Satisfiability Data Mining for Binary Data Classification Problems

Fred Glover
University of Colorado
Boulder, Colorado 80309-0419

fred.gover@colorado.edu

April 8, 2008

Abstract

Satisfiability Data Mining (SAT-DM) is a new method for binary data classification problems, based on generating a collection of logical clauses, or equivalently a collection of inequalities in zero-one variables, for each group of points representing a given classification. A point with unknown membership is classified as belonging to a particular group based on comparing the number or proportion of the inequalities it satisfies for that group versus the number or proportion it satisfies for other groups. We make use of a fundamental observation which states that inequalities are satisfied by a subset of elements of a particular group (and correspondingly violated by a subset of elements from a complementary group) if and only if these inequalities correspond to feasible solutions to a special variant of a satisfiability problem. Based on this, we propose a method for generating membership-defining systems of inequalities that provide a filter for segregating points lying in different groups.

Satisfiability data mining may be viewed as a procedure for generating multiple hyperplanes that segregate points of different groups by isolating their logical properties. The inequalities produced by SAT-DM capture classification regions in feature space that are more varied and complex than those derived from hyperplane separating procedures such as those used in support vector machines (SVMs) and related procedures based on linear programming and convex analysis. A particularly useful feature is the ability to generate the collections of segregating inequalities (complementary half-spaces) in a highly efficient manner, allowing the approach to handle large data sets without difficulty. The underlying processes can also be used for feature selection, or more generally attribute selection, to isolate a subset of attributes from large data sets that yield a high classification power while reducing the time and complexity of classification.

1. Introduction

We introduce a *Satisfiability Data Mining* (SAT-DM) approach for classification and feature selection in applications involving data vectors of binary valued attributes. The problem addressed is as follows. Let G_k, $k \in K$, denote a collection of groups, each consisting of a set of data points (vectors) $x^i = (x_1^i, \ldots, x_n^i)$, for $i \in G_k$, where the components x_j^i, $j \in N = \{1,\ldots,n\}$ of each vector x^i are binary (0 or 1). We seek a rule for determining the membership of a binary vector $x^o = (x_1^o, \ldots, x_n^o)$ in one of the groups G_k, $k \in K$, in order to establish which of these groups contains data points that x^o is "most like" or has "the most in common with". Example applications include diagnosing patients for disease, determining membership of biological organisms among groups exhibiting certain properties, classifying chemical compounds according to anticipated behaviors or functions, classifying investments for profitability, classifying drugs for their efficacy in treating specific conditions, and many others (see, e.g., Schlkopf and Smola, 2002; Dai, 2004; Wang, 2005).

Satisfiability data mining may be viewed from the perspective of logical (Boolean) analysis to consist of a method for generating clauses C_k in disjunctive normal form for each group G_k, subject to the condition that each clause in C_k is satisfied (i.e., is true) for all or a specified fraction $f|G_k|$ of the elements x^i, $i \in G_k$, where $1 \geq f > 0$. Then x^o is assigned to the group G_k such that minimizes an evaluation $E_k(x^o)$ measuring the degree to which x^o violates the truth conditions of the clauses in C_k, as where $E_k(x^o)$ identifies the number or proportion of clauses of C_k for which x^o is false. We generate the clauses implicitly rather than explicitly, by instead generating sets of inequalities in the binary variables x_j that are satisfied if and only if the associated clauses are satisfied.

The mechanism that enables these inequalities to be obtained efficiently is a one-one correspondence between the inequalities and feasible solutions to an associated covering/anti-covering satisfiability problem. By drawing on this relationship, we provide surrogate constraint algorithms for generating systems of inequalities that are particularly effective for differentiating among points that belong to different groups, leading to more compact and economical sets of inequalities for determining membership.

2. Inequalities Representing Logical Clauses

We begin by reviewing useful connections between logical clauses and associated systems of inequalities. With each component x_j^i of x^i, $j \in N = \{1,\ldots n\}$, we associate a term x_j that can be interpreted either as a logical statement about the value x_j^i or as a corresponding Boolean (binary valued) variable associated with x_j^i. From a logical frame of reference, we take x_j to be the statement "$x_j^i = 1$" and the negation $\tilde{x}_j$ of x_j to be the statement "$x_j^i = 0$." For example, if $x^1 = (1\ \ 1\ \ 0\ \ 0)$ and $x^2 = (0\ \ 1\ \ 0\ \ 1)$, then the disjunctive clause $x_1 \cup \tilde{x}_2$ (where "$\cup$" represents "or") is true for x^1 but not x^2. By the usual association between logical statements and Boolean variables, the logical statement x_j is represented by the Boolean assignment "$x_j = 1$" and the negation $\tilde{x}_j$ is represented by "$x_j = 0$," or equivalently by "$\tilde{x}_j = 1$" where $\tilde{x}_j = 1 - x_j$. By means of the correspondence

between the logical operator "$\cup$" and the arithmetic operator "+", the clause $x_1 \cup \tilde{x}_2$ corresponds to the Boolean inequality $x_1 + \tilde{x}_2 \geq 1$ (which, like the clause, is satisfied by x^1 and not by x^2.)

To represent more general clauses and their associated inequalities, let N_1 refer to a subset of N associated with statements denoted by x_j (equivalently, with assignments $x_j = 1$) and N_0 refer to a subset of N associated with statements denoted by $\tilde{x}_j$ (equivalently, with assignments $\tilde{x}_j = 1$). Then we are interested in identifying clauses expressed as logical disjunctions, taking the form

$$(\cup(x_j: j \in N_1)) \cup (\cup(\tilde{x}_j: j \in N_0)) \qquad (1.0)$$

Our first goal is to identify collections of clauses (1.0) that can be used to characterize elements of a particular group, which we will denote by G_α, and whose corresponding data points are represented by $\{x^i: i \in G_\alpha\}$. For example, if $x^1 = (1\ \ 1\ \ 0\ \ 0)$ and $x^2 = (0\ \ 1\ \ 0\ \ 1)$ are members of G_α, then we are interested in clauses that are true for both of them.[1] (The clauses $x_1 \cup x_2$ and $x_1 \cup x_4$ both work in this case, and $x_1 \cup x_2$ can be reduced to the more compact clause consisting of the single statement x_2.)

In terms of Boolean variables, we seek to generate inequalities of the form

$$\sum(x_j: j \in N_1) + \sum(\tilde{x}_j: j \in N_0) \geq 1 \qquad (1.1)$$

where, correspondingly, the initial goal is to identify instances of (1.1) that are satisfied by all or a usefully large portion of the elements x^i in G_α.

Going a step farther, we are interested in clauses (1.0) and hence inequalities (1.1) that are not only satisfied by elements of G_α, but that are violated by elements of another group G_β. That is, while we want elements of G_α to satisfy (1.1), we want elements of G_β instead to satisfy

$$\sum(x_j: j \in N_1) + \sum(\tilde{x}_j: j \in N_0) \leq 0 \qquad (1.2)$$

A natural choice is to designate G_β to be the complement of G_α given by $G_\beta = \cup(G_k: k \in K) - G_\alpha$. Then if an inequality (1.1) is satisfied by all elements of G_α and if (1.2) is satisfied by all elements of its complement G_β (or equivalently, if (1.1) is violated by all elements of G_β), we have a useful means for classifying new points x^o of unknown membership, by designating them to lie within or not within G_α according to whether or not they satisfy (1.1) or (1.2). In sum, our goal is to produce inequalities that include as many points as possible of G_α and exclude as many points as possible of G_β. We summarize the form of our procedure as follows.

[1] We sometimes refer to vectors as elements of groups, as a shorthand for saying that their indexes belong to the indicated sets; for example, speaking of a vector as belonging to G_α as a shorthand for saying that it belongs to the set $\{x^i: i \in G_\alpha\}$.

Summary of Method

1. Generate a collection of clauses C_k, expressed as a system of inequalities in binary variables, for each group G_k, $k \in K$. The inequalities for G_k take the form of (1.1) and (1.2), by letting $G_\alpha = G_k$ and G_β = the complement of G_k.
2. The collection of inequalities for G_k is constructed by generating solutions to an associated satisfiability problem. Each solution generated translates into an inequality that becomes a filter for differentiating points of G_k from points of other groups, and the resulting inequality captures a combination of properties of the points in G_k that differ from those captured by other inequalities in the collection.
3. The classification of a point x^o of unknown membership as belonging to G_k is determined by comparing an evaluation $E_k(x^o)$ (derived from C_k) with corresponding evaluations $E_h(x^o)$ for other groups G_h, $h \in K - \{k\}$.

The method we propose has an additional important property. Simultaneously with generating the collection C_k, $k \in K$, the method operates as a feature selection approach, isolating a subset of data features that by themselves are sufficient for defining effective instances of the collection, and that in general provide a more robust classification with reduced risk of over fitting.

Links to Other Work in Logic Applied to Data Mining and in Zero-One Optimization

An important body of work in applying logic to data mining is represented by the "Logical Analysis of Data (LAD)" approach of Boros et al. (2000a, 2000b). The LAD approach similarly generates binary inequalities by reference to clauses as in classical satisfiability problems. These inequalities are related to the canonical 0-1 cuts for integer programming in Balas and Jeroslow (1972) and strengthened for mixed integer programming by Glover (2008).

Our approach adopts a different perspective than employed in LAD, by drawing on surrogate constraint analysis and metaheuristic guidance to generate the 0-1 inequalities. By this means it becomes possible to isolate selected subsets of the inequalities that are both parsimonious and highly effective, without having to generate a potentially vast set of alternatives and then to try retrospectively to discover which inequalities may prove more desirable.

A Connection with Hyperplane Separation Models

The widely used hyperplane separation models, proposed in various forms and more recently included among support vector machine (SVM) procedures, exemplify the theme of seeking inequalities that separate the space into disjoint regions with the goal of including as many points of a given group as possible within one of the regions and as many points of an alternative group in another. (See, e.g., Christiani and Shawe-Taylor, 2000; Schlkopf and Smola, 2002; Glen, 2003; Glover, 2006.)

Our generation of the pair of inequalities (1.1) and (1.2) can be viewed from the standpoint of these hyperplane separation models as corresponding to generating the hyperplane

$$\sum(x_j: j \in N_1) + \sum(\tilde{x}_j: j \in N_0) = .5$$

Then we may classify a point as belonging to G_α if it lies on the "> .5 side" of the hyperplane or as belonging to G_β if it lies on the "< .5 side". Since we deal only in binary vectors of data points, and the coefficients of the inequality are all integers, the "> .5 side" is the same as (1.1) and the "< .5 side" is the same as (1.2).

However, our approach differs from the customary hyperplane separation approaches in three main ways, previously intimated: (i) The inequality pair (1.1) and (1.2) is produced by logical analysis (as opposed, for example, to statistical analysis or linear programming); (ii) We generate not a single inequality pair but many of them – a collection for each group G_α (or each pair G_α, G_β of interest, if G_β is not simply the complement of G_α); (iii) The classification of a point x^o is made according to evaluations x^o receives from each collection of inequalities (related to the number and proportion of the inequalities (1.1) it satisfies for each group G_k in the role of G_α), instead of basing the classification relative to a single hyperplane.

There exist some special variations of hyperplane separation approaches that share features in common with our approach. In particular, in connection with the theme of (ii), classification procedures that utilize multiple separating hyperplanes, notably including hyperplanes generated by tree-based analysis, have been proposed in Glover (1990), Bennet and Blue (1998), Glen (2003), Better, Glover and Samorani (2006) and Glover (2006). The present framework can also be used in a natural way within the tree-based analysis approaches, giving another way of exploiting the inequalities (1.1) and (1.2). Details of how this may be accomplished are given in Section 10.

3. Fundamental Relationships and the Quasi Covering/Anti-Covering System

As a foundation for algorithms subsequently described, we establish a useful connection between inequalities of the form of (1.1) and (1.2) and feasible solutions to another set of inequalities. In the same way that $\tilde{x}_j$ identifies the complement of the variable x_j, let $\tilde{x}_j^i$ denote the complement of the binary constant x_j^i, that is, $\tilde{x}_j^i = 1 - x_j^i$. We introduce a binary variable y_{1j} associated with each x_j^i and a binary variable y_{0j} associated with each $\tilde{x}_j^i$. Then we define the *Quasi Covering/Anti-Covering System* (QC/AC) as follows:

(QC/AC):

$$\sum(x_j^i\, y_{1j}: j \in N) + \sum(\tilde{x}_j^i\, y_{0j}: j \in N) \geq 1 \quad i \in G_\alpha \qquad (2.1)$$

$$\sum(x_j^i\, y_{1j}: j \in N) + \sum(\tilde{x}_j^i\, y_{0j}: j \in N) \leq 0 \quad i \in G_\beta \qquad (2.2)$$

$$y_{1j} + y_{0j} \leq 1 \quad j \in N \qquad (2.3)$$

$$y_{1j}, y_{0j} \in \{0,1\} \quad j \in N \qquad (2.4)$$

The Quasi Covering/Anti-Covering term comes from the fact that (2.1) defines a collection of covering inequalities, and (2.2) defines a collection of what may be called anti-covering inequalities. The inclusion of (2.3) removes both of these components from strictly belonging to a covering or anti-covering category, and hence we use the term "quasi" to make this distinction.

Fundamental Observation: Let G_α^* be any subset of G_α and let G_β^* be any subset of G_β. The inequality (1.1) is valid for all x^i, $i \in G_\alpha^*$ and the inequality (1.2) is valid for all x^i, $i \in G_\beta^*$ if and only if N_1 and N_0 correspond to a feasible solution y_{1j}^*, y_{0j}^*, $j \in N$, to (QC/AC) by the relationship

$$y_{1j}^* = 1 \leftrightarrow j \in N_1 \text{ and } y_{0j}^* = 1 \leftrightarrow j \in N_0$$

The significance of this observation is that we can pursue the goal of generating inequality pairs (1.1) and (1.2) that will respectively be satisfied by large subsets G_α^* of G_α and G_β^* of G_β by seeking feasible solutions to the system (QC/AC) that correspondingly satisfy large numbers of the inequalities (2.1) and (2.2). The (QC/AC) system, by reference to the Fundamental Observation, allows us to produce pairs of such inequalities (1.1) and (1.2) that are useful for the purpose of creating a collection to differentiate points of G_α from those of G_β.

We will show how it is possible to exploit the foregoing observation to generate a collection of inequalities of the form (1.1) and (1.2) given by

$$\sum(x_j: j \in N_1(p)) + \sum(\tilde{x}_j: j \in N_0(p)) \geq 1 \quad p \in P \qquad (1.1P)$$
$$\sum(x_j: j \in N_1(p)) + \sum(\tilde{x}_j: j \in N_0(p)) \leq 0 \quad p \in P \qquad (1.2P)$$

Assume that having produced (1.1P) and (1.2P) we identify the subsets $G_\alpha^*(p)$ of G_α and $G_\beta^*(p)$ of G_β for each $p \in P$ such that all points x^i for $i \in G_\alpha^*(p)$ and all points x^i for $i \in G_\beta^*(p)$ satisfy the p^{th} instances of (1.1P) and (1.2P). For this particular p, it is desirable to have (1.1P) be a prime implicant for points x^i, $i \in G_\alpha^*(p)$ – that is, to have (1.1P) be irreducible (non-redundant) in the sense that neither of the sets $N_1(p)$ or $N_0(p)$ can be replaced by a proper subset and still allow the associated instance of (1.1P) to be satisfied by all of these points. The reduction of an inequality to a prime implicant is also desirable from the standpoint of feature selection, since it also reduces the number of features (indexes j) the indicated inequality relies on.

More generally, in addition to reducing each instance of a collection (1.1P) to become a prime implicant, it is of further value from the standpoint of feature selection to generate inequalities that minimize the size of $N_1(p) \cup N_0(p)$. More broadly, we give preference to smaller $N_1(p) \cup N_0(p)$ sets subject to the goal of producing inequalities (1.1P) and (1.2P) that are respectively satisfied by large numbers of points in within G_α and G_β. This goal translates into seeking a solution y_{1j}^*, y_{0j}^*, $j \in N$ to the system (QC/AC) that assigns $y_{1j}^* = 1$ and $y_{0j}^* = 1$ to a relatively small number of variables. Consequently, we are interested in solutions that achieve the secondary (approximate) goal of minimizing the sum of the y_{1j} and y_{0j} variables.

These goals can be made more restrictive by seeking to strictly maximize the sum of satisfied inequalities and subject to this to strictly minimize the sum of the y_{1j} and y_{0j} variables. These restrictive goals are narrower than desirable, however, given that we want to generate not just a single inequality but a collection of inequalities. In addition, the restrictive goals entail the solution of NP hard problems and hence can consume large amounts of computation time. As will be shown, strategies are available for achieving the broader goals we have proposed that succeed in generating a collection of varied inequalities highly efficiently. Finally, the more restrictive goals can rule out the generation of inequalities that may nevertheless be valuable for the purpose of differentiating elements of group G_α from those of group G_β.

Thus, in pursuing the broader satisfiability objectives of satisfying a large number of the inequalities (1.1P) and (1.2P), and using a small number of variables to do this, we embrace the imprecision of referring any solution that achieves these objectives as a *strongly satisfying solution*. Consequently, we are interested in the following type of satisfiability problem, which we call the *Quasi Covering/Anti-Covering Satisfiability Problem*:

SAT(QC/AC): Generate a collection of strongly satisfying solutions to the problem

Minimize $\sum((y_{1j} + y_{0j}): j \in N)$

subject to

$$\sum(x_j^i\, y_{1j}: j \in N) + \sum(\tilde{x}_j^i\, y_{0j}: j \in N) \geq 1 \quad i \in G_\alpha \qquad (2.1)$$

$$\sum(x_j^i\, y_{1j}: j \in N) + \sum(\tilde{x}_j^i\, y_{0j}: j \in N) \leq 0 \quad i \in G_\beta \qquad (2.2)$$

$$y_{1j} + y_{0j} \leq 1 \quad j \in N \qquad (2.3)$$

$$y_{1j}, y_{0j} \in \{0,1\} \quad j \in N \qquad (2.4)$$

While imprecise in its specification of a strongly satisfying solution, the formulation of SAT(QC/AC) leads to a procedure that achieves its purposes quite effectively. Moreover, the objective of minimizing the sum of the y_{1j} and y_{0j} variables has the bonus of reinforcing the pursuit of the objectives of satisfying large numbers of the constraints (2.1) and (2.2). In fact, we show that it is possible to generate solutions that embody an asymmetric form of strong satisfiability by satisfying all inequalities of (1.1P), and demonstrate that this asymmetric form is valuable for separating G_k from other groups.

4. The Quasi Covering Problem and Surrogate Constraint Strategies

As a first step toward producing strongly satisfying solutions for SAT(QC/AC), it is useful to consider the simplified situation where G_β is chosen to be the empty set. By this choice, we are interested in generating inequalities that are satisfied by as many points of G_α as possible, without being concerned about separating these points from those of another set (such as the complement of G_α). Under the assumption $G_\beta = \varnothing$, we can disregard the inequalities (1.2P) and focus only on the quasi covering portion of the problem SAT(QC/AC), dropping the inequalities (2.2) associated with (1.2P). We call this simplified problem the *Quasi Covering Satisfiability Problem*, expressed as follows.

SAT(QC): Generate a collection of strongly satisfying solutions to the problem

Minimize $\sum((y_{1j} + y_{0j}): j \in N)$

subject to

$$\sum(x_j^i\, y_{1j}: j \in N) + \sum(\tilde{x}_j^i\, y_{0j}: j \in N) \geq 1 \quad i \in G_\alpha \qquad (2.1)$$

$$y_{1j} + y_{0j} \leq 1 \quad j \in N \qquad (2.3)$$

$$y_{1j}, y_{0j} \in \{0,1\} \quad j \in N \qquad (2.4)$$

The constraints of this problem, which again for convenience are given the same labeling as in the (QC/AC) system, correspond to those of SAT(QC/AC) except that the constraint corresponding to (2.2) is removed.

An ability to generate good solutions to SAT(QC) is a cornerstone of generating good solutions to SAT(QC/AC) for the following reason. Every assignment of values to the y_{1j} and y_{0j} variables, regardless of whether the problem considered is SAT(QC/AC) or SAT(QC), gives rise to the pair of inequalities (1.1P) and (1.2P). Hence, each assignment generated for solving SAT(QC) can be evaluated relative to both problems. In particular, a solution to SAT(QC) can be evaluated for its quality as a solution to SAT(QC/AC) simply by identifying the number of points of G_β that satisfy the associated inequalities (1.2P). Evidently, from among the large number of solutions that can be evaluated for the two problems, those that are best for SAT(QC) may not be best for SAT(QC/AC). Nevertheless, a procedure for generating good solutions to the simpler problem SAT(QC) provides insights into a procedure for generating good solutions to the more complex problem SAT(QC/AC). In addition, the SAT(QC) problem is important in its own right, because applications exist where the essential goal is to capture the nature of a group G_α without concern for differentiating it from a complementary group G_β.

Surrogate Constraint Strategy

The structure of SAT(QC) makes it susceptible to exploitation by a surrogate constraint strategy (see, e.g., Glover, 1965, 1968, 2003; Greenberg and Pierskalla, 1970, 1973), whose form we briefly sketch as follows.

A surrogate constraint consists of a non-negative linear combination of a selected subset of the problem constraints, and can be expressed in the context of the SAT(QC) problem by the inequality

$$\sum(a_{1j}y_{1j}: j \in N) + \sum(a_{0j}y_{0j}: j \in N) \geq a_o \qquad (3)$$

The coefficients a_{1j}, a_{0j} and a_o are determined in this case by reference to the inequalities of (2.1); i.e., for a chosen collection of non-negative weights w_i, $i \in G_\alpha$, we have $a_{1j} = \sum(x_j^i w_i: i \in G_\alpha)$, $a_{0j} = \sum(\tilde{x}_j^i w_i: i \in G_\alpha)$ and $a_o = \sum(w_i: i \in G_\alpha)$ (We do not include reference to constraints other than those of (2.1) in forming the surrogate constraint, because the requirements of these other constraints are handled as part of the approach for exploiting (3).)

A straightforward surrogate constraint strategy for SAT(QC) arises by successively selecting variables y_{1j} and y_{0j} to set to 1 (thus implicitly setting y_{vj} to 0 if y_{vj} is set to 1, v = 0 or 1), until enough variables have been selected to satisfy all or a desired portion of the inequalities of (2.1). All remaining unselected variables are given values of 0. The surrogate constraint (3) plays a central role in the process by providing the basis for choosing the specific y_{1j} and y_{0j} variables to be assigned values of 1.

A commonly used rule for selecting a variable to set to 1 in covering problems translates in the present case to picking a variable y_{1j} or y_{0j} that has the largest coefficient a_{1j} or a_{0j} in the surrogate constraint (3). This simply chooses the variable that makes the best contribution toward satisfying the surrogate constraint (motivated by the fact that we seek to minimize the sum of the variables).

Once the variable y_{vj} is selected, for v = 1 or 0, then the assignment $y_{vj} = 1$ is plugged into SAT(QC). The problem thus shrinks by removing y_{vj} and $y_{(1-v)j}$ from further consideration, and removing all component inequalities of (2.1) that are thereby satisfied. The remaining variables and component inequalities of (2.1) then generate a new surrogate constraint (3) derived from the reduced set G_α (and defined over the reduced N). The process repeats until all inequalities of (2.1) are satisfied or until N becomes empty. In either case, the currently generated solution satisfies all inequalities of SAT(QC) that have been removed by shrinking the index set G_α.

In keeping with the motive of producing a procedure that is both easy and fast to execute, we generate the surrogate constraint (3) by the *surrogate sum* method, which chooses all weights of the linear combination to be 1, and hence generates a_{1j} and a_{0j} as the sum of the coefficients x_j^i and $\tilde{x}_j^i$ in (2.1); i.e.,

$$a_{1j} = \sum(x_j^i: i \in G_\alpha) \text{ and } a_{0j} = \sum(\tilde{x}_j^i: i \in G_\alpha).$$

The surrogate sum rule has been used in a variety of successive assignment procedures for generating good solutions to covering problems (e.g., Chvatal, 1979; Balas and Ho, 1980; Feo and Resende, 1989; Caprara, Fischetti and Toth, 1999; Yagiura, Kishida and Ibaraki, 2006).[2] Within the context of the SAT(QC) problem we have an added motivation for using it. It is easy to see that a_{1j} and a_{0j} identify the number of component inequalities of (2.1) that will be satisfied by setting y_{1j} or $y_{0j} = 1$, respectively. Consequently, the surrogate sum rule pursues the objective of maximizing the number of inequalities that are satisfied while simultaneously pursuing the objective of minimizing the sum of the y_{1j} and y_{0j} variables. This synergy between the objectives underlying the definition of a strongly satisfying solution, when the objectives are pursued by a surrogate constraint strategy, is a useful side benefit of our approach.

[2] However, weights for the constraints other than all 1's have proved superior in a number of applications; see, e.g., Gavish and Pirkul, 1985; Gavish, Glover and Pirkul, 1991; Freville and Plateau, 1993; Lokketangen and Glover, 1997; Albanedo and Rego, 2005. The Appendix gives another way to choose such weights.

We next examine how the surrogate constraint strategy can be used to generate a single solution to SAT(QC) and hence a corresponding inequality of the form (1.1P). Afterward we show how to extend the process to generate multiple solutions for the purpose of generating a collection of strongly satisfying solutions to SAT(QC), and hence to produce a collection of useful inequalities (1.1P).

5. Creating a Single Inequality via SAT(QC)

The method we identify for generating a single solution to SAT(QC) constitutes the core process of our satisfiability data mining procedure. To give this process an efficient structure, we work with an implicit rather than an explicit representation of the problem SAT(QC). Let N^o and $G_\alpha{}^o$ denote the original form of N and G_α, so that the process starts from $N = N^o$ and $G_\alpha = G_\alpha{}^o$. The operation of implicitly assigning values to variables y_{1j} and y_{0j}, which results in removing elements from N and G_α at each iteration, causes the current N to contain the indexes of variables not yet assigned and G_α to contain the indexes of inequalities (3 .1) not yet satisfied. Indexes from G_α are added to a set $G_\alpha{}^*$ (which begins empty) so that $G_\alpha{}^*$ identifies the inequalities of (2.1) that are satisfied at each stage; i.e., $G_\alpha{}^* = G_\alpha{}^o - G_\alpha$. The current surrogate sum coefficients a_{1j} and a_{0j}, which result by summing over constraints of (2.1) that remain, can be determined by identifying a_{1j} (respectively, a_{0j}) as the number of remaining row vectors x^i, $i \in G_\alpha$ such that $x_j{}^i = 1$ (respectively, $x_j{}^i = 0$). Including a_o we therefore have

$$a_{vj} = |\{i \in G_\alpha: x_j{}^i = v\}|, \text{ for } v \in \{0,1\} \text{ and } a_o = |G_\alpha|.$$

The inequality (1.1) (the single instance of a collection (1.1P)) is generated by constructively building its associated index sets N_1 and N_0, starting with N_0 and N_1 empty. Then at each iteration, the surrogate sum rule is used to pick the index $v^* \in \{0,1\}$ and the index $j^* \in N$ that yields

$$a_{v^*j^*} = \text{Max}(a_{vj}: v \in \{0,1\}, j \in N).$$

This implicitly corresponds to choosing $y_{v^*j^*}$ as the variable to set to 1 on the current iteration. Accordingly, the index j^* is added to N_1 if $v^* = 1$ and is added to N_0 if $v^* = 0$. The sets N and G_α are updated by setting

$$N := N - \{j^*\} \text{ and } G_\alpha := G_\alpha - \{i: x_{j^*}{}^i = v^*\}.$$

If it is desired for the resulting inequality (1.1) to be valid for all points x^i in the original set $G_\alpha{}^o$, the process can continue to choose additional v^* and j^* values until (2.1) is fully satisfied, i.e., until G_α shrinks to become empty. However, to give the procedure greater flexibility, we instead allow the procedure to terminate once it has generated an

inequality (1.1) that is satisfied by a specified positive fraction $f \leq 1$ of the elements of G_α^o. This translates into the condition $|G_\alpha^*| \geq f|\, G_\alpha^o\,|$.[3]

Drawing on the preceding observations, we summarize the SAT-DM Core method to generate a single inequality as follows.

SAT-DM Core Method.
0. Begin with $G_\alpha = G_\alpha^o$, $G_\alpha^* = \varnothing$, $N = N^o$, and $N_0 = N_1 = \varnothing$.
1. Identify $v^* = 0$ or 1 and $j^* \in N$ such that
   $(v^*, j^*) = \arg\max(a_{vj}: v \in \{0,1\}, j \in N)$
   (i.e., $a_{v^*j^*} = \text{Max}(a_{vj}: v \in \{0,1\}, j \in N)$)
2. Set $N_{v^*} := N_{v^*} \cup \{j^*\}$, $N := N - \{j^*\}$, $G_\alpha^* := G_\alpha^* \cup \{i \in G_\alpha: x_{j^*}^i = v^*\}$ and $G_\alpha := G_\alpha - \{i \in G_\alpha: x_{j^*}^i = v^*\}$.
3. If $|G_\alpha^*| \geq f|\, G_\alpha^o\,|$ or $N = \varnothing$ proceed to Step 4. Otherwise, determine the updated values $a_{vj} = |\{i \in G_\alpha: x_j^i = v\}|$, for $v \in \{0,1\}$, $j \in N$, and return to Step 1.
4. Generate the inequality (1.1), satisfied by all $x = x^i$ for $i \in G_\alpha^*$, given by
$$\sum(x_j: j \in N_1) + \sum(\tilde{x}_j: j \in N_0) \geq 1 \qquad (1.1^*)$$
End of Core Method

We have represented the instance of (1.1) generated in Step 4 by (1.1*) to emphasize that it has been generated in association with the set G_α^* in the Core Method. This inequality may not be a prime implicant (i.e., it may not be undominated), and can be readily checked to see if tightening is possible. An undominated instance can be produced by dropping any element $j \in N_1$ or $j \in N_0$ whose removal will still allow (1.1*) to be satisfied by all elements of G_α^* (or by $f|G_\alpha^o\,|$ elements of the original G_α). The process then repeats until no elements can be dropped. This tightening approach can be reiterated by using simple memory analogous to that specified in the next section, as a way to identify additional undominated inequalities if more than one is contained within the present (1.1*). In addition, in Section 8 we give a process associated with the method for SAT(QC/AC) that generates a special instance of a prime implicant that is relevant for that problem.

Remark 1. In applying the Core Method, it is unnecessary to generate or record the problem matrix associated with the variables y_{0j}, $j \in N$, since each coefficient $\tilde{x}_j^i$ is known automatically from the corresponding coefficient x_j^i.

It will be seen that the organization of the Core Method implicitly takes this remark into account by means of the observation $a_{vj} = |\{i \in G_\alpha: x_j^i = v\}|$, for $v \in \{0,1\}$. Thus, the method makes no reference to the coefficients $\tilde{x}_j^i$.

[3] Nevertheless, we will argue that $f = 1$ is often desirable. An analogous condition, which may be viewed as determining a value for f that can change from one inequality to another, is determined adaptively in the more general procedure we subsequently specify for SAT(QC/AC).

As an accompaniment to Remark 1, we can also accelerate the determination of the surrogate constraint coefficients a_{vj} by computing them only for v = 1, as noted next.

Remark 2. The value of a_{0j} is given by $a_{0j} = a_o - a_{1j}$ (and, correspondingly, $a_{0j} = a_o - a_{1j}$).

These remarks disclose that the generation and examination of the coefficients a_{vj} in the Core Method should proceed by immediately examining the index pair (0,j) after the pair (1,j) rather than first examining all pairs (1,j) and then all pairs (0,j). Note that computation can be further streamlined by combining the update of the coefficients a_{vj} in Step 3 with the subsequent operation of identifying a_{v*j*} in Step 1.

We later make additional remarks that can improve the efficiency of the procedures we propose and isolate preferable instances of the inequalities (1.1).

6. Generating Multiple Inequalities

The SAT-DM Core Method can be extended to generate multiple inequalities by introducing a simple memory structure to oversee the process. Let n(1:j) and n(0:j) identify the number of inequalities (1.1P) of Section 3 in which j is added to N_1 and to N_0, respectively. An initialization step that precedes Step 0 sets n(v,j) = 0 for all $j \in N$, $v \in \{0,1\}$,. At the conclusion of Step 4, the n(v,j) values are updated by setting n(v,j) := n(v,j) + 1 for each $j \in N_v$, $v \in \{0,1\}$. For the indexed collection of inequalities (1.1P) generated, we refer to N_v as $N_v(p)$ fov v = 0 and 1.

We stipulate that each instance of (1.1P) must contain at least one $j \in N_v(p)$, $v \in \{0,1\}$, such that n(v,j) = 0, thus automatically assuring every instance will be different. Let L denote a user-selected limit on the number of inequalities generated. Then, multiple inequalities can by generated by the following two modifications of the Core Method.

(A) The method returns to Step 0 after each execution of Step 4, as long as n(v,j) = 0 for at least one pair v, j such that $j \in N$, $v \in \{0,1\}$, and as long as fewer than L inequalities have been generated,

(B) Each time Step 1 is visited immediately after Step 0 (to select the first v* and j* for the new inequality (1.1P)), we additionally require n(v*,j*) = 0, and the method terminates once this condition cannot be met on such a "first execution" of Step 1. Thus, on visiting Step 1 immediately after Step 0, the value v* and the index j* are selected by the rule

$$(v^*,j^*) = \arg\max(a_{vj}: v \in \{0,1\}, j \in N, n(v,j) = 0)$$

There is one further element to consider in the present case. If on some execution of Step 1 for iter = 1 we have $a_{v*j*} = 0$, this implies that choosing v*, j* (implicitly setting $y_{v*j*} = 1$) can not satisfy any of the inequalities (2.1) for $i \in G_\alpha^o$. In particular, $x_{j*}^i = 0$ for all $i \in G_\alpha^o$ if v* = 1 and $\tilde{x}_{j*}^i := 0$ for all $i \in G_\alpha^o$ if v* = 0. Consequently, this choice of v* and j* makes no contribution and can be disregarded. Since no other choice for v* and j* can do better, the procedure can terminate at this point. The condition $a_{v*j*} = 0$ cannot occur if

iter > 1, since the fact that x_j^i or $\tilde{x}_j^i = 1$ for every i implies either a_{1j} or a_{0j} must be positive for every j.

For additional control, the n(1:j) and n(0:j) values can also be constrained not to exceed some specified limit in subsequent iterations of Step 1, in order to assure that particular variables do not appear a disproportionate number of times in the inequalities generated.

We summarize these observations in the following method. The former Step 0 for the SAT-DM Core Method has been moved outside the procedure to become Step 1.0 of the more general method.

Basic SAT-DM Method for Generating Multiple Inequalities.

0.0 (Initialization) Select a limit L on the number p of inequalities to generate. Set p = 0, n(v,j) = 0, $v \in \{0,1\}$, $j \in N^o$.

1.0 (Prepare for the Core Method.) Set $G_\alpha = G_\alpha^o$, $G_\alpha^* = \varnothing$, $N = N^o$, $N_0 = N_1 = \varnothing$. and iter = 0.

SAT-DM Core Method (to Generate a Single Inequality).

1. Set iter := iter + 1. Identify $v^* = 0$ or 1 and $j^* \in N$ such that
   If iter = 1: $(v^*,j^*) = \arg\max(a_{vj}: v \in \{0,1\}, j \in N, n(v,j) = 0))$. If $a_{v^*j^*} = 0$, stop.
   If iter > 1: $(v^*,j^*) = \arg\max(a_{vj}: v \in \{0,1\}, j \in N)$
2. Set $N_{v^*} := N_{v^*} \cup \{j^*\}$, $N := N - \{j^*\}$, $G_\alpha^* := G_\alpha^* \cup \{i \in G_\alpha: x_{j^*}^i = v^*\}$ and $G_\alpha := G_\alpha - \{i \in G_\alpha: x_{j^*}^i = v^*\}$.
3. If $|G_\alpha^*| \geq f|G_\alpha^o|$ or $N = \varnothing$ proceed to Step 4. Otherwise, determine the updated values $a_{vj} = |\{i \in G_\alpha: x_j^i = v\}|$, for $v \in \{0,1\}$, $j \in N$, and return to Step 1.
4. Set p := p + 1 and, after reduction to a prime implicant, generate the associated instance of the inequality (1.1P), satisfied by all $x = x^i$ for $i \in G_\alpha^*$, given by

$$\sum(x_j: j \in N_1(p)) + \sum(\tilde{x}_j: j \in N_0(p)) \geq 1 \qquad (1.1(p))$$

   Set n(v,j) := n(v,j) + 1 for each $j \in N_v(p)$, $v \in \{0,1\}$ for each $j \in N_1(p) \cup N_0(p)$.

End of Core Method

2.0 If p = L, or if $|G_\alpha^*| < f|G_\alpha^o|$ (implying $N = \varnothing$), or if n(v,j) > 0 for all $j \in N$ and for both v = 0 and 1, then stop. Otherwise return to Step 1.0.

End of Method

The instance of (1.1P) generated in Step 4 is represented as (1.1(p)) to emphasize its association with a particular index p at that step. Under the stopping condition $|G_\alpha^*| < f|G_\alpha^o|$ of Step 2.0, the last inequality generated can be dropped, since it does not succeed in being satisfied by at least $f|G_\alpha^o|$ elements of G_α^o.

7. Improved Method and Feature (or Attribute) Selection

The observations at the conclusion of Section 5 concerning ways to reduce computation of the Core Method are likewise applicable to the Basic SAT-DM Method. We make

additional observations to improve the method at a similar rudimentary level and then examine improvements at a deeper level.

Remark 3. When an iteration of the Core Method occurs for $a_{1j} = a_o$ or 0 (hence $a_{oj} = a_o$) for some $j \in N$, the method can directly identify all pairs (v^*,j^*) such that $a_{v^*j^*} = a_o$, and augment the collection of inequalities (1.1(p)) by implementing Step 4 for each of these pairs (v^*,j^*) independent of the others (skipping Steps 2 and 3, except for setting $N_{v^*} := N_{v^*} \cup \{j^*\}$ separately for each pair (v^*,j^*)).

The condition $a_{1j} = a_o$ or 0 identified in Remark 3 is the condition that causes Step 4 to be visited (after updating for selecting (v^*,j^*)) as a result of satisfying $|G_\alpha^*| \geq f| G_\alpha^o |$ for $f = 1$, which may also be expressed as $G_\alpha^* = G_\alpha^o$ (or $G_\alpha = \varnothing$). The remark observes that the condition can be spotted in a simple way before updating and that a number of inequalities may possibly be produced from it at once rather than generating only one inequality at Step 4 at a time. A special case of Remark 3 can provide further computational savings.

Remark 4. If the condition $a_{1j} = a_o$ or 0 occurs on the first iteration of Step 1, when iter = 1 and p = 0, then after creating all of the inequalities indicated in Remark 3, each index j* that belongs to one of the pairs (v*, j*) can be permanently excluded from the set N in all subsequent iterations of the method.

The next special case embraces a potentially larger set of conditions.

Remark 5. If the condition $a_{1j} = a_o$ or 0 occurs on the second iteration of Step 1, when iter = 2 (and p has any value), then after creating all of the inequalities indicated in Remark 3, the specific index pair (v_1,j_1) that was chosen as (v^*,j^*) for iter = 1 (hence the pair such that $j^* \in N_{v^*}$ before the updates of iter = 2) can permanently be dropped from consideration on all future iterations.

The justification of this remark follows from the fact that the structure of the Basic SAT-DM method assures that all undominated inequalities that include (v_1, j_1) have been generated by the method using the approach of Remark 3.

Remark 5 raises an interesting point. The Basic SAT-DM method will automatically disregard the pair (v_1,j_1) whenever iter = 1 due to the use of the values n(v,j), but Remark 5 goes farther by stipulating that (v_1,j_1) can be disregarded for values of iter > 1 as well. Nevertheless, the memory embodied in the n(v,j) values may be unduly restrictive in spite of the ability to override it in this special case, because it can exclude many inequalities that may be desirable to generate.

A telling limitation of this memory structure comes from the fact that it operates in opposition to the goals of feature selection, where we seek a subset N′ containing features $j \in N$, typically much smaller than N itself, and classify new points by limiting consideration to N′. The organization of the Basic SAT-DM method, which forces a new

pair (v,j) to be introduced at each first step (iter = 1) of generating a new inequality, evidently tends to work against the goals of feature selection.

It is possible to get around this limitation by applying the foregoing method within the framework of a tree search memory, and thereby generate all sequentially undominated inequalities that can be derived as feasible solutions to the constraints of SAT(QC). ("Sequentially undominated" is defined in the natural way as an inequality that cannot be reduced if the sequence of introducing its terms is fixed, i.e., if the pair (v*,j*) added at a given iteration must require the inclusion of pairs add up through all previous iterations.) While such a tree search approach can readily be adapted to the present setting (see, e.g., Glover, 1965), it has three important disadvantages: (a) it generates far more inequalities than are needed: (b) the inequalities include all possible features, and hence must be significantly culled to meet the goals of feature selection; (c) the inequalities are produced in a lock-step sequence that exhausts all possible sequentially undominated inequalities for each pair (v*,j*) chosen in sequence before generating any inequalities that lack (v*,j*) in this sequence. That means the type of variation admitted by performing a truncated form of this tree search is exceedingly limited. Recall that, in addition to feature selection, we wish to generate inequalities that are parsimonious in the sense of having a small number of terms.

To combat these difficulties we introduce a more flexible memory structure based on ideas from the adaptive memory framework of tabu search. In fact, we may employ a relatively simple tabu search design that incorporates a common "recency-based" tabu list applied to choices at iter = 1, and an associated tabu restriction at iter = 2 that assures no duplications can occur. Within the modest limitations imposed by these two restrictions, the method is given a broad latitude to follow the guidance of the surrogate sum choice rule that favors parsimonious inequalities. Moreover, as will be shown, the method can be organized to pursue the goals of feature selection, enabling the feature selection problem to be handled simultaneously with the generation of inequalities for classification.

Attribute Selection Versus Feature Selection

We find it useful in the present setting not simply to seek a small subset N' of the features $j \in N$ that provides a source of effective inequalities, but look for a more refined choice by reference to *attributes* rather than features.[4] An attribute of a solution to SAT(QC/AC) or SAT(QC) is given not just by the index j, but by the pair (v,j). That is, each solution to the problems SAT(QC/AC) and SAT(QC) (and each inequality (1.1P) generated as a consequence) is uniquely determined by the set of pairs (v,j) that define the composition of $N_1(p)$ and $N_0(p)$. Accordingly, we are interested in the *attribute selection problem* that seeks a small set of attributes (v,j) that are capable of yielding strongly satisfying solutions to the inequalities (2.1) to (2.3). The imprecision of the word "small" in the feature selection literature carries over to attribute selection, but is mitigated in the

[4] In some papers on feature selection, the word "attribute" is used as synonymous with "feature." However, we differentiate the two by using the word "attribute" in the sense conveyed in memory-based search procedures (see, e.g., Glover and Laguna, 1997).

present context, as we have seen, by the fact that pursuit both of strongly satisfying solutions and solutions containing a modest number of attributes is reinforced by the surrogate constraint choice rules. This mutual reinforcement yields further useful outcomes when implemented within an appropriate memory framework.

To pursue the attribute-based perspective, let NA refer to an attribute-based form of N, consisting of ordered pairs (v,j), i.e.,

$$NA = \{(v,j): v \in \{0,1\}, j \in N\}.^5$$

and let NA(p) denote the attribute-based equivalent of $N_1(p) \cup N_0(p)$ given by

$$NA(p) = \{(v,j) \in NA: v = 1, j \in N_1(p) \text{ or } v = 0, j \in N_0(p)\}.$$

The inequality (1.1(p)) generated at Step 4 of the Basic SAT-DM method can therefore be written as

$$\sum(x_{vj}: (v,j) \in NA(p)) \geq 1$$

where we define $x_{1j} \equiv x_j$ and $x_{0j} \equiv \tilde{x}_j$.

Basic Memory Structures

As intimated, our memory structure employs two kinds of tabu lists, each consisting of selected elements of NA. The first, Tabu1, is a list of attributes (v,j) that have been selected as (v*,j*) on previous executions of Step 1 for iter = 1. The purpose of Tabu1 is to prevent the choice of the first (v*,j*) used to produce the inequality (1.1(p)) at Step 4 from duplicating previous choices. This does not by itself prevent the current (1.1(p)) from duplicating a previous (1.1(p)) (without incorporating the second tabu list), but it provides the potential to generate a variety of inequalities that are not compelled to keep the same first choice intact until all possible inequalities are generated that include this choice. Thus, Tabu1 induces a richer variety in the inequalities produced, while duplicate inequalities are avoided by including the second tabu list.

As in rudimentary tabu list constructions, Tabu1 contains only a limited number of the attributes most recently selected in Step 1. We denote this limitation of the number of elements within Tabu1 by TabuLim. In addition, from the standpoint of the attribute selection problem, let AttLim denote a limit on the number of attributes permitted to be embodied in the inequalities (1.1(p)). Thus, we seek to generate inequalities subject to the (approximate) restriction

$$|\cup(NA(q), q = 1,\ldots,p)| \leq \text{AttLim}$$

[5] An updated NA can be conveniently recorded and processed using two bit elements for each $j \in N$, $Bit_1(j)$ and $Bit_0(j)$, where $Bit_v(j) = 1$ if and only if attribute $(v,j) \in NA$.

Here p takes the value assigned at the most recent execution of Step 4, so that the indicated restriction applies to all inequalities produced up to a current execution of this step. TabuLim is related to AttLim by selecting TabuLim < AttLim (e.g., TabuLim = AttLim/2).

The second tabu list is defined relative to a specified attribute $(v',j') \in NA$, and is given by

$$\text{TabuMatch}(v',j') = \{(v,j) \in NA(q): (v',j') \in NA(q): \quad q = 1, \ldots p\}$$

In words, TabuMatch(v′,j′) consists of the attributes (v,j) that have appeared in the same inequalities as the attribute (v′,j′) on previous passes of the method. TabuMatch(v′,j′) begins empty for all pairs (v′,j′). ∈ NA, and stays empty until an inequality is generated that contains (v′,j′) and at least one additional attribute. This particular tabu list is updated each time a new inequality is generated at Step 4. It is used on subsequent steps of selecting (v*,j*) in Step 1 to insure that the second (v*,j*) selected during a given pass will not have appeared, together with the first (v*,j*) selected, in any inequality previously generated. This condition assures that no two inequalities can have the same composition of NA(p), and thus no two inequalities are the same (unless the first or second (v*,j*) chosen is dropped in producing a prime implicant at Step 4) .

A useful consequence of this organization is that the number of nonempty lists TabuMatch(v′,j′) will be limited to at most AttLim, and in large problems AttLim will normally be chosen to be only a small fraction of |NA|.

Exploiting the Memory Structures Efficiently

A key ingredient of an approach for SAT(QC) is to identify a way to exploit its memory structures efficiently, without requiring supporting structures that consume an excessive amount of memory. However, the goals of efficient computation and economical use of memory do not always go hand in hand. For example, a convenient way to exploit TabuMatch(v′,j′) is to create a supporting 0-1 matrix PairMatch((v_1,j_1),(v_2,j_2)), where PairMatch((v_1,j_1),(v_2,j_2)) = 1 if and only if (v_1,j_1) and (v_2,j_2) appear in the same inequality (1.1(q)) for some q = 1, …, p. However, such a matrix consumes a prohibitive amount of space when |NA| is large.

To obtain a more favorable balance between memory requirements and computational efficiency, we keep a bit-valued array TabuBit(v,j) initialized by setting TabuBit(v,j) = 0 for all (v,j) ∈ NA. The SAT-DM method can take advantage of this array by performing a *Set Bit* operation relative to the list TabuList(v*,j*) that consists of making a single pass of the list to set TabuBit(v,j) = 1 for each attribute (v,j) on this list. This allows a very fast determination of whether a given element (v,j) subsequently examined is an element of TabuList(v*,j*) simply by checking whether TabuBit(v,j) = 1 or 0. Once the use of the array TabuBit(v,j) is completed, it is restored to its initial "all 0" state by a *Re-Set Bit* operation in which one additional pass of TabuList(v*,j*) identifies the elements of TabuBit(v,j) to be changed from 1 back to 0.

An analogous process can also be used to create an efficient update of the relevant lists TabuList(v,j) that are affected each time a new inequality (1.1(p)) is generated at Step 4. For this, we exploit the fact that (v,j) ∈ TabuList(v*,j*) if and only if (v*,j*) ∈ TabuList(v,j). First the Set Bit operation is performed relative to TabuList(v*,j*) as previously indicated. Then, a pass is made of the elements (v,j) ∈ NA(p), considering only elements such that TabuBit(v,j) = 0. For each of these elements, the attribute (v*,j*) is added to TabuList(v,j) and (v,j) is added to TabuList(v*,j*). Finally, once again the Re-Set Bit operation is performed to restore TabuBit(v,j) to its all zero state, in this case referring only to the subset of TabuList(v*,j*) that was used in the Set Bit operation.

The exploitation of the Tabu1 list, to avoid re-selecting pairs (v,j) on the first iteration of Step 1 that were selected on the first iteration when generating recent previous inequalities, can be handled in a simpler fashion by using a dedicated TabuBit1(v,j) array where TabuBit1(v,j) = 1 if (v,j) ∈ Tabu1 and TabuBit1(v,j) = 0 otherwise. However, if an additional saving of memory is desired, the membership in Tabu1 can be tracked by a procedure analogous to that indicated for handling TabuList(v*,j*).

To summarize, the key updates used in the improved SAT-DM method can be described by reference to the following elementary operations.

For updating Tabu1 in Step 1 for iter = 1:

Let (v′,j′) denote the element (v,j) that was added to Tabu1 on the step that occurred TabuLim executions in the past (if at least this many executions of Step 1 for iter = 1 have occurred).

Tabu1 and TabuBit1 Update.

For (v*,j*) chosen at Step 1 for iter = 1: Let Tabu1 = Tabu1∪(v*,j*) – (v′,j′). Set TabuBit1(v′,j′) = 0 and TabuBit1(v*,j*) = 1.

For checking and updating TabuList(v,j*)) in Step 1 for iter = 2 and for updating TabuList(v*,j*) and associated lists TabuList(v,j) in Step 4:*

Let NS denotes an arbitrary subset of NA (e.g., NS = TabuList(v*,j*)).

Set Bit(NS)

For each (v,j) ∈ NS, set TabuBit(v,j) = 1.

Re-Set Bit(NS)

For each (v,j) ∈ NS, set TabuBit(v,j) = 0.

Finally, to achieve the goals of the attribute selection problem, we use an attribute list AttList to record the attributes (v,j) that have been selected to create the inequalities generated. Once the SAT-DM method fills this list with a number of attributes sufficient to yield |AttList| ≥ AttLim, we restrict the method from then on to generate inequalities only from the attributes (v,j) ∈ AttList. This policy assures that the inequalities embody only a limited set of attributes, and also substantially reduces the computation from this

point onward. Thus, once |AttList| ≥ AttLim, the SAT-DM method prepares for the Core Routine by setting NA (the set of attributes accessed in this routine) to AttList, instead of setting NA to the original set of attributes NA^o. AttList is updated by the operation AttList := AttList∪NA* in Step 4 (as long as |AttList| < AttLim), and this may be done efficiently by keeping a bit list AttBit(v,j) that is devoted to identifying membership in AttList, or, if memory is desired to be conserved, by re-using the bit list TabuBit(v,j) with the Set Bit and Re-Set Bit operations.

On each first iteration of the Core Method (when iter = 1), we restrict consideration to elements j in a set NA1 which differs from NA if the attribute list AttList is full (|AttList| ≥ AttLim). In this case, in preparation for the Core Method we set NA1 = NA – {(v,j) ∈ AttList: |TabuList(v,j)| ≥ AttLim – 1}, thus removing from NA all those attributes (v,j) that have been paired with at least AttLim – 1 other attributes, since if we select any (v,j) in NA1 then we will be unable to match it with another attribute by the rule of iter = 2. (We specify AttLim – 1 instead of AttLim, to account for the fact that the attribute (1 – v,j) will not be listed on TabuList(v,j) but (v,j) still cannot be paired with it.) Though we do not bother to give the details, an array such as AttBit(v,j) can similarly be used to facilitate the updating and checking of NA1.

The improved form of the SAT-DM method that results from incorporating these observations is as follows.

Improved SAT-DM Method for Generating Multiple Inequalities.

0.0 (Initialization) Select a limit L on the number of inequalities to generate and a limit AttLim on the number of attributes to be used over all inequalities generated. Initialize p = 0, and set TabuBit(v,j) = TabuBit1(v,j) = AttBit(v,j) = 0 for all (v,j) ∈ NA^o. Let AttList = ∅.

1.0 (Prepare for the Core Method.) Set $G_\alpha = G_\alpha^o$, G_α* = ∅, $N_0 = N_1$ = ∅, NA* = ∅ and iter = 0.

Attribute List Activation Check:

If |AttList| ≥ AttLim (AttList is full, and is no longer allowed to grow)
  Set NA = AttList.
  NA1 = NA – {(v,j) ∈ AttList: |TabuList(v,j)| ≥ AttLim – 1}
  If NA1 = ∅, terminate the method.
Else (if |AttList| < AttLim)
  Set NA = NA^o.
  NA1 = NA
Endif

SAT-DM Core Method.

1. Set iter := iter + 1. Identify v* = 0 or 1 and j* ∈ N such that
  If iter = 1: (v*,j*) = arg max(a_{vj}: (v,j) ∈ NA1, TabuBit1(v,j) = 0).
    Execute *Tabu1 and TabuBit1 Update*. Set (v_1,j_1) = (v*,j*).
  If iter = 2: Execute *Set Bit*(TabuList(v_1,j_1))
    (v*,j*) = arg max(a_{vj}: (v,j) ∈ NA, TabuBit(v,j) = 0)
    Execute *Re-Set Bit*(TabuList(v_1,j_1))

If iter > 2: $(v^*,j^*) = \arg\max(a_{vj}: (v,j) \in NA)$
If $a_{v^*j^*} = 0$:
If iter = 1, terminate the Core Method.
If iter > 1, go to Step 4.
If $a_{v^*j^*} > 0$: continue to Step 2.

2. Set $N_{v^*} := N_{v^*} \cup \{j^*\}$, $NA := NA - \{0,j^*\} - \{1,j^*\}$, $G_\alpha^* := G_\alpha^* \cup \{i \in G_\alpha: x_{j^*}^i = v^*\}$ and $G_\alpha := G_\alpha - \{i \in G_\alpha: x_{j^*}^i = v^*\}$. Execute *Set Bit*(TabuList(v^*,j^*)) and let TabuSave = TabuList(v^*,j^*).[6] Then for each $(v,j) \in NA^*$ such that TabuBit(v,j) = 0,
Add (v^*,j^*) to TabuList(v,j) and add (v,j) to TabuList(v^*,j^*).
Execute *Re-Set Bit*(TabuSave) and set $NA^* := NA^* \cup \{(v^*,j^*)\}$
3. If $|G_\alpha^*| \geq f|\ G_\alpha^o\ |$ or $NA = \varnothing$ proceed to Step 4. Otherwise, determine the updated values $a_{vj} = |\{i \in G_\alpha: x_j^i = v\}|$, for $(v,j) \in NA$, and return to Step 1.
4. Set p := p + 1 and generate the inequality (1.1(p)), satisfied by all $x = x^i$ for $i \in G_\alpha^*$, given by

$$\sum(x_j: j \in N_1(p)) + \sum(\tilde{x}_j: j \in N_0(p)) \geq 1 \qquad (1.1(p))$$

If (1.1(p)) can be reduced, perform the reduction to replace it by a prime implicant, and remove from NA^* any attributes dropped in such a reduction. If |AttList| < AttLim, then for each $(v,j) \in NA^*$ such that AttBit(v,j) = 0, set AttList :=AttList$\cup$(v,j) and AttBit(v,j) = 1..

End of Core Method

2.0 If p = L or if $|G_\alpha^*| < f|\ G_\alpha^o\ |$, then stop. Otherwise return to Step 1.0.
End of Method

The comments of Remarks 1 through 5 can be used to reduce the computation of the method. A significant further reduction in computation can be achieved by using a candidate list CanAtt to restrict the number of attributes examined during the choice step – in this case, constituting a set of attributes (v,j) that are permitted to be examined in selecting (v^*,j^*) within Step 1. We briefly sketch how CanAtt can be incorporated within the SAT-DM method.

CanAtt operates much like AttList, except that it is larger than AttList (though still, as a rule, substantially smaller than NA^o) and is filled faster than AttList. This accelerated filling of CanAtt allows it to take over the job of restricting the number of choices examined while waiting for AttList to be filled. To accomplish this, CanAtt includes not only the attributes selected to generate the inequalities (1.1(p)), but also additional attributes that receive high evaluations as candidates for the pair (v^*,j^*) selected in Step1.

Thus, to start, CanAtt receives the following assignment of attributes in the Initialization Step 0.0

$$\text{CanAtt} = \{(v,j) \in NA^o: a_{vj} := \text{one of the } h_o \text{ largest } a_{vj} \text{ values for } (v,j) \in NA^o\}.$$

[6] TabuSave can be identified simply by flagging the last element in the current TabuList(v^*,j^*).

The parameter h_o is fairly small relative to AttLim (e.g., h_o = AttLim/20), subject to requiring, for example, $h_o \geq 10$. Subsequently, during iterations of the Core Method, when iter ≥ 2 we identify a set

$$\text{Can}' = \{(v,j) \in \text{NA}: a_{vj} \text{ is one of the } h_o \text{ largest } a_{vj} \text{ values for } (v,j) \in \text{NA}'\}$$

where

$$\text{NA}' = \{(v,j) \in \text{NA}: \text{TabuBit}(v,j) = 0\} \text{ for iter} = 2 \text{ and } \text{NA}' = \text{NA} \text{ for iter} > 2.$$

Then CanAtt is augmented to include new elements of Can′ (if any exist not already in CanAtt) by setting CanAtt := CanAtt∪NA′, taking advantage of a bit set CanBit(v,j) exactly analogous to the set AttBit(v,j).

Finally, a limit CanLim can be employed that operates for CanAtt in the same way as the limit AttLim operates for AttList (where, for example, CanLim = vAttLim for v between 1.5 and 3). When CanAtt is filled to the point where |CanAtt| ≥ **C**anLim, then the set NA is set equal to CanAtt in Step 1.0, until at last AttList is filled and NA is set equal to AttList from then on.

The foregoing method can be used to differentiate elements of G_α from those of G_β by a two-pass application in which G_β adopts the role of G_α on the second pass. The restricted set of attributes embodied in AttList is carried forward from the first pass to the second. Then, a new point of unknown membership becomes classified as belonging to G_α or G_β according to the number or proportion of the inequalities that the point satisfies for each of these two groups. In the next section we address the challenge of creating a unified method that simultaneously considers both G_α and G_β during the generation of each inequality, by referring to the problem SAT(QC/AC) in place of SAT(QC).

8. SAT-DM Method for SAT(QC/AC)

The description of SAT(QC/AC) is duplicated below as a basis for extending the foregoing ideas to generate inequalities relative to this more general problem.

SAT(QC/AC): Generate a collection of strongly satisfying solutions to the problem

Minimize $\sum((y_{1j} + y_{0j}): j \in N)$

subject to

$$\sum(x_j^i\, y_{1j}: j \in N) + \sum(\tilde{x}_j^i\, y_{0j}: j \in N) \geq 1 \quad i \in G_\alpha \qquad (2.1)$$

$$\sum(x_j^i\, y_{1j}: j \in N) + \sum(\tilde{x}_j^i\, y_{0j}: j \in N) \leq 0 \quad i \in G_\beta \qquad (2.2)$$

$$y_{1j} + y_{0j} \leq 1 \quad j \in N \qquad (2.3)$$

$$y_{1j}, y_{0j} \in \{0,1\} \quad j \in N \qquad (2.4)$$

We again make recourse to surrogate constraint analysis to exploit the problem structure. In the case of SAT(QC/AC), we generate two surrogate constraints, one for the inequalities (2.1) and one for the inequalities (2.2).

To be consistent with the statement of the Fundamental Observation, the index set G_β^* is used to identify the subset of inequalities of (2.2) that are satisfied at any stage of executing the SAT-DM method (just as G_α^* identifies the subset of the inequalities of (2.1) that are satisfied at any stage). Consequently, G_β^* initially starts out as $G_\beta^* = G_\beta^o$, in contrast to G_α^* which starts out as $G_\alpha^* = \varnothing$. The update of G_β^* is carried out in the same manner as the update of G_α (which, analogously to G_β^*, starts out $G_\alpha = G_\alpha^o$). Consequently, while G_α identifies the portion of G_α^o corresponding to constraints (2.1) not yet satisfied, G_β^* identifies the portion of G_β^o corresponding to constraints (2.2) not yet violated. (In reverse, while G_α^* identifies the constraints of (2.1) currently satisfied, G_β identifies the constraints of (2.2) currently violated.)

The surrogate constraint for (2.1) for the SAT(QC/AC) problem can then be written in the same way as for the SAT(QC) problem, i.e.

$$\sum(a_{1j}y_{1j}: j \in N) + \sum(a_{0j}y_{0j}: j \in N) \geq a_o \qquad (3)$$

Correspondingly, we write the surrogate constraint for (2.2) for the SAT(QC/AC) problem as

$$\sum(b_{1j}y_{1j}: j \in N) + \sum(b_{0j}y_{0j}: j \in N) \leq 0 \qquad (4)$$

As previously observed, the coefficients a_{1j} and a_{0j} produced by the surrogate sum rule are given by

$$a_{1j} = \sum(x_j^i: i \in G_\alpha) \text{ and } a_{0j} = \sum(\tilde{x}_j^i: i \in G_\alpha).$$

By contrast, in view of the preceding observations, the coefficients b_{1j} and b_{0j} are given by

$$b_{1j} = \sum(x_j^i: i \in G_\beta^*) \text{ and } b_{0j} = \sum(\tilde{x}_j^i: i \in G_\beta^*)$$

(i.e., the latter are created by summing over G_β^* rather than G_β). For the chosen attribute (v^*,j^*), the coefficient $b_{v^*j^*}$ identifies the number of inequalities $i \in G_\beta^*$ that will become violated by the assignment $y_{v^*j^*} = 1$, as opposed to the coefficient $a_{v^*j^*}$, which identifies the number of inequalities $i \in G_\alpha$ of (2.1) that will become satisfied by this assignment. (It is possible that $b_{v^*j^*} = 0$ and hence no new inequalities $i \in G_\beta^*$ will become violated.) We may equivalently write $a_{vj} = |\{i \in G_\alpha: x_j^i = v\}|$ and $b_{vj} = |\{i \in G_\beta^*: x_j^i = v\}|$, for $v \in \{0.1\}$.

By the discussion of the SAT-DM method for the SAT(QC) problem, we are assured of being able to select v^* and j^* so that $a_{v^*j^*} > 0$ on each iteration of Step 1 (except possibly when iter = 1 for the Basic Method, in which case the method terminates). There is no virtue in a choice of v^* and j^* that yields $a_{v^*j^*} = 0$, given that setting $y_{v^*j^*} = 1$ can not improve the number of inequalities of (2.2) that are satisfied in this case.

We propose two approaches for taking advantage of the surrogate constraint (4) to create a choice rule for selecting v* and j*. For the first, write (4) as

$$\sum(-b_{1j}y_{1j}: j \in N) + \sum(-b_{0j}y_{0j}: j \in N) \geq 0 \qquad (4a)$$

Then we create a single surrogate constraint as a composite of (3) and (4a) by choosing a positive weight w to multiply by (3) and add to (4a), thus yielding

$$\sum(wa_{1j} - b_{1j})y_{1j}: j \in N) + \sum(wa_{0j} - b_{0j}y_{0j}): j \in N) \geq 0 \qquad (5)$$

The value w captures the tradeoff between the amount of emphasis placed on satisfying constraints of (2.1) versus violating constraints of (2.2). A natural choice for w is $w = |G_\beta^*| / |G_\alpha|$, which gives an equal emphasis to proportional changes in $|G_\beta^*|$ and $|G_\alpha|$. ($|G_\alpha| \geq 1$ holds whenever v* and j* are chosen, because if G_α becomes empty the method does not return to Step 1 but generates a new inequality at Step 4.) Then the associated rule for choosing v* and j* can be expressed as

$$(v^*,j^*) = \arg\max(wa_{vj} - b_{vj}: v \in \{0,1\}, j \in N).$$

The surrogate constraints (3) and (4) can be used in an additional way. Let T_α be a threshold establishing a lower bound on the improvement sought in G_α, and correspondingly, let T_β be a threshold establishing an upper bound on the deterioration allowed in G_β. Then the rule for choosing v* and j* that takes these thresholds into account may be specified as

Rule 1: $(v^*,j^*) = \arg\max(wa_{vj} - b_{vj}: a_{vj} \geq T_\alpha, b_{vj} \leq T_\alpha, v \in \{0,1\}, j \in N).$

This rule includes the previous one as a special case by selecting both T_α and T_β redundant, as by $T_\alpha = 0$ and T_β = large (where $|G_\beta^o|$ suffices for "large"). Extreme examples include

$T_\alpha = \text{Max}(a_{vj}: v \in \{0,1\}, j \in N)$, T_β = large, and
$T_\alpha = 0$, $T_\beta = \text{Min}(b_{vj}: v \in \{0,1\}, j \in N)$.

Based on the fact that G_α is the focal group, and that $a_{vj} > 0$ is preferable to assure, an appealing form of Rule 1 is to take T_β = large and to choose T_α from the interval

$$\text{Mean}(a_{vj} > 0: v \in \{0,1\}, j \in N) \leq T_\alpha \leq \text{Max}(a_{vj}: v \in \{0,1\}, j \in N).$$

The second approach for exploiting the pair of surrogate constraints (3) and (4) is to maximize the ratio a_{vj}/b_{vj} (representing the increase in the number of constraints satisfied in (2.1) to the increase in the number of constraints violated in (2.2)), or more precisely to maximize the ratio a_{vj}/b_{vj}^e, selecting the exponent $e > 1$ (e.g., e = 1.5 or 2), to emphasize the goal of avoiding the case where a choice will result in (or lead to) violating a significant number of the inequalities of (2.2). We replace b_{vj}^e in the ratio a_{vj}/b_{vj}^e by

adding a moderately small positive value ε (e.g., $\varepsilon \leq .1$) to the denominator to avoid complication when $b_{vj} = 0$. Then we obtain the rule

Rule 2: $(v^*,j^*) = \arg\max((a_{vj}/(b_{vj}^e + \varepsilon))\text{:}, v \in \{0,1\}, j \in N)$

For definiteness, we will incorporate Rule 2 into our description of the SAT-DM method for SAT(QC/AC), though Rule 1 can be used as well.

Alternating Construction and Destruction

The SAT-DM method for SAT(QC/AC) begins in the same way as the version for SAT(QC), by constructively choosing a new attribute (v*,j*) to augment the set NA(p) at each iteration of the Core Method. The only difference is in the rule for selecting (v*,j*). The fraction f is taken to be 1 to allow the constructive process to continue until $G_\alpha^* = G_\alpha^o$ ($G_\alpha = \varnothing$) when this is possible, in order to satisfy all of the inequalities of (2.1). This asymmetric solution, which satisfies all of (2.1), has a special role in a tree-based classification procedure as we show later. However, to compensate for this asymmetry, we follow the constructive process of the Core Method with a destructive process to successively remove selected attributes from NA(p) to create a balance between the number of data points satisfying (2.1) and the number violating (2.2), respectively identified by $|G_\alpha^*|$ and $|G_\beta^*|$.

Each destructive iteration decreases both $|G_\alpha^*|$ and $|G_\beta^*|$, hence improving $|G_\beta^*|$ while worsening $|G_\alpha^*|$. (Reducing (2.1(p)*) to a prime implicant before the destructive step removes the possibility that $|G_\alpha^*|$ will remain unchanged.) We again make use of surrogate constraint coefficients that identify the change in these two values, and select a preferred (v*,j*) to drop by a ratio rule related to Rule 2, by identifying

$$\text{Cover}_\alpha(i) = \sum(x_j^i\text{: } j \in N_1(p)) + \sum(\tilde{x}_j^i\text{: } j \in N_0(p)), \text{ for } i \in G_\alpha^o \qquad (6)$$

and similarly

$$\text{Cover}_\beta(i) = \sum(x_j^i\text{: } j \in N_1(p)) + \sum(\tilde{x}_j^i\text{: } j \in N_0(p)), \text{ for } i \in G_\beta^o. \qquad (7)$$

The inequality (2.1) is satisfied for a given $i \in G_\alpha^o$ if $\text{Cover}_\alpha(i) \geq 1$ and the inequality (2.2) is satisfied for a given $i \in G_\beta^o$ if $\text{Cover}_\beta(i) \leq 0$. We exploit this fact as follows.

For $j \in N_1(p)$, define

$$a_{1j} = \sum(x_j^i\text{: } i \in G_\alpha^o, \text{Cover}_\alpha(i) = 1) \text{ and } b_{1j} = \sum(x_j^i\text{: } i \in G_\beta^o, \text{Cover}_\beta(i) = 1) \qquad (8)$$

Similarly, for $j \in N_0(p)$, define

$$a_{0j} = \sum(\tilde{x}_j^i\text{: } i \in G_\alpha^o, \text{Cover}_\alpha(i) = 1) \text{ and } b_{0j} = \sum(\tilde{x}_j^i\text{: } i \in G_\beta^o, \text{Cover}_\beta(i) = 1) \qquad (9)$$

Then a_{vj} identifies the number of new inequalities for $i \in G_\alpha^o$ that become violated by dropping (v,j) from NA(p) (by implicitly setting $y_{vj} = 0$) and b_{vj} identifies the number of new inequalities for $i \in G_\beta^o$ that become satisfied by dropping (v,j) from NA(p) (likewise by setting $y_{vj} = 0$) . Thus, to assure an appropriate tradeoff between improving $|G_\beta^*|$ and worsening $|G_\alpha^*|$, we require $b_{vj}/a_{vj} > R$ where R is a selected ratio such as $R = |G_\beta^o|/|G_\alpha^o|$. First, to remove the possibility $a_{vj} = 0$, we reduce the inequality to a prime implicant by the choice rule

Prime Implicant Choice Rule: $(v^*,j^*) = \arg\max(b_{vj}: a_{vj} = 0, (v,j) \in NA(p))$

After updating by dropping (v^*,j^*) from NA(p) and computing the new a_{vj} and b_{vj} values, the process repeats. Once this rule can no longer be applied, because $a_{vj} > 0$ for all $(v,j) \in$ NA(p), the inequality is a prime implicant. Thereafter we apply the choice rule

Ratio Choice Rule: $(v^*,j^*) = \arg\max(b_{vj}^e/a_{vj}: (v,j) \in NA(p))$.

This process is similarly repeated until $a_{v^*j^*} \leq R$. At that point an addition inequality pair (1.1P) and (1.2P) is obtained, and the method goes back to launch the next execution of the Core Method, starting with its constructive component.

There is one final consideration to be mentioned regarding these choice rules. From the standpoint of a more robust classification, there can be value in not merely satisfying the inequalities of the original SAT(QC/AC) system, but in producing values of $Cover_\alpha(i)$ as large as possible for $i \in G_\alpha^o$ and values of $Cover_\beta(i)$ as small as possible for $i \in G_\beta^o$. Thus, we not only strive to yield $Cover_\alpha(i) \geq 1$ and $Cover_\beta(i) \leq 0$, but to encourage $Cover_\alpha(i)$ to increase still further while preventing $Cover_\beta(i)$ from increasing more than necessary, subject to meeting our first objective. Consequently, it is useful to extend the interpretation of the arg max function used in the Constructive and Destructive components of the method, to endow it with a tie-breaking function. In the Constructive Component, where we choose $(v^*,j^*) = \arg\max(a_{vj}/(b_{vj}^e + \varepsilon): (v,j) \in NA)$, there can well be situations where different attributes (v,j) yield the same a_{vj} and b_{vj} values that result in ties for choosing (v^*,j^*). We propose to break ties by reapplying the same choice rule, but replacing the current a_{vj} and b_{vj} values with their original values determined relative to the sets G_α^o and G_β^o. These original values can be computed and stored just once, in a preprocessing step, so that they can be instantly accessed without re-computation as a basis for resolving ties relative to the current values. The same comments apply to the determination of (v^*,j^*) by reference to the choice rules used in the Destructive Component.

The complete SAT-DM method for SAT(QC/AC) may now be described as follows.

SAT-DM Method for SAT(QC/AC).

0.0 (Initialization) Select a limit L on the number of inequalities to generate and a limit AttLim on the number of attributes to be used over all inequalities generated. Initialize p = 0, and set TabuBit(v,j) = TabuBit1(v,j) = AttBit(v,j) = 0 for all $(v,j) \in NA^o$. Let AttList = ∅.

1.0 (Prepare for the Core Method.) Set $G_\alpha = G_\alpha^o$, $G_\alpha^* = \varnothing$, $G_\beta = G_\beta^*$, $G_\beta = \varnothing$, $N_0 = N_1 = \varnothing$, NA* = ∅ and iter = 0. Set $\text{Cover}_\alpha(i) = 0$ for $i \in G_\alpha^o$ and $\text{Cover}_\beta(i) = 0$ for $i \in G_\beta^o$.

Attribute List Activation Check:

If |AttList| ≥ AttLim (AttList is full, and is no longer allowed to grow)

Set NA = AttList.

NA1 = NA – {(v,j) ∈ AttList: |TabuList(v,j)| ≥ AttLim – 1}

If NA1 = ∅, terminate the method.

Else (if |AttList| < AttLim)

Set $NA = NA^o$.

NA1 = NA

Endif

SAT-DM Core Method for SAT(QC/AC)

Constructive Component

1. Set iter := iter + 1. Identify (v*,j*) ∈ NA such that

If iter = 1: $(v^*,j^*) = \arg\max(a_{vj}/(b_{vj}^e + \varepsilon)$: (v,j) ∈ NA1, TabuBit1(v,j) = 0).

Execute *Tabu1 and TabuBit1 Update*. Set $(v_1,j_1) = (v^*,j^*)$.

If iter = 2: Execute *Set Bit*(TabuList(v_1,j_1))

$(v^*,j^*) = \arg\max(a_{vj}/(b_{vj}^e + \varepsilon)$: (v,j) ∈ NA, TabuBit(v,j) = 0)

Execute *Re-Set Bit*(TabuList(v_1,j_1))

If iter > 2: $(v^*,j^*) = \arg\max(a_{vj}/(b_{vj}^e + \varepsilon)$: (v,j) ∈ NA)

If $a_{v^*j^*} = 0$:

If iter = 1, terminate the Core Method.

If iter > 1, go to Step 4.

If $a_{v^*j^*} > 0$: continue to Step 2.

2. Set $N_{v^*} := N_{v^*} \cup \{j^*\}$, $NA := NA - \{0,j^*\} - \{1,j^*\}$, $G_\alpha^* := G_\alpha^* \cup \{i \in G_\alpha: x_{j^*}^i = v^*\}$ and $G_\alpha := G_\alpha - \{i \in G_\alpha: x_{j^*}^i = v^*\}$. $G_\beta := G_\beta \cup \{i \in G_\beta: x_{j^*}^i = v^*\}$, $G_\beta^* := G_\beta^* - \{i \in G_\beta: x_{j^*}^i = v^*\}$, Also set $\text{Cover}_\alpha(i) := \text{Cover}_\alpha(i) + 1$ for $i \in G_\alpha^o$: $x_{j^*}^i = v^*$, and $\text{Cover}_\beta(i) := \text{Cover}_\beta(i) + 1$ for $i \in G_\beta^o$: $x_{j^*}^i = v^*$.

Execute *Set Bit*(TabuList(v*,j*)) and let TabuSave = TabuList(v*,j*).[7] Then for each (v,j) ∈ NA* such that TabuBit(v,j) = 0:

Add (v*,j*) to TabuList(v,j) and add (v,j) to TabuList(v*,j*).

Execute *Re-Set Bit*(TabuSave) and set $NA^* := NA^* \cup \{(v^*,j^*)\}$.

3. If $|G_\alpha^*| \geq |G_\alpha^o|$ or NA = ∅ proceed to Step 4. Otherwise, determine the updated values $a_{vj} = |\{i \in G_\alpha: x_j^i = v\}|$, for (v,j) ∈ NA, and return to Ste(1.1(p)), p 1.

4. Set p := p + 1 and generate the inequality (1.1(p)), satisfied by all $x = x^i$ for $i \in G_\alpha^*$ and $i \in G_\beta$ and the inequality (1.2(p)), satisfied by all $x = x^i$ for $i \in G_\alpha$ and $i \in G_\beta^*$ given by

$$\sum(x_j: j \in N_1(p)) + \sum(\tilde{x}_j: j \in N_0(p)) \geq 1 \qquad (1.1(p))$$

$$\sum(x_j: j \in N_1(p)) + \sum(\tilde{x}_j: j \in N_0(p)) \leq 0 \qquad (1.2(p))$$

End of Constructive Component

[7] TabuSave can be identified simply by flagging the last element in the current TabuList(v*,j*).

Destructive Component

Generate a_{vj} and b_{vj} by (8) and (9) for $(v,j) \in NA(p)$.

Phase 1.

Set iter1 = 0.

D1. Select (v^*,j^*) by the Prime Implicant Choice Rule:

$$(v^*,j^*) = \arg\max(b_{vj}: a_{vj} = 0, (v,j) \in NA(p))$$

If (v^*,j^*) is not defined: proceed to Step D2 if iter1 > 0 and to Step D3 if iter1 = 0. Otherwise, if (v^*,j^*) is defined: set iter1 := iter1 + 1, update NA(p) by dropping (v^*,j^*), and update $Cover_\alpha(i)$ and $Cover_\beta(i)$ and a_{vj} and b_{vj} by (6) – (9); then repeat D1.

D2. (iter > 0: NA(p) has been reduced in D1 on previous Phase 1 iterations). Redefine (1.1(p)) and (1.2(p)) relative to the reduced set NA(p).

D3. (Inequality (1.1(p)) is a prime implicant). If |AttList| < AttLim, then for each $(v,j) \in NA^*$ such that AttBit(v,j) = 0, set AttList :=AttList$\cup${v,j) and AttBit(v,j) = 1.

Phase 2.

Set iter2 = 0.

D4. Select (v^*,j^*) by the Ratio Choice Rule:

$$(v^*,j^*) = \arg\max(b_{vj}^e/a_{vj}: (v,j) \in NA(p))$$

If $a_{v^*j^*} \leq R$:

If iter2 > 0, proceed to Step D5, while if iter2 = 0, terminate the Destructive Component (hence terminating the Core Method).

Otherwise, if $a_{v^*j^*} > R$:

Set iter2 := iter2 + 1, update NA(p) by dropping (v^*,j^*), and update $Cover_\alpha(i)$ and $Cover_\beta(i)$ and a_{vj} and b_{vj} by (6) – (9). Then repeat D4.

D5. Set p := p + 1 and generate the balanced inequalities

$$\sum(x_j: j \in N_1(p)) + \sum(\tilde{x}_j: j \in N_0(p)) \geq 1 \qquad (1.1(p))$$

$$\sum(x_j: j \in N_1(p)) + \sum(\tilde{x}_j: j \in N_0(p)) \leq 0 \qquad (1.2(p))$$

End of Destructive Component and End of Core Method

2.0 If $p \geq L$, then stop. Otherwise return to Step 1.0.

End of Method

In the pursuit of creating inequalities (1.1(p) and (1.2(p)) yielding classifications of greater robustness, we can go a step beyond our convention of interpreting the arg max function so that it breaks ties to increase the values $Cover_\alpha(i)$ and decrease the values of $Cover_\beta(i)$, by adding a final "Re-constructive Component" after the Destructive Component of the Core Method. In this final component, the method looks for choices that can promote a still better distribution of $Cover_\alpha(i)$ and $Cover_\beta(i)$ values, without permitting any $Cover_\beta(i)$ values that are 0 to become positive. Likewise, it is possible to use the $Cover_\alpha(i)$ and $Cover_\beta(i)$ values in the rules for choosing (v^*,j^*) to differentially encourage smaller $Cover_\alpha(i)$ values to grow end larger $Cover_\beta(i)$ values not to grow. Whether the added computational expense of such refinements is warranted will depend on the setting.

As in the case of the method of Section 7 for the SAT(QC) problem, the preceding method can be executed in two passes where the second interchanges the roles of G_α and G_β.

A Staged Variant

A useful variant of the foregoing method arises by generating more than one inequality in the Constructive Component by means of a staged process as follows. The choice rule for selecting (v*,j*) in the Constructive Method, given by

$$(v^*,j^*) = \arg\max(a_{vj}/(b_{vj}^e + \varepsilon): (v,j) \in NA)$$

will automatically select $a_{v^*j^*} > 0$ when possible, and if the outcome $a_{v^*j^*} = 0$ results the Constructive Component terminates. (If iter = 1 when $a_{v^*j^*} = 0$ the Core Method itself, terminates.) Given $a_{v^*j^*} > 0$ the preceding rule will also choose $b_{v^*j^*} = 0$ whenever the latter is also possible, and in this circumstance the method avoids increasing the number of inequalities in the system (2.2) that are currently violated. Furthermore, if the exponent e is made large enough the rule for choosing (v*,j*) will always favor a smaller $b_{v^*j^*}$ value over a larger one. Hence in this case the succession of choices will always give priority to creating the smallest deterioration (increase) in the set G_β (which identifies the violated inequalities of (2.2) and the points x^i that are covered by (1.2(p))).

Such an approach is appealing from a conservative perspective of seeking to limit the growth in G_β at each step, producing a pattern where successive executions of the Core Method are induced to generate inequalities (1.1(p)) and (1.2(p)) for smaller G_β sets before generating inequalities for larger G_β sets. Under such a "staged" organization, the Constructive Component of the Core Method can appropriately record additional inequalities generated along the way, saving the inequality for each of the conditions $|G_\beta| = 0$, $|G_\beta| = 1$, $|G_\beta| = 2$, etc., that yields the maximum size of the set G_α* (which identifies the satisfied inequalities of (2.1)) for this particular value of $|G_\beta|$. This organization also permits the Constructive Component to be terminated once $|G_\beta|$ reaches a selected size, thus limiting the number of violated inequalities from the system (2.2) that will be tolerated. Each of the inequalities (1.1(p)) and (1.2(p)) thus recorded for different $|G_\beta|$ values is relevant in a different way for classifying an unknown point correctly. For example, we have already suggested two alternatives for classifying a point of unknown membership as belonging to one of the original group G_α^o or G_β^o, giving it a vote of either 1 or $|G_\alpha^*|/|G_\alpha^o|$ for belonging to G_α^o if it satisfies a given inequality (1.1(p)), and similarly giving it a vote of either 1 or $|G_\beta^*|/|G_\beta^o|$ for belonging to G_β^o if it satisfies a given inequality (1.2(p)). The record of additional inequalities for different numbers of elements of G_α* and G_β (hence of $G_\beta^* = G_\beta^o - G_\beta$) expands the representation of this process. It also allows reference to be made to ratios such as $|G_\alpha^*|/(|G_\alpha^*| + |G_\beta^*|)$ $|G_\beta^*|/(|G_\alpha^*| + |G_\beta^*|)$ to further refine such votes.

The indicated Staged Method has another potential advantage that derives from its ability to be organized in a different manner. Rather than assigning a large value to the exponent

e in the rule for choosing (v*,j*), the rule can be broken into the following two parts:

Identify $b^* = \text{Min}(b_{vj}: (v,j) \in NA)$
Select $(v^*,j^*) = \arg\max(a_{vj}: b_{vj} = b^*, (v,j) \in NA)$

The utility of this organization comes from the following observation. After each choice of (v*,j*), the value of $|G_\beta|$ becomes increased by b*, and hence if b* = 0, the size of $|G_\beta|$ does not change. In addition, the outcome b* > 0 implies that the inequality pair (1.1(p)) and (1.2(p)) (produced by the current composition of $N_1(p)$ and $N_0(p)$ before selecting the new (v*,j*)) gives the locally maximum value of $|G_\alpha^*|$ for the current value of $|G_\beta|$. Thus, b* > 0 signals that the current pair (1.1(p)) and (1.2(p)) should be identified in order to record the desired inequality for each value of $|G_\beta|$. (Such a record is excluded if iter = 1, because at this point G_α^* is still empty and no implied inequality pair (1.1(p)) and (1.2(p)) yet exists.)

This two-part choice rule can be executed with particular efficiency if more than one iteration occurs in succession with b* = 0. Let NA(b*) denote be the reduced subset of NA over which the selection of (v*,j*) is implicitly restricted by the two-part rule; i.e., $NA(b^*) = \{(v,j) \in NA: b_{vj} = b^*\}$. Then, during any series of iterations in which b* = 0, the set NA(b*) (= NA(0)) cannot grow, but can only decrease in size. Whenever b* = 0, b* will again be 0 on the next iteration if and only if NA(0) remains non-empty when updated in the same way that NA is updated; i.e., NA(0) := NA(0) – {0,j*} – {1,j*} for each choice of j* once b* = 0 occurs (either on iter = 1 or immediately after the latest iteration when b* > 0). Moreover, as long as NA(0) remains non-empty by this update, then the next choice of (v*,j*) is simplified by $(v^*,j^*) = \arg\max(a_{vj}: (v,j) \in NA(0))$, and there is no need to first identify b* from its definition.

A special condition needs to be heeded for iter = 2 and for the case where NA is initialized to equal AttList (in preparation for the Core Method). Then possibly the choice of $a_{v^*j^*}$ may yield $a_{v^*j^*} = 0$ even if NA(0) is non-empty,. (This outcome can't occur if NA is instead initialized by $NA = NA^o$ and if iter ≠ 2 because $a_{v^*j^*} = 0$ can only occur if $(v,j) \in NA$ implies $(1 – v,j) \notin NA$, which is not true until NA becomes a subset of AttList or the special rule of iter = 2 is used to select (v*,j*).) Thus, whenever $a_{v^*j^*} = 0$ or whenever NA(0) becomes empty, the method reverts to the complete two-part rule to first compute b* and then identify (v*,j*) without taking advantage of the simplified rule.

9. A Compound Variable SAT(QC/AC) Method

The SAT(QC/AC) Method is susceptible to an enhancement that separates the groups G_α and G_β more effectively by introducing additional binary variables as products of other variables, e.g., representing a product such as $x_1x_2\tilde{x}_3$ (= $x_1x_2(1-x_3)$) as an additional binary variable. We show how this can be done adaptively.

The Compound Variable SAT-DM Method embodies a modification of the Constructive Component of the SAT-DM Core Method, but retains most of its present structure to initiate the choice of each first component term x_j or $\tilde{x}_j$ of a compound variable. The

choice step consists of selecting an index j* and a 0-1 value v*, as before, to make an assignment $x_{j*}^{i} = v^*$, which by the Fundamental Observation translates into creating a term x_{j*} and designating j* to be an element of $N_1(p)$ if v* = 1, or of creating a term $\tilde{x}_{j*}$, and designating j* to be an element of $N_0(p)$, if v* = 0.

The change entails introducing an Inner Loop into the Core Method after selecting the first term of a compound variable. The Inner Loop then selects additional terms x_j or $\tilde{x}_j$, by means of additional choices of pairs (j*,v*), until the complete compound variable is produced, whereupon it is treated as if it were a "normal" 0-1 variable, y_h, which in this case is identified by

$$y_h = \prod (x_j: j \in N_1(h)) \prod(\tilde{x}_j: j \in N_0(h)),$$

By convention, the product over $N_1(h)$ or $N_0(h)$ is understood to equal 1 if the corresponding set is empty.

The complete set of compound variables that compose the current inequality pair (1.1(p)) and (1.2(p)) is denoted by y_h, $h \in H(p)$. Consequently, the new compound forms of the inequalities are denoted by (1.1-H(p)) and (1.2-H(p)) , and given the representation

$$\sum(y_h: h \in H(p)) \geq 1 \qquad \text{(1.1-H(p))}$$
$$\sum(y_h: h \in H(p)) \leq 0 \qquad \text{(1.2-H(p))}$$

A new y_h variable is created at each iteration, iter, of the Core Method, and hence |H(p)*| = iter each time the full inequality pair (1.1-H(p)) and (1.2-H(p)) is generated.

From a logical standpoint, a compound variable corresponds to a conjunction rather than a disjunction of terms. Hence y_h corresponds to a conjunction of each of its x_j and $\tilde{x}_j$ components. As a result, each new component added to y_h creates the following effect.

Let I_α and I_β* denote the intersection of elements of G_α and G_β generated by the compound variable y_h. When the first term (x_{j*} or $\tilde{x}_{j*}$) of the y_h is created by the initial choice of the pair (j*,v*), we set $I_\alpha = \{i \in G_\alpha: x_{j*}^{i} = v^*\}$ and $I_\beta^* = \{i \in G_\beta: x_{j*}^{i} = v^*\}$. The choice of each new pair (j*,v*) produces the updated form of the intersections I_α and I_β* by setting $I_\alpha := \{i \in I_\alpha: x_{j*}^{i} = v^*\}$ and $I_\beta^* := \{i \in I_\beta^*: x_{j*}^{i} = v^*\}$. Hence, each updated instance of I_α and I_β* is a subset of its previous instantiation. Our goal remains to cover as much of the original G_α^o as possible and as little of the original G_β^o as possible, and hence to maximize the size of I_α and minimize the size of I_β*. The fact that the compound variables generate intersections hinders the maximization objective but helps the minimization objective. If we obtain a compound variable that results in making I_β* empty (as successive intersections help to foster), then a gain has been achieved by covering all of the original G_α^o that falls in I_α while avoiding the inclusion of any part of the original G_β^o.

The surrogate constraint choice rule used in the SAT(QC/AC) method continues to be applicable to selecting each successive term of a compound variable, except that we

replace the reference to G_α and G_β* by a reference instead to I_α and I_β*. Thus, within the Inner Loop of the Compound Variable Method the coefficients a_{vj} and b_{vj} of the surrogate constraints (3) and (4) are generated relative to this replacement by

$$a_{1j} = \sum(x_j^i : i \in I_\alpha) \text{ and } a_{0j} = \sum(\tilde{x}_j^i : i \in I_\alpha),$$
$$b_{1j} = \sum(x_j^i : i \in I_\beta^*) \text{ and } b_{0j} = \sum(\tilde{x}_j^i : i \in I_\beta^*)$$

By its nature, the surrogate constraint rule that chooses $(v^*,j^*) = \arg\max(a_{vj}/(b_{vj}^e + \varepsilon)$ will thereby favor making I_α large and I_β* small, since by the intersection update, the coefficients a_{vj} and b_{vj} identify the new cardinality of each of these sets as a result of choosing (v^*,j^*).

The SAT(QC/AC) method becomes simplified in one respect by including compound variables, in that the Core Method no longer needs to include a Destructive Component. In particular, the generation of compound variables takes care of the goal of establishing an improved tradeoff between satisfying constraints associated with G_α and violating constraints associated with G_β, and hence the Destructive Component is not required to achieve this goal. However, the handling of tabu restrictions becomes more subtle. Now, instead of seeking to avoid a duplicate pairing of choices on iter = 2 in the Core Method (i.e., in its Constructive Component), we seek simply to assure that at least one compound variable has a different composition than any previous compound variable. This is done by a design similar to that previously used to handle the case for iter = 2 in the Core Method, but applied to iter0 = 2, where iter0 identifies the iteration count for the Inner Loop for a given iteration of the Core Method. More precisely, we now avoid a pairing when iter = 1 and iter0 = 2 that has occurred in any previous compound variable, since this will assure the resulting new compound variable creates no duplications.

There is one more subtlety, however, because it is possible that the method will only generate a single variable and not a compound variable when iter = 1 and iter0 = 2. We make use of an indicator Compound = *false* if such a compound variable is not created and Compound = *true* otherwise. When iter = 2 and Compound = *false*, then we use the rule previously identified for the case of iter = 2 in the Core Method. The rule has a different significance, however, because the array TabuList(v,j) now stores just the attributes that have been paired in (v,j) in creating compound variables. Hence, whenever (v^*,j^*) is selected to avoid a pairing, the choice refers strictly to a pairing involving such compound variables. This means that TabuList(v,j) is updated strictly within the Inner Loop, where compound variables are created, rather than externally in the Core Method as previously.

For simplicity, we continue to refer to NA as the source of the pairs (v,j) from which (v^*,j^*) is chosen, but NA is no longer updated by removing both (v^*,j^*) and $(1 - v^*,j^*)$ following the selection of (v^*,j^*), but by removing only $(1 - v^*,j^*)$, since it is possible that the choice (v^*,j^*) may be useful within another compound variable on the same execution of the Inner Loop. Our handling of TabuList(v,j) permits such repetition to occur while avoiding a repetition among combinations of variables that compose a compound variable.

Finally, we impose an aspiration threshold on the amount of improvement that occurs as a result of choosing (j*,v*). Using the choice rule of the preceding section, which selects $(v^*,j^*) = \arg\max(a_{vj}/(b_{vj}^e + \varepsilon): (v,j) \in NA)$, we want to be sure that the ratio that determines this choice does not deteriorate below some minimum value MinRatio over successive iterations, and thus will make some minimum level of progress toward covering G_α and/or shrinking G_β. For convenience we select MinRatio to be the value $a_{v^*j^*}/(b_{v^*j^*}^e + \varepsilon)$ achieved by the first (v*,j*) choice on the Inner Loop that generates the compound variable y_h.[8] Consequently, whenever the choice of (v*,j*) made after this first choice results in $(a_{v^*j^*}/(b_{v^*j^*} + \varepsilon) \leq$ MinRatio, the Inner Loop is terminated. (For convenience we allow y_h to refer to a single variable x_j or $\tilde{x}_j$ when a compound variable consisting of additional terms fails to be generated.)

Drawing on these observations, the compound variable method may be stated as follows.

Compound Variable SAT-DM Method for SAT(QC/AC).

0.0 (Initialization) Select a limit L on the number of inequalities to generate and a limit AttLim on the number of attributes to be used over all inequalities generated. Initialize p = 0, and set TabuBit(v,j) = TabuBit1(v,j) = AttBit(v,j) = 0 for all $(v,j) \in NA^o$. Let AttList = ∅. Set h = 0 (to index the compound variables y_h generated within the Core Method). Set $NA = NA^o$

1.0 (Prepare for the Core Method.) Set $G_\alpha = G_\alpha^o$, $G_\alpha^* = \emptyset$, $G_\alpha^* = \emptyset$, $G_\beta = G_\beta^*$, $G_\beta = \emptyset$, $NA^* = \emptyset$ and iter = 0.

Attribute List Activation Check:

If |AttList| ≥ AttLim (AttList is full, and is no longer allowed to grow)

Set NA = AttList.

NA1 = NA – {(v,j) ∈ AttList: |TabuList(v,j)| ≥ AttLim – 1}

If NA1 = ∅, terminate the method.

Else (if |AttList| < AttLim)

Set $NA = NA^o$.

NA1 = NA

Endif

SAT-DM Core Method for SAT(QC/AC)

1. Set iter := iter + 1. Identify $(v^*,j^*) \in NA$ such that

If iter = 1: $(v^*,j^*) = \arg\max(a_{vj}/(b_{vj}^e + \varepsilon): (v,j) \in NA1$, TabuBit1(v,j) = 0.)

Execute *Tabu1 and TabuBit1 Update*. Set $(v_1,j_1) = (v^*,j^*)$.

Set Compound = *false* (to be set to *true* in the Inner Loop if the right conditions occur).

If iter = 2 and Compound = *false*: Execute *Set Bit*(TabuList(v_1,j_1))

$(v^*,j^*) = \arg\max(a_{vj}/(b_{vj}^e + \varepsilon): (v,j) \in NA$, TabuBit(v,j) = 0)

Execute *Re-Set Bit*(TabuList(v_1,j_1))

Otherwise: $(v^*,j^*) = \arg\max(a_{vj}/(b_{vj}^e + \varepsilon): (v,j) \in NA)$

NA := NA – {(1 – v*,j*)}.

[8] This value for MinRatio can undoubtedly be improved by experimentation.

If |AttList| < AttLim and if AttBit(v*,j*) = 0:
set AttList := AttList$\cup${v*,j*) and AttBit(v*,j*) = 1, and if now |AttList| $\geq$ AttLim, set NA := NA$\cap$AttList.

Inner Loop for Compound Variables

I0. iter0 = 1 and CompoundList(iter0) = (v*,j*)
h := h + 1
Set $y_h := x_{j*}$ if v* = 1 and $y_h = \tilde{x}_{j*}$ if v* = 0.
Set $I_\alpha = \{i \in G_\alpha: x_{j*}^{\,i} = v*\}$ and $I_\beta* = \{i \in G_\beta: x_{j*}^{\,i} = v*\}$.
Set MinRatio = $a_{v*j*}/(b_{v*j*}^{\,e} + \varepsilon)$
If $|I_\beta*| = 0$ ($I_\beta* = \varnothing$, or equivalently $b_{v*j*} = 0$) then y_h is complete without including additional component variables and the Inner Loop terminates.
Otherwise, execute the following steps.

I1. iter0 := iter0 + 1
If iter0 = 2 and iter = 1: Execute *Set Bit*(TabuList(v_1,j_1))
(v*,j*) = arg max($a_{vj}/(b_{vj}^{\,e} + \varepsilon)$: (v,j) $\in$ NA, TabuBit(v,j) = 0)
Execute *Re-Set Bit*(TabuList(v_1,j_1))
Otherwise: (v*,j*) = arg max($a_{vj}/(b_{vj}^{\,e} + \varepsilon)$: (v,j) $\in$ NA)
Ratio Improvement Test:
If $(a_{v*j*}/(b_{v*j*}^{\,e} + \varepsilon) \leq$ MinRatio terminate the Inner Loop. (The threshold of improvement is not satisfied.)
Otherwise, if $(a_{v*j*}/(b_{v*j*}^{\,e} + \varepsilon) >$ MinRatio:
If iter0 = 2 and iter = 1 set Compound = *true* (a compound variable has been successfully created for iter0 = 2 and iter = 1)
$I_\alpha := \{i \in I_\alpha: x_{j*}^{\,i} = v*\}$ and $I_\beta* := \{i \in I_\beta*: x_{j*}^{\,i} = v*\}$.
Add j* to $N_{v*}(h)$ and set $y_h := y_h x_{j*}$ if v* = 1 and $y_h = y_h \tilde{x}_{j*}$ if v* = 0.
(Hence $y_h = \prod(x_j: j \in N_1(h)) \prod(\tilde{x}_j: j \in N_0(h))$.)
NA := NA – {(1 – v*,j*)}.
If |AttList| < AttLim and if AttBit(v*,j*) = 0:
set AttList := AttList$\cup${v*,j*) and AttBit(v*,j*) = 1. If now |AttList| $\geq$ AttLim, set NA := NA$\cap$AttList.

I2. Execute *Set Bit*(TabuList0(v*,j*)) and let TabuSave = TabuList(v*,j*).
Then for each (v,j) $\in$ CompoundList(q), q = 1, …, iter0 –1 such that TabuBit(v,j) = 0:
Add (v*,j*) to TabuList(v,j) and add (v,j) to TabuList(v*,j*).
Execute *Re-Set Bit*(TabuSave) and set CompoundList(iter0) = (v*,j*)
If $|I_\beta*| = 0$ (equivalently $b_{v*j*} = 0$) terminate the Inner Loop. Otherwise, update the surrogate constraint coefficients a_{vj} and b_{vj} (relative to I_α and $I_\beta*$) and return to Step I1.

End of Inner Loop

2. Set $G_\alpha* := G_\alpha* \cup I_\alpha$ and $G_\alpha := G_\alpha – I_\alpha$. $G_\beta := G_\beta \cup I_\beta*$, $G_\beta* := G_\beta* – I_\beta*$
3. If $|G_\alpha*| \geq f| G_\alpha^{\,o}|$ or if or NA = $\varnothing$ proceed to Step 4. Otherwise, determine the updated values $a_{vj} = |\{i \in G_\alpha: x_j^{\,i} = v\}|$ and $b_{vj} = |\{i \in G_\beta*: x_j^{\,i} = v\}|$, for (v,j) $\in$ NA and return to Step 1.

4. Set p := p + 1 and for H(p) = {h_o, …, h} generate the inequality (1.1-H(p)) satisfied by all x = x^i for i ∈ G_α* and i ∈ G_β and the inequality (1.2-H(p)) satisfied by all x = x^i for i ∈ G_α and i ∈ G_β* given by

$$\sum(y_h: h \in H(p)) \geq 1 \qquad (1.1\text{-}H(p))$$
$$\sum(y_h: h \in H(p)) \leq 0 \qquad (1.2\text{-}H(p))$$

End of Core Method

2.0 If p ≥ L, then stop. Otherwise return to Step 1.0.
End of Method

10. Tree-Based SAT-DM Methods

The inequalities generated by the SAT-DM method[9] can be used to create a tree-based classification process using the design for such a process in the context of hyperplane separation methods. We discuss only the basic ideas of such a design here, and refer the reader to Glover (2006) for a more advanced treatment.

Theoretically, the rules of a decision tree process within the present framework will generate a collection of logical clauses equivalent to those produced by the compound variable method, assuming every possible decision tree and every possible compound variable inequality is generated. (This follows from the fact that the rules of a binary decision tree can be re-expressed as a set of logical clauses whose disjunctive normal form corresponds to a collection of inequalities that can be represented in the same way as (1.1-H(p)) and (1.2-H(p).) Given that the collection of all possible decision trees (and all compound variable inequalities) can be exponentially large, the subset generated by a practical method can be much smaller than the total, and hence a Compound Variable method and a tree-based method may produce somewhat different results.

We identify the fundamental structure of a SAT-DM decision tree approach as follows. Let $G_\alpha^o(s)$ and $G_\beta^o(s)$ denote the subsets of G_α^o and G_β^o associated with a given node s of the decision tree, where s = 1 identifies the initial (root) node of the tree (and hence $G_\alpha^o(1) = G_\alpha^o$ and $G_\beta^o(1) = G_\beta^o$). Each node s is the source of two branches, the first branch corresponding to the inequality (1.1(p)) and the second branch corresponding to the inequality (1.2(p)). Let G_α*(s), G_β*(s), G_α(s), and G_β(s) denote the sets identified as G_α*, G_β*, G_α, and G_β in Step 4 (or Step D5) of the SAT-DM method, at the point of generating (1.1(p)) and (1.2(p)). Thus, in particular, G_α*(s) and G_β(s) constitute the portions of the original sets $G_\alpha^o(s)$ and $G_\beta^o(s)$ whose points satisfy (1.1(p)), and G_β*(s) and G_α(s) constitute the portions of $G_\alpha^o(s)$ and $G_\beta^o(s)$ whose points satisfy (1.2(p)). (We continue to adopt the convention of referring to a point x^i as belonging to a set, with the interpretation that its index i belongs to the set.)

[9] In this section, the term "SAT-DM method" will refer to the version for the SAT(QC/AC) problem described in Section 8, but it can also refer to the Compound Variable Version of this method by means of evident qualifications, such as replacing the inequalities (1.1(p)) and (1.2(p)) by the inequalities (1.1-H(p)) and (1.2-H(p)),

New decision nodes at the end of these two branches for node s are denoted by s′ = s+1 and s′ = s+2, where the node s′ for the first branch (for (1.1(p))) is associated with the new sets $G_\alpha^o(s') = G_\alpha^*(s)$ and $G_\beta^o(s') = G_\beta(s)$, and the node s′ for the second branch (for (1.2(p))) is associated with the new sets $G_\alpha^o(s') = G_\alpha(s)$ and $G_\beta^o(s') = G_\beta^*(s)$. (Consequently, each node represents the portions of $G_\alpha^o(s)$ and $G_\beta^o(s)$ whose points satisfy the inequalities associated with its respective branch.) To continue the decision tree construction, we now perform additional executions of the Core Method, not by repeatedly generating new inequalities for the original $G_\alpha^o(s)$ and $G_\beta^o(s)$ sets, but instead by generating inequalities for each of the residual sets $G_\alpha^o(s')$ and $G_\beta^o(s')$. This produces a conditional separation process, generally shrinking the size of the sets as the depth of the tree increases.

In an ideal situation the branches created for a node s will produce a perfect separation of its sets $G_\alpha^o(s)$ and $G_\beta^o(s)$. In this case, upon reaching Step 4 (or D5) of the Core Method we will have $G_\alpha^*(s) = G_\alpha^o(s)$ and $G_\beta^*(s) = G_\beta^o(s)$ (hence $G_\alpha(s)$ and $G_\beta(s)$ are both empty). Consequently, all points of $G_\alpha^o(s)$ satisfy (1.1(p)) and all points of $G_\beta^o(s)$ satisfy (1.2(p)), and the nodes s′ = s+1 and s′ = s+2 are terminal nodes of the tree. A terminal node can also occur on just one of the two branches. For example, if $G_\alpha^*(s) = G_\alpha^o(s)$ (and hence $G_\alpha(s) = \varnothing$) but $G_\beta^*(s) \neq G_\beta^o(s)$ (and hence $G_\beta(s) \neq \varnothing$), then the node s′ of the first branch has both of its new sets $G_\alpha^o(s')$ and $G_\beta^o(s')$ nonempty, but node s′ of the second branch has only $G_\beta^o(s')$ nonempty. This means all points of the second branch are correctly classified as belonging to $G_\beta^o(s)$, and more generally as belonging to the original G_β^o itself. Consequently, the node s′ of the second branch is a terminal node, while the node s′ of the first branch is not. In reverse, if $G_\beta^*(s) = G_\beta^o(s)$ but $G_\alpha^*(s) \neq G_\alpha^o(s)$, then all points associated with the node s′ of the first branch are correctly classified as belonging to $G_\alpha^o(s)$, but the node s′ of the second branch contains points of both $G_\alpha^o(s)$ and $G_\beta^o(s)$, and hence the former node is a terminal node terminal node while the latter is not.

A terminal node can be produced in one other situation, where the sets $G_\alpha^o(s')$ and $G_\beta^o(s')$ for one of the nodes s′ are both empty. Hence these sets for the other node s′ are given by $G_\alpha^o(s') = G_\alpha^o(s)$ and $G_\beta^o(s') = G_\beta^o(s)$, disclosing that no differentiation between the starting sets $G_\alpha^o(s')$ and $G_\beta^o(s)$ has been achieved. We call this latter node an *unresolved terminal node*, because it contains points that have not been classified. We also call the points associated with this node *unresolved points*. By contrast, each of the terminal nodes produced in the tree preceding situations are called *resolved terminal nodes*, because all points in their final sets are correctly classified, and we call their points *resolved points*. In practice, a node can be treated as a terminal node if it lies at a sufficiently large depth from the root even if one of the preceding conditions does not occur. In this case we also identify it as a unresolved terminal node and identify its associated points as unresolved points.

Organization to Generate Multiple Decision Trees.

The SAT-DM method can thus be used to generate multiple decision trees as a basis for a global decision rule. We first observe that the uses of the tabu conditions to influence the choice rules for iter = 1 and iter = 2 in the Constructive Component of the method can be carried over to generating multiple decision trees as follows. Each time the construction of a new tree is launched, the choice rule for iter = 1 is applied to the root node s = 1 in generating the two branches s′ = 2 and 3. Then the choice rule for iter = 2 is applied in generating the branches derived from each of these latter nodes.

Beyond this, we organize the multiple tree generation process so that future trees give a higher priority to correctly classifying points that were unresolved in previous trees. To do this, a record is kept for each point, relative to a given tree that gives the depth of the terminal node where this point ends up. The depths for unresolved terminal nodes are increased by adding to them the value of the maximum depth of a resolved terminal node. Thus, unresolved nodes are assigned greater depth values than resolved nodes.

Then, on a pass to generate a new tree, each point is assigned a priority value equal to the sum of its depth values over previously generated trees. (The sum can be weighted so that trees generated more recently receive larger weights than those generated farther in the past.) Points that were more often (or more recently) unresolved receive greater priority values than those that were more often (or more recently) resolved. Likewise, resolved nodes whose resolution occurred at greater depths over previous trees receive greater priority values than those whose resolution occurred at smaller depths.

These priority values are used to produce a new tree that gives greater emphasis to correctly classifying higher priority points, and classifying them at earlier depths of the tree, than it gives to the lower priority points. The mechanism for accomplishing this is provided by the surrogate constraint choice rule, by generating the surrogate constraint coefficients a_{vj} and b_{vj} in relation to weights produced by the priority values. The process results by replacing the previous "simple sum" surrogate constraint choice rule with one based on introducing a weight w_i for each point x^i as a basis for creating the a_{vj} and b_{vj} coefficients. Thus, we write

$$a_{1j} = \sum(x_j^i w_i : i \in G_\alpha) \text{ and } a_{0j} = \sum(\tilde{x}_j^i w_i : i \in G_\alpha)$$

$$b_{1j} = \sum(w_i x_j^i : i \in G_\beta^*) \text{ and } b_{0j} = \sum(w_i \tilde{x}_j^i : i \in G_\beta^*).$$

where $w_i > w_{i'}$ if x^i receives a higher priority than $x^{i'}$. The weights can be chosen, if desired, to give a pre-emptive emphasis on some points over others, in effect creating different b_{vj} values for different sets of component constraints, and then assigning a strict priority to the order in which these values are considered by the choice criterion arg $\max(a_{vj}/(b_{vj}^e + \varepsilon) : (v,j) \in NA)$. This altered definition of the a_{vj} and b_{vj} coefficients entails a slight increase in computation, but produces decision trees that are collectively structured to produce more effective classifications, given that it is out of the question to

generate more than a small fraction of the enormous number of decision trees that are possible.

Global Decision Rule.

A global decision rule for classifying a new point of unknown membership operates as follows. Each decision tree casts a vote for assigning the point to membership in G_α^o or G_β^o. If the branching process of a given tree assigns a point to either G_α^o or G_β^o by placing it in a resolved terminal node, then the point receives a vote of 1 for the indicated membership.

But if a point winds up being assigned to an unresolved terminal node s′, then it receives a vote of $|G_\alpha^o(s')|/|G_\alpha^o(s')| + |G_\beta^o(s')|)$ for membership in G_α^o and a vote of $|G_\beta^o(s')|/|G_\alpha^o(s')| + |G_\beta^o(s')|)$ for membership in G_β^o. Finally, the membership category for the point is determined as the one that receives the greatest number of votes over all trees.[10]

Variants arise by combining the preceding approach with the form of the SAT-DM method described in Section 8 (that does not rely on a tree-based decision process). In this ease, the tree is truncated by restricted its generation to a relatively small depth. Each unresolved terminal node s′ of the truncated tree then becomes the source of multiple inequalities by treating the two sets $G_\alpha^o(s')$ and $G_\beta^o(s')$ exactly as if they were the sets G_α^o and G_β^o of the method of Section 8.

The ability of this decision tree process to uncover possible useful classification rules that are not equivalent to those produced by the Compound Variable Method, and vice versa, encourages the use of these two approaches in tandem.

11. Conclusions

Satisfiability data mining provides a method for separating and classifying points that opens up new avenues for determining group membership. The main contributions of this paper, in addition to linking logical clauses and their associated inequalities to group separation by means of the Fundamental Observation, may be summarized as follows:

- Identifying effective ways to generate a subset of preferred inequalities from the full collection possible, making use of surrogate constraint analysis and memory-based designs of tabu search.
- Handling the feature (attribute) selection problem simultaneously, as a natural concomitant of generating preferred inequalities, and without requiring the solution of an auxiliary optimization problem.
- Introducing a coordinated procedure that accommodates criteria for balancing trade-offs between satisfying inequalities defining membership in the two groups

[10] The two groups G_α^o and G_β^o can be interchanged to carry out this process.

being compared, including a staged version of the procedure that offers a useful alternative from a conservative perspective.

- Providing an related process that generates compound variables to enhance the ability to separate selected groups, utilizing surrogate constraint evaluations in a manner that permits the determination of appropriate compound variables again without the need to solve an auxiliary problem or resort to an external algorithm.
- Disclosing how to embed the classification process in a decision tree framework by an adaptation of processes designed to produce decision trees for separating hyperplane analysis.

Empirical research to explore the promise of these new developments will be the topic of future work.

References

Albanedo, J., and C. Rego (2005) "Surrogate Constraint Normalization for the Set Covering Problem," School of Business Administration, University of Mississippi.

Balas, E. and A. Ho (1980) "Set Covering Algorithms Using Cutting Planes, Heuristics and Subgradient Optimization: A Computational Study," *Mathematical Programming* 12, pp. 37-60.

Balas, E. and R. Jeroslow (1972) "Canonical Cuts on the Unit Hypercube," *SIAM Journal of Applied Mathematics, 23*, No. 1, pp. 60-69.

Bennett, K. P. and J.A. Blue (1998) "A Support Vector Machine Approach to Decision Trees," *Neural Networks Proceedings, IEEE World Congress on Computational Intelligence,* Vol. 3, pp. 2396-2401.

Better, M., F. Glover and M. Samorani (2006) "Multi-Hyperplane Formulations for Classification and Discrimination Analysis," Research Paper, University of Colorado, Presented at the Western Decision Sciences Institute, 36th Annual Meeting, Denver, CO.

Boros, E., P. L. Hammer, T. Ibaraki, A. Kogan, E. Mayoraz and I. Muchnik (2000a) "An Implementation of Logical Analysis of Data," IEEE Transactions on Knowledge and Data Engineering, Vol 12, No. 2, pp. 292-306.

Boros, E. T. Horiyama, T. Ibaraki, K. Makino and M. Yagiura (2000b) "Finding Small Sets of Essential Attributes in Binary Data," *RUTCOR Research Report*, RRR-13-2000, Rutgers University.

Caprara, A. M. Fischetti and P. Toth (1999) "A Heuristic Method for the Set Covering Problem," *Operations Research*, Vol 47, pp 730-743.

Christiani, N. and J. Shawe-Taylor (2000) *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*, Cambridge University Press, Cambridge, UK.

Chvátal, V. (1979). A greedy heuristic for the set covering problem," *Mathematics of Operations Research*, Vol. 4, pp. 233-235.

Dai, H. (2004) *Advances in Knowledge Discovery and Data Mining*, Springer Publishing.

Feo, T. and M. Resende (1989) "A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem," *Operations Research Letters*, Vol. 8, pp. 67-71.

Fréville, A. and G. Plateau (1993) "An exact search for the solution of the surrogate dual of the 0-1 bidimensional knapsack problem," *European Journal of Operational Research*, Vol. 68, pp. 413-421.

Gavish, B. and H. Pirkul (1985) "Efficient algorithms for solving multiconstraint zero-one knapsack problems to optimality," *Mathematical Programming*, Vol. 31, 78-105.

Gavish, B., F, Glover and H. Pirkul (1991) "Surrogate constraints in integer programming," *Journal of Information and Optimization Sciences*, Vol. 12, No. 2, 219-228.

Glen, J.J. (2003) "An iterative mixed integer programming method for classification accuracy maximizing discriminant analysis," *Computers and Operations Research,* 30/181-198.

Glover, F. (1965) "A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem," *Operations Research*, Vol. 13, No. 6, pp. 879-919.

Glover, F. (1968). Surrogate Constraints. *Operations Research*, Vol. 16, No. 4, pp, 741-749.

Glover, F. (1975). Surrogate Constraint Duality in Mathematical Programming. *Operations Research*, Vol. 23, No. 3., pp. 434-451.

Glover, F. (1977). Heuristics for integer programming using surrogate constraints. *Decision Sciences*, Vol. 8, pp. 156-166.

Glover, F. (1990) "Improved Linear Programming Models for Discriminant Analysis," *Decision Sciences*, Vol, 21, No. 4, pp. 771-785.

Glover, F. (2003) "Tutorial on Surrogate Constraint Approaches for Optimization in Graphs," *Journal of Heuristics,* Kluwer Academic Publishers, Boston, pp. 175-228, 2003.

Glover, F. (2006) "Improved Classification and Discrimination by Successive Hyperplane and Multi-Hyperplane Separation," Research Report, University of Colorado, Boulder.

Glover, F. (2008) "Inequalities and Target Objectives for Metaheuristic Search – Part I: Mixed Binary Optimization," in *Advances in Metaheuristics for Hard Optimization*, P. Siarry and Z. Michalewicz, eds. Springer, New York, pp. 439-474.

Glover, F. and G. Kochenberger (2006) “New Optimization Models for Data Mining,” *International Journal of Information Technology & Decision Making (IJITDM)*. Vol. 5, No. 4, pp. 1-6.

Glover, F. and M. Laguna (1997) *Tabu Search*, Kluwer Academic Publishers, Boston.

Greenberg, H.J. and W.P. Pierskalla (1970). Surrogate Mathematical Programs. *Operations Research*, Vol. 18, 924-939.

Greenberg, H.J. and W.P. Pierskalla (1973). Quasi-conjugate functions and surrogate duality. *Cahiers due Centre d’Etudes de Recherche Operationelle*, Vol.15, 437-448.

Kochenberger, G., A. McCarl and F. Wyman (1974) “A Heuristic for General Inteeger Programming,” *Decision Sciences*, Vol. 5, No. 1, pp. 36-45.

Løkketangen, A. and F. Glover. (1997) “Surrogate Constraint Analysis - New Heuristics and Learning Schemes for Satisfiability Problems,” *Satisfiability Problem: Theory and Applications. DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Eds.: D. Du, J. Gu, and P. M. Pardalos, Vol. 35, pp. 537-572.

Schlkopf, B. and A. J. Smola (2002) *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond (Adaptive Computation and Machine Learning)*, Cambridge, MA: MIT Press.

Wang, L. (2005) *Support Vector Machines: Theory and Applications*, Springer-Verlag, New York.

Yagiura, M., M. Kishida and T. Ibaraki (2006) ``A 3-Flip Neighborhood Local Search for the Set Covering Problem," *European Journal of Operational Research*, Vol. 172. pp. 472-499.