Chapter 15

Operations Research Methods in Constraint Programming

J. N. Hooker

A number of operations research (OR) methods have found their way into constraint programming (CP). This development is entirely natural, since OR and CP have similar goals.

OR is essentially a variation on the scientific practice of mathematical modeling. It describes phenomena in a formal language that allows one to deduce consequences in a rigorous way. Unlike a typical scientific model, however, an OR model has a prescriptive as well as a descriptive purpose. It represents a human activity with some freedom of choice, rather than a natural process. The laws of nature become constraints that the activity must observe, and the goal is to maximize some objective subject to the constraints.

CP's constraint-oriented approach to problem solving poses a prescriptive modeling task very similar to that of OR. CP historically has been less concerned with finding optimal than feasible solutions, but this is a superficial difference. It is to be expected, therefore, that OR methods would find application in solving CP models.

There remains a fundamental difference, however, in the way that CP and OR understand constraints. CP typically sees a constraint as a procedure, or at least as invoking a procedure, that operates on the solution space, normally by reducing variable domains. OR sees a constraint set as a whole cloth; the solution algorithm operates on the entire problem rather than the constraints in it. Both approaches have their advantages. CP can design specialized algorithms for individual constraints or subsets of constraints, thereby exploiting substructure in the problem that OR methods are likely to miss. OR algorithms, on the other hand, can exploit global properties of the problem that CP can only partially capture by propagation through variable domains.

15.1 Schemes for Incorporating OR into CP

CP's unique concept of a constraint governs how OR methods may be imported into CP. The most obvious role for an OR method is to apply it to a constraint or subset of constraints in order to reduce variable domains. Thus if the constraints include some linear

inequalities, one can minimize or maximize a variable subject to those inequalities, thereby possibly reducing the variable's domain. The minimization or maximization problem is a linear programming (LP) problem, which is an OR staple.

This is an instance of the most prevalent scheme for bringing OR into CP: create a *relaxation* of the CP problem in the form of an OR model, such as an LP model. Solution of the relaxation then contributes to domain reduction or helps guide the search. Other OR models that can play this role include mixed integer linear programming (MILP) models (which can themselves be relaxed), Lagrangean relaxations, and dynamic programming models. OR has also formulated specialized relaxations for a wide variety of common situations and provides tools for relaxing global constraints.

A relaxation provides several benefits to a CP solver. (a) It can tighten bounds on a variable. (b) Its solution may happen to be feasible in the original problem. (c) If not, the solution can guide the search in a promising direction. (d) The solution may allow one to filter domains in other ways, for instance by using reduced costs or Lagrange multipliers, or by examining the state space in dynamic programming. (e) In optimization problems, the solution can provide a bound on the optimal value that can be used to prune the search tree. (f) More generally, by pooling relaxations of several constraints in a single OR-based relaxation, one can exploit global properties of the problem that are only partially captured by constraint propagation.

Other hybridization schemes decompose the problem so that CP and OR can attack the parts of the problem to which they are best suited. To date, the schemes receiving the most attention have been branch-and-price algorithms and generalizations of Benders decomposition. CP-based branch and price typically uses CP for "column generation"; that is, to identify variables that should be added dynamically to improve the solution during a branching search. Benders decomposition often uses CP for "row generation"; that is, to generate constraints (nogoods) that direct the main search procedure.

OR/CP combinations of all three types can bring substantial computational benefits. Table 15.1 lists a sampling of some of the more impressive results. These represent only a small fraction, however, of hybrid applications; over 70 are cited in this chapter.

Even this collection omits entire areas of OR/CP cooperation. One is the use of concepts from operations research to design filters for certain global constraints, such as the application of matching and network flow theory to all-different, cardinality, and related constraints, and particularly to "soft" versions of these constraints. These ideas are covered in Chapter 7 and are therefore not discussed here. Two additional areas are heuristic methods and stochastic programming, both of which have a long history in OR. These are discussed in Chapters 8 and 21, respectively.

15.2 Plan of the Chapter

This chapter surveys the three hybridization schemes mentioned above: relaxation, branch-and-price methods, and Benders decomposition.

Sections 15.3–15.9 are devoted to relaxation, and of these the first four deal primarily with linear relaxations. Section 15.3 summarizes the elementary theory of linear programming (LP), which is used repeatedly in the chapter, and the role of LP in domain filtering. Section 15.4 briefly describes the formulation of MILP models. These are useful primarily because one can find LP relaxations for a wide variety of constraints by creating a MILP

Table 15.1: Sampling of computational results for methods that combine CP and OR.

Problem	Contribution to CP	Speedup
CP plus relaxations similar to those used in MILP		
Lesson timetabling [51]	Reduced-cost variable fixing using an assignment problem relaxation.	2 to 50 times faster than CP.
Minimizing piecewise linear costs [105]	Convex hull relaxation of piecewise linear function	2 to 200 times faster than MILP. Solved two instances that MILP could not solve.
Boat party & flow shop scheduling [77]	Convex hull relaxation of disjunctions, covering inequalities	Solved 10-boat instance in 5 min that MILP could not solve in 12 hours. Solved flow shop instances 3 to 4 times faster.
Product configuration [121]	Convex hull relaxation of element constraints, reduced cost variable fixing.	30 to 40 times faster than MILP (which was faster than CP).
Automatic digital recording [113]	Lagrangean relaxation	1 to 10 times faster than MILP (which was faster than CP).
Stable set problems [66]	Semi-definite programming relaxation.	Significantly better suboptimal solutions than CP in fraction of the time.
Structural design [23]	Linear quasi-relaxation of nonlinear model with discrete variables.	Up to 600 times faster than MILP. Solved 2 problems in < 6 min that MILP could not solve in 20 hours.
Scheduling with earliness and tardiness costs [14]	LP relaxation.	Solved 67 of 90 instances, while CP solved only 12.
CP-based branch and price		
Traveling tournament scheduling [44]	Branch-and-price framework.	First to solve 8-team instance.
Urban transit crew management [133]	Branch-and-price framework.	Solved problems with 210 trips, while traditional branch and price could accommodate only 120 trips.
Benders-based integration of CP and MILP		
Min-cost multiple machine scheduling [81]	MILP master problem, CP feasibility subproblem	20 to 1000 times faster than CP, MILP.
Min-cost multiple machine scheduling [120]	Updating of single MILP master (branch and check)	Additional factor of 10 over [81]
Polypropylene batch scheduling [122]	MILP master problem, CP feasibility subproblem.	Solved previously insoluble problem in 10 min.
Call center scheduling [16]	CP master, LP subproblem.	Solved twice as many instances as traditional Benders.
Min cost and min makespan planning & cumulative sched. [71]	MILP master problem, CP optimization subproblem	100 to 1000 times faster than CP, MILP. Solved significantly larger instances.
Min no. late jobs and min tardiness planning & cumulative sched. [72]	MILP master problem, CP . optimization subproblem with LP relaxation	Min late jobs 100-1000 times faster than MILP, CP; min tardiness significantly faster, better solutions when suboptimal.

model for them and dropping the integrality restrictions on the variables. Section 15.5 is a brief introduction to cutting planes, which can strengthen LP relaxations. Section 15.6 describes linear relaxations for some popular global constraints, while Section 15.7 provides continuous relaxations for piecewise linear constraints and disjunctions of nonlinear systems. Sections 15.8 and 15.9 deal with Lagrangean relaxation and dynamic programming, which can also provide useful relaxations.

Sections 15.10 and 15.11 are devoted to the remaining hybridization schemes discussed here, branch-and-price methods and Benders decomposition. The final section briefly explores the possibility of full CP/OR integration.

15.3 Linear Programming

Linear programming (LP) has a number of advantages that make it the most popular OR model discussed here. Although limited to linear inequalities (or equations) with continuous variables, it is remarkably versatile for representing real-world situations. It is even more versatile as a relaxation. It has an elegant duality theory that lends itself to sensitivity analysis and domain filtering. Finally, the LP problem is extremely well solved. It is rare for a practical LP instance, however large, to present any difficulty for a state-of-the-art solver.

LP relaxation provides all of the benefits of relaxation that were mentioned earlier. In particular, a solution that is infeasible in the original problem can guide the search by suggesting how to branch. If a variable x_j is required to be integral in the original problem, then an nonintegral value $\bar{x}_j$ in the solution of the LP relaxation suggests branching by requiring $x_j \leq \lfloor \bar{x}_j \rfloor$ in one branch and $x_j \geq \lceil \bar{x}_j \rceil$ in the other. Rounding of LP solutions, a technique widely used in approximation algorithms, can also be used a guide to backtracking [58].

Semidefinite programming [4, 129] generalizes LP and has been used in a CP context as a relaxation for the stable set problem [66]. It can also serve as a basis for approximation algorithms [57].

15.3.1 Optimal Basic Solutions

Without loss of generality an LP problem can be written

$$\begin{array}{l} \min\ cx \\ Ax \geq b,\ x \geq 0,\ x \in \Re^n \end{array} \tag{15.1}$$

where A is an $m \times n$ matrix. This can be read, "minimize cx subject to the constraints $Ax \geq b,\ x \geq 0$." In OR terminology, any $x \in \Re^n$ is a *solution* of (15.1), and any $x \geq 0$ for which $Ax \geq b$ is a *feasible* solution. The problem is *infeasible* if there is no feasible solution. It is *unbounded* if (15.1) is feasible but has no optimal solution.

The feasible set of (15.1) is a polyhedron, and the vertices of the polyhedron correspond to *basic* feasible solutions. Since the objective function cx is linear, it is intuitively clear that some vertex is optimal unless the problem is unbounded. It is useful to develop this idea algebraically.

The LP problem is first rewritten in equality form

$$\begin{aligned} &\min\ cx \\ &Ax = b,\ x \geq 0,\ x \in \Re^n \end{aligned} \tag{15.2}$$

An inequality constraint $ax \geq a_0$ can always be converted to an equality constraint by introducing a surplus variable $s_0 \geq 0$ and writing $ax - s_0 = a_0$.

Assume for the moment that (15.2) is feasible. Suppose further that $m \leq n$ and A has rank m. If A is partitioned as $[B\ N]$, where B is any set of m independent columns, then (15.2) can be written

$$\begin{aligned} &\min\ c_B x_B + c_N x_N \\ &Bx_B + Nx_N = b, \quad x_B, x_B \geq 0 \end{aligned} \tag{15.3}$$

The variables x_B that correspond to the columns of B are designated *basic* variables because B is a basis for $\Re^m$. One can solve the equality constraints for x_B in terms of the nonbasic variables x_N:

$$x_B = B^{-1}b - B^{-1}Nx_N \tag{15.4}$$

Thus any feasible solution of (15.4) has the form $(x_B, x_N) = (B^{-1}b - B^{-1}Nx_N, x_N)$ for some $x_N \geq 0$. Setting $x_N = 0$ yields a basic solution $(B^{-1}b, 0)$, which corresponds to a vertex of the feasible polyhedron if $B^{-1}b \geq 0$.

Substituting (15.4) into the objective function of (15.3) allows cost to be expressed as a function of the nonbasic variables x_N:

$$c_B B^{-1}b + (c_N - c_B B^{-1}N)x_N$$

Thus $c_B B^{-1}b$ is the cost of the basic solution $(B^{-1}b, 0)$. The row vector $r = c_N - c_B B^{-1}N$ contains the *reduced costs* associated with the nonbasic variables x_N. Since every feasible solution of (15.3) can be obtained by setting x_N to some nonnegative value, the cost can be smaller than $c_B B^{-1}b$ only if at least one reduced cost is negative. So the basic solution $(B^{-1}b, 0)$ is optimal if $r \geq 0$.

15.3.2 Simplex Method

Given a basic feasible solution $(B^{-1}b, 0)$, the *simplex method* can find a basic optimal solution of (15.3) or show that (15.3) is unbounded. If $r \geq 0$, the solution $(B^{-1}b, 0)$ is already optimal. Otherwise increase any nonbasic variable x_j with negative reduced cost r_j. If the column of $B^{-1}N$ in (15.4) that corresponds to x_j is nonnegative, then x_j can increase indefinitely without driving any component of x_B negative, which means (15.3) is unbounded. Otherwise increase x_j until some basic variable x_i hits zero. This creates a new basic solution. The column of B corresponding to x_i is moved out of B and the column of N corresponding to x_j is moved in. B^{-1} is quickly recalculated and the process repeated.

The procedure terminates with an optimal or unbounded solution if one takes care not to cycle through solutions in which one or more basic variables vanish (*degeneracy*). A starting basic feasible solution can be obtained by solving a "Phase I" problem in which the objective is to minimize the sum of constraint violations. The starting basic variables

in the Phase I problem are temporary slack or surplus variables added to represent the constraint violations that result when the other variables are set to zero.

More than half a century after its invention by George Dantzig, the simplex method is still the most widely used method in state-of-the-art solvers. *Interior point* methods are competitive for large problems and are also available in commercial solvers.

15.3.3 Duality and Sensitivity Analysis

The *dual* of a linear programming problem (15.1) is

$$\begin{array}{l} \max \lambda b \\ \lambda A \leq c, \;\; \lambda \geq 0, \;\; \lambda \in \Re^m \end{array} \tag{15.5}$$

The dual can be understood as seeking the tightest lower bound v on the objective function cx that can be inferred from the constraints $Ax \geq b,\ x \geq 0$. One consequence of the Farkas Lemma, a classical result of mathematical programming, is that $cx \geq v$ can be inferred from a feasible system $Ax \geq b, x \geq 0$ if and only if some nonnegative linear combination $\lambda Ax \geq \lambda b$ of $Ax \geq b$ dominates $cx \geq v$. Since $\lambda Ax \geq \lambda b$ dominates $cx \geq v$ when $\lambda A \leq c$ and $\lambda b \geq v$, (15.5) is simply the problem of finding the tightest lower bound v. The dual (15.5) and the *primal* problem (15.1) therefore have the same optimal value if both are feasible and unbounded (*strong duality*).

The dual provides sensitivity analysis, which in turn leads to domain filtering. Note first that the value λb of any dual feasible solution provides a lower bound on the value cx of any primal feasible solution (*weak duality*). This is because $\lambda b \leq \lambda Ax \leq cx$, where the first inequality is due to $Ax \leq b$ and $\lambda \geq 0$, and the second inequality to $\lambda A \geq c$ and $x \geq 0$. Now suppose x^* is optimal in the primal and λ^* is optimal in the dual. If the right-hand side b of the primal constraints is perturbed to obtain a new problem with constraints $Ax \geq b + \Delta b$, only the objective function of the dual changes, specifically to $\lambda(b + \Delta b)$. Thus λ^* is still dual feasible and provides a lower bound $\lambda^*(b + \Delta b)$ on the optimal value of the perturbed primal problem. In other words, the perturbation increases the optimal value $\lambda^* b$ of the original problem by at least $\lambda^* \Delta b$.

The dual multipliers in λ^* are readily available when the primal is solved. In fact, $\lambda^* = c_B B^{-1}$, as can be verified by writing the dual of (15.3). These multipliers indicate the sensitivity of the optimal cost to small changes in b. In addition the reduced costs are closely related to the dual multipliers, since $r = c_N - c_B B^{-1} N = c_N - \lambda^* N$. The reduced cost of a single nonbasic variable x_j is $r_j = c_j - \lambda^* A_j$, where A_j is the column of N (and of A) corresponding to x_j.

A related property of the dual solution is *complementary slackness*, which means that a dual variable can be positive only if the corresponding primal constraint is tight in an optimal solution. Thus if x^* and λ^* are optimal in the primal and dual, respectively, then $\lambda^*(Ax^* - b) = 0$. This is because $\lambda^* b = cx^*$, by strong duality, which together with $\lambda^* b \leq \lambda^* Ax^* \leq cx^*$ implies $\lambda^* b = \lambda^* Ax^*$ or $\lambda^*(Ax^* - b) = 0$.

15.3.4 Domain Filtering

The dual solution can help filter variable domains. Suppose that the LP problem (15.1) is a relaxation of a problem that is being solved by CP. There is an upper bound U on the cost cx. For instance, U might be the cost of the best feasible solution found so far in a

cost minimization problem, and there is no need to consider solutions with cost greater than U. Suppose (15.1) has optimal value v^*, and the optimal dual solution is λ^*. Suppose further that $\lambda_i^* > 0$, which means the constraint $A^i x \geq b_i$ is tight (i.e., $A^i x^* = b_i$), due to complementary slackness. If the solution of (15.1) were to change, the left-hand side of the ith constraint $A^i x \geq b_i$ of (15.1) could change, say by an amount Δb_i. This would affect the optimal value of (15.1) as much as changing the constraint $A^i x \geq b_i$ to $A^i x \geq b_i + \Delta b_i$, which is to say it would increase the optimal value by at least $\lambda_i^* \Delta b_i$. Since the optimal value cannot rise to a value greater than U, one must have that $\lambda_i^* \Delta b_i \leq U - v^*$, or $\Delta b_i \leq (U - v^*)/\lambda_i$. Since $\Delta b_i = A^i x - A^i x^* = A^i x - b_i$, this yields the valid inequality

$$A^i x \leq b_i + \frac{U - v^*}{\lambda_i} \tag{15.6}$$

for each constraint i of (15.1) with $\lambda_i^* > 0$. The inequality (15.6) can now be propagated, which is particularly useful if some of the variables x_j have integer domains in the original problem.

One can reduce the domain of a particular nonbasic variable x_j by considering the nonnegativity constraint $x_j \geq 0$. Since the reduced cost of x_j measures the effect on cost of increasing x_j, the dual multiplier associated with $x_j \geq 0$ is the reduced cost $r_j = c_j - \lambda^* A_j$. So (15.6) becomes $x_j \leq (U - v^*)/r_j$. If x_j has an integer domain in the original problem, one can say $x_j \leq \lfloor (U - v^*)/r_j \rfloor$.

CP applications of reduced-cost-based filtering include the traveling salesman problem with time windows [96], product configuration [121], fixed charge network flows [86], lesson timetabling [51], and the traveling salesman problem with time windows [51]. The additive bounding procedure [50], which uses reduced costs, has been applied to limited discrepancy search [91].

15.3.5 Example: Traveling Salesman with Time Windows

A traveling salesman problem with time windows provides an example of domain filtering [51]. Suppose a salesman (or delivery truck) must make several stops, perhaps subject to such additional constraints as time windows. The objective is to minimize the total travel time, which has upper bound U. The *assignment problem relaxation* of the constraint set is

$$\begin{aligned} &\min \sum_{ij} c_{ij} x_{ij} \\ &\sum_j x_{ij} = \sum_j x_{ji} = 1, \text{ all } i, \qquad x_{ij} \in \{0,1\}, \text{ all } i,j \end{aligned} \tag{15.7}$$

where c_{ij} is the travel time from stop i to stop j. Variable x_{ij} is 1 when the salesman visits stop j immediately after stop i and is zero otherwise. One can now solve an LP relaxation of (15.7) obtained by replacing $x_{ij} \in \{0,1\}$ with $0 \leq x_{ij} \leq 1$. If r_{ij} is the reduced cost associated with x_{ij} and v^* the optimal value of (15.7), then x_{ij} can be fixed to zero if $(U - v^*)/r_{ij} < 1$. This particular LP problem can be solved very rapidly, since there are specialized methods (e.g. the Hungarian algorithm) for assignment problems.

Solving the LP relaxation of an assignment problem (15.7) actually solves the problem itself, since every basic solution of (15.7) is integral. This is due to the total unimodularity of the matrix of constraint coefficients, which means that every square submatrix has a determinant of 1, -1, or 0.

15.4 Mixed Integer/Linear Modeling

A *mixed integer/linear programming* (MILP) problem is an LP problem with the additional restriction that certain variables must take integer values. It is a (pure) integer/linear programming (ILP) problem when all the variables are integer-valued, and a 0-1 linear programming problem when all the variables have domain $\{0, 1\}$.

MILP problems are solved by a *branch-and-bound* search mechanism. An LP relaxation of the problem is solved at each node of a search tree. If the optimal value of the relaxation is greater than or equal to the value of the best candidate solution found so far, the search backtracks. Otherwise, if all variables in the LP solution are integral, then it becomes a candidate solution. If one or more variables are nonintegral, the search branches on one of the nonintegral variables by splitting its domain. Cutting planes are commonly added at the root node and possibly at other nodes, resulting in a *branch-and-cut* method.

Although MILP problems are generally much harder to solve that LP problems, the solution technology has been the subject of intense development for at least three decades. Commercial solvers have achieved orders-of-magnitude speedups through the right combination of cutting planes, branching heuristics, and preprocessing.

The primary role of MILP in CP, however, is to provide an LP relaxation of a constraint or subset of constraints. One formulates an MILP model and drops the integrality condition. MILP is a highly versatile modeling language if one is sufficiently ingenious. Writing a model with a "good" LP relaxation, however, is often more an art than a science [124]. A good relaxation is generally viewed as one whose optimal value is close to that of the MILP problem.

15.4.1 MILP Representability

It is known precisely what sort of feasible set can be represented by an MILP model. A subset S of $\Re^n$ is MILP-representable if and only if S is the union of finitely many polyhedra, all of which have the same recession cone. The *recession cone* of a polyhedron P is the set of directions in which P is unbounded, or more precisely, the set of vectors $r \in \Re^n$ such that, for some $u \in P$, $u + \alpha r \in P$ for all $\alpha \geq 0$.

The intuition behind this fact can provide a tool for writing MILP models when S has a fairly simple structure. Since S is a union of polyhedra, it is described by a disjunction of linear systems:

$$\bigvee_{k \in K} A^k x \leq b^k \tag{15.8}$$

in which each system $A^k x \leq b^k$ represents a polyhedron. To obtain an MILP model of (15.8), one can introduce 0-1 variables y_k that take the value 1 when the kth disjunct holds:

$$\begin{aligned} & A^k x^k \leq b^k y_k, \ k \in K \\ & x = \sum_{k \in K} x^k, \ \ \sum_{k \in K} y_k = 1 \\ & x, x^k \in \Re^n, \ y_k \in \{0, 1\}, \ k \in K \end{aligned} \tag{15.9}$$

Note that the vector x of continuous variables is disaggregated into variables x^k. Thus when $y_\ell = 1$ and the other y_ks are zero, the constraints force $x = x^\ell$ and therefore require x to satisfy $A^\ell x \leq b^\ell$.

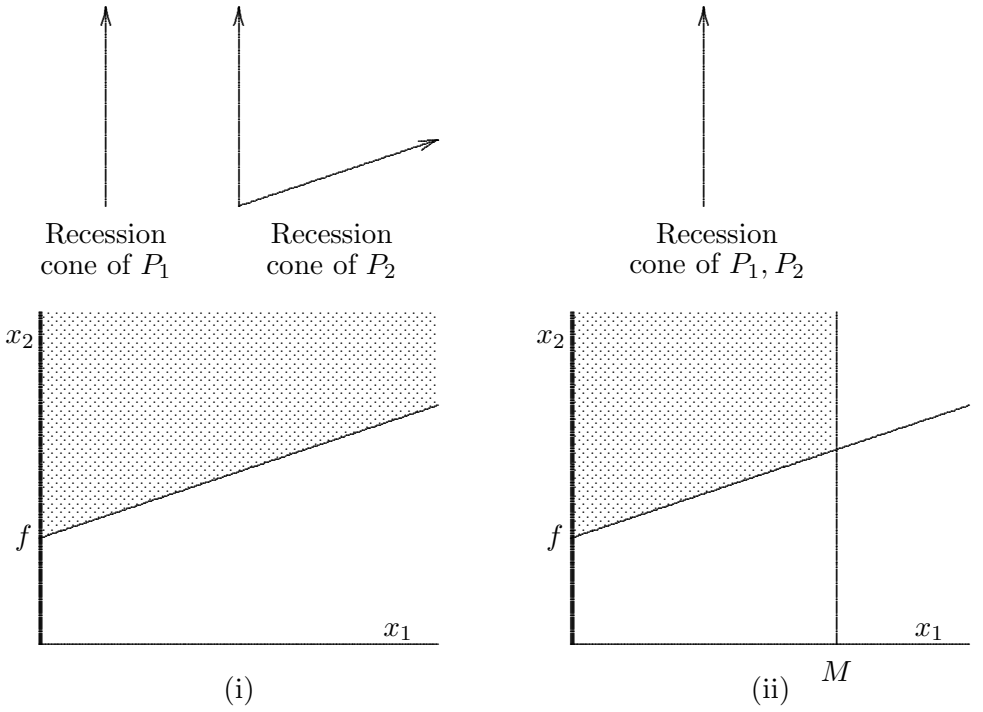

Figure 15.1: *(i) Feasible set of a fixed charge problem, consisting of the union of polyhedra P_1 (heavy line) and P_2 (shaded area). (ii) Feasible set of the same problem with the bound $x_1 \leq M$, where P_2' is the shaded area.*

There are a number of devices for writing MILP formulations when (15.10) does not yield a practical model. A comprehensive discussion of these may be found in [127].

15.4.2 Example: Fixed Charge Function

MILP representability is illustrated by the fixed charge function, which occurs frequently in modeling. Suppose the cost x_2 of producing quantity x_1 is bounded below by zero when the production quantity x_1 is zero, and $f + cx_1$ otherwise, where f is the fixed cost and c the unit variable cost. If S is the set of feasible points (x_1, x_2), then S is the union of two polyhedra P_1 and P_2 (Fig 15.1a). The recession cone of P_1 is P_1 itself, and the recession cone of P_2 is the set of all vectors (x_1, x_2) with $x_2 \geq cx_1 \geq 0$. Since these cones are not identical, S is not MILP-representable.

However, in practice one can put a sufficiently large upper bound M on x_1. Now the recession cone of each of the resulting polyhedra P_1, P_2' (Fig. 15.1b) is the same (namely, P_1), and the feasible set $S' = P_1 \cup P_2'$ is therefore MILP-representable. P_1 is the polyhedron described by $x_1 \leq 0,\ x_1, x_2 \geq 0$, and P_2' is described by $cx_1 - x_2 \leq -f,\ x_1 \leq$

$M,\ x_1 \geq 0$. So (15.9) becomes

$$\begin{array}{llll} x_1^1 \leq 0 & cx_1^2 - x_2^2 \leq -fy_2 & x_1 = x_1^1 + x_1^2 & y_1 + y_2 = 1 \\ x_1^1, x_2^2 \geq 0 & 0 \leq x_1 \leq My_2 & x_2 = x_2^1 + x_2^2 & y_1, y_2 \in \{0,1\} \end{array} \tag{15.10}$$

As often happens, (15.10) simplifies. Only one 0-1 variable appears, which can be renamed y. Since x_1^1 is forced to zero, $x_1 = x_1^2$, and the resulting model is $x_2 \geq fy + cx_1, x_1 \leq My$. Obviously y encodes whether the quantity produced is zero or positive, in the former case ($y = 0$) forcing $x_1 = 0$, and in the latter case incurring the fixed charge f. "Big M" constraints like $x_1 \leq My$ are common in MILP models.

15.4.3 Relaxing the Disjunctive Model

The disjunctive model (15.10) has the advantage that its continuous relaxation, obtained by replacing $y_i \in \{0,1\}$ with $0 \leq y_i \leq 1$, is a *convex hull relaxation* of (15.8)—the tightest possible linear relaxation. The convex hull of a set $S \in \Re^n$ is the intersection of all half planes that contain S. The LP relaxation of (15.10) is a convex hull relaxation in the sense that the projection of its feasible set onto the original variables x_1, x_2 is the convex hull of the feasible set of (15.8).

A model of (15.8) with fewer variables is

$$\begin{array}{l} A^k x \leq b^k + M^k(1 - y_k),\ k \in K \\ \displaystyle\sum_{k \in K} y_k = 1,\ \ x \in \Re^n,\ y_k \in \{0,1\},\ k \in K \end{array} \tag{15.11}$$

where each component of M^k is a valid upper bound on the corresponding inequality of $A^k \leq b^k$. The kth disjunct is enforced when $y_k = 1$. The LP relaxation of (15.11) is not in general a convex hull relaxation, but (15.11) is a correct model even if the polyhedra described by the systems $A^k x \geq b^k$ do not have the same recession cone. The LP relaxation of (15.10), incidentally, is a valid convex hull relaxation of (15.10) even when the polyhedra do not have the same recession cone.

The LP relaxation of (15.11) simplifies when each system $A^k \leq b^k$ is a single inequality $a^k x \leq a_k$ and $0 \leq x \leq m$ [13]. The variables y_k drop out and the relaxation becomes

$$\left(\sum_{k \in K} \frac{a^k}{M_k} \right) x \leq \sum_{k \in K} \frac{b_k}{M_k} + |K| - 1,\ \ 0 \leq x \leq m$$

where $M_k = b_k - \sum_j \min\{0, a_j^k\} m_j$.

15.5 Cutting Planes

Cutting planes, variously called *cuts* or *valid inequalities*, are linear inequalities that can be inferred from an integer or mixed integer constraint set. Cutting planes are added to the constraint set to "cut off" noninteger solutions of its LP relaxation, thus resulting in a tighter relaxation.

Cutting planes can also exclude redundant partial assignments (redundant compound labels), even if is not their intended purpose. In some cases they may achieve consistency

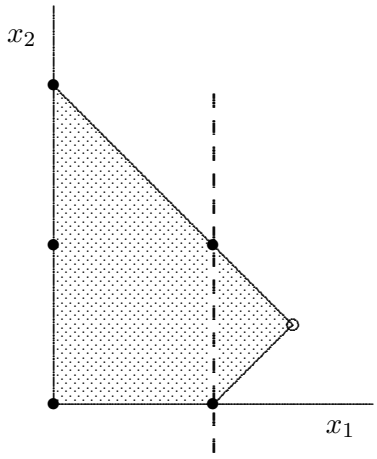

Figure 15.2: *Feasible set (shaded area) of the LP relaxation of a system* $x_1 + x_2 \leq 2$, $x_1 - x_2 \leq 0$ *with domains* $x_j \in \{0, 1, 2\}$, *and a cutting plane* $x_1 \leq 1$ *(dashed line).*

of one kind or another, even though the concept of consistency never developed in the OR community. It is therefore likely that cutting planes reduce backtracking in branch-and-bound algorithms by excluding redundant partial assignments, quite apart from their role in strengthening relaxations.

For example, $x_1 \leq 1$ is a cutting plane for the system $x_1 + x_2 \leq 2$, $x_1 - x_2 \leq 1$ in which each x_j has domain $\{0, 1, 2\}$. As Fig. 15.2 shows, $x_1 \leq 1$ is valid because it cuts off no feasible (0-1) points. Yet it cuts off part of the feasible set of the LP relaxation and therefore strengthens the LP relaxation, in fact resulting in a convex hull relaxation. Adding the cut also achieves arc consistency for the constraint set, since it reduces x_1's domain to $\{0, 1\}$.

A few basic types of cutting planes are surveyed here. General references on cutting planes include [93, 99, 130], and the role of cutting planes in CP is further discussed in [20, 22, 38, 52, 69, 77]. CP-based application of cutting planes include the orthogonal Latin squares problem [6], truss structure design [23], processing network design [60, 77], single-vehicle routing [110], resource-constrained scheduling [40], multiple machine scheduling [22], boat party scheduling [77], and the multidimensional knapsack problem [100]. Cutting planes for disjunctions of linear systems have been applied to factory retrofit planning, strip packing, and zero-wait job shop scheduling [111].

15.5.1 Chvátal-Gomory Cuts

One can always generate a cutting plane for an integer linear system $Ax \leq b$ by taking a nonnegative linear combination $uAx \leq ub$ of the inequalities in the system and rounding down all fractions that result. This yields the cut $\lfloor uA \rfloor x \leq \lfloor ub \rfloor$, where $u \geq 0$. For example, one can obtain the cut $x_1 \leq 0$ from $x_1 + x_2 \leq 1$ and $x_1 - x_2 \leq 0$ (Fig. 15.2) by giving each a multiplier of $\frac{1}{2}$ in the linear combination.

After generating a cut of this kind, one can add it to $Ax \leq b$ and repeat the process. Any cut generated recursively in this fashion is a *Chvátal-Gomory cut*. A fundamental result of cutting plane theory is that every cut is a Chvátal-Gomory cut [31].

A subset of Chvátal-Gomory cuts are enough to achieve consistency. If two particular types of cuts, *resolvents* and *diagonal sums*, are recursively derived from 0-1 inequalities that are dominated by inequalities in $Ax \leq b$, then every inequality implied by $Ax \leq b$ (up to equivalence) is implied by one of the generated cuts; see [67, 70] for details. This fact leads to a logic-based method for 0-1 linear constraint solving [12]. The derivation of resolvents alone is enough to achieve strong n-consistency and therefore hyperarc consistency.

15.5.2 General Separating Cuts

Since it is impractical to generate all Chvátal-Gomory cuts, a more common approach is to identify one or more cuts that exclude or "cut off" a nonintegral solution of the current LP relaxation. These are known as *separating cuts* because they separate the nonintegral solution from the convex hull of the original problem's feasible set. For instance, if the LP relaxation of the example in Fig. 15.2 has the solution $(\frac{3}{2}, \frac{1}{2})$, then $x_1 \leq 1$ is a separating cut.

A general-purpose separating cut can be generated by solving the LP relaxation of an integer system in equality form (15.2) to obtain a basic solution $(\hat{b}, 0)$, where $\hat{b} = B^{-1}b$. If x_i is any basic variable with a nonintegral value in this solution, the valid inequality

$$x_i \leq \lfloor \hat{b}_i \rfloor - \lfloor \hat{N}_i \rfloor x_N \tag{15.12}$$

cuts off $(\hat{b}, 0)$. Here $\hat{N} = B^{-1}N$, and $\hat{N}_i$ refers to the ith row of $\hat{N}$. This cut is popularly known as a *Gomory cut*.

For many years Gomory cuts were believed to be ineffective, but now it is known that they can be very useful if sufficiently many are generated. Multiple cuts can be obtained by generating one for each nonintegral x_i, or perhaps by re-solving the LP with cuts added and generating further Gomory cuts.

Gomory cuts have an analog for MILP systems, known as *separating mixed integer rounding cuts* [94], that are equally important in practice. Let the MILP system in equality form be $A_1 y + A_2 x = b$ with $x, y \geq 0$ and y integer. Let B and N be basic and nonbasic columns in the LP relaxation as before, and suppose that a basic variable y_i is nonintegral. Define J to be the set of indices j for which y_j is basic and K to be the set of indices for which x_j is basic. Also let $J_1 = \{j \in J \mid \text{frac}(\hat{N}_{ij}) \geq \text{frac}(\hat{b}_i)\}$ and $J_2 = J \setminus J_1$, where $\text{frac}(\alpha) = \alpha - \lfloor \alpha \rfloor$ is the fractional part of α. Then the following is a mixed integer rounding cut that cuts off the nonintegral solution:

$$y_i \geq \lfloor \hat{b}_i \rfloor - \sum_{j \in J_1} \lceil \hat{N}_{ij} \rceil y_j - \sum_{j \in J_2} \left(\lfloor \hat{N}_{ik} \rfloor + \frac{\text{frac}(\hat{N}_{ij})}{\text{frac}(\hat{b}_i)} \right) y_j - \frac{1}{\text{frac}(\hat{b}_i)} \sum_{j \in K} \hat{N}^+_{ij} x_j$$

where $\hat{N}^+_{ij} = \max\{0, \hat{N}_{ij}\}$.

15.5.3 Knapsack Cuts

Knapsack cuts, also known as *lifted covering inequalities*, are defined for an integer linear inequality $ax \leq \alpha$ with $a \geq 0$. (One can always obtain $a \geq 0$ by replacing each x_j that

has a negative coefficient with $U_j - x_j$, where U_j is an upper bound on x_j.) Knapsack cuts do not affect consistency, since they are generated for one constraint at a time, but they tighten the LP relaxation and are often useful in practice.

Suppose first that $ax \leq \alpha$ is a 0-1 inequality. A *cover* for $ax \leq \alpha$ is an index set $J \in \{1,\dots,n\}$ for which $\sum_{j\in J} a_j > \alpha$. A cover is *minimal* if no proper subset it is a cover. If J is a cover, the *covering inequality* $\sum_{j\in J} x_j \leq |J| - 1$ is obviously valid for $ax \leq \alpha$. (Only minimal covers need be considered, since nonminimal covering inequalities are redundant of minimal ones.) For example, $J = \{1,2,3,4\}$ is a minimal cover for the inequality

$$5x_1 + 5x_2 + 5x_3 + 5x_4 + 8x_5 + 3x_6 \leq 17$$

and gives rise to the covering inequality $x_1 + x_2 + x_3 + x_4 \leq 3$.

A covering inequality can often be strengthened by adding variables to the left-hand side; that is, by *lifting* the inequality into a higher dimensional space. *Sequential lifting* is presented here, in which terms are added one at a time; there are also techniques for adding several terms simultaneously [93]. If $\sum_{j\in J} x_j \leq |J| - 1$ is a covering inequality, then $\pi_k x_k + \sum_{j\in J} x_j \leq |J| - 1$ is also valid for $ax \leq \alpha$, provided

$$\pi_k \leq |J| - 1 - \max\left\{ \sum_{j\in J} x_j \;\middle|\; \sum_{j\in J} a_j x_j \leq \alpha - a_k,\ x_j \in \{0,1\} \text{ for } j \in J \right\}$$

For example, $x_1+x_2+x_3+x_4 \leq 3$ can be lifted to $x_1+x_2+x_3+x_4+2x_5 \leq 3$. If it is lifted further by adding x_6, the inequality is unchanged since $\pi_6 = 0$. The order of lifting can affect the outcome; if x_6 is added before x_5, the resulting cut is $x_1+x_2+x_3+x_4+x_5+x_6 \leq 3$. Lifted inequalities are useful only when they can be generated quickly, as when the covering inequality has only a few variables, or the problem has special structure. The coefficients π_k can be computed sequentially by dynamic programming [93].

Covering inequalities can also be derived for an inequality $ax \geq \alpha$ with general integer variables. Here J is a cover if $\sum_{j\in J} a_j \overline{x}_j > \alpha$, where $\overline{x}_j$ is an upper bound on x_j. Any cover J yields the covering inequality $\sum_{j\in J} x_j \leq \alpha / \max_{j\in J}\{a_j\}$. Lifting is possible but more difficult than in the 0-1 case.

15.6 Relaxation of Global Constraints

Linear relaxations for a few common global constraints are presented here: element, all-different, circuit and cumulative. Some relaxations are continuous relaxations of MILP models, and others are derived directly without recourse to an MILP model. These and other relaxations are surveyed in [70, 106].

15.6.1 Element Constraint

An element constraint has the form

$$\text{element}(x, (t_1,\dots,t_m), v) \tag{15.13}$$

and requires that $v = t_x$. (15.13) implies the disjunction $\bigvee_{k\in D_x}(v = t_k)$, where D_x is the current domain of x. This disjunction can be given an MILP model (15.10), which

leads immediately to a convex hull relaxation of (15.13). If each t_k is a constant, then the relaxation is trivial: $\min_{k\in D_x}\{t_k\} \le v \le \max_{k\in D_x}\{t_k\}$. However, if each t_k is a variable with interval domain $[L_k, U_k]$, (15.10) yields a more interesting convex hull relaxation:

$$\begin{aligned}
&L_j y_k \le t_j^k \le U_j y_k,\ j,k \in D_x \\
&v = \sum_{k\in D_x} t_k^k,\quad \sum_{k\in D_x} y_k = 1 \\
&t_k = \sum_{j\in D_x} t_k^j,\ y_k \ge 0,\ k \in D_x
\end{aligned}$$

Since this relaxation contains a large number of variables t_i^k, one may wish to use the simpler relaxation

$$\begin{aligned}
&\sum_{k\in D_x} t_k - (|D_x|-1)U_{\max} \le v \le \sum_{k\in D_x} t_k - (|D_x|-1)L_{\min} \\
&L_k \le t_k \le U_k,\ k \in D_x
\end{aligned} \tag{15.14}$$

where $L_{\min} = \min_{k\in D_x}\{L_k\}$ and $U_{\max} = \max_{k\in D_x}\{U_k\}$. This is a convex hull relaxation when $L_k = L_{\min}$ and $U_k = U_{\max}$ for all k [70]. One can also use the LP relaxation of (15.11). Another relaxation is

$$\begin{aligned}
&\left(\sum_{k\in D_x} \frac{1}{U_{\max} - L_k}\right) v \le \sum_{k\in D_x} \frac{t_k}{U_{\max} - L_k} + |D_x| - 1 \\
&\left(\sum_{k\in D_x} \frac{1}{U_k - L_{\min}}\right) v \ge \sum_{k\in D_x} \frac{t_k}{U_k - L_{\min}} - |D_x| + 1 \\
&L_k \le t_k \le U_k,\ k \in D_x
\end{aligned}$$

This is in general not redundant of (15.14), unless of course (15.14) is a convex hull relaxation.

15.6.2 All-Different Constraint

The constraint

$$\text{all-different}(y_1, \ldots, y_n)) \tag{15.15}$$

requires that $y_1, \ldots, y_n$ be distinct. If the domain D_{y_i} of each y_i is a finite set of real numbers and $\bigcup_{i=1}^n D_{y_i} = \{a_1, \ldots, a_m\}$, (15.15) can be given the MILP formulation:

$$\begin{aligned}
&y_i = \sum_{j=1}^m a_j x_{ij},\quad \sum_{j=1}^m x_{ij} = 1,\quad i \in \{1, \ldots, n\} \\
&\sum_{i=1}^n x_{ij} \ge 1,\ j \in \{1, \ldots, m\} \\
&x_{ij} = 0, \text{ all } i,j \text{ with } j \notin D_{y_i}
\end{aligned} \tag{15.16}$$

where the binary variable $x_{ij} = 1$ when $y_i = j$. A continuous relaxation can be obtained by replacing $x_{ij} \in \{0, 1\}$ with $x_{ij} \geq 0$. This is a convex hull relaxation, in the sense that the projection of its feasible set onto the variables y_i is the convex hull of the feasible set of (15.15).

The MILP formulation (15.16) is convenient for some problems and not for others. Suppose for instance the problem is to find a minimum-cost assignment of jobs to workers, where c_{ij} is the cost of assigning job j to worker i. The constraint (15.15) requires that every worker get a different job, and the problem is to minimize $\sum_{i=1}^{n} c_{iy_i}$ subject to (15.15). MILP solves the problem by minimizing $\sum_{ij} c_{ij} x_{ij}$ subject to (15.16).

However, if one wishes to find a minimum-cost route in a traveling salesman problem, the objective function becomes nonlinear in the integer model. (15.15) is a correct model for this problem only if y_i is the ith stop visited, rather than the stop visited immediately after i as in (15.7). So if c_{ij} is the travel time from i to j as before, the problem is to minimize $\sum_{i=1}^{n} c_{y_i y_{i+1}}$ subject to (15.15), where y_{n+1} is identified with y_1. The integer model must minimize the nonlinear expression $\sum_{ijk} c_{jk} x_{ij} x_{i+1,k}$ subject to (15.16). However, there are linear 0-1 models for this problem, the most popular of which is presented in the next section.

If D_{y_i} is the same set of numbers $\{a_1, \ldots, a_m\}$ for each i, with $a_1 \leq \cdots \leq a_m$, a convex hull relaxation [70, 75, 128] can be written in the original variables:

$$\sum_{j=1}^{|J|} a_j \leq \sum_{j \in J} x_j \leq \sum_{j=m-|J|+1}^{m} a_j, \quad \text{all } J \subset \{1, \ldots, n\}$$

There are exponentially many constraints, but one can start by using only the constraints

$$\sum_{j=1}^{n} a_j \leq \sum_{j=1}^{n} x_j \leq \sum_{j=m-n+1}^{m} a_j$$

and bounds on the variables, and then generate separating cuts as needed. Let $\bar{x}$ be the solution of the current relaxation of the problem, and renumber the variables so that $\bar{x}_1 \leq \cdots \leq \bar{x}_n$. Then for each $i = 2, \ldots, n-1$ one can generate the cut

$$\sum_{j=1}^{i} x_j \geq \sum_{j=1}^{i} a_j$$

if $\sum_{j=1}^{i} \bar{x}_j < \sum_{j=1}^{i} a_j$. Also for each $i = n-1, \ldots, 2$ generate the cut

$$\sum_{j=i}^{n} x_j \leq \sum_{j=m-n+i}^{m} a_j$$

if $\sum_{j=i}^{n} \bar{x}_j > \sum_{j=m-n+i}^{m} a_j$. There is no separating cut if $\bar{x}$ lies within the convex hull of the alldiff feasible set.

Relaxation of multiple all-different constraints is discussed in [5] and of two overlapping all-different constraints in [34].

15.6.3 Circuit Constraint

The constraint

$$\text{circuit}(y_1, \ldots, y_n) \tag{15.17}$$

requires that $z_1, \ldots, z_n$ be a permutation of $1, \ldots, n$, where each $z_i = y_{z_{i-1}}$ (and z_0 is identified with z_n). The domain D_{y_i} of each y_i is a subset of $\{1, \ldots, n\}$. The elements $1, \ldots, n$ may be viewed as vertices of a directed graph G that contains an edge (i, j) whenever $j \in D_{y_i}$. An edge (i, j) is selected when $y_i = j$, and (15.17) requires that the selected edges form a hamiltonian circuit.

The circuit constraint can be modeled with a traveling salesman formulation. Let the 0-1 variable x_{ij} (for $i \neq j$) take the value 1 when $y_i = j$:

$$\begin{aligned} &\sum_{j=1}^{n} x_{ij} = \sum_{j=1}^{n} x_{ji} = 1, \ i \in \{1, \ldots, n\} && (a) \\ &\sum_{(i,j)\in\delta(S)} x_{ij} \geq 1, \ \text{all } S \subset \{1, \ldots, n\} \text{ with } 2 \leq |S| \leq n-2 && (b) \end{aligned} \tag{15.18}$$

Here $\delta(S)$ is the set of edges (i, j) of G for which $i \in S$ and $j \notin S$. If $j \notin D_{y_i}$, the formulation (15.18) omits variable x_{ij}. Constraints (a) comprise the assignment relaxation (15.7) already discussed. The *subtour elimination constraints* (b) exclude circuits within proper subsets S of $\{1, \ldots, n\}$ by requiring that at least one edge connect a vertex in S to one outside S.

The traveling salesman problem minimizes $\sum_i c_{ix_i}$ subject to (15.17). Its MILP formulation minimizes $\sum_{ij} c_{ij}x_{ij}$ subject to (15.18) and $x_{ij} \in \{0, 1\}$. There are exponentially many subtour elimination constraints in this formulation, but one can begin with a relaxation of the problem (such as the assignment relaxation) and add separating cuts as needed. If $\bar{x}$ is a solution of the current relaxation, let the *capacity* of edge (i, j) be $\bar{x}_{ij}$. Select a proper subset S of the vertices for which the total capacity of edges leaving S is a minimum. The subtour elimination constraint (15.18b) corresponding to S is a separating cut if the minimum capacity is less than 1. There are fast algorithms for finding S ([49, 103]).

If $j \in D_{y_i}$ if and only if $i \in D_{y_j}$, and $c_{ij} = c_{ji}$, for every pair i, j, the problem becomes the *symmetric* traveling salesman problem and can be given a somewhat more compact model that uses a *2-matching* relaxation. The OR literature has developed different (albeit related) analyses and algorithms for the symmetric and asymmetric problems; see [61] for a comprehensive discussion.

Several families of cutting planes have been developed to strengthen the LP relaxation. By far the most widely used are *comb inequalities*. Suppose H is a subset of vertices of G, and $T_1, \ldots, T_m$ are pairwise disjoint sets of vertices (where m is odd), such that $H \cap T_k$ and $T_k \setminus H$ are nonempty for each k. H is the *handle* of the comb and each T_i is a *tooth*. Then the following is a comb inequality for the asymmetric problem:

$$\sum_{(i,j)\in\delta(H)} x_{ij} + \sum_{k=1}^{m} \sum_{(i,j)\in\delta(T_k)} x_{ij} \geq \tfrac{1}{2}(3m+1)$$

One can get some intuition as to why the cut is valid by considering a comb with three teeth and six vertices. Various proofs of validity are given in [97]. The comb inequalities are facet-defining when G is a complete graph.

15.6.4 Cumulative Constraint

The fundamental constraint for cumulative scheduling is

$$\text{cumulative}(s, p, c, C) \tag{15.19}$$

where variables $s = (s_1, \ldots, s_n)$ represent the start times of jobs $1, \ldots, n$. Parameters $p = (p_1, \ldots, p_n)$ are the processing times and $c = (c_1, \ldots, c_n)$ the resource consumption rates. Release times and deadlines are implicit in the domains $[R_j, D_j]$ of the variables s_j. The constraint (15.19) requires that the total resource consumption rate at any time be less than or equal to C. That is, $\sum_{j \in J_t} c_j \leq C$ for all t, where $J_t = \{j \mid s_j \leq t < s_j + p_j\}$.

The most straightforward MILP model discretizes time and introduces a 0-1 variable x_{jt} that is 1 when job j starts at time t. The variable appears for a particular pair j, t only when $R_j \leq t \leq D_j - p_j$. The model is

$$\begin{aligned} &\sum_j \sum_{t' \in T_{jt}} c_j x_{jt'} \leq C, \text{ all } t && (a) \\ &\sum_t x_{jt} = 1, \quad \text{all } j && (b) \end{aligned} \tag{15.20}$$

where each $x_{ij} \in \{0, 1\}$ and $T_{jt} = \{t' \mid t - p_j < t' \leq t\}$. Constraints (a) enforce the resource limit, and (b) requires that each job start at some time.

Model (15.20) is large when there are a large number of discrete times. In such cases it may be advantageous to use one of two discrete-event formulations that employ continuous-time variables, although these models provide weaker relaxations. In each model there are $2n$ events, each of which can be the start of a job or the finish of a job. The continuous variable s_k is the time of event k.

In one model, the 0-1 variable $x_{jkk'}$ is 1 if event k is the start of job j and event k' is the finish of job j. The inventory variable z_k keeps track of how much resource is being consumed when event k occurs; obviously one wants $z_k \leq C$. The model for (15.19) is

$$\begin{aligned} &z_k = z_{k-1} + \sum_j \sum_{k'>k} c_j x_{jkk'} - \sum_j \sum_{k'<k} c_j x_{jk'k}, \quad \text{all } k && (a) \\ &z_0 = 0, \quad 0 \leq z_k \leq C, \quad \text{all } k && (b) \\ &\sum_k \sum_{k'>k} x_{jkk'} = 1, \quad \text{all } j && (c) \\ &s_{k'} - s_k \geq \sum_j p_j x_{jkk'}, \quad \text{all } k, k' \text{ with } k < k' && (d) \\ &s_k \geq \sum_j \sum_{k'>k} x_{jkk'} R_j, \quad \text{all } k && (e) \\ &t_k \leq D_{\max} \left(1 - \sum_j \sum_{k'<k} x_{jk'k} \right) + \sum_j \sum_{k'<k} x_{jk'k} D_j, \quad \text{all } k && (f) \end{aligned} \tag{15.21}$$

where each $x_{jkk'} \in \{0, 1\}$ and $D_{\max} = \max_j \{D_j\}$. Constraint (a) keeps track of how much resource is being consumed, and (b) imposes the upper limit. Constraint (c) makes sure that each job starts once and ends once. Constraint (d) presents jobs from overlapping,

and (e) enforces the release times. Constraint (f) uses a big- M construction (where $D_{\max}$ is the big-M) to enforce the deadlines.

Model (15.21) can grow quite large if there are too many events, due to the triply indexed variables $x_{jkk'}$. An alternative is to use separate variables for start-events and finish-events, which requires that the deadlines be enforced in a different way. This reduces the triple index to a double index at the cost of producing a weaker relaxation. Let the 0-1 variable x_{jk} be 1 when event k is the start of job j, and $y_{jk} = 1$ when event k is the finish of job j. The new continuous variable f_j is the finish time of job j.

$$
\begin{aligned}
& z_k = z_{k-1} + \sum_j c_j x_{jk} - \sum_j c_j y_{jk}, \quad \text{all } k && (a)\\
& z_0 = 0, \quad z_k \le C, \quad \text{all } k && (b)\\
& \sum_k x_{jk} = 1, \quad \sum_k y_{jk} = 1, \quad \text{all } j && (c)\\
& \sum_j x_{jk} + y_{jk} = 1, \quad \text{all } k && (d)\\
& s_{k-1} \le s_k, \quad x_{jk} \le \sum_{k'<k} y_{jk'}, \quad \text{all } k > 1 && (e)\\
& s_k \ge \sum_j R_j x_{jk}, \quad \text{all } k && (f)\\
& s_k + p_j x_{jk} - D_{\max}(1 - x_{jk}) \le f_j \\
& \qquad\qquad \le s_k + D_{\max}(1 - y_{jk}), \quad \text{all } j,k && (g)\\
& f_j \le d_j, \quad \text{all } j && (h)
\end{aligned}
$$

where each $x_{jk}, y_{jk} \in \{0,1\}$. Constraints (a) and (b) perform the same function as before. Constraints (c) and (d) require that each job start once and end once but not as the same event. Constraints (e) are redundant but may tighten the relaxation. One constraint requires the events to occur in chronological order, and one requires a job's start-event to have a smaller index than its finish-event. Constraint (f) observes the release times. The new element is constraint (g). The first inequality defines the defines the finish time f_j of each job by forcing it to occur no earlier than p_j time units after the start time. The second inequality forces the time associated with the finish-event to be no earlier than the finish time. Finally, constraint (h) enforces the deadlines.

A fourth relaxation uses only the original variables s_j [75]. Let $J = \{j_1, \ldots, j_m\}$ be any subset of the jobs $\{1, \ldots, n\}$, indexed so that $p_{j_1} c_{j_1} \le \cdots \le p_{j_m} c_{j_m}$. Then

$$
\begin{aligned}
& \sum_{j \in J} s_j \ge m R_{\min} + \frac{1}{C} \sum_{i=1}^{m} (m - i + 1) p_{j_i} c_{j_i} - \sum_{i=1}^{m} p_{j_i}\\
& \sum_{j \in J} s_j \le m D_{\max} - \frac{1}{C} \sum_{i=1}^{m} (m - i + 1) p_{j_i} c_{j_i}
\end{aligned}
$$

where $R_{\min} = \min_i \{R_{j_i}\}$ and $D_{\max} = \max_i \{D_{k_i}\}$. A relaxation can be created by using these inequalities for selected subsets of jobs. One need only consider subsets J of the form $\{j \mid [R_j, D_j] \subset [R_i, D_k]\}$ for pairs i, k with $R_i < D_k$. Larger subsets tend to yield much stronger inequalities.

15.7 Relaxation of Piecewise Linear and Disjunctive Constraints

Specialized relaxations can be devised for a number of additional constraints that commonly occur in modeling. Two such constraints are bounds on piecewise linear, semicontinuous cost functions, and disjunctions of nonlinear inequality systems. There are also convex hull relaxations (not discussed here) for various logical constraints [70], such as cardinality rules requiring that if at least k of a given set of propositions are true, then at least ℓ of another set are true ([132], generalized in [10]).

15.7.1 Semicontinuous Piecewise Linear Functions

Piecewise linear functions are often useful in modeling, partly because they can approximate nonlinear functions. A semicontinuous piecewise linear function $f(x)$ can be written

$$f(x) = \frac{U_k - x}{U_k - L_k} c_k + \frac{x - L_k}{U_k - L_k} d_k \;\; \text{for } x \in [L_k, U_k], \;\; k = 1, \ldots, m$$

and is illustrated in Fig. 15.3.

The function $f(x)$ can be given an MILP model by replacing all occurrences of $f(x)$ with a new continuous variable v and writing

$$v = \sum_k \lambda_k a_k + \mu_k b_k, \;\; x = \sum_k \lambda_k L_k + \mu_k U_k, \;\; \sum_k \lambda_k + \mu_k = 1, \;\; \sum_k y_k = 1$$
$$0 \le \lambda_k \le y_k, \;\; 0 \le \mu_k \le y_k, \;\; \text{all } k$$

where each $y_k \in \{0, 1\}$. An LP relaxation is obtained by replacing $y_j \in \{0, 1\}$ by $0 \le y_j \le 1$. The MILP model for continuous piecewise linear functions is slightly simpler; see [127]. An MILP model is used with a probe backtrack algorithm in [2].

Computational studies [101, 102, 105] suggest that it is often more efficient to write a convex hull relaxation directly in the original variables and dispense with the auxiliary variables λ_k, μ_k, y_k. One simply writes an inequality in v, x to represent each edge of the convex hull, illustrated in Fig. 15.3.

15.7.2 Disjunctions of Nonlinear Systems

Methods for relaxing disjunctions of linear systems were presented in Section 15.4.3. They have proved useful for relaxing problems that combine discrete and continuous linear elements. Many applications, however, mix discrete and continuous *nonlinear* elements. Fortunately, the convex hull relaxation for the linear case can be generalized to the convex nonlinear case. It has been used to solve several chemical process engineering problems [89, 90, 104, 112, 125]

The task is to relax

$$\bigvee_{k \in K} g^k(x) \le 0 \tag{15.22}$$

where $g^k(x)$ is a vector of convex functions and $x \in \Re^n$. It is assumed that x and each $g^k(x)$ are bounded, and in particular that $L \le x \le U$. A convex hull relaxation for (15.22)

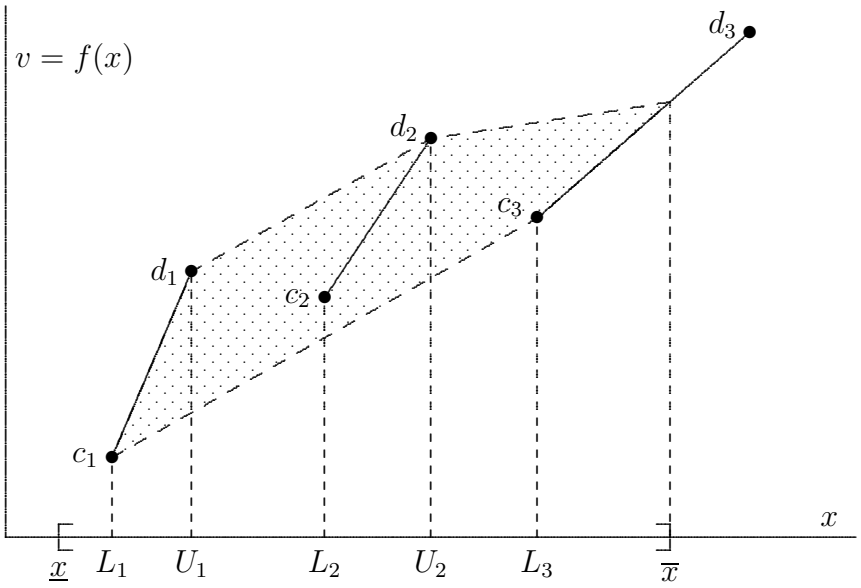

Figure 15.3: *Semicontinuous piecewise linear function $f(x)$ (solid line segments) and its convex hull relaxation (shaded area), where $[\underline{x}, \overline{x}]$ is the current domain of x.*

can be derived by writing x as a convex combination of points $\hat{x}^k$ that respectively satisfy the disjuncts of (15.22):

$$x = \sum_{k \in K} \beta_k \hat{x}^k, \quad \sum_{k \in K} \beta_k = 1$$
$$g^k(\hat{x}^k) \leq 0, \quad \beta_k \geq 0, \quad L \leq \hat{x}^k \leq U, \quad \text{all } k \in K$$

Using the change of variable $x^k = \beta_k \hat{x}^k$ and multiplying the nonlinear constraints by β_k yields a continuous relaxation for (15.22):

$$\begin{aligned} & x = \sum_{k \in K} x^k, \quad \sum_{k \in K} \beta_k = 1 \\ & \beta_k g^k \left(\frac{x^k}{\beta_k} \right) \leq 0, \quad \beta_k \geq 0, \quad \beta_k L \leq x^k \leq \beta_k U, \quad \text{all } k \in K \end{aligned} \tag{15.23}$$

It can be shown that the functions $\beta_k g^k(x^k/\beta_k)$ are convex and moreover that (15.23) is a convex hull relaxation of (15.22) [89, 119]. Since β_k can vanish, it is common in practice to introduce a perturbation ϵ, which preserves convexity. The nonlinear constraints of (15.23) become

$$(\beta_k + \epsilon) g^k \left(\frac{x^k}{\beta_k + \epsilon} \right) \leq 0, \quad \text{all } k \in K$$

One can also relax (15.22) with a big-M relaxation similar to the LP relaxation of (15.11):

$$g^k(x) \le M^k(1-\beta_k), \quad \beta_k \ge 0, \quad \text{all } k \in K$$
$$\sum_{k\in K} \beta_k = 1$$

where each component of M^k is an upper bound on the corresponding function in $g^k(x)$.

15.8 Lagrangean Relaxation

Many relaxations are formed simply by removing constraints. *Lagrangean relaxation* refines this process by imposing a penalty for violation of the deleted constraints, provided they have inequality form. This tends to make the minimum value of the relaxed problem larger and may therefore provide a tighter bound on the optimal value of the original problem.

A Lagrangean relaxation is actually a family of relaxations, since one can choose the penalty (i.e., the Lagrange multiplier) for each constraint violation. The problem of selecting the multipliers that result in the tightest bound is the *Lagrangean dual*.

Since solution of the Lagrangean dual tends to be time consuming, the general practice in CP applications is to solve it only at the root node and use the resulting Lagrange multipliers at all subsequent nodes. These multipliers not only define a relaxation that can be quickly solved or propagated, but they allow domain filtering in the same fashion as the dual multipliers of an LP relaxation. In fact, dual multipliers are a special case of Lagrange multipliers.

Lagrangean relaxation is particularly useful in CP when a full linear relaxation is incompatible with the very rapid processing of search tree nodes that often characterizes CP methods. In such cases one may be able to solve a Lagrangean dual of the LP at the root node, which is itself an LP problem, to obtain a weaker but specially-structured LP relaxation that can be solved quickly at the remaining nodes.

CP-based Lagrangean methods have been applied to network design [36], automatic digital recording [113], traveling tournament problems [17], the resource-constrained shortest path problem [54], and the general problem of filtering domains [85].

15.8.1 The Lagrangean Dual

Lagrangean relaxation is applied to problems of the form

$$\begin{aligned} &\min f(x) \\ &g_i(x) \le 0, \quad i = 1,\dots,m \\ &x \in S, \quad x \in \Re^n \end{aligned} \tag{15.24}$$

where $x \in S$ represents an arbitrary constraint set but in practice is carefully chosen to have special structure. Lagrangean relaxation *dualizes* the constraints $g_i(x) \le 0$ by penalizing their violation in the objective function:

$$\begin{aligned} &\min \theta(x,\lambda) = f(x) + \sum_{i-1}^{m} \lambda_i g_i(x) \\ &x \in S, \quad x \in \Re^n \end{aligned} \tag{15.25}$$

where each $\lambda_i \geq 0$ is a *Lagrange multiplier*. The motivation for using the relaxation is that the special structure of the constraints $x \in S$ makes it easy to solve. The term "penalty" is not quite right for the expression $\lambda_i g_i(x)$, however, since a penalty should vanish when the solution is feasible; $\lambda_i g_i(x)$ can go negative when x is feasible, resulting in a "saddle" function.

Clearly $\theta(x, \lambda) \leq f(x)$ for any x that is feasible in the relaxation (15.25), since $\lambda \geq 0$ and each $g_i(x) \leq 0$. Thus for any $\lambda \geq 0$, the optimal value $\theta(\lambda)$ of the Lagrangean relaxation (15.25) is a lower bound on the optimal value v^* of the original problem (15.24) (*weak duality*). The tightest lower bound on v^* is obtained by solving the *Lagrangean dual* problem: maximize $\theta(\lambda)$ subject to $\lambda \geq 0$. The amount by which this bound falls short of v^* is the *duality gap*.

15.8.2 Solving the Dual

Since the Lagrangean function $\theta(\lambda)$ is concave, it can be maximized by finding a local maximum. A popular approach is *subgradient optimization*, which begins with a starting value $\lambda^0 \geq 0$ and sets each $\lambda^{k+1} = \lambda^k + \alpha_k \sigma^k$, where σ^k is a subgradient of $\theta(\lambda)$ at $\lambda = \lambda^k$. Conveniently, if $\theta(\lambda^k) = f(x^k, \lambda)$, then $(g_1(x^k), \ldots, g_m(x^k))$ is a subgradient. The stepsize α_k should decrease as k increases, but not so quickly as to cause premature convergence. A simple option is to set $\alpha_k = \alpha_0/k$, or perhaps $\alpha_k = \alpha_0(\overline{\theta} - \theta(\lambda^k))/||\sigma^k||$, where $\overline{\theta}$ is a dynamically adjusted upper bound on the maximum value of $\theta(\lambda)$. The stepsize is highly problem-dependent and must be tuned for every problem class. Solution methods are further discussed in [11, 98].

Typically the Lagrangean dual is solved only at the root node. As one descends into the search tree, branching adds constraints to the problem, and the dual solution λ^* ceases to be optimal. Nonetheless the optimal value of the Lagrangean relaxation, with λ set to λ^* and branching constraints added, continues to be a valid lower bound of the optimal solution. In fact there is no need to obtain optimal λ_is even at the root node, and frequently the subgradient algorithm is terminated early.

One must take care that branching constraints do not destroy the special structure of the Lagrangean relaxation. For instance, one may wish to branch by fixing one or variables rather than adding inequality constraints.

15.8.3 Special Case: Linear Programming

The Lagrangean dual of an LP problem is easy to solve, since it is equivalent to the LP dual. Suppose the original problem (15.24) is an LP problem

$$\begin{aligned} &\min cx \\ &Ax \geq b, \quad Dx \geq d, \quad x \geq 0 \end{aligned} \tag{15.26}$$

in which the linear system $Dx \geq d$ has some kind of special structure. If $Ax \geq b$ is dualized, then $\theta(\lambda)$ is the optimal value of

$$\begin{aligned} &\min \theta(x, \lambda) = cx + \lambda(b - Ax) = (c - \lambda A)x + \lambda b \\ &Dx \geq d, \quad x \geq 0 \end{aligned} \tag{15.27}$$

Suppose x^* is an optimal solution of the original LP problem (15.26), and (λ^*, μ^*) is an optimal dual solution in which λ^* corresponds to $Ax \geq b$ and μ^* to $Dx \geq d$. Thus

$$cx^* = \lambda^* b + \mu^* d \tag{15.28}$$

by strong duality. It will be shown below that $\theta(\lambda^*) = cx^*$. This implies that λ^* is an optimal dual solution of the Lagrangean dual problem, since $\theta(\lambda)$ is a lower bound on cx^* for any $\lambda \geq 0$.

One can therefore solve the Lagrangean dual of (15.26) at the root node by solving its LP dual. If (λ^*, μ^*) solves the LP dual, then λ^* solves the Lagrangean dual in which $Ax \geq b$ is dualized. At subsequent nodes one solves the specially structured LP problem (15.27), with λ set to the value λ^* obtained at the root node, to obtain a valid lower bound on the value of the LP relaxation at that node.

To see that $\theta(\lambda^*) = cx^*$, note first that x^* is feasible in

$$\min_{x \geq 0} \{(c - \lambda^* A)x \mid Dx \geq d\} \tag{15.29}$$

and μ^* is feasible in its LP dual

$$\max_{\mu \geq 0} \{ud \mid \mu D \leq c - \lambda^* A\} \tag{15.30}$$

where the latter is true because (λ^*, μ^*) is dual feasible for (15.26). But the corresponding objective function value of (15.29) is

$$(c - \lambda^*)x^* = cx^* + \lambda^*(b - Ax^*) - \lambda^* b = cx^* - \lambda^* b$$

where the second equation is due to complementary slackness. This is equal to the value $\mu^* d$ of (15.30), due to (15.28). So $cx^* - \lambda^* b$ is the optimal value of (15.29), which means that cx^* is the optimal value $\theta(\lambda^*)$ of (15.27) when $\lambda = \lambda^*$.

15.8.4 Domain Filtering

Lagrangean duality provides a generalization of the filtering mechanism based on LP duality. Suppose that there is an upper bound U on the cost $f(x)$ in (15.24), and let $v^* = \theta(\lambda^*)$ be the optimal value of the Lagrangean dual. Let x^* solve (15.25) when $\lambda = \lambda^*$, so that $\theta(\lambda^*) = \theta(\lambda^*, x^*)$, and suppose further that $g_i(x^*) = 0$. If the solution of (15.24) were to change, function $g_i(x)$ could decrease, say by an amount Δ_i. This would increase the optimal value of (15.24) as much as changing the the constraint $g_i(x) \leq 0$ to $g_i(x) + \Delta_i \leq 0$. The function $\theta(\lambda)$ for the altered problem is $\theta'(\lambda) = \min_{x \in S}\{\theta'(\lambda, x)\}$, where $\theta'(\lambda, x) = f(x) + \sum_j \lambda_j g_j(x) + \lambda_i \Delta_i$. Since $\theta'(\lambda^*, x)$ differs from $\theta(\lambda^*, x)$ only by a constant, any x that minimizes $\theta(\lambda^*, x)$ also minimizes $\theta'(\lambda^*, x)$. So

$$\theta'(\lambda^*) = \theta'(\lambda^*, x^*) + \lambda_i^* \Delta_i = v^* + \lambda_i^* \Delta_i$$

is a lower bound on the optimal value of the altered problem, by weak duality. Thus one must have $v^* + \lambda^* \Delta_i \leq U$, or $\Delta_i \leq (U - v^*)/\lambda_i^*$. Since $\Delta_i = g_i(x^*) - g_i(x) = -g_i(x)$, this yields a valid inequality parallel to (15.6) that can be propagated:

$$g_i(x) \geq -\frac{U - v^*}{\lambda_i^*} \tag{15.31}$$

If constraint i imposes a lower bound L on a variable x_j (i.e., $-x_j + L \leq 0$), then (15.31) becomes $x_j \leq L + (U - v^*)/\lambda_i^*$. This can be used to reduce the domain of x_j if $\lambda_i^* > 0$, and similarly if there is an upper bound on x_j.

15.8.5 Example: Generalized Assignment Problem

The *generalized assignment problem* is an assignment problem (15.7) with the complicating constraint that the jobs j assigned to each resource i satisfy $\sum_j \alpha_{ij} x_{ij} \leq \beta_i$. Let's suppose that an LP relaxation of the problem is to be solved at each node of the search tree to obtain bounds. If solving this LP with a general-purpose solver is too slow, the complicating constraints can be dualized, resulting in a pure assignment problem with cost function $\sum_{ij}(c_{ij} - \lambda_i^* \alpha_{ij})x_{ij}$. The optimal multipliers λ_i^* can be obtained at the root node by solving the full LP relaxation.

At subsequent nodes one can solve the Lagrangean relaxation very quickly with the Hungarian algorithm. The relaxation provides a weak bound, but the dual variables allow useful domain filtering. The search branches by setting some x_{ij} to 0 or 1, which in turn can be achieved by giving x_{ij} a very large or very small cost in the objective function of the Lagrangean relaxation, thus preserving the problem structure.

15.9 Dynamic Programming

Dynamic programming (DP) exploits recursive or nested structure in a problem. A DP model provides one more opportunity for an OR-based relaxation of a CP problem, and the DP model can itself be relaxed to reduce time and space consumption. Since DP models express the optimal value as a recursively defined function, they can often be coded in a CP modeling language. DP models are also particularly amendable to filtering discrete domains, including cost-based filtering.

15.9.1 Recursive Optimization

Given the problem of minimizing $f(x)$ subject to $x \in S$, a DP model defines a sequence of *state variables* $s_1, \ldots, s_n$. The original variables $x = (x_1, \ldots, x_n)$ are viewed as a sequence of *controls*. Each control x_k brings about a transition from state s_k to state $t_k(x_k, s_k)$. The optimization problem is viewed as finding a minimum-cost sequence of controls.

The key property of a DP model is that each state s_k must contain enough information to determine the set $X_k(s_k)$ of feasible controls and the cost $g_k(s_k, x_k)$ of applying each control x_k, without knowing how one got to state s_k. Thus if each $x_k \in X_k(s_k)$ and each $s_{k+1} = t_k(x_k, s_k)$, then $x \in S$ and $f(x) = \sum_{k=1}^{n} g_k(s_k, x_k)$.

This structure immediately leads to recursive equations, known as *Bellman's equations*, that solve the problem. The computation works backward from a final state s_{n+1}:

$$f_k(s_k) = \min_{x_k \in X_k(s_k)} \{g_k(s_k, x_k) + f_{k+1}(t(x_k, s_k))\}, \quad k = n, \ldots, 1 \tag{15.32}$$

where s_1 is the initial state and $f_1(s_1)$ is the optimal value $min\{f(x) \mid x \in S\}$. The cost $f_k(s_k)$ is interpreted as the minimum cost of going from state s_k to a final state. The final costs $f_{n+1}(s_{n+1})$ are given as *boundary conditions*.

A DP algorithm compiles a table of each $f_k(s_k)$ for all values of s_k based on the previously computed table of $f_{k+1}(s_{k+1})$. At each stage k the set $X_k^*(s_k)$ of controls that yield the minimum in (15.32) for each s_k is recorded. When $f_1(s_1)$ is obtained, the algorithm works forward to obtain an optimal solution $(x_1^*, \ldots, x_n^*)$ and optimal sequence of states

$s_1^*, \ldots, s_n^*$ by letting each x_k^* be any element of $X_k^*(s_k^*)$ and each $s_k^* = g_k(x_{k-1}^*, s_{k-1}^*)$, where $s_1^* = s_1$.

For instance, suppose one wishes to find a shortest path from every vertex of a directed acyclic graph (V, E) to vertex $n+1$. If the length of edge (i,j) is c_{ij} with $c_{ii} = 0$, the recursion is $f_k(i) = \min_{j \in E(i)} \{c_{ij} + f_{k+1}(j)\}$, where $E(i) = \{i\} \cup \{j \mid (i,j) \in E\}$. The cost $f_k(i)$ is interpreted as the length of the shortest path from vertex i to vertex $n+1$ having $n-k+1$ or fewer edges. The boundary condition is $f_{n+1}(n+1) = 0$.

The art of dynamic programming lies in identifying state variables s_k that do not have too many possible values. Only certain problems have a recursive structure that can be exploited in this manner, but many examples of these can be found in [19, 42]

15.9.2 Domain Filtering

An important special case arises when DP is applied to finding feasible solutions of a constraint C, since this allows one to achieve hyperarc consistency for C. In this case one can view the objective function $f(x)$ as equal to 1 when x violates C and 0 otherwise. The boundary conditions are defined by letting $f_{n+1}(s_{n+1})$ be 0 for all final states s_{n+1} that satisfy C and 1 for all other states. All other costs $g_k(s_k, x_k) = 0$. So if $f_k(s_k) = 0$, $X_k^*(s_k)$ is the set of all controls x_k that can lead to a feasible solution when they are applied in state s_k. This means $\bigcup_{s_k | f_k(s_k)=0} X_k^*(s_k)$ is the set of values of x_k that can lead to a feasible solution, and hyperarc consistency is achieved by reducing the domain of x_k to this set.

15.9.3 Example: Cost-Based Filtering

Suppose that a cost constraint $ax \le b$ is given, and one wishes to filter the variable domains D_{x_k}. A classical DP recursion for finding all feasible solutions defines the state variable s_k to be the sum $\sum_{j<k} a_j x_j$ of the first $k-1$ terms on the left-hand side of $ax \le b$. The recursion is

$$f_k(s_k) = \min_{x_k \in D_{x_k}} \{f_{k+1}(s_k + a_k x_k)\}$$

with the boundary condition $f_{n+1}(s_{n+1}) = 0$ for $s_{n+1} \le b$ and $f_{n+1}(s_{n+1}) = 1$ for $s_{n+1} > b$. Note that $g_k(s_k, x_k) = 0$ in this recursion.

If the absolute value of the coefficients a_k is bounded, then the number of states (values of s_k for all k) is polynomially bounded and the DP recursion has polynomial complexity.

DP therefore provides a pseudopolynomial algorithm that achieves hyperarc consistency for $ax \le b$. For instance, if $ax \le b$ is $4x_1 + 2x_2 + 3x_3 \le 12$ and each $D_{x_k} = \{1, 2\}$, then possible states s_k are illustrated in Fig. 15.4. Every solution (x_1, x_2, x_3) defines a path $s_1 \to s_2 \to s_3 \to s_4$ through the network. Heavy lines show paths that correspond to optimal solutions (i.e., $f(x) = 0$) and therefore feasible solutions of $4x_1 + 2x_2 + 3x_3 \le 12$. For instance, $f_3(6) = f_3(8) = 0$ since $s_3 = 4x_1 + 2x_2$ can be either 6 or 8 in an optimal solution, but $f_3(s_3) = 1$ for all other s_3.

One can read the filtered domains from the network. For instance, the filtered domain of x_3 is $\bigcup_{s_3 | f(s_3)=0} X_3^*(s_3) = X_3^*(6) \cup X_3^*(8) = \{1,2\} \cup \{1\} = \{1,2\}$. The reduced domains of x_1 and x_2 are $\{1\}$ and $\{1,2\}$, respectively. This idea is developed further in [123].

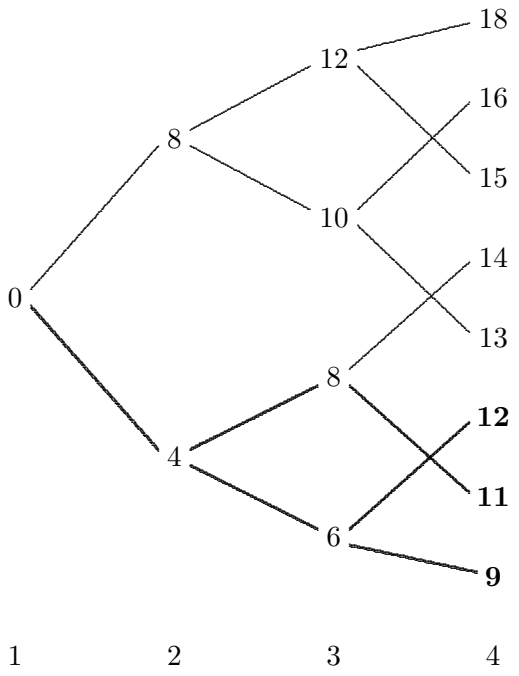

Figure 15.4: *States s_k for the constraint $4x_1 + 2x_2 + 3x_3 \leq 12$ with $x_j \in \{1, 2\}$. Edges leaving state 0 correspond to $x_1 = 1, 2$, and similarly for other edges. Costs associated with the terminal states ($k = 4$) are 0 for the boldface states and 1 for the others. Heavy lines correspond to paths that lead to optimal solutions (i.e. solutions feasible for $4x_1 + 2x_2 + 3x_3 \leq 12$).*

15.9.4 State Space Relaxation

When the state variables s_k have too many values, *state space relaxation* can reduce the number of values and make the problem easier to solve. State space relaxation in effect uses a hash code for the states; it defines a function $\phi(s_k)$ that maps many values of s_k to the same state. Every control $x_k \in X_k(s_k)$ is mapped to a control that takes the system from $\phi(s_k)$ to $\phi(g_k(s_k, x_k))$. The cost function $\hat{g}_k$ for the relaxation must satisfy $\hat{g}_k(\phi(s_k), \phi(y_k)) \leq g_k(s_k, x_k)$. The optimal value of the relaxed problem is a lower bound on the optimal value of the original recursion.

This can be illustrated by relaxing a DP formulation of the traveling salesman problem on a graph (V, E). The objective is to minimize $\sum_i c_{x_i x_{i+1}}$ subject to alldiff$(x_1, \ldots, x_n)$ and $(x_i, x_{i+1}) \in E$. Let x_1 be the first customer visited after leaving home base i_0, and fix the last customer visited x_n to be home base. The classical DP formulation defines state variable s_k to be (i, V_k), where V_k is any set of $n - k + 1$ vertices and $i \in V_k$. The cost $f_k(i, V_k)$ is interpreted as the cost of the minimum-cost tour that starts at i and covers all

the vertices in V_k. If $E_i = \{j \mid (i,j) \in E\}$, the recursion is

$$f_k(i, V_k) = \min_{x_k \in (V_k \cap E_i) \setminus \{i\}} \{c_{ix_k} + f_{k+1}(x_k, V_k \setminus \{i\})\} \tag{15.33}$$

with boundary condition $f_n(i,\{i\}) = c_{ii_0}$ for all vertices i with $i_0 \in E_i$ and $f_n(i,\{i\}) = \infty$ for all other i.

The recursion (15.33) is not computationally useful because there are exponentially many states (i, V_k). However, this DP model can be relaxed, for example by mapping all states (i, V_k) for a given k to $\phi(i, V_k) = (i,k)$. The cost of a transition from (i,k) to $(j, k+1)$ is the same as that from any (i, V_k) to any (j, V_{k+1}), namely c_{ij}. The recursion (15.33) becomes

$$\hat{f}_k(i) = \min_{x_k \in E_i} \left\{ c_{ix_k} + \hat{f}_{k+1}(x_k) \right\} \tag{15.34}$$

with the same boundary condition as before. A CP solution of (15.34) and other issues are discussed in [53].

15.9.5 Nonserial Dynamic Programming

DP is based on the principle that each state s_k depends only on the previous state s_{k-1} and control x_{k-1}. In *nonserial dynamic programming* (NSDP), a state may depend on several previous states.

NSDP has been known in OR for more than 30 years [18]. Essentially the same idea has surfaced in a number of contexts, including Bayesian networks [88], belief logics [115, 117], pseudoboolean optimization [35], location theory [28], k-trees [7, 8], and bucket elimination [39].

In the simplest form of NSDP, the state variables s_k are the original variables x_k. This is the form most relevant to CP, since it permits solution of a constraint set C in time that is directly related to the width of the dependency graph G of C.

The *width* of a directed graph G is the maximum in-degree of vertices of G. The *induced width* of G with respect to an ordering of vertices $1, \ldots, n$ is the width of G' with respect to this ordering, where G' is constructed as follows. Remove vertices $n, n-1, \ldots, 1$ from G one a time, and when each vertex i is removed, add edges as necessary so that the vertices adjacent to i at the time of its removal form a clique. Then G' consists of G plus all edges added in this process.

The dependency graph G for constraint set C contains a vertex for each variable x_j of C and an edge (x_i, x_j) when x_i and x_j occur in a common constraint. Let S_k be the set of vertices in $\{1, \ldots, k-1\}$ that are adjacent to k in G', and let x^i be the set of variables in constraint $C_i \in C$. Define the cost function $c_i(x^i)$ to be 1 if x^i violates C_i and 0 otherwise. Then the NSDP recursion again works backward:

$$f_k(S_k) = \min_{x_k \in D_{x_k}} \left\{ \sum_{i \in I_k} c_i(x^i) + \sum_{j \in J_k} f_j(S_j) \right\}, \quad k = 1, \ldots, n \tag{15.35}$$

where I_k is the set of indices i for which x^i contains x_k but none of $x_{k+1}, \ldots, x_n$. J_k is the set of indices $j \in \{k+1, \ldots, \ldots, n\}$ for which S_j contains x_k but none of $x_{k+1}, \ldots, x_n$. Note that the computation of $f_k(S_k)$ may use previously computed costs $f_i(S_i)$ for several

$i \in \{k+1, \ldots, n\}$. The process gets started by computing $f_n(S_n)$, which requires no previous results. The optimal value $\sum_{k|S_k=\emptyset} f_k(\emptyset)$ is 0 if and only if C has a feasible solution. A feasible solution is recovered by recording for each k the set $X_k^*(S_k)$ of values of x_k that achieve the minimum value of zero in (15.35). The for $k = 1, \ldots n$ one can select any $x_k^* \in X_k^*(S_k^*)$, where S_k^* contains the previously selected values for $x_j \in S_k$.

The complexity of the recursion (15.35) is at worst proportional to nD^{w+1}, where D is the size of the largest variable domain, and w is the size of the largest set S_k. But w is the width of G' and therefore the induced with of G with respect to the ordering $x_1, \ldots, x_n$. So the complexity of solving a constraint set C by NSDP is at worst exponential in the induced width of C's dependency graph with respect to the reverse order of recursion. These ideas are further discussed in [70].

15.10 Branch-and-Price Methods

Branch and price is a well-known OR technique that is applied to MILP problems with a very large number of variables. In fact, the MILP model is sometimes reformulated so as to have exponentially many variables, since this may simplify the model while yet allowing solution by branch and price.

The basic idea is to solve the MILP initially with only a few variables, and add variables to the problem as they are needed to improve on the current solution. A subproblem is solved to identify promising variables. Since a variable is added to the problem by adding a column to the MILP constraint matrix, this approach is often known as *column generation*. Complicating constraints can be dealt with implicitly by restricting what sort of columns can be generated by the subproblem.

It is in the column generation phase that CP can be useful, since the subproblem may have complicated constraints that make it more amenable to solution by CP rather than OR methods. CP-based branch and price has proved successful in several applications that involve assignment of resources under complex constraints.

15.10.1 The Algorithm

Branch and price is applied to an MILP problem

$$\begin{aligned} &\min cy \\ &Ay = b, \;\; y \geq 0, \;\; y_\ell \text{ integer for } \ell \in I \end{aligned} \tag{15.36}$$

where $I \subset \{1, \ldots, n\}$. The algorithm is a branch-and-bound search that solves the LP relaxation at each node of the search tree by column generation.

The column generation procedure begins with an LP relaxation that contains a subset of the variables:

$$\begin{aligned} &\min \sum_{\ell \in L} c_\ell y_\ell \\ &\sum_{\ell \in L} A_\ell y_\ell = b; \;\; y_\ell \geq 0, \;\; \ell \in L \end{aligned} \tag{15.37}$$

where A_ℓ is column ℓ of A and $L \subset \{1, \ldots, n\}$. If $\lambda^* = (\lambda_1^*, \ldots, \lambda_m^*)$ is the optimal dual solution of (15.37), then any nonbasic variable y_ℓ has reduced cost $c_\ell - \lambda^* A_\ell$ (see

Section 15.3). A subproblem is solved to find a column (c_ℓ, A_ℓ) with the smallest reduced cost. Thus the subproblem minimizes $z_0 - \sum_{i=1}^m \lambda_i^* z_i$ subject to $(z_0, z_1, \ldots, z_m) \in Z$, where Z is the set of all columns (c_ℓ, A_ℓ). If a column with negative reduced cost is found, it is added to (15.37). The process is repeated until no column with negative reduce cost is found in the subproblem, whereupon (15.37) is solved. If all goes well, only a small fraction of the columns of A will have been generated.

15.10.2 Example: Generalized Assignment Problem

Most applications of branch and price involve an assignment problem with complicating constraints. One example is the generalized assignment problem, which again is an assignment problem (15.7) with the complicating constraint that the jobs j assigned to each resource i satisfy $\sum_j \alpha_{ij} x_{ij} \le \beta_i$.

The problem is reformulated for branch and price by letting k index all possible assignments of jobs to a given resource i that satisfy $\sum_j \alpha_{ij} x_{ij} \le \beta_i$. The 0-1 variable $y_{ik} = 1$ if the kth assignment to resource i is selected. So if $\delta_{ijk} = 1$ when job j is assigned to resource i in assignment k, an LP relaxation of the problem has the form (15.36) with $\ell = (i,k)$:

$$\begin{aligned}
&\min \sum_{ik} \left(\sum_j c_{ij} \delta_{ijk} \right) y_{ik} \\
&\sum_{ik} \delta_{ijk} y_{ik} = 1, \quad \text{all } j \qquad (a) \\
&\sum_k y_{ik} = 1, \quad \text{all } i \qquad (b) \\
&y_{ik} \ge 0, \quad \text{all } i,k
\end{aligned} \tag{15.38}$$

If dual variables λ_j are associated with constraints (a) and μ_i with constraints (b), the reduced cost of a variable y_{ik} in (15.38) is

$$\sum_j c_{ij} \delta_{ijk} - \sum_j \lambda_j \delta_{ijk} - \mu_i = \sum_j (c_{ij} - \lambda_j) \delta_{ijk} - \mu_i$$

The subproblem of finding a variable y_{ik} with negative reduced cost can be solved by examining each resource i separately and solving the 0-1 knapsack problem

$$\begin{aligned}
&\min \sum_j (c_{ij} - \lambda_j) z_j - \mu_i \\
&\sum_j \alpha_{ij} z_j \le \beta_i; \quad z_j \in \{0,1\}, \quad \text{all } j
\end{aligned}$$

0-1 knapsack problems can be solved by a number of methods [95], including CP [48].

15.10.3 Other Applications

One of the most successful applications of CP-based branch and price is to airline crew assignment and crew rostering [26, 47, 82, 87, 114]. In [47], for example, a path constraint

is used to obtain a permissible roster with a negative reduced cost. The search tree is pruned by solving a relaxation of the problem (a single-source shortest path problem) so as to obtain a lower bound on the reduced cost.

CP-based branch and price has also been applied to transit bus crew scheduling [133], aircraft scheduling [59], vehicle routing [109], network design [27], employee timetabling [41], physician scheduling [55], and the traveling tournament problem [43, 44]. Several implementation issues are discussed in [45, 108].

15.11 Benders Decomposition

Benders decomposition was developed in the context of mathematical programming, but the root idea has much wider application. It solves a problem by enumerating, in a *master problem*, possible values of a pre-selected subset of variables. Each set of values that might be assigned to these variables defines the *subproblem* of finding the best values for the remaining variables. Solution of the subproblem generates a nogood, known as a *Benders cut*, that excludes that particular assignment to the master problem variables, and perhaps other assignments that can be no better. The master problem is re-solved with the new Benders cut in order to find a better solution, and the process continues until no further improvement is possible.

Benders decomposition can profitably combine OR and CP, since one approach can be applied to the master problem and one to the subproblem, depending on which best suits the problem structure. This sort of combination has yielded substantial speedups in computation.

15.11.1 Benders Decomposition in the Abstract

In classical Benders decomposition, the subproblem is a linear or nonlinear programming problem [15, 56], and the Benders cuts are generated using dual or Lagrange multipliers. However, if one recognizes that LP duality is a special case of a more general *inference duality*, the concept of a Benders cut can be generalized [70, 80]. In fact the basic idea of Benders decomposition is best seen in this more general setting and then specialized to the classical case.

Benders decomposition applies to problems of the form

$$\begin{array}{l} \min f(x,y) \\ (x,y) \in S, \;\; x \in D_x, \;\; y \in D_y \end{array} \tag{15.39}$$

Each iteration k of the Benders algorithm begins with a fixed value x^k for x and solves a subproblem for the best y:

$$\begin{array}{l} \min f(x^k,y) \\ (x^k,y) \in S, \;\; y \in D_y \end{array} \tag{15.40}$$

Solution of the subproblem yields an optimal solution y^k and an optimal value $v_k = f(x^k, y^k)$. The solution process is analyzed to identify a proof that $f(x^k, y) \geq v_k$; such a proof can be regarded as solving the inference dual of (15.40). (The inference dual of an LP problem is the classical LP dual.) If x^k is changed to some other value x, this same proof may still show that $f(x, y)$ is bounded below by some function $B_k(x)$. This yields

the Benders cut $z \geq B_k(x)$, where $B_k(x^k) = v_k$ and z is a variable indicating the optimal value of the original problem (15.39).

At this point one solves the master problem, which contains the Benders cuts so far generated.

$$\begin{aligned} &\min z \\ &z \geq B_i(x), \;\; i = 1, \ldots k, \;\; x \in D_x \end{aligned} \tag{15.41}$$

The solution x^{k+1} of (15.41) begins the next iteration. Since the master problem (15.41) is a relaxation of the original problem and the subproblem (15.40) a restriction if it, the optimal value z_{k+1} of the master problem is a lower bound on the optimal value of (15.39), and the optimal value of any subproblem is an upper bound. The algorithm terminates with an optimal solution when the two bounds converge; that is, when $z_{k+1} = \min_{i \in \{1,\ldots,k\}}\{v_i\}$. They converge finitely under fairly weak conditions, for instance if the domain D_x is finite, as it is in examples considered here.

15.11.2 Classical Benders Decomposition

The historical Benders decomposition applies to problems of the form

$$\begin{aligned} &\min f(x) + cy \\ &g(x) + Ay \geq b, \;\; x \in S, \;\; y \geq 0, \;\; x \in D_x, \;\; y \in \Re^n \end{aligned}$$

The subproblem (15.40) is the LP problem

$$\begin{aligned} &\min f(x^k) + cy \\ &Ay \geq b - g(x^k), \;\; y \geq 0, \;\; y \in \Re^n \end{aligned} \tag{15.42}$$

Suppose first that (15.42) has optimal value v^k and an optimal dual solution λ^k. By strong duality $v^k - f(x^k) = \lambda^k(b - g(x^k))$, which means

$$B_k(x) = f(x) + \lambda^k(b - g(x)) \tag{15.43}$$

is the tightest lower bound on cost when $x = x^k$. That is, λ^k specifies a proof of the lower bound v^k by defining a linear combination $\lambda^k Ay \geq \lambda^k(b - g(x^k))$ that dominates $cy \geq v^k - f(x^k)$. But since λ^k remains dual feasible when x^k in (15.42) is replaced by any x, (15.43) remains a lower bound on cost for any x; that is, λ^k specifies a proof of the lower bound $B_k(x)$. This yields the Benders cut

$$z \geq f(x) + \lambda^k(b - g(x)) \tag{15.44}$$

If the dual of (15.42) is unbounded, there is a direction or ray λ^k along which its solution value can increase indefinitely. In this case the Benders cut is $\lambda^k(b - g(x)) \leq 0$ rather than (15.44). The Benders cuts $z \geq B_k(x)$ in the master problem (15.41) therefore take the form (15.44) when the subproblem dual is bounded in iteration k, and the form $\lambda^k(b - g(x)) \leq 0$ when the subproblem dual is unbounded. The master problem can be solved by any desired method, such as branch and bound if it is an MILP problem.

15.11.3 Example: Planning and Scheduling

A basic planning and scheduling problem illustrates the use of nonclassical Benders to combine MILP and CP [71]. Each n jobs must be assigned to one of m facilities for processing. Each job j has processing time p_{ij} and uses c_{ij} units of resource on facility i. Every job has release time 0 and deadline d. Jobs scheduled on any facility i may run simultaneously so long as the total resource consumption at any one time is no greater than C_i (cumulative scheduling). The objective is to minimize makespan (i.e., finish the last job as soon as possible).

If job j has start time s_j on machine y_j, the problem can be written

$$\begin{aligned}&\min z\\ &z \geq s_j + p_{y_j j}, \quad 0 \leq s_j \leq d - p_{y_j j}, \quad \text{all } j\\ &\text{cumulative}\big(s^i(y), p^i(y), c^i(y), C_i\big), \quad \text{all } i\end{aligned}$$

where $s^i(y) = (s_j \mid y_j = i)$ and similarly for $p^i(y), c^i(y)$.

The master problem assigns jobs to facilities and is well suited for MILP solution. The subproblem schedules jobs assigned to each facility and is suitable for a CP approach.

Given an assignment y^k obtained by solving the master problem, the subproblem separates into an independent scheduling problem for each facility i:

$$\begin{aligned}&\min z_i\\ &z_i \geq s_i + p_{ij}, \quad 0 \leq s_j \leq d - p_{ij}, \quad \text{all } j \in J_{ki}\\ &\text{cumulative}(s^i(y^k), p^i(y^k), c^i(y^k), C_i)\end{aligned}$$

where $J_{ki} = \{j \mid y^k_j = i\}$ is the set of jobs assigned to facility i. Let z_{ik} be the optimal makespan obtained in the subproblem P_i for facility i. A Benders cut can be constructed by reasoning as follows.

First let $\hat{P}_i$ be the problem that results when jobs in set R are removed from problem P_i, and let $\hat{z}$ be the optimal makespan for $\hat{P}_i$. Then

$$\hat{z} \geq z_{ik} - \Delta \tag{15.45}$$

where $\Delta = \sum_{j \in R} p_{ij}$. To see this, construct a solution S for P_i by scheduling the jobs in R sequentially after the last job in the optimal solution of $\hat{P}_i$. The resulting makespan is $\hat{z} + \Delta$. If $\hat{z} + \Delta \leq d$, then S is feasible for P_i, so that $z_{ik} \leq \hat{z} + \Delta$ and (15.45) follows. On the other hand, if $\hat{z} + \Delta > d$, then (15.45) follows because $z_{ik} \leq d$.

Since the master problem will be solved by MILP, it is convenient to write the Benders cuts in terms of 0-1 variables x_{ij}, where $x_{ij} = 1$ when $y_j = i$. In subsequent iterations of the Benders algorithm, the jobs in J_{ki} that are removed from facility i are those for which $x_{ij} = 0$. So (15.45) yields the following lower bound on the minimum makespan z for facility i, which is also a lower bound on the minimum makespan z for the problem as a whole:

$$z \geq z_{ik} - \sum_{j \in J_{ki}} p_{ij}(1 - x_{ij}) \tag{15.46}$$

Each Benders cut $z \geq B_k(x)$ in the master problem is therefore actually a set of inequalities (15.46), one for each facility i. The master problem becomes

$$\begin{aligned}
&\min z \\
&\sum_i x_{ij} = 1, \quad \text{all } j && (a) \\
&z \geq z_{i\ell} - \sum_{j \in J_{i\ell}} p_{ij}(1 - x_{ij}), \quad \text{all } i, \ell = 1, \ldots, k && (b) \\
&z \geq \frac{1}{C_i} \sum_{j=1}^{n} c_{ij} p_{ij} x_{ij}, \quad \text{all } i && (c)
\end{aligned} \tag{15.47}$$

The expression (c) is a simple relaxation of the subproblem, which can be important to obtain good computational performance.

The Benders cuts (b) in (15.47) use only the solution of the subproblem and no information regarding the solution process. If additional information is available from the CP solver, one can trace which jobs play no role in proving optimality. These jobs can be removed from the sets $J_{i\ell}$ in the Benders cuts as described in [74], resulting in stronger cuts. The solution of the master problem can be accelerated by updating the solution of the previous master, as proposed in [70] and implemented in [120].

15.11.4 Other Applications

Classical Benders can be applied in a CP context when the subproblem is an LP problem, leaving CP to solve the master problem. This approach was used in [46] to solve minimal perturbation scheduling problems in which the sequencing is decided in the master problem and the assignment of start times in the subproblem. A similar approach was applied to scheduling the workforce in a telephone call center [16].

Most applications, however, have used nonclassical Benders methods in which CP or logic-based techniques solve the subproblem. CP is a natural approach to solving the inference dual of the subproblem, since inference techniques play a major role in CP solvers. Explanations [25, 83, 84, 118] for CP solutions are particularly relevant, since an explanation is in effect a proof of correctness or optimality and therefore solves the inference dual.

Nonclassical Benders was first used to solve circuit verification problems [80], and underlying theory was developed in [70, 78]. Application to CP-based planning and scheduling was proposed in [70] and has been implemented for min-cost planning and disjunctive scheduling [81] (later extended to multistage problems [63]), and planning and cumulative scheduling to minimize cost and makespan [71, 30] as well as tardiness [72]. Similar methods were applied to dispatching of automated guided vehicles [34], steel production scheduling [62], batch scheduling in a chemical plant [92] and polypropylene batch scheduling in particular [122]. CP was used to solve the master problem in a nonclassical Benders approach to real-time scheduling of computer processors [24]. In [131] a traffic diversion problem is solved with a Benders method that has an LP subproblem but generates specialized cuts that are not based on dual variables.

Nonclassical Benders methods for integer programming are studied in [29, 78] and for the propositional satisfiability problem in [70, 78].

15.12 Toward Integration of CP and OR

Importing OR into CP is part of a trend that has been taking shape over the last decade: the integration of OR and CP, or at least portions of them, into a unified approach to problem solving. Several paradigms for integration have emerged.

One general scheme is *double modeling*: some (or all) constraints are formulated in a CP model, some in an MILP model, and some in both. The two models then exchange information during the solution process (e.g., [64, 107, 126]). The relaxation and decomposition strategies discussed here may be seen as special cases in which one of the two models is subservient to the other (see also [37]).

Double modeling, however, is perhaps more a matter of cooperation than unification. Schemes that move closer to full integration point to underlying commonality between CP and OR. Types of commonality include: the role of logical inference in CP and OR [68, 70]; the parallel between CP's filtering algorithms and domain store on the one hand, and MILP's cutting planes and linear constraint store on the one hand [20, 21, 69]; formulability in a single modeling framework based on conditional constraints [76, 77, 116]; analogous roles of duality [79]; and a common search-infer-and-relax algorithmic structure [9, 73]. The ultimate goal is to view OR and CP as special cases of a single methodology.

The evolution of ideas in this area, as well as growing interest in integrated methods, can be traced in the development of such hybrid solvers as ECL ^{i}PS^e, OPL Studio, Mosel, SCIP, and SIMPL. ECL^{i}PS^e [107] is a Prolog-based constraint logic programming system that provides an interface with linear and MILP solvers. The CP solver in ECL ^{i}PS^e communicates tightened bounds to the MILP solver, while the MILP solver detects infeasibility and provides a bound on the objective function that is used by the CP solver. The optimal solution of the linear constraints in the problem can be used as a search heuristic. Recent developments are described in [3].

OPL Studio [65] provides an integrated modeling language that expresses both MILP and CP constraints. It sends the problem to a CP or MILP solver depending on the nature of constraints. A script language allows one to implement cooperation between the CP and MILP solvers.

Mosel [32, 33] is "both a modeling and programming language" that interfaces with various solvers, including MILP and CP solvers. SCIP [1] is a programming language that gives the user "total control" of a solution process that can involve both CP and MILP solvers. SIMPL [9] uses a high-level modeling language in which the choice of constraints and their parameters determine how techniques interact at the micro level.

There will perhaps always be a role for specialized solvers. However, one can also forsee a future in which today's general-purpose CP-based and OR-based solvers evolve into integrated systems in which there is no longer a clear distinction, nor any need to make a distinction, between CP and OR components.

Bibliography

[1] T. Achterberg. SCIP: A framework to integrate constraint and mixed integer programming. ZIB-report, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 2004.

[2] F. Ajili and H. El Sakkout. LP probing for piecewise linear optimization in scheduling. In C. Gervet and M. Wallace, editors, *Proceedings of the International Workshop on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming for Combintaorial Optimization Problems (CPAIOR 2001)*, Ashford, U.K., 2001.

[3] F. Ajili and M. Wallace. Hybrid problem soloving in ECLiPSe. In M. Milano, editor, *Constraint and Integer Programming: Toward a Unified Methodology* , pages 169–206. Kluwer, Dordrecht, 2004.

[4] F. Alizadeh. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM Journal of Optimization*, 5:13–51, 1995.

[5] G. Appa, D. Magos, and I. Mourtos. Linear programming relaxations of multiple all-different predicates. In J. C. Régin and M. Rueher, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2004)*, volume 3011 of *Lecture Notes in Computer Science*, pages 364–369. Springer, 2004.

[6] G. Appa, I. Mourtos, and D. Magos. Integrating constraint and integer programming for the orthogonal Latin squares problem. In P. Van Hentenryck, editor, *Principles and Practice of Constraint Programming (CP2002)*, volume 2470 of *Lecture Notes in Computer Science*, pages 17–32. Springer, 2002.

[7] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Jorunal on Algebraic and Discrete Mathematics*, 8:277–284, 1987.

[8] S. Arnborg and A. Proskurowski. Characterization and recognition of partial k-trees. *SIAM Jorunal on Algebraic and Discrete Mathematics*, 7:305–314, 1986.

[9] I. Aron, J. N. Hooker, and T. H. Yunes. SIMPL: A system for integrating optimization techniques. In J. C. Régin and M. Rueher, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2004)*, volume 3011 of *Lecture Notes in Computer Science*, pages 21–36. Springer, 2004.

[10] E. Balas, A. Bockmayr, N. Pisaruk, and L. Wolsey. On unions and dominants of polytopes. *Mathematical Programming*, 99:223–239, 2004.

[11] F. Barahona and R. Anbil. The volume algorithm: Producing primal solutions with a subgradient algorithm. *Mathematical Programming*, 87:385–399, 2000.

[12] P. Barth. *Logic-based 0-1 Constraint Solving in Constraint Logic Programming* . Kluwer, Dordrecht, 1995.

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

[14] C . Beck and P. Refalo. A hybrid approach to scheduling with earliness and tardiness costs. *Annals of Operations Research*, 118:49–71, 2003.

[15] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

[16] T. Benoist, E. Gaudin, and B. Rottembourg. Constraint programming contribution to Benders decomposition: A case study. In P. Van Hentenryck, editor, *Principles and Practice of Constraint Programming (CP2002)*, volume 2470 of *Lecture Notes in Computer Science*, pages 603–617. Springer, 2002.

[17] T. Benoist, F. Laburthe, and B. Rottembourg. Lagrange relaxation and constraint programming collaborative schemes for traveling tournament problems. In

C. Gervet and M. Wallace, editors, *Proceedings of the International Workshop on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming for Combintaorial Optimization Problems (CPAIOR 2001)* , Ashford, U.K., 2001.

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

[19] D. P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 1 and 2. Athena Scientific, Nashua, NH, 2001.

[20] A. Bockmayr and T. Kasper. Branch-and-infer: A unifying framework for integer and finite domain constraint programming. *INFORMS Journal on Computing*, 10:287–300, 1998.

[21] A. Bockmayr and T. Kasper. Branch-and-infer: A framework for combining CP and IP. In M. Milano, editor, *Constraint and Integer Programming: Toward a Unified Methodology*, pages 59–88. Kluwer, Dordrecht, 2004.

[22] A. Bockmayr and N. Pisaruk. Detecting infeasibility and generating cuts for mixed integer programming using constraint progrmaming. In M. Gendreau, G. Pesant, and L.-M. Rousseau, editors, *Proceedings of the International Workshop on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming for Combintaorial Optimization Problems (CPAIOR 2003)* , Montréal, 2003.

[23] S. Bollapragada, O. Ghattas, and J. N. Hooker. Optimal design of truss structures by mixed logical and linear programming. *Operations Research*, 49:42–51, 2001.

[24] H. Cambazard, P.-E. Hladik, A.-M. Déplanche, N. Jussien, and Y. Trinquet. Decomposition and learning for a hard real time task allocation problem. In M. Wallace, editor, *Principles and Practice of Constraint Programming (CP2004)* , volume 3258 of *Lecture Notes in Computer Science*, pages 153–167. Springer, 2004.

[25] H. Cambazard and N. Jussien. Identifying and exploiting problem structures using explanation-based costraint programming. In R. Barták and M. Milano, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2005)* , volume 3524 of *Lecture Notes in Computer Science*, pages 94–109. Springer, 2005.

[26] A. Chabrier. A cooperative CP and LP optimizer approach for the pairing generation problem. In *Proceedings of the International Workshop on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming for Combintaorial Optimization Problems (CPAIOR 1999)* , Ferrara, Italy, 2000.

[27] A. Chabrier. Heuristic branch-and-price-and-cut to solve a network design problem. In M. Gendreau, G. Pesant, and L.-M. Rousseau, editors, *Proceedings of the International Workshop on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming for Combintaorial Optimization Problems (CPAIOR 2003)*, Montréal, 2003.

[28] D. Chhajed and T. J. Lowe. Solving structured multifacility location problems efficiently. *Transportation Science*, 28:104–115, 1994.

[29] Y. Chu and Q. Xia. Generating benders cuts for a class of integer programming problems. In J. C. Régin and M. Rueher, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2004)*, volume 3011 of *Lecture Notes in Computer Science*, pages 127–141. Springer, 2004.

[30] Y. Chu and Q. Xia. A hybrid algorithm for a class of resource-constrained scheduling problems. In R. Barták and M. Milano, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2005)*, volume 3524 of *Lecture Notes in Computer Science*, pages 110–124. Springer, 2005.

[31] V. Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 4:305–337, 1973.

[32] Y. Colombani and S. Heipcke. Mosel: An extensible environment for modeling and programming solutions. In N. Jussien and F. Laburthe, editors, *Proceedings of the International Workshop on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2002)*, Le Croisic, France, 2002.

[33] Y. Colombani and S. Heipcke. Mosel: An overview. White paper, Dash Optimization, 2004.

[34] A. I. Corréa, A. Langevin, and L. M. Rousseau. Dispatching and conflict-free routing of automated guided vehicles: A hybrid approach combining constraint programming and mixed integer programming. In J. C. Régin and M. Rueher, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2004)*, volume 3011 of *Lecture Notes in Computer Science*, pages 370–378. Springer, 2004.

[35] Y. Crama, P. Hansen, and B. Jaumard. The basic algorithm for pseudoboolean programming revisited. *Discrete Applied Mathematics*, 29:171–185, 1990.

[36] W. Cronholm and Farid Ajili. Strong cost-based filtering for Lagrange decomposition applied to network design. In M. Wallace, editor, *Principles and Practice of Constraint Programming (CP2004)*, volume 3258 of *Lecture Notes in Computer Science*, pages 726–730. Springer, 2004.

[37] E. Danna and Claude Le Pape. Two generic schemes for efficveint and robust cooperative algorithms. In M. Milano, editor, *Constraint and Integer Programming: Toward a Unified Methodology*, pages 33–58. Kluwer, Dordrecht, 2004.

[38] I. de Farias, E. L. Johnson, and G. L. Nemhauser. Branch-and-cut for combinatorial optimization problems without auxilliary variables. In C. Gervet and M. Wallace, editors, *Proceedings of the International Workshop on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming for Combintaorial Optimization Problems (CPAIOR 2001)*, Ashford, U.K., 2001.

[39] R. Dechter. Bucket elimination: A unifying framework for several probabilistic inference algorithms. In *Proceedings of the Twelfth Annual Conference on Uncertainty in Artificial Intelligence (UAI 96)*, pages 211–219, Portland, OR, 1996.

[40] S. Demassey, C. Artiques, and P. Michelon. A hybrid constraint propagation-cutting plane prtocedure for the RCPSP. In N. Jussien and F. Laburthe, editors, *Proceedings of the International Workshop on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2002)*, Le Croisic, France, 2002.

[41] S. Demassey, G. Pesant, and L.-M. Rousseau. Constraint-programming based column generation for employee timetabling. In R. Barták and M. Milano, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2005)*, volume 3524 of *Lecture Notes in Computer Science*, pages 140–154. Springer, 2005.

[42] E. V. Denardo. *Dynamic Programming: Models and Applications*. Dover Publications, Mineola, NY, 2003.

[43] K. Easton, G. Nemhauser, and M. Trick. The traveling tournament problem description and benchmarks. In T. Walsh, editor, *Principles and Practice of Constraint Programming (CP2001)*, volume 2239 of *Lecture Notes in Computer Science*, pages 580–584. Springer, 2001.

[44] K. Easton, G. Nemhauser, and M. Trick. Solving the traveling tournament problem: A combined integer programming and constraint programming approach. In *Proceedings of the International Conference on the Practice and Theory of Automated Timetabling (PATAT 2002)*, 2002.

[45] K. Easton, G. Nemhauser, and M. Trick. CP based branch and price. In M. Milano, editor, *Constraint and Integer Programming: Toward a Unified Methodology* , pages 207–231. Kluwer, Dordrecht, 2004.

[46] A. Eremin and M. Wallace. Hybrid Benders decomposition algorithm in constraint logic programming. In T. Walsh, editor, *Principles and Practice of Constraint Programming (CP2001)*, volume 2239 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2001.

[47] T. Fahle, U. Junker, S. E. Karish, N. Kohn, M. Sellmann, and B. Vaaben. Constraint programming based column generation for crew assignment. *Journal of Heuristics*, 8:59–81, 2002.

[48] T. Fahle and M. Sellmann. Constraint programming based column generation with knapsack subproblems. In *Proceedings of the International Workshop on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming for Combintaorial Optimization Problems (CPAIOR 2000)* , pages 33–44, Paderborn, Germany, 2000.

[49] M. Fischetti, A. Lodi, and P. Toth. Solving real-world atsp instances by branch-and-cut. In M. Jünger, G. Reinelt, and G. Rinaldi, editors, *Combinatorial Optimization—Eureka, You Shrink!, Papers Dedicated to Jack Edmonds*, volume 2570 of *Lecture Notes in Computer Science*, pages 64–77. Springer, 2003.

[50] M. Fischetti and P. Toth. An additive bounding procedure for combinatorial optimization problems. *Operations Research*, 37:319–328, 1989.

[51] F. Focacci, A. Lodi, and M. Milano. Cost-based domain filtering. In J. Jaffar, editor, *Principles and Practice of Constraint Programming (CP1999)* , volume 1713 of *Lecture Notes in Computer Science*, pages 189–203. Springer, 1999.

[52] F. Focacci, A. Lodi, and M. Milano. Cutting planes in constraint programming: An hybrid approach. In R. Dechter, editor, *Principles and Practice of Constraint Programming (CP2000)*, volume 1894 of *Lecture Notes in Computer Science*, pages 187–201. Springer, 2000.

[53] F. Focacci and M. Milano. Connections and integrations of dynamic programming and constraint programming. In C. Gervet and M. Wallace, editors, *Proceedings of the International Workshop on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming for Combintaorial Optimization Problems (CPAIOR 2001)*, Ashford, U.K., 2001.

[54] T. Gellermann, M. Sellmann, and R. Wright. Shorter-path constraints for the resource constrained shortest path problem. In R. Barták and M. Milano, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2005)* , volume 3524 of *Lecture Notes in Computer*

Science, pages 201–216. Springer, 2005.

[55] B. Gendron, H. Lebbah, and G. Pesant. Improving the cooperation between the master problem and the subproblem in constraint programming based column generation. In R. Barták and M. Milano, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2005)*, volume 3524 of *Lecture Notes in Computer Science*, pages 217–227. Springer, 2005.

[56] A. M. Geoffrion. Generalized benders decomposition. *Journal of Optimization Theory and Applications*, 10:237–260, 1972.

[57] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using using semidefinite programming. *Journal of the ACM*, 42:1115–1145, 1995.

[58] C. P. Gomes and D. B. Shmoys. The promise of LP to boost CSP techniques for combinatorial problems. In N. Jussien and F. Laburthe, editors, *Proceedings of the International Workshop on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2002)*, Le Croisic, France, 2002.

[59] M. Grönkvist. Using constraint propagation to accelerate column generation nin aircraft scheduling. In M. Gendreau, G. Pesant, and L.-M. Rousseau, editors, *Proceedings of the International Workshop on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming for Combintaorial Optimization Problems (CPAIOR 2003)*, Montréal, 2003.

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

[61] G. Gutin and A. P. Punnen, editors. *The Traveling Salesman Problem and Its Variations*. Kluwer, Dordrecht, 2002.

[62] I. Harjunkoski and I. E. Grossmann. A decomposition approach for the scheduling of a steel plant production. *Computers and Chemical Engineering*, 25:1647–1660, 2001.

[63] I. Harjunkoski and I. E. Grossmann. Decomposition techniques for multistage scheduling problems using mixed-integer and constraint programming methods. *Computers and Chemical Engineering*, 26:1533–1552, 2002.

[64] S. Heipcke. Integrating constraint programming techniques into mathematical programming. In H. Prade, editor, *Proceedings, 13th European Conference on Artificial Intelligence*, pages 259–260. Wiley, New York, 1999.

[65] P. Van Hentenryck, I. Lustig, L. Michel, and J. F. Puget. *The OPL Optimization Programming Language*. MIT Press, Cambridge, MA, 1999.

[66] W. J. Van Hoeve. A hybrid constraint programming and semidefinite programming approach for the stable set problem. In F. Rossi, editor, *Principles and Practice of Constraint Programming (CP2003)*, volume 2833 of *Lecture Notes in Computer Science*, pages 407–421. Springer, 2003.

[67] J. N. Hooker. Generalized resolution for 0-1 linear inequalities. *Annals of Mathematics and Artificial Intelligence*, 6:271–286, 1992.

[68] J. N. Hooker. Logic-based methods for optimization. In A. Borning, editor, *Principles and Practice of Constraint Programming (CP2002)*, volume 874 of *Lecture Notes in Computer Science*, pages 336–349. Springer, 1994.

[69] J. N. Hooker. Constraint satisfaction methods for generating valid cuts. In D. L. Woodruff, editor, *Advances in Computational and Stochastic Optimization, Logic Programming and Heuristic Search*, pages 1–30. Kluwer, Dordrecht, 1997.

[70] J. N. Hooker. *Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction*. Wiley, New York, 2000.

[71] J. N. Hooker. A hybrid method for planning and scheduling. In M. Wallace, editor, *Principles and Practice of Constraint Programming (CP2004)*, volume 3258 of *Lecture Notes in Computer Science*, pages 305–316. Springer, 2004.

[72] J. N. Hooker. Planning and scheduling to minimize tardiness. In *Principles and Practice of Constraint Programming (CP2005)*, volume 3709 of *Lecture Notes in Computer Science*, pages 314–327. Springer, 2005.

[73] J. N. Hooker. A sesarch-infer-and-relax framework for integrating solution methods. In R. Barták and M. Milano, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2005)*, volume 3524 of *Lecture Notes in Computer Science*, pages 243–257. Springer, 2005.

[74] J. N. Hooker. A hybrid method for planning and scheduling. *Constraints*, to appear.

[75] J. N. Hooker. *Integrated Methods for Optimization*. To appear.

[76] J. N. Hooker, H.-J. Kim, and G. Ottosson. A declarative modeling framework that integrates solution methods. *Annals of Operations Research*, 104:141–161, 2001.

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

[78] J. N. Hooker and G. Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 96:33–60, 2003.

[79] J. N. Hooker, G. Ottosson, E. S. Thornsteinsson, and H.-J. Kim. A scheme for unifying optimization and constraint satisfaction methods. *Knowledge Engineering Review*, 15:11–30, 2000.

[80] J. N. Hooker and H. Yan. Logic circuit verification by benders decomposition. In V. Saraswat and P. Van Hentenryck, editors, *Principles and Practice of Constraint Programming: The Newport Papers*, pages 267–288, Cambridge, MA, 1995. MIT Press.

[81] V. Jain and I. E. Grossmann. Algorithms for hybrid MILP/CP models for a class of optimization problems. *INFORMS Journal on Computing*, 13:258–276, 2001.

[82] U. Junker, S. E. Karish, N. Kohl, B. Vaaben, T. Fahle, and M. Sellmann. A framework for constraint programming based column generation. In J. Jaffar, editor, *Principles and Practice of Constraint Programming (CP1999)*, volume 1713 of *Lecture Notes in Computer Science*, pages 261–275. Springer, 1999.

[83] N. Jussien. The versatility of using explanations within constraint programming. Research report, École des Mines de Nantes, France, 2003.

[84] N. Jussien and S. Ouis. User-friendly explanations for constraint programming. In *Eleventh Workshop on Logic Programming environments (WLPE 2001)*, Paphos, Cyprus, 2001.

[85] M. O. Khemmoudj, H. Bennaceur, and A. Nagih. Combining arc consistency and dual Lagrangean relaxation for filtering csps. In R. Barták and M. Milano, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2005)*, volume 3524 of *Lecture Notes in Computer Science*, pages 258–272. Springer, 2005.

[86] H.-J. Kim and J. N. Hooker. Solving fixed-charge network flow problems with a

hybrid optimization and constraint programming approach. *Annals of Operations Research*, 115:95–124, 2002.

[87] N. Kohl. Application of or and cp techniques in a real world crew scheduling system. In *Proceedings of the International Workshop on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming for Combintaorial Optimization Problems (CPAIOR 2000)* , Paderborn, Germany, 2000.

[88] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society B*, 50:157–224, 1988.

[89] S. Lee and I. Grossmann. Generalized disjunctive programming: Nonlinear convex hull relaxation and algorithms. *Computational Optimization and Applications* , 26:83–100, 2003.

[90] S. Lee and I. E. Grossmann. Global optimization of nonlinear generalized disjunctive programming with bilinear equality constraints: Applications to process networks. *Computers and Chemical Engineering* , 27:1557–1575, 2003.

[91] A. Lodi and M. Milano. Discrepancy-based additive bounding. In M. Gendreau, G. Pesant, and L.-M. Rousseau, editors, *Proceedings of the International Workshop on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming for Combintaorial Optimization Problems (CPAIOR 2003)* , Montréal, 2003.

[92] C. T. Maravelias and I. E. Grossmann. Using MILP and CP for the scheduling of batch chemical processes. In J. C. Régin and M. Rueher, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2004)*, volume 3011 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2004.

[93] H. Marchand, A. Martin, R. Weismantel, and L. Wolsey. Cutting planes in integer and mixed integer programming. *Discrete Applied Mathematics*, 123:397–446, 2002.

[94] H. Marchand and L. A. Wolsey. Aghgregation and mixed integer rounding to solve mips. *Operations Research*, 49:363–371, 2001.

[95] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. Wiley, New York, 1990.

[96] M. Milano and W. J. van Hoeve. Reduced cost-based ranking for generating promising subproblems. In P. Van Hentenryck, editor, *Principles and Practice of Constraint Programming (CP2002)*, volume 2470 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2002.

[97] D. Naddef. Polyhedral theory and branch-and-cut algorithms for the symmetric TSP. In G. Gutin and A. P. Punnen, editors, *The Traveling Salesman Problem and Its Variations*, pages 29–116. Kluwer, Dordrecht, 2002.

[98] A. Nedic and D. P. Bertsekas. Incremental subgradient methods for nondifferentiable pptimization. *SIAM Journal on Optimization* , 12:109–138, 2001.

[99] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization* . Wiley, New York, 1999.

[100] M. Osorio and F. Glover. Logic cuts using surrogate constraint analysis in the multidimensional knapsack problem. In C. Gervet and M. Wallace, editors, *Proceedings of the International Workshop on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming for Combintaorial Optimiza-*

tion Problems (CPAIOR 2001), Ashford, U.K., 2001.

[101] G. Ottosson, E. Thorsteinsson, and J. N. Hooker. Mixed global constraints and inference in hybrid IP-CLP solvers. In *Proceedings of CP99 Post-Conference Workshop on Large-Scale Combinatorial Optimization and Constraints* , http://www.dash.co.uk/wscp99, pages 57–78, 1999.

[102] G. Ottosson, E. Thorsteinsson, and J. N. Hooker. Mixed global constraints and inference in hybrid CLP-IP solvers. *Annals of Mathematics and Artificial Intelligence* , 34:271–290, 2002.

[103] M. Padberg and G. Rinaldi. An efficient algorithm for the minimum capacity cut problem. *Mathematical Programming*, 47:19–36, 1990.

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

[105] P. Refalo. Tight cooperation and its application in piecewise linear optimization. In J. Jaffar, editor, *Principles and Practice of Constraint Programming (CP1999)* , volume 1713 of *Lecture Notes in Computer Science*, pages 375–389. Springer, 1999.

[106] P. Refalo. Linear formulation of constraint programming models and hybrid solvers. In R. Dechter, editor, *Principles and Practice of Constraint Programming (CP2000)*, volume 1894 of *Lecture Notes in Computer Science*, pages 369–383. Springer, 2000.

[107] R. Rodošek, M. Wallace, and M. Hajian. A new approach to integrating mixed integer programming and constraint logic programming. *Annals of Operations Research*, 86:63–87, 1997.

[108] L.-M. Rousseau. Stabilization issues for constraint programming based column generation. In J. C. Régin and M. Rueher, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2004)*, volume 3011 of *Lecture Notes in Computer Science*, pages 402–408. Springer, 2004.

[109] L. M. Rousseau, M. Gendreau, and G. Pesant. Solving small VRPTWs with constraint programming based column generation. In N. Jussien and F. Laburthe, editors, *Proceedings of the International Workshop on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming for Combintaorial Optimization Problems (CPAIOR 2002)* , Le Croisic, France, 2002.

[110] R. Sadykov. A hybrid branch-and-cut algorithm for the one-machine scheduling problem. In J. C. Régin and M. Rueher, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2004)*, volume 3011 of *Lecture Notes in Computer Science*, pages 409–415. Springer, 2004.

[111] N. W. Sawaya and I. E. Grossmann. A cutting plane method for solving linear generalized disjunctive programming problems. Research report, Department of Chemical Engineering, Carnegie Mellon University, 2004.

[112] N. W. Sawaya and I. E. Grossmann. Computational implementation of non-linear convex hull reformulations. Research report, Department of Chemical Engineering, Carnegie Mellon University, 2005.

[113] M. Sellmann and T. Fahle. Constraint programming based Lagrangian relaxation for a multimedia application. In C. Gervet and M. Wallace, editors, *Proceedings of the International Workshop on Integration of Artificial Intelligence and Operations*

Research Techniques in Constraint Programming for Combintaorial Optimization Problems (CPAIOR 2001), Ashford, U.K., 2001.

[114] M. Sellmann, K. Zervoudakis, P. Stamatopoulos, and T. Fahle. Crew assignment via constraint programming: Integrating column generation and heuristic tree search. *Annals of Operations Research*, 115:207–225, 2002.

[115] G. Shafer, P. P. Shenoy, and K. Mellouli. Propagating belief functions in qualitative markov trees. *International Journal of Approximate Reasoning*, 1:349–400, 1987.

[116] H. M. Sheini and K. A. Sakallah. A SAT-based decision procedure for mixed logical/integer linear problems. In R. Barták and M. Milano, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2005)*, volume 3524 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 2005.

[117] P. P. Shenoy and G. Shafer. Propagating belief functions with local computation. *IEEE Expert*, 1:43–52, 1986.

[118] M. H. Sqalli and E. C. Freuder. Inference-based constraint satisfaction supports explanation. In *National Conference on Artificial Intelligence (AAAI 1996)*, pages 318–325, 1996.

[119] R. Stubbs and S. Mehrotra. A branch-and-cut method for 0-1 mixed convex programming. *Mathematical Programming*, 86:515–532, 1999.

[120] E. Thorsteinsson. Branch and check: A hybrid framework integrating mixed integer programming and constraint logic programming. In T. Walsh, editor, *Principles and Practice of Constraint Programming (CP2001)*, volume 2239 of *Lecture Notes in Computer Science*, pages 16–30. Springer, 2001.

[121] E. Thorsteinsson and G. Ottosson. Linear relaxations and reduced-cost based propagation of continuous variable subscripts. *Annals of Operations Research*, 115:15–29, 2001.

[122] C. Timpe. Solving planning and scheduling problems with combined integer and constraint programming. *OR Spectrum*, 24:431–448, 2002.

[123] M. Trick. A dynamic programming approach for consistency and propagation for knapsack constraints. In C. Gervet and M. Wallace, editors, *Proceedings, Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2001)*, pages 113–124, Ashford, U.K., 2001.

[124] M. Trick. Formulations and reformulations in integer programming. In R. Barták and M. Milano, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2005)*, volume 3524 of *Lecture Notes in Computer Science*, pages 366–379. Springer, 2005.

[125] A. Vecchietti, S. Lee, and I. E. Grossmann. Characterization and formulation of disjunctions and their relaxations. In *Proceedings of Mercosul Congress on Process Systems Engineering (ENPROMER 2001)*, volume 1, pages 409–414, Santa Fe, Chile, 2001.

[126] M. Wallace, M. S. Novello, and J. Schimpf. ECLiPSe: A platform for constraint logic programming. *ICL Systems Journal*, 12:159–200, 1997.

[127] H. P. Williams. *Model Building in Mathematical Programming, 4th Ed.* Wiley, New York, 1999.

[128] H. P. Williams and H. Yan. Representations of the all_different predicate of constraint satisfaction in integer programming. *INFORMS Journal on Computing*, 13:96–103, 2001.

[129] H. Wolkowicz, R. Saigal, and L. Vandenberghe, editors. *Handbook of Semidefinite Programming*. Kluwer, Dordrecht, 2000.

[130] L. A. Wolsey. *Integer Programming*. Wiley, New York, 1998.

[131] Q. Xia, A. Eremin, and M. Wallace. Problem decomposition for traffic diversions. In J. C. Régin and M. Rueher, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2004)* , volume 3011 of *Lecture Notes in Computer Science*, pages 348–363. Springer, 2004.

[132] H. Yan and J. N. Hooker. Tight representations of logical constraints as cardinality rules. *Mathematical Programming*, 85:363–377, 1995.

[133] T. H. Yunes, A. V. Moura, and C. C. de Souza. Hybrid column generation approaches for urban transit crew management problems. *Transportation Science*, to appear.

Index