Explanation-Based Learning for Diagnosis

Yousri El Fattah
fattah@ics.uci.edu
Paul O'Rorke
ororke@ics.uci.edu

Technical Report 92–21
(supersedes TR 91-06 and 91-64)

February 20, 1992

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.

Contents

Explanation-Based Learning for Diagnosis

Yousri El Fattah and Paul O'Rorke

Department of Information and Computer Science
University of California, Irvine, CA 92717-3425

Electronic Mail: fattah@ics.uci.edu, ororke@ics.uci.edu
Phone: (714) 856-6226, (714) 854-2894
Fax: (714) 856-4056

Running head: *Learning for diagnosis*

Keywords: Explanation-based learning, model-based reasoning, rule-based expert systems, diagnosis

Abstract

Diagnostic expert systems constructed using traditional knowledge-engineering techniques identify malfunctioning components using rules that associate symptoms with diagnoses. Model-based diagnosis (MBD) systems use models of devices to find faults given observations of abnormal behavior. These approaches to diagnosis are complementary. We consider hybrid diagnosis systems that include both associational and model-based diagnostic components. We present results on explanation-based learning (EBL) methods aimed at improving the performance of hybrid diagnostic problem solvers. We describe two architectures called STATIC and EBL(p). STATIC pre-compiles models into associations, and at run-time the diagnostic system is purely associational. In EBL(p), the run-time diagnosis system is a hybrid: learned associational rules are preferred but the MBD component is activated whenever the performance falls below a threshold p. We present results of empirical studies comparing MBD without learning versus STATIC and EBL(p). The main conclusions are as follows. STATIC 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.

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 expert's 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, while associational systems are relatively efficient. We consider hybrid diagnosis systems that include both associational and model-based components in this paper.

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, e.g., for multiple 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 the paper 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, STATIC 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

The MBD system is based on the theory of diagnosis given by Reiter (1987) and emulates the GDE system of de Kleer and Williams (1987). Diagnosis, as shown in figure 1, is a 3-step process:

Prediction by propagating observations of premise variables 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

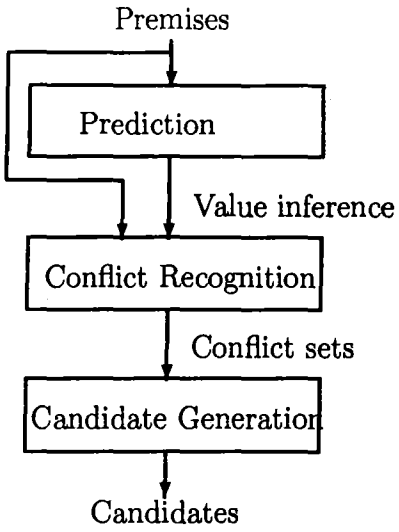

Figure 1. Model-Based Diagnosis

2.1 Prediction

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 is described as follows (see table 1). We record for each inference an assumption and a dependency label. 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 are conjunctions of the *ab* literals. The condition for a clause represents the minimal assumptions under which the prediction is valid. They are the same as ATMS labels (de Kleer & Williams, 1987).

Table 1. The Prediction Algorithm **Propagate**

Given: Premises.
Initialize: Value inferences as premises with empty assumption and dependency labels.

1. *Change* $\leftarrow$ *false*
2. For each value inference rule $X \rightarrow Y$ do:
   (a) For each tuple of X whose dependencies do not include Y do:
      i. Determine value inference for Y
      ii. Propagate the assumptions and dependencies of X to determine Y's labels
      iii. If the value inference is not subsumed by previous inferences then do:
         A. Assert the current inference
         B. Retract existing inferences subsumed by the current inference
         C. *Change* $\leftarrow$ *true*
3. If *Change* then go to 2.

Example 2.1 Consider the polybox circuit depicted in figure 2 with the input-output (I/O) premises:

$$A = 3,\ B = 2,\ C = 2,\ D = 3,\ E = 3,\ F = 10,\ G = 12 \tag{1}$$

In this circuit, $M1, M2$ and $M3$ are multipliers, while $A1$, and $A2$ are adders. The prediction process asserts the following clauses for the outputs:

$$\neg ab(M1) \wedge \neg ab(M2) \wedge \neg ab(A1) \rightarrow F = 12 \tag{2}$$
$$\neg ab(M2) \wedge \neg ab(M3) \neg ab(A2) \rightarrow G = 12 \tag{3}$$
$$\neg ab(M1) \wedge \neg ab(M3) \wedge \neg ab(A1) \neg ab(A2) \rightarrow F = 12 \tag{4}$$
$$\neg ab(M1) \wedge \neg ab(M3) \wedge \neg ab(A1) \neg ab(A2) \rightarrow G = 10 \tag{5}$$

Here, $ab(.)$ is the abnormality predicate, after Reiter (1987). Eqn. 2 says that under the assumption that none of the components $M1, M2$, and $A1$ are abnormal, the output F is predicted to be 12. Note that constraint propagation goes forward and backward. Eqs. 4 & 5 result from propagating the output value backward through the adder constraint. Note also that the I/O premises are absent in the conditions of the predictions. The predictions are all made in the context of those premises.

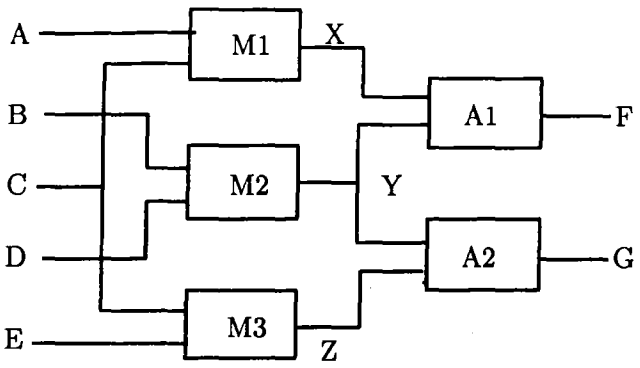

Figure 2. The Polybox Circuit

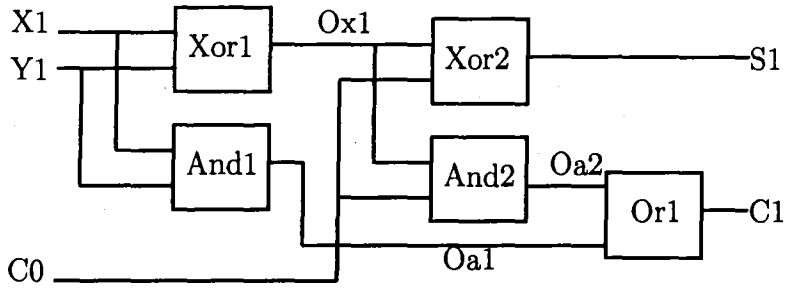

Figure 3. A Full Adder

Example 2.2 Consider the 1-bit adder circuit in figure 3, with the input-output:

$$X1 = 0,\ Y1 = 0,\ C0 = 0,\ S1 = 1,\ C1 = 0 \tag{6}$$

The prediction process asserts the following clauses for the outputs:

$$\neg ab(Xor1) \wedge \neg ab(Xor2) \rightarrow S1 = 0 \tag{7}$$

$$\neg ab(And1) \wedge \neg ab(And2) \wedge \neg ab(Or1) \rightarrow C1 = 0 \tag{8}$$

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. If there is a discrepancy, then the support set of the prediction inference is declared as a conflict set.

Example 2.3 The polybox circuit with inputs and outputs as given in Example 2.1 results in two conflicts:

$$ab(M1) \vee ab(M2) \vee ab(A1) \tag{9}$$
$$ab(M1) \vee ab(M3) \vee ab(A1) \vee ab(A2) \tag{10}$$

Example 2.4 The 1-bit adder with inputs and outputs as given in Example 2.2 results in one conflict set:

$$ab(Xor1) \vee ab(Xor2) \tag{11}$$

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). In our implementation, we do not consider two of Reiter's heuristics: 1) the "reusing node labels" heuristic, and 2) the tree-pruning heuristic 3-iii .[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 2.5 The polybox circuit with the two conflicts of example 2.3 results in four minimal candidates:

$$ab(M1) \tag{12}$$
$$ab(A1) \tag{13}$$
$$ab(M2) \wedge ab(M3) \tag{14}$$
$$ab(M2) \wedge ab(A2) \tag{15}$$

Example 2.6 The 1-bit adder with the one conflict of example 2.4 results in two minimal candidates:

$$ab(Xor1) \tag{16}$$
$$ab(Xor2) \tag{17}$$

[1] An unintentional interaction between the tree-pruning heuristics was responsible for a flaw in Reiter's algorithm, subsequently corrected by Greiner, Smith & Wilkerson (1989).

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* $\rightarrow$ *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

The procedure Propagate, table 1, must be applied by MBD for every new problem, even if a problem has been seen before. 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. A p-rule has the form: $P \wedge A \wedge C \rightarrow V$, where $P, and A$ are premise and assumption tuples, and V is a variable whose predicted value may depend on P via condition C. The p-rules may then replace the propagation procedure performed by Propagate. This has the following benefits:

1. The problem of finding predictions becomes backtrack-free.[2]
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.[3] The predictions made on previous problems may not have been useful for those problems in terms of dicovering 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 2.1. The procedure EBL-Propagate compiles the following p-rules for the output variable F,

$$\neg ab(M1) \wedge \neg ab(M2) \wedge \neg ab(A1) \wedge A = a \wedge B = b \wedge C = c \wedge D = d \rightarrow F = a * c + b * d \quad (18)$$

$$\neg ab(M1) \wedge \neg ab(M3) \wedge \neg ab(A1) \wedge \neg ab(A2) \wedge A = a \wedge C = c \wedge E = e \wedge G = g \rightarrow F = a * c + (g - c * e) \quad (19)$$

Similar rules are compiled for G.

[2]This reduces the search for inference chains and subsumption testing between labels.

[3]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).

Example 3.2 Consider the adder example 2.2. The procedure EBL-Propagate compiles the following p-rules for the output variables,

$$\neg ab(Xor1) \wedge \neg ab(Xor2) \wedge$$
$$X1 = x1 \wedge Y1 = y1 \wedge C0 = c0 \wedge s = (x1 \oplus y1 \oplus c0) \rightarrow S1 = s1 \quad (20)$$
$$\neg ab(Xor1) \wedge \neg ab(And1) \wedge \neg ab(And2) \wedge \neg ab(Or1) \wedge$$
$$X1 = 0 \wedge Y1 = 0 \rightarrow C1 = 0 \quad (21)$$
$$\neg ab(And1) \wedge \neg ab(And2) \wedge \neg ab(Or1) \wedge$$
$$X1 = 0 \wedge C0 = 0 \rightarrow C1 = 0 \quad (22)$$
$$\neg ab(And1) \wedge \neg ab(And2) \wedge \neg ab(Or1) \wedge$$
$$Y1 = 0 \wedge C0 = 0 \rightarrow C1 = 0 \quad (23)$$

For the given premise instance, either of the p-rules 22 or 23 is all that is needed for prediction. They both have the same assumption label, and that label subsumes that of rule 21. If we substitute for the premises, rules 21 & 22 will degenerate to prediction 8 of example 2.2. Although redundant for the given premises, rules 21 & 22 may be irredundant for other instances. For example, if the premise was: $\{X1 = 0, Y1 = 1, C0 = 0\}$, then rules 21 & 23 are not applicable, but rule 22 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 STATIC-Propagate. In EBL-Propagate we require that the learnt p-rules be consistent with the given premise instance (table 2, step 2(a)iv). STATIC-Propagate is the same as EBL-Propagate, except that step 2(a)iv is replaced by general satisfiability, instead of satisfiability for a given premise instance. The rules compiled by STATIC are to apply to all possible instantiations of the premise set, rather than to only a generalization of an initially given one as in EBL-Propagate. See table 3.

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

Example 3.4 For the 1-bit adder circuit, STATIC-Propagate compiles the following p-rules in addition to those compiled by EBL-Propagate (example 3.2),

$$\neg ab(And1) \wedge \neg ab(Xor2) \wedge \neg ab(And2) \wedge \neg ab(Or1) \wedge$$
$$X1 = 0 \wedge C0 = c0 \wedge S1 = s1 \wedge s1 = (0 \oplus c0) \rightarrow C1 = 0 \quad (24)$$
$$\neg ab(And1) \wedge \neg ab(Xor2) \wedge \neg ab(And2) \wedge \neg ab(Or1) \wedge$$
$$Y1 = 0 \wedge C0 = c0 \wedge S1 = s1 \wedge s1 = (0 \oplus c0) \rightarrow C1 = 0 \quad (25)$$
$$\neg ab(And1) \wedge \neg ab(Or1) \wedge$$

Table 2. **EBL-Propagate**, a "Learning While Doing" Prediction Algorithm

Input: Premise variables and their value assignments.
Output: All p-rules applicable to generalization of the premise assignments.
Initialization: For each premise variable assert a p-rule: $P \wedge A \wedge C \rightarrow V$, where assumption A and condition C are nil and prediction V has the same value (generalized) as that of the premise P
Description:

1. $Change \leftarrow false$
2. For each value inference rule R: $X \rightarrow Y$ do:
   (a) For each tuple of X whose p-rules' dependencies do not include Y do:
      i. Merge the premises for the p-rules of X. Let the result be Px.
      ii. Merge constraint relations for the p-rules of X in terms of premise Px.
      iii. Unify the merged constraint relation with R to determine a relation Cx betweeen Px and Y
      iv. Verify that $Cx \wedge Px$ is satisfiable for the given value assignments of the premise variables
      v. Propagate the assumptions and dependencies of X to determine labels for Y. Let the assumption label be Ax
      vi. If the p-rule: $Px \wedge Ax \wedge Cx \rightarrow Y$ is not subsumed by a prior rule then do:
         A. Assert the current rule
         B. Retract existing rules subsumed by the current one
         C. $Change \leftarrow true$
3. If *Change* then go to 2.

Table 3. **STATIC-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 2(a)iv. Instead of that step do: Verify that $Cx \wedge Px$ is satisfiable for some instantiation of premise variables from their domain.

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

$$X1 = 1 \wedge Y1 = 1 \rightarrow C1 = 1 \tag{26}$$

$$\neg ab(Xor1) \neg ab(And2) \wedge \neg ab(Or1) \wedge X1 = x1 \wedge Y1 = y1, C0 = 1 \wedge 1 = (x1 \oplus y1) \rightarrow C1 = 1 \tag{27}$$

$$\neg ab(Xor2) \neg ab(And2) \wedge \neg ab(Or1) \wedge C0 = 1 \wedge S1 = 0 \rightarrow C1 = 1 \tag{28}$$

The reason the above rules are not compiled by EBL-Propagate is that their conditions are incompatible with the given instance of input-output values. In general, the p-rules learnt by EBL-Propagate depend on the particular premise instance. In general, EBL-Propagate requires multiple examples to learn all the p-rules that are learnt by STATIC-Propagate. For the polybox circuit, one example will suffice 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 required.

3.2 Conflict Recognition

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. Non-minimal conflicts must be discarded. Step 3 in GET-CONFLICT (table 4) eliminates p-rules that could lead to non-minimal conflicts.

Table 5. A Candidate 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
   (d) Return hit 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). 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. That indexing will eliminate search when retrieving the applicable d-rule. The procedure, ALL-DIAG, to determine all diagnoses is given in table 5.

3.4 Learning Architectures

We consider two learning architectures: EBL(p) and STATIC. They both integrate EBL with MBD and associative diagnosis. STATIC 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.

3.4.1 STATIC

A block diagram of STATIC is shown in fig. 4. The function of STATIC is to generates diagnostic hypotheses consistent with the input observations. STATIC performs that task in terms of two sub-tasks: 1. conflict recognition, and 2. candidate generation. Conflict

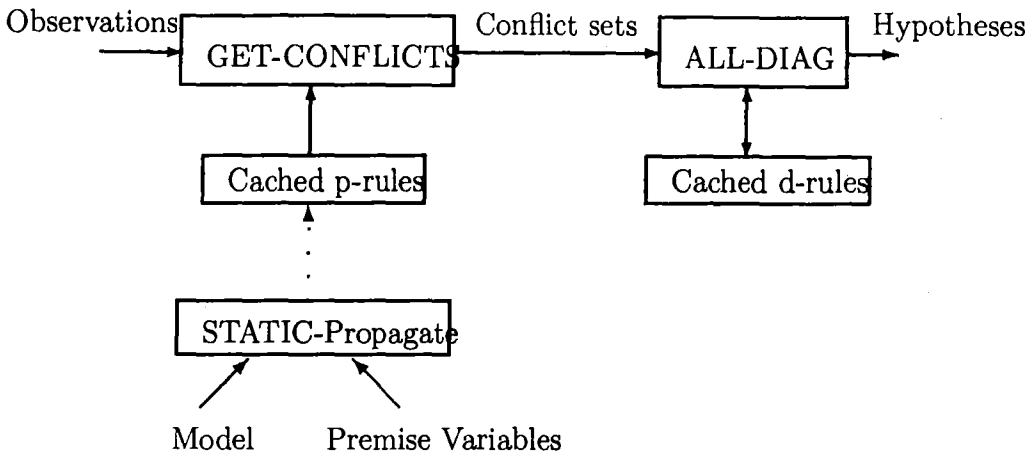

Figure 4. The STATIC Learning Architecture for Diagnosis

recognition is performed by the procedure GET-CONFLICTS. Candidate generation is performed by the procedure ALL-DIAG. All possible predictions are compiled in advance by the procedure STATIC-Propagate, in terms of the device model and the variables designated as observable (premise variables). STATIC-Propagate creates a cache of p-rules that is to be used by GET-CONFLICTS. Note the dashed line between STATIC-Propagate and the cache, indicating that the compilation is done in advance prior to the diagnostic task. Note also that the device model is not subsequently used by SATIC. Compilation of d-rules by ALL-DIAG is performed at diagnosis time. Note that as more d-rules are compiled, STATIC will operate entirely as an associative system.

3.4.2 EBL(p)

A block diagram of EBL(p) is shown in fig. 5. Like STATIC, the function of EBL(p) is to generate diagnostic hypotheses consistent with the input observations. Unlike STATIC, 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 fig. 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 run in parallel as a training system. EVAL-PERF outputs a binary signal to activate or de-activate EBL-Propagate. That signal is determined by comparing the average satisfaction with a threshold, p. EBL-

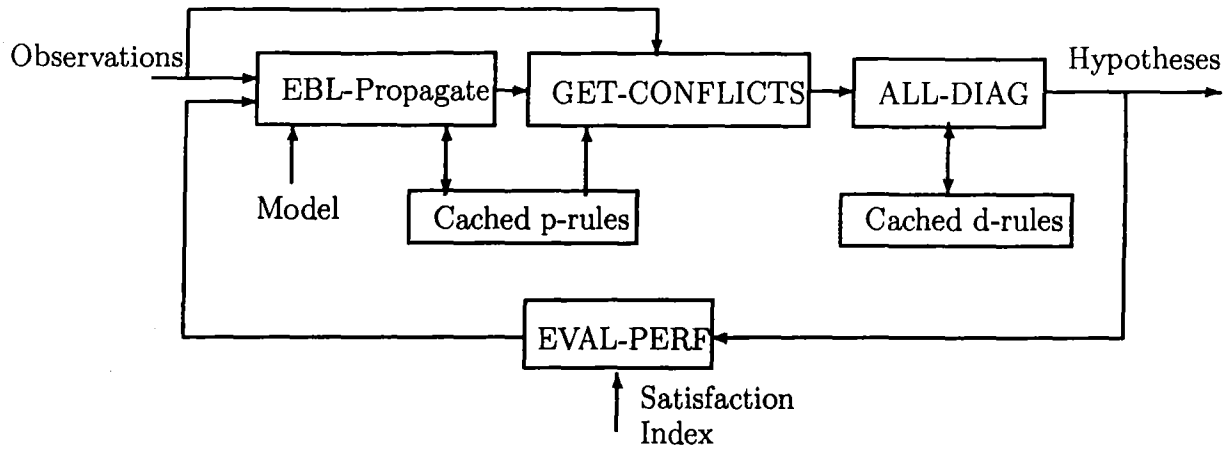

Figure 5. The EBL(p) Learning Architecture for Diagnosis

Propagate remains inactive as long as the average satisfaction is greater than the threshold; otherwise it is active.

If the activation signal is off, EBL(p) carries out its diagnostic task as if it were STATIC. That is, EBL(p) assumes that its p-rules are sufficient to generate all minimal conflicts. If EBL-Propagate was not activated on a sufficient number of examples, the generated hypotheses could be unsatisfactory. If unsatisfactory hypotheses persist then the activation of EBL-Propagate will occur and continue until average satisfaction again reaches the threshold. Notice that when EBL-Propagate is activated the generated hypotheses are identical to those of a model-based system. The difference from a model-based system 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).

3.4.3 Why Hypotheses Evaluation for EBL(p) is Required

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 incorrect, leading to overgeneral diagnoses.

EBL can be used to learn the predictions as associations with subsets of the premises and the assumptions. This provides sufficient conditions for the prediction but not the necessary conditions needed in MBD. One way of overcoming this difficulty is to use constraint-suspension and the model to check diagnoses proposed by learned rules. An

Table 6. Procedural Description of EBL(p)

Input: Observations.
Output: Hypotheses.
Initialization: {Sat is the average satisfaction}
$Activate \leftarrow Yes$, $Problem \leftarrow 0$, $Sat \leftarrow 0$.
Description:

1. If $Sat > p$ then $Activate \leftarrow No$
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}
   $Sat = Sat + (Index - Sat)/Problem$

example follows.

Consider the 1-bit full adder shown in figure 3. Given the assignments [1,1,1] for the inputs: $[X1, Y1, C0]$, and [0,0] for the outputs: $[C1, S1]$, the prediction rules (p-rules) learnt for the outputs are as follows:

$$\neg ab(Xor1) \wedge \neg ab(Xor2) \wedge X1 = x1 \wedge Y1 = y1 \wedge C0 = c0 \wedge s = (x1 \oplus y1 \oplus c0) \rightarrow S1 = s1 \quad (29)$$

$$\neg ab(And1) \wedge \neg ab(Or1) \wedge X1 = 1 \wedge Y1 = 1 \rightarrow C1 = 1 \quad (30)$$

$$\neg ab(Xor2) \wedge \neg ab(And2) \wedge \neg ab(Or1) \wedge S1 = 0 \wedge C0 = 1 \rightarrow C1 = 1 \quad (31)$$

Consider the subsequent example where the input is [0,1,1] and the output [0,0]. Obviously rules 29 and 30 both apply, making predictions for both outputs. This is not a complete set of prediction rules, since the following p-rule is applicable to the example, and provides a new assumption label.

$$\neg ab(Xor1) \neg ab(And2) \wedge \neg ab(Or1) \wedge X1 = x1 \wedge Y1 = y1, C0 = 1 \wedge 1 = (x1 \oplus y1) \rightarrow C1 = 1 \quad (32)$$

The missing rule 32 causes the conflict set $\{\neg ab(Xor1), \neg ab(And2), \neg ab(Or1)\}$ to be missed so the diagnosis proposed by the learned rules (29- 31) is too general.

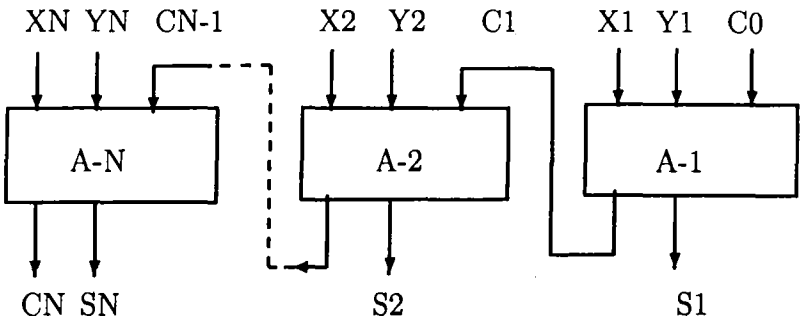

Figure 6. N-Bit Adder

4 Empirical Results

We have carried out an empirical study to compare the performance of STATIC, EBL(p) and MBD. We studied the 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-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 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-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 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.

4.1 STATIC

The parameters studied are the cumulative number of d-rules, and the cumulative time. The p-rules are learnt once per experiment, before feeding the diagnostic problems. During actual diagnosis the model is never used. The cumulative time for STATIC includes the initial compilation time. The minimal candidates produced by STATIC and MBD are verified to be the same for every problem.

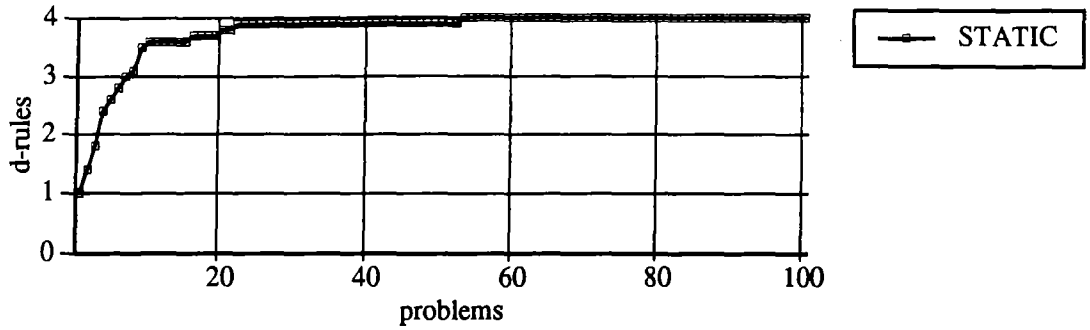

Figure 7. Polybox—Cumulative Number of D-Rules for STATIC (±10%)

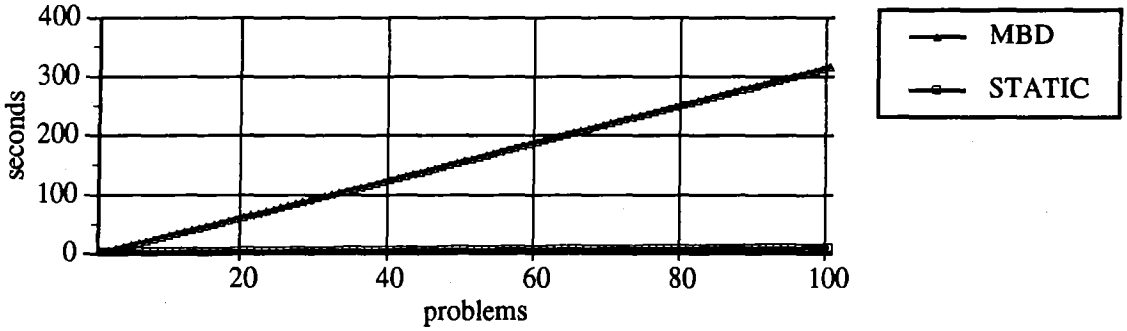

Figure 8. Polybox—Cumulative Time for STATIC Versus MBD (±2%)

4.1.1 Polybox

The number of d-rules (figure 7) rises sharply for the first few problems, and then levels off as the number of problems increases. After 50 problems all four d-rules are learnt. The cumulative time for MBD and STATIC rises almost linearly with the number of problems. The slope of STATIC is dramatically flatter than that of MBD (approx. 3%). See figure 8. This indicates that the matching cost is negligible compared to the time it takes to search the model for all value inferences and to compute the hit sets.

4.1.2 1-bit Adder

The number of d-rules (figure 9) increases steadily with the number of problems. The increase is more appreciable for the first 50 problems than for the last 50. (The increase of the d-rules is an indication of the effectiveness of the fault generator in spanning the space of diagnostic hypotheses.) The cumulative time for MBD and STATIC rises almost linearly with the number of problems. The slope of STATIC is dramatically flatter than that of MBD (approx. 6%). See figure 10.

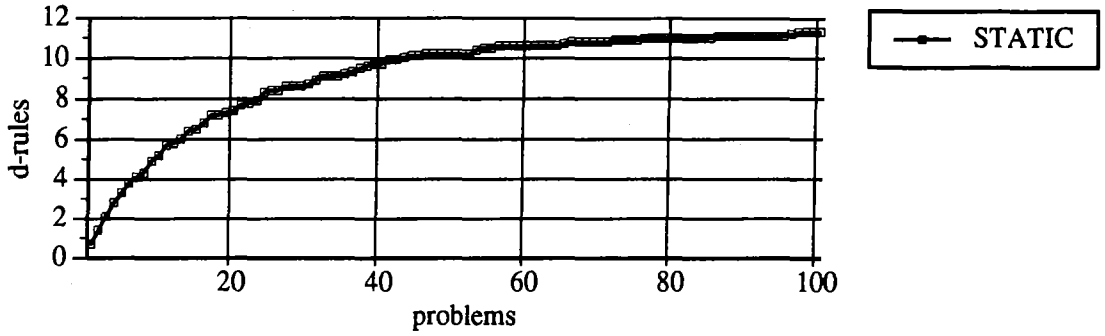

Figure 9. 1-Bit Adder—Cumulative Number of D-rules for STATIC (7%±)

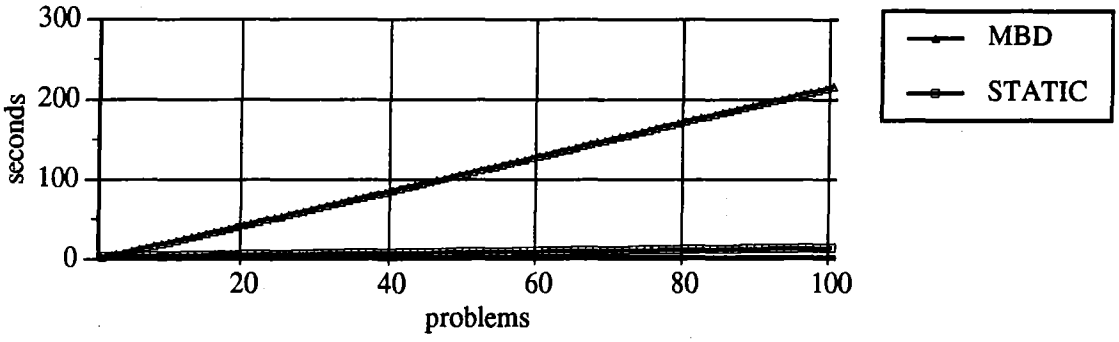

Figure 10. 1-Bit adder—Cumulative Time for STATIC Versus MBD (±2%)

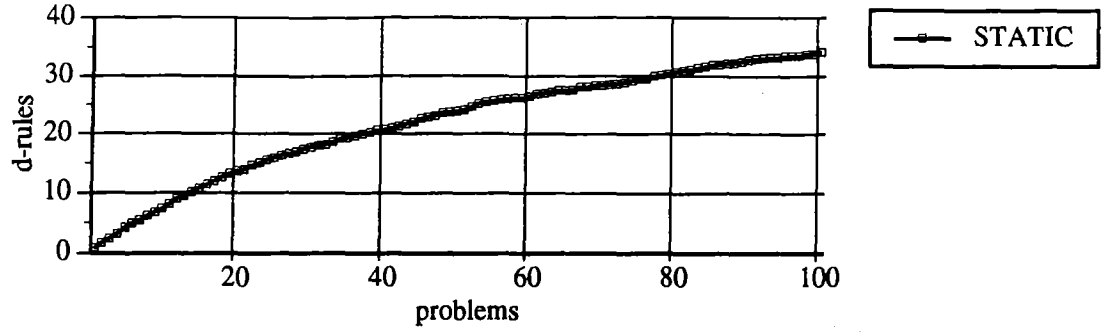

Figure 11. 2-Bit Adder—Cumulative Number of D-Rules for STATIC (±3%)

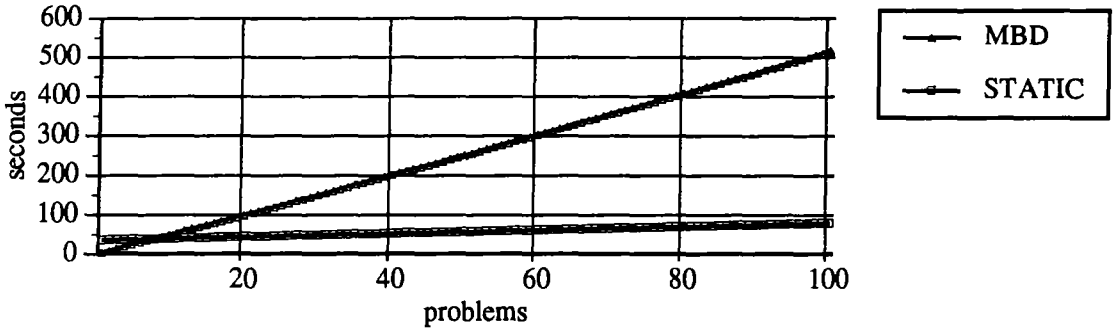

Figure 12. 2-Bit Adder—Cumulative Time for STATIC Versus MBD (±10%)

4.1.3 2-bit Adder

The number of d-rules (figure 11) increases steadily with the number of problems. The cumulative time for MBD and STATIC rises almost linearly with the number of problems. The slope of STATIC is about 25% of MBD. See fig. 12. The initial offset of STATIC, due to the initial compilation time, produces a relatively significant impact in comparison with the 1-bit adder (figure 10). STATIC produces a performance speed-up in comparison with MBD after diagnosing a few problems. The number of problems corresponds to the intersection point between the STATIC and MBD cpu-time curves. The cross-over point occurs at about 10 problems. See fig 12.

4.1.4 3-bit Adder

The number of d-rules (figure 13) increases steadily with the number of problems. The rate of the increase is even more uniform than it is for the 2-bit adder. This is due to the random nature of the fault generator and the size of the circuit, making it more likely to see a new collection of conflicts for every problem. The cumulative time for MBD and

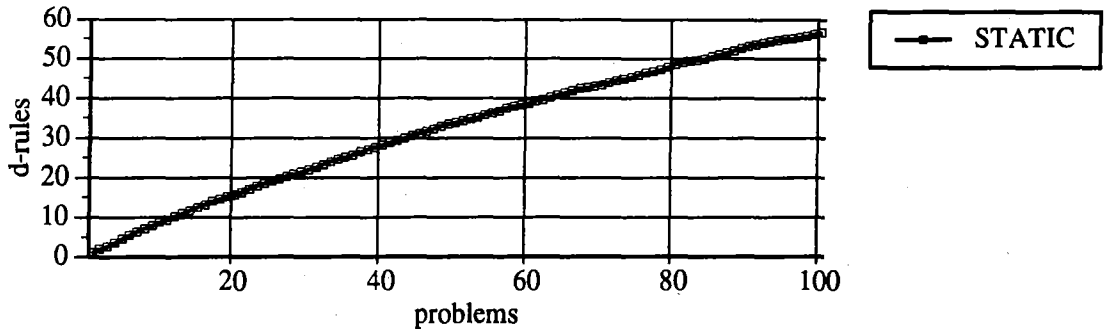

Figure 13. 3-Bit Adder—Cumulative Number of D-Rules for STATIC (±10%)

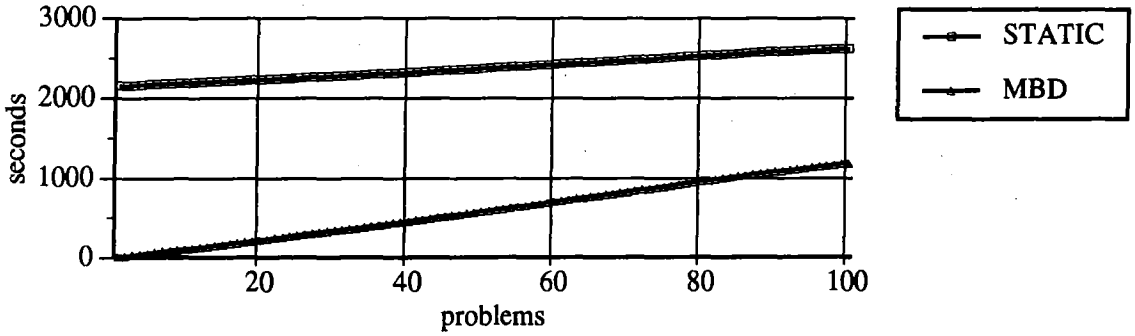

Figure 14. 3-Bit Adder—Cumulative Time for STATIC (±10%) Versus MBD (±30%)

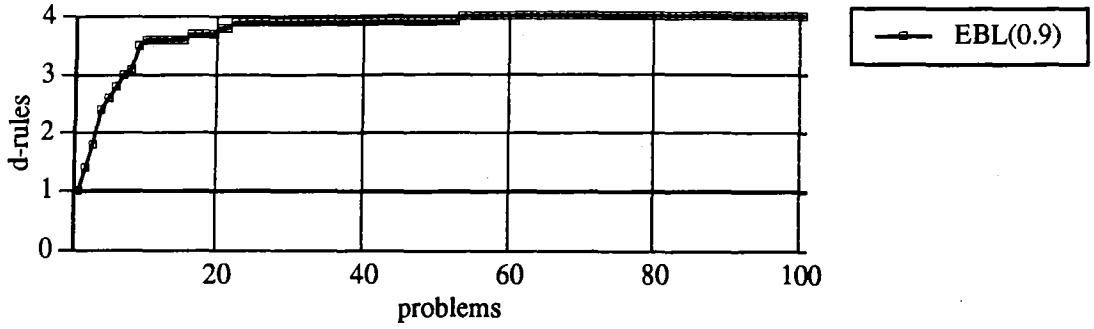

Figure 15. Polybox—Cumulative Number of D-Rules for EBL(p) (±10%)

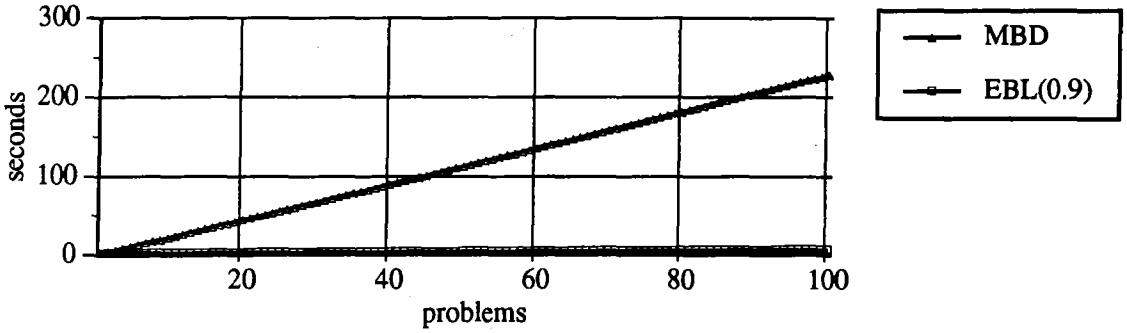

Figure 16. Polybox—Cumulative Time for EBL(p) Versus MBD (±3%)

STATIC rises almost linearly with the number of problems. The slope of STATIC is now a significant 75% of MBD. See figure 14. The time to compile the p-rules for STATIC offsets its initial cumulative time appreciably. STATIC does not produce a performance speed-up in comparison with MBD over the range of 100 problems. The cross-over point to obtain speed-up appears to be about 300 problems.

4.2 EBL(p)

The parameters studied are the cumulative number of p-rules, d-rules, the performance (ratio of correctly diagnosed problems), and the cumulative time. The p-rules are learnt only when learning is switched on, otherwise conflict set recognition is based solely on previously acquired rules. Notice that learning is switched off when performance exceeds the threshold p, else it is switched on. The d-rules are continually learnt, just as in STATIC.

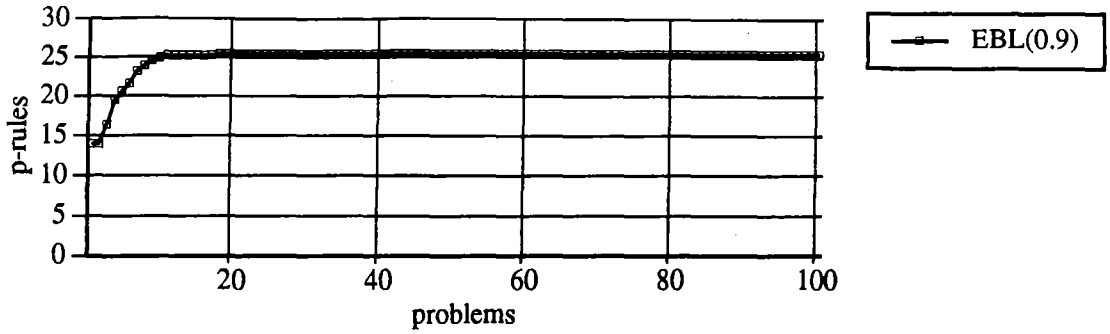

Figure 17. 1-Bit Adder—Cumulative Number of P-Rules for EBL(p) (±3%)

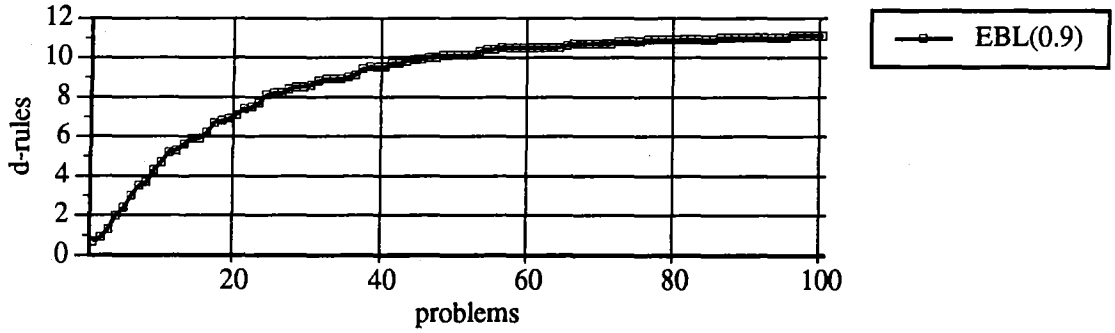

Figure 18. 1-Bit Adder—Cumulative Number of D-Rules for EBL(p) (±7%)

4.2.1 Polybox

The threshold p is set to 0.9. All the p-rules (20 of them) are learnt from the first example. So EBL(p) never turns learning on following the first example. The d-rules soon converges to 4, which are all the rules to be learnt. See figure 15. The performance remains constant at 1.0 throughout. The cumulative time of MBD and EBL(0.9) grows linearly with the actual number of problems, but the slope of EBL(0.9) is only a mere 3% of that of MBD. This is identical to the results obtained from STATIC.

4.2.2 1-bit Adder

The number of p-rules rises sharply following few examples (first 10) to a steady state value. The entire set of p-rules that can be learnt is 26. See figure 17. The number of d-rules rises with a higher rate at the beginning and then with a lower rate later on. The number of d-rules approaches 11 on average toward the end of a 100 problem series. See figure 18. The shallowness of the increase of the p- and d-rules with the increase of number of problems gives a good indication of the effectiveness of learning, since rules are often reused. The

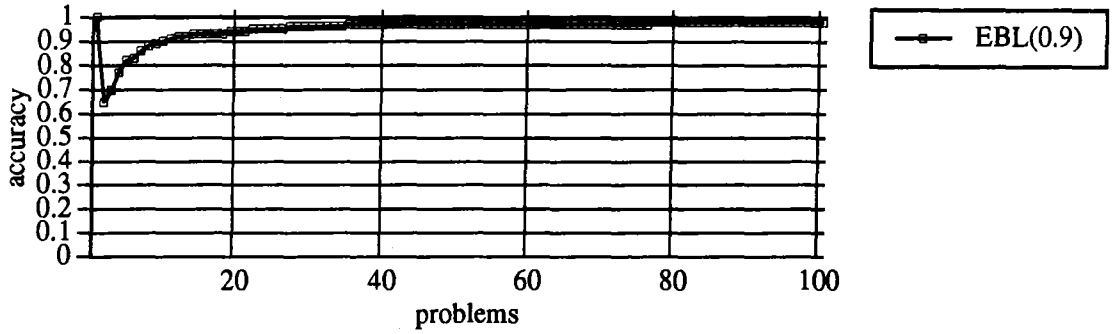

Figure 19. 1-Bit Adder—Performance (±2%)

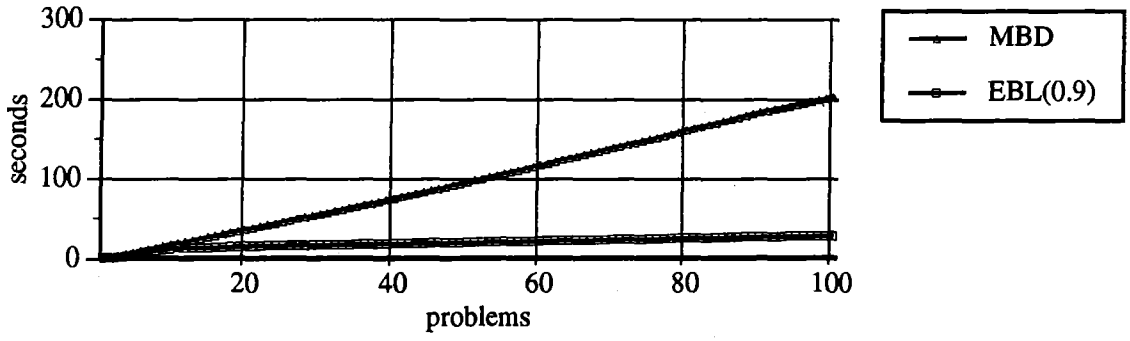

Figure 20. 1-Bit Adder—Cumulative Time for EBL(p) (±70%) Versus MBD (±30%)

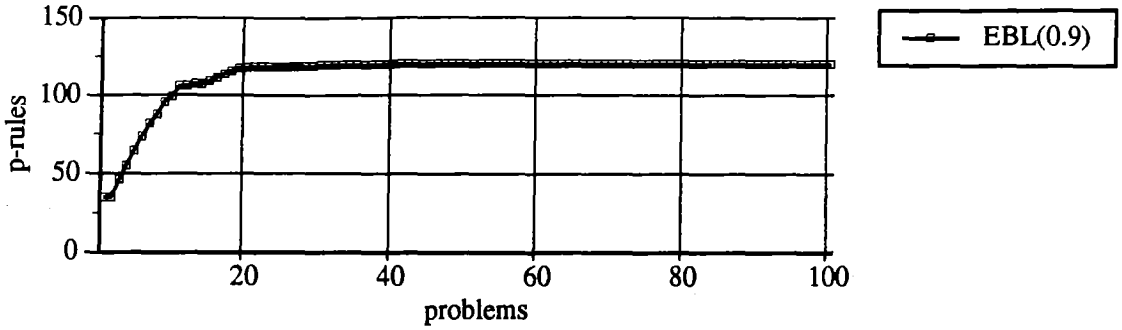

Figure 21. 2-Bit Adder—Cumulative Number of P-Rules for EBL(p) (±8%)

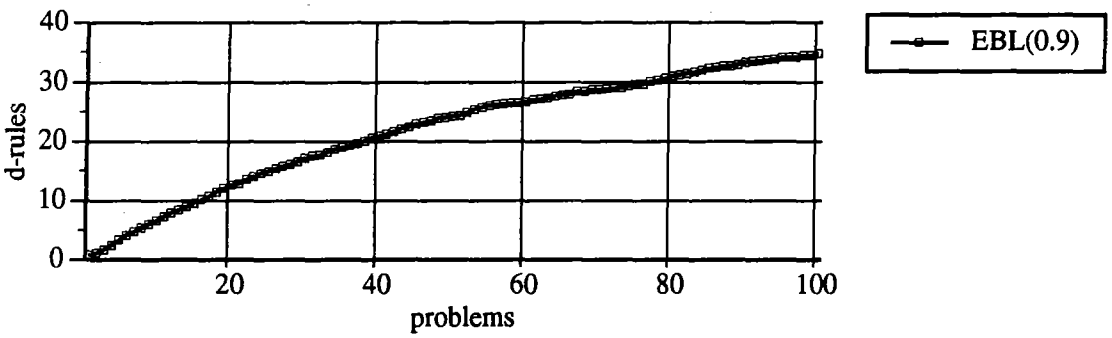

Figure 22. 2-Bit Adder—Cumulative Number of D-rules for EBL(p) (±10%)

performance curve shows a sharp dip at the beginning, where performance falls below the 0.9 threshold, then it rises steadily due to the effect of learning. Performance reaches the threshold value on its way up after about 10 problems. From then on, performance remains above the threshold, and consequently no learning is needed. See figure 19. The cumulative time rises almost linearly for MBD, but displays a "knee effect" for EBL(0.9). See figure 20.

4.2.3 2-bit Adder

Almost all the p-rules are learnt after 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.) See figure 21. The number of d-rules rises with a higher rate at the beginning and then with a lower rate later on. The number of d-rules reaches 35 on average toward the end of a 100 problem series. See figure 22. The effectiveness of d-rules learning is almost 65% as measured by the percentage of rules reused. The performance curve shows a sharp dip at the beginning, where performance falls below the 0.9 threshold, then it rises steadily due to the effect of learning. Performance reaches the threshold value on its way up after about

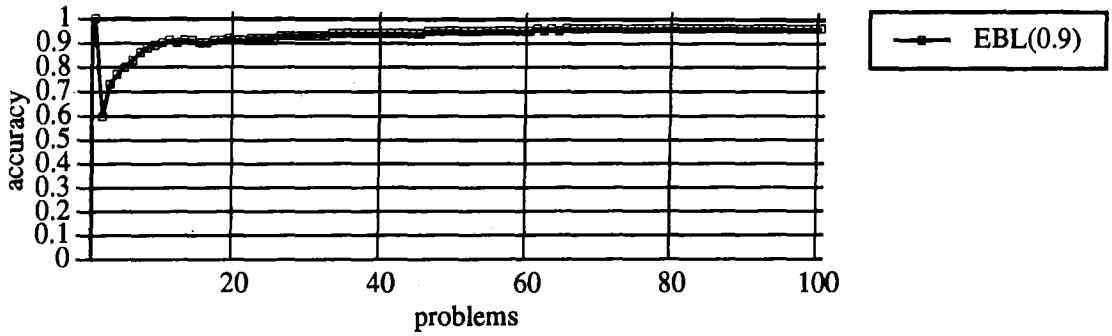

Figure 23. 2-Bit Adder—Performance (±2%)

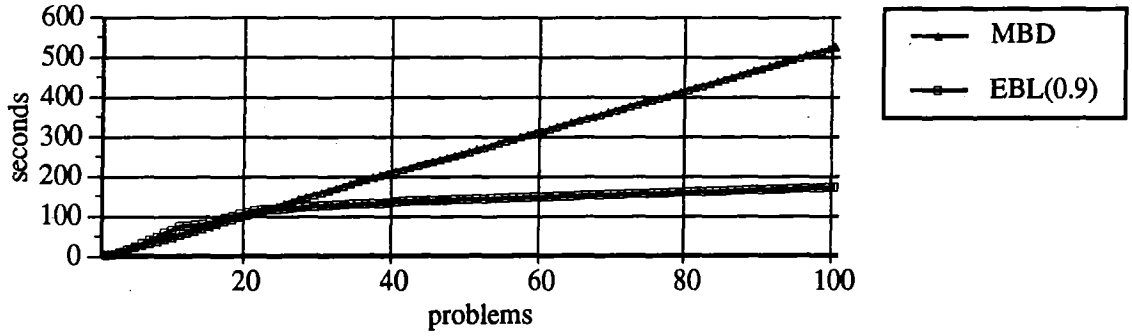

Figure 24. 2-Bit Adder—Cumulative Time for EBL(p) (±40%) Versus MBD (±10%)

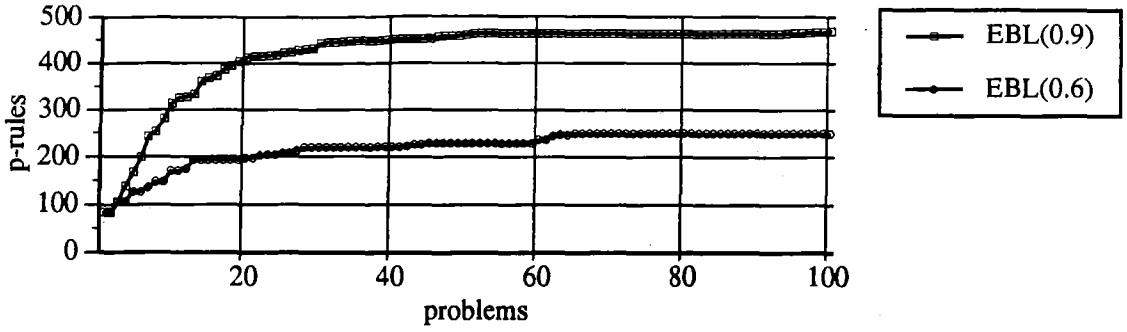

Figure 25: 3-Bit Adder—Cumulative Number of P-Rules for EBL(0.9) (±20%) and EBL(0.6) (±30%)

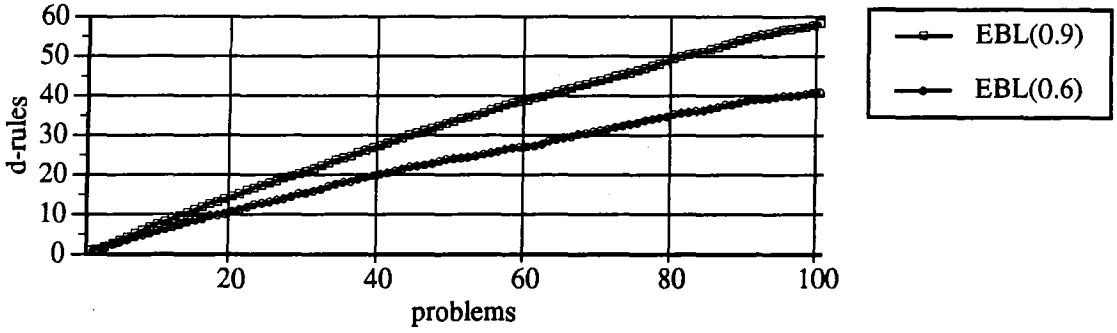

Figure 26: 3-Bit Adder—Cumulative Number of D-Rules for EBL(0.9) (±10%) and EBL(0.6) (±13%)

10 problems. Except for some rippling around the threshold for the range up to about 20 problems, the performance remains above the threshold and no learning is needed. See figure 23. The cumulative time rises almost linearly for MBD, but displays a "knee effect" for EBL(0.9). See figure 24. The knee position is located at the point where learning ends, and associative diagnosis takes over.

4.2.4 3-bit Adder

Here we experimented with two thresholds, 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 performance remains above the threshold.

Figure 25 shows the number of p-rules learnt by EBL(p). As the threshold drops from 0.9 to 0.6, the number of rules drops on average by almost 50%. Notice that EBL(0.9) learns approximately half of the total number of p-rules (918) which would have been learnt by EBL(1). This means that approximately 50 % of the p-rules contributes to 90% of the

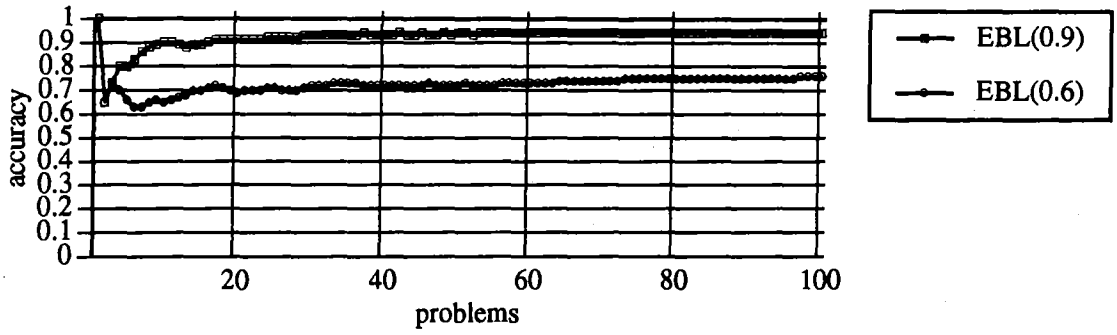

Figure 27. 3-Bit Adder—Performance for EBL(0.9) (±2%) and EBL(0.6) (±10%)

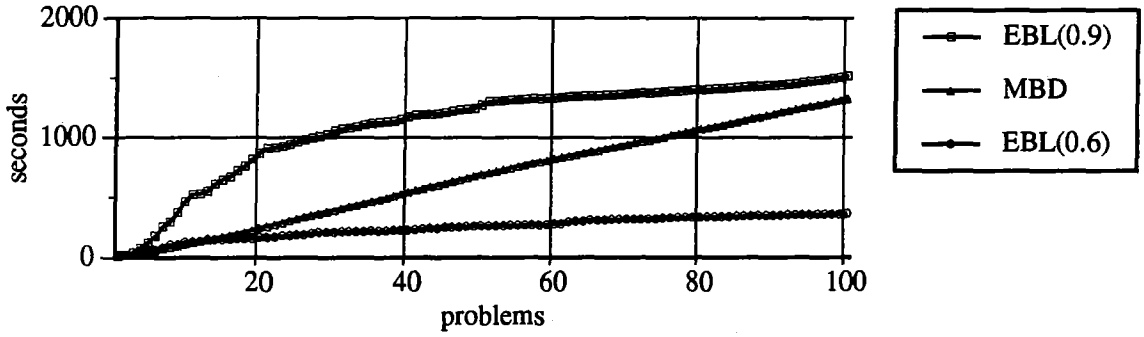

Figure 28: 3-Bit Adder—Cumulative Time for MBD (±30%) and EBL(0.9) (±30%) and EBL(0.6) (±40%)

performance. The number of d-rules grows almost linearly with the number of problems. EBL(0.9) learns on average 3 d-rules every 5 problems, while EBL(0.6) learns 2 d-rules every 5 problems. This means that EBL(0.6)'s abstraction of the diagnostic space is less detailed than that of EBL(0.9). In other words, classification of the diagnosis space for EBL(0.6) is at a coarser level than that for EBL(0.9).

For EBL(0.9), the performance curve shows a sharp dip at the beginning, where performance falls below the 0.9 threshold, then it rises steadily due to the effect of learning. On average, EBL(0.6) performance remains above the threshold and learning after the first problem is rarely invoked. See figure 27. For EBL(0.9), the cumulative time rises steeply 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 comparable to model-based prediction. See figure 28. However in most problems the rate of time increase is much flatter than MBD. For EBL(0.6), speed-up effects are evident compared to MBD. See figure 28.

5 Discussion

5.1 STATIC

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 STATIC. This is only the dynamic cost due to matching and HS-Tree. The preprocessing cost incurred by STATIC for compiling the p-rules is denoted by C_{st}. Based on the empirical results, we can fairly represent the cumulative (average) cost versus the number of problems for STATIC and MBD by linear relations, as depicted in figure 29. The cost of solving N problems by MBD is: $C_{mbd} \times N$. The cost of solving N problems by STATIC is $C_{st} + C_{ass} \times N$. The cross-over point N^* is the number of problems for which the cumulative time of STATIC and MBD is the same. That is,

$$N^* = \left\lceil \frac{C_{st}}{(C_{mbd} - C_{ass})} \right\rceil$$

Based on the empirical results, we can make the following observation. As the number of components increases, two things happen:

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

Table 7 provides numerical values for those parameters for the N-bit adder for increasing N. The results lead us to the following conclusion. STATIC achieves net speedup over

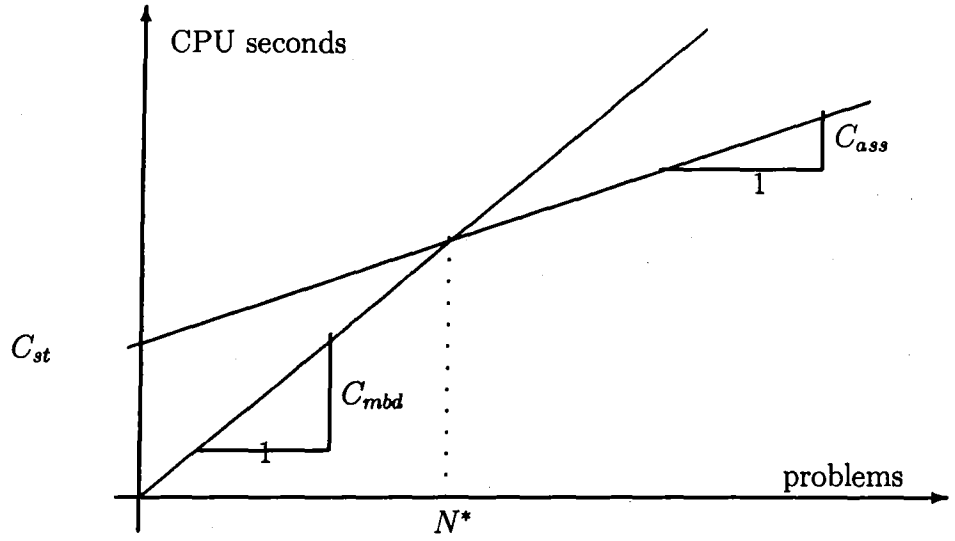

Figure 29. The Cross-Over Point for STATIC Versus MBD

Table 7. STATIC Versus MBD for N-bit Adder

N	C_{st}	C_{ass}	C_{mbd}	N^*
1	3	0.09	2.05	2
2	36	0.44	5.17	8
3	2145	4.68	11.86	299

MBD when precomputation is included only for small devices. As the number of components increases, STATIC becomes worse than MBD because the size of the p-rules grows exponentially. 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 required 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 number of problems tends to increase as the number of components increases. For example, compare figs. 13,11 and 9. This can be attributed to the random nature of the fault generator, making it more likely to see new collections of conflicts for new problems as the number of components increases. This also indicates that the learning of d-rules does not pay off very much on the 3-bit adder over the range of 100 problems. Almost 60 rules are learnt in that range, meaning that savings only occurred over 40 problems (in the sense that no new rules were required).

5.2 EBL(p)

For small size circuits (up to 10 components) the conflict set recognition is the most expensive subtask in MBD. As the size of the circuit increases, the candidate generation subtask becomes as or even more expensive. 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. This is confirmed by our results. See figures 18, 22, and 26. This implies that the use of d-rules will be effective only after a large set of problems. This means that eventually we will have to cache all possible d-rules. If the probability of invoking those rules is uniformly distributed, then it seems that we have not gained very much. Indeed we would only have traded space for time. Compiled d-rules will occupy an exponential space, but they can be applied in constant time. The HS-tree algorithm does not use exponential space but it may require exponential computation.

The above indicates a utility problem in learning for MBD. Learning may provide speed-up only if the rules have high likelihood of being applicable, and irredundant. Note that 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 STATIC, and no overall speed-up effect will be obtained except on small devices.

EBL(p) 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. See figures 25, 26. Note that a diagnosis is considered to be satisfactory if it either consists of the same collection of minimal candidates as are produced by MBD, or else it includes the actual fault. EBL(p) is capable of providing speed-up provided that p is sufficiently small.

Further observations regarding the empirical results in section 4.2 are as follows:

1. For the N-bit adder, the knee effect for EBL(p) (figs. 20, 24,& 28) can be explained as follows. Initially, MBD and EBL are "on," thus the time per problem is comparable to MBD. As soon as the learning phase ends (the knee point), the slope changes and the cpu time curve gets flatter due to the speed up provided by the associative operating mode.

2. For EBL(0.6) on the 3-bit adder, speed-up effects are evident compared to MBD. See figure 28. The reasons are: 1) the number of rules is much less than that of EBL(0.9)— thus reducing matching effort and space requirement; 2) learning is rarely invoked so there are no steep rises for the initial problems as in the case of EBL(0.9).

6 Related Work

Preliminary versions of the results reported here appeared in abridged form in two conference papers (Fattah & O'Rorke, 1991a; Fattah & O'Rorke, 1992). In Fattah & 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 speed-up occurred on the small circuits we initially studied. Unfortunately, more recent studies of parameterized devices (reported in the present paper) indicate that learning in advance is infeasible for large devices.

In (Fattah & 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 paper include averages, over ten experimental runs, of important measurements of this method's performance.

Other works that have explored the use of EBL for MBD include (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 has been more evident in recent works. See de Kleer (1990), El Fattah and O'Rorke (1991c, 1991b), Friedrich, Gottlob & Nejdl (1990). But results on the empirical evaluation of EBL for MBD have been meager and sketchy.

6.1 Knowledge Compilation

Davis (1987) 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, e.g., (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 turn knowledge associated with models into rules 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 STATIC. 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 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_{st} in figure 29 by the number of diagnostic systems produced.

According to Davis (1987), 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.

It is our view that the controversy around knowledge compilation is an indication that many issues are not yet understood. See Goel (1991). We think that more empirical studies are needed to form useful theories about the utility of knowledge transformations in specific task areas such as diagnosis.

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 STATIC. The issue of interpreted versus compiled in the reasoner-CMS architecture is discussed in 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 Utility of Diagnosis

Provan and Poole (1991) define diagnosis as a logical formula such that "all of the models of the formula are use equivalent." A diagnosis that is overgeneral in accordance with

Reiter's definition of minimal diagnosis (Reiter, 1987) could be use-equivalent if it contains the actual fault—thus leading to the right action. We adopted this definition of correct diagnosis in our empirical study of EBL(p). Notice that insisting on having identical collections of minimal candidates to qualify diagnosis as correct will only increase the learning effort and the rules that need to be cached. Also, our notion of approximate diagnosis may be regarded as a definition of class equivalence, except that our notion is empirically-based.

6.4 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 probable 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.

6.5 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-divers environments may be biased towards effective associations, (2) Learning in critical environments 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 STATIC, 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.

STATIC 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.

STATIC 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. STATIC 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 STATIC is subject to the exponential growth associated with MBD. As the size of the device grows, STATIC 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 cross-over point where STATIC 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, STATIC is the preferred alternative at diagnosis

time since it is essentially an extremely fast lookup operation.

EBL(p) provides speed-up over MBD provided that the required accuracy is not too high. EBL(p) alternates between a relatively high cost per problem (incurred when MBD and learning are turned off) and a low cost per problem (incurred by the associational rules). 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. This method is preferable in situations where we are willing to tolerate some errors. In realistic situations, observed faults will tend to form clusters in the space of possible faults. EBL(p) 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 trade-offs reducing costs by sacrificing performance. Discussions with them of 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. We also thank the graduate students and faculty of the AI and machine learning community 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.

References

Davis, R. (1987). Form and content in model based reasoning. In *Working Notes, AAAI-87 Workshop on Model Based Reasoning*, (pp. 11–27).

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

de Kleer, J. (1990). Exploiting locality in a TMS. In *Proceedings, AAAI-90*, (pp. 264–271).

de Kleer, J. (1991). Focusing on probable diagnosis. In *Proceedings, AAAI-91*, (pp. 842–848).

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 8th European Conference on Artificial Intelligence*, (pp. 408–413).

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

Etzioni, O. (1990). Why PRODIGY/EBL works. In *Proceedings, AAAI-90*, (pp. 916–922).

Fattah, Y. E. & O'Rorke, P. (1991a). Learning multiple fault diagnosis. In *Proceedings, The Seventh IEEE Conf. on Artificial Intelligence Applications*, (pp. 235–239).

Fattah, Y. E. & O'Rorke, P. (1991b). On tractability and learning in model based diagnosis. In *Working Notes, AAAI Workshop on Model Based Reasoning.*

Fattah, Y. E. & O'Rorke, P. (1991c). The role of compilation in constraint based reasoning. In *Working Notes, AAAI Spring Symp. on Constraint Based Reasoning*, (pp. 225–241).

Fattah, Y. E. & O'Rorke, P. (1992). Learning approximate diagnosis. In *Proceedings, The Eighth IEEE Conf. on Artificial Intelligence Applications.*

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 Math. Applic., 20*(9/10), 57–72.

Goel, A. (1991, April). Knowledge compilation—a symposium. *IEEE Expert*, 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. *J. Symbolic Computation, 9*, 185–206.

Koseki, Y. (1989). Experience learning in model-based diagnosis. In *Proceedings, IJCAI-89*, (pp. 1356–1362).

Minton, S. (1988). Quantitative results concerning the utility of explanation-based learning. In *Proceedings, AAAI-88*, (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*, 47–80.

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

Provan, G. & Poole, D. (1991). The utility of consistency-based diagnostic techniques. In *Proceedings, KR'91*, (pp. 461–472).

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, AAAI-87*, (pp. 183–188).

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

Zercher, K. (1988). Model-based learning of rules for error diagnosis. In W. Hoeppner (Ed.), *Proceedings GWAI-88* (pp. 196–205). Berlin: Springer Verlag.