A Scheme for Unifying Optimization and Constraint Satisfaction Methods

John Hooker,[†] Greger Ottosson,[‡]
Erlendur S. Thorsteinsson,[†] Hak-Jin Kim[†]

[†] Graduate School of Industrial Administration
Carnegie Mellon University
Pittsburgh, PA 15213, U.S.A.

[‡] Department of Information Technology
Computing Science, Uppsala University
Box 311, S-751 05 Uppsala, Sweden

April 1999; Revised October 1999

Abstract

Optimization and constraint satisfaction methods are complementary to a large extent, and there has been much recent interest in combining them. Yet no generally accepted principle or scheme for their merger has evolved. We propose a scheme based on two fundamental dualities, the duality of search and inference and the duality of strengthening and relaxation. Optimization as well as constraint satisfaction methods can be seen as exploiting these dualities in their respective ways. Our proposal is that rather than employ either type of method exclusively, one can focus on how these dualities can be exploited in a given problem class. The resulting algorithm is likely to contain elements from both optimization and constraint satisfaction, and perhaps new methods that belong to neither.

The last several years have seen increasing interest in combining the models and methods of optimization with those of constraint satisfaction. Integration of the two was initially impeded by their different cultural origins, one having developed largely in the operations research community and the other in the computer science and artificial intelligence communities. The advantages of merger, however, are rapidly overcoming this barrier.

There is a growing body of evidence that a hybrid approach can bring both modeling and algorithmic advantages. It includes computational studies involving chemical process design [36, 46], distillation network design [20, 27, 37], truss structure design [10], machine scheduling [21, 31], scheduling with resource constraints [34], highly combinatorial scheduling [27, 44], dynamic scheduling [15], production planning and transportation with piecewise linear functions [33, 38], warehouse location [9, 48], traveling salesman problem with time windows [16], hoist scheduling [43] and problems with specially ordered sets [14].

A recent commercially available modeling system, OPL, invokes both linear programming (ILOG Planner/CPLEX) and constraint programming (ILOG Solver) solvers [48].

Despite these developments, no generally accepted principle or scheme has evolved for the merger of optimization and constraint satisfaction. The purpose here is to propose such a scheme for unifying the solution methods of the two fields. We address elsewhere [27, 28, 30] the issue of a unified modeling framework.

Our scheme is based on exploiting two dualities: the duality of search vs. inference and the duality of strengthening vs. relaxation. Some of these ideas are anticipated in [22].

Branching algorithms provide one example of these dualities at work. The search/inference duality is evident in Bockmayr and Kasper's observation [9] that both optimization and constraint satisfaction rely on "branch and infer." Branching is a search mechanism. During the branching process, one can generate inferences in the form of cutting planes (as in optimization) or constraint propagation to achieve domain reduction (as in constraint satisfaction). The combination of branching and inference is usually much more effective than either alone.

The strengthening/relaxation duality is also evident in branching algorithms. When one branches on the possible values of a variable, the resulting subproblems are strengthenings of the original problem in the sense of a restriction; they have an additional constraint that fixes the value of the variable and therefore shrinks the feasible set. In optimization one typically solves a relaxation of the problem at each node of the search tree in order to obtain bounds on the optimal value, often a continuous relaxation such as a linear programming or Lagrangean relaxation. The reduced variable domains obtained in a constraint satisfaction algorithm in effect represent a relaxation of the problem at that node of the search tree. In any feasible solution the variables must take values in these domains, but an arbitrary selection of values from the domains need not comprise a feasible solution. Again enumeration of strengthenings is more effective when combined with relaxation of some kind.

The thesis of this paper is that because both constraint programming and optimization use problem-solving strategies based on the same dualities, their methods can be naturally combined. Rather than employ optimization methods exclusively or constraint satisfaction methods exclusively, one can focus on the how these dualities can be exploited in a given problem class. The resulting algorithm is likely to contain elements from both optimization and constraint satisfaction, and perhaps new methods that belong to neither.

Section 1 begins the paper with a simple illustration of how the dualities might operate in a branching context. It does so first in a constraint satisfaction and in an integer programming setting. It then combines the two approaches.

The next two sections explore the dualities more deeply and propose methods that belong to neither constraint satisfaction nor integer programming. Section 2 investigates the search/inference duality. It explains how constraint programmers exploit problem structure to apply effective inference algorithms. It also frames the search/inference duality as a formal optimization duality that generalizes classical linear programming duality. This perspective forges a link between the concept of a nogood in constraint satisfaction and Benders decomposition in optimization. It also provides a general method for sensitivity analysis.

Section 3 takes up the duality of strengthening and relaxation. This duality is studied in optimization under the guise of Lagrangean and surrogate duality, which finds a strong relaxation by searching over a parameterized family of relaxations. Both are defined only for inequality constraints, but both are special cases of a general relaxation duality that can be developed for a much wider range of problems.

A particularly promising maneuver is to combine constraint programming's approach to inference with optimization's approach to relaxation. When formulating a problem, the constraint programmer often identifies a group of constraints that show special structure and represents them with a single *global* constraint, such as `all-different`, `element` or `cumulative`. The optimizer often relaxes a problem by transforming it to an instance of a specially structured class of problems that can be solved to optimality, such as a linear programming problem. Both of these techniques are key to the success of the respective fields. There is a natural way to combine them: design relaxations of the sort used in optimization for global constraints of the sort used in constraint programming [33]. This idea is illustrated in Section 4 by presenting relaxations for element constraints.

The paper concludes by suggesting issues for future research.

1 A Motivating Example

A small example can illustrate how dualities can operate in a constraint satisfaction and in an integer programming setting as well as in a combined mode. One example can illustrate only a few of the relevant ideas, but it will help make the discussion to follow more concrete.

Consider the following optimization problem:

$$\begin{array}{ll} \text{minimize} & 4x_1 + 3x_2 + 5x_3 \\ \text{subject to} & 4x_1 + 2x_2 + 4x_3 \geq 17, \\ & \texttt{all-different}\{x_1, x_2, x_3\}, \\ & x_j \in \{1, \ldots, 5\}. \end{array} \tag{1}$$

The set $D_j = \{1, \ldots, 5\}$ is the initial *domain* of each of the variables x_j. The optimal solution is $(x_1, x_2, x_3) = (2, 3, 1)$ with optimal value 22.

We will solve the problem by constraint satisfaction methods, by integer programming, and finally by a combined approach. The particular methods illustrated are not the best available for either constraint satisfaction or integer programming. They are chosen because they help illustrate how methods may be combined. The intent is not to compare the performance of constraint programming and integer programming but to present a combined approach.

Node	D_1 / D_2 / D_3	z	*Value*	*Branches*
1.	12345 / 12345 / 12345	∞	12..60	
2.	1 / 2345 / 2345	∞	20..44	$D_1 = \{1\}$
3.	1 / 2 / 345	∞	25..35	$D_2 = \{2\}$
4.	1 / 2 / 3	∞	25	$D_3 = \{3\}$
5.	1 / 2 /	25	∞	$D_3 = \{4, 5\}$
6.	1 / 3 / 2	25	23	$D_2 = \{3, 4, 5\}$
7.	23 / 123 / 12	23	16..22	$D_1 = \{2, 3, 4, 5\}$
8.	2 / 3 / 1	23	22	$D_1 = \{2\}$
9.	3 / 1 /	22	∞	$D_1 = \{3, 4, 5\}$

Table 1: Solution of a constraint satisfaction problem by branching and domain reduction. The "value" shown is the value or domain of $4x_1 + 3x_2 + 5x_3$ at a leaf node of the search tree. A value of ∞ indicates the lack of a feasible solution.

1.1 Constraint Satisfaction

A constraint satisfaction method can solve (1) by solving the feasibility problem

$$4x_1 + 3x_2 + 5x_3 < z, \tag{2}$$
$$4x_1 + 2x_2 + 4x_3 \geq 17, \tag{3}$$
$$\texttt{all-different}\{x_1, x_2, x_3\}, \tag{4}$$
$$x_j \in \{1, \ldots, 5\}. \tag{5}$$

Initially $z = \infty$. Each time a feasible solution is found, the search continues with z set to the value of the solution. A possible search tree is shown in Table 1. The nodes are traversed in the depth-first order shown. At node 1, where $D_1 = D_2 = D_3 = \{1, 2, 3, 4, 5\}$, the search branches on x_1. This creates a subproblem at node 2 by setting $D_1 = \{1\}$, and one at node 7 by setting $D_1 = \{2, 3, 4, 5\}$. (Other branching schemes are possible.) Similarly, the search branches on x_2 at node 2, creating nodes 3 and 6.

Because $x_1 = 1$ at node 2, setting $x_2 = 1$ or $x_3 = 1$ would be inconsistent with the `all-different` constraint. The domains of x_1, x_2 are therefore reduced to $D_1 = D_2 = \{2, 3, 4, 5\}$. There are several different *domain reduction* algorithms that remove domain elements that are inconsistent with `all-different`, varying in degree of efficiency and completeness [32, 40].

A feasible solution is found at node 4 that permits one to set $z = 25$. At node 5 another type of domain reduction, based on maintaining *bounds consistency*, can be applied [32]. It infers from (2) that

$$x_3 \leq \tfrac{1}{5}(z - 4\min D_1 - 3\min D_2) \tag{6}$$

where $\min D_j$ is the smallest element in D_j, and similarly for x_1 and x_2. This changes neither $D_1 = \{1\}$ nor $D_1 = \{2\}$ but reduces $D_3 = \{4, 5\}$ to the empty set because (6) implies that $x_3 \leq 3$. In general, the implications of one constraint are *propagated* to other constraints by means of a *constraint store*, which in the present case consists of the reduced domains that are inferred from one constraint and made available to others.

The subproblem at node 5 is therefore infeasible. Node 6 reveals a solution of value $z = 23$, and finally the optimal value $z = 22$ is discovered at node 8, and the search completes its proof of optimality in 9 nodes. Constraint satisfaction therefore solves (1) with a combination of search (branching) and inference (domain reduction).

1.2 Integer Programming

Node	$\bar{z}$	*Value*	(x_1, x_2, x_3)	(y_{12}, y_{13}, y_{23})	*Branches*
1.	∞	20	$(3,1,1)$	$(0,0,\frac{1}{5})$	
2.	∞	21	$(2\frac{1}{2},2,1)$	$(\frac{1}{10},0,0)$	$y_{23} = 0$
3.	∞	$21\frac{2}{3}$	$(2,2\frac{1}{3},1\frac{1}{3})$	$(\frac{4}{15},\frac{1}{15},0)$	$x_1 \leq 2$
4.	∞	∞			$x_2 \leq 2$
5.	∞	22	$(2,3,1)$	$(1,0,0)$	$x_2 \geq 3$
6.	22	23	$(3,2,1)$	$(0,0,0)$	$x_1 \geq 3$
7.	22	21	$(2,1,2)$	$(0,\frac{1}{5},1)$	$y_{23} = 1$
8.	22	25	$(3,1,2)$	$(0,0,1)$	$y_{13} = 0$
9.	22	$21\frac{1}{2}$	$(1\frac{1}{2},1,2\frac{1}{2})$	$(\frac{1}{10},1,1)$	$y_{13} = 1$
10.	22	22	$(1,1,3)$	$(\frac{1}{5},1,1)$	$x_1 \leq 1$
11.	22	26	$(2,1,3)$	$(0,1,1)$	$x_1 \geq 2$

Table 2: Solution of an integer programming problem by branching, relaxation, and cutting plane generation.

Integer programming exploits the search/inference duality along with a duality of strengthening and relaxation. Due to the restricted vocabulary of integer programming, however, the model is more complex.

Variables y_{jk} $(j < k)$ are introduced to enforce the `all-different` constraint, with $y_{jk} = 1$

when $x_j < x_k$ and $y_{jk} = 0$ when $x_j > x_k$.

$$\begin{array}{rll}
\text{minimize} & 4x_1 + 3x_2 + 5x_3 & (7)\\
\text{subject to} & 4x_1 + 2x_2 + 4x_3 \geq 17, & (8)\\
& x_j \leq (x_k - 1) + 5(1 - y_{jk}), \ \ \forall j,k \text{ with } j < k, & (9)\\
& x_k \leq (x_j - 1) + 5y_{jk}, \ \ \forall j,k \text{ with } j < k, & (10)\\
& 1 \leq x_j \leq 5 \text{ and } x_j \text{ integer}, \ \ j = 1,2,3, & \\
& y_{jk} \in \{0,1\}, \ \ \forall j,k \text{ with } j < k. &
\end{array}$$

Constraints (9)–(10) illustrate the ubiquitous "big-M" constraint of integer programming; here $M = 5$. When $y_{jk} = 1$, constraint (9) is enforced while (10) is vacuous, and vice-versa when $y_{jk} = 0$.

There are other and better integer programming models for this particular problem. Model (7)–(10) is used here in order to illustrate the big-M constraints and how they may be avoided in general.

The problem is again solved by branching. This time, however, a *continuous relaxation* of the problem is solved each node of the search tree. The continuous relaxation of (7)–(10), which is solved at the root node, is obtained by deleting the integrality constraints on x_j and replacing $y_{jk} \in \{0,1\}$ with $0 \leq y_{jk} \leq 1$. The optimal value of the resulting linear programming relaxation provides a lower bound on the optimal value of (7)–(10).

In integer programming, inference often takes the form of *cutting plane* generation. Cutting planes are inequalities that are satisfied by every integer solution of the continuous relaxation but possibly violated by some noninteger solutions. By "cutting off" noninteger solutions, cutting planes can provide a tighter bound when added to the constraint set of the continuous relaxation. In this case one might add the cutting planes,

$$\begin{array}{c} x_1 + x_2 + x_3 \geq 5 \\ 2x_1 + x_2 + 2x_3 \geq 9 \end{array} \qquad (11)$$

(These cutting planes are derived from the inequality constraints alone.) There is a vast literature describing how cutting planes may be generated for highly structured problem classes, such as traveling salesman, job shop scheduling, set covering, set packing, and a host of other problems.

A search tree for the problem represented by (7)–(10) and (11) appears in Table 2. The search branches on variables that have nonintegral values in the continuous relaxation, in the order x_1, x_2, x_3, y_{12}, y_{13}, y_{23}. At node 1, $y_{23} = \frac{1}{5}$, and one branches by setting $y_{23} = 0$ and $y_{23} = 1$, creating nodes 2 and 7. At node 2, $x_1 = 2\frac{1}{2}$, and the branches are defined by $x_1 \leq 2$ and $x_1 \geq 3$. The branching constraints are added to the relaxation at each node. For example, $y_{23} = 0$ is added at node 2. The first feasible (i.e., integral) solution is found at node 5. Its optimal value provides an upper bound $\bar{z} = 22$ on the optimal value of the original problem, i.e., the constraint $4x_1 + 3x_2 + 5x_3 \leq 22$ is added to the problem at node 5. At node 6 the value of the relaxation is 23, so further branching at node 6 cannot lead to an optimal solution, and the tree is pruned at this point. This bounding mechanism provides the name, *branch-and-bound,* for this particular kind of search.

Branch-and-bound search relies on the interplay of the search/inference and strengthening/relaxation dualities. It searches by branching and draws inferences by cutting plane generation (which results in *branch and cut*). Branching likewise creates strengthenings of the problem by adding constraints. A relaxation of this strengthened problem is created by dropping integrality requirements and applying a linear programming algorithm.

1.3 A Combined Approach

Node	D_1	D_2	D_3	$\bar{z}$	*Value*	(x_1, x_2, x_3)	*Branches*
1.	12345	12345	12345	∞	20	$(3, 1, 1)$	
2.	12345	2345	12345	∞	21	$(2\frac{1}{2}, 2, 1)$	$x_2 \geq 2$
3.	12	2345	12345	∞	$21\frac{1}{2}$	$(2, 2, 1\frac{1}{2})$	$x_1 \leq 2$
4.	2	345	1	∞	22	$(2, 3, 1)$	$x_3 \leq 1$
5.	$\varnothing$			22	∞		$x_3 \geq 2$
6.	$\varnothing$			22	∞		$x_1 \geq 3$
7.	$\varnothing$			22	∞		$x_3 \geq 2$

Table 3: Solution of a constraint satisfaction problem by branching, domain reduction, relaxation, and cutting plane generation.

The respective advantages of constraint satisfaction and integer programming are easily combined in this case. Domain reduction and cutting plane generation are simply two forms of inference, and both can be used. In fact, domain reduction can be applied to the cutting planes (11) as well as the original constraint (8). The advantages of relaxation are also available. The integer programming relaxation was obtain by dropping integrality constraints from the large model (7)–(10). But the largest part of this model, (9)–(10), adds little to the quality of the relaxation. One can use the original model (1) as a problem statement and for feasibility checks, but create a continuous relaxation that consists of (7)–(8), the cutting planes (11) and the bounds $1 \leq x_j \leq 5$. Because the relaxation is distinguished from the model, both are more succinct.

A search tree appears in Table 3. At each node constraint propagation is first applied to the original problem, and if successful, the bounds in the relaxation are adjusted accordingly (we add the branching constraints both to the original model and the relaxation). As soon as a feasible solution is found, a constraint is added to the problem indicating the bound on the solution and it is updated as necessary.

The search branches on the alternatives $x_1 \geq 2$, $x_3 \geq 2$ at node 1 because the solution of the relaxation sets $x_2 = x_3 = 1$; the alternatives are obviously exhaustive. At node 2 the search branches on a nonintegral variable x_1. Because the solution of the relaxation at node 3 is integral and satisfies `all-different`, it is feasible, and no further branching is needed.

Due to the combined effects of constraint propagation and relaxation, a combined approach may produce a search tree is smaller than those that result from constraint programming or integer programming methods. In addition processing may be faster at each node than in integer programming, because the relaxation is smaller. There may also be nodes at which one need not solve the relaxation, because constraint propagation alone (which is often faster than solving an LP) may determine that the problem is infeasible.

2 Duality of Search and Inference

A *search* method examines possible values of the variables until an acceptable solution is found. An *inference* method attempts to derive a desired implication from the constraint set. Popular search methods include branching (which examines partial solutions) and local search heuristics (which examine complete solutions). Inference methods include cutting plane methods (in which inequalities are inferred) and domain reduction (in which smaller domains are inferred).

Search and inference tend to work best in combination. Search alone may happen upon a good solution early in the process, but it must examine many other solutions before determining that it is good. Inference alone can rule out whole families of solutions as inferior, but this is not the same as finding a good solution. Working together, search and inference can find and verify good solutions more quickly.

As illustrated above, a common strategy for running search and inference in parallel is to use inference in the context of a branching search. Constraint satisfaction infers smaller domains, for instance by maintaining consistency. Smaller domains result in less branching. The example illustrates the maintenance of bounds consistency for inequalities and hyperarc consistency [32] for `all-different` constraints. The cutting planes of optimization are designed to strengthen continuous relaxations but can reduce domains as well, for example if one maintains bounds consistency for them.

Combining search and inference also provides an effective way to exploit problem structure. Inferences drawn from specially structured constraints can reveal which regions of the solution space are unproductive and need not be examined. This is discussed first below. Finally, the interaction of search and inference can be interpreted as a formal duality. This leads to a link between nogoods and Benders decomposition as well as a general approach to sensitivity analysis.

2.1 Inference and Structure

One advantage of using inference in the context of search is that it can exploit special structure. If the problem or some part if it exhibits a pattern that has been closely analyzed offline in order to identify strong implications, these implications can be generated quickly. The practical success of optimization and constraint satisfaction owes much to this strategem.

The two fields have developed different and complementary approaches to recognizing structure. Constraint programmers identify sets of constraints, at the modeling stage, that can be recognized as a single global constraint (e.g., [6, 11, 12, 39, 41, 42]). The `cumulative` constraint, for example, requires that a set of tasks be scheduled so that, at any given time, their total consumption of resources is within bounds. A variety of scheduling problems are special cases of this general pattern. When they are formulated as such, the solver can apply domain reduction procedures that deal with the constraints *globally* rather than one at a time, thus resulting in much greater reduction of domains.

In optimization, the modeler typically identifies a problem as an instance of a class for which solution methods have been designed, such as linear programming, network flow, or 0–1 programming problems. Beyond this point the recognition of structure is often automated as part of the solution algorithm. The problem is scanned for opportunities to generate knapsack cuts, fixed charge cuts, covering inequalities, etc. In some cases substructures are identified, as for example subgraphs in a traveling salesman problem that give rise to *comb* inequalities. Another difference with the constraint satisfaction community is that constraint generation is aimed at strengthening a continuous relaxation rather than raising the degree of consistency of the constraint set. Interestingly, for a brief period in the early days of operations research, optimizers used constraints that were unrelated to the continuous relaxation. They were part of the *implicit enumeration* schemes of that day (e.g., [19]). Perhaps such constraints have since been neglected because the community, unaware of the theory of consistency, has not had a clear understanding of how they might accelerate search.

One of the most impressive traits of the human mind is its pattern-recognition ability. The constraint satisfaction approach uses this ability to identify structure primarily in the modeling stage. Optimization uses it primarily in the design of the solution algorithms that automatically detect patterns. To restrict oneself to one approach or the other seems a mistake. The modeler's insight into the practical situation should be used, as should the mathematician's analysis when the problem is sufficiently stylized to apply it.

2.2 Inference Duality in Optimization

One way to capture the duality of search and inference in a more rigorous setting is to state it as a formal optimization duality. For this purpose a general optimization problem might be written

$$\begin{array}{ll} \underset{x \in D}{\text{minimize}} & f(x) \\ \text{subject to} & x \in S \end{array} \tag{12}$$

where S is the feasible set and $D = D_1 \times \ldots \times D_n$ the domain. This can be viewed as a search problem: find an $x \in S \cap D$ that minimizes $f(x)$. The inference dual is

$$\begin{array}{ll} \text{maximize} & z \\ \text{subject to} & (x \in S) \overset{D}{\rightarrow} (f(x) \geq z) \end{array} \tag{13}$$

where the arrow indicates implication: for all $x \in D$, if $x \in S$ then $f(x) \geq z$. If an optimal value exists for (12), it is the same as the optimal value of (13). So optimization can be viewed as an inference problem: what is the tightest lower bound on $f(x)$ one can infer from $x \in S$?

The optimization literature has closely studied inference duality in the special case of linear programming. Here (12) becomes

$$\begin{array}{ll} \underset{x \in \mathbb{R}^n}{\text{minimize}} & cx \\ \text{subject to} & Ax \geq b, \ x \geq 0 \end{array} \tag{14}$$

where A is an $m \times n$ matrix. The dual problem is to infer the strongest possible inequality $cx \geq z$ (i.e., the tightest lower bound z) from $Ax \geq b$, $x \geq 0$. A fundamental result of linear programming (the Farkas Lemma) states that if $Ax \geq b$, $x \geq 0$ is feasible, it implies $cx \geq z$ if and only if some nonnegative linear combination $uA \geq ub$ of $Ax \geq b$ dominates $cx \geq z$. That is, $uA \leq c$ and $ub \geq z$ for some $u \geq 0$. So the dual problem can be written

$$\begin{array}{ll} \underset{u \in \mathbb{R}^m}{\text{maximize}} & ub \\ \text{subject to} & uA \leq c, \ u \geq 0 \end{array} \tag{15}$$

This is the classical linear programming dual. It has the same (possibly infinite) optimal value z^* as (14) unless both (14) and (15) are infeasible. The solution u of the dual problem can be viewed as encoding a *proof* that $cx \geq z^*$, because it specifies a linear combination of $Ax \geq b$ that dominates $cx \geq z^*$.

Linear programming has the convenient property that a solution of the dual always has polynomial length (i.e., linear programming belongs to both NP and co-NP). A proof of optimality can in general be exponentially long. For example, if x in (14) is restricted to be integer, so that (14) becomes an integer programming problem, then the inference dual can no longer be written in the form (15). The dual must be solved by a proof of optimality that most commonly takes the form of an exhaustive search tree—which has exponential size in general.

A major benefit of inference duality is that it provides a scheme for sensitivity analysis. This kind of analysis is very important in practice because it indicates how the solution would be affected by perturbations of the problem data. It allows one to focus on data that really matter.

Up to a point, sensitivity analysis is straightforward. Given an optimal solution of the primal problem (12), one can analyze under what data alterations this solution remains feasible. But this says nothing about whether it remains optimal, and this is where the inference dual comes into play. Because a solution of the inference dual is a proof, one can analyze under what data perturbations the proof remains valid and the solution therefore remains optimal (assuming it remains feasible as well).

This scheme works out nicely in linear programming. Let x^* be an optimal solution of (12) and u^* an optimal solution of the dual problem (15). Let (14) be perturbed so that it minimizes $(c + \Delta c)x$ subject to $(A + \Delta A)x \geq (b + \Delta b)$. Obviously x^* remains feasible if $\Delta A x^* \geq \Delta b$. But it remains optimal as well if u^* remains a valid proof that $(c+\Delta c)x \geq z^*$; i.e., if $u^* \Delta A \leq \Delta c$ and $u^* \Delta b \geq z^*$.

Until very recently the optimization community has approached the sensitivity question for discrete problems in a different fashion, using the concepts of value function, super-additive duality, etc. These approaches tend to make sensitivity analysis computationally very difficult. Both optimization and constraint satisfaction problems could benefit from the inference duality approach. The branch-and-bound tree for an integer programming problem, for example, can be analyzed to determine under what problem perturbations the tree remains a proof of optimality [13, 23, 24]. If branching and domain reduction prove a problem instance to be infeasible, one can examine for what problem perturbations this proof remains valid.

More generally, a solution of the inference dual provides an *explanation* (in the form of a proof) for why a solution is optimal, or why the problem is infeasible. In practice, an explanation of the solution could be more valuable than the solution itself. Perhaps methods can be developed to reduce this proof to its bare essentials, delineating as clearly as possible the reason for optimality or infeasibility. These ideas remain largely unexplored.

2.3 Nogoods and Benders Decomposition

If in the midst of search a trial solution is found to be unsatisfactory, the reasons for its failure can be analyzed. This analysis may lead to a constraint that rules out many solutions that fail for the same reason. By adding this constraint to the problem one can avoid unnecessary search. Such a constraint is a *nogood*, a well-known idea in the constraint satisfaction literature [45]. A nogood is a way of learning from one's mistakes. It combines search and inference in a particular way: the inferred constraints (nogoods) are occasioned by the discovery of bad solutions.

A related idea has evolved in the optimization literature. One way to combine search and inference is to search values of *some* of the variables and use inference to project the constraints onto these variables. Let the variables of (12) be partitioned as follows.

$$\begin{array}{ll} \underset{x \in D_x,\, y \in D_y}{\text{minimize}} & f(x,y) \\ \text{subject to} & (x,y) \in S \end{array} \tag{16}$$

We will search over values of x. If we examine a particular value $\bar{x}$, we can find optimal values for the remaining variables subject to $x = \bar{x}$. This poses the *subproblem*

$$\begin{array}{ll} \underset{y \in D_y}{\text{minimize}} & f(\bar{x},y) \\ \text{subject to} & (\bar{x},y) \in S \end{array} \tag{17}$$

The variables are partitioned in such a way that the subproblem has special structure that makes it easier to solve. The variables may decouple, for example.

The next step is to solve the inference dual of the subproblem by generating a proof of optimality for its optimal solution $y_{\bar{x}}$. This proof might take the form of a branching tree. By examining the conditions under which this proof is valid, we may be able to define a function $B_{\bar{x}}(x)$ that provides a lower bound on the optimal value of (16) for a given value of x. Two examples of this are provided below. Obviously $B_{\bar{x}}(\bar{x}) = f(\bar{x}, y_{\bar{x}})$, but even when x

has some value other than $\bar{x}$, it may be possible to determine what kind of lower bound the dual proof still provides. If z is the objective function value of (16), this analysis yields the nogood $z \geq B_{\bar{x}}(x)$. It states that there is no point in examining solutions x for which the objective function value is less than $B_{\bar{x}}(x)$. The *master problem* minimizes the objective function subject to nogoods that have been accumulated so far:

$$\begin{array}{ll} \underset{x \in D_x}{\text{minimize}} & z \\ \text{subject to} & z \geq B_{x^k}(x), \quad k = 1, \ldots, K \end{array} \tag{18}$$

where $x^1, \ldots, x^K$ are the nogoods generated so far. Each time the master problem is solved, the solution $\bar{x}$ generates another nogood. The nogoods in effect project the constraint set onto the variables x. It is usually unnecessary to generate all of the nogoods that define the projection, because the process stops when a nogood is satisfied by the previous $\bar{x}$.

When this strategy is applied to a problem with the following form, the result is *Benders decomposition.*

$$\begin{array}{ll} \underset{x \in D_x, y \in \mathbb{R}^n}{\text{minimize}} & f(x) + cy \\ \text{subject to} & g(x) + Ay \geq b \end{array} \tag{19}$$

The subproblem is a linear programming problem.

$$\begin{array}{ll} \underset{y \in \mathbb{R}^n}{\text{minimize}} & f(\bar{x}) + cy \\ \text{subject to} & Ay \geq b - g(\bar{x}) \end{array} \tag{20}$$

The dual solution $u_{\bar{x}}$ of (19) proves bound $z \geq u_{\bar{x}}b - u_{\bar{x}}g(\bar{x})$. Because the proof remains valid when x has values other than $\bar{x}$, we have the nogood $z \geq B_{\bar{x}}(x) = u_{\bar{x}}b - u_{\bar{x}}g(x)$, also known as a *Benders cut.*

When the subproblem (17) is solved by branching, one may be able to construct a boolean formula $P(x)$ such that the branching tree remains a proof of optimality of $y_{\bar{x}}$ whenever $P(x) = 1$. It is shown in [29], for example, that in integer programming the branching proof can be viewed as encoding a resolution proof whose premises are implied by constraints that are violated at the leaf nodes of the tree. The violated constraints imply the premises when $x = \bar{x}$. One can let $P(x) = 1$ for all values of x for which these constraint continue to imply the premises. Then $z \geq B_{\bar{x}}(x) = f(\bar{x}, y_{\bar{x}})P(x)$ is a valid nogood (if $z \geq 0$).

Nogoods can be combined with branching. Suppose that at a given node of the branching tree, constraint propagation or cutting planes prove infeasibility, or more generally prove $z \geq \underline{z}$ where $\underline{z} = \infty$ in the case of infeasibility. One can view this proof as solution of the inference dual of a subproblem containing the variables that have not yet been fixed at that node. Then one might derive a nogood $z \geq B_{\bar{x}}(x)$, where x is the vector of variables that have been fixed. This nogood can serve as a valid constraint throughout the rest of the tree search. Thus at each node one can generate complementary constraints: constraints involving the unfixed variables by means of constraint propagation and cutting plane methods, and nogoods involving the fixed variables.

Again there is cross-fertilization. The optimization community has apparently never used nogoods in branching search. The constraint satisfaction community has apparently never used generalized Benders decomposition as a means to generate nogoods, although Beringer and de Backer have done related work [1, 7]. The ability of Benders decomposition to exploit structure could give new life to the idea of a nogood, which has received limited attention in practical algorithms.

3 Duality of Strengthening and Relaxation

A *strengthening* of a problem shrinks the feasible set, and a *relaxation* enlarges it. Solving a strengthened minimization problem provides an upper bound on the optimal value of the original problem. Relaxing the problem provides a lower bound.

The interplay of strengthening and relaxation is an old theme in optimization that goes under the name of *primal-dual* methods. These appear, for example, in dual-ascent and other Lagrangean methods for discrete optimization, out-of-kilter and related methods for network flow problems [4], and the primal-dual simplex method for linear programming. All of these exploit the same formal duality, which is defined below. Branch-and-bound search can be regarded as a primal-dual method in a somewhat different sense that will also be discussed.

Properly chosen strengthenings and relaxations may be much easier to solve than the original problem. Solving several of them and choosing the tightest bounds that result may therefore provide a practical way of bracketing the optimal value of a problem that cannot be solved to optimality.

As noted earlier, enumeration of strengthenings usually takes the form of branching or local search. It is less obvious how to enumerate relaxations of a problem, but a clever method has evolved over the years: one *parameterizes* relaxations. Each parameter setting yields a different relaxation. The problem of finding parameters that yield the tightest bound might be called the *relaxation dual* problem. Several well-known dualities in optimization are special cases, including the linear programming dual, the Lagrangean dual, and the surrogate dual.

These classical duals, however, apply only to problems with inequality constraints. The general relaxation dual may provide a key to relaxing constraints that take other forms. This is particularly important for bringing the advantages of relaxation to constraint satisfaction methods, which permit a much broader repertory of constraints than the inequality constraints of mathematical programming. The first section below suggests how this might be done.

Branch-and-bound search dualizes strengthening and relaxation in a different way. The second section suggests how generalization of this idea can also result in new methods.

3.1 Relaxation Duality

We begin by defining strengthening and relaxation more carefully. The definition stated above assumes that the objective function in a strengthening or relaxation is the same as in the original problem. It need not be. The following problem is a *strengthening* of (12) in a more general sense if $S' \subset S$ and $f'(x) \geq f(x)$ for $x \in S'$.

$$\begin{array}{ll} \underset{x \in D}{\text{minimize}} & f'(x) \\ \text{subject to} & x \in S' \end{array} \tag{21}$$

Problem (21) is a *relaxation* of (12) if $S' \supset S$ and $f(x) \leq f'(x)$ for $x \in S$. Equivalently, let the *epigraph* E of an optimization problem (12) be the set $\{(z, x) \mid z \geq f(x),\ x \in S\}$. A strengthening's epigraph is a subset of E, and a relaxation's epigraph is a superset.

Suppose that a family of relaxations is parameterized by $\lambda \in \Lambda$, so that $f'(x) = f(x, \lambda)$ and $S' = S(\lambda)$. Each relaxation is written

$$\theta(\lambda) = \min_{x \in D}\{f(x, \lambda) \mid x \in S(\lambda)\} \tag{22}$$

This is a valid relaxation if:

$$\begin{array}{l} S(\lambda) \supset S, \text{ all } \lambda \in \Lambda \\ f(x, \lambda) \leq f(x), \text{ all } x \in S,\ \lambda \in \Lambda \end{array} \tag{23}$$

The problem of finding a relaxation that gives the tightest lower bound is the *relaxation dual*,

$$\underset{\lambda \in \Lambda}{\text{maximize}} \quad \theta(\lambda) \tag{24}$$

The classical *Lagrangean relaxation* replaces the objective function with a lower bound that is obtained by penalizing infeasible solutions and perhaps rewarding feasible ones. It is defined only when the constraints have inequality form, so that $S = \{x \mid g_i(x) \leq 0,\ i \in I\}$. It is obtained by setting $f(x, \lambda) = f(x) + \sum_{i \in I} \lambda_i g_i(x)$ and $S(\lambda) = D$ for $\lambda \geq 0$. Note that the feasible set is the same for all λ. This is a valid relaxation because clearly $S(\lambda) \supset S$, and $f(x) + \sum_{i \in I} \lambda_i g_i(x) \leq f(x)$ for all $\lambda \geq 0$ and all $x \in S$ (so that $g_i(x) \leq 0$). In this case the relaxation dual is the *Lagrangean dual*, which is widely used in integer and nonlinear programming to obtain bounds.

The *surrogate relaxation* leaves the objective function untouched but replaces the inequality constraints with a nonnegative linear combination of those constraints. It is obtained by setting $f(x, \lambda) = f(x)$ and $S(\lambda) = \{x \mid \sum_{i \in I} \lambda_i g_i(x) \geq 0\}$, with $\lambda \geq 0$. The relaxation dual in this instance is the *surrogate dual*, which can also be used to obtained bounds for integer and nonlinear programming. The Lagrangean and the surrogate dual of a linear programming problem are both equivalent to the linear programming dual.

A relaxation dual can be defined for a much wider range of problems than those involving inequality constraints. It is necessary only that the relaxation observe the formal properties (23). This can be illustrated by the traveling salesman problem. The traditional continuous relaxation requires that the problem be written with inequality constraints, resulting in a long problem statement (exponentially long in the most popular formulation). However, the problem can be written very succinctly as follows.

$$\begin{array}{ll} \underset{x \in D}{\text{minimize}} & \sum_i c_{y_i, y_{i+1}} \\ \text{subject to} & \texttt{all-different}\{y_1, \ldots, y_n\} \end{array}$$

where $y_{n+1} = y_1$. Here y_i is the i-th city visited and c_{jk} the cost on arc (j,k). One can of course write a relaxation for this formulation by reverting to the inequality model and relaxing the integrality constraints. But this is not a practical option when an inequality formulation of the problem at hand is unavailable, too large, or has a weak relaxation. A generalized Lagrangean relaxation can perhaps accommodate such cases.

In the case of the traveling salesman problem, a generalized Lagrangean relaxation might be given by

$$f(x,\lambda) = \sum_i c_{y_i,y_{i+1}} + \sum_j \lambda_j (N_j - 1) \tag{25}$$

and $S(x,\lambda) = \{1,\ldots,n\}$, where N_j is the number of x_i's equal to j [26]. Because (25) can be written

$$f(x,\lambda) = \sum_i c_{y_i,y_{i+1}} + \sum_i (\lambda_{y_i} - \lambda_i)$$

the value $\theta(\lambda)$ can be readily computed by dynamic programming. The dual (24) can be solved by subgradient optimization, because $(N_1 - 1, \ldots, N_n - 1)$ is a readily available subgradient.

In other types of problems a concept from constraint satisfaction may help provide a useful relaxation of the constraint set. The *dependency graph* G for a problem (12) indicates the extent to which variables decouple [45]. It contains a vertex for each variable and an edge (i,j) when variables x_i and x_j occur in the same constraint or in the same term of the objective function (which we may, for simplicity, assume to be a sum of terms). If vertices (along with adjacent edges) are removed from G in order $1,\ldots,n$, the *induced width* of G with respect to this ordering is the maximum degree of a vertex at the time it is removed.

Problem (12) can be solved by nonserial dynamic programming [8] in time that is exponential in the induced width of G. Although the induced width is normally too large for this to be practical, relaxations can be defined for which it is small. This might be done by removing several arcs from G to obtain $G(\lambda)$, where λ is a list of the arcs removed. For each arc (x_i, x_j) removed, replace each constraint containing both x_i and x_j with two projections of the constraint. (The objective function can be analogously treated.) The projections are obtained by projecting the constraint onto all of its variables except x_i and onto all of its variables except x_j. Once the projections are computed, resulting problem has dependency graph $G(\lambda)$.

The relaxed set $S(x,\lambda)$ is now defined to be the projected problem for $G(\lambda)$, and $\theta(\lambda)$ is computed by nonserial dynamic programming. The dual problem (23) might be attacked by local search methods over the space of λ's.

These represent only two examples of how discrete relaxations might be parameterized and bounds obtained by solving a relaxation dual. The potential of this approach is largely unexplored.

3.2 Searches that Combine Strengthening and Relaxation

The most popular strategy for combining strengthening and relaxation in a search procedure is to enumerate strengthenings and solve a relaxation of each. In integer programming, for instance, one might enumerate strengthenings in a branch-and-bound tree and solve the continuous relaxation of the strengthened problem at each node. A rationale for this strategy is that it hedges against the liabilities of both strengthening and relaxation: strengthenings may not be easy to solve until they become very strong (i.e., almost all variables are fixed), and whereas a continuous relaxation may be easy to solve, it may also be very weak. By solving a relaxation at each node of a search tree, one solves an easy problem that may nonetheless be a relatively strong relaxation because several variables have been fixed. Bounds derived from the relaxations can be used in a branch-and-bound scheme.

This represents only one way that strengthening and relaxation can interact. There are others. For example, the reverse strategy is seldom recognized: one can solve strengthenings of a relaxation. The only requirement is that the relaxation remain an easy problem when strengthened, for example when variables are fixed. This is normally the case. A feasible solution is found when the solution of a strengthening is feasible in the original problem. One backtracks whenever a feasible solution is found, or it can be determined that no solution of the current strengthening is feasible in the original problem. Bounding can be used as before.

In integer programming, the reverse strategy is identical to the original strategy, because continuous relaxation and variable fixing are commutative functions. Fixing a variable in a continuous relaxation has the same effect as relaxing the problem after fixing that variable. Perhaps this is why the reverse strategy has not been noticed.

In general relaxation and strengthening do not commute. For example, if $x_1, x_2 \geq 0$, the constraint $x_1 x_2 \geq 4$ can be relaxed to $x_1 + x_2 \geq 4$ by writing a first-order Taylor series approximation at the point $(x_1, x_2) = (2, 2)$. The relaxation becomes $x_2 \geq 0$ when x_1 is fixed to, say, 4. Reversing the direction, fixing $x_1 = 4$ changes the nonlinear constraint to $x_2 \geq 1$, which relaxes to itself and is different from $x_2 \geq 0$.

Viewing search consciously as an interplay of strengthening and relaxation can therefore lead one to combine them in different ways and obtain new methods. The effectiveness of these new methods has yet to be tested.

4 Generating Relaxations via Inference

As mentioned earlier, the global constraints of constraint programming provide an opportunity to exploit structure not only for purposes of domain reduction, but for relaxation as well.

To clarify this point, it should be acknowledged that a global constraint is sometimes relaxed in order to compute reduced domains. The result is not, however, normally the sort of relaxation that is recommended here; namely, one that can be solved to optimality in order to obtain useful bounds, such as a linear programming relaxation.

It is true that domain reduction can itself be viewed as a process that generates a relaxation. It in effect derives “in-domain” constraints that restrict each variable to a reduced domain. The constraint programming community normally views in-domain constraints as comprising a *constraint store* that propagates the implications of one constraint to other constraints. But they can also be viewed as comprising a relaxation that is easily solved: merely choose one value from each domain [9]. It may even be practical to optimize the objective function subject to the in-domain constraints. But even in this case, the result is unlikely to provide a useful bound.

The desired sort of relaxation has usually been obtained in the form of cutting planes that are derived from inequality constraints. This imposes a severe limitation, because most useful global constraints are neither expressed nor easily expressible in inequality form. It is often possible, however, to derive linear inequality relaxations, as well as other soluble relaxations, from constraints other than inequalities. This has been done even in traditional operations research for disjunctions of linear inequalities [2, 3, 5]. This and more recent work are summarized in [27].

As an illustration, we present here continuous relaxations for element constraints, which are important due to their role in implementing variable subscripts. To highlight the overall strategy of attaching both domain reduction procedures and relaxations to a global constraint, we also analyze domain reduction for element constraints.

4.1 Discrete Variable Subscripts

Variable subscripts are rapidly becoming ubiquitous in modeling of combinatorial optimization problems. The `element` constraint is well known in the constraint programming world [47, 32] as a way of indexing discrete variables, but variable subscripts have now also been introduced in mathematical modeling languages such as AMPL [17, 18] and OPL [48]. While domain reduction ensuring arc- or hyperarc consistency is relatively simple for indexing over discrete variables, the case of continuous variables is much less explored.

The `element` constraint is written,

$$\texttt{element}(y, (v_1, \ldots, v_k), z). \tag{26}$$

In the simplest case, y is a single variable whose initial domain is $\{1, \ldots, k\}$, and $(v_1, \ldots, v_k)$ is a list of values. The variable z can be discrete or continuous. The constraint says that z must take the y-th value in the list.

An `element` constraint of this form implements a term with a variable subscript. A term of the form $c_{f(y)}$, where $f(y)$ is a function of the variable y, is implemented by imposing the constraint

$$\texttt{element}(y, (c_{f(1)}, \ldots, c_{f(k)}), z)$$

and replacing all occurrences of $c_{f(y)}$ with z. For example, if $y \in \{1, 2, 3, 4\}$ the term $c_{y,y+1}$ is replaced by z and the constraint

$$\texttt{element}(y, (c_{12}, c_{23}, c_{34}, c_{45}), z).$$

Because the simplest `element` constraint (26) contains only two variables, arc consistency is equivalent to full consistency. It is achieved in the obvious way. Let D_z, D_y be the current domains of z and y, respectively, and let $\bar{D}_z, \bar{D}_y$ be the new, reduced domains. Then the two rules, $\bar{D}_z = D_z \cap \{v_j \mid j \in D_y\}$ and $\bar{D}_y = D_y \cap \{j \mid v_j \in D_z\}$, in a fix-point iteration, will achieve arc consistency.

Indexing among values is just an instance of the general case of variables (as opposed to constants) with variable subscripts. Because `element` now contains $k+2$ variables, arc consistency does not imply hyperarc consistency. However, full hyperarc consistency can be obtained as follows (of which the rules above are a special case).

(a) The domain of z must be a subset of the combined domains of the variables x_j for which j belongs to the domain of y. So

$$\bar{D}_z = D_z \cap \bigcup_{j \in D_y} D_{x_j}.$$

(b) The domain of y is restricted to indices j for which the domain of z intersects the domain of x_j. Thus

$$\bar{D}_y = D_y \cap \{j \mid D_x \cap D_{x_j} \neq \emptyset\}.$$

(c) The domain of x_j can be restricted if j is the only index in the new domain $\bar{D}_y$ of y.

$$\bar{D}_{x_j} = \begin{cases} \bar{D}_z, & \text{if } \bar{D}_y = \{j\}, \\ D_{x_j}, & \text{otherwise.} \end{cases}$$

Example 1 Consider the `element` constraint

$$\texttt{element}(y, (x_1, x_2, x_3, x_4), z)$$

where initially the domains are:

$$\begin{aligned} D_z &= \{20, 30, 60, 80, 90\} \\ D_y &= \{1, 3, 4\} \\ D_{x_1} &= \{10, 50\} \\ D_{x_2} &= \{10, 20\} \\ D_{x_3} &= \{40, 50, 80, 90\} \\ D_{x_4} &= \{40, 50, 70\} \end{aligned}$$

Rules (a), (b) and (c) imply that the reduced domains are:

$$\begin{aligned} \bar{D}_z &= \{20, 30, 60, 80, 90\} \cap \{10, 40, 50, 70, 80, 90\} = \{80, 90\} \\ \bar{D}_y &= \{1, 3, 4\} \cap \{3\} = \{3\} \\ \bar{D}_{x_1} &= D_{x_1} \\ \bar{D}_{x_2} &= D_{x_2} \\ \bar{D}_{x_3} &= \bar{D}_z = \{80, 90\} \\ \bar{D}_{x_4} &= D_{x_4} \end{aligned}$$

Thus y is fixed to 3, which means that $x_3 = z$. The common domain of x_3 and z is the intersection of their original domains. □

4.2 Continuous Variable Subscripts

While the discrete cases above are well known, variable subscripts in continuous linear inequalities are far less explored. In this case, constraint propagation is replaced by cutting plane generation. Suppose for example that x is a continuous variable in the constraint $x_{f(y)} \geq \beta$. The inequality $z \geq \beta$ is inserted into the LP model along with additional constraints that define z. If the value of y is fixed, e.g, to 1, one adds the constraint $z = x_{f(1)}$. If the current domain of y is $\{1,2\}$, however, the constraint that defines z is a disjunction:

$$(x_{f(1)} = z) \vee (x_{f(2)} = z). \tag{27}$$

Although (27) cannot be added to a linear model, a linear relaxation of it, defined by cutting planes, can be used instead.

In general a subscripted variable $x_{f(y)}$ is represented by replacing it with z and the linear relaxation of the general disjunction

$$\bigvee_{i \in D_y} x_{f(i)} = z. \tag{28}$$

In order to be useful, the variables x_j must also have bounds, such as $0 \leq x_{f(j)} \leq m_{f(j)}$ for $j \in D_y$. We assume in what follows that $|D_y| \geq 2$.

If each upper bound is the same value m_0, we get the following valid inequalities for (28):

$$\sum_{i \in D_y} x_{f(i)} \leq z + (|D_y| - 1)m_0, \tag{29}$$

$$\sum_{i \in D_y} x_{f(i)} \geq z, \tag{30}$$

$$0 \leq x_{f(i)} \leq m_0, \quad i \in D_y, \tag{31}$$

$$0 \leq z \leq m_0. \tag{32}$$

The inequality (30) is a surrogate inequality [3] and the bounds (31)–(32) are from before.

Let $\mathcal{C}$ be the polyhedron defined by (28) and (31)–(32), and let $\mathcal{P}$ be the polyhedron defined by (29)–(32) It can be easily shown that $\mathcal{C} \subseteq \mathcal{P}$, since points in $\mathcal{C}$ are convex combinations of points where at least one of the x_i's is equal to z. Equations (29)–(30) follow immediately.

We note that the inequalities defining $\mathcal{P}$ are facets of the convex hull relaxation of $\mathcal{C}$. The $|D_y|+1$ points, $(0, m_0, \ldots, m_0, 0)$, $\ldots$, $(m_0, \ldots, m_0, 0, 0)$ and $(m_0, \ldots, m_0)$ in $\mathcal{C}$ are affinely independent and satisfy (29) at equality. Also, the $|D_y| + 1$ points $(m_0, 0, \ldots, 0, m_0)$, $\ldots$, $(0, \ldots, 0, m_0, m_0)$ and $(0, \ldots, 0)$ in $\mathcal{C}$ are affinely independent and satisfy (30) at equality. The bounds (31)–(32) are obviously facets of $\mathcal{C}$. It is shown in [25] that $\mathcal{P}$ is in fact the convex hull of the disjunction (28).

If the upper bounds differ the convex hull relaxation can be much more complex. In this case one can use a weaker and simpler relaxation by letting $m_0 = \max_i\{m_{f(i)}\}$ in (29) and (32) and replacing (31) with the actual bounds.

One can augment this relaxation with second relaxation. First write the disjunction (28) in the weaker form of two disjunctions,

$$\bigvee_{i \in D_y} (x_{f(i)} - z \geq 0), \tag{33}$$

$$\bigvee_{i \in D_y} (-x_{f(i)} + z \geq 0), \tag{34}$$

and replace each with the linear "elementary" relaxation described in [5]. This yields,

$$\sum_{i \in D_y} \frac{x_i}{m_{f(i)}} - \left(\sum_{i \in D_y} \frac{1}{m_{f(i)}} \right) z \geq -|D_y| + 1, \tag{35}$$

$$-\sum_{i \in D_y} \frac{x_i}{m_{f(i)}} + \left(\sum_{i \in D_y} \frac{1}{m_{f(i)}} \right) z \geq -|D_y| + 1, \tag{36}$$

When all upper bounds are the same, $m_0 = \max_i \{m_{f(i)}\}$, (35)–(36) become the following, which are dominated by (29)–(30):

$$\sum_{i \in D_y} x_{f(i)} \geq |D_y| z - (|D_y| - 1) m_0, \tag{37}$$

$$\sum_{i \in D_y} x_{f(i)} \leq |D_y| z + (|D_y| - 1) m_0. \tag{38}$$

But when the upper bounds differ, it is advantageous to use both (29)–(30) and (35)–(36) along with the upper bounds and (32), where $m_0 = \max_i \{m_{f(i)}\}$ in (29)–(30).

Example 2 The goal is to generate linear inequalities to represent $x_y \geq \beta$ in a linear programming solver, where the current domain of y is $\{1, 2\}$. Suppose initially that $0 \leq x_j \leq 5$ for $j = 1, 2$. Then one can generate the inequality $z \geq \beta$ and define z with the relaxation of (27). The latter is given by (29)–(32), which in this case is

$$0 \leq x_1 + x_2 - z \leq 5, \tag{39}$$

$$0 \leq x_1, x_2, z \leq 5, \tag{40}$$

$$0 \leq z \leq 5, \tag{41}$$

Thus the constraints $z \geq \beta$ and (39)–(41) are added to the LP model.

Now suppose the upper bounds are different: $0 \leq x_1 \leq 4, 0 \leq x_2 \leq 5$. The elementary relaxation in (35)–(36) becomes:

$$5x_1 + 4x_2 - 9z \leq 20,$$
$$5x_1 + 4x_2 - 9z \geq -20.$$

These inequalities are combined with $z \geq \beta$, (39), (41) and the bounds $0 \leq x_1 \leq 4, 0 \leq x_2 \leq 5$ in the LP model. $\square$

An alternative to the disjunctive relaxation discussed above would be to use a variant of the conventional "big-M" formulation. Equations (33)–(34) would then form a relaxation of (28) as

$$\begin{aligned} x_{f(i)} - z &\geq -M(1 - y_i), & i \in D_y, \quad (42)\\ -x_{f(i)} + z &\geq -M(1 - y_i), & i \in D_y, \quad (43)\\ \sum_{i \in D_y} y_i &= 1, & (44)\\ 0 \leq y_i &\leq 1, & \forall i \in D_y, \quad (45) \end{aligned}$$

where $M = \max_i \{m_{f(i)}\}$. Note that we introduce $|D_y|$ new continuous variables in this relaxation. The projection of (42)–(45) on $(x_{f(1)}, \ldots, x_{f(|D_y|)}, z)$ is very weak so as a relaxation the big-M formulation is not only more costly ($|D_y|$ new variables and $2|D_y| + 1$ new constraints) but is almost useless. Its use is in a search framework where y is needed for branching purposes, i.e., where the framework does not allow branching on constraints, e.g., on parts of a disjunction. Instead, the y's in the big-M formulation above are used to simulate that capability.

4.3 Incremental Cutting Plane Generation

Although useful on their own, the relaxations for variable subscripts described in the previous section are mainly intended for use within a branch-and-bound search. Due to branching and inference (such as constraint propagation), the domain of the indexing variable y will shrink as we descend in the search tree. This means that $x_{f(i)}$ should be removed from the equations when $i \notin D_y$ (by setting the corresponding coefficient to zero), and also that the coefficients $|D_y|$ and m_j need to be updated in equations (29)–(38) when D_y is modified.

This last comment is related to how the M's in a conventional "big-M" formulation are selected and handled. Ideally, they should be updated when the variable bounds change, to obtain the strongest possible relaxation, but often this seems to be neglected.

5 Future Research Directions

A number of research directions are identified in the foregoing. They may be summarized as follows.

- *Deciding what to relax.* Learn how to identify subsets of constraints that have a useful continuous relaxation.
- *Continuous relaxations for global constraints.* Find useful continuous relaxations for common global constraints.
- *Relaxation duals.* Use the idea of a relaxation dual to create discrete relaxations for common global constraints.

- *Sensitivity analysis.* Develop inference-based sensitivity analysis for problem classes by analyzing when problem perturbations leave the proof of optimality (or infeasibility) intact. Also, learn how to generalize this analysis so as to explain the solution.
- *Using nogoods in branch-and-cut search.* Investigate the possibility of using nogoods obtained by inference-based Benders decomposition as cuts that are complementary to the traditional cuts in branch-and-cut search.
- *Finding nogoods that exploit structure.* Use generalized Benders decomposition as a means to identify nogoods that exploit problem structure and perhaps thereby improve the utility of nogoods.
- *Strengthening and relaxation.* Experiment with new ways for combining strengthening and relaxation during search, for instance by solving strengthenings of a relaxation.
- *Unified solution technology.* Solve a wide variety of problems with a view to how the search/inference and strengthening/relaxation dualities may be exploited, with the aim of building a solution technology that unifies and goes beyond classical optimization and constraint satisfaction methods.

References

[1] Bruno De Backer and Henry Beringer. A CLP language handling disjunctions of linear constraints. In David S. Warren, editor, *Proceedings of the Tenth International Conference on Logic Programming*, pages 550–563, Budapest, Hungary, 1993. The MIT Press.

[2] E. Balas. Disjunctive programming: Cutting planes from logical conditions. *Nonlinear Programming 2*, pages 279–312, 1975.

[3] E. Balas. Disjunctive programming. In P. L. Hammer, E. L. Johnson, and B. H. Korte, editors, *Discrete Optimization II, 5*, pages 3–51, Amsterdam, 1979. Annals of Discrete Mathematics, North-Holland.

[4] M. S. Bazaraa, J. J. Jarvis, and H. D. Sherali. *Linear Programming and Network Flows.* Wiley, New York, 1990.

[5] N. Beaumount. An algorithm for disjunctive programs. *European Journal of Operational Research*, 48:362–371, 1990.

[6] Nicolas Beldiceanu and E. Contejean. Introducing global constraints in CHIP. *Mathematical and Computer Modelling*, 20(12):97–123, 1994.

[7] H. Beringer and B. De Backer. Combinatorial problem solving in constraint logic programming with cooperating solvers. In C. Beierle and L. Plümer, editors, *Logic Programming: Formal Methods and Practical Applications*, Studies in Computer Science and Artificial Intelligence, chapter 8, pages 245–272. Elsevier, 1995.

[8] U. Bertele and F. Brioschi. *Nonserial Dynamic Programming.* Academic Press, New York, 1972.

[9] A. Bockmayr and T. Kasper. Branch-and-infer: A unifying framework for integer and finite domain constraint programming. *INFORMS J. Computing*, 10(3):287 – 300, 1998.

[10] S. Bollapragada, O. Ghattas, and J. N. Hooker. Optimal design of truss structures by mixed logical and linear programming. *Operations Research, to appear*, 1995.

[11] Y. Caseau and F. Laburthe. Solving various weighted matching problems with constraints. In *Principles and Practice of Constraint Programming*, volume 1330 of *Lecture Notes in Computer Science*, pages 17–31, 1997.

[12] Yves Caseau and François Laburthe. Solving small TSPs with constraints. In Lee Naish, editor, *Proceedings of the 14th International Conference on Logic Programming*, pages 316–330, Cambridge, July 8–11 1997. MIT Press.

[13] M. Dawande and J. N. Hooker. Inference-based sensitivity analysis for mixed integer/linear programming. *Operations Research, to appear.*

[14] I. R. de Farias, E. L. Johnson, and G. L. Nemhauser. A branch-and-cut approach without binary variables to combinatorial optimization problems with continuous variables and combinatorial constraints. *Knowledge Engineering Review, special issue on AI/OR, submitted*, 1999.

[15] H. El Sakkout, T. Richards, and M. Wallace. Minimal perturbance in dynamic scheduling. In Prade [35], pages 504–508.

[16] Filippo Focacci, Andrea Lodi, and Michela Milano. Solving TSP with time windows with constraints. In *Sixteenth International Conference on Logic Programming*, November 1999.

[17] R. Fourer. Extending a general-purpose algebraic modeling language to combinatorial optimization: A logic programming approach. In *Advances in Computational and Stochastic Optimization, Logic Programming, and Heuristic Search*, Dordrecht, 1994. Kluwer.

[18] R. Fourer, D.M. Gay, and B.W. Kernighan. *AMPL – A Modeling Language for Mathematical Programming.* The Scientific Press, South San Francisco, 1993.

[19] R. Garfinkel and G. Nemhauser. Optimal political districting by implicit enumeration techniques. *Management Science*, 16, 1970.

[20] I. E. Grossmann, J. N. Hooker, R. Raman, and H. Yan. Logic cuts for processing networks with fixed charges. *Computers and Operations Research*, 21:265–279, 1994.

[21] S. Heipcke. Integrating constraint programming techniques into mathematical programming. In Prade [35], pages 259–260.

[22] J. N. Hooker. Logic-based methods for optimization. In Alan Borning, editor, *Principles and Practice of Constraint Programming*, volume 874 of *Lecture Notes in Computer Science*. Springer, May 1994. (PPCP'94: Second International Workshop, Orcas Island, Seattle, USA).

[23] J. N. Hooker. Inference duality as a basis for sensitivity analysis. In *Principles and Practice of Constraint Programming–CP96*, volume 1118 of *Lecture Notes in Computer Science*, 1996.

[24] J. N. Hooker. Inference duality as a basis for sensitivity analysis. *Constraints*, 4:101–112, 1999.

[25] J. N. Hooker. *Logic-Based Methods for Optimization*. Wiley, New York, 2000.

[26] J. N. Hooker and Hak-Jin Kim. Obtaining bounds from the relaxation dual. *In preparation*, 1999.

[27] J. N. Hooker and M. A. Osorio. Mixed logical/linear programming. *Discrete Applied Mathematics*, 96–97(1–3):395–442, 1999.

[28] John N. Hooker, Hak-Jin Kim, and Greger Ottosson. A declarative modeling framework that integrates solution methods. *Annals of Operations Research, Special Issue on Modeling Languages and Approaches, to appear*, 1998.

[29] John N. Hooker and Greger Ottosson. Logic-based benders decomposition. *In preparation.*, 1999.

[30] John N. Hooker, Greger Ottosson, Erlendur S. Thorsteinsson, and Hak-Jin Kim. On integrating constraint propagation and linear programming for combinatorial optimization. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pages 136–141. AAAI, The AAAI Press/The MIT Press, July 1999.

[31] Vipul Jain and Ignacio E. Grossmann. Algorithms for hybrid MILP/CLP models for a class of optimization problems. Technical report, Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, PA, USA, September 1999.

[32] Kim Marriott and Peter J. Stuckey. *Programming with Constraints: An Introduction*. MIT Press, 1998.

[33] Greger Ottosson, Erlendur S. Thorsteinsson, and John N. Hooker. Mixed global constraints and inference in hybrid CLP–IP solvers. In Susanne Heipcke and Mark Wallace, editors, *Proceedings of the Fifth International Conference on Principles and Practice of Constraint Programming (CP-99)'s Post-Conference Workshop on Large Scale Combinatorial Optimisation and Constraints (LSCO&C)*, volume 4 of *Electronic Notes in Discrete Mathematics, www.elsevier.nl/locate/endm*. Elsevier Science, October 1999.

[34] J. M. Pinto and I. E. Grossmann. A logic-based approach to scheduling problems with resource constraints. *Computers and Chemical Engineering*, 21:801–818, 1997.

[35] Henri Prade, editor. *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI-98)*. John Wiley & Sons, 1998.

[36] R. Raman and I. E. Grossmann. Relation between MILP modeling and logical inference for chemical process synthesis. *Computers and Chemical Engineering*, 15:73–84, 1991.

[37] R. Raman and I. E. Grossmann. Modeling and computational techniques for logic based integer programming. *Computers and Chemical Engineering*, 18:563–578, 1994.

[38] Philippe Refalo. Tight cooperation and its application in piecewise linear optimization. In Joxan Jaffar, editor, *Principles and Practice of Constraint Programming*, volume 1713 of *Lecture Notes in Computer Science*. Springer, October 1999.

[39] J-C Régin. A filtering algorithm for constraints of difference in CSPs. Technical Report R.R.LIRMM 93-068, LIRM, Dec 1993.

[40] J.-C. Régin. A filtering algorithm for constraints of difference in CSPs. In *Proc. of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 362–367, 1994.

[41] J.-C. Regin and J.-F. Puget. A filtering algorithm for global sequencing constraints. In *Principles and Practice of Constraint Programming*, volume 1330 of *Lecture Notes in Computer Science*, pages 32–41, 1997.

[42] Jean-Charles Régin. Arc consistency for global cardinality constraints with costs. In Joxan Jaffar, editor, *Principles and Practice of Constraint Programming*, volume 1713 of *Lecture Notes in Computer Science*. Springer, October 1999.

[43] R. Rodosek and M. Wallace. A generic model and hybrid algorithm for hoist scheduling problems. In *Principles and Practice of Constraint Programming*, volume 1520 of *Lecture Notes in Computer Science*, pages 385–399, 1998.

[44] Robert Rodošek, Mark Wallace, and Mozafar Hajian. A new approach to integrating mixed integer programming and constraint logic programming. *Baltzer Journals*, 1997.

[45] E. Tsang. *Foundations of Constraint Satisfaction*. Computation in Cognitive Science. Academic Press, 1993.

[46] M. Türkay and I. E. Grossmann. Logic-based MINLP algorithms for the optimal synthesis of process networks. *Computers and Chemical Engineering*, 20:959–978, 1996.

[47] Pascal Van Hentenryck. *Constraint Satisfaction in Logic Programming*. Logic Programming Series. MIT Press, Cambridge, MA, 1989.

[48] Pascal Van Hentenryck. *The OPL Optimization Programming Language*. MIT Press, 1999.