The Steel Mill Slab Design Problem Revisited

P. Van Hentenryck[1] and L. Michel[2]

[1] Brown University, Box 1910, Providence, RI 02912
[2] University of Connecticut, Storrs, CT 06269-2155

Abstract. Recently, Gargani and Refalo (G&R) presented an elegant model for the Steel Mill Slab Design Problem (Problem 38 in the CSP LIB). Contrary to earlier approaches, their model does not use 0/1 variables but exploits the traditional expressiveness of constraint programming. G&R indicated that static symmetry-breaking constraints proposed earlier are not effective on this model, as these interact with their heuristic. Instead they use large neighborhood search to obtain solutions quickly. This paper shows that a simple search procedure breaking symmetries dynamically leads to a constraint program solving the problem in a few seconds, while maintaining the completeness of the approach and removing the need for large neighborhood search.

1 Introduction

The steel mill slab design problem (problem 38 in the CSP Library) has attracted significant interest in the community. The problem consists of packing a set of orders into slabs, minimizing the total capacity of the slabs needed while satisfying the capacity and order compatibility constraints on the slabs. The CSPLIB proposes an instance with 111 orders which could not be solved to optimality by constraint-programming approaches until last year. Earlier work included the presentation of different models in [1], the study of symmetry breaking in [2], the hybridization of constraint and mathematical programming in [4] which solves a sub-instance of the CSP Lib problem with 30 orders in about 1000s, and the local search solver WSAT(OIP) for pseudo boolean variables [7] which solves the decision problem with 111 orders in about 2000s.

Last year, Gargani and Refalo [3] reconsidered the problem using a constraint-programming approach. They stated that "*the models used for [earlier] constraint programming approaches to this problem were basically linear models over binary variables. While such models are suited for integer programming solvers that can tighten the formulation by cutting-plane generation, these models are notoriously not well suited to a constraint programming approach because of the limited domain reductions they produce.*" They introduced a natural constraint-programming model using logical and global constraints exploiting the structure of the problem. By designing a specific strategy for variable and value selection and combining the heuristic with a large-neighborhood search, they showed how to solve the largest instance with 111 orders in just 3s using the Ilog constraint-programming solver.

Gargani and Refalo also studied how to add (static) constraints in order to prevent the search strategy from producing symmetrical solutions. Their experimental results show that these constraints are useful for small instances but negatively impact performance on larger instances. They argued that "*the symmetry-breaking constraints prevent our strategy from finding good solutions causing the loss in performance*" and explained the interference between the search heuristics and the symmetric-breaking constraints. As a result, they stated: "*For these reasons, we have not used symmetry breaking constraints in our constraint programming solution, and we have dramatically improved the convergence of the search by using a local search approach.*"

This paper shows that their constraint-programming model with a simple, dynamic symmetry-breaking scheme leads to a constraint program solving the problem in a few seconds, while maintaining the completeness of the approach and removing the need for large neighborhood search.

2   The Steel Mill Slab Design Problem

The problem consists in producing n orders using a set of slabs. Each order o has a color c_o and a weight w_o representing the slab capacity it takes. Each slab has a capacity that must be chosen from the increasing set of capacities $\{u_1, u_2, ..., u_k\}$. A solution is an assignment of orders to slabs such that

1. the total weights of the orders in a slab must not exceed the slab capacity;
2. the orders in a slab can be of two different colors only.

The objective is to minimize the sum of the weights of the slabs used in the solution or, equivalently, the sum of losses (unused capacity) in the slabs used in the solution.

3   The Constraint Program

Figure 1 depicts the constraint program for solving the steel mill slab problem in COMET. Lines 1–16 are essentially the model of Gargani and Refalo, while lines 18–24 are the new search procedure including the dynamic symmetry breaking.

The ingenuity in their model is in the expression of the objective function. Indeed, the model uses two sets of decision variables: variable x[o] specifies the slab assigned to order o, while variable l[s] represents the load of slab s. Once the load of a slab is known, it is easy to compute its loss: simply take the smallest capacity supporting the load. Line 6 computes an array of losses for each possible capacity, while the objective function in line 12 uses the element constraint to compute the loss of each slab. Note that a slab with no order incurs no loss. Gargani and Refalo use a global packing constraint [5] for computing the weight: this constraint is semantically equivalent to

```
forall(s in Slabs)
  cp.post(sum(o in Orders) weight[o] * (x[o] == s) == l[s]);
```

```
1  int capacities[Caps] = ...;
2  int weight[Orders] = ...;
3  int color[Orders] =...;
4  set{int} colorOrders[c in Colors] = filter(o in Orders) (color[o] == c);
5  int maxCap = max(i in Caps) capacities[i];
6  int loss[c in 0..maxCap] = min(i in Caps: capacities[i] >= c) capacities[i] − c;
7
8  Solver<CP> cp();
9  var<CP>{int} x[Orders](cp,Slabs);
10 var<CP>{int} l[Slabs](cp,0..maxCap);
11
12 minimize<cp> sum(s in Slabs) loss[l[s]]
13 subject to {
14     cp.post(packing(x,weight,l));
15     forall(s in Slabs)
16         cp.post(sum(c in Colors) (or(o in colorOrders[c]) (x[o] == s)) <= 2);
17 } using {
18     forall(o in Orders) by (x[o].getSize(),−weight[o]) {
19         int ms = max(0,maxBound(x));
20         tryall<cp>(s in Slabs: s <= ms + 1)
21             cp.label(x[o],s);
22         onFailure
23             cp.diff(x[o],s);
24     }
25 }
```

Fig. 1. The Constraint-Programming Model in COMET

and the experimental results will discuss its importance. The second set of constraints are meta-constraints specifying that the orders can be of at most two different colors.

The search procedure in lines 18–24 is the main novelty here. It iterates on the orders, selecting first the orders with the smallest domains (first-fail principle) and breaking ties by choosing orders with the largest weight (line 18). The search procedure then considers the slabs to assign to the selected order: it only considers slabs in which some orders have been placed as well as *one* additional empty slab. Line 19 computes the already used slabs (i.e., $1..ms$), while line 20 is a nondeterministic instruction trying to assign the slabs to variable x[o].

Observe that the entire model is 25 lines of COMET, does not include large neighborhood search, and is guaranteed to be complete, since two empty slabs are equivalent for allocating an order. These value symmetries were studied theoretically in [6] and have been used in several constraint programs for graph coloring, scene allocation, deployment of serializable services to name only a few. Note also that there are other symmetries that could be broken: two slabs with the same capacities and the same colors are also symmetric, but it was not necessary to break these symmetries to achieve good performance.
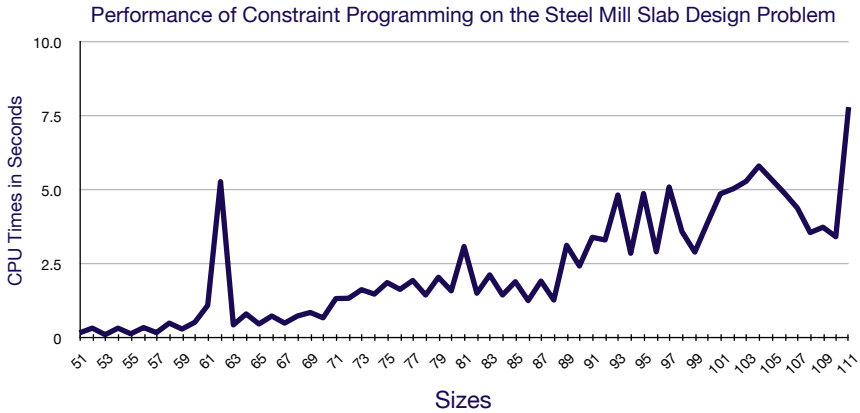
Fig. 2. Performance of the COMET Program on the Steel Mill Slab Design Problem

4 Experimental Results

Figure 2 depicts the experimental results on a 2.16 GHz Intel processor running
Mac OS X 10.5.1. As can be seen, the COMET program solves all instances within
less than 8 seconds, indicating that this problem has become extremely easy for
constraint programming.

Readers may wonder how much of the efficiency is due to the global constraint
for packing. To determine its contribution, it was replaced by the constraints

```
forall(s in Slabs)
  cp.post(sum(o in Orders) weight[o] * (x[o] == s) == l[s]);
cp.post(sum(o in Orders) weight[o] == sum(s in Slabs) l[s]);
```


Fig. 3. Performance of the COMET Program with no Global Constraint

The first set of constraints was discussed earlier and captures the semantics of the global constraint, while the last constraint is semantically redundant and expresses that the total weight of the orders is equal to the total load of the slabs. Figure 3 depicts the experimental results. They indicate that all the instances are now solved within 10 seconds, showing that the global constraint is not strictly necessary here. Observe also the similar shape of the computation results.

5   Conclusion

In recent work, Gargani and Refalo (G&R) presented an elegant model for the Steel Mill Slab Design Problem (Problem 38 in the CSP LIB). Contrary to earlier approaches, their model does not use 0/1 variables but exploits the traditional expressiveness of constraint programming. G&R indicated that static symmetry-breaking constraints proposed earlier are not effective on this model, as these interact with their heuristic. Instead they use large neighborhood search to obtain solutions quickly. This paper showed that an simple search procedure using the first-fail principle and dynamic symmetry breaking leads to a constraint program solving the problem in a few seconds, while maintaining the completeness of the approach and removing the need for large neighborhood search.

It is interesting to observe that the steel mill slab design problem is now solved efficiently using technology and concepts which were all available in 1991.

References

1. Frisch, A., Miguel, I., Walsh, T.: Modelling a steel mill slab design problem. In: Proceedings of the IJCAI 2001 Workshop on Modelling and Solving Problems with Constraints (2001)
2. Frisch, A., Miguel, I., Walsh, T.: Symmetry and implied constraints in the steel mill slab design problem. In: Walsh, T. (ed.) CP 2001. LNCS, vol. 2239, pp. 77–92. Springer, Heidelberg (2001)
3. Gargani, A., Refalo, P.: An efficient model and strategy for the steel mill slab design problem. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, Springer, Heidelberg (2007)
4. Hnich, B., Kiziltan, Z., Miguel, I., Walsh, T.: Hybrid modelling for robust solving. Annals of Operations Research 130(1–4), 19–39 (2004)
5. Shaw, P.: A Constraint for Bin-Packing. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 648–662. Springer, Heidelberg (2004)
6. Van Hentenryck, P., Flener, P., Pearson, J., Ågren, M.: Tractable symmetry breaking for csps with interchangeable values. In: International Joint Conference on Artificial Intelligence (IJCAI 2003) (2003)
7. Walser, J.: Solving linear pseudo-boolean constraints with local search. In: Proceedings of the Eleventh Conference on Artificial Intelligence, pp. 269–274 (1997)