Modelling a Steel Mill Slab Design Problem

Alan M. Frisch, Ian Miguel, Toby Walsh
Artificial Intelligence Group
University of York
York, England
{frisch, ianm, tw}@cs.york.ac.uk

Abstract

We consider an industrial steel mill slab design problem. This problem is an instance of a class of difficult problems where the problem structure (in this case, the number and size of slabs) is not fixed initially, but determined as part of the solution process. Since a natural CSP encoding does not immediately suggest itself, the choice of model for this type of problem may be crucial in successfully solving it. This paper develops three different models to tackle the slab design problem. Model A uses a conservatively large number of variables to represent the slabs, some of which will typically be redundant in an optimal solution. Model B operates in two phases, first solving an abstraction of the original problem before solving further sub-problems to provide a solution to the original problem. The third model is a dual model combining models A and B, allowing search and propagation on both sets of variables.

1 Introduction

Many problems exhibit some flexibility in portions of their structure (such as the number required of a certain type of variable) which must be determined by the solution procedure. Steel mill slab design, considered in this paper, is an example of such a problem. Steel is produced by casting molten iron into slabs, of which a steel mill is capable of making a finite number of weights. Given a set of orders with desired weights, the flexibility lies in the number and size of slabs chosen to fulfil the orders. These choices specify a secondary problem of assigning orders to slabs such that waste is minimised.

This paper develops three models, A, B, and a combined model A/B, to solve the slab design problem. Model A avoids the problem of an unknown number of slabs by establishing an upper bound on the number of slabs required and creating that number of slab variables. This model has the advantage of simplicity and that it provides a solution in a single step (as opposed to model B below). However, since generally some of the slab variables are unnecessary, it can suffer the twin problems of an overly large problem and the introduction of a lot of symmetry.

Model B discards individual slab identifiers initially, focusing instead on the abstraction of the original problem of assigning orders to a particular slab size. The solution of the abstract problem dictates the structure of a further set of sub-problems, each of which consists of assigning a subset of the orders to a fixed number of slabs with a common size. This model has the advantage of compactness, since no redundant variables are required. However, due to the abstract nature of the initial problem, one or more of the secondary sub-problems may be inconsistent, necessitating a (potentially expensive) cycle of repairs at the abstract level, followed by the solution of further sub-problems.

Other problems have a similar nature, and therefore present similar modelling problems to which the models developed in this paper should be adaptable. Consider, for example, a warehouse location problem [van Hentenryck, 1999] in which the number of warehouses is unknown. The decision as to how many warehouses are necessary has a direct effect on how their placement can be optimised.

The remainder of this paper is structured as follows. The next section introduces the steel mill slab design in more detail, and provides an example problem. Models A and B are introduced in sections 3 and 4 respectively, and illustrated using the example problem. Model A/B is discussed in section 5. Finally, section 6 concludes the paper and points out future work.

2 A Steel Mill Slab Design Problem

A finite number, σ, of *slab sizes* is available. Each of the j input orders has two properties, a *colour* corresponding to the route required through the steel mill and a *weight*. An order cannot be split between slabs. The problem is to pack orders onto slabs such that the total slab capacity is minimised. There are two types of constraint:

1. **Capacity constraints**. The total weight of orders assigned to a slab cannot exceed the slab capacity.
2. **Colour constraints**. Each slab can contain at most p of k total colours (p is usually 2). This constraint arises because it is expensive to cut the slabs

up in order to send them to different parts of the mill.

Dawande et al [Dawande *et al.*, 1998] describe an asymptotic polynomial time approximation scheme for this problem. They also present more practical 3-approximation algorithms for realistic-sized instances.

2.1 An Example

There follows a small illustrative example problem which will be used throughout the paper. For this problem, $p = 2$.

- Slab sizes available ($\sigma = 3$): {1, 3, 4}
- Colours ($k = 5$): {Red, Green, Blue, Orange, Brown}

Table 1 presents the input orders to this problem and table 2 presents an example solution.

Order	Weight	Colour
1	2	Red
2	3	Green
3	1	Green
4	1	Blue
5	1	Orange
6	1	Orange
7	1	Orange
8	2	Brown
9	1	Brown

Table 1: Input Orders for the Example Problem

Slab	Size	Orders Assigned
1	4	1, 8
2	3	2
3	1	3
4	1	4
5	3	5, 6, 7
6	1	9

Table 2: A Solution to the Example Problem

3 Model A - Redundant Variables

Given an input set of orders with known weights and colours, the orders seem to be good candidates for being variables. Hence, j input orders are modelled by a list, O, of order variables $\{o_1, ..., o_j\}$ with domains consisting of identifiers for a particular slab.

The number of slabs is not fixed, although if it is assumed that the weight of each order does not exceed the maximum possible slab size, the maximum number of slabs required is j. One possibility, therefore, is to have a list, S, of slab variables, $\{s_1, ..., s_j\}$, each with a domain consisting of the possible slab sizes. The cost function for calculating the quality of a solution is then simply the sum of the assignments to each s_i. Generally, a subset of the slab variables will be *redundant*, i.e. an optimal solution does not assign any orders to these slabs. In order to cater for this, an extra element, 0, is added to the domain of each slab variable such that, if slab s_i is not necessary to solve the problem, $s_i = 0$ in the solution.

Since the slab variables are indistinguishable, model A in its current form is likely to suffer from symmetry problems: the slab sizes assigned to the variables in S may be permuted without affecting the solution (assuming order assignments are updated appropriately). This can be counteracted effectively through the use of binary symmetry-breaking constraints of the form $s_1 \geq s_2$, $s_2 \geq s_3$, etc.

3.1 Model A of the Example Problem

For the example problem, $o_1, ..., o_9$ and $s_1, ..., s_9$ are required, as presented in table 3. The former have identical initial domains $\{1, ..., 9\}$, denoting the available slabs. The latter also have identical domains $\{0, 1, 3, 4\}$ denoting the available slab sizes, and with 0 indicating that the slab is not used.

Order Variable	Domain	Slab Variable	Domain
o_1	{1, ..., 9}	s_1	{0, 1, 3, 4}
o_2	{1, ..., 9}	s_2	{0, 1, 3, 4}
o_3	{1, ..., 9}	s_3	{0, 1, 3, 4}
o_4	{1, ..., 9}	s_4	{0, 1, 3, 4}
o_5	{1, ..., 9}	s_5	{0, 1, 3, 4}
o_6	{1, ..., 9}	s_6	{0, 1, 3, 4}
o_7	{1, ..., 9}	s_7	{0, 1, 3, 4}
o_8	{1, ..., 9}	s_8	{0, 1, 3, 4}
o_9	{1, ..., 9}	s_9	{0, 1, 3, 4}

Table 3: Variables for Model A of the Example Problem

For reference, table 4 presents one possible solution to model A of the example problem. This is a 'perfect' solution since the total weight of the slabs used (4 + 2 x 3 + 3 x 1 = 13) is equal to the total weight of the orders.

Order Variables	Slab Variables
$o_1 = 1$	$s_1 = 4$
$o_2 = 2$	$s_2 = 3$
$o_3 = 3$	$s_3 = 1$
$o_4 = 4$	$s_4 = 1$
$o_5 = 5$	$s_5 = 3$
$o_6 = 5$	$s_6 = 1$
$o_7 = 5$	$s_7 = 0$
$o_8 = 1$	$s_8 = 0$
$o_9 = 6$	$s_9 = 0$

Table 4: Solution: Model A of the Example Problem

3.2 Constraints

Weight constraints express that the combined weight of the orders assigned to a particular slab cannot exceed the size assigned to that slab. For example, if size 3 is assigned to s_1, orders 2 and 3 cannot be assigned to slab 1, since their combined weight is 4. A total of j 'weighted occurrence' constraints of arity $j+1$ are used to achieve this, each ranging over the entire set of order variables and one of the slab variables. A weighted occurrence constraint is simply an occurrence constraint [Régin, 1996] that takes into account the weight of each variable (in this case, an order variable) that is assigned the target value (the identifier of the slab variable).

A simple method of implementing the colour constraints is to create a set of 'not-all-same' constraints of arity $p+1$ (recall that p denotes the maximum number of colours allowed on one slab) over order variables with different colours. An example of such a constraint is $c(o_1, o_2, o_4)$: since the colours of orders 1, 2 and 4 are red, green and blue respectively they cannot be assigned to the same slab. For this problem, 51 such ternary constraints are required. In general, the number of colour constraints is bounded above by ${}^jC_{p+1}$[1], when each order has one of j unique colours.

The Model A Colour Daemon

The problem with this initial scheme is the number of colour constraints required. Real slab design problems typically have hundreds or thousands of orders and colours, producing far too many colour constraints to deal with. A remedy for this is to replace the large set of constraints with a single 'daemon' (i.e. a large non-binary constraint, defined intensionally) that ranges over the entire set of order variables. The description below is with respect to performing forward checking following an order variable assignment, but the daemon could be extended to maintain generalised arc consistency.

Two definitions are required to describe the daemon:

- *colourSet*$_i$. As the order variables are assigned to slab s_i, the colour daemon maintains a record of the set of colours assigned to it in *colourSet*$_i$.
- *saturation*. Once a slab has orders with p different colours assigned to it, it is said to be saturated.

Once s_i is saturated, propagation on the remaining order variables can take place. The daemon iterates through those order variables that are not yet instantiated and, for those whose associated colour is not present in *colourSet*$_i$, the domain element i (if present) may be pruned. Instantiating such an order variable to s_i would violate the colour constraint of allowing only p colours per slab.

The extra space requirement for the model A colour daemon is in terms of the colour sets associated with each slab. The maximum space required for these is $O(jp)$, much less than that required by ${}^jC_{p+1}$ individual constraints with arity $p+1$. The time required by the propagation process is $O(j^2)$, since j order variables must be examined, each with a maximum of j domain elements.

[1] ${}^nC_r$ denotes the number of ways of selecting r of n objects and is defined: $\frac{n!}{r!(n-r)!}$

3.3 Implied Constraints for Model A

The combined weight of the input orders provides a unary implied constraint on the lower bound of the optimisation variable: the total slab weight must be at least this value. In addition, a lower bound on the number of slabs required can be found by simply dividing the total weight of orders by the largest available slab size (13 and 4 in the example, so at least 4 slabs are required). This lower bound decomposes into unary constraints on the slab variables, allowing the removal of 0 from the domain of the first i variables, if i is the minimum number of slabs required.

4 Model B - Abstraction

Model B consists of two phases. In the first phase, an abstraction of the original problem is constructed and solved (hopefully quickly). Phase 2 consists of a set of independent sub-problems, each centred around assigning a set of orders to a number of slabs of a particular size, as defined by the solution of phase 1. Solving the phase 2 sub-problems either provides a solution to the original problem, or, if any of the sub-problems are over-constrained, new constraints which restrict the set of solutions at phase 1. Phase 1 must then be solved again with respect to the new constraints, producing a new set of sub-problems at phase 2, and so on.

4.1 Phase 1 Variables

Model B considers the different slab sizes as variables, $z_1, z_2, ...$ with domains {0, ..., j} denoting the number of slabs of the corresponding size used. For clarity, the index of the z variable indicates the size it represents. Since the number of slab sizes is typically far less than the number of orders, this leads to a more compact representation than model A. The cost function for model B sums the number of slabs of each size used with each term multiplied by the slab size.

Modelling the order variables then becomes more difficult, however. Since model B does not identify individual slabs, the domains of the order variables can no longer consist of the slab to which the order is currently assigned to. Instead, the domain of an order variable contains the *size* of the slab to which this order is assigned.

4.2 Model B, Phase 1, of the Example Problem

Using this model, the example problem requires variables $o_1, ..., o_9$, all with the domain {1, 3, 4}, and z_1, z_3, z_4, all with the domain {0, ..., 9} (see table 5). This is fewer variables than are required by model A, and also eliminates the symmetry on the slab variables.

Table 6 presents a solution to the example problem represented using model B. This is a 'perfect' (though ambiguous) solution since the total weight of the slabs used (3 x 3 + 1 x 4 = 13) is equal to the total weight of the orders.

Order Variable	Domain	Slab Size Variable	Domain
o_1	{1, 3, 4}	z_1	{0, ..., 9}
o_2	{1, 3, 4}	z_3	{0, ..., 9}
o_3	{1, 3, 4}	z_4	{0, ..., 9}
o_4	{1, 3, 4}		
o_5	{1, 3, 4}		
o_6	{1, 3, 4}		
o_7	{1, 3, 4}		
o_8	{1, 3, 4}		
o_9	{1, 3, 4}		

Table 5: Variables for Model B of the Example Problem, Phase 1

Order Variables	Slab Size Variables
$o_1 = 3$	$z_1 = 0$
$o_2 = 3$	$z_3 = 3$
$o_3 = 3$	$z_4 = 1$
$o_4 = 3$	
$o_5 = 3$	
$o_6 = 3$	
$o_7 = 4$	
$o_8 = 4$	
$o_9 = 4$	

Table 6: Solution: Model B of the Example Problem, Phase 1

4.3 Phase 1 Constraints

Weight constraints range over the entire set of order variables $o_1, ..., o_j$ and one slab size variable, z_i. They express that there must be a sufficient number of slabs of size i to accommodate the total weight of orders assigned to size i slabs. This can be implemented via a weighted occurrence constraint. The example problem requires 3 such constraints. In general, the number of weight constraints required by model B, phase 1 is σ.

As per model A, colour constraints can be stated individually. In this case, each is an occurrence constraint with arity $k+1$, ranging over a subset of the order variables with all different colours and a slab size variable, z_i. They express that $pz_i \geq \omega$, where ω is the number of orders assigned to the slab size i. That is, the assignment to z_i must be sufficient to accommodate the total number of orders assigned to slab size i with respect to their colours. One such colour constraint for the example problem is $c(o_1, o_2, o_4, o_5, o_8, z_1)$. This problem requires 36 such constraints in total. In general, the number of colour constraints for model B, phase 1 is bounded above by ${}^jC_k\sigma$ when each order has one of j unique colours.

Hence, model B specified in this way would also suffer from a crippling number of colour constraints. Again, a daemon can be used to remedy this problem, in this case ranging over both the order variables and the size variables. The model B colour daemon maintains a record of the set of colours assigned to each slab size as the order variables are instantiated, requiring space $O(\sigma k)$. This information is used to trigger propagation on both the order variables and the slab size variables. Taking the latter case first, the lower bound of z_i (assuming integer division) is:

$$\frac{|colourSet_i|}{p} + 1$$

As $|colourSet_i|$ increases, this propagation is triggered whenever $|colourSet_i|\ (mod\ p) = 1$. This type of propagation simply requires the re-calculation of the lower bound of z_i which can be done in constant time.

Propagation involving the order variables is based on the colour-saturation of a slab size variable in much the same way as for the model A colour daemon. When a slab size variable, z_i say, is instantiated and $\frac{|colourSet_i|}{p} = z_i$ (exactly) then i can be removed from the domains of all orders that contain it and whose associated colour is not in $colourSet_i$. This propagation takes time $O(j\sigma k)$, since j orders must be examined, each with maximum domain size σ and their associated colours must be searched for in $colourSet_i$, with maximum size k.

4.4 Implied Constraints for Model B, Phase 1

As per model A, a unary implied constraint exists on the optimisation variable such that its lower bound is the combined weight of all the input orders. In addition, the lower bound on the number of slabs required, as described in section 3.3, can be enforced on the sum of the assignments to the z variables.

A unary constraint on the domain of each order variable can be derived by considering the weight of an order in relation to the available slab sizes. The lower bound on the domain of an order variable is the smallest slab size that is greater than the weight of the associated order.

4.5 Model B, Phase 2

As noted above, without an explicit representation of individual slabs, model B phase 1 is ambiguous: a solution to the problem modelled in this way does not provide the orders-to-slabs assignments required by the original problem statement. On the other hand, a solution to a model B phase 1 solution does provide the number and sizes of the slabs required, the size of slab each order is assigned to and the quality of the final solution.

The phase 1 solution presented in table 6 is used to construct a much simplified phase 2 problem (see table 7). Since the number and size of the slabs is known, slab variables are not required. In addition, the domain of an order variable now contains identifiers for particular slabs of the size specified in the solution to the phase 1 problem. Indeed, since only those orders assigned to the same slab size interact, this second problem may be decomposed into a number of sub-problems, each of which considers packing the slabs of a particular size.

Constraints are restricted to combinations of order variables with identical domains - i.e. those that were assigned to a common slab size in phase 1. Under this restriction, colour constraints for phase 2 are stated as

Sub-problem	Slab	Slab Size	Order Variable	Domain	Colour	Weight
1	1	3	o_1	{1, 2, 3}	Red	2
	2	3	o_2	{1, 2, 3}	Green	3
	3	3	o_3	{1, 2, 3}	Green	1
			o_4	{1, 2, 3}	Blue	1
			o_5	{1, 2, 3}	Orange	1
			o_6	{1, 2, 3}	Orange	1
2	4	4	o_7	{4}	Orange	1
			o_8	{4}	Brown	2
			o_9	{4}	Brown	1

Table 7: Model B of the Example Problem, Phase 2

per model A. Weight constraints are stated similarly, disallowing order combinations that exceed the slab size chosen.

Table 8 presents a solution to the problem constructed for phase 2 of model B of the example problem. It represents an alternative solution to the original problem to that given in table 4.

Slab	Slab Size	Order Variables
1	3	$o_1 = 1$
2	3	$o_2 = 2$
3	3	$o_3 = 1$
4	4	$o_4 = 3$
		$o_5 = 3$
		$o_6 = 3$
		$o_7 = 4$
		$o_8 = 4$
		$o_9 = 4$

Table 8: Solution: Model B of the Example Problem, Phase 2

4.6 The Price of Ambiguity

Given the ambiguity of model B phase 1, it is possible that one or more of the sub-problems constructed in phase 2 might be over-constrained. Table 9 presents a simple problem that demonstrates this. Given that the only slab size available is size 4, the weight constraints require there to be two such slabs to accommodate the combined weight of the orders. Similarly, if $p = 1$ the colour constraints require that there are two slabs since there are two colours.

Order	Domain	Weight	Colour	Slab Size	Domain
o_1	{4}	3	Red	z_4	{0,...,4}
o_2	{4}	3	Red		
o_3	{4}	1	Blue		
o_4	{4}	1	Blue		

Table 9: Problem for which Model B Produces an Inconsistent Phase 2 Problem, when $p = 1$

However, when the phase two problem is constructed, as presented in table 10, it is inconsistent. Consider that colour constraints prevent o_1 from being on the same slab as either o_3 or o_4 and a weight constraint prevents o_1 from being on the same slab as o_2. Hence, o_1 must be on a slab by itself, leaving one slab for the remaining three orders. Due to weight and colour constraints amongst o_2, o_3 and o_4, the problem is therefore inconsistent.

Slab	Size	Order	Domain	Colour	Weight
1	4	o_1	{1,2}	Red	3
2	4	o_2	{1,2}	Red	3
		o_3	{1,2}	Blue	1
		o_4	{1,2}	Blue	1

Table 10: Inconsistent Model B, Phase 2 Problem

Conflict Recording

From the above example, it might be inferred that phase 1 simply underestimates the value of the optimisation variable or the number of a particular slab size. However, this is not necessarily the case: the value of the optimisation variable may be correct, but with respect to a different combination of slab sizes. Or, the combination of slab sizes may be correct, but not the assignment of orders to sizes. The question, therefore, is how to isolate the reasons for failure at phase 2 and post appropriate constraints at phase 1 to prevent the failure from re-occurring.

The sub-division of phase 2 into independent sub-problems aids this process. When a phase 2 sub-problem is over-constrained, the arity of the phase 1 constraint implied by this failure is (at most) one more than the number of variables in the sub-problem. In the example shown in tables 9 and 10 above, the following constraint can be posted at phase 1:

$$o_1 = 4 \wedge o_2 = 4 \wedge o_3 = 4 \wedge o_4 = 4 \rightarrow z_4 > 2$$

If the phase 1 problem is now solved again, the same inconsistent sub-problem cannot re-occur and a different sub-problem (or sub-problems in the general case) will be generated for phase 2.

Generally, it is possible that a cycle of several such failures followed by the addition of new constraints may be necessary before a final solution is found. Therefore, it is important to maximise the efficiency of this process.

Order (A)	Domain	Slab (A)	Domain	Order (B)	Domain	Slab Size (B)	Domain
o_{A1}	{1,...,9}	s_1	{0,1,3,4}	o_{B1}	{1,3,4}	z_1	{0,...,9}
o_{A2}	{1,...,9}	s_2	{0,1,3,4}	o_{B2}	{1,3,4}	z_3	{0,...,9}
o_{A3}	{1,...,9}	s_3	{0,1,3,4}	o_{B3}	{1,3,4}	z_4	{0,...,9}
o_{A4}	{1,...,9}	s_4	{0,1,3,4}	o_{B4}	{1,3,4}		
o_{A5}	{1,...,9}	s_5	{0,1,3,4}	o_{B5}	{1,3,4}		
o_{A6}	{1,...,9}	s_6	{0,1,3,4}	o_{B6}	{1,3,4}		
o_{A7}	{1,...,9}	s_7	{0,1,3,4}	o_{B7}	{1,3,4}		
o_{A8}	{1,...,9}	s_8	{0,1,3,4}	o_{B8}	{1,3,4}		
o_{A9}	{1,...,9}	s_9	{0,1,3,4}	o_{B9}	{1,3,4}		

Table 11: Variables for Dual Model A/B of the Example Problem

Firstly, since the constraints added at phase 1 are not directly implied by the phase 1 representation, this process can be viewed as constraint *restriction* in the sense of restriction/relaxation-based dynamic CSP [Dechter and Dechter, 1988]. Efficient dynamic CSP algorithms can be employed to solve this evolving problem without redoing much of the work from scratch as is naively the case.

A second improvement is in the arity of the constraints recorded. It is likely that, in general, only a subset of the variables in an over-constrained phase 2 sub-problem are in conflict. If smaller conflicts can be identified, the resulting phase 1 constraints will also be smaller, leading to more pruning and therefore more quickly to a solution.

5 Model A/B: A Dual Model

A Dual model can also be considered, combining model A and phase 1 of model B. This approach removes the need for the cyclical repair strategy described in section 4.6 while retaining the power of model B to avoid symmetry problems.

5.1 Variables

As table 11 shows, model A/B contains both explicit slab variables (s_i) and slab-size variables (z_i). Both order variables with domains referring to explicit slabs (o_{Ai}) and domains referring just to slab sizes (o_{Bi}) are also included.

5.2 Channelling Constraints

The constraints required for the individual models are as specified in sections 3.2 and 4.3. In order to maintain consistency and to aid pruning between the two models, additional channelling constraints are required. For example, channelling constraints between the s and z variables state that the number of occurrences of assignment i in $s_1, ..., s_j$ must be equal to the assignment of z_i. This ensures consistency in the number of each size of slab. Such constraints (of which model A/B requires σ in general) are again easily implemented by an occurrence constraint.

Channelling constraints are also necessary between the order variables. These constraints express the implication $o_{Ai} = j \rightarrow o_{Bi} = s_j$, i.e. that the size of the of slab referred to by the model A order variable must match that assigned to the the model B order variable. Note that this constraint is not of the form 'iff', because of the ambiguity inherent to model B: the size assigned to o_{Bi} might match that of several model A slab variables.

5.3 Search Strategy

When attempting to solve a model A/B problem, the variable instantiation order has a marked effect on the benefit derivable from maintaining both basic models. If, for example, the model A variables are instantiated first, the channelling constraints will ensure that all model B variables are also instantiated. This process is analogous to simply solving the pure model A problem, and will therefore not provide any further pruning of the search tree.

The opposite strategy is to search initially with the model B variables, then attempt to instantiate the model A variables. Since model B is ambiguous, the channelling constraints do not force the instantiation of the model A variables as the model B variables are assigned. Model B variable assignments do, however, constrain the model A variables. Hence, when the search continues on to the model A variables, failures are identified more quickly than by considering model A alone. This process is analogous to the 2-phase pure model B approach, with dead ends uncovered during the search on the model A variables corresponding to inconsistent phase 2 sub-problems.

Other search strategies specific to model A/B could also be considered, interleaving the instantiation of the variables from the two basic models to obtain the most efficient pruning of the search space.

6 Conclusion

The models described here have thus far been tested only on small problem instances created from subsets of real industrial data. Since, for these problems, the number of sizes available is relatively close to the number of orders, model A performs the most strongly - sometimes solving the problem more quickly even than model B phase 1. This trait is not expected to scale to larger problems: as the ratio of orders to available slab sizes increases, so will the ratio of the size (in terms of number of variables) of model A to model B and model B phase 1 should be easier to solve.

The next step, therefore, is to develop further the models described and apply them to more realistic problem instances. The use of colour daemons as opposed to many individual colour constraints should enable the models to scale effectively. A more realistic and more detailed comparison will allow a better judgement of the relative merits of models A, B and A/B and the identification of circumstances in which we can expect one to perform better than the others.

It would also be interesting to model the problem using set variables, with each variable representing a slab, and its domain being the set of orders assigned to it. Partition constraints on the set variables would then encapsulate the occurrence constraints used in the current models. A model C might also be considered, utilising activity-based dynamic CSP [Mittal and Falkenhainer, 1990]. Here, model A slab variables would be used, but only 'activated' as needed, according to the remaining capacity of the activated slabs. This approach would avoid the existence of a large number of redundant variables and constraints.

Acknowledgements

The authors are grateful to Andrew Davenport and Jayant Kalagnanam for information about the problem and instances, and to our anonymous reviewer for useful comments. The authors are supported by EPSRC Grant GR/N16129[2]. The third author is supported by an EPSRC advanced research fellowship.

References

[Dawande *et al.*, 1998] M. Dawande, J. Kalagnanm, and J. Sethurmana. Variable sized bin packing with color constraints. tr 21350, IBM T J Watson Research Center, 1998.

[Dechter and Dechter, 1988] R. Dechter and A. Dechter. Belief maintenance in dynamic constraint networks. *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 37–42, 1988.

[Mittal and Falkenhainer, 1990] S. Mittal and B. Falkenhainer. Dynamic constraint satisfaction problems. *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 25–32, 1990.

[Régin, 1996] J-C. Régin. Generalized arc consistency for global cardinality constraints. *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 209–215, 1996.

[van Hentenryck, 1999] P. van Hentenryck. *The OPL Optimization Programming Language.* MIT Press, 1999.

[2] `http://www.cs.york.ac.uk/aig/projects/implied/index.html`