The Role of Integer Programming Techniques in Constraint-Programming's Global Constraints

Michela Milano • Greger Ottosson
Philippe Refalo • Erlendur S. Thorsteinsson

DEIS – Università di Bologna, V.le Risorgimento, 2, 40136 Bologna, Italy

Computing Science Department, Uppsala University, PO Box 311, S-751 05 Uppsala, Sweden

ILOG S.A., Les Taissounieres, 1681, route des Dolines, 06560 Sophia Antipolis, France

Graduate School of Industrial Administration, Carnegie Mellon University, Schenley Park, Pittsburgh, PA 15213, USA

mmilano@deis.unibo.it • greger@acm.org
refalo@ilog.fr • esth@cmu.edu

Efforts aimed at combining operations research and constraint programming have become increasingly prominent and successful in the last few years. It is now widely recognised that integration, e.g., inference in the form of constraint propagation and relaxation in the form of linear programming, can yield substantial results. In this paper, we argue the benefits of constraint programming's global constraints as a basis for such an integration and discuss the advantages along with some examples. We illustrate the integration on the global cardinality structure, on piecewise linear functions, on variable subscripts, on the cycle structure and on resource constraints. Each example is completed with a case study. (*Integer Programming; Constraint Programming; Global Constraints; Integration; Hybrid Methods*)

1 Introduction

Since its origins, constraint programming (CP) has integrated algorithms from different areas (Jaffar and Maher 1994) such as mathematical programming, networks, and computational logic. Earliest constraint logic programming systems like `Prolog III` (Colmerauer 1990), `CHIP` (Dincbas et al. 1988) and `CLP(`$\mathcal{R}$`)` (Jaffar et al. 1992) solve constraints in different domains using constraint-solving algorithms such as the simplex method for solving linear constraints (Refalo and Van Hentenryck 1996), boolean unification or resolution for logical constraints (Colmerauer 1990), constraint-propagation techniques for solving constraints on finite domain variables (Van Hentenryck 1989) and many others ranging from quadratic programming to interval-analysis techniques (Benhamou

and Older 1997). In this paper, we focus on finite domain constraint solving, where variables, describing problem entities, range on a finite domain of objects (say integers) and are subject to constraints.

One of the most successful integrations is the combination of the models and methods of operations research (OR), in particular those of integer programming (IP), with the models and methods of finite-domain constraint programming, including constraint propagation. The initial cultural barriers have largely disappeared and the gains deriving from the integration of inference in the form of constraint propagation, and, e.g., continuous relaxations through linear programming (LP), are now widely acknowledged (Bockmayr and Kasper 1998, Hooker 2000, Rodosek et al. 1999).

A fundamental modelling capability of CP systems are global constraints that capture interesting substructures of a problem. Global constraints serve both declaratively as building blocks of the problem statement, and operationally as software components. They encapsulate dedicated inference algorithms based on *feasibility* reasoning. Global constraints achieve *domain reduction*: they infer which values are infeasible for a variable w.r.t. the structure they represent. Industrial implementations, e.g., CHIP (Beldiceanu and Contejean 1994), ECL^{i}PS^e (IC-Parc 2001), and ILOG Solver (ILOG 2000) have shown the effectiveness of global constraints for solving practical problems in different areas, such as rostering, scheduling, resource allocation, and configuration (Baptiste et al. 2001, Cheng et al. 1996, Régin 2001). However, reasoning mainly on feasibility is a weakness of CP solvers when dealing with an objective function. Most of the CP systems do not use any relaxation and do not provide any lower bounds. In fact, CP systems implement a simple form of branch and bound: each time a solution is found, a new constraint (called the *bounding constraint*) is posted, stating that further solutions should be better than the last one found. When the link between problem decision variables and the objective function is loose, then the bounding constraint prunes only a limited portion of the search tree, thus affecting performance.

On the other hand, IP techniques apply to a set of linear constraints. IP in general tackles *optimality* reasoning better than CP. In IP branch-and-bound approaches, the continuous (linear) relaxation of the original problem is defined and solved to optimality. This solution provides a bound on the optimal solution of the original problem, which enables pruning sub-optimal parts of the search tree and guiding the search towards promising regions. Clearly, the closer the bound to the optimal solution, the larger the search tree pruned. It is thus important to have tight linear relaxations of the problem. Efforts were made to study polyhedral structures of subproblems, ranging from the 0–1 knapsack structure and graph-based structures to the travelling-salesman structure. Exploiting the problem structure enables IP solvers to efficiently infer valid inequalities (cutting planes) in order to tighten the linear formulation (Wolsey 1998).

Due to their similarities and complementarities, techniques from IP are increasingly being applied within the CP framework, and global constraints have often been considered as a suitable basis for a close integration. The purpose

of this paper is to summarise and discuss the role of IP techniques within CP global constraints. Examples of this integration are numerous. Among them we survey in this paper:

- The integration of specialised graph algorithms, such as matchings (Focacci et al. 1999) and network-flow algorithms (Régin, 1999) for reducing variable domains in `alldifferent` and global cardinality constraints;
- The use of global constraints for optimality reasoning; values are removed from domains if proven sub-optimal, e.g., using reduced-cost-based propagation (Focacci et al. 1999, Thorsteinsson and Ottosson 2001);
- The use of relaxations in addition to constraint-propagation techniques, such as linear relaxations (Ottosson et al. 2002, Refalo 1999, 2000a, Rodosek et al. 1999, Thorsteinsson and Ottosson 2001) or Lagrangian relaxations (Caseau and Laburthe 1997a, Focacci et al. 2000b) to provide tight bounds to the constraint-programming solvers;
- Embedding cutting-plane algorithms in global constraints, such as polyhedral cuts (Bockmayr and Kasper 1998, Focacci et al. 2000b, Refalo 2000b, Thorsteinsson and Ottosson 2001) or local cuts derived from the constraint propagation (Ottosson et al. 2002, Refalo 1999).

To facilitate this discussion we begin with a short overview of the basic concepts of CP and IP in Section 2 and motivate the integration of CP and IP in Section 3. We then describe and illustrate the integration of CP and IP techniques on the global cardinality structure (Section 4), on piecewise linear functions (Section 5), on variable subscripts (Section 6), on the cycle structure (Section 7) and on resource constraints (Section 8). Finally, Section 9 concludes the paper.

2 Background

In this section, we provide some basic ideas about IP and CP. Since the readership of this journal comes predominantly from the mathematical-programming community, we focus mainly on the CP side, giving particular attention to global constraints.

2.1 Integer-Programming Basic Concepts

In IP, the linear relaxation is often used as part of the solution process. This very restricted form of problem representation also permits formalisation and modelling of high levels problems. IP uses 0–1 variables extensively. They specify the available decision choices (often exhaustively) and the associated constraints encode, in linear form, which decisions can be made simultaneously. The decisions are then implicitly enforced through branching on the values of the

0–1 variables. Modelling and solving integer programs is extensively described in textbooks such as Nemhauser and Wolsey (1988) and Wolsey (1998).

The lack of high-level modelling constructs in IP for disjunctions and combinatorial constraints has some drawbacks. First, the modelling process is harder and more error-prone. Second, and more importantly, much of the problem structure is lost before the problem reaches the solver algorithms. To obtain stronger linear formulation (e.g., by cutting-plane generation), the solver has to *recover* the structure of the problem, despite the fact that the structure was already known to the modeller, or *assume* what the structure of the model was.

Recent work in IP attempts to solve these limitations. It involves enhancing the basic modelling with linear constraints by allowing special structures to be defined in the modelling language. For instance this is what is done in Belvaux and Wolsey (1996) for lot-sizing problems. Closer to the modelling in CP, a recent work by Bockmayr and Kasper (1998) addressed this issue by showing how to model IP through the composition of global structures (high-level constraints) that preserve the structure of the problem or part of it. They introduce global `tsp` and `assign` constraints exploiting various results from polyhedral theory. Declaratively, these constraints correspond to sets of linear inequalities, while operationally they embed cutting-plane generators deriving problem-specific inequalities. Also, in Kasper (1998) a nonlinear global constraint is defined that operationally derives linear inequalities that are efficiently handled by the solver.

The advantages of using global structures are clearly due to the increased expressive power of the resulting language, more readable and concise models maintaining the problem structure, and the increased efficiency if problem-specific constraints are introduced. In addition and an important point: different constraint abstractions can be used together in one model when the problem is not pure but consists of a composition of several (possibly overlapping) subproblems.

2.2 Constraint-Programming Basic Concepts

CP is a declarative programming paradigm that has had more influence from computer science and programming-language design than from mathematics. This has affected not only solution strategies, but also the interface and modelling practices. We focus here on finite-domain CP, henceforth referred to as CP(FD). Problems are modelled by defining a set of variables representing problem entities, taking their values from a finite domain of objects of arbitrary type (generally integers), and linked by a set of constraints. The model description and the solution algorithm (embedded in the solver) are kept separate. In this setting, "constraint programming represents one of the closest approaches computer science has made to the Holy Grail of programming: the user states the problem, the computer solves it" (Freuder 1996).

2.2.1 Constraint Propagation

Constraints in CP(FD) are stored in the *constraint store* and can be divided into *passive* and *active* constraints (Jaffar and Maher 1994). Passive constraints are unary constraints making the variables take their values from finite domains of values, while active constraints are defined on domain variables and embed constraint-propagation algorithms that are triggered each time a variable domain changes.

The constraint solver is based on two interleaved steps: constraint propagation and search for solutions. Constraint propagation is aimed at removing assignments from variable domains that certainly lead to a failure. In general, it is an incomplete procedure, in the sense that values left in the domains are not always proven to be consistent. For this reason, search is needed to explore the solution space. Domain reduction and constraint propagation can be compared to pre-processing and probing techniques proposed by Savelsberg (1994). The main difference is that in CP, they are performed at each step of the search process, while in general in the IP setting they are performed only at the root node to get a simpler and stronger linear formulation.

The basic domain-reduction technique used in finite-domain CP is called *arc consistency* (Mackworth 1977). A constraint problem is arc-consistent if and only if for each pair of variables (say X and Y) linked by a constraint, and for each value in the domain of variable X, there is a consistent value in the domain of Y, and vice-versa. Bound consistency is simpler than arc consistency. It acts only on bounds and it is not affected by value-domain reduction if this value is not a bound of the domain. Arc and bound consistency are not sufficiently powerful for real applications and hardly applicable to non-binary constraints (see Tsang 1993 for a survey). For these reasons, global constraints embedding smart domain-reduction algorithms have been introduced in CP.

Constraint propagation is an iterative process: domain reductions performed by a single constraint are communicated to other constraints through events connected to domains of shared variables. Events triggering the propagation could be the instantiation of a variable, bound reduction, or domain reduction. Propagation reaches a fixed point when no domain can be pruned any more.

2.2.2 Global Constraints

Global constraints are relations among several variables representing suitable abstractions that enable a declarative statement of the problem and an operational behaviour matching the best available domain-reduction techniques. A typical example is the global constraint

$$\texttt{alldifferent}([x_1, \ldots, x_n]),$$

which constrain a set of variables to be assigned different values (Régin 1994). It is declaratively equivalent to a conjunction of simpler constraints

$$x_i \neq x_j \quad \text{for } i, j \in \{1, \ldots, n\},\ i < j.$$

Another typical example is a resource constraint in scheduling models. Assume that n tasks are required to be processed on a machine, and that this machine can process only one task at a time. The global constraint that states this requirement is

$$\texttt{resource1}([x_1, \ldots, x_n], [d_1, \ldots, d_n]),$$

where each x_i is a variable representing the starting date of task i and d_i is its duration. This constraint is declaratively equivalent to a conjunction of simpler constraints

$$(x_i >= x_j + d_j) \vee (x_j >= x_i + d_i) \quad \text{for } i, j \in \{1, \ldots, n\},\ i < j.$$

The constraints `alldifferent` and `resource1` are said to be *global* in the sense that they work on a set of variables (not only two) and they are declaratively equivalent to a conjunction of simpler constraints. From an operational point of view, however, they are much more powerful since they reason globally.

Global constraints can be seen as CP *software components*, from both a problem-modelling and solving perspective. A CP model, in fact, decomposes the problem into sub-structures, each of which is handled separately by a global constraint. The effect of the domain reduction of individual constraints is communicated to other constraints through the *constraint store*, which contains the domains of shared variables. This communication process is called *constraint propagation*. Constraint propagation is performed until a fixed point is reached, i.e., when the variable domains cannot be reduced any further by any constraint.

Global constraints encapsulate specialised domain-reduction techniques. To illustrate it, consider the constraint system

$$\begin{cases} \texttt{alldifferent}([x_1, x_2, x_3, x_4]) \\ x_1 + x_3 = 4 \end{cases}$$

and the constraint store

$$\{\ x_1 \in \{1,2\}, x_2 \in \{1,2\}, x_3 \in \{1,2,3\}, x_4 \in \{1,2,3,4,5\}\ \}.$$

Domain reduction on the `alldifferent` constraint can achieve a strong form of consistency. In fact, it removes all infeasible values from the domain of the variables. Consequently, the new store is

$$\{\ x_1 \in \{1,2\}, x_2 \in \{1,2\}, x_3 \in \{3\}, x_4 \in \{4,5\}\ \}.$$

Propagation then triggers the constraint $x_1 + x_3 = 4$ and removes the value 2 from the domain of x_1. The constraint store becomes

$$\{\ x_1 \in \{1\}, x_2 \in \{1,2\}, x_3 \in \{3\}, x_4 \in \{4,5\}\ \}.$$

The `alldifferent` constraint is triggered again and a fixed point is reached:

$$\{\ x_1 \in \{1\}, x_2 \in \{2\}, x_3 \in \{3\}, x_4 \in \{4,5\}\ \}.$$

In many cases, domain-reduction techniques have their origin in results from OR and discrete mathematics. For the `alldifferent` constraint, the problem of pruning all inconsistent values from the domains of the variables can be recast as the problem of finding all maximal bipartite matchings in the corresponding value graph. A graph-theory result of Berge (1970) allows this to be done in a less naïve and more efficient manner by using a flow algorithm and calculating strongly connected components (Régin 1994).

For the `resource1` constraint, the problem of removing all inconsistent values is NP-complete. Therefore, a weaker consistency is maintained: the domain-reduction algorithm maintains the tightest bounds on each starting date of a task assuming that the other tasks are interruptible (Baptiste et al. 2001). Several implementations of domain reduction for this constraint have been investigated (Baptiste et al. 1995b, Caseau and Laburthe 1994, Nuijten and Aarts 1996). Most of them are extensions of the *edge-finding* technique for scheduling that originated in the work of Carlier and Pinson (1990, 1994).

Global-constraints designers often try to solve the most general structure. Despite this, a large range of practical applications has triggered the invention of quite a number of global constraints. Among them, we can cite the sort constraint (Guernalec and Colmerauer 1997) and the permutation constraint (Zhou 1997). The global cardinality constraint (Régin 1996) derives its properties and bases its propagation on graph theory and graph-based algorithms. In fact, the consistency of a global cardinality constraint is based on the existence of a maximum flow in a network associated with the constraint. Also, filtering of inconsistent values is decided by computing strongly connected components in the corresponding residual graph, i.e., a graph where each arc is labelled with the difference between the capacity of the arc and the flow on it. There is also a global constraint combining a linear sum and a set of linear inequalities (Régin and Rueher 2000). The propagation in this constraint intensively uses the shortest-path algorithm, previously proposed by Dechter et al. (1991) to cope with temporal distance constraints. Another interesting use of global constraints is to solve the subproblem in a column-generation application. This promising approach is described in Junker et al. (1999) where a special path constraint is used to generate groups of columns for an airline application.

The list above is not exhaustive. Several global constraints have been defined for different applications. A classification of many global constraints based on their graph properties has been proposed in Beldiceanu (2000).

A drawback of global constraints appears when trying to solve a problem whose structure or substructures do not fit any global constraints available in the CP system. This is why most of those systems give facilities to the user to program the domain-reduction algorithms of their own global constraints.

2.2.3 Object Oriented Modelling and Solving

It is interesting to examine the parallels between modelling and solving using global constraints, and object oriented software design and development (OOSDD).

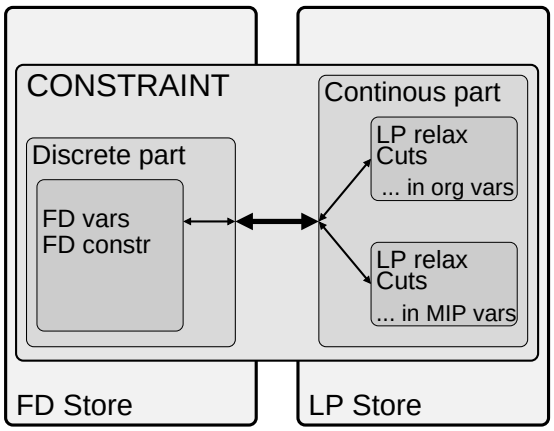

Figure 1: The Structure of Global Constraints

OOSDD builds on the principle that each class captures specific functionality in the program and has built-in methods to operate on its data. Defining a specific but reusable class is a balance between focusing on this particular functionality and using its special features for designing efficient data structures and algorithms, and at the same time being general enough so that the class can be reused in a different context.

Similarly, a global constraint captures a specific substructure in the problem, e.g., an all-different requirement on a set of variables, a piecewise linear function, a variable subscripted coefficient or variable, and allows that substructure to be passed around and operated on as a whole, e.g., propagation and dynamically stated linear relaxation. It should be specific enough so that an efficient propagation can be implemented and a tight linear relaxation can be derived. On the other hand, it should also be general enough so that it can be used in a different problem where a similar structure appears.

Given these parallels, the methodology of OOSDD can be applied to a large degree to the process of modelling and solving of combinatorial optimisation problems, as it relates to breaking the problem structurally into coherent pieces and providing filtering algorithms for each one, hence the term *object oriented modelling and solving (OOMS)*. The correspondence between the two can also provide a roadmap to how a modeller/solver system can be implemented in software (Michel and Van Hentenryck 2001, Puget and Leconte 1995, Thorsteinsson 2001), both in terms of how a mixed global constraint breaks down into individual parts and how they communicate (Figure 1), and in terms of the interface between global constraints and the rest of the system (Figure 2).

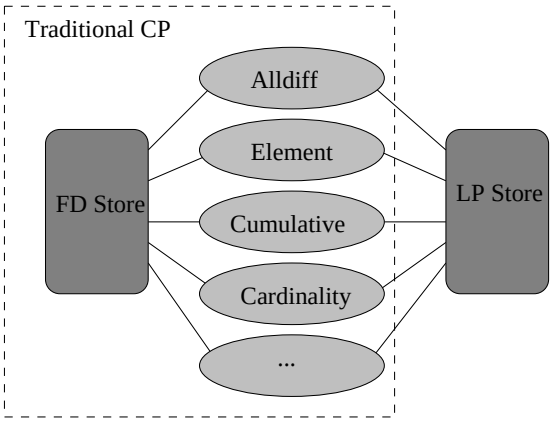

Figure 2: The Interactions of Global Constraints

3 Motivations for the Integration

Modelling with global constraints leads to smaller and more expressive models whose variables and constraints have a closer mapping to the problem at hand, compared to modelling solely using equations. The global constraints simplify the task of modelling and maintenance. As described above, global constraints also enable effective domain reduction on the basis of *feasibility reasoning*. It is important to note that domain reduction is performed locally on a constraint. In fact, constraints communicate only through the constraint store and ignore each other. Moreover, global constraints do no take into account the objective function for domain reduction and therefore lack *optimality reasoning*.

IP solvers, in contrast, perform *optimality reasoning* on a global linear relaxation of the problem. The objective function can then be taken into account very early in the search process.

Thus, the first benefit coming from the integration of IP and CP is to provide a CP solver with relaxations. Relaxations can be embedded in or derived from global constraints and structured problems. In this way, relaxation and inference can be combined and applied to the same structures. In particular, this allows constraint propagation and linear programming to meet, despite the difference in the models they traditionally are applied to. Global constraints are a suitable mean of integration also because they behave in an incremental and dynamic way; relaxations can be dynamically updated as domains change during search, allowing a tighter formulation to be maintained.

Furthermore, relaxations of overlapping global constraints give a second and strong channel for communication, as depicted in Figure 2. In CP, constraints communicate only through the *constraint store*, called the FD Store in Figure 2. A global relaxation (e.g. a linear program) adds a separate means of interaction. Inference on one constraint will here strengthen the LP relaxation, which

therefore also strengthens the relaxations of other global constraints since their relaxations are also a part of the same LP.

Using a relaxation can also serve an operational purpose: values are removed from domains if proven sub-optimal (Focacci et al. 1999, Thorsteinsson and Ottosson 2001).

In what follows, we illustrate these type of integrations by studying a series of cases where integration efforts based on global constraints have shown the benefits of combining relaxations and constraint propagation.

4 The Global Cardinality Structure

4.1 The `alldifferent` and `gcc` Global Constraints

The simplest matching structure arises in the global constraint

$$\texttt{alldifferent}([x_1, \ldots, x_n])$$

that requires the variables x_i to be assigned all different values. A generalisation of the `alldifferent` constraint is the global cardinality constraint

$$\texttt{gcc}([x_1, \ldots, x_n], [v_1, \ldots, v_k], [q_1, \ldots, q_k])$$

that requires the number of variables from the set $[x_1, \ldots, x_n]$ that have the value v_i to be equal to the variable q_i. It is easy to see that when the values v_i are from the union of domains of variables x_i and the variables q_i have $\{0, 1\}$ as domains, then this constraint is equivalent to an `alldifferent` constraint.

These structures are very common and occur as basic building blocks of many real-life applications like assignment, scheduling, and timetabling problems.

By considering the bipartite graph whose first set of nodes is indexed by variables and whose second set of nodes is indexed by the values of variables domains, and by creating an arc between each variable and the nodes indexed by a value of its domain, finding a solution to the `alldifferent` constraint amounts to finding a maximum matching in that graph. This has been exploited to develop efficient and incremental algorithms for enforcing arc consistency of both structures (Régin 1994, Régin 1996). These algorithms are based on results coming from graph-flow theory, well known in OR, and can be considered as a successful integration of these fields.

The example below illustrates the use of a global cardinality constraint `gcc` on a car-sequencing problem. The emphasis in this problem is not to find an optimal solution since there is no objective function, but just to find a feasible solution. Large instances of this problem are frequent in industry and CP has been successfully used in solving it.

The car-sequencing problem. *Cars in production are placed on an assembly line that moves through production units responsible for installing options such as special paint, radios, etc. There are n classes of cars to produce. The assembly line is composed of ns slots and each car is allocated to a single slot. A*

type of car t is characterised by a quantity q_t to produce. There are m possible options. Options cannot be installed on all cars of a production line; there are setup times that create capacity constraints. An option o can be installed only on the set of type of cars T_o and only on r_o out of p_o cars. The car-sequencing problem amounts to finding an assignment of cars to slots that satisfies the capacity constraints. For modelling this problem, we introduce ns variables, one for each slot, that are assigned to the type of car to produce:

$$S_i \text{ in } \{1,\ldots,n\}, \text{ for } i \in \{1,\ldots,ns\}$$

Let m be the number of options; we also introduce $n \times m$ variables

$$Q_i^o \text{ in } \{0,1\}, \text{ for } i \in \{1,\ldots,ns\}, o \in \{1,\ldots,m\}$$

indicating whether the option o is installed on the slot i. To constrain the quantities required, we state the constraint

$$\texttt{gcc}([S_1,\ldots,S_{ns}],[1,\ldots,n],[q_1,\ldots,q_n])$$

To model the capacity constraints, we state the following set of linear constraints for each option o:

$$Q_i^o + \cdots + Q_{i+r_o}^o \leq p_o \text{ for } i \in \{1,\ldots,ns-r_o\}$$

To link the variables S_i and Q_i^o, the following higher-order constraints are stated:

$$Q_i^o = \sum_{t \in T_o} (S_i = t) \text{ for } i \in \{1,\ldots,ns\}, o \in \{1,\ldots,m\}$$

that force Q_i^o to be 1 if the type of car in slot i requires option o (that is, if $S_i \in T_o$) and 0 otherwise.

4.2 Handling an Objective Function

In optimisation problems, besides a global cardinality constraint

$$\texttt{gcc}([x_1,\ldots,x_n],[v_1,\ldots,v_k],[q_1,\ldots,q_k])$$

we have to consider a cost c_{ij} for assigning a variable x_i to a value v_j. The goal is to find solutions to the cardinality constraint that minimise the overall cost. By introducing a binary variable y_{ij} for each couple (x_i, v_j), an optimal solution of this problem can be found by solving the IP problem

$$\begin{aligned}
\min \quad & \sum_{i\in\{1,\ldots,n\}} \sum_{j\in\{1,\ldots,k\}} c_{ij} y_{ij} \\
\text{s.t.} \quad & \sum_{j\in\{1,\ldots,k\}} y_{ij} = 1, \quad \text{for } i \in \{1,\ldots,n\}, \\
& \sum_{i\in\{1,\ldots,n\}} y_{ij} = q_j, \quad \text{for } j \in \{1,\ldots,k\}, \\
& y_{ij} \in \{0,1\}, \quad \text{for } i \in \{1,\ldots,n\},\ j \in \{1,\ldots,k\}.
\end{aligned}$$

The matrix of this formulation is totally unimodular; thus its optimal continuous solution is integral. This formulation can therefore be solved by LP techniques. In the special case where $n = k$ and the `gcc` constraint reduces to an `alldifferent` constraint, this problem is an assignment problem (AP). Consequently, it can be solved in polynomial time by specialised incremental algorithms, e.g., the Busaker and Gowen flow algorithm or the Hungarian algorithm (Carpaneto et al. 1988).

Caseau and Laburthe (1997b) first exploited this special case when embedding lower bounds in global constraints. They also defined an approximative regret function that was used as information for search heuristics.

Focacci et al. (1999) have extended the idea of regret with that of reduced costs by incorporating the AP structure within the `alldifferent` and `path/cycle` constraints. A linear formulation of the AP is embedded in the global constraint and solved to optimality while computing the corresponding reduced costs. The assignment $y_{ij} = 1$ corresponds to the CP variable instantiation $x_i = v_j$ in `alldifferent`$([x_1, \ldots, x_k])$. The reduced-cost propagation rule is that if $c\bar{y} + \bar{c}_{ij} > cy^*$ then $x_i \neq v_j$ (for a non-basic variable y_{ij}, incumbent solution y^*, current solution $\bar{y}$, and its reduced costs $\bar{c}$). This rule is used within a standard fixed-point propagation loop and can thus interact and communicate with other constraints. From a CP perspective, this is a new way of effectively using the objective function for domain reduction in specific global constraints/structures. On the other hand, from an OR point of view, reduced-cost fixing is given added value through its integration with other inference algorithms. Here the inference algorithms for the `alldifferent` constraint work in combination with a relaxation to reduce the search space.

Régin (1999) has proposed an algorithm that maintains arc-consistency for a global cardinality constraint and a cost function. The algorithm is based on the computation of minimum-cost flow in the value network with cost less than or equal to a given upper bound. For filtering purposes a second graph is built, called a *residual graph*. The nodes are the same as for the value graph. Each arc is labelled with the difference between its capacity and the flow on it. Moreover, each arc has an associated cost corresponding to its reduced cost. Filtering of value a for variable x is based on the computation of the shortest path in the residual graphs between a and x.

Finally, Refalo (2000b) has used the linear formulation above in to automatically provide linear formulations for CP models containing `alldifferent` and `gcc` constraints for using LP relaxations together with constraint propagation. This approach handles relaxations of different global constraints in the same linear program. The solution method, a simplex algorithm, is not dedicated to that particular structure but since it has a global view, it gives more accurate optimal information such as reduced costs and dual values. The relaxed optimal solution given by the simplex algorithm guides the search toward optimal solutions.

5 Piecewise-Linear Functions

Piecewise-linear functions are well studied in the literature and widely used in industrial applications. A function is piecewise-linear when its values lie on linear segments, as illustrated in Figure 3. Optimising a problem having constraints $y = f(x)$ where f is piecewise-linear is called *piecewise-linear optimisation*. A problem having arbitrary piecewise-linear functions is generally equivalent to an IP. The following example describes a transportation problem having piecewise-linear costs.

A transportation problem. *A company must transport boats from n factories to m shops. In other words it must ship ships. Each factory can supply at most q_i ships and each shop needs at least d_i ships. There is a transportation cost for shipping a ship from a factory to a shop, i.e.:*

- *120 for quantities between 0 and 200,*
- *80 for quantities between 201 and 400,*
- *50 for quantities greater than 401.*

The objective is to minimise the total shipping cost while satisfying the demands and respecting the supply quantities. To model this problem, we introduce $n \times m$ positive variables x_{ij} representing the quantity to ship from factory i to shop j. There are also $n \times m$ positive variables y_{ij} representing the cost for shipping from factory i to shop j. A CP model of this problem is thus

$$
\begin{array}{lll}
\min & \displaystyle\sum_{i\in\{1,\ldots,n\},\, j\in\{1,\ldots.m\}} y_{ij} & \\
\textit{s.t.} & \textbf{\textit{piecewise-linear}}(x_{ij,}[0,0], & \\
& \quad [120 \to 200, 80 \to 400, 50], y_{ij}), & \textit{for } i \in \{1,\ldots,n\}, j \in \{1,\ldots,m\}, \\
& \displaystyle\sum_{j\in\{1,\ldots,m\}} x_{ij} \le p_i, & \textit{for } i \in \{1,\ldots,n\}, \\
& \displaystyle\sum_{i\in\{1,\ldots,n\}} x_{ij} \ge d_j, & \textit{for } j \in \{1,\ldots,m\}.
\end{array}
$$

The global constraint $\textbf{\textit{piecewise-linear}}(x, [a,b], [p_1 \to s_1, \ldots, p_k \to s_k, s_{k+1}], y)$ *defines that $y = f(x)$ where f is a piecewise-linear function whose slope is s_i between values p_i and p_{i+1} on the x-axis and $x = a, y = b$ is a solution to the constraint. This is sufficient to define the shape of the function fully.*

An equivalent IP formulation of a piecewise-linear optimisation problem can be obtained by reformulating each constraint $y = f(x)$ into

$$
\begin{aligned}
& x = \lambda_1 b_1 + \cdots + \lambda_k b_k, && (1)\\
& y = \lambda_1 f(b_1) + \cdots + \lambda_k f(b_k), && (2)\\
& \lambda_1 + \cdots + \lambda_k = 1, && (3)\\
& \lambda_i \ge 0, \quad i = 1,\ldots,k, && (4)\\
& \text{SOS–2}(\lambda_1,\ldots,\lambda_k), && (5)
\end{aligned}
$$

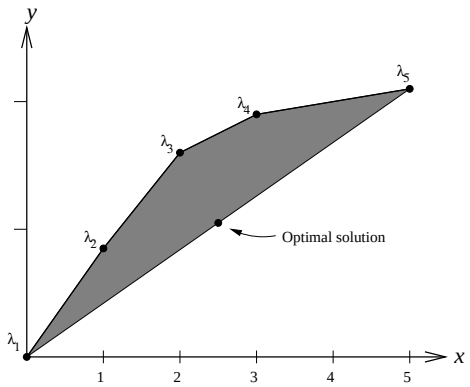

Figure 3: Piecewise-Linear Function Illustrating the SOS–2 Formulation

where b_i are the x-coordinates at which the slope of the function changes. The linear constraints in this formulation express that the solution set contains solutions that are convex combinations of the extreme points of the functions, i.e., of points $(b_i, f(b_i))$ by way of positive λ_i variables. Thus the linear constraints (1)–(4) represent the convex hull of the solution set. The SOS–2 requirement (5) forces the number of λ_i variables that are different from zero to be at most two and also adjacent in the set. This constraint is usually handled with specialised branching in IP solvers (cf. Beale and Forrest 1976), not with constraint propagation, as is the case with cardinality constraints in CP.

Although the goal of the SOS–2 requirement is to provide structural information to the IP model, traditional MIP solvers consider all these constraints independently. As a consequence, they lose the global structure of piecewise-linear functions and they do not update the linear formulation correctly during the search.

Recently, Ottosson et al. (2002) and Refalo (1999) have shown how to preserve the piecewise structure as a global constraint that maintains a tight formulation during the search.

In Ottosson et al. (2002) the constraint is given a linear convex hull formulation in the original space, i.e., the two-dimensional space (x, y) given by the two axes, and thus no new variables need to be introduced. If one discrete indicator variable is also introduced, then the segments can be semi-continuous. Besides an improved relaxation, which is dynamically updated and tightened during the search as a part of the constraint propagation, the global piecewise constraint can also provide information for better search strategies. The black dot on the cut in Figure 3 denotes the LP-values of x, y in a typical situation where the LP relaxation has been solved and the minimisation has identified the optimal solution. A natural branching strategy is now to focus the search around the indicated segment, e.g, by splitting the domain of x into two parts, or into three parts, forcing the solution up onto the piecewise function. This is obvious and easy to calculate for a solver that treats this structure as a global constraint

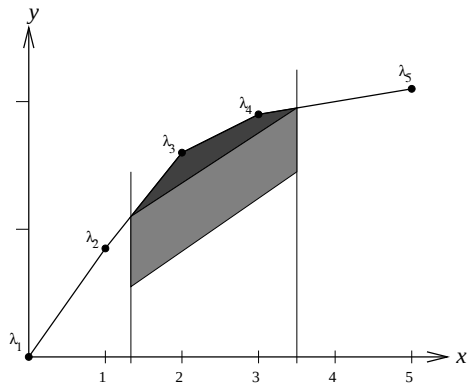

Figure 4: Piecewise-Linear Functions Illustrating How Constraint Propagation on Bounds Can Give a Tighter Relaxation (Dark Shaded Area)

In Refalo (1999) the interaction between constraint propagation and linear formulation is investigated. A tight cooperation scheme is introduced that for each constraint $y = f(x)$ maintains the convex hull of the formulation, taking into account the domain reduction performed during the search. To maintain this convex hull, cutting planes are added in the two-dimensional space (x, y). The benefits of this approach are illustrated in Figure 4: assume the bounds of x have been tightened. Then a tighter formulation (shown in dark) can be constructed by intersecting these new bounds with the solution set of $y = f(x)$. On some transportation problems, like the one given above, this cutting-plane generation can outperform IP approaches. Note that these cuts are only locally valid since they depend on domain reductions performed on x and y at a particular node of the search tree. Consequently, they must be removed when backtracking, similarly to standard constraint propagation.

6 The Variable-Subscripts Structure

6.1 Simple Variable Subscripts

The global constraint

$$\texttt{element}(y, [c_1, \ldots, c_k], z)$$

is ubiquitous in many optimisation problems. It constrains the variable z to be equal to the value c_y. It is commonly used to look up a cost associated with a discrete decision variable. Bound-consistency on the `element` constraint involves producing increasingly smaller domains or tighter bounds on z as the domain D_y of y is reduced. A straightforward linearisation of this constraint is identical to how this structure (i.e., a disjunction) is modelled in IP (Refalo 2000b, Thorsteinsson and Ottosson 2001). We introduce decision variables $b_1, \ldots, b_k$ for $D_y = \{1, \ldots, k\}$, where b_i is 1 if $y = i$ and 0 otherwise, and we

have the equivalence

$$\texttt{element}(y, [c_1, \ldots, c_k], z) \Leftrightarrow \begin{cases} z = c_1 b_1 + \cdots + c_k b_k, \\ b_1 + \cdots + b_k = 1, \\ b_i \in \{0, 1\} \end{cases}$$

This linear relaxation is no stronger than bound consistency on the `element` constraint (actually equivalent to it). Thus there is little incentive to use this higher-dimensional relaxation unless these new variables $(b_1, \ldots, b_k)$ connect in a beneficial way to some other linear constraints, and useful information (dual values, reduced costs, etc.) can be derived and used.

We believe there are many problems where bounds-consistency techniques can eliminate the introduction of 0–1 variables, and effectively reduce the LP size. Furthermore, in this case, branching on y implicitly performs a search that is similar to SOS-style branching in IP, without the need for the modeller to specify this explicitly.

6.2 Quantified Variable Subscripts

A variant of the variable-subscripted constraint discussed in the previous section was investigated by Thorsteinsson and Ottosson (2001). Occurring in a configuration problem, a type and quantity are to be decided for each component, which results in a term of the form $c_y x = z$. Here, y indicates the type of the component, x its quantity, and z the resulting cost.

The configuration problem. *Many industrial products come in different configurations, aiming to satisfy the needs of individual customers closely. In one class of such problems the product is made up of individual components, each one of a set of possible types, and the aim is to find a feasible configuration with a type and quantity for each component that optimises some criteria. A computer is an example of such a product.*

Components supply or consume attributes/resources (such as weight, cost, or effect), and these resources are constrained, the resource *constraints, or are a part of the objective. We will assume that all quantities are integral and that in addition there is a set of logical side-constraints, the* configuration *constraints.*

In general we are given a set of components C, a set of possible component types T_i for each component i, and a set of attributes R. Let variable t_i be the type of component i, variable q_i be the quantity of component i, and variable r_k be the quantity of attribute/resource k. It is convenient to have r_k as a separate variable although its value is determined by the values of t_i and q_i; see (6)*. The weight/cost per unit of attribute/resource k is denoted by c_k. The parameter R_k is the minimum amount of attribute/resource k produced/consumed and $A_{k,i,j}$ defines how many units of attribute/resource k are produced/consumed by each component i if it is of type j.*

Let $q = (q_i)_{i \in C}$ and $t = (t_i)_{i \in C}$. Then the problem can be written

$$\begin{aligned}
\min \quad & \sum_{k \in R} c_k r_k \\
s.t. \quad & r_k = \sum_{i \in C} q_i A_{kit_i}, && \forall k \in R, && (6) \\
& h_l(q,t), && \forall l \in L, && (7) \\
& r_k \geq R_k, && \forall k \in R, && (8) \\
& t_i \in T_i,\ q_i \in \mathbb{Z}_+, && \forall i \in C.
\end{aligned}$$

Note that the resource-consumption "lookup" is handled in (6) *through a variable subscript on A, i.e., A_{k,i,t_i} where the* variable t_i *is the type of component i. This term, $q_i \times A_{k,i,t_i}$, is the core of the problem. Equation* (7) *states the set of configuration constraints, e.g., $q_1 > 0 \Rightarrow q_2 = 0$, $\mathtt{alldiff}(t_1, \ldots, t_n)$, or some type of layout constraints.*

A linear relaxation of the quantified variable subscript

$$c_y x = \mathtt{element}(y, [c_1, \ldots, c_k] \times x, z)$$

is achieved by disaggregating x into bins x_i, and linking to the corresponding decision variable z:

$$\begin{aligned}
& 0 \leq b_i \leq 1, && \forall i, && (9) \\
& b_1 + \cdots + b_k = 1, && && (10) \\
& x = x_1 + \cdots + x_k, && && (11) \\
& z = c_1 x_1 + \cdots + c_k x_k, && && (12) \\
& x_i \geq 0, && \forall i, && (13) \\
& x_i \leq M b_i, && \forall i. && (14)
\end{aligned}$$

Equations (9)–(10) are the same as for c_y above (the plain `element` constraint), but in addition, x is disaggregated to $x_1, \ldots, x_k$, and connected through the big-M constraints (14) to $b_1, \ldots, b_k$.

This formulation is not the smallest convex-hull relaxation possible. The purpose of the variables $b_1, \ldots, b_k$ and the big-M constraints (14) is only to ensure that at most one of $x_1, \ldots, x_k$ is non-zero. This is unnecessary since this is implicitly enforced by the connection to y. Thus, we can dispose of (9), (10), and (14), given that we enforce $x_i = 0$ upon domain reduction $i \notin D_y$ of y. This is similar to the modelling and branching with SOS–1 variables in IP, but again, it is implicit within this global constraint. Note that the resulting relaxation (11)–(13) is the convex hull formulation of the disjunction (Balas 1979) underlying the `element` constraint.

Computational tests (Thorsteinsson and Ottosson 2001) show that this tight relaxation is an invaluable tool if combined with constraint propagation. Furthermore, reduced-cost-based propagation (Thorsteinsson and Ottosson 2001) can be applied to the variables $x_1, \ldots, x_k$, achieving domain reduction on y. A

hybrid approach using this relaxation and reduced-cost-based propagation along with regular CP propagation is shown to outperform both pure CP and IP, and is able to solve instances larger than the other two methods can handle alone.

The two variants of the `element` constraint described in this and the previous section illustrate three of the benefits of global constraints: *smaller relaxations*, *improved search*, and *improved propagation*, e.g., by using the reduced costs.

7 The Cycle Structure

The cycle structure is expressed in constraint programs through the global constraint

$$\texttt{cycle}(k, [x_1, \ldots, x_n])$$

where k is a positive integer and x_i are variables having domains $D_i \subset \{1, \ldots, n\}$. This constraint is satisfied when the values of the variables define k disjoint circuits in a directed graph such that each node is visited exactly once. For instance, if we have $x_1 \in \{2, 3\}$, $x_2 \in \{1, 3\}$, $x_3 \in \{2, 3\}$, the constraint cycle$(1, [x_1, x_2, x_3])$ is satisfied by the assignment $x_1 = 3$, $x_2 = 1$, $x_3 = 2$.

This constraint is used in practice to model transportation problems. The following example shows how to model a travelling salesman problem (TSP) with multiple travellers with the constraint `cycle`.

The multiple travelling salesman problem. *Given a set of n visits to be done once by k travelling salesmen, and a distance c_{ij} between each visit, the problem is to find which travelling salesmen visit which city so as to minimise the overall distance travelled. This problem is modelled in the following way: for each visit i we introduce a variable x_i in $\{1, \ldots, n\}$ that is assigned to the visit to do next, and a variable y_i assigned to the cost of the travel from visit i. Thus the problem can be modelled by means of cycle and element constraints in the following way:*

$$\begin{aligned}
\min \quad & z = \sum_{i=0}^{n} y_i \\
s.t. \quad & x_i \in \{1, \ldots, n\}, \quad \textit{for } i \in \{1, \ldots, n\}, \\
& y_i \in \{c_{i1}, \ldots, c_{in}\}, \quad \textit{for } i \in \{1, \ldots, n\}, \\
& \texttt{element}(x_i, [c_{i1}, , c_{in}], y_i), \quad \textit{for } i \in \{1, \ldots, n\}, \\
& \texttt{cycle}(k, [x_1, \ldots, x_n]).
\end{aligned}$$

Note that in (Focacci et al. 1999) the above model is embedded in a single (extended) cycle constraint whose parameters, beside k and $[x_1, \ldots, x_n]$, are a cost matrix $CostMtx$ and an objective function Z.

A simple relaxation of the `cycle` constraint is the `alldifferent`$([x_1, \ldots, x_n])$ constraint. Indeed, the linear formulation of the `alldifferent` constraint is used in IP codes for solving the one-tour travelling salesman problem (Padberg

and Rinaldi 1990). Of course, this formulation is tightened by several classes of cutting planes ranging from subtour elimination to comb inequalities.

In CP the `alldifferent` constraint has been used as a relaxation for a cycle constraint in a similar way to the one described in Section 4.

A first improvement to pure CP techniques for solving the one-tour TSP problems was given by Caseau and Laburthe (1997a), where a propagation scheme for the `cycle` constraint was proposed together with branching strategies and bounding methods that further improved the performances of CP systems.

In Focacci et al. (1999) the use of reduced costs has been shown to be even more effective. In fact, experimental results show that the information on the lower bound derived from solving an assignment problem (AP) to optimality enables one to solve small TSPs that are not solvable without this information. In addition, the use of reduced-cost-based propagation leads to a 10–20-fold average reduction in the number of failures, with respect to using the lower bounds only. For pure TSPs, despite the performance improvement achieved with reduced-cost-based propagation, the state-of-the-art OR branch-and-cut algorithms are still superior. When TSP variants are considered, however, the techniques described above obtain results that are comparable to state-of-the-art OR approaches (Ascheuer et al. 2000). An improvement to this work has been obtained in Focacci et al. (2000b) by adding a set of subtour-elimination cuts (Padberg and Rinaldi 1990) to the AP formulation, aimed at strengthening the linear formulation of the AP. The resulting problem is again solved to optimality and used for domain pruning based on reduced costs and improved bounding. As before, the relaxation combines with the propagation of the `cycle` constraint (Caseau and Laburthe 1997a) within the global constraint.

In Refalo (2000b) a linear formulation for the k-cycle constraint is proposed. It is based on traditional cut-set and subtour-elimination inequalities (Wolsey 1998). The formulation of the problem above is used to automatically produce an IP problem that is solved by branch-and-cut. The search strategy is described in terms of the original CP model using the values of variables given by the LP relaxation. Choice points are set by stating either $x_i = j$ or $x_i \neq j$. At each node of the search tree, the global constraints generate cutting planes that are part of the formulation, as long as they are violated. This simple approach was able to solve problems having up to a few hundred visits.

8 Resource Constraints

Many real-life applications have to deal with limited resource capacity. One of the most successful examples of integration of OR techniques in CP is the use of *edge-finding* techniques first presented by Carlier and Pinson (1995), which have been embedded in CP global resource constraints (Baptiste et al. 1995a). Analogous techniques have been presented in Beldiceanu and Contejean (1994) for the `cumulative` constraint. The algebraic description of the `resource1` constraint was given in Section 2.2.2.

We now restrict our presentation to a single machine constraint, where a set

of activities should be scheduled on a single-capacity machine. Let us consider a set of activities Ω. Each activity $a \in \Omega$ is associated with two variables $Start(a)$ and $End(a)$ ranging in $[est(a), \ldots, lst(a)]$ and $[ect(a), \ldots, lct(a)]$ respectively, where $est(a)$ and $lst(a)$ are the earliest and latest start time of a respectively, and $ect(a)$ and $lct(a)$ are the earliest and latest completion time of a respectively. The duration of activity a is represented as $d(a) = End(a) - Start(a)$. The edge-finding technique is based on the following theorem.

Theorem *Assume a single-capacity machine m, $a \in \Omega$ and $S \subset \Omega$, such that $a \notin S$ and all activities in $S \cup \{a\}$ are to be scheduled on m. If*

$$est(S \cup \{a\}) + d(S \cup \{a\}) > lct(S),$$

then $est(a)$ can be set to $\max_{S' \subseteq S}\{est(S') + pt(S')\}$.

The theorem is justified by reasoning on the total sum of processing times required on machine m by the set of activities $S \cup \{a\}$. Thus, it is possible to update $est(a)$ and $lct(a)$ for all $a \in \Omega$ in $O(nlog(n))$ time.

These theorems are valid on problems both with and without sequence-dependent setup times. Brucker and Thiele (1996) extend the previous theorem for cases where setup times are defined. The formula in the above theorem can be changed as follows:

$$est(S \cup \{a\}) + pt(S \cup \{a\}) + Setup(S \cup \{a\}) > lct(S),$$

where $Setup(S \cup \{a\})$ is the minimal total setup time to schedule all activities in S followed by activity a.

However, the computation of $Setup(S)$ is an NP-hard problem since it is equivalent to an asymmetric travelling salesman problem. Thus, all the methods discussed for the `cycle` structure in Section 7 can be applied to compute a bound on $Setup(S)$, and encapsulated in a global constraint for resource capacity when a setup matrix is defined, as proposed in Focacci (2000) and described in the following example.

Scheduling with sequence-dependent setup times. *(Focacci et al. 2000a) We are given a set of n activities $A_1, ..., A_n$ and a set of m unary resources (resources with maximal capacity equal to one) $R_1, ..., R_m$. Each activity A_i has to be processed on a resource R_j for $duration_i$ time units starting at time $start_i$ and ending at time end_i. Sequence-dependent setup times exist among activities. Given a setup-time matrix S^k (square matrix of dimension n), s^k_{ij} represents the setup time between activities A_i and A_j if A_i and A_j are scheduled sequentially on the same resource R_k. In such a case, $start_j \geq end_i + s^k_{ij}$. Activities may be linked by precedence relations $A_i \rightarrow A_j$. In this case activity A_j cannot start before the end of activity A_i. The goal is to minimise the sum of setup times.*

In the model we propose, for each activity we have variables representing the start, the end, and an additional variable $next_i$ representing the activity following A_i on the same resource. The schedule horizon is limited by the End parameter. Clearly, the array of variables start, end, and next should be linked by a constraint `link`$(start, end, next)$ *that ensures the propagation in both ways (from next to start and end, and vice versa). The set of precedence constraints*

are taken into account by the symbolic constraint $\texttt{precedence}(start, end)$ *that declaratively corresponds to the set of precedences imposed by the problem. In addition, for each resource, besides the resource constraint shown in this section, we have a setup constraint ensuring the respect of setup times between subsequent activities on the same resource. We have grouped variables corresponding to activities on the same resource with a superscript* k*, i.e.,* $start^k$*.*

$$
\begin{aligned}
\min \quad & z = \sum_{i=0}^{n} setup_i \\
s.t. \quad & start_i \in \{1, \ldots, End - duration_i\}, \quad \textit{for } i \in \{1, \ldots, n\}, \\
& end_i \in \{duration_i, \ldots, End\}, \quad \textit{for } i \in \{1, \ldots, n\}, \\
& next_i \in \{1, \ldots, n\}, \quad \textit{for } i \in \{1, \ldots, n\}, \\
& \texttt{link}(start, end, next) \\
& s_i \in \{s_{i1}, \ldots, s_{in}\}, \quad \textit{for } i \in \{1, \ldots, n\}, \\
& \texttt{element}(next_i, [s_{i1}, ..., s_{in}], setup_i), \quad \textit{for } i \in \{1, \ldots, n\}, \\
& \texttt{cycle}(m, [next_1, \ldots, next_n]) \\
& \texttt{precedence}(start, end) \\
& \texttt{setup}(start^k, end^k, setup^k) \quad \textit{for } k \in \{1, \ldots, m\}, \\
& \texttt{resource1}(start^k, duration^k) \quad \textit{for } k \in \{1, \ldots, m\}.
\end{aligned}
$$

Note that the cycle constraint in the model allows one to use a relaxation (as discussed in Section 7) on the sum of the setup times.

9 Conclusion

Modelling and solving are tightly integrated activities, and it is no surprise that the different underlying solution technologies of CP and IP have shaped the modelling traditions into different forms. When the two techniques now are being merged on the technology level, we also have to re-evaluate our modelling practices. So far, most efforts of integration have been done within the CP community, which also traditionally has had the more flexible framework.

As part of this, global constraints play a very important role. They preserve structure, allowing improved inference and search to be performed more effectively. We believe this also holds true when adding relaxations, and this is illustrated on several examples. Not only can OR methods be incorporated within these constraint abstractions, but in several cases the relaxations can be made sharper and more effective than in IP solvers operating only on linear formulations. Add to this the combination of constraint propagation and relaxations that enhance each other, and we have a clear indication that global constraints are worth further study and application in hybrid CP–IP contexts.

References

Ascheuer, N., M. Fischetti, M. Grötschel. 2000. A polyhedral study of the asymmetric travelling salesman problem with time windows. *Networks* **36** 69–79.

Balas, E. 1979. Disjunctive programming. P. L. Hammer, E. L. Johnson, B. H. Korte, eds. *Discrete Optimization II, 5, Annals of Discrete Mathematics.* North-Holland, Amsterdam, The Netherlands. 3–51.

Baptiste, P., C. L. Pape, W. Nuijten. 1995a. Efficient operations research algorithms in constraint-based scheduling. *Proceedings of IJCAI'95, Montreal, Quebec, Canada.*

Baptiste, P., C. L. Pape, W. Nuijten. 1995b. Incorporating efficient operations research algorithms in constraint-based scheduling. *Proceedings of 1st Joint Workshop on Artificial Intelligence and Operational Research, Timberline Lodge, Oregon, USA.*

Baptiste, P., C. L. Pape, W. Nuijten. 2001. *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems.* Kluwer Academic Publishers, Boston, Massachusetts, USA.

Beale, E., and J. Forrest. 1976. Global optimization using special ordered sets. *Mathematical Programming* **10** 52–69.

Beldiceanu, N. 2000. Global constraints as graph properties on a structured network of elementary constraints of the same type. *Proceedings of the* 6^{th} *International Conference CP 2000, Singapore, September 2000*, Springer-Verlag, Heidelberg, Germany. 52–66.

Beldiceanu, N., E. Contejean. 1994. Introducing global constraints in CHIP. *Mathematical and Computer Modelling* **20** 97–123.

Belvaux, G., L. Wolsey. 1996. Lot-sizing problems: modelling issues and a specialized branch-and-cut system bc-prod. Technical Report CORE Discussion Paper 9849, Center for Operations Research and Econometrics, Universite Catholique de Louvain, Louvain, Belgium.

Benhamou, F., W. Older. 1997. Applying interval arithmetic to real, integer and boolean constraints. *Journal of Logic Programming* **32** 1–24.

Berge, C. 1970. *Graphes et hypergraphes.* Dunod, Paris, France.

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

Brucker, P., O. Thiele. 1996. A branch and bound method for the general shop problem with sequence dependent setup-times. *OR Spektrum* **18** 145–161.

Carlier, J., E. Pinson. 1990. A practical use of Jackson's preemptive schedule for solving the job-shop problem. *Annals of Operations Research* **26** 269–287.

Carlier, J., E. Pinson. 1994. Adjustment of heads and tails for the job-shop problem. *European Journal of Operational Research* **78** 146–161.

Carlier, J., E. Pinson. 1995. An algorithm for solving job shop scheduling. *Management Science* **35** 164–176.

Carpaneto, G., S. Martello, P. Toth. 1988. Algorithms and codes for the assignment problem. B. Simeone, P. Toth, G. Gallo, F. Maffioli, S. Pallottino, eds. *Fortran Codes for Network Optimization, Annals of Operations Research* **13** 193–223.

Caseau, Y., F. Laburthe. 1994. Improved CLP scheduling with task intervals. P. Van Hentenryck, ed. *Proceedings of the Eleventh International Conference on Logic Programming*, MIT Press, Cambridge, Massachusetts, USA. 369–383.

Caseau, Y., F. Laburthe. 1997a. Solving small TSPs with constraints. L. Naish, ed. *Proceedings of the 14th International Conference on Logic Programming*, MIT Press, Cambridge, Massachusetts, USA. 316–330.

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

Cheng, B., J. Lee, J. Wu. 1996. A constraint-based nurse rostering system using a redundant modeling approach. *Eighth IEEE International Conference on Tools with Artificial Intelligence, Toulouse, France, November 1996*, IEEE Computer Society Press. 140–148.

Colmerauer, A. 1990. An introduction to Prolog III. *Communications of the ACM* **33** 69–91.

Dechter, R., I. Meiri, J. Pearl. 1991. Temporal constraint networks. *Artificial Intelligence* **49** 61–95.

Dincbas, M., P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf, F. Berthier. 1988. The Constraint Logic Programming Language CHIP. *FGCS-88: Proceedings International Conference on Fifth Generation Computer Systems*, ICOT, Tokyo, Japan. 693–702.

Focacci, F. 2000. *Solving Combinatorial Optimization Problems in Constraint Programming.* PhD thesis, University of Ferrara, Ferrara, Italy.

Focacci, F., P. Laborie, W. Nuijten. 2000a. Solving scheduling problems with setup times and alternative resources. *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling, AIPS'00*, AAAI Press, Menlo Park, California, USA. 92–111.

Focacci, F., A. Lodi, M. Milano. 1999. Cost-based domain filtering. J. Jaffar, ed. *Proceedings of the Fifth International Conference on Principles and Practice of Constraint Programming, CP'99, Lecture Notes in Computer Science* **1713** 189–203.

Focacci, F., A. Lodi, M. Milano. 2000b. Cutting planes in constraint programming: a hybrid approach. *Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming, CP'00, Lecture Notes in Computer Science* **1894** 187–201.

Freuder, E. C. 1996. In pursuit of the holy grail. *ACM Computing Surveys* **28** 63.

Guernalec, M. B., A. Colmerauer. 1997. Narrowing a $2n$-block of sorting in $o(n\log n)$. *Proceedings of* 3^{rd} *International Conference CP'97, Linz, Austria.* 2–16.

Hooker, J. N. 2000. *Logic-Based Methods for Optimization.* Wiley, New York City, New York, USA.

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

Hooker, J. N., G. Ottosson, E. S. Thorsteinsson, H.-J. Kim. 1999. On integrating constraint propagation and linear programming for combinatorial optimization. *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, The AAAI Press/The MIT Press, Menlo Park, California, USA. 136–141.

Hooker, J. N., G. Ottosson, E. S. Thorsteinsson, H.-J. Kim. 2000. A scheme for unifying optimization and constraint satisfaction methods. *Knowledge Engineering Review, Special Issue on Artifical Intelligence and Operations Research* **15** 11–30.

IC-Parc. 2001. *ECL^{i}PS^e User Manual Release 5.3.* IC-Parc, Imperial College, London, UK.

ILOG. 2000. *Ilog Solver 5.0 User Manual.* ILOG SA, Gentilly, France.

Jaffar, J., M. Maher. 1994. Constraint logic programming: a survey. *Journal of Logic Programming* **19-20** 503–582.

Jaffar, J., S. Michaylov, P. Stuckey, R. Yap. 1992. The CLP($\mathcal{R}$) language and system. *ACM Transactions on Programming Languages and Systems* **14**.

Junker, U., S. E. Karisch, N. Kohl, B. Vaaben, T. Fahle, M. Sellmann. 1999. Framework for constraint programming based column generation. *Proceedings of* 5^{th} *International Conference CP'99, Alexandria, Virginia, USA, October 1999*, Springer-Verlag, Heidelberg, Germany. 261–274.

Kasper, T. 1998. *A unifying logical framework for Integer Linear Programming and Finite Domain Constraint Programming.* PhD thesis, Fachbereich Informatick, University of Saarlandes, Saarbrücken, Germany.

Mackworth, A. 1977. Consistency in networks of relations. *Journal of Artifical Intelligence* **8** 99–118.

Michel, L., P. Van Hentenryck. 2001. Modeler++: a modeling layer for constraint programming libraries. *Proceedings of the Third International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR-01)*, Wye College (Imperial College), Ashford, UK.

Nemhauser, G., L. Wolsey. 1988. *Integer and Combinatorial Optimization.* Wiley, New York City, New York, USA.

Nuijten, W. P. M., E. H. L. Aarts. 1996. A computational study of constraint satisfaction for multiple capacitated job shop scheduling. *European Journal of Operational Research* **90** 269–284.

Ottosson, G., E. S. Thorsteinsson, J. N. Hooker. 2002. Mixed global constraints and inference in hybrid CLP–IP solvers. *Annals of Mathematics and Artificial Intelligence, Special Issue on Large Scale Combinatorial Optimisation and Constraints* **34** 271–290.

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

Puget, J.-F., M. Leconte. 1995. Beyond the glass box: constraints as objects. J. Lloyd, ed. *ILPS'95: Proceedings 5rd International Logic Programming Symposium*, MIT Press, Cambridge, Massachusetts, USA. 513–527.

Refalo, P. 1999. Tight cooperation and its application in piecewise linear optimization. J. Jaffar, ed. *Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* **1713** 375–389.

Refalo, P. 2000a. Linear formulation of constraint programming models. *Proceedings of the Second International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR-00)*, University of Paderborn, Paderborn, Germany.

Refalo, P. 2000b. Linear formulation of constraint programming models and hybrid solvers. *Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming, CP'00, Lecture Notes in Computer Science* **1894** 369–383.

Refalo, P., P. Van Hentenryck. 1996. `CLP(`$\mathcal{R}_{lin}$`)` revised. *Proceedings of the Joint International Conference and Symposium on Logic Programming (JICSLP'96)*, Bonn, Germany. 1–14.

Régin, J.-C. 1994. A filtering algorithm for constraints of difference in CSPs. *Proceedings of the Twelfth National Conference on Artificial Intelligence AAAI-94, Seattle, Washington, USA, August 1994*, AAAI Press, Menlo Park, California, USA. 362–367.

Régin, J.-C. 1996. Generalized arc consistency for global cardinality constraint. *Proceedings of the Thirteenth National Conference on Artificial Intelligence AAAI-96, Portland, Oregon, USA, August 1996*, AAAI Press, Menlo Park, California, USA. 209–215.

Régin, J.-C. 1999. Arc consistency for global cardinality constraints with costs. J. Jaffar, ed. *Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* **1713** 390–404.

Régin, J.-C. 2001. Minimization of the number of breaks in sports scheduling problems using constraint programming. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* **57** 115–129.

Régin, J.-C., M. Rueher. 2000. A global constraint combining a sum constraint and binary inequalities. *Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming, CP'00, Lecture Notes in Computer Science* **1894** 384–395.

Rodosek, R., M. Wallace, M. Hajian. 1999. A new approach to integrating mixed integer programming and constraint logic programming. *Annals of Operational Research, Recent Advances in Combinatorial Optimization* **86** 63–87.

Savelsbergh, M. 1994. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing* **6** 445–454.

Thorsteinsson, E. S. 2001. *Hybrid Approaches to Combinatorial Optimisation.* PhD thesis, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA.

Thorsteinsson, E. S., G. Ottosson. 2001. Linear relaxations and reduced-cost based propagation of continuous variable subscripts. *Annals of Operations Research, Special Issue on Integration of Constraint Programming, Artificial Intelligence and Operations Research Methods.* To appear.

Tsang, E. 1993. *Foundations of Constraint Satisfaction.* Academic Press, London, UK and San Diego, California, USA.

Van Hentenryck, P. 1989. *Constraint Satisfaction in Logic Programming.* MIT Press, Cambridge, Massachusetts, USA.

Wolsey, L. 1998. *Integer Programming.* Wiley, New York City, New York, USA.

Zhou, J. 1997. A permutation-based approach for solving the job-shop problem. *Constraints* **2** 185–213.