

Hybrid Modeling

J. N. Hooker

Abstract The modeling practices of constraint programming (CP), artificial intelligence, and operations research must be reconciled and integrated if the computational benefits of combining their solution methods are to be realized in practice. This chapter focuses on CP and mixed integer/linear programming (MILP), in which modeling systems are most highly developed. It presents practical guidelines and supporting theory for the two types of modeling. It then suggests how an integrated modeling framework can be designed that retains, and even enhances, the modeling power of CP while allowing the full computational resources of both fields to be applied and combined. A series of examples are used to compare modeling practices in CP, MILP, and an integrated framework.

1 Modeling as a Key to Hybrid Problem Solving

The solution methods of constraint programming, artificial intelligence, and operations research have complementary strengths. Recent research shows that these strengths can be profitably combined in hybrid algorithms, many of which are described in subsequent chapters of this book. Under the right conditions, one need not choose between CP, AI, and OR, but can have the best of all three worlds.

There is more to integration, however, than combining algorithmic techniques. There is also the issue of problem formulation. CP, AI, and OR have developed their own distinctive modeling styles, which poses the question of how to formulate problems that are to be solved by hybrid methods. Whereas the solution methods of the three fields can be seen as related and complementary, the modeling styles seem very different and possibly irreconcilable.

For example, one can contrast CP models with mixed integer/linear programming (MILP) models developed in the OR community. CP organizes its models around

high-level *global constraints*, each of which represents a structured collection of simpler constraints. The solver may have a large library of global constraints for which special-purpose algorithms have been designed. The modeler selects global constraints that correspond to the major structural elements of the problem and combines them with some low-level constraints as needed to complete the formulation.

An advantage one might claim for this approach is that the selection of global constraints reveals the special structure of the problem and allows the solver to exploit it. It may also result in a fairly succinct model that is easier to read and debug because the global constraints reflect how the modeler thinks about the problem. Such a model can be seen as a theory or explanation that helps us understand the phenomenon described by the model. Scientific theories, after all, are essentially explanatory models of phenomena. On the other hand, a model that uses high-level global constraints may have to be reformulated for solvers that use different sets of global constraints.

An MILP model takes the opposite approach. It uses a very small set of primitive terms, namely linear inequalities. The problem is broken down into elementary ideas that can be captured with inequality constraints, perhaps using auxiliary variables and other devices. Advanced modeling systems allow one to abbreviate the formulation with loops and if-then statements that generate a large number of constraints, but the formulation must nonetheless be conceived in terms of inequality constraints.

An advantage cited for this approach is the independence of model and method. Once the model is written, it can be submitted to any MILP solver. In fact, there are libraries of MILP instances that have been used, without alteration, as testbeds for several generations of solvers. In addition, the user need not be familiar with a library of meta-constraints. Finally, inequality constraints are suitable for the highly developed relaxation technology of MILP, including strong cutting planes, Lagrangean relaxation, and so forth. On the negative side, the model may be long, nonintuitive, and hard to debug. The solver may be unable to exploit substructure that global constraints would have revealed, aside from a few special types of structure that can be automatically recognized by the more sophisticated solvers.

The modeling issue must be resolved if hybrid solvers are to harness the complementary strengths of CP and MILP. CP methods rely heavily on the application of specialized filtering methods to global constraints and must therefore know where global constraints appear in the problem. MILP methods rely heavily on relaxation methods and cutting planes that have been developed for inequality constraints and therefore require that inequality constraints appear in the model.

The modeling issue is important in practice. Developing a formulation and the associated data is an expensive undertaking. Models must typically be reformulated, updated, and debugged many times. While reasonably fast solution is desirable, it is equally essential that solution software support and simplify modeling activities. Practitioners frequently report that the greatest benefit of developing a model is not so much the ability to obtain a solution as the clearer understanding of the problem one obtains from the modeling exercise.

Practical application also requires interaction between modeling and the solution process. There are typically alternative ways to formulate a problem, some of which result in much faster solution than others. A practitioner can begin with a straightforward model and solve it on a small problem instance. The model can then be altered and refined as the instances are scaled up, so as to maintain tractability. A practical modeling system should support this kind of trial-and-error process.

This chapter focuses on integrating the modeling styles of CP and MILP in particular, because it is in these areas that modeling systems are most highly developed. After a brief review of current hybrid modeling systems and some basic terminology, the chapter is organized in three major sections. The first two sections develop guidelines for CP and MILP modeling, respectively. They introduce the necessary theory and present a series of examples to illustrate good modeling practice. The third section suggests how these modeling practices may be merged into a unified approach, based on ideas that have evolved in the CP-AI-OR community over the last decade or more. It illustrates the ideas by showing how each of the examples discussed earlier may be rendered in an integrated modeling framework. The paper concludes by assessing the extent to which the particular strengths of CP and MILP modeling carry over into an integrated framework.

2 Modeling Systems

A number of modeling systems implement CP/MILP hybrid modeling to a greater or lesser extent. A pioneering effort is *ECLⁱPS^e* [2, 4, 14, 42], a Prolog-based constraint logic programming system that provides an interface with linear and MILP solvers. *ECLⁱPS^e* was recently revised to accept models written in MiniZinc, a CP-based modeling system [35]. OPL Studio [19] provides a modeling language that expresses both MILP and CP constraints. A script language allows one to write algorithms that call the CP and MILP solvers repeatedly. Mosel [15, 16] is a specialized programming language that interfaces with various solvers, including MILP and CP solvers.

SCIL [3] is an MILP solver with a modeling language that designates specially structured sets of inequalities. SIMPL [5, 52] is a hybrid solver with a high-level modeling language that integrates CP and MILP modeling. The solver processes each constraint with both CP-based and MILP-based techniques that are combined in a branch-infer-and relax algorithmic framework. SCIP [1] is a callable library that gives the user control of a solution process that can involve both CP and MILP solvers. G12 [44] is a CP-based and hybrid system that accepts models written in the Zinc modeling language and uses a mapping language (Cadmium) to associate the models with underlying solvers and/or search strategies.

There has been some investigation of integrated modeling beyond the CP/MILP interface. The modeling language in the Comet system [18], which evolved from an earlier system Localizer [34], allows CP and MILP constraints as well as high-level constraint-based specifications of local search. The global optimization pack-

age BARON [45, 46] combines nonlinear (as well as linear) integer programming with CP-style domain reduction, although it uses a modeling system (AIMMS) that does not support CP-style constraints. Some general discussions of integrated modeling include [23, 26, 52].

No attempt is made here to describe currently available modeling languages, because they evolve rapidly. The emphasis is on general principles that should inform the design of any integrated modeling system. In fact, none of the existing systems implement all of these principles or fully integrate CP and MILP modeling. Yet the necessary concepts and technology have reached a stage where a seamlessly integrated modeling system is within reach. Perhaps the discussion to follow will help encourage efforts in this direction.

3 Basic Terminology

For present purposes, a *problem* consists of a stock of *variables* x_1, \dots, x_n and a set of *constraints* C_1, \dots, C_m . Each variable x_j is associated with a *domain* that can be viewed as a set of permissible values for x_j . A *solution* is any tuple $x = (x_1, \dots, x_n)$ for which each x_j belongs to its domain. Each constraint is associated with a set of solutions that *satisfy* it. The goal is to find a *feasible* solution, which is a solution that satisfies all the constraints.

When there is an *objective function* $f(x)$, the goal is to find an *optimal* solution, which can without loss of generality be defined as a feasible solution that minimizes $f(x)$ subject to the constraints. That is, an optimal solution \bar{x} is one such that $f(\bar{x}) \leq f(x)$ for all feasible solutions x .

The terminology here is borrowed from both CP and OR, with constraints and domains defined roughly as in CP, and the various types of solutions as in OR. It is important to note that the domain of a variable need not consist of numbers, although this is the normal practice in OR. In CP, a domain may consist of arbitrary objects, or even sets of objects.

CP methods are typically designed to find feasible solutions and OR methods to find optimal solutions, but this is not a fundamental difference. CP methods can find optimal solutions by adding the constraint $f(x) \leq U$ to the problem and gradually reducing the bound U until no feasible solution can be found.

The *modeling* task is (a) to identify variables, domains, constraints, and perhaps an objective function that formulate the desired problem, and (b) to express the constraints and objective function in a form that allows solution by available software.

4 CP Modeling

The concepts of domain consistency, filtering, propagation, and global constraints are essential to understanding CP modeling practice. Once these are defined, guide-

lines for CP modeling can be stated and illustrated with a series of examples. These examples will recur in later sections to show how they can be formulated as MILP models and in an integrated modeling context. For further practice, one can consult the tutorial on CP modeling at [41, 43].

4.1 Consistency, Filtering, and Propagation

A central concept in CP solution technology is *domain consistency*, also known as *generalized arc consistency* or *hyperarc consistency*. A problem is domain consistent if every element of every domain is consistent with the constraint set. That is, for each element v in the domain of any variable x_j , there is at least one feasible solution in which $x_j = v$. Another way to put this is that each variable's domain is equal to the projection of the feasible set onto that variable.

CP solvers typically achieve or approximate domain consistency for individual constraints by means of *filtering* algorithms that remove inconsistent values from domains. The smaller domains obtained by filtering a constraint become the starting point for filtering another constraint, in a process known as *constraint propagation*. It is important to note that constraint propagation does not necessarily achieve domain consistency for the problem as a whole, even if the filtering algorithms achieve it for every individual constraint.

The advantage of filtering domains is that the search algorithm spends less time enumerating values that cannot be part of a feasible solution. If filtering reduces every domain to a singleton and achieves domain consistency as well, then a feasible solution is at hand.

A somewhat weaker form of consistency is *bounds consistency*, which applies when there is a natural ordering for the elements of a domain. A problem is bounds consistent if for each domain, its smallest value is consistent with the constraint set, and likewise for its largest value.

4.2 Global Constraints

CP modeling relies heavily on the use of *global constraints*. A global constraint represents a set of more elementary constraints that exhibit special structure when considered together. Each individual constraint typically involves only a few of the variables that appear in the global constraint and might be viewed as “local” in that sense.

A practice of using global constraints, rather than writing out the more elementary constraints, has several advantages: (a) it is more convenient; (b) it yields a more natural and readable model that is more easily debugged; (c) it alerts the solver that the model contains special structure that might have been overlooked if the elementary constraints had been written. In particular, filtering algorithms designed

for a global constraint can generally remove more values than filtering algorithms designed for the more elementary constraints.

An example of a global constraint is the well-known *all-different* constraint, which can be written `alldiff(X)` and requires that the variables in the set $X = \{x_1, \dots, x_k\}$ take pairwise distinct values. It replaces a set of more elementary constraints in the form of inequations $x_i \neq x_j$ for $1 \leq i < j \leq k$.

Filtering is more effective when applied to an `alldiff` constraint than when applied to the individual inequations it represents. Suppose, for example, that variables x_1, x_2, x_3 all have domain $\{a, b\}$. The constraint `alldiff({x1, x2, x3})` allows filtering to reduce each domain to the empty set if domain consistency is achieved. By contrast, achieving domain consistency for the individual inequations $x_1 \neq x_2$, $x_1 \neq x_3$, $x_2 \neq x_3$ removes no values from the domains. It is therefore better modeling practice to use the `alldiff`.

4.3 Example: Sudoku Puzzles

The popular sudoku puzzle (Fig. 1) illustrates how global constraints can be used in modeling, in this case `alldiff` constraints. A sudoku puzzle consists of a 9×9 grid whose cells must be filled with digits $1, \dots, 9$ so that each row and each column of the grid contains nine distinct digits. In addition, each of the nine 3×3 subsquares of the grid must contain nine distinct digits. Some of the cells are preassigned digits.

6		7	4	1	2			5
				8		4		
		4			3	8	2	6
	2					1	6	3
			5		6			
3	4	6					5	
4	7	3	2			5		
		9		5				
5			1	7	9	3		2

Fig. 1 A sudoku puzzle.

The first task in formulating a model is normally to define the variables. In this case, a natural scheme is to let x_{ij} be the digit in row i and column j , so that each x_{ij} has domain $\{1, \dots, 9\}$. The domain could of course be any set of nine distinct objects, not necessarily numbers, without affecting the problem. Let X_i be the set of variables in the i th row, namely $\{x_{i1}, \dots, x_{i9}\}$, and let X_j be the set of variables in the j th column. Also let X^{kl} contain the variables corresponding to the cells in the 3×3 square in position k, ℓ , for $k, \ell \in \{1, 2, 3\}$. Suppose the content of cell i, j is preassigned a_{ij} for all $(i, j) \in S$. Then the problem can be formulated

$$\begin{aligned}
& \text{alldiff}(X_i), \text{alldiff}(X_j), i = 1, \dots, 9 \\
& \text{alldiff}(X^{k\ell}), k, \ell = 1, 2, 3 \\
& x_{ij} = a_{ij}, \text{all } (i, j) \in S \\
& x_{ij} \in \{1, \dots, 9\}, \text{all } i, j
\end{aligned} \tag{1}$$

Propagation is more effective if the `alldiffs` are filtered simultaneously. If the modeling system has a multiple `alldiff` constraint (not yet standard), the first two lines of (1) should be replaced with

$$\text{multiAlldiff} \left(\begin{array}{l} X_i, X_j, i = 1, \dots, 9, \\ X^{k\ell}, k, \ell = 1, \dots, 3 \end{array} \right) \tag{2}$$

Moreover, the `alldiffs` in the first line of (1) have special structure, in that they define a Latin square. If at some point a specialized filter is developed for Latin squares, a global constraint `LatinSquare(X)` can be added to the model, where X is the matrix of variables x_{ij} . The new constraint is redundant of the `alldiffs`, but redundancy can result in better propagation.

4.4 CP Modeling Guidelines

At least four principles should guide the formulation of CP models. They will also carry over into an integrated modeling framework.

1. A specially-structured subset of constraints should be replaced by a single global constraint that captures the structure, when a suitable one exists. This produces a more succinct model and can allow more effective filtering and propagation.
2. A global constraint should be replaced by a more specific one when possible, to exploit more effectively the special structure of the constraints.
3. The addition of redundant constraints (i.e., constraints that are implied by the other constraints) can improve propagation.
4. When two alternate formulations of a problem are available, including both (or parts of both) in the model may improve propagation. This is especially helpful when some constraints are hard to write in one formulation but suitable for the other. The dual formulations normally contain different variables, which should be defined in terms of each other through the use of *channeling* constraints.

The sudoku formulation illustrates Principle 1, because each `alldiff` constraint represents many inequations. If the `multiAlldiff` constraint is used as in (2), this again accords with Principle 1, because the `multiAlldiff` replaces twenty-seven `alldiff` constraints with overlapping variable sets.

The sudoku model (1) also illustrates Principle 3, because one of the `alldiff` constraints is redundant. If the numbers in rows 1 through 8 are all different, and those in columns 1 through 9 are all different, then row 9 necessarily contains nine

different numbers. Nonetheless it is good modeling practice to include a redundant `alldiff` constraint for row 9.

A practice of including redundant constraints may appear contrary to the goal of writing perspicuous models, because it makes the models longer. Yet redundant constraints can sometimes result in a more intuitive statement of the problem, as in the sudoku example. When redundancy is introduced by dual formulations, the two formulations can be separated in the model statement for clarity. It is natural (as well as computationally advantageous) to make each one as complete as possible, rather than arbitrarily dropping some constraints for the sake of removing redundancy.

In other cases, however, the modeler may become aware of redundant constraints after completing the formulation. Adding them complicates the model, as in the case of the `LatinSquare` constraint in the sudoku model. Yet the redundant constraints can be written separately for clarity, and the model continues to provide an explanatory theory—perhaps even a better theory, because it includes some “theorems” (redundant constraints) along with the “axioms” (original constraints). It can be a good exercise to think through some of the consequences of a model by deriving redundant constraints.

4.5 Example: Car Sequencing

A car sequencing example, adapted from [43], introduces three important global constraints. An assembly line makes fifty cars a day. To simplify matters, suppose that only four types of cars are manufactured, although in practice there could be hundreds. Each car type is defined by the options installed, as indicated in Table 1. In this example, the only available options are air conditioning and a sun roof. The table also shows how many cars of each type are required on a given day.

Table 1 Options and production level required for each car type.

Type	Air cond.	Sun roof	Production
a	no	no	20
b	yes	no	15
c	no	yes	8
d	yes	yes	7

The problem is to sequence the car types so as to meet production requirements while observing the capacity constraints of the assembly line. The constraints are that at most three cars in any sequence of five can be given air conditioning, and at most one in any sequence of three can be given a sun roof. Figure 2 shows a feasible solution for a smaller instance of the problem.

A natural decision variable for this problem is the type t_i of car to assign to each position i in the sequence. To make sure that the production requirements are met, a constraint is needed that counts the number of times each type occurs in the

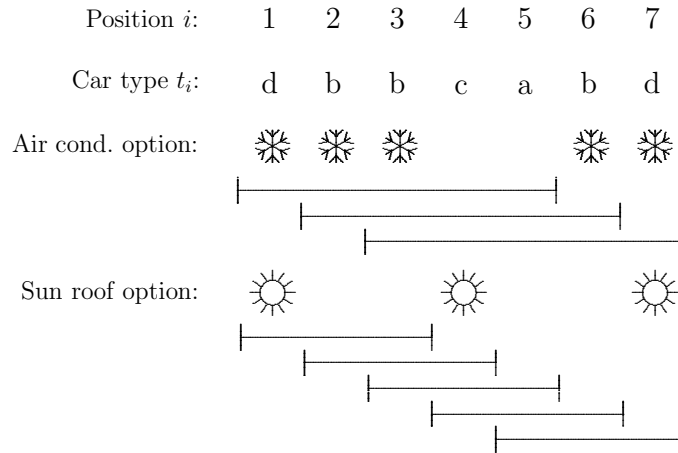


Fig. 2 A feasible solution for a small instance of the car sequencing problem in which the production requirements are $(D_a, D_b, D_c, D_d) = (1, 3, 1, 2)$. The brackets indicate subsequences in which assembly line capacity constraints are enforced.

sequence. The *cardinality* constraint, also known as the *generalized cardinality* or *gcc* constraint, serves the purpose [39, 40]. It is written $\text{cardinality}(X, \mathbf{v}, \mathbf{l}, \mathbf{u})$, where X is a set of variables, $\mathbf{v} = (v_1, \dots, v_k)$ a tuple of values (not necessarily numerical), $\mathbf{l} = (\ell_1, \dots, \ell_k)$ a tuple of lower bounds, and $\mathbf{u} = (u_1, \dots, u_k)$ a tuple of upper bounds. The constraint says that, for each i , at least ℓ_i and at most u_i of the variables in X must take the value v_i . The production requirements for the car sequencing problem can be written with a single cardinality constraint,

$$\text{cardinality}(\{t_1, \dots, t_{50}\}, (a, b, c, d), (20, 15, 8, 7), (20, 15, 8, 7)) \quad (3)$$

The `alldiff` constraint is a special case of a cardinality constraint in which each value is allowed to appear at most once. The `alldiff` constraints in the sudoku model (1) can therefore be replaced with cardinality constraints. It is best to follow Principle 2, however, by using the more specific `alldiff` constraint. Although filtering methods designed specifically for `alldiff` are likely to have the same result as `cardinality` filters (either typically achieves domain consistency), a `alldiff` filter generally runs faster.

The capacity constraints in the car sequencing problem limit the number of times each option occurs in subsequences of a specified length. However, the variables x_i indicate what type of car occurs in position i of the sequence, not which option. We know only that types b and d use the air conditioning option, and types c and d use the sun roof option. This requires a slightly different kind of counting than provided by the cardinality constraint. One must count the number of times type b or d occurs in a subsequence, and the number of times type c or d occurs. The *among* constraint was developed for such situations [10]. It can be written $\text{among}(X, S, \ell, u)$, where X is a set of variables, S a set of values (not necessarily numerical), and ℓ and u

are lower and upper bounds. The constraint requires that at least ℓ and at most u variables in X have a value that is among those in S .

Constraints of this kind can enforce the capacity constraints by applying them to every subsequence of five variables for air conditioners and every subsequence of three variables for sun roofs:

$$\begin{aligned} &\text{among}(\{t_j, \dots, t_{j+4}\}, \{b, d\}, 0, 3), j = 1, \dots, 46 \\ &\text{among}(\{t_j, t_{j+1}, t_{j+2}\}, \{c, d\}, 0, 1), j = 1, \dots, 48 \end{aligned}$$

Although this is a correct formulation, it fails to recognize that the `among` constraints are closely related. They apply to overlapping subsequences of variables of equal length. Faster and more effective filters can be designed if one exploits this structure (Principle 1). For this reason, scheduling formulations frequently use the constraint `sequence`($\mathbf{x}, S, q, \ell, u$), where $\mathbf{x} = (x_1, \dots, x_n)$ is a tuple of variables, S a set of values, and q an integer [10, 20, 32]. The constraint says that in any sequence of q consecutive variables x_j, \dots, x_{j+q-1} , at least ℓ and at most u variables must take a value in S . The capacity constraints can now be more compactly written as the second and third constraints in the complete car sequencing model that appears below.

$$\begin{aligned} &\text{cardinality}(\{t_1, \dots, t_{50}\}, \{a, b, c, d\}, (20, 15, 8, 7), (20, 15, 8, 7)) \\ &\text{sequence}((t_1, \dots, t_{50}), \{b, d\}, 5, 0, 3) \\ &\text{sequence}((t_1, \dots, t_{50}), \{c, d\}, 3, 0, 1) \\ &t_i \in \{a, b, c, d\}, i = 1, \dots, 50 \end{aligned} \tag{4}$$

4.6 Example: Employee Scheduling

Employee scheduling is one of the most successful application areas for CP, and several well-studied global constraints have been developed for it. A small nurse scheduling problem, adapted from [23, 41], illustrates dual formulations (Principle 4) and channeling constraints.

Four nurses are to be assigned to eight-hour shifts. Shift 1 is the daytime shift, while shifts 2 and 3 occur at night. The schedule repeats itself every week. In addition,

1. Every shift is assigned exactly one nurse.
2. Each nurse works at most one shift a day.
3. Each nurse works at least five days a week.
4. To ensure a certain amount of continuity, no shift can be staffed by more than two different nurses in a week.
5. To avoid excessive disruption of sleep patterns, a nurse cannot work different shifts on two consecutive days.
6. Also, a nurse who works shift 2 or 3 must do so at least two days in a row.

We can formulate the problem by assigning nurses to shifts or by assigning shifts to nurses. Rather than select one alternative, Principle 4 recommends using both. This not only sidesteps a difficult modeling decision but can result in faster solution than either model would permit if used in isolation. Table 2 displays a feasible assignment of nurses to shifts, and Table 3 displays the equivalent assignment of shifts to nurses.

We first write a model that assigns nurses to shifts. Let w_{sd} be the nurse assigned to shift s on day d , where the domain of w_{sd} is the set of nurses $\{A, B, C, D\}$. Constraint 1 is satisfied automatically, by virtue of the notation. Constraint 2 says in effect that three different nurses work each day:

$$\text{alldiff}(w_{1d}, w_{2d}, w_{3d}), \quad d = 1, \dots, 7 \quad (5)$$

Because there are 21 shifts in a week, constraint 3 implies that each nurse will work at least five and at most six days a week. This is readily expressed with a cardinality constraint:

$$\text{cardinality}(W, (A, B, C, D), (5, 5, 5, 5), (6, 6, 6, 6)) \quad (6)$$

where W is the set of variables w_{sd} .

Constraint 4 requires that for each shift s , at most two nurses are assigned to the variables w_{s1}, \dots, w_{s7} corresponding to the seven days of the week. Thus while constraint 2 counts the number of times a value occurs, constraint 3 counts the number of different values that occur. One therefore uses the $\text{nvalues}(X, \ell, u)$ global constraint [8, 11], which requires that the variables in X take at least ℓ and at most u different values.

$$\text{nvalues}(\{w_{s1}, \dots, w_{s7}\}, 1, 2), \quad s = 1, 2, 3$$

Table 2 Employee scheduling viewed as assigning workers to shifts. A feasible solution is shown.

	Sun	Mon	Tue	Wed	Thu	Fri	Sat
<i>Shift 1</i>	A	B	A	A	A	A	A
<i>Shift 2</i>	C	C	C	B	B	B	B
<i>Shift 3</i>	D	D	D	D	C	C	D

Table 3 Employee scheduling viewed as assigning shifts to workers (shift 0 corresponds to a day off). A feasible solution is shown.

	Sun	Mon	Tue	Wed	Thu	Fri	Sat
<i>Worker A</i>	1	0	1	1	1	1	1
<i>Worker B</i>	0	1	0	2	2	2	2
<i>Worker C</i>	2	2	2	0	3	3	0
<i>Worker D</i>	3	3	3	3	0	0	3

Because `alldiff` is a special case of `nvalues` with $\ell = u = n$, one could have used `nvalues` rather than `alldiff` to express constraint 2. However, again following Principle 2, it is better to use the more specific constraint.

The remaining constraints are difficult to express in terms of variables w_{sd} , and there are no obvious global constraints that capture them. One can therefore move to an alternative model in which variable t_{id} is the shift assigned to nurse i on day d , and where shift 0 denotes a day off. Constraint 1 is satisfied in this model by assigning different shifts to the nurses on each day:

$$\text{alldiff}(t_{Ad}, t_{Bd}, t_{Cd}, t_{Dd}), \quad d = 1, \dots, 7$$

Constraint 2 is automatically satisfied by virtue of the notation. Constraint 3 says that each nurse gets at most two days off in a week (and at least one, because the other nurses get at most two days off):

$$\text{cardinality}(\{t_{i1}, \dots, t_{i7}\}, 0, 1, 2), \quad i = A, B, C, D$$

Thus Constraints 1–3 are enforced in both models, a redundancy that can speed solution.

Constraint 4 is not easily expressed in terms of the variables t_{id} . However, these variables are suitable for Constraints 5 and 6, which refer to patterns of shifts that each nurse may work. This allows one to use the *stretch* constraint [12, 17, 37], written `stretch`(x, v, ℓ, u, P), where $x = (x_1, \dots, x_n)$ and P is a set of *patterns*. The constraint requires, for each v_i , that any *stretch* of variables with value v_i in the sequence x_1, \dots, x_n has length at least ℓ_i and at most u_i . A stretch is a maximal subsequence of consecutive variables that take the same value. A pattern is a pair (v, v') of distinct values. The constraint requires that whenever a stretch of value v immediately precedes a stretch of value v' , the pair (v, v') occurs in P . Constraints 5 and 6 can be written

$$\text{stretch-cycle}((t_{i1}, \dots, t_{i7}), (2, 3), (2, 2), (6, 6), P), \quad i = A, B, C, D$$

where P consists of all patterns that include a day off

$$P = \{(s, 0), (0, s) \mid s = 1, 2, 3\}$$

One must use the cyclic version of `stretch` because the schedule is cyclic, and a stretch can extend across the weekend.

Constraints 5 and 6 can also be written as a `regular` constraint [38], which generalizes `stretch` and can often be processed at least as efficiently. A `regular` constraint models allowable sequences of values as expressions in a regular language, which can in turn be represented by a deterministic finite automaton. However, it is not straightforward to represent cyclic stretches with an automaton, due in part to the necessity of introducing additional variables. The more specific `stretch` constraint is therefore used in the present model.

Finally, the two models are linked with channeling constraints that relate the variables w_{sd} with the variables t_{id} :

$$\begin{aligned} w_{t_{id}d} &= i, \quad \text{all } i, d \\ t_{w_{sd}s} &= s, \quad \text{all } s, d \end{aligned} \tag{7}$$

The first constraint says that the nurse assigned to the shift to which nurse i is assigned on day d should be nurse i , and analogously for the second constraint. For (7) to be valid, one must interpret w_{0d} as the nurse that is off duty on day d . When a problem instance requires more than one nurse to be off duty on a given day, it is necessary to define a dummy shift (representing a day off) for each nurse.

Note that variables occur as subscripts in the constraints (7). This powerful modeling device does not occur in MILP but is frequent in CP. A CP or integrated modeling system can parse an expression of the form x_t , where t is a variable with a finite domain, by replacing it with a new variable y and adding the constraint `element($t, (x_{v_1}, \dots, x_{v_k}), y$)`. The values v_1, \dots, v_k belong to the domain of t , and the `element` constraint requires that y take the same value as the variable in the item x_{v_1}, \dots, x_{v_k} whose subscript is the value of t . Thus the modeling system interprets the constraints (7) as

$$\begin{aligned} y_{id} &= i, \quad \text{all } i, d \\ z_{sd} &= s, \quad \text{all } s, d \\ \text{element}(t_{id}, (w_{0d}, \dots, w_{3d}), y_{id}), & \quad \text{all } i, d \\ \text{element}(w_{sd}, (t_{Ad}, \dots, t_{Dd}), z_{sd}), & \quad \text{all } s, d \end{aligned}$$

Filtering methods have been developed for the `element` constraint as for any other global constraint, and they allow the channeling constraints (7) to improve propagation when the dual formulation is used. The dual formulation also speeds solution by allowing all the elements of the problem to be written as high-level global constraints (in one formulation or the other), which appears to be impossible if either formulation is used alone.

4.7 Assignment and Circuit Problems

Assignment and circuit problems illustrate several lessons in CP, MILP, and integrated modeling. The *assignment* problem can be viewed as one in which the objective is find a minimum cost assignment of m tasks n to workers ($m \leq n$). Each task is assigned to a different worker, and no two workers are assigned the same task. If $m < n$, some of the workers will not be assigned tasks. If assigning worker i to task j incurs cost c_{ij} , the problem is simply stated:

$$\begin{aligned} \min \sum_i c_{ix_i} \\ \text{alldiff}(x_1, \dots, x_n), \quad x_i \in D_i, \quad i = 1, \dots, n \end{aligned} \quad (8)$$

where x_i is the worker assigned to task i . The domain of x_i is the set of workers that can do task i .

The assignment problem can be viewed as a sequencing problem in which the cost depends on which item appears in each position of the sequence. The *circuit* problem, by contrast, is a sequencing problem in which the cost depends on which item follows which in the sequence. The sequence is circular in the sense that the first item is viewed as following the last. The problem can be written

$$\begin{aligned} \min \sum_i c_{x_i x_{i+1}} \\ \text{alldiff}(x_1, \dots, x_n), \quad x_i \in D_i, \quad i = 1, \dots, n \end{aligned} \quad (9)$$

where $c_{n,n+1}$ is identified with c_{n1} . If each domain D_i is $\{1, \dots, n\}$, the problem can be viewed as seeking a shortest hamiltonian circuit on a complete directed graph, where c_{ij} is the length of edge (i, j) . This is the famous *traveling salesman problem*, in which the vertices are cities that the salesman must visit before returning to his home city. One can deal with edges (i, j) that are missing from the graph by setting $c_{ij} = \infty$. Unfortunately, filtering the `alldiff` has no effect because every domain is complete.

Although the two formulations (8) and (9) are almost identical, they represent vastly different problems. The assignment problem is very easy to solve, whereas the traveling salesman problem is notoriously hard.

An alternate formulation for the circuit problem uses the `circuit`(y_1, \dots, y_n) constraint [13, 28]. Here y_i denotes the vertex after vertex i in the circuit, and the constraint requires that the y_i s describe a hamiltonian circuit. The circuit problem can be written

$$\begin{aligned} \min \sum_i c_{iy_i} \\ \text{circuit}(y_1, \dots, y_n), \quad y_i \in D'_i, \quad i = 1, \dots, n \end{aligned} \quad (10)$$

This formulation has the advantage that missing edges can be explicitly represented in the domains. Domain D'_i of y_i contains j if and only if (i, j) is an edge of the graph. Filtering the `circuit` constraint can therefore have an effect. On the other hand, achieving domain consistency is much harder for `circuit` than for `alldiff` [23, 28].

Rather than choose between formulations (9) and (10), Principle 4 recommends using both. The channeling constraints are $x_{i+1} = y_{x_i}$ for $i = 1, \dots, n-1$, and $x_1 = y_{x_n}$. The ‘‘first’’ vertex in the circuit can be arbitrarily defined to be vertex 1 (i.e., $x_1 = 1$) without loss of generality. Propagation of incomplete domains D'_i through these constraints can remove elements from the domains D_i , so that `alldiff` filtering can now have an effect. Both objective functions can be used as bounds on cost, so that the dual formulation becomes

$$\begin{aligned}
& \min z \\
& z \geq \sum_i c_{x_i x_{i+1}}, \quad z \geq \sum_i c_{y_i y_i} \\
& \text{alldiff}(x_1, \dots, x_n), \quad x_i \in D_i, \quad i = 1, \dots, n \\
& \text{circuit}(y_1, \dots, y_n), \quad y_i \in D'_i, \quad i = 1, \dots, n \\
& x_1 = y_{x_n} = 1, \quad x_{i+1} = y_{x_i}, \quad i = 1, \dots, n-1 \\
& x_i \in \{1, \dots, n\}, \quad y_i \in D_i, \quad i = 1, \dots, n.
\end{aligned} \tag{11}$$

5 MILP Modeling

An MILP model is an optimization problem in which the objective function is linear and the constraints are linear inequalities. Some or all of the variables are restricted to integer values. A major advantage of such a model is that it provides a ready-made linear relaxation, obtained simply by dropping the integrality constraints. Solution of this continuous relaxation provides a bound on the optimal value that can be very useful in a solution algorithm, for example by pruning the search tree. In addition, the solution of the relaxation may be integral, in which case it solves the original MILP. Even when the solution is nonintegral, it may provide valuable clues on how to conduct a branching search.

Because integer-valued variables are present in an MILP model, its continuous relaxation can often be tightened by the addition of *cutting planes*. These are *valid* inequalities that are satisfied by all the feasible solutions of the MILP but “cut off” part of the feasible set of the continuous relaxation. MILP solution methods rely heavily on relaxation and cutting plane technology. Both general-purpose and special-purpose families of cutting planes have been developed, the latter for problems with special structure (see [33] for a survey).

A wide variety of problems can be given MILP models, although it is often not obvious how to do so, and the MILP formulation may require additional variables and constraints that obscure the underlying structure of the problem. Yet even when an MILP format is not appropriate for the original model, it may be advantageous for the solver to reformulate parts of the problem as an MILP to harness the power of the relaxation technology. The best strategy for using MILP formulations is therefore to write constraints in MILP format when it is natural to do from a modeling point of view, and to allow the solver to reformulate other constraints as MILPs when this benefits the solution process. The present section focuses on how to write MILP formulations, while Section 6 explores the role of these formulations in integrated modeling.

The key to building MILP formulations is to recognize the structure of feasible sets that are representable by MILP models. This section therefore begins with a presentation of MILP representability theorems. These lead to guidelines for writing MILP models, after which several examples are presented. Further examples and discussion of MILP modeling can be found in [47, 48, 49, 50].

5.1 MILP Representability

An MILP model can always be written in the form

$$\min \{ cx \mid Ax \geq b, x \in \mathbb{R}^n \times \mathbb{Z}^p \}$$

Note that variables x_1, \dots, x_n are real valued, while variables x_{n+1}, \dots, x_{n+p} are required to take integer values. Because of the restriction to linear inequalities, the question arises as to what kind of problems can be represented with MILP models. The answer is known, due to theorems proved in [27] and generalized in [25].

MILP representability is defined in such a way that the MILP model is allowed to contain auxiliary variables in addition to the variables of the original problem space. Such variables can be either continuous or discrete and frequently appear in practical models. Thus a subset S of \mathbb{R}^{n+p} is *MILP representable* if it is the projection onto x of the feasible set of a model of the form

$$\begin{aligned} Ax + Bu + Dy &\geq b \\ x \in \mathbb{R}^n \times \mathbb{Z}^p, u \in \mathbb{R}^m, \delta \in \{0, 1\}^q \end{aligned}$$

Some of the auxiliary variables are real-valued (u_j) and some are binary (δ_j).

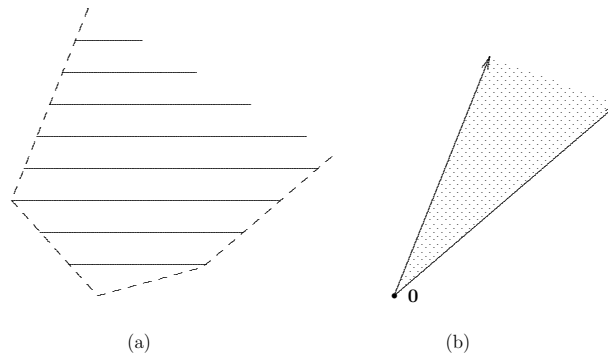


Fig. 3 (a) A mixed integer polyhedron Q (horizontal lines) where $Q = P \cap (\mathbb{R} \times \mathbb{Z})$ and P is bounded by the dashed line. (b) The recession cone of Q .

To state the representability theorems, some definitions are necessary. Let a *mixed integer polyhedron* be the nonempty intersection of any polyhedron in \mathbb{R}^{n+p} with $\mathbb{R}^n \times \mathbb{Z}^p$. Such a polyhedron is illustrated in Fig. 3. A vector $r \in \mathbb{R}^{n+p}$ is a *recession direction* of a polyhedron $P \in \mathbb{R}^{n+p}$ if one can go forever in the direction r without leaving P . That is, for any $x \in P$, $x + \alpha r \in P$ for all $\alpha \geq 0$. A rational vector r is a recession direction of a mixed integer polyhedron Q if it is a recession direction of a polyhedron whose intersection with $\mathbb{R}^n \times \mathbb{Z}^p$ is Q . The *recession cone* of a mixed integer polyhedron is the set of all its recession directions (Fig. 3). The following is proved in [25].

Theorem 1. *A nonempty set $S \in \mathbb{R}^n \times \mathbb{Z}^p$ is MILP representable if and only if it is the union of finitely many mixed integer polyhedra in $\mathbb{R}^n \times \mathbb{Z}^p$ having the same recession cone.*

Each mixed integer polyhedron in the finite union can be described by a system of linear inequalities. The theorem therefore states in effect that any MILP representable set can be modeled as a disjunction of linear systems (i.e., at least one of the systems must be satisfied). This suggests a principle for creating MILP models, because practical problems frequently present a set of alternative actions or situations. If the alternatives can be modeled by linear systems, then the problem takes the form of a disjunction of linear systems. An MILP model can now be written, provided the linear systems describe mixed integer polyhedra with the same recession cone. The recession cone condition is normally satisfied in practice by adding a few innocuous constraints.

There remains the question, however, as to how to convert a disjunction of linear systems into an MILP model. There are two standard ways: a big- M formulation and a convex hull formulation. Both require 0-1 auxiliary variables, and the convex hull formulation requires continuous auxiliary variables as well.

First, the big- M formulation. Consider any nonempty MILP representable set $S \in \mathbb{R}^n \times \mathbb{Z}^p$. From Theorem 1, S is the union of mixed integer polyhedra $Q_k = P_k \cap (\mathbb{R}^n \times \mathbb{Z}^p)$ for $k \in K$, where the Q_k s have the same recession cone. If each polyhedron P_k is described by the linear system $A^k x \geq b^k$, S has the *sharp big- M formulation*

$$\begin{aligned} A^k x &\geq b^k - M^k(1 - \delta_k), \quad k \in K \\ x &\in \mathbb{R}^n \times \mathbb{Z}^p, \quad \delta_k \in \{0, 1\}, \quad k \in K \end{aligned} \quad (12)$$

where

$$M^k = b^k - \min_{\ell \neq k} \left\{ \min_x \left\{ A^k x \mid A^\ell x \geq b^\ell, x \in \mathbb{R}^n \times \mathbb{Z}^p \right\} \right\} \quad (13)$$

Note that there are binary auxiliary variables δ_k . When $\delta_k = 1$, the k th linear system $A^k x \geq b^k$ is enforced. When $\delta_k = 0$, the k th system is deactivated by subtracting a vector M^k of large numbers from the right-hand side. The formulation is sharp in the sense that the components of M^k are chosen to be as small as possible. Let Q_k be the mixed integer polyhedron $\{x \in \mathbb{R}^n \times \mathbb{Z}^p \mid A^k x \geq b^k\}$.

Theorem 2. *A set $S \in \mathbb{R}^n \times \mathbb{Z}^p$ is MILP representable if and only if it has a sharp big- M representation (12), where $S = \bigcup_{k \in K} Q_k$.*

The convex hull formulation introduces continuous auxiliary variables by disaggregating x into a sum $\sum_{k \in K} x^k$, where each x^k corresponds to Q_k . The formulation is

$$\begin{aligned} x &= \sum_{k \in K} x^k \\ A^k x^k &\geq b^k \delta_k, \quad k \in K \\ \sum_{k \in K} \delta_k &= 1 \\ x &\in \mathbb{R}^n \times \mathbb{Z}^p, \quad \delta_k \in \{0, 1\}, \quad k \in K \end{aligned} \quad (14)$$

Theorem 3. *A set $S \in \mathbb{R}^n \times \mathbb{Z}^p$ is MILP representable if and only if it can be formulated as (14), where $S = \bigcup_{k \in K} Q_k$. Furthermore, (14) is a convex hull formulation if $A^k x \geq b^k$ is a convex hull formulation of Q_k for each $k \in K$.*

Model (14) is a *convex hull formulation* of S when its continuous relaxation describes the closure of the convex hull of S . The continuous relaxation is obtained by dropping the integrality constraints; that is, by replacing the last line of (14) with

$$x \in \mathbb{R}^{n+p}, \quad 0 \leq \delta_k \leq 1, \quad k \in K$$

The system $A^k x \geq b^k$ is a convex hull formulation of Q_k when it describes the closure of the convex hull of Q_k . A convex hull formulation has the tightest possible continuous relaxation.

It can be shown that (14) provides a convex hull relaxation of S when the polyhedra Q_k do not have the same recession cone, even though (14) is not a representation of S in this case. That is,

Theorem 4. *The continuous relaxation of (14), when projected onto x , describes the closure of the convex hull of $S = \bigcup_{k \in K} Q_k$ even when the Q_k s do not have the same recession cone.*

Similarly, the big- M formulation (12) provides a valid relaxation of S even when the polyhedra Q_k do not have the same recession cone.

5.2 Example: Fixed-Charge Problems

A simple fixed-charge problem illustrates MILP model construction. The cost z of manufacturing quantity x of some product is zero when $x = 0$ and is $f + cx$ when $x > 0$, where f is the fixed cost and c the unit variable cost. The problem is to minimize cost.

There are two alternatives, corresponding to a zero or positive production level, each giving rise to a different cost calculation. The feasible set S is illustrated in Fig. 4. It is described by a disjunction of linear systems that, in this case, contain only continuous variables:

$$\left(\begin{array}{l} x = 0 \\ z \geq 0 \end{array} \right) \vee \left(\begin{array}{l} x \geq 0 \\ z \geq cx + f \end{array} \right) \quad (15)$$

The disjuncts respectively describe the polyhedra P_1 and P_2 in Fig. 4(a). The recession cone of P_1 is P_1 itself, and the recession cone of P_2 is $\{(z, x) \mid z \geq cx \geq 0\}$. Thus, by Theorem 1, S is not MILP representable. For example, one can write the big- M formulation (13) as

$$\begin{array}{ll} 0 \leq x \leq M_1^1 \delta & x \geq 0 \\ z \geq -M_2^1 \delta & z \geq cx + f - M_2^2(1 - \delta) \\ \delta \in \{0, 1\} & \end{array}$$

where the 0-1 variables δ_1, δ_2 are replaced by $1 - \delta$ and δ (because they sum to one). But this formulation is not well defined because from (13),

$$M_1^1 = -\min\{-x \mid x \geq 0, z \geq cx + f\} = \infty$$

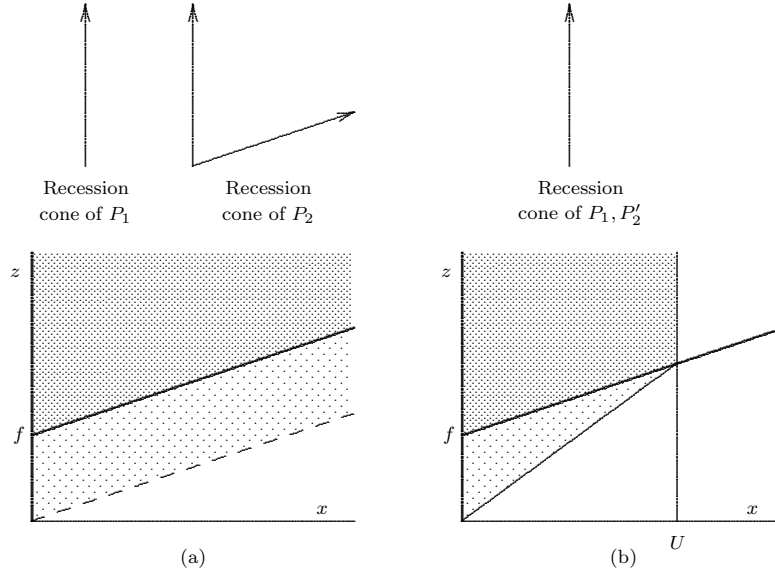


Fig. 4 (a) Feasible set of a fixed-charge problem, consisting of the union of polyhedra P_1 (heavy vertical line) and P_2 (darker shaded area). (b) Feasible set of the same problem with the bound $x \leq U$, where P_2' is the darker shaded area. In both (a) and (b), the convex hull of the feasible set is the entire shaded area.

Also the convex hull formulation (14) of (15) becomes

$$\begin{aligned} x &= x^1 + x^2 & z &= z^1 + z^2 \\ x^1 &= 0 & x^2 &\geq 0 \\ z^1 &\geq 0 & z^2 &\geq cx^2 + f\delta \\ \delta &\in \{0, 1\} \end{aligned} \tag{16}$$

One can eliminate the constraint $x^1 = 0$ (and the corresponding aggregation constraint $x = x^1 + x^2$) by replacing x^2 with x , and similarly for the constraint $z^1 \geq 0$. So (16) simplifies to

$$x \geq 0, z \geq cx + f\delta, \delta \in \{0, 1\}$$

which does not correctly represent the feasible set. However, its continuous relaxation correctly describes the closure of the convex hull of the feasible set, as pre-

dicted by Theorem 4. For if one replaces $\delta \in \{0, 1\}$ with $0 \leq \delta \leq 1$ and projects out δ , the result is $z \geq cx$, $x \geq 0$. This is illustrated in Fig. 4(a).

The recession cones can be equalized by placing an upper bound U on x in the second disjunct of (15). The recession cone of each of the resulting polyhedra P_1, P'_2 is the same, as illustrated in Fig. 4(b), and the feasible set is therefore MILP representable. The big- M formulation becomes

$$\begin{aligned} 0 \leq x &\leq M_1^1 \delta & 0 \leq x &\leq U + M_1^2(1 - \delta) \\ z &\geq -M_2^1 \delta & z &\geq cx + f - M_2^2(1 - \delta) \\ \delta &\in \{0, 1\} \end{aligned} \quad (17)$$

From (13), $(M_1^1, M_2^1, M_1^2, M_2^2) = (U, -f, 0, f)$. So (17) simplifies to

$$0 \leq x \leq U\delta, \quad z \geq cx + f\delta, \quad \delta \in \{0, 1\} \quad (18)$$

which is a correct formulation. The convex hull formulation becomes

$$\begin{aligned} x &= x^1 + x^2 & z &= z^1 + z^2 \\ 0 &= x^1 & 0 &\leq x^2 \leq U\delta \\ z^1 &\geq 0 & z^2 &\geq cx^2 + f\delta \\ \delta &\in \{0, 1\} \end{aligned}$$

which again simplifies to (18). In this case, the big- M formulation happens to be a convex hull formulation.

5.3 MILP Modeling Guidelines

Theorems 1–3 imply that an MILP model can be formulated by regarding the problem as a disjunction of linear systems corresponding to discrete alternatives. Some or all of the variables in the linear systems may be required to be integer valued. The disjunction is then converted to a big- M or convex hull formulation. In practice, it may be convenient to identify several disjunctions, each of which is converted to a set of MILP constraints. This suggests the following guidelines for MILP modeling:

1. Try to conceive the problem as posing one or more choices among discrete alternatives.
2. Formulate each alternative using a system of linear inequalities in which some variables may be restricted to be integer.
3. Write each choice of alternatives as a disjunction of linear systems. Some of the disjunctions may have only one disjunct, indicating that there is only one alternative.
4. Adjust the linear systems so that in each disjunction, all of the disjuncts describe mixed integer polyhedra with the same recession cone.

5. Convert each disjunction to a big- M or a convex hull formulation. The convex hull formulation normally has a tighter continuous relaxation, unless the big- M model happens to be a convex hull formulation as well. The big- M formulation normally contains fewer variables, particularly when there are many disjuncts, unless the convex hull formulation can be simplified. The choice between the formulations rides on whether the tighter relaxation is worth the overhead of additional variables.

Unfortunately, not all useful MILP models evolve naturally from this disjunctive approach. An example is the standard 0-1 model for the circuit (traveling salesman) problem, discussed below.

5.4 Example: Facility Location

A capacitated facility location problem illustrates the above modeling guidelines. There are m possible locations for factories, and n customers who obtain products from the factories. A factory installed at location i incurs fixed cost f_i and has capacity C_i . Each customer j has demand D_j . Goods are shipped from factory i to customer j on trucks, each with capacity K_{ij} , and each incurring a fixed cost c_{ij} . The problem is to decide which facilities to install, and how to supply the customers, so as to minimize total cost.

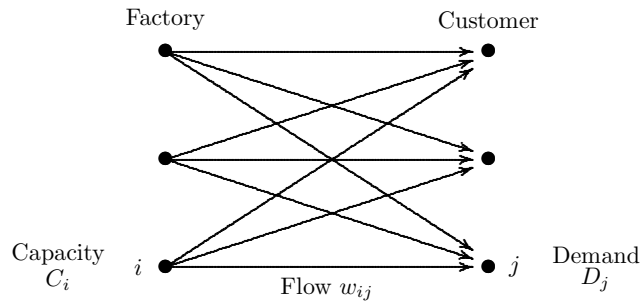


Fig. 5 A facility location problem.

The basic decision, for each location i , is whether to install a factory at that location. This presents two discrete alternatives that can be represented as a disjunction. To describe each alternative with linear inequalities, let x_{ij} be the quantity of goods shipped from factory i to customer j , and let w_{ij} be the number of trucks on which they are transported. Then if z_i is the total cost incurred at location i , the two alternatives for location i are represented by the disjunction

$$\left(\begin{array}{l} \sum_j x_{ij} \leq C_i \\ 0 \leq x_{ij} \leq K_{ij}w_{ij}, \text{ all } j \\ z_i \geq f_i + \sum_j c_{ij}w_{ij} \\ w_{ij} \in \mathbb{Z}, \text{ all } j \end{array} \right) \vee \left(\begin{array}{l} x_{ij} = 0, \text{ all } j \\ z_i \geq 0 \end{array} \right) \quad (19)$$

The alternative on the left corresponds to installing a factory at location i . The first constraint enforces the factory's capacity limit, and the second does the same for the truck capacities. The third constraint computes the cost incurred at location i . Note that each w_{ij} is integer valued, which means that this disjunct describes a mixed integer polyhedron. The disjunct on the right corresponds to the case in which no factory is installed at location i .

Customer demand can be satisfied by imposing the constraint $\sum_i x_{ij} \geq D_j$ for each customer j . Each of these constraints can be viewed as a separate disjunction with only one alternative. The objective is to minimize total cost, given by $\sum_i z_i$.

The polyhedra defined by the two disjuncts in (19) have different recession cones. The cone for the first polyhedron is $\{(x_i, w_i, z_i) \mid x_i = 0, w_i \geq 0, z_i \geq \sum_j c_{ij}w_{ij}\}$ where $x_i = (x_{i1}, \dots, x_{in})$ and $w_i = (w_{i1}, \dots, w_{in})$, while the cone for the second is $\{(x_i, w_i, z_i) \mid x_i = 0, z_i \geq 0\}$. However, if the innocuous constraints $w_{ij} \geq 0$ and $z_i \geq \sum_j c_{ij}w_{ij}$ are added to the second disjunct, the two disjuncts have the same recession cone. This yields a disjunction that can be given an MILP model:

$$\left(\begin{array}{l} \sum_j x_{ij} \leq C_i \\ 0 \leq x_{ij} \leq K_{ij}w_{ij}, \text{ all } j \\ z_i \geq f_i + \sum_j c_{ij}w_{ij} \\ w_{ij} \in \mathbb{Z}, \text{ all } j \end{array} \right) \vee \left(\begin{array}{l} x_{ij} = 0, \text{ all } j \\ w_{ij} \geq 0, \text{ all } j \\ z_i \geq \sum_j c_{ij}w_{ij} \end{array} \right) \quad (20)$$

If it is too hard to identify the recession cones, one can always ensure that every disjunct has the same recession cone (namely, the origin) by imposing fixed lower and upper bounds on every variable.

Using (14), the convex hull formulation of (20) is

$$\begin{aligned} x_{ij} &= x_{ij}^1 + x_{ij}^2, \quad w_{ij} = w_{ij}^1 + w_{ij}^2, \quad z_i = z_i^1 + z_i^2, \quad \text{all } j \\ \sum_j x_{ij}^1 &\leq C_i \delta_i & x_{ij}^2 &= 0, \quad w_{ij}^2 \geq 0 \quad \text{all } j \\ 0 \leq x_{ij}^1 &\leq K_{ij}w_{ij}^1, \quad \text{all } j & z_i^2 &\geq \sum_j c_{ij}w_{ij}^2 \\ z_i^1 &\geq f_i \delta_i + \sum_j c_{ij}w_{ij}^1 & \delta_i &\in \{0, 1\}, \quad w_{ij} \in \mathbb{Z}, \quad \text{all } j \end{aligned}$$

Because the auxiliary 0-1 variables corresponding to the two disjuncts sum to one, they can be written as δ_i and $1 - \delta_i$; the latter does not appear because the right-hand sides in the second disjunct are all zero. The constraints $x_{ij}^2 = 0$ can be dropped (along with the aggregation constraints $x_{ij} = x_{ij}^1 + x_{ij}^2$) if x_{ij}^1 is replaced by x_{ij} , and

similarly for the constraints $w_{ij}^2 \geq 0$ and $z_i^2 \geq \sum_j c_{ij} w_{ij}^2$. Because z_i can be replaced by $f_i \delta_i + \sum_j c_{ij} w_{ij}$ in the objective function $\sum_j z_j$, the complete MILP model becomes

$$\begin{aligned}
& \min \sum_i \left(f_i \delta_i + \sum_j c_{ij} w_{ij} \right) \\
& \sum_j x_{ij} \leq C_i \delta_i, \quad \text{all } i \\
& 0 \leq x_{ij} \leq K_{ij} w_{ij}, \quad \text{all } i, j \\
& \sum_i x_{ij} \geq D_j, \quad \text{all } j \\
& \delta_i \in \{0, 1\}, \quad w_{ij} \in \mathbb{Z}, \quad \text{all } i, j
\end{aligned} \tag{21}$$

Although each disjunction (20) is given a convex hull formulation in the MILP model (21), the model as a whole is not a convex hull formulation of the problem.

Using (12), the big- M model for the disjunction (20) is

$$\begin{aligned}
& \sum_j x_{ij} \leq C_i + M_{1i}^1 (1 - \delta_i) & 0 \leq x_{ij} \leq M_{1ij}^2 \delta_i, \quad \text{all } i, j \\
& 0 \leq x_{ij} \leq K_{ij} w_{ij} + M_{2ij}^1 (1 - \delta_i), \quad \text{all } j & w_{ij} \geq -M_{2ij}^2 \delta_i, \quad \text{all } i, j \\
& z_i \geq f_i \delta_i + \sum_j c_{ij} w_{ij} - M_{3i}^1 (1 - \delta_i) & z_i^2 \geq \sum_j c_{ij} w_{ij} - M_{3i}^2 \delta_i, \quad \text{all } i \\
& \delta_i \in \{0, 1\}, \quad w_{ij} \in \mathbb{Z}, \quad \text{all } j
\end{aligned} \tag{22}$$

It can be verified from (13) that $M_{1i}^1 = -C_i$, $M_{1ij}^2 = C_i$, and all the other big- M s are zero in the sharp formulation. The big- M formulation (22) therefore reduces to the same model as the convex hull formulation.

5.5 Example: Car Sequencing

It will be useful to compare a MILP model of the car sequencing problem with the CP model developed earlier. In this problem, there are four discrete alternatives at each position i in the manufacturing sequence—car types a, b, c, and d. Each alternative implies a choice of options. If we let $AC_i = 1$ when air conditioning is installed at position i , and $SR_i = 1$ when a sun roof is installed, the four alternatives can be written as follows for each position i :

$$\left(\begin{array}{l} AC_i = 0 \\ SR_i = 0 \end{array} \right) \vee \left(\begin{array}{l} AC_i = 1 \\ SR_i = 0 \end{array} \right) \vee \left(\begin{array}{l} AC_i = 0 \\ SR_i = 1 \end{array} \right) \vee \left(\begin{array}{l} AC_i = 1 \\ SR_i = 1 \end{array} \right)$$

The convex hull formulation of this disjunction is

$$\begin{aligned}
AC_i &= AC_i^a + AC_i^b + AC_i^c + AC_i^d \\
SR_i &= SR_i^a + SR_i^b + SR_i^c + SR_i^d \\
AC_i^a &= 0, \quad AC_i^b = \delta_{ib}, \quad AC_i^c = 0, \quad AC_i^d = \delta_{id} \\
SR_i^a &= 0, \quad SR_i^b = 0, \quad SR_i^c = \delta_{ic}, \quad SR_i^d = \delta_{id} \\
\delta_{ia} + \delta_{ib} + \delta_{ic} + \delta_{id} &= 1 \\
\delta_{ij} &\in \{0, 1\}, \quad j = a, b, c, d
\end{aligned}$$

This simplifies to

$$\begin{aligned}
AC_i &= \delta_{ib} + \delta_{id}, \quad SR_i = \delta_{ic} + \delta_{id} \\
\delta_{ib} + \delta_{ic} + \delta_{id} &\leq 1 \\
\delta_{ij} &\in \{0, 1\}, \quad j = b, c, d
\end{aligned}$$

The complete MILP model can now be written by combining the above with constraints that meet demand and observe the assembly line capacity constraints:

$$\begin{aligned}
AC_i &= \delta_{ib} + \delta_{id}, \quad SR_i = \delta_{ic} + \delta_{id}, \quad i = 1, \dots, 50 \\
\delta_{ib} + \delta_{ic} + \delta_{id} &\leq 1, \quad i = 1, \dots, 50 \\
\delta_{ij} &\in \{0, 1\}, \quad j = b, c, d, \quad i = 1, \dots, 50 \\
\sum_{i=1}^{50} \delta_{ia} &= 20, \quad \sum_{i=1}^{50} \delta_{ib} = 15, \quad \sum_{i=1}^{50} \delta_{ic} = 8, \quad \sum_{i=1}^{50} \delta_{id} = 7, \quad i = 1, \dots, 50 \\
\sum_{j=i}^{i+4} AC_j &\leq 3, \quad i = 1, \dots, 46 \\
\sum_{j=j}^{i+2} SR_j &\leq 1, \quad j = 1, \dots, 48
\end{aligned} \tag{23}$$

5.6 Network Flow Models

The continuous relaxation of a MILP model sometimes describes an integral polyhedron, meaning that every vertex of the polyhedron has integral coordinates. In such cases one can easily solve the MILP model by solving its continuous relaxation with a linear programming algorithm that finds a vertex solution. There is a strong incentive to use an MILP formulation when it has this integrality property.

In practice, the most common MILP models with the integrality property are capacitated network flow models, of which assignment models are a special case. The matrix of constraint coefficients in these problems is totally unimodular, which ensures that the continuous relaxation of the model describes an integral polytope if all the right-hand sides are integral.

A network flow model is defined on a directed network in which the net supply S_i of flow at each node i is given. If (i, j) represents an arc that is directed from node i to node j , then C_{ij} is the arc capacity and variable y_{ij} represents the flow on (i, j) . If

arc (i, j) is missing from the network, one can nonetheless include y_{ij} in the model and set $C_{ij} = 0$. The flow model is

$$\begin{aligned} \sum_j y_{ij} - \sum_j y_{ji} &= S_i, \quad \text{all } i \\ 0 \leq y_{ij} &\leq C_{ij}, \quad \text{all } i, j \end{aligned} \quad (24)$$

There is typically an objective function that measures cost, such as $\sum_{ij} c_{ij} y_{ij}$, where c_{ij} is the unit cost of sending flow on arc (i, j) .

Due to total unimodularity, the model (24) describes an integral polytope if the supplies and capacities are all integral. This means that if the flows are restricted to be integral, as may occur, for example, when they consist of discrete vehicles, the resulting MILP model can be solved by solving its continuous relaxation (24). This is a particularly easy problem to solve because there are specialized algorithms for computing minimum cost network flows.

The network model does not evolve from the disjunctive modeling process described above. However, it is best conceived as a linear programming model that becomes an MILP model when integrality constraints are imposed on the flows. The MILP modeling guidelines assume a knowledge of linear modeling and attempt only to show how MILP models can be built from disjunctions of linear inequality systems.

5.7 Assignment Problems

The assignment problem discussed in Section 4.7 assigns m tasks to n workers ($m \leq n$). It is a special case of a network flow problem, which means that the MILP model is totally unimodular. This provides a strong incentive to use an MILP formulation at some stage of the solution process.

The flow network corresponding to an assignment problem is bipartite, as illustrated in Fig. 6. If $m < n$, then dummy task nodes are created so that supply balances demand. The cost c_{ij} is set to zero for a dummy task i . A unit flow $y_{ij} = 1$ indicates that worker i is assigned task j . The flow model therefore reduces to

$$\begin{aligned} \sum_{j=1}^n y_{ij} &= \sum_{j=1}^n y_{ji} = 1, \quad i = 1, \dots, n \\ y_{ij} &\geq 0, \quad \text{all } i, j \end{aligned} \quad (25)$$

This model can be solved very rapidly with specialized algorithms. Obviously the solution is meaningful only if each $y_{ij} \in \{0, 1\}$, but this is assured by total unimodularity.

The assignment model (25) has more of a combinatorial flavor than the more general network flow model, and it can in fact be derived disjunctively using the MILP modeling guidelines. The assignment problem implies the disjunction $\bigvee_j (x_i = j)$

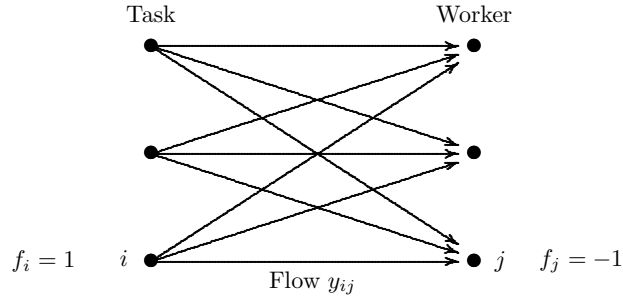


Fig. 6 A flow model for an assignment problem.

for each i , because each task i is assigned a different worker. When $m = n$, it also implies the disjunction $\bigvee_i (x_i = j)$ for each j , because each worker j is assigned a task. The convex hull formulations of these disjunctions are

$$\begin{aligned} x_i &= \sum_j j y_{ij}, \quad \sum_j y_{ij} = 1, \quad y_{ij} \in \{0, 1\}, \quad \text{all } i, j \\ x_i &\geq j \delta_{ji}, \quad \sum_i \delta_{ji} = 1, \quad \delta_{ij} \in \{0, 1\}, \quad \text{all } i, j \end{aligned} \quad (26)$$

But $y_{ij} = \delta_{ji}$, because either variable is 1 when task i is assigned to worker j . So (26) simplifies to the standard model (25), plus a formula $x_i = \sum_j j y_{ij}$ that defines the original variables x_i in terms of the auxiliary variables y_{ij} .

5.8 Circuit Problems

1

Circuit problems have been given several MILP formulations [36], but by far the most popular is the subtour elimination formulation. If binary variable $y_{ij} = 1$ when vertex j immediately follows i in the hamiltonian circuit, then the traveling salesman problem on n cities can be written

$$\begin{aligned} \min & \sum_{ij} c_{ij} y_{ij} \\ \sum_j y_{ij} &= \sum_j y_{ji} = 1, \quad \text{all } i \\ \sum_{(i,j) \in \delta(S)} y_{ij} &\geq 1, \quad \text{all } S \subset \{1, \dots, n\} \text{ with } 2 \leq |S| \leq n-1 \\ y_{ij} &\in \{0, 1\}, \quad \text{all } i, j \end{aligned} \quad (27)$$

where S is a subset of vertices and $\delta(S)$ is the set of edges (i, j) for which $i \in S$ and $j \notin S$. The assignment constraints (line 2) ensure that exactly one vertex precedes, and exactly one vertex follows, each vertex in the tour. Line 3 contains the subtour

elimination constraints, which rule out circuits on fewer than n vertices. This is accomplished by requiring, for each proper subset S of the vertices, that at least one edge in the circuit connect a vertex in S with a vertex outside S .

In practice, the formulation is not actually written out because it contains exponentially many constraints. Rather, a traveling salesman solver generates *separating* subtour elimination constraints as they are needed. A separating constraint is one that cuts off the solution of the current linear relaxation without cutting off any feasible solutions. Several families of strong cutting planes have been identified for the problem, along with separation heuristics. A survey of this work can be found in [6].

5.9 Example: Sudoku Puzzles

The sudoku problem can be formulated with assignment constraints, although a large number of 0-1 variables are necessary to do so. Let 0-1 variable $y_{ijt} = 1$ when digit t appears in cell i, j . Let $J_{k\ell}$ be the set of cells (i, j) in the 3×3 square in position k, ℓ . Then the MILP model is

$$\begin{aligned}
 \sum_{j=1}^9 y_{ijt} &= \sum_{j=1}^9 y_{jit} = 1, \quad i, t = 1, \dots, 9 & (a) \\
 \sum_{(i,j) \in J_{k\ell}} y_{ijt} &= 1, \quad k, \ell = 1, 2, 3, \quad t = 1, \dots, 9 & (b) \\
 \sum_{t=1}^9 y_{ijt} &= 1, \quad i, j = 1, \dots, 9 & (c) \\
 y_{ija_{ij}} &= 1, \quad \text{all } (i, j) \in F & (d) \\
 y_{ijt} &\in \{0, 1\}, \quad \text{all } i, j, t & (28)
 \end{aligned}$$

Constraints (a) enforce the `alldiff` condition for the rows and columns, and constraints (b) do the same for the 3×3 squares. Constraints (c) ensure that exactly one digit appears in each cell. Constraints (d) take care of the preassigned cells.

6 Integrated Modeling

It is time to address the question as to how the seemingly incompatible modeling styles of CP and MILP can be integrated. CP relies on the use of global constraints, so that it can exploit problem substructure with its filtering algorithms and propagation methods. MILP requires that the problem be reduced to linear inequality constraints, so that it can obtain linear relaxations and strengthen them with cutting planes.

A simple solution is to follow the CP practice of building a model around global constraints, but to create new global constraints that represent sets of inequalities. Different constraints can be designed for inequality sets with different kinds of special structure, so that the solver can take advantage of this structure when it generates cutting planes. A general-purpose constraint can be defined for an MILP inequality set that has no particular structure.

Because MILP models can always be constructed by conceiving the problem as a disjunction of linear system, it may be more natural in many cases to write the problem in disjunctive form rather than translate the disjunctions to MILP models. The solver can make the translation automatically. It is therefore useful for a general-purpose MILP global constraint to accept disjunctions of linear systems as well as a single linear system.

The proposal, therefore, is that the modeler write parts of the problem with CP-style global constraints and other parts with global constraints that represent structured sets of linear inequalities, depending on which is more natural and best reveals the structure of the problem to the solver.

Such a model allows the solver to take full advantage of both CP and MILP solution technology. CP-style global constraints are explicitly present, which allows the solver to apply its repertory of filtering and propagation techniques. Inequality constraints are also explicitly present, identified by their structure, which allows the solver to generate MILP-based relaxations and cutting planes. This already an advance over commercial MILP systems, which do not permit the user to identify most types of special structure in subsets of constraints.

Furthermore, MILP relaxation technology can be applied even when a constraint is written in CP style. If a CP constraint in the model has an alternative MILP formulation, the solver always has the option of generating the MILP formulation along with any useful cutting planes, for the sake of obtaining an relaxation. Alternatively, the solver may generate a relaxation that is based on a polyhedral analysis of that particular constraint, rather than on an MILP model of it. The overall advantage of this scheme is that it allows the solver to exploit the wide variety of filtering and relaxation methods that appear in the CP, MILP, and CP-AI-OR literatures.

There are some technical issues that must be clarified if the generated inequality relaxations are to replicate the full advantage of MILP technology. One is a variable mapping issue that arises when MILP translations of global constraints create auxiliary variables. When the auxiliary variables map to the same set of variables in the original model, the solver must use the same auxiliary variables in all the translations. Also, care must be taken when simplifying the individual MILP translations, so that the combined translations provide a correct model.

Not only are the full computational resources of CP and MILP simultaneously available, but they are mutually reinforcing in all the ways that have been described in the literature and the remainder of this book. CP-based filtering, for example, results in tighter MILP models and relaxations, which in turn provide bounds for more effective domain reduction. Thus an integrated model supports integrated problem solving.

The remainder of this section begins with a summary of guidelines for integrated modeling. It then reviews the problems that have been so far introduced and indicates how they might be formulated in an integrated modeling system. It shows how relaxations can be generated and addresses the technical issues just mentioned. Because hybrid methods often use decomposition methods to combine solution techniques, the section concludes with two illustrations of how decomposition can be introduced into a model. This not only tells the solver how the problem may be decomposed, but it may allow the model to be written with high-level global constraints that would not otherwise be applicable.

6.1 Integrated Modeling Guidelines

Because it is proposed that an integrated model be built around global constraints much as in CP modeling, CP modeling guidelines continue to apply. They must be augmented, however, with principles for incorporating MILP-based global constraints. In view of the above discussion, the following principles seem appropriate.

1. A specially-structured subset of constraints should be replaced by a single global constraint that captures the structure, when a suitable one exists.
2. A global constraint can be one familiar to the CP community or a collection of inequalities whose structure has been studied in the MILP literature.
3. Constraints that are more naturally formulated as disjunctions of linear systems should normally be left in this form, rather than converting them to MILP models.
4. A global constraint should be replaced by a more specific one when possible.
5. Redundant constraints can improve propagation. However, the solver should be relied upon to generate the MILP equivalent of a CP constraint in the model.
6. When two formulations of a problem are natural and intuitive, both (or parts of both) may be included in the model to improve propagation. This is especially helpful when some constraints are hard to write in one formulation but suitable for the other. Channeling constraints should be used to define variables in the two formulations in terms of each other.
7. Decomposition can be introduced into a model when it would alert the solver to a useful decomposition strategy, or when it would permit the use of high-level global constraints that would not otherwise be applicable.

6.2 Example: Facility Location

The facility location problem is naturally expressed as an MILP model. The primary elements of the problem are a disjunction of alternatives (install the factory or not), and additional linear inequalities (customer demand). Following Principle 3, it should be written in disjunctive form rather than converting it to an MILP.

It is therefore convenient to invent a general-purpose MILP global constraint `linear($\bigvee_k S_k$)`, which enforces the disjunction of the linear systems S_k . A special case is `linear(S)`, which enforces a single linear system S . Any integrality restrictions on the variables are given when the variable domains are specified. The facility location model can be written

$$\begin{aligned} & \text{linear} \left(\left(\begin{array}{l} \sum_j x_{ij} \leq C_i \\ 0 \leq x_{ij} \leq K_{ij} w_{ij}, \text{ all } j \\ z_i \geq f_i + \sum_j c_{ij} w_{ij} \\ w_{ij} \in \mathbb{Z}, \text{ all } j \end{array} \right) \vee \left(\begin{array}{l} x_{ij} = 0, \text{ all } j \\ z_i \geq 0 \end{array} \right) \right), \text{ all } i \\ & \text{linear} \left(\begin{array}{l} \min \sum_i z_i \\ \sum_i x_{ij} \geq D_j, \text{ all } j \end{array} \right) \\ & x_{ij}, z_i \in \mathbb{R}, w_{ij} \in \mathbb{Z}, \text{ all } i, j \end{aligned} \quad (29)$$

The objective function is placed in a `linear` constraint because it can be viewed as a linear inequality, namely $\sum_i z_i \leq U$, where U is any upper bound on the minimum cost.

As noted in Section 5.4, the mixed integer polyhedra described by the two terms of the disjunction in (29) have different recession cones. It is therefore impossible for the solver to generate an MILP model for the disjunction. However, the purpose of creating an MILP model is to obtain its continuous relaxation. The continuous relaxations of the convex hull and big- M formulations are valid relaxations for the problem, even though the formulations do not correctly model the problem.

This suggests that the `linear` constraint can be accompanied by a parameter that has three possible values.

Relaxation only. The solver simply generates a valid relaxation of the disjunction based on the big- M or convex hull formulation. It does not strengthen the relaxation with cutting planes, because these formulations may not be valid MILP models.

User-defined recession cones. The solver assumes that the modeler has equalized the recession cones by hand, perhaps simply by placing reasonable bounds on the variables. The solver creates a valid MILP model and perhaps strengthens it with cutting planes.

System-defined recession cones. The solver automatically equalizes the recession cones, again perhaps by placing bounds on the variables. It is a research issue how these bounds can be adjusted automatically so that the resulting relaxation is reasonably tight.

6.3 Network Flow Problems

An MILP model is the preferred choice for a network flow problem, not only because it is a natural and intuitive formulation, but also because it has the substantial advantage of total unimodularity. However, there is no need to write out the individual network flow constraints, and even if one did, the solver might not recognize the network flow structure. Principles 1 and 2 call for a global constraint to represent the inequality set. Such a constraint might be written

$$\text{networkFlow}(\mathbf{y}, \mathbf{f}, \mathbf{C}) \quad (30)$$

where \mathbf{y} is a matrix of flow variables y_{ij} , \mathbf{f} is a vector of net supplies f_i , and \mathbf{C} is a matrix of arc capacities C_{ij} . On encountering a constraint of the form (30), the modeling system automatically generates the MILP model (24).

A minimum cost network flow problem can be stated by minimizing the objective function $\sum_{ij} y_{ij}$ subject to (30).

6.4 Assignment Problems

The assignment problem is naturally written in its CP form (8), using the `alldiff` constraint. The MILP model (25) is also useful due to its total unimodularity, but the solver should be relied upon to generate it (Principle 5). Generating this model raises the important technical issue of variable mapping.

The solver generates the MILP assignment constraints (25) when it encounters the `alldiff` constraint in the CP model. But this alone is not adequate. Fast algorithms for the MILP assignment model use an objective function $\sum_{ij} c_{ij} y_{ij}$ that is expressed in terms of the 0-1 variables y_{ij} , not the objective function $\sum_i c_{ix_i}$ that appears in the CP model (8).

The difficulty can be addressed as follows. Recall that the variable subscript in $\sum_i c_{ix_i}$ is parsed by generating an `element` constraint. The CP model actually sent to the solver is therefore

$$\begin{aligned} \min \sum_i z_i \\ \text{element}(x_i, (c_{i1}, \dots, c_{in}), z_i), \quad i = 1, \dots, n \\ \text{alldiff}(x_1, \dots, x_n), \quad x_i \in D_i, \quad i = 1, \dots, n \end{aligned} \quad (31)$$

The solver can now create MILP translations for the `element` constraints as well as the `alldiff`. The i th element constraint can be given an MILP model by writing a convex hull formulation of the disjunction $\bigvee_j (z_i = c_{ij})$:

$$z_i = \sum_{j=1}^n c_{ij} y'_{ij}, \quad \sum_{j=1}^n y'_{ij} = 1, \quad y'_{ij} \in \{0, 1\}, \quad j = 1, \dots, n$$

The following MILP model is therefore generated for (31):

$$\begin{aligned}
 & \min \sum_i z_i \\
 & z_i = \sum_{j=1}^n c_{ij} y'_{ij}, \quad \sum_{j=1}^n y'_{ij} = 1, \quad i = 1, \dots, n \\
 & \sum_{j=1}^n y_{ij} = \sum_{j=1}^n y_{ji} = 1, \quad i = 1, \dots, n \\
 & y'_{ij} \in \{0, 1\}, \quad i = 1, \dots, n
 \end{aligned} \tag{32}$$

The solver relates the new 0-1 variables y_{ij}, y'_{ij} to the original variables x_i by way of internal variable mapping constraints

$$x_i = \sum_{j=1}^n j y_{ij} = \sum_{j=1}^n j y'_{ij}, \quad i = 1, \dots, n$$

Problem (32) is an assignment problem with the integrality property, provided the solver can recognize that the 0-1 variables y_{ij} and y'_{ij} are semantically the same. This is accomplished by mapping the variables x_i to the *same* set of 0-1 variables y_{ij} whenever those variables are created for an MILP translation. That is, the y_{ij} s should be *global* variables rather than local to a specific MILP translation. Then if the original model contains only an `alldiff` constraint and an objective function, the solver can exploit the total unimodularity of the MILP translation once it verifies that the translated objective function has the right form.

The practice of mapping variables in the original model to global variables in the MILP translations can be called *global variable mapping*. Such a practice ensures that one can use the succinct CP model (8) for an assignment problem without sacrificing any of the advantages of an MILP model. However, MILP variables that are not mapped to an original variable should remain local. This is illustrated in the car sequencing example below.

6.5 Circuit Problems

From a modeling point of view, a CP formulation of the circuit (traveling salesman) problem is superior to an MILP formulation. A CP formulation is more natural and contains only one constraint, as opposed to exponentially many constraints in the most popular MILP model. The question remains, however, as to which of the two CP formulations is better in an integrated modeling context—the `alldiff` formulation (9) or the `circuit` formulation (10).

The `circuit` formulation seems more intuitive, because it is conceived in terms of a circuit, as opposed to an assignment. The variables y_i in the `circuit` formula-

tion refer to the next vertex in the hamiltonian circuit, which allows one to indicate missing edges from the graph by removing elements from the variable domains.

The `circuit` formulation is superior from a technical point of view as well, because it allows for more effective propagation and relaxation. Consider the situation with the `alldiff` formulation. No filtering can take place because the variable domains are complete. In addition, the totally unimodular MILP model of `alldiff` constraint is of little use in the context of a circuit problem, because it has no variables in common with the MILP translation of the objective function. To see this, recall that the `alldiff`(x_1, \dots, x_n) constraint is translated

$$\sum_{j=1}^n y_{ij} = \sum_{j=1}^m y_{ji} = 1, \quad y_{ij} \in \{0, 1\}, i = 1, \dots, n \quad (33)$$

where the x_i s are mapped to the auxiliary variables y_{ij} by

$$x_i = \sum_j j y_{ij} \quad (34)$$

However, the objective function $\sum_i c_{x_i x_{i+1}}$ of (9) is parsed as

$$\sum_i z_i, \quad \text{element}((x_i, x_{i+1}), \mathbf{C}, z_i), \quad i = 1, \dots, n \quad (35)$$

where \mathbf{C} is the matrix of coefficients c_{ij} . The i th element constraint sets z_i equal to the element of \mathbf{C} in position (x_i, x_{i+1}) . This can be translated to an MILP model by writing a convex hull formulation of the disjunction $\bigvee_{jk} (z_i = c_{jk})$:

$$z_i = \sum_{jk} c_{jk} \delta_{ijk}, \quad \sum_{jk} \delta_{ijk} = 1, \quad \delta_{ijk} \in \{0, 1\}, \quad \text{all } j, k$$

where the x_i s are mapped to the auxiliary variables δ_{ijk} by

$$\delta_{ijk} = 1 \Leftrightarrow (x_i, x_{i+1}) = (j, k) \quad (36)$$

Because (34) and (36) are different mappings, the solver does not (and should not) identify the variables δ_{ijk} with the variables y_{ij} . So the MILP translation (33) of the `alldiff` has no variables in common with the MILP translation (35) of the objective function, and the resulting MILP model of the problem is useless as a relaxation.

The `circuit` formulation, on the other hand, allows for filtering, even though achieving domain consistency is much harder than for `alldiff`. Also the `circuit` constraint allows one to harness the advanced relaxation methods that have been developed for the MILP formulation (27) of the constraint. The solver would not actually generate the entire MILP formulation, because of its exponential size, but would generate separating subtour elimination constraints and strong separating cuts as needed, much as a specialized traveling salesman solver would do. The difficulty that arose with the `alldiff` constraint does not occur here, because the variables

y_{ij} that occur in the MILP translation (27) of `circuit` also occur in the MILP translation of the objective function $\sum_i c_{iy_i}$. The latter is simply $\sum_{ij} c_{ij}y_{ij}$, and both MILP translations use the same variable mapping $y_i = \sum_j j y_{ij}$.

In addition, it is possible to write a relaxation of the `circuit` constraint solely in terms of the original variables y_i , provided they take numerical values. It is argued in [29] that a proper choice of these values can exploit structure in the objective function.

The dual model (11), which uses both `alldiff` and `circuit` constraints, may be advantageous when some other constraints in the problem are best expressed in terms of the x_i variables (Principle 6). In such cases, the MILP-based relaxation of `alldiff` may be useful even though it does not connect with the objective function.

6.6 Example: Sudoku Puzzles

The sudoku puzzle is most naturally modeled with `alldiff` constraints, as in (1). It may also be advantageous for the solver to generate the MILP model (34), which is not totally unimodular but may provide a useful relaxation. The solver can easily generate the assignment constraints for each `alldiff`. If the solver uses global variable mapping, the combined assignment constraints provide the desired relaxation.

MILP-based relaxations for `alldiff` contain 0-1 variables y_{ij} , not the original variables x_i . Polyhedral relaxations in the original variables have been studied for the `alldiff` constraint [21, 51] and the `multiAlldiff` constraint [31, 30], provided those variables take numerical values such as $1, \dots, n$. The solver may choose to generate these in addition to the MILP model, particularly if the y_{ij} s do not occur in the relaxations of the objective function or other constraints.

6.7 Example: Car Sequencing

The CP-based formulation (45) of the car sequencing model is very appropriate for an integrated modeling context, due to its simplicity and the fact that it harnesses the filtering power of `cardinality` and `sequence` constraints. The MILP model (23) can provide a useful relaxation, but generating the model raises another important technical point.

The `cardinality` constraint in the CP model (45) translates immediately to the desired MILP constraints in (23), namely

$$\sum_{i=1}^{50} \delta_{ia} = 20, \quad \sum_{i=1}^{50} \delta_{ib} = 15, \quad \sum_{i=1}^{50} \delta_{ic} = 8, \quad \sum_{i=1}^{50} \delta_{id} = 7, \quad i = 1, \dots, 50 \quad (37)$$

using the variable mapping

$$(\delta_{ij} = 1) \Rightarrow (t_i = j), \quad j = a, b, c, d \quad (38)$$

There may appear to be a difficulty in translating the two sequence constraints, however. For the first sequence constraint in (45), disjunctions of the form

$$(AC_i = 0) \vee (AC_i = 1) \vee (AC_i = 0) \vee (AC_i = 1) \quad (39)$$

are converted to the MILP model

$$\begin{aligned} AC_i &= \delta_{ib} + \delta_{id}, \quad i = 1, \dots, 50 \\ \delta_{ia} + \delta_{ib} + \delta_{ic} + \delta_{id} &= 1 \\ \sum_{j=i}^{i+4} AC_j &\leq 3, \quad i = 1, \dots, 46 \end{aligned} \quad (40)$$

which simplifies to

$$\begin{aligned} AC_i &= \delta_{ib} + \delta_{id}, \quad i = 1, \dots, 50 \\ \delta_{ib} + \delta_{id} &\leq 1 \\ \sum_{j=i}^{i+4} AC_j &\leq 3, \quad i = 1, \dots, 46 \end{aligned} \quad (41)$$

If global variable mapping is used, the variables δ_{ij} in (41) are the same those in (37), because they are mapped to the same original variables t_i using (38). Similarly, the second sequence constraint yields the MILP model

$$\begin{aligned} SR_i &= \delta_{ic} + \delta_{id}, \quad i = 1, \dots, 50 \\ \delta_{ia} + \delta_{ib} + \delta_{ic} + \delta_{id} &= 1 \\ \sum_{j=i}^{i+2} SR_j &\leq 1, \quad i = 1, \dots, 48 \end{aligned} \quad (42)$$

which simplifies to

$$\begin{aligned} SR_i &= \delta_{ic} + \delta_{id}, \quad i = 1, \dots, 50 \\ \delta_{ic} + \delta_{id} &\leq 1 \\ \sum_{j=i}^{i+2} SR_j &\leq 1, \quad i = 1, \dots, 48 \end{aligned} \quad (43)$$

The variables AC_i and SR_i are local to either MILP translation because they are not mapped to any of the original variables.

The difficulty is that when the MILP formulations (41) and (43) are merged, the constraints involving $\delta_{i,j}$ s are not equivalent to the corresponding constraints in the desired MILP model (23):

$$\begin{aligned} AC_i &= \delta_{ib} + \delta_{id}, \quad SR_i = \delta_{ic} + \delta_{id}, \quad i = 1, \dots, 50 \\ \delta_{ib} + \delta_{ic} + \delta_{id} &\leq 1, \quad i = 1, \dots, 50 \end{aligned} \quad (44)$$

The two inequalities $\delta_{ib} + \delta_{id} \leq 1$ and $\delta_{ic} + \delta_{id} \leq 1$ are not equivalent to the inequality $\delta_{ib} + \delta_{ic} + \delta_{id} \leq 1$ in (44).

The problem is that (40) and (41) are equivalent only in the sense that they specify the same disjunction (39). Because the auxiliary variables δ_{ijs} are global variables, the two formulations must be equivalent in the space that includes the δ_{ijs} as well as the AC_i s. In general, when an MILP translation based on disjunctions is simplified, it must be simplified to a formulation that is equivalent in the auxiliary variables as well as the variables in the disjuncts, if the auxiliary variables are global. It is therefore essential to use the formulation (40) rather (41), and similarly to use (42) rather than (43). If this is done, the result is the desired MILP model (23).

6.8 Example: Employee Scheduling

The CP idiom is especially well suited for employee scheduling problems, because several global constraints are expressly designed for this purpose—such as the `stretch` constraint in the CP model of Section 4.6. Writing an MILP model for `stretch` is not straightforward and should not be attempted in the original model. How the solver might generate an MILP translation presents an interesting research issue, as does the task of finding linear relaxations that are not based on MILP formulations.

6.9 Decomposition: Machine Assignment and Scheduling

A machine assignment and scheduling problem illustrates the usefulness of decomposition in modeling (Principle 7). A set of n jobs are to be assigned to m machines. The jobs assigned to each machine must be scheduled so that they do not overlap and are processed within their time windows. The time window for each job j consists of a release time r_j and a deadline d_j . Each job j has a processing time of p_{ij} on machine i . For simplicity, suppose the cost of assigning job j to machine i is a constant c_{ij} .

The assignment portion of the problem is modeled simply by letting x_j be the machine assigned to job j . For the scheduling component, a number of well-studied global constraints are available. The simplest is a disjunctive scheduling constraint, which might be written `disjunctive` ($\{t_1, \dots, t_n\}$), where t_j is the start time of job j . The constraint requires that start times be set so that each job runs inside its time window and starts after the previous job finishes. The time window of each job j is enforced by setting the domain of each x_j to the interval $[r_j, d_j - p_j]$, where p_j is the processing time of job j . Filtering algorithms for the constraint use edge finding and other methods to reduce the domains of the t_j s [].

One would like to write a model for the assignment and scheduling problem that uses `disjunctive` constraints:

$$\begin{aligned}
& \text{linear} \left(\begin{array}{l} \min \sum_j c_{x_j j} \\ r_j \leq t_j \leq d_j - p_{x_j j}, \text{ all } j \end{array} \right) \\
& \text{disjunctive}(\{t_j \mid x_j = i\}), \text{ all } i \\
& t_j \in \mathbb{R}, x_j \in \{1, \dots, m\}, \text{ all } j
\end{aligned} \tag{45}$$

Each `disjunctive` constraint schedules the jobs on one of the machines. The difficulty is that the `disjunctive` constraints are not well defined until the values of the x_j s are known, because the variable list in the constraints depends on the x_j s. In principle, an enhanced `disjunctive` constraint could be designed to filter the t_j and x_j domains simultaneously, but there is apparently no such enhanced constraint in current systems.

By introducing decomposition into the model, however, one can retain the `disjunctive` constraints. One approach is to define a global constraint that specifies a subproblem in which the values of certain variables are assumed to be known. The constraint could be written

$$\text{subproblem}(X, C_1, \dots, C_k)$$

to enforce constraints C_1, \dots, C_k after the values of the variables in set X are fixed. The model (45) can be written

$$\begin{aligned}
& \text{linear} \left(\min \sum_j c_{x_j j} \right) \\
& \text{subproblem} \left(\begin{array}{l} \{x_1, \dots, x_n\}, \\ \text{disjunctive}(\{t_j \mid x_j = i\}), \text{ all } i, \\ \text{linear}(r_j \leq t_j \leq d_j - p_{x_j j}, \text{ all } j) \end{array} \right) \\
& t_j \in \mathbb{R}, x_j \in \{1, \dots, m\}, \text{ all } j
\end{aligned} \tag{46}$$

The variables x_j function as constants inside the subproblem, which means that the `disjunctive` constraints are well defined. Also the time window constraints set the variable domains to ranges appropriate for the assigned machine.

The solver may be able to exploit the decomposition structure that is identified in the model. In this case, it might use a Benders method, because the outer problem and the subproblem use disjoint sets of variables (the x_j s and the t_j s, respectively). The `disjunctive` constraint would be associated with an algorithm that generates a logic-based Benders cut. The cut is added to the main problem, which is then re-solved. For example, if the scheduling problem on machine i is infeasible, the `disjunctive` filter may discover that a small subset J of the jobs assigned to machine i are responsible for the infeasibility. Then a Benders cut $\{x_j \neq i \mid j \in J\}$ can be added to the constraint set outside the subproblem. This constraint set is then re-solved to obtain new trial values of the x_j s, and so on until an optimal solution is obtained. This process has been used in a number of contexts (e.g., [22, 24]) and is discussed further in Chapter 4.

An important generalization of disjunctive scheduling is *cumulative* scheduling, which allows tasks to be run simultaneously subject to one or more resource constraints. Each task consumes each resource at a certain rate, and there is a limit on the total rate of consumption—a limit that may vary with time. Several versions of the `cumulative` global constraint exist for this situation, and logic-based Benders cuts have been developed for some of them as well as for the `disjunctive` constraint. Although filtering technology for `disjunctive` and `cumulative` is highly developed [7], it may be useful to generate MILP formulations or linear relaxations that are not based on MILP models [23].

6.10 Decomposition: Routing and Frequency Assignment

A final example, adapted from [43], illustrates decomposition in a more complex setting. The arcs of a directed network represent optical fibers with limited capacity (Fig. 7). There are requests for communication channels to be established between certain pairs of nodes. The channels must be routed over the network, and they must be assigned frequencies so that all the channels passing along any given arc have different frequencies. There are a limited number of frequencies available, and it may not be possible to establish all the channels requested. The objective is to maximize the number of channels established.

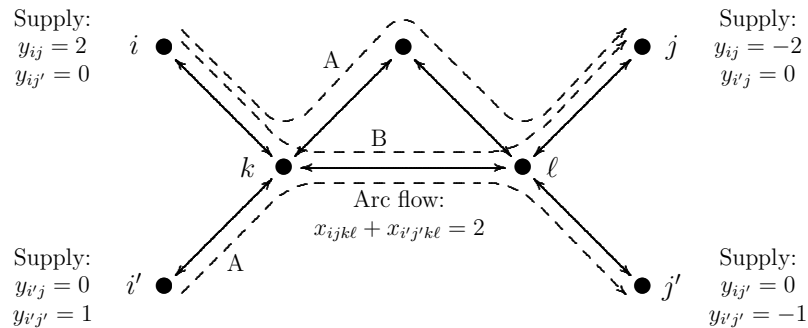


Fig. 7 A message routing and frequency assignment problem. Two message channels are requested from i to j and one from i' to j' . The arcs have capacity 2, and frequencies A, B are available. The dashed lines show an optimal solution.

The problem can be decomposed into its routing and frequency-assignment elements. The routing problem is amenable to an MILP formulation, and the frequency assignment problem is conveniently written with `alldiff` constraints—provided that a `subproblem` constraint is used to fix the flows before the frequency assignment problem is stated.

The routing problem is similar to the well-known multi-commodity network flow problem. This problem generalizes the capacitated network flow problem discussed

above by distinguishing several commodities that must be transported over the network. There is a net supply of each commodity at each node, and the total flow on each arc must be within the arc capacity. In the message routing problem, each origin-destination pair represents a different commodity.

The message routing problem is not identical to the multicommodity flow problem because the net supplies are not fixed, due to the fact that some requests may not be satisfied. As a result, one would not be able to use a global constraint designed for multicommodity flow problems, even if one existed. Nonetheless, it is fairly easy to write the MILP constraints directly.

For each pair of nodes (i, j) , let D_{ij} be the number of i -to- j channels requested (possibly zero). A key decision is which requests to honor, and one can therefore let integer variable y_{ij} be the number of channels from i to j that are actually established. (It is assumed here that different channels from i to j can be routed differently.) The net supply of commodity (i, j) is y_{ij} at node i , $-y_{ij}$ at node j , and zero at other nodes. Let $x_{ijk\ell}$ be the flow of commodity (i, j) on arc (k, ℓ) , and $C_{k\ell}$ the capacity of the arc. To simplify notation, arcs missing from the network can be viewed as arcs with a capacity of zero. The flow model is

$$\text{linear } \left(\begin{array}{l} \max \sum_{ij} y_{ij} \\ \sum_{\ell \neq i} x_{ij\ell} - \sum_{k \neq i} x_{ijk} = y_{ij}, \text{ all } i, j \\ \sum_{\ell \neq j} x_{ij\ell} - \sum_{k \neq j} x_{ijk} = -y_{ij}, \text{ all } i, j \\ \sum_{\ell \neq i, j, k} x_{ijk\ell} - \sum_{\ell \neq i, j, k} x_{ij\ell k} = 0, \text{ all } i, j, k \text{ with } k \neq i, j \\ \sum_{ij} x_{ijk\ell} \leq C_{k\ell}, \text{ all } k, \ell \\ x_{ijk\ell} \geq 0, \text{ all } i, j, k, \ell \\ 0 \leq y_{ij} \leq D_{ij}, \text{ all } i, j \end{array} \right)$$

$$x_{ijk\ell}, y_{ij} \in \mathbb{Z}, \text{ all } i, j, k, \ell$$

Once the communications channels are routed, a frequency f_{ij} can be assigned to each pair i, j so that the frequencies assigned to channels passing through any given arc are all different. The model is therefore completed by writing

$$\text{subproblem } \left(\begin{array}{l} \{x_{ijk\ell}, \text{ all } i, j, k, \ell\}, \\ \text{alldiff}(\{f_{ij} \mid x_{ijk\ell} > 0\}), \text{ all } k, \ell \end{array} \right)$$

$$f_{ij} \in F, \text{ all } i, j \text{ with } i \neq j$$

where F is the set of available frequencies.

7 Conclusions

A scheme for integrating CP and MILP modeling styles has been proposed, in which structured sets of MILP inequalities appear as global constraints alongside the global constraints of CP. It remains to assess, however, whether a scheme of this sort can deliver the advantages of both CP and MILP modeling in a single framework.

It has already been argued that the full computational resources of both CP and MILP are available in an integrated setting, where they can also be combined for greater effect. CP-style global constraints continue to appear in the model whenever they provide the best modeling approach, and they can be subjected to any filtering or propagation methods available to a CP solver. MILP relaxation technology can also be brought to bear, even in cases where it would not be applied in a commercial MILP solver, because structured sets of inequalities are identified. Even when constraints are not written in inequality form, the integrated solver can generate MILP translations when they are useful. This results in an MILP-based relaxation that is as effective as any obtained from a conventional MILP model, provided certain technical issues are handled correctly. Global constraints can be given linear relaxations that are not based on an MILP model and therefore cannot be used in MILP solvers. Beyond this, the potential of integrated problem solving can be tapped, because CP filtering and MILP relaxations are mutually reinforcing.

The key advantages of CP-based modeling are the power of the modeling language, the relative conciseness and naturalness of its formulations, and their ability to reveal problem structure to the solver. These advantages are clearly retained, and again enhanced, because the lexicon of global constraints is increased to encompass structural ideas from MILP.

The key advantages of MILP modeling—aside from its ability to harness MILP relaxation technology, which is retained—are its reliance on a small set of primitives (linear inequalities) and the relative independence of model and solution method. Integrated modeling obviously sacrifices the first advantage, because it relies on a sizable collection of global constraints. Yet it must be asked whether this is actually a sacrifice.

The difficulty with using global constraints is presumably that one must be familiar with a large number of them to be able to write a model. Yet one frequently writes an MILP model, or at least important parts of it, by identifying such patterns as flow balances, fixed charges, packing or covering constraints, and so forth, and then reducing them to inequality form. One must therefore be familiar with a number of modeling ideas in any case. Integrated modeling only spares one the labor (and errors) of writing micro-constraints that can be generated automatically. Moreover, a well-organized list of constraints can alert the modeler to patterns that might otherwise have been overlooked in the problem. It can provide a vocabulary that helps one to learn and distinguish modeling ideas, much as a technical vocabulary assists learning in any field.

Global constraints seem to be proliferating by the day, but new constraints tend to be variations on old ones. A well-known global constraints catalog [9] lists 313

constraints, but on close examination one can identify about thirty basic modeling ideas among these, of which the other constraints are variations and extensions. A good modeling system can organize constraints along various dimensions, so that one can generally find what one needs, much as one finds the relevant function in a spreadsheet.

Independence of model and method presents a more serious challenge to integrated modeling, and the matter deserves careful examination. It should be acknowledged at the outset, however, that the issue is not independence of model and *method*, because no modeling language achieves it, but independence of model and *solver*. Good MILP modelers know that one must think about the solution method when writing a model. The constraints must be chosen to result in a tight relaxation, the variables chosen to allow effective branching, redundant constraints added, symmetry considered, specially ordered sets used when applicable, and so forth. Nonetheless, MILP does achieve independence of model and solver (with the possible exception of specially ordered sets), because a given MILP model will run on almost any solver.

Reliance on global constraints undeniably links the model with the solver, because different solvers offer different libraries of constraints. The library of available constraints grows as solvers advance, and the best way to write a model evolves accordingly. Static collections of models used for benchmarking software, such as MIPLIB, are an impossibility, because the formulations must change with the solution technology to take full advantage of the software.

Independence of model and solver is frequently discussed as though it were an unmitigated advantage, when in fact it has both positive and negative aspects. A fixed modeling language provides the convenience of being able to run a model on any solver, but that very characteristic blocks progress in solution technology. Integrated methods, for example, can sometimes yield orders of magnitude in computational speedup, but only if one is willing to move beyond traditional MILP modeling.

As for benchmarking, a standard set of MILP models allows one to compare a wide variety of solvers, but at the cost of restricting one's attention to certain kinds of solvers. Benchmarking sets can equally well consist of problem *statements* (as opposed to models), so that one can reformulate the problems as necessary for new solvers. This allows one to monitor progress in modeling practices as well as algorithms. Some popular benchmarking libraries, such as MIPLIB, contain models for which the underlying problems are actually unknown, which means that they cannot be reformulated. This practice does not seem optimal for progress in either modeling or solution technology.

In summary, integrated modeling forgoes the convenience of solver independence, but it compensates with more convenient modeling and a wider repertory of solution methods. Even the inconvenience of incompatible solvers may fade over time, because software vendors will have an incentive to converge toward a universal set of global constraints. They may want to satisfy as many customers as possible by implementing all the global constraints they prefer to use.

The discussion here has focused on CP and MILP, but integrated modeling can in principle be broadened to encompass nonlinear constraints, local search heuristics and other AI-based search procedures, stochastic models and methods, and even simulation. Ideally, a single modeling system would allow one to write problem formulations to which the solver can apply any combination of methods that might be effective.

References

1. Achterberg, T., Berthold, T., Koch, T., Wolter, K.: A new approach to integrate CP and MIP. In: L. Perron, M.A. Trick (eds.) Proceedings of the International Workshop on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2008), *Lecture Notes in Computer Science*, vol. 5015, pp. 6–20. Springer (2008)
2. Ajili, F., Wallace, M.: Hybrid problem solving in ECLiPSe. In: M. Milano (ed.) Constraint and Integer Programming: Toward a Unified Methodology, pp. 169–206. Kluwer, Dordrecht (2004)
3. Althaus, E., Bockmayr, A., Elf, M., Kasper, T., Jünger, M., Mehlhorn, K.: SCIL—Symbolic constraints in integer linear programming. In: 10th European symposium on Algorithms (ESA 2002), *Lecture Notes in Computer Science*, vol. 2461, pp. 75–87. Springer (2002)
4. Apt, K., Wallace, M.: Constraint Logic Programming Using ECLiPSe. Cambridge University Press (2006)
5. Aron, I., Hooker, J.N., Yunes, T.H.: SIMPL: A system for integrating optimization techniques. In: J.C. Régin, M. Rueher (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2004), *Lecture Notes in Computer Science*, vol. 3011, pp. 21–36. Springer (2004)
6. Balas, E., Fischetti, M.: Polyhedral theory for the asymmetric traveling salesman problem. In: G. Gutin, A.P. Punnen (eds.) The Traveling Salesman Problem and its Variations, pp. 117–168. Kluwer, Dordrecht (2002)
7. Baptiste, P., Pape, C.L., Nuijten, W.: Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems. Kluwer, Dordrecht (2001)
8. Beldiceanu, N.: Pruning for the *minimum* constraint family and for the *number of distinct values* constraint family. In: T. Walsh (ed.) Principles and Practice of Constraint Programming (CP 2001), *Lecture Notes in Computer Science*, vol. 2239, pp. 211–224. Springer (2001)
9. Beldiceanu, N., Carlsson, M., Rampon, J.X.: Global constraint catalog. <http://www.emn.fr/x-info/sdemasse/gccat/> (updated 2009)
10. Beldiceanu, N., Contejean, E.: Introducing global constraints in CHIP. *Mathematical and Computer Modelling* **12**, 97–123 (1994)
11. Bessiere, C., Hebrard, E., Hnich, B., Kiziltan, Z., Walsh, T.: Filtering algorithms for the *nvalue* constraint. In: R. Barták, M. Milano (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2005), *Lecture Notes in Computer Science*, vol. 3524, pp. 79–93. Springer (2005)
12. Bourdais, S., Galinier, P., Pesant, G.: Hibiscus: A constraint programming application to staff scheduling in health care. In: F. Rossi (ed.) Principles and Practice of Constraint Programming (CP 2003), *Lecture Notes in Computer Science*, vol. 2833, pp. 153–167. Springer (2003)
13. Caseau, Y., Laburthe, F.: Solving small TSPs with constraints. In: L. Naish (ed.) Proceedings, Fourteenth International Conference on Logic Programming (ICLP 1997), vol. 2833, pp. 316–330. The MIT Press (1997)
14. Cheadle, A.M., Harvey, W., Sadler, A.J., Schimpf, J., Shen, K., Wallace, M.G.: ECLiPSe: A tutorial introduction. Technical report IC-Parc-03-1, IC-Park, Imperial College London (2003)

15. Colombani, Y., Heipcke, S.: Mosel: An extensible environment for modeling and programming solutions. In: N. Jussien, F. Laburthe (eds.) Proceedings of the International Workshop on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2002), pp. 277–290. Le Croisic, France (2002)
16. Colombani, Y., Heipcke, S.: Mosel: An overview. white paper, DASH Optimization (2004)
17. Hellsten, L., Pesant, G., van Beek, P.: A domain consistency algorithm for the stretch constraint. In: M. Wallace (ed.) Principles and Practice of Constraint Programming (CP 2004), *Lecture Notes in Computer Science*, vol. 3258, pp. 290–304. Springer (2004)
18. Hentenryck, P.V., Michel, L.: Constraint Based Local Search. MIT Press, Cambridge, MA (2005)
19. Hentenryck, P.V., Michel, L., Perron, L., Régin, J.C.: Constraint programming in OPL. In: International Conference on Principles and Practice of Declarative Programming (PPDP 1999), Paris (1999)
20. van Hoeve, W.J., Pesant, G., Rousseau, L.M., Sabharwal, A.: Revisiting the sequence constraint. In: F. Benhamou (ed.) Principles and Practice of Constraint Programming (CP 2006), *Lecture Notes in Computer Science*, vol. 4204, pp. 620–634. Springer (2006)
21. Hooker, J.N.: Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction. Wiley, New York (2000)
22. Hooker, J.N.: A hybrid method for planning and scheduling. In: M. Wallace (ed.) Principles and Practice of Constraint Programming (CP 2004), *Lecture Notes in Computer Science*, vol. 3258, pp. 305–316. Springer (2004)
23. Hooker, J.N.: Integrated Methods for Optimization. Springer (2007)
24. Hooker, J.N.: Planning and scheduling by logic-based Benders decomposition. *Operations Research* **55**, 588–602 (2007)
25. Hooker, J.N.: A principled approach to mixed integer/linear problem formulation. In: J.W. Chinneck, B. Kristjansson, M. Saltzman (eds.) Operations Research and Cyber-Infrastructure (ICS 2009 Proceedings), pp. 79–100. Springer (2009)
26. Hooker, J.N., Kim, H.J., Ottosson, G.: A declarative modeling framework that integrates solution methods. *Annals of Operations Research* **104**, 141–161 (2001)
27. Jeroslow, R.G.: Representability in mixed integer programming, I: Characterization results. *Discrete Applied Mathematics* **17**, 223–243 (1987)
28. Kaya, L.G., Hooker, J.N.: A filter for the circuit constraint. In: F. Benhamou (ed.) Principles and Practice of Constraint Programming (CP 2006), *Lecture Notes in Computer Science*, vol. 4204, pp. 706–710. Springer (2006)
29. Kaya, L.G., Hooker, J.N.: The circuit polytope. Manuscript, Carnegie Mellon University (2008)
30. Kruk, S.: Some facets of multiple alldifferent predicate. In: P. Belotti (ed.) Workshop on Bound Reduction Techniques for Constraint Programming and Mixed-Integer Nonlinear Programming, at CPAIOR (2009)
31. Magos, D., Mourtos, I., Appa, G.: A polyhedral approach to the *alldifferent* system. Technical report, Technological Educational Institute of Athens (2008)
32. Maher, M.J., Narodytska, N., Quimper, C.G., Walsh, T.: Flow-based propagators for the SEQUENCE and related global constraints. In: P.J. Stuckey (ed.) Principles and Practice of Constraint Programming (CP 2008), *Lecture Notes in Computer Science*, vol. 5202, pp. 159–174. Springer (2008)
33. Marchand, H., Martin, A., Weismantel, R., Wolsey, L.: Cutting planes in integer and mixed integer programming. *Discrete Applied Mathematics* **123**, 397–446 (2002)
34. Michel, L., Hentenryck, P.V.: Localizer: A modeling language for local search. *INFORMS Journal on Computing* **11**, 1–14 (1999)
35. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: Minizinc: Towards a standard CP modelling language. In: C. Bessiere (ed.) Principles and Practice of Constraint Programming (CP 2007), *Lecture Notes in Computer Science*, vol. 4741, pp. 529–543. Springer (2007)

36. Orman, A.J., Williams, H.P.: A survey of different integer programming formulations of the travelling salesman problem. In: E.J.K. and C. Gatu (ed.) *Optimisation, Economics and Financial Analysis, Advances in Computational Management Science*, pp. 933–106. Springer, Berlin (2006)
37. Pesant, G.: A filtering algorithm for the stretch constraint. In: T. Walsh (ed.) *Principles and Practice of Constraint Programming (CP 2001)*, *Lecture Notes in Computer Science*, vol. 2239, pp. 183–195. Springer (2001)
38. Pesant, G.: A regular language membership constraint for finite sequences of variables. In: M. Wallace (ed.) *Principles and Practice of Constraint Programming (CP 2004)*, *Lecture Notes in Computer Science*, vol. 3258, pp. 482–495. Springer (2004)
39. Quimper, C.G., López-Ortiz, A., van Beek, P., Golynski, A.: Improved algorithms for the global cardinality constraint. In: M. Wallace (ed.) *Principles and Practice of Constraint Programming (CP 2004)*, *Lecture Notes in Computer Science*, vol. 3258, pp. 542–556. Springer (2004)
40. Régin, J.C.: Generalized arc consistency for *global cardinality* constraint. In: *National Conference on Artificial Intelligence (AAAI 1996)*, pp. 209–215. AAAI Press (1996)
41. Régin, J.C.: Modeling problems in constraint programming. In: Tutorial presented at conference on *Principles and Practice of Constraint Programming (CP 2004)*. Toronto (2004)
42. Rodošek, R., Wallace, M., Hajian, M.: A new approach to integrating mixed integer programming and constraint logic programming. *Annals of Operations Research* **86**, 63–87 (1999)
43. Simonis, H.: Modelling in CP: Tutorial presented at CPAIOR 2009. <http://4c.ucc.ie/hsimonis/slidescpaior.pdf> (2009)
44. Stuckey, P.J., de la Banda, M.G., Maher, M., Marriott, K., Slaney, J., Somogyi, Z., Wallace, M., Walsh, T.: The G12 project: Mapping solver independent models to efficient solutions. In: P. van Beek (ed.) *Principles and Practice of Constraint Programming (CP 2005)*, *Lecture Notes in Computer Science*, vol. 3668, pp. 314–327. Springer (2005)
45. Tawarmalani, M., Sahinidis, N.V.: *Convexification and Global Optimization in Continuous and Mixed-integer Nonlinear Programming: Theory, Algorithms, Software, and Applications*. Springer (2002)
46. Tawarmalani, M., Sahinidis, N.V.: Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. *Mathematical Programming* **99**, 563–591 (2004)
47. Williams, H.P.: *Model Building in Mathematical Programming*, 4th Ed. Wiley, New York (1999)
48. Williams, H.P.: The formulation and solution of discrete optimization models. In: G. Appa, L. Pitsoulis, H.P. Williams (eds.) *Handbook on Modelling for Discrete Optimization*, pp. 3–38. Springer (2006)
49. Williams, H.P.: *Logic and Integer Programming*. Springer, Berlin (2009)
50. Williams, H.P., Brailsford, S.C.: The splitting of variables and constraints in the formulation of integer programming models. *European Journal of Operational Research* **100**, 623–628 (1997)
51. Williams, H.P., Yan, H.: Representations of the all_different predicate of constraint satisfaction in integer programming. *INFORMS Journal on Computing* **13**, 96–103 (2001)
52. Yunes, T.H., Aron, I., Hooker, J.N.: An integrated solver for optimization problems. *Operations Research* (to appear)