TUTORIAL ON SURROGATE CONSTRAINT APPROACHES FOR OPTIMIZATION IN GRAPHS

Fred Glover

Leeds School of Business
University of Colorado
Boulder, CO 80309-0419

December 2002

Abstract

Surrogate constraint methods have been embedded in a variety of mathematical programming applications over the past thirty years, yet their potential uses and underlying principles remain incompletely understood by a large segment of the optimization community. In a number of significant domains of combinatorial optimization, researchers have produced solution strategies without recognizing that they can be derived as special instances of surrogate constraint methods. Once the connection to surrogate constraint ideas is exposed, additional ways to exploit this framework become visible, frequently offering opportunities for improvement.

We provide a tutorial on surrogate constraint approaches for optimization in graphs, illustrating the key ideas by reference to independent set and graph coloring problems, including constructions for weighted independent sets which have applications to associated covering and weighted maximum clique problems. In these settings, the surrogate constraints can be generated relative to well-known packing and covering formulations that are convenient for exposing key notions. The surrogate constraint approaches yield widely used heuristics for identifying independent sets as simple special cases, and also afford previously unidentified heuristics that have greater power in these settings. Our tutorial also shows how the use of surrogate constraints can be placed within the context of vocabulary building strategies for independent set and coloring problems, providing a framework for applying surrogate constraints that can be used in other applications.

At a higher level, we show how to make use of surrogate constraint information, together with specialized algorithms for solving associated sub-problems, to obtain stronger objective function bounds and improved choice rules for heuristic or exact methods. The theorems that support these developments yield further strategies for exploiting surrogate constraint relaxations, both in graph optimization and integer programming generally.

*Published in the *Journal of Heuristics* 9: 175-227, 2003

1. Introduction.

Surrogate constraint methods were originally introduced as a way to improve decision rules and bounding information in integer programming algorithms (Glover, 1965). They yield stronger relaxations for combinatorial optimization than Lagrangian methods, and have given rise to a well delineated duality theory (Greenberg and Pierskalla, 1970, 1973; Glover, 1975; Karwan and Rardin, 1979; Dyer, 1980; Fréville and Plateau, 1992, 1993; Fréville and Hanafi, 2000). While surrogate constraints have sometimes been used to drive exact solution methods (Dinkel and Kochenberger, 1980; Gavish and Pirkul, 1985; Joseph, Bryson and Gass, 1996), or to generate cutting planes (Glover, Sherali and Lee, 1999) their most prominent use results by applying them within heuristic methods (Kochenberger; McCarl and Wyman, 1973; Klingman and Karney, 1979; Hanafi, 1993; Glover and Kochenberger, 1996; Løkketangen and Glover, 1997; Yu, 1998; Hanifi and Fréville, 2001; Osorio, Glover and Hammer, 2002).

Outside of methods that explicitly use surrogate constraint heuristics, there are many more that embody instances of such heuristics without formal recognition that a link to surrogate constraints exists. Once such a connection is exposed, additional heuristic alternatives emerge that are typically more powerful. Opportunities to improve widely used heuristics have occurred particularly in applications such as covering, multidimensional knapsack problems, scheduling, binary quadratic programming, and satisfiability (SAT). (See, e.g., the references cited above.)

The present tutorial addresses the use of surrogate constraint heuristics within optimization problems over graphs, focusing on independent set problems, with application to graph coloring and associated covering and clique problems, including their weighted versions. In these cases, the surrogate constraints can be generated relative to independent set formulations that have exceedingly simple structures as 0-1 packing or covering problems. Commonly used procedures to create surrogate constraints simplify for these formulations to give a convenient basis for presenting basic surrogate constraint ideas. The derivation yields heuristics that are often embedded in solution methods for independent set problems, and also yields other heuristics not previously considered for these problems. At more advanced levels, dynamic surrogate constraint approaches are introduced that simultaneously generate and solve surrogate constraint relaxations that include clique inequalities, and by extension, inequalities from structures called q-complete systems. As part of this development, we provide theorems that demonstrate how to obtain stronger bounds and improved decision criteria for surrogate constraint approaches in a wide range of optimization settings.

2. Background and Motivation: Links Between Independent Set and Graph Coloring Problems.

Maximum cardinality independent set problems and minimum cardinality coloring problems in graphs are usefully interrelated. Methods for finding maximum independent sets can be embedded in methods for graph coloring, based on the fact that a set of nodes assigned a given color in a coloring problem constitutes an independent set, in which no two nodes are joined by an edge. Thus, it is natural in coloring problems to look for means of restructuring independent sets, which correspond to different assigned colors, to increase the overall average set size and thus reduce the total number of sets (and colors). Successful methods for coloring problems that utilize this relationship between colorings and independent sets have been developed by Hertz and de Werra (1987), Fleurent and Ferland (1995), Morgenstern (1996), Dorne and Hao (1998) and Galinier and Hao (1999). Independent set problems are also of interest because of their equivalence to maximum cardinality clique problems and associated covering problems. An examination of these problems from a surrogate constraint perspective has the useful consequence of yielding results that readily generalize to weighted versions of independent set, clique and covering problems.

To further motivate the use of surrogate constraint strategies in these applications, we stress the utility of applying a vocabulary building process to generate sub-graphs over which independent sets are sought. The vocabulary building framework (Glover and Laguna, 1993; Glover, 1999; Glover, Laguna and Marti, 2000) affords a means to generate specific sub-problems and solutions as a foundation for solving larger problems in which these problems are embedded. Within the present setting such a process gives a natural way to generate and exploit sub-graphs to which the surrogate constraint methods are applied.

3. Preliminary Surrogate Constraint Development: Strategies for Finding a Maximum Independent Set.

Surrogate constraint strategies are often conveniently developed by reference to standard mathematical programming formulations. As a starting point for illustrating this, we represent a selected graph of interest by $G = (N,E)$, where $N = \{1,\ldots, n\}$ denotes the set of nodes of the graph and E denotes the set of edges.

For each node $i \in N$, define

EdgeStar(i) = $\{\{i,j\} \in E\}$,
NodeStar(i) = $\{j: \{i,j\} \in E\}$,
SizeStar(i) = |*NodeStar*(i)| (equivalently, |*Edgestar*(i)|)

More generally, for any subset I of N, define

EdgeStar(I) = $\{\{i,j\} \in E: i \in I\}$ and
NodeStar(I) = $\{j: \{i,j\} \in E, i \in I\}$.
SizeStar(I) = |*NodeStar*(I)| (equivalently, |*Edgestar*(I)|)

By convention, if I is empty, then so are *EdgeStar*(I) and *Nodestar*(I). We also subsequently apply these definitions relative to an induced sub-graph $G' = (N', E')$ of G, where N′ is a subset of N and E′ is the (largest) subset of E determined by N′.

The customary mathematical programming formulation for the maximum independent set problem associates a 0-1 integer variable x_i with each node $i \in N$, where $x_i = 1$ if and only if node i is chosen as an element of the independent set. Then the problem can be expressed as an integer programming (IP) problem, as an instance of a packing problem, as follows.

IP1: Maximize $x_o = \sum (x_i : i \in N)$
subject to $x_i + x_j \le 1 \quad \{i,j\} \in E$
x_i binary $\quad i \in N$

As also commonly observed, the problem can be transformed into an equivalent instance of a covering problem by defining $y_j = 1 - x_j$, whereupon the preceding formulation acquires the form

IP2: Minimize $y_o = \Sigma(y_i: i \in N)$

subject to $y_i + y_j \geq 1 \quad \{i,j\} \in E$

y_i binary $\quad i \in N$

3.1 Foundation for a Surrogate Constraint Heuristic.

A convenient type of surrogate constraint heuristic for such problems, when the goal is to generate approximate solutions quickly, results by weighting the original inequalities from simple normalizations to create the surrogate constraint. To do this, the inequalities are put in a form where all coefficients are nonnegative (as automatically happens here), and the weights are derived by reference to the right hand sides of the inequalities and the sums of their coefficients. The weighted inequalities are then summed to produce the surrogate constraint, which for the present formulations can be represented by

$\Sigma(a_ix_i: i \in N) \leq a_o$ for IP1
$\Sigma(b_iy_i: i \in N) \geq b_o$ for IP2

Thus, we explicitly define the surrogate constraint coefficient values, we explicitly represent the associated surrogate constraint problems derived from IP1 and IP2 by

SC1: Maximize $x_o = \Sigma(x_i: i \in N)$
subject to $\Sigma(a_ix_i: i \in N) \leq a_o$
x_i binary $\quad i \in N$

SC2: Minimize $y_o = \Sigma(y_i: i \in N)$
subject to $\Sigma(b_iy_i: i \in N) \geq b_o$
y_i binary $\quad i \in N$

A general way to produce normalization-based surrogate constraints for 0-1 problems, which we can use to identify the a_i and b_i coefficients for SC1 and SC2, and which will also be relevant to later concerns, may be summarized as follows. Consider a typical "≤ inequality" of a system such as IP1, indexed over some set $h \in H$ and written in the form

$\Sigma(a_{hi}x_i: i \in N) \leq a_{ho}$.

Assume the 0-1 variables x_i are complemented as necessary to assure $a_{hi} \geq 0$ for all $i \in N$ (where the identity of variables complemented may differ for different constraints). Then an appropriate weight w_h to multiply by such an inequality to create a normalized constraint is

$w_h = (\Sigma(a_{hi}: i \in N) - a_{ho})/a_{ho}$.

The weight applies to the form of the original inequality in which the variables have not been complemented as well as to the form used to define w_h.

Similarly, we may consider a typical "≥ inequality" of a system such as IP2 written in the form

$\Sigma(b_{hi}y_i: i \in N) \geq b_{ho}$.

Again, assuming variables have been complemented as necessary to assure $b_{hi} \geq 0$ for all $i \in N$, a corresponding normalization weight is given by

$w_h = b_{ho}/(\Sigma(b_{hi}: i \in N) - b_{ho})$.

Variations of such normalizations raise the indicated weights to some selected power.

Although we need not be concerned with problems that include both negative and positive coefficients here, we remark that advantages are gained for creating and processing surrogate constraints by explicitly complementing variables to maintain the problem in nonnegative-coefficient form. Surrogate constraints can then include both original variables and their complements, which permits a more informed analysis and improved decision rules. (See, for example, Lokketangen and Glover, 1997.)

The special structure of the formulations IP1 and IP2 applicable to the independent set problem assure that the weights to produce the normalization all equal 1, and the normalized inequalities are the same as the original inequalities, thus creating what is called a *simple-sum* surrogate constraint which results by summing the original inequalities without modification. Consequently, the a_o and b_o values in the surrogate constraints for SC1 and SC2 equal the sum of the right hand sides (hence equal the number of inequality constraints), and the a_i and b_i values equal the sum of the unit coefficients for the x_i and y_i variables that appear in these constraints.

By our preceding notation, this gives:

$a_o = b_o = |E|$

$a_i = b_i = \mathit{SizeStar}(i)$

3.2. A Simple Surrogate Constraint Choice Rule.

A choice rule often used in first stage surrogate constraint heuristics can be applied to SC1 and SC2 by selecting the variable x_r or y_r that gives the best ratio (largest for maximization, smallest for minimization) of the objective function coefficient to the surrogate constraint coefficient. The selected variable is set equal to 1, and the problem is reduced by eliminating variables that receive forced value assignments and by removing redundant constraints. Then a surrogate constraint is constructed relative to the new problem and the process repeats.

Such a process becomes quite straightforward in the present setting due to the special structure of the problem. Since all objective function coefficients are 1, the choice rule reduces to selecting x_r and y_r to yield $a_r = \mathrm{Min}(a_i)$ and $b_r = \mathrm{Max}(b_i)$. In terms of the graph G the rule can be expressed as follows.

Choose $r \in N$, for setting $x_r = 1$ or $y_r = 1$, by identifying

$\mathit{Sizestar}(r) = \mathrm{Min}(\mathit{SizeStar}(i): i \in N)$ for IP1

$\mathit{Sizestar}(r) = \mathrm{Max}(\mathit{SizeStar}(i): i \in N)$ for IP2

These simple choices for IP1 and IP2 correspond to decision rules frequently embedded in constructive strategies and branch and bound approaches applied to maximum independent set problems, as in the studies of Friden Hertz and de Werra (1989, 1990), Feo, Resende and Smith (1994), Homer and Peinado (1996), Dorne and Hao (1998), and Abello, Pardelos and Resende (1999) among others. (These rules were first proposed by Johnson (1974)[1], about a decade after the introduction of surrogate constraint strategies.)

As we will see, the surrogate constraint framework can be used to provide strategies that are substantially more advanced. First, however, we complete the preliminary connections by identifying the problem updates that occur both by reference to the mathematical programming formulations and the underlying graph structures. The outcomes illustrate a characteristic feature of surrogate constraint approaches applied to graph problems generally. Whenever the decision

[1] This paper formulated such choice criteria for the maximum clique problem, but they translate directly into the rules described here. More general cases of the covering problem were also considered in the same paper, similarly relying on a rule equivalent to the simple-sum surrogate constraint rule. Such a rule is less effective for these cases than using more general normalizations, but the focus of Johnson (1974) was on complexity analysis rather than on heuristic efficacy. Similar proposals and an extended complexity analysis also appear in Chvatal (1979).

variables correspond to operations on a graph (such as adding or deleting nodes and edges), then the updated mathematical programming formulations and the new surrogate constraints that result from each choice can be expressed directly in terms of the corresponding graph update. Similarly, implications of the decision such as forcing particular variables to 0 or 1 and causing constraints to become redundant can also be reflected by corresponding changes in the graph structure.

In the present case, according to the problem considered, the updates for problems IP1 and IP2 correspond to simple graph reductions as follows.[2]

IP1 Update: Setting $x_r = 1$ in IP1 forces $x_j = 0$ for the variables x_j in each of the constraints $x_r + x_j \le 1$, which makes the updated form of these constraints redundant. In the graph G this corresponds to dropping all the associated nodes j and their incident edges, thereby defining a new node set N′ and a new edge set E′ by

$$N' = N - \mathit{NodeStar}(r)$$
$$E' = E - \mathit{EdgeStar}(\mathit{NodeStar}(r))$$

The reduced problem has exactly the same form as the original, by replacing G = (N,E) with the reduced graph G′ = (N′,E′). Thus, defining *SizeStar′*(i) relative to G′, for each node $i \in \mathit{NodeStar}(\mathit{NodeStar}(r))$ yields

$$\mathit{NodeStar}'(i) = \mathit{NodeStar}(i) - \mathit{NodeStar}(\mathit{NodeStar}(r))$$
$$\mathit{EdgeStar}'(i) = \mathit{EdgeStar}(i) - \mathit{EdgeStar}(\mathit{NodeStar}(r)).$$

These outcomes can be directly identified by examining each $j \in \mathit{NodeStar}(r)$ and then, for each $i \in \mathit{NodeStar}(j)$, removing node j from *NodeStar*(i) and removing the edge {i,j} from *EdgeStar*(i) (reducing *SizeStar*(i) by 1 for each such step). Hence in the surrogate constraint SC1 the coefficients are changed so that

$$a'_i = a_i - |\mathit{NodeStar}(i) \cap \mathit{NodeStar}(r)|$$
$$a'_o = a_o - \mathit{SizeStar}(\mathit{NodeStar}(r)).$$

The problem can be further reduced by setting $x_j = 1$ for all isolated nodes $j \in N'$ (all of whose constraints reduce to the form $x_j \le 1$), and removing these nodes from N′.

IP2 Update: Setting $y_r = 1$ in IP2 makes each inequality $y_r + y_j \ge 1$ in which y_r appears redundant. Hence removing y_r and these redundant constraints results in defining

$$N' = N - r$$
$$E' = E - \mathit{EdgeStar}(r).$$

As in the case of IP1, the reduced problem has exactly the same form as the original, by replacing G = (N,E) with the reduced graph G′ = (N′,E′). Defining *SizeStar′*(i) relative to G′ yields

$$\mathit{SizeStar}'(i) = \mathit{SizeStar}(i) - 1 \text{ for each } i \in \mathit{EdgeStar}(r).$$

Hence in the surrogate constraint SC2 the coefficients are changed so that

$$b'_i = b_i - 1 \text{ for each } i \in \mathit{EdgeStar}(r)$$
$$b'_o = b_o - \mathit{SizeStar}(r).$$

[2] Such reductions are now common. We include them for completeness to show their connection with customary surrogate constraint processes and to provide a background for implementing procedures described later.

Nodes i that are isolated in N′ (i.e., that have no incident edges) correspond to variables that appear only in redundant inequalities of the form $y_i \geq 0$, and hence these nodes can all be dropped from N′ together with setting $y_i = 0$.

Additional dominance criteria can be applied to assign values to variables in IP1 and IP2. However, the illustrated surrogate constraint heuristics have the convenient feature that their choice criteria automatically assure each assignment is the same as if such dominance criteria had been explicitly identified and used to assign values to the variables. For classical definitions of dominance, this property holds not only for this special class of graph problems but for integer programming problems in general.

4. More General Surrogate Constraint Heuristics.

Beyond the fact that the two elementary strategies illustrated for IP1 and IP2 correspond to popular constructive procedures, it is easy to see that the heuristic described for IP2 can be applied directly to the IP1 formulation, since each choice $y_r = 1$ corresponds to a choice $x_r = 0$. The updated graph G′ for IP1 in this case is exactly as specified for IP2. This affords a way to extend the preceding approaches within the preliminary framework discussed so far. As characteristically done in surrogate constraint methods, the choice rules can be applied to select a variable to set either to 1 or to 0 at each step, an option that gives a more flexible heuristic than the two separate methods described for IP1 and IP2.

Additional possibilities result by taking fuller advantage of the surrogate constraint framework. We can divide these into two main types of approaches. The first involves the use of quickly executed look-ahead strategies, which are commonly used with surrogate constraint choice rules to identify the consequences of alternative assignments, enabling better choices to be made by accounting for these projected outcomes. The second involves the use of bounding information, making it possible to exploit the fact that the surrogate constraint creates a relaxation for the original problem. This provides a means to develop choice rules based on considerations that go beyond the considerations examined so far, giving an opportunity to develop more advanced forms of look-ahead strategies.

Quickly executed strategies are important in a variety of settings, and although they do not provide the primary focus of this tutorial, we describe these types of surrogate constraint approaches in Appendix 1. Such methods can be used to supplement those we address in the following sections, where we begin with procedures that are fairly elementary, and which are likewise fast to execute (if not quite as fast as those in Appendix 1).

5. Enhanced Choices Using Surrogate Constraint Bounding Information.

We restrict the discussion in this and remaining sections of the paper to the formulation IP1 and the choice rules for setting $x_r = 1$. Drawing on the preceding observations our comments also apply to the formulation IP2 and the choice rules for setting $y_r = 1$ (or, equivalently, for setting $x_r = 0$ in IP1).

5.1 An Optimal Surrogate Constraint Solution.

The "ratio choice rule" identified in Section 3 for successively choosing variables x_r to set equal to 1 (which reduces in the present context to identifying r to yield the minimum a_i value), also solves the linear programming relaxation of the surrogate constraint problem; i.e., it yields a solution to maximize x_o subject to the surrogate constraint. Due to the fact that the objective function for the maximum independent set problem contains only unit coefficients, we can also obtain an optimal IP solution to the surrogate constraint problem SC1.

Specifically, assume the coefficients a_i are placed in ascending order. Then the optimum value x_o for the surrogate constraint relaxation SC1 is given by

$$x_o = \text{Max}(h:\ a_1 + ... + a_h \leq a_o)$$

which corresponds to the solution obtained by setting $x_j = 1$ for $j \leq h$, and $x_j = 0$ for $j > h$.[3]

The bounding information from the solution to SC1 can be used to improve quickly implemented look-ahead rules, as discussed in Appendix 1. However, we can carry the analysis farther. The succeeding sections show how to generate information at another level.

5.2 A Higher Level Surrogate Constraint Solution.

An important strategy in surrogate constraint applications is to identify additional problem constraints, either explicit or implied, that can be adjoined to the surrogate constraint problem while creating a structure that permits the problem to be solved efficiently. Bounded sums of variables that satisfy a nesting property are useful for generating such a structure (Glover, 1971), and we can conveniently extract such constraints as a subset of the inequalities of IP1. As before, our comments for IP1 can be translated into related observations for IP2. In particular, we consider the special case of nesting where the component constraints are disjoint, i.e., the sums of variables are defined over disjoint sets. The corresponding surrogate constraint problem then becomes

$$\begin{array}{lll} \textbf{SC}\text{: Maximize } x_o & = \sum (x_i : i \in N) & \\ \quad\text{subject to} & \sum (a_i x_i : i \in N) \leq a_o & \\ & x_i + x_j \leq 1 & \{i,j\} \in E' \\ & x_i \text{ binary} & i \in N \end{array}$$

where the edges in E′ are pairwise node disjoint, and consequently each variable x_i, $i \in N$, appears in at most one inequality of the collection. It is easy to see that an optimal solution to this problem results as follows. Under the assumed ascending order of the a_i coefficients, define

$$N' = N - \{\ i \in N:\ \{i,j\} \in E' \text{ and } i > j\ \}$$

Hence N′ is obtained from N by dropping exactly one node i from each edge $\{i,j\} \in E'$, which is associated with the larger (or equal) of the two corresponding coefficients in the surrogate constraint.

Then the auxiliary constraints over E′ can be removed and the problem reduces to:

$$\begin{array}{lll} \textbf{SC}'\text{: Maximize } x_o & = \sum (x_i : i \in N') & \\ \quad\text{subject to} & \sum (a_i x_i : i \in N') \leq a_o & \\ & x_i \text{ binary} & i \in N' \end{array}$$

Since SC′ has the same form as the original surrogate constraint problem SC1, we may identify an optimal solution exactly as in Section 5.1. However, since N′ removes roughly half the coefficients from N, the solution is more restricted and SC′ will generally yield a somewhat stronger bound on x_o than SC1.

[3] A useful observation in this setting often escapes notice. If the integer a_i coefficients fall within a modest range, or do not have many gaps between them, they can be rapidly sorted by a single pass that records each index $i \in N$ in a list for the associated integer $v = a_i$. Then a pass of the integers v between Min(a_i) and Max(a_i) recovers the a_i coefficients in sorted order, stopping as soon as the "a_h term" is identified. (The popularity of more complex sorting techniques has obscured the merits of this simple alternative.) Such an approach is also relevant for sorting integer values $U_o(i)$ which are referenced by additional algorithms discussed later.

A rule for implicitly choosing the set E′ to generate N′ and the reduced problem SC′ is as follows (again assuming the a_i coefficients are in ascending order):

Method to Create SC′ from SC.

1. Let i = 1 and N′ = N.
2. Identify the first (smallest) j > i such that {i,j} ∈ E and j ∈ N′.
   (a) If no such node j exists, proceed directly to Step 3.
   (b) If node j exists, remove node j from N′ (implicitly, designating {i,j} to belong to E′).
3. Let i′ = Min {q : q > i and q ∈ N′}. If i′ does not exist, or is the last (largest index) node in N′, stop: N′ now has its final form. Otherwise, let i : = i′ and return to step 2.

The preceding method is designed to efficiently obtain a good N′ (and E′) to produce the reduced surrogate constraint problem. Each successive choice of i identifies a node that is undominated over choices of nodes to remain in N′, given choices previously made, and each choice of j is locally best for the associated node i.

An analogous method can go through the indexes i in reverse order, seeking the largest j < i such that {i, j} ∈ E and j ∈ N′. In this case node i is the one dropped from N′ and node j must be marked to prevent it from being chosen as a future node i. Other more complex variants are also possible, such as choosing an edge {i, j} at each step, for j > i, to minimize $a_j – a_i$, although such variants involve greater computation.

6. Exploiting the Reduced Surrogate Constraint Problem.

The reduced surrogate constraint problem SC′ can be exploited exactly as the original surrogate constraint problem, but we propose a more ambitious method for doing this. First we identify a subset N* of N that contains candidate nodes we wish to evaluate more fully. (For example, N* can be taken to have the form identified in the threshold-based strategy in Appendix 1, where the determination of node h in Section 5.1 is made relative to N′ rather than N.) The following method can be executed independently for each node i ∈ N*, and hence can be efficiently exploited by parallel processing.

Node - Specific SC Bounding and Choice Rule

1. For each i ∈ N*, tentatively perform the assignment $x_i = 1$. Reduce the original problem IP1 as indicated in Section 3 to produce a problem denoted IP1(i).
2. Generate the surrogate constraint problems SC and SC′ for IP1(i), representing these problems as SC(i) and SC′(i). Denote the optimum x_o value for SC(i) and SC′(i) by x_o (SC(i)).
3. Identify r ∈ N* to give the choice $x_r = 1$ by the rule
   x_o (SC(r)) = Max (x_o (SC(i)) : i ∈ N*)

If the value x_o (SC(i)) found in Step 2 above does not exceed x_o*, the x_o value for the best known solution to IP1, then the assignment $x_i = 0$ is appropriate and the problem IP1 can be simplified. A strategy that presupposes a restrictive value for x_o* can be useful to induce such a problem reduction heuristically.

We now show how to strengthen the x_o(SC(i)) values to obtain more restrictive bounds on x_o and to yield more refined decisions.

7. First Principle of Conditionally Shared Limitations.

In this section we take advantage of what may be called the *First Principle of Conditionally Shared Limitations*, which may expressed as follows: Knowledge of bounds on x_o for conditionally constrained problems, where values (or bounds) are tentatively assigned to specific x_i variables, can be used to derive stronger bounds on x_o in other conditionally constrained problems, defined relative to assigning values (or bounds) to other x_i variables.

To see how this principle can be applied in the present setting, let N(r) be the subset of N identifying the nodes i over which the surrogate problem SC(r) is defined (after the reduction of setting $x_r = 1$), and let N′(r) be the further reduced set of nodes for the associated problem SC′(r). Also, let NO(r) be the *optimal node set* for SC′(r) (and hence SC(r)), where setting $x_i = 1$ for $i \in NO(r)$ yields an optimal solution to SC′(r). Consequently, $|NO(r)| = x_o(SC(r))$.

The value $x_o(SC(r))$ provides an upper bound for x_o in the problem IP1 when $x_r = 1$, and we undertake to find an improved bound, taking advantage of information about the composition of NO(r). In general, for an arbitrary node i, let $U_o(i)$ denote an upper bound for x_o when $x_i = 1$. Then, more precisely, we use the knowledge that $U_o(i) \leq x_o(SC(i))$, together with the composition of NO(r), to obtain a tighter value for $U_o(r)$ than $x_o(SC(r))$.

The rationale underlying our approach may be sketched as follows. If $x_o(SC(i)) < x_o(SC(r))$ for any node $i \in NO(r)$, then node i cannot belong to an independent set of size $x_o(SC(r))$. However, since node i has been used to yield $x_o(SC(r))$ (by solving SC′(r)), this discloses that the value $x_o(SC(r))$ is larger than needed for $U_o(r)$. We can conclude that $U_o(r)$ should be no larger than $x_o(SC(i))$, or else node i should not belong to NO(r). But the exclusion of node i from NO(r), hence from N(r), imposes an added restriction on the problem SC(r), compelling $x_i = 0$. Hence this exclusion also affords the possibility to reduce $U_o(r)$ below $x_o(SC(r))$.

Denote the set of nodes in NO(r) that create this possibility, by

$$I(r) = \{i \in NO(r): U_o(i) < U_o(r)\}$$

where initially we begin by setting $U_o(i) = x_o(SC(i))$ for all $i \in N$. Let SC(r\i) denote the surrogate constraint problem SC(r) subject to $x_i = 1$. Then for any $i \in I(r)$, the previous observations lead to the conclusion that we can compel $U_o(r)$ to satisfy

$$U_o(r) \leq \mathrm{Max}(\, U_o(i),\ x_o(SC(r\backslash i)).$$

We increase the generality of the foregoing observation as follows. Let SC(r\I) for an arbitrary subset I of N denote the surrogate constraint problem SC(r) subject to $x_i = 0$ for $i \in I$. That is, in this problem all nodes of I are excluded from belonging to the same independent set as node r. Note that the solution of the problem SC(r\I) requires removing the nodes of I from N in the SC problem, i.e., setting $x_i = 0$ for $i \in I$, before the transformation that creates SC' from SC. Then we can state the following result.

Theorem 1. If I is any nonempty subset of I(r), then a legitimate value for $U_o(r)$ is given by

$$U_o(r) = \mathrm{Max}(x_o(SC(r\backslash I)), \mathrm{Max}(U_o(i): i \in I)).$$

Proof: The result follows directly from the reasoning already stated, together with the fact that either (a) $x_i = 0$ for all $i \in I$ or (b) $x_i = 1$ for at least one $i \in I$. Condition (a) underlies the term $x_o(SC(r\backslash I)$ and condition (b) underlies the term $\mathrm{Max}(U_o(i): i \in I)$. □

The value $x_o(SC(r\backslash I))$ in the theorem can be replaced by $x_o(r\backslash I)$, the corresponding value of x_o in the original IP1 problem (i.e., the value in IP1 that results by setting $x_r = 1$ and $x_i = 0$ for

$i \in I$). This produces a tighter limit for $U_o(r)$, but requires considerably more computational effort to exploit.

To take full advantage of Theorem 1, we seek to identify a set I that gives a smallest limiting value for $U_o(r)$ by the stipulations of the theorem. This goal may be achieved as follows.

Assume $I(r)$ is nonempty and denote the distinct values taken by the bounds $U_o(1)$ for $i \in I(r)$, listed in ascending order, by $u_1 < \ldots < u_k$, $k \geq 1$. Define the collection of sets $I_j = \{i \in I(r): U_o(i) \leq u_j\}$ for j = 1 to k. (Hence, each of these sets is contained in the next and $I_k = I(r)$.)

Theorem 2. A set $I = I^*$ that yields a tightest bound for $U_o(r)$ by Theorem 1, and the corresponding $U_o(r)$ value, are given by:

(a) If $u_1 \geq x_o(SC(r\backslash I_1))$, then $I^* = I_1$ and $U_o(r) = u_1$
(b) If $u_k \leq x_o(SC(r\backslash I_k))$, then $I^* = I_k$ and $U_o(r) = x_o(SC(r\backslash I_k))$
(c) If neither (a) nor (b) apply, then define
$p = \text{Max}(j: u_j \leq x_o(SC(r\backslash I_j))$
$q = \text{Min}(j: u_j > x_o(SC(r\backslash I_j))$ $(q = p + 1)$
If $u_q \leq x_o(SC(r\backslash I_p))$, then $I^* = I_q$ and $U_o(r) = u_q$
Otherwise, $I^* = I_p$ and $U_o(r) = x_o(SC(r/I_p))$

Proof: First, we show that a tightest bound can be found by restricting attention to the sets I_j, $j = 1, \ldots, k$, as candidates for I*. Assume that a given set I* is optimal, giving a smallest value for $U_o(r)$ by Theorem 1. Identify the value u_j such that $u_j = \text{Max}(U_o(i): i \in I^*)$. Then $\text{Max}(U_o(i): i \in I^*) = \text{Max}(U_o(i): i \in I_j)$ and in addition, since $I^* \subseteq I_j$, it follows that $x_o(SC(r\backslash I_j)) \leq x_o(SC(r\backslash I^*))$. Hence I_j must also qualify to be I*. Next, it is clear that if the conditions specified in either (a) or (b) hold, then the conclusions stipulated in these cases are valid. For (c), if $j < p$, then $x_o(SC(r\backslash I_j)) \geq x_o(SC(r\backslash I_p))$, while if $j > q$ then $\text{Max}(U_o(i): i \in I_j) > \text{Max}(U_o(i): i \in I_q)$. Consequently, the $U_o(r)$ value specified by Theorem 1 in each respective case cannot be smaller than that specified by taking $I^* = I_p$ or $I^* = I_q$. The preferred choice between I_p and I_q therefore determines I*, as stipulated. (The result is also valid if (c) is modified to replace "≤" by "<" and to replace ">" by "≥".) □

To implement Theorems 1 and 2, parallel processing can allow the determination of tightened $U_o(r)$ values to be executed simultaneously for each r in N, or for each r in some candidate subset N* of N. If improved $U_o(i)$ values are produced for some indexes i, the entire process can be repeated to determine whether these values may again imply a smaller $U_o(i)$ value for some $i \in N$. The smallest $U_o(i)$ value cannot change, and if the process is applied serially, going to the index i that yields the next largest $U_o(i)$ value at each step, then exactly one pass of the nodes in N will complete the application. However, a parallel approach will usually be significantly faster.

We will revisit Theorems 1 and 2 in Section 11, where we provide a generalization that derives still stronger bounds by reference to ideas introduced in the next three sections.

8. Second Principle of Conditionally Shared Limitations

The *Second Principle of Conditionally Shared Limitations* complements the first, by undertaking to take advantage of it in a special way. This second principle may be summarized by the statement: Conditionally Shared Limitations can be exploited by creating a specific optimization problem that imposes conditional constraints on x_o.

We demonstrate this principle by showing how the bounds $U_o(i)$ for $i \in N(r)$ (the index set for SC(r)) can be embedded in an optimization problem that is specifically designed to tighten the bound $U_o(r)$. For convenience, we develop the relevant results in connection with the problem SC, understanding that they apply to SC(r) simply by enforcing $x_r = 1$ and redefining SC to be the new problem that results after the appropriate reductions. Noting that $x_o(SC) \leq U_o(j)$ must hold for each $j \in N$ such that $x_i = 1$ in the solution that yields the optimum value $x_o(SC)$, we can formulate a secondary surrogate problem SS associated with the problem SC as follows:

SS: Maximize $x_o = \sum(x_i: i \in N)$

$$\begin{array}{ll} \sum(x_i: i \in N) \leq U_o(j)x_j + |N|(1-x_j) & j \in N \\ x_i + x_j \leq 1 & \{i,j\} \in E'' \\ x_i \text{ binary} & i \in N \end{array}$$

The inequalities for $j \in N$ can also be replaced by the nonlinear inequality

$$\sum(x_i: i \in N) \leq \text{Min}\,(U_o(i): x_i = 1, i \in N).$$

As in the case for the edge set E′ in problem SC, we assume E″ is a subset of E whose edges share no nodes in common (hence E″ is a matching on the graph G), and we will identify a way to quickly extract a good candidate for E″. The problem SS can be used to find a bound on x_o for IP1, but our primary interest will be to use the version SS(r) when N is replaced by N(r) in order to identify an optimum value $x_o(SS(r))$. This latter value, like the value $x_o(SC(r))$, gives an upper bound $U_o(r)$ for x_o when $x_r = 1$.

First we identify how to solve SS (and hence, using the same rules, to solve SS(r)). Although the structure of SS is significantly different from that of SC, an optimal solution can be obtained by a similar method. Just as the a_i coefficients are indexed in ascending order to extract a good set E′ from E for SC, and to identify a reduced problem SC′, analogous operations are applied to extract a good set E″ from E for SS, and to identify a reduced problem SS″, but by reference to an ordering of the $U_o(i)$ values.

Theorem 3. For each edge $\{i, j\} \in E''$, indexed so $U_o(j) \leq U_o(i)$, remove node j from N (setting $x_j = 0$). Denote the set of nodes that remain by N″, and define SS″ to be the problem that results from SS by replacing N with N″ and by dropping the inequalities over E″. Then problem SS″ shares at least one optimal solution in common with SS.

Proof: Suppose there is an edge $\{i, j\}$ where $U_o(j) \leq U_o(i)$ and $x_j = 1$, $x_i = 0$ in an optimal solution to SS while $j \notin N''$. But then the solution to SS can be changed by setting $x_j = 0$ and $x_i = 1$ without violating any constraints and without altering x_o (SS). Repeating the process ultimately yields a solution to SS″ with $x_o = x_o(SS)$. But $x_o(SS'') \leq x_o(SS)$ since SS″ is more restricted than SS, and hence the resulting solution must be optimal for SS″. □

Theorem 4. Index the values $U_o(i)$, $i \in N''$ in descending order ($U_o(i) \geq U_o(i+1)$ for $i, i + 1 \in N''$). Let $p = \text{Max}(i \in N'' : i \leq U_o(i))$. Then an optimal solution to SS″ is obtained by setting $x_i = 1$ for $i \in N''(r)$ if and only if $i \leq p$, yielding $x_o(SS'') = p$.

Proof: First, it is clear that the indicated solution is feasible for the inequalities $\sum(x_i : i \in N'') \leq \text{Min}(U_o(i): x_i = 1, i \in N'')$. Moreover, if any $x_j = 1$ for $j > p$, then $U_o(j) \leq U_o(p)$ and the foregoing inequality implies $\sum(x_j : i \in N'') \leq U_o(j)$, establishing that $x_o(SS'') \leq p$. □

The preceding theorems show that each problem SS(r) can be transformed into a problem SS″(r) and solved with essentially the same level of effort to transform SC(r) into SC′(r) and solve the result. Similarly, the solution of each SS(r) problem can be carried out in parallel.

Finally, we observe that an edge set E″ for SS can be generated by exactly the same rule for generating an edge set E′ for SC, by replacing the a_i coefficients with the values $U_o(i)$, and reversing the ordering. (The correspondence can be achieved more precisely by reference to an ascending ordering of the $-U_o(i)$ values.)

9. Generalizations to Additional Optimization Problems

As already noted, the method based on the first principle of conditionally shared limitations is exceedingly general, and applies to a wide range of optimization problems. The method described for exploiting the second principle of conditionally shared limitations can similarly be generalized.

The procedure can immediately be extended from the maximum cardinality independent set problem to the maximum weight independent set problem, while exploiting the ability to create reductions of the surrogate constraint problem SC and the secondary surrogate problem SS. For this, it suffices to modify the previous formulations by introducing a positive weight c_i associated with each node i in the objective function.

9.1. The Weighted SC Problem

The weighted SC problem acquires the form

W-SC: Maximize $x_o = \sum(c_i x_i : i \in N)$

$$\begin{aligned} \text{subject to} \quad & \sum(a_i x_i : i \in N) \leq a_o \\ & x_i + x_j \leq 1 \qquad \{i,j\} \in E' \\ & x_i \text{ binary} \qquad i \in N \end{aligned}$$

The edges in E′ are pairwise node disjoint, as before. We solve the continuous (LP) relaxation of this problem by allowing the binary conditions to be replaced by $0 \leq x_i \leq 1$, $i \in N$. In the case where the inequalities over E′ are not present, then as previously intimated an optimal continuous solution can be obtained by the following well-known rule.

LP Knapsack Solution

1. Index the variables so that $c_i/a_i \geq c_{i+1}/a_{i+1}$ for all $i, i + 1 \in N$.
   Define $A(0) = 0$, and for $p \in N$; define $A(p) = (\sum a_i : i \leq p, i \in N)$
2. Identify $q = \text{Max}(p \in N : A(p) \leq a_o)$. Then set
   $x_i = 1$ for $i \leq q$ (if $q \geq 1$)
   $x_{q+1} = (a_o - A(q))/a_{q+1}$ (if $q < n$)
   $x_i = 0$ for $i > q + 1$ (if $q + 1 < n$)

The solution to W-SC can be obtained by a similar rule after a preliminary transformation to create a related problem W-SC′ defined over a set of nodes N′ with the inequalities over E′ removed.

Transforming W-SC into W-SC′

0. Begin with N′ = N. For each $\{i, j\} \in E'$, assume an indexing so that $c_i/a_i \geq c_j/a_j$, and if the s are equal, then $a_i \geq a_j$.
1. For each $\{i, j\} \in E'$, if either $c_i \geq c_j$ or $a_i \geq a_j$, then drop j from N′. Otherwise if $c_j > c_i$ and $a_j > a_i$, create a new variable $z_i = x_j + x_i$ to replace x_i. Then z_i receives the coefficients c_i and a_i formerly associated with x_i, and x_j receives new coefficients given by
$$c_j := c_j - c_i$$
$$a_j := a_j - a_i$$
2. Problem W-SC′ is the resulting form of W-SC that drops the inequalities over E′ and replaces N by N′.

We can now state:

Theorem 5: An optimal LP solution to W-SC is obtained by applying the LP Knapsack Solution Rule to W-SC′ (treating the z_i variables in the same way as x_i variables). If an x_j variable is positive in this knapsack solution for some $\{i, j\} \in E'$, then the solution also yields $z_i = 1$, and the corresponding optimal solution to W-SC assigns x_i the value $x_i := z_i - x_j$. Optimal values for remaining variables in the W-SC solution are those specified by the knapsack solution, without modification.

Proof: First, under the assumed indexing of Step 0, we show that if $a_i \geq a_j$ for $\{i, j\} \in E'$, then $x_j = 0$ in an optimal LP solution to W-SC. Let x_i' and x_j' be optimal LP values for x_i and x_j, and suppose $x_j' > 0$. Consider a solution x″ where all variables except x_i and x_j retain their current values, while $x_j'' = 0$ and $x_i'' = x_i' + (a_i/a_j)x_j'$. Then $x_i'' + x_j'' = x_i'' \leq x_i' + x_j'$, and hence the solution satisfies the inequality $x_i + x_j \leq 1$. Also it is easy to verify that
$$a_ix_i'' + a_jx_j'' = a_ix_i' + a_jx_j' \text{ and } c_ix_i'' + c_jx_j'' = c_ix_i' + c_jx_j'.$$
Hence the solution x″ satisfies the knapsack constraint and gives an x_o value at least as large as the solution x′. We therefore assume henceforth that $a_j > a_i$. The indicated transformation of variables to create $z_i = x_i + x_j$ (≤ 1) clearly yields a relaxation of W-SC, which becomes equivalent to W-SC if $x_i = z_i - x_j$ is assured to be nonnegative. Denote the new coefficients for x_j by $c_j' = c_j - c_i$ and $a_j' = a_j - a_i$. If $c_i > c_j$, then $c_j' \leq 0$ and $a_j' > 0$, which implies $x_j = 0$ in an optimal solution to the relaxation, and hence j may be discarded as specified in the theorem. We are left with the case $c_j' > 0$ and $a_j' > 0$. The indexing assumptions imply $c_i/a_i > c_j'/a_j'$. As a result, $x_j > 0$ implies $z_i = 1$ in the LP Knapsack Solution to W-SC′, and the feasibility (and hence optimality) of this solution for W-SC is established.[4] □

[4] A method equivalent to the approach provided by this theorem, but which does not introduce an explicit transformation of variables and which is slightly more cumbersome to state and prove, is provided in Glover (1971).

Theorem 5 finds its usefulness in the present context by enabling maximum weight independent set problems to be handled in a manner analogous to maximum cardinality problems. By the now-familiar design, we may create associated reduced problems by setting $x_r = 1$ for candidate variables, and solving the resulting W-SC(r) problems in parallel.

An appropriate choice of an edge set E′ to be exploited by Theorem 5 can also be made by methods resembling those previously described for unweighted problems. The procedure by which N′ is generated from N (when W-SC is transformed into W-SC′) suggests that E′ be derived from E by successively choosing "best remaining ratios" to identify both i and j for the pair $\{i, j\} \in E'$.

9. 2 The Weighted SS Problem

The weighted SS problem is expressed by

W-SS: Maximize $x_o = \sum (c_i x_i : i \in N)$

$$\begin{array}{lll} \text{subject to} \quad \sum (c_i x_i : i \in N) \le U_o(j)x_j + U_o(1-x_j) & j \in N \\ x_i + x_j \le 1 & \{i,j\} \in E'' \\ x_i \text{ binary} & i \in N \end{array}$$

where U_o is an upper bound on x_o (e.g. $U_o = \sum (c_i : i \in N)$). The $U_o(i)$ terms are defined as before, taking into account the changed form of x_o. Also, E″ is a set of node disjoint edges, and the inequalities over $j \in N$ can alternately be expressed as

$$\sum (c_i x_i : i \in N) \le \text{Min}(U_o(i) : x_i = 1, i \in N).$$

In the absence of the inequalities over E″, the solution to W-SS, perhaps surprisingly, takes no account of the c_i values in creating a priority ordering of the variables.

Specifically, define the version of W-SS that excludes the inequalities over E″ as the *Conditional Objective Knapsack Problem* (since the objective value is conditional upon the bounds $U_o(i)$). Then the continuous relaxation of this problem can be solved as follows.

LP Conditional Objective Knapsack Solution

```
1. Index the variables so that $U_0(1) \ge U_0(2) \ge \ldots \ge U_0(n)$. Define $C(0) = 0$ and for each
   $p \in N$ define $C(p) = \sum (c_i : i \le p, i \in N)$
2. Define $q = \text{Max}(p: C(p) \le U_0(p))$ and set
       $x_i = 1$ for $i \le q$   (if $q \ge 1$)
   If $q < n$, then
       (a) if $C(q) \ge U_0(q+1)$ then $x_i = 0$ for $i > q$ and $x_0 = C(q)$
       (b) if $C(q) < U_0(q+1)$ then $x_{q+1} = (U_0(q+1) - C(q)) / c_{q+1}$,
           $x_i = 0$ for $i > q+1$ (if $q+1 < n$), and $x_0 = U_0(q+1)$
```

The division between the cases 2(a) and 2(b), and the fact that the indexing makes no reference to the c_i values, constitute noteworthy differences between the rule for generating the LP Conditional Objective Knapsack Solution and the rule for generating the more traditional LP Knapsack Solution. Nevertheless, the computational effort is approximately the same.

To solve the more complex W-SS problem, which includes the constraints over E″, requires a method that departs in a still more significant way from the method for W-SC, although the execution is still exceedingly efficient. In this case, W-SC is not transformed into a second problem in advance, but is solved directly by a method partly analogous to the preceding solution approach.

A descending ordering of the $U_0(i)$ values again identifies a critical priority relationship for solving the problem. This suggests that the set E″ can be generated by reference to this ordering, using rules similar to those previously described for generating node disjoint subsets of edges from E.

Method for W-SS

```
1. Index the variables so $U_0(1) \geq U_0(2) \geq \ldots \geq U_0(n)$. Start with $q = 1$ and $CSUM = 0$.
2. (a)  If there exists an edge $\{i, q\} \in E''$ with $i < q$ then:
            (1) if $c_i \geq c_q$, set $x_q = 0$
            (2) if $c_i < c_q$ let $d = c_q - c_i$. Then
                (i)  if $CSUM + d \leq U_0(q)$ set $x_q = 1$, $x_i = 0$ and $CSUM := CSUM + d$
                (ii) if $CSUM + d > U_0(q)$ set $x_q = f$ and $x_i = 1 - f$ for $f = (U_0(q) - CSUM)/d$
                     and then set $CSUM = U_0(q)$
   (b)  Otherwise, if there is no edge $\{i, q\} \in E''$ with $i < q$, then
            (1) if $CSUM + c_q \leq U_0(q)$ set $x_q = 1$ and $CSUM := CSUM + c_q$
            (2) if $CSUM + c_q > U_0(q)$ set $x_q = (U_0(q) - CSUM)/c_q$ and then set
                $CSUM = U_0(q)$
3. If $q = n$, $x_0 = CSUM$ and the method stops. Otherwise:
   (a)  If $CSUM \geq U_0(q+1)$ then $x_0 = CSUM$, $x_i = 0$ for $i > q$ and the method stops.
   (b)  If $CSUM < U_0(q+1)$, then set $q := q + 1$ and return to Step 2.
```

Theorem 6. The LP Conditional Objective Knapsack Solution and the Method for W-SS give optimal solutions for their respective problems.

Proof: The validity of the LP Conditional Objective Knapsack Solution can be verified by observing that if $x_i > 0$ for any $i > q$ in case 2(a), or for any $i > q + 1$ in case 2(b), then x_0 is compelled to be no larger than specified in these cases. Moreover, if x_i is reduced in value for any $i \leq q$ or for $i = q + 1$ in Case 2(b), respectively, then $x_i > 0$ for some $i > q$ or $i > q + 1$, as previously identified. The validity of the Method for W-SS rests on the validity of the LP Conditional Objective Knapsack Solution, and on the fact that each iteration of the method identifies an optimal allocation of values between x_q and x_i in the case where $\{i,q\}$ is an edge of E″. □

As in the solution method for W-SC, the method for W-SS can be applied to evaluate the consequences of setting selected variables $x_r = 1$, by solving associated problems W-SS(r) in parallel. These methods give new ways to determine bounds and choice rules for weighted independent set problems, and can also be applied to other 0-1 problems by disregarding the special rules to handle constraints over the edges of E′ and E″ (in cases where such constraints do not exist).

10. Surrogate Constraint Strategies for Additional Improvement

Returning to basics, we observe as in the surrogate constraint proposals of Glover (1965, 1971) that additional strengthening can be obtained by generating surrogate constraints from inequalities obtained as logical implications of the original constraints. In the present setting, a well-known source of logical implications is provided by clique inequalities (Padberg, 1973). Whenever a set C of nodes constitutes a clique, i.e., each pair of nodes in C is joined by an edge, then clearly at most one of the nodes in C can belong to an independent set. Consequently, we can write

$$\sum (x_i : i \in C) \leq 1.$$

If C contains more than two nodes, the clique inequality is stronger than any of the component inequalities

$$x_i + x_j \leq 1 \quad i \neq j, \quad i, j \in C$$

and can replace all of these inequalities.

The clique inequality is also stronger than a simple surrogate constraint formed without reference to logical implications. To illustrate, consider a clique composed of nodes 1, 2 and 3, whose associated component inequalities are

$$\begin{array}{lllll} x_1 + & x_2 & & \leq & 1 \\ x_1 & & + \ x_3 & \leq & 1 \\ & x_2 + & x_3 & \leq & 1. \end{array}$$

The simple-sum surrogate constraint for this system is

$$2x_1 + 2x_2 + 2x_3 \leq 3$$

and is dominated by the logically derived clique inequality

$$x_1 + x_2 + x_3 \leq 1.$$

Larger cliques provide still greater degrees of strengthening. For example, the component inequalities for a clique on nodes 1, 2, 3 and 4 yield a simple-sum surrogate constraint of

$$3x_1 + 3x_2 + 3x_3 + 3x_4 \leq 6.$$

Evidently, this is only “half as strong” as the clique inequality

$$x_1 + x_2 + x_3 + x_4 \leq 1.$$

These observations show that the generation of surrogate constraints by including reference to logically implied inequalities is particularly relevant in the present setting. As a result of dominating the component two-variable inequalities that serve as their “building blocks,” the clique inequalities are preferable to use in the role of source inequalities for generating surrogate constraints over larger numbers of variables. Such surrogate constraints can then be used to create stronger versions of the sub-problems SC1 and SC2. These versions can be additionally improved by the opportunity to use surrogate constraint normalizations whose weights differ for clique inequalities over differing numbers of variables. More general normalization rules such as those indicated in Section 3.1 become applicable.

Another important consequence derives from identifying clique inequalities. Disjoint subsets of these inequalities can also be extracted to provide stronger sub-problem relaxations than the SC and SS models previously discussed. Using common terminology, a collection of disjoint sums of variables each bounded above by 1 is called a GUB (“generalized upper bound”) system, and thus by means of clique inequalities we find it useful to address more general GUB-constrained versions of the SC and SS problems.

It may be possible that clique inequalities containing more than two variables are few in number or do not exist. However, even where cliques beyond size 2 are rare, their discovery can be valuable. As the previous illustration has suggested, accounting for components of cliques separately rather than considering their combined implications produces a misleading evaluation, and an exaggeration of an accompanying estimate of the size of an independent set. Effort spent to remove this element of exaggeration by identifying and exploiting clique inequalities can therefore be well rewarded.

This raises two primary issues: (1) to identify collections of inequalities that lead to strong instances of GUB-constrained SC and SS models, and (2) to develop algorithms that solve these models highly efficiently. We show that it is possible to handle both of these considerations simultaneously, by a dynamic solution strategy that generates the clique inequalities for the GUB system and solves the associated sub-problems at the same time. In addition, for problems that hide their complexity in structures more general or more subtle than cliques, we observe in Section 12 how these ideas can be extended to *q-complete systems*, which give rise to inequalities that include clique inequalities as a special case.

11. Creating and Exploiting Clique Inequalities Dynamically.

We represent a collection of disjoint cliques that will be generated dynamically by the notation

$$\text{Clique}(p): \; p = 1,\ldots,nc,$$

where the number of cliques nc is determined automatically by the process of generating the cliques themselves. The clique generation method is a constructive procedure embodied within the algorithm for exploiting the GUB system defined by the cliques.

The approaches to be described can also be advantageously used within multi-start methods by varying the choice rules of the dynamic generation process on different passes. Such a repeated application provides an opportunity to create a larger collection of cliques, which in turn can be used to produce stronger surrogate constraints. The solution procedures can also draw upon cliques from the pool already established, selecting new sub-collections of disjoint members from this pool rather than creating new ones. Such a pool can also be created by preprocessing, using variations of the same processes embedded within a multi-start method.

The guidance provided by surrogate constraints in these processes additionally affords a means to supplement (or in some cases replace) the use of separation algorithms designed to identify cliques in branch and cut methods. Such applications can take advantage of dual multipliers produced by solving LP problems within branch and cut methods, giving weights for generating surrogate constraints that can be further enhanced by procedures described in Appendix 2.

The first and simplest problem we address by a dynamic method is the GUB-constrained version of SC, which can be written as follows.

SC:GUB: Maximize $x_o = \sum (x_i : i \in N)$

subject to $\sum (a_i x_i : i \in N) \le a_o$

$\sum (x_i : i \in \text{Clique}(p)) \le 1$ for $p = 1, \ldots, nc$

x_i binary $\quad i \in N$

To describe the method for solving SC:GUB, we say that Clique(p) can be augmented by node i if adding node i to Clique(p) creates a larger clique, i.e., if $\{i, j\} \in E$ for all $j \in \text{Clique}(p)$.[5] The method applies as well to situations where cliques of size greater than 2 may not exist, by assembling good choices for cliques composed of the edges $\{i,j\} \in E$. (There is no requirement that the indicated cliques must include all nodes of N.) The basic method can be summarized in the following simple form.

[5] Checking this condition is facilitated in the following method by keeping a bit matrix M(i,j), where M(i,j) = 1 if $\{i,j\} \in E$ and M(i,j) = 0 otherwise.

Dynamic Method for SC:GUB.

0. Index the variables so that $a_i \le a_{i+1}$, for i, i+1 $\in$ N. Let $x_i = 0$ for i = 0 and i $\in$ N , ASUM = 0 and i = 1.
1. If ASUM + $a_i > a_o$, stop, the current solution is optimal.
2. If some existing Clique(p), $1 \le p \le nc$, can remain a clique by adding node i, choose such a clique and add i to Clique(p).
3. Otherwise, if no clique exits that can be augmented by node i, create a new clique to contain node i: Set nc := nc + 1, Clique(nc) = {i}, $x_i = 1$, $x_o := x_o + 1$, ASUM := ASUM + a_i.
4. If i = n, stop. Otherwise set i := i + 1 and return to step 1.

In the preceding method, all variables start 0, and a variable x_i is set to 1 if node i is the first node to become an element of a clique that is newly created. For other nodes i subsequently added to the clique, the variables x_i retain their initial 0 value. Once no more variables can be set to 1 without violating the surrogate constraint inequality, the method stops. It is possible at the end that one or more of the Clique(p) sets are *extraneous*, in the sense that they contain a single node i, whose corresponding clique inequality $x_i \le 1$ is redundant. (The associated variables are nevertheless relevant in that they are assigned values of 1 in the solution process.)

Various rules can be used in Step 2 for choosing the particular Clique(p) to augment by node i, when more than one choice exists. A simple rule is to examine the cliques either in ascending or descending order of the indexes p (i.e., in FIFO or LIFO order), and to choose the first (respectively, smallest or largest) p for which Clique(p) can be successfully augmented. At the same level of simplicity, the method can alternate between FIFO and LIFO choices, and a variety of other rules are evidently possible.

Since the foregoing method is very fast, and solves the SC:GUB problem at the same time that it generates it, an opportunity arises to implement more than one choice rule in Step 2 and then select the best resulting bound for x_o.

In addition, the method can be accelerated by considering Clique(p) as a candidate to be augmented only if it does not exceed a specified size s. Restricting s to 2 or 3 (i.e., restricting the largest clique size to 3 or 4), may appreciably speed the method, although the outcome depends on the problem instance. Tradeoffs in the quality of the outcomes produced by restricting clique size enter into consideration, since larger cliques can more strongly constrain the set of feasible solutions. An approach that keeps track of the sizes of cliques produced during successive passes of a multi-start method, accounting for the fact that clique sizes will drop as larger numbers of variables are assigned values, can provide reasonable choices for s on later passes as a result of information derived from earlier passes.

11. 2 Solving the GUB-Constrained SS Problem.

In a manner analogous to the formulation of the SC:GUB problem, we may formulate a GUB-constrained version of the SS problem as follows.

SS:GUB: Maximize $x_o = \sum (x_i : i \in N)$

subject to $\sum(x_i: i \in N) \le U_o(j)x_j + |N|(1-x_j) \quad j \in N$

$\sum (x_i : i \in \text{Clique}(p)) \le 1$ for p = 1, …,nc

x_i binary $\quad i \in N$

As in the case of the SS problem, the inequalities over $j \in N$ above can also be replaced by the nonlinear inequality

$$\Sigma(x_i: i \in N) \leq \text{Min}\ (U_o(i): x_i = 1, i \in N).$$

By direct extension of Theorem 4, the method for SS:GUB acquires a structure similar to that of the SC:GUB method, and can be expressed even slightly more succinctly.

Dynamic Method for SS:GUB.

```
0. Index the variables so that $U_o(i) \geq U_o(i+1)$, for $i, i+1 \in N$.  Let $x_i = 0$ for $i = 0$ and
   $i \in N$, and $i = 1$.
1. If $x_o > U_o(i)$, stop, the current solution is optimal.
2. If some existing Clique(p), $1 \leq p \leq nc$, can remain a clique by adding node i, choose
   such a clique and add i to Clique(p).
3. Otherwise, if no clique exits that can be augmented by node i, create a new clique to
   contain node i:  Set $nc := nc + 1$, Clique(nc) = $\{i\}$, $x_i = 1$, $x_o := x_o + 1$.
4. If $i = n$, stop.  Otherwise set $i := i + 1$ and return to step 1.
```

The previous comments about the choices in Step 2 of the SC:GUB method apply to the SS:GUB method as well.

11. 3 The Weighted SS:GUB Problem.

The GUB-constrained version of the weighted SS problem, which we denote by W-SS:GUB, can be written

W-SS:GUB: Maximize $x_o = \Sigma\ (c_i x_i : i \in N)$

$$\begin{aligned} \text{subject to}\quad & \Sigma(c_i x_i: i \in N) \leq U_o(j)x_j + U_o(1-x_j) \quad j \in N \\ & \Sigma\ (x_i : i \in \text{Clique}(p)) \leq 1 \quad \text{for } p = 1, \ldots, nc \\ & x_i \text{ binary} \qquad i \in N \end{aligned}$$

where U_o is an upper bound on x_o. This problem requires a somewhat more complex method than the SS:GUB problem, just as the W-SS problem requires a more complex method than the SS problem. However, the earlier W-SS method extends smoothly to the GUB case. The analysis of Theorem 6 discloses that the primary change required is to keep track of a special node for each Clique(p), which we denote by MaxWtNode(p), which is the node in Clique(p) with the largest c_i (weight) value. We identify how this is done as follows.

Method for W-SS:GUB

```
1. Index the variables so $U_0(1) \geq U_0(2) \geq \ldots \geq U_0(n)$. Start with q = 1 and CSUM = 0.
2. If some existing Clique(p), $1 \leq p \leq nc$, can be augmented by node q, choose such a
   clique and set i = MaxWtNode(p) (for MaxWtNode(p) as previously determined in
   Step 2(3) or 3(1)).  Then
      (1) add q to Clique(p).
      (2) if $c_i \geq c_q$, set $x_q = 0$
      (3) if $c_i < c_q$ set MaxWtNode(p) = q, and let $d = c_q - c_i$. Then
            (i)  if $CSUM + d \leq U_0(q)$ set $x_q = 1$, $x_i = 0$ and $CSUM := CSUM + d$
            (ii) if $CSUM + d > U_0(q)$ set $x_q = f$ and $x_i = 1 - f$ for $f = (U_0(q) - CSUM)/d$
                 and then set $CSUM = U_0(q)$
```

3. Otherwise, if no existing clique can be augmented by node q, then
   (1) create a new clique to contain node q: Set nc := nc + 1, Clique(nc) = {q}, and MaxWtNode(nc) = q
   (2) if CSUM + $c_q \le U_0(q)$ set $x_q = 1$ and CSUM : = CSUM + c_q
   (3) if CSUM + $c_q > U_0(q)$ set $x_q = (U_0(q) - \text{CSUM})/c_q$ and then set CSUM = $U_0(q)$
4. If q = n, x_0 = CSUM and the method stops. Otherwise:
   (1) If CSUM $\ge U_0(q+1)$ then x_0 = CSUM, $x_i = 0$ for i > q and the method stops.
   (2) If CSUM $< U_0(q+1)$, then set q: = q + 1 and return to Step 2.

The validity of this method derives by logical extension of the proof of Theorem 6. As in the case of the simpler W-SS problem, the approach is directly applicable to solving weighted independent set problems, and can be adapted by a change of variables to give a method for solving the associated covering problems.

11.4 The Weighted SC:GUB Problem.

We have inverted the order of considering the weighted versions of the SS:GUB problem and the SC:GUB problem, because the latter problem requires a somewhat more intricate method to handle. We can write the problem to be addressed as follows.

W-SC:GUB: Maximize $x_o = \sum (c_i x_i : i \in N)$
subject to $\sum (a_i x_i : i \in N) \le a_o$
$\sum (x_i : i \in \text{Clique}(p)) \le 1$ for p = 1, …,nc
x_i binary $i \in N$

To solve this problem, we proceed as in the case of the W-SC problem by transforming it into a new problem, which in the present instance we denote as T-SC ("T" for "transformed"). The new problem has the same form as W-SC:GUB, except that we change the values of some of the c_i and a_i coefficients, dropping some of the variables and removing the GUB constraints. Thus the problem acquires the form of an ordinary knapsack problem, and we can solve its continuous version by the LP Knapsack Solution rule.

Let Clique(p:q) be defined to be the subset of Clique(p) given by

Clique(p:q) = $\{i \in \text{Clique}(p): i > q\}$.

The following transformation reindexes the elements of Clique(p), possibly more than once for some elements, and on each occasion Clique(p:q) is assumed to be defined relative to the current indexing. As variables are dropped, their indexes are removed from Clique(p) and Clique(p:q), and hence also removed from N, leaving a residual N we denote by N′.

Transforming W-SC:GUB into T-SC.

0. Begin with N′ = N, and record the original c_i and a_i values as c_i^o and a_i^o. Perform the following steps for each Clique(p), $1 \le p \le nc$, beginning the execution for each p by setting q = 0.
1. For all $i,j \in$ Clique(p:q), index the variables so that $c_i/a_i \ge c_j/a_j$, and if the ratios are equal, then $a_i \ge a_j$.
2. Let i = Min($h \in$ Clique(p:q)), and for all $j \in$ Clique(p:i) such that either $c_i^o \ge c_j^o$ or $a_i^o \ge a_j^o$, drop j from N′ (and hence from Clique(p) and Clique(p:i)).

3. If Clique(p:i) is now empty, the transformation for Clique(p) is complete. Otherwise, replace x_i by a new variable z_i defined by $z_i = x_i + \sum (x_j : j \in \text{Clique}(p{:}i))$. Then z_i receives the coefficients c_i and a_i associated with x_i, and each x_j, $j \in$ Clique(p:i), receives new coefficients given by

   $c_j := c_j^o - c_i^o$

   $a_j := a_j^o - a_i^o$

   Let $q = i$ and return to Step 1.
4. Once the transformation is completed for every Clique(p), the final form of T-SC is defined by reference to the new z_i variables and the x_i variables that were not eliminated, where each variable receives the final c_i and a_i coefficients identified for that variable.

Theorem 7. An optimal LP solution to W-SC:GUB is obtained by applying the LP Knapsack Solution Rule to the transformed problem T-SC (treating the z_i variables in the same way as the x_i variables). Optimal solution values for the original x_i variables are recovered by the rule:

(1) $x_i = z_i - z_j$ for each $i,j \in$ Clique(p) such that z_j was created immediately following the creation of z_i, and

(2) $x_i = z_i - x_j$ if z_i was the last z variable created for Clique(p), and x_j was the single remaining variable not eliminated.

All x_i variables not transformed into z variables retain their assigned values.

Proof: Follows by inductive application of the reasoning of the proof of Theorem 5. □

We now describe a method to take advantage of this theorem as the GUB sets are generated dynamically. To facilitate the description, we introduce some additional arrays and make reference to a standard linked list Next(i) to order the variables. The linked list, which orders the variables by starting with i = FirstIndex and visiting all $i \in N$ by repeatedly setting $i :=$ Next(i), is constructed by the rule of defining Next(i) = h, for all $i,h \in N$, so that[6]

$c_i/a_i \geq c_h/a_h$ and $a_i \geq a_h$ if the ratios are equal.

As a clique is being dynamically constructed (or as its elements are being visited) in the sequence determined by the following algorithm, we keep track of the index i = Current(p) for each Clique(p) which identifies the "current x_i" for that clique. We also keep track of "Updated" nodes in the clique, by setting Update(j) = i, if x_j has been processed by updating its coefficients according to the rule $c_j = c_j^o - c_i^o$ and $a_j = a_j^o - a_i^o$. The update is not required for each x_j for $j \in$ Clique(p), because we can defer the update until x_j is itself a candidate to be selected as x_i.

Finally, we require an array CliqueID(j) to keep track of which clique a node j belongs to, setting CliqueID(j) = p if node j belongs to Clique(p), and setting CliqueID(j) = 0 if node j does not belong to a clique.

[6] Because the GUB-Constrained Knapsack method changes some problem coefficients and thus changes the linking by Next(i), an accompanying reverse order linked list is useful to execute such changes.

Dynamic GUB-Constrained Knapsack Method.

0. To begin, set nc = 0, and for all $j \in N$, set: $x_j = 0$, Update(j) = 0, CliqueID(j) = 0. Set ASUM = 0 and j = FirstIndex. Notationally, denote the last $i \in N$ visited by the Next(i) ordering as LastIndex.

1. Let p = CliqueID(j). If p = 0, proceed to Step 2, and otherwise go to Step 3.
2. (Node j does not yet belong to a clique.) Check if node j can augment an existing clique:
   - (a) Node j does not enlarge an existing clique (or no cliques exist): Create a new clique to contain node j: Set nc := nc + 1, Clique(nc) = {j}, CliqueID{j} = nc, Current(nc) = j (x_j is treated as the current x_i of the T-SC transformation) and Update(j) = j (x_j is self-updated, and does not change its coefficients). Set p = nc and proceed to Step 3.
   - (b) Node j augments an existing clique: Select such a Clique(p), add j to Clique(p), and set CliqueID(j) = p.
3. (Node j belongs to the existing Clique(p).) Let i = Current(p) (identifying the node i of Clique(p) that qualifies as the current x_i of the T-SC transformation). Check if Update(j) = i:
   - (a) Update(j) = i (x_j has been updated by x_i). Assign x_j its current value:
     - (1) If ASUM + $a_j > a_o$, then
       - Set $x_j = (\text{ASUM} - a_o)/a_j$ and ASUM = a_o.

       Otherwise:
       - Set $x_j = 1$ and ASUM := ASUM + a_j.
     - (2) If $j \neq i$ (x_j has not yet taken the role of x_i), then:
       - Set $x_i := x_i - x_j$ and Current(p) = j (hence x_j becomes the new x_i for Clique(p)).
     - (3) If ASUM = a_o or if j = LastIndex, the solution is complete. Otherwise, set j := Next(j) and go to Step 1.
   - (b) Update(j) $\neq$ i (x_j has not been updated by the current x_i):
     - (1) If $c_j^o \le c_i^o$ or $a_j^o \le a_i^o$, then drop node j; i.e., x_j retains its preassigned 0 value. If j = LastIndex, stop. Otherwise, set j := Next(j) and return to Step 1.
     - (2) If $c_j^o > c_i^o$ and $a_j^o > a_i^o$, then: Set $c_j = c_j^o - c_i^o$, $a_j = a_j^o - a_i^o$ and Update(j) = i. If j = LastIndex, stop. Otherwise, set PossibleNext = next(j): Reinsert j in its proper position on the linked list, based on its updated coefficients. If j does not change its position, i.e., is still the first index on the list remaining to be examined, go to Step 3. But if j is no longer in the first position, set j = PossibleNext (the node now in the first position), and go to Step 1.

11.5 Generalized Results for Exploiting Conditionally Shared Limitations

We now provide results that link the first and second principles of conditionally shared limitations, and allow them to be applied to more general contexts. We start by giving a general problem formulation and identifying three related problems, followed by the main theorem that gives bounds for the original problem by reference to the others. We also describe a method for implementing the theorem and provide an example to demonstrate how the method can be applied.

We denote the problem we are interested in solving by

Problem P: Maximize $x_o = f(x)$
subject to $x \in X$

where $x \in X$ implies $x_i = 0$ or 1 for $i \in I \subseteq N$.[7] Associated with Problem P, we identify the relaxed problem PR given by:

Problem PR: Maximize $x_o = g(x)$
subject to $x \in X^o$

The set X^o is a superset of X and the function g(x) is an overestimator of f(x), i.e., $\text{Max}(g(x): x \in X^o) \geq \text{Max}(f(x): x \in X)$. Define $U_o(i)$ for $i \in I$ to be a conditional upper bound for x_o in Problem P when $x_i = 1$, i.e., $U_o(i)$ is a value that satisfies the inequality

$$U_o(i) \geq \text{Max}(f(x): x \in X, \text{ and } x_i = 1).$$

Then we identify the special relaxation PR* of P given by:

Problem PR*: Maximize $x_o = g(x)$
subject to $x \in X^o$
$x_o \leq \text{Min}(U_o(i): x_i > 0, i \in I)$

We observe that Problem PR* is a generalization of the problems SS and W-SS, including their GUB-constrained versions.

Following our previous conventions, we let $x_o(P)$, $x_o(PR)$, etc., denote the optimum x_o value for the corresponding problem. (We interpret x_o to be $-\infty$ if the associated problem has no feasible solution.) Given knowledge of a bound U_o on x_o in Problem P as a result of solving PR (i.e., $U_o = x_o(PR)$), we can always assume $U_o(i) \leq U_o$. For example, U_o can initially be obtained by solving PR (to give $U_o = x_o(PR)$), accompanied by solving instances of PR in which selected x_i variables are given the assignment $x_i = 1$ to obtain associated values $U_o(i) \leq U_o$, and we may take $U_o(i) = U_o$ for the remaining x_i variables for $i \in I$.

It is clear that PR* is a valid relaxation of P. Moreover, PR* is a stronger relaxation than PR if $U_o(i) < U_o$ for at least one x_i, $i \in I$, such that $x_i > 0$ in an optimal solution to PR. Consequently, we are interested in solving PR* to obtain a tighter bound U_o, and also in solving versions of PR* subject to $x_i = 1$for selected variables x_i, to obtain tighter bounds $U_o(i)$ than would be possible by solving PR. Since these latter "constrained versions" of PR* have the same form as PR* upon making the assignment $x_i = 1$ and eliminating x_i, we state all our results in terms of PR*.

[7] The set I, which identifies the integer variables, is different from the set I of Section 7.

Let U be the set of the distinct values taken by the bounds $U_o(i)$, i.e., $U = \{u: u = U_o(i), i \in I\}$, and define $I[u] = \{ i \in I: U_o(i) \leq u\}$. Then we identify a family of problems over $u \in U$

$$\textbf{Problem PR}[u]: \quad \begin{array}{ll} \text{Maximize } x_o = g(x) & \\ \text{subject to} \quad x \in X^o & \\ \qquad\qquad\quad x_i = 0, \quad i \in I[u] & \end{array}$$

Our main result, which generalizes and strengthens Theorems 1 and 2 of Section 7, demonstrates that solving PR[u] for any selected u = u* either gives an optimal solution to PR*, or else bounds $x_o(PR^*)$ and allows U to be reduced in a manner that allows the best bound on $x_o(PR^*)$ (and hence on $x_o(P)$) to be found by a "better than binary" search.[8]

Theorem 8. Let u* be any element of U, and let x* denote an optimal solution to PR[u*], with $x_o^* = x_o(PR[u^*])$. Then x* either solves PR* or the pair (u^*, x_o^*) bounds $x_o(PR^*)$ as follows.

Case 1. If $u^* \leq x_o^*$, then:

(1) $x_o(PR^*) \leq x_o^*$.

(2) Let $u^+ = \text{Min}(U_o(i): x_i^* > 0)$ (where $u^+ = \infty$ if all $x_i^* = 0$).

(a) If $u^+ \geq x_o^*$, then x* is optimal for PR* and $x_o(PR^*) = x_o^*$.

(b) If $u^+ < x_o^*$, then a stronger bound on $x_o(PR^*)$ than x_o^* exists by solving PR[u] only if u lies in the range $u^+ \leq u < x_o^*$.

Case 2. If $u^* > x_o^*$, then:

(3) $x_o(PR^*) \leq u^*$.

(4) Let $u' = \text{Max}(u \in U: u \leq x_o^*)$. If a stronger bound on $x_o(PR^*)$ than u* exists by solving PR[u], then such a bound can be found for u in the range $u' \leq u < u^*$.

Proof: Suppose $x_o(PR^*) > x_o^*$. Then since $x_i^* = 0$ for all $U_o(i) \leq u^*$ by the formulation of PR[u*], we must have $x_i > 0$ for some $U_o(i) \leq u^*$ in an optimal solution to PR*, which implies $x_o(PR^*) \leq u^*$. This establishes $x_o(PR^*) \leq \text{Max}(u^*, x_o^*)$, which implies both (1) and (3). Next, to establish 2(a), $u^+ \geq x_o^*$ implies x* is feasible for PR*, since the constraint $x_o \leq \text{Min}(U_0(i): x_i > 0)$ is satisfied by x* and x_o^*. If x* is not optimal for PR*, i.e., if $x_o(PR^*) > x_o^*$, then an optimal solution x to PR* must satisfy $x_i > 0$ for some $i \in I[u^*]$ (since otherwise the PR[u*] requirement that $x_i = 0$ for $U_o(i) \leq u^*$ is compatible with optimality of x* for PR*). But then PR* requires $x_o \leq u^*$ for feasibility, hence $x_o(PR^*) \leq x_o^*$, a contradiction. To establish 2(b), the previous justification of (1) and (3) shows that $\text{Max}(u, x_o(PR[u])) < x_o^*$ must hold if solving PR[u] gives a stronger bound than x_o^*. Hence $u < x_o^*$, and in addition, $x_o(PR[u]) < x_o^*$ implies $u > u^*$. For u in the interval $u^* < u < u^+$, we have $x_o(PR[u]) = x_o^*$, because x* remains feasible for PR[u] through this interval and PR[u] is more constrained than PR[u*]. Thus, $u^+ \leq u < x_o^*$. It remains only to establish (4). To obtain a tighter bound than u* on $x_o(PR^*)$ we must have $u^* > \text{Max}(u, x_o(PR[u]))$, hence $u < u^*$, which implies in turn that $x_o(PR[u]) \geq x_o(PR[u^*]) = x_o^*$. Since $x_o^* < u^*$ in Case 2, w also have $x_o(PR[u]) \geq x_o^*$, hence $x_o(PR[u]) \geq u$, for all $u \leq x_o^*$. Thus

[8] To increase the usefulness of this result for the general case, the variables x_i of the original problem can be complemented where appropriate to assure that $U_o(i)$ is the tighter of the two upper bounds for x_o when x_i is alternately assigned the values 0 and 1. The result further applies to general mixed integer programming problems by allowing "$x_i = 0$ or 1" to be a shorthand notation for "$x_i \leq v$ or $x_i \geq v + 1$," for some integer variable x_i and a chosen integer value v.

for $u \le x_o^*$ the strongest bound results for $u = u' = \mathrm{Max}(u \in U: u \le x_o^*)$, which confirms the conclusion of (4). □

The usefulness of the conditions of Theorem 8 for finding a best pair (u^*, x_o^*) for bounding $x_o(PR^*)$ can be seen by noting that only part of these conditions, the inequalities $u < x_o^*$ in 2(b) and $u < u^*$ in (4), are sufficient to allow such a pair to be found by a binary search over elements of U. The remaining inequalities in 2(b) and (4), as well as those in 2(a) for identifying optimality, additionally constrain the search.

In many applications, particularly graph problems such as the independent set problem, the relevant values of u will be relatively small in number. For example, values beyond a certain size are not likely to be useful for the goal of restricting $x_o(P)$, and hence can be disregarded (set to U_o) in the formulation of PR*. Consequently, the set U will often contain only a fairly small number of elements, and a search guided by Theorem 8 can quickly identify the best one.

To conveniently describe a method for implementing Theorem 8, we enlarge U to contain one additional element $u_o = -\infty$, and index the elements of U so that $U = \{u_o, u_1, u_2, \ldots, u_k\}$, where $u_o < u_1 < u_2 < \ldots < u_k$.[9] We observe that $I[u_o]$ is empty and hence $PR[u_o]$ is the problem PR. Also, $u_o \le x_o(PR[u_o]) = x_o(PR)$, and hence Theorem 8 remains valid for this enlarged form of U.

It is also useful to create an additional element $u_{k+1} = \infty$, although u_{k+1} is not added to U. This convention obviates the need to include special qualifications in the descriptions of results and methods that follow. We first state a Corollary of Theorem 8, which is a direct generalization of Theorems 1 and 2.

Corollary. For all $u^* \in U$, and for x^* and x_o^* $(= x_o(PR[u^*]))$ as in Theorem 8:

$$x_o(P) \le x_o(PR^*) \le \mathrm{Max}(u^*, x_o^*)$$

Further, identify the index h such that $u_h = u^*$. Then x^* solves PR* if $x_o^* \le u_{h+1}$ (or if $x_o^* = u_h$, as a special case). Finally, let $p = \mathrm{Max}(j: 0 \le j \le k: u_j \le x_o(PR[u_j]))$. Then the best bound by Theorem 8 is given by

$$x_o(PR^*) \le \mathrm{Min}(x_o(PR[u_p]), u_{p+1}).$$

Proof: The assertions follow at once from Theorem 8. □

We now extract additional elements of Theorem 8 to provide a method that efficiently identifies the strongest bound on $x_o(PR^*)$. Let $x_o^\#$ denote the best candidate value for $x_o(PR[u_p])$, and $u^\#$ denote the best candidate value for u_{p+1}.

Fast Determination of the Bound on $x_o(PR^*)$

```
0. Begin with $x_o^\# = u^\# = \infty$, kMin = 0, kMax = k.
1. Select h: kMin $\le$ h $\le$ kMax, and let $u^* = u_h$.
2. Solve $PR[u^*]$, and denote the solution by $x^*$, with $x_o^* = x_o(PR[u^*])$.
    (1) If $u^* \le x_o^*$, then: Let $x_o^\# = x_o^*$ and $u_j = \mathrm{Min}(U_o(i): x_i^* > 0)$ (or $j = k+1$, hence
        $u_j = \infty$, if all $x_i^* = 0$).
        (a) If $u_j \ge x_o^*$, then $x_o(PR) = x_o^*$ and the search ends.
        (b) If $u_j < x_o^*$, then set kMin = j and kMax := $\mathrm{Max}(q \le \text{kMax}: u_q < x_o^*)$.
```

[9] By this organization, the set $I[u_j]$ for $j > 0$ constitutes a generalized form of the set I_j of Section 7.

(2) If $u^* > x_o^*$, then: let $u^\# = u^*$ and if $u_{kMin} \le x_o^*$, set
kMin = Max($q \le$ kMax: $u_q \le x_o^*$). Set kMax := h – 1.

3. If kMin > kMax, the search ends and $x_o(PR^*) \le \text{Min}(x_o^\#, u^\#)$.
Otherwise, return to Step 1.

The condition "kMin > kMax" of Step 3 is checked each time kMin or kMax changes in Step 2, and the search ends as indicated if the condition holds.

A Numerical Example.

To illustrate the foregoing method, we consider a problem that includes the constraints of both the SC and SS problems of Section 7. Instead of being compelled to solve these problems separately, to obtain a (possibly different) bound on x_o from each, we are able to combine the two problems into a single new problem by taking advantage of Theorem 8. SC takes the role of the relaxed problem PR, and the associated problem PR* thereby combines SC and SS. The formulation of PR*, where in this instance f(x) = g(x) and I = N, is:

$$\begin{array}{lll} \textbf{PR*:} \text{ Maximize } x_o = & \sum (x_i : i \in N) & \\ \text{subject to} & \sum (a_i x_i : i \in N) \le a_o & \\ & x_i + x_j \le 1 & \{i,j\} \in E' \\ & x_i \text{ binary} & i \in N \\ & \sum(x_i : i \in N) \le \text{Min}\,(U_o(i): x_i > 0,\ i \in N).^{10} & \end{array}$$

The following two tables illustrate the solution of the problems PR[u] for various values of u, as a means of bounding $x_o(PR^*)$. Table 1 gives the numerical data for a 16 variable problem, where the variables are indexed for convenience to that the edges $\{i,j\} \in E'$ (extracted as a set of node-disjoint edges from E) are {1,2}, {3,4}, {5,6}, etc.

Table 2 gives the solution values for the x_i variables for each problem PR[u], where the values of u range over U = {- ∞, 2,3,4,5,6,7,8} (corresponding to the different $U_o(i)$ values identified in the first table, together with $u_o = -\infty$). Variables that are forced to 0, by the restriction $x_i = 0$ for $i \in I[u]$, are shown by an "X entry" in Table 2. Zero-one solution values for remaining variables are as indicated. The variables are pre-indexed in an order that makes it easy to see the solution to SC at each stage (obtained by successively setting $x_i = 1$ in the order from smaller to larger a_i values, allowing at most one assignment $x_i = 1$ for each edge, until no more assignments can be made subject to $\sum a_i x_i \le a_o = 72$).[11]

Table 1. Problem Data

i	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	
a_i	3	12	7	11	12	16	13	15	17	30	18	29	31	34	32	33	≤72
$U_0(i)$	3	4	3	3	4	8	2	3	5	7	3	6	7	4	3	7	

[10] This constraint is equivalent to the constraint $\sum(x_i: i \in N) \le \text{Min}\,(U_o(i): x_i = 1, i \in N)$ of Problem SS, since x is a binary vector in the SC formulation. As noted before, if a bound on setting $x_i = 0$ is more limiting than $U_o(i)$ for setting $x_i = 1$, the variable is simply complemented to define $U_o(i)$ relative to $y_i = 1 – x_i$, since $y_i = 1$ when $x_i = 0$.

[11] The $U_o(i)$ values of Table 1 may be assumed to come from setting $x_i = 1$ in an original problem P (not shown), followed by generating and solving an associated PR problem. Or less stringently, they can come from setting $x_i = 1$ in PR and solving the resulting problem. Similarly, the current PR* problem itself may come from setting $x_r = 1$ in P or PR for some x_r (e.g., x_{17}) to produce the data in Table 1, for the goal of obtaining a stronger bound $U_o(r)$.

Table 2. Solution Values (x_i) for PR[u]

u	x_0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	$\Sigma a_i x_i$
$-\infty$	6	1	0	1	0	1	0	1	0	1	0	1	0	0	0	0	0	70
2	6	1	0	1	0	1	0	X	1	1	0	1	0	0	0	0	0	72
3	4	X	1	X	X	1	0	X	X	1	0	X	1	0	0	X	0	70
4	3	X	X	X	X	X	1	X	X	1	0	X	1	0	X	X	0	62
5	2	X	X	X	X	X	1	X	X	X	0	X	1	0	X	X	0	45
6	2	X	X	X	X	X	1	X	X	X	1	X	X	1	X	X	0	61
7	1	X	X	X	X	X	1	X	X	X	X	X	X	X	X	X	X	16
8	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0

Proceeding down the rows of Table 2, the progression by which x_o decreases as u increases becomes evident. If the solution method first selects u = 4, then the method reduces kMax and increases kMin so that the only remaining value of u to examine is u = 3, and the solution to PR[3] immediately confirms that $x_o(PR^*) = 4$. If u = 3 is chosen first, the method at once verifies that the associated solution is optimal.

If instead of first picking a "middle" u value, the solutions are generated by starting from $u = -\infty$, and successively increasing the value of u, then the value of kMax progressively drops as u and kMin increase. Proceeding in the opposite direction, the value of kMin increases as u and kMax decrease.

If the SC and SS problems are solved independently of each other, then $x_o(SC) = 6$ and $x_o(SS) = 5$, and hence the bound $x_o(PR^*) = 4$ obtained above proves to be better than available from these component problems (as may be expected). In addition, the foregoing approach can be applied in a stronger way by allowing the set E′ to be regenerated from the original problem to exploit the knowledge that x_i is forced to 0 for $U_o(i) \le u$. Such a strategy is facilitated by the fact that all variables x_i compelled to be 0 for $u = u_{kMin}$ are 0 for all larger values of u, and since kMin progressively grows, these x_i are also compelled to be 0 on all subsequent iterations. The effort of applying the approach can also be reduced by only regenerating E′ for the values of u that give the strongest bounds when the regeneration process is not applied.

To show how the strengthening can occur, suppose we undertake to re-generate (a part of) E′ in order to re-evaluate the solution for u = 3, which is the value that previously determined the strongest bound for $x_o(PR^*)$. Also assume the edge set E contains the edge {2,5}. Applying the rule in Section 7 for choosing E′, we introduce this edge into E′ for u = 3 in place of edge {5,6}. (The rule makes this choice since the edge {1,2} no longer exists when u = 3, as a result of compelling $x_1 = 0$.) Then the inequality $x_2 + x_5 \le 1$ yields a solution with $x_o(PR[3]) = 3$, which the method at once confirms to be optimal, and hence which tightens the bound given by $x_o(PR^*)$ from 4 to 3.

12. Q-Complete Systems and Associated Inequalities

We undertake to show that the ideas of the preceding sections can readily be adapted to exploit more general structures, giving rise to inequalities that can either be used to generate surrogate constraints or that can be embedded in a dynamic solution process.

12. 1 Inequalities for q-Complete Systems.

Consider a system of inequalities

$$\textbf{(V:m)} \quad \sum (x_i: i \in Q) \leq m, \qquad Q \in S(V)$$

where S(V) is a specified collection of subsets of the set V. We say the system (V:m) is *q-complete* (relative to S(V)) if S(V) consists of all q-element subsets of V; i.e., if $S(V) = \{Q \subseteq V: |Q| = q\}$. We assume the inequality is non-trivial, i.e., $m \geq 1$, and also assume $q \geq m + 1$. This latter inequality represents a non-redundancy condition in the case where the variables are 0-1.

For independent set problems, if $E(C) \subseteq E$ is the set of edges over a clique C, then the collection of inequalities

$$x_i + x_j \leq 1, \quad \{i,j\} \in E(C)$$

constitutes a 2-complete system. (That is, the edges {i,j} constitute the sets Q, and C takes the role of V.)

The fact that this system implies the clique inequality

$$\sum(x_i: i \in C) \leq 1$$

is a manifestation of a simple but useful result that applies more generally to any q-complete system of inequalities for which the variables are non-negative and integer-valued.

To see how this occurs, we first note that if (V:m) is q-complete and $|V| = q+1$, then $\sum(x_i: i \in V) \leq m$. This inequality results by creating a simple-sum surrogate from the $q + 1$ inequalities $\sum (x_i: i \in Q) \leq m$ of (V:m) as Q ranges over q-element subsets of V. Each variable x_i appears in q of these $q + 1$ inequalities, and so the simple-sum surrogate constraint is given by $q(\sum(x_i: i \in V)) \leq (q + 1)m$, or equivalently $(\sum(x_i: i \in V)) \leq ((q + 1)/q)m$. The assumption $q \geq m +1$ implies the right hand side is less than $m + 1$, and thus the inequality $\sum(x_i: i \in V) \leq m$ follows.

The same result holds more generally without restricting the size of V.

Remark 1. If (V:m) is q-complete (for any V such that $|V| \geq q$), then $\sum(x_i: i \in V) \leq m$.

The remark may be justified by our preceding observations as follows. Since the inequality holds when $|V| = q + 1$ (and trivially when $|V| = q$), it suffices when $|V| > q + 1$ to create the inequalities $\sum(x_i: i \in Q') \leq m$ for all $q + 1$ element supersets Q' of Q in V. Then, letting Q range over these sets Q' establishes that (V:m) is "q + 1 complete" and the process is repeated for $q := q + 1$, continuing until $q + 1 = |V|$. (As in the case of clique inequalities, the inequality of Remark 1 dominates the separate inequalities of (V:m) and hence can replace them.)

Related (but slightly different) results also hold for "≥" inequality systems, as represented by

$$\textbf{(V: }\boldsymbol{\geq}\textbf{m)} \quad \sum(x_i: i \in Q) \geq m \quad Q \in S(V)$$

(For notational symmetry, the system (V:m) may alternately be represented by (V: ≤m).) Corresponding to our previous definition, we likewise define the system (V: ≥m) to be q-complete (relative to S(V)) if S(V) consists of all q-element subsets of V. We continue to assume $m \geq 1$ and $q \geq m + 1$. (The latter represents an additional non-triviality assumption in the case of 0-1 problems, since $q < m$ is infeasible and $q = m$ forces $x_i = 1$ for all $i \in Q$.)

As a variation on our earlier observation, we cannote in the present case that if (V: ≥m) is q-complete and $|V| = q+1$, then $\sum(x_i: i \in V) \geq m + 1$. The inequality again results by creating a simple-sum surrogate constraint, this time from the $q + 1$ inequalities of (V: ≥m), which gives $(\sum(x_i: i \in V)) \geq ((q + 1)/q)m$. The assumptions $m \geq 1$ and $q \geq m + 1$ insure that the right hand side lies strictly between m and m + 1, producing $\sum(x_i: i \in V) \geq m + 1$. But the result again is more general, extending to any V (such that $|V| \geq q$) and we can state

Remark 2. If (V: ≥m) is q-complete, then $\sum(x_i: i \in V) \geq m + (|V| - q)$.

Remark 2 follows from the same reasoning that establishes Remark 1, noting that the right hand side grows by 1 each time q is incremented by setting q := q + 1.

For the "≥ case" considered here, removing the limitation $q \geq m + 1$ (which is relevant for 0-1 problems), permits the inequality derived from the simple-sum surrogate constraint to be stronger. The derivation of a succession of such inequalities when $|V| > q + 1$ (by considering successively larger sets within V) similarly produces a stronger inequality than the one given in Remark 2. Nevertheless, even by retaining the limitation $q \geq m + 1$ it is clear that in the 0-1 case the inequality of Remark 2 dominates the inequalities of the system (V: ≥m) and can replace them. Without requiring $q \geq m + 1$, but retaining the assumption $q \geq 2$ (previously implicit), the stronger inequality that can result by the simple-sum surrogate constraint likewise dominates the inequalities of (V: ≥m).

12. 2 Weak q-Complete Systems.

We generalize the previous observations for the "≤ case" to increase their applicability. Corresponding results can be stated for the "≥ case," by applying arguments related to those that follow, employing the pattern of development in Section 12.1. (In the situation where the variables are 0-1, associated results for the "≥ case"can be obtained by the device of complementing the variables.)

Consider the system

[V:m] $\sum (x_i: i \in Q \cup P(Q)) \leq m, \quad Q \in S(V)$

where each P(Q), whose identity may vary according to the identity of Q, can be any subset of N – Q, and where, as before, S(V) is a specified collection of subsets of the set V. We say [V:m] is a *weak q-complete system* (relative to S(V)) if the removal of each set P(Q) would make [V:m] a q-complete system. In other words, we stipulate once again that S(V) consists of all q-element subsets of V; hence $S(V) = \{Q \subseteq V: |Q| = q\}$. Also, as before, we assume $m \geq 1$ and $q \geq m + 1$.

We observe at once that the conclusion of Remark 1 holds if [V:m] replaces (V:m), i.e., the inequality $\sum(x_i: i \in V) \leq m$ is still valid, since $\sum (x_i: i \in Q \cup P(Q)) \leq m$ implies $\sum (x_i: i \in Q) \leq m$. This observation, although entirely apparent, is useful for making it possible to derive new inequalities by Remark 4, without requiring that all "component inequalities" take precisely the form of those in a q-complete system. However, we can increase the relevance of Remark 4 more significantly by translating its basic observations into a stronger result.

For this, let R(Q) = P(Q) – V and let $R(V) = \cup (R(Q): Q \in S(V))$. Then define $R = \{j \in R(V): \exists$ at least one $i \in V$ such that $j \in R(Q)$ for every Q that contains i$\}$. (Or more formally, if $S_i(V) = \{Q \in S(V): i \in Q\}$ and $R_j(V) = \{Q \in S(V): j \in R(Q)\}$, then $R = \{j \in R(V): \exists$ at least one $i \in V$ such that $S_i(V) \subseteq R_j(V)\}$.)

Theorem 9. If [V:m] is a weak q-complete system, then
$\sum (x_i: i \in V \cup R) \leq m$.

Proof: The preceding arguments establish that the inequality of Theorem 9 holds with R removed. Moreover, each x_j, $j \in R$, appears in all inequalities that some x_i, $i \in V$ would appear in, if the system [V:m] were relaxed by setting coefficients of x_i to 0 for $i \in V - Q$ in each of the inequalities $\sum (x_i: i \in Q \cup P(Q)) \leq m$. The simple-sum surrogate constraints used to derive Remark 1 give x_i a unit coefficient in the inequality $\sum (x_i: i \in V) \leq m$, based only on whether or not $i \in Q$ holds for specified sets Q. Hence each variable x_j must receive a coefficient at least as large as an associated x_i variable in this surrogate constraint. Consequently, x_j must receive a positive coefficient in the inequality of Theorem 9 over $V \cup R$. □

We observe several consequences of this result that further extend its utility.

Corollary 9.1. The conclusion of Theorem 9 holds for a weak q-complete system whose inequalities are given by

$$\sum (x_i: i \in Q \cup P_o(Q)) + \sum (x_j: j \in R) \leq m, \quad Q \in S(V)$$

where $R \subseteq N - V$ and $P_o(Q) \subseteq V$ for each $Q \in S(V)$.

Proof: The Corollary is readily verified to be a special case of the theorem.□

The next result demonstrates that inequalities of greater strength are "hidden" within the statement of Theorem 9.

Corollary 9.2. Assume [V:m] is a weak q-complete system for $q > m + 1$. Define $q^* = m + 1$, and let V^* be any proper subset of V containing $|V| - (q - q^*)$ elements. Then the weak q^*-complete system $[V^*:m]$, given by $\sum (x_i: i \in Q^* \cup P(Q^*)) \leq m$ for $Q^* \in S(V^*)$, where $S(V^*) = \{Q^* \subseteq V^*: |Q^*| = q^*\}$, yields an inequality that dominates the inequality of Theorem 9. In particular, the inequality

$$\sum (x_i: i \in V^* \cup R^*) \leq m$$

for R^* defined relative to V^* as R is defined relative to V, is at least as strong as the inequality

$$\sum (x_i: i \in V \cup R) \leq m$$

taken directly from [V:m], as determined by $V \cup R \subseteq V^* \cup R^*$.

Proof: The result follows by noting that every q^*-element subset Q^* of V^* is contained a q element subset Q of V, where the system [V:m] can be written as

$$\sum (x_i: i \in (Q^* \cup (Q - Q^*)) \cup P(Q)) \leq m, \quad Q \in S^* \quad (1)$$

$$\sum (x_i: i \in Q \cup P(Q)) \leq m, \quad Q \in S(V) - S^* \quad (2)$$

and where $S^* = \{Q \in S(V): Q^* \subset Q \text{ for some } Q^* \in S(V^*)\}$. The system $[V^*:m]$, given by

$$\sum (x_i: i \in Q^* \cup P(Q^*)) \leq m, \quad Q^* \in S(V^*)$$

is in fact the same as inequality (1) of [V:m], i.e., $P(Q^*) = (Q - Q^*) \cup P(Q)$. No element $j \in Q - Q^*$ in [V:m] qualifies to belong to R (since this element is in Q), whereas all elements of $Q - Q^*$ belong to R^*. Moreover, if P(Q) for $Q \in S(V)$ contains any $j \in R$, the fact that (1) ranges over a proper subset of the inequalities for [V:m] assures that $j \in R^*$, and hence we conclude $V \cup R \subseteq V^* \cup R^*$, which establishes that the inequality over $V^* \cup R^*$ is at least as strong as the one over $V \cup R$. □

Corollary 9.2 is useful not only for the opportunity to obtain stronger inequalities, but for the fact that it permits the search for a weak q-complete system to be carried out by restricting attention to $q = m + 1$, which involves less work than the search for such a system with a larger value of q.

12. 3 Expansions by Lower Order Progressions

Theorem 9 and the associated observation of Corollary 9.2 have a further important implication, which involves the ability to expand the inequality $\sum (x_i: i \in V \cup R) \leq m$ by a progression that creates new inequalities over lower order systems. In particular, consider any inequality $\sum (x_i: i \in V_1) \leq m$ (where V_1 may be given by $V_1 = V \cup R$ from Theorem 9 or $V_1 = V^* \cup R^*$ from Corollary 9.2). We seek to enlarge V_1 to include one or more additional variables, i.e., to find some $j \notin V_1$ that can be added to V_1 and still permit $\sum (x_i: i \in V_1) \leq m$ to hold. We show that repeated application of Theorem 9 makes this possible and also has the added benefit of restricting attention to smaller values of q than given by the stipulated lower bound of $q \geq m + 1$. This not only saves effort in searching for such systems, but gives a systematic design for strengthening the inequality $\sum (x_i: i \in V_1) \leq m$ by enlarging V_1 step by step.

The mechanism for achieving this may be described as follows. If we assume $|V_1| \geq m + 1$, then our previous results show that $\sum (x_i: i \in V_1) \leq m$ is equivalent to a collection of associated inequalities $\sum (x_i: i \in Q) \leq m$, where Q ranges over all q-element subsets of V_1, for some $q \geq m + 1$. Thus, if $j \notin V_1$ and we are able to identify a set of inequalities $\sum (x_i: i \in Q) + x_j \leq m$, $Q \in S(V_1) = \{Q \subseteq V_1: |Q| = q\}$, but where q now is only required to satisfy $q \geq m$, then this identifies system that is equivalent to a set of inequalities over all subsets of $q \geq m + 1$ elements of $V_1 \cup \{j\}$.

By means of this observation, the ability to expand an inequality system by looking at lower order systems (for smaller values of q) can be expressed as follows, where we replace j more generally by a set of elements R_1 from $N - V_1$ to take fuller advantage of Theorem 9.

Corollary 9.3. (First Expansion by Lower Order) Assume $V_1 \subset N$, $|V_1| \geq m + 1$, $R_1 \subseteq N - V_1$ and $|R_1| \geq 1$. Then the two-part system

$$\sum (x_i: i \in V_1) \leq m$$
$$\sum (x_i: i \in Q) + \sum (x_j: j \in R_1) \leq m, \quad Q \in S_1(V_1)$$

where $S_1(V_1) = \{Q: Q \subseteq V_1 \text{ and } |Q| = q\}$, for some $q \geq m$, implies the inequality

$$\sum (x_i: i \in V_1) + \sum (x_j: j \in R_1) \leq m.$$

Proof: Follows by preceding arguments.□

The ability to take advantage of smaller order systems does not stop here, and we can express the result at a deeper level as follows.

Corollary 9.4. (Second Expansion by Lower Order) For V_1 and R_1 as given in Corollary C, assume $V_2 \subset V_1$, $|V_2| \geq m$, $R_2 \subseteq V_1 - V_2$ and $|R_2| \geq 1$. Then the three-part system

$$\sum (x_i: i \in V_1) \leq m$$
$$\sum (x_i: i \in V_2) + \sum (x_j: j \in R_1) \leq m$$
$$\sum (x_i: i \in Q) + \sum (x_j: j \in R_1 \cup R_2) \leq m, \quad Q \in S_2(V_2)$$

where $S_2(V_2) = \{Q: Q \subseteq V_2 \text{ and } |Q| = q\}$, for some $q \geq m - 1$, implies the inequality

$$\sum (x_i: i \in V_2) + \sum (x_j: j \in R_1 \cup R_2) \leq m.$$

Proof: Follows by preceding arguments.□

Comparison of Corollaries 9.3 and 9.4 discloses the pattern for going to deeper levels. The result applicable to an arbitrary level h, for $h \geq 2$, identifies a corresponding (h+1)-part system based on stipulating $V_h \subset V_{h-1}$, $|V_h| \geq m + 2 - h$, $R_h \subseteq V_{h-1} - V_h$, $|R_h| \geq 1$, and $S_h(V_h) = \{Q: Q \subseteq V_h \text{ and } |Q| \geq m + 1 - h\}$.

12. 4 Inequalities from Sub-Complete Systems.

The basic surrogate constraint analysis that led to Theorem 9 and its corollaries can be carried a step farther. We develop a more general inequality system by reference to building blocks that exhibit features analogous to the building blocks of q-complete systems, which are the simplest non-trivial forms of these systems where $|V| = q + 1$. These minimal q-complete systems are a special case of a structure we call *q-sub-complete* systems. We identify the special character of these more general systems and the inequalities they generate as follows.

Consider the system

(V:m,r)

$$\sum (x_i: i \in V) \leq m$$
$$\sum (x_i: i \in Q) + \sum (x_j: j \in R) \leq m, \quad Q \in S(V)$$

where $R \neq \emptyset$, $R \cap V = \emptyset$ and $R \cap Q = \emptyset$ for $Q \in S(V)$. (If the set $R_o = \cap(Q - V: Q \in S(V))$ is nonempty, then we replace each Q by $Q - R_o$ and add R_o to R.)

To complete the characterization of the system, define $S_i(V) = \{Q \in S(V): i \in Q\}$ and let $r(i) = |S_i(V)|$, that is, r(i) is the number of inequalities from the subsystem

$$\sum (x_i: i \in Q) + \sum (x_j: j \in R) \leq m, \quad Q \in S(V)$$

in which x_i appears with a unit coefficient. If $i \notin Q$ for all $Q \in S(V)$, then $S_i(V) = \emptyset$ and by convention $r(i) = 0$.

The *rank* r of (V:m,r) is then defined by

$$r = \text{Min}(r(i): i \in V).$$

We assume $r \geq 1$, which is equivalent to the requirement that the sets $Q \in S(V)$ cover V, i.e., $V \subseteq \cup(Q \in S(V))$.

Finally, we define $q = |S(v)|$, hence q is the number of inequalities of (V:m,r) excluding $\sum (x_i: i \in V) \leq m$. Clearly $q \geq r$, and we may suppose $q > r$, or else $V \subseteq Q$ for all $Q \in S(V)$ and the inequality $\sum (x_i: i \in V) \leq m$ is redundant.

We call (V:m,r) a q-sub-complete system when the foregoing assumptions hold; i.e., in summary, when $R \neq \emptyset$, $R \cap V = \emptyset$, $R \cap Q = \emptyset$ for $Q \in S(V)$, and $q = |S(v)| > r \geq 1$.

Remark 3. A minimal q-complete system (V:m) (for $|V| = q + 1$) is a q-sub-complete system (V:m,r) of rank $r = q - 1$.

This observation, which follows directly from the preceding definitions and comments, shows the increased generality afforded by q-sub-complete systems as building blocks for generating inequalities. Our key result for exploiting the generality provided by these systems may be expressed as follows.

Theorem 10. The q-sub-complete system (V:m,r) implies the inequality

$$\sum (x_i: i \in V) + \sum (x_j: j \in R) \leq m$$

for any m that satisfies $m < q/(q - r)$ (where $q = |S(V)|$).

Proof: To establish the theorem, create a surrogate constraint that gives a weight of $q - r$ to $\sum (x_i: i \in V) \leq m$, and a weight of 1 to each of the remaining inequalities of (V:m,r). This yields a surrogate constraint $\sum (a_ix_i: i \in Q_o) + \sum (a_jx_j: j \in R) \le a_o$, where $Q_o = \cup(Q \in S(V))$, and by construction $a_i = (q - r) + r(i)$ ($\ge q$) for each $i \in V$, $a_j = q$ for each $j \in R$, and $a_o = (q - r)m + qm$. We relax the surrogate constraint by setting $a_i = q$ for $i \in V$ (if any coefficients exist with $a_i > q$) and by discarding variables x_i for $i \in Q_o - V$ (whose coefficients are smaller than q).[12] Thus, we obtain

$$q(\sum (x_i: i \in V) + \sum (x_j: j \in R)) \le (q - r)m + qm$$

or equivalently

$$\sum (x_i: i \in V) + \sum (x_j: j \in R) \le ((q - r)/q)m + m$$

In consequence, we then have

$$\sum (x_i: i \in V) + \sum (x_j: j \in R) \le m$$

provided the right hand side $((q - r)/q)m + m$ of the surrogate constraint is less than $m + 1$, or in short, provided $m < q/(q - r)$ as stipulated.□

For the special case of a minimal q-complete system, where $r = q - 1$, the condition $m < q/(q - r)$ of the theorem translates into the familiar condition $q \ge m + 1$.

Theorem 10 evidently remains valid if the inequality $\sum (x_i: i \in V) \le m$ of (V:m,r) is replaced by $\sum (x_i: i \in V \cup V_o) \le m$, for any set V_o. For a statement of the theorem that involves such a replacement, we may assume $V_o \cap R = \emptyset$ to avoid triviality, and we may also suppose $r(i) < r$ for $i \in V_o$ (since any $i \in V_o$ such that $r(i) \ge r$ can be moved from V_o to V, to give a stronger inequality by Theorem 10).

12. 5 Inequalities for Special Sub-Complete Systems.

Two special sub-complete systems, those for $m = 1$ and those for zero-one variables, provide additional results.

We first observe that $m = 1$ is a privileged value for m, since the condition $m < q/(q - r)$ always hold for this case. Consequently, we have the following immediate outcome of Theorem 10.

Corollary 10.1. If S(V) is any collection of sets Q that cover V, then the inequalities

$$\sum (x_i: i \in V) \le 1 \text{ and}$$
$$\sum (x_i: i \in Q) + \sum (x_j: j \in R) \le 1, \quad Q \in S(V)$$

imply

$$\sum (x_i: i \in V) + \sum (x_j: j \in R) \le 1.$$

The strongest form of the inequality produced by the preceding corollary of course results by restricting attention to minimal covers. A more restricted special case arises in the situation where the inequalities of Corollary 10.1 represent clique inequalities. Then Corollary 10.1 yields the following outcome.

Corollary 10.2. Let V and the sets $Q \in S(V)$ be cliques on a graph G, and assume the elements of S(V) provide a cover of V. Then the expanded set $V \cup R$ is also a clique, for $R = \cap(Q - V: Q \in S(V))$.

[12] These steps correspond to including inequalities $x_i \ge 0$, in the form $- x_i \le 0$, within the surrogate constraint construction.

These rather apparent corollaries are accompanied by a less obvious result for 0-1 problems.

Corollary 10.3. Let $r'(i)$, $i \in V$, be a set of values such that $r'(i) \geq r(i)$, and let $r' = \text{Min}(r'(i): i \in V)$. Assume $r' > r$ and define $s = \sum (r'(i) - r(i): i \in V)$. Then for a 0-1 problem, a q-sub-complete system (V:m,r) implies the inequality

$$\sum (x_i: i \in V) + \sum (x_j: j \in R) \leq m$$

provided $m < (q - s)/(q - r')$.

Proof: We modify the surrogate constraint construction in the proof of Theorem 10 by adding a subset of the inequalities $x_i \leq 1$, $i \in V$, to yield a new surrogate constraint. To do this, we identify indexes i which include those such that $r(i) = r$, and add a quantity to each of these $r(i)$ values to give new values $r'(i)$, as specified. For the value r' that results, the associated surrogate constraint created by the construction in the proof of Theorem 10 becomes

$$q(\sum (x_i: i \in V) + \sum (x_j: j \in R)) \leq (q - r')m + qm + s$$

or equivalently

$$\sum (x_i: i \in V) + \sum (x_j: j \in R) \leq ((q - r')m + s)m + m.$$

The right hand side is less than m + 1 if and only if $m < (q - s)/(q - r')$, which yields the conclusion of the Corollary.□

The potential improvement provided by Corollary 10.3 is constrained by relationships between r', r, s and q, as disclosed by the following observation, whose proof is immediate.

Remark 4. The limit on m given in Corollary 10.3 is less restrictive than the limit $m < q/(q - r)$ of Theorem 10 only if $q(r' - r) > s(q - r)$.

The potential strengthening that arises by Corollary 10.3 can also be undertaken by an alternative device, which often yields an inequality at least as strong and applies to a more general system where the variables are not required to be 0-1.

To see how this occurs, define a q-sub-complete system to be *tight* if q cannot be reduced except by reducing r; i.e., no inequality over $Q \in S(V)$ can be discarded without reducing $\text{Min}(r(i): i \in V)$, or equivalently, for every $Q \in S(V)$ there is at least one $i \in Q$ such that $r(i) = r$. The following result shows that tight systems yield the best opportunities for strengthening a set of inequalities by Theorem 10 and also impose limitations on the relationship between q and r.

Corollary 10.4. Let (V:m,r) satisfy the conditions of a q-sub-complete system, and let v denote the number of variables x_i, $i \in V$, such that $r(i) = r$. Then if (V:m,r) is tight, $q \leq r + v - 1$. Moreover, if (V:m,r) is not tight, then it is possible to identify a set $Q' \in S(V)$ and remove the associated inequality

$$\sum (x_i: i \in Q') + \sum (x_j: j \in R) \leq m,$$

by redefining $S(V) := S(V) - Q'$, so that the new resulting (V:m,r) system imposes less restrictive conditions on m to yield a valid inequality by Theorem 10.

Proof: The relationship $q \leq r + v - 1$ for a tight (V:m,r) system can be established by induction on v, first observing that $v = 1$ implies $q = r$. For the remaining part of the proof, it suffices to choose Q' to be any element of S(V) such that $r(i) > r$ for all $i \in Q'$. The rest of the argument is apparent.□

The key to applying the foregoing results is to focus on denser sub-matrices within the coefficient matrix for a system of inequalities. The generation of new inequalities relative to these sub-matrices then introduces new sub-matrices of increased density (relative to variables with unit coefficients in the new inequalities), thus potentially allowing a repetition of the process to generate additional new inequalities.

In general, the significance of Theorem 10 for q-sub-complete systems lies in providing a structure to create inequalities that are more comprehensive than those produced by the simpler forms of q-complete systems.

13. Embedded Approaches Using Vocabulary Building

The results of preceding sections can be incorporated within vocabulary building strategies to provide additional approaches for independent set and graph coloring problems. Vocabulary building operates by creating a pool of solution fragments, which are successively assembled, disassembled and modified to produce new fragments that are ultimately transformed into trial solutions. A useful analogy exists to the use of surrogate constraint processes to build a pool of inequalities, following the types of designs previously discussed. As shown in Section 12, it is possible to derive inferences for obtaining stronger inequalities – better members of the pool – by a process of joining smaller sub-systems before proceeding to larger ones. In the same way, vocabulary building applied to pools of solutions gains advantages by decomposing solutions into components, and then proceeding through stages of building larger components from smaller ones. Prominent application of vocabulary building methods occurs, for example, by creating *structured combinations* of solution fragments (Glover 1994), where the values assigned to variables become votes that are translated into decision criteria and objective functions for methods to generate new solutions.

Simple ways to create sub-problems associated with solution fragments in the present setting consist of generating sub-graphs by extracting them as unions, intersections and differences of independent sets derived from previously generated solutions (primarily elite solutions and solutions chosen for their diversity). In the simplest case, the pool of sub-graphs can be generated from independent sets obtained from a single current solution structure.

Applied to maximum independent set problems, such an approach is entirely straightforward, by assembling sub-graphs composed of selected independent sets from a vocabulary building pool. The goal is simply to find larger independent sets within these sub-graphs in order to identify new candidates for the maximum independent set.

In the context of coloring problems, the process has additional features. The sub-graphs in this case may be composed of a chosen number of independent sets determined by a selected coloring, or determined by a partial coloring derived as an assembly from a vocabulary pool. (The number of sets selected to compose such a sub-graph can be as small as 2, and generally may be allowed to range over an interval of values.) Then, by generating larger independent sets within such a sub-graph, the process gives a means for achieving the goal of reducing the total number of independent sets, and thus reducing the number of colors required.

Within the body of such a procedure, an auxiliary objective may appropriately be introduced that simultaneously seeks to assign nodes to other pre-existing independent sets determined by current graph colorings. For example, if the sub-graph under consideration is derived from independent sets from a single coloring, then nodes of this sub-graph that are excluded from the new independent set should include as many as possible that can be assigned to other independent sets for the coloring. Choices that set variables $x_r = 0$ (or $y_r = 1$ in the IP2 formulation) may accordingly be biased in favor of nodes r that can be added to such other independent sets, or which connect to as few nodes as possible in one of these sets. If a current re-coloring attempt does not succeed, some node assignments may violate the condition that all sets are independent. In such cases, the sub-graph selected by the processes described here may include component node sets that are not independent.

These auxiliary choice criteria can be handled by focusing on assignments of the form $x_r = 0$ during the first steps of the procedure for extracting an independent set from G, for the purpose of initially removing some number of nodes from G that can be re-assigned to other independent sets. Such criteria can alternatively be brought into play as tie breaking rules which are invoked only when the current choices do not offer any strong winner for a node to be added or excluded from the independent set under construction.

Similarly, if a current sub-graph is drawn from node sets that belong to more than one coloring, then the auxiliary criteria can undertake to assign excluded nodes (determined by choices $x_r = 0$) to independent sets from each of the colorings considered, eventually focusing on the coloring to which this assignment proves most successful.

A key premise underlying these types of strategies, both for coloring and independent set problems, is that a method which focuses on creating desired structures from sub-graphs rather than from the entire graph will have a greater likelihood of success, provided the selected sub-graphs are appropriately chosen. (This is one of the motivating premises of vocabulary building processes generally.)

The indicated strategies are highly compatible with a variety of metaheuristic procedures. For example, adaptive memory approaches can be used to control the choices of sub-graphs treated, and evolutionary procedures can be used to provide combined solutions as sources of the fragments within the vocabulary building pool. By considering different numbers and choices of fragments simultaneously, the strategy is susceptible to incorporation in a multilivel cooperative search process, employing designs such as proposed in Toulouse, Thulasiram and Glover (1999) and Ouyang et al. (2000a, 2000b).

14. Additional Surrogate Constraint Refinements and Uses.

Surrogate constraints can be generated in additional ways apart from the use of normalizations. For example, the solution to the dual of the linear programming relaxation of an IP problem gives a valid surrogate constraint. A stronger surrogate constraint can be generated by applying a method described in Glover (1965), which solves the knapsack problem created by the surrogate constraint, and then increases the weights on violated constraints by a specified formula, repeating until either all original constraints are satisfied or conditions disclose that the strongest surrogate constraint has been obtained. (A description of this method appears in Appendix 2.)

In the setting of independent set problems, a similar approach can be used to modify the choices made by the constructions described in the preceding sections. At the conclusion of generating a trial solution that yields an independent set (which will be maximal, but not necessarily maximum), a composite surrogate constraint can be generated from the collection of surrogate constraints used to make the choices that produced this solution. One possibility is to sum each of these prior surrogate constraints to create the composite constraint. (This would normally require the weights that produce these surrogate constraints to be applied to the original problem constraints. In this case, however, the surrogate constraints at each stage are generated relative to some subset of the original constraints, rather than to a reduced form of these constraints. Hence no "recovery step" is needed.)

Such an approach will give greater emphasis to those original inequalities that have endured through a larger number of iterations before being eliminated as redundant.

Consequently, the resulting surrogate constraint will amplify the size of the a_i coefficients for variables that are among the last to be given values, either by choice or by a forced assignment. As a result, these variables that most recently received their current values will appear least attractive to receive these assignments (by comparison to assignments selected for other variables) and hence will appear to be the best candidates for changing their assignments.

Such a bias can have undesirable consequences. A better surrogate constraint for analyzing current choices is likely to be obtained by recovering and using an earlier surrogate constraint in which a number of tied evaluations appeared. Rather than bothering to recover the constraint, however, the choice process can be simplified by keeping track of prior evaluations, as they have evolved over successive iterations.

This type of memory (also used in tabu search) can be useful for more advanced strategies. For example, choices that received high evaluations throughout a series of iterations may be implemented at an earlier stage of a subsequent constructive pass. This early stage implementation of such choices can cause other ensuing evaluations to change, leading to different choices at later stages and thus producing different final solutions. Choices that received enduring high evaluations, but which eventually were discarded (and hence never implemented) as a result of making other choices, particularly invite examination. (Systematic ways to apply these types of strategies are described in Glover, 2000.) Alternatively, evaluations for current choices can be generated relative to surrogate constraints produced by subgradient updates, as also elaborated in Appendix 2.

15. Conclusions.

Surrogate constraint approaches can be applied in a variety of ways for optimization problems in graphs, by mechanisms that also afford an understanding of how these methods can be used in a number of other settings. The basic ideas underlying these approaches are especially convenient to illustrate in application to independent set problems, by strategies that seek maximum independent sets over strategically selected sub-graphs.

A useful relationship for exploiting surrogate constraint strategies in graphs derives from the fact that each value assignment in a standard mathematical programming formulation corresponds to a natural modification of the graph structure, so that the outcome is again a graph and all evaluation criteria and updating operations can be carried out by reference to this same structure. The ability to take advantage of the representational properties of graphs provides a particularly efficient basis for implementation.

Surrogate constraint approaches can be applied in quickly executed strategies that generate trial solutions very rapidly, or can be the basis for more advanced strategies that exploit bounding information at multiple levels. As our development has shown, the power of surrogate constraint approaches can be amplified by making use of principles that involve conditionally shared limitations. Theorems for exploiting these principles offer new constructions to generate bounds and associated choice rules, while still affording advantages of efficient implementation. These procedures generalize to the setting of GUB-constrained surrogate constraint relaxations, where the GUB systems come from clique inequalities. Additional advantage can be taken of these GUB-constrained relaxations by dynamic solution approaches that generate the relevant inequalities and also solve the surrogate constraint problems based on them at the same time. We have further observed how these approaches can be applied with more general quasi-clique inequalities, and can take advantage of useful structures called q-complete (and sub-complete) systems. The underlying results carry over to additional settings, including zero-one and general integer programming.

Finally, we have noted how surrogate constraint heuristics such as those illustrated in the preceding sections acquire additional scope by implementing them within the framework of vocabulary building methods. This affords a mechanism to identify sub-problems that give rise to new trial solutions by strategically isolating relevant sub-graphs of the problem graph. Such an integration of surrogate constraint approaches with vocabulary building methods can also conveniently be exploited by metaheuristics such as tabu search, evolutionary methods and multilevel cooperative search.

References:

Abello, J., P.M. Pardalos and M.G.C. Resende (1999). On maximum clique problems in very large graphs. *External memory algorithms and visualization*, J. Abello and J. Vitter, eds. *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, American Mathematical Society, Vol. 50, 199-230.

Chvatal, Vasek (1979). A greedy heuristic for the set-covering problem. *Mathematics of Operations Research* 4 , 233-235.

Dyer, M.F. (1980), Calculating surrogate constraints. *Mathematical Programming,* Vol.19, 255-278.

Dorne, R. and J.K. Hao (1998). A new genetic local search algorithm for graph coloring. *Lecture Notes in Computer Science 1498*, Springer-Verlag, 745-754.

Feo, Thomas A., Resende, Mauricio G.C., and Smith, Stuart H. (1994). A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, Vol. 42: 5, 860-878.

Fréville, A. and S. Hanafi (2000). Des Bornes Duales Robustes pour le Sac à Dos Bidimensionnel en Variables 0-1. *ROADF'2000*, Nantes, France.

Fréville, A. and Plateau, G. (1992). An implicit enumeration code for the solution of the 0-1 bidimensional knapsack problem based on surrogate duality. *Graphs and Optimization Colloquium*, Grimentz, Switzerland.

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

Friden, C., Hertz, A. and Werra, D. de (1989). Stabulus: a technique for finding stable sets in large graphs with tabu search. *Computing,* Vol. 42, 35-44.

Friden, C., Hertz, A. and Werra, D. de (1990). Tabaris: An exact algorithm based on Tabu Search for finding a maximum independent set in a graph. *Computers & Operations Research*, Vol. 155, 437-445.

Galinier, P. and J.K. Hao (1999). Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization.* Vol. 3, No. 4, 379-397.

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

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

Glover, F. (1965). A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problems. *Operations Research*, Vol. 13, No. 6, 879-893.

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

Glover, F. (1971). Flows in Arborescences. *Management Science,* Vol.17, No. 9, 568-586.

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

Glover, F. (1977). Heuristics for Integer Programming Using Surrogate Constraints. *Decision Sciences*, Vol. 8, No. 1. 156-166.

Glover, F. (1994). Tabu Search for Nonlinear and Parametric Optimization (with Links to Genetic Algorithms). *Discrete Applied Mathematics* Vol. 49, 231-255.

Glover, F. and D. Klingman (1988). Layering Strategies for Creating Exploitable Structure in Linear and Integer Programs. *Mathematical Programming*, Vol. 40, No. 2, 165-182.

Glover, F. and M. Laguna (1993). Tabu Search. *Modern Heuristic Techniques for Combinatorial Problems*, C. Reeves, ed., Blackwell Scientific Publishing, 71-140.

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

Glover, F. and G. Kochenberger (1996). Critical event tabu search for multidimensional knapsack problems. *Meta-Heuristics: Theory & Applications*, J.H. Osman and J.P. Kelly, eds., Kluwer Academic Publishers, 407-427.

Glover, F., H. D. Sherali and Y. Lee (1997). Generating Cuts from Surrogate Constraint Analysis for Zero-One and Multiple Choice Programming. *Computational Optimization and Applications*, Volume 8, Number 2, 151-172.

Glover, F. (2000). Multi-Start and Strategic Oscillation Methods – Principles to Exploit Adaptive Memory. *Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research*, M. Laguna and J.L. Gonzales Velarde, eds., Kluwer Academic Publishers, 1-24.

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

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

Hanafi S., (1993). Contribution à la résolution de problèmes duaux de grande taille en optimisation combinatoire. PhD thesis, Université de Valenciennes et du Hainaut-Cambrésis, France.

Hanafi, S. and A. Fréville (2001). Résolution du Dual Composite du Sac à Dos Bidimensionnel en variables 0-1 par une méthode de Branch-and-Bound. Francoro III Conference, Quebec, Canada.

Homer, S. and M. Peinado (1996). Experiments with Polynomial-time CLIQUE Approximation Algorithms on Very Large Graphs, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol. 26, 147-155.

Johnson, D.S. (1974). Approximation algorithms for combinatorial problems, *Journal of Computer and System Sciences*, Vol 9, 256-278.

Johnson, D.S., Trick M.A eds (1996). *Clique, Coloring and Satisfiability, Second Implementation Challenge*, DIMACS, AMS.

Joseph, A., S. Gass, and N. Bryson (1998). An objective hyperplane procedure for solving the general all-integer linear programming (ILP) problem. *European Journal of Operational Research,* Vol. 104, 601-614.

Klingman, D. and D. Karney (1979). A study of alternative relaxation approaches for a manpower planning problem. *Quantitative Planning and Control*, Ijiri and Whinston, eds., Academic Press, Inc., NY, 141-164.

Kochenberger, G.A., B.A. McCarl, and F.P. Wyman (1974). A Heuristic for General Integer Programming. *Decision Sciences*, Vol. 5, No. 1, 36-44.

L∅kketangen, A. and F. Glover (1997). Surrogate Constraint Analysis – New Heuristics and Learning Schemes for Satisfiability Problems. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol. 35, 537-572.

Osorio, M.A., F. Glover and P. Hammer (2002). Surrogate constraint analysis for improved multidimensional knapsack solutions. To appear in *Discrete Applied Mathematics*.

Ouyang, M., M. Toulouse, K. Thulasiraman, F. Glover, J.S. Deogun (2000a). Multilevel Cooperative Search for the Circuit/Hypergraph Partitioning Problem. To appear in *IEEE Transactions on Computer-Aided Design*.

Ouyang, M., M. Toulouse, K. Thulasiraman, F. Glover, J.S. Deogun (2000b). Multi-Level Cooperative Search: Application to the Netlist/Hypergraph Partitioning Problem. *International Symposium on Physical Design*, ACM Press, 192-198.

Padberg, M.W. (1973). On the facial structure of set packing polyhedra. *Mathematical Programming*, Vol. 5, 199-215.

Pardalos, P., J. Xue (1994). The maximum clique problem. *J. of Global Optimization*, Vol. 4, No. 3, 286-301.

Resende,M.G.C., T.A. Feo and S.H. Smith (1998). Algorithm 787: Fortran subroutines for approximate solution of maximum independent set problems using GRASP. *ACM Transactions on Mathematical Software*, Vol. 24, 386-394.

Toulouse, M., K. Thulasiram and F. Glover (1999). Multi-Level Cooperative Search. *Lecture Notes in Computer Science*, Vol. 1685, Springer-Verlag, 533-542.

Yu, G. (1998). Min-max Optimization of Several Classical Discrete Optimization Poblems. *OJournal of Optimization Theory and Applications,* Vol. 98, No. 1, 221-242.

APPENDIX 1: Fast Look-Ahead Methods

We sketch the foundations for creating quickly implemented procedures of the type that are natural accompaniments of more advanced surrogate constraint approaches.

Look-Ahead Criteria for Setting $x_r = 1$.

A natural goal in making a choice of the form $x_r = 1$ is to create as many small *NodeStar* sets as possible, since the existence of nodes i with small values of *SizeStar*(i) produces stronger choices on subsequent steps (by the criterion of minimizing a_i values). This goal may be pursued by seeking an assignment $x_r = 1$ that will cause a maximum number of edges to be removed from G. Consequently, a useful auxiliary rule for choosing r is as follows. Let I be a subset of N that includes the nodes that qualify to be selected as node r, i.e., that yield a minimum value for *SizeStar*(i). Then we choose r so that

$$\mathit{SizeStar}(\mathit{NodeStar}(r)) = \mathrm{Max}(\mathit{SizeStar}(\mathit{NodeStar}(i)): i \in I).$$

This rule requires additional effort to apply, beyond the simple rule of the preceding section, although the effort can be reduced by restricting I to be somewhat smaller than N, as in the extreme case by limiting consideration to nodes that yield Min(*SizeStar*(i)). Specifically, for each such i, the rule requires examining each j in *NodeStar*(i) and accumulating the sum of the values *SizeStar*(j). (Alternately, for the same amount of work, the operation can identify Max(*SizeStar*(j): $j \in$ *NodeStar*(i)), and use this value to choose a node i that yields a maximum of these values.) Other choice rules can be similarly based on the idea of reducing *SizeStar*(i) for the choices that will be made in future steps.

Look-Ahead Criteria for Setting $x_r = 0$.

Similarly, in selecting $x_r = 0$, we may seek to leave the largest NodeStar(i) undiminished in size, or to leave as many of the large *NodeStar*(i) sets undiminished as possible, as a basis for enabling future choices of the form $x_r = 0$ to be as strong as possible. Thus, requiring I in this case to include the set of nodes i that yield the maximum value of *NodeStar*(i), we look for a node r in I that yields

$$\mathrm{Min}(\mathit{SizeStar}(i): i \in I)$$

or

$$\mathrm{Min}(\mathrm{Max}(\mathit{SizeStar}(j): j \in \mathit{NodeStar}(i)): i \in I).$$

The identification of such an r requires the same order of work as in the choice of r for setting $x_r = 1$. For a strategy that alternates decisions of the form $x_r = 1$ with those of the form $x_r = 0$, the rule for choosing r can be amended by reference to criteria similar to those identified.

Thresholding Approach.

Further refinement results by establishing an upper bound threshold value a* for the coefficients a_i, and defining the node set $N^* = \{i \in N: a_i \le a^*\}$. For example, we may define $a^* = \mathrm{Max}\ (a_i : a_i + ... + a_{h-1} + a_i \le a_o)$, where the a_i coefficients are arranged in ascending order and the index h identifies an optimal x_o value for SC1, i.e.,

$$x_o = \mathrm{Max}(h:\ a_1 + ... + a_h \le a_o)$$

as noted in Section 5.1.

Clearly, in order for a variable x_i to be able to receive an assignment $x_i = 1$ in an optimal solution to SC1, it is necessary for i to be in N*. Consequently, we are motivated to modify the look-ahead choice criterion by treating N* as a candidate list to identify possible "next choices" for x_r, for at least some number of steps after the current choice. Hence, instead of examining the effect of the forced assignments $x_j = 0$ on all nodes $k \in$ NodeStar(j) (for those assignments forced by currently setting $x_r = 1$), it is relevant to examine the effect only on nodes k that belong to *SubStar*(j) = *NodeStar*(j) ∩ N*.

To be precise, for each node $i \in I$, for the set I as identified in Section 4.1, identify each $j \in$ *NodeStar*(i) (which corresponds to an x_j to be forced to 0 if i is chosen as r). Then, sum the values |*SubStar*(j)| to find the total reduction in a_k values produced by setting $x_i = 1$. The coefficient a_i itself will be reduced to 0 by this, and so to adjust for the effect on a_i (in the case where I is allowed to contain coefficients other than those with a minimum a_i value), the choice criterion becomes to choose $r \in I$ to

Maximize(∑(|*SubStar*(j)|: $j \in$ *NodeStar*(i)) - *SizeStar*(i)): $i \in I$)

(This uses the fact that a_i = *NodeStar*(i).) The amount of work to apply this refined look-ahead rule is approximately the same as to apply the simpler rule in Section 4.1.

We may also choose a* to be smaller than specified above, motivated by the observation that the set N* will contain more elements than contained in an optimal surrogate constraint solution, except in the special case where $|N^*| = x_o(SC1)$. (Even then, N* will contain more elements than found in an optimal solution to the original problem, unless there is no surrogate duality gap, and $x_o(SC1)$ is the actual optimum objective function value for this problem.) The value of a* may therefore be reduced to yield a set N* that one hopes to contain fewer elements which are not part of an optimal solution.

A counter to this rationale derives from considering that if a* is not reduced (or even increased) and N* is "too large," there may be elements in this larger set that are missing from the smaller one and that will be in an optimal solution. To account for the effect of these elements, it is reasonable to occasionally select a larger a*. However, attending this, N* will contain a larger number of nodes that are certain not to be in an optimal solution. Empirical testing may be useful to determine a good balance between these competing concerns.

APPENDIX 2: Iterative Surrogate Constraint Generation

Surrogate constraints derived from normalizations are typically better than those based on simple-summing, but they are still subject to a number of weaknesses. For example, if some inequalities are copies or "near-copies" of others, after scaling, then these closely related inequalities can receive a distorted emphasis – e.g., by an over-representation analogous to including a particular inequality multiple times within the overall collection. More generally, it is characteristically preferable to give emphasis to constraints that are binding at optimality. A lack of knowledge about the identity of such constraints also limits the value of normalizations, which are myopic in their handling of such considerations. To give a more sophisticated basis for creating surrogate constraints, it is therefore useful to consider iterative constructions, where trial solutions generated relative to the surrogate constraints provide feedback to guide the creation of modified forms of these constraints.

To describe fundamental ideas for iterative surrogate constraint generation, we introduce some notation and also review parts of the general perspective underlying the use of surrogate constraints. Consider the integer programming problem in the form

$$\begin{array}{lll} \text{(IP) Maximize} & cx & \\ \quad\quad\text{s.t.} & Ax \le b & \\ & x \in X & \end{array}$$

where c is a $1xn$ vector, A is an mxn matrix, b is an $mx1$ vector, and x is an $nx1$ vector of variables. The stipulation $x \in X$ is understood to restrict the components x_j of x to be non-negative integers, and in addition may impose other restrictions such as bounds and supplementary constraints that are convenient to keep separate.

At the most basic level, a surrogate constraint for the problem (IP) consists of a non-negative linear combinations of its component constraints

$$A_ix \le b_i,\ i \in M = \{1, ..., m\},$$

where A_i represents the *ith* row of A. Surrogate constraints may also include cutting planes iteratively derived from the original constraints and their linear combinations, which we may imagine as appended to enlarge the system $Ax \le b$ and hence the index set M (Glover, 1965). At any given point in the process, we may therefore represent a surrogate constraint relative to the current A matrix as

$$wAx \le wb$$

where w is a non-negative $1xm$ vector. The goal of generating such a constraint is to capture useful information not present in the individual constraints $A_ix \le b_i$, taken in isolation.

The associated surrogate constraint relaxation of (IP) given by

$$\begin{array}{ll} \text{(SR) Maximize } cx & \\ \quad\quad\text{s.t.} & wAx \le wb \\ & x \in X \end{array}$$

A *strongest* surrogate constraint (or a strongest (SR)), is one that is most restrictive relative to the shared objective of (IP) and (SR) and hence that yields the minimum value of the objective Maximize cx.

Basic Results for Generating Surrogate Constraints.

The iterative generation of surrogate constraints by subgradient approaches enjoys a special advantage that is not shared by Lagrangian versions of such approaches: the ability to identify precisely a step size, given any relevant direction vector, that will assure an improvement in the strength of the relaxation (if improvement in the specified direction is possible).

Let $w = w^*$ denote a currently selected vector for generating a surrogate constraint, and denote the associated surrogate constraint relaxation by (SR:w^*). If x^* is optimal for (SR:w^*) and is also feasible for (IP), then evidently x^* is optimal for (IP) and (SR:w^*) automatically qualifies as a strongest relaxation. Also, if x^* is "approximately optimal" for (SR:w^*) by a heuristic procedure, and is feasible for (IP), we may consider (SR:w^*) to be a heuristically strongest relaxation (relative to the procedure employed). In general, it is useful to include the goal of generating *heuristically strong* surrogate constraints, and associated relaxations (SR), which give restrictive bounds on cx within the margin of error determined by the efficacy of the heuristic involved.

When a solution x^* for (SR:w^*) is not feasible for (IP), the possibility of obtaining a form of (SR) stronger than (SR:w^*) depends on identifying the constraints of (IP) that are violated by x^*. For this purpose, define the index sets of violated, exactly satisfied and strictly satisfied constraints as:

$$V = \{i \in M: A_i x^* > b_i\}$$
$$E = \{i \in M: A_i x^* = b_i\}$$
$$S = \{i \in M: A_i x^* < b_i\}.$$

A step toward strengthening (SR) is to modify w^* by increasing the weights associated with violated constraints. The nature and definition of a subgradient, however, is different from that for the Lagrangian case, as shown in Glover (1975). Let d denote a direction vector designed to produce such a change in w^*, and let v denote a scalar step-size for moving in the direction d, so that the new vector w that replaces w^* may be represented as

$$w = w^* + vd.$$

It is possible to allow d to be any vector such that $dAx > db$**.** However, it is usually preferable to have

$$d_i > 0, \quad i \in V$$
$$d_i \geq 0, \quad i \in E$$
$$d_i = 0, \quad i \in S$$

A reason for choosing $d \geq 0$ (and specifically $d_i = 0$ for $i \in S$) is to avoid the need for stepwise changes in d as v increases, which otherwise might be required to prevent one or more components of w^* from becoming negative. However, there is a further advantage to choosing values for the d_i coefficients that yield $d \geq 0$.

Remark 1. A simple choice that assures $dAx^* > db$ and $d \geq 0$ is as follows. First, create an initial surrogate constraint with all weights positive. Then, each time a new surrogate constraint is to be generated (from a weight vector $w = w^* + vd$), select

$$d_i = w_i^* \quad \text{for } i \in V$$
$$d_i = 0 \text{ or } w_i^* \quad \text{for } i \in E$$
$$d_l = 0 \quad \text{for } i \in S$$

Implications of Remark 1 will be noted subsequently.

The strongest surrogate constraint that can be generated in the direction d (i.e., the strongest constraint that can be generated from $w = w^* + vd$, for some nonnegative step-size v)

can be determined as follows. Let ε refer to an arbitrarily small positive value. The main result consists of two parts.

Theorem (Strongest Surrogate Relaxation).

(I) Assume $w^* \geq 0$, x^* is feasible for (SR:w^*) and d is a direction vector such that $dAx^* > db$. Then x^* is also feasible for (SR:$w^* + vd$), for all nonnegative step-sizes v satisfying $v \leq v^*$, where

$$v^* = w^*(b - Ax^*)/d(Ax^* - b).$$

(II) Let $v' = v^* + \varepsilon$ and define $w' = w^* + v'd$, for w^*, x^* and v^* as given in (I). Further assume x^* is optimal for (SR:w^*), x' is optimal for (SR:w') and $d \geq 0$.

(a) If $cx^* < cx'$, then $dAx' \leq db$, and the relaxation (SR:w^*) is the strongest surrogate relaxation in the direction d.

(b) If $dAx' \leq db$ and $cx^* > \underline{cx'}$ then (SR:w') is a strongest surrogate relaxation in the direction d.

Proof: For part (I) of the theorem, the solution x^* will remain feasible for (SR) in the direction d if and only if $wAx^* \leq wb$ for $w = w^* + vd$, hence if and only if $w^*Ax^* + vdAx^* \leq w^*b + vdb$. Rearranging gives $v(dAx^* - db) \leq w^*(b - Ax^*)$, and the fact that $dAx^* > db$ leads to the conclusion $v \leq v^*$.

For part (II), $d \geq 0$ implies that $dAx \leq db$ is a surrogate constraint, and the problem (SR:$w^* + vd$) results by assigning a unit weight to the surrogate constraint $w^*Ax \leq w^*d$ and adding v times the surrogate constraint $dAx < db$ that is, (SR:$w^* + vd$) is a surrogate constraint relaxation of the two-inequality problem whose constraints are $w^*Ax \leq w^*d$ and $dAx \leq db$. For (a), $dAx' \leq db$ implies x' is feasible for the second surrogate constraint, and $\mathbf{x}'$ must also be feasible for all relaxations (SR:$w^* + vd$) such that $v \geq v'$. Hence $cx' \leq cx$ for optimal solutions x to all such relaxations, and (SR:w') must be the strongest of these relaxations. Consequently, for (b), the condition $dAx' \leq db$ assures that (SR:w^*) or (SR:w') or both give a strongest relaxation in the direction d, according to the relative size of cx' and cx^*. But if $dAx' > db$, then x' must be feasible for (SR:w^*), and hence $cx' \leq cx^*$. □

Corollary 1. Repeated application of Theorem 1, maintaining d unchanged and repetitively choosing $v' = v^* + \varepsilon$ to generate new surrogate constraints, will produce a succession of relaxations (SR:w') and associated solutions x' with the following properties:

(a) Each solution x' is different from all previous solutions generated (including the original x^*).

(b) The value cx' is monotonically nonincreasing and each relaxation (SR:w') is as strong or stronger than the previous one, until a first (critical) w' is reached such that the solution x' satisfies $dAx' \leq db$.

(c) Denote the predecessor to the critical w' of (b) by w'', and denote the solution corresponding to w'' by x''. Then (SR:w') or (SR:w'') is a strongest relaxation in the direction d, according to whether $cx' \leq cx''$ or $cx' \geq cx''$.

Corollary 2. Assume the surrogate constraint $w^*Ax \leq w^*b$, where w^* has at least two positive components, is divided into two component surrogate constraints

$$w_1^*Ax \leq w_1^*b \quad \text{and} \quad w_2^*Ax \leq w_2^*b$$

where $w_1^* + w_2^* = w^*$, and w_1^* and w_2^* are both nonnegative and nonzero. Then a strongest surrogate constraint that can be created from these two surrogate constraints occurs either for

$w = v_1w_1^* + w_2^*$ or for $w = w_1^* + v_2w_2^*$, where v_k is the smallest nonnegative value such that an optimal solution to (SR:w) satisfies $w_k^*Ax \le w_k^*b$, for $k = 1, 2$.

Remark 2. Assume that d is chosen as indicated in Remark 1, and let w^* be given as in Corollary 2 so that $w^* = w_1^* + w_2^*$, based on selecting $w_2^* = d$. Then, the vector w' of Theorem 1 results by

$$w_1' = w_1^* \quad \text{and} \quad w_2' = (u + \varepsilon)w_2^*,$$

where $u = w_2^*(b - Ax^*)/w_1^*(Ax^* - b)$.

Corollary 1 and Corollary 2 follow directly from Theorem 1, and Remark 2 follows from the definition of v′ in Theorem 1, upon substituting and simplifying terms. Remark 1 and Remark 2 are the basis for the special case of Theorem 1 proved in Glover (1965), which also implies a special instance of Corollary 1.

These results raise the question of how many iterated applications of Theorem 1 are likely to be required to generate a strongest relaxation in the direction d. Empirical outcomes for special cases of Theorem 1 indicate that by using the value v^* to guide a modified binary search, the total number of iterations is often only 3 to 8 (Gavish, Glover and Pirkul, 1991; Fréville and Plateau, 1993). However, Theorem 1 can clearly be applied by generating a new direction d before obtaining the greatest possible improvement in a given direction, and in general the theorem provides a variety of unexplored options that invite empirical examination.