Symmetry and Implied Constraints in the Steel Mill Slab Design Problem

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

Abstract

Steel mill slab design, which reduces to variable-sized bin packing with colour constraints, is a challenging problem for constraint satisfaction techniques. Starting from a basic model, we add symmetry-breaking and implied constraints which greatly reduce the amount of search required. We further discuss how each of these constraints might be generated automatically.

1 Introduction

We consider an industrial steel mill slab design problem, which is a type of bin-packing problem with colour side-constraints [3]. 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. Basic models of this problem struggle to cope with even very small instances [7]. In order to improve a basic model, it is important to a) deal with the symmetry in the problem, and b) add implied constraints (i.e. constraints which logically follow from the initial specification) which help the solver to reduce search. This paper develops a model which addresses both of these issues. The model was formulated by hand, but we will discuss it in terms of how it could be generated automatically from a basic version.

1.1 Automatic Generation of Implied Constraints

Extending a constraint satisfaction problem with implied constraints can often result in a marked reduction in the amount of search needed to solve the problem [10]. Consequently, we often add such constraints to an initial problem formulation. Typically, however, this process is performed by hand. We have begun the process of automating the generation of implied constraints, using a system based on proof planning [6], a technique used for guiding the search for a proof in automated theorem proving [1]. Common patterns in proofs are identified and encapsulated in methods, which the planner selects and applies successively to reduce goals to sub-goals. Methods have strong preconditions which limit their applicability and prevent combinatorially explosive search. Rather than a goal-directed search for proof, our system begins from a basic specification of the problem and forward chains, in a search for useful implied constraints. We use methods which encapsulate common patterns in generating implied constraints by hand. The lessons learnt from generating implied constraints for the steel mill problem will be used to expand the current set of methods and inform the way in which the proof planner performs its search.

2 A Steel Mill Slab Design Problem

Steel is produced by casting molten iron into slabs. A finite number, σ, of *slab sizes* is available. An order has two properties, a *colour* corresponding to the route required through the steel mill and a *weight*. The problem is to pack the d input 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.

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

Table 1 shows the input orders ($d = 9$) to this problem, and 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

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

Table 1: Input Orders and a Solution for the Example Problem

3 A Matrix Model

We focus on developing the most successful of three basic models for the slab design problem [7]. Potentially redundant variables are used to cope with the fact that that the number of slabs required by an optimal solution is unknown. If we assume that the greatest order weight does not exceed the maximum slab size, the worst case consists of assigning each order to an individual slab. Hence, a maximum of d slabs are used, giving slab variables, $s_1, ...s_d$ with domains of size σ consisting of the size of each slab. This gives a simple expression for the total weight of slabs used, which must be minimised:

$$\sum_i s_i = totalWeight$$

Generally, some slab variables will remain unused. 0 is added to the domain of each slab variable such that if $s_i = 0$, s_i is not necessary to solve the problem.

3.1 The Order Matrix

Rather than d order variables with domains identifying the slab that each order is assigned to, as presented in [7], a $d \times d$ 0-1 matrix, $order_A$, is used:

$$\begin{array}{cc} & \begin{array}{cccc} o_1 & o_2 & ... & o_d \end{array} \\ \begin{array}{c} s_1 \\ s_2 \\ ... \\ s_d \end{array} & \begin{pmatrix} 0/1 & 0/1 & ... & 0/1 \\ 0/1 & 0/1 & ... & 0/1 \\ ... & ... & ... & ... \\ 0/1 & 0/1 & ... & 0/1 \end{pmatrix} \end{array}$$

Constraints on the rows ensure that the slab capacity is not exceeded,

$$\forall j \sum_i weight(o_i) \times order_A[i,j] \leq s_j$$

Constraints on the columns ensure that each order is assigned to one and only one slab.

$$\forall i \sum_j order_A[i,j] = 1$$

The justification for this matrix model is that it allows us to remove the symmetry from the problem, as discussed in section 4.

3.2 The Colour Matrix

In [7], we advocated the use of a specialised daemon to enforce the colour constraints. A (less opaque) alternative is to use a second 0-1 matrix, $colour_A$ with dimensions $k \times d$.

$$\begin{array}{c} \\ s_1 \\ s_2 \\ \ldots \\ s_d \end{array} \begin{array}{c} \begin{array}{cccc} red & green & \ldots & brown \end{array} \\ \begin{pmatrix} 0/1 & 0/1 & \ldots & 0/1 \\ 0/1 & 0/1 & \ldots & 0/1 \\ \ldots & \ldots & \ldots & \ldots \\ 0/1 & 0/1 & \ldots & 0/1 \end{pmatrix} \end{array}$$

Constraints link $order_A$ to $colour_A$.

$$\forall i \forall j . order_A[i,j] = 1 \rightarrow colour_A[colour(o_i), j] = 1$$

Constraints on the rows of $colour_A$ ensure that orders with at most p colours are assigned to each slab.

$$\forall j \sum_i colour_A[i,j] \leq p$$

3.3 A Matrix Model Solution

A solution using the matrix model is presented below. In this solution, only 4 slabs are used, so the remaining variables and corresponding rows of $order_A$ and $colour_A$ are set to 0.

$$\begin{array}{c} \\ s_1 = 4 \\ s_2 = 3 \\ s_3 = 3 \\ s_4 = 3 \\ \ldots \end{array} \begin{array}{c} \begin{array}{ccccccccc} o_1 & o_2 & o_3 & o_4 & o_5 & o_6 & o_7 & o_8 & o_9 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \qquad \begin{array}{c} \\ s_1 = 4 \\ s_2 = 3 \\ s_3 = 3 \\ s_4 = 3 \\ \ldots \end{array} \begin{array}{c} \begin{array}{ccccc} red & green & blue & orange & brown \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

4 The Role of Symmmetry

Symmetry-breaking constraints are not implied, but are very useful in reducing the amount of search required. Furthermore, the generation of some implied constraints is not possible without the prior existence of symmetry-breaking constraints (see [6], and one of the first examples in [8]). Therefore, symmetry in the slab design problem is discussed first.

The slab variables, $s_1, \ldots, s_d$ are indistinguishable: the slab sizes assigned to each of these variables may be permuted without affecting the solution (assuming order assignments are updated appropriately). The proof planner contains a `symmetry` method designed to identify indistinguishable variables and terms and add appropriate symmetry-breaking constraints, as described in detail in [6]. Firstly, a normal form is imposed on the set of constraints which describe the problem. All occurrences of two candidate variables or terms, x and y are swapped. Following re-normalisation, if the set of constraints derived matches the original set, x and y are indistinguishable. The symmetry breaking constraints imposed by this method take the form of ordering constraints (assuming algebraic constraints). Hence, in this case we have:

$$s_1 \geq s_2 \geq \ldots \geq s_d$$

However, symmetry persists in the problem in two areas. Firstly, columns of $order_A$ associated with identical orders (i.e. same weight and colour) are symmetrical. Exploiting the matrix model, a powerful method of removing this symmetry is to impose a lexicographic order on the symmetrical columns. Hence,

$$weight(o_i) = weight(o_j) \wedge colour(o_i) = colour(o_j) \rightarrow \forall h\, order_A[i,h] \leq_{lex} order_A[j,h]$$

where $\leq_{lex}$ represents lexicographically less or equal to. The current `symmetry` method is capable of detecting this symmetry via term comparison. However, to add the lexicographic ordering it is useful to know that these terms represent the columns of a matrix. This, coupled with the common occurrence of matrix models (see [5]) suggests that matrices should be made first class objects in the input language to the proof planner. A further benefit, given that symmetry detection is a potentially expensive process, is that matrix rows and columns are heuristically strong things to compare for symmetry [4].

Secondly, rows of $order_A$ associated with slabs assigned the same size are indistinguishable. In any solution, the collection of orders assigned to two such slabs may be swapped. Again, we can use a lexicographic ordering to remove this symmetry. The order matrix allows us to reason about the collection of orders assigned to symmetrical slabs in a way that is difficult using single order variables. In order to generate the lexicographic ordering constraints automatically, the proof planner must recognise that symmetry can occur based on partial assignments, and compare variables and terms under the assumption that certain assignments have been made. For example, the following pair may be compared under the assumption that s_1 and s_2 are equal.

$$\sum_i weight(o_i) \times order_A[i,1] \leq s_1$$
$$\sum_i weight(o_i) \times order_A[i,2] \leq s_2$$

Since $\forall i\, weight(o_i)$ are constant, we concentrate on the rows of 0-1 variables. Using the existing `symmetry` method, i.e. exchanging every occurrence of the terms $\forall i\, order_A[i,1]$ and $\forall i\, order_A[i,2]$ throughout the constraint set and normalising, $\forall i\, order_A[i,1]$ and $\forall i\, order_A[i,2]$ are clearly symmetrical under our assumption. Hence, we can impose:

$$s_1 = s_2 \rightarrow \forall i\, order_A[i,1] \leq_{lex} order_A[i,2]$$

Lexicographic orderings imposed on more than one dimension of a matrix must be chosen carefully to avoid conflicts [4]. Clearly, it would be very expensive generally to test arbitrary terms for symmetry under partial assignments, reinforcing the need for a heuristic approach based on high level constructs such as matrix objects.

5 Implied Constraints

A variety of implied constraints exist for the slab design problem which can dramatically improve performance on this problem. We consider first some unary constraints that can be derived in a straightforward manner. Unary implied constraints are desirable because they lead directly to domain reductions without recourse to any search. A first example follows directly from the symmetry-breaking constraints on the slab variables. Given that s_1 is always the largest, or largest-equal slab:

$$s_1 \geq max_i(weight(o_i))$$

That is, the first slab must be able to accommodate the largest order. Otherwise, the order is too large to be assigned to any slab or $\exists s_i\, i > 1 \wedge s_i > s_1$, which violates the ordering constraints.

The combined weight of the input orders provides a lower bound on the total weight of steel to be produced (since all orders must be assigned).

$$\sum_i weight(o_i) \leq totalWeight$$

A lower bound can also be derived on the number of slabs required by dividing the combined weight of the input orders by the largest available slab size.

$$\frac{\sum_i weight(o_i)}{max(slabSizes)}$$

When combined with the binary ordering constraints on the slab variables, this decomposes into unary constraints, allowing the removal of the 0 element from the first few slab variables.

The unary implied constraints described above require fairly extensive algebraic manipulation. The proof planner currently contains some simple algebraic manipulation methods, such as `isolate`, `collect` and `attract` from the PRESS system [2]. In the future, we plan to interface with computer algebra packages to make these sorts of manipulations for us.

5.1 Assigned Weight Variables

This section gives an example of the utility of adding new variables to the problem representation, which we encapsulate as the `introduce` method. Since the uncontrolled introduction of new variables would quickly cause an explosion in the size of the problem representation, we must be very careful about its use. One possible criterion is that a variable should only be added if we can immediately prune its domain. In addition, adding a set of similar variables is justified if we can make use of an efficient constraint (e.g. all-different) to constraint its elements.

We introduce d new variables $assWt_i$ which contain the weight of orders assigned to s_i.

$$assWt_i = \sum_j order_A[j, i] \times weight(o_j)$$

The reason for introducing these new variables is that their domains can be pruned more effectively than through the use of bounds consistency alone. This is achieved using a dynamic programming approach based on [11] that reasons about reachable values, taking into account both available order sizes and their colours. Figure 1 presents part of the network that is constructed by this process for the example problem. A non-horizontal directed arc represents setting an entry in $order_A$ to 1, hence adding the weight of the corresponding order to the assigned weight. Similarly, a horizontal directed arc represents the case where the entry in $order_A$ is set to 0. At each reachable node, a 'history' list is kept of the colours of the order combinations that have been used to reach this value thus far. In the example problem, the assigned weight 2 can be reached using just the colour red (o_1), or colours green and blue (o_3 and o_4) or various other combinations. Each of these combinations are added as an entry in the history list of the appropriate node.

When considering whether or not an order o_i can be added to the current combination (i.e. a non-horizontal move in the network), the following are checked:

- Whether adding o_i would exceed the maximum slab size.
- The colour of o_i. To be added, an order's colour must be compatible with at least one entry in the history for the current node. That is, its colour must be present in the entry or there must be a free slot in the entry (recall that up to p colours are allowed per slab).

The history list for the node reached by adding the weight of a new order is constructed from the combination of the new order's colour and the compatible history entries. History lists from two arcs incident into the same node are merged. In a horizontal move, no new orders are assigned, hence the history list is the same as that of the adjacent reachable node to the left.

One special case exists: if one entry in the list is subsumed by another, it is removed. In figure 1, for instance, (green) subsumes (red, green) in the entry for o_3, assigned weight 3. Hence the latter can be discarded. It is safe to do so because (green) supports a superset of the additional orders that (red, green) does.

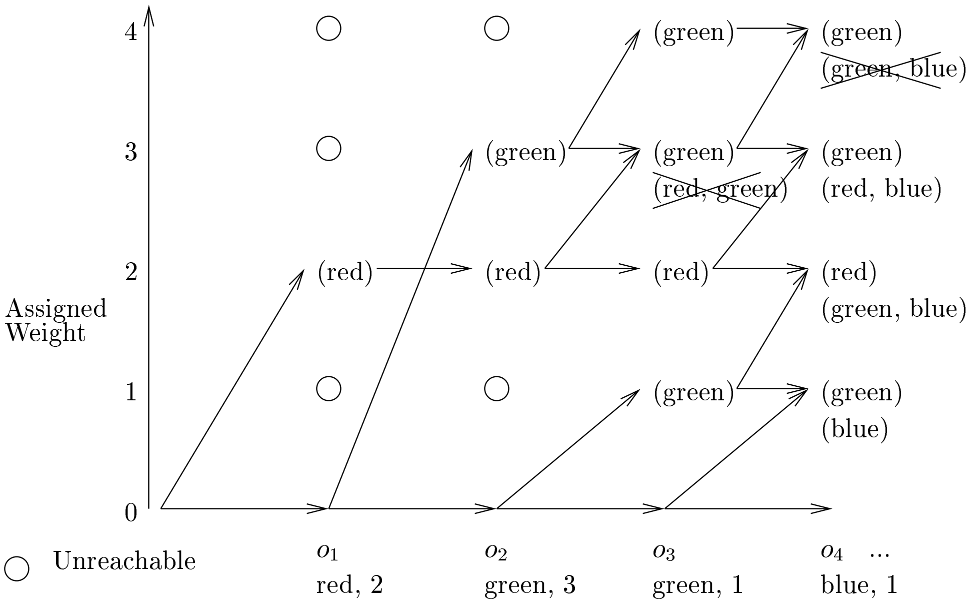

Figure 1: Reasoning about reachable assigned weight values.

Once the network is constructed, all unreachable values can be pruned from the domain of every $assWt_i$. This doesn't do anything for the example problem because the slab sizes involved are very small, but can lead to a lot of pruning in more realistic instances. Currently, this process is used as a pre-processing step. However, we hope it will be much more effective during search. As soon as entries are set in $order_A$, this further constrains the reachable values, allowing still more pruning of the $assWt$ variables. At first sight, this is a very specialised piece of reasoning. However, knapsack constraints (equivalent to the rows of the order matrix) are a sufficiently common occurrence to warrant specific mechanisms to deal with them. The colour constraints involved in the above method *are* quite specialised, but we envisage a basic knapsack pruning method, with the ability to accept certain forms of side-constraints.

This process also provides a maximum assigned weight for any slab. So the lower bound on the number of slabs required can be revised:

$$\frac{\sum_i weight(o_i)}{maxAssWt}$$

5.2 Wasteage Variables

As a further application of the `introduce` method, another set of d variables, $waste_i$, are now introduced, representing the unused (i.e. wasted) portion of each slab.

$$waste_i = s_i - assWt_i$$

We can put an upper bound on the total waste in a solution by considering again the worst case where each order is assigned to a single slab. Under this assumption, we choose for each order the smallest slab that is large enough to contain it. More formally:

1. $\forall i \forall j. i \neq j \rightarrow o_i \neq o_j$
2. $\forall i. o_i = j \rightarrow \neg \exists a. a \in slabSizes \wedge a < s_j \wedge a \geq weight(o_i)$

The maximum total waste is found by summing the waste for each such assignment. This allows us to put an upper bound on the optimisation variable.

$$totalWeight \leq maxTotalWaste + \sum_i weight(o_i)$$

We can put an upper bound on each $waste_i$. This is simply the largest of the cases encountered when computing the maximum total waste. If this value is exceeded for a particular slab, we know we can do better simply by assigning all the orders on that slab to individual slabs of their own.

Wasteage variables have been found to be useful previously, for example in rack configuration [9] and template design [8]. This suggests that wasteage is a feature that the proof planner should be able to recognise and reason about.

6 Results

Table 2 presents results generated from (small) subsets of industrial data, implemented in ILOG Solver 5.0. The basic model cannot solve and find a proof of optimality to any of these instances. Adding symmetry-breaking constraints alone drastically improves performance, enabling the solver to find and prove optimal solutions for all but the largest instances. Adding the implied constraints improves the model further still, with the new bounds on the optimisation variable especially helpful to the proof of optimality.

Orders	Optimal	Basic Model	Basic + Sym-Breaking	Basic + Sym-Breaking + Implied
12	77	80:285607,23.7s 79:288871,24.0s 77:289337,24.1s P: —	83:21,0.2s 78:446,0.1s 77:4954,1.1s P:23165,3.8	83:4,0.1 78:165,0.2 77:1245,0.5 P:1254,0.6
13	79	80:285607,25.2,0.1s 79:290989,25.6s —	83:21,0.12s 80:829,0.3s 79:4545,0.9s P:39513,6.7s	83:4,0.11s 80:68,0.15s 79:941,0.4s P:951,0.6s
14	87	—	95:46,0.19s 89:1128,0.5s 88:1187,0.6s 87:34071,6.4s P:179336,31.8s	95:5,0.2s 89:831,0.42s 88:845,0.5s 87:8572,2.1s P:8584,2.2s
15	92	—	95:76,0.2s 94:8476,1.9s 93:8617,2.0s 92:63709,12.7s P:643459,119.2s	95:4,0.2s 94:4446,1.1s 93:4461,1.3s 92:16741,3.7s P:16749,3.7s
16	99	—	107:100,0.2s 101:6323,1.4s 100:6690,1.6s 99:377970,78.1s P:—	107:5,0.2s 101:5100,1.3s 100:5117,1.4s 99:93627,20.3s P:93641,20.2s
17	103	—	107:64,0.3s 105:19541,5.4s 104:38167,10.3 —	107:5,0.2s 105:9473,2.8s 104:9550,3.0s 103:213618,59.6s
18	110	—	119:371,0.32s 111:9241,2.7s —	119:6,0.2s 111:3941,1.3s 110:741206,227.13s P:741221,228.3s

Table 2: Results for Model A, Model A/B (Solution Quality: Cumulative Fails, Cumulative Time). 'P:' is proof of optimality. A dash means that the search was halted after 1,000,000 fails.

The real problems from which these instances are derived are orders of magnitude larger than those considered here, and are solved to within a bound of optimal via linear programming techniques [3]. Despite being unable (yet) to compete with these techniques, it is encouraging to note that the best model of the problem enables the solver to get close to the optimal solution very rapidly.

7 Conclusion

We have developed a model for steel mill slab design, a difficult problem to solve successfully via constraint satisfaction. Starting from a basic model, we showed how the removal of symmetry and the addition of implied constraints lead to significant improvements in performance. We described how some of the symmetry-breaking and implied constraints can already be generated automatically via a proof-planning system, and discussed how that system could be extended to generate the remainder. In order to solve realistic-sized instances, further significant improvements to the current model are necessary. We expect a substantially improved proof planning system to be able to help in the search for these improvements.

Acknowledgements

This project and the second author are supported by EPSRC Grant GR/N16129[1]. The third author is supported by an EPSRC advanced research fellowship.

References

[1] A. Bundy. A science of reasoning. In J-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 178–198. MIT Press, 1991.

[2] A. Bundy and B. Welham. Using meta-level inference for selective application of multiple rewrite rules in algebraic manipulation. *Artificial Intelligence*, 16(2):189–212, 1981.

[3] M. Dawande, J. Kalagnanam, and J. Sethurmana. Variable sized bin packing with color constraints. Technical Report TR 21350, IBM T J Watson Research Center, 1998.

[4] P. Flener, A.M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Symmetry in matrix models. Technical Report APES-30-2001, APES Research Group, 2001. URL: http://www/dcs.st-and.ac.uk/apes/reports/apres-30-2001.ps.gz. Submitted to SymCon'01 CP2001 workshop.

[5] P. Flener, A.M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Matrix modelling. Technical Report APES-36-2001, APES Research Group, 2001. URL: http://www/dcs.st-and.ac.uk/apes/reports/apres-36-2001.ps.gz. Submitted to Formul'01 CP2001 workshop.

[6] A. Frisch, I. Miguel, and T. Walsh. Extensions to proof planning for generating implied constraints. In S Linton and R Sebastiani, editors, *Proceedings of the 9th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning (CALCULEMUS-01)*, 2001.

[7] A.M. Frisch, I. Miguel, and T. Walsh. Modelling a steel mill slab design problem. *Proceedings of the IJCAI-01 Workshop on Modelling and Solving Problems with Constraints*, pages 39–45, 2001.

[8] L. Proll and B. Smith. Ilp and constraint programming approaches to a template design problem. *INFORMS Journal of Computing*, 10:265–275, 1998.

[9] D. Sabin and E. Freuder. Optimization methods for constraint resource problems. *Proceedings of the AAAI Workshop on Configuration. Tech Report WS-99-05*, 1999.

[10] B.M. Smith, K. Stergiou, and T. Walsh. Using auxiliary variables and implied constraints to model non-binary problems. *Proceedings of the 16th National Conference on Artificial Intelligence*, pages 182–187, 2000.

[11] M Trick. A dynamic programming approach for consistency and propagation for knapsack constraints. *Proceedings of the Third International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR-01)*, pages 113–124, 2001.

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