University of Leeds

SCHOOL OF COMPUTER STUDIES

RESEARCH REPORT SERIES

Report 97.16

ILP and Constraint Programming Approaches to a Template Design Problem

by

Les Proll & Barbara Smith

May 1997

Abstract

We describe a design problem arising in the colour printing industry and discuss a number of integer linear programming and constraint programming approaches to its solution. Despite the apparent simplicity of the problem it presents a challenge for both approaches. We present results for three typical cases and show that the constraint programming approach provides better results, although in some cases after considerable handcrafting. We also show that the results obtained by constraint programming can be improved by a simple goal programming model.

1 Introduction

The problem we consider in this paper arises from a local colour printing firm which produces a variety of products from thin board, including cartons for human and animal food and magazine inserts. Food products, for example, are often marketed as a basic brand with several variations (typically flavours). Packaging for such variations usually has the same overall design, in particular the same size and shape, but differs in a small proportion of the text displayed and/or in colour. Two variations of a cat food carton may differ only in that on one is printed 'Chicken Flavour' on a blue background whereas the other has 'Rabbit Flavour' printed on a green background. In the case of magazine inserts, the difference may simply be a small code number on the reply paid section of the insert which allows the client to identify from which magazine the reply came and thus to improve targetting of future advertising. A typical order is for a variety of quantities of several design variations. Because each variation is identical in dimension, we know in advance exactly how many items can be printed on each mother sheet of board, whose dimensions are largely determined by the dimensions of the printing machinery. Each mother sheet is printed from a template, consisting of a thin aluminium sheet on which the design for several of the variations is etched. The problem is to decide how many distinct templates to produce and which variations, and how many copies of each, to include on each template.

The following example is based on data from an order for cartons for different varieties of dry cat-food.

Variation	**Order Quantity**
Liver	250,000
Rabbit	255,000
Tuna	260,000
Chicken Twin	500,000
Pilchard Twin	500,000
Chicken	800,000
Pilchard	1,100,000
Total	3,665,000

Each design of carton is made from an identically sized and shaped piece of board, shown in Figure 1. Nine cartons can be printed on each mother sheet.

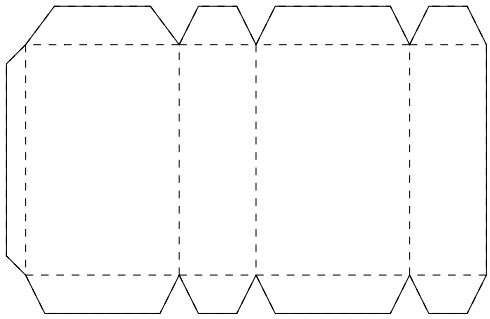

Figure 1: 2-dim. net for a catfood carton

Because in this example there are more slots in each template (9) than there are variations (7), it would be possible to fulfil the order using just one template. The optimum way of doing this is to allocate two slots to the two largest orders (Chicken and Pilchard), one each to the rest, and print 550,000 sheets of card, as in Figure 2.

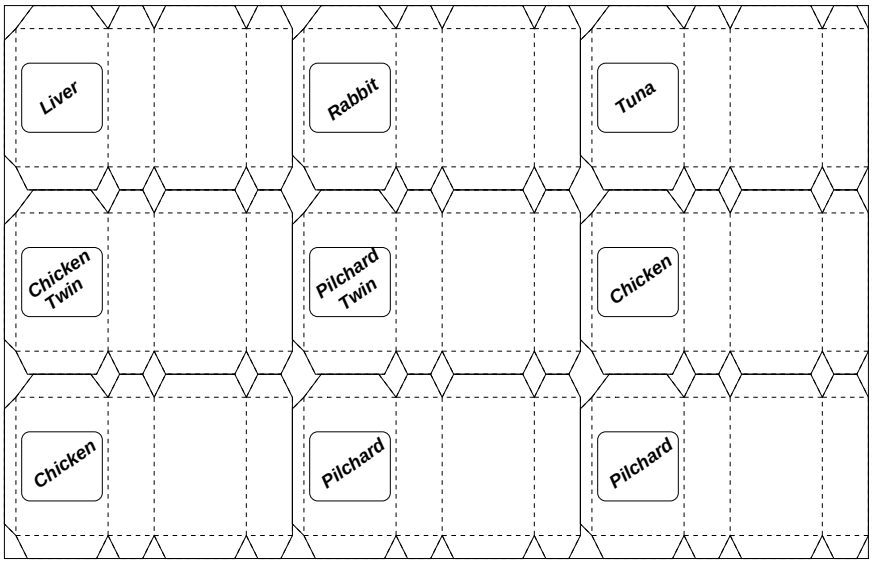

Figure 2: Optimal one-template solution for the catfood data

This would create an enormous amount of waste, however: 4,950,000 cartons would be produced, which is 35% more than the total order quantity. On the other hand, it minimises the cost of producing the templates.

At the other extreme, we could produce one template for each variation; this would result in virtually no over-production, but would be expensive in terms of template production. (In fact, even if we wanted to minimise over-production by itself, this solution is wasteful in requiring unnecessary templates, since it is normally possible to find a solution with no waste production with far fewer templates than the number of variations.)

The problem was originally posed to us as one of designing the minimum number of distinct templates to satisfy the order to within a specified tolerance of the order

quantity for each variation. Although the production cost of a template may not be great in absolute terms, the industry is a highly competitive one and small differences to the cost per carton can make a difference to the company's competitive edge.

We describe both integer linear programming and constraint programming approaches to the template design problem. We compare the two approaches and present results obtained for three typical real cases: the catfood problem specified above, and those listed in Table 1.

Problem	Slots per Template	Variations	Order Quantities (/1000)
Herb cartons	42	30	60,60,70,70,70,70,70,70,70,80, 80,80,80,90,90,90,90,90,90,100, 100,100,100,150,230,230,230,230,280,280
Magazine inserts	40	50	50,53,55,60,85,90,100,100,105,110, 137,140,140,140,150,150,150,150,150,150, 150,150,168,170,170,195,195,200,200,200, 210,210,225,230,230,230,250,250,250,250, 250,250,250,250,265,270,270,375,375,405

Table 1:

The results suggest that the problem is more difficult to solve in general than might at first appear.

2 ILP Approaches

At first sight, the template design problem has some similarity to the cutting stock problem [1] and similar formulations to that might be expected to be fruitful. Thus we can formulate the template design problem as an ILP in which we select templates from a predetermined list of possible templates as follows:

Let Q_i = order quantity for variation i
l_i = lower quantity (fractional) tolerance for variation i
u_i = upper quantity (fractional) tolerance for variation i
p_{ij} = number of slots in template j in which variation i appears
T_j = 1 if template j is used, = 0 if template j is not used
R_j = number of pressings of template j,
i.e. the number of mother sheets printed with template j

Then we have

$$\begin{aligned} \text{MIN} \quad & \sum T_j && (1)\\ s.t. \quad & (1-l_i)Q_i \;\le\; \sum p_{ij}R_j \;\le\; (1+u_i)Q_i \;\forall i && (2)\\ & R_j \;\le\; M_jT_j \;\forall j && (3)\\ & R_j \;\ge\; 0,\; T_j \;=\; 0 \text{ or } 1 \end{aligned}$$

(2) ensures that the order quantities for each variation are met within the tolerances. (3) models the logical implication $T_j = 0 \Rightarrow R_j = 0$ provided M_j is large enough [9]. Although nominally the variables R_j should be integer valued, there is no difficulty in allowing them to assume fractional values given their expected magnitude. We refer to this model as the global model. It requires enumeration of all possible templates, which is combinatorially explosive. Indeed the enumeration of templates is even more of a problem than is the enumeration of cutting patterns in the cutting stock problem since, because all variations are of the same size, any allocation of slots to variations which fills the template is feasible. Similarly development of a restricted global model, in which only a pre-selected 'reasonable' subset of the possible templates is considered, seems unpromising since it is difficult to determine selection criteria local to individual templates. However, we return to this model later.

Our next attempt is a sequential approach in which a local ILP model generates one template on each run. It is based on the recognition that if a small number of templates is used to satisfy the order, each of the templates is likely to have a large number of pressings. Thus we attempt to find templates with the maximum number of pressings consistent with demand, using the model:

$$\begin{aligned} \text{MIN} \quad & W && (4)\\ s.t. \quad & \sum p_i = S && (5)\\ & p_i \;\le\; (1+u_i)Q_iW \;\forall i && (6)\\ & W \;\ge\; 0,\; p_i \;\ge\; 0 \text{ and integer} \end{aligned}$$

(5) ensures that the template is full, (6) that the upper limit on quantity produced is respected.

where S = no. of slots in a template,
W = $1/R$ (R = no. of pressings of the template),
p_i = no. of slots occupied by variation i
$Q_i,\ u_i$ are as above.

Suppose the first run of the model results in the template

$$[p_1^1,\ p_2^1,\ \ldots,\ p_n^1]$$

The residual demand for the variations is

$$Q_i - \lfloor R \rfloor \times p_i^1$$

Any variation for which

$$Q_i - \lfloor R \rfloor \times p_i^1 \ \leq \ l_i Q_i$$

is regarded as complete.

Second and further runs of the model are now made considering only the incomplete variations and their residual demands. The process stops when all variations are complete. For the catfood problem, using a 10% order tolerance for all variations, this approach results in the template shown in Table 2. This solution gives an overall

Template	Pressings
$[0,0,0,1,1,3,4]$	266,000
$[2,2,2,2,1,0,0]$	117,000
$[0,0,0,0,9,0,0]$	13,000

Table 2: Templates for catfood problem using ILP

deviation from demand of -2.76% with a maximum deviation per variation of -10%. The solution can be improved by a variety of simple goal programming models [3]. For example we can seek a solution, using the templates generated by the sequential approach, which gives minimum total deviation from demand whilst remaining within the individual order quantity tolerances, using:

$$\begin{aligned}
\text{MIN} \quad & \sum U_i + O_i && (7)\\
s.t. \quad & \sum p_{ij} R_j + U_i - O_i = Q_i && (8)\\
& \sum p_{ij} R_j \geq (1 - l(i)) Q_i && (9)\\
& \sum p_{ij} R_j \leq (1 + u(i)) Q_i && (10)\\
& R_j, \ U_i, \ O_i \geq 0
\end{aligned}$$

where U_i = underproduction of variation i,
O_i = overproduction of variation i,
R_j = no. of pressings of template j,
$Q_i, \ l_i, \ u_i, \ p_{ij}$ are as defined previously

(9) and (10) ensure that the quantity produced of each variation stays within the allowed tolerances, (8) defines U_i, O_i. The models involved in the above approach are small and solvable in trivial time with a commercial ILP package such as XPRESS-MP [2], the one used in this study. Following the goal programming step, the sequential ILP approach gives the results in Table 3.

Although for the purposes of this study, the sequential approach was performed with manual intervention, it could easily be automated, either by using the ILP solver as a callable subroutine or by using the ability to connect to a spreadsheet. The small size of the goal program in particular makes it possible to generate a variety of possible solutions with the templates generated by the sequential approach by varying the tolerances. In particular we can set l_i to 0 to generate a solution with no underproduction.

Problem	No. of Templates	Templates	Pressings	Overall % Deviation	Maximum % Deviation
Catfood cartons	3	[0,1,1,1,1,2,3] [2,0,0,2,2,1,2] [0,0,0,0,0,9,0]	260,000 120,000 17,778	-2.59	-7.27
Herb cartons	3	[0,0,1,1,1,1,1,1,1,1, 1,1,1,1,1,1,1,1,1,1, 1,1,1,2,3,3,3,3,3,4] [5,5,0,0,0,0,0,0,0,1, 1,1,1,2,2,2,2,2,2,3, 1,3,3,0,0,0,0,0,6,0] [10,10,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0, 22,0,0,0,0,0,0,0,0,0]	70,000 10,000 909	-2.91	-8.70
Magazine inserts	4	[1,0,0,1,0,0,1,0,1,1, 1,1,1,0,1,1,1,0,1,1, 1,1,1,1,0,1,0,1,1,1, 0,1,1,1,0,1,1,1,2,1, 1,1,1,0,2,2,1,1,0,1] [1,1,4,2,1,1,1,2,1,0, 1,1,0,1,1,1,0,0,1,0, 1,0,1,1,1,0,0,0,0,0, 1,1,1,0,0,0,0,0,2,1, 0,1,2,2,1,1,1,0,1,1] [2,1,0,0,1,1,0,1,2,0, 2,2,1,1,1,1,0,2,1,0, 2,0,2,0,1,2,2,0,0,0, 2,0,2,0,1,1,0,2,0,1, 1,2,0,0,0,0,0,0,0,0] [0,0,0,0,7,0,0,0,0,0, 0,0,0,8,0,0,0,0,0,0, 0,0,0,0,7,0,0,0,0,0, 0,0,0,0,11,0,0,0,0,0, 0,0,0,7,0,0,0,0,0,0]	145,000 60,250 24,750 2295	-2.49	-10.00

Table 3: Results using the Sequential ILP Approach

At this stage the problem owner felt that the solutions obtained with the simple sequential approach were capable of improvement and, in particular, was confident that the catfood problem was capable of solution with only two templates. As an attempt to test this assertion we used a form of the restricted global model in which the template population was generated by multiple runs of the sequential approach. At each stage, instead of using only the optimal template provided by the previous stage to derive the residual demand, we use each of the best (up to 10) templates found during the branch and bound search as a starting point. Each of these generates several more templates so that a population of templates is generated in a tree-like fashion. Given this population of templates we attempt to find a solution using a

fixed number, K, of templates using the ILP model:

$$\begin{aligned}
\text{MIN} \quad & \sum U_i + O_i && (11)\\
s.t. \quad & \sum p_{ij} R_j + O_i - U_i = Q_i && (12)\\
& R_j \leq M_j T_j \ \forall j && (13)\\
& \sum T_j = K && (14)\\
& \sum_{p_{ij}>0} T(j) \geq 1 \ \forall i && (15)
\end{aligned}$$

where T_j, U_i, O_i, R_j, Q_i, p_{ij}, M_j are as previously defined. As before (13) models the logical implication $T_j = 0 \Rightarrow R_j = 0$. (14) fixes the number of templates. (15) is a cut ensuring that every variation appears in at least one selected template. This proved useful in strengthening the formulation. With this procedure and the addition of the 'unit' templates, i.e. templates of the form:

$$[0, 0, \ ..., S, \ ..., 0]$$

we generated a population of 90 templates for the catfood problem from which we were able to establish that a two template solution with an overall deviation from demand of 3.25% was possible. However, it may be noted that the restricted global model does not include constraints on individual deviation from demand and that this result required 23.25% overproduction of the Rabbit variation. When we attempted to incorporate constraints on individual deviation, no two template solution could be found. This approach was not pursued further as it required a substantial amount of manual intervention and it seemed plausible that a constraint programming approach might be more fruitful.

3 Constraint Programming Approaches

In this section we briefly introduce the concepts of constraint satisfaction and constraint programming, before discussing their application to the template design problem. For a description of constraint satisfaction problems and algorithms for solving them, see [7].

A constraint satisfaction problem (CSP) consists of:

- a set of *variables* X = $\{x_1, ..., x_n\}$;
- for each variable x_i, a finite set D_i of possible values (its *domain*);
- and a set of *constraints* restricting the values that the variables can simultaneously take.

In contrast to ILP, the constraints of a CSP need not be linear, but can be any relation on the domains of the subset of variables affected by the constraint. A solution to

a CSP is an assignment of a value from its domain to every variable, in such a way that every constraint is satisfied.

Constraint programming(CP) tools such as CHIP [8] and ILOG Solver [5] provide straightforward means of defining the variables, domains and constraints of a CSP, together with algorithms for solving them. In this study, we use ILOG Solver. The algorithms provided in constraint programming tools are enumeration methods using backtracking search, combined with *constraint propagation*. Constraint propagation uses the values already assigned (if only temporarily) to variables during search in order to prune values from the domains of not-yet-assigned variables which can now no longer be chosen. The importance of constraint propagation in reducing search means that constraints play a very different role in CP than in ILP. As will be seen, it is often beneficial to add constraints to a basic model in order to facilitate constraint propagation and so allow the search algorithms to find a solution more quickly.

Although algorithms for solving CSPs are aimed at finding a feasible solution, they can be adapted to finding an optimal solution. For instance, a constrained variable can be created to represent the objective function. An initial solution is found and then a new constraint is introduced that the value of the objective variable must be better than in the initial solution. This is done repeatedly, tightening the constraint on the objective variable as each solution is found until the problem becomes insoluble: the last solution found is then the optimal solution.

3.1 The Template Design Problem as a CSP

The obvious advantage of a constraint programming approach to the template design problem arises from the fact that the quantity produced of variation i is $\sum_j p_{ij} R_j$. Since both p_{ij} and R_j are decision variables, we have an inherently nonlinear problem. We have to find means of linearising the problem to employ the ILP approaches. However, in a constraint programming approach we can handle the nonlinearity directly.

By the time that we began to investigate this approach, we had unfortunately lost contact with the original problem owner. However, we continued the investigation because of the tantalising difficulty of the problem. We also re-interpreted the problem as one of meeting minimum requirements for each variation while minimizing wastage, i.e. minimizing the total number of sheets printed, for a given number of templates. Although minimizing wastage was not part of the original problem description, clearly if two plans were produced, both meeting the same order quantity tolerances with the same number of templates, the one with less wastage would be preferable.

For each set of data, we have attempted to find an optimal solution, as just defined, for one template, two templates and so on, until a solution is found which meets the specified upper and lower tolerances on the order quantities for each variation. Adding more templates at that point would not result in any further improvement.

If an upper tolerance on the order quantities is specified, there may not be a feasible solution with a given number of templates; in those cases we have found the best solution with unlimited over-production allowed, to give an upper bound to the cost,

i.e. the number of sheets to be printed, if another template is added. (Although the minimum cost with t templates and unlimited over-production is not provably higher than the minimum cost with $t+1$ templates and limited over-production, in practice it is.) In particular, we use a one-template solution, which is trivial to compute, to give a good starting point for the search for a two-template solution.

3.2 The Basic Formulation

The following are the principal variables:

p_{ij} = number of slots allocated to variation i in template j
R_j = the run length, i.e. the number of pressings, for template j
$Production = \sum_j R_j$ i.e. the total number of pressings.

The following constants also appear in the model:

$S =$	number of slots in each template
$t =$	number of templates
$n =$	number of variations
$d_i =$	minimum production quantity for variation i $(= (1 - l_i)Q_i)$
$l =$	current upper limit on total production
$\underline{l} =$	minimum number of sheets which will meet the total demand, i.e. $\left\lceil \frac{\sum_i d_i}{S} \right\rceil$

l is initially set by the user. For the one-template problem, a suitable value is $2 \times \underline{l}$. For the t-template problem it should be set to one less than the optimal value of *Production* in the $(t-1)$-template problem. d_i, R_j, l, $\underline{l}$ are all in suitable units, e.g. 1000's.

The constraints are:

$$\sum_j p_{ij} = S \text{ for } j = 1, 2, ..., t \tag{16}$$

$$\sum_i p_{ij} R_j \geq d_i \text{ for } i = 1, 2, ..., n \tag{17}$$

The objective is to minimize *Production*; Solver's built-in minimization function `IlcMinimize` is used. This function first searches for a solution with *Production* $\leq l$; the solution is output, and if in this solution *Production* $= l'$, say, a new solution is sought satisfying *Production* $\leq l' - 1$. This process is repeated until the problem becomes infeasible; the last solution found must then be optimal.

3.3 Solution Strategy

Although, in theory, the formulation just described is sufficient to define the problem, and the search algorithms provided in Solver should be able to find an optimal solution to it, in practice this would take far too long for all but very small problems. In order to find an optimal solution, or even a good solution, in a reasonable time, it is necessary to add further constraints and to think carefully about the decisions that the search algorithm makes. This is usually true when solving complex problems using constraint programming; in this problem the number of additional constraints which could be devised, and which were necessary for solving the larger instances, was unusual.

We describe here the final model which we developed, although many of these refinements were unnecessary to solve the catfood problem; this will be discussed further in section 4.

3.3.1 Variable ordering

The order in which variables are considered for instantiation often has a dramatic effect on the time taken to solve a CSP, as does the order in which each variable's values are considered.

A common variable ordering heuristic is to choose next the variable with smallest remaining domain, so that the order is not fixed but depends on the effects of the choices already made. In this case, it seemed better to consider the variables in a fixed order, first assigning the number of slots in each template occupied by each variation, and considering the variations in ascending order of demand, d_i. Finally, the run length for each template is determined (a variation on this strategy is discussed below). Hence the variables are considered in the order $p_{1,1}, p_{1,2}, ..., p_{1,t}, p_{2,1}, ..., p_{n,t}, R_1, R_2, ..., R_t$, where $d_1 \leq d_2 \leq .... \leq d_n$.

When the current limit on *Production* is much higher than the optimum, it often happens that $R_1, R_2, ..., R_t$ are not completely determined by the time that the p_{ij} variables have all been assigned values. In that case, Solver finds the values of the R_j variables which give the minimum value of *Production* while keeping the p_{ij} variables fixed. However, when the current best solution is close to optimal, much more constraint propagation can take place, so that whenever a feasible assignment to the p_{ij} variables can be found, the values of the R_j variables have usually been determined in the process.

The values of all variables are considered in ascending order. An implication of this, given other choices discussed below, is that Solver will first look for a solution in which the variations with smallest demand (those considered first) occupy no slots in the first template, which is the one which will be printed fewest times.

3.3.2 Symmetry Constraints

In many problems, some of the problem entities are indistinguishable in the CSP formulation. If there are any solutions at all, there are classes of equivalent solutions. In the template design problem, variations with identical demands are indistinguishable; if $d_i = d_{i+1}$, and we have a solution with $p_{ij'} \neq p_{i+1,j'}$ for some template j', then the values of p_{ij} and $p_{i+1,j}$ for $j = 1, 2, ..., t$ can be interchanged to yield a different but equivalent solution. Similarly, the templates are indistinguishable: the values of $p_{1j}, p_{2j}, ..., p_{nj}$ and R_j can be exchanged with those of $p_{1k}, p_{2k}, ..., p_{nk}$ and R_k to derive another solution. These alternative solutions are for all practical purposes identical. Such symmetries in the problem may cause difficulties for a search algorithm: if the problem has no solution, or the algorithm is exploring a branch of the search tree which does not lead to a solution, then all symmetrical assignments will be explored in turn. This is a waste of effort, because if one such assignment is infeasible, then they all are. Symmetries should be avoided, if possible, by including additional constraints in the formulation which will allow only one solution from each class of equivalent solutions [4].

To do this, the following constraints have been introduced:

if $d_i = d_{i+1}$ then:

1. $p_{i1} \leq p_{i+1,1}$
2. if $p_{i1} = p_{i+1,1}$ then $p_{i2} \leq p_{i+1,2}$
3. if $p_{ij} = p_{i+1,j}$ for j = 1,2,...,$r-1$ then $p_{ir} \leq p_{i+1,r}$ for r = 3,..., t.

Designs with the same demand usually, but not always, occupy the same number of slots in every template. If they do not, the constraints just given serve to distinguish between them.

For the two-template problem, another symmetry constraint has been introduced:

$$\text{if } d_i = d_{i+1} \text{ and } p_{i1} < p_{i+1,1} \text{ then } p_{i2} > p_{i+1,2} \tag{18}$$

Similar constraints could be devised for $t = 3$, but this would be more complex, and has not so far been done: the resulting constraint would be used only rarely.

To distinguish between the templates, we add the constraint:

$$R_1 \leq R_2 \leq ... \leq R_t \tag{19}$$

3.3.3 Pseudo-symmetry Constraints

A further set of constraints has been added to the model which again eliminate equivalent solutions, as with symmetry constraints, but do not arise from indistinguishable

entities; for want of a better term, these will be called 'pseudo-symmetry' constraints. Suppose we have two variations i and $i+1$, with $d_i < d_{i+1}$; these two variations are not in general indistinguishable. However, if we have a solution in which the amount produced of variation i is greater than the amount produced of variation $i+1$, then we can derive a solution in which the values of $p_{i1}, p_{i2}, ..., p_{it}$ are exchanged with those of $p_{i+1,1}, p_{i+1,2}, ..., p_{i+1,t}$. From the point of view of the CSP model, the two solutions are equivalent, and one should be eliminated; from the practical point of view, the second solution is better.

So, if $d_i < d_{i+1}$ we can add a constraint:

$$\sum_j p_{ij} R_j \leq \sum_j p_{i+1,j} R_j$$

3.3.4 Implied Constraints: Upper Limits on Production

In general, the constraint propagation methods used in constraint programming work on constraints involving only two variables, or on constraints in which all but two of the variables have been assigned values (although for some special types of constraint more powerful propagation methods have been developed, e.g. [6]). In this problem, we do not always specify an upper tolerance on production of each variation in the CP model, so that any amount of over-production is allowed, consistent with the other constraints. The only upper limit is then the overall limit given by the optimization constraint on *Production*. This constraint indirectly involves all the problem variables (since *Production* is determined by the sum of the run lengths, and the values of these variables are assigned last, after the p_{ij} variables); hence it is very weak at detecting choices which must eventually result in the constraint being violated. Even when an upper tolerance on the amount produced is specified, it may only give a fairly loose constraint (for instance if we allow under-production at the same time). In both situations, we may be able to improve performance by looking for new constraints implied by those already in the model which propagate better. Unlike symmetry constraints, implied constraints do not affect the set of solutions. In the template planning problem, we can derive a new set of constraints from the optimization constraint which give upper limits on production of each variation, or set of variations.

In any solution, the number of surplus items to be printed is given by:

$$Surplus = Production \times S - \sum_i d_i$$

As the upper limit on *Production* gets closer to its minimum possible value $\underline{l}$, the number of surplus items which can be produced gets smaller. Since there is an upper limit on the total number of surplus items, the same upper limit applies to the number that can be produced of any variation, or of any set of variations.

This gives:

$$\sum_j p_{kj} R_j - d_k \leq Surplus \text{ for } k = 1, 2, ..., n \tag{20}$$

We can also have a constraint on the number of surplus items resulting from the production of the first k variations:

$$\sum_{m=1}^{k} \left(\sum_j p_{mj} R_j - d_m \right) \leq Surplus \text{ for } k = 2, ..., n-1 \tag{21}$$

These constraints become progressively tighter, and so lead to more constraint propagation, as successively better solutions are found. They help to avoid making mistakes early in the search which would be very time-consuming to correct.

Note that because the variables are assigned values in the order described earlier, the constraints (21) are more useful than they might appear. When the values of $p_{kj} (j = 1, 2, ..., t)$ are being assigned, the only variables in the kth cumulative constraint which have not yet been assigned values are p_{kj} and R_j, for $j = 1, 2, ..., t$, and $Surplus$. Hence (21) gives a tighter constraint on these variables at that point than that given in (20), which is useful for initial pruning of each variable's domain.

3.3.5 Implied Constraints on Run Length

If there are two templates, then one is used for at most half the total number of printings and the other for at least half. We can add new constraints specifying this, since we have already specified in section 3.3.2 that $R_1 \leq R_2$:

$$\begin{aligned} R_1 &\leq Production/2 \\ R_2 &\geq Production/2 \end{aligned} \tag{22}$$

and similarly when $t = 3$:

$$\begin{aligned} R_1 &\leq Production/3 \\ R_2 &\leq Production/2 \\ R_3 &\geq Production/3 \end{aligned} \tag{23}$$

and so on.

4 Results from Constraint Programming Approach

4.1 The Catfood Problem

For the catfood problem, we found solutions with one, two and three templates, and initially specified no upper limit on the amount produced of each variation. As already mentioned, the one-template solution is not practically useful, but serves to provide an upper limit on cost for the two-template problem. The two and three template solutions are shown in Table 4. The solutions shown are optimal, i.e. they are the least cost solutions for that number of templates, within the tolerances specified.

Specification	Templates	Pressings	Overall % Deviation	Maximum % Deviation
2 templates	[0,0,0,0,0,2,7]	158,000	2.6	4.5
$l_i = 0$	[1,1,1,2,2,2,0]	260,000		
3 templates	[0,5,3,0,0,1,0]	51,000	0.2	0.6
$l_i = 0$	[0,0,1,0,0,7,1]	107,000		
	[1,0,0,2,2,0,4]	250,000		
2 templates	[0,0,0,0,0,2,7]	142,000	-7.7	-10
$l_i = 10\%$	[1,1,1,2,2,2,0]	234,000		
3 templates	[0,0,0,0,0,3,6]	64,000	-9.9	-10
$l_i = 10\%$	[0,3,3,0,0,1,2]	78,000		
	[1,0,0,2,2,2,2]	225,000		

Table 4: Results for Catfood Problem using Constraint Programming

With three templates, it is possible to meet all the order quantities exactly (or at least to within 6,000 cartons overall, which is as near as it is possible to get, given that the order quantities are in thousands and each mother sheet produces 6 cartons).

As can be seen in Table 4, we found a two-template solution in which all the order quantities were met, with an overall over-production of 2.6% and a maximum over-production for any variation of 4.5%. This means that the problem owner's belief that the problem could be solved with two templates, within acceptable tolerances, was correct.

All the optimal solutions were well within the upper tolerance of 10%. We also found solutions with a lower tolerance of 10%, shown in Table 4. However, since the result is that, for *all* variations, the amount produced is less than the order quantity, it is not certain that this would in fact be acceptable (although meeting the problem owner's specification).

Apart from an initial attempt, described below, these solutions could be found very quickly with the Solver program. A common measure of the amount of effort required to find a solution to a CSP is the number of times the algorithms has to backtrack to a previous choice point, having encountered a failure during search. This gives an idea of the size of the search tree explored by the algorithm before finding a solution,

or proving that there is no solution.

We first looked for solutions using a basic program implementing none of the strategy described in section 3.3, apart from ordering the variables and their values as in section 3.3.1. We also did not specify an upper tolerance on over-production. Finding the optimal solution and proving optimality took a long time. For instance, with $l_i = 0$, finding the optimal two-template solution took only 5 backtracks (0.05 cpu secs.[1]); proving optimality took 29,600 backtracks (17.7 secs.). For the three-template problem, proving optimality was trivial, since the optimal solution required the minimum possible number of pressings, $\underline{l}$; but finding the optimal solution took 6,460,000 backtracks (3,800 secs.). Clearly, solving larger problems with this basic program would not be practicable. Specifying an upper tolerance of 10% on over-production makes a significant difference to these timings (although not to the solutions found): the two-template problem was solved in 139 backtracks (0.06 secs.) and the three-template problem in 186,000 backtracks (93 secs.). Implementing all the elements of the solution strategy described in section 3.3 gave a further reduction, especially in the three-template problem which could then be solved in only 109 backtracks (0.09 secs.); the two-template problem took 29 backtracks (0.04 secs.). These results show the importance of having an upper limit on the amount produced of each variation; both the upper tolerance (u_i) and the implied constraints using the currently available surplus (section 3.3.4) are useful. In the catfood problem, the symmetry constraints on variations with the same order quantity are not so important, since only two of the seven variations are affected.

4.2 The Herbs Problem

The herbs problem is much larger than the catfood problem, having 30 variations rather than 7. Nevertheless, optimal solutions have been found.

We first specified that no under-production should be allowed, and that over-production of any variation should be at most 10%. With these restrictions, there is no solution with only two templates; this can be proved in 45 backtracks (0.11 secs.). However, we removed the limit on over-production, and found the least cost two-template solution, to give a starting point for the search for a three-template solution; the result is shown in Table 5. A solution requiring 88,000 pressings was found very quickly (1431 backtracks, 2.8 secs.), but finding the optimal two and three template solutions at first proved very time-consuming. We switched to an alternative strategy in which the value of *Production* is specified in each run of the program, and the run-length variables R_j are assigned first, before the p_{ij} variables. Since the number of pressings required for the two-template solution is 87,000 and for this problem $\underline{l}$ is 84 (thousand), we needed to run the program five times, with *Production* = 87 and 86 for the two-template problem and *Production* = 86, 85 and 84 for the three-template problem, to find the optimal solution with *Production* = $\underline{l}$. These five runs took a total of 11.3 cpu secs.; however, this does not include the time that the original version of the program spent trying to find an improvement on the two-template solution with *Production* = 88 before it was stopped.

[1] on a Silicon Graphics O2.

We also found the optimal solution using two templates with at most 10% under-production or 10% over-production of any variation (277 backtracks, 0.89 secs.). It is probable that an acceptable solution could be found, somewhere between the two-template solutions shown in Table 5, with an overall deviation closer to zero.

Specification	Templates	Pressings	Overall % Deviation	Maximum % Deviation
2 templates $l_i = 0$	[4,4,0,0,0,0,0,0,0,1,1,1,1,2,2, 2,2,2,2,2,2,2,2,1,1,1,1,1,0,5] [0,0,1,1,1,1,1,1,1,1,1,1,1,1,1, 1,1,1,1,1,1,1,1,2,3,3,3,3,4,3]	15,000 72,000	4.4	13.3
3 templates $l_i = 0$ u_i =10%	[0,0,0,0,0,0,0,0,0,0,0,0,0,3,4, 4,4,4,4,3,3,3,3,5,0,0,0,0,0,2] [5,5,0,0,0,0,0,0,0,1,1,1,1,7,1, 1,1,1,1,2,2,2,2,0,2,2,2,2,0,0] [0,0,1,1,1,1,1,1,1,1,1,1,1,0,1, 1,1,1,1,1,1,1,1,2,3,3,3,3,4,4]	2,000 12,000 70,000	0.8	2.5
2 templates l_i =10% u_i =10%	[0,0,0,0,0,0,0,0,0,1,1,1,1,2,2, 2,2,2,2,2,2,2,2,1,1,1,1,1,0,13] [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1, 1,1,1,1,1,1,1,1,2,3,3,3,3,4,1]	15,000 64,000	−5.2	−10, +6.7

Table 5: Results for Herbs Problem using Constraint Programming

The optimal three template solution with $l_i = 10\%$ and $u_i = 10\%$ is not shown since, as before, all variations were underproduced by up to 10% and this is unlikely to be acceptable.

4.3 The Magazine Inserts Problem

The order quantities for the magazine inserts problem are very diverse (ranging from 50 to 405), and the number of variations is greater than the number of slots in each template. These factors together mean that the order quantities cannot be even approximately satisfied with only two templates: most variations can appear in only one template, so that the range of order quantities cannot be adequately covered. It was easily proved that no two-template solution with $< 10\%$ over-production exists, even if up to 10% under-production is allowed. In the best solutions found, shown in Table 6, there is unacceptable over-production of the variations with smallest order quantities. The same problem arises with three templates if there is no under-production, but a solution with less than 10% over-production can be found if 10% under-production is also allowed; again, it can be easily proved that it is necessary to allow under-production in order to achieve $< 10\%$ over-production. With four templates, a solution with no under-production and very close to the minimum possible number of pressings $\underline{l}$ (238,000 compared with 234,000) was found.

Specification	Templates	Pressings	Overall % Deviation	Maximum % Deviation
2 templates $l_i = 0$	[1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2, 2,2,2,2,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,1,1,1] [0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0, 0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2]	100,000 168,000	14.6	100
3 templates $l_i = 0$	[0,0,0,0,0,0,1,1,2,3,0,0,0,0,1,1,1,1,1,1,1,1,6,0,0, 0,0,1,1,1,0,0,0,0,0,0,2,2,2,2,2,2,2,2,0,0,0,0,0,0] [1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,2,2, 2,2,2,2,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,1,1,0] [0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0, 0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,3]	4,000 98,000 147,000	6.4	96
4 templates $l_i = 0$, u_i= 10%	[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,2,2,2,2,2,2,2,8,2,3,3,5,5,0] [0,0,2,2,3,3,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,1,1,1, 0,0,0,0,0,0,0,0,1,1,1,1,3,3,3,3,3,3,3,0,0,0,0,0,0,0] [1,1,0,0,0,0,0,2,2,2,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 1,1,1,1,1,4,4,1,1,1,1,0,0,0,0,0,0,0,4,2,2,2,1,1,2] [0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1, 1,1,1,1,1,0,0,1,1,1,1,1,1,1,1,1,1,1,0,1,1,1,2,2,2]	5,000 30,000 53,000 150,000	1.0	9.5
2 templates $l_i = 10\%$	[1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2, 2,2,2,2,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,1,1,1] [0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0, 0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2]	90,000 152,000	3.4	80
3 templates $l_i = 10\%$ $u_i = 10\%$	[0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,1,1,1,1,3,3,3,3,3,3,3,3,0,0,0,5,5,0] [1,1,1,1,1,1,2,2,2,2,0,0,0,0,0,0,0,0,0,0,0,3,3,3, 1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,2,0,0,1,1,2] [0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0, 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,1,1,2]	30,000 54,000 135,000	−6.4	−10, +8

Table 6: Results for Magazine Inserts Problem using Constraint Programming

Some of these solutions were hard to find, and only the first shown in Table 6 has been proved optimal. For this two-template problem, the program takes a long time to find any solution requiring fewer than 309,000 pressings; better solutions were found by solving a simpler problem in which all the order quantities are divided by 10, and showing that for the minimum number of pressings required in that case is 270,000 pressings. Then the strategy described in the last section was used to look for solutions for the original data with *Production* successively equal to 269, 268 and 267; the program (eventually) proved that there is no solution with *Production* = 267, so the solution given in the table, with 268,000 pressings, is optimal.

Another device was also used to find this solution: if a solution has been found with $Production = l$, and there are solutions with $Production = l - 1$, it is likely that one of them will have run lengths very similar to the solution we already have. For instance, the solution found with $Production = 269$ has $R_1 = 100$, $R_2 = 169$; the next solution found has $R_1 = 100$, $R_2 = 168$ (although the layout of the templates is significantly different). The programs have been modified so that a small range of values in which the values of each R_j might be expected to lie can be specified; where there is some evidence on which to base the guess, this can be a very successful way of improving on a solution found by some other means. It does not, of course, help with proving optimality, or in cases where there is a larger change in the run lengths from one solution to the next.

The same strategy was used to find the solution given for two templates with $l_i = 10\%$, and for three templates with no under-production; these are not known to be optimal.

On the other hand, the three and four template solutions with $u_i = 10\%$ were found very quickly, in 3500 backtracks (8.4secs.) and 9530 backtracks (9.6 secs.) respectively. This again demonstrates the importance in reducing the search space of an upper limit on the amount produced of each variation. No attempt has been made to reduce the number of pressings required for the three template problem, because this would only worsen the overall under-production. The four template solution is already close to the minimum possible number of pressings. All attempts to improve on this solution have so far proved extremely time-consuming and have not succeeded. However, in practice one or other of these solutions would be acceptable to the company.

5 Improving the Constraint Programming Solutions

The results in section 4 for cases in which the full range of upper and lower tolerances is used generally exhibit underproduction. Whilst this does meet the original problem specification, it is unlikely to be acceptable if it happens consistently. Improvement of the solutions obtained from the constraint programming approach can be attempted using the goal programming model (7) - (10). This model permits a variety of different solutions to be generated by adjusting the number of pressings of

a set of specified templates. Use of the goal programming model with some sample cases from Tables 3, 4 and 5 gives the improved results in Table 7. Running the goal

Specification	Templates	Pressings	Overall % Deviation	Maximum % Deviation
Catfood 3 templates $l_i = 10\%$	[0,0,0,0,0,3,6] [0,3,3,0,0,1,2] [1,0,0,2,2,2,2]	71,667 85,000 250,000	−1.4	−1.9
Herbs 2 templates l_i =10% u_i =10%	[0,0,0,0,0,0,0,0,0,0,1,1,1,1,2,2, 2,2,2,2,2,2,2,2,1,1,1,1,1,0,13] [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1, 1,1,1,1,1,1,1,1,2,3,3,3,3,4,1]	16,500 66,000	−3.2	+10, -6.7
Magazines 3 templates l_i = 10% u_i = 10%	[0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,1,1,1,1,3,3,3,3,3,3,3,3,0,0,0,5,5,0] [1,1,1,1,1,1,2,2,2,2,0,0,0,0,0,0,0,0,0,0,0,3,3,3, 1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,2,0,0,1,1,2] [0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,0,0,0, 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,1,1,2]	35,000 55,000 145,000	− 2.4	−8.3, +7.4

Table 7: Results for Goal Program Adjustment of Constraint Programming Solutions

programming model takes negligible time so it is viable to use different conditions on under and over production and different objective functions to generate alternative solutions.

6 Conclusions

The attempts described earlier to solve the template design problem using ILP have proved to be capable of producing solutions, but for the three test problems, the solutions are often not good enough. The difficulty lies in the fact that the problem cannot easily be expressed in the form of linear constraints. Constraint programming does much better, producing (with some effort) acceptable solutions for all three test problems which in many cases have been proved to be optimal.

This performance is somewhat surprising: although it is easy to *express* non-linear constraints using CP tools, it is not necessarily easy to solve the resulting CSPs. In this case, there are constraints (both non-linear and linear) involving large numbers of variables, and such constraints are usually hard to deal with: they do not allow the effects of variable assignments to be propagated to other variables, so that mistakes can be detected, until nearly all the variables involved have been assigned. In particular, the objective constraint involves all the decision variables (and there are 204 such variables in the four-template magazine inserts problem); this would suggest that until nearly all the decision variables have been assigned, it will be impossible to tell whether the current limit on *Production* can be met. However, because of the

constraints expressing an upper limit (if there is one) on the amount produced of each variation, and other constraints which can be added to the formulation as described in section 3.3, potential errors can in fact be detected much earlier.

It is particularly impressive that the program easily detects, in the two larger test problems, when the upper and lower tolerances cannot be met, and that several solutions have been proved to be optimal. When all that is required is to find a feasible solution, good heuristics can sometimes guide the search so that a solution is found very quickly, even when the search space is very large. Proving that there is no solution or proving optimality requires a complete search and there are no short-cuts.

The problem remains difficult, however, and some manual intervention was required to find solutions to the larger problems. For the largest problem, although good solutions have been found which cannot be far from optimal, proving optimality or finding better solutions has so far proved impossible. Although finding provably optimal solutions would not be necessary in practice, this does suggest that the program is operating near its limit, and that its performance on similar problems might not be robust.

We have shown in section 6 that ILP does have a role to play in improving constraint programming solutions. The template layouts from the CP solution are kept and solutions which more closely meet the order quantities can be found. This takes negligible time, since the constraints are linear; clearly, attempting to do this using CP, when ILP is so well suited to it, would be pointless.

It is unlikely that our experience with the template design problem will carry over directly to other problems. However, some broad general lessons can be drawn. The problem was a difficult one for both ILP and CP, for different reasons. The fact that it could not easily be expressed in terms of linear constraints ultimately defeated ILP, despite our best efforts. Since CP can in principle handle non-linear constraints, this is an obvious alternative to try in such cases. However, devising a successful strategy for solving the problem required considerable ingenuity; a straightforward formulation with only sufficient constraints to ensure that solutions were correct only solved the smallest of the problems. Because the CSP is a direct representation of the original problem, and the search algorithms used are basically very simple, it is relatively easy to understand where the difficulties lie and to incorporate insights into the problem into the problem-solving strategy. Although the template design problem required a more sophisticated strategy than usual, the need to devise such a strategy is common when using CP to solve complex problems. The fact that insights based on human problem-solving ingenuity can be built in is one of the strengths of constraint programming.

References

[1] A. R. Brown. *Optimum Packing and Depletion.* Macdonald, 1971.

[2] Dash Associates. XPRESS-MP Reference Manual, 1990.

[3] J. P. Ignizio. *Linear Programming in Single and Multiple Objective Systems.* Prentice-Hall, 1982.

[4] J.-F. Puget. On the Satisfiability of Symmetrical Constrained Satisfaction Problems. In *Proceedings of ISMIS'93*, 1993.

[5] J.-F. Puget. A C++ Implementation of CLP. In *Proceedings of SPICIS94 (Singapore International Conference on Intelligent Systems)*, 1994.

[6] J.-C. Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings AAAI-94*, volume 1, pages 362–367, 1994.

[7] E. Tsang. *Foundations of Constraint Satisfaction.* Academic Press, 1993.

[8] P. van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, 1989.

[9] H. P. Williams. *Model Building in Mathematical Programming.* John Wiley & Sons, 1993.