Machine Learning, 13, 35–70 (1993)

Explanation-Based Learning for Diagnosis

YOUSRI EL FATTAH FATTAH@ICS.UCI.EDU
Department of Information and Computer Science, University of California, Irvine, CA 92717-3425

PAUL O'RORKE ORORKE@ICS.UCI.EDU
Department of Information and Computer Science, University of California, Irvine, CA 92717-3425

Abstract. We present explanation-based learning (EBL) methods aimed at improving the performance of diagnosis systems integrating associational and model-based components. We consider multiple-fault model-based diagnosis (MBD) systems and describe two learning architectures. One, EBLIA, is a method for "learning in advance." The other, EBL(p), is a method for "learning while doing." EBLIA precompiles models into associations and relies only on the associations during diagnosis. EBL(p) performs compilation during diagnosis whenever reliance on previously learned associational rules results in unsatisfactory performance—as defined by a given performance threshold p. We present results of empirical studies comparing MBD without learning versus EBLIA and EBL(p). The main conclusions are as follows. EBLIA is superior when it is feasible, but it is not feasible for large devices. EBL(p) can speed-up MBD and scale-up to larger devices in situations where perfect accuracy is not required.

Keywords. explanation-based learning, model-based reasoning, rule-based expert systems, diagnosis

1. Introduction

Diagnostic expert systems constructed using traditional knowledge-engineering techniques identify malfunctioning components using rules that associate symptoms with diagnoses (Feigenbaum, 1979). Model-based diagnosis (MBD) systems use models of devices to find faults given observations of abnormal behavior (Davis & Hamscher, 1988). These approaches to diagnosis are complementary. The associational approach takes advantage of human experts' empirical knowledge of the behavior of faulty devices in practice. MBD takes advantage of models of devices that can be generated during design, circumventing the knowledge engineering process and eliminating the need for a human who is an expert at diagnosing the device. MBD systems can cope with novel and multiple-faults but at a computational price. MBD is combinatorially explosive (de Kleer, 1991), while associational systems are relatively efficient. In this article, we consider hybrid diagnosis systems that include both associational and model-based components.

A principal shortcoming of existing diagnosis systems is that they learn nothing from any given task. Upon facing the same task a second time, they will incur the same computational expenses as were incurred the first time. We describe several architectures that integrate learning with associational and model-based diagnosis. The architectures take advantage of the strengths of both diagnosis methods while attempting to avoid the weaknesses. In these architectures, diagnostic associations are preferred because they tend to be more efficient, but model-based reasoning is available for multiple and novel faults.

We use explanation-based learning (EBL) (DeJong & Mooney, 1986; Mitchell, Keller, & Kedar-Cabelli, 1986) to transform knowledge contained in device models into associational rules.

The structure of this article is as follows. Section 2 states the MBD task and describes the performance element. Section 3 describes how EBL can be integrated with MBD and presents two learning architectures, EBLIA and EBL(p). Section 4 provides a detailed description of the results of computational experiments evaluating the learning methods. Section 5 provides discussions of the results. Section 6 points out related works. Section 7 gives general conclusions.

2. Model-based diagnosis

Following Reiter (1987) and de Kleer, Mackworth, and Reiter (1992), we define *model-based diagnosis* in terms of a 3-tuple (*SD*, *COMPS*, *OBS*) where

1. *SD*, the system description, is a set of first-order sentences;
2. *COMPS*, the system components, is a finite set of constants;
3. *OBS*, the observation, is a finite set of first-order sentences.

The system description, *SD*, consists of the structural and functional description of the device. The structure consists of the connections between the various components and the mappings between various variables. The function is described by a set of constraints for the various components. A constraint is represented as a set of value inference rules, defined as follows.

Definition 2.1. A *value inference rule* $r(c, X \rightarrow Y)$ *for a component* $c \in COMPS$ *is an implication,* $x \rightarrow y$, *whose condition is a value assignment tuple,* $x = (x_1, x_2, \ldots, x_n)$ *for a subset of the component variables* $\{X_1, X_2, \ldots, X_n\} \subset vars(C)$, *and its conclusion is a value assignment* y *for a variable* $Y \in vars(C)$, $Y \notin X$. *A value assignment for a condition variable* X_i *can either be a specific value in the domain of* X_i, *or a logical variable that matches any value in that domain. The value assignment for the conclusion variable* Y *is either a specific value in the domain of* Y, *or a function of the logical variables appearing in the assignment of* X.

Example 2.1. A component of type multiplier whose input is X, Y and whose output is Z can be described by the following value inference rules:

$$(X = x, Y = y) \rightarrow Z = x * y \tag{1}$$

$$(Y = y, Z = z) \rightarrow X = z/y \tag{2}$$

$$(X = x, Z = z) \rightarrow Y = z/x \tag{3}$$

Example 2.2. Consider a component whose function is to output the logical and of its inputs. Let the inputs be X, Y and the output Z. The component can be described by the following rules:

$$(X = 1, Y = 1) \rightarrow Z = 1 \tag{4}$$

$$(Z = 1) \rightarrow X = 1 \tag{5}$$

$$(Z = 1) \rightarrow Y = 1 \tag{6}$$

$$(X = 0) \rightarrow Z = 0 \tag{7}$$

$$(Y = 0) \rightarrow Z = 0 \tag{8}$$

Implicit in the system description is the assumption that the system is behaving "normally." Abnormal behavior assumes no constraint on the system variables; anything can be happening. To make the normality/abnormality assumptions explicit in our inferences, we associate each constant $c \in COMPS$ with abnormal literals $ab(c)$ or $\neg ab(c)$, where $ab(c)$ means "c is abnormal" while $\neg ab(c)$ means "c is ok." We will make use of the following definitions.

Definition 2.2. *For any subset $C \subseteq COMPS$, the predicate normal(C) is defined as the conjunction*

$$normal(C) = \bigwedge_{c \in C} \neg ab(c)$$

corresponding to the condition that every component in C is not abnormal.

Definition 2.3. *For any subset $C \subseteq COMPS$, the predicate faulty(C) is defined as the conjunction*

$$faulty(C) = \bigwedge_{c \in C} ab(c)$$

corresponding to the condition that every component in C is abnormal.

Intuitively, a diagnosis is a smallest set of components such that the assumption that each of these components is faulty (abnormal), together with the assumption that all other components are behaving correctly (not abnormal), is consistent with the system description and the observations. This is formalized by the following definition.

Definition 2.4. *A* diagnosis *for (SD, COMPS, OBS) is a minimal set $\Delta \subseteq COMPS$ such that*

$$SD \cup OBS \cup faulty(\Delta) \cup normal(COMPS - \Delta)$$

is consistent.

The MBD system discussed in this article is based on the theory of diagnosis given by Reiter (1987) and emulates the GDE system of de Kleer and Williams (1987). The method for determining all diagnoses for (*SD, COMPS, OBS*) is based on the concept of a conflict set, originally due to de Kleer (1976).

Definition 2.5. *A* conflict set *for* (*SD, COMPS, OBS*) *is a set* $CONF \subseteq COMPS$ *such that*

$$SD \cup OBS \cup normal(CONF)$$

is inconsistent. A conflict set for (*SD, COMPS,* OBS) *is* minimal *iff no proper subset of it is a conflict set for* (*SD, COMPS, OBS*). *A conflict set CONF corresponds to a clause,*

$$\bigvee_{c \in CONF} ab(c)$$

called a conflict. That clause is entailed by $SD \cup OBS$.

A result by Reiter (1987) (theorem 4.4) shows that $\Delta \subseteq COMPS$ is a *diagnosis* for (*SD, COMPS, OBS*) iff Δ is a minimal set cover (hitting set) for the collection of (minimal) conflict sets for (*SD, COMPS, OBS*). A cover can be defined: given a set of subsets F, a set C is a cover of F iff any set in F contains an element in C.

The task of computing all diagnoses for (*SD, COMPS, OBS*) can be represented as a three-step process, as shown in figure 1, and is described as follows:

Prediction by propagating observations through all constraints;
Conflict recognition by determining all (minimal) assumptions responsible for discrepancies between predictions and observations;
Candidate generation by finding all minimal set covers of the collection of conflicts.

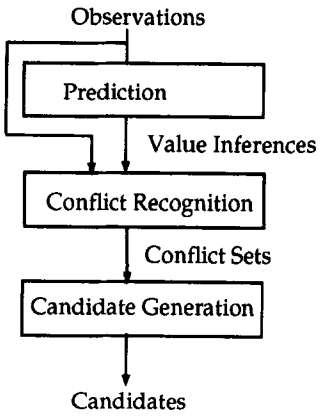

Figure 1. Model-based diagnosis.

This diagnostic task is only one phase in the diagnosis cycle, which is followed by the task of selecting a test or a probe for discrimination between diagnostic candidates. The task of test/probe selection is not addressed in this article, although most of the results here serve as a basis for the computations underlying that task.

2.1. Prediction

Prediction is the key to model-based diagnosis. Given the model and the observations, prediction consists in determining for each (variable, value) pair all the assumptions that entail it. Intuitively, the prediction task involves making inferences about the overall behavior of the device based on the assumption that the various components are behaving normally. These inferences are defeasible.

Prediction is performed as a value inference constraint propagation process, triggered by the values of observed variables (called premises). An example of a premise is a value assignment for the input and output variables of a device. In diagnosis, the input assignment corresponds to some test vector, and the output assignment corresponds to observed outputs. The prediction process uses an ATMS (de Kleer, 1986) as an intelligent cache for the value inferences. Value inferences are stored with associated labels, where an ATMS label describes the set of minimal environments (sets of assumptions) in which the associated value inference is verified. The prediction process integrates a value-inference engine with an ATMS cache and is described in table 1. The system description is specified by a set

Table 1. The prediction algorithm **Propagate**.

Input: (*SD*, *COMPS*, *OBS*)

Output: Value inferences for system variables and associated minimal sets of (normality) assumptions in which each inference is valid.

Initialize: For each observation in *OBS* assert the observed value for the corresponding variable as a premise and assign to it an empty assumption label and an empty dependency label.

Description:

1. *Change* ← *false*
2. For each component $c \in COMPS$
   For each rule $r(c, X \rightarrow Y) \in SD$ do:
   (a) For each set of values for X that satisfy the rule condition and whose dependencies do not include Y do:
      i. Determine value inference for Y
      ii. Set the dependency label for Y to be the union of X and the dependency labels for X; Set the assumption label for Y to be the union of $\{c\}$ and the assumption labels for X.
      iii. If the inference for Y is not subsumed by a previous inference then do:
         A. Assert the current inference
         B. Retract existing inferences for Y subsumed by the current inference
         C. *Change* ← *true*
3. If *Change* then go to 2.

of production rules whose conditions include assumptions and value assignments for variables, whose conclusions are value inferences for variables. Prediction is a forward-chaining value inference process triggered by the observation. When a value inference is made, an assumption and a dependency label are also determined. Only new value inferences with minimal assumptions are recorded. This is done by checking whether the value inference is subsumed by a previous one (step 2(a)iii). If not, then we assert it and retract all previous inferences subsumed by the current inference (step 2(a)iiiB). To see the need for this step, consider the case where the first time the value inference is made the label is non-minimal.

The following two examples show the predictions derived by the procedure *Propagate* for the outputs of two simple circuits. The predictions are represented as Horn clauses whose conditions consist of the minimal set of normality assumptions for which the prediction is valid.

Example 2.3. Consider the polybox circuit depicted in figure 2 with the input–output (I/O) observations (premises)

$$A = 3,\ B = 2,\ C = 2,\ D = 3,\ E = 3,\ F = 10,\ G = 12 \tag{9}$$

In this circuit, $M1$, $M2$, and $M3$ are multipliers, while $A1$, and $A2$ are adders. Propagating the premises $A = 3$ and $C = 2$ through the multiplier $M1$ produces the prediction $X = 6$ with the label $normal([M1])$, and propagating $B = 2$, $D = 3$ through $M2$ produces $Y = 6$ with the label $normal([M2])$. Propagating the inferences $X = 6$, $Y = 6$ through the adder $A1$ produces the prediction $F = 12$. The assumption label for that prediction is $normal([M1, M2, A1])$, obtained by propagating the assumptions for X and Y. This amounts to the assertion that under the assumption that none of the components $M1$, $M2$, $A1$ is abnormal, the output F is predicted to be 12. This can be expressed as the logical formula

$$normal([M1, M2, A1]) \rightarrow F = 12 \tag{10}$$

Similarly, we can conclude that the output G should be 12 under the assumption that the components $M2$, $M3$, and $A2$ are not behaving abnormally, i.e.,

$$normal([M2, M3, A2]) \rightarrow G = 12. \tag{11}$$

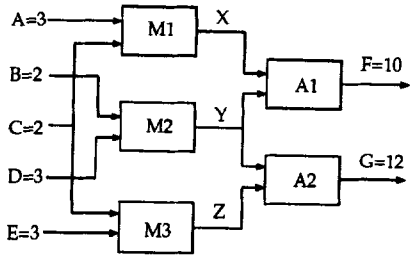

Figure 2. The polybox circuit.

Propagating the output $G = 12$ and the prediction $Z = 6$ (whose label is $[M3]$) through the adder $A2$ produces the prediction $Y = 6$ and the label $normal([A2, M3])$. Propagating that prediction for Y along with the prediction $X = 6$ (whose label is $normal([M1])$ through the adder $A1$ produces the prediction that F should be 12 with the label $normal([M1, M3, A1, A2])$. This corresponds to the formula

$$normal([M1, M3, A1, A2]) \rightarrow F = 12 \tag{12}$$

Similary, we can conclude that under the assumption that $A1$, $A2$, $M1$, $M3$ are working correctly, G should be 10, i.e.,

$$normal([M1, M3, A1, A2]) \rightarrow G = 10 \tag{13}$$

Note that the I/O premises do not appear in the conditions of the prediction formulas (10)–(13); the predictions are all made in the context of those premises.

Example 2.4. Consider the one-bit adder circuit in figure 3, with the input–output

$$X1 = 0,\ Y1 = 0,\ C0 = 0,\ S1 = 1,\ C1 = 0. \tag{14}$$

Propagating the input bits $X1$, $Y1$ through the exclusive-or gate $Xor1$ produces the prediction $Ox1 = 0$ with the label $normal([Xor1])$. Propagating that prediction along with the input carry $C0 = 0$ through the exclusive-or gate $Xor2$ produces the prediction that the sum bit $S1$ should be 0 under the assumption that $Xor1$ and $Xor2$ are functioning correctly. That is,

$$normal([Xor1, Xor2]) \rightarrow S1 = 0. \tag{15}$$

Also, propagating either one of the input bits $X1$, $Y1$ through the and-gate $And1$ produces the prediction that $Oa1$ should be 0 provided that $And1$ is not abnormal. Propagating the input carry $C0 = 0$ through the and gate $And2$ produces the prediction that $Oa2 = 0$ provided that $And2$ is not abnormal. Then propagating those predictions for $Oa1$, $Oa2$ through the or gate $Or1$ produces the prediction $C1 = 0$ provided that components $And1$, $And2$, and $Or1$ are all functioning correctly. That is,

$$normal([And1, And2, Or1]) \rightarrow C1 = 0. \tag{16}$$

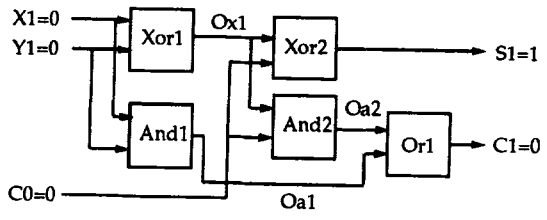

Figure 3. A full adder.

2.2. *Conflict recognition*

Conflict recognition consists in identifying sets of default normality assumptions that lead to predictions that are inconsistent with the observations. Conflict recognition is performed by comparing predictions with premise assignments recording observed values. If there is a discrepancy, then the support set of the prediction inference is declared as a conflict set.

Example 2.5. The polybox circuit with inputs and outputs as given in example 2.3 results in two conflicts. One conflict results from the prediction $F = 12$, equation (10), and the observation, $F = 10$. Using the ATMS terminology, the label of the prediction $F = 12$ becomes a nogood set, meaning that the assumptions that $A1$, $M1$, $M2$ are all working correctly cannot be part of any consistent environment; thus the conflict

$$ab(M1) \vee ab(M2)\ X\ ab(A1). \tag{17}$$

The other conflict results from either the prediction that $F = 12$, equation (12), and the observation, $F = 10$, or the prediction $G = 10$, equation (13), and the observation, $G = 12$. That conflict says that the components $M1$, $M3$, $A1$, $A2$ cannot be all working correctly; one of them must be faulty, i.e.,

$$ab(M1) \vee ab(M3) \vee ab(A1) \vee ab(A2). \tag{18}$$

Example 2.6. The one-bit adder with inputs and outputs as given in example 2.4 results in one conflict, namely, between the prediction of the sum bit $S1 = 0$ and the observation $S1 = 1$. The conflict set consists of the components $X\ or\ 1$ and $X\ or\ 2$; at least one of these components must be faulty, i.e.,

$$ab(Xor1) \vee ab(Xor2). \tag{19}$$

2.3. *Candidate generation*

Candidate generation consists in determining minimal sets of abnormality assumptions whose conjunction covers (accounts for) all known conflicts. This amounts to saying that if $ab(C1) \wedge ab(C2)$ is a candidate, then the suspension of the normal constraint for components $C1$ and $C2$ removes all conflicts (i.e., restores consistency). A candidate set is minimal if it does not include a subset that is also a candidate.

For the candidate generation step, we implemented an HS-Tree algorithm, based on Reiter (1987). Each node in the HS-tree is labeled with a conflict set, and each edge to its children is labeled with an element from that set (corresponding to a system component). Define the path label $H(n)$ of a node n to be the set of edge labels from the root of the HS-tree to the node. The HS-tree is built up breadth-first such that each node n's label is disjoint with its path label, $H(n)$. If no such label exists for a node, then that node is labeled by $\checkmark$. The path label to any node labeled by $\checkmark$ is a hitting set. Reiter assumes the existence of a theorem prover to be called by the HS-tree algorithm to find conflict sets for the node

labels. 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 theorem prover, Reiter (1987) provides the following heuristics for generating a pruned HS-tree:

1. **Reusing node labels:** If node n has already been labeled by a set S and if n' is a new node such that the path label to that node is disjoint with S, then label n' by S.
2. **Tree pruning**
   (a) **Closing rule-1.** If node n is labeled by $\checkmark$ 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.
   (b) **Closing rule-2.** If node n has been generated and node n' is such that $H(n') = H(n)$, then close n'.
   (c) **Remove redundant edges.** If node n and n' have been labeled by sets S and S', 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.

In our implementation, we do not consider two of Reiter's heuristics: 1) the "Reusing node labels" heuristic, and 2) the "Remove redundant edges" tree-pruning heuristic.[1] In our case the reuse heuristic is not needed, since we determine the entire collection of conflict sets prior to determining the hitting sets. The reason for not pruning is that the implementation is simpler without it. Our algorithm may generate a larger tree than necessary, but we are guaranteed not to miss any minimal hitting set.

Example 2.7. The polybox circuit with the two conflicts of example 2.5 results in four minimal candidates:

$$ab(M1) \tag{20}$$

$$ab(A1) \tag{21}$$

$$ab(M2) \wedge ab(M3) \tag{22}$$

$$ab(M2) \wedge ab(A2) \tag{23}$$

Example 2.8. The one-bit adder with the conflict of example 2.6 results in two minimal candidates:

$$ab(Xor1) \tag{24}$$

$$ab(Xor2) \tag{25}$$

3. Explanation-based learning

Explanation-based learning (EBL) is one proposal to speed up MBD, by accumulating problem-solving experience and using past experience on new problems. Experience is represented using rules of the form *Situation* → *Conclusion*; whenever faced with *Situation*, then jump directly to *Conclusion*. We now consider in detail how EBL can impact on the various phases of the diagnosis task.

3.1. Prediction

Traditional MBD must make predictions anew for every problem, even if a similar problem has been seen before. Prediction entails search in the assumption lattice to find the minimal support environments for all possible value inferences. Our proposal is to exploit the results of the search made on current problems in the prediction phase for use on future problems. The main intuition for applying EBL to the prediction phase is as follows. While making value inferences, the inference rules themselves are also propagated and unified to form what we call p-rules (prediction rules).

Definition 3.1. *Let vars(SD) be the system variables and vars(OBS) be the observation (premise) variables. A p-rule* $p(C, X \rightarrow Y)$ *is an implication*:

$$normal(C) \wedge x \rightarrow y$$

The condition of a p-rule is a conjunction of the normality predicate $normal(C)$, $C \subset COMPS$ *and a value assignment tuple,* $x = (x_1, x_2, \ldots, x_n)$ for a subset of observation variables; $\{X_1, X_2, \ldots, X_n\} \subset vars(OBS)$. *The conclusion of a p-rule is value assignment y for a system variable* $Y \in vars(SD)$. *A value assignment for a condition variable* X_i *can either be a specific value in the domain of* X_i, *or a logical variable that matches any value in that domain. The value assignment for the conclusion variable Y is either a specific value in the domain of Y, or a function of the logical variables appearing in the assignment of X.*

P-rules may replace the propagation procedure performed by Propagate. This has the following benefits:

1. The problem of finding predictions becomes backtrack-free:
   (a) The p-rules specify explicitly the minimal environment in which an inference is valid. Without p-rules, value inferences are retracted when they are subsumed by other environments.
   (b) The p-rules eliminate the need to search for inference chains, since they associate directly the observed (premise) variables with the system variables.
2. Inferences are no longer made for internal variables.

Learning p-rules is a way of allowing the "reuse" of search efforts on previous diagnosis problems.[2] The predictions made on previous problems may not have been useful for those problems in terms of discovering conflict sets. But the cached p-rules may be useful for new problems (see example 3.2 below).

The application of EBL to the prediction phase is performed by the procedure EBL-Propagate (see table 2). The following are examples of applying that procedure.

Example 3.1. Consider the polybox example 2.3. The procedure EBL-Propagate compiles the following p-rules for the output variable F:

$$normal([M1, M2, A1]) \wedge (A = a, B = b, C = c, D = d) \rightarrow F = a * c + b * d \quad (26)$$

$$normal([M1, M3, A1, A2]) \wedge (A = a, C = c, E = e, G = g) \rightarrow F = a * c + (g - c * e) \quad (27)$$

Similar rules are compiled for G.

Example 3.2. Consider the adder example 2.4. The procedure EBL-Propagate compiles the following p-rules for the output variables,

$$normal([X\ or\ 1, X\ or\ 2]) \wedge (X1 = x1, Y1 = y1, C0 = c0) \rightarrow S1 = x1 \oplus y1 \oplus c0 \quad (28)$$

$$normal([Xor1, And1, And2, Or1]) \wedge (X1 = 0, Y1 = 0) \rightarrow C1 = 0 \quad (29)$$

$$normal([And1, And2, Or1]) \wedge (X1 = 0, C0 = 0) \rightarrow C1 = 0 \quad (30)$$

$$normal([And1, And2, Or1]) \wedge (Y1 = 0, C0 = 0) \rightarrow C1 = 0 \quad (31)$$

For the given premise instance, either of the p-rules (30) or (31) is all that is needed for prediction. They both have the same assumption label, and that label subsumes that of rule (29). If we substitute for the premises, rules (29) and (30) will degenerate to prediction (16) of example 2.4. Although redundant for the given premises, rules (29) and (30) may be irredundant for other instances. For example, if the premise is $\{X1 = 0, Y1 = 1, C0 = 0\}$, then rules (29) and (31) are not applicable, but rule (30) is.

When EBL-Propagate is made to cover not only the given example of value assignments to the premise variables, but also all other possible assignments, the procedure becomes what we call EBLIA-Propagate. In EBL-Propagate we require that the learnt p-rules be consistent with the given premise instance (table 2, step 2c). EBLIA-Propagate is the same as EBL-Propagate, except that step 2c is replaced by general satisfiability, instead of satisfiability for a given premise instance. The rules compiled by EBLIA are to apply to all

Table 2. **EBL-Propagate**, a "learning while doing" prediction algorithm.

Input: (*SD*, *COMPS*, *OBS*)

Output: All p-rules applicable to generalization of the observations in *OBS*.

Initialization: For each observed variable V assert a p-rule $normal([\,]) \wedge (V = v) \rightarrow V = v$, where v is a logical variable that matches any value in the domain of V. Set the p-rules' dependencies to nil.

Description:

1. *Change* ← *false*
2. For each component $c \in COMPS$
   For each inference rule $r(c, X \rightarrow Y) \in SD$
   For each collection S of p-rules $p(C_i, Z_i \rightarrow X_i) \mid X_i \in X)$ whose dependencies do not include Y do:
   (a) Unify the conclusions of the p-rules with the conditions of the inference rule.
   (b) Set Z to be the union of Z_i for all p-rules in S.
   (c) Verify that the condition set on Z is satisfiable by *OBS*.
   (d) Form a new p-rule $p(C, Z \rightarrow Y)$. C is the union of $\{c\}$ and C_i for all p-rules in S. Set the dependency label for that rule to be the union of X and the dependencies for all p-rules in S.
   (e) If the new p-rule is not subsumed by a prior rule then do:
      i. Assert the current rule
      ii. Retract existing rules subsumed by the current one
      iii. *Change* ← *true*
3. If *Change* then go to 2.

Table 3. **EBLIA-Propagate**, a "learning in advance" prediction algorithm.

Input: Premise variables.

Output: All p-rules covering every possible instantiation of premise variables from their domain.

Description: Follow every step in EBL-Propagate except for step 2c. Instead of that step do: Verify that the condition set on Z is satisfiable for some instantiation of premise variables from their domain.

possible instantiations of the premise set, rather than to only a generalization of a given instance as in EBL-Propagate (see table 3).

Example 3.3. For the polybox circuit, applying EBLIA-Progagate produces the same p-rules as in example 3.1.

Example 3.4. For the one-bit adder circuit, EBLIA-Propagate compiles the following p-rules in addition to those compiled by EBL-Propagate (example 3.2):

$$normal([And1, Xor2, And2, Or1]) \wedge (X1 = 0, C0 = c0, S1 = c0) \rightarrow C1 = 0 \tag{32}$$

$$normal([And1, Xor2, And2, Or1]) \wedge (Y1 = 0, C0 = c0, S1 = c0) \rightarrow C1 = 0 \tag{33}$$

$$normal([And1, Or1]) \wedge (X1 = 1, Y1 = 1) \rightarrow C1 = 1 \quad (34)$$

$$normal([Xor1, And2, Or1]) \wedge (X1 = x1, Y1 = \neg x1, C0 = 1) \rightarrow C1 = 1 \quad (35)$$

$$normal([Xor2, And2, Or1]) \wedge (C0 = 1, S1 = 0) \rightarrow C1 = 1 \quad (36)$$

The reason the above rules are not compiled by EBL-Propagate is that their conditions are incompatible with the given input–output values. In general, the p-rules learnt by the EBL-Propagate depend on the particular observation instance. In general, EBL-Propagate requires multiple examples to learn all the p-rules that are learnt by EBLIA-Propagate. For the polybox circuit, one example is sufficent for EBLIA to learn all prediction rules. This is so because the constraints are independent of special instantiations of the premise set. For logic circuits, multiple examples are still required.

3.2. Conflict recognition

The EBL impact on this task is through the p-rules, which state explicitly all the minimal assumptions for various predictions. Conflict recognition becomes a simple matching of the p-rule conditions and comparing the p-rule prediction against the observation.

The procedure to determine the conflict sets is shown in table 4. As shown in example 3.2, the p-rules may include pairs of rules that are applicable in a given premise instance but one rule's assumption is subsumed by the other. For conflict recognition we are only interested in minimal conflicts. We discard non-minimal conflicts. Step 3 in GET-CONFLICT (table 4) eliminates p-rules that could lead to non-minimal conflicts.

Table 4. A conflict set generation algorithm: GET-CONFLICTS.

Input: Set of p-rules and a premise.

Output: Collection of all minimal conflicts.

Description:

1. Sort the p-rules in increasing order of their assumption set cardinality.
2. Begin with the first p-rule.
3. If the rule's condition holds and the rule's prediction conflicts with a premise then declare the rule's assumption as a conflict set and remove all remaining rules whose supports are subsumed by the current rule.
4. If there is a next rule then go to 3 else return all conflict sets.

3.3. Candidate generation

For the candidate generation phase of the diagnostic process, the learning component caches associational rules between collections of conflict sets and collections of minimal set covers (hit sets). We call those associations d-rules, formally defined as follows:

Definition 3.2. *A d-rule is a propositional rule,*

$$\mathcal{C} \rightarrow \mathcal{D}$$

associating a collection $\mathcal{C}$ *of all minimal conflict sets for* (*SD*, *COMPS*, *OBS*),

$$\mathcal{C} = \{C_i \mid C_i \subseteq COMPS,\ i = 1, \ldots, n\}$$

with the corresponding collection $\mathcal{D}$ *of all diagnoses for* (*SD*, *COMPS*, *OBS*),

$$\mathcal{D} = \{D_j \mid D_j \subseteq COMPS,\ j = 1, \ldots, m\}$$

where each $D_j \in \mathcal{D}$ *is a minimal hit set (set cover) for* $\mathcal{C}$.

Example 3.5. *For the polybox example, the d-rule that can be learned is as follows:*

$$\{[A1, M1, M2], [A1, A2, M1, M3]\} \rightarrow \{[A1], [M1], [A2, M2], [M2, M3]\}$$

The d-rules are indexed by the collection of minimal conflicts, so that finding all diagnoses using a d-rule takes constant time. After learning, there is no search involved. However, the number of d-rules may grow exponentially with the size of the device, and they can occupy exponential space.

The indexing of the d-rules can be achieved as follows. Each time a new conflict set appears, a counter is incremented and the value of that counter is assigned as an index for that conflict. A collection of conflict sets will be indexed as the ordered set of its conflict set indexes. The procedure, ALL-DIAG, to determine all diagnoses is given in table 5.

Table 5. A candiate generation algorithm: ALL-DIAG.

Input: Collection of minimal conflict sets.

Output: All minimal hit sets.

Description:

1. If the present collection of conflict sets has been seen and a d-rule already exists then return the associated collection of hit sets,
2. else, do:
   (a) apply HS-Tree to the collection of conflict sets,
   (b) record and index new conflict sets,
   (c) assert a d-rule associating conflict indices with hit sets, and
   (d) return the hit sets.

3.4. Summary and discussion

Diagnosis is determined by a minimal set of components with the following property: the assumption that each of these components is abnormal, together with the assumption that all other components are not abnormal, is consistent with the system description and the observation. Computing diagnoses involves three subtasks: prediction, conflict recognition, and candidate generation. Prediction is done by making value inferences for various variables and recording the corresponding *minimal* consistent sets of assumptions, called labels. Recognizing conflicts amounts to comparing observations with predictions and identifying their labels as *nogoods* or conflict sets in the event of inconsistencies. Generating candidates involves finding all minimal set covers, or hit sets, of the collection of conflicts. The worst-case complexity for the task of computing the collection F of all minimal conflicts (nogoods) is exponential in the number of components $|COMPS|$. Given a collection F of subsets of $COMPS$, and a positive integer $K \leq |COMPS|$, the task of determining whether there is a hit set of cardinality less than or equal to K is also known to be NP-complete, even if each set in F has at most two elements (Garey & Johnson, 1979). Consequently, the overall complexity of a model-based diagnosis algorithm based on Reiter (1987) or the GDE system of de Kleer and Williams (1987) is likely to be (if $P \neq NP$) exponential in the number of components.

We propose the use of EBL to acquire two categories of production rules, called p-rules and d-rules. The p-rules associate observations and assumptions with value inferences (predictions) for various variables. Those rules can replace the search for value inferences in the assumption lattice, reducing conflict recognition to the task of matching the condition of a p-rule with the observations and declaring the rule's assumption as a conflict when the value inference in the rule's conclusion differs from the observation. The d-rules associate collections of conflicts with a collection of minimal covers, or all diagnoses. The p-rules may include first-order predicates, while the d-rules are strictly propositional. The d-rules allow the candidate generation to be a deterministic table look-up, involving no search, as the rules are indexed by the collection of conflicts.

3.5. Learning architectures

We consider two learning architectures: EBLIA and EBL(p). They both integrate EBL with MBD and associative diagnosis. EBLIA compiles in advance all p-rules, while EBL(p) compiles those rules while performing the diagnostic task. The candidate generation procedure in both systems is identical. Rules called d-rules are compiled by both systems at diagnosis time, associating conflict sets with minimal candidates. The following sections describe the two architectures in detail.

3.5.1. EBLIA

A block diagram of EBLIA is shown in figure 4. All possible predictions are compiled in advance by the procedure EBLIA-Propagate, in terms of the device model and the variables

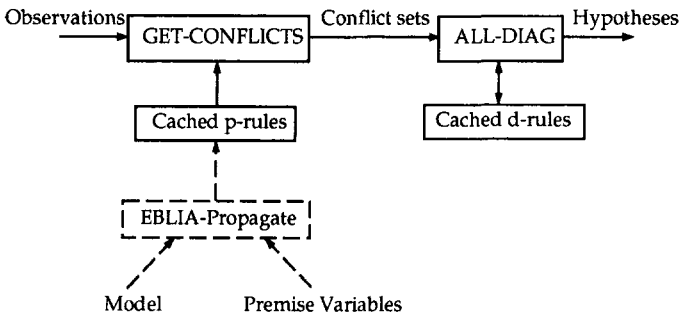

Figure 4. The EBLIA learning architecture for diagnosis.

designated as observable (premise variables). EBLIA-Propagate creates a cache of p-rules that is to be used for the conflict recognition performed by GET-CONFLICTS. Note the dashed lines indicating that the compilation is done in advance—prior to the diagnostic task. Note also that the device model is not subsequently used by EBLIA.

The function of GET-CONFLICTS is to take as input the observations and produce as output the collection of all minimal conflict sets. The collection of conflicts found by GET-CONFLICTS is then input to ALL-DIAG, whose function is to output the corresponding minimal covers (hit sets). If a d-rule exists for the collection of conflicts, then the associated collection of minimal candidates will be produced as the output. Otherwise, the collection of conflicts has not been seen before, and the HS-tree algorithm is used by ALL-DIAG to compute and output all the minimal covers. Then a d-rule is cached for possible use on future problems. Unlike the pre-compilation of p-rules, the compilation of d-rules by ALL-DIAG is performed at diagnosis time. Note that as more d-rules are compiled, EBLIA will operate entirely as an associative system.

EBLIA differs from EBL in that all p-rules are compiled in advance, thus covering the entire observation space, while EBL will cover only the part of that space corresponding to actual examples. The advantage of EBLIA over EBL is that the collection of minimal conflicts found by EBLIA are guaranteed to be the same as those that are model-based, while EBL may miss some minimal conflicts or produce some non-minimal ones. The advantage of EBL over EBLIA is that the space required by the p-rules may be more economical, since rules that do not correspond to some occurring problems will not be learned.

3.5.2. EBL(p)

As pointed out earlier, the p-rules learned by EBL may be incomplete. The conflict sets found based only on the p-rules may fail to detect other existing conflicts. However, for the sake of efficiency, we adopt the closed-world assumption and use negation by failure when recognizing conflicts. If predictions made by the p-rules do not conflict with observations, then no conflict exists. The question that naturally follows is what consequence this assumption has on diagnostic performance. The answer is this: if only a partial set

of conflicts is found but other conflicts are undetected, then the resulting diagnosis will be overgeneral. An overgeneral diagnosis is formally defined as follows.

Definition 3.3. *A set* $\Delta \subseteq COMPS$ *is an* overgeneral *diagnosis for* (*SD*, *COMPS*, OBS) *if there exists a set* $\Delta' \subseteq COMPS$ *such that* Δ' *is a diagnosis and* $\Delta \subset \Delta'$.

Note that an overgeneral diagnosis is *not* a (minimal) diagnosis, according to definition 2.4. Unless the empty set is a diagnosis, the empty set is always an overgeneral diagnosis. An overgeneral diagnosis lacks specificity as to what components must be incriminated in order to provide a hypothesis that is consistent with the model and the observations. This is explained further in example 3.6 below.

Overgeneral diagnoses may not lead to performance errors. This is the case under the following circumstances:

1. If the repair action based on an overgeneral diagnosis "covers" the actual fault.
2. If another generated candidate is a correct diagnosis that subsumes the actual fault (see example 3.6).

In general, however, an overgeneral diagnosis may increase the troubleshooting costs due to the following:

1. Effective probes for discriminating between diagnoses cannot be determined.
2. Troubleshooting may take longer due to inefficient repair decisions.

The following example illustrates the relationships between overgenerality and performance errors.

Example 3.6. Consider the polybox circuit with the input–output observation as shown in figure 2. Assume that existing p-rules lack equation (27). The EBL system based on the closed-world assumption will then conclude that only one conflict set [$A1$, $M1$, $M2$] exists, and it will miss the other conflict set [$M1$, $M3$, $A1$, $A2$]. Based on that erroneous conclusion, three single fault diagnoses namely, [$M1$], [$A1$],[$M2$], will be conjectured. The correct diagnoses based on the two conflicts are [$M1$], [$A1$], [$M2$, $A2$], [$M2$, $A3$]. Therefore the consequence of missing the second conflict, [$M1$, $M3$, $A1$, $A2$], is that one of the proposed diagnoses, [$M2$], is overgeneral. The other two diagnoses, [$M1$], [$A1$], are correct. [$M2$] is overgeneral because suspending the constraint for component $M2$ alone while assuming that all remaining components are behaving correctly will lead to inconsistency with the model and the observation. To see this, we suspend the $M2$ constraint and determine whether the observation and the remaining constraints can derive a contradiction. Propagating the inputs through the multiplier constraints $M1$ and $M3$ yields $X = 6$ and $Z = 6$, respectively. Propagating $Z = 6$ and $G = 12$ through the adder constraint of

$A2$ yields $Y = 6$. But then $X = 6$, $Y = 6$, and $F = 10$ is inconsistent with the $A1$ adder constraint. A contradiction exists, and $[M2]$ cannot be a diagnosis. $[M2]$ is a subset of the (more specific) diagnoses: $[M2, A2]$, $[M2, M3]$. If the actual fault is one of the single-fault hypotheses $A1$ or $M1$, then the overgenerality is of no consequence. On the other hand, if the actual fault is subsumed by one of the double faults $[M2, A2]$ or $[M2, M3]$, then the overgenerality may be harmful. If we repair $M2$ based on the overgeneral diagnosis, the fault will not disappear and the troubleshooting process will need to continue. The correct, more specific diagnoses $[M2, A2]$ and $[M2, M3]$ are more likely to lead to a shorter troubleshooting sequence. For instance, we may measure X and then measure Z, and if both values are as predicted, then we may replace both components $M2$ and $A2$.

One on hand, producing some overgeneral diagnoses on some problems may result in an overall performance that is inferior to MBD. On the other hand, the use of a limited number of p-rules to directly find conflicts is likely to produce more efficient computation, in comparison with MBD. Trading accuracy for efficiency may prove useful in practical applications. To that end, we propose the use of a threshold p as a parameter that determines a lower bound on the performance level of an EBL system that we call EBL(p).

A block diagram of EBL(p) is shown in figure 5. Like EBLIA, the function of EBL(p) is to generate diagnostic hypotheses consistent with the input observations. Unlike EBLIA, EBL(p) compiles the p-rules at diagnosis time. In EBL(p), the p-rules compilation is done by the procedure EBL-Propagate, using the device model and the observations.

EBL-Propagate is turned on and off by a performance evaluation unit, EVAL-PERF, as shown in figure 5. EVAL-PERF does its evaluation task by averaging satisfaction indices received on previous problem-solving. The satisfaction index may be a binary variable: 0 if hypotheses are satisfactory and 1 otherwise. Satisfaction is input by an external unit that could be the human trouble-shooter, or a model-based reasoning system that runs in parallel as a training system.

EVAL-PERF outputs a binary signal to activate or deactivate EBL-Propagate. That signal is determined by comparing the average satisfaction with a threshold, p. EBL-Propagate remains inactive as long as the average satisfaction is greater than the threshold; otherwise it is active.

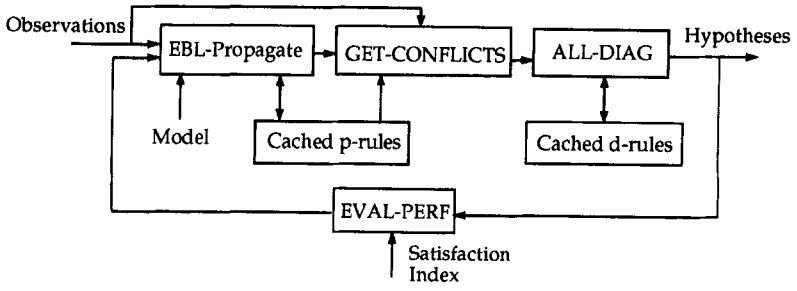

Figure 5. The EBL(p) learning architecture for diagnosis.

Table 6. Procedural description of EBL(p).

Input: Observations.

Output: Hypotheses.

Initialization: {*Sat* is the average satisfaction}
Problem ← 0, *Sat* ← 0.

Description:

1. If *Sat* > *p* then *Activate* ← *No* else *Activate* ← *Yes*
2. If *Activate* = *Yes* then apply EBL-Propagate
3. Apply GET-CONFLICTS
4. Apply ALL-DIAG
5. *Problem* = *Problem* + 1
6. {*Index* is the satisfaction index for output hypotheses}
   Sat = *Sat* + (*Index* − *Sat*)/*Problem*

If the activation signal is off, EBL(p) carries out its diagnostic task as if it were EBLIA. That is, EBL(p) assumes that its p-rules are sufficient to generate all minimal conflicts. If EBL-Propagate is not activated on a sufficient number of examples, the generated hypotheses may be unsatisfactory. If unsatisfactory hypotheses persist, then the activation of EBL-Propagate will occur and continue until average satisfaction again reaches the threshold. When EBL-Propagate is activated, the generated hypotheses are identical to those of MBD. The difference is that caching of p-rules takes place so that prediction can be performed associatively on future problems. Table 6 gives a procedural description of EBL(p).

3.5.3. Evaluation of hypotheses

In standard EBL one learns a sufficient characterization of a concept by generalizing an example instance using a given theory. A problem arises in applying standard EBL to MBD as formulated by Reiter (1987). MBD requires the knowledge of *all* conflict sets. If we miss some minimal conflict sets, then the minimal diagnoses may be overgeneral, as discussed in example 3.6. Overgenerality may adversely affect the troubleshooting performance: effective probes may be overlooked or fruitless repairs may be undertaken.

This raises the need to evaluate generated hypotheses, and how such evaluation is done becomes an important issue. A simple approach to evaluation is to determine (either through an external teacher or at the end of troubleshooting) whether the actual fault is or was one of the generated hypotheses. However, a minimal diagnosis may only be a subset of the actual faulty components. If single faults are dominant, this evaluation seems reasonable, but it will err on multiple faults.

Another possible evaluation method is to test whether diagnoses found by EBL are overgeneral. An overgeneral diagnosis may provide poor guidance to troubleshooting, for example, by suggesting inefficient probes or inadequate repairs (see example 3.6). One way

to test for overgenerality is by constraint-suspension (Davis & Hamscher, 1988), which amounts to verifying whether it is consistent that all components other than the suspects appear to be working correctly. This method is elegant and has the advantage that it can be integrated with explanation-based learning so that learning occurs when constraint suspension uncovers new conflicts. However, constraint suspension can be expensive when the number of suspects is large and when multiple faults are possible.

In a supervised learning mode, MBD can be used by a teacher to provide feedback on whether proposed diagnoses are overgeneral, and the actual faults can be used to test whether proposed diagnoses are sufficiently specific for all practical purposes. We adapt this approach in our experiments, since our aim is to evaluate the accuracy of the generated candidates with regard to how diagnoses will be used in guiding the overall troubleshooting process.

4. Empirical results

We have carried out an empirical study to compare the performance of EBLIA, EBL(p), and MBD. We studied their performance on the polybox (figure 2) and the N-bit parallel adder (figure 6).

Diagnostic problems are generated using a fault simulator module. The number of faults for each problem ranges between 1 and 3, with higher probability assigned for single faults. The locations of faults cover the various components at random. For the N-bit adder, a faulty component is simulated by complementing its normal output. For the polybox, a fault for a multiplier is simulated by subtracting 1 from its normal output, and for an adder by adding 1 to its normal output. The input values are independently and randomly generated from their allowed value set. The value set for the N-bit adder is [0, 1], while for the polybox we chose [2, 3]. The fault simulator produces the output corresponding to the assigned faults and inputs. A diagnostic problem consists of a set of input and output values (called premises) and a set of actual faults. For each device, an experiment consists of feeding 10 series of 100 problems simultaneously to both systems with and without learning. We monitor the values of interesting parameters (such as the cumulative cpu-time) versus the number of problems in each series. We then compute the average value and the standard deviation of those parameters versus the number of problems.

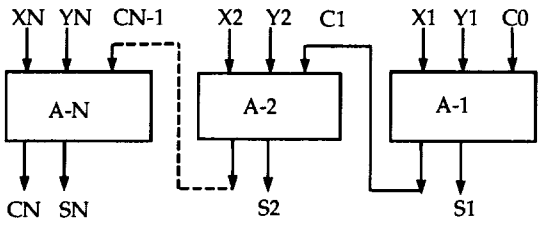

Figure 6. N-bit adder.

4.1. EBLIA

EBLIA learns p-rules from the device description once per experiment, before the diagnosis problems are solved. During actual diagnosis the model is never used. See section 3.5.1 for a more complete description of EBLIA.

Here, we study the cumulative number of d-rules and the cumulative time. The cumulative time for EBLIA includes the initial compilation time. The minimal candidates produced by the implementations of EBLIA and MBD are verified to be the same for every problem to help avoid coding errors in the implementations.

4.1.1. D-Rules

The curves in figure 7 show the number of d-rules learned by EBLIA on the polybox and n-bit adder for n = 1, 2, and 3. The curves are averages over ten problem sets. The values for each curve vary from the average by at most 10%.

There are four d-rules that can be learned in the case of the polybox. EBLIA learns most of them in the first ten problems. After approximately 50 problems, all four d-rules are learnt. On the n-bit adder, the number of d-rules increases monotonically with the number of problems. The increase is more appreciable for the initial problems than for the remaining problems. However, the rate of the increase drops more slowly as the size of the device grows.

4.1.2. Computation time

Figure 8 shows cumulative time curves for EBLIA and MBD on the two-bit adder. The curves are averages over ten problem sets. The values for each curve vary by at most 10%.

In general, the cumulative time curves for EBLIA and MBD rise almost linearly with the number of problems. Table 7 shows key statistics that characterize the curves. The *time per problem* column contains the slopes of lines fitted to the cumulative cpu-time curves. This is a measure of the computation time spent on problems in a diagnostic series. The y-intercepts of the cpu-time lines are given in the *preprocessing* column. The y-intercepts

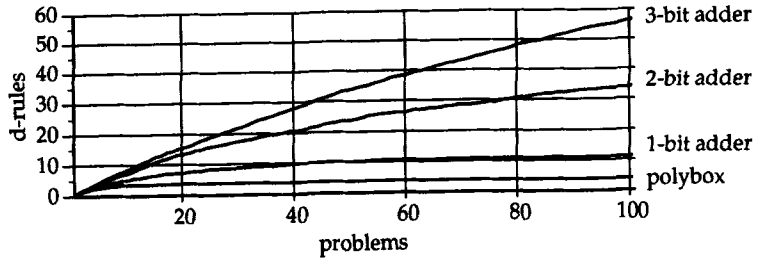

Figure 7. Cumulative number of d-rules for EBLIA.

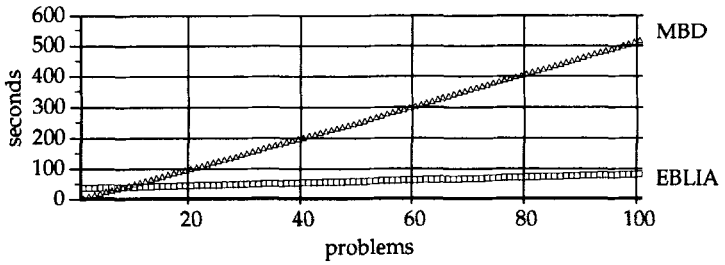

Figure 8. Cumulative time for EBLIA versus MBD on the two-bit adder.

Table 7. Computation time for EBLIA versus MBD.

Problem	Time per problem MBD	Time per problem EBLIA	Preprocessing EBLIA	Crossover
polybox	3.16	.0749	2.96	1.01
1-bit adder	2.16	.104	3.28	1.69
2-bit adder	5.14	.441	35.9	8.14
3-bit adder	12.1	4.96	2126	299

of the cpu-time lines for EBLIA indicate how much time it spends preprocessing a given device prior to diagnosis. The *crossover* column shows the intersections of the MBD and EBLIA time lines. This is a measure of the number of problems that must be solved before EBLIA's overall computation time becomes smaller than MBD's.

On the polybox, the cumulative time curves are within 2% of the average. This indicates that the average is a good representative of the diagnosis problem sets that contributed to it. A linear curve fit on the average times for EBLIA yields the line *seconds* = .0749 × *problems* + 2.96. A linear curve fit on MBD yields the line *seconds* = 3.16 × *problems* − .155. The entries in the polybox lines of table 7 were derived by rounding computed slopes and intercepts to three places. The entry in the crossover column was computed by finding a simultaneous solution of computed linear equations and then rounding.

A comparison of the slopes of the lines on the polybox indicates that the cpu-time per problem for EBLIA is approximately 2% (.0749/3.16) of the time for MBD (see table 7). This indicates that the additional cost of matching against learned rules and the cost of any additional search they engender is negligible compared to the time it takes to search the model for all value inferences and to compute the hit sets. EBLIA's preprocessing time for this device is small, and it improves upon MBD's performance immediately after the first problem.

On the one-bit adder, the slope for EBLIA is approximately 6% of the slope for MBD. The preprocessing time is small, and EBLIA's cumulative time curve crosses over MBD's immediately after the first problem.

On the two-bit adder, the slope for EBLIA is about 9% of the slope for MBD. The initial compilation time produces a relatively significant impact in comparison with the one-bit adder. However, EBLIA produces a net speed-up in comparison with MBD after diagnosing less than ten problems.

On the three-bit adder, the slope of EBLIA is now a significant 41% of the slope of MBD. The preprocessing time required to compile the p-rules for EBLIA is substantial (over 2000 cpu seconds). EBLIA does not produce a performance speed-up in comparison with MBD over the range of 100 problems. The crossover point to obtain the speed-up on this device appears to be about 300 problems.

4.2. EBL(p)

EBL(p) learns d-rules continually, just as in EBLIA. P-rule learning is on in EBL(p) if and only if accuracy is below the threshold percentage p. When learning is off, conflict set recognition is based solely on previously acquired rules. See section 3.5.2 for a description of the EBL(p) architecture.

In this experiment, the threshold parameter p is set to 0.9. A set of candidate diagnoses produced by EBL(p) on a given problem is considered to be *satisfactory* if it is exactly the same as the output of MBD or if the set of faults used to generate the problem is a member of the set of candidates. See section 3.5.3 on hypothesis evaluation for alternative definitions of *satisfactory* and section 3.5.2 for a discussion of the role of hypothesis evaluation in EBL(p). Note that either one of the disjuncts in the requirement used here may be satisifed without satisfying the other (neither implies the other). This satisfaction requirement is strictly weaker than the requirement that EBL(p) and MBD produce identical results. We argue that it is a reasonable way to weaken this requirement. Consider a set of symptoms that produces the conflict sets $[a, b]$ and $[a, c]$. MBD produces the following set of diagnoses in this case: $\{[a], [b, c]\}$. If EBL(p) is unaware of the conflict set $[a, c]$, it will produce the set of candidate diagnoses: $\{[a], [b]\}$. This output would be unsatisfactory according to a strict test requiring EBL(p)'s output to be identical to the output of MBD. But on a problem where a is actually faulted, rather than b and c, this set of candidates is perfectly satisfactory for all practical purposes. For a more realistic case, see example 3.6. An advantage of the definition of satisfactory candidate used here is that it takes into account whether the candidate will successfully guide troubleshooting for the problem at hand, and does not require the candidate to be correct for all possible problems that might have produced similar symptoms.

The measures studied include the cumulative number of d-rules, p-rules, and the cumulative time. In addition, the current accuracy score (the radio of correctly diagnosed problems to the number of problems tried) is recorded after each problem is solved.

4.2.1. D-rules

The number of do-rules learned by EBL(.9) is nearly the same as for EBLIA (see figure 7).

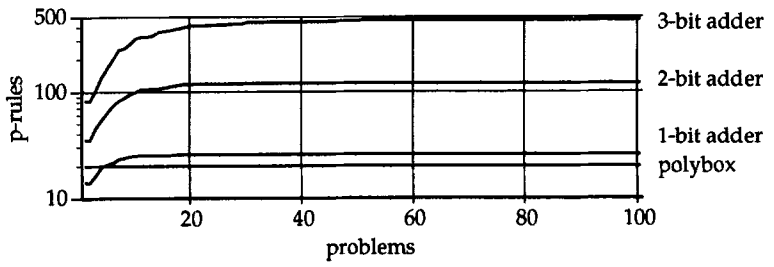

Figure 9. Cumulative number of p-rules for EBL(.9).

4.2.2. P-rules

In the polybox, there is a limited number of p-rules to be learned (20 of them). All of these p-rules are learnt from the first example. So EBL(p) never turns learning on following the first example. In the one-bit adder, the entire set of p-rules that can be learned is 26, so this number serves as an upper bound for the one-bit adder curve in figure 9. This curve shows the cumulative number of p-rules learned averaged over ten runs. The individual curves differ from the average by at most 3%. Nearly all of these p-rules are learned on the first ten examples. Note that the graph in figure 9 uses a logarithmic vertical axis.

Almost all the p-rules learned for the two-bit adder are learnt from the first 10 to 20 problems. The number of p-rules then remains constant around 120 rules. (There are 137 rules that can be learnt.)

4.2.3. Accuracy

Accuracy curves are shown in figure 10. For the polybox, the accuracy remains constant at 1.0. For the n-bit adders, the accuracy curves show a sharp dip at the beginning, where accuracy falls below the 0.9 threshold; then they rise steadily due to learning. Accuracy

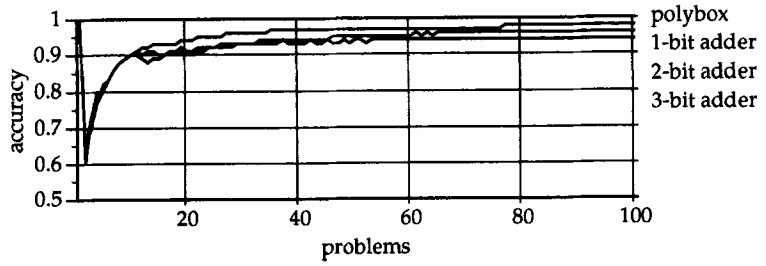

Figure 10. Accuracy of EBL(.9).

reaches the threshold value on its way up after about 10 problems. Except for some rippling around the threshold for the range up to about 20 problems, the accuracy remains above the threshold and no further learning is needed. The top (most accurate) curve in the figure is the polybox, then the one-bit adder, the two-bit adder; the curve that represents the lowest accuracy is for the three-bit adder. In general, the larger the circuit, the lower the accuracy. Note, however, that all three curves end up significantly above the required 90% accuracy mark.

4.2.4. Computation time

In the polybox, the cumulative time of MBD and EBL(.9) grows linearly with the number of problems, but the slope of EBL(.9) is only about 3% of that of MBD. This is nearly the same as the result obtained by EBLIA.

On the n-bit adders, the cumulative time rises almost linearly for MBD, but displays a "knee effect" for EBL(.9) (see figures 11 to 13). The figures show average cumulative time curves. The knee position is located at the point where learning ends and associative diagnosis takes over.

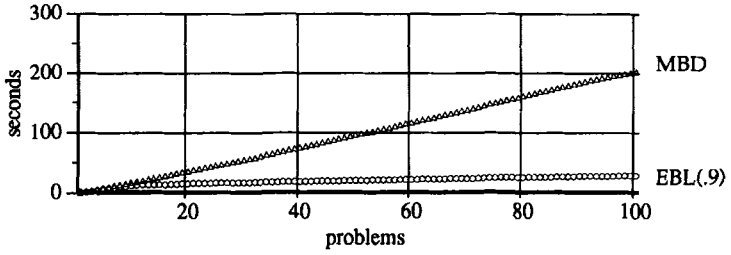

Figure 11. Cumulative time for EBL(.9) versus MBD on the one-bit adder.

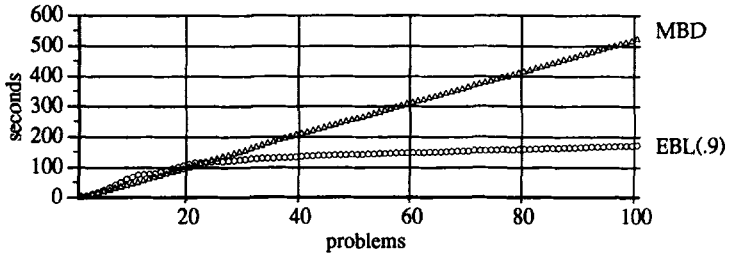

Figure 12. Cumulative time for EBL(.9) versus MBD on the two-bit adder.

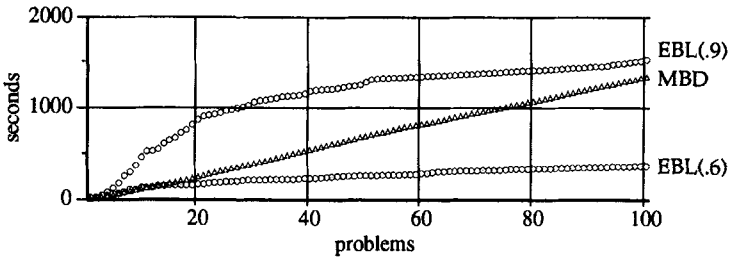

Figure 13. Cumulative time for MBD and EBL(.9) and EBL(.6) on the three-bit adder.

4.2.5. On the effects of varying the required accuracy

In order to explore potential tradeoffs between accuracy and efficiency, we experimented with EBL(p) using two thresholds, $p = 0.6$ and 0.9. The higher the threshold the more p- and d-rules will be acquired. EBL(p) will learn only enough rules to meet the requirement that the average accuracy remains above the threshold. See figures 15 and 16.

Figure 14 compares the accuracy of EBL(.6) against the accuracy of EBL(.9). For EBL(.9), the accuracy curve shows a sharp dip at the beginning, where accuracy falls below the 0.9 threshold; then it rises steadily due to the effect of learning. On average, the accuracy of EBL(.6) remains above the threshold and learning after the first problem is rarely reinvoked.

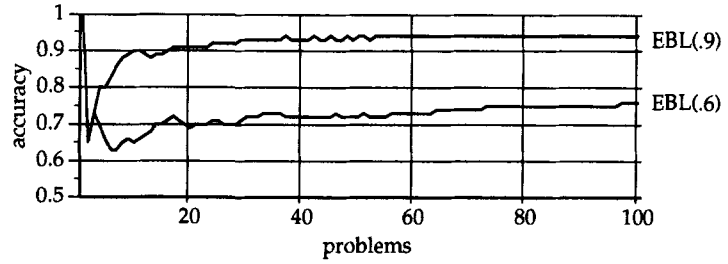

Figure 14. Accuracy for EBL(.9) versus EBL(.6) on the three-bit adder.

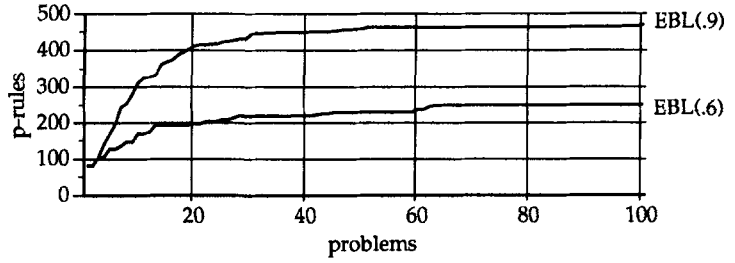

Figure 15. Cumulative number of p-rules for EBL(.9) and EBL(.6) on the three-bit adder.

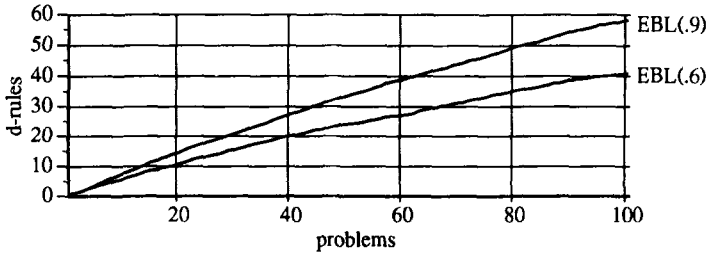

Figure 16. Cumulative number of d-rules for EBL(.9) and EBL(.6) on the three-bit adder.

Figure 15 shows the number of p-rules learnt by EBL(p) on the three-bit adder. EBL(.9) learns approximately half of the 918 p-rules learned by EBLIA and approximately 64% of the 735 p-rules learned by EBL(1). As the threshold drops from 0.9 to 0.6, the number of rules drops again by almost 50%.

For EBL(.9), the cumulative time rises relatively quickly for the first 20 problems. This is the range where learning is most frequent. For later problems the average time taken by the associative mode varies rather widely. Due to the large number of rules, p-rule matching costs are significant. However, after learning, the rate of time increase is much flatter than for MBD. For EBL(.6), speedup effects are evident compared to MBD within 10 to 20 problems (see figure 13).

4.3. Summary

Here we pause to briefly summarize the empirical results prior to a full discussion. The most important results of the empirical studies of EBLIA are as follows:

- For a fixed device, EBLIA and MBD both compute diagnoses in time linear in the number of diagnosis problems. They each spend a constant time per problem. EBLIA is significantly faster than MBD on each problem.
- For the devices studied, there is often a crossover point. EBLIA takes more time to solve initial problems prior to the crossover point, but after a certain number of problems EBLIA achieves speedup over MBD.
- The time spent by EBLIA preprocessing the device model is not significant on small devices, but it grows rapidly with the size of the device.
- The number of p-rules computed by EBLIA prior to diagnostic problem-solving and the number of d-rules computed during diagnosis grow rapidly with the size of the device.

The most important results of the empirical studies of EBL(p) are as follows:

- The cumulative cpu-time curves for EBL(p) are nonlinear. They exhibit a "knee effect;" initially they rise with a slope higher than the slope of MBD and then they level off.

- EBL(p) cpu-time curves tend to crossover MBD at a point that depends on the accuracy parameter and the size of the device. Speedup over MBD is obtained sooner for lower accuracies and smaller devices.
- The accuracy drops rapidly on initial problems then rises when learning is activated until it crosses over the required accuracy. It rarely drops below the threshold on subsequent problems. The final accuracy scores are significantly higher than the required accuracy. On the devices studied with random faults, for a given accuracy requirement, EBL(p) was slightly less accurate as the size of the device increased.
- Significantly fewer d-rules and substantially fewer p-rules were learned by EBL(p) given lower accuracy requirements.

5. Discussion

5.1. EBLIA

Here, we discuss the empirical results of section 4.1 in an effort to draw some general conclusions. The first general conclusion that can be drawn is that EBLIA achieves net speedup over MBD—even when precomputation is included—given a sufficiently large number of diagnosis problems. But the question is how much precomputation is required and how many problems must be solved before the costs of the precompilation are repaid.

For the purpose of analysis, let us introduce the following notations. Let C_{MBD} be the average cost per problem for MBD. Let C_{ASS} be the average cost per problem for the associational problem solver of EBLIA. This is the cost due to matching and HS-Tree. The preprocessing cost incurred by EBLIA for compiling the p-rules is denoted by C_{IA}. Based on the empirical results, we can fairly represent the cumulative (average) cost versus the number of problems for EBLIA and MBD by linear relations, as depicted in figure 17.

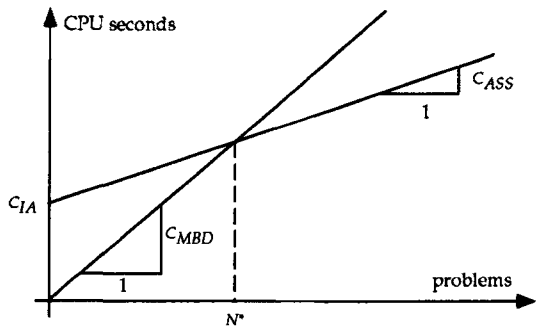

Figure 17. The crossover point for EBLIA versus MBD.

The cost of solving N problems by MBD is $C_{MBD} \times N$. The cost of solving N problems by EBLIA is $C_{IA} + C_{ASS} \times N$. The crossover point N^* is the number of problems for which the cumulative time of EBLIA and MBD is the same. That is,

$$N^* = \left\lceil \frac{C_{IA}}{(C_{MBD} - C_{ASS})} \right\rceil$$

We know that C_{MBD} increases exponentially with the size of the device. As the number of components increases,

1. C_{IA} increases exponentially, and
2. C_{ASS} increases as a result (also exponentially).

Table 7 in section 4.1 provides numerical values for those parameters for the N-bit adder for increasing N. The results lead us to the following conclusion. EBLIA is feasible and will achieve net speedup over MBD for small devices. As the number of components increases, EBLIA can become infeasible. Even if it is feasible, EBLIA may be worse than MBD over a fixed number of problems. The main reason for this is that the number of p-rules that must be precompiled grows exponentially with the size of the device. As a consequence, conflict set recognition using the p-rules becomes more costly. This is in part due to the non-minimality problem pointed out in section 3.2. In addition, the space used to store the rules grows exponentially.

One further observation with respect to the empirical results of section 4.1 is the following. The rate of increase of d-rules with the number of problems tends to increase as the number of components increases (see figure 7). This is due to the random nature of the fault generator and the size of the device. It is more likely to see new collections of conflicts on new problems when diagnosing larger devices. As a consequence, on the three-bit adder the learning of d-rules does not pay off as much as it did on smaller devices over the range of 100 problems since almost 60 rules are learnt in that range.

5.2. EBL(p)

Initially, EBL(p) is in learning mode, and the cpu-time per problem is comparable to MBD. As soon as the learning phase ends, the slope of the cpu-time curve changes and the curve gets flatter due to the speedup provided by the associative operating mode. This is why EBLIA exhibits a "knee effect" (e.g., see figure 12). For EBL(.6) on the three-bit adder, learning is rarely invoked after the first few problems, so the number of rules acquired is much less than that of EBL(.9). This reduces the matching effort and space requirement so that EBL(.6) quickly achieves a net speedup compared to MBD.

Learning provides speedup only if the learned rules have high likelihood of being applicable and irredundant. The results of sections 4.2 and 4.2.5 indicate a utility problem in learning for MBD. If the problems have faults that are randomly distributed and uncorrelated, then the likelihood that a d-rule is going to be useful for the next problem decreases as the number of components increases. This is so because the number of conflict sets

(and hence their possible combinations) increases exponentially with the number of components. If the faults that occur in practice cover all possible minimal candidates, and we are required to be complete (i.e., no erroneous diagnoses can be tolerated), then the best we can do is to learn all possible p-rules (and d-rules). This will degenerate to EBLIA, and no overall speedup will be obtained except on small devices.

EBL(p) relaxes the requirement for perfect accuracy. It is biased to learning p-rules that are necessary to meet bounds on performance. The threshold p reduces the number of p-rules that need to be learnt. As p decreases, the number of p-rules decreases, and as a result also the number of d-rules.

Table 8 summarizes relevant data from section 4.2.5. The table shows that the reduction in rules required for decreasing accuracy is more substantial for p-rules than for d-rules.

With respect to d-rules, the EBLIA results (shown in figure 7) carry over to EBL(.9) with little change. When the accuracy threshold drops from .9 to .6, the number of d-rules (see figure 16) acquired on the three-bit adder is roughly 70% of the number required for $p = .9$. In rough terms, EBL(.9) learns three d-rules every five problems, while EBL(.6) learns two d-rules every five problems. This means that EBL(.6)'s abstraction of the diagnostic space is less detailed than that of EBL(.9). In other words, classification of the diagnosis space for EBL(.6) is at a coarser level than that for EBL(.9).

With respect to p-rules, EBL(.9) learns 64% of the total number of p-rules which would have been learned by EBL(1). This means that approximately 64% of the p-rules contributes to 94% of perfect performance. As the accuracy threshold drops from .9 to .6, the number of rules drops again by almost 50%. In general, even though the problems are randomly distributed, a relatively small set of p-rules covers the majority of the problems and contributes the majority of the accuracy.

The results shown in table 8 provide a good indication of "transfer." They measure how often rules learned by both forms of learning in EBL(p) are useful in subsequent situations. For a given accuracy, smaller numbers of learned rules indicate more transfer. One reason there is more transfer for p-rules than for d-rules is that they encode different kinds of knowledge. Another possible reason is that the p-rules are first-order predicate calculus rules, and p-rule learning benefits from explanation-based generalization. In contrast, d-rules are propositional, and d-rule learning is a form of rote-learning.

Table 8. Rules learned versus accuracy on the three-bit adder.

Method	P-rules	D-rules	Accuracy
EBL(1)	735	60	1.0
EBL(.9)	470	58	.94
EBL(.6)	250	41	.76

6. Related work

Preliminary versions of the results reported here appeared in abridged form in two conference papers (see El Fattah & O'Rorke, 1991a, 1992). In El Fattah and O'Rorke (1991a), the diagnosis system used constraint suspension testing to double-check diagnoses. The system learned immediately from new constraint violations that occurred during the checking. But as the size of the device increased, the cost of constraint suspension testing quickly overcame the benefits of EBL.

Discussions with Oren Etzioni of his work on an alternative to EBL "learning while doing" for planning (Etzioni, 1990) led us to consider the merits of "learning in advance" for diagnosis. When the associational rules were all learned in advance and the system operated in associational mode at run time, substantial speedup occurred on the small circuits we initially studied. Unfortunately, more recent studies of parameterized devices (reported in this article) indicate that learning in advance is infeasible for MBD of large devices.

In El Fattah and O'Rorke (1992), we allowed the EBL system to make errors as long as the percentage remained below a pre-assigned threshold. Instead of testing proposed diagnoses against the model, this diagnosis system tests against reality (or an external "teacher"). Results of the present article include averages, over ten experimental runs, of important measurements of this method's performance.

Other works have explored the use of EBL for MBD (see Resnick, 1989; Zercher, 1988, Koseki, 1989), but these works are limited to single-fault diagnosis. Interest in the proposal of embedding compilation in problem-solving environments is evident in recent works by de Kleer (1990), El Fattah and O'Rorke (1991c, 1991b), and Friedrich, Gottlob, and Nejdl (1990).

6.1. Knowledge compilation

A controversy surrounding knowledge compilation is relevant to the present work. It is our view that the controversy is an indication that many issues are not yet understood (see Goel, 1991).

Davis (1989) has argued strongly against efforts to compile causal models into associational rules. According to him, turning a model into a set of rules is

> . . . **misguided**, if rule is taken to mean conditional statement, because form alone is not the source of speed.

We agree with Davis that form alone is not the source of speed. EBL has been demonstrated to provide speedup in numerous problem-solving situations (for example, see O'Rorke, 1989). But there are factors that diminish the improvement offered by EBL—for example, the utility problem (Minton, 1988). If too many useless rules are learned, EBL may degrade problem-solving performance instead of improving it.

We consider the question of whether to transform knowledge associated with models using EBL to be an empirical question. Our results indicate that, for sufficiently small

devices, it makes sense to convert the entire model into rules. See the results in section 4.1 on EBLIA. As the number of components increases, this approach becomes less feasible, but it still makes sense if we are willing to invest substantial computation up front, prior to fault diagnosis, and if quick response at diagnosis time is important and a large memory is available at diagnosis time. In addition, in situations where diagnostic tools are mass produced, the initial computations can be amortized over problems encountered by each tool. In this case, one can divide the preprocessing cost C_{IA} in figure 17 by the number of diagnostic systems produced.

According to Davis (1989), turning a model into a set of rules is

> . . . **impossible**, if rule is taken to mean empirical association and the causal model is strictly deterministic, because empirical information is (by definition) available only from observing nature.

We believe it is possible to use knowledge from first principles and from observing actual occurrences of faults to compile empirical associations. This seems to be what humans do to become experts. We claim that EBL(p) automates the acquisition of some empirical associations, since it is driven by observations of actual faults.

In a reply to Davis in defence of compilation, Keller (1989) has argued that empirical studies are needed to form useful theories about the utility of knowledge transformations. Keller (1991) reports on the results of an initial feasibility study in the context of NASA's Hubble Space Telescope. We agree with Keller's position that the tradeoffs inherent in knowledge compilation for model-based reasoning merit empiricial study. We also agree with his view that it is worthwhile to consider how to best integrate model-based reasoning and techniques such as EBL. We offer the present study as a contribution to work on methods for integrating model-based diagnosis and knowledge compilation.

6.2. Clause management systems

In Reiter and de Kleer (1987), a problem-solving environment consists of a domain-dependent reasoner and a domain-independent Clause Management System (CMS). The reasoner can query the CMS about the set of minimal support clauses for a given propositional clause. The set of minimal supports for a query can be computed trivially from the set of prime implicates of the CMS database. Two approaches are proposed: the interpreted versus the compiled. This is somewhat similar to MBD versus EBLIA. The issue of interpreted versus compiled in the reasoner-CMS architecture is discussed by Kean and Tsiknis (1990), who claim that the compiled approach is "more suitable for CMS in both question-answering and explanation-based problem solving environments."

6.3. Focusing diagnosis

A "focused" MBD system was introduced by de Kleer (1991), based on the idea of focusing the reasoning on "what will ultimately be the most probably diagnosis." The distinction

between us and de Kleer is that while he recomputes for each problem predictions that focus on the most probable candidates, we cache all p-rules. Like de Kleer's, our approach is also a means of limiting the predictions that need to be made. But our approach could benefit from probabilistic focusing, and we view this as an important topic for future work.

6.4. Quality of learning

Van de Velde (1988) discusses three criteria to evaluate the quality of learning problem-solving associations: correctness, effectiveness, and level of abstraction. In general, the higher the correctness, the lower the effectiveness and the level of abstraction. These criteria determine the bias of the learning system. According to Van de Velde, the bias will be dictated by three characteristics of the learning situation: criticality, diversity, and background knowledge:

(1) learning in non-diverse environments may be biased towards effective associations,
(2) learning in critical enviornments must be biased toward correct associations,
(3) with background knowledge, learning may be biased towards abstract associations.

Our EBL(p) system for MBD is formulated to strike a balance between the first two biases. The third bias is an integral part of our EBL/MBD framework.

7. Conclusions

We described two general approaches integrating EBL with model-based and associative diagnosis. The first approach is a form of "learning in advance." Learning occurs in a training phase prior to diagnosis of examples of faults. The second approach is a form of "learning while doing." Learning takes place as faults are diagnosed. In both approaches, rules called p-rules associate observations and assumptions with predictions, and d-rules associate conflict-sets with minimal diagnoses. In the first approach, implemented in a system called EBLIA, all p-rules are compiled in advance. In the second approach, implemented in a system called EBL(p), the p-rules are compiled at diagnosis time. D-rules are compiled at diagnosis time in both approaches.

EBLIA avoids a problem with the straightforward application of EBL to diagnosis. The obvious approach to integrating EBL and diagnostic hybrids is to transform the results of model-based diagnosis into associations between observations, constraint violations, and diagnoses. But if EBL is used to learn p-rules and d-rules while doing MBD, the resulting rules can suggest incorrect diagnoses. If too few examples have been observed, the system may not have encountered relevant constraint violations. As a result, the rules may suggest diagnoses that are too general, missing faulty components. This problem can be solved by doing constraint suspension testing of the diagnoses suggested by the rules and by learning when this leads to unforseen constraint violations. Unfortunately, this form of "doublechecking" is prohibitively expensive. Its cost overwhelms the speedup provided by EBL on large devices.

EBLIA solves this problem by eliminating the need to double-check proposed diagnoses. It also eliminates the need for diagnostic examples altogether, since it considers all possible constraint violations in advance. EBLIA analyzes the model and compiles it into abstract constraints between inputs and outputs.

EBL(p) allows for relaxation of the requirement that the diagnostic system perform with perfect accuracy. It assumes that existing associational rules are applicable to new situations, analyzing and learning only when this assumption leads to unacceptable errors. When too many errors have been made, EBL is activated and new rules are acquired until the diagnostic accuracy rises above the given threshold percentage p. Constraint suspension testing is not performed. Instead an external agent is charged with the task of verifying that the proposed diagnoses are correct. If not, then an error is counted against the diagnosis system, lowering its running accuracy score.

We presented results of computational experiments on the polybox and on digital logic devices with increasing number of components. The experiments were carried out for independent randomly distributed faults spanning all components. We allowed multiple faults of up to three components.

The experimental results show that EBLIA is subject to the exponential growth associated with MBD. As the size of the device grows, EBLIA incurs a large time cost in advance of diagnosis and a large space cost at diagnosis time. The results show that if costs are measured purely in terms of cpu-time (without regard for such variables as the utility of correct diagnoses) the number of problems that must be diagnosed before the crossover point where EBLIA intersects MBD soon becomes large. With more powerful computers and more massive memories becoming available, this approach may be warranted for important diagnosis problems. When feasible, EBLIA is the preferred alternative at diagnosis time, since it is essentially an extremely fast lookup operation.

EBL(p) provides speedup over MBD if we are willing to tolerate some errors. EBL(p) alternates between a relatively high cost per problem (incurred when MBD and learning are turned on) and a low cost per problem (incurred by associational rules). If relatively few rules cover many examples with high accuracy, the lower cost dominates the higher cost, so that EBL(p) outperforms MBD after a number of examples. The lower the required accuracy, the sooner this crossover point occurs. In our tests, even though faults were distributed randomly, subsets of the possible diagnostic rules provided high degrees of accuracy. In realistic situations, observed faults will tend to form clusters in the space of possible faults. Novel faults will occur less frequently than previously learned faults. The EBL(p) architecture takes advantage of this fact to improve efficiency while making acceptable sacrifices in accuracy.

Acknowledgments

We thank Pat Langley, Deepak Kulkarni, and other researchers at NASA's Ames Research Center for encouraging us to look at tradeoffs reducing costs by sacrificing performance. Discussions with them about the need for resource-limited computing at NASA helped motivate our work on learning approximate diagnoses. Thanks also to Oren Etzioni for discussions of "learning in advance" in the context of planning. Brian Williams provided

helpful comments in the early stages of this work. Steven Minton and three anonymous reviewers provided encouraging feedback and many excellent suggestions for improving this article. We also thank the AI graduate students and faculty at UCI for serving as helpful critics. This research was supported in part by National Science Foundation grant number IRI-8813048 and by grant number 90-117 from Douglas Aircraft Company and the University of California Microelectronics and Computer Research Opportunities Program.

Notes

1. An unintentional interaction between the tree-pruning heuristics was responsible for a flaw in Reiter's algorithm, subsequently corrected by Greiner, Smith, and Wilkerson (1989).
2. The search effort is exponential in the number of components in the worst case. This is due to the fact that predictions must be made for *all* possible environments (sets of assumptions).

References

Davis, R. (1989). Form and content in a model-based reasoning. In E. Searl (Ed.), *Proceedings of the Workshop on Model Based Reasoning* (pp. 11-27). Detroit, MI: Boeing Computer Services.

Davis, R., & Hamscher, W. (1988). Model-based reasoning: Troubleshooting. In H.E. Shrobe (Ed.), *Exploring Artificial Intelligence*, chapter 8 (pp. 297-346). San Mateo, CA: Morgan Kaufmann.

de Kleer, J. (1976). *Local methods for localizing faults in electronic circuits* (AI Memo 394). Cambridge, MA: MIT Artificial Intelligence Laboratory.

de Kleer, J. (1986). Problem solving with the ATMS. *Artificial Intelligence*, 28, 197-224.

de Kleer, J. (1990). Exploiting locality in a TMS. In *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 264-271). Boston, MA: AAAI Press/The MIT Press.

de Kleer, J. (1991). Focusing on probable diagnoses. In *Proceedings of the Ninth National Conference on Artificial Intelligence* (pp. 842-848). Anaheim, CA: AAAI Press/The MIT Press.

de Kleer, J., Mackworth, A., & Reiter, R. (1992). Characterizing diagnoses and systems. *Artificial Intelligence*, 56, 197-222.

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

de Velde, W.V. (1988). Quality of learning. In *Proceedings of the Eighth European Conference on Artificial Intelligence* (pp. 408-413). London: Pitman.

DeJong, G.F., & Mooney, R. (1986). Explanation-based learning: An alternative view. *Machine Learning*, 1(2), 145-176.

El Fattah, Y., & O'Rorke, P. (1991a). Learning multiple fault diagnosis. In *Proceedings of the Seventh IEEE Conference on Artificial Intelligence Applications* (pp. 235-239). Los Alamitos, CA: IEEE Computer Society Press.

El Fattah, Y., & O'Rorke, P. (1991b). On tractability and learning in model based diagnosis. In *Proceedings of the Workshop on Model Based Reasoning*. Anaheim, CA: AAAI.

El Fattah, Y., & O'Rorke, P. (1991c). The role of compilation in constraint based reasoning. In *Working Notes of the AAAI Spring Symposium on Constraint Based Reasoning* (pp. 225-241). Stanford, CA: AAAI.

El Fattah, Y., & O'Rorke, P. (1992). Learning approximate diagnosis. In *Proceedings of the Eighth IEEE Conference on Artificial Intelligence Applications* (pp. 150-156). Los Alamitos, CA: IEEE Computer Society Press.

Etzioni, O. (1990). Why PRODIGY/EBL works. In *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 916-922). Menlo Park, CA: AAAI Press/The MIT Press.

Feigenbaum, E.A. (1979). Themes and case studies of knowledge engineering. In D. Michie (Ed.), *Expert systems in the micro electronic age* (pp. 3-25). Edinburgh: Edinburgh University Press.

Friedrich, G., Gottlob, G., & Nejdl, W. (1990). Generating efficient diagnostic procedures from model-based knowledge using logic programming techniques. *Computers Mathematical Applications*, 20(9/10), 57-72.

Garey, M., & Johnson, D. (1979). *Computers and Tractability.* New York: W.H. Freeman and Company.

Goel, A.K. (1991). Knowledge compilation: A symposium. *IEEE Expert*, 6(2), 71–93.

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

Kean, A., & Tsiknis, G. (1990). An incremental method for generating prime implicants/implicates. *Journal of Symbolic Computation*, 9, 185–206.

Keller, R.M. (1990). In defense of compilation: A response to Davis' "Form and content in model-based reasoning." In E. Scarl, (Ed.), *Proceedings of the Workshop on Model Based Reasoning* (pp. 22–31). Boston, MA: Boeing Computer Services.

Keller, R.M. (1991). Applying knowledge compilation techniques to model-based reasoning. *IEEE Expert*, 6(2), 82–87.

Koseki, Y. (1989). Experience learning in model-based diagnosis. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 1356–1362). Detroit, MI: Morgan Kaufmann.

Minton, S. (1988). Quantitative results concerning the utility of explanation-based learning. In *Proceedings of the Seventh National Conference on Artificial Intelligence* (pp. 564–569). St. Paul, MN: Morgan Kaufmann.

Mitchell, T.M., Keller, R.M., & Kedar-Cabelli, S.T. (1986). Explanation-based generalization: A unifying view. *Machine Learning*, 1(1), 47–80.

O'Rorke, P. (1989). LT revisited: Explanation-based learning and the logic of Principia Mathematica. *Machine Learning*, 4(2), 117–159.

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

Reiter, R., & de Kleer, J. (1987). Foundations of assumption-based truth maintenance systems. In *Proceedings of the Sixth National Conference on Artificial Intelligence* (pp. 183–188). Seattle, WA: Morgan Kaufmann.

Resnick, P. (1989). Generalizing on multiple grounds: Performance learning in model-based troubleshooting (Technical Report AI—TR 1052). Cambridge, MA: MIT Artificial Intelligence Laboratory.

Zercher, K. (1988). Model-based learning of rules for error diagnosis. In W. Hoeppner (Ed.), *Proceedings of the German Workshop on Artificial Intelligence* (pp. 196–205). Berlin: Springer Verlag.

Received February 20, 1992
Accepted October 8, 1992
Final Manuscript October 28, 1992