

On Integrating Constraint Propagation and Linear Programming for Combinatorial Optimization

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

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

[‡] Computing Science Dept., Uppsala University
PO Box 311, S-751 05 Uppsala, Sweden

Abstract

Linear programming and constraint propagation are complementary techniques with the potential for integration to benefit the solution of combinatorial optimization problems. Attempts to combine them have mainly focused on incorporating either technique into the framework of the other — traditional models have been left intact. We argue that a rethinking of our modeling traditions is necessary to achieve the greatest benefit of such an integration. We propose a declarative modeling framework in which the structure of the constraints indicates how LP and CP can interact to solve the problem.

Introduction

Linear programming (LP) and constraint propagation (CP) are techniques from different fields that tend to be used separately in integer programming (IP) and constraint (logic) programming (CLP), respectively. They have the potential for integration to benefit the solution of combinatorial optimization problems. Yet only recently have attempts been made at combining them.

IP has been successfully applied to a wide range of problems, such as capital budgeting, bin packing, crew scheduling and traveling salesman problems. CLP has in the last decade been shown to be a flexible, efficient and commercially successful technique for scheduling, planning and allocation. These problems usually involve permutations, discretization or symmetries that may result in large and intractable IP models.

Both CLP and IP rely on branching to enumerate regions of the search space. But within this framework they use dual approaches to problem solving: inference and search. CLP emphasizes inference in the form of constraint propagation, which removes infeasible values from the variable domains. It is not a search method, i.e., an algorithm that examines a series of complete labellings until it finds a solution. IP, by contrast, does exactly this. It obtains complete labellings by solving linear programming relaxations of the problem in the branching tree.

IP has the advantage that it can generate cutting planes (inequalities implied by the constraint set) that strengthen the linear relaxation. This can be a powerful technique when

the problem is amenable to polyhedral analysis. But IP has the disadvantage that its constraints must be expressed as inequalities (or equations) involving integer-valued variables. Otherwise the linear programming relaxation is not available. This places a severe restriction on IP's modeling language.

In this paper we argue that the key to effective integration of CP and LP lies in the design of the modeling language. We propose a language in which conditional constraints indicate how CP and LP can work together to solve the problem.

We begin, however, by reviewing efforts that have hitherto been made toward integration.

Previous Work

Several articles compare CLP and IP (Smith *et al.* 1995; Darby-Dowman & Little 1998). They report experimental results that illustrate some key properties of the techniques: IP is very efficient for problems with good relaxations, but it suffers when the relaxation is weak or when its restricted modeling framework results in large models. CLP, with its more expressive constraints, has smaller models that are closer to the problem description and behaves well for highly constrained problems, but it lacks the “global perspective” of relaxations.

Some attempts have been made at integration. Early out was (Beringer & Backer 1995) in which the idea is explored of coupling CP and LP solvers with bounds propagation and fixed variables. In (Rodosek, Wallace, & Hajian 1997), CP is used along with LP relaxations in a single search tree to prune domains and establish bounds. A node can fail either because propagation produces an empty domain, or the LP relaxation is infeasible or has an optimal value that is worse than the value of the optimal solution (discussed below). A systematic procedure is used to create a “shadow” MIP model for the original CLP model. It includes reified arithmetic constraints (which produce big-M constraints, illustrated below) and `all different` constraints. The modeler may annotate constraints to indicate which solver should handle them — CP, LP or both.

Some research has been aimed at incorporating better support for symbolic constraints in IP. (Hajian, Rodosek, & Richards 1996; Hajian 1996) show how disequalities ($X_i \neq X_j$) can be handled (more) efficiently in IP solvers. Fur-

¹Copyright © 1999, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

ther, they give a linear modeling of the alldifferent constraint.

Bockmayr and Kasper propose an interesting framework in (Bockmayr & Kasper 1998) for combining CLP and IP, in which several approaches to integration or synergy are possible. They investigate how symbolic constraints can be incorporated into IP much as cutting planes are. They also show how a linear system of inequalities can be used in CLP by incorporating it as a symbolic constraint. They also discuss a closer integration in which both linear inequalities and domains appear in the same constraint store.

Characterization

We start with a basic characterization of CLP and IP.

Constraint (Logic) Programming

In Finite Domain CLP each integer variable x_i has an associated *domain* D_i , which is the set of possible values this variable can take on in the (optimal) solution. The cartesian product of the domains, $D_1 \times \dots \times D_n$, forms the solution space of the problem. This space is finite and can be searched exhaustively for a feasible or optimal solution, but to limit this search CP is used to infer infeasible solutions and prune the corresponding domains. From this viewpoint, CP operates on the set of possible solutions and narrows it down.

Integer Programming

In contrast to CLP, IP does not maintain and reduce a set of solutions defined by variable domains. It generates a series of complete labellings, each obtained at a node of the branching tree by solving a relaxation of the problem at that node. The relaxation is usually constructed by dropping some of the constraints, notably the integrality constraints on the variables, and perhaps by adding valid constraints (cutting planes) that make the relaxation tighter. In a typical application the relaxation is rapidly solved to optimality with a linear programming algorithm.

If the aim is to find a feasible solution, branching continues until the solution of the relaxation happens to be feasible in the original problem (in particular, until it is integral). Relaxations therefore provide a heuristic method for identifying solutions. In an optimization problem, relaxation also provides bounds for a branch-and-bound search. At each node of the branching tree, one can check whether the optimal value of the relaxation is better than the value of the best feasible found so far. If not, there is no need to branch further at that node.

The dual of the LP relaxation can also provide useful information, perhaps by fixing some integer variables or generating additional constraints (nogoods) in the form of “Benders cuts.” (Benders 1962; Geoffrion 1974). We will see how the latter can be exploited in an integrated framework.

Comparison of CP and LP

CP can accelerate the search for a solution by

- reducing variable domains (and in particular by proving infeasibility),

- tightening the linear relaxation by adding bounds and cuts in addition to classical cutting planes, and
- eliminating search of symmetric solutions, which are often more easily excluded by using symbolic constraints.

LP can enhance the solver by

- finding feasible solutions early in the search by “global reasoning,” i.e., solution of an LP relaxation,
- similarly providing stronger bounds that accelerate the proof of optimality, and
- providing reasons for failure or a poor solution, so as to produce nogoods.

Modeling for Hybrid Solvers

The approaches taken so far to integration of CP and LP are (a) to use both models in parallel, and (b) to try to incorporate one within the other. The more fundamental question of whether a *new* modeling framework should be used has not yet been explored in any depth. The success of (a) depends on the strength of the links between the models and to what degree the overhead of having two models can be avoided. Option (b) is limited in what it can achieve. The high-level symbolic constraints of CLP cannot directly be applied in an IP model, and the same holds for attempts to use IP’s cutting planes and relaxations in CLP.

We will use a simple multiple-machine scheduling problem to illustrate the advantages of a new modeling framework. Assume that we wish to schedule n tasks for processing on as many as n machines. Each machine m runs at speed r_m . The objective is to minimize the total fixed cost of the machines we use. Let R_j be the release time, P_j the processing time for speed $r_m = 1$ and D_j the deadline for task j . Let C_m be the fixed cost of using machine m and t_j the start time of task j .

We first state an IP model, which uses 0–1 variables x_{ij} to indicate the sequence in which the jobs are processed. Let $x_{ij} = 1$ if task i precedes task j , $i \neq j$, on the same machine, with $x_{ij} = 0$ otherwise. Also let 0–1 variable $y_{mj} = 1$ if task j is assigned to machine m , and 0–1 variable $z_m = 1$ if machine m is used. Then an IP formulation of this problem is,

$$\min \sum_m C_m z_m$$

$$\text{s.t. } z_m \geq y_{mj}, \quad \forall m, j, \quad (1)$$

$$\sum_m y_{mj} = 1, \quad \forall j, \quad (2)$$

$$t_j + \sum_m \frac{P_j}{r_m} y_{mj} \leq D_j, \quad \forall j,$$

$$R_j \leq t_j, \quad \forall j,$$

$$t_i + \sum_m \frac{P_i}{r_m} y_{mi} \leq t_j + M(1 - x_{ij}), \quad \forall i \neq j, \quad (3)$$

$$x_{ij} + x_{ji} \geq y_{mi} + y_{mj} - 1, \quad \forall m, i < j, \quad (4)$$

$$z_m, y_{mj}, x_{ij} \in \{0, 1\}, t_j \geq 0, \quad \forall m, i, j.$$

Constraint (3) is a “big-M” constraint. If M is a sufficiently large number, the constraint forces task i to precede task j if $x_{ij} = 1$ and has no effect otherwise.

A CLP model for the same problem is

$$\begin{aligned}
\min \quad & \sum_m C_m z_m \\
\text{s.t.} \quad & \text{if } m_i = m_j \text{ then} \\
& (t_i + \frac{P_i}{r_{m_i}} \leq t_j) \vee (t_j + \frac{P_j}{r_{m_j}} \leq t_i), \quad \forall i, j, \quad (5) \\
& t_j + \frac{P_j}{r_{m_j}} \leq D_j, \quad \forall j, \\
& R_j \leq t_j, \quad \forall j, \\
& \text{if at least } (m, [m_1, \dots, m_n], 1) \\
& \text{then } z_m = 1 \text{ else } z_m = 0, \quad \forall m, \quad (6) \\
& m, m_j \in \{1, \dots, n\}, t_j \geq 0, \quad \forall j,
\end{aligned}$$

where m_j is the machine assigned to task j . The CLP model has the advantage of dispensing with the doubly-subscripted 0–1 variables x_{ij} and y_{mi} , which are necessary in IP to represent permutations and assignments. This advantage can be pronounced in larger problems. A notorious example is the progressive party problem (Smith et al., 1995), whose IP model requires an enormous number of multiply-subscripted variables.

The IP model has the advantage of a useful linear programming relaxation, consisting of the objective function, constraints (1)–(2), and bounds $0 \leq y_{mj} \leq 1$. The 0–1 variables y_{mj} enlarge the model but compensate by making this relaxation possible. However, the IP constraints involving the permutation variables x_{ij} yield a very weak relaxation and needlessly enlarge the LP relaxation.

Somehow we must combine the succinctness of the CLP model with an ability to create a relaxation from that portion of the IP model that has a useful relaxation. To do this we propose taking a step back to investigate how one can design a model to suit the solvers rather than adjust the solvers to suit the traditional models.

Mixed Logical/Linear Programming

We begin with the framework of Mixed Logical/Linear Programming (MLLP) proposed in (Hooker 1994; Hooker & Osorio 1997; Hooker, Kim, & Ottosson 1998). It writes constraints in the form of conditionals that link the discrete and continuous elements of the problem. A model has the form

$$\begin{aligned}
\min \quad & cx \quad (7) \\
\text{s.t.} \quad & h_i(y) \rightarrow A^i x \geq b^i, \quad i \in I, \\
& x \in R^n, y \in D,
\end{aligned}$$

where y is a vector of discrete variables and x a vector of continuous variables. The antecedents $h_i(y)$ of the conditionals are constraints that can be treated with CP techniques. The consequents are linear inequality systems that can be inserted into an LP relaxation.

A linear constraint set $Ax \geq b$ which is enforced unconditionally may be so written for convenience, with the understanding that it can always be put in the conditional form

$(0 = 0) \rightarrow Ax \geq b$. Similarly, an unconditional discrete constraint h can be formally represented with the conditional $\neg h \rightarrow (1 = 0)$.

The absence of discrete variables from the objective function will be useful algorithmically. Costs that depend on discrete variables can be represented with conditional constraints. For example, the objective function $\sum_j c_j x_j$, where $x_j \in \{0, 1\}$, can be written $\sum_j z_j$ with constraints $(x_j = 1) \rightarrow (z_j = c_j)$ and $z_j \geq 0$ for all j .

A useful modeling device is a *variable subscript*, i.e., a subscript that contains one or more discrete variables. For example, if c_{jk} is the cost of assigning worker k to job j , the total cost of an assignment can be written $\sum_j c_{jy_j}$, where y_j is a discrete variable that indicates the worker assigned job j . The value of c_{jy_j} is in effect a function of $y = (y_1, \dots, y_n)$ and can be written $g_j(y)$, where function g_j happens to depend only on y_j . The MLLP model can incorporate this device as follows:

$$\begin{aligned}
\min \quad & cx \\
\text{s.t.} \quad & h_i(y) \rightarrow L_i(x, y), \quad i \in I, \\
& x \in R^n, y \in D,
\end{aligned}$$

where

$$L_i(x, y) = \sum_{k \in K_i(y)} a_{ik}(y) x_{j_{ik}(y)} \geq b_i(y).$$

Note that the model also allows for a summation taken over a *variable index set* $K_i(y)$, which is a set-valued function of y , as well as real-valued “variable constants” $b_i(y)$.

Models of this sort can in principle be written in the more primitive form (7) by adding sufficiently many conditional constraints. For example, the constraint $z \geq \sum_j c_j y_j$ can be written $z \geq \sum_j z_j$, if the following constraints are added to the model

$$(y_j = k) \rightarrow (z_j = c_{jk}), \quad \text{all } j, k \in \{1, \dots, n\},$$

where each $y_j \in \{1, \dots, n\}$. It is preferable, however, for the solver to process variables subscripts and index sets directly.

The Solution Algorithm

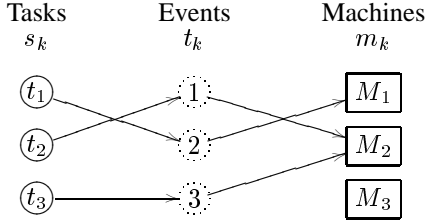
An MLLP problem is solved by branching on the discrete variables. The conditionals assign roles to CP and LP: CP is applied to the discrete constraints to reduce the search and help determine when partial assignments satisfy the antecedents. At each node of the branching tree, an LP solver minimizes cx subject to the inequalities $A^i x \geq b^i$ for which $h_i(y)$ is determined to be true. This delayed posting of inequalities leads to small and lean LP problems that can be solved efficiently. A feasible solution is obtained when the truth value of every antecedent is determined, and the LP solver finds an optimal solution subject to the enforced inequalities.

Computational tests reported in (Hooker & Osorio 1997) suggest that an MLLP framework not only has modeling advantages but can often permit more rapid solution of the problem than traditional MILP solvers. However, a number of issues are not addressed in this work, including: (a)

systematic implementation of variable subscripts and index sets, (b) taking full advantage of the LP solution at each node, and (c) branching on continuous variables and propagation of continuous constraints.

An Example

We can now formulate the multiple machine scheduling problem discussed earlier in an MLLP framework. Let k index a sequence of *events*, each of which is the start of some task. In the following model we will focus on the events, using mappings from events to tasks and events to machines, respectively.



Variable t_k will now be the start time of event k , s_k the task that starts, and m_k the machine to which it is assigned. The formal MLLP model is,

$$\begin{aligned} \min \quad & \sum_m f_m \\ \text{s.t.} \quad & (m_k = m_l) \rightarrow (t_k + \frac{P_{s_k}}{r_{m_k}} \leq t_l), \quad \forall k < l, \\ & t_k + \frac{P_{s_k}}{r_{m_k}} \leq D_{s_k}, \quad \forall k, \\ & R_{s_k} \leq t_k, \quad \forall k, \\ & \text{alldifferent}\{s_1, \dots, s_n\}, \\ & f_{m_k} = C_{m_k}, \quad f_m \geq 0, \quad \forall k, m. \end{aligned}$$

The model shares CLP's succinctness by dispensing with doubly-subscripted variables. To obtain the relaxation afforded by IP, we can simply add the objective function and constraints (1)–(2) to the model, and link the variables y_{mi} to the other variables logically in (10).

$$\begin{aligned} \min \quad & \sum_m C_m z_m \\ \text{s.t.} \quad & (m_k = m_l) \rightarrow (t_k + \frac{P_{s_k}}{r_{m_k}} \leq t_l), \quad \forall k < l, \\ & t_k + \frac{P_{s_k}}{r_{m_k}} \leq D_{s_k}, \quad \forall k, \\ & R_{s_k} \leq t_k, \quad \forall k, \\ & \text{alldifferent}\{s_1, \dots, s_n\}, \\ & z_m \geq y_{m,j}, \quad \forall m, j, \quad (8) \\ & \sum_m y_{m,j} = 1, \quad \forall j, \quad (9) \\ & y_{m_k s_k} = 1, \quad \forall k. \quad (10) \end{aligned}$$

The relaxation now minimizes $\sum_m C_m z_m$ subject to (8), (9), $0 \leq y_{m,j} \leq 1$, and $y_{m,j} = 1$ for all $y_{m,j}$ fixed to 1 by (10). One can also add a number of additional valid constraints involving the $y_{m,j}$'s and the t_k 's.

A Perspective on MLLP

The framework for integration described in (Bockmayr & Kasper 1998) provides an interesting perspective on MLLP. The CLP literature distinguishes between *primitive* and *non-primitive* constraints. Primitive constraints are “easy” constraints for which there are efficient (polynomial) satisfaction and optimization procedures. They are maintained in a *constraint store*, which in finite-domain CLP consists simply of variable domains. Propagation algorithms for nonprimitive constraints retrieve current domains from the store and add the resulting smaller domains to the store.

In IP, linear inequalities over continuous variables are primitive because they can be solved by linear programming. The integrality conditions are (the only) nonprimitive constraints.

Bockmayr and Kasper propose two ways of integrating LP and CP. The first is to incorporate the LP part of the problem into a CLP framework as a nonprimitive constraint. Thus LP becomes a constraint propagation technique. It accesses domains in the form of bounds from the constraint store and add new bounds obtained by minimizing and maximizing single variable.

A second approach is to make linear inequalities primitive constraints. The constraint store contains continuous inequality relaxations of the constraints but excludes integrality conditions. For example, discrete constraints $x_1 \vee x_2$ and $\neg x_1 \vee x_2$ could be represented in the constraint store as inequalities $x_1 + x_2 \geq 1$ and $(1 - x_1) + x_2 \geq 1$ and bounds $0 \leq x_j \leq 1$. If constraint propagation deduced that x_2 is true, the inequality $x_2 \geq 1$ would be added to the store. This is an instance of what has long been known as “preprocessing” in MILP, which can therefore be viewed as a special case of this second kind of integration.

MLLP is a third type of integration in which two constraint stores are maintained (see Figure 1). A classical fi-

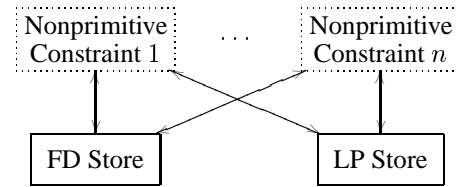


Figure 1: Constraint stores and nonprimitive constraints in MLLP

nite domain constraint store, S_{FD} , contains domains, and the LP constraint store, S_{LP} , contains linear inequalities and bounds. The nonprimitive constraints can access and add to both constraint stores. Since only domain constraints $x_i \in D_i$ exist in the FD store, integrality constraints can remain therein as primitive constraints. There are no continuous variables in the CP store and no discrete variables in the LP store. The conditional constraints of MLLP act as the prime inference agents connecting the two stores, reading domains of the CP store and adding inequalities to the LP store (Figure 2).

In the original MLLP scheme, the conditionals are unidirectional, in the sense that they infer from S_{FD} and post

to S_{LP} and not vice-versa. This is because the solution algorithm branches on discrete variables. As the discrete domains are reduced by branching, the truth value of more antecedents is inferred by constraint propagation, and more inequality constraints are posted. However, conditionals in the opposite direction could also be used if one branched on continuous variables by splitting intervals. The antecedents would contain continuous numerical constraints (not necessarily linear inequalities), and the consequents would contain primitive discrete constraints, i.e., restrictions on discrete variable domains. The truth value of the antecedents might be inferred using interval propagation.

We will next give two more examples of nonprimitive constraints in MLLP; a generalized version of the `element` constraint for handling variable subscripts, and a constraint which derives nogoods from S_{LP} .

Variable Subscripts

As seen before, MLLP provides variable subscripts as a modeling component, but an expansion to conditional constraints is in most cases not tractable. Instead a nonprimitive constraint, or inference agent, can be designed to handle variable subscripts more efficiently.

There are basically two cases in which a variable subscript can occur — in a discrete constraint or in a continuous inequality. In the former case it can either be a vector of constants or a vector of discrete variables; both of these correspond to the traditional use of the `element/3` (Marriott & Stuckey 1998) constraint found in all major CP systems and libraries (e.g. (Dincbas *et al.* 1988; Carlsson 1995)). This constraint takes the form `elementFD(I, [X1, ..., Xn], Y)`, where I is an integer variable with domain $D_I = \{1, \dots, n\}$, indexing the list, and $Y = X_I$.

Here we will consider the second case, `elementLP(I, [X1, ..., Xn], Y)`, where I is still a discrete, indexing variable, but x_i and Y are *continuous* variables or constants. Propagating this constraint can be done almost as before. Let the interval $[\min(x_i), \max(x_i)]$ be the domain of x_i . Then upon change of the domain of I , we can compute

$$\begin{aligned} \min &= \{\min(x_i) \mid i \in D_I\} \\ \max &= \{\max(x_i) \mid i \in D_I\} \end{aligned}$$

reading D_I from the S_{CP} and the adding new bounds

$$\min \leq Y \leq \max$$

to S_{LP} . Similarly, bounds of Y can be used to prune D_I . (Stronger bounds for variables in LP can be obtained by minimizing and maximizing the variable subject to the linear inequalities in S_{LP} , which for some cases might be beneficial.)

The important point here is not the details of how we can propagate this constraint, but rather to exemplify how an inference agent can naturally access both constraint stores.

Infeasible LP

When the LP is infeasible, any dual solution specifies an infeasible linear combination of the constraint set. For each conditional constraint $y_i \rightarrow A^i x \geq b^i, i \in I$ where some

$ax \geq b \in A^i x \geq b^i$ has a corresponding nonzero dual multiplier, we can form the logical constraint

$$\bigvee_{i \in I} \neg y_i$$

This nogood (Tsang 1993) must be satisfied by any solution of the problem, because its corresponding set of linear inequalities forms an infeasible combination. This scheme can naturally be encapsulated within a nonprimitive constraint, `nogood`, reading from S_{LP} and writing to S_{FD} . This agent can collect, merge and maintain no-goods for any combination of nonzero dual values in any infeasible LP node, and can infer primitive and nonprimitive constraints which will strengthen S_{FD} . A related use of the infeasible combination has previously been explored in the context of intelligent backtracking (De Backer & Beringer 1991). Figure 2 shows our `nogood` constraint and the other basic nonprimitive constraints of MLLP.

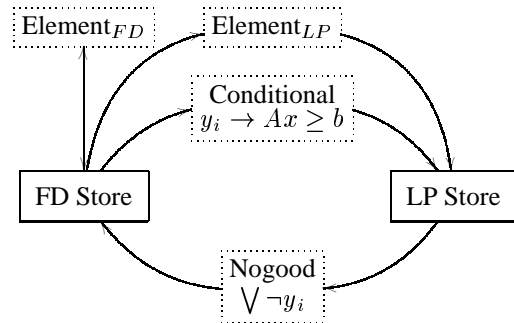


Figure 2: Nonprimitive constraints in MLLP

Feasible LP

In IP, the solution of the relaxation provides a complete labelling of the variables. This sort of labelling is not immediately available in MLLP, because the relaxation involves only continuous variables. However, the solution \bar{x} of a feasible relaxation can be heuristically extended to a complete labelling (\bar{x}, \bar{y}) that may satisfy the constraints. (Because y does not occur in the objective function, its value will not affect the optimal value of the problem.) Given any conditional $h_i(y) \rightarrow A^i x \geq b^i, h_i(\bar{y})$ must be false if $A^i \bar{x} \not\geq b^i$, but it can be true or false if $A^i \bar{x} \geq b^i$. One can therefore employ a heuristic (or even an exhaustive search) that tries to assign values \bar{y}_j to the y_j 's from their current domains so as to falsify the antecedents that must be false.

Conclusion

LP and CP have long been used separately, but they have the potential to be integrated as complementary techniques in future optimization frameworks. To do this fully and in general, the modeling traditions of mathematical programming and constraint programming also must be integrated. We propose a unifying modeling and solution framework that aims to do so. Continuous and discrete constraints are naturally combined using conditional constraints, allowing

a clean separation and a natural link between constraints amenable to CP and continuous inequalities efficiently handled by LP.

References

- Benders, J. F. 1962. Partitioning procedures for solving mixed-variables programming problems. *Numer. Math.* 4:238–252.
- Beringer, H., and Backer, B. D. 1995. Combinatorial problem solving in constraint logic programming with cooperating solvers. In Beierle, C., and Plümer, L., eds., *Logic Programming: Formal Methods and Practical Applications*, Studies in Computer Science and Artificial Intelligence. Elsevier. chapter 8, 245–272.
- Bockmayr, A., and Kasper, T. 1998. Branch-and-infer: A unifying framework for integer and finite domain constraint programming. *INFORMS J. Computing* 10(3):287–300.
- Carlsson, M. 1995. SICStus Prolog User’s Manual. SICS research report, Swedish Institute of Computer Science. URL: <http://www.sics.se/isl/sicstus.html>.
- Darby-Dowman, K., and Little, J. 1998. Properties of some combinatorial optimization problems and their effect on the performance of integer programming and constraint logic programming. *INFORMS Journal on Computing* 10(3):276–286.
- De Backer, B., and Beringer, H. 1991. Intelligent backtracking for CLP languages: An application to CLP(R). In Saraswat, V., and Ueda, K., eds., *Logic Programming, Proceedings of the 1991 International Symposium*, 405–419. San Diego, USA: The MIT Press.
- Dincbas, M.; Van Hentenryck, P.; Simonis, H.; Aggoun, A.; Graf, T.; and Berthier, F. 1988. The Constraint Logic Programming Language CHIP. In *FGCS-88: Proceedings International Conference on Fifth Generation Computer Systems*, 693–702. Tokyo: ICOT.
- Geoffrion, A. 1974. Lagrangian relaxation for integer programming. *Mathematical Programming Study* 2:82–114.
- Hajian, M.; Rodosek, R.; and Richards, B. 1996. Introduction of a new class of variables to discrete and integer programming problems. *Baltzer Journals*.
- Hajian, M. T. 1996. Dis-equality constraints in linear/integer programming. Technical report, IC-Parc.
- Hooker, J. N., and Osorio, M. A. 1997. Mixed logical/linear programming. *Discrete Applied Mathematics*, to appear.
- Hooker, J. N.; Kim, H.-J.; and Ottosson, G. 1998. A declarative modeling framework that integrates solution methods. *Annals of Operations Research, Special Issue on Modeling Languages and Approaches*, to appear.
- Hooker, J. N. 1994. Logic-based methods for optimization. In Borning, A., ed., *Principles and Practice of Constraint Programming*, volume 874 of *Lecture Notes in Computer Science*, 336–349.
- Marriott, K., and Stuckey, P. J. 1998. *Programming with Constraints: An Introduction*. MIT Press.
- Rodosek, R.; Wallace, M.; and Hajian, M. 1997. A new approach to integrating mixed integer programming and constraint logic programming. *Baltzer Journals*.
- Smith, B.; Brailsford, S.; Hubbard, P.; and Williams, H. P. 1995. The Progressive Party Problem: Integer Linear Programming and Constraint Programming Compared. In *CP95: Proceedings 1st International Conference on Principles and Practice of Constraint Programming*.
- Tsang, E. 1993. *Foundations of Constraint Satisfaction*. Academic Press.