Inference of a Minimum Size Boolean Function from Examples by Using a New Efficient Branch-and-Bound Approach

EVANGELOS TRIANTAPHYLLOU
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) triantd@lsu.edu

(Received: 1 October 1992; accepted: 22 September 1993)

Abstract. This paper deals with the problem of identifying a hidden Boolean function $\mathcal{F}$: $\{0,1\}^t \rightarrow \{0,1\}$ from positive and negative examples. This problem is of paramount importance in many real life applications of artificial intelligence. The method proposed in this paper is based on a branch-and-bound approach. This approach is an expansion of some earlier work (Triantaphyllou *et al.*, 1994). Computational results, comparing the new method with one based on Karmakar's interior point method, suggest that the new method is very efficient.

Key words. Inductive inference, Boolean functions, clause satisfiability problem, maximum clique, interior point method, artificial intelligence.

1. Introduction

This paper deals with the problem of *inductive inference*. That is, given sets of examples infer a set of rules. This problem is of considerable importance in artificial intelligence and, in particular, in machine learning. A number of algorithms which implement learning from examples can be found in [5, 8, 20, 21, 22, 23, 15]. An excellent survey on inductive inference approaches is given in [1]. Complexity issues of this type of learning have been studied by [27, 28, 17, 19].

In the type of learning considered in this paper, examples are classified either as positive or as negative. Then, the issue is to determine a Boolean expression which classifies all the positive and negative examples correctly. An early definition of this problem was given in Bongard [2] (the original book was published in Russian in 1967). This problem is *NP-complete* (see, for instance, [3, 9]). Some recent developments can be found in [23, 15]. Usually, such a Boolean function is expressed in the conjunctive normal form (CNF) or in the disjunctive normal form (DNF). See, for instance [4, 7, 10, 11, 12, 13, 14, 29, 30]. Peysakh in [18] describes an algorithm for converting any Boolean expression into CNF.

The general form of a CNF and DNF system (i.e., a Boolean function) is

Journal of Global Optimization **5**: 69–94, 1994.

defined as (I) and (II), respectively. That is:

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

and

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

where a_i is either A_i or $\bar{A}_i$. That is, a CNF expression is a conjunction of disjunctions, while a DNF expression is a disjunction of conjunctions. These conjunctions and disjunctions are also called *logical clauses*.

Let $\{A_1, A_2, A_3, \ldots A_t\}$ be a set of t Boolean *predicates* or *atoms*. Each atom A_i $(i = 1, 2, 3, \ldots, t)$ can be either true (denoted by 1) or false (denoted by 0). Let $\mathcal{F}$ be a *Boolean function* over these atoms. That is, $\mathcal{F}$ is a mapping from $\{0,1\}^t \rightarrow \{0,1\}$ which determines for each combination of truth values of the arguments $A_1, A_2, A_3, \ldots, A_t$ of $\mathcal{F}$, whether $\mathcal{F}$ is true or false (denoted as 1 and 0, respectively). For each Boolean function $\mathcal{F}$, the *positive examples* are the vectors $v \in \{0,1\}^t$ such that $\mathcal{F}(v) = 1$. Similarly, the *negative examples* are the vectors $v \in \{0,1\}^t$ such that $\mathcal{F}(v) = 0$. Therefore, given a function $\mathcal{F}$ defined on the t atoms $\{A_1, A_2, A_3, \ldots, A_t\}$, then a vector $v \in \{0,1\}^t$ is either a positive or a negative example. Equivalently, we say that a vector $v \in \{0,1,\}^t$ is *accepted* (or *rejected*) by a Boolean function $\mathcal{F}$ if and only if the vector v is a positive (or a negative) example of $\mathcal{F}$.

In the present paper, a set of positive examples will be denoted as E^+. Similarly, a set of negative examples will be denoted as E^-. The cardinalities of the two sets E^+ and E^- will be denoted as M_1 and M_2, respectively. Given two sets of positive and negative examples, then the constraints to be satisfied by a system (i.e., a Boolean function) are as follows. In the CNF case, each positive example should be accepted by all the disjunctions in the CNF expression and each negative example should be rejected by at least one of the disjunctions. In the case of DNF systems, any positive example should be accepted by at least one of the conjunctions in the DNF expression, while each negative example should be rejected by all the conjunctions.

The *main idea of this paper* is described in the following illustration. A series of examples (binary vectors) is somehow known. These examples are classified as either positive or negative. For this illustration suppose that the following positive and negative examples are known:

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

What we seek is a set of logical clauses which correctly classify *all the examples*. One such set of logical clauses (in CNF form) is as follows:

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

The *main problem* examined in this paper is how to construct a set of logical clauses (i.e., a Boolean function) which satisfies the requirements of two collections of examples E^+ and E^-. Given two sets of positive and negative examples, it is possible that more than one Boolean function satisfy the corresponding constraints. For reasons of simplicity, however, we are interested in a Boolean function with the *minimum number* of conjunctions or disjunctions (if it is in the DNF or CNF form, respectively).

2. Some Background Information

The following two sub-sections briefly describe two clause inference algorithms. Both algorithms use collections of positive and negative examples as the input data. The first algorithm is based on a branch-and-bound approach and infers CNF clauses [23]. The second algorithm infers DNF clauses and is based on formulating a *clause satisfiability* (*SAT*) *problem* [15] and then solving this SAT by using an interior point method described in [16].

Besides the fact that the first algorithm infers CNF systems, while the second infers DNF systems, the two approaches have another major difference. The first approach uses a heuristic which returns a very small (not necessarily minimal) number of disjunctions in the proposed CNF system. However, the second approach *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. Therefore, by using successively lower k values the SAT approach can be used to determine a *minimum size* DNF system (by minimum size we mean a system with the minimum number of conjunctions).

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). In [24] it is shown that it is possible to derive either a CNF or DNF system using *any clause inference algorithm*. Furthermore, some decomposition approaches for large scale inference problems can be found in [25]. Finally, an approach for *guided learning* can be found in [26]. Guided learning can be used to derive the examples in such a way that a hidden Boolean function can be inferred by using only a few examples (when compared with *random* collections of examples).

2.1. THE ONE CLAUSE AT A TIME APPROACH

In [23] an algorithm which infers CNF systems from positive and negative examples is developed. In that approach, CNF clauses are generated in a way which approximates the minimum 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, rejects all the negative examples and accepts 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 it 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 (i.e., the first) clause so far. The iterations continue until a set of clauses is constructed which reject all the negative examples in the original E^- set (see also Figure 1). Recall that M_2 denotes the number of the negative examples. Then the following theorem states a critical property of the OCAT approach.

THEOREM 1 [23]. *The OCAT approach terminates within* M_2 *iterations.*

The core of the OCAT approach is step 2, in Figure 1. In [23] a branch-and-bound based algorithm is presented which solves the problem posed in step 2. The OCAT approach returns the set of desired clauses (i.e., the CNF system) as set C. Because the proposed branch-and-bound approach is an extension of the branch-and-bound approach developed in [23], the next sub-section briefly highlights the old approach.

$i = 0$; $C = \emptyset$;
DO WHILE ($E^- \neq \emptyset$)
  Step 1: $i \leftarrow i + 1$; /*i indicates the i-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;

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

2.2. THE ORIGINAL BRANCH-AND-BOUND APPROACH

The original branch-and-bound (B & B) approach is best described with an illustrative example. Consider the E^+ and E^- examples given in the introduction section. We number the positive examples as $(1, 2, 3, 4)$ and the negative examples as $(1, 2, 3, 4, 5, 6)$. The B & B approach will determine a *single clause* which accepts all the positive examples in the E^+ set, while rejecting as many negative examples from the current E^- set as possible.

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 $(1, 1, 0, 0)$ indicates 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 $\{A_1, \bar{A}_2, \bar{A}_3, A_4\}$.

At this point 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 current illustrative example the NEG(A_k) sets are presented in Table I. Furthermore, denote by ATOMS(v) the set of the atoms that are true in a particular (either positive or negative) example v (where $v \in \{1, 0\}^t$, and t is the number of atoms). In the light of the sets with the negative examples in Table I and the requirement the proposed clause to accept all the four positive examples and reject as many negative examples as possible, the following minimization problem is derived:

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

subject to:

$$\beta_1 \in B_1 = \{\{2, 4\}, \{3, 6\}, \{2, 4, 5\}, \{1, 4, 5, 6\}\}$$
$$\beta_2 \in B_2 = \{\{1, 3, 5, 6\}, \{3, 6\}, \{2, 4, 5\}, \{1, 4, 5, 6\}\}$$
$$\beta_3 \in B_3 = \{\{2, 4\}, \{1, 2, 4, 5\}, \{1, 3, 6\}, \{2, 3\}\}$$
$$\beta_4 \in B_4 = \{\{1, 3, 5, 6\}, \{1, 2, 4, 5\}, \{2, 4, 5\}, \{2, 3\}\}.$$

Table I. 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}

This formulation leads to a B & B search as follows. The search has *four stages* (or levels in the search tree); one stage for each of the four positive examples. The root of the search tree is the empty set. Each interior node (i.e., nodes with descendents), say at level h (where $1 \leq h < 4$), is connected to t (where t is the number of atoms) nodes in the next higher level via t arcs. These t arcs represent the atoms that are true in the h-th positive example (i.e., the members of the set ATOMS(α_h), where α_h is the h-th positive example). The nodes (or *search states*) in this graph represent sets of negative examples.

The set of negative examples which corresponds to a node (state) is the set of all the negative examples accepted by the atoms which correspond to the arcs that connect that node with the root node. That is, if one is at node (state) Y_K and follows the arc which corresponds to the atom A_i, then the resulting state, say Y_L, is:

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

The search tree for the current illustrative example is depicted in Figure 2. Note that *not all* states need to be expanded. This occurs because each node of the tree is examined in terms of two fathoming tests. If any of these two tests succeeds, then the 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 which correspond to the leaves (terminal nodes) which 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) which 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 which has another state (in the current stage) as a subset, then that state (node) *can be fathomed* without eliminating any optimal solutions. This is the first fathoming test.

For the second fathoming test, 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. The way these fathoming tests were applied and more on this B & B search, along with some computational results, can be found in [23].

2.3. CLAUSE INFERENCE AS A SATISFIABILITY PROBLEM

In [15] 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

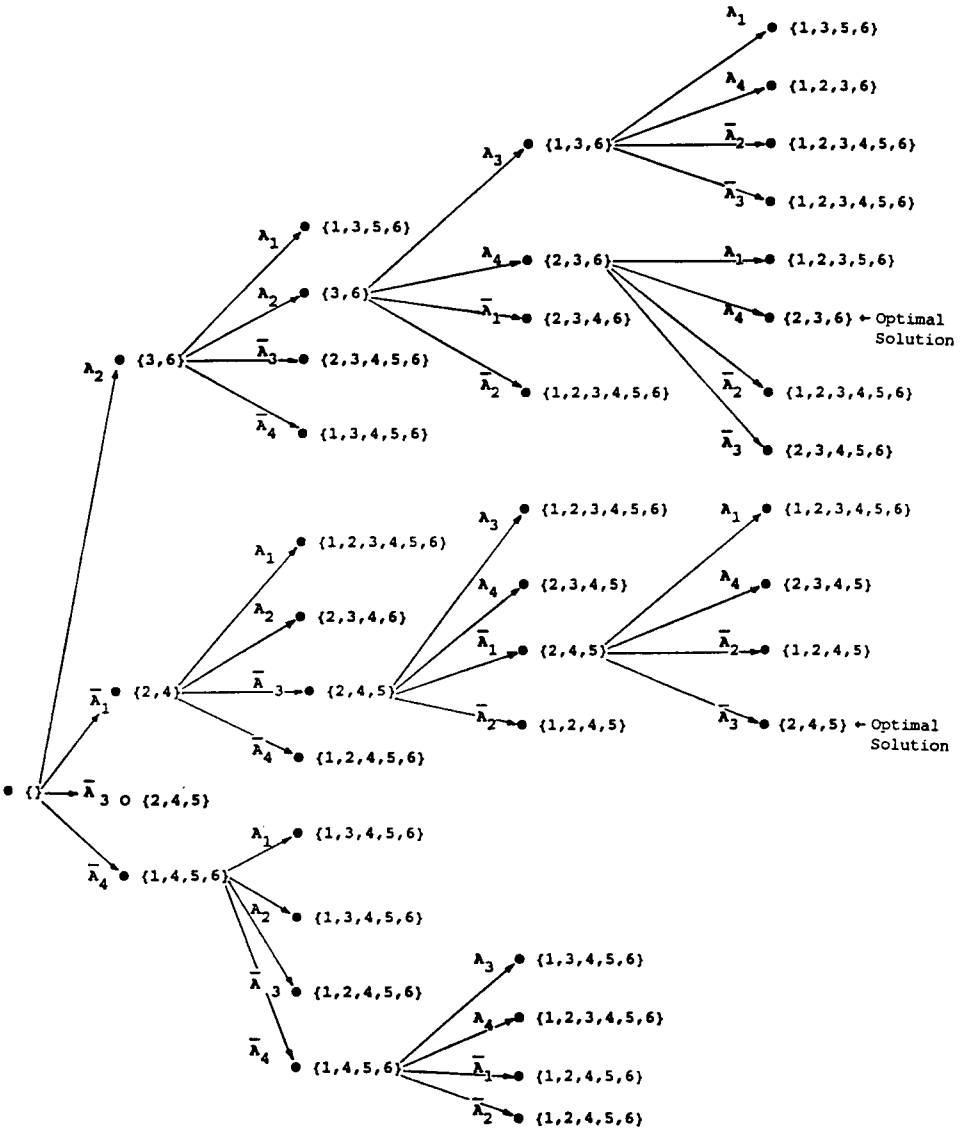


Fig. 2. The search tree when the original B & B approach is applied on the illustrative example.

interior point method developed in [16] to solve it. This approach *pre-assumes the value of k*; the number of conjunctions in the DNF system. The SAT problem uses the following Boolean variables [15]:

$$s_{ji} = \begin{cases} 0 & \text{if } A_1 \text{ is in the } j\text{-th conjunction} \\ 1 & \text{if } A_i \text{ is not in the } j\text{-th conjunction} \end{cases}$$

$$s'_{ji} = \begin{cases} 0 & \text{if } \bar{A}_1 \text{ is in the } j\text{-th conjunction} \\ 1 & \text{if } \bar{A}_i \text{ is not in the } j\text{-th conjunction} \end{cases}$$

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

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

Then, the clauses of this SAT problem are as follows (where n is the number of atoms):

$$s_{ji} \vee s'_{ji}, \quad i = 1, \ldots, n, \quad j = 1, \ldots, k \tag{1}$$

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

$$\bigvee_{j=1}^{k} z_j^{\alpha}, \quad \alpha = 1, \ldots, M_1 \tag{3}$$

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

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 conjunction. In general, this SAT problem has $k(n(M_1 + 1) + M_2) + M_1$ clauses, and $k(2n + M_1)$ Boolean variables. A detailed example of this formulation can be found in [15].

3. The New Branch-and-Bound Algorithm

The new branch-and-bound algorithm will also be demonstrated on the examples presented in the first section. Recall that these examples are as follows:

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

At first, it will be shown how the algorithm can derive a CNF clause. Next, the basic algorithm will be modified to derive DNF clauses. Recall that for the CNF case, the requirement is the clause to accept all the positive examples, while rejecting as many negative examples as possible.

3.1. GENERATING A CNF CLAUSE

Define as POS(A_k) the set of the positive examples which are accepted by a CNF clause when the atom A_k is included in that clause. The new B & B algorithm also uses the concepts of the NEG(A_k) and ATOMS(v) sets, as they were defined in the previous section. The POS(A_k) sets for the current illustrative example are given in Table II. For the new search, the NEG(A_k) sets in Table I are also going to be used.

Now the *search states* are described in terms of *two sets*. The first set refers to the *positive examples* which are accepted by the atoms which correspond to the arcs which connect that state (node) with the root node. Similarly, the second set refers to the *negative examples* which are accepted by the atoms which correspond to the arcs which connect that state with the root node. Suppose that we are at state $S_i = [P_i, N_i]$ (where P_i, N_i correspond to the previous two sets of positive and negative examples, respectively). Now assume that the search considers the state (node) which is derived by following the arch which corresponds to the atom A_k. Then, the new state is: $S_j = [P_j, N_j]$, where the new sets P_j and N_j are defined as follows:

$$P_j = P_i \cup \text{POS}(A_k), \quad \text{and}$$
$$N_j = N_i \cup \text{NEG}(A_k).$$

Therefore, the search continues until terminal states are reached. A state $S_i = [P_i, N_i]$ is a *terminal state* if and only if the set P_i refers to *all positive* examples. That is, if and only if $P_i = \{1, 2, 3, \ldots, M_1\}$. Apparently, a terminal state with a *minimum cardinality* of the set N_i is *optimal* (in the OCAT sense). In the light of the previous considerations, the problem to be solved by the B & B search can be summarized as follows (where a_i is either A_i or $\bar{A}_i$):

Table II. The POS(A_k) sets for the illustrative example

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

Find a set of atoms S such that the following two conditions are true:

$$\left| \bigcup_{a_i \in S} \text{NEG}(a_i) \right| = \text{MINIMUM}, \quad \textit{and}$$

$$\bigcup_{a_i \in S} \text{POS}(a_i) = E^+ .$$

Given the above definitions some useful derivations are possible. We say that a state S_i *absorbs* another state S_j if by expanding the state S_j we cannot reach any better terminal state than the ones derived by expanding the state S_i. In such a case we call the state S_j an *absorbed state*. From the previous considerations it becomes obvious that once a state can be identified to be an absorbed state, then it can be dropped from further consideration. Then the following two theorems are applicable only when a CNF clause is to be generated and they provide some conditions for identifying absorbed states. The proofs of these theorems follow directly from the previous definitions and discussion.

THEOREM 2. *The state* $S_i = [P_i, N_i]$ absorbs *the state* $S_j = [P_j, N_j]$ *if the following condition is true:*

$$P_j \subseteq P_i \quad \textit{and} \quad N_i \subseteq N_j .$$

THEOREM 3. *Suppose that* $S_i = [P_i, N_i]$ *is a* terminal *state. Then, any state* $S_j = [P_j, N_j]$, *such that* $|N_j| \geq |N_i|$, *is* absorbed *by the state* S_i.

The search tree for the current illustrative example is depicted in Figure 3. Observe that the arcs starting from a given node (search state) *are not* in the order $A_1, A_2, A_3, \ldots, \bar{A}_4$ (as was the case with the original B & B search) but, instead, they are *ranked*. They are ranked in terms of two criteria as follows. The first criterion is to rank the atoms in descending order of the size of the corresponding $\text{POS}(A_k)$ set (as given in Table II). If there is a tie, then they are ranked in ascending order in terms of the size of the corresponding $\text{NEG}(A_k)$ set (as given in Table I). Therefore, the resulted ranking is as follows: $\bar{A}_3, A_2, A_4, \bar{A}_1, \bar{A}_4, A_1, \bar{A}_2, A_3$. In this way, it is more likely to reach terminal states quickly, and thus the fathoming test of Theorem 2 can be utilized more frequently.

The search tree in Figure 3 indicates that fathoming of states may occur very often. In this figure, Rule #2 refers to the application of Theorem 3. For instance, the state $[\{1, 2, 4\}, \{1, 2, 3, 5\}]$ needs not to be expanded (i.e., it is an absorbed state) because there is a terminal state which has a size of the N_i set equal to 3 (the absorbed state has a corresponding value of 4). Since the process was successful in determining a terminal state very early, new states are generated by considering *at first* their N_i sets. If the N_i sets have a size greater or equal to 3, then they are fathomed.

This is the reason why their P_i sets *do not need* to be considered. For this

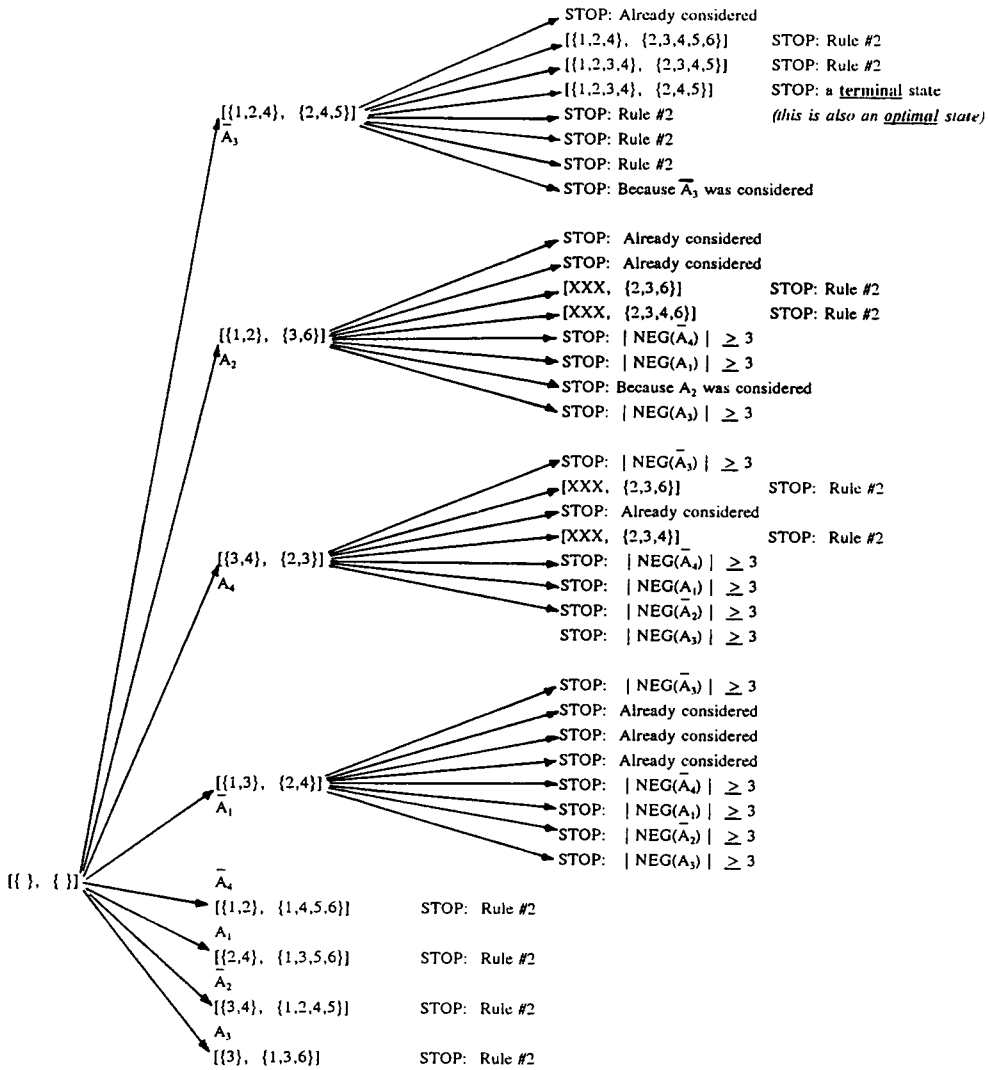


Fig. 3. The search tree under the *New* Branch-and-Bound approach.

reason they are indicated with the symbol XXX in Figure 3. It is interesting to observe that any arc A_k for which the size of the NEG(A_k) set is greater or equal to 3 necessarily leads to an absorbed state. Therefore, all states derived by such arcs are fathomed. Now consider the state $[\{1,2\},\{3,6\}]$. This state was created by applying the atom A_2 on the root state. If we apply the negation of A_2 (i.e., $\bar{A}_2$) on any of its descendent states, then the new state would have *both* the atoms A_2 and $\bar{A}_2$. However, if that happens then that state would accept all the negative (and positive) examples. This is the reason why the seventh child state of

$[\{1,2\}, \{3,6\}]$ is fathomed in the B & B search. In this illustrative example Theorem 1 was never utilized.

In Figure 3 only the arcs of the first level bear the indication of the appropriate atom. Arcs in the second level bear no indication of atom because of space limitations in this figure. However, the atoms in the order $\bar{A}_3$, A_2, A_4, $\bar{A}_1$, $\bar{A}_4$, A_1, $\bar{A}_2$, A_3 are assumed on these arcs as well. Since the state $[\{1,2,3,4\}, \{2,4,5\}]$ is the only unfathomed terminal state, this is also an optimal state. Therefore, the derived CNF clause is $(\bar{A}_3 \vee \bar{A}_1)$. This clause accepts all the positive examples and also accepts the negative examples $\{2,4,5\}$ (that is, it rejects the negative examples $\{1,3,6\}$).

From the previous considerations it follows that there is a great advantage to reach terminal nodes early in the search process. In this way, the minimum size of their N_i sets can be used to effectively fathom search states. This situation suggests the application of the B & B search in *two phases* (as was the case with the original B & B search). During the first phase only a very small number (say, 10) of active states is allowed. If there are more than 10 active states, then they are ranked according to their P_i and N_i sizes (i.e., in a manner similar to ranking the atoms). In this way, the states with the highest potential of being optimal are kept into memory. This is the principle of *beam search* in artificial intelligence (see, for instance, [8]). At the end of phase one, a terminal state of small cardinality becomes available. Next, phase two is initiated. During the second phase a larger number (say, 50) of active states is allowed. However, states now can be fathomed more frequently because the size of a small N_i set of a terminal state is known. Also note that the new B & B search does not have to follow a fixed number of stages (as was the case with the original B & B search).

An important issue with the previous two phases is to be able to decide when a terminal state is optimal (in the OCAT sense). As it was mentioned above, memory limitations may force the search to *drop* states which are *not* absorbed by any other state. Therefore, *there is a possibility to drop a state which could have lead to an optimal state* (*and thus to an optimal clause*).

Suppose that L non-absorbed states had to be dropped because of memory limitations. Let $K_1, K_2, K_3, \ldots, K_L$ represent the cardinalities of their corresponding N_i sets. Next, define the quantity K_{MIN} as the minimum of the previous L numbers. Similarly, suppose that the B & B process has identified N terminal states. Let $Y_1, Y_2, Y_3, \ldots, Y_N$ represent the cardinalities of their corresponding N_i sets. Define as Y_{MIN} the minimum of the previous N cardinalities. Then, the previous considerations lead to the proof of the following theorem which states a condition for establishing optimality.

THEOREM 4. *A terminal state* $S_i = [P_i, N_i]$ *is also an* optimal state *if the following two conditions are true:*

$$|N_i| = Y_{\text{MIN}} \quad \textit{and} \quad K_{\text{MIN}} \geq Y_{\text{MIN}} .$$

Note that this theorem can be applied after each one of the two phases. Obviously, if it is applicable after the first phase, then the second phase does not need to be initiated. The following lemma states a fact when optimality is not provable.

LEMMA 1. *Suppose that the following relation is true:* $K_{\text{MIN}} < Y_{\text{MIN}}$. *Then, an optimal clause accepts no less than* K_{MIN} *negative examples.*

This lemma indicates that if optimality cannot be proven, then it is still possible to establish a *lower limit* on the number of negative examples which can be accepted by an optimal clause (or, equivalently, an upper limit on the number of negative examples which can be *rejected* by an optimal clause).

Finally, it should be stated here that the CNF version of the B & B algorithm was implemented in FORTRAN and used in the computational experiments described in a following section. The next sub-section briefly describes a modification of this B & B approach which can derive DNF systems.

3.2. GENERATING A DNF CLAUSE

The previously described B & B search can be easily modified to generate DNF clauses. Recall that the requirements now are as follows. Every positive example should be accepted by *at least one* of the conjunctions (single DNF clause). Also, every negative example should be rejected by *all* the conjunctions. Therefore, the DNF clause proposed by the B & B approach during a single OCAT iteration should reject all the negative examples and accept as many positive examples as possible.

The search states are defined in a manner analogous to that of the CNF case. Each state is described in terms of two sets. The first set (denoted by P_i) refers to the *positive examples* which are accepted by the atoms which correspond to the arcs which connect that state (node) with the root node. Similarly, the second set (denoted by N_i) refers to the *negative examples* which are accepted by the atoms which correspond to the arcs which connect that state with the root node. Suppose that we are at state $S_i = [P_i, N_i]$. Now assume that the search considers the state (node) which is derived by following the arch which corresponds to the atom A_k. Then, the new state is: $S_j = [P_j, N_j]$, where the new sets P_j and N_j are defined as follows:

$$P_j = P_i \cap \text{POS}(A_k), \quad \text{and}$$
$$N_j = N_i \cap \text{NEG}(A_k).$$

In other words, instead of set union now there is set *intersection*. A state $S_i = [P_i, N_i]$ is a *terminal state* if and only if the set N_i is equal to the *empty set*. That is, a state is terminal if it rejects all the negative examples. A terminal state with the *maximum cardinality* of the set P_i is *optimal* (in the OCAT sense).

The concept of *absorbed* states is the same as in the CNF case. However, in the light of the previous definitions Theorem 3 takes the following form:

THEOREM 5. *Suppose that* $S_i = [P_i, N_i]$ *is a* terminal *state. Then, any state* $S_j = [P_j, N_j]$, *such that* $|P_j| \leq |P_i|$, *is* absorbed *by the state* S_i.

The atoms in the arcs now are ranked in an analogous manner as in the previous sub-section. The first criterion is to rank the atoms in ascending (instead of descending) order of the size of the corresponding NEG(A_k) set. If there is a tie, then they are ranked in descending (instead of ascending) order in terms of the size of corresponding POS(A_k) set.

As was the case with the CNF version of the B & B algorithm, suppose that L non-absorbed states had to be dropped because of memory limitations. Let K_1, K_2, $K_3, \ldots, K_L$ represent the cardinalities of their corresponding P_i sets (note that before we considered the N_i sets). Next, define the quantity K_{MAX} as the *maximum* of the previous L numbers (before we considered the minimum values). Similarly, suppose that the B & B process has identified N terminal states. Let Y_1, Y_2, $Y_3, \ldots, Y_N$ represent the cardinalities of their corresponding P_i sets. Define as Y_{MAX} the *maximum* of the previous N cardinalities. Then, the previous considerations lead to the proof of a theorem, analogous to Theorem 4, regarding the establishment of optimality.

THEOREM 6. *A terminal state* $S_i = [P_i, N_i]$ *is also an* optimal state *if the following conditions are true:*

$$|P_i| = Y_{\text{MAX}} \quad \textit{and} \quad K_{\text{MAX}} \leq Y_{\text{MAX}}.$$

The following section briefly describes a modification of the original SAT method. As it was described in Section 2.3, the original SAT method derives DNF systems. The modified version derives CNF systems. After that development both the OCAT with the B & B algorithm and the SAT approach can derive CNF or DNF systems.

4. An SAT Approach for Inferring CNF Clauses

The SAT formulation for deriving CNF systems is based on the original SAT formulation for deriving DNF systems (as described in Section 2.3). The variables used in the new formulation are similar to the ones used in the DNF case. They are defined as follows:

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

$$s'_{ji} = \begin{cases} 0 & \text{if } \bar{A}_1 \text{ is in the } j\text{-th disjunction} \\ 1 & \text{if } \bar{A}_i \text{ is not in the } j\text{-th disjunction} \end{cases}$$

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

$$z^{\beta}_{j} = \begin{cases} 1 & \text{if the negative example } \beta \text{ is accepted by the } j\text{-th disjunction} \\ 0, & \text{otherwise .} \end{cases}$$

The clauses of this SAT formulation for deriving a CNF system which has up to k disjunctions are as follows (where n is the number of atoms):

$$s_{ji} \vee s'_{ji}, \quad i = 1, \ldots, n, \quad j = 1, \ldots, k \tag{1}$$

$$\left(\bigvee_{i \in P_r} \bar{s}_{ji} \right) \vee \left(\bigvee_{i \in \bar{P}_r} \bar{s}'_{ji} \right), \quad j = 1, \ldots, k, \quad r = 1, \ldots, M_1 \tag{2}$$

$$\bigvee_{j=1}^{k} z^{\beta}_{j}, \quad \beta = 1, \ldots, M_2 \tag{3}$$

$$\sigma^{\alpha}_{ji} \vee \bar{z}^{\beta}_{j}, \quad i = 1, \ldots, n, \quad j = 1, \ldots, k, \quad \beta = 1, \ldots, M_2, \tag{4}$$

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

Clauses of type (1) ensure that *never both* A_i and $\bar{A}_i$ will appear in any disjunction at the same time. Clauses of type (2) ensure that each positive example will be accepted by *all* k disjunctions. Clauses of type (3) ensure that each negative example will be rejected by at *least one* of the k disjunctions. Finally, clauses of type (4) ensure that $\bar{z}^{\beta}_{j} = 1$ if and only if the negative example β is rejected by the j-th conjunction. In general, this SAT problem has $k(n(M_2 + 1) + M_1) + M_2$ clauses, and $k(2n + M_2)$ Boolean variables.

5. The Rejectability Graph of Two Collections of Examples

This section presents the motivation and definition of a special graph, called the *rejectability graph*, which can be easily derived from positive and negative examples. The concept of this graph was first introduced in [25]. This graph can be used to establish a *lower limit* on the number of clauses which can be derived from positive and negative examples. When this lower limit is equal to the number of clauses derived by OCAT, then *the conclusion is that OCAT derived a system with the minimum number of clauses.*

To understand the motivation for introducing this graph consider a situation with $t = 5$ atoms. Suppose that the vector $v_1 = (1, 0, 1, 0, 1)$ *is a positive example* while the two vectors $v_2 = (1, 0, 1, 1, 1)$ and $v_3 = (1, 1, 1, 0, 1)$ are *negative examples*. Recall that ATOMS(v) denotes the set of the atoms which are true in a particular (either positive or negative) example v (where $v \in \{1, 0,\}^t$, and t is the number of atoms). Now observe that there is no *single CNF clause* which can *simultaneously* reject both the two negative examples v_2 and v_3, while at the same time accept the positive example v_1.

This is true because any clause which simultaneously rejects the two examples v_2 and v_3, should not contain any of the atoms in the *union* of the two sets ATOMS(v_2) and ATOMS(v_3). But if none of the atoms of the set $\{A_1, A_2, \bar{A}_2, A_3, A_4, \bar{A}_4, A_5\} = \text{ATOMS}(v_2) \cup \text{ATOMS}(v_3)$ is present in the clause, then it is *impossible* to accept the positive example $v_1 = (1, 0, 1, 0, 1)$. Therefore, given any clause which accepts the positive example v_1, the previous two negative examples v_2 and v_3 cannot also be *rejected* by this clause. In general, given a set of positive examples E^+, then two negative examples v_1 and v_2 are rejectable by a single clause if and only if the condition in the following theorem is satisfied:

THEOREM 7 [25]. *Let E^+ be a set of positive examples and v_1, v_2 be two negative examples. There exists a clause which accepts all the positive examples and rejects both negative examples v_1 and v_2 if and only if:*

$$\text{ATOMS}(v_i) \not\subseteq \text{ATOMS}(v_1) \cup \text{ATOMS}(v_2), \quad \textit{for each positive example } v_i \in E^+ .$$

The above theorem follows directly from the previous considerations. Given two collections of positive and negative examples, denoted as E^+ and E^-, respectively, Theorem 7 motivates the construction of a graph $G = (V, E)$ as follows:

$V = \{V_1, V_2, V_3, \ldots, V_{M_2}\}$

where M_2 is the size of the E^- set,

$E = \{(V_i, V_j)$ *if and only if the i-th and the j-th examples in E^- are rejectable by a single clause (subject to the examples in E^+)}.*

We denote this graph as the *rejectability graph* of E^+ and E^-. The previous theorem indicates that it is computationally straightforward to construct this graph. If there are M_2 negative examples, then the maximum number of edges is $M_2(M_2 - 1)/2$. Therefore, the rejectability graph can be constructed by performing $M_2(M_2 - 1)/2$ simple rejectability examinations. The rejectability graph G of a set of positive and a set of negative examples possesses a number of interesting properties (for more details see [25]). The following theorem refers to *any clique* of the rejectability graph.

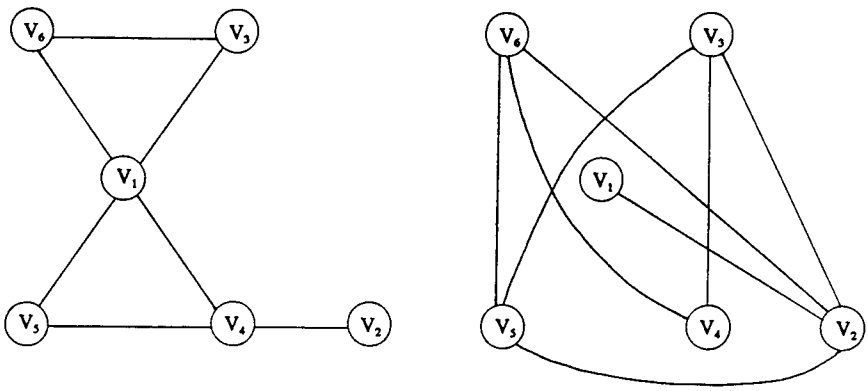
THEOREM 8 [25]. *Suppose that the two sets E^+ and E^- are given and $\mathcal{F}$ is a subset of k negative examples from E^- ($k \leq$ size of set E^-) with the property that the subset can be rejected by a single CNF clause which also accepts each of the positive examples in E^+. Then, the vertices corresponding to the k negative examples in the rejectability graph G form a clique of size k.*

The previous theorem states that any set of negative examples which can be rejected by a single clause corresponds to a clique in the rejectability graph. However, *the converse is not true*. That is, not every clique in the rejectability graph corresponds to a set of negative examples which can be rejected by a single clause. Let $\bar{G}$ be the *complement* of the rejectability graph G of the two sets of examples. At this point let $\omega(\bar{G})$ be the *size of the maximum clique* of the graph $\bar{G}$. Then, the following theorem states a *lower bound* on the *minimum number* of clauses which can reject all the negative examples in E^-, while accepting all the positive examples in E^+.

THEOREM 9 [25]. *Suppose that E^+ and E^- are the sets of the positive and negative examples, respectively. Let r be the minimum number of clauses which reject all the examples in E^-, while accepting all the examples in E^+. Then, the following relation is true:*

$$r \geq \omega(\bar{G}) .$$

As an illustrative example, consider the E^+ and E^- examples given in the introduction section. The corresponding rejectability graph and its complemented graph are depicted in Figure 4. The size of the maximum clique of the complemented graph is 3. Thus, a lower limit on the number of clauses is 3. Since



The Rejectability Graph

The Complemented Graph

Fig. 4. The rejectability and its complemented graph of E^+ and E^-.

the OCAT approach yields 3 clauses for these data, and 3 is a lower bound, then we can conclude that for this case *OCAT derived à minimum size system.*

6. Some Computational Results

Several computational experiments were conducted in order to gain a better understanding of the performance of the new branch-and-bound approach. These experiments were conducted in a manner similar to the experiments reported in [15]. At first, a system was chosen to be the "hidden logic". This system is *hidden* in the sense that we try to infer it from limited collections of positive and negative examples. The systems used as "hidden logic" are exactly the same as the ones also used in [15]. These are the 15 systems depicted in Table III.

Once a "hidden logic" is selected, a number of random examples was generated. Each random example was classified according to the "hidden logic"

Table III. Description of the systems used as "hidden logic" in the computer experiments

8A	$(A_4 \vee \bar{A}_7) \wedge (\bar{A}_3 \vee A_4) \wedge (A_1 \vee A_2 \vee \bar{A}_6)$	16D	$(\bar{A}_5 \vee \bar{A}_8 \vee \bar{A}_{10} \vee A_{16}) \wedge (\bar{A}_2 \vee \bar{A}_{12} \vee \bar{A}_{16}) \wedge (\bar{A}_1 \vee \bar{A}_{12}) \wedge (A_3 \vee \bar{A}_5 \vee A_6)$
8B	$(\bar{A}_1 \vee \bar{A}_4 \vee A_6) \wedge (A_2) \wedge (\bar{A}_2 \vee A_8)$	16E	$(A_1 \vee \bar{A}_2 \vee A_3 \vee \bar{A}_4) \wedge (A_5 \vee A_6 \vee \bar{A}_7 \vee A_8) \wedge (A_9 \vee \bar{A}_{10} \vee \bar{A}_{11} \vee \bar{A}_{12}) \wedge (\bar{A}_{13} \vee A_{14} \vee \bar{A}_{15} \vee A_{16})$
8C	$(A_5) \wedge (A_6 \vee \bar{A}_8) \wedge (A_7)$	32A	$(A_1 \vee \bar{A}_{12}) \wedge (A_2 \vee \bar{A}_5 \vee A_{32}) \wedge (A_{19} \vee \bar{A}_{23} \vee A_{26})$
8D	$(\bar{A}_6) \wedge (\bar{A}_2) \wedge (\bar{A}_3 \vee \bar{A}_7)$	32B	$(A_1 \vee A_2 \vee \bar{A}_9 \vee \bar{A}_{12} \vee A_{31}) \wedge (A_{19} \vee \bar{A}_{23} \vee A_{26}) \wedge (A_2 \vee \bar{A}_5 \vee \bar{A}_{20} \vee A_{32})$
8E	$(A_8) \wedge (A_2 \vee A_5) \wedge (\bar{A}_3 \vee A_5)$	32C	$(A_2 \vee \bar{A}_9 \vee \bar{A}_{12} \vee A_{31}) \wedge (A_2 \vee \bar{A}_{20} \vee A_{32}) \wedge (A_1 \vee A_2 \vee A_{19} \vee \bar{A}_{23} \vee A_{26})$
16A	$(A_1 \vee \bar{A}_{12}) \wedge (A_2 \vee A_3 \vee \bar{A}_5) \wedge (A_9) \wedge (\bar{A}_7)$	32D	$(A_4 \vee A_{11} \vee \bar{A}_{22}) \wedge (A_2 \vee A_{12} \vee \bar{A}_{15} \vee \bar{A}_{29}) \wedge (\bar{A}_3 \vee A_9 \vee A_{20}) \wedge (\bar{A}_{10} \vee A_{11} \vee \bar{A}_{29} \vee A_{32})$
16B	$(A_3 \vee A_{12} \vee A_{15}) \vee (\bar{A}_3 \vee \bar{A}_{11}) \wedge (\bar{A}_2 \vee \bar{A}_{10} \vee \bar{A}_{16}) \wedge (A_1 \vee A_2)$	32E	$(A_9 \vee A_{10} \vee A_{23}) \wedge (A_2 \vee A_{29} \vee \bar{A}_{31}) \wedge (A_2 \vee \bar{A}_4 \vee A_6 \vee \bar{A}_7 \vee A_{19} \vee \bar{A}_{32})$
16C	$(A_4 \vee \bar{A}_7 \vee A_{11}) \wedge (A_4 \vee A_{10} \vee A_{14}) \wedge (\bar{A}_9 \vee \bar{A}_{14} \vee A_{15}) \wedge (\bar{A}_3 \vee A_8)$		

as either positive or negative. The random examples used in these experiments are *identical* with the ones used in [15]. After the collections of the E^+ and E^- examples were generated this way, the OCAT approach with the CNF version of the new B & B algorithm were applied. In order to derive a DNF system, the main result in [24] was applied.

To see how this can be accomplished let v be an example (either positive or negative). Then, $\bar{v}$ is defined as the *complement* of example v. For instance, if $v = (1, 0, 0)$, then $\bar{v} = (0, 1, 1)$. The following definition introduces the concept of the complement of a set of examples. Let E be a collection of examples (either positive or negative). Then, $\bar{E}$ is defined as the *complement of the collection E*. Recall that in the first section the general form of a CNF and DNF system was defined as (I) and (II), respectively.

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

and

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

where a_i is either A_i or $\bar{A}_i$. Then, the following theorem states an important property which exists when CNF and DNF systems are inferred from collections of positive and negative examples.

THEOREM 10 [24]. *Let E^+ and E^- be the sets of positive and negative examples, respectively. A CNF system given as (I) satisfies the constraints of the E^+ and E^- sets if and only if the DNF system given as (II) satisfies the constraints of $\bar{E}^-$ (considered as the positive examples) and $\bar{E}^+$ (considered as the negative examples).*

Furthermore, in these experiments the rejectability graph and the maximum clique of its complemented graph were determined as well. For the maximum clique the computer code reported in [6] was used. Then Theorem 9 was used to establish the *lower limit* reported in Table IV.

When a system was inferred, it was compared in terms of 10,000 random examples with the "hidden logic". That is, 10,000 random examples were generated and then they were classified according to these two systems. The percentage of the times the two systems agreed, was reported as the *accuracy* of the inferred system. For the case of the systems which were defined on 8 atoms (i.e., systems 8A1, 8A2, 8A3, . . . , 8E2), all possible 256 examples were considered. The active list in the B & B search contained up to 10 search states at any time. However, this restriction did not prevent the search from reaching optimal

Table IV. Solution statistics

Problem Characteristics		SAT Solution Characteristics				OCAT Solution Characteristics				
Problem ID	No of Examples	k value	No of Clauses	CPU time	Accuracy	No of Clauses	Lower limit	CPU time	Accuracy	CPU time comparison with SAT
8A1	10	3	3	0.42	0.86	3	2	0.004	0.81	105
8A2	25	6	3	21.37	0.87	3	2	0.004	0.86	5,343
8A3	50	6	3	29.77	0.92	3	3	0.007	0.92	4,253
8A4	100	6	3	9.33	1.00	3	3	0.015	1.00	622
8B1	50	3	2	2.05	0.97	2	2	0.002	0.97	1,025
8B2	100	6	3	97.07	0.97	3	3	0.006	0.99	16,178
8B3	150	10	3	167.07	0.96	3	3	0.007	0.96	23,867
8B4	200	6	3	122.62	1.00	3	3	0.009	0.97	13,624
8C1	50	10	2	8.02	0.94	2	2	0.002	0.94	4,010
8C2	100	10	3	84.80	1.00	3	3	0.005	0.94	16,960
8D1	50	10	3	116.98	0.94	3	3	0.004	1.00	29,245
8D2	100	10	3	45.80	1.00	3	3	0.006	1.00	7,620
8E1	50	10	3	122.82	1.00	3	3	0.005	1.00	24,564
8E2	100	10	3	16.68	1.00	3	3	0.007	1.00	2,383
16A1	100	15	4	2,038.55	1.00	4	3	0.065	1.00	31,362
16A2	300	6	4	607.80	1.00	4	4	0.134	1.00	4,536
16B1	200	8	5	78.27	0.99	4	4	0.402	1.00	195
16B2	400	4	4	236.07	1.00	4	4	0.284	1.00	831
16C1	100	20	5	757.55	0.87	4	4	0.314	1.00	2,413
16C2	400	4	4	520.60	1.00	4	4	0.477	1.00	1,091
16D1	200	10	4	1,546.78	1.00	4	4	0.089	1.00	17,380
16D2	400	4	4	544.25	1.00	4	4	0.105	1.00	5,183
16E1	200	15	5	2,156.42	0.99	5	4	1.545	1.00	1,396
16E2	400	4	4	375.83	1.00	4	4	2.531	1.00	149
32A1	250	3	3	176.73	1.00	3	3	0.134	1.00	1,319
32B1	50	3	3	5.02	0.83	2	1	0.044	0.93	114
32B2	100	3	3	56.65	0.96	3	2	0.058	0.96	977
32B3	250	3	3	189.80	0.97	3	2	0.143	0.97	1,327
32B4	300	3	3	259.43	1.00	3	2	0.198	0.99	1,310
32C1	50	3	3	23.85	0.80	2	1	0.019	0.92	1,255
32C2	100	3	3	9.38	0.88	2	1	0.020	0.91	469
32C3	150	3	3	14.27	0.92	3	1	1.969	0.91	7
32C4	1000	3	3	154.62	1.00	3	3	0.654	1.00	236
32D1	50	4	4	65.65	0.74	3	1	0.197	0.81	333
32D2	100	4	4	178.10	0.91	3	2	0.308	0.92	578
32D3	400	4	4	1,227.40	1.00	4	2	1.103	1.00	1,113
32E1	50	3	2	8.33	0.86	2	1	0.015	0.83	555
32E2	100	3	3	9.67	0.97	2	1	0.096	0.99	101
32E3	200	3	3	132.83	0.98	3	2	0.105	0.98	1,265
32E4	300	3	3	276.93	0.98	3	2	0.209	0.99	1,325
32E5	400	3	3	390.22	0.98	3	2	0.184	0.98	2,121

solutions in any OCAT iteration. It should be stated here that only one phase was enough to derive an optimal clause.

The computational results are reported in Table IV. The same table also depicts the results derived by using the SAT approach. The SAT results were derived by using a VAX 8700 running 10th Edition UNIX. The code for the SAT tests was written in FORTRAN and in C. The SAT results were originally reported in [15].

The OCAT results were derived by using an IBM 3090–600S computer and the code was written in FORTRAN. As it can be seen from this table the OCAT approach outperformed the SAT approach in an order of many times. If we exclude the case of the test with ID 32C3, then for all the other cases OCAT was 101 to 31,362 times faster than the SAT approach. On the average, OCAT was 5,579 times faster.

Observe at this point that a direct comparison would have required the SAT results to be generated *either* in FORTRAN or in C (not both) and the OCAT approach on the same (single) language as the SAT approach. However, this is impractical and it seems that one has to compromise in comparing the two codes. Thus, the current results can convey only a flavor on the relative performance of the two methods, and by no means should be considered as a direct comparison of the two approaches.

In terms of the accuracy index, both the SAT and OCAT approaches performed considerably well. It should also be stated here that when the lower limit is equal to the number of clauses derived by OCAT, then we can conclude that OCAT derived a minimum size (in terms of the number of clauses) system. Note that this situation occurred in these experiments 51.2% of the time. However, if the lower limit is less than the number of clauses, then this *does not necessarily imply* that OCAT failed to derive a minimum size system.

Recall, that if optimality (i.e., the minimum number of inferred clauses) is not proven in the OCAT case, then the SAT approach can be applied with *successively decreasing k* values. When an infeasible SAT problem is reached, the conclusion is that the *last feasible* SAT solution yielded an *optimal system*. Finally, it is interesting to emphasize here that in these computational experiments it was found that, *most of the time*, OCAT derived a minimum size system. Therefore, it is anticipated that under the proposed strategy the SAT approach (which is very CPU time consuming) does not have to be used very often. Optimality (i.e., the minimum number of clauses) can be checked by comparing the number of clauses derived by OCAT with the lower limit established in Theorem 9.

Table V presents the results of solving some *large problems*. In these tests the number of atoms is 32, the total number of examples is equal to 1,000, and each "hidden logic" was assumed to have 30 clauses. The strategy followed in these

Table V. Solution statistics of some large test problems (number of atoms = 32)

Problem Characteristics					OCAT Solution Characteristics			
Problem ID	No. of Clauses	Total No. of Examples	$\|E^+\|$	$\|E^-\|$	CPU Time	No. of Clauses	Lower Limit	Accuracy
32H1	30	1,000	943	57	135.71	17	3	84.13%
32H2	30	1,000	820	180	45.18	10	3	93.83%
32H3	30	1,000	918	18	175.50	7	2	95.85%
32H4	30	1,000	944	56	64.16	20	2	82.84%
32H5	30	1,000	988	12	13.41	5	2	97.83%

Table VI. Descriptions of the systems in the first test problem (32H1)

The "Hidden Logic" System

CLAUSE: 1 = 2	6	11	14	16	21	25	27	−10	−13	−17	−18	−29			
CLAUSE: 2 = 15	17	19	26	31	−4	−8	−20								
CLAUSE: 3 = 24	25	−1	−10	−17	−18	−20	−26								
CLAUSE: 4 = 1	9	11	17	32	−10	−15	−16	−21	−22						
CLAUSE: 5 = 4	8	14	15	17	18	24	32	−11	−16	−21	−22	−28			
CLAUSE: 6 = 9	10	12	16	19	−4	−5	−8	−14	−25	−30	−31				
CLAUSE: 7 = 1	12	−8	−9	−11	−13	−15	−17	−26	−27	−28	−31				
CLAUSE: 8 = 1	12	17	26	−2	−5	−8	−20	−22	−24	−28	−31				
CLAUSE: 9 = 1	9	17	26	−2	−4	−6	−8								
CLAUSE: 10 = 5	7	19	−2	−12	−17	−18	−20	−23	−26	−28					
CLAUSE: 11 = 20	25	31	32	−2	−3	−8	−12	−13	−15	−22	−28				
CLAUSE: 12 = 13	25	−6	−19	−30	−32										
CLAUSE: 13 = 11	19	21	23	24	27	31	−4	−8	−9	−12	−13	−15	−26	−28	
CLAUSE: 14 = 5	8	20	21	23	−2	−9	−16								
CLAUSE: 15 = 3	4	13	16	−6	−10	−12	−26	−28	−29	−31					
CLAUSE: 16 = 4	17	19	24	26	32	−1	−5	−10	−13	−18	−21	−28			
CLAUSE: 17 = 14	25	28	−11	−12	−17	−22	−29								
CLAUSE: 18 = 2	3	7	11	23	24	27	31	32	−5	−8	−12	−14	−26	−29	
CLAUSE: 19 = 2	12	22	29	−8	−18	−27	−30	−31							
CLAUSE: 20 = 3	17	25	−4	−6	−24	−26	−30	]							
CLAUSE: 21 = 8	9	12	22	25	−3	−18	−24	−26	−29	]					
CLAUSE: 22 = 1	2	8	9	10	11	22	24	27	32	−3	−7	−14	−16	−29	−30
CLAUSE: 23 = 15	18	21	23	−2	−17										
CLAUSE: 24 = 10	13	16	17	18	−23	−26	−27								
CLAUSE: 25 = 3	6	16	19	20	21	25	−1	−12	−14	−15	−18	−23	−26	−32	
CLAUSE: 26 = 1	2	8	9	13	20	27	28	−5	−16	−23	−25	−26			
CLAUSE: 27 = 5	7	15	19	25	28	−6	−8	−10	−11						
CLAUSE: 28 = 1	8	9	11	15	20	21	22	29	−13	−24	−28	−31	−32		
CLAUSE: 29 = 1	5	9	15	25	29	−13	−26	−32							
CLAUSE: 30 = 17	18	22	23	30	32	−6	−8	−9	−10	−12	−24	−27	−29	−31	

The Inferred System

CLAUSE: 1 = 13	15	25	−6	−19	−30	−32					
CLAUSE: 2 = 15	18	20	21	23	25	−2	−32				
CLAUSE: 3 = 15	18	21	23	25	−2	−4	−17	−30			
CLAUSE: 4 = 5	14	15	21	25	−1	−2	−8	−10	−28	−32	
CLAUSE: 5 = 5	17	21	25	−2	−4	−6	−28	−30	−32		
CLAUSE: 6 = 18	25	29	−6	−8	−19	−30	−32				
CLAUSE: 7 = 15	18	21	23	−2	−9	−28					
CLAUSE: 8 = 1	9	17	−2	−4	−6	−8	−27				
CLAUSE: 9 = 2	5	11	15	17	29	−23	−26	−27	−28	−32	
CLAUSE: 10 = 3	11	12	17	22	25	29	−4	−8	−28		
CLAUSE: 11 = 3	12	15	17	29	−4	−8	−13	−18	−20	−32	
CLAUSE: 12 = 10	12	25	−14	−23	−24	−26	−28	−32			
CLAUSE: 13 = 3	7	11	14	15	17	18	25	28	−8	−20	−26
CLAUSE: 14 = 3	5	6	15	24	25	27	−4	13	−18		
CLAUSE: 15 = 10	16	21	23	31	−4	−8	−14	−20	−28		
CLAUSE: 16 = 7	16	30	−9	−13	−15	−17	−26	−28	−29		
CLAUSE: 17 = 12	25	−19									

experiments is the same as in the previous tests. The numbers of positive and negative examples are shown as well. Table VI presents the exact structure of the "hidden" and inferred systems of the first of these test problems (i.e., problem 32H1). In Table IV only the indexes of the atoms are depicted (in order to save space). For instance, the first clause of the inferred system is represented by the list [13, 15, 25, −6, −19, −30, −32] which implies the CNF clause:

$$(A_{13} \vee A_{15} \vee A_{25} \vee \bar{A}_6 \vee \bar{A}_{19} \vee \bar{A}_{30} \vee \bar{A}_{32}) .$$

Observe that now the CPU times are considerably (with the OCAT standards) higher. However, relatively speaking these times are still kept in low levels. The lower limit, however, is not tight enough. Furthermore, determining the maximum clique of the complemented rejectability graph took considerably more time than determining the infrared system. It should also be stated here that the "hidden" systems were not defined in terms of a minimum representation. That is, it might be possible to represent an equivalent system with less than 30 clauses. The OCAT approach always returned, compared to the original "hidden" system, a very compact system.

Finally, the accuracy of the inferred system was rather high. The size of the population of all possible examples is $2^{32} = 4.29496 \times 10^9$. Out of these examples, the tests considered only 1,000 random inputs. This represents a very small sample of the actual population and, therefore, the corresponding accuracy values can be considered rather high. The computational results in Tables II and V suggest that the OCAT approach, when it is combined with the new branch-and-bound algorithm, constitutes an efficient and effective strategy for inferring a logical system from examples.

7. Conclusions

The results of the computational experiments suggest that the proposed OCAT approach, when it is combined with the new branch-and-bound algorithm, provides a very efficient way for inferring logical clauses from positive and negative examples. It is interesting to observe that OCAT also derived systems for which very often it could be proved (by using the idea of the rejectability graph) to be of minimum size. Furthermore, the OCAT and the SAT approaches can be combined into a single strategy in order to efficiently derive a minimum size CNF and DNF system.

The high CPU time efficiency and effectiveness of the proposed method make it to be a practical method for inferring clauses from examples. Future work can focus on inferring *Horn clauses* (and not just general CNF or DNF systems as is the case currently). Another interesting expansion of this work is to apply these concepts on *partially defined* examples. That is, examples now are not defined in

the domain $\{0,1\}^r$, but instead in the domain $\{0,1,*\}^r$, where * indicates unknown value.

The problem of learning rules from past experience is the keystone in building truly intelligent systems. Furthermore, more efficient decomposition approaches are required in order to make learning feasible for large scale applications. More research in this area has the potential of making more contributions in this vital area of artificial intelligence and operations research.

Acknowledgements

The author would like to thank Professor P. M. Pardalos from the University of Florida for providing him with the clique code used in the computational experiments. He would also like to thank Dr. M. G. C. Resende from AT & T for providing him with the data used in deriving the computational results presented in Table IV.

References

1. Angluin, D. and C. H. Smith (1983), Inductive Inference: Theory and Methods, *Computing Surveys* **15**, 237–265.
2. Bongard, M. (1970), *Pattern Recognition*, Spartan Books, New York, NY.
3. Brayton, R. K., G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli (1985), *Logic Minimization Algorithms for VLSI Minimization*, Kluwer Academic Publishers, Dordrecht.
4. Blair, C. E., R. G. Jersolow, and J. K. Lowe (1985), Some Results and Experiments in Programming Techniques for Propositional Logic, *Computers and Operations Research* **13**, 633–645.
5. Carbonell, J. G., R. S. Michalski, and T. M. Mitchell (1983), An Overview of Machine Learning from Examples, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (eds.), *Machine Learning: An Artificial Intelligence Approach*, Tioga Publishing Company, Palo Alto, CA, 3–23.
6. Carraghan, R. and P. M. Pardalos (1990), An Exact Algorithm for the Maximum Clique Problem, *Operations Research Letters* **9**(11), 375–382.
7. Cavalier, T. M., P. M. Pardalos, and A. L. Soyster (1990), Modeling and Integer Programming Techniques Applied to Propositional Calculus, J. P. Ignizio (ed.), *Computers and Operations Research* **17**(6), 561–570.
8. Dietterich, T. C. and R. S. Michalski (1983), A Comparative Review of Selected Methods for Learning from Examples, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (eds.), *Machine Learning: An Artificial Intelligence Approach*, Tioga Publishing Company, Palo Alto, CA, 41–81.
9. Gimel, J. F. (1965), A Method of Producing a Boolen Function Having an Arbitrarily Prescribed Prime Implicant Table, *IEEE Trans. Computers* **14**, 485–488.
10. Hooker, J. N. (1988a), Generalized Resolution and Cutting Planes, R. G. Jeroslow (ed.), *Annals of Operations Research* **12**(1–4), 217–239.
11. Hooker, J. N. (1988b), A Quantitative Approach to Logical Inference, *Decision Support Systems*, North-Holland, 4, 45–69.
12. Jeroslow, R. G. (1988), Computation-Oriented Reductions of Predicate to Prepositional Logic, *Decision Support Systems*, North-Holland, 4, 183–197.
13. Jeroslow, R. G. (1989), *Logic-Based Decision Support*, North-Holland, Amsterdam.
14. Kamath, A. P., N. K. Karmakar, K. G. Ramakrishnan, and M. G. C. Resende (1990), Computational Experience with an Interior Point Algorithm on the Satisfiability Problem, *Annals*

of Operations Research, P. M. Pardalos and J. B. Rosen (eds.), Special issue on: Computational Methods in Global Optimization, **25**, 43–58.
15. Kamath, A. P., N. K. Karmakar, K. G. Ramakrishnan, and M. G. C. Resende (1992), A Continuous Approach to Inductive Inference, *Math. Programming* **57**(2), 215–238.
16. Karmakar, N. K., M. G. C. Resende, and K. G. Ramakrishnan (1991), An Interior Point Algorithm to Solve Computationally Difficult Set Covering Problems, *Math. Programming* **52**(3), 597–618.
17. Kearns, M., Ming Li, L. Pitt, and L. G. Valiant (1987), On the Learnability of Boolean Formulae, *Journal of the Association for Computing Machinery* **34**(9), 285–295.
18. Peysakh, J. (1987), A Fast Algorithm to Convert Boolean Expressions into CNF, *IBM Computer Science RC 12913 (#57971)*, Watson, NY.
19. Pitt, L. and L. G. Valiant (1988), Computational Limitations on Learning from Examples, *Journal of the Association for Computing Machinery* **35**(4), 965–984.
20. Quinlan, J. R. (1979), 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.
21. Quinlan, J. R. (1983), Learning Efficient Classification Procedures and Their Application to Chess and Games, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (eds.), *Machine Learning: An Artificial Intelligence Approach*, Tioga Publishing Company, Palo Alto, CA, 3–23.
22. Quinlan, J. R. (1986), Induction of Decision Trees, *Machine Learning* **1**(1), 81–106.
23. Triantaphyllou, E., A. L. Soyster and S. R. T. Kumara (1994), Generating Logical Expressions from Positive and Negative Examples via a Branch-and-Bound Approach, *Computers and Operations Research* **21**(2), 185–197.
24. Triantaphyllou, E. and A. L. Soyster (1994a), A Relationship between CNF and DNF Systems Derivable from Examples, *ORSA Journal on Computing*, to appear.
25. Triantaphyllou, E. and A. L. Soyster (1994b), On the Minimum Number of Logical Clauses which Can be Inferred from Examples, *Working Paper*, Department of Industrial and Manufacturing Systems Engineering, Louisiana State University, Baton Rouge, LA 70803-6409, U.S.A.
26. Triantaphyllou, E. and A. L. Soyster, (1994c), An Approach to Guided Learning of Boolean Functions From Examples, *Working Paper*, Department of Industrial and Manufacturing Systems Engineering, Louisana State University, Baton Rouge, LA 70803-6409, U.S.A.
27. Valiant, L. G. (1984), A Theory of the Learnable, *Comm. of ACM* **27**(11), 1134–1142.
28. Valiant, L. G. (1985), Learning Disjunctions of Conjunctives, *Proceedings of the 9th IJCAI*, 560–566.
29. Williams, H. P. (1986), Linear and Integer Programming Applied to Artificial Intelligence, *Preprint series*, University of Southampton, Faculty of Mathematical Studies, 1–33.
30. Williams, H. P. (1987), Linear and Integer Programming Applied to the Propositional Calculus, *International Journal of Systems Research and Information Science* **2**, 81–100.