A Correction to the Algorithm in Reiter's Theory of Diagnosis

Russell Greiner*
Department of Computer Science
University of Toronto
Toronto, Ontario M5S 1A4

Barbara A. Smith† **and Ralph W. Wilkerson**
Computer Science Department
University of Missouri at Rolla
Rolla, Missouri 65401

Abstract

Reiter [1987] has developed a general theory of diagnosis based on first principles. His algorithm computes all diagnoses which explain the differences between the predicted and observed behavior of a given system. Unfortunately, Reiter's description of the algorithm is incorrect in that some diagnoses can be missed under certain conditions. This note presents a revised algorithm and a proof of its correctness.

1 Introduction

Many researchers have developed systems for diagnosis which use a "first principles approach" using a representation language generally based on first-order logic. Both Reiter [1987] and deKleer and Williams [1987] use the concept of a conflict set as the basis of their methods. While Reiter's algorithm can make use of conflict sets which are not minimal, de Kleer and Williams's algorithm requires that minimal conflict sets be determined by the underlying inference mechanism.[1] However, it is the application of a technique for handling the non-minimal conflict sets that introduces a bug into Reiter's algorithm.

Section 2 presents a brief review of the pertinent definitions and a statement of Reiter's algorithm for computing minimal hitting sets. For the sake of completeness, we reproduce these concepts and definitions from Reiter's paper essentially unchanged. Section 3 presents an example where the Reiter's algorithm fails to find all of the minimal hitting sets. Section 4 contains a revised algorithm and a proof of correctness for this algorithm.

2 Definitions

A *hitting set* for a collection of sets C is a set $H \subseteq \bigcup_{S \in C} S$ such that $H \cap S \neq \{\}$ for each $S \in C$. A hitting set for C is minimal if and only if no proper subset of it is a hitting set for C.

A *system* to be diagnosed is defined by a set of COMPONENTS, a system description SD, and a set of observations, OBS (the latter two are sets of propositions). A *diagnosis* for (SD, COMPONENTS, OBS) is defined to be a minimal set $\Delta \subseteq$ COMPONENTS such that

$$\begin{aligned} \text{SD} \cup \text{OBS} \quad & \cup \{ \neg\text{AB}(c) \mid c \in \text{COMPONENTS} - \Delta \} \\ & \cup \{ \text{AB}(c) \mid c \in \Delta \} \end{aligned}$$

is consistent, where AB is a predicate indicating that a component is abnormal. The method of computing diagnoses is based on the determination of minimal hitting sets, since a diagnosis can be defined in terms of minimal hitting sets.

Reiter proposes a characterization of a diagnosis which uses the concept of a conflict set. A *conflict set* for (SD, COMPONENTS, OBS) is a set $\{c_1, \ldots, c_k\} \subseteq$ COMPONENTS such that SD $\cup$ OBS $\cup \{\neg\text{AB}(c_1), \ldots \neg\text{AB}(c_k)\}$ is inconsistent. A conflict set for (SD, COMPONENTS, OBS) is minimal if and only if no proper subset of it is a conflict set for (SD, COMPONENTS, OBS).

Two of the main results of Reiter's work are (1) the following theorem which relates diagnoses, conflict sets, and hitting sets and (2) a method for computing minimal hitting sets.

Theorem 1 *[Reiter, 1987; Theorem 4.4]* $\Delta \subseteq$ COMPONENTS *is a diagnosis for (*SD, COMPONENTS, OBS*) if and only if* Δ *is a minimal hitting set for the collection of conflict sets for (*SD, COMPONENTS, OBS*).*

The minimal hitting sets are computed by constructing a hitting set tree (HS-tree). An HS-tree is defined as follows.

Definition 1 *Let* C *be a collection of sets. An HS-tree for* C*, call it* T*, is a smallest edge-labeled and node-labeled tree with the following properties:*

1. *The root is labeled by* $\surd$ *if* C *is empty. Otherwise the root is labeled by an arbitrary set of* C*.*
2. *For each node* n *of* T*, let* $H(n)$ *be the set of edge labels on the path in* T *from the root node to* n*. The label for* n *is any set* $\Sigma \in C$ *such that* $\Sigma \cap H(n) = \{\}$*, if such a set* Σ *exists. Otherwise, the label for* n *is* $\surd$*. If* n *is labeled by the set* Σ*, then for each* $\sigma \in \Sigma$*,* n *has a successor,* n_σ*, joined to* n *by an edge labeled by* σ*.*

---

*Current address: Siemens Corporate Research, 755 College Road East, Princeton, NJ 08540-6632.

†Current address: Computer Science Department, University of Dayton, Dayton, OH 45469-2160.

[1] Many others, including [Provan, 1987], have discussed mechanisms for handling only minimal conflict sets. This note reports on a slightly different process, one which can accommodate non-minimal conflict sets as well.

Reiter identifies two properties of an HS-tree for a collection of sets C. First, for any node n labeled by $\surd$, $H(n)$ is a hitting set for C. Second, every minimal hitting set for C is $H(n)$ for some node n which is labeled by $\surd$. Reiter states these properties without proof.

For the diagnostic problem, the sets in the collection which are used as node labels are conflict sets for (SD, COMPONENTS, OBS). These sets are not explicitly known and are calculated as needed by an underlying theorem prover. In the algorithm for the construction of an HS-tree, the set to be used as the label of a node is determined by an access to the collection C. However, in diagnosis, the set to be used as a label of a node is determined by a call to an underlying theorem prover. As Reiter points out, the computation of a conflict set by the theorem prover must be treated as computationally expensive.

In order to (1) keep the HS-tree as small as possible, (2) calculate only minimal hitting sets, and (3) minimize the number of calls to the underlying theorem prover, Reiter provides an algorithm for generating a pruned HS-tree. The method is:

1. Generate the pruned HS-tree breadth first, generating all nodes at any fixed level in the tree before descending to generate the nodes at the next level.
2. Reusing node labels: If node n has already been labeled by a set $S \in C$, and if n' is a new node such that $H(n') \cap S = \{\}$, then label n' by S. Such a node n' requires no access to the theorem prover. (In our diagrams, we underline the label of node to indicate that this label is determined by reusing an existing label.)
3. Tree pruning:
   (a) If node n is labeled by $\surd$ and node n' is such that $H(n) \subseteq H(n')$, then close the node n'. A label is not computed for n' nor are any successor nodes generated. (In our diagrams, $\times$ indicates a closed node.)
   (b) If node n has been generated and node n' is such that $H(n') = H(n)$, then close node n'.
   (c) If nodes n and n' have been labeled by sets S and S' of C, respectively, and if S' is a proper subset of S, then for each $\alpha \in S - S'$ mark as redundant the edge from node n labeled by α. A redundant edge, together with the subtree beneath it, may be removed from the HS-tree while preserving the property that the resulting pruned HS-tree will yield all minimal hitting sets for C.

3 Problems with the algorithm

It should be clear that pruning by removing redundant edges (pruning rule 3c) is applicable only when there is at least one set in the collection which is a strict superset of some other set in the collection. Recall that for the problem of diagnosis, the minimal hitting sets of the conflict sets are the diagnoses. As already pointed out, an advantage of Reiter's method is that the conflict sets determined by the underlying theorem prover need not be minimal. However, this type of pruning can result in an incomplete diagnostician, as it is possible to lose minimal hitting sets, and therefore, diagnoses.

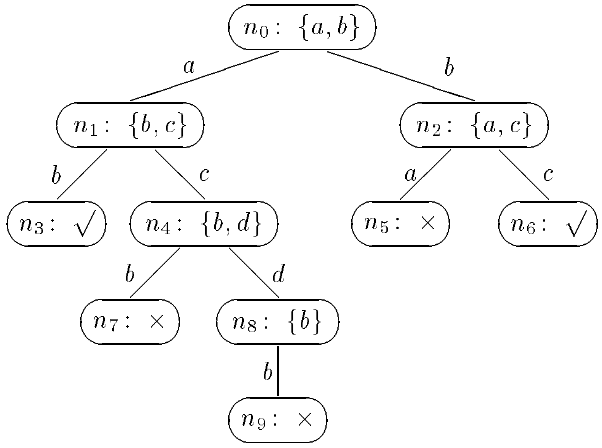

Figure 1: HS-tree illustrating the problem with pruning.

Consider the collection of sets: $\{ \{a,b\}, \{b,c\}, \{a,c\}, \{b,d\}, \{b\} \}$. Without pruning by removing redundant edges, the HS-tree shown in Figure 1 would be generated. Identifying node labels have been added. Note that nodes n_5, n_7, and n_9 have been closed by the subset rule (pruning rule 3a) since n_3 is labeled $\surd$, $H(n_3) \subseteq H(n_5)$, $H(n_3) \subseteq H(n_7)$, and $H(n_3) \subseteq H(n_9)$. The set labeling node n_8, $\{b\}$, is a proper subset of the sets labeling nodes n_0, n_1, and n_4. If the redundant branches from n_0, namely the branch labeled "a" is pruned, the remaining tree contains only the nodes n_0, n_2, n_5, and n_6. The minimal hitting set $\{a,b\}$ is no longer represented in the tree.

The problem arises from the interaction of the pruning rule which removes redundant edges (rule 3c) and the closing rules (rules 3a and 3b). A closing rule will close the node n when it finds another node n' which will lead to the same minimal hitting set(s). This, of course, assumes that the node n' will remain in the HS-tree. The pruning rule, however, may remove the node n', meaning that the path to any potential hitting sets will be totally lost—lost from the node n path when node n was closed and lost from the node n' path when node n' was pruned.

Before presenting the solution to this problem, we first clarify one point in Reiter's original algorithm. Pruning by the removal of redundant edges also requires that the pruned node be relabeled. Consider the collection of sets: $\{ \{a,b\}, \{a\}, \{b\} \}$. Without pruning, the HS-tree in Figure 2 would be generated. As $\{b\} \subset \{a,b\}$, the "a" branch under n_0 would be pruned. However, if n_0 is not relabeled by the set $\{b\}$, then the "b" branch under n_0 would be pruned as $\{a\} \subset \{a,b\}$. The surviving HS-tree would contain the single node n_0 which is not labeled by $\surd$.

The pruning method is based on the argument (presented in Reiter's paper) that when a node n is labeled by a set S' and there is a set $S \in C$ where $S \subset S'$, then n could be labeled by S rather than S'. This justifies removing the edges descending from n which are labeled

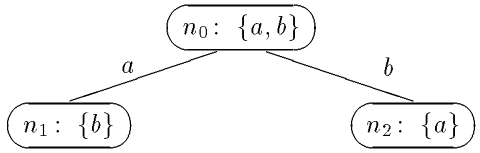

Figure 2: HS-tree illustrating the need for node relabelling.

by the members of $S' - S$, leaving only the edges labeled by members of S. In the text of the paper, Reiter discusses relabeling the node, but this point is not stated in the algorithm.

4 Revised Algorithm

Reiter's description (in the text) of the process for computing the minimal hitting sets is basically correct. However, the algorithm did not accurately follow his text. The HS-DAG algorithm, shown below, is more faithful to that description. It involves using a directed acyclic graph, dag, to compute the minimal hitting sets rather than a tree. To simplify the description, we assume that the collection of sets is ordered. This allows us to specify the algorithm deterministically, as we can now select a member of this collection rather than assume that a member is chosen arbitrarily.

We begin by defining the HS-DAG$_0$ algorithm for constructing the HS-dag for an ordered collection of sets, F.

1. Let D represent the growing dag. Generate a node which will be the root of the dag. This node will be processed in Step 2 below.
2. Process the nodes in D in a breadth first order. To process a node n:
   (a) Define $H(n)$ to be the set of edge labels on the path in D from the root down to node n.
   (b) If for all $x \in F$, $x \cap H(n) \neq \{\}$ then label n by $\surd$. Otherwise, label n by Σ where Σ is the first member of F for which $\Sigma \cap H(n) = \{\}$.
   (c) If n is labeled by a set $\Sigma \in F$, then for each $\sigma \in \Sigma$, generate a new downward arc labeled by σ. This arc leads to a new node m with $H(m) = H(n) \cup \{\sigma\}$. The new node m will be processed (labeled and expanded) after all nodes in the same generation as n have been processed.
3. Return the resulting dag, D.

This algorithm corresponds to Reiter's basic algorithm for constructing the HS-tree algorithm, without pruning. It differs only by labeling a node by the first member of F that qualifies, rather than by an arbitrary member. Note, also, that as a result of ordering the collection of sets, the algorithm will reuse node labels wherever possible.

Following Reiter, we propose three pruning enhancements to the HS-DAG$_0$ algorithm in order to reduce the size of the dag and also generate only the minimal hitting sets.

1. Reusing Nodes: This algorithm will not always generate a new node m as a descendant of node n. There are two cases to consider:
   (a) If there is a node n' in D such that $H(n') = H(n) \cup \{\sigma\}$, then let the σ-arc under n point to this existing node n'. Hence, n' will have more than one parent.
   (b) Otherwise, generate a new node, m, at the end of this σ-arc as described in the basic HS-DAG$_0$ algorithm.
2. Closing: If there is node n' which is labeled by $\surd$ and $H(n') \subset H(n)$, then close node n. A label is not computed for n nor are any successor nodes generated.
3. Pruning: If the set Σ is to label a node and it has not been used previously, then attempt to prune D as described in the following.
   (a) If there is a node n' which has been labeled by the set S' of F where $\Sigma \subset S'$, then relabel n' with Σ. For any α in $S' - \Sigma$, the α-edge under n' is no longer allowed. The node connected by this edge and all of its descendants are removed, except for those nodes with another ancestor which is not being removed. Note that this step may eliminate the node that is currently being processed.
   (b) Interchange the sets S' and Σ in the collection. (Note that this has the same effect as eliminating S' from F.)

Figure 3 shows a partial HS-dag for the collection of sets used earlier, namely, $\{\ \{a,b\},\ \{b,c\},\ \{a,c\},\ \{b,d\},\ \{b\}\ \}$. When the set $\{b\}$ is first used as a label, the dag is pruned as shown in Figure 4. Note that node n_3 still has a parent and so remains in the dag. Thus, the minimal hitting set $\{a,b\}$ is not lost as was the case with the HS-tree algorithm.

Let HS-DAG refer to the overall algorithm with the pruning rules included. Note that the particular HS-dag which is returned by the algorithm depends on how F is ordered. Using $\Pi(F)$ to refer to the Π rearrangement of F, HS-DAG(F) and HS-DAG$(\Pi(F))$ will lead to different HS-dags. We prove below that these two graphs will produce the same minimal hitting sets where each hitting set is $H(n)$ for some node n labeled by $\surd$.

Theorem 2 (Correctness of HS-DAG Algorithm) *Given the ordered collection F, the HS-DAG algorithm returns a particular labeled dag.*

1. *For all nodes n labeled by $\surd$, $H(n)$ is a minimal hitting set.*
2. *Every minimal hitting set for F is $H(n)$ for some node n whose label is $\surd$.*

Proof: It is sufficient to prove the following three points: (1) The basic HS-DAG$_0$ algorithm (without the pruning rules) will find all of the minimal hitting sets, (2) the

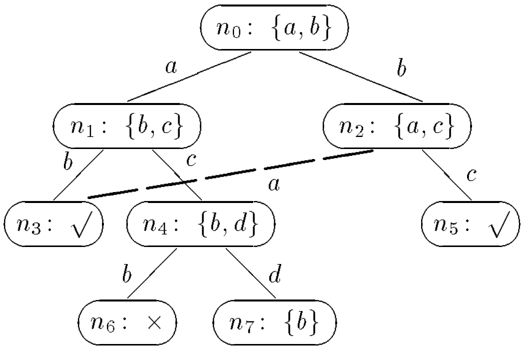

Figure 3: HS-dag before pruning.

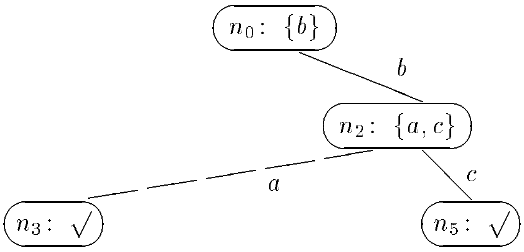

Figure 4: HS-dag after pruning.

pruning rules will not eliminate any of the minimal hitting sets, (3) the pruning rules will eliminate all of the non-minimal hitting sets.

1. This claim is stated, without proof in [Reiter, 1987; p. 72]. The claim applies to Reiter's basic algorithm for constructing an HS-tree without pruning. Obviously, if it is true when the members of F are selected arbitrarily, it must be true for any particular ordering, that is, for our related basic HS-DAG$_0$ algorithm. We prove it below in Lemma 1.

2. It suffices to show that none of the three pruning rules will remove any $\surd$-labeled node.

   (a) The process of reusing nodes does not remove any nodes. It is simply used to encode an HS-tree as a dag. Notice that we can recapture the tree information by replicating each node that has multiple parents. For each node n with multiple parents: Assume that n is connected to its m_i parent by the α_i-labeled branch, that is, the α_i branch of m_i leads down to n for $i = 1,\ldots,k$. The sub-dag rooted at n can be replicated k times, with the α_i branch of m_i pointing to the i^{th} copy. We obtain a tree by expanding each such node. As this step has no effect on any of the $\surd$-labeled nodes, it does not interfere with either of the other pruning rules nor with other applications of itself.

   (b) The process of closing nodes does remove some nodes from an HS-dag. By construction, it only removes a node n if there is a $\surd$-labeled node n' for which $H(n')$ is a proper subset of $H(n)$. Note that $H(n')$ is a hitting set since n' is labeled $\surd$. The basic HS-DAG$_0$ algorithm would have left node n and all of its descendants. Let c be any $\surd$-labeled node in the sub-dag rooted at node n. Notice that $H(c)$ must be a superset of $H(n)$ and thus it would be a strict superset of $H(n')$. As $H(n')$ is a hitting set, $H(c)$ cannot be a minimal hitting set.

   (c) Pruning transforms an HS-dag into another one. That is, it produces the dag associated with HS-DAG$(\Pi(F))$ for some rearrangement Π, rather than the one begun for HS-DAG(F). In general, when a node's label is changed from S to S', then $\Pi(F)$ will interchange S and S' in F. Note that there is always some permutation of the members of F such that pruning will never apply.

3. Recall that, by definition, for every non-minimal hitting set h there is a minimal hitting set h_m such that $h_m \subset h$. Suppose that there is a node n in the unpruned HS-dag which the basic construction algorithm would have produced such that $H(n)$ is this non-minimal hitting set h. From point 2 above, we know that this HS-dag will include a node labeled $\surd$ whose H-set is h_m. Call this node n_m. Notice that n_m will appear closer to the root of the dag than will node n. By virtue of the algorithm's breadth-first ordering, n_m will be generated before n. Enhancement (2) (Closing) of the HS-DAG construction algorithm would prevent this node n from being generated and labeled as this rule would close node n as soon as it was considered. This enhancement might actually close the ancestors of n. $\Box$

Lemma 1 *The basic* HS-DAG$_0$ *algorithm, without any of the pruning enhancements, will find all of the minimal hitting sets. That is, let T be the HS-dag that it returns and let h be any minimal hitting set. Then T will include a node n such that (1) $H(n)$ is h and (2) the label for n is $\surd$.*

Proof: By induction on the cardinality of F:

Base. If $|F| = 0$ then the only minimal hitting set for F is the empty set. Notice that the only HS-dag for F is the degenerate tree consisting of a single node n_0 labeled by $\surd$. As $H(n_0) = \{\}$, every possible HS-dag for F includes all of the minimal hitting sets for F.

Induction. Let T be any HS-dag for the collection F where $|F| = n+1$. Let its root node be labeled by f_0 where $f_0 \in F$ and $f_0 = \{m_1,\ldots,m_k\}$. Define F_i to be the members of F which do not include the element m_i, *i.e.*, $F_i = \{f \in F \mid m_i \notin f\}$. Notice that the sub-dag under m_i is an HS-dag for the collection F_i and that $|F_i| < |F|$. By inductive assumption, the H-sets of the $\surd$-labeled nodes of the HS-dag for F_i include all of the minimal hitting sets for F_i. This means that the H-sets for each of the $\surd$-labeled nodes in the HS-dag for F is of the form $h_i \cup \{m_i\}$ where h_i is a minimal hitting set for F_i. It suffices to show that this accounts for all of the minimal hitting sets for F.

Let h be any of the minimal hitting sets for F. By definition, there is an element, call it m_i, such that $m_i \in h$ and $m_i \in f_0$. This m_i is sufficient to account for every $f_j \in F$ for which $m_i \in f_j$. The remaining elements of h must (minimally) hit each of the remaining members of

F, that is, $h - \{m_i\}$ must be a minimal hitting set for F_i. This is true by construction. $\square$

5 Conclusion

This note demonstrates that Reiter's algorithm for computing minimal hitting sets fails under certain circumstances: While the basic algorithm for constructing a hitting set tree is correct, the application of pruning techniques can result in the loss of minimal hitting sets. We present a revised algorithm which uses a directed acyclic graph rather than a tree structure and prove it is correct.

Acknowledgements

Barbara Smith and Ralph Wilkerson were supported in part under grants from McDonnell Douglas Research Laboratories (Independent Research and Development) and the Missouri Research Assistance Act. Russell Greiner was supported by a grant from Canada's National Science and Engineering Research Council. We would also like to thank Ruth Aydt for pointing out the need for a clarification on node relabeling.

References

[deKleer and Williams, 1987] Johan deKleer and Brian C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, April 1987.

[Provan, 1987] Gregory Provan. Complexity analysis of multiple-context TMSs in scene representation. In *Proceedings of the American Association for Artificial Intelligence*, pages 173–177, 1987.

[Reiter, 1987] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–96, April 1987.

Received July 1988; revised version received January 1989