Published in:

Computers and Operations Research, Vol. 21, No. 2, pp. 185-179, 1994.

GENERATING LOGICAL EXPRESSIONS FROM POSITIVE AND NEGATIVE EXAMPLES VIA A BRANCH-AND-BOUND APPROACH

Evangelos Triantaphyllou[1], Allen L. Soyster[2], and Soundar R.T. Kumara[3]

1: Assistant Professor. Dept. of Industrial and Manufacturing Systems Engineering, Louisiana State University, 3134C CEBA Building, Baton Rouge, LA 70803-6409, U.S.A., E-mail: IETRIAN@LSUVM.SNCC.LSU.EDU
Web: http://www.imse.lsu.edu/vangelis

2: Professor and Head. Dept. of Industrial and Management Systems Engineering, Penn State University, 207 Hammond Building, Univ. Park, PA 16802, U.S.A. (Since 1998 Dr. Soyster is Dean of Engineering at Northeastern University in Boston, MA)

3: Associate Professor. Dept. of Industrial and Management Systems Engineering, Penn State University, 207 Hammond Building, Univ. Park, PA 16802, U.S.A.

ABSTRACT: Consider a logical system with N entities which assume binary values of either TRUE (1) or FALSE (0). There are 2^N vectors, each with N components, of this type. Even with a modest value of N, e.g. $N = 50$, the number of such vectors exceeds one quadrillion. We assume that an "expert" exists which can ascertain whether a particular vector (observation) such as (1,1,0,0,1,0,…,1) is allowable or not. This "expert" can be a human expert or an unknown system whose rules have to be inferred. Further, we assume that a sampling of m observations has resulted in M_1 instances which the "expert" has classified as allowable and $M_2 = m - M_1$ instances which are not allowable. We call these instances positive and negative examples, respectively. The objective of this research is to infer a set of logical rules for the entire system based upon the m, and possibly, additional sample observations.

The proposed algorithm in this paper is based on an highly efficient branch and bound formulation. This algorithm configures a sequence of logical clauses in conjunctive normal form (CNF), that when are taken together, accept all the positive examples and reject all the negative examples. Some computational results indicate that the proposed approach can process problems that involve hundreds of positive and negative examples in a few CPU seconds and with small memory requirements.

KEY WORDS: Learning from examples, conjunctive normal form (CNF), satisfiability problem, branch-and-bound search.

1. INTRODUCTION

This paper deals with the problem of inferring logical expressions given a set of positive (or allowable states) and a set of negative examples (or non-allowable states). This is a typical representative problem of learning from examples. Complexity issues of this type of learning have been studied by Valiant [1], [2], and Pitt and Valiant [3]. A number of algorithms which implement learning from examples were presented by Michalski [4], Dietterich and Michalski [5], Helft [6], Quinlan [7], [8], and Utgoff [9].

Recent research has been focused on the **logical inference** or **satisfiability problem**. In the logical inference problem, one is given a set of logical clauses and the problem is to determine whether certain assertions are true or false. This problem has been examined from an integer programming point of view with considerable success (see, for example, [10], [11], [12], [13], [14], and [15]). The conjunctive normal form (CNF) is used in all these treatments. In the logical inference problem the CNF clauses (rules) are given and a set of assertions are examined to see whether or not they are implied by these clauses. In contrast, in this paper we are given a set of positive examples and a set of negative examples and the main issue is to infer an appropriate set of clauses (rules). By an appropriate set of clauses we mean a set of clauses that can infer all the positive examples while they do not infer any of the negative examples.

The motivation for this research is best illustrated via an example. Consider a Rule-Based System (RBS) that involves the following 4 atoms (or premises): A_1, A_2, A_3, A_4. In any situation each atom can either be TRUE (designated by 1) or FALSE (designated by 0). For instance, in state (0,1,1,0) atoms A_2, A_3, $\bar{A}_1$, $\bar{A}_4$, are true or, equivalently, A_1, A_4, $\bar{A}_2$, $\bar{A}_3$, are false. There are $2^4 = 16$ possible states in the system. If an RBS is specified, then each of these 16 states could be categorized as either **allowable** or **non-allowable**. A state is allowable if and only if it is satisfied by each clause in the system. For example, the state (0,1,0,1) satisfies the clause: $(A_1 \text{ V } A_2 \text{ V } \bar{A}_3)$ (where V stands for the logical OR and Λ stands for the logical AND). Similarly, a state is non-allowable if it violates at least one of the clauses in the RBS. Consider the following RBS:

$(\bar{A}_1 \text{ V } \bar{A}_2 \text{ V } A_3 \text{ V } A_4)$

$(A_1 \text{ V } A_2)$

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

Then, the 16 states are characterized as (1,1,1,1) allowable, (1,0,0,0) non-allowable, (1,1,0,0) non-allowable, etc.

In this paper it is assumed that any Boolean expression (and consequently any rule) is expressed in the CNF form. An example of a statement in CNF is:

$$(A_1 \vee A_3 \vee A_4) \wedge (A_2 \vee \bar{A}_7) \wedge (A_1 \vee \bar{A}_6),$$

which is simply a conjunction of disjunctions. The above statement is true if and only if the three disjunctions are true.

More formally, a Boolean expression is in CNF if it is in the form (where a_j is either A_j or $\bar{A}_j$):

$$\bigwedge_{i=1}^{N} \left(\bigvee_{j=1}^{M_i} a_j \right)$$

Similarly, a Boolean expression is in disjunctive normal form (DNF) if it is in the form:

$$\bigvee_{i=1}^{N} \left(\bigwedge_{j=1}^{M_i} a_j \right)$$

In other words, a CNF expression is a conjunction of disjunctions, while a DNF expression is a disjunction of conjunctions.

Any expression in CNF form is true if and only if all the associated disjunctions are true. Since the rule base of any expert system contains a finite selection of rules and any rule can be reduced to clausal form [16], it follows that any rule base can be expressed as a finite set of disjunctions (clauses in CNF form). Therefore, we can assume that any RBS is in CNF form.

The general problem we analyze in this paper is the construction of a set of Boolean expressions (clauses in CNF form) which correctly classify a set of **sampled** states. We assume that each of these states can be correctly classified (by an oracle or "expert") as either allowable (also called a **positive example**) or non-allowable (also called a **negative example**). The "expert" somehow knows the correct identification of any state. However,

the underlying RBS is not explicitly known. We believe that this is quite a common situation. The "expert" somehow can identify (probably through experience) the nature of any particular state but lacks an ability to characterize the decision-making rules. We seek methods to approximate the RBS in situations in which the nature of finite numbers of states is known.

We will consider this problem in a practical and applied context. Instead of four (4) atoms, consider the case in which we may have 50 atoms. Here the number of all the possible states is: $2^{50} = 1{,}125{,}899{,}906{,}842{,}624$. It would be impractical to generate all possible states. However, one may be able to generate and categorize 500 or 1,000 sampled states. From this **partial** set of states, we determine a set of CNF clauses which correctly classify **all** the sampled states and, hopefully, a large proportion of the remaining states.

2. GENERATING CLAUSES FROM NEGATIVE EXAMPLES

Consider any state (example) α defined on N atoms. For instance, if N = 5, then consider a state such as (1,0,1,1,0). Observe that the CNF clause

$$(\bar{A}_1 \text{ V } A_2 \text{ V } \bar{A}_3 \text{ V } \bar{A}_4 \text{ V } A_5),$$

is satisfied by all states $(d_1, d_2, d_3, d_4, d_5)$, where $d_i \in \{0, 1\}$, except (1,0,1,1,0). A clause C_α can always be constructed which rejects any single state α while it accepts all other possible states. To formalize this let ATOMS(α) be the set of indices of the atoms which are true in the state α. For instance, ATOMS((1,0,1,1,0)) = $\{ 1, \overline{2}, 3, 4, \overline{5} \}$. If the clause C_α is defined as:

$$C_\alpha = \beta_1 \; v \; \beta_2 \; v \; \beta_3 \; v \ldots v \; \beta_N,$$

$$\text{where: } \beta_i = \left\{ \begin{array}{l} \bar{A}_i \quad iff \quad i \in ATOMS \quad (\alpha) \\ \\ A_i, \quad otherwise \end{array} \right\} \quad for\ each\ i = 1,2,3,\ldots,N,$$

then the clause C_α will reject only the state (example) α and accept any other state.

Suppose that m states are somehow generated. Define E^+ as the set of M_1 states which are classified as allowable (positive examples) and E^- as the set of states which are not allowable (negative examples). For each of the $M_2 = m - M_1$ states in E^-, generate the unique clause as defined above. Each of these clauses rejects

one and only one state and, hence, accepts all the states in E^+. This set of M_2 clauses precisely satisfies the objective of this paper. However, this approach is **impractical** for large selections of negative examples, since it results in large numbers of clauses. Therefore, it is important to have an approach that constructs a rather small (relative to the number M_2) number of clauses. The method described in the following section is such an approach.

3. THE ONE CLAUSE AT A TIME APPROACH

The proposed **One Clause At a Time (or OCAT)** approach, which is a **greedy** one, uses as input data the two collections of positive and negative examples. It determines a set of CNF clauses that, when taken together, reject all the negative examples and each of them accepts all the positive examples. The OCAT approach is sequential. In the first iteration it determines a clause in CNF form that accepts all the positive examples in the E^+ set while it rejects as many negative examples in the current E^- set as possible. In the second iteration it performs the same task using the original E^+ set but the current E^- set has only those negative examples (non-allowable states) that have not been rejected by any clause so far. The iterations continue until a set of clauses is constructed which reject all the negative examples. Figure 1 summarizes the iterative nature of OCAT.

$i = 0$; $C = \phi$

DO WHILE ($E^- \neq \phi$)

Step 1: $i \leftarrow i + 1$

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^- that 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 OCAT Approach

The core of the OCAT approach is step 2 in figure 1. In section 5 a branch and bound based algorithm is presented that solves the problem posed in step 2. The OCAT approach returns the set of desired clauses as the

set C. The following theorem states a critical property of the OCAT approach.

THEOREM 1:

The OCAT approach terminates within M_2 iterations.

Proof:

From Section 2 it is always possible to construct a clause C_α that rejects only one negative example while it accepts any other possible example. At worse, in step 2 OCAT could propose a clause that rejects only one negative example. Therefore, it follows that the maximum number of iterations of the OCAT approach is M_2.

EOP (End Of Proof)

4. CLAUSE INFERENCE AS A SATISFIABILITY PROBLEM

In [17] it is shown that given two collections of positive and negative examples then, a DNF system can be inferred to satisfy the requirements of these examples. This is achieved by formulating a satisfiability (SAT) problem and then using the interior point method of Karmakar [18] as a solution strategy. This approach requires the specification of the number of conjunctions in the DNF system. The SAT problem uses the following Boolean variables [17]:

$$s_{ji} = \begin{cases} 0 & \text{if } A_i \text{ is in the j-th conjunction} \\ 1 & \text{if } A_i \text{ is not in the j-th conjunction} \end{cases}$$

$$s'_{ji} = \begin{cases} 0 & \text{if } \bar{A}_i \text{ is in the j-th conjunction} \\ 1 & \text{if } \bar{A}_i \text{ is not in the j-th conjunction} \end{cases}$$

$$\sigma^{\alpha}_{ji} = \begin{cases} s'_{ji} & \text{if } A_i = 1 \text{ in the positive example } \alpha \in E^+ \\ s_{ji} & \text{if } A_i = 0 \text{ in the positive example } \alpha \in E^+ \end{cases}$$

$$z^{\alpha}_{j} = \begin{cases} 1 & \text{if the positive example } \alpha \text{ is accepted by the j-th conjunction} \\ 0, & \text{otherwise} \end{cases}$$

Then, the clauses of this SAT problem are as follows:

$$s_{ji} \vee s'_{ji},\ i = 1,\ldots,n,\ j = 1,\ldots,K \quad (1)$$

$$\left(\bigvee_{i\in P_r} \bar{s}'_{ji}\right) \vee \left(\bigvee_{i\in \bar{P}_r} \bar{s}_{ji}\right),\ j = 1,\ldots,K,\ r = 1,\ldots,M_2 \quad (2)$$

$$\bigvee_{j=1}^{K} z_j^{\alpha},\ \alpha = 1,\ldots,M_1 \quad (3)$$

$$\sigma_{ji}^{\alpha} \vee \bar{z}_j^{\alpha},\ i = 1,\ldots,n,\ j = 1,\ldots,K,\ \alpha = 1,\ldots,M_1 \quad (4)$$

where P_r is the set of indices of A for which $A_i = 1$ in the negative example $r \in E^-$. Similarly, $\bar{P}_r$ is the set of indices of A for which $A_i = 0$ in the negative example $r \in E^-$.

Clauses of type (1) ensure that never both A_i and $\bar{A}_i$ will appear in any conjunction. Clauses of type (2) ensure that each negative example is rejected by all conjunctions. Clauses of type (3) ensure that each positive example is accepted by at **least one** conjunction. Finally, clauses of type (4) ensure that $z_j^{\alpha} = 1$ if and only if the positive example α is accepted by the j-th disjunction. In general, this SAT problem has $K(n(M_1 + 1) + M_2) + M_1$ clauses, and $K(2n(1 + M_1) + n M_2 + M_1)$ Boolean variables. A detailed example on this formulation can be found in [17].

Besides the fact that OCAT infers CNF systems, while the SAT approach infers DNF systems, the two approaches have another major difference. The OCAT approach attempts to minimize the number of disjunctions in the proposed CNF system. However, the SAT approach **pre-assumes** a given number, say k, of conjunctions in the DNF system to be inferred and solves a SAT problem. If this SAT problem is infeasible, then the conclusion is that there is no DNF system which has k or less conjunctions and satisfies the requirements imposed by the examples. It should be emphasized here that it is not very critical whether an inference algorithm determines a CNF or DNF system (i.e., CNF or DNF Boolean function). As shown in [19], either a CNF or DNF system can be derived by using **either algorithm**.

5. THE BRANCH-AND-BOUND APPROACH OF THE OCAT PROBLEM

The branch-and-bound (B&B) approach is best described with an illustrative example. Suppose that the following are the two sets (it is assumed that N = 4, i.e., the system involves 4 atoms) E^+ and E^- with the positive

and negative examples of cardinality M_1 and M_2, respectively.

$$E^+ = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix} \qquad E^- = \begin{vmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{vmatrix}$$

We number the positive examples as (1,2,3,4) and the negative examples as (1,2,3,4,5,6). For instance, the set of the negative examples {1,3} means the set of the first and the third negative examples. The B&B approach will determine a single clause that accepts all the positive examples in the E^+ set, while rejecting as many negative examples from the current E^- set as possible. Before proceeding with a description of the B&B approach, it is instructive to compare it with a complete enumeration methodology.

Consider the first positive example (0,1,0,0). Observe that in order to accept this positive example at least one of the four atoms A_1, A_2, A_3, A_4 must be specified as follows: (A_1 = FALSE, i.e., $\bar{A}_1$ = TRUE), (A_2 = TRUE), (A_3 = FALSE , i.e., $\bar{A}_3$ = TRUE), and (A_4 = FALSE, i.e., $\bar{A}_4$ = TRUE). Hence, any valid CNF clause must include $\bar{A}_1$, or A_2, or $\bar{A}_3$, or, $\bar{A}_4$. Similarly, the second positive example (1,1,0,0) means that any valid CNF clause must include A_1, or A_2, or $\bar{A}_3$ or $\bar{A}_4$. In this manner, all valid CNF clauses must include at least one atom as specified from each of the following sets: $\{\bar{A}_1, A_2, \bar{A}_3, \bar{A}_4\}$, $\{A_1, A_2, \bar{A}_3, \bar{A}_4\}$, $\{\bar{A}_1, \bar{A}_2, A_3, A_4\}$, and $\{A_1, \bar{A}_2, \bar{A}_3, A_4\}$.

This is a special case of the set covering problem which we denote as the **minimum cardinality problem** (or MCP). Let $|s|$ denote the cardinality of a set s. For the clause inference problem, the corresponding MCP problem takes the following form:

$$\textit{MINIMIZE} \quad \Big|\bigcup_{i=1}^{M_1} \beta_i\Big|$$

$$\textit{subject to:}\ \ \beta_i \in B_i \quad \textit{for } i=1,2,3,\ldots,M_1,$$

$$\textit{where the } B_i\ (i=1,2,3,\ldots,M_1) \textit{ are finite sets, each of which contains exactly } N \textit{ elements, and each element is a subset of } \{1,2,3,\ldots,M_2\}.$$

The MCP formulation for the current clause inference problem is developed as follows. Define as $NEG(A_k)$ the set of the negative examples which are accepted by a clause when the atom A_k is included in that clause. For the example in this section the $NEG(A_k)$ sets are presented in Table 1. In the light of the definition of the sets $NEG(A_k)$ and the sets $ATOMS(\alpha)$ (as defined in Section 2), the sets B_i in problem MCP are defined as follows:

$$B_i = \{NEG(A_k),\ \textit{for each } k \in ATOMS(\alpha_i)\},$$

$$\textit{where } \alpha_i \textit{ is the i-th positive example in } E^+.$$

Table 1. The $NEG(A_k)$ Sets for the Illustrative Example

Atom	Set of Negative Examples	Atom	Set of Negative Examples
A_1	$NEG(A_1) = \{1,3,5,6\}$	$\bar{A}_1$	$NEG(\bar{A}_1) = \{2,4\}$
A_2	$NEG(A_2) = \{3,6\}$	$\bar{A}_2$	$NEG(\bar{A}_2) = \{1,2,4,5\}$
A_3	$NEG(A_3) = \{1,3,6\}$	$\bar{A}_3$	$NEG(\bar{A}_3) = \{2,4,5\}$
A_4	$NEG(A_4) = \{2,3\}$	$\bar{A}_4$	$NEG(\bar{A}_4) = \{1,4,5,6\}$

Therefore, the previous minimization problem takes the form:

$$\textit{MINIMIZE} \quad \Big|\bigcup_{i=1}^{M_1} \beta_i\Big| \qquad \textbf{(I)}$$

$$\textit{subject to:}\ \ \beta_i \in B_i \quad \textit{for } i = 1,2,3,\ldots,M_1,$$

$$\textit{where:}\quad B_i = \{NEG(A_k),\ \textit{for each } k \in ATOMS\ (\alpha_i)\}$$

$$\textit{and } \alpha_i \textit{ is the i-th positive example in } E^+.$$

Using the data presented in Table 1, formulation (I), takes the following form for the case of the illustrative example of this section:

$$\textit{MINIMIZE} \quad \left| \bigcup_{i=1}^{4} \beta_i \right|$$

subject to:

$$\beta_1 \in B_1 = \{ \{2,4\},\{3,6\},\{2,4,5\},\{1,4,5,6\} \}$$

$$\beta_2 \in B_2 = \{ \{1,3,5,6\},\{3,6\},\{2,4,5\},\{1,4,5,6\} \}$$

$$\beta_3 \in B_3 = \{ \{2,4\},\{1,2,4,5\},\{1,3,6\},\{2,3\} \}$$

$$\beta_4 \in B_4 = \{ \{1,3,5,6\},\{1,2,4,5\},\{2,4,5\},\{2,3\} \}$$

An **exhaustive enumeration** approach to solve this MCP problem is to construct a tree that has nodes arranged in 4 (= M_1) levels. In the description of the search that follows, we call these levels **stages**. These levels correspond to the 4 positive examples enumerated as #1, #2, #3, and #4, in E^+. Each interior node (i.e., nodes with descendents), say at level h (where $1 \leq h < 4$), is connected to N nodes in the next higher level via N arcs. These N arcs represent the atoms that are true at the h-th positive example (i.e., the members of the set ATOMS(α_h) and α_h is the h-th positive example), as described in Section 2. The nodes (or **search states**) in this graph represent sets of negative examples, in our illustrative example these are subsets of the set {1,2,3,4,5,6}.

For instance, the state {2, 3, 5} refers to the second, third, and fifth negative examples in the set E^-. The set of negative examples that corresponds to a node (state) is the set of all the negative examples accepted by the atoms that correspond to the arcs that connect that node with the root node. That is, if one is at node (state) Y_k and one follows the arc that corresponds to the atom A_i, then the resulting state, say Y_L, is:

$$Y_L = Y_K \cup \text{NEG}(A_i).$$

If the above strategy is followed, then the current illustrative example would create 4x4x4x4 = 256 terminal nodes and, in the general case, N^{M_1} terminal nodes (where: M_1 is $|E^+|$). Then, the clause which accepts all the positive examples and rejects as many negative examples as possible is found by simply selecting

the terminal node that corresponds to a state with the **minimum cardinality**. This is true because that state accepts the **minimum** number (or equivalently, rejects the maximum number) of negative examples.

Apparently, an exhaustive enumeration strategy is impractical. This is true because an exhaustive enumeration will require one to construct a search tree with different terminal nodes (final states). However, our proposed B&B approach which is based on the previous tree, is very fast because it is capable of **pruning** this tree very efficiently. Each node of the tree is examined in terms of two tests. If any of these two tests succeeds, then that node is **fathomed** and it is not expanded further.

Consider the two nodes which correspond to the two states {2, 4, 5} and {1, 2, 4, 5, 6} in the second stage of the search tree (see also figure 2). Clearly, the states that correspond to the leaves (terminal nodes) that have the state {1, 2, 4, 5, 6} as an ancestor are going to have **at least as** many members (i.e., negative examples) as the states of the leaves (terminal nodes) that have as ancestor the state {2, 4, 5}. This is true because subsequent states are derived by performing union operations on these two states with the same sets. Therefore, if at any stage of building the search tree there is a state that has another state (in the current stage) as a subset then, that state (node) can be fathomed without eliminating any optimal solutions. This characterization of the states is formalized by the following definition, which is derived from the above discussion.

DEFINITION 1:

*A state S_k is a **dominated state** if there is another state S_j in the same stage which is a **proper** subset of S_k, i.e., $S_i \subset S_k$. Otherwise, the state S_k is an **undominated** state.*

The notion of dominated states leads to an important simplification of the (MCP) problem. Define (MCP') as the problem derived from (MCP) when all dominated states are eliminated. Then, the previous definition and discussion about dominated and undominated states are summarized in the following theorem:

THEOREM 2:

An optimal solution to (MCP') is also optimal to (MCP).

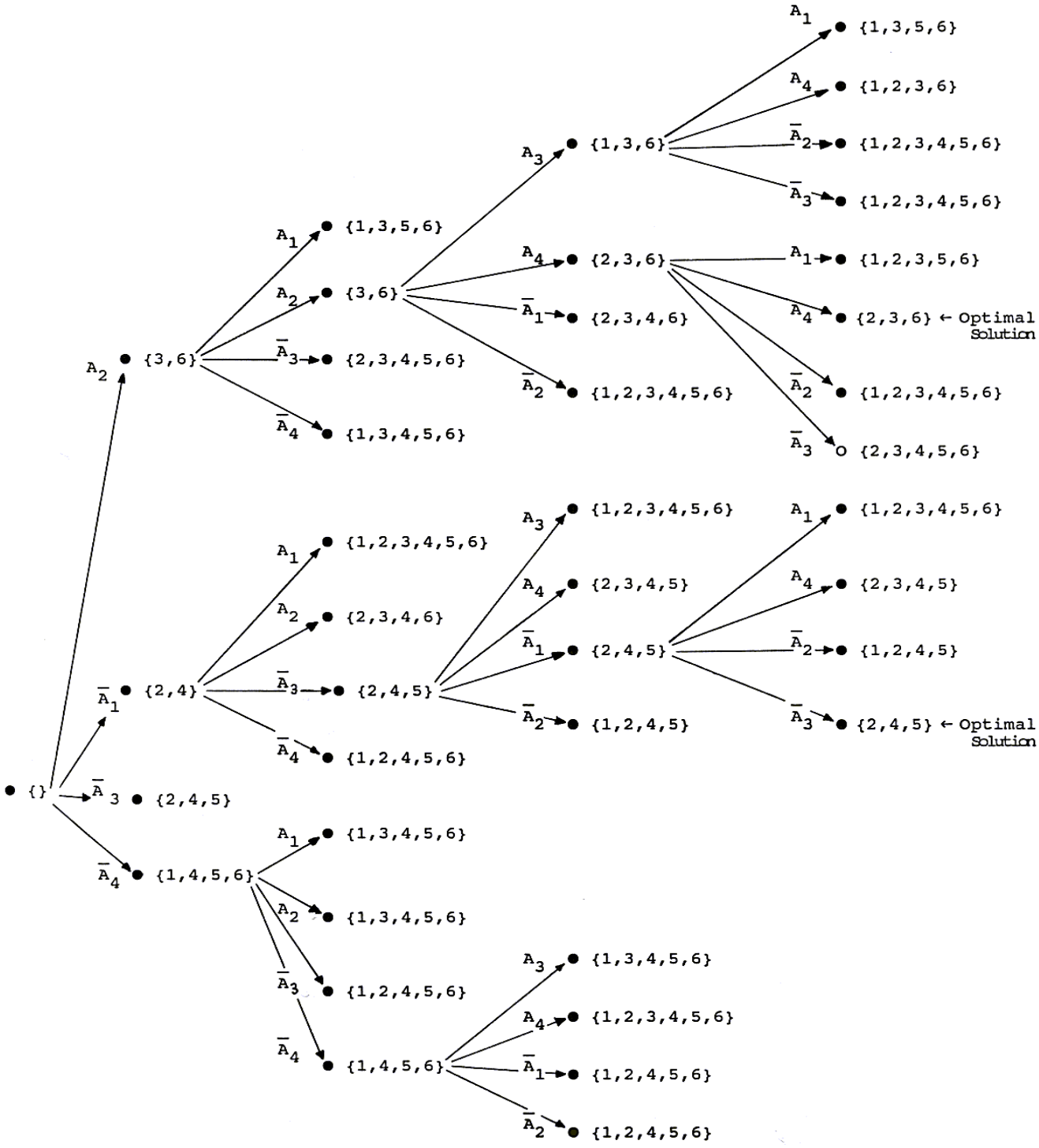

Figure 2. The Branch-And-Bound Search

Corollary 1:

The optimal solution of the original MCP problem, given as (I), and the following MCP problem are identical.

$$MINIMIZE\ |f| = \left|\bigcup_{i=1}^{i=M_1} \beta_i\right|$$

$$subject\ to:\quad \beta_i \in \hat{B}_i \quad for\ i = 1,2,3,\ldots,M_1,$$

where $\hat{B}_i$ $(i=1,2,3,\ldots,M_1)$ *is the set that has as members only the undominated members of the set* B_i.

The previous corollary can be used for **problem pre-processing**. That is, when an MCP problem formulated as (I) is given, it is beneficial to first transform it to the problem MCP' (as described above). In this way, the number of options (arcs in the B&B search graph) available at each node (state) of the search graph will be the same or **smaller** than in the original MCP problem. Clearly, this means that the search can be done faster than in the original MCP problem.

In figure 2 the B&B tree for the current illustrative example is presented. The states in the last stage (i.e., the leaves of the tree) with the minimum number of negative examples indicate an optimal solution. In this example there are two such minimum size states. They are: {2, 3, 6} and {2, 4, 5}. The first optimal state (i.e., {2, 3, 6}) is derived from the clause $(A_2\ V\ A_4)$. The atoms A_2 and A_4 are the only atoms (as indicated by the B&B tree) which are involved in the decisions that generate the state {2, 3, 6}. Similarly, the second optimal state (i.e., {2, 4, 5}) is derived from the clause $(\bar{A}_1\ V\ \bar{A}_3)$.

An alternative and more efficient implementation of this B&B formulation is to keep in memory only the nodes (states) of the current level (stage). Then, when an optimal state S is determined at the last stage, the optimal clause can be found by simply activating (i.e., setting as true in the current clause) all the atoms which make only the negative examples included in this state S to be accepted.

Note that the optimal solution $(A_2\ V\ A_4)$ does not rejects the second, third, and sixth current non-allowable states (negative examples in E^-). Hence, the remaining negative examples are:

$$E^- = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

Similarly, the second OCAT iteration, when applied to the E^+ set and the new E^- set, yields the clause $(\bar{A}_2 \vee \bar{A}_3)$. The remaining negative examples are:

$$E^- = [0\ 0\ 0\ 1]$$

Iterating further, the third OCAT iteration yields the clause $(A_1 \vee A_3 \vee \bar{A}_4)$. That is, the system of the CNF clauses which are generated from the original examples E^+ and E^- are:

Clause 1: $(A_2 \vee A_4)$

Clause 2: $(\bar{A}_2 \vee \bar{A}_3)$

Clause 3: $(A_1 \vee A_3 \vee \bar{A}_4)$.

It can be easily verified that the previous three clauses, when taken together, reject all the negative examples in E^-. Moreover, each of the three clauses accepts all the positive examples in E^+.

There is another observation that allows a further reduction in the number of states in the B&B search. Suppose that it is known (possibly via a heuristic) that one of the terminal states in the B&B search (not necessarily an optimal one) has k elements. Then, at any stage of the B&B approach, all states which have more than k elements can be deleted from further consideration. This is true because any descendent of a state may only get larger at subsequent stages. This observation is summarized in the following theorem:

THEOREM 3:

Suppose some feasible solution to MCP has cardinality k. Then, an optimal solution to a modified B&B problem in which all states that have more than k members are deleted, is also optimal for MCP. Furthermore, if this B&B problem is infeasible, then an optimal solution to MCP has cardinality k.

Corollary 2:

The optimal solution of the original MCP problem, given as (I), and the following problem are identical.

$$MINIMIZE\ |f| = \left|\bigcup_{i=1}^{i=M_1} \beta_i\right|$$

$$subject\ to:\quad \beta_i \in \hat{B}_i \quad for\ i=1,2,3,\ldots,M_1,$$

where $\hat{B}_i$ $(i=1,2,3,\ldots,M_1)$ *is the set which has as members only the members of the set* B_i *which have less than or equal to k members.*

6. A HEURISTIC FOR PROBLEM PRE-PROCESSING

The previous corollary can be used for **problem pre-processing**. That is, when an MCP problem is formulated as (I) then, it is a good idea first to run a heuristic (as it will be described next) that very quickly yields a good feasible solution of size k (i.e., k is small) to the original MCP problem. When a value for k is available, the B&B search does not need to expand nodes in the search graph that have cardinality greater than k. This is true even for nodes that correspond to undominated states. In this way, the number of nodes to be expanded in the B&B search tree will, in general, be **smaller** than in the original MCP problem. This step has the potential to expedite the B&B search.

Theorem 3 can further improve the performance of the proposed B&B search. When the B&B search is performed, the number of states at each stage (i.e., level of the search tree) may increase dramatically. Therefore, the time and memory requirements of the search may increase dramatically. An efficient way to overcome this complication is to run the B&B search in two phases. In the first phase the B&B search is applied by allowing up to a small number, say 5, of states (i.e., nodes in the search tree) to be considered at any stage (i.e., level of the search tree). These 5 states are the ones with minimal cardinality. That is, if more than 5 states (nodes) are formed at any stage, then only the 5 states with the smallest cardinality will be considered for the next stage. This type of search is used in the AI literature often and is called **beam search** (see, for instance, [22]).

Since up to 5 states are allowed to be considered at any stage of the B&B search and the number of stages is equal to the number of positive examples, it follows that the first phase will terminate quickly. Furthermore, the terminal nodes (final states) of the search tree will tend to represent states which have a tendency to have small

cardinalities. This is expected to be the case because at each stage only the 5 states with the minimal cardinality are considered.

Suppose that in the first phase of the B&B process more than 5 states were generated at some stage. Let k be the cardinality of the smallest state that is dropped from further consideration due to the upper limit of 5 states per stage. Then, if one of the terminal nodes has cardinality **less than k**, then one can conclude that this node (state) represents an optimal solution. This is true because in this case none of the deleted states could lead to a terminal state with cardinality less than k. If there is no terminal state with cardinality less than k, then a terminal node (search state) with the minimal cardinality represents a good feasible solution which may or may not be optimal. It should be emphasized here that by an optimal solution we mean the one that represents a clause which accepts all the positive examples in E^+ while it rejects as many negative examples in the current E^-set as possible.

If after the first phase optimality is not provable, then the second phase is initiated. In the second phase, the B&B process is repeated with an upper limit, say 20, states per stage. As in the first phase, these 20 states are the states with the 20 smallest cardinalities. Suppose that L is the cardinality of the solution obtained in the first phase. Then in the second phase, Theorem 3 is applied by eliminating any state that has cardinality greater than L. However, memory limitations may prohibit this B&B search from reaching an optimal solution. It should be stated here that if a too large number of states is allowed to be considered at any stage, then the B&B approach will take excessive time in ranking these states. The previous limit of 20 states was empirically found to be a reasonable choice.

As it was done in the first phase, if more than 20 states are generated at any stage, then only 20 states are allowed at each stage. Similarly to the first phase, let k be the cardinality of the smallest state that was dropped from further consideration due to the upper limit of 20 states per stage. Then, if one of the terminal nodes has cardinality **less than k** one can conclude that this node (state) represents an optimal solution. Otherwise optimality is not provable. In this case one may want to proceed with a third phase, or fourth phase until optimality is reached.

Table 2.
Some Computational Results When N=30 and the OCAT Approach is Used

$\|E^o\|$	$\|E^+\|$	$\|E^-\|$	S	time	$\|E^o\|$	$\|E^+\|$	$\|E^-\|$	S	time
100	9	91	4	2	400	10	390	6	10
100	5	95	4	2	400	7	393	6	10
100	15	85	4	7	400	36	364	13	282
100	7	93	4	6	400	47	353	6	97
100	3	97	4	1	400	49	351	12	400
100	8	92	4	2	400	15	385	5	7
100	7	93	4	2	400	5	395	5	3
100	1	99	4	1	400	17	383	6	23
100	7	93	4	3	400	16	384	6	8
100	5	95	4	3	500	35	465	12	194
200	5	195	4	2	500	16	484	5	38
200	2	198	5	1	500	7	493	6	15
200	18	182	5	18	500	34	466	7	73
200	6	194	5	2	500	13	487	6	8
200	1	199	4	1	500	20	480	5	19
200	11	189	7	38	500	6	494	5	13
200	19	181	4	4	600	83	517	6	300
200	51	149	12	212	600	49	551	15	315
200	10	190	4	6	600	44	556	5	41
200	4	196	5	2	600	8	592	6	16
300	22	278	8	70	600	23	577	12	184
300	14	286	6	25	600	11	589	6	15
300	14	286	7	29	700	56	644	16	467
300	2	298	5	1	700	18	682	6	30
300	22	278	10	102	700	19	681	6	15
300	36	264	11	243	700	19	681	9	60
300	24	276	4	12	700	13	687	6	26
300	71	229	14	524	800	64	736	18	739
300	3	297	5	2	900	72	828	17	836
300	17	283	11	107	1000	47	953	14	80

NOTE: Time is in seconds.

Some computational experiments indicate that Theorems 2 and 3 provide a very efficient way for keeping the states at each stage in a manageable number and the resulting CPU requirements are dramatically reduced. For instance, a case with N equal to 10, 50 positive examples, and with 170 negative examples required more than 1,100 CPU seconds on an IBM ES/3090-600S machine running an integer programming implementation of the OCAT approach by using MPSX. However, the same problem took less than 30 CPU seconds with the proposed B&B formulation. Other similar comparisons also demonstrated significant improvement in time performance for the B&B approach.

7. SOME COMPUTATIONAL RESULTS

In order to gain some computational experience with OCAT and the B&B formulation, some random problems were generated and tested. The derived computational results are depicted in table 2. For these problems N, the number of atoms, was set equal to 30. First a set of 40 random clauses (disjunctions) was generated (the number 40 is arbitrary). Each such clause included, on the average, 5 atoms (as was the case with the experiments reported in [13]). The range of the number of variables per clause was from 1 to 10. Next, a collection E^o of random examples was generated. In these experiments we generated groups of 100, 200, 300,..., 1000 random examples.

Each such random example was classified, according to the previous 40 clauses, either as a positive or as a negative example. With 40 clauses, this process resulted in more negative than positive examples. Because the stages in the B&B algorithm correspond to positive examples, problems with higher percentages of positive examples would demand more CPU time.

Next, the OCAT approach was applied on the previous positive and negative examples. The computational results are shown in table 2. In this table the number of clauses derived by OCAT is denoted as S. The CPU time of the OCAT approach was recorded as well. This simulation program was written in PL/I and run on an IBM ES/3090-600S computer.

Each entry in table 2 represents the performance of a single test problem, rounded to the nearest integer. Recall that $|S|$ indicates the **size** of a set S. The computational results in table 2 strongly suggest that the B&B approach is computationally tractable. For instance, no test problem took more than 836 CPU seconds (with an overage of 96.17 CPU seconds). As it was anticipated, the number of clauses created by OCAT increases with the number of input examples.

It is also interesting to observe the behavior of the CPU time used by OCAT under the B&B formulation.

Since the number of stages in the B&B search is equal to the number of positive examples, the CPU time increases with the size of the set of the positive examples. Furthermore, the total number of examples $|E^o|$ is critical too.

In [19] an approach is presented to derive DNF expressions by using the OCAT approach. Some ways for partitioning large scale learning problems of the type described in this paper are also discussed in [20]. In [21] an approach for **guided learning** is described. In that approach examples are not randomly generated, but instead, they are generated in a manner which attempts to derive the correct system by considering only few new examples.

In these test problems the B&B formulation was applied as follows. During the first phase up to 5 states were allowed. If after the final stage optimality was not proved, the best (i.e., the one with the smallest cardinality) solution available at this point was kept and the B&B approach was repeated by allowing up to 20 B&B states per stage (20 was an upper limit for memory considerations). These 20 states were selected as follows. If more than 20 B&B states were generated at some stage, then these states were ranked in descending order according to the number of elements (negative examples) per state and the top 20 states were selected.

In this second phase of the B&B search, the best solution found at the end of the first phase was used to reduce the state space at each stage (i.e., Theorem 3 was applied to reduce the memory requirements). The process was terminated after this second phase (in which the 20 states per stage limit was imposed) whether the current best solution could be confirmed as optimal or not. It should be mentioned here that if a higher limit of states was used, then the B&B approach takes more time because at each stage more states need to be considered. Some computational tests indicated that the previous limits (i.e., 5 and 20 states) seem to be reasonable. In 83% of the problems examined, confirmation of optimality could be made. The very low CPU times strongly indicate that the B&B approach is very efficient both in terms of CPU time and memory requirements.

Tables 3 and 4 present some computational results when the SAT approach is used. These results are the ones originally reported in [17]. The CPU times are approximated to the closest integer value (in seconds). These experiments were performed on a VAX 8700 running UNIX. The program was written in FORTRAN and C. The strategy of generating and testing the random problems is similar to the one mentioned in the OCAT case. The only difference is that now the "hidden system" is in DNF form and consists of a few conjunctions (three to four). Recall, that in the OCAT case the "hidden logic" was a system in CNF form consisted of 40 randomly generated disjunctions.

Table 3.

Some Computational Results When N = 16 and the SAT Approach is Used

$\|E^o\|$	Problem id	k	Vars	Clauses	time
100	16A1	15	1,650	19,368	2,039
100	16C1	20	1,580	16,467	758
200	16D1	10	1,230	15,901	1,547
200	16E1	15	1,245	14,766	2,156
300	16A2	6	1,602	23,281	608
300	16B1	8	1,728	24,792	78
400	16B2	4	1,076	16,121	236
400	16C2	4	924	13,803	521
400	16D2	4	836	12,461	544
400	16E2	4	532	7,825	376

NOTE: Time is in seconds.

The main point with the SAT results is that even for a small number of (positive and negative) examples the CPU times are rather high. This happens because the resulted SAT problems (as it was indicated in formulas presented in section 4) require many variables and clauses (as it is shown under the *"Vars"* and *"Clauses"* columns in tables 3 and 4). In table 4 the test problems considered 32 atoms. The CPU are smaller than the ones with 16 atoms (in table 3) because now k was allowed to take much smaller values (3 or 4). In the 16 atom case, however, k was allowed to take larger values (4 to 20).

In other words, the CPU requirements increase dramatically with the number of conjunctions assumed in the SAT formulation (denoted with k). This behavior is in direct agreement with the formulas mentioned in Section 4. However, if the original k value is **too small**, then infeasibility will be reached and the SAT problem needs to run again (with a larger k value) until a feasible solution is reached. This situation may increase the **actual** CPU requirements even more dramatically than the figures shown in tables 3 and 4.

Table 4.

Some Computational Results When N=32 and the SAT Approach is Used

$\|E^o\|$	Problem id	k	Vars	Clauses	time
50	32B1	3	228	1,374	5
50	32C1	3	225	1,280	24
50	32D1	4	332	2,703	66
50	32E1	3	222	1,186	8
100	32B2	3	261	2,558	57
100	32C2	3	249	2,182	9
100	32D2	4	404	5,153	178
100	32E2	3	267	2,746	10
150	32C3	3	279	3,272	14
200	32E3	3	330	5,680	133
250	32A1	3	459	9,212	177
250	32B3	3	348	5,734	190
300	32B4	3	381	6,918	259
300	32E4	3	387	7,106	277
400	32D3	4	824	19,478	1,227
400	32E5	3	450	9,380	390
1000	32C4	3	759	20,862	155

NOTE: Time is in seconds.

8. CONCLUDING REMARKS

In general, the expert in a domain somehow can categorize a given state as being positive or negative. However, the underlying RBS may not be explicitly known. In this paper we consider the problem of learning the structures of clauses from selections of positive and negative examples. Since the rules in any rule based system can be transformed into a set of clauses in CNF form [16] the problem of being able to infer clause structures is

very critical in RBS development. As described earlier, most often the structure of the rules in the knowledge base of an RBS is not known and has to be derived from examples.

The approach proposed in this paper helps in efficiently learning the rules (in CNF) of an RBS through sets of positive and negative examples. The OCAT approach is based on an efficient B&B algorithm. This approach attempts to derive as few logical clauses (i.e., rules) as possible. A satisfiability based approach, proposed in [17], can actually determine the minimum number of clauses, given the sets of positive and negative examples. However, the SAT approach pre-assumes an upper limit on the number of clauses and it is very CPU time consuming. Therefore, if the absolute minimum number of clauses is required, the OCAT approach maybe used first to derive an upper limit for the SAT approach. It should also be stated here that in [19] it is demonstrated that either approach can derive a CNF or DNF system of clauses. Furthermore, in [19] and [20] it is shown how to solve large scale problems of inductive inference.

A related issue is to examine whether the inferred systems are correct in a domain. In other words, what is the accuracy of these inferred systems for unseen examples in that particular domain. Although this is an important issue, it is not examined in this paper. However, this can be one of the extensions of the present work.

Although considerable work has been done on the logical inference problem, the problem of inferring clause structures from positive and negative examples has not been examined adequately in the literature. The present paper demonstrates that a branch-and-bound approach has the potential to efficiently solve large learning from examples problems.

ACKNOWLEDGEMENTS

The authors would like to thank the referees for their thoughtful comments which significantly improved the quality of this paper.

REFERENCES

1. L. G. Valiant, A Theory of the Learnable, *Comm. of ACM.*, **27** (11), 1134-1142 (1984).

2. L. G. Valiant, Learning Disjunctions of Conjunctives, *Proceedings of the 9th IJCAI.*, 560-566 (1986).

3. L. Pitt. and L.G. Valiant, Computational Limitations on Learning from Examples, *Journal of the Association for Computing Machinery*. **35** (4), 965-984 (1988).

4. R. S. Michalski, Machine Learning Research in the Artificial Intelligence Laboratory at Illinois, *Machine Learning: A Guide to Current Research*, T.M. Mitchell, J.G. Carbonell, R.S. Michalski (Eds.), 193-198, Kluwer Academic Publishers, Boston, Mass, (1986).

5. T. C. Dietterich and R.S. Michalski, A Comparative Review of Selected Methods for Learning from Examples, *Machine Learning: An Artificial Intelligence Approach*. R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (Eds.), Tioga Publishing Company, Palo Alto, CA, 41-81 (1983).

6. N. Helft, Learning Systems of First-Order Rules, *Proceedings of the Fifth International Conference on Machine Learning*, John Laird (Ed.), University of Michigan, Ann Arbor, 395-401 June 12-14, 1988.

7. J. R. Quinlan, Discovering Rules by Induction from Large Numbers of Examples: A Case Study, D. Michie (ed.). *Expert Systems in the Micro-Electronic Age*. Edinburgh University Press (1979).

8. J. R. Quinlan, Induction of Decision Trees, *Machine Learning*. **1** (1), 81-106, Kluwer Academic Publishers (1986).

9. P. E. Utgoff, ID5: An Incremental ID3, *Proceedings of the Fifth International Conference on Machine Learning*. John Laird (ed.). University of Michigan, Ann Arbor, 107-120 (1988).

10. H. P. Williams, Linear and Integer Programming Applied to Artificial Intelligence, reprint series, University of Southampton, Faculty of Mathematical Studies, 1-33 (July 1986).

11. R. G. Jeroslow, Computation-Oriented Reductions of Predicate to Propositional Logic, *Decision Support Systems*, **4**, 183-197, North-Holland (1988).

12. J. N. Hooker, Generalized Resolution and Cutting Planes, *Annals of Operations Research*, R.G. Jeroslow (Ed.), **12** (1-4) 217-239 (1988).

13. J. N. Hooker, Resolution vs. Cutting Plane Solution of Inference Problems: Some Computational Experience, *Operations Research Letters*, **7** (1), 1-7 (1988).

14. T.M. Cavalier, P.M. Pardalos, A.L. Soyster, Modeling and Integer Programming Techniques Applied to Propositional Calculus. *Computers & Operations Research*, **17** (6) 561-570 (1990).

15. A. P. Kamath, N.K. Karmakar, K.G. Ramakrishnan, and M.G.C. Resende, Computational Experience with an Interior Point Algorithm on the Satisfiability Problem, *Annals of Operations Research*, 27 (1991).

16. C. E. Blair, R.G. Jeroslow, and J.K. Lowe, Some Results and Experiments in Programming Techniques for Propositional Logic, *Computers & Operations Research*, **13** (5) 633- 645 (1986).

17. A.P. Kamath, N.K. Karmakar, K.G. Ramakrishnan, and M.G.C. Resende, A Continuous Approach to

Inductive Inference. *Mathematical Programming*, to appear (1992).

18. N. K. Karmakar, M.G.C. Resende, and K.G. Ramakrishnan, An Interior Point Algorithm to Solve Computationally Difficult Set Covering Problems. *Mathematical Programming,* to appear (1992).

19. E. Triantaphyllou and A.L. Soyster, A Relationship Between CNF and DNF Systems Derivable from Examples. *ORSA Journal on Computing,* to appear (1993).

UPDATED REFERENCE:

E. Triantaphyllou and A.L. Soyster, A Relationship Between CNF and DNF Systems Derivable from Examples. *ORSA Journal on Computing,* Vol. 7, No. 3, pp. 283-285, (1995).

20. E. Triantaphyllou and A.L. Soyster, On the Minimum Number of Logical Clauses Which Can be Inferred From Examples, *Working Paper,* Kansas State University, Dept. of Industrial Engineering (1992).

UPDATED REFERENCE:

E. Triantaphyllou and A.L. Soyster, On the Minimum Number of Logical Clauses Which Can be Inferred From Examples, *Computers and Operations Research,* Vol. 23, No. 8, pp. 783-799, (1996).

21. E. Triantaphyllou and A.L. Soyster, An Approach to Guided Learning of Boolean Functions, *Working Paper,* Kansas State University, Dept. of Industrial Engineering (1992).

UPDATED REFERENCE:

E. Triantaphyllou and A.L. Soyster, An Approach to Guided Learning of Boolean Functions, *Mathematical and Computer Modelling,* Vol. 23, No. 3, pp. 69-86, (1995).

22. T.C. Dietterich and R.S. Michalski, Inductive Learning of Structural Descriptions, *Artificial Intelligence*, **16** (1981).