Surrogate Constraint Analysis for New Heuristics and Learning Schemes for Satisfiability Problems

Arne Løkketangen and Fred Glover

ABSTRACT. Surrogate constraint analysis has been applied effectively to a variety of combinatorial optimization problems, as a foundation for both exact and heuristic methods. In the heuristic domain, surrogate constraint methods are particularly suited to the creation of associated learning procedures and to the application of probabilistic decisions. We show that these approaches are natural and effective for satisfiability (SAT) problems. Added motivation comes from observing that the current best exact and heuristic procedures for multidimensional knapsack problems are provided independently by surrogate constraint methods and probabilistic methods that use memory and learning structures (derived from tabu search). We show that the SAT problem can be formulated as a special instance of a binary-choice multidimensional knapsack problem (or equivalently, a binary-choice generalized covering problem), and demonstrate how surrogate constraint analysis can be specialized in a particularly convenient way to exploit the structure of this problem. Our approach incorporates simple (first order) instances of adaptive memory structures characteristic of tabu search implementations, to give a learning effect to guide the search. This use of memory adds a dimension to the solution process that has not adequately been examined in the past.

We find that the combination of surrogate constraint analysis and simple learning proves more effective than probabilistic search designs, including those that encompass probabilistic rules that have been highly favored in previous SAT approaches. These outcomes motivate a closer look at surrogate strategies and more advanced ways of integrating them with adaptive memory and learning procedures.

1. Introduction

Surrogate constraints are designed to capture information from an original system of constraints that is not available from its individual members. This information capturing function is achieved by strategies for combining and processing the original constraints to create new ones, with the goal of analyzing and tracing consequences of the original system that may not otherwise be readily accessible. Surrogate constraint analysis has established itself as a useful tool for exploiting system structures, with particular application to combinatorial search. (See, for example, Glover (1965, 1975), Greenberg and Pierskalla (1970, 1973), Karwan and Rardin (1979), Gavish and Pirkul (1985),

1991 Mathematics Subject Classification. Primary 05A99; Secondary 05-04

.

Freville and Plateau (1993), Yu 1994)). For a survey on algorithms for the satisfiability problem, see Gu at. al. (1995).

Our goal in the present paper is to show how surrogate constraint analysis can be used to guide heuristic approaches for the satisfiability problem. In particular, we give surrogate constraint generation procedures and associated decision rules to construct advanced starting solutions, and then to provide a basis for more refined improvement approaches. We show that our surrogate constraint design is conveniently suited to incorporate learning procedures for generating improved surrogate constraints, and hence for providing better decision rules, as the solution effort continues.

Experience with surrogate constraint approaches for combinatorial optimization problems has shown that using different weights for different constraints, by well-defined processes, proves significantly more effective than choosing all weights the same. Our study examines the outcome of using weights prescribed by these more general processes, specifically making use of weights that are fast to update, involving the same order of computation as required when all weights are chosen equal to 1. This use of surrogate constraints also has the side benefit of exploiting the presence of dominated and "nearly-dominated" constraints, without the effort of preprocessing to identify such constraints.

The application of surrogate constraint analysis to the SAT problem (by our formulation) discloses an interesting connection between this analysis and the "move evaluation" rules of the popular GSAT code (see Selman et al. 1992), and its more recent extensions: the GSAT rules arise as a simple instance of rules standardly used in surrogate constraint methods, by restricting weights for all constraints to equal 1. Thus, our approach establishes a continuity with the approaches derived from GSAT, yielding a larger class of choices by surrogate constraint analysis.

We are especially interested in combining surrogate constraint analysis with adaptive memory mechanisms as proposed in tabu search, as a basis for establishing learning effects. In this paper we examine only very simple first order uses of such memory, which we implement both with deterministic and probabilistic decision rules. The tabu search orientation gives an appealing foundation for the probabilistic element, because the rules of probabilistic tabu search (PTS) bear the same relation to the probabilistic component of GSAT as the evaluation rules of surrogate constraint analysis bear to the evaluation rules of GSAT; i.e., GSAT implicitly chooses all probabilities for the moves on its candidate list to be the same, whereas PTS employs a rationale that causes these probabilities (potentially) to be different (See, for example, Løkketangen and Glover, 1995). (Recent extensions of GSAT, which use a candidate list known as an "essential move" candidate list, implicitly use two different probabilities.) A natural continuum therefore also results between the probabilistic component of our approach and approaches popularly applied to SAT problems, by means of a framework that offers broader strategic options than previously investigated. Our study examines a few simple PTS variants for selecting probabilities, including designs that have proven effective in other combinatorial optimization settings.

Computational experiments are reported for different variants of our surrogate constraint approach, applied to satisfiability problems from a benchmark testbed. From this, we identify surrogate constraint strategies that prove effective in the satisfiability context, and indicate possible extensions of our approaches to other combinatorial problem settings.

The layout of this paper is as follows. After this introduction, a mathematical programming formulation of the satisfiability problem is given in Section 2. The

fundamentals of surrogate constraints are given in Section 3, and the use of surrogate constraints to guide a constructive heuristics is described in Section 4. Section 5 similarly gives ways for using surrogate constraints in an iterative improvement search phase. In Section 6 learning is introduced to give additional guidance, and Section 7 outlines aspects of probabilistic tabu search. Section 8 relates the use of surrogate constraints to other approaches and Section 9 gives computational results. Finally conclusions and directions for further research are in Section 10 and references in Section 11.

2. Formulation.

The satisfiability problem can be given a mathematical programming formulation in several ways, each seeking a feasible solution to a system of inequalities, with implicit or explicit inclusion of binary choice constraints. A standard representation replaces literals by variables x_j and their complements $1 - x_j$, to give rise to the system

$$\text{(I)} \qquad \begin{aligned} &\mathbf{A}x \geq b \\ &x \text{ binary} \end{aligned}$$

where **A** is an $m*n$ matrix of 0´s, 1´s and –1´s, and b and x are n*1 column vectors. The i^{th} constraint of the system,

$$\mathbf{A}_i x \geq b_i$$

has the property that the number of –1´s in the row vector $\mathbf{A}_i$ equals $1 - b_i$, where b_i is an integer ≤ 1.

Equivalent formulations can be created by a device that allows any pure zero-one integer linear programming problem to be expressed as a generalized covering problem or a multidimensional knapsack problem, subject to binary choice constraints. Specifically, we rewrite an arbitrary **A** matrix as the difference of two nonnegative matrices, $\mathbf{A} = \mathbf{P} - \mathbf{Q}$, where the nonzeroes of **P** consist of the positive entries of **A** and the nonzeroes of **Q** consist of the absolute values of the negative entries of **A**. Then, for $d = b + \mathbf{Q}e$, where e represents the column n-vector of 1´s, the system (I) can be expressed in the following binary choice generalized covering formulation:

$$\text{(GC)} \qquad \begin{aligned} &\mathbf{P}x + \mathbf{Q}y \geq d \\ &x + y = e \\ &x, y \text{ binary} \end{aligned}$$

The covering terminology results because of the nonnegativity of **P** and **Q** (where $d > 0$ may be assumed after removing trivially redundant inequalities), and the binary choice terminology results because the equality constraints have the form $x_j + y_j = 1$, presenting a binary choice between the two variables x_j and y_j.

For the SAT problem, d is an m vector of 1´s, and the inequalities of the system are those of a simple covering problem (since **P** and **Q** in this case are matrices of 0´s and 1´s).

A corresponding expansion, based on expressing the original system in the form $\mathbf{A}´x \leq$ b´, where $\mathbf{A}´ = -\mathbf{A}$ and $b´ = -b$, creates the following multidimensional knapsack representation:

(MK)
$$\begin{aligned} &\mathbf{Q}x + \mathbf{P}y \le g \\ &x + y = e \\ &x, y \text{ binary} \end{aligned}$$

The matrices **P** and **Q** here are the same as in the (GC) formulation, and $g = \mathbf{P}e - b$. The (MK) formulation also can be obtained by substituting $e - x$ for y and $e - y$ for x in (GC), noting that g can also be written as $g = \mathbf{P}e + \mathbf{Q}e - d$. The multidimensional knapsack terminology results because of the nonnegativity of the problem matrices and vectors (where $g \ge 0$ is required to assure feasibility) which causes the formulation to adhere to a standard multidimensional knapsack format, with the inequality portion of the system in the reverse direction from that of (GC). It should be noted that the composition of **P** and **Q** in both (GC) and (MK) is not rigidly determined, in the sense that the columns $\mathbf{P}_j$ and $\mathbf{Q}_j$ of **P** and **Q** can be interchanged by renaming their associated variables x_j and y_j.

Although the three formulations (I), (GC) and (MK) are entirely equivalent, the representations of (GC) and (MK) are useful for enabling certain connections to be seen more readily. The (GC) formulation is clearly a natural one in the present context, since it resembles the statement of the SAT problem in its customary logical representation. We will see that it has other conveniences for analyzing the SAT problem. The (MK) formulation, however, can be useful for generating insights in other settings. (An interesting exercise is to translate the procedures we subsequently specify for the (GC) formulation into equivalent steps for the (MK) formulation.)

In this paper we apply surrogate constraint analysis to the SAT problem in the following two ways:

(1) to generate a collection of advanced starting solutions by a constructive approach—including a component that is able to learn how to modify previous decisions.

(2) to introduce a related exchange procedure for obtaining improved solutions, similarly based on surrogate constraint evaluations and learned modifications, to extend the advanced starting approach to yield a general heuristic.

In the process we discuss a number of strategic considerations that are also relevant to other types of combinatorial optimization problems.

3. Surrogate constraint fundamentals.

The most common form of surrogate constraint generation, which we employ here, results by creating surrogate constraints as nonnegative linear combinations of the constraints of the original system (expressed as inequalities). For convenience we write the inequality portion of (GC) in a compact form by letting $\mathbf{D} = (\mathbf{P}\ \mathbf{Q})$ and $z = (x\ y)^{\mathrm{T}}$ (hence, $\mathbf{D}_{j+n} = \mathbf{Q}_j$ and $z_{j+n} = y_j$), to yield the representation

$$\mathbf{D}z \ge d$$

Then we introduce a nonnegative vector w of weights w_i, to generate a surrogate constraint

$$sz \ge s_o$$

where $s = w\mathbf{D}$ and $s_o = wd$. (The surrogate constraint therefore results by weighting each constraint $\mathbf{D}_i z \ge d_i$ by w_i and summing.) The component constraints $x_j + y_j = 1$ of $x + y =$

e (and more particularly the associated inequalities $-x_j + -y_j \geq -1$) will also be used to create the final form of the surrogate constraint. As will soon be seen, the appropriate weights in this case are evident and we do not need additional notation to represent them.

3.1. Surrogate constraint uses for the SAT problem. Since the surrogate constraint $sz \geq s_o$ is implied by the original system, any solution satisfying (GC) must also satisfy the surrogate constraint. Assuming the weights for generating the constraint can effectively capture the "combined influence" of the inequalities $\mathbf{D}_i z \geq d_i$, then the variable z_j that provides the greatest contribution to satisfying $sz \geq s_o$ may also provide the greatest contribution to satisfying $\mathbf{D}z \geq d$. This statement is heuristic, but in practice a variety of choices of weights often exist that support its general premise. Situations in which several different key variables z_j must be assigned a value of 1 to satisfy (GC), for example, may be accompanied by the existence of different weightings that cause each such z_j in turn to appear the most important for satisfying the surrogate constraint—and it is only necessary to identify one of these weightings to make a valid choice for assigning a value to a variable. An illustration of this is provided shortly.

Superficially, it may be supposed that the variable z_j with the largest coefficient s_j in the surrogate constraint $sz \geq s_o$ would provide the greatest contribution to satisfying this constraint. However, the binary choice constraints must first be taken into account. Let $j_\#$ denote the index of the variable that is the complement of z_j (hence $j_\# = j + n$ if z_j corresponds to an element of x, and $j_\# = j - n$ if z_j corresponds to an element of y). Then the binary choice constraint $z_j + z_{j\#} = 1$ discloses that the *relative contribution* of z_j is not s_j but $s_j - s_{j\#}$, since selecting $z_j = 1$ forgoes the opportunity of selecting $z_{j\#} = 1$. Hence, by adopting such an opportunity cost criterion, we may judge the most attractive variable z_j to be one that maximizes the value of $s_j - s_{j\#}$.

This analysis corresponds to using the inequalities $-z_j - z_{j\#} \geq -1$ to give the "final form" of the surrogate constraint, where we choose a weight for each such inequality of $\text{Min}(s_j, s_{j\#})$. Then one of z_j or $z_{j\#}$ will have a coefficient equal to 0 and the other will have a coefficient equal to the absolute value of $s_j - s_{j\#}$ in the resulting surrogate constraint. Choosing the largest coefficient in this constraint corresponds to choosing a variable that yields a maximum value of $s_j - s_{j\#}$ We will call this surrogate constraint (which has a coefficient for each z_j of $s_j - \text{Min}(s_j, s_{j\#})$) a *derived* surrogate constraint.

AN EXAMPLE. We illustrate the preceding considerations by a system that consists of 5 inequalities and 3 pairs of complementary variables: (z_1,z_4), (z_2,z_5) and (z_3,z_6). The weight associated with each inequality is listed to its right, below. (Note that in such an inequality system representing the (GC) formulation, complementary variables z_j and $z_{j\#}$ cannot appear together in any inequality; i.e., the columns $\mathbf{D}_j$ and $\mathbf{D}_{j\#}$ cannot have unit coefficients in the same row.)

$$
\begin{array}{cccccccccccccc}
z_1 & + & z_2 & & & & & & & & & \geq & 1 & (w_1) \\
z_1 & & & + & z_3 & & & & & & & \geq & 1 & (w_2) \\
 & & z_2 & + & z_3 & & & & & & & \geq & 1 & (w_3) \\
 & & & & & z_4 & + & z_5 & & & & \geq & 1 & (w_4) \\
z_1 & & & & & & & & + & z_6 & & \geq & 1 & (w_5)
\end{array}
$$

We generate three surrogate constraints for this example, the first by setting all $w_i = 1$ (which simply sums the preceding inequalities), the second by setting $w_2 = 1$, $w_3 = 1$ and $w_4 = 1$ (with other weights 0), and the third by setting $w_2 = 1$, $w_4 = 2$ and $w_5 = 1$. These constraints and their corresponding derived surrogate constraints are shown as follows:

$$
\begin{array}{llllllllllll}
3z_1 & + & 2z_2 & + & 2z_3 & + & z_4 & + & z_5 & + & z_6 & \geq & 5 & (1) \\
z_1 & + & z_2 & + & 2z_3 & + & z_4 & + & z_5 & + & z_6 & \geq & 3 & (2) \\
2z_1 & & & + & z_3 & + & 2z_4 & + & 2z_5 & + & z_6 & \geq & 4 & (3) \\
 & & & & & & & & & & & & & \\
2z_1 & + & z_2 & + & z_3 & & & & & & & \geq & 2 & (1\text{-derived}) \\
 & & & & z_3 & & & & & & & \geq & 1 & (2\text{-derived}) \\
 & & & & & & & & 2z_5 & & & \geq & 1 & (3\text{-derived})
\end{array}
$$

Surrogate constraints (1) and (2) each have a unique maximum coefficient, $s_1 = 3$ and $s_3 = 2$, respectively, which suggests that z_1 and z_3 may be good variables to receive a value of 1. No clear choice of a "good variable" emerges from surrogate constraint (3). On the other hand, a clear winner emerges in each of the three derived surrogate constraints. In fact the second and third derived constraints demonstrate that z_3 and z_5 may be compelled to equal 1.

Although it can be valuable to identify compulsory possibilities, we will not undertake to use surrogate constraints in the present setting to determine when variables may be forced to receive a value of 1, but instead will focus on using these constraints to provide information for making choices. This policy stems from two related reasons. First, simple (if sometimes less powerful) tests exist for determining compulsory assignments by reference to updated forms of the original inequalities of (GC). Second, it can be somewhat time consuming to discover a surrogate constraint that yields a compulsory assignment (assuming one can be found) -- and, more importantly from a heuristic standpoint, a less restrictive constraint may yield information that is just as good for determining when a variable should receive a particular value.

In the present illustration, for example, there exists a surrogate constraint that discloses z_1 is compelled to equal 1, which is the derivative of the surrogate constraint obtained by setting $w_1 = 1$, $w_2 = 1$ and $w_5 = 2$. The less restrictive surrogate constraint (1) and its derived constraint, which do not yield this information, nevertheless identify z_1 as a preferred variable to select.

3.2 A specific choice rule. In combinatorial problems, subtle differences in evaluations can sometimes have significant consequences, and therefore it can be useful to construct several different surrogate constraints (by different weightings) and allow them to "vote" on a preferred variable to set equal to 1. For purposes of speed we will confine such a voting process only to the single surrogate constraint $sz \geq s_o$ and its derived surrogate constraint, making the derived constraint the dominant partner. In particular, we will select a maximum $s_j - s_{j\#}$ value (equivalently, a maximum $s_j - \text{Min}(s_j, s_{j\#})$ value) and break ties in favor of larger s_j values. This may be expressed as a parameterized choice rule which chooses a variable z_j to set to 1 that yields a maximum of $ps_j + (s_j - s_{j\#})$ where the parameter p is given a very small positive value. Hereafter we will refer to this rule as the *surrogate constraint choice rule*. Since all the information necessary for this choice is available from the single constraint $sz \geq s_o$, we will continue to speak of it as the surrogate constraint that guides our approach.

Dominance Connections: -- We now motivate the surrogate constraint choice rule more rigorously. If the binary choice constraints are eliminated, leaving the system $\mathbf{D}z \geq d$ subject only to z binary, the result is the set of constraints for a classical covering problem. Since we are interested only in feasibility, without concern for an objective function, a standard indication that a variable z_j dominates another variable z_h is $\mathbf{D}_j \geq \mathbf{D}_h$, which implies we may enforce $z_j \geq z_h$. In addition, given the unit vector d, setting $z_j = 1$ renders $z_h = 1$ superfluous.

This type of dominance does not carry the same interpretation for the binary choice covering formulation of (GC), and so we will refer to it as *quasi-dominance*. Instead, to assure (full) dominance in (GC) we require the two conditions $\mathbf{D}_j \geq \mathbf{D}_h$ and $\mathbf{D}_{j\#} \leq \mathbf{D}_{h\#}$. In the context of (GC), these conditions allow us to conclude $z_j \geq z_h$ (and symmetrically, $z_{h\#} \geq z_{j\#}$).

Naturally, it would be advantageous to identify quasi-dominance and dominance when they occur. However, the effort of checking for such conditions is prohibitive, particularly when repeated each time the system $\mathbf{D}z \geq d$ changes during a constructive procedure (as a result of removing assigned variables and redundant constraints). Fortunately, surrogate constraint analysis effectively makes it possible to avoid this effort. That is, the existence of these dominance conditions automatically causes $z_j = 1$ to be preferentially selected over $z_h = 1$ by relying on the surrogate constraint $sz \geq s_o$ and its derived surrogate constraint to determine the choice. We state this result as follows.

REMARK 1. Quasi-dominance of z_j over z_h implies the coefficient condition $s_j \geq s_h$ holds in the surrogate constraint $sz \geq s_o$, and (full) dominance of z_j over z_h implies the coefficient condition $s_j - s_{j\#} \geq s_h - s_{h\#}$ holds in the derived surrogate constraint. In addition, if $w > 0$, the inequalities indicated for the surrogate constraint coefficients are strict whenever the dominance conditions are nonreversible (i.e., do not allow z_h to dominate z_j).

PROOF. $\mathbf{D}_j \geq \mathbf{D}_h$ implies $w\mathbf{D}_j \geq w\mathbf{D}_h$ for $w \geq 0$, and also implies $w\mathbf{D}_j > w\mathbf{D}_h$ if $\mathbf{D}_j \neq \mathbf{D}_h$] and $w > 0$. The conclusions of the remark then result by applying the definitions for generating the surrogate constraint coefficients.

Remark 1 shows that the surrogate constraint choice rule will choose $z_j = 1$ in preference to $z_h = 1$ whenever dominance occurs, and will break ties to reflect the influence of quasi-dominance. The result also readily generalizes to problems with explicit objective functions. Of course, the choice rule also yields guidance in situations where dominance does not occur.

3.3. Determining the Surrogate Constraint Weights. No matter how subtle the choice rule, its effectiveness must ultimately depend on selecting surrogate constraint weights appropriately. Methods for determining weights w_i take several forms. We are particularly interested in procedures that generate and update surrogate constraints rapidly, and consequently we determine the weights by successive normalization.

Define $n_i = \mathbf{D}_i e$, where the vector e is understood here to be a $2n$ dimensional vector of 1´s. For the SAT problem, n_i is therefore the number of unit elements in the row vector $\mathbf{D}_i = (\mathbf{P}_i\ \mathbf{Q}_i)$ (equivalently, the number of nonzero elements in the row vector $\mathbf{A}_i$ for formulation (I)).

The successive normalization process then operates by the following two steps.

STEP 1. Make any (identifiable) compulsory or dominating assignments of values to variables, and eliminate any constraints determined to be redundant or dominated.

STEP 2. Generate the surrogate constraint for the system that remains after applying Step 1 by choosing w_i to be an increasing function of d_i and a decreasing function of n_i.

(For a system such as (MK), whose inequalities are expressed in "less than or equal to" form, the words "increasing" and "decreasing" are interchanged in the conditions of Step 2.)

The foregoing steps are applied iteratively in branch and bound methods and in constructive and destructive heuristics such as those we subsequently identify. These approaches update the system at each iteration to incorporate currently selected value assignments as well as to reflect changes from compulsory assignments and eliminated constraints.

In particular, we define a *constructive process* to be one that starts with all variables "unassigned" (implicitly 0) and successively selects some variable z_j to receive a value of 1 (hence automatically assigning the complementary variable $z_{j\#}$ an explicit value of 0). Variables currently assigned explicit values are removed from consideration and the vector d is updated appropriately (by subtracting the sum of those columns $\mathbf{D}_j$ whose associated variables z_j equal 1).

Similarly, we define a *destructive process* as one that begins with all variables "overassigned" (implicitly equal 1), and successively chooses some variable z_j to equal 0 (hence automatically assigning its complementary variable an explicit value of 1). We will not make use of a destructive process in the present study, but note its relevance to *strategic oscillation* procedures that alternate between series of constructive and destructive moves, which constitute one of the principal strategies for applying surrogate constraint analysis. Our subsequent observations can be applied directly within strategic oscillation procedures as well.

The following sections first describe how surrogate constraint analysis can be embedded in a constructive procedure for generating advanced starts, and then show how these ideas can be used to guide processes that interchange the values of complementary variables. We then assemble a complete surrogate constraint method from these components.

4. A constructive surrogate constraint procedure.

In common with other procedures described later, a constructive surrogate constraint method for the SAT problem can operate without explicitly storing the matrix **D**, since all information about its structure, both original and updated, can be maintained by sets identifying its current nonzero (unit) elements and associated variables and constraints. We represent these sets in their initial form by the following notation:

J = the index set of the variables z_j: $J = \{1, ..., 2n\}$.
$J(i)$ = the index set of variables z_j that have unit coefficients in row $\mathbf{D}_i$ of **D**: $J(i) = \{j: \mathbf{D}_{ij} = 1\}$. (Note that the cardinality of $J(i)$ is n_i.)
I = the index set of the rows $\mathbf{D}_i$ of **D**: $I = \{1, ...,m\}$.
$I(j)$ = the index set of rows $\mathbf{D}_i$ that have unit coefficients for variable z_j: $I(j) = \{i: \mathbf{D}_{ij} = 1\}$.

As a constructive method progresses, variables are assigned explicit values and constraints become redundant, causing these sets to alter their form. We refer to updated (current) solution information by the following related notation:

J^* = the index set of currently unassigned variables z_j (hence $j^* \in J^*$ implies $j\# \in J^*$).
$J^*(i)$ = the index set of unassigned variables z_j with unit coefficients in row $\mathbf{D}_i$: $J^*(i) = J(i) \cap J^*$. (Accordingly, we denote the cardinality of this set by n.)
z^* = the current solution vector, where implicitly $z = 0$ for $j \in J^*$.
d = the current value of d_i: $d = d_i - \mathbf{D}_i z^*$. (Note $\mathbf{D}_i z^*$= the number of elements j of $J(i)$ such that $z_j = 1$.)
I^* = the index set of rows $\mathbf{D}_i$ associated with currently nonredundant constraints: $I^* = \{i: d = 1\}$.
$I^*(j)$ = the index set of nonredundant constraint rows that have a unit coefficient for z_j: $I^*(j) = I(j) \cap I^*$.

Each time an assignment $z_j^* = 1$ and $z_{j\#}^* = 0$ is made, the set I^* is updated by deleting all elements of $I^*(j)$ (or equivalently all elements of $I(j)$ in I^*), since d becomes reduced from 1 to 0 for precisely the elements of this set. At the same time, for each element i of $I^*(j\#)$, the element $j\#$ is deleted from $J^*(i)$, which correspondingly reduces the value of n by 1. (Additional updates for $i \in I - I^*$ are unnecessary, since a constraint that becomes redundant remains so thereafter during the constructive process.) Detailed instructions for these updates are provided later.

4.1. Surrogate constraint generation: effective normalizations. The fundamental issue is to identify the rules for generating and updating the surrogate constraint $sz \geq s_o$. Specifically, we are left to identify precisely how the weights w_i are to be selected, and how the changes in these weights can be efficiently reflected by associated changes in the surrogate constraint coefficients. The issue is simplified for the constructive approach, since only one (current) value of d is relevant, namely $d = 1$. Hence, following the guidelines indicated earlier for determining weights by a successive normalization procedure, we only need to choose a normalization that makes w_i a decreasing function of n (for $i \in I^*$).

A customary normalization for surrogate constraints achieves this effect by setting w_i proportional to $1/n$ (where the factor of proportionality is determined by d). In the present case, we consider multiple normalization possibilities for determining w_i, as a basis for generating an advanced start for each choice (and for providing a comparison of outcomes). Since the "best" advanced start relative to reducing infeasibility may not always be the one that leads most quickly to a solution that satisfies (GC) (if the advanced start itself fails to provide such a solution), the use of multiple possibilities for determining weights also serves as a strategy for a multistart method.

Consequently, we are motivated not simply to identify a good normalization but a *good collection* of normalizations—a collection such that the advanced start generated by one or more of its members will lead to a solution satisfying (GC) in a relatively short time (where the meaning of "short" depends on the problem difficulty). Rather than arbitrarily step through a range of parameter values to test alternative normalizations, however, we begin with six *basic normalizations* and then construct new normalizations

from these by a learning approach. In this way we are able to go beyond the rules that determine w_i as a function of n (and d), and create modified weights based on the outcomes of applying prior normalizations.

We first examine these basic normalizations and specify the constructive heuristic based on them, and then indicate the learning approach that provides new normalizations.

4.2 Basic normalizations. By extension of the usual normalization approach that sets w_i proportional to $1/n$ for a given d value, we will replace n by an increasing (nondecreasing) function $\mathbf{F}(n)$, and set $w_i = M/\mathbf{F}(n)$, where M represents the "factor of proportionality" (a constant here, since we are only interested at the moment in the case $d = 1$).

The six basic forms of $\mathbf{F}(n)$ we use are as follows:

$\mathbf{F1}(n) = 1$
$\mathbf{F2}(n) = n$
$\mathbf{F3}(n) = n(n + 1)/2$
$\mathbf{F4}(n) = n^2$
$\mathbf{F5}(n) = n^3$
$\mathbf{F6}(n) = n^4$

(The value for $\mathbf{F3}(n)$ is the mean of $\mathbf{F2}(n)$ and $\mathbf{F4}(n)$.)
We segregate n = 1 as a special case due to the following observation.

REMARK 2. The condition $n = 1$ for $i \in I^*$ implies the compulsory assignment $z = 1$ and $z_{\#} = 0$ for $j \in J^*(i)$.

PROOF. The remark is immediate since n is the cardinality of $J^*(i)$, and hence the inequality for the current row i has the form $z_j \geq 1$ where j is the unique element of $J^*(i)$.

After eliminating all compulsory assignments that arise by the condition of Remark 2 (and disposing of any implicitly violated constraints that result for $n = 0$), we may restrict attention to the case $n \geq 2$.

We will monitor the condition $n = 1$ in a way that implicitly assigns a "preemptively large" weight w_i to row i when this condition occurs in the constructive approach. The reason for this is to allow compulsory assignments to be made in a strategically preferred order, in situations where such assignments may not all be mutually compatible. This is important to reduce the number of violated inequalities that result. (This might have justified using $n - 1$ as a component of the preceding functions, but will use n instead, as this will let us incorporate learning approaches in the selection between compulsory approaches, as well as letting us be able to handle single literal clauses in the improvement heuristics in a straightforward manner, using the same normalization functions.)

The preceding functions are illustrated in the following table.

n_i*	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
F1	1	1	1	1	1	1	1	1
F2	1	2	3	4	5	6	7	8
F3	1	3	6	10	15	21	28	36
F4	1	4	9	16	25	36	49	64
F5	1	8	27	64	125	218	243	512
F6	1	16	81	256	625	1308	2401	4096

The numerator M can be chosen so that all the weights w_i will yield distinct integer values, thus enabling faster integer arithmetic on the weights. This study will use floating point weights (due to easier monitoring of learning effects), and consequently a value of $M = 1$.

4.3. Surrogate constraint updating. Once the original surrogate constraint is generated, we are concerned with identifying a procedure for updating it efficiently. To be comprehensive, we will indicate complete subroutines to initiate and update all data structures for a constructive surrogate constraint approach, since these operations are intimately tied together with the operations of updating the surrogate constraint itself. On this basis we will then be able to summarize the steps of the constructive method in a succinct form.

As previously noted, we handle the special condition $n = 1$ separately in the surrogate constraint update, since it yields a compulsory assignment -- and such assignments must be performed in a judicious sequence (likewise governed by surrogate constraint information) in order to reduce constraint violations that may occur, in case these assignments are not mutually compatible. For this purpose, we maintain an array t_j in addition to the surrogate constraint coefficient array s_j, where t_j records the surrogate constraint coefficient of z_j that applies only to the compulsory inequalities $z_j \geq 1$ -- that is, we just sum these inequalities (since they all have the same n value, $n = 1$) to give the coefficient t_j. (We will the weights w for these singular clauses instead, to be able to incorporate learning effects. Recall that initially (without learning) these weights are all equal to 1).

The "complete" surrogate constraint coefficient that results in the presence of these compulsory inequalities thus may be viewed as obtained by preemptively weighting t_j and adding it to s_j. Since this is only relevant when $n = 1$ occurs -- a condition that must be checked for anyway -- the computation of maintaining t_j is trivial. Accompanying the use of t_j we maintain a list JT*, which identifies those elements of J* such that $t_j > 0$.

We assume that $n_i > 0$ holds initially for all $i \in I$, or else the problem begins with an unsatisfiable constraint. We also refer to elements of J, J* and JT* by the index h, to avoid confusion with the particular index j such that a variable z_j is selected to receive a value of 1.

4.4. Initializing and updating routines. The various routines to handle the surrogate constraint maintenance are as follows:

INITIALIZATION ROUTINE:

Set $I^* = I$, $J^* = J$, $J^*(i) = J(i)$ for all $i \in I$, $I^*(h) = I(h)$ for all $h \in J$.

Set $s_h = 0$ and $t_h = 0$ for all $h \in J$.

Initialize the index set V of violated constraints and the index set JT* to be

empty.
Then, for each $i \in I^*$:
- set $d = d_i$ (= 1) and $n = n_i$ (the cardinality of $J(i)$).
- Set $w_i = M/\mathbf{F}(n_i^*)$
- if $n > 1$, then for each $h \in J^*(i)$, set $s_h = s_h + w_i$
- if $n = 1$, for the unique $h \in J^*(i)$, set $t_h = t_h + w_i$, and add h to JT^* if $t_h = 1$

SURROGATE CONSTRAINT UPDATE ROUTINE:
(After each assignment $z_j = 1$ and $z_\# = 0$):
Delete j and $j\#$ from J^*
For each $i \in I^*(j)$:
(d changes from 1 to 0, and constraint i is eliminated)
- delete i from I^*
- delete j from $J^*(i)$
- for each $h \in J^*(i)$, set $s_h = s_h - w_i$

For each $i \in I^*(j\#)$:
(d is unchanged, but $J^*(i)$ shrinks)
- delete $j\#$ from $J^*(i)$
- set $n_i^* = n_i^* - 1$
- if $n_i^* = 0$, delete i from I^* and add i to V
- Identify the new w_i value $v = M/\mathbf{F}(n_i^*)$
- Let $\Delta = v - w_i$ and update $w_i = v$
- if $n_i^* = 1$, for the unique $h \in J^*(i)$ set $s_h = s_h - w_i$, set $t_h = t_h + w_i$, and add h to JT^* if $t_h = 1$
- if $n_i^* > 1$, then, for each $h \in J^*(i)$, set $s_h = s_h + \Delta$

ALTERNATIVE UPDATE ROUTINE: Recompute the surrogate constraint from scratch.
Delete j and $j\#$ from J^*
Set $s_h = 0$ for all $j \in J^*$
Delete each $i \in I^*(j)$ from I^*
For each $i \in I^*(j\#)$:
- delete j# from J*(i)
- set $n_i^* = n_i^* - 1$
- if $n_i^* = 0$, delete i from I^* and add i to V
- Identify the current value of $w_i = M/\mathbf{F}(n_i^*)$
- if $n_i^* = 1$, for the unique $h \in J^*(i)$, set $t_h =$ th $+ w_i$, and add h to JT* if $t_h = 1$.
- if $n_i^* > 1$ then for each $h \in J^*(i)$, set $s_h = s_h + w_i$

Note, the uses we make of the surrogate constraint in the present setting do not require explicit knowledge of s_o, and so we do not refer to it in the preceding updates. Its value can be computed simply by adding the instruction "s_o: = s_o + change" at each place where "$s_h = s_h$ + change" occurs. Also note that the test for $t_h = 1$ in the preceding surrogate constraint update routines really means if this is the first time this condition (that variable h is forced) is encountered, and the appropriate set updates must be made.

To achieve the greatest efficiency, the operation of updating the surrogate constraint will normally be somewhat faster than recomputing it from scratch, since (assuming

sparsity) the union of the sets $I^*(j)$ and $I^*(j\#)$ will typically be smaller than the set $I^* - I^*(j)$, except perhaps in final iterations of the method.

A final observation sets the stage for specifying the constructive surrogate constraint approach in detail.

REMARK 3. Whenever $s_j = 0$ for some $j \in J^*$, the assignment $z_j^* = 0$ and $z_{j\#}^* = 1$ may be made without changing the set of feasible assignments for remaining unassigned variables.

PROOF. The indicated assignment is justified by domination, noting that the use of positive weights assures that $s_j = 0$ implies $\mathbf{D}_{ij} = 0$ for all $i \in I^*$, and hence there is no loss in setting $z = 0$.

The observation of Remark 3 is applied immediately after disposing of compulsory assignments that result from the condition $n = 1$, at the beginning of each iteration of a constructive approach. (Since the updates previously specified are based on the assignment $z_j^* = 1$ and $z_{j\#}^* = 0$, the indexes j and $j\#$ of Remark 3 must be interchanged for the purpose of these updating operations.) We note the surrogate constraint choice rule must be performed to take the positive entries of t_j into account when such compulsory assignments arise (signalled by JT^* not empty). Thus we apply the surrogate constraint choice rule first to isolate elements j of JT^* that yield a maximum value of $t_j - t_{j\#}$ breaking ties in favor of larger t_j values. (The $t_{j\#}$ value will be 0, and hence $j\#$ will not belong to JT^*, unless there is a conflict that must result in violating a constraint. Such conflicts, existing and potential, are the reason for choosing the sequence of compulsory assignments with care.)

Ties may normally be expected among the best elements identified, and these will be settled by applying the usual form of the surrogate constraint choice rule (by reference to the coefficients s_j instead of t_j), thereby determining the variable z_j selected. (Such tie breaking is important, and should not be left to simple randomization except to dispose of ties at the "lowest level.") Upon setting $z_j^* = 1$ and $z_{j\#}^* = 0$, the updating operations are performed as customary before again examining JT^*.

We now state the method that results from the foregoing observations.

CONSTRUCTIVE METHOD FOR (GC).

0. Initialize I^*, J^*, etc., as previously specified. Each time an assignment is performed, update (or recompute) the surrogate constraint.
1. If I^* is empty, or becomes empty at any point during this step, go to step 3.
   - (A) If JT^* is not empty, apply the surrogate constraint choice rule to select j from JT^*, and set $z_j^* = 1$ and $z_{j\#}^* = 0$, removing j from JT^* and also removing $j\#$ if it belongs to JT^*. Repeat (A) as long as JT^* is not empty (updating the problem arrays and the surrogate constraint after each assignment).
   - (B) In the current surrogate constraint, if $s_j = 0$ for some $j \in J^*$, set $z_j^* = 0$ and $z_{j\#}^* = 1$ (breaking "ties" arbitrarily where both $s_j = 0$ and $s_{j\#} = 0$), and return to the start of Step 1 after all such assignments are made. Otherwise, if the condition of (B) does not hold, continue to step 2.

2. Apply the surrogate constraint choice rule to select an unassigned variable z_j, $j \in J^*$, and make the assignment $z_j^* = 1$ and $z_{j\#}^* = 0$. Then return to step 1.
3. (I^* is empty.) If J^* is not empty, set $z_j^* = 1$ and $z_{j\#}^* = 0$ arbitrarily for remaining pairs of unassigned variables. If V is empty, the satisfiability problem is solved; otherwise, V identifies the constraints violated by the trial solution z^*.

5. Improvement method.

To use surrogate constraint analysis to guide an improvement method, we must slightly redesign the steps to handle moves that exchange values of variables -- i.e., moves that replace an assignment $z_j^* = 1$ and $z_{j\#}^* = 0$ by the reverse assignment $z_j^* = 0$ and $z_{j\#}^* = 1$. We let $J0$ and $J1$ respectively denote the indexed sets of variables assigned the values 0 and 1, and assume the instructions given in the Appendix are used in the constructive approach to partition the sets $J(i)$, $i \in I$, into the component sets $J0(i) = \{j \in J(i): z = 0\}$ and $J1(i) = \{j \in J(i): z = 1\}$. These instructions also give rules for updating these sets in the improvement approach we now describe.

To remain consistent with the notation of the constructive approach, we continue to express the goal of each step as that of identifying a best assignment of the form $z_j^* = 1$ and $z_{j\#}^* = 0$. Thus we view the current solution z^* as represented by pairs $(j, j\#)$ for $j \in J0$ (corresponding to $z_j^* = 0$ and $z_{j\#}^* = 1$).

We are concerned primarily with three main issues: (1) generating and updating the surrogate constraint, (2) selecting a “best move” by reference to the surrogate constraint, (3) using learning processes to create improved surrogate constraints over time, and to refine and diversify the move selection process.

5.1 Surrogate constraint generation and updating. Since the sets J, I, $J(i)$ and $I(j)$ are not progressively reduced in the improvement procedure, as they are in the constructive procedure, we maintain them in their original form. Similarly, we do not replace n_i by n, but keep n_i at its original value.

We add a new consideration in the improvement approach by seeking to account for the influence of a larger number of component constraints in generating the surrogate constraint. Specifically, we still retain reference to the updated d value ($= d_i - \mathbf{D}_i z^*$), where $d = 1$ is the signal that the inequality $\mathbf{D}_i z \geq d_i$ is violated by the current solution $z = z^*$. However, instead of restricting attention only to the inequalities with $d = 1$ as a basis for generating the surrogate constraint, we include the option of incorporating “nearly violated” inequalities that yield $d = 0$ and $d = -1$.

The inequalities for $d = 0$ and $d = -1$ should receive significantly smaller weights than those for $d = 1$ (particularly for equal values of n_i), and we handle this by subdividing the surrogate constraint into corresponding components. In particular, we continue to let $sz \geq s_o$ denote the surrogate constraint (component) generated from inequalities for which $d = 1$, and additionally let $s'z \geq s_o'$ and $s''z \geq s_o''$ denote the surrogate constraint components generated for $d = 0$ and $d = -1$, respectively. Likewise, we let w' and w'' denote the weight vectors for these components. (In this way, we avoid explicitly using weights that assure $w >> w' >> w'$, but simply use the coefficients of $s'z \geq s_o'$ and $s''z \geq s_o''$, in turn, to resolve ties created by the choice rules applied to $sz \geq s_o$.)

5.2. Surrogate constraint coefficients for exchange moves. Exchange moves can be evaluated efficiently by slightly modifying the way surrogate constraint coefficients are generated. To evaluate an exchange that replaces the assignment $z_j^* = 0$ and $z_{j\#}^* = 1$ by the reverse assignment $z_j^* = 1$ and $z_{j\#}^* = 0$, it is necessary first to "undo" the half-assignment $z_{j\#}^* = 1$, thereby restoring the implicit assignment $z_j^* = 0$ and $z_{j\#}^* = 0$ which forms the basis for the evaluation rules of the preceding sections. Fortunately, it is unnecessary to introduce an extra step to accomplish this restoration. Instead, we maintain a mixed surrogate constraint, which does not alter the coefficient s_j (corresponding to $z_j^* = 0$), but modifies the coefficient $s_{j\#}$ (corresponding to $z_{j\#}^* = 1$) to take the value it would receive if instead $z_{j\#}^* = 0$. This is done for each such $s_{j\#}$ coefficient independently, since our analysis requires the assumption that only a single $z_{j\#}^*$ value changes from 1 to 0, while the other values remain unchanged.

To accomplish this, we observe that changing $z_{j\#}^*$ from 1 to 0 causes the vector d^* to be replaced by the vector $d^* + \mathbf{D}_{j\#}$; that is, d is replaced by $d + 1$ for $i \in I(j\#)$. Thus, the "adjusted" surrogate constraint coefficient for $s_{j\#}$ is generated simply by using the weights w_i that apply under the assumption that d is 1 larger than its current value for $i \in I(j\#)$. Again letting the index h represent a general element of J (differentiated from the index $j \in J0$ of the current assignment $z_j^* = 0$ and $z_{j\#}^* = 1$), the rules to initialize and update the surrogate constraint components are as follows. (The weight w_i in each case is assumed as before to be given by $w_i = M/\mathbf{F}(n_i)$, where w' and w'' are given by replacing M and $\mathbf{F}$ by M' and $\mathbf{F}'$, and by M'' and $\mathbf{F}''$, respectively.)

INITIALIZATION ROUTINE:

Begin with s_h, s_h' and $s_h'' = 0$ for $h \in J$.

For $i \in I$:

If $d = 1$:	For $h \in J0$(i): set $s_h = s_h + w_i$
If $d = 0$:	For $h \in J1$(i): set $s_h = s_h + w_i$
	For $h \in J0$(i): set $s_h' = s_h' + w_i'$
If $d = -1$:	For $h \in J1$(i): set $s_h' = s_h' + w_i'$
	For $h \in J0$(i): set $s_h'' = s_h'' + w_i''$
If $d = -2$:	For $h \in J1$(i): set $s_h'' = s_h'' + w_i''$

The update of the surrogate constraints after each new assignment is made can be done incrementally, similarly to the update in the constructive case. However, with the addition of learning effects (see Section 6), these calculations can become quite messy, and we use the initialization routine above for updating the weights as well.

We observe several things about these operations. First, if attention is restricted only to the surrogate constraint component $sz \geq s_o$ the amount of updating is significantly reduced. The updating can also be considerably reduced by dropping reference to $s''z \geq s_o''$ and by choosing M' "relatively small" so that the maximum w_i' value is not large. Then, making M moderately larger than its usual value can assure that the w_i values dominate the w_i' values, allowing the component constraint $s'z \geq s_o'$ to be embedded directly within $sz \geq s_o$. A goal of our computational testing is to explore the tradeoffs in "speed versus information" by incorporating surrogate constraint components at different levels.

5.3 Significance of the mixed surrogate constraint evaluation. The mixed surrogate constraint evaluation that is incorporated into the preceding updates allows an

exchange move to have the same basis of evaluation as a constructive move. Although the analysis is straightforward in the context of the (GC) formulation, it is less apparent in the context of the (I) formulation. Relative to the inequality system $\mathbf{A}x \geq b$ of (I), the evaluation implies that the negative coefficients of **A** are treated differently than the positive coefficients of **A** in generating appropriate surrogate constraint weights. This seems at first counter to the definition of a surrogate constraint, but in fact it results from the fact that an exchange move implicitly requires reference to multiple surrogate constraints -- the surrogate constraint that applies to the current solution, and an additional surrogate constraint for each $z_{j\#}$ such that $z_{j\#}^* = 1$, where the constraint results by setting $z_{j\#}^* = 0$. The useful consequence of our analysis is that we may generate appropriate coefficients for all cases with essentially the same effort as required to generate coefficients for a single surrogate constraint. While entirely natural relative to the (GC) formulation, the analysis becomes somewhat convoluted relative to the (I) formulation. This shows how alternative equivalent formulations can sometimes be useful for developing useful insights.

5.4. Explicit form of the choice rules. We use exactly the same surrogate constraint choice rule as in the constructive approach for evaluating and selecting a preferred move, based on identifying a variable z_j that maximizes the value $(1+p)s_j - s_{j\#}$ where p receives a small positive value as a "first level tie breaker." However, when w' and w'' are used (or just w' by itself) to extend the surrogate constraint, we incorporate the variant that displaces the tie breaking use of p to a lower decision level. In this variant, we first seek a maximum value of $s_j - s_{j\#}$ then resolve ties from a maximum value of $s_j' - s_{j\#}'$, and finally (when w'' is included) from a maximum value of $s_j'' - s_{j\#}''$. Only in the rare event that ties are not resolved in this manner do we include reference to the maximum s_j value (implicitly incorporating a very small value of p). When s_j' is embedded in s_j, of course, as in the previous modified update routine, the original choice rule applies without change. (Following standard procedure, ties that are not broken otherwise are broken randomly.)

In each of these cases, the index j is restricted to $j \in J0$, since we only allow choices that change a current assignment. It is possible that a current best choice has a zero or negative evaluation, but this is not cause for rejecting it.

6. Learning and multiple starts.

As previously noted, the advanced starting solutions generated by the constructive procedure can be used as the basis for a multistart method. One way to generate a variety of solutions, in addition to those that result from the six basic normalizations, is to include additional normalizations (as by taking various convex combinations of the functions **F1** through **F6**), and by selecting different values for the parameter p of the surrogate constraint choice rule (instead of choosing p small).

The alternative we propose instead is to use the knowledge of the index set V of violated constraints as a basis for learning appropriate changes in the weights w_i -- i.e., changes designed to generate new solutions that differ from previous ones, and that are more likely to satisfy previously violated constraints. For this purpose if we simply assigned dominant weights w_i for i in V, then a new constructive solution pass would devote its first steps to satisfying the constraints violated on the previous pass, assuring the two successive solutions would be different. We seek to modify the weights w_i for i in V in a more judicious fashion, in order to maintain the influence of the normalization

functions. In addition, we modify the weights in a way that assures the effect of previous modifications is not lost, so that the learning component is not myopically determined by the solution obtained immediately before the current one. Similar considerations can be made for learning in the improvement procedure.

6.1. Learning in the constructive procedure. We propose a design based on introducing a *modification factor* $f(i)$ for each constraint $i \in I$. Then, instead of setting $w_i = M/\mathbf{F}(n_i^*)$, we set $w_i = f(i)M/\mathbf{F}(n_i^*)$. This embodies an implicit combination of frequency based and recency based memory, as customarily used in tabu search. This straightforward design entails no added computational effort for updating the surrogate constraint after the assignment $z_j^* = 1$ and $z_{j\#}^* = 0$, other than to introduce (at most) two multiplications for each $i \in I^*$. In addition, a variety of strategies can conveniently exert their influence by this design, using a simple rule to progressively modify $f(i)$ itself, which we describe as follows.

To begin, we set $f(i) = 1$ for all $i \in I$, so that w_i receives its usual value on the first constructive solution pass. Then we update $f(i)$ on successive passes by introducing a multiplier $v(i)$ and making an assignment: $f(i) = v(i)f(i)$. A natural implementation of this model is to select $v(i) = 1$ if constraint i is not currently violated, and $v(i) > 1$ otherwise. By this means, a constraint that is violated on a given pass will receive an "enduring bias" to be satisfied on subsequent passes, so that the learning process does not completely disregard the outcomes of previous construction attempts. Moreover, a constraint that is repeatedly violated gradually receives more and more emphasis until it effectively achieves a top priority for being satisfied. We use a rule for applying this approach based on choosing two parameters, a value v_o which gives the starting value assigned to $v(i)$ when constraint i is violated, and a value Δv which successively modifies the value assigned to $v(i)$. The explicit form of the rule, including the associated determination of weights, is as follows.

MULTIPLIER UPDATE LEARNING RULE (CONSTRUCTIVE PROCEDURE):

0. Initially set $f(i) = 1$ for $i \in I$ and set $v = v_o$.
1. For the current pass, define $w_i = f(i)M/\mathbf{F}(n_i^*)$ for $i \in I^*$.
2. At the end of each pass, set $v(i) = 1$ for $i \in I - V$, $v(i) = v$ for $i \in V$, and $f(i) = v(i)\, f(i)$ for all $i \in I$.
   Then, set $v = v + \Delta v$, and return to 1 for the next pass.

The special instance of this rule that sets $\Delta v = 0$, and hence always assigns $v(i)$ the value v_o for i in V evidently embodies a form of frequency based memory that gives a uniform bias to satisfying constraints that have been violated the same number of times. For example, for the choice $v_o = 2$, the initial w_i value is changed to become twice as large for a constraint that has been violated on one pass, four times as large for a constraint violated on two passes, and so forth. (The effect is subtler than this, because the value of w_i also changes as n changes. The multiplier effect holds constant across a given value of n.) The choice $v_o = 2$ thus provides a relatively strong impetus to satisfy a constraint previously violated, and especially a constraint that is violated more than once.

The variant that maintains $\Delta v = 0$ has a potential shortcoming, however. The use of a uniform bias for frequency can allow a scenario in which, for example, the first third of the constraints are violated on the first pass, then (precisely) the next third of the

constraints are violated on the second pass, and finally the last third of the constraints are violated on the third pass. At the end of the third pass the factor $f(i)$ which has been updated by $f(i) = v(i)\ f(i)$ after each pass will now have the same value for all constraints $i \in I$, and so the approach will cause the fourth pass to give the same solution as on the first -- continuing to cycle every three passes. Such a scenario is not highly probable (since it would be rare to obtain a starting solution that allowed fully a third of the constraints to be violated, and other interactions would likely rule out such an elementary type of cycling), but the uniform frequency bias nevertheless can create periodicities that may undermine progress toward a goal of satisfying increased numbers of constraints (which at least loosely is one of the goals we consider desirable to achieve).

Consequently, the incrementing parameter Δv is included to overcome this potential pitfall, which it achieves by introducing a recency based element into the learning process to complement the frequency based element. As time goes on, a positive Δv value progressively increases the value v to be assigned to the multiplier $v(i)$ when constraint i is violated. (Equivalently, we could let v stay unchanged, and let old $f(i)$ values diminish, but this would require considerably more computation for updating.) As a result, a constraint violated on pass $k + 1$ receives a higher bias for being satisfied on the next pass than one violated on pass k. Ideally, such a rule might include a short term memory design that "forgets" (or greatly reduces) earlier biases after a sufficient span of time, to introduce a thresholding effect. The integration of frequency based and recency based memory over short term and longer term horizons represents a first order approximation to more advanced strategies in tabu search, which is convenient for study because it relies solely on the parameter v.

Just as there is a potential danger if frequency is allowed to be uniform, inversely there is a potential danger if recency is allowed to become too strongly skewed. In particular, if the increment Δv is made too large, then we encounter a situation where the outcome of the immediately preceding pass dominates all prior outcomes, which can cause an erratically zigzagging change in successive solutions that invites short term cycling. On the other hand, if the increment is selected too small, the outcome will scarcely be differentiable from that of uniform frequency.

6.2. Learning in the improvement procedure. As with the constructive method, we are concerned with modifying surrogate constraint weights based on learning. Quite simply, by direct extension of our earlier approach, we seek to increase the weights for constraints that are more often violated, taking frequency and recency considerations into account. Within the setting of an improvement procedure, we speculate that frequency becomes somewhat more important than recency and consequently we will organize the learning process around the use of frequency measures.

Let *SUM* be the number of violated constraints on a given iteration. We can then use the following rule to update the weights for the currently violated constraints.

WEIGHT UPDATE LEARNING RULE (IMPROVEMENT PROCEDURE):
Let $\Delta w = q/SUM$
For each $i \in V$.
Set $w_i = w_i + \Delta w$

Instead of using SUM in the denominator for forming Δw above, one might use SUM^2 (dampening the effect) or 1 (amplifying the effect). The value of q determines the overall effect assigned to the learning.

The intuition that constraints that are violated more often should receive larger weights is reinforced by considering the effect of constraint dominance. Note that in the (GC) formulation for the SAT problem, where $d_i = 1$ for all i, we can say that constraint i dominates constraint k if $\mathbf{D}_i \leq \mathbf{D}_k$. In such a case, constraint k can be effectively treated as redundant and eliminated from consideration.

As in the corresponding conditions for a variable to dominate another, the effort of checking for constraint dominance may be more than is worthwhile to invest. In addition, there can be conditions of "almost dominance" that may be desirable to account for, as an approximate indicator of the weights that should be assigned to different constraints. The conditions require even greater work to identify.

The surrogate constraint normalization that assigns w_i a larger weight for a smaller n_i value already implicitly will give greater emphasis to dominating constraints than to dominated constraints. Adding in the Δw values for the violated constraints, which gives a measure of the frequency of violation, allows us to magnify this emphasis in the proper direction.

REMARK 4. If constraint $\mathbf{D}_i z \geq d_i$ dominates constraint $\mathbf{D}_k z \geq d_k$, then $q_i \geq q_k$ holds over every subset of iterations that q_i and q_k are both updated.

PROOF. The result is a simple consequence of the fact that a violation of $\mathbf{D}_k z \geq d_k$ will assure a violation of $\mathbf{D}_i z \geq d_i$, while the reverse is not true.

Remark 4 discloses how frequency measures are useful for reflecting the influence of dominance and "almost-dominance". We observe that the remark also holds in the more general case where q_i and q_k are not simply incremented by 1 when their associated constraints are violated, but where they may be incremented by a variable amount (as long as the increment is the same for each term on any given iteration). This allows a recency element to be incorporated, by which the increment gradually increases over time. By this means, if the search leaves a region where particular constraints have tended to be violated, the relative emphasis on these constraints can be diminished more significantly than by relying on a uniform increment at each iteration. (A "time threshold," as often used in recency based memory in tabu search, allows a further discounting of the past.)

7. Probabilistic tabu search

As sometimes noted in the tabu search literature, "controlled randomization" (which uses the biases of probability) may be viewed as a substitute for memory -- when we are ignorant of how memory should advantageously be used. But just as there are multiple kinds of memory that can supplement each other, probabilities may find their best uses in various supplementary roles. The ideas that are tested in this paper originate in part from Glover (1989), and have been extensively tested in Løkketangen and Glover (1995).

For clarification, since more than one interpretation is possible, what is generally regarded as probabilistic tabu search is usually applied to the move acceptance function. Among conjectures for why these probabilistic measures may work better than the

deterministic analog, one may guess that move evaluations have a certain "noise level" that causes them to be imperfect -- so that a "best evaluation" may not correspond to a "best move". Yet the imperfection is not complete, or else there would be no need to consider evaluations at all (except perhaps from a thoroughly local standpoint -- noting the use of memory takes the evaluations beyond such a local context). The issue then is to find a way to assign probabilities that somehow compensates for the noise level.

One may also view controlled randomization as a means for obtaining diversity without reliance on memory. In this respect it represents a gain in efficiency by avoiding the overhead otherwise incurred in the use of long term memory. However, it also implies a loss of efficiency as potentially unproductive wanderings and duplications may occur, that a more systematic approach would seek to eliminate.

A method like GSAT also relies on the use of randomization to obtain this diversification effect (See Selman et. al. 1992). The use here is more implicit, and relies on the plateaus in the move evaluations, when using the change in the number of satisfied clauses as move evaluations.

Simulated Annealing (SA) also uses probabilistic measures for move acceptance, but in a significantly different way than they are used by TS. SA samples the neighborhood (typically randomly) by a design that accepts any improving move encountered, and that accepts nonimproving moves with a decreasing probability as the search progresses. By contrast, PTS creates a strategically composed candidate list of moves (sorted by some measure of goodness) and uses a biased probability to select from this list, with the first move on the list (i.e. the move that is considered best according to some criterion) having the greatest chance for being selected. This selection is done after the usual tabu restrictions and aspiration criteria have been applied. PTS is thus much more aggressive, and gives more guidance to the search process. For a general description of SA, see Dowsland (1993), while Connolly (1994) describes the use of SA as a general search tool component for pure ILP problems.

Hart and Shogan (1987) describes the use of probabilistic measures for move selection in greedy heuristics. Here the move is selected with uniform probability either among the *n* best moves, or among the moves better than some threshold (compared to the best move), but only considering improving moves.

7.1. Probabilistic tabu search for surrogate constraints. To be able to apply the general probabilistic move acceptance approach as outlined in the previous section, we seek a move evaluation function that in some way reflects the *true merit* of all the moves in the candidate list. Usually this is based on mapping the move evaluations into positive weights, and using these weights to obtain probabilities by dividing by the sum of the weights, where the highest evaluations receive weights that disproportionately favor their selection.

The use of surrogate constraints to guide the search gives a larger range of move evaluations than customarily used for SAT problems, and consequently fewer moves tie for first place in the sorted candidate list. The inclusion of learning effects amplifies this trend. As the only diversification present in our surrogate constraint procedures relies on "equal best moves", the inclusion of more sophisticated move evaluations reduces this diversifying effect. The notion of PTS is therefore added to be able to increase the diversification of the search, but by measures making it more controllable than just relying on plateaus in the move evaluation function.

Our surrogate constraint move evaluation function for the improvement procedure contains three levels of surrogate constraint coefficients for each variable (see Section

5.2). It is therefore difficult to assign a good comparable numerical measure which reflects the *true merit* of all the moves. However, based on the dominance arguments put forth in previous sections, there is good reason to believe that the *relative* ranking of the moves in the candidate list is quite good, and that a move should be selected among the first few in the candidate list, if possible.

We therefore propose to introduce controlled randomization in the move selection process by exponentially decreasing the move acceptance probability when traversing the candidate list, thus relying solely on the individual move rankings, and not using the actual move evaluations.

7.2. Exponentially decreasing move acceptance. The basis for this approach is that we regard the relative ranking of the moves on the candidate list to be a good approximation to their *true merit*.

The general outline of the method is as follows. Let p be the probability threshold for acceptance of a move, and let r be a randomly generated number (both in the range 0 –1). Only the basic move selection core of the method is shown.

EXPONENTIALLY DECREASING MOVE ACCEPTANCE:

0. Generate the candidate list in the usual way.
1. Take the first (potentially best) move from the candidate list.
2. Generate r.
   If $r > p$, reject. Goto Step 3.
   If $r \leq p$, accept move, exit.
3. Select the next move on the candidate list. Goto Step 2.

If all acceptable moves are rejected, select the move randomly among all candidates.

The method is intriguing because of its simplicity. The value of p should not be too small, as we would usually like to usually select one of the top few moves. Testing will show good values for p, but as an example, consider $p = 1/3$. The probability of selecting each of the first d moves is then:

$$1/3,\ 2/9,\ 4/27,\ 8/81,\ ...,\ 2^{d-1}/3^d\ .$$

The probability of not choosing one of the first d moves is $2^d/3^d$, so $p = 1/3$ gives a very high probability of picking one of the top moves: about .87 for picking one of the top 5, and about .98 for picking one of the top 10.

The effective value of p can also be viewed as a function of the quality of the move evaluation function. The better the move evaluation function, the higher the expected value of p for which the best solutions are obtained.

8. Relation to other approaches.

If, for the surrogate constraint improvement procedure (see Section 5) we restrict attention only to the surrogate constraint component $sz \geq s_o$ based on w, without reference to components based on w' and w'', an interesting connection to some previous approaches can be developed. To do this, we must additionally restrict attention to the normalization based on the "uniform weighting function" **F1** (which gives $\mathbf{F1}(n_i) = 1$ for all i, see Section 4.2). Then, as previously observed, we can simply

choose the scaling factor $M = 1$, and it is easy to show that the surrogate constraint coefficients s_j and $s_{j\#}$ respectively count the number of violated constraints that would be satisfied by setting $z_j^* = 1$, and the number of satisfied constraints that would be violated by setting $z_{j\#}^* = 0$. Thus the coefficient $s_j - s_{j\#}$ of the derived surrogate constraint corresponds to the "net improvement" (or disimprovement) in the number of satisfied constraints that results by reversing the current assignment $z_j^* = 0$ and $z_{j\#}^* = 1$.

This simple counting function corresponds to the choice rule used in the GSAT procedure of Selman, Levesque and Mitchell (1992), and is also embodied in the more recent random walk enhancement of GSAT by Selman, Kantz and Cohen (1995). In this respect, some of the choice rules that are currently popular for solving SAT problems share common ground with rules that result from a special instance of surrogate constraint analysis. On the other hand, surrogate constraint analysis provides a much wider range of options (resulting from different weights and choice rules over different ranges of constraints), and bases these options on a framework for capturing useful information from the problem constraints. In the mathematical optimization setting, this framework provides a duality theory that further motivates its application. (See, e.g., Glover (1965, 1975), Greenberg and Pierskalla (1970, 1973), Freville and Plateau (1993).)

9. Computational testing.

To test our heuristic, we used the same test-cases from the *aim-** and *jnh** group as used by Resende and Feo (1994), both groups being in the test case portfolio of the *2nd DIMACS Challenge on Satisfiability Problems*. The instance class *aim-**, submitted by E. Miyano, are artificially generated 3-SAT problems, with the property that each instance has at most one satisfiable truth assignment. The problems in *jnh**, submitted by John Hooker, are random instances, generated to be difficult by rejecting unit clauses and setting the density to a hard value.

In addition we tested our procedures on some structured real world problems from the areas of asynchronous circuit synthesis and technology mapping (mapping the gate level circuit into an implementation using standard cell library) submitted by Jun Gu to the *DIMACS Workshop on the Satisfiability Problem: Theory and Applications*, with class names *as** and *tm**

Our test code is implemented in C++, running on a 75 Mhz Pentium PC under Windows. The code is highly experimental, designed to more easily focus on the behavior of the search process, as suggested by Hooker (1996). We have not sought to streamline our code for speed. As a comparison, one iteration of the constructive phase takes about 10 times the amount of time used by an iteration requiring a similar level of computation in GRASP, as reported in Resende and Feo (1994).

9.1. Tests of constructive methods. A key question is which normalization strategy is generally most effective at reducing V (the set of violated inequalities)? It should be noted that the normalization based on **F1**, which yields $w_i = 1$ for all i with $d = 1$, leads to a rule of reducing V by a maximum amount at each step if a maximum s_j value is the basis of choice. Similarly, it leads to a rule of reducing V by a maximum "opportunity cost" amount at each step if a maximum $s_j - s_j^*$ value is the basis of choice. Thus, **F1** yields rules that are directly designed for the objective of reducing V. (This simple counting normalization also is the basis of the constructive heuristic of Chvtal (1979) for covering problems. As previously noted, in the setting of improvement methods, **F1** also

yields rules that correspond to those proposed more recently in currently popular SAT methods.)

However, in spite of its direct link with the objective of reducing *V*, this function may not be the one that actually succeeds in reducing *V* the most. Surrogate constraint analysis in fact suggests that the other **F**'s (**F2** to **F6**) are likely to yield somewhat better outcomes, though this would seem counterintuitive from a traditional orientation (since it suggests that using a "wrong" objective function is better than using the "right" one). Consequently, it is of interest to determine the effect on *V* created by the choice of normalizations.

Table I shows the size of *V* for the different **F**'s for the *aim* problem class. A modest 100 iterations were run for each case. Recall that the only diversification employed is the random selection between the equal top contenders of the candidate list. Entries marked with an asterix indicate that the optimum was found within 100 iterations in an ensuing improvement phase. Thus, in this case we are simply using the constructive part

Table I. *V* versus **F** for *aim**. Constructive procedure. No learning.

Name	F1	F2	F3	F4	F5	F6
aim-50-1_6_yes1-1	1	1	1	1	1	1
aim-50-1_6_yes1-2	1	1	0	1	1	1
aim-50-1_6_yes1-3	1	1	1	0	0	0
aim-50-1_6_yes1-4	1	1	1	1	1	1
aim-50-2_0_yes1-1	2	1	1	1	1	1
aim-50-2_0_yes1-2	2	1	1	1	1	1
aim-50-2_0_yes1-3	2	1	1	1	1	1
aim-50-2_0_yes1-4	3	1	0	0	0	0
aim-50-3_4_yes1-1	9	3	2	1	2	2
aim-50-3_4_yes1-2	10	2	1*	1*	1*	1*
aim-50-3_4_yes1-3	11	2	3*	3	3	3
aim-50-3_4_yes1-4	12	3	3	3	3	3
aim-50-6_0_yes1-1	20*	2*	0	0	0	0
aim-50-6_0_yes1-2	24*	0	0	0	0	0
aim-50-6_0_yes1-3	23*	0	0	0	0	0
aim-50-6_0_yes1-4	22*	0	0	0	0	0
aim-100-1_6_yes1-1	2	1	1	2	2	2
aim-100-1_6_yes1-2	3	1	1	1	1	1
aim-100-1_6_yes1-3	1	1	1	1	1	1
aim-100-1_6_yes1-4	1	1	1	1	1	1
aim-100-2_0_yes1-1	6	1	1	1	1	1
aim-100-2_0_yes1-2	6	2	1	1	1	1
aim-100-2_0_yes1-3	7	2	1	2	2	2
aim-100-2_0_yes1-4	6	1	1	1	1	1
aim-100-3_4_yes1-1	23	8	4	5	6	6
aim-100-3_4_yes1-2	20	8	5	6	5	5
aim-100-3_4_yes1-3	23	8	4	5	6	6
aim-100-3_4_yes1-4	26	8	7	7	6	6
aim-100-6_0_yes1-1	47	25	0	5*	0	0
aim-100-6_0_yes1-2	48	13*	0	0	4*	0
aim-100-6_0_yes1-3	47	11*	0	0	0	0
aim-100-6_0_yes1-4	48	6*	0	0	0	0
aim-200-6_0_yes1-1	98	46*	41*	1*	1*	1*
aim-200-6_0_yes1-2	103	18*	23*	23*	15*	11*
aim-200-6_0_yes1-3	105	32*	15*	17*	15*	9*
aim-200-6_0_yes1-4	105	33*	29*	2*	14*	32

Table II. *V* versus **F** for *as** and *tm**. Constructive procedure. No learning.

Name	F1	F2	F3	F4	F5	F6
as2-yes	8*	5*	3*	2*	1*	0
as3-yes	8*	4*	3*	2*	1*	0
as4-yes	55	33	17	16	17	9
as6-yes	31	15	13	11*	5*	0
as8-yes	10*	6	5	6*	4	4*
as10-yes	44	22	13*	8*	11	1*
as11-yes	22	10*	8*	8*	6*	1*
as12-yes	16	11*	7*	7*	7*	7*
as13-yes	33	14	6*	6	5*	3*
as14-yes	5*	4*	4*	2*	2*	2*
as15yes	89	30	16	13	13	13*
tm2-yes	1318	266*	118*	74*	19*	-

of our approach to initiate a multistart method, as in surrogate constraint strategies of Glover (1977). For these tests we do not exploit PTS notions beyond the starting solutions. (We note that a similar multistart design has been adopted by GRASP, though relying chiefly on randomization effects rather than on the use of surrogate constraint guidance.) As is clearly evident from the table, using the F3 or F4 normalization often succeeds in significally reducing the cardinality of the best *V* found. (Similar improvements were also found for worst and average solutions.)

The issues connected with reducing infeasibility during the constructive phase acquire relevance primarily under the assumption that higher quality starting solutions, which we are implicitly defining as solutions with smaller sets *V* of violated inequalities, tend to enable a problem to be solved more effectively in a subsequent improvement phase. As can be seen from the tables, the ensuing local search phase finds the optimum easier when the cardinality of the best *V* found is reduced.

Table II similarly gives the results for the *as** and *tm** test cases (the dash in the table entry for **F6** and *tm2-yes*, is because a floating point error occurred for this configuration. This is not too surprising, as *tm2-yes* has clauses with a lot of literals).

Table III. *V* versus **F** for *jnh**. Constructive procedure. No learning.

Name	F1	F2	F3	F4	F5	F6
jnh1	33	4	3	2	2	2
jnh2	43	8	3	1*	0	0
jnh12	39	6	3	3	1	0
jnh17	44	7	4	2	1*	1
jnh201	32	4	1*	1*	0	0
jnh204	40	5	2	1	4	1
jnh205	39	4	3	2	1	1
jnh207	34	5	3	1	1	1
jnh209	39	7	4	2	2	2
jnh210	38	6	0	1	2	3*
jnh212	33	6	3	2	2	1
jnh213	36	9	1	4	0	1
jnh217	37	7	3	1*	1	0
jnh218	40	5	2*	1	0	0
jnh220	35	5	3	1*	3	1
jnh301	37	9	4	3	2	2

Table IV. $v(i)$ vs. **F** for *as2-yes*.

v(i)	F1	F2	F3	F4	F5	F6
1	8	5	3	2	1	0
1.1	8	3	3	0	0	0
1.2	9	2	0	0	0	0
1.5	8	0	0	0	0	0
2	8	0	0	0	0	0

Table V. Results for *as2-yes* with learning

as2-yes	F1	F2	F3	F4	F5	F6
Best found	8	0	0	0	0	0
Opt at	-	35	22	33	5	0
Avg. values	31.9	6.4	7.5	5.7	3	0

Table III gives the results for the *jnh** test cases. As can be seen from the tables, using the normalization F1 generally fails to produce good results. Progressively better results for **F2**, **F3** and **F4**. The inclusion of **F5** and **F6** produces similar, and for some test case classes better, results than using **F4**.

Table VI. V versus **F** for *aim**. Constructive procedure. $v(i) = 1.5$.

Name	F1	F2	F3	F4	F5	F6
aim-50-1_6_yes1-1	1	1	1	1	1	0
aim-50-1_6_yes1-2	1	1	1	0	0	1
aim-50-1_6_yes1-3	1	0	0	0	0	0
aim-50-1_6_yes1-4	1	1	0	0	0	0
aim-50-2_0_yes1-1	2	1	1	1	1	1
aim-50-2_0_yes1-2	3	1	1	1	1	1
aim-50-2_0_yes1-3	2	1	1	1	1	1
aim-50-2_0_yes1-4	3	0	0	0	0	0
aim-50-3_4_yes1-1	9	4	2	2	2	0
aim-50-3_4_yes1-2	11	3	0	0	1	1
aim-50-3_4_yes1-3	12	3	1	0	0	0
aim-50-3_4_yes1-4	11	4	0	0	0	0
aim-50-6_0_yes1-1	21	1*	0	0	0	0
aim-50-6_0_yes1-2	21	0	0	0	0	0
aim-50-6_0_yes1-3	21	4*	0	0	0	0
aim-50-6_0_yes1-4	21	3*	0	0	0	0
aim-100-1_6_yes1-1	1	1	1	1	1	1
aim-100-1_6_yes1-2	1	1	1	1	1	1
aim-100-1_6_yes1-3	1	1	1	1	1	1
aim-100-1_6_yes1-4	1	1	1	0	1	1
aim-100-2_0_yes1-1	5	1	1	1	1	1
aim-100-2_0_yes1-2	6	2	1	2	2	1
aim-100-2_0_yes1-3	7	2	1	1	2	1
aim-100-2_0_yes1-4	6	1	1	1	1	1
aim-100-3_4_yes1-1	23	11	8	6	5	3
aim-100-3_4_yes1-2	23	9	7	6	3	5
aim-100-3_4_yes1-3	24	5	7	4	4	1
aim-100-3_4_yes1-4	23	10	6	6	3	5
aim-100-6_0_yes1-1	52	19*	6*	0	0	0
aim-100-6_0_yes1-2	53	18*	0	0	0	0
aim-100-6_0_yes1-3	52	18*	11*	0	0	3*
aim-100-6_0_yes1-4	49	18*	0	1*	1*	0
aim-200-6_0_yes1-1	103	58	8*	4*	0	0
aim-200-6_0_yes1-2	103	44*	12*	45	38*	47
aim-200-6_0_yes1-3	105	60	36*	21	24*	0
aim-200-6_0_yes1-4	103	55	39	2*	29*	17

Table VII. *V* versus **F** for *as** and *tm**. Constructive procedure. $v(i) = 1.5$.

Name	F1	F2	F3	F4	F5	F6
as2-yes	8	0	0	0	0	0
as3-yes	6*	1*	0	0	0	0
as4-yes	96	17	19	2	0	9
as6-yes	33*	8	0	4*	0	0
as8-yes	12*	2	3*	2	2*	2*
as10-yes	35	1*	0	1*	0	0
as11-yes	12*	3*	0	0	0	0
as12-yes	16*	5*	6*	6*	5*	5*
as13-yes	37	5	4*	3*	3*	3*
as14-yes	7*	1*	1*	1*	2*	2*
as15yes	100	21*	18	15	9	13*
tm2-yes	1318	257*	30*	8*	2*	-

9.2 Learning and PTS in the constructive procedure. We investigate whether the learning strategies cause the constructive part of our method to yield smaller sizes for *V*, the set of unsatisfied clauses, and whether these strategies also provide advantages that carry over to the subsequent improvement phases.

We used *as2-yes* as a test-case to identify good values for the clause weight multiplier $v(i)$. Table IV shows the effects of different values of $v(i)$ for the different weights. As can be seen, the important aspect here is to include *some* amount of learning. Another interesting question is how fast the methods converges to an optimum. In Table V this is illustrated for the same *as2-yes* test case. The rows show the best value of *V* found, the iteration number where this V was found, and the average value of *V* for the iterations up to (and including) the point where the best *V* was found. A value of $v(i) = 1.5$ was used.

This example illustrates very nicely that by using the more extreme **F**'s, the search yields *better* average values for each iteration, and converges *faster* to the optimum. The inclusion of the learning mechanisms clearly has a beneficial effect.

For the general testing of learning we used the same value of $v(i) = 1.5$. The test

Table VIII. *V* versus **F** for *jnh**. Constructive procedure. $v(i) = 1.5$.

Name	F1	F2	F3	F4	F5	F6
jnh1	27	6	3	2	1	2
jnh2	38	9	4	2	0	0
jnh12	43	6	4*	4	0	1
jnh17	43	9	2	2*	1*	1
jnh201	27*	7	1	0	0	0
jnh204	37	7	3*	2	1	1
jnh205	42	5	3	2	1	1
jnh207	38	7	5	1	1	1
jnh209	41	11	4	2	1*	0
jnh210	35	6	3*	1	0	1*
jnh212	36	5	2	2	2	1
jnh213	40	8	4	1	0	1*
jnh217	30	8	4	1	0	1*
jnh218	36	7	3*	1	0	1
jnh220	35	8	4	2	2	2
jnh301	37	9	4	2	1	2

Table IX. PTS on *jnh1*. No learning.

p	Best	Wor	Avg.
0.1	7	24	15.0
0.2	4	23	12.1
0.3	3	18	9.8
0.4	2	19	9.3
0.5	2	17	8.3
0.6	3	15	8.3
0.7	3	16	7.6
0.8	2	14	7.4
0.9	2	13	6.85
1.0	2	14	7.2

Table X. PTS on *jnh1*. $v(i) = 1.5$.

p	Best	Wor	Avg.
0.1	6	28	15.8
0.2	4	20	11.7
0.3	2	18	10.1
0.4	2	16	7.9
0.5	1	14	7.3
0.6	1	14	6.5
0.7	1	17	6.5
0.8	2	12	5.9
0.9	1	13	5.6
1.0	1	13	5.4

setup follows the same design as the tests without learning. The results are shown in Tables VI, VII and VIII. (Recall that entries marked with an asterix indicate that the optimum was found within 100 iterations in an ensuing improvement phase.) (The "dash" in the entry for *tm2-yes* and **F6** was due to floating point overflow, due a some clauses with large number of literals.)

It is evident from the tables that the learning mechanism generally has a very beneficial effect, and the optimum (or a near optimum) is found for most test cases, when a sufficient large **F** is used, in spite of the lack of sophistication embodied in the improving phase of the search. It is also evident that the learning has very little effect when **F1** is used.

We also tested the effect of introducing controlled probabilistic measures to the move selection process, as outlined in Section 7. As a test case we used jnh1, and ran tests both with, and without learning. **F5** was used for normalization. The rest of the test setup was as for the other tests in this section.

The results are shown in Tables IX and X. As can be seen, neither of the tests shows any benefit from using probabilities for selecting moves during the constructive phase, hence in this phase a deterministic search seems as useful as a probabilistic one. This can be taken as an indication of the focusing ability of both the surrogate constraint and learning mechanisms. Preferred moves, which end up on the top of the candidate list as a result of these mechanisms, are indeed better than the ones further down. If PTS is to be used in the constructive phase, then a value of *p* fairly close to 1 should be used, such that the probabilistic component only exerts a small diversifying effect.

9.3. Tests of improvement methods. As pointed out in Section 5, the surrogate constraint improvement procedure is quite different from the constructive procedure. For example, in a 3-SAT problem, which contains 3 literals in every clause (and hence n_i is the same for all i), there is no difference between the uniform weighting provided by the **F1** function and the weightings provided by other functions. (In a constructive method, by contrast, differences among these weightings emerge because the n_i values change.)

We ran initial tests with random initialization, 5 runs and 10,000 iterations per run, and with a depth of 3 for the surrogate constraint weights. (See Section 5.2). These tests

Table XI. *V* for aim*. Improvement procedure. $\Delta w = 1$.

Name	Best	Wor	Avg.	Min Flips	Max Flips
aim-50-1_6_yes1-1	0	0	0	275	8036
aim-50-1_6_yes1-2	0	0	0	197	2800
aim-50-1_6_yes1-3	0	0	0	251	9230
aim-50-1_6_yes1-4	0	1	0.8	89	-
aim-50-2_0_yes1-1	0	0	0	207	5513
aim-50-2_0_yes1-2	0	2	0.5	220	-
aim-50-2_0_yes1-3	0	0	0	370	2002
aim-50-2_0_yes1-4	0	0	0	103	1163
aim-50-3_4_yes1-1	0	0	0	89	3952
aim-50-3_4_yes1-2	0	0	0	62	1045
aim-50-3_4_yes1-3	0	0	0	60	883
aim-50-3_4_yes1-4	0	0	0	95	620
aim-50-6_0_yes1-1	0	0	0	42	416
aim-50-6_0_yes1-2	0	0	0	61	189
aim-50-6_0_yes1-3	0	0	0	19	247
aim-50-6_0_yes1-4	0	0	0	28	225
aim-100-1_6_yes1-1	0	1	0.4	2686	-
aim-100-1_6_yes1-2	0	1	0.8	7234	-
aim-100-1_6_yes1-3	0	1	0.4	1666	-
aim-100-1_6_yes1-4	0	0	0	1211	3095
aim-100-2_0_yes1-1	0	1	0.6	3019	-
aim-100-2_0_yes1-2	0	4	1	1020	-
aim-100-2_0_yes1-3	0	3	1	905	-
aim-100-2_0_yes1-4	0	1	0.4	1496	-
aim-100-3_4_yes1-1	0	0	0	590	8676
aim-100-3_4_yes1-2	0	5	1.8	510	-
aim-100-3_4_yes1-3	0	4	0.8	575	-
aim-100-3_4_yes1-4	0	0	0	1501	6225
aim-100-6_0_yes1-1	0	0	0	242	472
aim-100-6_0_yes1-2	0	0	0	75	503
aim-100-6_0_yes1-3	0	0	0	172	425
aim-100-6_0_yes1-4	0	0	0	219	780
aim-200-6_0_yes1-1	0	0	0	555	5398
aim-200-6_0_yes1-2	0	0	0	439	1101
aim-200-6_0_yes1-3	0	0	0	452	2021
aim-200-6_0_yes1-4	0	0	0	304	7429

generally failed to find any solutions, apart from the highly structured *tm2-yes* and some from the simpler *aim-6_0* group. One reason for the failure to produce solutions is the obvious lack of diversification present when using only surrogate constraint evaluations to guide the search. Following the tabu search scheme that the best form of diversification is based on the use of adaptive memory and learning, as illustrated by our use of learning in the constructive phase, we therefore embedded a simple learning scheme in all remaining tests of the improvement method. (See next Section).

9.4 Learning and PTS in the in the improvement procedure. To test the effect of learning in the improvement phase, we used $\Delta w = 1$. (See Section 6.2). This ensures that the learning effect is fairly large compared to the initial clause weights, and should give a significant contribution early. The results are presented in Tables XI, XII and XIII, for 5 runs of 10,000 iterations each. The tables gives best, worst and average values for the

Table XII. *V* for *as** and *tm**. Improvement procedure. $\Delta w = 1$.

Name	Best	Wor	Avg.	Min Flips	Max Flips
as2-yes	0	0	0	57	100
as3-yes	0	0	0	56	267
as4-yes	0	0	0	396	1405
as6-yes	0	0	0	177	343
as8-yes	0	0	0	37	407
as10-yes	0	0	0	160	501
as11-yes	0	0	0	64	352
as12-yes	0	0	0	52	163
as13-yes	0	0	0	130	327
as14-yes	0	0	0	40	117
as15yes	0	0	0	284	484
tm2-yes	0	0	0	107	127

cardinality of *V*, and the minimum and maximum number of flips used. (A dash in the "Max Flips" column indicates that the maximum of 10,000 iterations was used. As can be seen, the inclusion of learning has a marked effect on the search, with the optimum found for all the test cases on most runs.

We tested the use of probabilities in this framework, and found similar results as reported for the constructive procedure.

More extensive testing of the various parameters for the learning mechanisms might be done, but we believe that our results clearly indicate the beneficial effect of including learning mechanisms in the search process.

The effect of learning should be discounted over time (by increasing Δw over time in the improvement learning procedure, or similarly setting $\Delta v > 0$ for the constructive learning procedure), if the search has to be run for a longer time (i.e. by including more restarts in the constructive procedure and more iterations in the improvement procedure.)

Table XIII. *V* for jnh*. Improvement procedure. $\Delta w = 1$.

Name	Best	Wor	Avg	Min Flips	Max Flips
jnh1	0	0	0	91	1254
jnh2	0	0	0	275	558
jnh12	0	0	0	514	1908
jnh17	0	0	0	183	814
jnh201	0	0	0	83	156
jnh204	0	0	0	248	564
jnh205	0	0	0	178	1168
jnh207	0	1	0.2	508	-
jnh209	0	0	0	224	1867
jnh210	0	0	0	132	856
jnh212	0	1	0.4	331	-
jnh213	0	0	0	132	607
jnh217	0	0	0	82	1749
jnh218	0	0	0	64	1093
jnh220	0	1	0.2	107	-
jnh301	0	0	0	452	9759

10. Conclusions and future studies.

We have shown how the use of surrogate constraint analysis gives rise to superior guidance when applied to both constructive and iterative procedures for the satisfiability problem, on both structured and random instances. As would be expected, the best guidance is provided for the structured test cases.

Our testing has centered on the goal of applying a surrogate constraint method that uses only the simplest forms of memory, without incorporating more advanced memory-based strategies proposed by tabu search. Nevertheless, we have shown that the associated learning mechanisms can improve the search, in some cases dramatically. In fact, we find that the benefits of this effect dominate those that result from probabilistic decision rules.

A strategy that has been highly effective with surrogate constraint approaches in other settings is strategic oscillation. One interesting area for future studies is to apply this type of procedure to take advantage of the "proximate optimality principle" (POP), which says roughly that good solutions at one level are likely to be found "close to" good solutions at an adjacent level. The relevance of this may be illustrated as follows. A "wrong move" can easily be made at a fairly early stage of a constructive approach, because the information available for making decisions is especially incomplete during early construction stages. Once such a misstep occurs, the evaluations for future moves will be accordingly influenced in such a way that the error becomes "reinforced," since future choices will be designed to favor conditions that support this move. Thus, additional associated errors will occur, each giving rise to future distortions in evaluations. These will consequently lead to choices that further lock in the unfortunate patterns laid down by earlier choices. Ultimately, this reinforced error will be nearly impossible to "undo" when an exchange approach is applied. (In a local sense, changing a previously wrong move can seriously disrupt the current pattern, in view of the layers of other choices subsequently made to support the earlier move.)

This type of scenario discloses the potential utility of the POP notion, which can be exploited by interrupting the constructive process and introducing exchange moves to improve each given level before progressing to the next. In this way, defects are prevented from accumulating. The defects that remain are "closer to the surface," and thus more accessible to being removed by the application of an improvement method when the construction is finally complete.

Another highly effective use of strategic oscillation occurs by means of a *critical event memory* design. Applying this form of memory in conjunction with surrogate constraints has produced the most effective known methods for multidimensional knapsack problems (Glover and Kochenberger (1996); Glover, Kochenberger and Alidaee (1996)). An appealing avenue for future study would be to explore ways to get the fullest advantage from applying strategies based on the POP notion and critical event memory within the surrogate constraint framework.

11. References

Chvtal, V., *A greedy heuristic for the set covering problem*, Mathematics of Operations Research **4** (1979), 233–235.

Connolly, D., *General Purpose Simulated Annealing*. Journal of the Operational Research Society, **43(5)** (1992), 495-505.

Dowsland, K. A., *Simulated Annealing*, in: *Modern Heuristics for Combinatorial Problems*, (Ed. C. R. Reeves), Blackwell, Oxford, (1993).
Freville, A. and Plateau, G., *An exact search for the solution of the surrogate dual of the 0-1 bidimensional knapsack problem*. European Journal of Operational Research **68** (1993), 413-421.
Gavish, B. and Pirkul, H., *Efficient algorithms for solving multiconstraint zero-one knapsack problems to optimality*, Mathematical Programming **31** (1985), 78–105.
Glover, F., *A multiphase-dual algorithm for the zero-one integer programming problem*. Operations Research, **13** (1965), 879–919.
Glover, F., *Surrogate constraint duality in mathematical programming*. Operations Research **23** (1975), 434–451.
Glover, F., *Heuristics for integer programming using surrogate constraints*. Decision Sciences **8** (1977), 156–166.
Glover, F., *Tabu search - Part I*. ORSA Journal on Computing **1(3)** (1989), 190–206.
Glover, F. , Kochenberger, G., *Critical Event Tabu search for Multidimensional Knapsack Problems*, Meta-Heuristics: Theory and Applications. (Eds. J. Kelly and I. Osman). Kluwer Scientific Publishers, (1996), 407–428.
Glover, F. , Kochenberger, G. and Alidaee, B. *Adaptive Memory Tabu Search for Binary Quadratic Programs*, School of Business, University of Colorado, Boulder, (1996).
Greenberg, H. J. and Pierskalla, W. P., *Surrogate Mathematical programs*. Operations Research **18** (1970), 924–939.
Greenberg, H. J. and Pierskalla, W. P., *Quasi-conjugate functions and surrogate duality*. Cahiers du Centre d´Etudes de Recherche Operationelle **15** (1973), 437–448.
Gu, J., Purdom, P. W., Franco, J. and Wah, B. W., *Algorithms for the Satisfiability Problem: A Survey*, working paper, Univ. of Calgary, Canada, (1995).
Hart, J. P. and Shogan A. W., *Semi-Greedy Heuristics: An Empirical Study*. Operations Research Letters, **6(3)** (1987), 107–114.
Hooker, J.. N., *Testing Heuristics: We Have It All Wrong*. Forthcoming in Journal of Heuristics, (1996).
Karwan, M. H. and Rardin, R. L., *Some relationships between Lagrangean and surrogate duality in integer programming*. Mathematical Programming **17** (1979), 230–334.
Løkketangen, A. and Glover, F., *Probabilistic Move Selection in Tabu Search for 0/1 Mixed Integer Programming Problems*, Meta-Heuristics: Theory and Applications. (Eds. J. Kelly and I. Osman). Kluwer Scientific Publishers, (1996), 467–488.
Resende, M. and Feo, T. *A., GRASP for satisfiability*. Research Report, AT&T Bell Laboratories, Murray Hill, NJ. (1994)
Selman, B., Levesque, H. and Mitchell, D., *A new method for solving hard satisfiability problems*, in Proceedings of AAAI-92, San Jose, CA, (1992), 440–446.
Selman, B., Kautz, H., and Cohen, B., *Local search strategies for satisfiability testing*, to appear in DIMACS Series in Discrete Mathematics and Theoretical Computer Science, **Vol 2** (1995).
Yu, G., *On the max-min 0-1 knapsack problem with robust optimization applications*. To appear in Operations Research.

INSTITUTE OF INFORMATICS, MOLDE COLLEGE, MOLDE, NORWAY
E-mail address: Arne.Lokketangen@hiMolde.no

SCHOOL OF BUSINESS, CB 419, UNIVERSITY OF COLORADO, BOULDER, CO, USA
E-mail address: Fred.Glover@colorado.edu