Linear Formulation of Constraint Programming Models and Hybrid Solvers

Philippe Refalo

ILOG, Les Taissounieres
1681, route des Dolines, 06560 Sophia Antipolis, France
refalo@ilog.fr

Abstract. Constraint programming offers a variety of modeling objects such as logical and global constraints, that lead to concise and clear models for expressing combinatorial optimization problems. We propose a way to provide a linear formulation of such a model and detail, in particular, the transformation of some global constraints. An automatic procedure for producing and updating formulations has been implemented and we illustrate it on combinatorial optimization problems.

Keywords: linear formulation, global constraints, hybrid solvers

1 Introduction

Constraint programming (CP) offers a variety of modeling facilities, such as logical and global constraints, that lead to concise and clear models for expressing combinatorial optimization problems. CP models contain useful structural information permitting the development of dedicated and efficient domain reduction algorithms for high-level constraints and the design of efficient ad hoc solution search strategies. Industrial implementations (CHIP [4] and ILOG Solver [18], for instance) have shown the effectiveness of CP for solving problems in different areas.

Mixed integer programming (MIP) techniques are also an effective and widely used method for solving combinatorial optimization problems. With MIP, the model is limited to a set of linear constraints over binary, integer, or real variables. A MIP formulation is sometimes far from natural and contains little structural information. The emphasis is on having good linear relaxations of the problem. Consequently, MIP solvers maintain a relaxed optimal solution of the linear relaxation and generate cutting planes to strengthen the relaxation [22].

In this paper, we propose a way to provide a MIP formulation of a CP model. The aim of this transformation is threefold. First, it permits the use of a MIP solver for solving a CP model. This is for solving models where MIP techniques are effective, without having to give a linear formulation. Second, it permits the use of constraint programming capabilities for designing search strategies on the high level model together with linear relaxations. Solving the linear relaxation provides a lower bound that prunes the search space, while the relaxed optimal

R. Dechter (Ed.): CP 2000, LNCS 1894, pp. 369–383, 2000.
© Springer-Verlag Berlin Heidelberg 2000

solution guides the search towards optimal solutions [2,13]. Third, it facilitates the use of hybrid solvers where domain reduction algorithms cooperate with a linear relaxation solver. This last approach has been shown to be promising (see [8,17,15,13]).

In the following we detail, in particular, the reformulation of the "all different" constraint [14]; constraints over occurrences of values such as "atmost", "atleast", and "among" [4]; constraints for variable indexing such as "element" [19]; and the "cycle" constraint [4]. The formulations presented can be viewed as an extension of the work done by Wallace et al. on the automatic transformation of a constraint logic program (including alldifferent constraints) to a MIP model [16] and of the work done on the transformation of logical formulas to MIP models [21,11] which is closely related to the seminal work of Balas and Jeroslow on disjunctive programming [3,10].

The linear formulation of a constraint is divided into two sets: the set of linear constraints that are *required* for the linearization and the set of linear constraints that are *delayed*. The latter are cutting planes that are added to the linear formulation when they are violated by the relaxed optimal solution. Cutting plane generation is embedded in the propagation process of the high-level constraint. Thus a global constraint can propagate not only new domains but also arbitrary constraints. Generating cutting planes from a global constraint by using the structure of the constraints is a clean integration of integer programming techniques into constraint programming which is related to the work of Bockmayer et al. [5]. However, this approach does not address the problem of systematically providing a linear formulation of models containing global constraints.

An automatic procedure for producing and updating formulations has been implemented and we illustrate it on problems such as the asymmetric traveling salesman problem, the facility location problem, and the quadratic assignment problem. Interestingly, our general reformulation of a natural CP model of a problem often gives a standard and even strong MIP formulation. Some experimental results are also given to illustrate the effectiveness of this method.

The rest of this paper is organised as follows. Section 2 gives details about how high-level constraints are reformulated in general. In Section 3 the reformulation of some global constraints is given. In Section 4, three applications are presented, and Section 5 concludes this article.

2 Constraint Reformulation

A constraint programming model is composed of variables and constraints. Constraints can be (1) domain constraints, (2) elementary constraints such as arithmetic constraints, and (3) high-level constraints which can be logical [20] or global constraints [4]. A linear formulation of these constraints needs to address two important issues. The first is the strength of the formulation that determines relevance of the linear relaxation; the second is maintaining this strength during the search.

2.1 Linear Reformulation

A linear reformulation of a set S of constraints over a set of variables V is a set S' of linear constraints over variables $V \cup V'$ such that both sets have the same solutions on variables V.

A set of linear constraints defines a convex set. Let Q be the solutions set of a high level constraint c. The strongest linear formulation of c represents the smallest convex set containing the solutions of Q. This set is called the *convex hull* of Q and the associated linear constraints represent a *sharp* formulation of Q [21]. The closer a formulation to a sharp one, the stronger it is.

Various sharp formulations can represent the same solution set. Moreover, in many cases the size of a strong formulation can be huge and it is desirable to avoid considering the whole reformulation at the same time. Therefore, the linear formulation $\mathcal{F}(c)$ of a constraint c is divided into two sets:

$$\mathcal{F}(c) = \mathcal{L}(c) \cup \mathcal{D}(c)$$

where $\mathcal{L}(c)$ is the set of linear constraints that are required in the reformulation and $\mathcal{D}(c)$ is a set of linear constraints whose addition is delayed. Constraints of $\mathcal{D}(c)$ are cutting planes that are not stored explicitly, but generated when needed. They are added when violated by the relaxed optimal solution. In the following, not specifying $\mathcal{D}(c)$ means that no constraint is delayed.

Constraints are not reformulated independently. Domain constraints are also considered. That is if S_D is the set of domain constraints of S, a formulation of $S_D \cup \{c\}$ is provided for c.

Since many logical conditions over linear constraints can be represented as a disjunction of convex sets, research in this field has focused on the linear representation of such disjunctions. A sharp formulation of $D = \{Ax \leq b \vee A'x \leq b'\}$ can have an exponential number of constraints when expressed on variables x. However, by introducing two new vectors of variables x^1 and x^2 and two binary variables γ_1 and γ_2, a sharp formulation of D is given by the system:

$$\mathcal{L}(D) = \begin{cases} Ax^1 \leq \gamma_1 b \\ A'x^2 \leq \gamma_2 b' \\ x = x^1 + x^2 \\ \gamma_1 + \gamma_2 = 1 \\ \gamma_i \in \{0,1\} \text{ for } i \in \{1,2\} \end{cases}$$

This fundamental result is known as the *disjunctive formulation*, which was developed by Balas and Jeroslow [3,10]. This transformation is applied in the following for constraints that can be restated as disjunctions of linear constraint sets.

2.2 Maintaining Sharpness under Domain Reduction

An important property of a linear formulation that is used together with a Branch and Bound search is *hereditary sharpness*, where the formulation remains sharp when variables are fixed from a parent node to a child node of the

search tree [21]. In constraint programming with hybrid solvers, we need to go further. Since new domains are inferred when going down the search tree, we must maintain the sharpness under domain reduction. In [13] a general principle called *tight cooperation* is presented where linear formulations of high level structures are dynamically updated with variable fixing and cutting planes generation when domains are reduced. This goes beyond the classical solvers cooperation framework, where only bounds on variables are exchanged between the domain reduction solver and the linear optimizer [2,17].

The same approach is applied herein. For each constraint, the formulation is updated so that the modifications done during search on the (high-level) constraint programming model are reflected on the (low-level) linear formulation. As an example, suppose that in the disjunctive formulation above, the alternative $Ax \leq b$ becomes unsolvable w.r.t. new domains on variables x at a node of the search tree. Fixing the variable γ_1 to 0 maintains the sharpness of the formulation w.r.t to new domains. Conversely, if $Ax \leq b$ is entailed w.r.t domains of x, the variable γ_1 must be fixed to 1.

3 Reformulation of Constraint Programming Models

This section details the reformulation of domain constraints and some high-level constraints, such as the alldifferent constraint, constraints over occurrences of variables, variations of the element constraint, the cycle constraint, and unary meta constraints.

3.1 Domain Constraint

A sharp linear formulation of a domain constraint $x \in D$ is obtained by the disjunctive formulation of $\bigvee_{a \in D}(x = a)$. For writing ease, let's denote the variable introduced for the alternative $x = a$ by $v_{x=a}$. It is assumed that $v_{x=a} = 0$ when $a \notin D$. The linear formulation of a domain constraint is thus:

$$\mathcal{L}(x \in D) = \begin{cases} \sum_{a \in D} v_{x=a} = 1 \\ 0 \leq v_{x=a} \leq 1 \text{ for } a \in D \\ \text{integer}(v_{x=a}) \text{ for } a \in D \end{cases}$$

$$\mathcal{D}(x \in D) = \left\{ x = \sum_{a \in D} a \times v_{x=a} \right. \tag{1}$$

The addition of constraints linking the binary variables and the original variable is delayed, because the linear formulation of high-level constraints often concerns the binary variables only. This is the case for the constraints presented below. Thus, delaying these constraints significantly reduces the size of the linear problem solved in practice.

The impact of constraint propagation algorithms is reduced domains for variables. The linear formulation of a domain constraint can be updated when the domain D is reduced to a new domain D'. To maintain the sharpness of the

formulation, variables are fixed depending on the values removed, according to the following propagation rules:

$$\begin{aligned} \forall \alpha \in D, \alpha \notin D' &\Leftrightarrow v_{x=\alpha} = 0 \\ D' = \{\alpha\} &\Leftrightarrow v_{x=\alpha} = 1 \end{aligned}$$

Conversely, if some variables of the reformulation are fixed with an integer programming technique such as reduced cost fixing [22], the original domain is updated. Observe that variable fixing ensures that the value of x in the relaxed optimal solution stays between the lower and upper bound of its domain.

All linear formulations below reuse the variables introduced for the reformulation of the domain constraint. Therefore, these rules are sufficient to maintain sharpness of all global constraints presented in the following. Note also that, in many cases, these updating rules are sufficient to ensure that the delayed constraint is satisfied. Consequently, it may never be added.

3.2 Alldifferent Constraint

The constraint alldifferent$(x_1, \ldots, x_n)$ is satisfied if variables $x_1, \ldots, x_n$ have different values. A sharp formulation of this constraint is well-known: it is that of a bipartite matching [22].

Let $K = \bigcup_{i=1}^{n} D_i$ be the union of domains. The linearization of this constraint formulates that each value of K can be given at most once to any of the variables $x_1, \ldots, x_n$. Hence we have

$$\mathcal{L}(\text{alldifferent}(x_1, \ldots, x_n)) = \left\{ \sum_{i=1}^{n} v_{x_i=j} \leq 1,\ j \in K \right\}$$

Example 1 *Assuming that* $x_1, x_2, x_3 \in \{1, 2, 3, 4\}$ *and* $\gamma_{ij} = v_{x_i=j}$ *we have*

$$\mathcal{L}(\mathit{alldifferent}(x_1, x_2, x_3)) = \left\{ \begin{array}{l} \gamma_{11} + \gamma_{21} + \gamma_{31} \leq 1 \\ \gamma_{12} + \gamma_{22} + \gamma_{32} \leq 1 \\ \gamma_{13} + \gamma_{22} + \gamma_{33} \leq 1 \\ \gamma_{14} + \gamma_{24} + \gamma_{34} \leq 1 \end{array} \right\}$$

3.3 Constraint over Occurrences of Values

Many requirements in practical problems limit the number of values that a set of variables can have in a solution. This is the case for the *among* [4] constraint of CHIP and the *distribute* constraint of ILOG Solver [18].

The among constraint restricts the number of variables from a set having a given value. The constraint

$$\text{among}(y, [x_1, \ldots, x_n], [a_1, \ldots, a_k])$$

where x_i and y are variables and a_i are real numbers, is satisfied when exactly y variables among $x_1, \ldots, x_n$ have their value in the set $\{a_1, \ldots, a_k\}$. The formulation of this constraint specifies that each of the values a_i must be taken y times by variables x_i:

$$\mathcal{L}(\text{among}(y, [x_1, \ldots, x_n], [a_1, \ldots, a_k])) = \left\{ y = \sum_{i=1}^{n} \sum_{j=1}^{k} v_{x_i=a_j} \right\}$$

Without going into details, this formulation is obviously sharp since the associated one line matrix is totally unimodular (see [22]).

Example 2 *Assuming that* $x_1, x_2, x_3 \in \{1, 2, 3, 4\}$ *and* $\gamma_{ij} = v_{x_i=j}$ *we have*

$$\mathcal{L}(\textit{among}(y, [x_1, x_2, x_3], [1, 3]) = \{y = \gamma_{11} + \gamma_{21} + \gamma_{31} + \gamma_{13} + \gamma_{23} + \gamma_{33}\}$$

The ILOG Solver constraint distribute($[x_1 \ldots, x_n]$, $[a_1, \ldots, a_k]$, $[y_1, \ldots, y_k]$) is satisfied if the number of variables among $x_1, \ldots, x_n$ having the value a_i is equal to the variable y_i. It has a sharp formulation when formulated as a conjunction of among constraints: among(y_1, $[x_1, \ldots, x_n]$, $[a_1]$) $\wedge \ldots \wedge$ among (y_k, $[x_1, \ldots, x_n]$, $[a_k]$)}.

Note that the cardinality constraint atmost($\alpha, [x_1, \ldots, x_n], \beta$) and the constraint atleast($\alpha, [x_1, \ldots, x_n], \beta$) which require the number of variables from $x_1, \ldots, x_n$ that take the value β to be respectively atmost or atleast α, are also a special case of among constraints, and thus can be given a sharp formulation the same way.

3.4 Element Constraint

The constraint element was probably one of the first global constraints introduced in constraint logic programming systems [19]. The syntax is:

$$\text{element}(x, [a_1, \ldots, a_n], z)$$

where z is a variable, x is a variable whose domain D_x is a subset of $\{1, \ldots, n\}$, and a_i are real values. This constraint is satisfied if z is equal to the x^{th} value of the array $[a_1, \ldots, a_n]$. In [9], this constraint is formulated by inequalities over x and z variables. Since we deal with domain constraints, we give a different formulation that has the advantage of being efficiently updated when domains are reduced.

Solutions of this constraint can also be represented by the disjunction $\bigvee_{i=1}^{n}(x = i \wedge z = a_i)$. Assuming that $D_x = \{1, \ldots n\}$, its disjunctive formulation gives the set:

$$\begin{cases} z = a_1\gamma_1 + \cdots + a_n\gamma_n \\ x = \gamma_1 + \cdots + n\gamma_n \\ \gamma_1 + \cdots + \gamma_n = 1 \\ 0 \leq \gamma_i \leq 1 \text{ for } \ i \in D_x \\ \text{integer}(\gamma_i) \text{ for } i \in D_x \end{cases}$$

Since all constraints, except the first one, formulate the domain constraint on x, we have

$$\mathcal{L}(\text{element}(x,[a_1,\ldots,a_n],z)) = \left\{ z = \sum_{i\in D_x} a_i v_{x=i} \right\}$$

Example 3

$$\mathcal{L}\left(\begin{array}{l} x \in \{1,2,3\} \\ \textit{element}(x,[7,8,12],z) \end{array}\right) = \left\{\begin{array}{l} x = \gamma_1 + 2\gamma_2 + 3\gamma_3 \\ \gamma_1 + \gamma_2 + \gamma_3 = 1 \\ z = 7\gamma_1 + 8\gamma_2 + 12\gamma_3 \\ 0 \le \gamma_i \le 1 \textit{ for } i \in \{1,2,3\} \\ \textit{integer}(\gamma_i) \textit{ for } i \in \{1,2,3\} \end{array}\right\}$$

Note that in this example, and in general, a compact reformulation is obtained if the variable $v_{x=i}$ is used for the alternative $y = a_i$. Consequently, the linear formulation of this constraint does not require any variables in addition to those that are introduced for domain constraints of x.

Element constraints can also be defined on higher dimensional arrays. Consider an element constraint element(x, y, A, z) over an $m \times n$ matrix A where x and y are variables having domains $D_x \subset \{1,\ldots,m\}$ and $D_y \subset \{1,\ldots,n\}$, which is satisfied when z is equal to the element $A_{x,y}$. This constraint has the following disjunctive formulation:

$$\bigvee_{i=1,\ldots,m \quad j=1,\ldots,n} (x = i \wedge y = j \wedge z = A_{ij})$$

Thus a sharp formulation is:

$$\mathcal{L}(\text{element}(x,y,A,z)) = \left\{\begin{array}{l} v_{x=i} = \sum\limits_{j=1,\ldots,n} (v_{z=A_{ij}}) \text{ for } i \in D_x \\ v_{y=j} = \sum\limits_{i=1,\ldots,m} (v_{z=A_{ij}}) \text{ for } j \in D_y \end{array}\right.$$

The variables γ_{ij} can be reused for $v_{z=A_{ij}}$ provided that the value A_{ij} appears only once in the matrix.

Example 4

$$\mathcal{L}\left(\begin{array}{l} x \in \{1,2\} \\ y \in \{1,2,3\} \\ \textit{element}(x,y,\begin{bmatrix} 7 & 8 & 4 \\ 3 & 1 & 2 \end{bmatrix},z) \end{array}\right) = \left\{\begin{array}{l} x = \gamma_1 + 2\gamma_2 \\ \gamma_1 + \gamma_2 = 1 \\ y = \delta_1 + 2\delta_2 + 3\delta_3 \\ \delta_1 + \delta_2 + \delta_3 = 1 \\ z = 7\lambda_1 + 8\lambda_2 + 4\lambda_3 + 3\lambda_4 + 1\lambda_5 + 2\lambda_6 \\ \lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 + \lambda_5 + \lambda_6 = 1 \\ \gamma_1 = \lambda_1 + \lambda_2 + \lambda_3 \\ \gamma_2 = \lambda_4 + \lambda_5 + \lambda_6 \\ \delta_1 = \lambda_1 + \lambda_4 \\ \delta_2 = \lambda_2 + \lambda_5 \\ \delta_3 = \lambda_3 + \lambda_6 \\ 0 \le \gamma_i \le 1, \textit{integer}(\gamma_i) \textit{ for } i \in \{1,2\} \\ 0 \le \delta_i \le 1, \textit{integer}(\delta_i) \textit{ for } i \in \{1,2,3\} \\ 0 \le \lambda_i \le 1, \textit{integer}(\lambda_i) \textit{ for } i \in \{1,\ldots,6\} \end{array}\right\}$$

This formulation can be easily extended to the case of an array of arbitrary dimensions.

3.5 Cycle Constraint

The constraint cycle$(k, [x_1, \ldots, x_n])$, where k is a positive integer and x_i are variables having domains $D_i \subset \{1, \ldots, n\}$, is mainly used to solve routing problems [4]. This constraint is satisfied when the values of the variables define k disjoint circuits in an directed graph such that each node is visited exactly once. Initially, a variable x_i is associated with the node i, the domain D_i corresponding to the set of possible nodes that can be visited from i.

A cycle in a set of variables $\{x_1, \ldots, x_n\}$ is a sequence of p indexes $C(1), ..., C(p)$ such that $C(i+1) \in D_{C(i)}$ for $i < p$ and $C(p) \in D_{C(1)}$. For instance if we have $x_1 \in \{2, 3\}$, $x_2 \in \{1, 3\}$, $x_3 \in \{2, 3\}$ the sequence $C(1) = 1$, $C(2) = 3$, $C(3) = 2$ is a cycle and thus the constraint cycle$(1, [x_1, x_2, x_3])$ is satisfied by the assignment $x_1 = 3$, $x_2 = 1$, $x_3 = 2$.

The formulation of the cycle constraint requires that each node must belong to a single circuit. This is equivalent to the formulation of an alldifferent constraint. To constrain the number of cycles to be at most k, the formulation also enforces connectivity on a partition of $k+1$ subset of $\{1, \ldots, n\}$. There must be at least one variable having its index in one set and its value in another set. The number of constraints enforcing this is exponential, thus their addition to the linear model is delayed. To constrain the number of cycles to be atleast k, the formulation enforces that for each set of $k-1$ disjoint cycles covering all nodes (called a $k-1$ cycle-cover), one of the cycles must be broken.

$$\mathcal{L}(\text{cycle}(k, [x_1, \ldots, x_n])) = \mathcal{L}(\text{alldifferent}([x_1, \ldots, x_n]))$$

$$\mathcal{D}(\text{cycle}(k, [x_1, \ldots, x_n])) =$$

$$\begin{cases} \sum_{e=1}^{k+1} \sum_{i \in S_e} \sum_{j \in P \setminus S_e} (v_{x_i = j}) \geq 1 \\ \qquad\qquad \text{for all partition } (S_1, \ldots, S_{k+1}) \text{ of } \{1, \ldots, n\},\ S_p \neq \emptyset \\ \sum_{e=1}^{k-1} \sum_{i \in C_e} (v_{x_{C_e(i)} = C_e(i+1)}) \leq n - 1 \\ \qquad\qquad \text{for all } k-1 \text{ cycle-cover } (C_1, \ldots, C_{k-1}) \text{ of } \{1, \ldots, n\} \end{cases}$$

Observe that when $k = 1$, the set of delayed constraints represents the family of subtour elimination inequalities [22]. Consequently, this formulation is not sharp in general but, as it will be shown in the section of experimental results, it is strong enough to handle cycle constraints involving a few hundred variables. Stronger formulations could be obtained by generalizing stronger cuts for the case where $k = 1$; that is other strong cuts used to solve traveling salesman problems.

3.6 Higher-Order Expressions

Higher order expressions are arithmetic constraints that appear as terms in a constraint. For instance, $(x + y \leq 1) + (y = 5) = 1$ contains two higher-order

expressions. The linear formulation of such constraints can be obtained by using the *big-M* formulation which introduces a binary variable for each constraint as a term [11]. It has been used in [16] for reformulating constraint logic programs as MIP models.

When using domain constraints, higher-order unary constraints must be handled differently. For instance, the occurrence of an expression $(x = 2)$ in a linear constraint should be replaced by the variable $v_{x=2}$ and not by the binary variable of the *big-M* formulation. More generally, here is how to reformulate unary higher order terms into linear terms in the general case.

$$\begin{aligned}
\mathcal{L}'((x = a)) &= (v_{x=a}) \\
\mathcal{L}'((x \neq a)) &= (1 - v_{x=a}) \\
\mathcal{L}'((x \geq a)) &= (\sum_{i \in D_x, i \geq a} v_{x=i}) \\
\mathcal{L}'((x \leq a)) &= (\sum_{i \in D_x, i \leq a} v_{x=i})
\end{aligned}$$

The mapping $\mathcal{L}'$ is an extension of the mapping $\mathcal{L}$ to the set of higher order terms. Note that, here again, the formulation does not introduce any new variables but reuses variables from the reformulation of the domain constraints.

4 Examples of Problem Solving Using Reformulation

A system has been implemented on the top of `ILOG Planner/CPLEX` [12] (for handling linear formulation) and `ILOG Solver` [18] (for domain reduction). It automatically reformulates a CP model into a MIP model. In this section, we detail the modeling of three classes of problems. For each of them we study the MIP formulation. To show the effectiveness of the method, some practical results are also given. In the following, all computation times are given in seconds on a Pentium II-300.

4.1 Facility Location Problem

This example illustrates the creation of a new global constraint. This is not for developing an efficient domain reduction algorithm, as is usually done in constraint programming, but to capture the structure of part of a model, whose sharp formulation must be delayed.

The facility location problem is to assign m clients to n depots that deliver to clients. A fixed cost f_j is associated with the use of depot j and a transportation cost c_{ij} if client i is delivered from depot j. A depot has a maximum number of clients w_j that it can serve. The problem is to decide which depots to open and which depot serves each client so as to minimize the fixed and transportation costs.

To model this problem, a binary variable y_j is introduced for each depot. It is equal to 1 if the depot j is open and to 0 otherwise. We also introduce for each client i, a variable x_i whose value corresponds to the depot that serves it. Thus

	Weak Model	Strong Model	
Problem	Time	Ch. Pts	Time
cap101	> 1000	0	3.9
cap102	> 1000	1	6.7
cap111	> 1000	2	11.0
cap112	> 1000	7	15.6
cap113	> 1000	14	21.8
cap114	> 1000	21	33.5
cap131	> 1000	0	21.3
cap132	> 1000	0	18.5
cap133	> 1000	0	20.7
cap134	> 1000	0	22.0

Fig. 1. Results on Facility Location Problems

we have the constraint $x_i \in \{1, \ldots, n\}$. The whole problem can be formulated in the following way:

$$\min \sum_{j=1}^{n} f_j y_j + \sum_{i=1}^{m} z_i$$

$$\text{s.t.} \begin{cases} y_j \in \{0,1\} \text{ for } j \in \{1,\ldots,n\} & \text{(f1)} \\ x_i \in \{1,\ldots,n\} \text{ for } i \in \{1,\ldots,m\} & \text{(f2)} \\ \text{element}(x_i, [c_{i1},\ldots,c_{in}], z_i) \text{ for } i \in \{1,\ldots,m\} & \text{(f3)} \\ (x_1 = j) + \cdots + (x_m = j) \leq w_j y_j \text{ for } j \in \{1,\ldots,n\} & \text{(f4)} \end{cases}$$

The constraint element associates a cost z_i with a customer i, depending on the warehouse x_i that serves it. The last n higher-order constraints enforce that not more than w_j customers from $x_1, \ldots, x_m$ can be assigned to a depot j whose decision variable is y_j.

It is well known that this formulation can be strengthened by constraints $v_{x_i=j} \leq y_j$, whose logical meaning is to force y_j to be 1 if the j^{th} depot serves customer i (see [22]). In fact, these constraints define the convex hull of the solution set of each constraint (f4). Since there are $n \times m$ strengthening constraints for a problem, it is efficient to delay their addition. To capture this structure, we introduce a new global constraint serve$(j, y, [x_1, \ldots, x_m], w)$ where j is a depot index, y is the binary variable associated with this depot, x_i are the variables associated with clients, and w is the depot capacity. This constraint is introduced in place of each constraint (f4). Its linear formulation is:

$$\mathcal{L}(\text{serve}(j, y, [x_1, \ldots, x_m], w)) = \left\{ \sum_{i=1}^{n} v_{x_i=j} \leq w \times y \right\}$$

$$\mathcal{D}(\text{serve}(j, y, [x_1, \ldots, x_m], w)) = \{v_{x_i=j} \leq y \text{ for } i = 1, \ldots, m\}$$

To illustrate the effect of this new formulation, experiments were made on a list of problems from the OR Library[1]. The problems considered have from 25 to 50 facilities and 50 clients. Problems are solved using constraint programming and linear relaxations only. That is we use `ILOG Solver` to program the search over the linear optimizer `ILOG Planner/CPLEX`. The search strategy is based on the relaxed optimal solution. It chooses the expression $(x_i = j)$ whose relaxed solution is the closest to 0.5. This is to apply the first fail principle: these expressions are, a priori, the most difficult to instantiate. A binary choice point is created to explore either the subproblem when stating $x_i = j$ or the one when stating $x_i \neq j$. A solution is found when all variables have integer values.

The results are presented in figure 1. A comparison is made between solving the model with the constraints (f4) (Weak Model) and the one with the global constraint "serve" (Strong Model). The Strong model is solved in less than a minute for all instances considered, while no weak models were solved within the time limit of 1000 seconds.

4.2 Traveling Salesman Problem

The traveling salesman problem (TSP) is to visit a set of cities exactly once while minimizing the sum of the costs c_{ij} of going from a city i to a city j. The cost c_{ij} can be different from c_{ji}, in which case the problem is asymmetric. It can be modeled very concisely in constraint programming with the constraint cycle. For n cities to visit, n variables x_i are introduced, whose domain is included in $\{1, \ldots, n\}$ and that are assigned in a solution to the city to visit after i. A constraint programming formulation uses element constraints to associate a cost with each visit leaving a city and a cycle constraint for enforcing that visits must be done in one tour. The term to minimize is the sum of each visiting cost.

$$\min \sum_{i=1}^{n} z_i$$
$$\text{s.t.} \begin{cases} \text{element}(x_i, [c_{i1}, \ldots, c_{in}], z_i) \text{ for } i \in \{1, \ldots, n\} \\ \text{cycle}(1, [x_1, \ldots, x_n]). \\ x_i \in D_{x_i},\ D_{x_i} \subset \{1, \ldots, n\} \text{ for } i \in \{1, \ldots, n\} \end{cases}$$

The linear constraints in the required part of the reformulation of this model define an assignment problem, which corresponds to the standard initial relaxation to solve this problem [22].

Two approaches to solve this model are compared. Some experimental results are given on small instances of symmetric and asymmetric traveling salesman problem (TSP) from the TSPLIB suite[2].

The first approach (Propag. + cost) uses the domain reduction techniques of `ILOG Solver` and, in addition, a constraint alldiff($[x_1, \ldots, x_n]$) that takes into account the objective function and performs pruning using an embedded assignment problem solver [7,6]. This approach outperforms previous pure constraint

[1] Problems are available at `http://mscmga.ms.ic.ac.uk/jeb/orlib/capinfo.html`
[2] Problems are available at `http://softlib.rice.edu/softlib/tsplib/`

Asymmetric TSP					Symmetric TSP				
	Propag. + cost		Linear Relax.			Propag. + cost		Linear Relax.	
Problem	Fails	Time	Ch. Pts	Time	Problem	Fails	Time	Ch. Pts	Time
br17	-	-	9	0.3	gr17	646	0.7	0	0.3
ry48p	100000	248.0	40	12.2	gr21	31	0.1	0	0.2
ft53	6900	630.0	272	47.7	gr24	120	0.3	5	0.7
ft70	113	3.7	5	5.4	fri26	1600	2.9	1	0.7
ftv33	-	-	2136	195.2	bays29	8800	13.7	8	1.5
ftv35	-	-	5547	4137.2					
ftv38	-	-	36987	2697.3					
ftv44	-	-	841	116.1					
ftv47	-	-	27910	2730.6					
kro124p	121000	1321.2	46	58.2					

Fig. 2. Results on Traveling Salesman Problems

programming approaches and compares well with a dedicated approach. Several search strategies have been used and we give for each problem the time and number of fails obtained by the best approach. The results concern the proof time only, that is the time spent after having found an optimal solution because the method was not able to find rapidly optimal solutions. Entries are blank for problems that have not been tried.

The second approach (Linear Relax.) uses the formulation above. It solves the problem by using linear relaxations only. It uses `ILOG Solver` to program the search over the linear optimizer `ILOG Planner/CPLEX`. The search strategy used is the same as that used for the facility location problem. Here full computation times are given, that is the time for finding an optimal solution plus the time spent to prove optimality.

The results are presented in figure 2. The approach using linear relaxations often gives better results on larger problems. This is not very surprising, since the relaxation provided by the cutting planes added by the cycle constraint is tighter than that of the assignment problem. More important is to observe that the search strategy using the relaxed optimal solution allows us to find an optimal solution quickly. Appropriate choices are made for the variables, due to the tightness of the relaxation. It is interesting to note that the cutting planes are added locally to a node of the search tree and are removed when backtracking. This eliminates automatically irrelevant cuts, since those generated on one branch are mostly useless on other branches at the same level. Note also that many fewer choice points are needed to solve the problem. However, the time spent per node is larger since cutting plane generation is more time consuming than domain reduction.

Results obtained when using domain reduction in addition to linear relaxation are not presented here, because we uniformly obtained a slight reduction of search space together with a slight slowdown.

	Propagation		Linear Relax		Cooperation	
Problem	Ch. Pts	Time	Ch. Pts	Time	Ch. Pts	Time
7×7 (1)	9207	3720.1	51	15.1	49	17.2
7×7 (2)	11249	4261.2	37	13.3	32	12.1
8×8		> 5000	184	109.2	154	90.3
9×9 (1)		> 5000	652	780.6	295	580.5
9×9 (2)		> 5000	1183	2280.7	621	1780.1

Fig. 3. Results on Quadratic Programming Problem Instances

4.3 Quadratic Assignment Problem

The quadratic assignment problem (QAP) problem consists of locating n factories in n cities. There is a flow A_{ij} between factory i and factory j and there is a cost d_{lk} for each unit of flow transported from city l to city k. The problem is to determine where to locate the factories while minimizing the flow transportation cost, that is, $A_{ij} \times d_{loc(i)loc(j)}$ where $loc(i)$ is the location of factory i. This problem is strongly NP-Hard and instances having $n \geq 15$ are difficult to solve in practice.

A constraint programming formulation is possible by way of element constraints on matrices. For each city, a variable $x_i \in \{1, \ldots, n\}$ is introduced whose value is the factory number that is build in this city. These variables are constrained to take different values. Also, $n \times n$ variables y_{ij} are introduced. Each of them is assigned to the cost of transporting the flow from city i to city j. The constraint programming formulation is thus

$$\min \sum_{i,j=1,\ldots,n} d_{ij} \times y_{ij}$$

$$\text{s.t.} \begin{cases} x_i \in \{1, \ldots, n\} \text{ for } i \in \{1, \ldots, n\} \\ y_{ij} \in \{A_{uv} \mid u, v = 1, \ldots, n\} \text{ for } i, j \in \{1, \ldots, n\} \\ \text{alldifferent}(x_1, x_2, \ldots, x_n) \\ \text{element}(x_i, x_j, A, y_{ij}) \text{ for } i, j \in \{1, \ldots, n\} \end{cases}$$

where A is the matrix of flow between factories.

Interestingly, the reformulation of this problem gives the tight linear formulation introduced recently by Adams and Jonhson [1]. Several lower bounds introduced for solving QAP with a MIP solver can be considered as approximations of the lower bound given by this formulation. Unfortunately, the reformulation introduces a large number of binary variables and constraints, which create formulations having a large size for $n \geq 8$.

Nevertheless, we have solved problems having with up to 9 cities with this approach. The search strategy is the same as that used for previous examples. Note that only variables x need to be instantianted, since fixing these variables implies that variables y are also fixed. The results presented in figure 3 show

the solving of the model with a pure CP approach (Propagation) using `ILOG Solver`, an approach using linear relaxation in constraint programming (Linear Relax.) and an approach using domain reduction together with linear relaxation (Cooperation).

We can observe that the pure domain reduction approach cannot solve problems having a size greater than 7. The approach using linear relaxations only finds optimal solutions rapidly thanks to the relaxed optimal solution. It is worth observing that the cooperative approach is faster than any of the others. This is mainly due to pertinent domain reduction done by global constraints that tighten the linear relaxation.

5 Conclusion

We have presented a linear formulation for commonly used global constraints like alldiff, atleast, among, element, cycle and meta constraints. Our linearization provides a set of required linear constraints and a set of delayed linear constraints added when violated. Specific formulations can also be defined to capture problem substructures having a strong linear formulation.

The first advantage is that the problem statement is more concise and expressive than in integer programming. Interestingly, formulations produced automatically are often standard and tight for the problem considered. Moreover, linear constraints and binary variables introduced can be fully ignored when designing search strategies. Practical results on combinatorial optimization problems show that this approach can be effective.

We believe that automatic reformulations will be a fundamental aspect of modeling systems since this allows the use of linear relaxations while avoiding the need to provide a linear formulation. Our current research is to experiment with more problems that can benefit from this approach, to identify useful substructures, and to provide a strong formulation for them.

References

1. W. Adams and T. Johnson. Improved linear programming based lower bounds for the quadratic assignment problem. In P. Pardalos and H. Wolkowicz, editors, *Quadratic Assignment and Related Problems*, number 16, pages 49–72. AMS, Providence, Rhode Island, 1994. 381
2. B. De Backer and H. Beringer. Cooperative solvers and global constraints: The case of linear arithmetic constraints. In *Proceedings of the Post Conference Workshop on Constraint, Databases and Logic Programming, ILPS'95*, 1995. 370, 372
3. E. Balas. Disjunctive programming and a hierarchy of relaxations for discrete optimization problems. *SIAM Journal Alg. Disc. Meth.*, 6(3):466–486, 1985. 370, 371
4. N. Beldiceanu and E. Contejean. Introducing global constraints in CHIP. *Mathl. Comput. Modelling*, 20(12), 1994. 369, 370, 373, 376
5. A. Bockmayr and T. Kasper. Branch and infer: A unifying framework for integer and finite domain constraint programming. *INFORMS Journal on Computing*, 10(3):287–300, 1998. 370

6. F. Focacci, A. Lodi, and M. Milano. Cost-based domain filtering. In *Proceedings of* 5^{th} *International Conference CP 99*, Alexandria, Virginia, October 1999. Springer-Verlag. 379
7. F. Focacci, A. Lodi, M. Milano, and D. Vigo. Solving tsp through the integration of cp and or methods. In *Proceedings of the CP 98 Workshop on Large Scale Combinatorial Optimization and Constraints*, 1998. 379
8. J. N. Hooker and M. A. Osorio. Mixed logical / linear programming. *Discrete Applied Mathematics*, 1996. to appear. 370
9. J. N. Hooker, G. Ottosson, E. S. Thornsteinsson, and Hak-Jin Kim. A scheme for unifying optimization and constraint satisfaction methods. *Knowledge Engineering Review*, 2000. to appear. 374
10. R. Jeroslow. Logic based decision support: Mixed integer model formulation. *Annals of Discrete Mathematics*, (40), 1989. 370, 371
11. G. Mitra, C. Lucas, S. Moody, and E. Hadjiconstantinou. Tools for reformulating logical forms into zero-one mixed integer programs. *European Journal of Operational Research*, (72):262–276, 1994. 370, 377
12. ILOG Planner 3.3. *User Manual*. ILOG, S. A., Gentilly, France, December 1999. 377
13. P. Refalo. Tight cooperation and its application in piecewise linear optimization. In *Proceedings of* 5^{th} *International Conference CP 99*, Alexandria, Virginia, October 1999. Springer-Verlag. 370, 372
14. J. C. Regin. A filtering algorithm for constraints of difference in csps. In *Proceedings of AAAI-94*, pages 362–367, Seattle, Washington, 1994. 370
15. R. Rodosek and M. Wallace. A generic model and hybrid algorithm for hoist scheduling problems. In *Proceedings of the* 4^{th} *International Conference on Principles and Practice of Constraint Programming – CP'98*, pages 385 – 399, Pisa, Italy, 1998. Also in LNCS 1520. 370
16. R. Rodosek, M. Wallace, and M. T. Hajian. A new approach to integrate mixed integer programming with CLP. In *Proceedings of the CP'96 Workshop on Constraint Programming Applications*, Boston, MA, USA, 1996. 370, 377
17. M. Rueher and C. Solnon. Concurrent cooperating solvers over the reals. *Reliable Computing*, 3(3):325–333, 1997. 370, 372
18. ILOG Solver 4.4. *User Manual*. ILOG, S. A., Gentilly, France, June 1999. 369, 373, 377
19. P. van Hentenryck. *Constraint Satisfaction in Logic Programming*. MIT Press, Cambridge, Mass., 1989. 370, 374
20. P. van Hentenryck, H. Simonis, and M. Dincbas. Constraint satisfaction using constraint logic programming. *Artificial Intelligence*, 58(1–3):113–159, 1992. 370
21. H. P. Williams. *Model Building in Mathematical Programming*. Wiley, 1999. 370, 371, 372
22. L. A. Wolsey. *Integer Programming*. Wiley, 1998. 369, 373, 374, 376, 378, 379