Linear Relaxations and Reduced-Cost Based Propagation of Continuous Variable Subscripts

Erlendur S. Thorsteinsson [a] Greger Ottosson [b]

[a] *Graduate School of Industrial Administration; Carnegie Mellon University; Schenley Park; Pittsburgh, PA 15213; U.S.A.*
E-mail: esth@cmu.edu

[b] *Computing Science Department, Uppsala University; PO Box 311, S-751 05 Uppsala; Sweden.*
E-mail: greger@acm.org

Abstract In hybrid solvers for combinatorial optimisation, combining Constraint (Logic) Programming (CLP) and Mixed Integer Programming (MIP), it is important to have tight connections between the two domains. We extend and generalise previous work on automatic linearisations and propagation of symbolic CLP constraints that cross the boundary between CLP and MIP. We also present how reduced costs from the linear programming relaxation can be used for domain reduction on the CLP side. Computational results comparing our hybrid approach with pure CLP and MIP on a configuration problem show significant speed-ups.

Keywords: Mixed Integer Programming, Constraint Logic Programming, Integration, Dynamic Linear Relaxations, Reduced Costs, Inference, Propagation, Variable Subscripts, Mixed Global Constraints.

AMS Subject classification: 68N99,68Q99,68T99,90C05,90C11,90C27.

1. Introduction

The topic of this paper can be seen as the merger of three lines of research in the area of hybrid techniques of Constraint (Logic) Programming (CLP) and Mixed Integer Programming (MIP).

Firstly, while studying a configuration problem, we continue along the lines of Refalo [21], Rodošek, Wallace and Hajian [22], and Williams and Yan [24] in providing (automatic) linearisations of arithmetic and symbolic CLP constraints. This avenue of research is becoming increasingly important as the integration of CLP and MIP is pursued. The object of study here are terms with variable subscripts in continuous constraints (c_y and $c_y x$), i.e., a structure which in part is modelled with the `element` constraint in CLP.

Secondly, we continue a line of research by Focacci, Lodi and Milano [7,8, 9,10], who have used the *reduced costs* of a separate assignment subproblem for propagation in a CLP framework. We generalise this by showing how reduced-cost based inference can be applied to constraints whose linearisation is a part of

a larger linear programming relaxation.

Finally, we show how this fits nicely into the modelling framework Mixed Logical/Linear Programming (MLLP) and a hybrid CLP–MIP solver, which is a part of our previous line of research [14,15,20].

This paper is structured as follows. In Sec. 2 we recapitulate the basic framework of MLLP. Section 3 introduces our application, a class of configuration problems, with CLP, MIP and hybrid MLLP models. Sections 4 and 5 describe how the variable subscripts are linearised in MLLP. Section 6 describes reduced costs and how they can be used for inference using those linearisations. Finally, Sec. 7 gives computational results, which demonstrate the importance of the linear relaxation and the advantage of CLP–MIP integration.

2. Mixed Logical/Linear Programming (MLLP)

To lay the basis for the subsequent discussion, we recapitulate the basic framework of MLLP, proposed in [11,12,13,14,15,20]. In that framework, constraints are in the form of conditionals that link the discrete and continuous elements of the problem. An MLLP model has the form

$$\begin{aligned} &\min\ cx \\ &\text{s.t. } h_i(y) \rightarrow A^i x \geq b^i, \quad i \in I, \\ &\qquad x \in \mathbb{R}^n,\ y \in D, \end{aligned} \tag{1}$$

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

A linear constraint set $Ax \geq b$ which is enforced unconditionally may be so written for convenience, with the understanding that it can always be put in the conditional form $\mathrm{T} \rightarrow Ax \geq b$. Similarly, an unconditional discrete constraint h can be formally represented with the conditional $\neg h \rightarrow (0x = 1)$.

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

The basic constraint of MLLP is the conditional constraint and declaratively all MLLP models can be expressed as a set of these conditional constraints.

However, for solution speed and modelling clarity, MLLP encompasses other constraints that typically span a larger number of variables and utilise specialised propagation techniques. In Constraint Programming terminology, these are commonly referred to as *global constraints*. In addition to purely discrete global constraints that express the deductions as domain prunings, MLLP also has constraints that initially construct (and subsequently refine) a linear relaxation in addition to the constraint propagation. The variable subscripts that are the topic of this paper are indeed handled in this manner, as we shall see.

3. A Configuration Problem

In order to motivate the discussion in the following sections, in particular why variable subscripts (the `element` constraints) are important, we will begin by describing an important application.

Many industrial products come in different configurations, aiming to closely satisfy the needs of individual customers. 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. 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.

A computer is an example of such a product. It has components such as power supplies, hard drives, processors and memory, that must be arranged in a case and connected together. Each component can be of a different type, e.g., a power supply can be of different wattage, e.g., 250 Watt or 400 Watt, and different form factors, e.g., AT or ATX. Each component will consume/contribute attributes/resources, e.g., a power supply will consume different amounts of the *physical space* resource[1] (height, width, length) and of the *weight* resource based on its type, and will contribute different amounts to the *power* resource. A hard drive will also consume different amounts of the physical space resource and of the weight resource based on its type, and will consume different amounts of the power resource and contribute different amounts to the *disk space* resource.

In addition to determining the type of a particular component, we also have to determine how many to install, which multiplies the resource consumption/contribution, yet observing some simple relationships between the resources, e.g., that the hard drives can not consume more power than provided by the power supplies and everything has to fit into the computer case. This is an example of the resource constraints.

There are also some direct relationships between the components, e.g. differ-

[1] Note that the computer case contributes to the physical space resource.

ent types of hard drives will also require different types of drive controllers (e.g., EIDE, SCSI) to be installed. This is an example of the configuration constraints. There are many others, e.g., physical limitations on the distance between the hard drive and the hard drive controller, limitations on the orientation of the hard drive, limitations on the placement of memory and add-on cards, etc.

In general we can describe the problem as follows. 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 eq. (2). 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

$$
\begin{aligned}
\min \quad & \sum_{k \in R} c_k r_k \\
\text{s.t.} \quad & r_k = \sum_{i \in C} \Big(q_i \times A_{k,i,t_i} \Big), && \forall k \in R, && (2) \\
& h_l(q,t), && \forall l \in L, && (3) \\
& r_k \geq R_k, && \forall k \in R, && (4) \\
& t_i \in T_i,\ q_i \in \mathbb{Z}_+, && \forall i \in C.
\end{aligned}
$$

Note that the resource consumption "lookup" is handled in (2) 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, and is, as we shall see, modelled differently in CLP and MIP. Equation (3) states the set of problem specific configuration constraints, e.g., $q_1 > 0 \Rightarrow q_2 = 0$, $\texttt{alldiff}(t_1, \ldots, t_n)$, or some type or layout constraints, such as in the computer example above.

3.1. A CLP Model

In CLP the configuration problem is modelled with variable subscripts, i.e., with $\texttt{element}(y, [c_1, \ldots, c_k], z)$ constraints. The `element` constraint constrains the variable z to be equal to the value c_y. This constraint is commonly used in CLP models to look up cost associated with a discrete decision variable.

We model the configuration problem in CLP in the following way (assuming that $T_i = \{1, \ldots, n_i\}$):

$$
\begin{aligned}
\min \quad & \sum_{k \in R} c_k r_k \\
\text{s.t.} \quad & r_k = \sum_{i \in C} q_i a_{ki}, && \forall k \in R, && (5)\\
& \texttt{element}(t_i, [A_{ki1}, \ldots, A_{kin_i}], a_{ki}), && \forall i \in C, k \in R, && (6)\\
& r_k \geq R_k, && \forall k \in R, && (7)\\
& t_1 = 1 \Rightarrow t_2 \in \{1, 2\}, && && (8)\\
& q_1 > 0 \Rightarrow q_2 = 0, && && (9)\\
& t_i \in T_i,\ q_i \in \mathbb{Z}_+,\ a_{ki} \in \mathbb{Q}, && \forall i \in C, k \in R. &&
\end{aligned}
$$

where a_{ki} is the consumption of resource k by component i, and the other variables and constants as before. Note that (5) is non-linear, both q_i and a_{ki} are variables, and that there are $|C| \times |R|$ `element` constraints. Equations (8)–(9) are two examples of a logical side-constraints, to demonstrate below how the configuration constraints have to be modelled in different modelling paradigms.

3.2. A MIP Model

In a MIP model, the lack of variable subscripts and the linear requirement means that the component types t_i have to be replaced with 0–1 variables t_{ij}, where t_{ij} is 1 if component i has type j, 0 otherwise, see eq. (11). Similarly, the quantity of the i-th component, q_i, is disaggregated into q_{ij} for types $j \in T_i$ in eq. (13). The type selection and the quantity are then linked using eq. (12):

$$
\begin{aligned}
\min \quad & \sum_{k \in R} c_k r_k \\
\text{s.t.} \quad & r_k = \sum_{i \in C} \sum_{j \in T_i} q_{ij} A_{kij}, && \forall k \in R, && (10)\\
& \sum_{j \in T_i} t_{ij} = 1, && \forall i \in C, && (11)\\
& q_{ij} \leq M t_{ij}, && \forall i \in C, j \in T_i, && (12)\\
& q_i = \sum_{j \in T_i} q_{ij}, && \forall i \in C, && (13)\\
& r_k \geq R_k, && \forall k \in R, && (14)\\
& t_{21} + t_{22} \geq t_{11}, && && (15)\\
& b \leq q_1 \leq M b,\ q_2 \leq M(1 - b), && && (16)\\
& t_{ij} \in \{0, 1\},\ q_i, q_{ij} \in \mathbb{Z}_+,\ b \in \{0, 1\}, && \forall i \in C, j \in T_i. &&
\end{aligned}
$$

Equations (15)–(16) in this model formulate the configuration constraints from before.

3.3. An MLLP Model

The MLLP model is similar to the CLP model, although the underlying solution strategy employs LP relaxations in conjunction with the constraint propagation:

$$
\begin{aligned}
\min \quad & \sum_{k \in R} c_k r_k \\
\text{s.t.} \quad & r_k = \sum_{i \in C} q_i A_{k i t_i}, && \forall k \in R, && (17)\\
& r_k \geq R_k, && \forall k \in R, && (18)\\
& t_1 \in \{1\} \Rightarrow t_2 \in \{1,2\}, && && (19)\\
& b \in \{0\} \Rightarrow q_1 = 0,\ b \in \{1\} \Rightarrow (q_1 \geq 1, q_2 = 0), && && (20)\\
& t_i \in T_i,\ q_i \in \mathbb{Z}_+,\ b \in \{0,1\}, && \forall i \in C. &&
\end{aligned}
$$

Note that the resource consumption "lookup" is handled through a variable subscript on A, i.e., by A_{kit_i} where the variable t_i is the type of component i, as in the original (abstract) model, which is compiled into a special type of the `element` constraint. The next two sections describe the linear relaxations used for these variable subscripts, and how the relaxations link the discrete and continuous parts of the MLLP model. Propagation rules for these variable subscripts are presented in [15,20].

4. Linear Relaxation of c_y

The term $q_i \times A_{k,i,t_i}$ is the core of the configuration problem. Again, note that what makes it complex is that the index t_i is not a parameter, but a variable. Further complicating matter, the variable-subscripted constant A is multiplied by a another variable, q_i.

Eventually we seek a linear relaxation of a term of this type, subscripted constant multiplied by a variable ($c_y x$), but we will begin with a simpler linearisation, of a single subscripted constant (c_y). This would be the term that would occur in the MLLP model if it was limited to unit quantities, i.e., if $q_i \equiv 1$, and is equivalent to the traditional `element` constraint in CLP. A subscripted constant, c_y, occurring alone in a term is compiled as in the following example:

$$
\begin{gathered}
2x + c_y \leq 18, \\
c \in \{1.0, 4.5, 6.1\},
\end{gathered}
$$

$$\Updownarrow$$

$$2x + z \leq 18,$$
$$\texttt{element}(y, [1.0, 4.5, 6.1], z).$$

Bounds-consistency on the `element` constraint involves producing increasingly tighter bounds on z as the domain D_y of y is reduced. A straight-forward linearisation of this constraint is identical to how this structure is modelled in MIP, i.e., 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 add

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

$$\Updownarrow$$

$$0 \leq b_i \leq 1, \ \forall i, \tag{21}$$
$$b_1 + \cdots + b_k = 1, \tag{22}$$
$$z = c_1 b_1 + \cdots + c_k b_k. \tag{23}$$

to the LP. This linear relaxation is no stronger than bounds-consistency on the `element` constraint (actually equivalent to it). Thus there is little incentive to use this relaxation unless these linear constraints connect in a beneficial way to some other linear constraints, or useful information (dual values, reduced costs, etc.) can be derived and used by including them.

4.1. Linear Relaxation of Subscripts in Bounds

Variable subscripts can occur in several places in a model, including in constraints, in the objective or in simple bounds on variables. One can reap benefits from treating them slightly differently depending on the context and we will illustrate this here by discussing the case of variable subscripts in bounds. For variable subscripts that occur in bounds there is a variant of the `element` constraint that enforces only one bound of z.

Allowing variable subscripts in bounds expressions, where the variable is declared, declaratively amounts to posting the constraints

$$l_y \leq x \leq u_y \tag{24}$$

where l [u] is the lower [upper] bound vector and y a discrete variable. Procedurally, the variable subscripted bounds have two drawbacks. First, they are naïvely compiled into

$$z_1 \leq x, \quad x \leq z_2, \tag{25}$$
$$\texttt{element}(y, [l_1, \ldots, l_n], z_1), \tag{26}$$
$$\texttt{element}(y, [u_1, \ldots, u_n], z_2), \tag{27}$$

which introduces two new variables, two `element` constraints and two linear inequalities. Secondly, and a more refined trap, the values of x, z_1 and z_2 in a

continuous solution may be such that no value $i \in D_y$ satisfies (26)–(27), despite the fact that some value $i \in D_y$ together with x satisfies the original variable subscripted bounds (24). This happens in practice, e.g., if the values of z_1 and z_2 are different from all the values in the lists $l_1, \ldots, l_n$ and $u_1, \ldots, u_n$.

These two problems are avoided by compiling the bounds (24) directly into

$$\texttt{element}_{\leq}(y, [l_1, \ldots, l_n], x), \tag{28}$$

$$\texttt{element}_{\geq}(y, [u_1, \ldots, u_n], x), \tag{29}$$

where (28) [(29)] only performs lower [upper] bound propagation, i.e., instead of requiring the output variable (x in this case) to be equal to l_y [u_y] then it should be less than or equal to l_y [greater than or equal to u_y].

Compiling the bounds (24) directly into (28)–(29) creates no new variables and no new linear inequalities. Also, given a value for x in a continuous solution, if there is a value $i \in D_y$ such that the subscripted bounds (24) are satisfied then that same i satisfies (28)–(29).

This inequality variant of the `element` can be relaxed in a similar fashion to the equality case, with the '=' in eq. (23) changed to '$\leq$' or '$\geq$' as appropriate. Naturally, since y is shared between (28) and (29) only one set of $b_1 + \cdots + b_k$ are introduced in the relaxation.

5. Linear Relaxation of $c_y x$

We now turn our attention back to the basic structure (2) of the configuration problem. The term $c_y x$ is a subscripted constant multiplied by a continuous variable. In the configuration problem it denotes the cost of buying x units of type y at cost c_y each. In MLLP this structure is replaced by a continuous variable z when the model is compiled and the constraint $\texttt{element}(y, [c_1, \ldots, c_k] \times x, z)$ is introduced into the problem. For example

$$\begin{gathered} 2x + c_y x \leq 18, \\ c \in \{1.0, 4.5, 6.1\}, \end{gathered}$$

$$\Updownarrow$$

$$\begin{gathered} 2x + z \leq 18, \\ \texttt{element}(y, [1.0, 4.5, 6.1] \times x, z). \end{gathered}$$

When propagating this variant of the `element` constraint upon a change of D_y, the domain of y, cuts of the form

$$c_{\min} x \leq z \qquad \text{and} \qquad c_{\max} x \geq z \tag{30}$$

are updated, where $c_{\min} = \min\{c_i : i \in D_y\}$ and $c_{\max} = \max\{c_i : i \in D_y\}$.

This is compact, but a stronger relaxation of $\texttt{element}(y, [c_1, \ldots, c_k] \times x, z)$ is achieved by disaggregating x into bins x_i, and linking to the corresponding decision variable z:

$$0 \leq b_i \leq 1, \qquad \forall i, \tag{31}$$
$$b_1 + \cdots + b_k = 1, \tag{32}$$
$$x = x_1 + \cdots + x_k, \tag{33}$$
$$z = c_1 x_1 + \cdots + c_k x_k, \tag{34}$$
$$x_i \geq 0, \qquad \forall i, \tag{35}$$
$$x_i \leq M b_i, \qquad \forall i. \tag{36}$$

Equations (31)–(32) are the same as for c_y above, but in addition, x is disaggregated to $x_1, \ldots, x_k$, and connected through the big-M constraints (36) to $b_1, \ldots, b_k$. Intuitively, this relaxation is stronger than bounds-consistency simply because all the coefficients are weighed into the LP relaxation. As we shall see in Sec. 7, this makes a big difference in computational efficiency.

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 (36) is only to ensure that at most one of $x_1, \ldots, x_k$ is non-zero. This is unnecessary in our framework since this is implicitly enforced by the connection to y. As an alternative to this intuitive explanation, consider deriving the convex hull formulation of this constraint from the general disjunctive formulation [1]. The disjunction equivalent of $\texttt{element}(y, [c_1, \ldots, c_k] \times x, z)$ is

$$\bigvee_{i \in D_y} (-z + c_i x = 0), \tag{37}$$

and the general disjunctive formulation a) applied to (37) gives b) since $\alpha = 0$, which then simplifies to c), which are equations (33)–(34) above.

$$\begin{array}{lll}
\bigvee a_i x \leq \alpha, & & \\
\quad \Updownarrow & (z, x) = \sum_i (z_i, x_i), & \\
x = \sum_i x_i, & & \\
a_i x_i \leq \alpha b_i, \quad \forall i, & -z_i + c_i x_i \leq 0, \quad \forall i, & z = \sum_i c_i x_i, \\
\sum_i b_i = 1. & z_i - c_i x_i \leq 0, \quad \forall i, & \\
 & \sum_i b_i = 1. & x = \sum_i x_i. \\
\text{a)} & \text{b)} & \text{c)}
\end{array}$$

Thus, we can dispose of (31), (32) and (36), given that upon domain reduction $i \notin D_y$ of y, we enforce $x_i = 0$. This is similar to the modelling and branching with Special Ordered Sets (SOS) of type 1 in MIP [2,6].

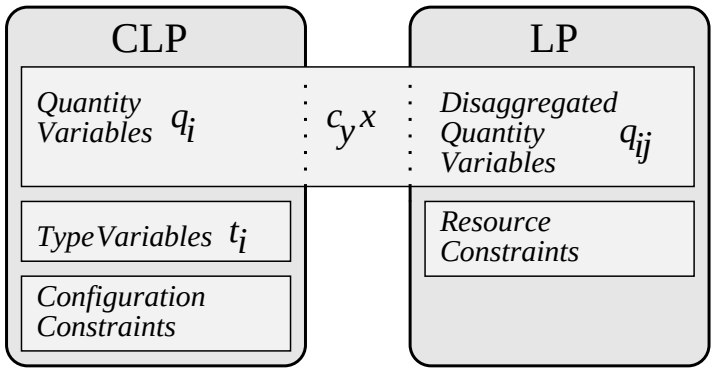

Figure 1. The build-up of the MLLP model.

This tight relaxation, which computational tests (Sec. 7) show is an invaluable tool, links the discrete and continuous parts of the MLLP model (Sec. 3.3), The continuous (linear) part includes the resource constraints (17), with $q_i A_{kit_i}$ linearised as detailed above.[2] Figure 1 illustrates the effect of this. The quantities are present in both the CLP and LP parts, although in different forms. CLP has q_i, whereas the linearisation introduces the q_{ij} variables into the LP. The component types are only present in the CLP part, because as proven above, they add nothing to the relaxation and are therefore not included.

6. Reduced costs

Using reduced costs for inference in optimisation methods is standard practice in OR and is available in several MIP solvers, e.g., CPLEX [16] and XPRESS MP [5]. We will only give a brief description of reduced costs here, for more details see [4,18]. In a basic solution to the Linear Program (LP),

$$\begin{aligned} \min \quad & z = cx \\ \text{s.t.} \quad & Ax = b,\ x \geq 0, \end{aligned}$$

the variables are partitioned into *basic* and *non-basic* variables [4], call them x_B and x_N, respectively. If we partition the constraint matrix $A = [B\ N]$ and the objective function $c = [c_B\ c_N]$ in the same manner, we can write the problem above as

$$\begin{aligned} \min \quad & z = c_B x_B + c_N x_N \\ \text{s.t.} \quad & Bx_B + Nx_N = b, \\ & x_B, x_N \geq 0. \end{aligned} \tag{38}$$

[2] To obtain the linearisation for the MLLP model replace x with q_i, y with t_i and c with A_{kit_i} in the discussion in this section.

The solution to equation (38) is $x_B = B^{-1}b - B^{-1}Nx_N$. In a *basic solution*, x_N will be 0 and therefore $x_B = B^{-1}b$. We note that the value of the objective function will then be

$$\begin{aligned} z = c_B x_B + c_N x_N &= c_B(B^{-1}b - B^{-1}Nx_N) + c_N x_N \\ &= c_B B^{-1}b + \underbrace{(c_N - B^{-1}N)}_{\bar{c}} x_N = c_B B^{-1}b. \end{aligned}$$

The reduced costs, $\bar{c}$, are defined for the non-basic variables $x_N = \{x_{i_1}, \ldots, x_{i_k}\}$ (which are all zero in the basic solution). The reduced cost $\bar{c}_{i_j}$ then corresponds to the *cost per unit increase* to the objective function value if x_{i_j} would take on a non-zero value. The basic solution is an *optimal basic solution* if all the reduced costs are non-negative, indicating that there is no benefit in having any of the non-basic variables taking on non-zero values.

A process called *reduced cost fixing* [25] uses the reduced costs of relaxed 0–1 variables in MIP problems, obtained from the optimal linear programming relaxation solution $\bar{x}$, to potentially fix variables to zero without having to branch on them. This can be done for a non-basic variable x_{i_j} if $c\bar{x} + \bar{c}_{i_j} > cx^*$, i.e., if the current objective value plus the reduced cost $\bar{c}_{i_j}$ of the variable x_{i_j} exceeds the objective value of the incumbent solution x^*, the best solution found so far in the branch-and-bound search. This generalises to variables at their lower *or* upper bound, and to general integer variables.

6.1. Reduced-Cost Based Propagation in CLP

Recently, Focacci et. al. [7,9,10] have adapted variable fixing to a CLP framework and used it successfully for the domain pruning of the `alldiff` constraint and the `path` constraint in ILOG Solver. This is done by shadowing these global constraints with a linear formulation of the assignment problem [25] and solving it to optimality while computing the corresponding reduced costs. This is done incrementally using the Hungarian algorithm, but it could also be done with linear programming. In the assignment problem, the variable $x_{ij} = 1$ corresponds to $x_i = j$ in $\texttt{alldiff}(x_1, \ldots, x_k)$. As before, if $c\bar{x} + \bar{c}_{ij} > cx^*$ for a non-basic variable x_{ij} and incumbent solution x^*, then $x_i \neq j$. This rule is used within a standard fix-point propagation loop and can thus interact and communicate with other constraints. From a CLP 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.

Note that the assignment problem is completely separate there from the rest of the model; it is only used in the propagation loop. The assignment problem is also *totally unimodular* [18], which means that the optimal solution to the linear relaxation will always be integral. This will not be the case for general hybrid

CLP–MIP models, but the reduced costs will still be well-defined for non-basic variables.

6.2. Reduced-Cost Based Propagation in MLLP

In a hybrid CLP–MIP solver, such as MLLP, where the inference of CLP is combined with a linear relaxation, we can also use reduced-cost based propagation, given that our constraints are linearised. The reduced cost propagation then becomes a part of the regular fixed-point propagation loop, which all the discrete and mixed constraints of MLLP are a part of.

Again we start with the simple subscripted constant, c_y (Sec. 4). Assume the current optimal LP solution is $\bar{x}$, with reduced costs $\bar{c}_1, \ldots, \bar{c}_k$ for the decision variables $b_1, \ldots, b_k$, and x^* is the incumbent MLLP solution. Then the following rule defines reduced-cost based domain reduction for the `element` constraint (assume minimisation), if linearised by (21)–(23):

$$\begin{array}{l}\texttt{do } \forall i \in D_y \\ \quad \texttt{if } c\bar{x} + \bar{c}_i \geq cx^* \texttt{ then } D_y = D_y - \{i\}\end{array}$$

This domain pruning is equivalent to standard constraint propagation, i.e., valid in this subproblem and all its extensions (this node and below in the search tree). As we noted earlier, remember that the linearisation of c_y is no stronger than simple bounds-consistency, but that it allows us to do this reduced-cost based propagation, which may be beneficial in some cases.

Returning to our configuration problem, a similar reduced-cost based propagation rule for the term $c_y x$ (Sec. 5) can be used on the linearisation of (31)–(36).

$$\begin{array}{l}\texttt{do } \forall i \in D_y \\ \quad \texttt{if } c\bar{x} + \bar{d}_i \geq cx^* \texttt{ then } D_y = D_y - \{i\} \\ \texttt{do } \forall i \in D_y \\ \quad x \leq \lfloor (cx^* - c\bar{x})/\bar{d}_i \rfloor\end{array}$$

where $\bar{d}_1, \ldots, \bar{d}_k$ are the reduced costs for $x_1, \ldots, x_k$, respectively. The last line bounds the quantity variable; the standard variable fixing in MIP, which is also applicable in MLLP.

7. Computational Testing

The first interesting object of study is to compare how CLP, MIP and MLLP perform on the configuration problem introduced earlier. Figures 2–3 show average number of nodes and average time[3] required to solve to proven optimality instances of different sizes (note the logarithmic scale). The first, easily solved,

[3] Sun Ultra 60 Model 2360 (2×360 MHz UltraSPARC-II) running Solaris 7.

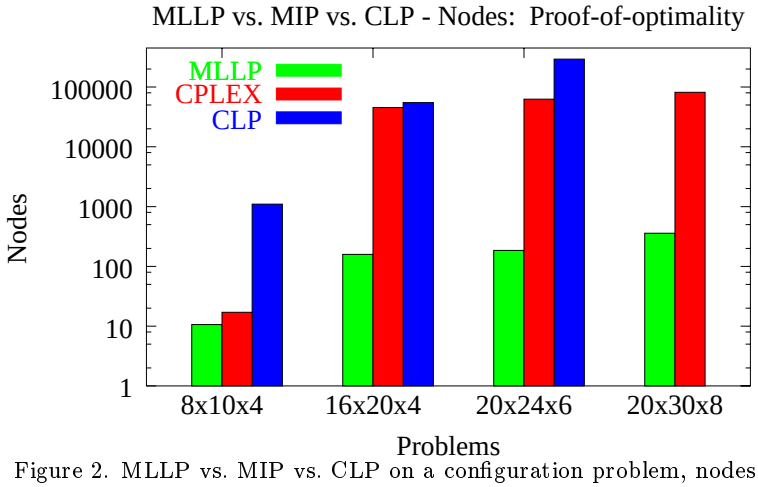

Figure 2. MLLP vs. MIP vs. CLP on a configuration problem, nodes.

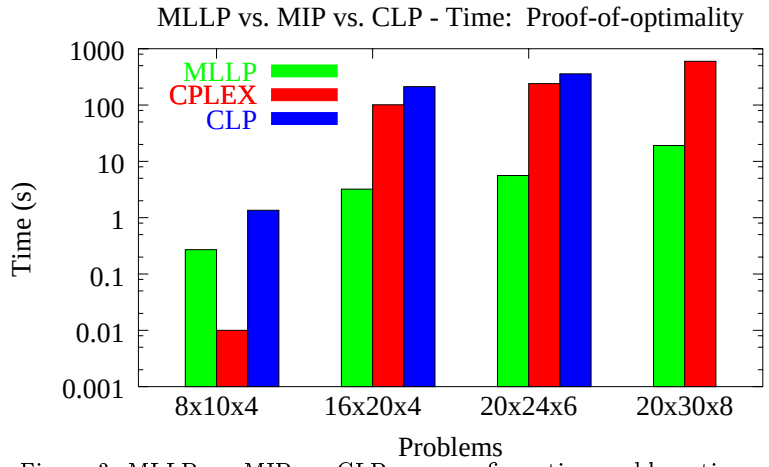

Figure 3. MLLP vs. MIP vs. CLP on a configuration problem, time.

class is a set of seven instances with real-world data; the following classes have ten instances each, with data generated to closely resemble the original instances.

For size '16x20' (16 components, 20 types), CPLEX solves 8 of the 10 instances to proven optimality within 100 000 nodes, at which point the search is aborted. For size '20x24' it only solves 4 of the 10 instances and for size '26x30' it only solves 3 of the 10 instances. SICStus Prolog [3] solves 6 instances of size '16x20' to proven optimality within 500 sec. (50 000–100 000 nodes), 7 instances of size '20x24' on and no '26x30' instances. MLLP solves all instances of all sizes, a magnitude, or more, faster. It should be emphasised that MLLP uses the relaxation of $c_y x$ described in Sec. 5, without which MLLP performs much worse. It also uses the reduced-cost based propagation.

The second method we want to study is the reduced-cost based propagation. It has been shown to be clearly beneficial in a CLP context [7,9,10], where it adds

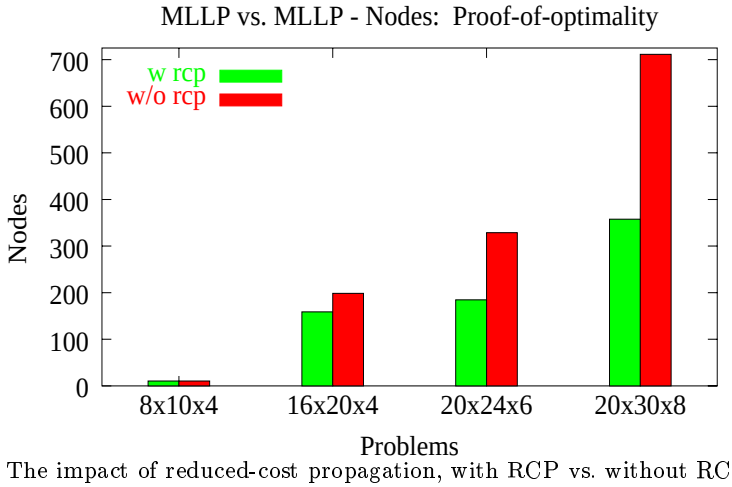

Figure 4. The impact of reduced-cost propagation, with RCP vs. without RCP, nodes.

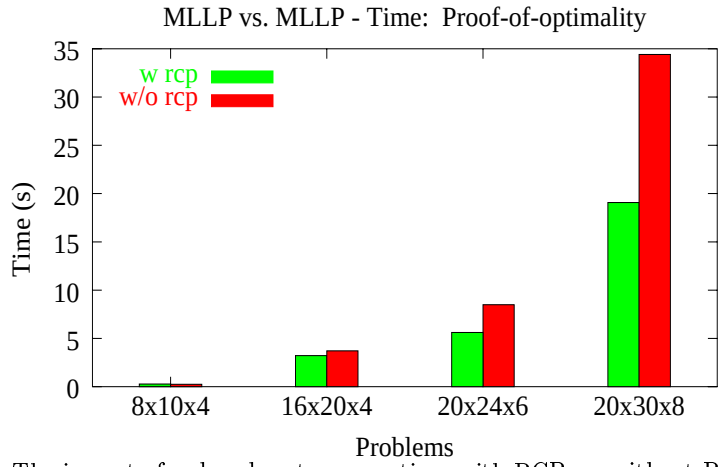

Figure 5. The impact of reduced-cost propagation, with RCP vs. without RCP, time.

optimisation oriented domain reduction, but is it equally effective in a framework like MLLP that is already focusing the search around the relaxation optimum? Figures 4–5 show the effect on the number of nodes and the time required to solve to proven optimality. The reduction in nodes is significant, around 10-50% is saved (higher percentage as the problems grow larger), and the overhead is not considerable so the CPU time saved is almost as much.

8. Conclusion

In this paper, we presented and compared CLP, MIP and hybrid MLLP models for a configuration problem. For the hybrid MLLP model, we showed how an extended variant of the `element` constraint, which is central to the problem,

can be linearised and how the reduced costs from the LP relaxation can be used for domain pruning. Computational results were included, comparing different models and techniques on a set of real-world instances. The results showed that the linear relaxation introduced is an invaluable tool and that the reduced cost propagation also contributes to an efficient solution. The hybrid MLLP approach was shown to out-perform both pure CLP and MIP, and was able to solve instances larger than the other two methods could handle.

Acknowledgements

We would like to thanks Tacton Systems [19,23] for providing the configuration problem and initial data, Mats Carlsson, Swedish Institute of Computer Science (SICS), for reading drafts of this paper, and Prof. John Hooker, Carnegie Mellon University (CMU), for general guidance and inspiration.

References

[1] E. Balas. Disjunctive programming. In P. L. Hammer, E. L. Johnson, and B. H. Korte, editors, *Discrete Optimization II, 5*, pages 3–51, Amsterdam, 1979. Annals of Discrete Mathematics, North-Holland.

[2] E.M.L. Beale and J.A. Tomlin. Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables. In John Lawrence, editor, *Proceeding of the Fifth International Conference on Operation Research*. Tavistock Publications, 1970.

[3] Mats Carlsson et al. SICStus Prolog User's Manual. SICS research report, Swedish Institute of Computer Science, 1995. URL: http://www.sics.se/sicstus.

[4] Vašek Chvátal. *Linear Programming*. W. H. Freeman and Company, New York, 1983.

[5] Dash Optimization, Inc. *XPRESS–MP: User Manuals*, 2000. Version 12.11.

[6] I. R. de Farias, E. L. Johnson, and G. L. Nemhauser. A branch-and-cut approach without binary variables to combinatorial optimization problems with continuous variables and combinatorial constraints. *Knowledge Engineering Review, special issue on AI/OR, submitted*, 1999.

[7] Filippo Focacci, Andrea Lodi, and Michela Milano. Cost-based domain filtering. In Joxan Jaffar, editor, *Principles and Practice of Constraint Programming*, volume 1713 of *Lecture Notes in Computer Science*. Springer, October 1999.

[8] Filippo Focacci, Andrea Lodi, and Michela Milano. Integration of CP and OR methods for matching problems. In *CP-AI-OR'99 Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems*, Feb 1999.

[9] Filippo Focacci, Andrea Lodi, and Michela Milano. Solving TSP with time windows with constraints. In *Sixteenth International Conference on Logic Programming*, November 1999.

[10] Filippo Focacci, Andrea Lodi, Michela Milano, and Danielo Vigo. Solving TSP through the integration of OR and CP techniques. In *CP98 Workshop on Large Scale Combinatorial Optimisation and Constraints*, October 1998.

[11] J. N. Hooker. Logic-based methods for optimization. In Alan Borning, editor, *Principles and Practice of Constraint Programming*, volume 874 of *Lecture Notes in Computer Science*. Springer, May 1994. (PPCP'94: Second International Workshop, Orcas Island, Seattle, USA).

[12] J. N. Hooker and M. A. Osorio. Mixed logical/linear programming. *Discrete Applied Mathematics*, 96–97(1–3):395–442, 1999.

[13] John N. Hooker, Hak-Jin Kim, and Greger Ottosson. A declarative modeling framework that integrates solution methods. *Annals of Operations Research, Special Issue on Modeling Languages and Approaches, to appear*, 1998.

[14] John N. Hooker, Greger Ottosson, Erlendur S. Thorsteinsson, and Hak-Jin Kim. On integrating constraint propagation and linear programming for combinatorial optimization. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pages 136–141. AAAI, The AAAI Press/The MIT Press, July 1999.

[15] John N. Hooker, Greger Ottosson, Erlendur S. Thorsteinsson, and Hak-Jin Kim. A scheme for unifying optimization and constraint satisfaction methods. *Knowledge Engineering Review, Special Issue on Artifical Intelligence and Operations Research*, 15(1):11–30, 2000.

[16] ILOG. *ILOG CPLEX Reference Manual*, 2001. Version 7.0.

[17] 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(2):263–277, 1994.

[18] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley, New York, 1988.

[19] Klas Orsvärn and Tomas Axling. The Tacton view of configuration tasks and engines. In *Workshop on Configuration, Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, 1999. www.tacton.com, Technical Report WS-99-05.

[20] Greger Ottosson, Erlendur S. Thorsteinsson, and John N. Hooker. 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*, March 2001. Accepted for publication.

[21] Philippe Refalo. Linear formulation of constraint programming models. In *CP-AI-OR'00 Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems*, March 2000.

[22] Robert Rodošek, Mark Wallace, and Mozafar Hajian. A new approach to integrating mixed integer programming and constraint logic programming. *Annals of Operations Research, Advances in Combinatorial Optimization*, 86:63–87, 1999.

[23] Tacton Systems. *Tacton Configurator User's Manual*, 1999. www.tacton.com.

[24] H.P. Williams and Hong Yan. Representations of the all_different predicate of constraint satisfaction in integer programming. *INFORMS Journal on Computing*, 2001. To appear.

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