Pergamon

Mathl. Comput. Modelling Vol. 27, No. 1, pp. 75–99, 1998
Copyright©1998 Elsevier Science Ltd
Printed in Great Britain. All rights reserved
0895-7177/98 $19.00 + 0.00

PII: S0895-7177(97)00255-0

A Greedy Randomized Adaptive Search Procedure (GRASP) for Inferring Logical Clauses from Examples in Polynomial Time and Some Extensions

A. S. DESHPANDE AND E. TRIANTAPHYLLOU*
Department of Industrial and Manufacturing Systems Engineering
Louisiana State University, 3128 CEBA Building, Baton Rouge, LA 70803-6409, U.S.A.
ietrian@lsuvm.sncc.lsu.edu
Web: http//www.imse.lsu.edu/vangelis

(Received September 1995; revised and accepted September 1996)

Abstract—Two new heuristics are presented for inferring a small size Boolean function from complete and incomplete examples in polynomial time. These examples are vectors defined in $\{1,0\}^n$ for the complete case, or in $\{1,0,*\}^n$ for the incomplete case (where n is the number of binary attributes or atoms and "$*$" indicates unknown value). Each example is either positive or negative, if it must be accepted or rejected by the target function, respectively. For the incomplete case, however, some examples may be unclassifiable. Moreover, computational results indicate that the proposed heuristics may also be effective in solving very large problems with thousands of examples.

Keywords—Boolean functions, CNF/DNF form, Learning from examples, Logical analysis, Clause satisfiability problem, GRASP, Randomized algorithms, Inductive inference.

1. INTRODUCTION

An interesting machine learning problem is how to infer a Boolean function from collections of positive and negative examples. This is also called the logical analysis problem and is a special case of inductive inference. This problem is essential when there is no stochastic or fuzzy behavior in the system and one is interested in extracting an underlying pattern in the form of a set of logical rules. This kind of knowledge extraction is desirable when one is interested in deriving a set of rules which, in turn, can be easily comprehended by a field expert. In many domains the end users lack sophisticated computer and modelling expertise. As result, systems which are based on techniques such as neural networks, statistics, or linear programming, are not appealing to them. On the other hand, a logical analysis approach, when it is applicable, can result in rules which are already known to the end user (thus increasing his/her confidence on the method) or lead to new discoveries.

The most recent advances in distinguishing between elements of two pattern sets can be classified into six distinct categories. These are: a clause satisfiability approach to inductive inference by Kamath *et al.* [1,2]; some modified branch-and-bound approaches of generating a small set of

The authors are very grateful to the two anonymous referees for their comments which significantly improved an earlier version of this paper.
*The second author gratefully recognizes the partial support from the Office of Naval Research (ONR) Grant N00014-95-1-0639 and the LaSPACE program at LSU. All correspondence should be addressed to the second author.

Typeset by $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX logical rules by Triantaphyllou *et al.* [3] and Triantaphyllou [4]; some improved polynomial time and NP-complete cases of Boolean function decomposition by Boros *et al.* [5,6]; linear programming approaches by Wolberg and Mangasarian [7], Mangasarian *et al.* [8], and Mangasarian [9]; some knowledge based learning approaches by combining symbolic and connectionist (neural networks) machine based learning as proposed by Shavlik [10], Fu [11], Goldman *et al.* [12], and Cohn *et al.* [13], and finally, some nearest neighbor classification approaches by Hattori and Torii [14], Kurita [15], Kamgar-Parsi and Kanal [16]. From the above six categories, the first three can be considered as logical analysis approaches.

More specifically, Gallant [17] and Fu [18] have proposed algorithms which formulate rules if a node's weighted input exceeds its threshold and convert each one of these situations into a rule. However, these methods can require an *exponential number of rules* for rule formulation [10]. Recall that in the present paper, emphasis is given in minimizing the number of rules. Towell and Shavlik [19] developed a method that produced comprehensible rules for each node while maintaining the accuracy of the network. However, their approach works well only on knowledge based networks as it requires those weights to cluster into a few groups. Knowledge based connectionism clusters the weights of the neurons to improve the search efficiency.

Fu [11] proposed an algorithm for extracting rules from a trained neural network. The algorithm heuristically searches through the rule space distinguishing between positive and negative attributes that link to positive and negative weights. Since the search procedure is done layer by layer, the algorithm would have an *exponential time complexity* with respect to the depth of the network.

Goldman and Sloan [12] studied the self directed learning approach in which the learner chooses the training examples to be supplied to the learning mechanism. Although they show that the time complexity is somewhat less than other learning mechanisms in this class of classification approaches, it essentially remains of exponential complexity.

This paper examines two interrelated problems. In both problems, given are collections of examples. In the first problem each example is a vector of size n (where n is the number of binary attributes or atoms pertinent to the current application) defined in the space $\{1,0\}^n$. Positive examples should be accepted by the inferred Boolean function while negative examples should be rejected. In the second problem, examples are defined in the space $\{1,0,*\}^n$, where "$*$" stands for a missing (i.e., unknown) value. Because of the presence of missing elements in the examples, now some examples may be *unclassifiable* (i.e., the expert, or oracle cannot determine whether the example is positive or negative). As result, the inferred Boolean function should neither accept nor reject the unclassifiable examples. In the above settings the focus is to infer a Boolean function in CNF (conjunctive normal form) or DNF (disjunctive normal form) of a very small, ideally minimum, number of disjunctions (for the CNF case) or conjunctions (for the DNF case). The motivation for this direction becomes evident when one considers that each CNF disjunction corresponds to a single logical rule. That is, to an "IF–THEN" type of logical structure.

The above clause minimization problem is NP-complete [2]. In the past, the second author and his associates [3,4] have developed some efficient (but still of exponential time complexity) branch-and-bound (B&B) approaches which can infer very small size Boolean functions (either in CNF or DNF form). The present paper presents the development of two randomized heuristics which can infer small size Boolean functions in *polynomial time*. These two heuristics can be used for solving the previous two research problems. A rigorous definition of these two problems and some pertinent notation is provided in the next section.

2. SOME DEFINITIONS AND TERMINOLOGY

Before proceeding to the actual problem description, some terminology is discussed. Let $\{A_1, A_2, \ldots, A_n\}$ be a set of n Booolean predicates (binary attributes or atoms). Each atom A_i,

$\{i = 1, 2 \ldots, n\}$ has a binary value, i.e., it can either be true (1) or false (0). Let F be a Boolean function over these predicates, i.e., F is a mapping from $\{0,1\}^n \rightarrow \{0,1\}$. In this paper, the terms Boolean functions, Boolean system, logical system and set of clauses will be interchangeably used to denote the same concept. For each combination of the $A_1, A_2, \ldots, A_n$ values, F takes the value true (1) or false (0). The positive set of examples are those combinations of the $A_1, A_2, \ldots, A_n$ values for which F takes the value of 1. In this paper, they will be represented as the E^+ set. The negative set of examples are those combinations of attribute values which evaluate the value of F as 0. They will be represented as the E^- set.

The Boolean expressions dealt with in this paper are of conjunctive normal form (CNF). Any Boolean expression can be transformed into the CNF or disjunctive normal form (DNF) [20]. The CNF and DNF representation schemes are defined as (I) and (II), respectively:

$$\bigwedge_{j=1}^{k} \left(\bigvee_{i \in \rho_j} a_i \right), \quad \text{and} \tag{I}$$

$$\bigvee_{j=1}^{k} \left(\bigwedge_{i \in \rho_j} a_i \right), \tag{II}$$

where a_i is either A_i or $\bar{A}_i$ and ρ_j is the superset of the indices of the atoms in the j^{th} conjunction or disjunction. In other words, a CNF expression is a conjunction of disjunctions, while a DNF expression is a disjunction of conjunctions. Triantaphyllou and Soyster [21] show how to use any DNF algorithm to derive a CNF expression (and *vice versa*).

To illustrate the central idea of this paper, consider the following two sets E^+ and E^- which represent the positive and negative examples, respectively:

$$E^+ = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix}, \quad E^- = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix}.$$

These examples are binary vectors of dimension 4 (i.e., $n = 4$). For instance, the first example in the E^+ set (i.e., $[0 \quad 1 \quad 0 \quad 0]$) implies that the values of (A_1, A_2, A_3, and A_4) are equal to (false, true, false, and false), respectively. Therefore, the desired Boolean function should evaluate to the true value (denoted as (1)) when it is fed with that positive example. Similar observations can be made for the rest of the positive and negative examples (for the negative examples the Boolean function must return negative value). Next, consider the following Boolean function which has three disjunctions:

$$(A_2 \vee A_4) \wedge (\bar{A}_2 \vee \bar{A}_3) \wedge (A_1 \vee A_3 \vee \bar{A}_4).$$

It can be easily verified that the above expression satisfies the requirements of the available positive and negative examples. That is, each positive example makes **each** disjunction to return true value. Also, each negative example is rejected by **at least one** of the disjunctions.

This in essence is Problem 1: how to construct a set (of hopefully small size) of clauses (i.e., terms of a Boolean function) which would correctly classify all the available positive and negative examples and hopefully classify new examples with high accuracy.

Next, to illustrate Problem 2, consider the sample of the input data shown below:

$$E^+ = \begin{bmatrix} 0 & * & 0 & 0 \\ 1 & 0 & * & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}, \quad E^- = \begin{bmatrix} 0 & 0 & * & 1 \\ * & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \quad E^U = \begin{bmatrix} 0 & 1 & 1 & * \\ 1 & 1 & * & * \end{bmatrix}.$$

In this paper the set E^U always denotes the set with the unclassifiable examples. The symbol "$*$" in the three data sets represents atoms whose values (i.e., true or false) are missing.

In the previous E^+ and E^- data sets it is assumed that the missing data values (indicated by "$*$") did not prohibit the user (i.e., the hidden function or oracle) from classifying the corresponding examples as positive or negative. For instance, the first positive example in E^+ (i.e., $[0, *, 0, 0]$) implies that this example should be positive *regardless* of the actual nature of the "$*$" element. This observation indicates that the following 2 examples (note that $2 = 2^1$, where 1 is the number of the missing elements in that example): $[0, 0, 0, 0]$ and $[0, 1, 0, 0]$ are also positive examples. That is, for positive and negative examples the missing values can be treated as "*do not care*" cases (i.e., they can be either 1 or 0 without changing the classification of that example). The notion of the "*do not care*" (or DC) concept was first introduced by Kamath *et al.* [2] in order to condense the information representation in this type of learning problems.

An obvious restriction for the data to be correct is that every possible pair of a positive and a negative example should have at least one of their common fixed atoms with a different value. For instance, when $n = 8$, then the examples $[1, 1, *, *, 0, 0, *, *]$ and $[1, 1, *, 0, 0, 0, *, 1]$ *cannot* belong to different classes (i.e., one to be positive and the other to be negative). This is true because the example $[1, 1, 0, 0, 0, 0, 0, 1]$ is implied by either one of the previous two examples with the missing (i.e., "*do not care*") elements.

To further illustrate the concept of the unclassifiable examples consider the following Boolean function defined on five atoms (i.e., $n = 5$):

$$(A_1 \vee A_4) \wedge (A_2 \vee \bar{A}_3 \vee A_5) .$$

The above Boolean function would make an example such as $[0, 1, 1, *, 1]$ unclassifiable chiefly because the value of A_4 is missing.

A naive way for dealing with data which have missing values would be to ignore the unclassifiable examples and concentrate the attention only on the positive and negative examples. That is, to ignore all unclassifiable examples, and expand all the positive and negative examples, and thus transform Problem 2 into a problem of Type 1. Recall that if a positive or negative (but not unclassifiable) example has k (where $k < n$) missing values, then it can be expanded into 2^k positive or negative examples defined in $\{1, 0\}^n$. However, if this is done, then one would ignore the information present in the unclassifiable examples. That is, by knowing that the inferred system should neither accept nor reject any of the unclassifiable examples, the search for an accurate Boolean function may be better directed.

3. A HEURISTIC FOR INFERRING A BOOLEAN FUNCTION FROM COMPLETE DATA

The aim of the clause inference strategies in this paper is to derive a very small (hopefully minimum) number of disjunctions (in the CNF case). Also recall that although the Boolean functions derived in the proposed approach are in CNF form, DNF functions can also be derived from the same data set [21] and *vice versa.*

A Boolean function in CNF must satisfy the following requirements:

(i) each clause in the derived system should accept all the examples in the E^+ set, and
(ii) all the clauses, when taken together, should reject all the examples in the E^- set.

In [3] an algorithm which infers CNF Boolean expressions of small size from positive and negative examples is developed. In that approach, CNF clauses are generated in a way which *attempts to minimize* the number of CNF clauses that constitute the recommended CNF system. In this way, a compact CNF system can be derived. The strategy followed there is called the *one clause at a time (or OCAT)* approach.

The OCAT approach is *greedy* in nature. It uses as input data two collections of positive and negative examples (denoted as E^+ and E^-, respectively). It determines a set of CNF clauses which, when taken together, reject all the negative examples and accept all the positive examples. The OCAT approach is sequential. In the first iteration it determines a single clause (i.e., a disjunction) which accepts all the positive examples in the E^+ set while it rejects as many negative examples in E^- as possible. This is the greedy aspect of the approach. In the second iteration if performs the same task using the original E^+ set but the revised E^- set has only those negative examples which have not been rejected by any clause (i.e., the first) so far. The iterations continue until a set of clauses is constructed which reject all the negative examples in the original E^- set. More on this approach can be found in [3,4]. Figure 1 summarizes the iterative nature of the OCAT approach.

```
$i = 0$ ; $C = \phi$; {initializations}
DO WHILE ($E^- \neq \phi$)
        Step 1: $i \leftarrow i + 1$; /* $i$ indicates the $i^{\text{th}}$ clause */
        Step 2: Find a clause $c_i$ which accepts all members of $E^+$
                while it rejects as many members of $E^-$ as possible
        Step 3: Let $E^-(c_i)$ be the set of members of $E^-$ which are rejected by $c_i$
        Step 4: Let $C \leftarrow C \cup c_i$
        Step 5: Let $E^- \leftarrow E^- - E^-(c_i)$
REPEAT;
```

Figure 1. The one clause at a time (OCAT) approach.

At this point please note that the number of atoms in a rule (logical clause) could be a measure of performance. However, the OCAT approach (which is the underlying philosophy of the proposed heuristics) creates CNF (or DNF) clauses which are of decreasing discriminatory power. This is true, because the first clause rejects many negative examples (while it accepts all the positive examples) and so on. Thus, the first clauses are defined on very few atoms while later clauses are increasingly defined on more and more atoms.

The core of the OCAT approach is Step 2, in Figure 1. In [3] a branch-and-bound (B&B) based algorithm is presented which solves the problem posed in Step 2 efficiently. A more efficient branch-and-bound algorithm (B&B), along with other enhancements, are described in [4]. The OCAT approach returns the set of desired clauses (i.e., the CNF system) as set C.

To offset the drawback of the exponential time complexity of the B&B algorithm in Step 2 of the OCAT approach, in the first proposed heuristic clauses are formed in a manner such that each clause accepts all the examples in the E^+ set while it attempts to reject many (as opposed to *as many as possible* in the B&B approaches) examples in the E^- set. Note that this is the main procedural difference between the B&B algorithms and the proposed heuristics. This is achieved in the proposed first heuristic by choosing the atoms to form a clause based on an evaluative function (to be described later). Only atoms with high values in terms of the evaluative function are included in the current clause. A single clause is completely derived when all the examples in the E^+ set are accepted. The clause forming procedure is repeated until all the examples in the E^- set are rejected by the proposed set of clauses. As some computational results (presented in a later section) indicate, this strategy may often result in Boolean functions with a small number of clauses.

Observe that if always the atom with the *highest value* of the evaluative function is included in the clause, then there is an inherent danger of being trapped in a local optimal point. To prevent the Boolean system from being degenerated as result of being trapped at a local optimal point, a *randomized* approach is used. In this randomized approach, instead of a single atom being included in a clause due to its highest value of the evaluative function, a candidate list is formed of atoms whose values in terms of the evaluative function are close to the highest value as derived from the evaluative function. Next, an atom is *randomly* chosen out of the candidate list and is included in the CNF clause being derived.

Please note that it is possible a CNF clause to reject as many negative examples as possible (and, of course, to accept all positive examples) but the entire system not to have a small (ideally minimum) number of clauses. Recall that the proposed heuristics follow the OCAT approach (see also Figure 1). That is, sometimes it may be more beneficial to have a less "effective" clause which does not reject a large number of negative examples, and still derive a system with very few clauses. Such systems are possible to derive with the use of randomized algorithms. A randomized algorithm, with a sufficiently large number of random replications, is difficult to be trapped by a local optimal point.

The first heuristic approach, termed RA1 (for *Randomized Algorithm 1*), is proposed to solve the first research problem considered in this paper. Before the RA1 heuristic is formally presented, some new definitions and terminology are needed to be introduced next.

Definitions

C	The set of atoms in the current clause (disjunction).
A_k	An attribute (atom) such that $A_k \in A$, where A is the set of all attributes $A_1, \ldots, A_n$.
POS(A_k)	The number of all positive examples in E^+ which would be accepted if attribute A_k is included in the current clause.
NEG(A_k)	The number of all negative examples in E^- which would be accepted if attribute A_k is included in the current clause.
l	The size of the candidate list.
ITRS	The number of times the clause forming procedure is repeated.

As an illustrative example of the above definitions, consider the following sets of positive and negative examples:

$$E^+ = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix}, \quad E^- = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix}.$$

The set A of all attributes for the above set of examples is:

$$A = \{A_1, A_2, A_3, A_4, \bar{A}_1, \bar{A}_2, \bar{A}_3, \bar{A}_4\}.$$

Therefore, the POS(A_k) and the NEG(A_k) values are:

$$\begin{array}{llll} \mathrm{POS}(A_1) = 2, & \mathrm{NEG}(A_1) = 4, & \mathrm{POS}(\bar{A}_1) = 2, & \mathrm{NEG}(\bar{A}_1) = 2, \\ \mathrm{POS}(A_2) = 2, & \mathrm{NEG}(A_2) = 2, & \mathrm{POS}(\bar{A}_2) = 2, & \mathrm{NEG}(\bar{A}_2) = 4, \\ \mathrm{POS}(A_3) = 1, & \mathrm{NEG}(A_3) = 3, & \mathrm{POS}(\bar{A}_3) = 3, & \mathrm{NEG}(\bar{A}_3) = 3, \\ \mathrm{POS}(A_4) = 2, & \mathrm{NEG}(A_4) = 2, & \mathrm{POS}(\bar{A}_4) = 2, & \mathrm{NEG}(\bar{A}_4) = 4. \end{array}$$

The problem now is to derive a small set of logical clauses which would correctly classify all the above examples. Suppose that there exists a "hidden" system given by the following Boolean function:

$$(A_2 \vee A_4) \wedge (\bar{A}_2 \vee \bar{A}_3) \wedge (A_1 \vee A_3 \vee \bar{A}_4).$$

It can be easily seen that the above Boolean function correctly classifies all the previous examples. Therefore, the first problem is to accurately estimate the above "hidden" system. This is accomplished by using heuristic RA1, which is described in Figure 2.

The following theorem states an upper bound on the number of clauses which can be inferred by RA1, and it is the same as a similar result first presented in [3].

THEOREM 1. *The RA1 approach terminates within at most m_2 iterations.*

PROOF. A clause C_x can always be formed which rejects only the x^{th} negative example while accepting all other examples. For instance, if the x^{th} negative example to be rejected is $[1,0,1,0]$, then the clause which rejects the x^{th} example, while accepting all other examples, is: $(\bar{A}_1 \vee A_2 \vee \bar{A}_3 \vee A_4)$. Therefore, in Step 2 of the RA1 procedure, a clause which at best rejects only one single example from the E^- set could be formed. As result, the maximum number of clauses required to reject all the E^- examples is m_2. ∎

```
DO for ITRS number of iterations
  BEGIN;
        DO WHILE ($E^- \neq \phi$)
          $C = \phi$; {initializations}
              DO WHILE ($E^+ \neq \phi$)
                Step 1: Rank in descending order all atoms $a_i \in a$ (where $a_i$ is either $A_i$ or $\bar{A}_i$)
                        according to their POS($a_i$)/NEG($a_i$) value, if NEG($a_i$) = 0, then
                        POS($a_i$)/NEG($a_i$) = 1,000 (an arbitrarily high value);
                Step 2: Form a candidate list of the atoms which have the $l$ top highest
                        POS($a_i$)/NEG($a_i$) values;
                Step 3: Randomly choose an atom $a_k$ from the candidate list;
                Step 4: Let the set of atoms in the current clause be $C \leftarrow C \cup a_k$;
                Step 5: Let $E^+(a_k)$ be the set of members of $E^+$ accepted when $a_k$ is included
                        in the current CNF clause;
                Step 6: Let $E^+ \leftarrow E^+ - E^+(a_k)$;
                Step 7: Let $a \leftarrow a - a_k$;
                Step 8: Calculate the new POS($a_k$) values for all $a_k \in a$;
              REPEAT
          Step 9:  Let $E^-(C)$ be the set of members of $E^-$ which are rejected by $C$;
          Step 10: Let $E^- \leftarrow E^- - E^-(C)$;
          Step 11: Reset $E^+$;
        REPEAT
  END;
CHOOSE the final Boolean system among the previous ITRS systems which has the smallest number of clauses.
```

Figure 2. The RA1 heuristic.

Next, let n be the number of atoms in the data set, m_1 be the number of examples in the E^+ set and m_2 be the number of examples in the E^- set. Then Theorem 2 states the time complexity of the RA1 algorithm.

THEOREM 2. *The RA1 algorithm has a polynomial time complexity of order $O(n(m_1+m_2)m_1m_2 \times \text{ITRS})$.*

PROOF. Calculating the values of the ratios POS(A_i)/NEG(A_i), for $i = 1$ to n, requires $n(m_1 + m_2)$ simple computations. To sort out the atoms in descending order of their POS(A_i)/NEG(A_i) value we can use the "quick sort" procedure [22] which has time complexity of order $O(n \log n)$. Each clause is completely formed when all the m_1 examples in the positive set are accepted. Each Boolean function is completely formulated when all the m_2 negative examples are rejected. The whole clause forming procedure is repeated ITRS number of times. Therefore, the time complexity of the RA1 algorithm is $O((n(m_1 + m_2) + n \log n)m_1m_2\text{ITRS}) = O(n(m_1 + m_2)m_1m_2\text{ITRS})$. ∎

From the way the POS(A_k) and NEG(A_k) values were defined, some critical observations can be made. When an atom with a rather high value of the POS function is included in the CNF clause being formed, then chances are that some additional positive examples will be accepted by that clause as result of the inclusion of that atom. Similarly, atoms which correspond to low NEG values, are likely not to cause many new negative examples to be accepted as result of the inclusion of that atom in the current clause. Therefore, it makes sense to include as atoms in the CNF clause under formation, atoms which correspond to high POS values and, at the same time, to low NEG values.

In this paper the notations POS(a_i)/NEG(a_i) and POS(A_k)/NEG(A_k) will be used interchangeably to denote the same concept. For the current illustrative example, the values of the POS(A_k)/NEG(A_k) ratios are:

$$\frac{\mathrm{POS}(A_1)}{\mathrm{NEG}(A_1)} = 0.5, \qquad \frac{\mathrm{POS}\,(\bar{A}_1)}{\mathrm{NEG}\,(\bar{A}_1)} = 1.0,$$

$$\frac{\mathrm{POS}(A_2)}{\mathrm{NEG}(A_2)} = 1.0, \qquad \frac{\mathrm{POS}\,(\bar{A}_2)}{\mathrm{NEG}\,(\bar{A}_2)} = 0.5,$$

$$\frac{\mathrm{POS}(A_3)}{\mathrm{NEG}(A_3)} = 0.33, \qquad \frac{\mathrm{POS}\,(\bar{A}_3)}{\mathrm{NEG}\,(\bar{A}_3)} = 1.0,$$

$$\frac{\mathrm{POS}(A_4)}{\mathrm{NEG}(A_4)} = 1.0, \qquad \frac{\mathrm{POS}\,(\bar{A}_4)}{\mathrm{NEG}\,(\bar{A}_4)} = 0.5.$$

The above discussion illustrates the motivation for considering as possible candidates for the evaluative function, the functions: POS/NEG, POS − NEG, or some type of a weighted version of the previous two expressions. Some exploratory computational experiments indicated that the evaluative function POS/NEG was the most effective one. That is, it lead to the formation of Boolean functions with less clauses than when the other evaluative functions were considered.

The randomization of the RA1 algorithm is done as follows. In Step 2, the first l atoms with the highest value of the POS(A_k)/NEG(A_k) ratio are chosen as the members of the candidate list and an atom in the list was randomly chosen out the candidate list in Step 3. This is done in order to obtain different solutions at each iteration and prevent the system from being trapped by a locally optimal point.

In the approach of choosing a fixed value l as the size of the candidate list, there is a possibility that an atom with a very low value of POS(A_k)/NEG(A_k) ratio could be selected if the value of l is large enough (how large depends on the current data). That could occur if there are not l atoms with a sufficiently high value of the POS(A_k)/NEG(A_k) ratio. If an atom with a low value of POS(A_k)/NEG(A_k) is chosen to be included in the clause, then the clause would accept less examples from the E^+ set or accept more examples from the E^- set, or both. All these three situations should be avoided as it would lead to an increase in the number of atoms in a clause (if it accepts less examples from the E^+ set) or an increase in the number of clauses (if the atom accepts more examples from the E^- set) or both. To prevent the above situation from happening, a candidate list is formed of atoms, each of whose POS(A_k)/NEG(A_k) value is within a certain percentage, say α%, of the highest value of the POS(A_k)/NEG(A_k) value in the current candidate list. This ensures that the atom (randomly chosen out of the candidate list) to be included in the clause has a value close to the highest value of the POS(A_k)/NEG(A_k) ratios.

The above idea of using randomization in a search algorithm has been explored recently by other researchers as well. For instance, Feo and Resende [23] have successfully used randomization to solve clause satisfiability (SAT) problems. Also, in a recent book [24] Motwani and Raghavan provide a comprehensive presentation of the theory on randomized algorithms. Randomization also offers a natural and intuitive way for implementing *parallelism* in algorithms.

To obtain a system with a very small number of clauses, the whole procedure is subjected to a certain number of iterations (denoted by the value of the ITRS parameter) and the system which has the least number of disjunctions is chosen as the final inferred Boolean system.

Referring to the previous illustrative example, if $l = 3$, then the values of the 3 best POS(A_k)/ NEG(A_k) ratios are: $\{1.0, 1.0, 1.0\}$ (note that it is a coincidence that the three values are identical) which correspond to the atoms $\bar{A}_1, A_2$, and A_4, respectively. Let atom A_2 be the randomly selected atom from the candidate list. Note that atom A_2 accepts examples number 2 and 3 from the current E^+ set. Therefore, at least one more atom is required to complete the formation of the current clause. The whole process of finding a new atom (other than atom A_2 which has already been selected) with a very high value of POS/NEG is repeated. Now, suppose that the atom with a high POS/NEG value happened to be A_4. Observe that atoms A_2 and A_4, when are combined together, accept all the elements in the E^+ set. Therefore, the first clause is $(A_2 \vee A_4)$.

This clause fails to reject examples number 2, 3, and 6 in the E^- set. Therefore, examples number 2, 3, and 6 in the original E^- set constitute the reduced (and thus new) E^- set. The above process is repeated until a set of clauses are formed which, when combined together, reject all the examples in the original E^- set. Therefore, a final Boolean function for this problem could be as follows (recall that the algorithm is a randomized one and thus it does not return a deterministic solution):

$$(A_2 \vee A_4) \wedge (\bar{A}_2 \vee \bar{A}_3) \wedge (A_1 \vee A_3 \vee \bar{A}_4) .$$

A very important factor in deriving a Boolean function from positive and negative examples is the number of examples needed to infer the logic. This is also known as the *sample complexity* of a given approach. The problem of inferring a pattern with a sequence of very few new examples has been examined by Bshouty *et al.* [25], Goldman and Sloan [12], and Triantaphyllou and Soyster [26]. This is also known as the *guided learning* approach. A guided learning approach (as, for instance, the one described by [26]) can be used in conjunction with the randomized algorithm RA1 to infer Boolean functions from positive and negative data.

4. AN APPROACH FOR INFERRING A BOOLEAN FUNCTION FROM INCOMPLETE DATA

The second algorithm deals with the case in which some of the examples contain missing values. That is, now the examples are defined in the $\{0, 1, *\}^n$ space. The consequences of having missing elements in the positive, negative, or unclassifiable examples have already been discussed in Section 2.

If an example is determined as unclassifiable by the "hidden" system, then it has also to remain unclassifiable by the derived Boolean system. In other words, the property for inferring Boolean functions when unclassifiable examples are also considered (along with positive and negative examples), is that none of the examples in the E^U set should neither be accepted nor rejected by the derived system.

To help fix ideas, consider the following Boolean function:

$$(A_2 \vee A_4) \wedge (\bar{A}_2 \vee \bar{A}_3) \wedge (A_1 \vee A_3 \vee \bar{A}_4) .$$

If an example in E^U has $A_2 = A_4 = 0$, then the first clause would reject that example. This, however, should not be permissible and thus the above function *cannot* be a candidate solution, *regardless* of what are the positive and negative examples. If, on the other hand, in a particular unclassifibale example $A_2 = 0$ and $A_4 = *$, then that example is neither rejected nor accepted. Therefore, in the later scenario, the previous function is a possible candidate as far as that single unclassifiable example is concerned.

The second algorithm, termed RA2 (for *Randomized Algorithm 2*), also works in conjunction with the OCAT approach (as described in Figure 1). That is, the RA2 generates a single clause at a time. Each clause accepts all the positive examples in the E^+ set, while it does not reject any examples in the unclassifiable set E^U and it also attempts to reject a large number (but not necessarily as many as possible) of negative examples in the E^- set.

```
DO for ITRS number of iterations
        BEGIN;
PHASE I:    DO WHILE ($E^- \neq \phi$)
              $C = \phi$; {initialization}
                  DO WHILE ($C$ does not reject any example from $E^U$)
                        DO WHILE ($E^+ \neq \phi$)
                          Step 1: Rank in descending order all atoms $a_i \in a$ (where $a_i$ is either $A_i$ or $\bar{A}_i$)
                                  according to their POS($a_k$)/NEG($a_k$) value, if NEG($a_k$) = 0, then
                                  POS($a_k$)/NEG($a_k$) = 1,000 (an arbitrarily very high value);
                          Step 2: Form a candidate list of the atoms which have the $l$ top highest POS($a_i$)/
                                  NEG($a_i$) values;
                          Step 3: Randomly select an atom $a_k$ from the candidate list;
                          Step 4: Let $E^+(a_k)$ be the set of members of $E^+$ accepted when $a_k$ is included
                                  in the current clause;
                          Step 5: Let $E^+ \leftarrow E^+ - E^+(a_k)$;
                          Step 6: Let the set of atoms in the current clause be $C \leftarrow C \cup a_k$;
                    REPEAT;
                    Step 7: $C \leftarrow C \cup a_j$ where $a_j$ is any one atom with a value of "*" in
                            each of the examples in $E^U$ which were rejected by the clause $C$.
                  REPEAT;
                  Step 8: Let $E^-(C)$ be the set of members of $E^-$ which are rejected by $C$;
                  Step 9: Let $E^- \leftarrow E^- - E^-(C)$;
                  Step 10: Reset $E^+$;
            REPEAT;

PHASE II:   Denote as $E^A$ the updated set of unclassifiable examples in $E^U$ accepted by the current set of
            clauses $C_1 \wedge C_2 \wedge \cdots \wedge C_m$, where $m$ is the total number of clauses formed so far.
            DO WHILE ($E^A \neq \phi$)
              Step 11: $m \leftarrow m + 1$;
              Step 12: Form (according to the proof of Theorem 3) a clause $C_m$ which does not
                       accept the first unclassifiable example from the $E^A$ set.
            REPEAT;
      END;
CHOOSE the final Boolean system among the previous ITRS systems which has the smallest number of clauses.
```

Figure 3. The RA2 heuristic.

Subsequent clauses are formed with a reduced E^- set (comprised by the negative examples which have not been rejected so far). When all the examples in the E^- set have been rejected, then the RA2 algorithm enters its second phase. In the second phase the whole set of clauses is tested against the E^U set to satisfy the necessary Boolean system forming condition for the unclassifiable examples. That is, all the clauses when are grouped together should not accept any example from the E^U set.

Recall that from Phase I the clauses do not reject any of the unclassifiable examples. Any unclassifiable examples which are accepted by the clauses which have been formed in the first phase, are grouped into the set E^A (that is: $E^A \subseteq E^U$). The clauses which were formed in the first phase are appended with a new set of clauses which are formed in the second phase. The new clauses, when are grouped together with the first set of clauses, do not accept any example in the set E^A (i.e., $E^A = \phi$). This is accomplished in a sequential manner. Figure 3 illustrates the specific steps of the RA2 heuristic.

Let the set E^A contain examples from the original E^U set which the derived Boolean function has accepted. Therefore, the maximum number of examples in the set E^A is the same as the cardinality of the E^U set (i.e., equal to m_3). Suppose that there are still m_3' (where $m_3' \leq m_3$) unclassifiable examples in the E^A set after all the examples from the E^- set are rejected and all the clauses are tested against the examples in the E^U set for non acceptance (i.e., now $E^A \neq \phi$ and $E^- = \phi$). Then, how does the RA2 heuristic terminate? The upper bound for the terminating condition is given in the following theorem.

THEOREM 3. *If $E^- = \phi$, the maximum number of additional conjunctions in Phase II in heuristic RA2 is m_3'.*

PROOF. A clause C_x can always be formed which does not accept the x^{th} unclassifibale example (i.e., it can either reject or make unclassifiable this example) from the E^A set while accepting *all* other examples (please refer to the proof of Theorem 1 for the validity of the previous statement). Since all the negative examples have already been rejected, a maximum of m_3' new clauses can be formed which do not accept the m_3' unclassifiable examples from the E^A set. For instance, if the x^{th} example in E^A is $[1, *, 0, *]$, then the clause $C_x : (\bar{A}_1 \vee A_2 \vee A_3 \vee A_4)$ would fail to accept as well as reject the x^{th} example. ∎

If n is the number of attributes and m_1, m_2 and m_3 are the cardinalities of the E^+, E^-, and the E^U sets, respectively, then the complexity of the RA2 algorithm is stated in the following theorem.

THEOREM 4. *The time complexity of the RA2 heuristic is $O(n(m_1 + m_2)m_1m_2m_3\text{ITRS})$.*

PROOF. Calculating the values of the $\text{POS}(A_i)/\text{NEG}(A_i)$ ratios (for $i = 1$ to n) requires $n(m_1 + m_2)$ computations. To sort the atoms in descending order of their $\text{POS}(A_i)/\text{NEG}(A_i)$ value we can use the "quick sort" which is of time complexity of order $O(n \log n)$ [22]. To form a Boolean system in which each clause accepts all the m_1 positive examples, rejects none of the m_3 unclassifiable examples and the whole set of clauses is rejecting all the m_2 negative examples is of order $m_1m_2m_3$. The complexity of Phase II is m_3n. This is indicated in the second loop in Figure 3. Therefore, the complexity of the RA2 heuristic is of order $O(((n(m_1 + m_2) + n \log n)m_1m_2m_3) + m_3n)\text{ITRS}) = O(n(m_1 + m_2)m_1m_2m_3\text{ITRS})$. ∎

Next, for demonstrative purposes of the above issues consider the following illustrative example. Let the three classes of data be as follows (observe that for simplicity, any positive or negative examples with missing values has been expanded to be in $\{1, 0\}^n$):

$$E^+ = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad E^- = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \text{and} \quad E^U = \begin{bmatrix} 1 & * & 0 & * \\ 1 & 0 & * & 0 \\ * & 1 & * & 0 \end{bmatrix}.$$

Then, the $\text{POS}(a_k)$, $\text{NEG}(a_k)$, and $\text{POS}(a_k)/\text{NEG}(a_k)$ values are:

$$\text{POS}(A_1) = 1, \quad \text{NEG}(A_1) = 2, \quad \frac{\text{POS}(A_1)}{\text{NEG}(A_1)} = 0.5,$$

$$\text{POS}(A_2) = 2, \quad \text{NEG}(A_2) = 1, \quad \frac{\text{POS}(A_2)}{\text{NEG}(A_2)} = 2.0,$$

$$\text{POS}(A_3) = 1, \quad \text{NEG}(A_3) = 2, \quad \frac{\text{POS}(A_3)}{\text{NEG}(A_3)} = 0.25,$$

$$\text{POS}(A_4) = 1, \quad \text{NEG}(A_4) = 2, \quad \frac{\text{POS}(A_4)}{\text{NEG}(A_4)} = 0.5,$$

$$\text{POS}(\bar{A}_1) = 2, \quad \text{NEG}(\bar{A}_1) = 2, \quad \frac{\text{POS}(\bar{A}_1)}{\text{NEG}(\bar{A}_1)} = 1.0,$$

$$\text{POS}(\bar{A}_2) = 1, \quad \text{NEG}(\bar{A}_2) = 3, \quad \frac{\text{POS}(\bar{A}_2)}{\text{NEG}(\bar{A}_2)} = 0.33,$$

$$\text{POS}(\bar{A}_3) = 2, \quad \text{NEG}(\bar{A}_3) = 2, \quad \frac{\text{POS}(\bar{A}_3)}{\text{NEG}(\bar{A}_3)} = 0.5,$$

$$\text{POS}(\bar{A}_4) = 2, \quad \text{NEG}(\bar{A}_4) = 2, \quad \frac{\text{POS}(\bar{A}_4)}{\text{NEG}(\bar{A}_4)} = 1.0.$$

If $l = 3$, the 3 highest ratios of the $\text{POS}(a_k)/\text{NEG}(a_k)$ values are $\{2.0, 1.0, 1.0\}$ which correspond to atoms $A_2, \bar{A}_1$, and $\bar{A}_4$, respectively. Let atom A_2 be the randomly selected atom from the current candidate list. If atom A_2 is introduced into the current clause, then as result this clause will accept the first and the second examples in the E^+ set. The whole process of finding the values of $\text{POS}(a_k)/\text{NEG}(a_k)$, (with $k \neq 2$) is repeated. For the next iteration suppose that atom A_3 is chosen. When atoms A_2 and A_3 are introduced into the clause, then this clause accepts all examples in the E^+ set. This set of atoms does not reject any example in the E^U set. Therefore, the first clause is $(A_2 \vee A_3)$. This process is repeated until $E^- = \phi$ and $E^A = \phi$. Therefore, a final Boolean function for this problem could be as follows (recall that the algorithm is a randomized one and thus it does not return a deterministic solution):

$$(A_2 \vee A_3) \wedge (\bar{A}_1 \vee \bar{A}_3 \vee A_4) \wedge (\bar{A}_1 \vee \bar{A}_3).$$

5. SOME COMPUTATIONAL RESULTS

A number of computer experiments were conducted on an IBM 3090-600S mainframe computer running the VM/XA operating system, in order to investigate the effectiveness of the RA1 and RA2 heuristics on different types of problems. Some interesting results were obtained and are discussed in the next sections.

The previous two heuristics RA1 and RA2 were tested on a number of different experiments. The first type of experiments used the well known Wisconsin breast cancer database (donated by Professor Mangasarian from the University of Wisconsin, Madison, and now it can be found in the University of California at Irvine Repository of Learning Databases and Domain Theories) [27]. This database contained (at the time it was obtained) 421 examples, 224 of which were corresponding to benign (or positive) and 197 to malignant (or negative) cases. The original data were defined on nine discrete variables, each variable assuming values from the integer set [1,10]. These data were converted into their equivalent binary data. That is, each variable was converted into four binary variables and thus the transformed database was defined on $36 (= 4 \times 9)$ binary variables.

Additionally, the RA1 heuristic was compared with the branch-and-bound method (described by [4]) by generating a challenging set of large random test problems. The first large random data set contained 18,120 examples defined on 15 atoms with a varied ratio of positive and negative examples. Note, that this set was almost 50 times larger than the size of the breast cancer data. A second large data set contained 3,750 examples defined on 14 atoms (binary variables).

The measures of performance considered in these tests are of three types:

(i) the accuracy of the derived system (Boolean function),
(ii) the number of clauses (CNF disjunctions) in the derived system, and
(iii) the CPU time required to derive a solution.

The method of determining accuracy of a proposed system (i.e., Boolean function) was defined in two different ways. When the Wisconsin breast cancer database was used, the accuracy of a solution was defined by comparing the way an inferred system (which was derived when a part of the available data was used as the training set) and the system derived when the entire data base is used, classified a random collection of 10,000 examples. Note, that this approach is similar to the testing procedures used in [1,3,4,21]. For the other cases the testing procedure was different. First a collection of random examples was formed. Next, a random Boolean function was formed and the previous examples were classified according to that function as either positive or negative examples. That function played the role of the *oracle* or "*hidden system*". Then, the goal of the inference algorithms was to infer a close approximation of that "*hidden*" function. Therefore, in this case the notion of accuracy was defined as the percentage of the times the proposed and the "*hidden*" system agreed in classifying a random collection of 10,000 examples.

Moreover, for testing the two heuristics on the breast cancer data, two categories of Boolean functions were derived. The first category of Boolean functions used the benign set as the E^+ set and the malignant set as the E^- set. This category of Boolean systems was denoted as system S1. The second category of systems treated the benign set as the E^- set and the malignant set as the E^+ set. This category of Boolean systems was denoted as system S2. The purpose of formulating two categories of systems (i.e., S1 and S2) was to study the effect of the number of examples (recall that the benign and the malignant observations were 224 and 197, respectively) on the accuracies and the number of clauses in the derived systems.

Since it was required that the number of clauses be kept at a very small level, the whole clause forming process was repeated a certain number of times (defined as the value of the parameter ITRS in Figures 2 and 3). The value of ITRS equal to 150 was determined after a brief pilot study. The results are tabulated below.

ITRS Value	Number of Clauses in System S1	System S2
50	14.40	23.52
100	12.20	20.69
150	8.86	19.92
200	8.84	20.12
500	8.83	19.79
1,000	8.82	19.78

These results suggest that higher values of ITRS did not generated much fewer clauses (although the CPU requirement is higher now). Thus, the value of ITRS equal to 150 is a reasonable one. Obviously, this empirical value cannot be generalized since for different data a different ITRS value may be more appropriate. Therefore, we suggest that a pilot study to be undertaken before an ITRS value is decided. Finally, please recall that the running time of the heuristic is directly proportional to the value of ITRS. This is indicated in the complexities of the RA1 and RA2 heuristics as seen in Figures 1 and 3, respectively.

For the randomization of the heuristics, a candidate list of a few atoms was formed among which the representative atom was randomly chosen. Only those atoms were chosen in the candidate list whose POS/NEG value was within a certain percentage, say $\alpha\%$, of the maximum POS/NEG value in the candidate list. This would assure that the atoms which are finally chosen are in a near neighborhood of the atom with the maximum value of the POS/NEG ratio in the candidate list. A good value of $\alpha\%$ seemed to be equal to 75% as it resulted in the highest accuracy in some exploratory computational experiments. These experiments are described in more detail in the following sections.

5.1. Results for the RA1 Algorithm on the Cancer Data

The results for Problem 1 (i.e., inference of a Boolean function with complete data) are presented in Table 1. The number of replications per case was equal to 50. The numbers in the parentheses indicate the standard deviations of the various observations. Individual accuracies for the benign and the malignant tumors are also presented. For instance, B(S1), M(S1), B(S2) and M(S2) represent benign and malignant accuracies for systems S1 and S2, respectively. Figure 4 shows the relationship of accuracy on the percentage of data used. With approximately 10% of the data used as the training data, accuracy of approximately 88% was achieved with system S1 while a higher accuracy rate of 92% was achieved with system S2. A peak accuracy of approximately 92% was obtained with system S1 while a peak of 94% was obtained with system S2. Figure 5 shows the number of clauses which were derived. The maximum number of clauses is nine for system S1 and 20 for system S2.

On the average, the time required to infer the nine clauses was *300 CPU seconds.* Resende and Feo [28] had reported that their SAT approach took more than *27,000 CPU seconds* (on a VAX system, which also is 4–5 times slower than the IBM 3090-600S computer used in our

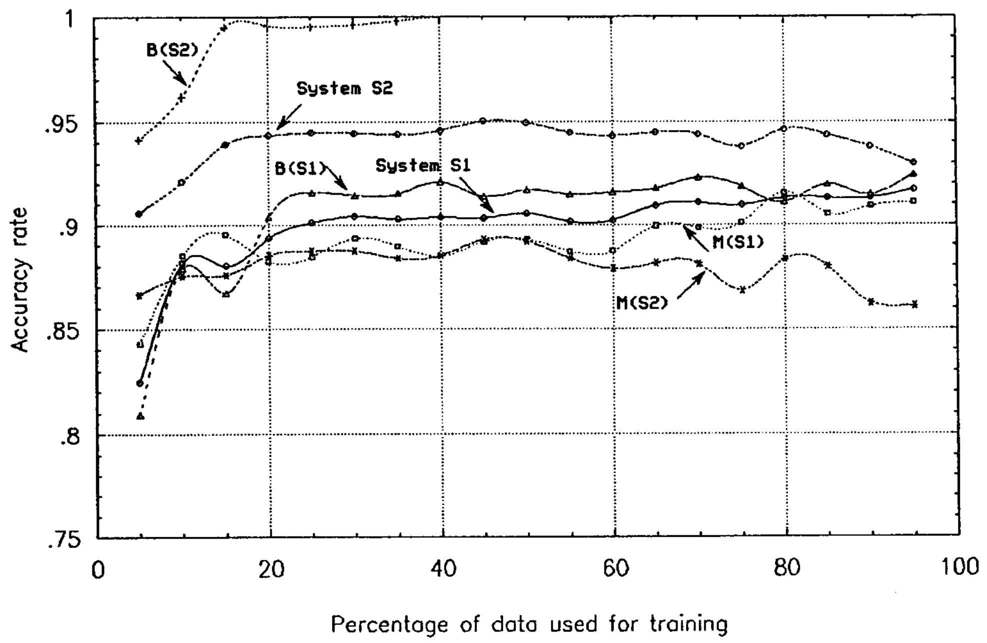

Figure 4. Accuracy rates for Systems S1 and S2 when heuristic RA1 is used on the Wisconsin Breast Cancer Data.

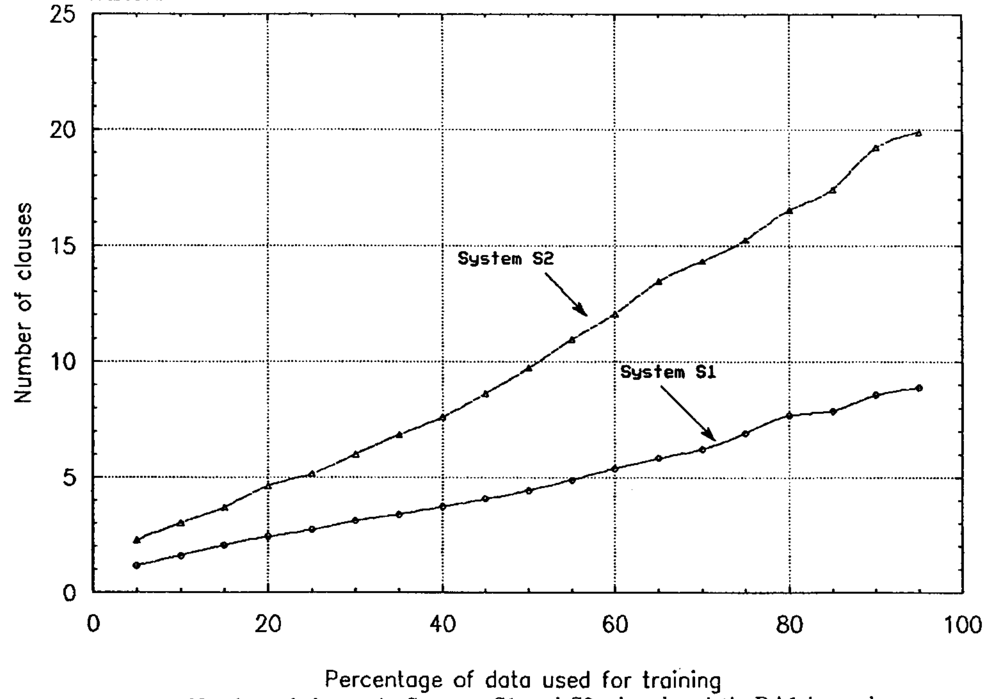

Figure 5. Number of clauses in Systems S1 and S2 when heuristic RA1 is used on the Wisconsin Breast Cancer Data.

tests) without being able to infer the nine clauses from the breast cancer database. Some typical Boolean functions for systems S1 and S2 are given in Figure 6. These clauses were derived from the entire set of cancer data. Recall that the cancer data had nine discrete valued attributes which were converted into 36 binary valued attributes, $A_1, \ldots, A_{36}$. System S2 was derived after treating the malignant examples as the E^+ data set and the benign examples as the E^- data set. As such, the set of clauses in system S2 approximates the complement of system S1. This is seen in Figure 6 where system S1 has nine clauses, each clause containing on an average ten atoms, whereas system S2 contains 20 clauses, each clause having, on the average, four atoms.

System S1:		
	Clause 1:	$A_1 \vee A_9 \vee A_{13} \vee A_{17} \vee A_{21} \vee A_{22} \vee A_{25} \vee A_{26} \vee A_{29} \vee A_{33} \vee A_{35}$
	Clause 2:	$A_1 \vee A_2 \vee A_9 \vee A_{13} \vee A_{15} \vee A_{21} \vee A_{25} \vee A_{30} \vee A_{33}$
	Clause 3:	$A_1 \vee A_3 \vee A_8 \vee A_9 \vee A_{17} \vee A_{21} \vee A_{25} \vee A_{33} \vee A_{35} \vee \bar{A}_{16}$
	Clause 4:	$A_1 \vee A_9 \vee A_{17} \vee A_{21} \vee A_{23} \vee A_{25} \vee A_{33} \vee A_{35} \vee \bar{A}_{16}$
	Clause 5:	$A_3 \vee A_9 \vee A_{12} \vee A_{17} \vee A_{21} \vee A_{25} \vee A_{33} \vee A_{35}$
	Clause 6:	$A_1 \vee A_{11} \vee A_{13} \vee A_{17} \vee A_{22} \vee A_{25} \vee A_{29} \vee A_{34} \vee A_{35} \vee \bar{A}_{12} \vee \bar{A}_{16} \vee \bar{A}_{20} \vee \bar{A}_{36}$
	Clause 7:	$A_{13} \vee A_{15} \vee A_{17} \vee A_{22} \vee A_{24} \vee A_{25} \vee A_{27} \vee A_{29} \vee A_{34} \vee \bar{A}_{16} \vee \bar{A}_{36}$
	Clause 8:	$A_4 \vee A_{23} \vee A_{35} \vee \bar{A}_2 \vee \bar{A}_{18} \vee \bar{A}_{20} \vee \bar{A}_{36}$
	Clause 9:	$A_8 \vee A_{10} \vee A_{15} \vee A_{21} \vee A_{23} \vee \bar{A}_{16} \vee \bar{A}_{20} \vee \bar{A}_{32}$
System S2:		
	Clause 1:	$A_{16} \vee A_{24} \vee \bar{A}_{21}$
	Clause 2:	$A_{24} \vee \bar{A}_{23} \vee \bar{A}_{26}$
	Clause 3:	$A_5 \vee \bar{A}_{21} \vee \bar{A}_{26}$
	Clause 4:	$A_{12} \vee A_{26} \vee \bar{A}_1$
	Clause 5:	$A_6 \vee \bar{A}_{18} \vee \bar{A}_{19}$
	Clause 6:	$\bar{A}_9 \vee \bar{A}_{26}$
	Clause 7:	$A_{10} \vee \bar{A}_{13}$
	Clause 8:	$A_{30} \vee \bar{A}_3 \vee \bar{A}_{10} \vee \bar{A}_{24}$
	Clause 9:	$A_4 \vee A_8 \vee \bar{A}_{11} \vee \bar{A}_{23}$
	Clause 10:	$A_{15} \vee \bar{A}_7 \vee \bar{A}_{10} \vee \bar{A}_{22}$
	Clause 11:	$A_{21} \vee \bar{A}_7 \vee \bar{A}_{14} \vee \bar{A}_{16}$
	Clause 12:	$A_6 \vee \bar{A}_{10} \vee \bar{A}_{23}$
	Clause 13:	$A_{12} \vee \bar{A}_{15} \vee \bar{A}_{22}$
	Clause 14:	$A_8 \vee A_{16} \vee \bar{A}_{30}$
	Clause 15:	$A_5 \vee A_{13} \vee \bar{A}_2 \vee \bar{A}_{11} \vee \bar{A}_{26}$
	Clause 16:	$A_3 \vee A_5 \vee A_{15} \vee A_{27} \vee \bar{A}_6$
	Clause 17:	$A_5 \vee A_{13} \vee A_{18} \vee \bar{A}_6 \vee \bar{A}_{28}$
	Clause 18:	$A_3 \vee A_{13} \vee \bar{A}_7 \vee \bar{A}_{25}$
	Clause 19:	$A_3 \vee A_{13} \vee A_{21} \vee A_{32} \vee \bar{A}_{11} \vee \bar{A}_{20}$
	Clause 20:	$A_3 \vee A_{15} \vee A_{21} \vee \bar{A}_7 \vee \bar{A}_{28} \vee \bar{A}_{30}$

Figure 6. Clauses for Systems S1 and S2 when the entire Breast Canter data are used.

5.2. Results for the RA2 Algorithm on the Cancer Data with Some Missing Values

The results for Problem 2 (i.e., inference of a Boolean function with incomplete data) are presented in Table 2. Heuristic RA2 was used to solve this probelm. The unclassifiable data were generated by "covering" (i.e., masking out) the actual values of some elements in some random examples taken from the cancer data base. When the covered values were enough not to allow for the classification of that example, that example was introduced into the E^U set and the covered attributes were assigned missing values. Two systems of rules were inferred. System SA was inferred with only the positive (i.e:, the E^+) and the negative (i.e, the E^-) data sets. On the other hand, system SB was inferred from E^+, E^- *as well as* the unclassifiable (i.e., the E^U) data set. The reason for doing this was to compare the relative benefit of including the unclassifiable data sets as opposed to inferring a system of rules without the inclusion of the unclassifiable data.

The number of replications for each case was also equal to 50 and the numerical resulst are presented in Table 2. That number of replications produced rather acceptable confidence intervals and it was limited due to the excessive CPU time requirements. The individual benign and malignant accuracies for systems SA and SB are denoted in Table 2 by B(SA), M(SA) and B(SB), M(SB), respectively. The graphs for the accuracies and the number of clauses derived by

Table 1. Results using the RA1 algorithm on the Wisconsin Breast Cancer database.

% Data	# of Rules (S1)	# of Rules (S2)	Accuracy (S1)	Accuracy B(S1)	Accuracy M(S1)	Accuracy (S2)	Accuracy B(S2)	Accuracy M(S2)
5.0	1.18 (0.38)	2.28 (0.49)	0.8248 (0.0584)	0.8095 (0.1074)	0.8430 (0.1060)	0.9061 (0.0680)	0.9413 (0.1012)	0.8666 (0.1078)
10.0	1.60 (0.57)	3.02 (0.55)	0.8816 (0.0319)	0.8791 (0.0573)	0.8852 (0.0548)	0.9211 (0.0521)	0.9618 (0.0730)	0.8752 (0.0605)
15.0	2.06 (0.54)	3.70 (0.67)	0.8801 (0.0271)	0.8670 (0.0593)	0.8955 (0.0348)	0.9391 (0.0308)	0.9948 (0.0197)	0.8756 (0.0620)
20.0	2.42 (0.53)	4.62 (0.75)	0.8934 (0.0229)	0.9039 (0.0401)	0.8824 (0.0337)	0.9435 (0.0267)	0.9956 (0.0145)	0.8851 (0.0482)
25.0	2.72 (0.60)	5.14 (0.89)	0.9010 (0.0168)	0.9158 (0.0303)	0.8845 (0.0359)	0.9446 (0.0200)	0.9950 (0.0157)	0.8873 (0.0407)
30.0	3.12 (0.71)	6.00 (1.31)	0.9043 (0.0138)	0.9146 (0.0290)	0.8933 (0.0330)	0.9445 (0.0183)	0.9959 (0.0113)	0.8872 (0.0350)
35.0	3.40 (0.72)	6.84 (1.10)	0.9027 (0.0170)	0.9151 (0.0295)	0.8896 (0.0357)	0.9438 (0.0177)	0.9977 (0.0081)	0.8837 (0.0332)
40.0	3.74 (0.66)	7.60 (1.34)	0.9038 (0.0146)	0.9209 (0.0232)	0.8848 (0.248)	0.9457 (0.185)	0.9997 (0.0015)	0.8850 (0.0369)
45.0	4.06 (0.88)	8.60 (1.71)	0.9034 (0.0183)	0.9140 (0.0275)	0.8918 (0.0327)	0.9502 (0.0155)	1.0000 (0.0000)	0.8928 (0.0321)
50.0	4.44 (0.78)	9.70 (1.57)	0.9056 (0.0175)	0.9170 (0.0315)	0.8929 (0.0372)	0.9494 (0.0166)	0.9998 (0.0011)	0.8920 (0.0338)
55.0	4.90 (0.88)	10.94 (1.63)	0.9015 (0.0160)	0.9149 (0.0341)	0.8870 (0.0325)	0.9448 (0.0164)	0.9996 (0.0020)	0.8837 (0.0321)
60.0	5.40 (0.92)	12.06 (1.54)	0.9022 (0.0229)	0.9159 (0.0372)	0.8872 (0.0450)	0.9432 (0.0183)	0.9996 (0.0029)	0.8787 (0.0376)
65.0	5.84 (1.03)	13.46 (2.02)	0.9095 (0.0228)	0.9177 (0.0280)	0.8995 (0.0400)	0.9448 (0.0197)	1.0000 (0.0000)	0.8813 (0.0395)
70.0	6.24 (0.93)	14.34 (2.06)	0.9109 (0.0217)	0.9230 (0.0383)	0.8985 (0.0419)	0.9440 (0.0211)	0.9997 (0.0021)	0.8810 (0.0428)
75.0	6.92 (1.07)	15.26 (1.71)	0.9096 (0.0255)	0.9186 (0.0397)	0.9009 (0.0364)	0.9381 (0.0238)	1.0000 (0.0000)	0.8683 (0.0477)
80.0	7.68 (0.88)	16.54 (1.56)	0.9132 (0.0321)	0.9110 (0.0459)	0.9156 (0.0502)	0.9462 (0.0239)	1.0000 (0.0000)	0.8830 (0.0491)
85.0	7.88 (0.99)	17.44 (1.73)	0.9131 (0.0399)	0.9197 (0.0520)	0.9054 (0.0583)	0.9436 (0.0290)	1.0000 (0.0000)	0.8798 (0.0607)
90.0	8.56 (0.75)	19.22 (1.71)	0.9131 (0.0464)	0.9147 (0.0590)	0.9091 (0.0714)	0.9379 (0.0330)	1.0000 (0.0000)	0.8623 (0.0771)
95.0	8.88 (0.86)	19.90 (1.81)	0.9171 (0.0725)	0.9242 (0.0810)	0.9109 (0.1031)	0.9299 (0.0536)	1.0000 (0.0000)	0.8608 (0.0971)

the RA2 algorithm with and without the inclusion of the unclassifiable data set E^U, are shown in Figures 7 and 8, respectively. As it was anticipated, the accuracy obtained with the inclusion of the unclassifiable data set E^U is *always higher* than the corresponding accuracy obtained without the inclusion of the E^U data set. Therefore, it is indicated in these experiments that the use of unclassifiable data set, along with the positive and negative data sets, indeed improves the quality of the improved Boolean system. That is, the inferred system is a better approximation of the "*hidden*" system.

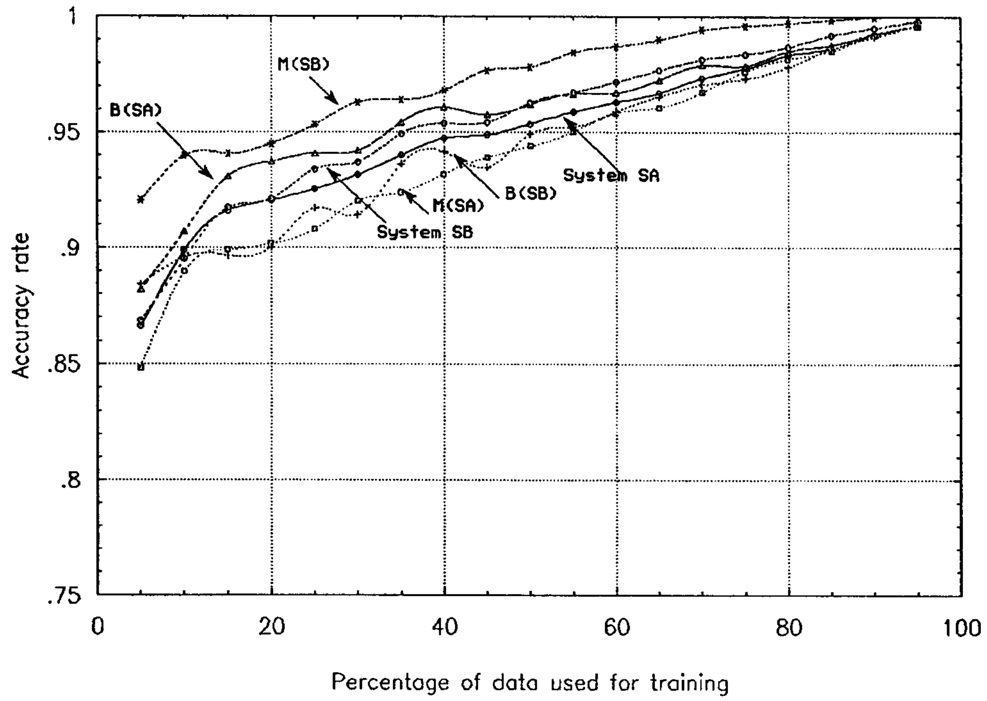

Figure 7. Accuracy rates for Systems SA and SB when heuristic RA2 is used on the Wisconsin Breast Cancer Data.

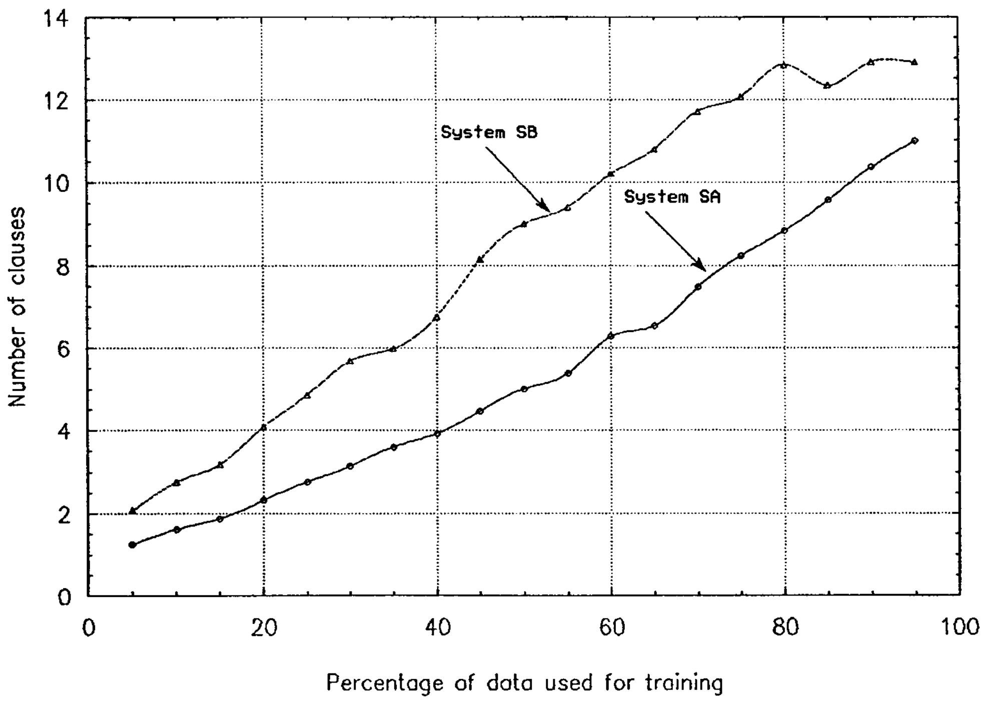

Figure 8. Number of clauses in Systems SA and SB when heuristic RA2 is used on the Wisconsin Breast Cancer Data.

5.3. Comparison of the RA1 Algorithm and the B&B Method Using Large Random Data Sets

To compare the RA1 heuristic with the B&B method, a large data set was randomly generated and used for clause inference. The difficulty of the problem is determined not only by the number of examples, but also by the percentage of examples used as compared to the total possible number of examples. For n atoms, the total number of possible distinct examples is 2^n. The problem

Table 2. Results using the RA2 algorithm on the Wisconsin Breast Cancer database.

% Data	# of Rules (SA)	# of Rules (SB)	Accuracy (SA)	Accuracy B(SA)	Accuracy M(SA)	Accuracy (SB)	Accuracy B(SB)	Accuracy M(SB)
5.0	1.26 (0.48)	2.08 (0.63)	0.8662 (0.0513)	0.8822 (0.0857)	0.8482 (0.0898)	0.8687 (0.0423)	0.8843 (0.0955)	0.9206 (0.0475)
10.0	1.62 (0.49)	2.76 (0.62)	0.8989 (0.0306)	0.9070 (0.0428)	0.8898 (0.0645)	0.8951 (0.0280)	0.8961 (0.0545)	0.9397 (0.0227)
15.0	1.88 (0.38)	3.18 (0.68)	0.9160 (0.0180)	0.9309 (0.0361)	0.8992 (0.0312)	0.9172 (0.0201)	0.8967 (0.0460)	0.9407 (0.0216)
20.0	2.32 (0.61)	4.08 (0.84)	0.9206 (0.0190)	0.9373 (0.0299)	0.9018 (0.0403)	0.9211 (0.0221)	0.9002 (0.0442)	0.9449 (0.0231)
25.0	2.76 (0.68)	4.84 (0.86)	0.9253 (0.0151)	0.9409 (0.0253)	0.9078 (0.0356)	0.9340 (0.0151)	0.9171 (0.0329)	0.9531 (0.0140)
30.0	3.14 (0.75)	5.68 (0.97)	0.9316 (0.0126)	0.9418 (0.0259)	0.9202 (0.0240)	0.9371 (0.0133)	0.9143 (0.0278)	0.9630 (0.0162)
35.0	3.60 (0.94)	5.98 (1.10)	0.9399 (0.0112)	0.9542 (0.0203)	0.9239 (0.0228)	0.9492 (0.0089)	0.9363 (0.0168)	0.9640 (0.0141)
40.0	3.92 (0.84)	6.76 (0.93)	0.9472 (0.0088)	0.9608 (0.0148)	0.9319 (0.210)	0.9538 (0.0089)	0.9416 (0.0174)	0.9680 (0.0135)
45.0	4.46 (1.08)	8.14 (1.15)	0.9488 (0.0104)	0.9575 (0.0165)	0.9390 (0.0167)	0.9543 (0.0104)	0.9347 (0.0127)	0.9768 (0.0122)
50.0	5.00 (0.98)	9.00 (1.36)	0.9536 (0.0087)	0.9620 (0.0151)	0.9439 (0.0150)	0.9627 (0.0102)	0.9492 (0.0189)	0.9783 (0.0094)
55.0	5.38 (1.09)	9.40 (1.11)	0.9588 (0.0088)	0.9666 (0.0128)	0.9501 (0.0169)	0.9672 (0.0088)	0.9522 (0.0164)	0.9845 (0.0092)
60.0	6.28 (1.10)	10.20 (1.25)	0.9630 (0.0091)	0.9671 (0.0136)	0.9583 (0.0141)	0.9718 (0.0077)	0.9588 (0.0150)	0.9871 (0.0076)
65.0	6.54 (1.08)	10.78 (1.38)	0.9669 (0.0079)	0.9726 (0.0111)	0.9607 (0.0160)	0.9768 (0.0071)	0.9655 (0.0139)	0.9899 (0.0082)
70.0	7.48 (1.30)	11.70 (1.00)	0.9734 (0.0072)	0.9790 (0.0104)	0.9673 (0.0126)	0.9815 (0.0062)	0.9705 (0.0115)	0.9941 (0.0053)
75.0	8.24 (1.05)	12.06 (1.24)	0.9775 (0.0070)	0.9788 (0.0121)	0.9763 (0.0122)	0.9838 (0.0056)	0.9733 (0.0112)	0.9959 (0.0043)
80.0	8.84 (1.21)	12.84 (1.17)	0.9833 (0.0080)	0.9850 (0.0106)	0.9818 (0.0113)	0.9869 (0.0058)	0.9782 (0.0101)	0.9970 (0.0040)
85.0	9.56 (1.17)	12.34 (0.97)	0.9862 (0.0055)	0.9875 (0.0069)	0.9852 (0.0090)	0.9917 (0.0048)	0.9860 (0.0085)	0.9984 (0.0032)
90.0	10.36 (0.97)	12.90 (0.83)	0.9920 (0.0043)	0.9925 (0.0059)	0.9919 (0.0065)	0.9950 (0.0033)	0.9912 (0.0058)	0.9996 (0.0014)
95.0	10.98 (0.97)	12.90 (0.92)	0.9960 (0.0025)	0.9960 (0.0035)	0.9960 (0.0040)	0.9980 (0.0023)	0.9964 (0.0039)	0.9999 (0.0007)

would be easy when there are very few examples or when the number of examples approaches 2^n. The upper limit on the number of examples was limited by the dynamic allocation of the random access memory (RAM memory) of the IBM 3090-600S mainframe computer at LSU. Thus, a maximum of 18,120 examples were randomly generated for the purpose of these computational experiments when the number of attributes was set equal to 15. A maximum of 32,768 distinct examples are possible when n is equal to 15. Therefore, this data set is a representation of one of the difficult problems possible with n equal to 15.

The RA1 algorithm terminates when all the clauses, when are put together, reject the entire set of negative examples denoted by E^-. A critical question is what happens if none of the clauses formed in the local search are able to reject even a single negative examples. In the proof of Theorem 1 it was shown that a clause can always be formed which rejects exactly one negative

```
DO WHILE ($E^- \neq \phi$)
        Call Procedure RA1 to form a single clause.
        If no examples from the $E^-$ set are rejected by that clause then:
                Call the B&B method to form a single clause.
REPEAT;
```

Figure 9. Using the RA1 heuristic in conjunction with the B&B method.

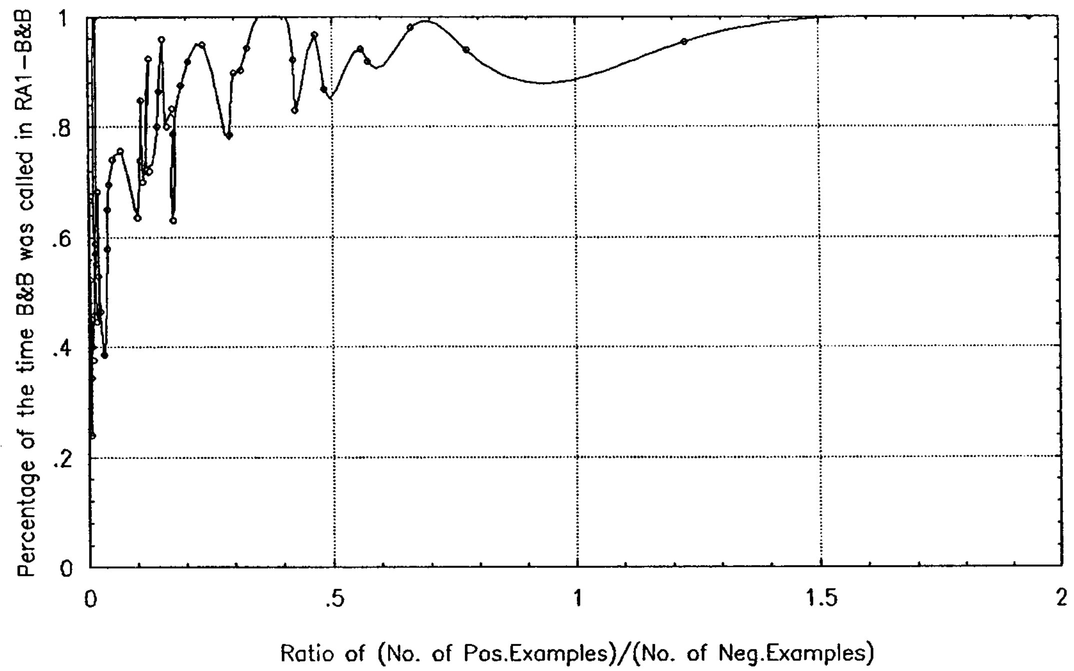

Figure 10. Percentage of the time the B&B was invoked in the combined RA1-B&B method.

example while accepting all positive examples. However, a disadvantage of that approach is that only one negative example would be rejected at a time by the inclusion of such one new clause. If the number of negative examples is large and if this boundary condition is initiated often, then the system could end up with an exceptionally large number of (degenerative) clauses.

A possible alternative solution to this situation is the use of the RA1 heuristic in conjunction with the B&B method in [4]. The B&B method always guarantees to return a clause, which most of the time, rejects more than just a single negative example. However, there is also a certain tradeoff to this implementation. The B&B method has an exponential CPU time complexity. Hence, for large size data sets, it has the potential to take large amounts of CPU time. Figure 9 best describes the idea of using the RA1 algorithm in conjunction with the B&B method (or for that matter, with any other single clause inference method).

Computational results are presented in Tables 3 and 4 which compare the combined RA1 heuristic and the B&B method with the stand alone B&B method, respectively. Two sizes of random data sets were used in these tests. One set of data contained a total of 18,120 examples with a varying ratio of the positive to the negative examples (See also Table 3). The number of atoms in this data set was equal to 15. In the other data set, the total number of examples was equal to 3,750. The number of atoms in this case was set equal to 14 (see also Table 4). These numbers of examples were fixed after determining the RAM memory restrictions of the IBM 3090-600S mainframe computer system at LSU.

Besides the CPU times consumed by the RA1 heuristic and the B&B approach in the RA1-B&B combination, the number of clauses inferred by each component method was also recorded.

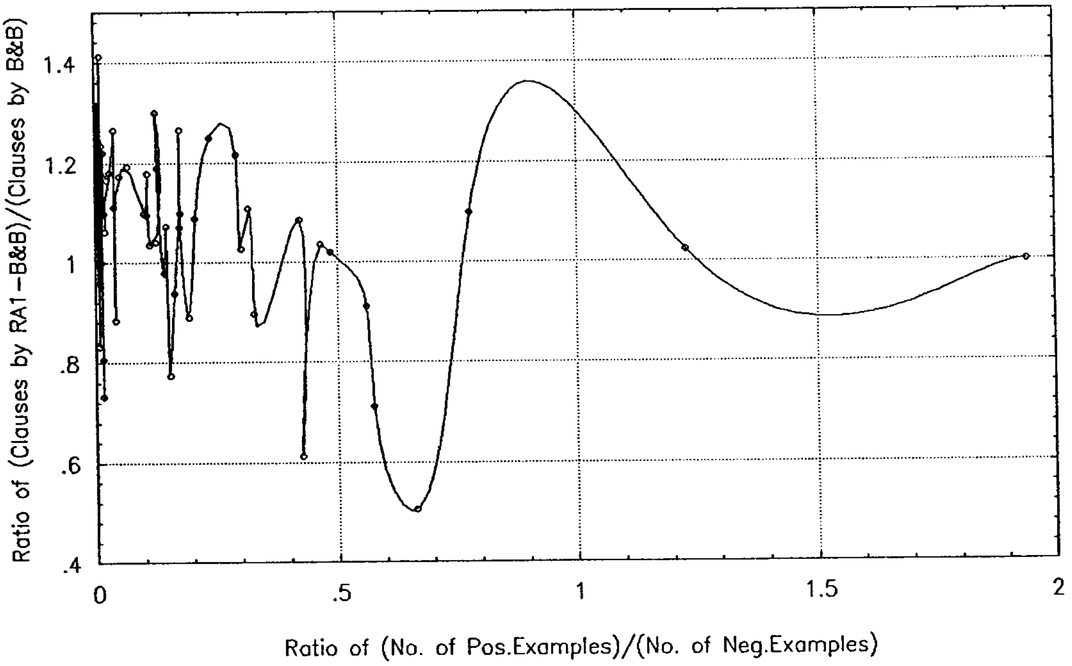

Figure 11. Ratio of the number of clauses by the RA1-B&B method and the number of clauses by the stand alone B&B method.

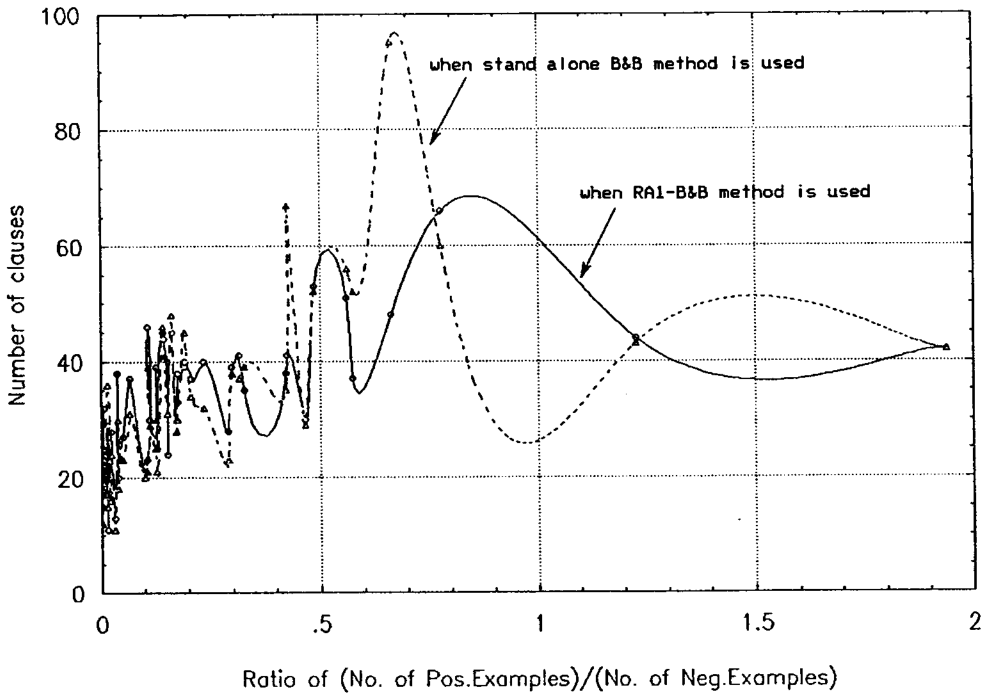

Figure 12. Number of clauses by the stand alone B&B and the RA1-B&B method.

Moreover, the number of times the boundary condition was invoked (i.e., the number of times the B&B method was used) in the RA1-B&B method, was recorded as well.

As it can be seen from the previous computational results, the combination of the RA1 heuristic with the B&B method performed significantly better than the stand alone B&B method when one focuses on the CPU times. Figure 10 shows the percentage of times the B&B method was invoked when it was combined with the RA1 heuristic. On the average, the B&B method was called approximately 60% of the time in the combined RA1-B&B approach.

Table 3. Comparison of the RA1 algorithm with the B&B method (number of examples 18,120; number of atoms = 15).

E^+	E^-	Rules by B&B in RA1/B&B	Rules in RA1	Rules in B&B	CPU Time RA1 (in seconds)	CPU Time B&B (in seconds)
264	17,856	11	15	15	267	267
690	17,430	13	20	18	614	1,265
730	17,390	16	23	26	2,243	3,302
856	17,264	20	27	23	5,781	10,983
1,739	16,381	17	23	21	5,266	6,244
1,743	16,377	36	45	46	3,442	6,016
1,773	16,347	39	46	39	5,150	10,020
2,013	16,107	24	26	25	2,058	2,000
2,298	15,822	38	44	41	4,777	4,891
2,396	15,724	23	24	31	2,816	2,583
2,400	15,720	36	45	48	3,719	4,827
2,913	15,207	35	40	45	4,344	4,532
3,090	15,030	34	37	34	4,889	4,945
3,459	14,661	38	40	32	12,187	14,547
3,574	14,546	34	41	67	4,980	9,245
3,917	14,203	46	53	52	10,588	12,232
4,781	13,339	47	48	95	10,243	19,475
5,750	12,370	29	30	29	7,959	7,944
6,503	11,617	48	51	56	5,316	9,688
6,608	11,512	34	37	52	3,887	12,632
6,989	11,131	62	66	60	16,719	17,626
9,981	8,139	42	44	43	12,232	12,146
10,554	7,566	42	42	42	12,681	12,523

E^+:	Number of Positive Examples.
E^-:	Number of Negative Examples.
Rules by B&B in RA1/B&B:	Number of rules contributed by B&B method in the RA1/B&B combined approach.
Rules in RA1:	Number of rules inferred by RA1 heuristic.
Rules in B&B:	Number of rules inferred by the B&B method.
CPU Time RA1:	CPU time taken by the RA1 heuristic in seconds.
CPU Time B&B:	CPU time taken by the B&B method in seconds.

Figure 11 depicts the ratio of the number of rules returned by the combination of the RA1-B&B methods as compared to the stand alone B&B method. As it can be seen from that figure, the number of rules returned by the combined methods is comparable to the number of rules returned when the more greedy (and by far more CPU time demanding) B&B is used alone. Figure 12 depicts the absolute values of the previous numbers of rules.

Figure 13 shows the ratio of the time taken by the stand alone B&B method to the time taken by the combined RA1-B&B method. These results indicate the relative benefits of using the proposed combined approach (i.e., the RA1-B&B method) as compared to the earlier stand alone methods (i.e., the stand alone RA1 or the stand alone B&B methods). The CPU time performance of the combined RA1-B&B methods, when it was tested in terms of the previous computational experiments was, on the average, two to three times faster as compared to the stand alone B&B method, with sometimes as much as being six times faster. Finally, Figure 14 shows the absolute values of the time taken by the two methods.

Table 4. Comparison of the RA1 algorithm with the B&B method (number of examples 3,750; number of atoms = 14).

E^+	E^-	Rules by B&B in RA1/B&B	Rules in RA1	Rules in B&B	CPU Time RA1 (in seconds)	CPU Time B&B (in seconds)
10	3,740	10	15	12	0.48	1.5
14	3,736	8	20	18	44	89
18	3,752	12	23	22	51	172
19	3,731	6	25	19	12	70
23	3,727	11	32	25	75	125
23	3,727	9	20	20	99	206
30	3,720	7	20	24	76	202
33	3,717	9	24	20	112	299
40	3,710	11	24	17	23	28
47	3,703	12	21	17	73	97
52	3,698	10	17	17	38	53
53	3,697	16	29	36	519	1,218
62	3,688	10	22	18	16	146
65	3,685	12	27	23	132	374
67	3,683	15	22	20	586	739
77	3,673	9	17	16	90	354
88	3,662	13	28	24	312	1,303
112	3,638	5	13	11	50	57
140	3.610	22	38	30	1,735	1,867
233	3,517	28	37	31	1,416	2,487
345	3,405	14	22	20	818	863
379	3,371	21	30	29	587	621
419	3,331	28	39	30	596	754
425	3,325	28	25	21	1,149	1,266
552	3,198	25	30	28	534	704
558	3,192	24	38	30	704	1,407
558	3,192	26	33	30	774	1,408
846	2,904	22	28	23	2,812	3,171
864	2,886	35	39	38	968	1,487
899	2,851	37	41	37	1,620	2,197
924	2,826	33	35	39	1,502	2,379
1,112	2,638	35	38	35	1,183	1,020

Note: The legend for this table is the same as for Table 3 shown on the previous page.

At present, previously obtained benchmark results which take into consideration unclassifiable examples are not available. Hence, computational results obtained with the RA2 algorithm were not compared with any other set of resulst. A logical extension of the work done so far is to develop a B&B method which would take into consideration the unclassifiable examples, and compare the RA2 alogorithm with this modified B&B method. It is quite possible that the RA2 algorithm in conjunction with the modified B&B method would perform better than the present method (i.e., the stand alone RA2 heuristic). Once this is developed, the combination of the RA2 algorithm and the modified B&B method could be tested on a large set of random examples to indicate with a high degree of certainty that the inclusion of unclassifiable examples indeed enhances the accuracy of the inferred rules.

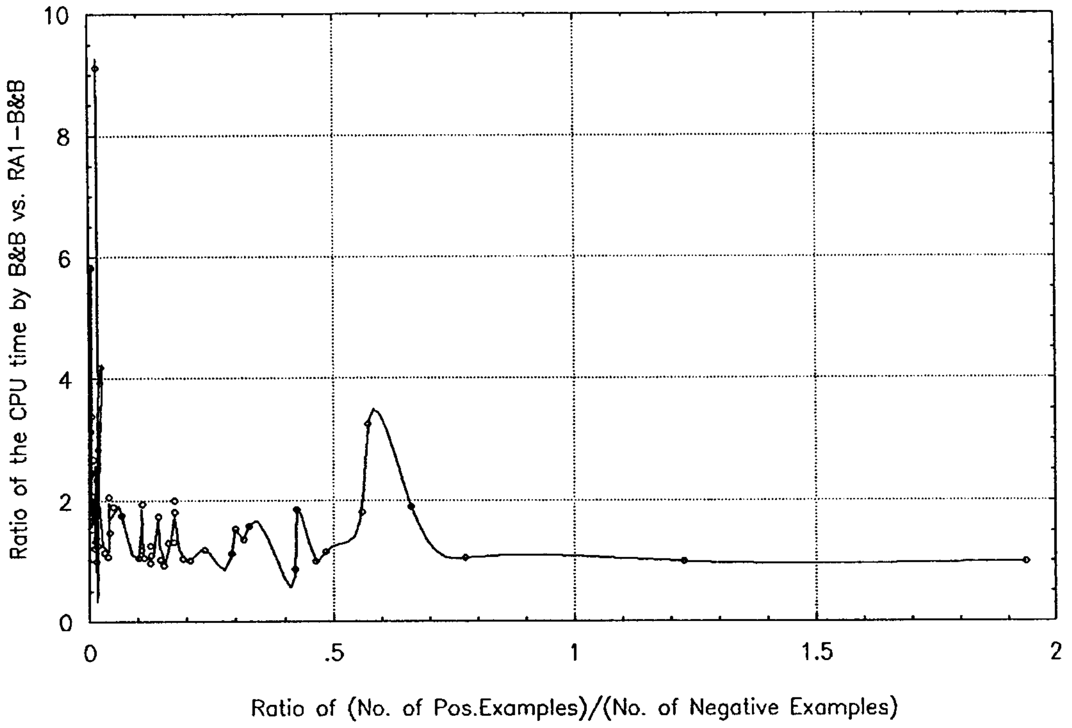

Figure 13. Ratio of the time used by the stand alone B&B and the time used by the RA1-B&B method.

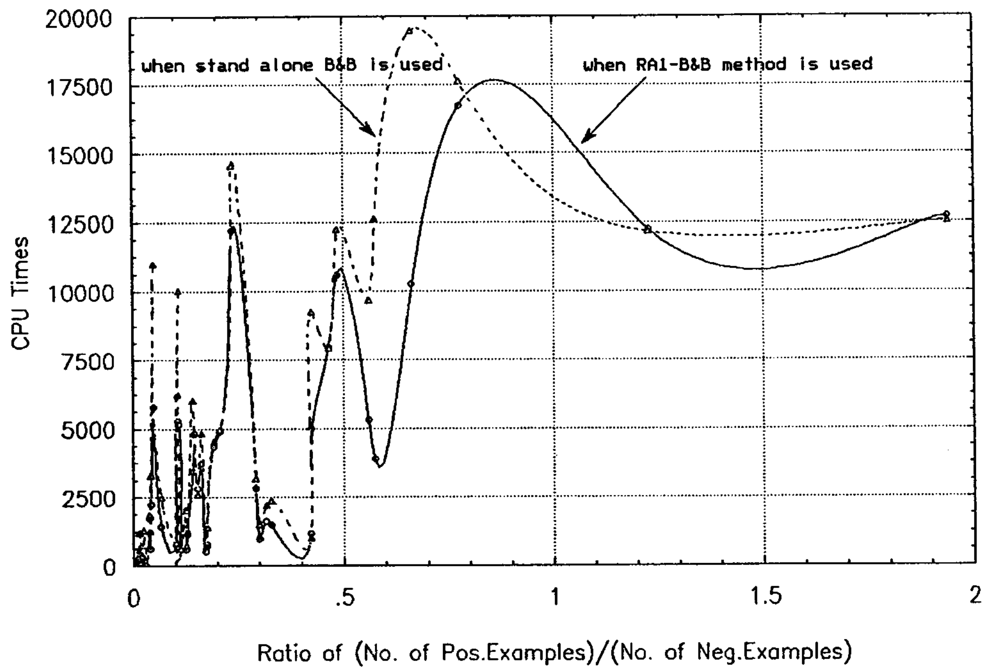

Figure 14. CPU times by the stand alone B&B and the RA1-B&B method.

6. CONCLUDING REMARKS

This paper discussed some new developments in two closely related areas. The first contribution is the development of a randomized search heuristic, termed as RA1 (for *Randomized Algorithm 1*). This heuristic takes as input positive and negative examples (i.e., binary vectors of size n) and infers a Boolean function which satisfies the requirements of the input examples. Unlike previous algorithms which were of exponential time complexity, the RA1 heuristic is of polynomial time complexity. However, it does not return small Boolean functions (in terms of the number of CNF clauses) as other more time demanding approaches (e.g., the ones in [1,3,4]).

However, computational results seem to indicate that the RA1 heuristic returns comparably small numbers of logical clauses when it is compared with the other exponential time approaches. Moreover, as it was shown in Figure 8 and supported by the computational results, the RA1 heuristic can be effectively combined with other methods (such as the B&B method developed in [4]) and solve very large problems. In this paper, we presented test results of solving problems with more than 18,000 examples defined on 15 binary variables. The previous best results were reported by [4] and involved problems with up to 1,000 examples defined on 32 atoms.

The second contribution of this paper was the development of the RA2 heuristic (for *Randomized Algorithm 2*). This algorithm can process examples in which some of the values may be missing. That is, now besides the ordinary positive and negative examples, some examples may be unclassifiable. This is the first time in the literature to deal with this kind of data. As it was anticipated, the inclusion of the unclassifiable data can significantly assist the search process. The above algorithms were tested on some large sets and on the well known breast cancer database which was originally compiled in the University of Wisconsin.

The problem of extracting a small set of logical rules via a logical analysis approach from classes of mutually exclusive observations is gradually gaining a wide interest in the computer science/operations reserach communities. This can be partly attributed to the failure of many neural network applications to gain the understanding and confidence by the end user (who usually does not have a computer/mathematical background). It should always be kept in mind, however, that the proposed methods of logical should be used in deterministic environments. Clearly, more research in this highly potential area is required.

REFERENCES

1. A.P. Kamath, N. Karmarker, K.G. Ramakrishnan and M.G.C. Resende, A continuous approach to inductive inference, *Math. Programming* **57** (2), 215–238 (1992).
2. A.P. Kamath, N. Karmarker, K.G. Ramakrishnan and M.G.C. Resende, An interior point approach to Boolean vector function synthesis, In *Proceedings of the 36th MSCAS*, 1–5, (1994).
3. E. Triantaphyllou, A.L. Soyster and S.R.T. Kumara, Generating logical expressions from positive and negative examples via a branch-and bound approach, *Computers and Operations Research* **21** (2), 185–197 (1994).
4. E. Triantaphyllou, Inference of a minimum size Boolean function from examples by using a new efficient branch-and- bound approach, *Journal of Global Optimization* **5**, 69–94 (1994).
5. E. Boros, Dualization of aligned Boolean functions, RUTCOR Reserach Report RRR, Rutgers University, pp. 9–94 (1994).
6. E. Boros, V. Gurvich, P.L. Hammer, T. Ibaraki and A. Kogan, Structural analysis and decomposition of partially defined Boolean functions, RUTCOR Research Report RRR, Rutgers University, 13–94 (1994).
7. W.H. Wolberg and O.L. Mangasarian, Multisurface method of pattern separation for medical diagnosis applied to breast cytology, In *Proceedings of the National Academy of Sciences of the USA* **87** (23), 9193–9196 (1990).
8. O.L. Mangasarian, R. Setiono and W.H. Wolberg, Pattern recognition via linear programming: Theory and application to medical diagnosis, In *Large Scale Numerical Optimization*, pp. 22–31 (1990), Philadelphia, PA; In *SIAM Proceedings of the Workshop on Large-Scale Numerical Optimization*, October 19–20, 1989, Cornell University, Ithaca, NY.
9. O.L. Mangasarian, Mathematical programming in neural networks, *ORSA Journal on Computing* **5** (4), 349–360 (1993).
10. J.W. Shavlik, Combining symbolic and neural learning, *Machine Learning* **14**, 321–331 (1994).
11. L.M. Fu, Knowledge-based connectionism for revising domain theories, *IEEE Transactions on Systems, Man and Cybernetics* **23** (1), 173–182 (1993).
12. S.A. Goldman and R.H. Sloan, The power of self directed learning, *Machine Learning* **14**, 271–294 (1994).
13. D. Cohn, L. Atlas and R. Ladner, Improving generalizing with active learning, *Machine Learning* **15**, 201–221 (1994).
14. K. Hattori and Y. Torri, Effective algorithms for the nearest neighbor method in the clustering problem, *Pattern Recognization* **26** (5), 741–746 (1993).
15. T. Kurita, An efficient agglomerative clustering algorithm using a heap, *Pattern Recognization* **24** (3), 205–209 (1991).

16. B. Kamgar-Parsi and L.N. Kanal, An improved branch and bound algorithm for computing k-nearest neighbors, *Pattern Recognition Letters* **3**, 7–12 (1985).
17. S.I. Gallant, Connectionist expert systems, *Communications of the ACM* **31**, 152–169 (1988).
18. L.M. Fu, Rule learning by searching on adapted nets, In Proceeedings of *The Ninth National Conference on Artificial Intelligence,* Anaheim, CA: AAAI Press, pp. 590–595, (1991).
19. G.G. Towell and J.W. Shavlik, Extracting refined rules from knowledge-based neural networks, *Machine Learning* **13**, 71–101 (1993).
20. J. Peysakh, *A Fast Algorithm to Convert Boolean Expressions into CNF,* IBM Computer Science RC 12913 (#57971), Watson, NY, (1987).
21. E. Triantaphyllou and A.L. Soyster, A relationship between CNF and DNF systems derivable from examples, *ORSA Journal on Computing* **7** (3), 283–285 (1995).
22. A.V. Aho, J.E. Hopcraft and J.D. Ullman, *The Design and Analysis of Computer Algorithms,* Addison-Wesley, Reading, MA, (1974).
23. T.A. Feo and M.G.C. Resende, Greedy randomized adaptive search procedures, *Journal of Global Optimization* **6**, 109–133 (1995).
24. R. Motwani and P. Raghavan, *Randomized Algorithms,* Cambridge University Press, New York, (1995).
25. N.H. Bshouty, S.A. Goldman, T.R. Hancock and S. Matar, Asking questions to minimize errors, In *Proceedings of the Sixth Annual ACM Conference on Computational Learning Theory,* pp. 41–50, (1993).
26. E. Triantaphyllou and A.L. Soyster, An approach to guided learning of Boolean functions, *Math. Comput. Modelling* **23** (3), 69–86 (1995).
27. P.M. Murphy and D.W. Aha, UCI repository of machine learning databases, *Machine Readable Data Repository,* University of California, Dept. of Information and Computer Science, Irvine, CA (1994).
28. M.G.C. Resende and T.A. Feo, A GRASP for MAX-SAT, TIMS/ORSA National Meeting in Boston, MA, USA (1994).