**ISTITUTO DI ANALISI DEI SISTEMI ED INFORMATICA**
**CONSIGLIO NAZIONALE DELLE RICERCHE**

**K. Truemper**

# EFFICIENT ALGORITHMS FOR SUBCLASSES OF THE FUTILE QUESTIONING PROBLEM

**R. 479 Novembre 1998**

**Klaus Truemper** – University of Texas at Dallas Computer Science Program, Box 830688, Richardson, Texas 75083-0688, U.S.A, e-mail: truemper@utdallas.edu.

This research was supported in part by the Office of Naval Research under Grant N00014-93-1-0096, and was done while the author was visiting IASI in October 1998.

Istituto di Analisi dei Sistemi ed Informatica, CNR
viale Manzoni 30
00185 ROMA, Italy

tel. ++39-06-77161
fax ++39-06-7716461
email: iasi@iasi.rm.cnr.it
URL: http://www.iasi.rm.cnr.it

## Abstract

Define the following problem to be the futile questioning problem of propositional logic. The input consists of two propositional logic formulas $S$ and $T$. Some variables of $S$ are declared to be question variables. One either must assign *True/False* values to the question variables of $S$ such that $T$ becomes a theorem of the modified $S$, or must conclude that no such assignment of *True/False* values is possible. The futile questioning problem as well as a generalization involving logic minimization are readily shown to be difficult in general. We describe efficient solution algorithms for large subclasses of these problems and discuss example applications involving expert systems.

## 1. Introduction

Suppose an expert system whose knowledge base is encoded by a propositional logic formula $S$ does not have sufficient data to prove a certain conclusion encoded by a propositional logic formula $T$. The expert system may decide to query the user for additional information, and may or may not succeed in establishing the conclusion $T$ once that information has been obtained. One can predict the latter, negative, outcome if the conclusion cannot be proved regardless how the user answers the queries. In that case, we say that questioning is *futile*. We call the task of detecting futile questioning the *futile questioning problem*. Specifically, that problem demands that either one declares questioning to be futile or one computes answers to the questions that would allow the conclusion $T$ to be proved.

The questions are represented by some of the variables of $S$. Thus, the answers are always *True/False* values for these *question variables*. We only consider *True/False* answers that a user may reasonably supply. We assume that such answers are characterized by the condition that they be part of some satisfying solution for $S$. We call such answers *$S$-feasible*.

In a slight abuse of language, we say in the case of futile questioning that *proving $T$ is futile* or, shorter, that *$T$ is futile*. With that terminology, we can rephrase the futile questioning problem. Given formula $S$, specified question variables, and conclusion $T$, one must decide if $T$ is futile. If $T$ is not futile, one must determine $S$-feasible answers for the questions for which $T$ can be proved to be a theorem of $S$.

In this paper, we develop methods for solving the futile questioning problem when $S$ is a formula in conjunctive normal form (CNF) and $T$ is a CNF clause. Thus, $S$ is a conjunction of clauses, each of which is a disjunction of possibly negated propositional variables, and $T$ is one disjunction of such variables. That case is of significant interest since the knowledge bases and conclusions of many expert systems can be compactly encoded by CNF formulas and CNF clauses, respectively. Furthermore, the case of general $S$ and $T$ can be reduced in linear time to one with CNF formula $S'$ and CNF clause $T'$, as follows. First, using the well-known technique relying on additional variables, one converts $S$ (resp. $\neg T$) in linear time to an equivalent CNF formula $G$ (resp. $H$). Second, one defines a new variable $t$ and adds it to each clause of $H$. Let $H'$ be the CNF formula so obtained from $H$. Finally, one defines $S' = GH'$ and $T' = t$ and declares the question variables of $S$ to be the question variables of $S'$. Validity of the conversion is easily verified.

In principle, each instance of the futile questioning problem considered here may be handled by enumerating all possible cases of $S$-feasible answers and solving for each case an instance of the satisfiability problem SAT of propositional logic. For this reason, we call the futile questioning problem considered here FUTILE SAT.

A generalization of FUTILE SAT involves that data of FUTILE SAT plus two rational costs for each nonquestion variable. One of the costs applies if the variable takes on the value *True*, while the other cost applies for the case of *False*. One must compute $S$-feasible answers for the question variables that prove $T$ to be not futile or that, if this case does not apply, force the total cost of a satisfying solution of $S\neg T$ that minimizes that total, to be as large as possible. In principle, the problem may be handled by enumerating all possible cases of $S$-feasible answers and solving for each case an instance of the logic minimization problem MINSAT of propositional logic, where one must prove unsatisfiability or provide a satisfying solution whose total cost, as determined by the costs associated with the *True/False* values, is minimum.

Real-world applications of FUTILE SAT and FUTILE MINSAT abound. For example, if futile questioning can be detected in a diagnostic expert system, then fruitless lines of questioning can be cut short. We discuss this case and others in detail in Section 7.

We show that FUTILE SAT and FUTILE MINSAT are difficult. Stockmeyer [1976] as well as Wrathall [1976] contain results for the polynomial-time hierarchy that directly prove the following problem to be $\Sigma_2^p$ complete. Instance: A propositional logic formula $g(V, W)$ where $V$ and $W$ are disjoint sets of variables. Question: Is it true that there exist *True/False* values for the variables of $V$ such that, for all *True/False* values of the variables of $W$, the function $g(V, W)$ has value *False*?

An instance $g(V, W)$ can be reduced to an instance of FUTILE SAT as follows. First, using a new variable $t$, one defines $S = g(V, W) \vee t$, lets $V$ be the set of question variables, and declares $T = t$. Second, one uses the earlier described method to convert this instance of the futile questioning problem to one of FUTILE SAT, say, with CNF formula $S'$ and CNF clause $T'$. It is easy to check that "Is $T'$ not futile?" and the question posed by the instance $g(V, W)$ of the original problem have the same answer.

The converse reduction is just as easy. Let $S$ and $T$ with set $V$ of question variables and set $W$ of nonquestion variables be given. We write $S$ and $T$ as $S(V, W)$ and $T(V, W)$, respectively, to emphasize the variables. Then the desired function is $g(V \cup Z, W) = [\neg S(V, Z)] \vee [S(V, W)\neg T(V, W)]$. Validity of the reduction is readily confirmed.

We conclude that answering "Is T not futile?" is a $\Sigma_2^p$ complete problem, and that the more general logic minimization version is at least as hard. Accordingly, we are justified to confine ourselves to subclasses of FUTILE SAT and FUTILE MINSAT. The subclasses considered here are motivated by real-world applications. The methods proposed for the subclasses are computationally effective and thus practically useful. The subclasses are defined via two decompositions of the given CNF formula $S$.

The first decomposition assumes that, roughly speaking, the number of ways in which $S$-feasible answers for the question variables can influence the values of the remaining variables is bounded. We call that decomposition *exact-linear* since it is a special case of a logic decomposition called *linear* in Truemper [1998] and since it involves certain satisfiability subproblems where some clauses must not be satisfied.

The second decomposition is in some sense the opposite of a logic decomposition called *monotone* in the cited reference. For that reason, the decomposition considered here is called *antimonotone*.

The two decompositions described in this paper are compatible with, and thus may be combined with, logic decompositions described in Truemper [1998]. As a result, the large polynomially solvable classes established in that reference for the satisfiability and logic minimization problems are readily modified to produce still-large polynomially solvable classes for FUTILE SAT and FUTILE MINSAT.

The paper proceeds as follows. Section 2 describes prior work. Section 3 introduces definitions and notation. Section 4 defines the exact linear decomposition and provides a solution algorithm based on that decomposition. Section 5 introduces the antimonotone decomposition and describes a related solution algorithm. Section 6 covers an extension of the antimonotone decomposition. Section 7 discusses several example applications. The final section, 8, contains the references.

## 2. Prior Work

FUTILE SAT is defined in Straach and Truemper [1998a, 1998b]. These references use the following simple sufficient test for showing futility of $T$. Delete from $S$ all question variables, and test the reduced CNF formula with $\neg T$ enforced for satisfiability. If that formula is satisfiable, then $T$ is futile.

Binary decision diagrams (BDDs) are powerful tools for solving difficult questions of propositional logic. The seminal paper is Bryant [1986], where a graph encoding of propositional logic is developed and proved to be optimal in a certain sense; see also Bryant [1992]. Subsequently, numerous refinements and extensions were developed; see, for example, the survey paper by Bryant [1995]. BDDs can solve FUTILE SAT since they can handle quantification of propositional variables. The process works well if $S$ is small. It is not clear how effective BDDs would be for large instances—in particular, for the instances of the subclasses considered in this paper, where size is unbounded.

## 3. Definitions and Notation

The methods of the subsequent sections utilize a number of results of Truemper [1998] that are most easily described using the operations and notation of that reference. In this section, we review those operations and the related notation.

Let $S$ be a CNF formula, say with variables $v_1$, $v_2, \ldots,$ $v_n$. An example is the formula composed of the clauses

$$\begin{aligned} & v_1 \vee v_3 \\ & \neg v_1 \vee v_2 \vee \neg v_3 \\ & \neg v_1 \vee \neg v_2 \end{aligned} \tag{3.1}$$

A well-known matrix representation of $S$, say by a matrix $B$, assigns to each variable (resp. clause) of $S$ one row (resp. column) of $B$. The entry of $B$ in column $j$ and row $i$ is 1 if the variable $v_j$ occurs in clause $i$, it is $-1$ if $v_j$ occurs negated in clause $i$, and it is 0 otherwise. Here we assume that both $v_j$ and $\neg v_j$ do not occur in a clause since any such clause would be trivially satisfied and could be eliminated.

As an example, the matrix $B$ for (3.1) is

$$B = \begin{array}{c} \begin{array}{ccc} v_1 & v_2 & v_3 \end{array} \\ \begin{pmatrix} 1 & 0 & 1 \\ -1 & 1 & -1 \\ -1 & -1 & 0 \end{pmatrix} \end{array} \tag{3.2}$$

For our purposes, the entries 0, 1 and $-1$ of $B$ have no numeric interpretation.

We utilize three binary operations $\odot$, $\oplus$, and $\ominus$ called IB-*multiplication*, IB-*addition*, and IB-*subtraction*. They are defined as follows.

For $\alpha, \beta \in \{0, \pm 1\}$, IB-multiplication is defined by

$$\alpha \odot \beta = \begin{cases} 1 & \text{if } \alpha = \beta = 1 \text{ or } \alpha = \beta = -1 \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

IB-addition and IB-subtraction are defined only for $\{0, 1\}$ elements. For $\alpha, \beta \in \{0, 1\}$, IB-addition is given by

$$\alpha \oplus \beta = \begin{cases} 1 & \text{if } \alpha = 1 \text{ or } \beta = 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

For $\alpha, \beta \in \{0, 1\}$, IB-subtraction is specified by

$$\alpha \ominus \beta = \begin{cases} 1 & \text{if } \alpha = 1 \text{ and } \beta = 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

IB-multiplication may be interpreted as an extension of the Boolean "if and only if" when a $+1$ of $\alpha$ or $\beta$ is interpreted as *True*, if a $-1$ is interpreted as *False*, and if a resulting 1 is viewed as *True* and a resulting 0 as *False*.

IB-addition may be interpreted as the inclusive "or" if a 1 is viewed as *True* and a 0 as *False*. Furthermore, IB-subtraction may be viewed as a sort of inverse operation of the inclusive "or."

The above operations are extended to matrices analogously to the case of integer matrix multiplication, addition, and subtraction. In particular, for $\{0, \pm 1\}$ matrices $A$ and $B$ of appropriate size, $C = A \odot B$ is defined to be the $\{0, 1\}$ matrix whose elements $C_{ij}$ are given by $C_{ij} = \bigoplus_k (A_{ik} \odot B_{kj})$.

Both G. W. Leibniz (1646–1716) and G. Boole (1815–1864) made use of a subtraction operation as inverse of the exclusive "or." For details, see Newman (1956) and Kneale and Kneale (1984). That subtraction operation does not lead to many interesting results for logic equations. Presumably for that reason, the subtraction operation was later abandoned. However, in connection with inequalities rather than equations, the subtraction operation defined here as a sort of inverse of the inclusive "or" is useful. Indeed, if one defines $1 > 0$, one can establish several interesting inequalities involving $\{0, 1\}$ vectors $a$, $b$, $c$, and $d$ and the operations $\oplus$ and $\ominus$; see Truemper (1998). In this paper, we make use of

$$a \oplus b \geq c \text{ if and only if } a \geq c \ominus b \tag{3.6}$$

$$a \geq b \text{ and } c \geq d \text{ imply } a \oplus c \geq b \oplus d \text{ and } a \ominus d \geq b \ominus c \tag{3.7}$$

Satisfiability of $S$ translates to existence of a solution for a certain inequality system involving $B$. That is, $S$ is satisfiable if and only if there exists a $\{\pm 1\}$ vector $s$ such that $B \odot s \geq \underline{1}$, where $\underline{1}$ is a vector of appropriate size containing only 1s. A value of 1 (resp. $-1$) for the element $s_j$ of a solution vector $s$ corresponds to assigning the value *True* (resp. *False*) to the related variable of $S$.

Since each entry of the vector $B \odot s$ is 0 or 1, requiring $B \odot s \geq \underline{1}$ is the same as demanding $B \odot s = \underline{1}$. That fact no longer holds when one replaces the right-hand-side vector $\underline{1}$ by an arbitrary $\{0, 1\}$ vector, as we shall do shortly.

One could allow $s$ of a solution for $B \odot s \geq \underline{1}$ to be a $\{0, \pm 1\}$ vector instead of a $\{\pm 1\}$ vector. The value 0 for $s_j$ would mean that the related variable of $S$ could be deleted while satisfiability of $S$ was maintained. Unless stated otherwise, we do not allow for that case and thus assume generally that $s$ is a $\{\pm 1\}$ vector.

There are several variations of the satisfiability problem that are readily handled by our notation. If satisfiability of a subset of the clauses of $S$ is to be checked, one replaces in the right-hand-side vector $\underline{1}$ of $B \odot s \geq \underline{1}$ all 1s corresponding to clauses that need not be satisfied by 0s, thus getting for some $\{0, 1\}$ vector $b$ an inequality system $B \odot s \geq b$. Instead of that inequality, one may also consider the equation $B \odot s = b$. It depicts that the clauses $i$ with $b_i = 1$ (resp. $b_i = 0$) must (resp. must not) to be satisfied. It is easily checked that solving $B \odot s = b$ is the same as solving $B \odot s \geq b$ with the added restriction that, for any row $i$ and any column $j$ with $B_{ij} = 1$ (resp. $B_{ij} = -1$) and $b_i = 0$, the vector $s$ must observe $s_j = -1$ (resp. $s_j = 1$). Let $J^+$ (resp. $J^-$) be the index sets of the $j$ for which $s_j = 1$ (resp. $s_j = -1$) must be enforced due to the rows $i$ with $b_i = 0$. If $J^+ \cap J^-$ is nonempty, the restrictions imposed on $s$ are contradictory, and $B \odot s = b$ has no solution. If $J^+ \cap J^-$ is empty, one solves $B \odot s = b$ by

solving the SAT instance $B \odot s \geq b$ while demanding $s_j = 1$, $j \in J^+$, and $s_j = -1$, $j \in J^-$. We call the problem of solving $B \odot s = b$ the *exact satisfiability problem* EXACT SAT. A similar reduction to SAT applies if just a subset of the inequalities of $B \odot s \geq b$ is replaced by equations. We refer to that more general situation as the SAT/EXACT SAT problem.

Let $T$ be a single CNF clause and thus a disjunction of possibly negated variables. The clause $T$ is a theorem of $S$ if $S \wedge \neg T$ is unsatisfiable. Since $\neg T$ is a conjunction, enforcing $\neg T$ amounts to fixing some variables of $S$ to certain *True/False* values. Suppose $S$ is specified by the inequality $B \odot s \geq b$ or the equation $B \odot s = b$. Then the fixing of a variable of $S$ to *True* (resp. *False*) is equivalent to demanding $s_j = 1$ (resp. $s_j = -1$). In the case of $B \odot s = b$, the condition $s_j = 1$ or $s_j = -1$ may be in conflict with one of the earlier cited conditions involving $J^+$ or $J^-$. If that is so, $T$ is trivially a theorem.

The above definitions and results have a straightforward extension to the logic minimization problem MINSAT. There, each variable $v_j$ of the given CNF formula $S$ is assigned a rational cost $c_j^+$ for the value *True* and a second rational cost $c_j^-$ for the value *False*. One must determine a satisfying solution that minimizes the total cost produced by that solution or must show $S$ to be unsatisfiable. Equivalently, one is to find a $\{\pm 1\}$ solution vector $s$ that satisfies $B \odot s \geq \underline{1}$ and that, subject to that condition, minimizes $\sum_{j \ni s_j = 1} c_j^+ + \sum_{j \ni s_j = -1} c_j^-$, or must declare that $B \odot s \geq \underline{1}$ has no solution. One could also define an exact satisfiability version of MINSAT, but we have no need for that problem here.

When we introduce into a matrix $E$ a horizontal (resp. vertical) partition that creates two submatrices $F$ and $G$, then we write $E = [F/G]$ (resp. $E = [F|G]$). A special case of horizontal partition is the partition of a column vector $e$ into subvectors $f$ and $g$. Accordingly, we write $e = [f/g]$.

We are prepared to discuss the exact-linear decomposition and a related solution algorithm for FUTILE SAT and FUTILE MINSAT.

## 4. Exact-Linear Decomposition

Let $B$ be the matrix representing a CNF formula $S$, say, with row index set $X$ for the clauses and column index set $Y$ for the variables. Let $Y_1 \subseteq Y$ be the index set of the question variables of $S$, and define $Y_2 = Y - Y_1$. Let $X_1 \subseteq X$ index the clauses of $S$ that do not contain any nonquestion variables, and define $X_2 = X - X_1$. When we partition $B$ according to $X_1$, $X_2$ and $Y_1$, $Y_2$, we get

$$B = \begin{array}{c|c|c} & Y_1 & Y_2 \\ \hline X_1 & A^1 & 0 \\ \hline X_2 & D & A^2 \\ \hline \end{array} \qquad (4.1)$$

Matrix $B$ for exact-linear decomposition

In terms of $S$, the submatrix $A^1$ represents the clauses of $S$ that specify only question variables. The submatrix $[D|A^2]$ represents the clauses containing nonquestion variables plus possibly question variables; the submatrix $D$ specifies the question variables, while $A^2$ specifies the nonquestion variables.

We decompose $B$ into two matrices $B^1$ and $B^2$ given by

$$B^1 = [A^1/D] \qquad B^2 = [D|A^2] \qquad (4.2)$$

and call these matrices the *components* of the exact-linear decomposition of $B$. We also refer to $B$ as the *exact-linear sum* of the components.

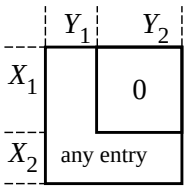
We want to capture the behavior of $D$ when it is $\mathbb{B}$-multiplied by all possible $\{\pm 1\}$ vectors. To this end, we use the following definition of *subrange* given in Truemper [1998] for a $\{0, \pm 1\}$ matrix $A$.

$$\text{subrange}(A) = \{b \mid b = A \odot s;\ s_j \in \{\pm 1\},\ \forall\ j\} \tag{4.3}$$

The cited reference contains several bounds for the cardinality of subrange$(A)$. We use the bound that for current purposes seems most useful. Let the BG-*rank* of a $\{0, \pm 1\}$ matrix $A$ be the rank of the matrix when the $\pm 1$s are replaced by reals that are algebraically independent over the rationals. According to Edmonds [1967], the BG-rank can be computed as follows, where we assume that $A$ has row index set $X$ and column index set $Y$. Create a bipartite undirected graph $G$ where $X$ is the node set on one side and where $Y$ is the node set on the other side. Connect node $x \in X$ with node $y \in Y$ by an undirected edge if the entry $A_{xy}$ is nonzero. Then BG-rank$(A)$ is equal to the size of a maximum cardinality matching of $G$. Such a matching can be efficiently found by network flow techniques and thus is easily computed; for details, see, for example, Ahuja, Magnanti, and Orlin (1993) or Cook, Cunningham, Pulleyblank, and Schrijver (1998). A byproduct of that way of computing BG-rank$(A)$ is the following result due to Edmonds [1967].

**(4.4) Theorem.** *Any* $\{0, \pm 1\}$ *matrix* $A$ *can be partitioned as*

$A =$




(4.5)

Partition of matrix $A$

*so that* $|X_2| + |Y_1| = \text{BG-rank}(A)$.

One may compute subrange$(A)$ as follows. $\mathbb{B}$-multiply an arbitrarily selected column of $A$ once by $+1$ and a second time by $-1$, and declare the resulting two $\{0, 1\}$ vectors to constitute the members of a set $U$. Process the remaining columns of $A$ one by one as follows: $\mathbb{B}$-multiply the selected column once by $+1$ and a second time by $-1$, thus getting two vectors, say $a$ and $b$; replace each vector $c$ of $U$ by the vectors $a \oplus c$ and $b \oplus c$.

Evidently, each $U$ at hand after the processing of an additional column of $A$ is the subrange of some column submatrix of $A$, and the final $U$ is equal to subrange$(A)$.

Let us call the above scheme Algorithm SUBRANGE. In general, the effort of that algorithm can vary significantly depending on the order in which the columns of $A$ are processed. But regardless of the sequence, the effort can always be uniformly bounded using BG-rank$(A)$, as shown by the next theorem.

**(4.6) Theorem.** *For each* $U$ *computed by Algorithm* SUBRANGE, $|U| \leq 2^{\text{BG-rank}(A)}$. *In particular,* $|\text{subrange}(A)| \leq 2^{\text{BG-rank}(A)}$.

**Proof.** Each $U$ is the subrange of some column submatrix of $A$. Since submatrix taking cannot increase the BG-rank, it suffices to show that $|\text{subrange}(A)| \leq 2^{\text{BG-rank}(A)}$, where $A$ is the matrix of (4.5). Apply Algorithm SUBRANGE to that matrix by selecting the columns from right to left. Suppose the columns indexed by $Y_2$ have been processed. Since in those columns only the rows indexed by $X_2$ contain nonzero entries, $U$ contains after processing of the columns of $Y_2$ at

most $2^{|X_2|}$ vectors. Hence, when the remaining columns indexed by $|Y_1|$ have been processed, $|U| = |\text{subrange}(A)| \leq 2^{|X_2|+|Y_1|}$. Since $|X_2| + |Y_1| = \text{BG-rank}(A)$, the desired conclusion follows. □

We also need a restricted version of the subrange set where some elements of the vectors $s$ producing the vectors $b = A \odot s$ are fixed to either $+1$ or $-1$. If the set of allowed vectors $s$ is, say, $N$, then we call the resulting subset of subrange the *subrange restricted by* $N$. The upper bound $2^{\text{BG-rank}(A)}$ on $|U|$ remains valid when Algorithm SUBRANGE is expanded in the obvious way to accommodate the restriction imposed by $N$. In a slight abuse of terminology, we still call the expanded scheme Algorithm SUBRANGE.

We are ready to tackle the problem FUTILE SAT. We are given the matrix $B$ of (4.1), which represents $S$. We partition the $\{\pm 1\}$ vector $s$ of $B \odot s \geq \underline{1}$ into two components $p$ and $q$, where $p$ (resp. $q$) is indexed by $Y_1$ (resp. $Y_2$) and corresponds to the question variables (resp. the nonquestion variables). Thus, $s = [p/q]$.

The negated conclusion, $\neg T$, implies a fixing of some of the variables. We enforce that fixing by demanding $p$ and $q$ to be in appropriately defined sets $P$ and $Q$. For example, if the $j$-th question variable must be fixed to *True* (resp. *False*), then among the conditions imposed on membership of $p$ in $P$ is the requirement $p_j = 1$ (resp. $p_j = -1$).

Recall that FUTILE SAT demands that we either show $T$ to be futile or compute $S$-feasible answers for the questions so that $S\neg T$ is unsatisfiable. There are trivial instances of the second case that we want to rule out. Suppose an answer of an $S$-feasible set is opposite to the value forced by $\neg T$. Then we have a case of unsatisfiable $S\neg T$, and thus $T$ is not futile. In practical applications, that case of nonfutility is not of interest, so from now on we require answers to questions not only to be $S$-feasible, but also to be consistent with the requirements imposed by $\neg T$. We call answers satisfying the latter requirement *nontrivial*. Throughout, we assume that there exists at least one set of nontrivial $S$-feasible answers.

In matrix notation, each set of nontrivial $S$-feasible answers corresponds to a vector $p^* \in P$ for which $B \odot [p^*/q] \geq \underline{1}$ has a solution. Such a $p^*$ establishes $T$ to be not futile if and only if $B \odot [p^*/q] \geq \underline{1}$ does not have a solution $q \in Q$. We determine whether such $p^*$ exists as follows.

Using the partition of $B$ given by (4.1), the inequality $B \odot [p/q] \geq \underline{1}$ can be rewritten as

$$\begin{aligned} A^1 \odot p &\geq \underline{1} \\ (D \odot p) \oplus (A^2 \odot q) &\geq \underline{1} \end{aligned} \tag{4.7}$$

We break up the inequalities of (4.7) into (4.8) and (4.9) below by introducing a vector $d \in \text{subrange}(D)$. That is,

$$\begin{aligned} A^1 \odot p &\geq \underline{1} \\ D \odot p &= d \end{aligned} \tag{4.8}$$

and

$$A^2 \odot q \geq \underline{1} \ominus d \tag{4.9}$$

One may link (4.7) with (4.8) and (4.9) as follows.

**(4.10) Theorem.** *A pair $(p, q)$ is a solution of (4.7) if and only if there exists a $d \in \text{subrange}(D)$ such that (4.8) and (4.9) are satisfied using $(p, q)$ and $d$.*

**Proof.** $\Rightarrow$: Given $(p, q)$ satisfying (4.7), let $d = D \odot p$. Then (4.8) clearly holds. We use (3.6) to deduce from $(D \odot p) \oplus (A^2 \odot q) = d \oplus (A^2 \odot q) \geq \underline{1}$ the inequality $A^2 \odot q \geq \underline{1} \ominus d$. Thus, (4.9) holds.

$\Leftarrow$: Let (4.8) and (4.9) be satisfied for some $d \in \text{subrange}(D)$. Then (4.8) implies the first inequality of (4.7). To prove the second inequality of (4.7), we use the equation of (4.8) in the inequality of (4.9), apply (3.6), and obtain $(D \odot p) \oplus (A^2 \odot q) \geq \underline{1}$. □

Theorem (4.10) suggests that we determine futility of $T$ by solving (4.8) and (4.9) repeatedly, each time using another $d \in \text{subrange}(D)$. A shortcut allows some $d \in \text{subrange}(D)$ to be skipped. Specifically, if $d^1 \leq d^2$, then a solution $q$ for (4.9) with $d = d^1$ is also a solution for (4.9) with $d = d^2$.

The above arguments also apply to the FUTILE MINSAT case, except that we solve MINSAT versions of (4.9) and search for a case where the minimum total cost is as large as possible.

As an aside, let us rewrite (4.9) to the equivalent $D \odot p = d$ and $(D \odot p) \oplus (A^2 \odot q) \geq \underline{1}$. Then we see that (4.8) and (4.9) involve the component matrices $B^1$ and $B^2$ of (4.2). Hence, we have transformed a problem involving $B$ into several problems involving the component matrices $B^1$ and $B^2$.

### (4.11) Algorithm SOLVE EXACT-LINEAR SUM FUTILE SAT/MINSAT.

*Input*: CNF formula $S$ and CNF clause $T$ encoded by a matrix $B$ and sets $P$ and $Q$. The latter sets represent $\neg T$. FUTILE MINSAT case: For each nonquestion variable $q_j$, rational costs $c_j^+$ and $c_j^-$.
Assumption: $A^1 \odot p \geq \underline{1}$, $p \in P$, has at least one solution. (If this does not hold, $T$ is trivially a theorem of $S$.)

*Output*: Either: A vector $p^* \in P$ that establishes $T$ to be not futile.
Or: FUTILE SAT case: "$T$ is futile." FUTILE MINSAT case: Vectors $p^* \in P$ and $q^* \in Q$ and scalar $z^*$ for which the following holds. If the answers are chosen according to $p^*$, then $q^*$ provides for the nonquestion variables a solution with minimum total cost, which is $z^*$. That minimum total cost is largest given all possible choices for $p$.

*Complexity*: Polynomial if BG-rank($D$) is bounded by a constant and if the SAT/EXACT SAT instances of Step 4 and the applicable SAT and MINSAT instances of Step 5 are polynomially solvable.

*Procedure*:

1. (Initialization) Determine the exact-linear decomposition of $B$ as given by (4.1). Compute the subrange restricted by $P$ for the submatrix $D$ of $B$. Initialize $U$ to that restricted subrange. FUTILE MINSAT case: Initialize $z^* = -\infty$.
2. (Termination test) If $U$ is empty: FUTILE SAT case: Declare $T$ to be futile, and stop. FUTILE MINSAT case: Output $p^*$, $q^*$, and $z^*$, and stop.
3. (Select next $d$) Select a $d \in U$ with minimum number of 1s.
4. (Test for existence of $p$) Check if

$$\begin{aligned} A^1 \odot p &\geq \underline{1} \\ D \odot p &= d \\ p &\in P \end{aligned} \tag{4.12}$$

   has a solution. If a solution does not exist, remove $d$ from $U$, and go to Step 2.
5. (Test for nonfutility of $T$) Check if

$$\begin{aligned} A^2 \odot q &\geq \underline{1} \ominus d \\ q &\in Q \end{aligned} \tag{4.13}$$

has a solution. If a solution does not exist, output the solution vector most recently obtained for (4.12) in Step 4 as a vector $p^*$ proving that $T$ is not futile, and stop.
Otherwise, remove from $U$ all vectors $d'$ satisfying $d' \geq d$, and do the following. FUTILE SAT case: Go to Step 2. FUTILE MINSAT case: Find a solution $q^*$ for (4.13) that minimizes total cost $\sum_{j \ni q_j = 1} c_j^+ + \sum_{j \ni q_j = -1} c_j^-$. If that total cost is larger than $z^*$, increase $z^*$ to that total cost, retain the corresponding solution vector for (4.13) in $q^*$, and retain the solution vector obtained for (4.12) in the most recent pass through Step 4 in $p^*$. Go to Step 2.

Truemper (1998) defines large polynomially solvable subclasses for SAT and MINSAT. If the instances encountered in Steps 4 and 5 fall into these classes, then Algorithm (4.11) is polynomial if BG-rank($D$) is bounded by a constant. The reference also describes a compiler that constructs SAT or MINSAT solution algorithms for the following setting. Given are a $\{0, \pm 1\}$ matrix $A$ and, in the MINSAT case, two rational cost vectors. The compiler constructs a solution algorithm that is valid for all SAT or MINSAT instances created by submatrix taking of $A$ and, if applicable, by corresponding subvector taking of the cost vectors. The compiler also supplies an upper bound on the run-time of the solution algorithm. When the compiler is applied to the matrix $[A^1/D]$ of (4.12) and to $A^2$ of (4.13) plus possibly the cost vectors, then the solution algorithms so obtained may be employed in Algorithm (4.11) to handle Steps 4 and 5. The upper bounds on run-time for these algorithms can be used to compute an upper bound for the entire scheme. The latter bound has several uses. For example, it supplies a performance guarantee for real-time applications.

The performance of Algorithm (4.11) crucially depends on BG-rank($D$). If that rank is not large and if the subproblems of Steps 4 and 5 can be well solved, the performance is good. These conditions are satisfied in many practical applications. However, there are application cases where the subproblems of Steps 4 and 5 can be readily solved, but where BG-rank($D$) is too large for Algorithm (4.11) to be useful. For those situations, the approach described in the next section or, more likely, its extension given in Section 6 may be appropriate.

## 5. Antimonotone Decomposition

In many practical applications, answers for the question variables are independent inputs into a logic formula, and the relations among the questions are completely captured by the clauses involving question variables only and are not influenced by the nonquestion variables. Thus, the clauses of $S$ that involve just question variables give a complete characterization of the $S$-feasible answers. We rely on that consideration in this section and the next one.

The antimonotone decomposition partitions the matrix $B$ as in (4.1). That is,

$$B = \begin{array}{c|c|c} & Y_1 & Y_2 \\ \hline X_1 & A^1 & 0 \\ \hline X_2 & D & A^2 \\ \hline \end{array} \qquad (5.1)$$

Matrix $B$ for antimonotone decomposition

where the submatrix $A^1$ represents the clauses involving only question variables, and where the submatrix $[D|A^2]$ represents the remaining clauses. However, additional requirements are imposed. Define a $\{0, \pm 1\}$ matrix to be *nearly negative* if each row of the matrix contains at most

one +1. The additional conditions are as follows: The submatrix $A^1$ must be nearly negative, and the submatrix $D$ must be nonnegative.

In Truemper (1998), a decomposition is described that is identical to the above one except that $D$ is required to be nonpositive. That decomposition is called *monotone*. Due to the switch from nonpositivity of $D$ of the monotone decomposition to nonnegativity demanded here, we call the decomposition of this section *antimonotone*. The component matrices of the antimonotone decomposition are

$$B^1 = A^1 \qquad\qquad B^2 = [D|A^2] \tag{5.2}$$

which also are the components for the monotone decomposition. We also say that $B$ is an *antimonotone sum* of $B^1$ and $B^2$. The earlier stated assumption about $S$-feasible answers can now be phrased as follows.

**(5.3) Assumption.** *Each vector $p$ satisfying $A^1 \odot p \geq \underline{1}$ corresponds to a set of $S$-feasible answers.*

The requirement that $A^1$ be nearly negative translates to the condition that each one of the related clauses of $S$ contains at most one nonnegated variable. Thus, those clauses have the well-known *Horn form*. Horn clauses induce a certain property that is useful for our purposes. Define a $\{0, \pm 1\}$ vector $e$ to be *minimum with respect to* +1 for a specified set of $\{0, \pm 1\}$ vectors $f$ of the same length as $e$ if, for any $f$ of the given set and any index $j$, $e_j = 1$ implies $f_j = 1$. We often abbreviate the terminology by omitting reference to the set of vectors $f$ when that set is clear from the context.

**(5.4) Theorem.** (Dowling and Gallier (1984)) *If $A \odot s \geq b$ with nearly negative $A$ and arbitrary $\{0, 1\}$ $b$ has a solution, then it has a solution that is minimum with respect to +1 for the set of solutions.*

A well-known linear-time solution algorithm for $A \odot s \geq b$ with nearly negative $A$ due to Itai and Makowsky (1982, 1987) either determines the inequality to have no solution or finds a solution vector that is minimum with respect to +1 for the set of solutions. We summarize the algorithm. It recursively fixes $s_j$ variables whose values are uniquely determined by some row $i$ of $A$ with just one nonzero entry and $b_i = 1$, reduces the inequality system accordingly, and either derives a contradictory situation or cannot continue since each row $i$ of $A$ with $b_i = 1$ has at least two nonzero entries. In the former case, $A \odot s \geq b$ has no solution. In the latter case, the algorithm fixes the $s_j$ that so far have not received a value to $-1$; the $\{\pm 1\}$ vector $s$ so determined is a solution that is minimum with respect to +1. Let us call this scheme Algorithm SOLVE NEARLY NEGATIVE SAT. We use the scheme as a subroutine in the next algorithm for solving FUTILE SAT or FUTILE MINSAT instances involving $B$. The algorithm is simple enough that a few comments added in parentheses supply the proof of validity.

**(5.5) Algorithm SOLVE ANTIMONOTONE SUM FUTILE SAT/MINSAT.**

*Input*: CNF formula $S$ and CNF clause $T$ encoded by a matrix $B$ and sets $P$ and $Q$. The latter sets represent $\neg T$. The matrix $B$ has an antimonotone decomposition. FUTILE MINSAT case: For each nonquestion variable $q_j$, rational costs $c_j^+$ and $c_j^-$.
Assumptions: $A^1 \odot p \geq \underline{1}$, $p \in P$ has at least one solution. (If this assumption does not hold, $T$ is trivially a theorem of $S$.) Also, Assumption (5.3) holds.

*Output*: Either: A vector $p^* \in P$ that establishes $T$ to be not futile.
Or: FUTILE SAT case: "$T$ is futile." FUTILE MINSAT case: Vectors $p^* \in P$ and $q^* \in Q$ and scalar $z^*$ for which the following holds. If the answers are chosen according to $p^*$, then $q^*$

provides for the nonquestion variables a solution with minimum total cost, which is $z^*$. That minimum total cost is largest given all possible choices for $p$.

*Complexity*: Polynomial if the applicable SAT and MINSAT instances of Step 2 are polynomially solvable.

*Procedure*:

1. Use Algorithm SOLVE NEARLY NEGATIVE SAT to obtain for

$$\begin{aligned} A^1 \odot p &\geq \underline{1} \\ p &\in P \end{aligned} \tag{5.6}$$

the solution vector $p^*$ that is minimum with respect to $+1$ for the set of solutions of (5.6). Let $d^* = D \odot p^*$. (By the input assumptions, $p^*$ must exist. Since $p^*$ is minimum with respect to $+1$ and since $D$ is nonnegative, the vector $d^*$ is minimum with respect to $+1$ for the set of vectors $d'$ obtained from solutions $p'$ of (5.6) by $d' = D \odot p'$. Hence, the inequality $A^2 \odot q \geq \underline{1} \ominus d'$, $q \in Q$, is hardest to satisfy if $d' = d^*$. The next step relies on this fact.)

2. Check if

$$\begin{aligned} A^2 \odot q &\geq \underline{1} \ominus d^* \\ q &\in Q \end{aligned} \tag{5.7}$$

has a solution. If a solution does not exist, output the solution vector obtained for (5.6) in Step 1 as a vector $p^*$ proving that $T$ is not futile, and stop.
Otherwise: FUTILE SAT case: Declare $T$ to be futile, and stop. FUTILE MINSAT case: Find a solution $q^*$ for (5.7) that minimizes total cost $\sum_{j \ni q_j = 1} c_j^+ + \sum_{j \ni q_j = -1} c_j^-$. Output $p^*$, $q^*$, and minimum total cost $z^*$, and stop.

Note that the algorithm does not make full use of Assumption (5.3). Indeed, it suffices that the solutions $p$ to (5.6) correspond to $S$-feasible answers. We can test for that reduced requirement by redefining $T$ so that $\neg T$ imposes just the restrictions of $P$ on $p$ and imposes no restrictions on $q$. Then the reduced requirement holds if and only the revised $T$ is futile. We may similarly test if Assumption (5.3) holds. This time, we take $T = \mathit{False}$. Then $\neg T = \mathit{True}$, no restrictions are imposed on $p$ and $q$, and $T$ is futile if and only if Assumption (5.3) holds.

Just as in the case of the exact-linear decomposition, one may use the large polynomially solvable subclasses for SAT and MINSAT of Truemper (1998) and the antimonotone decomposition to construct still-large polynomially solvable subclasses of FUTILE SAT and FUTILE MINSAT. One may also use the compiler of that reference analogously to the case of Algorithm (4.11).

There is one troublesome aspect, though. Logic formulations of real-world problems tend not to admit the antimonotone decomposition. This is due to the fact that the clauses of applications often are implications of the form "some question variables imply some nonquestion variables." A simple example is $v \Rightarrow w$, where $v$ is a question variable and $w$ is a nonquestion variable. In CNF form, that implication becomes $\neg v \vee w$ and thus results in a $-1$ entry in the submatrix $D$ of $B$. But there is a process that reduces a matrix without antimonotone decomposition to a matrix that has one. We cover that process next.

## 6. Extension of Antimonotone Decomposition

Suppose the matrix of a FUTILE SAT or FUTILE MINSAT instance does not admit an antimonotone decomposition. We apply a two-step process to reduce the given matrix to one with the desired property.

We need an expanded notation. Let $\tilde{S}$ be the given CNF system and $\tilde{T}$ be the CNF clause of the FUTILE SAT or FUTILE MINSAT problem. Define $\tilde{B}$ with column index set $\tilde{Y}$ and row index set $X$ to be the matrix representing $\tilde{S}$. We partition $\tilde{Y}$ into $\tilde{Y}_1$ and $Y_2$ where $\tilde{Y}_1$ corresponds to the question variables. We partition $X$ into $X_1$ and $X_2$ where $X_1$ corresponds to the clauses that contain just question variables.

The two-step process is as follows.

First, we remove from $\tilde{B}$ some columns $y \in \tilde{Y}_1$. Later, we use enumeration to account for the removed columns.

Second, we scale the retained columns of $\tilde{Y}_1$ by suitably selected $\{\pm 1\}$ factors. The scaling by $-1$ corresponds to replacing a question variable of $\tilde{S}$ by its complement, while scaling by $+1$ has no effect.

The two steps are so carried out that the reduced matrix has an antimonotone decomposition. The reduction is most effective when as few columns of $\tilde{Y}_1$ are removed from $\tilde{B}$ as possible. We formulate the selection problem as well as the choice problem of the scaling factors as an integer program (IP). For each $y \in \tilde{Y}$, define two $\{0,1\}$ variables $w_y^+$ and $w_y^-$ as follows.

$$\begin{aligned} w_y^+ &= \begin{cases} 1 & \text{if column } y \text{ is retained and scaled by } +1 \\ 0 & \text{otherwise} \end{cases} \\ w_y^- &= \begin{cases} 1 & \text{if column } y \text{ is retained and scaled by } -1 \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{6.1}$$

Evidently, a column $y \in \tilde{Y}_1$ is deleted from $\tilde{B}$ if and only if $w_y^+ = w_y^- = 0$. We can scale just one way, so we must enforce, for all $y \in \tilde{Y}_1$,

$$w_y^+ + w_y^- \leq 1 \tag{6.2}$$

If a column $y$ of $D$ contains a $+1$ (resp. $-1$), then scaling that column by $-1$ (resp. $+1$) is not allowed. Hence, for each $y \in \tilde{Y}_1$, if there exists an $x \in X_2$ such that $D_{xy} = +1$, then

$$w_y^- = 0 \tag{6.3}$$

and if there exists an $x \in X_2$ such that $D_{xy} = -1$, then

$$w_y^+ = 0 \tag{6.4}$$

The scaling must convert the submatrix of $\tilde{B}$ defined by the retained columns of $\tilde{Y}_1$ and by the row index set $X_1$ to nearly negative form. This is achieved by demanding, for each $x \in X_1$,

$$\sum_{y \ni \tilde{B}_{xy} = 1} w_y^+ + \sum_{y \ni \tilde{B}_{xy} = -1} w_y^- \leq 1 \tag{6.5}$$

Since the goal is the retention of a maximum number of columns $y \in \tilde{Y}_1$, the objective function is

$$\max \sum_y w_y^+ + w_y^- \tag{6.6}$$

The IP given by (6.1)–(6.6) is nothing but a version of the following problem (see Chandru and Hooker (1992) or Truemper (1998)): Given a $\{0, \pm 1\}$ matrix, select a column submatrix with maximum number of columns such that the selected submatrix can be column scaled to become nearly negative. Let us call the latter problem MAX NEARLY NEGATIVE SUBMATRIX. Since that problem is known to be $\mathcal{NP}$-hard (see the cited references), the problem considered here is $\mathcal{NP}$-hard as well.

A certain polynomial heuristic method described in Truemper (1998) is very effective in finding good if not optimal solutions for the problem MAX NEARLY NEGATIVE SUBMATRIX. Hence that method can be expected to perform well when applied to the IP (6.1)–(6.6). Due to space limitations, we skip details of the heuristic method and mention only that it is based on linear programming and limited enumeration of certain subcases. Let us call that method Heuristic MAX NEARLY NEGATIVE SUBMATRIX.

Suppose we have applied Heuristic MAX NEARLY NEGATIVE SUBMATRIX to a given input matrix $\tilde{B}$. Let $Z$ be the subset $\tilde{Y}_1$ indexing the deleted columns, and define $Y_1 = \tilde{Y}_1 - Z$. Declare $E$ to be the column submatrix of $\tilde{B}$ indexed by $Z$. Let $F$ (resp. $G$) be the row submatrix of $E$ indexed by $X_1$ (resp. $X_2$). Thus, $E = [F/G]$.

Define $B$ to be the submatrix of the retained and scaled columns. By the derivation, $B$ is of the form (5.1), and its partition provides an antimonotone decomposition. Therefore, the submatrix $A^1$ of $B$ is nearly negative, and the submatrix $D$ is nonnegative.

As before, we use vectors $p$ and $q$ to represent the question and nonquestion variables associated with $B$. In addition, we define a new vector $r$ for the question variables associated with $E$.

Analogously to the partition of $\tilde{B}$ into $B$ and $E$, we need a partitioned representation of the negated version of the CNF clause $\tilde{T}$. That is, a set $R$ represents the part of $\neg\tilde{T}$ concerning the vector $r$, while $P$ and $Q$ represent the portions of $\neg\tilde{T}$ concerning $p$ and $q$, respectively.

With this notation, the FUTILE SAT problem for $\tilde{B}$ demands the following: Find $r \in R$ and $p \in P$ for which there exists a $q$ such that

$$(E \odot r) \oplus (B \odot [p/q]) \geq \underline{1} \tag{6.7}$$

has a solution and such that (6.7) with the added condition $p \in Q$ has no solution, or conclude that no such $r \in R$ and $p \in P$ exist. In the first case, $\tilde{T}$ is nonfutile, while in the second case it is futile. The FUTILE MINSAT problem is similar, except that in the case of futile $\tilde{T}$ one must produce $r \in R$ and $p \in P$ for which minimum total cost defined by $q$ is largest.

When we expand the inequality of (6.7) using the submatrices $F$ and $G$ of $E$ as well as the submatrices $A^1$, $D$, and $A^2$ of $B$, we get

$$\begin{aligned} (F \odot r) \oplus (A^1 \odot p) &\geq \underline{1} \\ (G \odot r) \oplus (D \odot p) \oplus (A^2 \odot q) &\geq \underline{1} \end{aligned} \tag{6.8}$$

Assumption (5.3) used earlier for the antimonotone decomposition case becomes the following statement.

**(6.9) Assumption.** *Every pair $(r, p)$ satisfying $(F \odot r) \oplus (A^1 \odot p) \geq \underline{1}$ corresponds to a set of $\tilde{S}$-feasible answers.*

To simplify the description of the solution algorithm, we let the input consist of the matrices $E$ and $B$ representing $\tilde{S}$, the sets $R$, $P$, and $Q$ representing $\neg\tilde{T}$, and, in the FUTILE MINSAT case, the rational cost vectors.

Caution: The columns of $B$ indexed by $Y_1$ are derived from $\tilde{B}$ of $\tilde{S}$ by scaling, so the set $P$ must account for that scaling if it is to represent its part of $\neg\tilde{T}$ properly. Similarly, any vector $p^*$ of the output must be scaled for correct interpretation in $\tilde{S}$.

We list the solution algorithm and subsequently prove its validity.

**(6.10) Algorithm SOLVE EXTENDED ANTIMONOTONE SUM FUTILE SAT/MINSAT.**

*Input*: Matrix $E = [F/G]$ as well as $B$ of (5.1) representing $\tilde{S}$, and sets $R$, $P$, and $Q$ representing $\neg\tilde{T}$. FUTILE MINSAT case: For each nonquestion variable $q_j$, rational costs $c_j^+$ and $c_j^-$.
Assumptions: $(F \odot r) \oplus (A^1 \odot p) \geq \underline{1}$, $r \in R$, $p \in P$, has at least one solution. (If this does not hold, $\tilde{T}$ is trivially a theorem of $\tilde{S}$.) Also, Assumption (6.9) holds.

*Output*: Either: Vectors $r^* \in R$ and $p^* \in P$ that establish $\tilde{T}$ to be not futile.
Or: FUTILE SAT case: "$\tilde{T}$ is futile." FUTILE MINSAT case: Vectors $r^* \in R$, $p^* \in P$, $q^* \in Q$, and scalar $z^*$ for which the following holds. If the answers are chosen according to $r^*$ and $p^*$, then $q^*$ provides for the nonquestion variables a solution with minimum total cost, which is $z^*$. That minimum total cost is largest given all possible choices for $r$ and $p$.

*Complexity*: Polynomial if BG-rank$(E)$ is bounded by a constant and if the applicable SAT and MINSAT instances of Step 5 are polynomially solvable.

*Procedure*:

1. (Initialization) Use Algorithm SUBRANGE to compute for $E$ the subrange restricted by $R$. Let $U$ be that restricted subrange. During the execution of Algorithm SUBRANGE, also create, for each $e \in U$, one $\{\pm 1\}$ vector $r(e) \in R$ so that $e = E \odot r(e)$. FUTILE MINSAT case: Initialize $z^* = -\infty$.
2. (Termination Test) If $U$ is empty: FUTILE SAT case: Declare $T$ to be futile, and stop. FUTILE MINSAT case: Output $r^*$, $p^*$, $q^*$, and $z^*$, and stop.
3. (Select next $e$) Select an $e = [f/g]$ of $U$ where $f$ has maximum number of 1s and where, subject to that condition, $g$ has minimum number of 1s.
4. (Test for existence of $p$) Use Algorithm SOLVE NEARLY NEGATIVE SAT to obtain for
$$\begin{aligned} &A^1 \odot p \geq \underline{1} \ominus f \\ &p \in P \end{aligned} \tag{6.11}$$
the solution vector $p(f)$ that is minimum with respect to $+1$ for the set of solutions of (6.11), or conclude that (6.11) has no solution. In the latter case, remove from $U$ all vectors $e' = [f'/g']$ where $f' \leq f$ and where $g'$ is arbitrary, and go to Step 2. In the former case, let $d = D \odot p(f)$.
5. (Test for nonfutility of $\tilde{T}$) Check if
$$\begin{aligned} &A^2 \odot q \geq \underline{1} \ominus (g \oplus d) \\ &q \in Q \end{aligned} \tag{6.12}$$
has a solution. If a solution vector does not exist, output $r^* = r(e)$ and $p^* = p(f)$ as vectors proving that $\tilde{T}$ is not futile, and stop.
Otherwise, remove from $U$ all vectors $e' = [f'/g']$ satisfying $f' \leq f$ and $g' \geq g$, and do the following. FUTILE SAT case: Go to Step 2. FUTILE MINSAT case: Find a solution $q^*$ for (6.12) that minimizes total cost $\sum_{j \ni q_j = 1} c_j^+ + \sum_{j \ni q_j = -1} c_j^-$. If that total cost is larger than $z^*$, increase $z^*$ to that total cost, retain the corresponding solution vector for (6.12) in $q^*$, and retain $r(e)$ in $r^*$ and $p(f)$ in $p^*$. Go to Step 2.

**Proof of Validity.** We first treat the FUTILE SAT case.

Suppose the algorithm stops in Step 5. Since $e = E \odot r(e) = [F \odot r(e)/G \odot r(e)] = [f/g]$, we have $f = F \odot r(e)$ and $g = G \odot r(e)$. Also, $p(f)$ satisfies (6.11), so $(F \odot r(e)) \oplus (A^1 \odot p(f)) = f \oplus (A^1 \odot p(f)) \geq \underline{1}$, $r(e) \in R$, $p(f) \in P$. Hence, Assumption (5.9) implies that the pair $(r(e), p(f))$ represents a set of $\tilde{S}$ feasible answers. Due to the termination in Step 5, we know that (6.12) with $g = G \odot r(e)$ and $d = D \odot p(f)$ has no solution. Thus, $\tilde{T}$ is not futile, as desired.

Suppose the algorithm stops in Step 2. Then $U$ is empty, and we must show $\tilde{T}$ to be futile. Take an arbitrary pair $(r', p')$, $r' \in R$, $p' \in P$, that represents a set of $\tilde{S}$-feasible answers. Let $f' = F \odot r'$, $g' = G \odot r'$, $e' = [f', g']$, and $d' = D \odot p'$. We must show that there exists $q' \in Q$ such that $A^2 \odot q' \geq \underline{1} \ominus (g' \oplus d')$. The vector $e'$ was in the original $U$. At some point, $e'$ was removed from $U$, say, when $e = [f/g]$ was processed and removed from $U$ at the same time. The two removals took place either in Step 4 or in Step 5. We examine the two cases.

Removal of $e$ and $e'$ from $U$ in Step 4 implies that (6.11) has no solution and that $f' \leq f$. But we assume that $(r', p')$ represents a set of $\tilde{S}$-feasible answers, so $p' \in P$ and

$$A^1 \odot p' \geq \underline{1} \ominus f' \geq \underline{1} \ominus f \tag{6.13}$$

which provides the contradictory conclusion that (6.11) has a solution. Hence, the removal of $e$ and $e'$ in Step 4 is not possible and must therefore take place in Step 5. The preceding Step 4 has produced $p(f)$ that is minimum with respect to $+1$ for the set of solutions to (6.11). In particular, we see from (6.13) that, for any $j$, $p(f)_j = 1$ implies $p'_j = 1$. Since $D$ is nonnegative, we therefore have $d' = D \odot p' \geq D \odot p(f) = d$. Using the inequality $g' \geq g$ of Step 5 and the solution for (6.12), say, $q^*$, found in that step, we have

$$A^2 \odot q^* \geq \underline{1} \ominus (g \oplus d) \geq \underline{1} \ominus (g' \oplus d') \tag{6.14}$$

Hence, $q' = q*$ satisfies $A^2 \odot q' \geq \underline{1} \ominus (g' \oplus d')$, and $\tilde{T}$ has been shown to be futile.

The above arguments also prove validity for the FUTILE MINSAT case once we show that, in the case of futile $\tilde{T}$, it suffices that we consider the vectors $e = [f/g]$ selected in Step 3 to obtain correct $r^*$, $p^*$, $q^*$, and $z^*$. But validity of that selection follows from the above arguments concerning $r'$ and $p'$. Indeed, by (6.14), these vectors cannot produce a larger minimum cost than the vectors $r(e)$ and $p(f)$. □

Similarly to the use of Assumption (5.3) in Algorithm (5.5), the above algorithm does not fully require Assumption (6.9). Indeed, it suffices that the pairs $(r(e), p)$ where $e = [f/g] \in U$ and where $p$ is a solution of (6.11) correspond to $\tilde{S}$-feasible answers. Testing for that reduced assumption or for the full Assumption (6.9) is done analogously to the case for Assumption (5.3) and Algorithm (5.5). That is, for the test of the reduced assumption, we redefine $\tilde{T}$ so that $\neg\tilde{T}$ imposes just the restrictions of $R$ on $r$ and of $P$ on $p$ and imposes no restrictions on $q$. For the test of the full Assumption (6.9), we take $\tilde{T} = \mathit{False}$ and impose no restrictions on $r$, $p$, and $q$.

One may combine the large polynomially solvable subclasses of SAT and MINSAT of Truemper (1998) with the class defined by the extended antimonotone decomposition with bounded BG-rank($E$) to construct still-large polynomially solvable subclasses of FUTILE SAT and FUTILE MINSAT. In contrast to the situation involving the antimonotone decomposition of Section 5, this time the resulting classes do include many instances arising from real-world applications and thus make the results of this section practically useful.

Up to this point, the decompositions of Sections 4 and 5 and of this section were selected independently of $T$ or $\tilde{T}$. That approach is appropriate if the upper bound on the solution effort is reasonable. However, if that bound is large, one may want to consider the following alternate procedure.

Given $T$ or $\tilde{T}$, one reduces $B$ or $\tilde{B}$ by fixing variables according to $\neg T$ or $\neg\tilde{T}$ and deleting satisfied clauses. Let $B'$ be the matrix derived from $B$ or $\tilde{B}$. One determines a decomposition for $B'$ and solves the FUTILE SAT or FUTILE MINSAT instance involving $B'$ and $T' = \mathit{False}$. If $B'$ is substantially smaller than $B$ or $\tilde{B}$, an improved bound on the solution effort as well as a shorter solution time may be attained. The drawback of the method is the extra computing effort for the analysis of $B$ or $\tilde{B}$ for each instance of $\tilde{T}$.

## 7. Applications

Many expert systems interactively acquire information from the user and make decisions based on those data. In the terminology of this paper, such systems pose questions that are answered by the user. For example, a diagnostic system may query the user to establish a diagnostic goal that the system has selected based on a preliminary analysis. If the goal happens to be unprovable regardless of the answers supplied by the user, then that goal is futile and should be abandoned. The algorithms of the preceding sections can decide whether that situation is at hand and thus can cut short fruitless lines of questioning.

In other expert systems, information for decision making is not acquired by interaction with the user, but is obtained by querying other systems. A cost may be explicitly or implicitly associated with each query, and one would want to avoid useless queries. Here, too, the case of futile questioning may arise and can be handled with the techniques of this paper.

In the two examples, futility of a goal or decision has a somewhat negative connotation. That is, one would prefer that futility would not occur and that one could establish the desired goal or decision. There are examples where the opposite is true and where futility is desirable. Consider an expert system for regulatory compliance. An example is the system for management of hazardous materials described in Straach and Truemper [1998a, 1998b]. The question variables depict the setting or situation to be managed. The clauses embody the applicable regulations imposed by various agencies. Let the statement $T$ represent the decision that a certain action is done. If $T$ (resp. $\neg T$) can be shown to be futile, then the regulations cannot force the action to be done (resp. not to be done) regardless how the questions are answered. Hence, if both $T$ and $\neg T$ are futile, then we have the attractive situation where according to the regulations, the action is optional and can be decided upon using other criteria—for example, cost or availability of resources.

In another setting of expert systems, only the minimum total cost of a logic minimization instance is needed to make a certain decision. If the system can prove that any answers for the currently open questions cannot increase minimum total cost beyond a certain value, then the system can make the decision right now, without getting answers for those questions.

There is yet another setting. This time, an expert system wants to assess how many undesirable or dangerous results represented by propositional *result variables* $t_i$ may simultaneously apply in a worst-case scenario. Each one of the possible scenarios is specified by a set of *True/False* values for a subset of the nonresult variables. We declare the nonresult variables of that subset to be the question variables.

We obtain a worst-case scenario as follows. Let $S$ be the CNF formula describing the relations between the question variables, the $t_i$ variables, and, possibly, auxiliary variables. To each $t_i$, we assign a cost of 1 (resp. 0) for the value *True* (resp. *False*). We take $T$ to be $T = \mathit{False}$ and solve the FUTILE MINSAT instance given by $S$, $T$, and the cost values for the $t_i$. If $T$ is not futile, then the logic formulation of $S$ is in error. So let $T$ be futile. We thus obtain *True/False* values for the question variables such that the minimum total cost is largest. Since that minimum cost tells how many $t_i$ are forced to have the value *True*, the *True/False* values for the question variables give the desired worst-case scenario.

In general, we cannot determine which of the results represented by the $t_i$ must occur under that scenario. For example, let $S$ consist of one question variable $v$, two conclusion variables $t_1$ and $t_2$, and the single clause $v \Rightarrow t_1 \vee t_2 = \neg v \vee t_1 \vee t_2$. Then the worst-case scenario is unique and is given by $v = \mathit{True}$. For that case, one of $t_1$ and $t_2$ must be *True*, but it cannot be determined which one.

However, one can try to settle for as many $t_i$ as possible whether they must be *True* under the worst-case scenario given by the FUTILE MINSAT solution. For this, we fix the question variables of $S$ to the just determined *True/False* values. Let $S'$ be the modified formula. In turn, we take each $t_i$ for which we had the value *True* in the FUTILE MINSAT solution and check whether $t_i$ is a theorem of $S'$. If $t_i$ is a theorem, then the corresponding result must occur.

In some cases, the undesirability of the various results varies. One then defines the cost of *True* for $t_i$ to be a positive integer that reflects the degree of undesirability. If the total undesirability of a scenario is reasonably expressed by the sum of the cost values of the $t_i$ with value *True*, then the above approach determines *True/False* values for the question variables that define a worst-case scenario.

**Acknowledgement:** We thank D. T. Huynh and R. E. Bryant for clarifying the computational complexity question and the role of binary decision diagrams, respectively, for the futile questioning problem FUTILE SAT.

## 8. References

Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993), *Network Flows*, Prentice-Hall, Englewood Cliffs, New Jersey, 1993.

Bryant, R. E. (1986), Graph-based algorithms for Boolean function manipulation, *IEEE Transactions on Computers* C-35 (1986) 677–691.

Bryant, R. E. (1992), Symbolic Boolean manipulation with ordered binary-decision diagrams, *ACM Computing Surveys* 24 (1992) 293–318.

Bryant, R. E. (1995), Binary decision diagrams and beyond: Enabling techniques for formal verification, Proceedings of *IEEE International Conference on Computer-Aided Design (ICCAD '95)*, San Jose, CA, 1995, pp. 236–243.

Chandru, V., and Hooker, J. N. (1992), Detecting embedded Horn structure in propositional logic, *Information Processing Letters* 42 (1992) 109–111.

Cook, W. J., Cunningham, W. H., Pulleyblank, W. R., and Schrijver, A. (1998), *Combinatorial Optimization*, Wiley, New York, 1998.

Dowling, W. F., and Gallier, J. H. (1984), Linear-time algorithms for testing the satisfiability of propositional Horn formulae, *Journal of Logic Programming* 1 (1984) 267–284.

Edmonds, J. (1967), Systems of distinct representatives and linear algebra, *Journal of Research of the National Bureau of Standards (B)* 71B (1967) 241–245.

Itai, A., and Makowsky, J. A. (1982), On the complexity of Herbrand's theorem, working paper 243, Department of Computer Science, Israel Institute of Technology, 1982.

Itai, A., and Makowsky, J. A. (1987), Unification as a complexity measure for logic programming, *Journal of Logic Programming* 4 (1987) 105–117.

Kneale, W., and Kneale, M. (1984), *The Development of Logic*, Clarendon Press, Oxford, 1984.

Newman, J. R. (1956), *The World of Mathematics*, Vols. 3 and 4, Simon and Schuster, New York, 1956.

Stockmeyer, L. J. (1976), The polynomial-time hierarchy, *Theoretical Computer Science* 3 (1976) 1–22.

Straach, J., and Truemper, K. (1998), Optimization in interactive expert systems, working paper, University of Texas at Dallas, 1998.

Straach, J., and Truemper, K. (1998), Learning to ask relevant questions, working paper, University of Texas at Dallas, 1998.

Truemper, K. (1998), *Effective Logic Computation*, Wiley, New York, 1998.

Wrathall, C. (1976), Complete sets and the polynomial-time hierarchy, *Theoretical Computer Science* 3 (1976) 23–33.