Pergamon

0305-0548(95)00075-5

Computers Ops Res. Vol. 23, No. 8, pp. 783–799, 1996
Copyright © 1996 Elsevier Science Ltd
Printed in Great Britain. All rights reserved
0305-0548/96 $15.00+0.00

ON THE MINIMUM NUMBER OF LOGICAL CLAUSES INFERRED FROM EXAMPLES

Evangelos Triantaphyllou[1,†] and Allen L. Soyster[2,‡]

[1]Department of Industrial and Manufacturing Systems Engineering, Louisiana State University, 3128 CEBA Building, Baton Rouge, LA 70803-6409, U.S.A. [2]Department of Industrial and Management Systems Engineering, Penn State University, 207 Hammond Building, University Park, PA 16802, U.S.A.

(Received February 1994; in revised form October 1995)

Scope and Purpose—One of the critical challenges in learning a set of rules (logical clauses) is to derive a very small number of rules, which still satisfy the pertinent requirements. These requirements are derived from positive (successes) and negative examples (failures). The present paper uses some graph theoretic approaches in establishing ways for partitioning large learning problems and also determining bounds on the minimum number of rules derivable from given sets of positive and negative examples.

Abstract—Given two sets of positive and negative examples, the inductive inference problem is to infer a small set of logical clauses which appropriately classify the examples. A graph theoretic approach is used to establish a lower limit on the minimum number of required clauses. Furthermore, the findings of this paper reveal methods for partitioning the original data and thus solving efficiently large scale problems. Copyright © 1996 Elsevier Science Ltd

1. INTRODUCTION

The capability of a system to learn from experience has long been accepted as the main requirement for building a truly intelligent system. Although there are many types of learning, learning from examples is the most widely examined type of learning. Complexity issues of this type of learning have been studied by Valiant [1,2], Kearns *et al.* [3], and Pitt and Valiant [4]. A number of algorithms which implement learning from examples can be found in Carbonell *et al.* [5], Dietterich and Michalski [6], Quinlan [7–9], Triantaphyllou *et al.* [10,11], Kamath *et al.* [12], and Utgoff [13].

In the type of learning considered in this paper examples are classified either as positive or negative. Then, the issue is to determine a Boolean expression which classifies all the positive and negative examples correctly. Usually, such a Boolean function is expressed in the conjunctive normal form (CNF) or in the disjunctive normal form (DNF). See, for instance, Blair *et al.* [14], Cavalier *et al.* [15], Hooker [16,17], Jeroslow [18,19], Kamath *et al.* [12,20], and Williams [21,22]. Peysakh in [23] describes an algorithm for converting any Boolean expression into CNF.

The objective of this paper is to determine a lower bound on the number of the CNF (or DNF) clauses which are needed to correctly classify a given set of examples. In addition, efficient approaches are derived which can effectively partition the input data into smaller groups before processing by a learning algorithm. In this way, large learning problems can be solved more efficiently.

†E. Triantaphyllou received a Dual Ph.D. in Industrial Engineering and Operations Research from the Pennsylvania State University in 1990. He also has an M.S. degree in Computer Science and a Dual M.S. degree in Environment and Operations Research from the same university. Currently, he is an Assistant Professor at Louisiana State University. His research interests include interfaces of artificial intelligence and operations research, mathematical programming, multi-criteria decision making, fuzzy sets, and machine learning. He is the author of more than 20 papers in the above areas which appeared in *Decision Support Systems, Fuzzy Sets and Systems, Optimization Theory and Applications, ORSA Journal on Computing, Computers and OR, OR Letters, Global Optimization, IEEE Transactions on Fuzzy Systems, and Approximate Reasoning,* among others.

‡A.L. Soyster is a Professor and Head of Industrial and Management Systems Engineering at the Pennsylvania State University. He received his Ph.D. from Carnegie Mellon University, his M.S. from Cornell University and his B.S. from the Pennsylvania State University. His research expertise is in optimization of energy distribution systems and planning methods, modeling economic energy processes, and transportation; as well as artificial intelligence and expert systems application to industrial engineering. He is a member of IIE, ORSA, and TIMS. Dr Soyster also served as Editor-in-Chief of *IIE Transactions*, an international journal sponsored by IIE. He was awarded with the IIE Fellow Award in May 1988.

The main idea of this paper is described in the following illustration. Suppose that a series of examples (binary vectors) is somehow known. These examples were classified as either *positive* examples or *negative* examples. Such a situation is common. For instance, applicants for admission to a particular college could be characterized by high school performance, college placement exams, etc. and are either admitted (positive examples) or rejected (negative examples). This paper considers the case in which the examples are vectors with binary components (0 or 1). For this illustration, suppose that the following positive (denoted as E^+) and negative (denoted as E^-) examples are known:

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

(The 4 rows in E^+ represent 4 positive examples and the 6 rows in E^- represent 6 negative examples.) What we seek is a set of Boolean clauses which correctly classify *all the examples*. One set of Boolean clauses (in so-called conjunctive normal form) is as follows:

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

If the vector (0,1,0,0) (i.e. the first positive example) is interpreted as (F,T,F,F), (i.e. $F=0$ and $T=1$), then observe that this vector, as well as each positive example, *satisfies* all three Boolean clauses. Furthermore, each negative example is *not satisfied* by at least one of the three clauses. Note that these three clauses correctly classify all ten examples. The main goal of this paper is to establish a *lower bound* on the number of clauses needed to correctly classify any set of positive and negative examples. One might inquire, for instance, whether it is possible that only two clauses are required to correctly classify all the examples in this illustration. If it were known that a lower bound was three, then, in terms of relative simplicity, these three clauses constitute an optimal set. These ideas and notation are formalized in the next section.

2. SOME DEFINITIONS AND TERMINOLOGY

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 F be a *Boolean function* over these atoms. For instance, the expression $(A_1 \vee A_2) \wedge (A_3 \vee \bar{A}_4)$ is such a Boolean function, where "$\vee$" and "$\wedge$" stand for the logical *"OR"* and *"AND"* operators, respectively. That is, 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 F, whether F is true or false (denoted as 1 and 0, respectively).

For each Boolean function F, the *positive examples* are the vectors $v \in \{0,1,\}^t$ such that $F(v)=1$. Similarly, the *negative examples* are the vectors $v \in \{0,1,\}^t$ such that $F(v)=0$. Therefore, given a function 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 F if and only if the vector v is a positive (or a negative) example of F. For instance, let F be the Boolean function $(A_1 \vee A_2) \wedge (A_3 \vee \bar{A}_4)$. Consider the two vectors $v_1=(1,0,0,0)$ and $v_2=(1,0,0,1)$. Then, it can be easily verified that $F(v_1)=1$. That is, the vector v_1 is a positive example of the function F. However, the vector v_2 is a negative example of F [since $F(v_2)=0$].

At this point some additional definitions are also introduced. Let $v \in \{0,1,\}^t$ be an example (either positive or negative). Then, $\bar{v} \in \{0,1\}^t$ is defined as the *complement of the example v*. Similarly, let E be a collection of examples. Then, $\bar{E}$ is defined as the *complement of the collection E*. A Boolean expression is in CNF or DNF if it is in the form (I) or (II), respectively:

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

and

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

where a_i is either A_i or $\bar{A}_i$ and ρ_j is the set of indexes.

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

The following theorem proved by Triantaphyllou and Soyster in [24] states an important property which exists when CNF and DNF systems are inferred from collections of positive and negative examples.

Theorem 1 (Triantaphyllou and Soyster [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).

This theorem is stated here because the graph theoretic developments throughout this paper assume that a system is derived in CNF form. However, since a clause inference algorithm which derives DNF expressions (such as, for instance, the SAT approach described in Kamath *et al.* [12]) can also derive CNF expressions (by applying the previous theorem), the findings of this paper are applicable both to CNF and DNF cases.

In summary, a set of positive examples is denoted as E^+ and a set of negative examples is denoted as E^-. Given these two sets of positive and negative examples, 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.

3. THE REJECTABILITY GRAPH OF TWO COLLECTIONS OF EXAMPLES

This section presents the motivation and definition of a special graph which can be easily derived from positive and negative examples. 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.* For the positive example v_1, note that A_1, $\bar{A}_2$, A_3, $\bar{A}_4$, and A_5 are true (or, equivalently, $\bar{A}_1$, A_2, $\bar{A}_3$, A_4 and $\bar{A}_5$ are false). Similar interpretations exist for the remaining two examples v_2 and v_3.

Denote by ATOMS(v) the set of the atoms that are true for a particular (either positive or negative) example v. With this definition, one obtains:

$$\text{ATOMS}(v_1) = \text{ATOMS}((1,0,1,0,1)) = \{A_1, \bar{A}_2, A_3, \bar{A}_4, A_5\}$$

$$\text{ATOMS}(v_2) = \text{ATOMS}((1,0,1,1,1)) = \{A_1, \bar{A}_2, A_3, A_4, A_5\}$$

$$\text{ATOMS}(v_3) = \text{ATOMS}((1,1,1,0,1)) = \{A_1, A_2, A_3, \bar{A}_4, A_5\}.$$

Now consider a single CNF clause (i.e. a disjunction), denoted as C, of the form: $\bigvee_{i=1}^{M} a_i$ (where a_i is either A_i or $\bar{A}_i$). Then, the clause C *accepts* an example v (i.e. v is a positive example of C) if and only if at least one of the atoms in the set ATOMS(v) is also one of the atoms in the expression $\bigvee_{i=1}^{M} a_i$. Otherwise, the example v is *not accepted* (i.e. v is a negative example of C). For instance, if the clause C is defined as: $C=(\bar{A}_2 \vee A_4)$, then the examples v_1 and v_2 are accepted by C, while the example v_3 is *not* accepted.

Now observe that there is no ***single CNF clause*** which can *simultaneously* reject 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 $\{A_1, A_2, \bar{A}_2, A_3, A_4, \bar{A}_4, A_5\}$ = ATOMS(v_2) ∪ 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.

From the above considerations it follows that given three examples v_1, v_2, and v_3, then the examples v_2 and v_3 are rejectable by a single clause (disjunction), subject to the example v_1, if and only if the following condition is true:

$$\text{ATOMS}(v_1) \not\subseteq \text{ATOMS}(v_2) \cup \text{ATOMS}(v_3).$$

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 2:

Let E^+ be a set of positive examples and v_1, v_2 be two negative examples. There exists a CNF clause which accepts all the positive examples and rejects both negative examples v_1 and v_2 if and only if:
ATOMS(v_i) $\not\subseteq$ ATOMS(v_1) ∪ ATOMS(v_2), 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 2 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 cardinality of E^- (i.e. each vertex corresponds to one negative example in E^-), and

$E = \{(V_i, V_j)$ 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* (or *the R-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 $\frac{M_2(M_2-1)}{2}$. Therefore, the rejectability graph can be constructed by performing $\frac{M_2(M_2-1)}{2}$ simple rejectability examinations.

An illustrative example

Consider the following E^+ and E^- sets:

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

Since there are 6 negative examples, there are $6 \times 5/2 = 15$ possible pairwise comparisons (i.e. single rejectability tests). For instance, the first (v_1) and third (v_3) negative examples correspond to the vertices V_1 and V_3, respectively. Since: ATOMS(v_1) ∪ ATOMS(v_3) = $\{A_1, A_2, A_3, A_4, \bar{A}_2, \bar{A}_4\} \not\supseteq$ ATOMS(v_i), for each $v_i \in E^+$, it follows that there is an edge which connects the vertices V_1 and V_3 in the rejectability graph. The rejectability graph G, which corresponds to this illustrative example, is presented in Fig. 1. □

4. PROPERTIES OF THE REJECTABILITY GRAPH

The rejectability graph G of a set of positive and a set of negative examples possesses a number of interesting properties. Two of these properties refer to the cliques of the rejectability graph. A *clique*

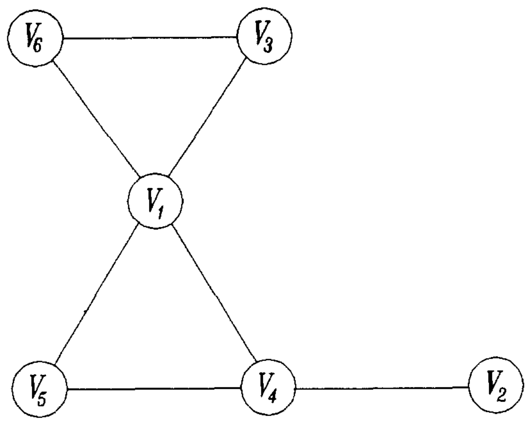

Fig. 1. The rejectability graph of E^+ and E^-.

of a graph is a subgraph in which all the nodes are connected with each other. The *minimum clique cover number* [denoted as $k(G)$] is the fewest number of cliques needed to cover the vertices of G (see, for instance, Golumbic [25]). The following theorem refers to any clique of the rejectability graph.

Theorem 3:

Suppose that the two sets E^+ and E^- are given and $\mathscr{F}$ 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.

Proof:

Consider any two examples v_1 and v_2 in the subset $\mathscr{F}$. Since all the members in $\mathscr{F}$ can be rejected by a single clause, obviously the examples v_1 and v_2 can be rejected by this single clause. From the definition of the rejectability graph G, it follows that there is an edge connecting the corresponding two nodes in G. Clearly this situation is true for any pair of examples in the subset $\mathscr{F}$. Therefore, the vertices which correspond to the k negative examples in $\mathscr{F}$ form a clique in G 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. To show this consider the following illustrative example.

An illustrative example.

Consider the following sets E^+ and E^-:

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

It can be easily verified that any pair of the three negative examples in E^- can be rejected by a single clause which also accepts the positive example in E^+. For instance, the first and second negative examples are rejected by the clause (A_3), which also accepts the positive example in E^+. Similarly, the first and third negative examples can be rejected by (A_2), while (A_1) rejects the second and third examples. In all cases, these clauses accept the single example in E^+. Therefore, the corresponding rejectability graph is a triangle (i.e. a clique with three nodes, see also Fig. 2). However, a clause which would reject all the three negative examples should *not include* any atoms from the following set:

$$\begin{aligned} &\text{ATOMS}(v_1) \cup \text{ATOMS}(v_2) \cup \text{ATOMS}(v_3) \\ &= \text{ATOMS}((1,0,0)) \cup \text{ATOMS}((0,1,0)) \cup \text{ATOMS}((0,0,1)). \\ &= \{A_1, A_2, A_3, \bar{A}_1, \bar{A}_2, \bar{A}_3\} \end{aligned}$$

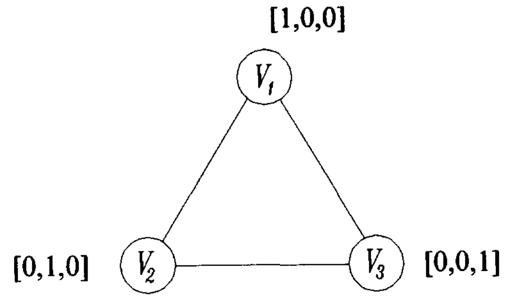

Fig. 2. The rejectability graph for the second illustrative example.

Obviously, no such clause exists. Therefore, a minimum size set of CNF clauses which satisfy the requirements of the current examples is: $(A_3) \vee (A_2)$, which is of size 2.□

5. ON THE MINIMUM CLIQUE COVER OF THE REJECTABILITY GRAPH

Consider two sets of positive and negative examples E^+ and E^-, respectively. Let $\bar{G}$ be the *complement* of the rejectability graph G of the two sets of examples. Recall that the complement of a graph is constructed as follows: the complement graph has exactly the same vertices as the original graph. There is an edge between any two vertices if and only if there is no edge between the corresponding vertices of the original graph. Next, define $\omega(\bar{G})$ as the *size of the maximum clique* of the graph $\bar{G}$ and $k(G)$ as the minimum clique cover number of the rejectability graph G. Let r be the minimum number of CNF clauses required to reject all the examples in E^-, while accepting all the examples in E^+. Then, the following theorem states a *lower bound* [i.e. the minimum clique cover $k(G)$] on the *minimum number* of clauses required to reject all the negative examples in E^-, while accepting all the positive examples in E^+.

Theorem 4:

Suppose that E^+ and E^- are the sets of the positive and negative examples, respectively. Then, the following relation is true: $r \geq k(G) \geq \omega(\bar{G})$.

Proof:

Denote as $\{C_1, C_2, C_3, \ldots, C_r\}$ a minimum set of r clauses which reject all the examples in E^-. By Theorem 3, the negative examples rejected by clause C_i $(1 \leq i \leq r)$, form a clique in the graph G and, by construction, each negative example is associated with at least one of these cliques. Hence, the cliques generated by the set $\{C_1, C_2, C_3, \ldots, C_r\}$ form a clique cover for G. Therefore, $r \geq k(G)$. It is well known (see, for instance, Golumbic [25]) that $k(G) \geq \omega()$. Hence, $r \geq k(G) \geq \omega(\bar{G})$. □

At this point it should be stated that according to this theorem the gap between r and $k(G)$ can be positive. The same is also true with the gap between $k(G)$ and $\omega(\bar{G})$. Therefore, there is a potential for the gap between r and $\omega(\bar{G})$ to be large (since the value of $\omega(\bar{G})$ can be arbitrarily large, see for instance [26]). Results from some related computational experiments seem to indicate that when the value of $\omega(\bar{G})$ is large, then the bound is rather tight (see also Tables 3 and 4). More on this is discussed later in Section 8.

Although finding $k(G)$ is NP-complete, the determination of $\omega(\bar{G})$ is also NP-complete, but there are more efficient enumerative algorithms. In Carraghan and Pardalos [27] a survey of algorithms which can find the maximum clique in any graph is presented. They also present a very efficient algorithm which uses a partial enumeration approach which outperforms any other known algorithm. In that treatment random problems with 3000 vertices and over one million edges were solved in rather short times (less than 1 h on an IBM ES/3090-600S computer). Some other related developments regarding the maximum clique of a graph can be found in [28–33].

6. PROBLEM DECOMPOSITION

The rejectability graph provides a framework for decomposing the determination of a lower bound for the number of clauses into a set of smaller problems. The decomposition is obtained through a partitioning of the rejectability graph. We consider two processes:

- Decomposition via Connected Components, and
- Decomposition via the Construction of a Clique Cover.

Connected components

In this case, one inspects the rejectability graph for a *natural* decomposition. A *connected component* of a graph is a maximal subgraph in which there is a path of edges between any pair of vertices. Hence, the vertices of the connected components are mutually exclusive and their union is exhaustive. The following corollary is derived from Theorem 3 and illustrates the relation of the connected components of G and the clauses which can be inferred from two collections of positive and negative examples.

Corollary 3.1:

Suppose that E^+ and E^- are the sets of the positive and negative examples, respectively. Then, any subset of negative examples in E^- which is rejected by a single CNF clause, subject to the examples in E^+, corresponds to a subset of vertices of the rejectability graph G which belong to the *same connected component* of the graph G.

Proof:

This is obvious since Theorem 3 implies that the above subset of vertices forms a clique in the graph G and a clique has to belong to the same connected component of G. □

Pardalos and Rentala in [34] present an excellent survey of algorithms which determine the connected components of a graph. Furthermore, they also propose a *parallel algorithm* which runs on an IBM ES/3090-400E computer (with four processors). That algorithm determines the connected components in super linear time.

The importance of Corollary 3.1 emerges when the sets of positive and negative examples are very large. First, one constructs the rejectability graph G. Next, one determines all the connected components of the rejectability graph by applying an algorithm (such as the one described in Pardalos and Rentala [34]) for finding the connected components. Then, one solves the smaller clause inference problems which are formed by considering *all the positive examples* and the *negative examples which correspond* to the vertices of the individual and distinct connected components in G.

In other words, if a graph has two or more connected components, then one can decompose the original problem into separate problems and *the aggregation of the optimal solutions (minimum number of CNF clauses) of the separate problems is an optimal solution for the original problem.* Observe that each such sub-problem (in the CNF case) is comprised of the negative examples for that component and *all* the positive examples, i.e. the positive examples are identical for each sub-problem.

Clique cover

The second approach is also motivated by partitioning the vertices of the rejectability graph into mutually disjoint sets. However, in this second approach, vertices are subdivided via a sequential construction of cliques.

First, the maximum clique of the rejectability graph is determined. The negative examples which correspond to the vertices of the maximum clique, along with *all* the positive examples, form the first sub-problem of this decomposition. Next, the maximum clique of the *remaining* graph is derived. The second sub-problem is formed by the negative examples which correspond to the vertices of the second clique and all the positive examples. This process continues until all the negative examples (or, equivalently, all the vertices in the rejectability graph) are considered.

We note that this sequence of cliques does not necessarily correspond to a minimum clique cover of the rejectability graph. This procedure is simply a *greedy* approach which *approximates* a minimum clique cover. Furthermore, it is possible that a single sub-problem (in which all the vertices in the rejectability graph form a clique) may yield *more than one* clause.

It should be noted at this point that the clique cover derived by using the above greedy approach may not always yield a minimum clique cover. Therefore, the number of cliques derived in that way, *cannot* be used as a lower bound on the number of clauses derivable from positive and negative examples. Obviously, if the number of cliques is equal to $\omega(\bar{G})$, then the previous clique cover is minimal. However, even if the previous clique cover is not of minimum size, it can still be very useful as it can lead to a decomposition of the original problem into a sequence of smaller problems. Some computational results described in Section 8, provide some insight into the effectiveness of such a decomposition approach.

The two problem decomposition approaches described in this section can be combined into one approach as follows. One first decomposes the original problem in terms of its connected components. Next, a clique cover, as described above, is derived for the individual problems which correspond to the connected components of the rejectability graph. This approach is further illustrated in the example presented in the following section.

7. AN EXAMPLE OF USING THE REJECTABILITY GRAPH

Consider the following sets of positive and negative examples:

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

One method to actually construct a set of clauses which accept all the positive examples while rejecting the negative ones is proposed in Triantaphyllou *et al.* [10] and Triantaphyllou [11]. That approach, denoted as the *One Clause At a Time (or OCAT)* method, is a greedy approach and is briefly described in the appendix. An application of OCAT in this illustrative example yields the following CNF system of *four* clauses:

$$(A_2 \vee A_3 \vee \bar{A}_5) \wedge (\bar{A}_1 \vee A_3 \vee \bar{A}_5) \wedge (A_1 \vee A_2 \vee A_4 \vee A_5) \wedge (\bar{A}_1 \vee A_2 \vee \bar{A}_3 \vee \bar{A}_4),$$

of course the *question* addressed in this paper is whether it is possible to derive another system with *fewer* clauses.

To help answer the previous question, we apply Theorem 4 to this illustrative example. Since there are 13 positive and 7 negative examples, the construction of the rejectability graph requires 21 simple rejectability examinations. When Theorem 2 is applied to this data, the rejectability graph shown in Fig. 3 is derived. For instance, there is an edge between vertices V_1 and V_6 because the first and sixth negative examples can be rejected by a single disjunction without violating the constraints imposed by the positive examples in E^+. A similar interpretation holds for the remaining edges in graph G.

The rejectability graph in the current illustrative example has *two connected components* (see Fig. 3). One component is comprised by the vertices V_1, V_2, V_4, V_5, V_6, V_7 and the second component has only the vertex V_3. Therefore, the original problem can be partitioned into **two independent** clause inference sub-problems.

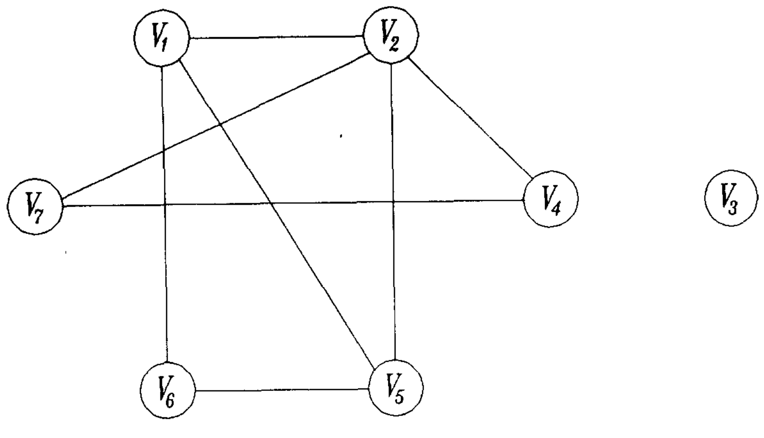

Fig. 3. The rejectability graph for E^+ and E^-.

Both sub-problems have the same positive examples. The first sub-problem has the same negative examples as in E^- *except* for the *third* negative example. The second problem has *only* the third negative example. The lower bound for the minimum number of CNF clauses required to appropriately classify the 20 examples is derived from the sum of the lower bounds for the two separate components. Since the rejectability graph of the second sub-problem contains only a single vertex, the size of the minimum clique cover is one. A minimum clique cover is also obvious for the first sub-problem, namely, the two sets $\{V_1, V_5, V_6\}$ and $\{V_2, V_4, V_7\}$. Hence, a minimum clique cover is two for the second sub-problem. Thus, an overall lower bound for the minimum number of CNF clauses required is three. Hence, it may well be possible that only three clauses are needed to appropriately classify all 20 examples.

There is another clause inference approach which can be used to determine a minimum size set of clauses. This method, denoted as SAT (for satisfiability), has been proposed in Kamath *et al.* [12] (also briefly described in the Appendix). In this approach one first specifies an upper limit on the number of clauses to be considered, say k. That is, the value of k must be *pre-assumed.* Next a clause satisfiability (SAT) model is formed and solved using an interior point method developed by Karmakar and his associates [35]. If the clause satisfiability problem is satisfied, it is possible to correctly classify all the examples with k or fewer clauses. If this SAT problem is infeasible, then one must increase k until feasibility is reached. In this manner, the SAT approach yields a system with the minimum number of clauses. It is very important one to observe at this point that computationally it is much harder to prove that a given SAT problem is infeasible than it is feasible. Therefore, trying to determine a minimum size Boolean function by using the SAT approach may be computationally too difficult. In this illustrative example, the SAT approach with $k=3$, is feasible and returns the following 3 clauses:

$$(A_1 \lor A_2 \lor A_3) \land (\bar{A}_1 \lor A_2 \lor \bar{A}_3 \lor \bar{A}_4) \land (\bar{A}_1 \lor A_3 \lor \bar{A}_5).$$

However, when the value $k=2$ is used, then the corresponding SAT formulation is infeasible. Therefore, this set of clauses is optimal in the sense of this paper. The last statement also follows from Theorem 4 since there exists a clique cover of 3 and a set of clauses has been derived with exactly this number of members.

8. SOME COMPUTATIONAL RESULTS

In this section we provide some computational insight into two issues. The first is the role and usefulness of the lower bound of Theorem 4. The second issue is on the potential benefit of using the decomposition approach which is based on a clique cover. The first issue is important when one is interested in minimizing the size of the inferred Boolean function. The lower limit described in Theorem 4 (i.e. the value of $\omega(\bar{G})$) can also give an idea of how far from optimality a given solution might be. The second issue is important when one wishes to control the CPU time requirement (for instance, when solving large problems). Although the clique decomposition approach results in

Table 1. Description of the systems used as *"hidden logic"* in the computer experiments.

System ID	System Description	System ID	System Description
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	$(A_3 \vee A_{12} \vee A_{15}) \wedge (\bar{A}_3 \vee \bar{A}_{11}) \wedge (\bar{A}_2 \vee \bar{A}_{10} \vee \bar{A}_{16}) \wedge (A_1 \vee A_2)$	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	$(A_4 \vee \bar{A}_7 \vee A_{11}) \wedge (A_4 \vee A_{10} \vee A_{14}) \wedge (\bar{A}_9 \vee \bar{A}_{14} \vee A_{15}) \wedge (\bar{A}_3 \vee A_8)$		

solving a sequence of smaller Boolean inference problems, there is a new time burden because one now also needs to determine a sequence of maximum cliques. Therefore, it is not immediately obvious that the decomposition approach will be less demanding in CPU time. A second issue is the size of the systems derived via the clique decomposition. The systems derived via the decomposition approach may be larger in size than otherwise. For these reasons, three series of test problems were performed.

The **first series of test problems** is based on the 15 systems given by Kamath *et al.* [12]. These systems use sets of clauses defined on 8, 16, and 32 atoms and are depicted in Table 1. The 5 systems with 8 atoms are labelled as 8A1, 8A2, 8A3, 8A4, and 8A5 in Table 2 (the secondary classification is based on the number of examples used as input). The same convention was also used for the rest of the systems in Table 1. Each of the 15 sets are used as a "*hidden logic*" for classifying randomly generated examples, i.e. the sets E^+ and E^-. Then, the OCAT algorithm is used to generate a set of CNF clauses which correctly classify each of the positive and negative examples.

As described earlier, the rejectability graph and the sequential (greedy) generation of maximum cliques are determined. Next, Theorem 4 is used to establish a lower bound on the required number of inferred clauses. Note that the SAT results were determined by using a VAX 8700 machine running the 10th edition of UNIX, written in FORTRAN (the SAT results were originally reported in Kamath et al. [12] and were not repeated during this investigation). The OCAT results were derived using an IBM/3090-600S machine (which is approximately 3 to 4 times faster than a VAX 8700 machine) and the code was written in FORTRAN.

The results of these experiments are provided in Table 2. These results include computations from SAT, OCAT and the lower bound, as determined according to Theorem 4. Please note that in these tests we did not decompose the problems by using the connected component approach described in Section 6 (no code was available to us at the time). The SAT results represent a *feasible* solution, not necessarily a solution with the minimum number of clauses. (To obtain the minimum number of clauses, one must iteratively reduce the value of k until infeasibility occurs, which is a very time consuming process.) Consider, for instance, the first case (problem 8A1) depicted in Table 2. Ten random examples were generated for system 8A (as defined in Table 1). With k fixed at 3 the SAT algorithm returned a feasible solution with 3 clauses. The lower bound from Theorem 4 is equal to 2.

Table 2. Solution statistics

Problem Characteristics			SAT solution		OCAT solution	ω(G bar): lower bound via Theorem 4
Problem ID	No. of examples	Clauses in "hidden logic"	k value	Number of clauses	Number of clauses	
8A1	10	3	3	3	3	2
8A2	25	3	6	3	3	2
8A3	50	3	6	3	3	3
8A4	100	3	6	3	3	3
8B1	50	3	3	2	2	2
8B2	100	3	6	3	3	3
8B3	150	3	10	3	3	3
8B4	200	3	6	3	3	3
8C1	50	3	10	2	2	2
8C2	100	3	10	3	3	3
8D1	50	3	10	3	3	3
8D2	100	3	10	3	3	3
8E1	50	3	10	3	3	3
8E2	100	3	10	3	3	3
16A1	100	4	15	4	4	3
16A2	300	4	6	4	4	4
16B1	200	4	8	5	4	4
16B2	400	4	4	4	4	4
16C1	100	4	20	5	4	4
16C2	400	4	4	4	4	4
16D1	200	4	10	4	4	4
16D2	400	4	4	4	4	4
16E1	200	4	15	5	5	4
16E2	400	4	4	4	4	4
32A1	250	3	3	3	3	3
32B1	50	3	3	3	2	1
32B2	100	3	3	3	3	2
32B3	250	3	3	3	3	2
32B4	300	3	3	3	3	2
32C1	50	3	3	3	2	1
32C2	100	3	3	3	2	1
32C3	150	3	3	3	3	1
32C4	1000	3	3	3	3	3
32D1	50	4	4	4	3	1
32D2	100	4	4	4	3	2
32D3	400	4	4	4	4	2
32E1	50	3	3	2	2	1
32E2	100	3	3	3	2	1
32E3	200	3	3	3	3	2
32E4	300	3	3	3	3	2
32E5	400	3	3	3	3	2

Hence, it may be possible to correctly classify the 10 examples with only 2 clauses. However, in these tests the value of k was not iteratively reduced to determine its minimum value.

The results of the first set of tests indicate just how well OCAT performs. Of the 24 problems with 16 or fewer atoms, OCAT generated a set of clauses *exactly* at the lower bound in 20 cases. In the other 4 cases, OCAT exceeded the lower bound by only 1 clause. For the 17 problems with 32 atoms, OCAT averaged about 1.23 more clauses than the lower bound.

It is also noteworthy to observe that the performance of SAT, OCAT and the Theorem 4 lower bound are not dramatically affected by the number of examples. As expected, as the number of examples increases, the number of required clauses also increases. This is illustrated, for instance, by example 32E. The OCAT approach generated 2 clauses with 50 and 100 examples while 3 clauses were needed for 200, 300 and 400 examples. As more examples are generated, the set of inferred

Table 3. Solution statistics when $N = 10$ and the total number of examples is 400

Problem				With out decomposition				With clique decomposition					
ID	M_1	M_2	K_0	K_1	Time$_1$	A_1	K_2	Limit $\omega(\bar{G})$	No. of cliques	Time$_2$	Time$_3$	Time$_4$	A_2
1A10	247	153	10	10	4.96	0.97	10	10	10	0.03	0.04	0.07	0.96
2A10	258	142	10	9	2.09	0.98	9	9	9	0,03	0.03	0.06	0.98
3A10	198	202	10	10	3.34	0.98	10	9	9	0.03	0.07	0.10	0.98
4A10	253	147	10	10	2.54	0.96	10	10	10	0.03	0.05	0.08	0.97
5A10	288	112	10	10	5.06	0.97	11	10	11	0.04	0.03	0.07	0.97
Mean:	**248.8**	**151.21**	**10.0**	**9.8**	**3.60**	**0.97**	**10.0**	**9.6**	**9.8**	**0.03**	**0.04**	**0.08**	**0.97**
1A20	193	207	20	14	14.00	0.96	18	14	15	0.05	0.25	0.30	0.95
2A20	266	134	20	18	55.22	0.94	22	18	19	0.08	0.04	0.12	0.95
3A20	329	71	20	12	44.7	60.95	16	12	14	0.07	0.01	0.08	0.94
4A20	291	109	20	16	57.51	0.95	20	15	17	0.08	0.02	0.10	0.93
5A20	133	267	20	13	4.29	0.97	16	13	14	0.03	200.60	200.60	0.97
Mean:	**242.4**	**157.6**	**20.0**	**14.6**	**35.16**	**0.95**	**18.4**	**14.4**	**15.8**	**0.06**	**40.18**	**40.24**	**0.95**
1A30	315	85	30	17	103.76	0.93	18	16	17	0.08	0.01	0.09	0.94
2A30	326	74	30	21	376.06	0.93	24	20	22	0.11	0.01	0.12	0.93
3A30	306	94	30	18	64.55	0.93	17	17	17	0.07	0.02	0.09	0.93
4A30	287	113	30	17	38.29	0.93	21	16	18	0.09	0.03	0.12	0.92
5A30	293	107	30	22	267.25	0.92	25	19	21	0.10	0.03	0.13	0.92
Mean:	**305.4**	**94.6**	**30.0**	**19.0**	**169.98**	**0.93**	**21.0**	**17.6**	**19.0**	**0.09**	**0.02**	**0.11**	**0.93**
1A40	317	83	40	26	316.21	0.89	24	20	22	0.11	0.01	0.12	0.90
2A40	252	148	40	22	159.09	0.90	24	21	22	0.08	0.05	0.13	0.89
3A40	286	114	40	20	129.13	0.91	26	19	21	0.10	0.04	0.14	0.92
4A40	328	72	40	19	220.66	0.92	20	17	18	0.09	0.01	0.10	0.93
5A40	271	129	40	22	139.31	0.90	27	20	24	0.11	0.06	0.17	0.90
Mean:	**290.8**	**109.2**	**40.0**	**21.8**	**192.87**	**0.90**	**24.2**	**19.4**	**21.4**	**0.10**	**0.03**	**0.13**	**0.91**
1A50	256	144	50	27	260.39	0.86	30	23	27	0.11	0.07	0.18	0.87
2A50	265	135	50	25	159.03	0.88	28	21	24	0.11	0.07	0.18	0.87
3A50	270	130	50	23	128.13	0.86	27	20	22	0.10	0.05	0.15	0.86
4A50	285	115	50	26	356.04	0.88	26	21	25	0.10	0.04	0.14	0.88
5A50	296	104	50	21	112.31	0.86	26	20	22	0.10	0.02	0.12	0.88
Mean:	**274.4**	**125.6**	**50.0**	**24.4**	**203.18**	**0.87**	**27.4**	**21.0**	**24.0**	**0.10**	**0.05**	**0.15**	**0.87**

clauses becomes a better approximation to the underlying *"hidden logic"*, which for example 32E is comprised of 3 clauses.

The **second and third series** of computational experiments were executed as follows. First, a random Boolean function with K_0 clauses (in CNF form) was determined. In order to have a good balance of positive and negative examples, each atom in any clause was present with probability 15 to 25%. Also, if atom A_i was selected to be in a clause, then atom $\bar{A}_i$ was not allowed to be present and vice-versa. That was done in order to avoid constructing clauses which would accept all examples (i.e. to avoid tautologies). A system derived in this way, was considered as the *"hidden logic"* system in these experiments. The number of atoms was set to be equal to 10 and 30 (i.e. $N = 10$ or 30). These systems are indexed by ID numbers, such as 1A10, 2A10,..., 1B10, 2B10 etc. and are available to interested readers from the first author of this paper. In the above coding scheme the first digit indicates the test number, "A" or "B" indicates whether, N, the number of atoms was equal to 10 or to 30, respectively, and the last two digits indicate the number of rules in the *"hidden logic"* (see also Table 3 and Table 4).

Next, a collection of 400 or 600 examples (400 examples were considered when $N = 10$ and 600 examples when $N = 30$) was randomly generated and classified according to the previous clauses. The computational results are presented in Table 3 and Table 4 (for $N = 10$ and $N = 30$, respectively). At first, the branch-and-bound (B & B) algorithm described in Triantaphyllou [11] was applied to the original problem consisted by M_1 positive and M_2 negative examples. The CPU time (in seconds on an IBM 3090-600E computer) and the number of clauses derived this way are presented in columns *"Time$_I$"* and *"K_I"*.

Since in these experiments the *"hidden logic"* is also known to us, the original system can be compared with the derived system by asking both systems to classify 10,000 randomly generated examples (this technique was also used in [12]). The corresponding accuracy rates are presented in column *"A_I"*. The next phase in these experiments was to apply the clique decomposition approach. The original problem with the M_1 and M_2 examples was decomposed into a number of smaller problems according to the clique cover approach described at the end of Section 7. Note that the

Table 4. Solution statistics when $N = 30$ and the total number of examples is 600

Problem				With out decomposition			With clique decomposition						
ID	M_1	M_2	K_0	K_1	Time$_1$	A_1	K_2	Limit $\omega(\bar{G})$	No. of cliques	Time$_2$	Time$_3$	Time$_4$	A_2
1B10	520	80	10	8	16.02	0.97	14	3	4	117.21	5.22	122.43	0.93
2B10	485	115	10	9	20.68	0.96	19	3	5	41.38	17,960.00	18,001.00	0.89
3B10	563	37	10	7	28.07	0.96	10	2	3	35.11	0.01	35.12	0.94
4B10	554	46	10	6	5.15	0.99	10	3	3	3.70	0.12	3.82	0.97
5B10	589	11	10	4	10.97	0.98	5	2	2	6.24	0.00	6.24	0.98
6B10	570	30	10	5	9.64	0.98	7	2	3	1.59	0.00	1.59	0.95
7B10	546	54	10	6	78.10	0.97	9	2	3	2.46	0.02	2.48	0.95
8B10	563	37	10	6	39.84	0.97	9	2	3	17.84	0.02	17.86	0.95
9B10	569	31	10	6	38.61	0.97	7	2	3	45.97	0.00	45.97	0.96
10B10	515	85	10	8	57.23	0.98	20	3	4	202.88	259.06	461.93	0.94
Mean:	**547.4**	**52.6**	**10.0**	**6.5**	**30.43**	**0.97**	**11.0**	**2.4**	**3.3**	**47.44**	**1822.45**	**1869.89**	**0.95**
1B30	586	14	30	5	22.95	0.97	6	2	2	10.44	0.00	10.44	0.97
2B30	583	17	30	4	1.76	0.99	5	3	3	0.21	0.00	0.21	0.98
3B30	533	67	30	10	383.66	0.96	17	3	4	546.33	5.89	552.22	0.92
4B30	574	26	30	5	66.23	0.97	7	3	3	1.67	0.00	1.67	0.96
5B30	559	41	30	10	1380.84	0.93	13	3	4	97.77	0.07	97.85	0.93
6B30	499	101	30	9	538.02	0.93	14	3	5	55.77	2375.00	2431.00	0.93
7B30	513	87	30	9	41.06	0.95	17	3	4	62.71	3.90	66.61	0.91
8B30	537	63	30	12	1501.81	0.91	17	3	3	903.19	0.47	903.66	0.91
9B30	553	47	30	11	1791.30	0.89	14	3	3	417.47	0.09	417.56	0.92
10B30	551	49	30	11	10,714.65	0.91	16	3	3	834.61	0.10	834.71	0.91
Mean:	**548.8**	**51.2**	**30.0**	**8.6**	**1644.23**	**0.94**	**12.6**	**2.8**	**3.4**	**293.01**	**238.55**	**531.57**	**0.93**
1B50	586	14	50	4	1.57	0.97	4	2	2	0.46	0.01	0.47	0.98
2B50	584	16	50	5	64.47	0.96	7	2	2	20.99	0.01	21.00	0.96
3B50	575	25	50	5	13.49	0.97	8	2	3	10.45	0.01	10.46	0.95
4B50	550	50	50	6	28.34	0.97	7	2	3	7.65	0.02	7.67	0.97
5B50	503	97	50	13	1465.44	0.89	17	3	4	277.57	281.24	558.81	0.86
6B50	512	88	50	13	2181.36	0.91	21	3	4	804.24	550.02	1354.26	0.86
7B50	489	111	50	19	7415.04	0.82	27	3	4	2438.29	4410.85	6849.14	0.83
8B50	501	99	50	15	5135.15	0.89	24	3	5	411.08	286.33	697.41	0.87
9B50	496	104	50	18	11,735.38	0.88	27	3	5	2422.45	1970.34	4392.79	0.80
10B50	517	83	50	15	7987.36	0.87	23	3	5	2594.78	125.63	2720.41	0.82
Mean:	**531.3**	**68.7**	**50.0**	**11.3**	**3602.76**	**0.91**	**16.5**	**2.6**	**3.7**	**898.80**	**762.44**	**1661.24**	**0.89**

natural decomposition via the connected component approach was not used in these tests. We only used the decomposition approach imposed by the sequence of the maximum cliques (as described in Section 6 and Section 7.

The number of cliques which was generated for this purpose is depicted under column *"No. of Cliques"*. The value of $\omega(\bar{G})$ is depicted under column *"Limit"*. The values under *"Time$_2$"* and *"Time$_3$"* present the CPU time required by the B & B algorithm (only) and the calculation of the cliques, respectively. The values under *"Time$_4$"* (total CPU time when clique decomposition is used) are the sum of the values in columns *"Time$_2$"* and *"Time$_3$"*. The number of clauses of the proposed systems are under column *"K_2"*. Finally, the values under *"A_2"* are the accuracy rates when the system proposed by the decomposition approach and the original system are compared.

The computational experiments in Table 3 were performed for groups of random *"hidden logics"* with 10, 20, 30, 40 and 50 clauses. These results indicate that the limit provided by Theorem 4 (i.e. the values of $\omega(\bar{G})$ in column *"Limit"*) is rather tight. For instance, when $K_0 = 40$ then the average number of clauses derived by using B & B without decomposition was equal to 21.8 vs 19.4 being the lower limit. That is, the B & B approach returned systems of very small or even probably of minimal size.

A *"hidden logic"* was generated randomly with a predetermined number of CNF clauses, say K_0. Suppose that two collections of positive and negative examples are generated such that all positive examples are accepted by the K_0 clauses, while each negative example is rejected by at least one of the previous clauses. If r denotes the minimum number of CNF clauses which satisfy the requirements of the previous collections of positive and negative examples, then from the previous consideration the following relation follows to be true: $K_0 \geq r$.

When one examines the sizes of the systems returned when the clique decomposition approach was used, it can be observed that most of the time the decomposition approach returned systems with at most 10% (on the average) more CNF clauses than without decomposition. However, this was done with a fraction of the CPU time when compared with no decomposition.

For instance, when $K_0 = 40$ then, then on the average, the decomposition approach returned systems with 24.2 clauses (vs systems with 21.8 clauses without decomposition) by consuming, on the average, 0.13 CPU seconds vs 192.87 CPU seconds without the clique decomposition. That is, in the above test problems one obtains a system with at most 10% more clauses, but in return, realizes a return of an almost 1500 times speedup on CPU time.

The results in Table 4 are similar to those in Table 3. However, the lower bound (i.e. the value of $\omega(\bar{G})$) described in Theorem 4 is considerably less tight. This is seen by the size of the gap between the values in column *"K_1"* (or *"K_2"*) and column *"Limit"*. Also, now the CPU times are significantly much higher, since we are dealing with larger and more difficult problems. At the same time, the CPU times are more unpredictable.

For instance, observe that when $K_0 = 10$, then problem 2B10 took 17,960.00 CPU seconds for computing the required cliques. This time is obviously excessive when compared with the times in the rest of the problems in these experiments. The CPU times became more variable when the *"hidden logics"* had more clauses (which naturally resulted in harder problems).

However, even now the results suggest that the proposed clique decomposition approach may *significantly reduce* the CPU requirements in solving large problems with only a moderate increase in the number of derived clauses. Finally, it is remarkable to observe that in terms of the accuracy rates the systems derived by using the decomposition approach are almost equally accurate as the systems derived without the clique decomposition (which are computed at higher CPU time requirements).

From the above analyses and computational results it becomes evident that the rejectability graph provides at least two benefits:

(i) When the value of $\omega(\bar{G})$ is rather high, then the value of $\omega(\bar{G})$ can serve as a tight lower limit of the minimum number of clauses derivable from positive and negative examples. Of course, the clique cover can still be used for decomposing a large inductive inference problem.

(ii) When the value of $\omega(\bar{G})$ is low (and hence there be a large gap between $\omega(\bar{G})$ and r), then the rejectability graph can still be useful in decomposing a large inductive inference problem because it may lead to a significant reduction of the CPU time.

These decompositions are based on constructing a sequence of cliques. This operation depends on the algorithm used to determine the maximum clique in a graph. Present algorithms are rather efficient when the graph is scarce (as is the case with the rejectability graphs in the problems described in Table 3). In our computational experiments we used the clique algorithm described in [27]. This algorithm is considered to be very good for scarce graphs and it is often used as a benchmark in the literature.

However, when the graphs become dense, then this clique algorithm becomes too slow. Very often (around 20–30% of the time) we had to abort tests running for the results in Table 4, because the clique construction phase of our program would take too long (more than 5 h on an IBM 3090-600E mainframe computer). We believe that other current algorithms may be more efficient for dense graphs (such as the clique algorithm described in [36] or the algorithms recently developed by Balas and his associates [32], [33]). One issue became profoundly apparent in this investigation: Future developments in determining a maximum clique in a graph, will directly benefit the efficient solution of large inference problems by employing the rejectability graph and the decomposition approaches described in this paper.

9. CONCLUDING REMARKS

The paramount importance of learning from examples, creates the demand of being able to process large collections of positive and negative examples. It also increases the pressure on creating Boolean expressions which have a small number of clauses. The rejectability graph, which was introduced and discussed in this paper, provides the means for establishing a lower bound on the number of CNF or DNF clauses which can be inferred from positive and negative examples.

This graph also provides an effective way for partitioning the original data and, thus, solve large scale learning problems. Furthermore, the rejectability graph suggests a time efficient approach for decomposing the original problem into a sequence of smaller problems and still infer a compact Boolean expression from the partial solutions of the smaller problems.

The previous findings were discussed in terms of two learning algorithms. The first, is a greedy approach and is based on a branch-and-bound algorithm developed by Triantaphyllou *et al.* [10] (for a new and much faster version of the branch-and-bound algorithm developed by see Triantaphyllou [11]). The second approach is based on formulating a satisfiability problem (Kamath *et al.* [12] and then solving it by using an interior point method (Karmakar *et al.* [35]). Finally, it is possible that the rejectability graph to have more interesting properties than the ones described in this paper. More research in this direction may reveal more connections between graph theory and the learning from examples problem.

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**, 1134–1142 (1984).
2. L. G. Valiant, Learning disjunctions of conjunctives, *Proc. of the 9th IJCAI*, pp. 560–566 (1985).
3. M. Kearns, Ming Li, L. Pitt and L.G. Valiant, On the learnability of Boolean formulae. *J. Assoc. Computng Mach.*, **34**, 285–295 (1987).
4. L. Pitt and L. G. Valiant, Computational limitations on learning from examples. *J. Assoc. Computng Mach.* **35**, 965–984 (1988).
5. J. G. Carbonell, R. S. Michalski and T. M. Mitchell, An overview of machine learning from examples. In R. S. Michalski, J. G. Carbonell and T.M. Mitchell (Eds), *Machine Learning: An Artificial Intelligence Approach*, pp.3–23. Tioga Publishing Company, Palo Alto, CA (1983).
6. T. C. Dietterich and R. S. Michalski, A comparative review of selected methods for learning from examples. In R. S. Michalski, J. G. Carbonell and T. M. Mitchell (Eds), *Machine Learning: An Artificial Intelligence Approach*, pp. 41–81. Tioga Publishing Company, Palo Alto, CA (1983).
7. J. R. Quinlan, Discovering rules by induction from large numbers of examples: a case study. In D. Michie (Ed.) *Expert Systems in the Micro-Electronic Age*. Edinburgh University Press (1979).
8. J. R. Quinlan, Learning efficient classification procedures and their application to chess and games. In R.S. Michalski, J.G. Carbonell and T.M. Mitchell (Eds), *Machine Learning: An Artificial Intelligence Approach*, pp. 3–23. Tioga Publishing Company, Palo Alto, CA, (1983).
9. J. R. Quinlan, Induction of decision trees, *Mach. Learn.* **1**, 81–106 (1986).
10. E. Triantaphyllou, A. L. Soyster and S. R. T. Kumara, Generating logical expressions from positive and negative examples via a branch-and-bound approach. *Comput. Ops Res.* **21**, 185–197 (1994).
11. E. Triantaphyllou, Inference of a minimum size Boolean function from examples by using a new efficient branch-and-bound approach, *J. Glob. Optimiz.* **5**, 69–94 (1994).
12. A. P. Kamath, N. K. Karmakar, K. G. Ramakrishnan and M. G. C. Resende, A continuous approach to inductive inference, *Math. Progm.* **57**, 215–238 (1992).
13. P. E. Utgoff, ID5: an incremental ID3, John Laird (Ed.). *Proc. 5th Int. Conf. on Machine Learning*, pp.107–120. University of Michigan, Ann Arbor (1988).
14. C. E. Blair, R. G. Jeroslow and J. K. Lowe, Some results and experiments in programming techniques for propositional logic. *Comput. Ops Res.* **13**, 633–645 (1985).
15. T. M. Cavalier, P. M. Pardalos and A. L. Soyster, Modeling and integer programming techniques applied to propositional calculus, J.P. Ignizio (Ed.). *Comput. Ops Res.* **17**, 561–570 (1990).
16. J. N. Hooker, Generalized resolution and cutting planes, R.G. Jeroslow (Ed.). *Ann. Ops Res.* **12**, 217–239 (1988).
17. J. N. Hooker, A quantitative approach to logical inference, *Dec. Supp. Syst.* **4**, 45–69 (1988).
18. R. G. Jeroslow, Computation-oriented reductions of predicate to prepositional Logic. *Dec. Supp. Syst.* **4**, 183–197 (1988).
19. R. G. Jeroslow, R. G., *Logic-Based Decision Support*. North-Holland, Amsterdam (1989).
20. 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*, P.M. Pardalos and J.B. Rosen (eds.). Special issue on: *Computational Methods in Global Optimization*, **25**, 43–58 (1990).
21. H. P. Williams, Linear and integer programming applied to artificial intelligence, (*Preprint Series*), pp.1–33. Faculty of Mathematical Studies, University of Southampton (1986).
22. H. P. Williams, Linear and integer programming applied to the propositional calculus. *Int. J. Syst. Res. Inf. Sci.* **2**, 81–100 (1987).
23. J. Peysakh, A fast algorithm to convert Boolean expressions into CNF. IBM Computer Science RC 12913 (No. 57971), Watson, NY (1987).
24. E. Triantaphyllou and A. L. Soyster, A relationship between CNF and DNF systems which are derived from the same positive and negative examples. *ORSA J. Comput.* **7**, 283–285 (1995).
25. M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York (1980).
26. B. Bollobás, *Graph Theory, An Introductory Course*. Springer, Berlin (1979).
27. R. Carraghan and P. M. Pardalos, An exact algorithm for the maximum clique problem, *Ops Res. Lett.* **9**, 375–382 (1990).
28. P. M. Pardalos and J. Xue, The Maximum Clique Problem. *J. Glob. Optimiz.* **4**, 301–328 (1994).
29. L. Babel and G. Tinhofer, A branch and bound algorithm for the maximum clique problem. *Meth. Models Ops Res.* **34**, 207–217 (1990).

30. L. Babel, Finding maximum cliques in arbitrary and in special graphs. *Computing* **46**, 321–341 (1991).
31. L. Babel, A fast algorithm for the maximum weight clique problem, *Computing*, to appear (1996).
32. E. Balas and J. Xue, Weighted and unweighted maximum clique algorithms with upper bounds from fractional coloring. Management Science Research Report No. MSRR-590, Carnegie Mellon University, Pittsburgh (1993).
33. E. Balas and W. Niehaus, Finding large cliques by bipartite matching. Management Science Research Report No. MSRR-597, Carnegie Mellon University, Pittsburgh (1994).
34. P. M. Pardalos and C. S. Rentala, Computational aspects of a parallel algorithm to find the connected components of a graph. Technical Report, Department of Computer Science, Pennsylvania State University (1990).
35. N. K. Karmakar, M. G. C. Resende and K. G. Ramakrishnan, An interior point algorithm to solve computationally difficult set covering problems, *Math. Progm.* **52**, 597–618 (1991).
36. P. M. Pardalos and G. P. Rogers, A branch and bound algorithm for the Maximum Clique Problem. *Comput. Ops Res.* **19**, 363–375 (1992).

Readers may address inquiries to Dr Triantaphyllou at: ietrian@isuvm.sncc.isu.edu

APPENDIX

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 Triantaphyllou *et al.* [10]. The second algorithm infers DNF clauses and is based on formulating a *clause satisfiability (SAT) problem* Kamath *et al.* [12] and then solving this SAT by using an interior point method described in Karmakar *et al.* [35]. These approaches are also used in the illustrative example presented in Section 6.

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 attempts to minimize the 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. 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). By applying Theorem 1, either a CNF or DNF system can be derived by using *any algorithm.*

The One Clause At a Time Approach

In Triantaphyllou *et al.* [10] an algorithm which infers CNF systems from positive and negative examples is developed. In that approach, CNF clauses are generated in a way which attempts to *minimize* the number of CNF clauses that constitute the recommended CNF system. In this way, a compact CNF system can be derived. The strategy followed there is called the *One Clause At a Time (or OCAT)* approach.

The OCAT approach is *greedy* in nature. It uses as input data two collections of positive and negative examples (denoted as E^+ and E^-, respectively). It determines a set of CNF clauses which, when taken together, reject all the negative examples and 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 clause (i.e. the first) so far. The iterations continue until a set of clauses is constructed which reject all the negative examples in the original E^- set. More on this approach can be found in Triantaphyllou *et al.* [10] and Triantaphyllou [11]. Figure 4 summarizes the iterative nature of the OCAT approach.

The core of the OCAT approach is Step 2, in Figure 4. In Triantaphyllou *et al.* [10] a branch-and-bound based algorithm is presented which solves the problem posed in Step 2 efficiently. A more efficient branch-and-bound algorithm, along with other enhancements, is described in Triantaphyllou [11]. The OCAT approach returns the set of desired clauses (i.e. the CNF system) as set C.

```
i = 0 ; C = $\phi$; { initializations }
DO WHILE  ($E^- \neq \phi$)
    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. 4. The One Clause At a Time (OCAT) approach.

Clause Inference as a Satisfiability Problem

Let M_1 and M_2 be the numbers of examples in the E^+ and E^- sets, respectively. In Kamath *et al.* [12] it is shown that given two collections of positive and negative examples (called the ON-set and OFF-set, respectively) 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 developed in Karmakar *et al.* [35] 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 (Kamath *et al.* [12]):

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

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

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

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

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

$$s_{ji} \bigvee s'_{ji}, \text{ for } i = 1, \ldots, n, \text{ and } j = 1, \ldots, k, \tag{1}$$

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

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

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

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 on this formulation can be found in Kamath *et al.* [12].