Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

STEFAN HEINZ*
THOMAS SCHLECHTE
RÜDIGER STEPHAN

Solving Steel Mill Slab Problems with Branch and Price

* Supported by the DFG Research Center MATHEON *Mathematics for key technologies* in Berlin.

ZIB-Report 09-14 (Mai 2010)

Solving Steel Mill Slab Problems with Branch and Price

Stefan Heinz*, Thomas Schlechte, and Rüdiger Stephan

Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany
{heinz,schlechte,stephan}@zib.de

Abstract. The steel mill slab design problem from the CSPLib is a binpacking problem that is motivated by an application of the steel industry and that has been widely studied in the constraint programming community. Recently, several people proposed new models and methods to solve this problem. A steel mill slab library was created which contains 380 instances. A closely related binpacking problem called multiple knapsack problem with color constraints, originated from the same industrial problem, were discussed in the integer programming community. In particular, a simple integer programming for this problem has been given by Forrest et al. [3]. The aim of this paper is to bring these different studies together. Moreover, we adopt the model of [3] for the steel mill slab problem. Using a state of the art integer program solver, this model is capable to solve *all* instances of the steel mill slab library, mostly in less than one second, to optimality. We improved, thereby, the solution value of 76 instances.

1 Introduction

The *steel mill slab problem* is motivated by a real world application from the steel industry. The problem consists of a set of n orders, each order j coming with a size $s_j \in \mathbb{N}$ and color $c_j \in C$, where C is a finite set. Furthermore, we are given a set of m capacities $K := \{k_1, \ldots, k_m\} \subset \mathbb{N}$. The task is to equip each used slab with one capacity and assign each order to exactly one slab with the requirements that the selected capacities are respected and that each slab only processes orders of at most two different colors. The objective is to minimize the *leftover* that is the total loss or equivalent the residual capacity.

The steel mill slab problem is problem number 38 of the CSPLib[1]. This library provides one instance which consists of 111 orders with 88 different colors, and 20 possible capacities. We call this instance the *original instance*. Furthermore, there exists a steel mill slab library [11]. This library contains 380 instances which are grouped into 19 classes each with 20 instances. These instances have been created by changing the set of possible capacities of the original instance.

* Supported by the DFG Research Center Matheon *Mathematics for key technologies* in Berlin.

[1] http://www.csplib.org/

This means, the orders are the same as the one of the original instance. The capacities are generated uniformly and range between 10 and 50; class i contains instances which have $i + 1$ possible capacities.

In the following section, we give a brief overview on different approaches to solve the steel mill slab problem and related binpacking problems. One of the presented models is a column generation approach of Forrest et al. [3] to the so-called *multiple knapsack problem with color constraints* which can be confessed as a slight generalization of the steel mill slab problem. In this paper, we adapt this approach to the steel mill slab problem, and using a state of the art integer program solver, we solve *all* instances of the steel mill slab library and the original instance to optimality. Thereby, we improved the solution of 76 instances and proved for all instances, even those with a leftover greater than zero, that the known solution values are optimality.

2 Related work

In the past, several different models have been proposed to solve the steel mill slab problem. A first set of constraint programming models has been presented by Frisch et al. [4] and first computational results for a (small) subset of orders of the original instance were given by the same authors in [5]. Dawande et al. [2] presented an asymptotic polynomial time approximation scheme and two 3-approximation algorithms. Hnich et al. [8] introduced an integer programming formulation, a constraint programming formulation, and a hybrid model and solved also one instance which consists of a subset of orders of the original instance. A first optimal solution of the original instance (total loss of zero) was given by Gargani and Refalo [6] using a large neighborhood search heuristic. Van Hentenryck and Michel [7] introduced a constraint programming model which can be used to solve the original instance using a heuristic approach. All these models, however, are not capable to solve all instances of the steel mill slab library [11] which was created after the original instance was solved.

Kalagnanam et al. [9] and Forrest et al. [3] studied a closely related binpacking problem called the *multiple knapsack problem with color constraints*. The problem provides another view on the same industrial application as the steel mill slab problem. The problem input consists of m slabs, each slab j coming with a capacity $k_j \in \mathbb{R}$, and n items, each item i coming with a size $s_i \in \mathbb{R}$, a color $c_i \in \mathbb{N}$, and a specification in form of a subset of slabs indicating from which slabs this item can be manufactured. We say that an item is *feasible* for a slab if the item can be manufactured from it. The goal is to find an assignment such that each slab contains feasible items of at most two different colors, the capacities of the slabs are respected, and the unused capacity of the used slabs is minimized. For this problem, Kalagnanam et al. [9] presented a compact integer programming formulation, while Forrest et al. [3] designed a simple column generation approach. Their computational results indicate that the column generation method is superior in practice.

One main reason why these two binpacking problems are hard to solve in practice is that the used models, excepted the column generation model of Forrest et al. [3], are symmetric. In these models, orders are explicitly assigned to slabs, and therefore, symmetry naturally arises by permutations of the used slabs. It is well known, for instance, that symmetry causes branch-and-bound algorithms to perform poorly, since the resulting problems change only marginally after branching, see Barnhart et al. [1]. In principle, one can respond to this difficulty by either adding symmetry breaking constraints to the given model or by avoiding such a symmetric model in advance. The first strategy were pursued by the most authors. Van Hentenryck and Michel [7] partly broke symmetry using a customized search routine. Other symmetry breaking techniques are discussed in [5]. The column generation approach of Forrest et al. [3], however, provides a model that avoids this kind of symmetry, which obviously explains the performance of their column generation algorithm.

In the following section, we adopt the column generation approach of Forrest et al. [3] for the steel mill slab problem.

3 Branch and price approach

Adapting the column generation approach of Forrest et al. [3], we obtain a similar model to solve the steel mill slab problem. This model does not contain the kind of symmetry mentioned in the previous section.

Let S be the set of all feasible packings of a slab. A packing s is an assignment vector $\lambda_s \in \{0,1\}^n$. This vector defines which orders belong to packing s. This means, order $j \in \{1,\ldots,n\}$ belongs to packing s if $(\lambda_s)_j$ is one. A packing is *feasible* if the total size is not greater than the largest available capacity and if s contains orders of at most two different color classes. Each packing s comes with an unique leftover l_s. Introducing for each feasible packing $s \in S$ a binary decision variable u_s which is one if s is used and zero otherwise, we can formulate the steel mill slab problem as an integer program, i.e., a set partitioning problem, as follows:

$$
\begin{aligned}
\min \quad & \sum_{s\in S} l_s u_s \\
\text{subject to} \quad & \sum_{s\in S} (\lambda_s)_j u_s = 1 && \forall j \in [n] \\
& u_s \in \{0,1\} && \forall s \in S,
\end{aligned}
$$

where $[n] := \{1,\ldots,n\}$. The objective function, is to minimize the total leftover. The equalities are set partitioning constraints to ensure that for each order j exactly one packing s is chosen. Finally, the last conditions state that all variables are binary. Note that, through the equalities all variables are implicitly binary. Hence, the upper bound constraints for the variables u_s can be ignored in the linear programming (LP) relaxation. This has an advantageous for the dual formulation for the LP-relaxation of the above integer program.

In contrast to the setting of Forrest et al. [3] we consider the case that all orders must be covered. This is simply reflected by the transition from packing to partitioning constraints. As a result we focus on pure minimizing of the total leftover whereas Forrest et al. [3] additionally consider to maximize satisfied orders, i.e., they combine both goals in one objective function. We propose, however, the same solution methodology to cope with such formulations.

Since the number of columns can become quite large, an integer program like the one above is usually solved with a branch-and-price algorithm. We assume the reader to be familiar with this method. Clearly, the performance of this algorithm heavily depends on the used branching rules and subroutines for solving the pricing problem.

At the root node of the branch-and-bound tree, an optimal solution of the LP-relaxation of the master problem has to be found. Using column generation, one starts with a feasible basis solution. In our case a basis is obtained, for instance, by setting $u_s := 1$ for all $s \in S'$ and $u_s := 0$ otherwise, where $S' \subset S$ is the set of feasible packings which only contain one order. Then, one successively improves the current solution of the restricted LP-relaxation. This is done by finding a non-basis variable with negative reduced cost with respect to the current dual solution to the LP-relaxation of restricted problem. Denoting by π_j the value of the dual variable associated with order $j \in [n]$, this problem, usually called *pricing problem*, can be modeled as follows:

$$
\begin{array}{lll}
\min & \displaystyle\sum_{i=1}^{m} y_i k_i - \sum_{j=1}^{n} s_j x_j - \sum_{j=1}^{m} \pi_j x_j & \\
\text{subject to} & \displaystyle\sum_{j=1}^{n} s_j x_j \leq \sum_{i=1}^{m} y_i k_i & (1)\\
& \displaystyle\sum_{i=1}^{m} y_i = 1 & (2)\\
& \displaystyle\sum_{c \in C} v_c \leq 2 & (3)\\
& x_j - v_c \leq 0 \qquad \forall c \in C,\ \forall j \in [m] : c_j = c & (4)\\
& x_j \in \{0,1\} \qquad \forall j \in [n] & \\
& y_i \in \{0,1\} \qquad \forall i \in [m] & \\
& v_c \in \{0,1\} \qquad \forall c \in C. &
\end{array}
$$

Here, the binary decision variables x define which order belongs to the packing. The binary decision variables y define the chosen capacity for this packing and equality (2) make sure that exactly one capacity is selected. The artificial binary variables v are used to ensure that at most two color classes are part of the packing. This is handled via the constraint (3) and the coupling constraints (4). Constraint (1) captures the fact that the chosen items do not exceed the selected capacity. Finally, the objective function is to minimize the reduced cost. Note that in case that the objective value of an optimal solution to this pricing problem is larger than zero the column generation is finished.

We want to point out explicitly that it is possible to decompose this pricing problem formulation into smaller subproblems. That is $\frac{1}{2}m|C|(|C|-1)$ knapsack problems, i.e., for each capacity and pair of colors one separate problem. Although knapsack problems are NP-hard, they can be solved very efficiently by dynamic programming in practice [10].

After solving the LP-relaxation of the master problem using column generation, we propose a branch-and-price routine to find and prove an optimal integer solution. The computational results to this model for the instances of the steel mill slab library, however, show that none of these sophisticated techniques is required to solve these problems to optimality.

4 Computational results

The introduced formulation is usually solved with a branch-and-price algorithm, since the number of variables can be huge. This, however, is not the case for the instances of the steel mill slab library. These instances have a number of binary variables between 7103 and 10011. Therefore, all variables can be generated, i.e., all feasible packings can be enumerated, in advance.

We used CPLEX 12.1.0 to solve the resulting integer programs. All computations were performed on computers with an Intel Core 2 Extreme CPU X9650 with 3 GHz, 6 MB cache, and 8 GB of RAM. We used the deterministic parallel mode with 4 threads of CPLEX. The remaining parameters a kept at their default values.

We applied our model to the 380 instances of the steel mill slab library [11]. The results are summarized in Table 1. Each row represents one capacity class which is indicated by the number of available capacities in this class which is given in the first column. The other columns state for the 20 (ordered) instances, which correspond to a capacity class, the optimal objective value. Values written in italic font indicate an improvement to the previous best known solution. Overall we improved 76 instances and proved for all instances optimality.

The running time for these instances were around one second except for four instances of the capacity class 2. Instance 5 took 192.0 seconds, instance 6 run 317.0 seconds, instance 8 needed 11.1 seconds, and instance 15 required 22.6 seconds.

5 Conclusion

We introduced a very simple integer programming model which can be used to solve the steel mill slab problem. The main feature of the proposed model is that all the naturally arising symmetries are removed. As a result, we can solve all instances of the steel mill slab library very efficiently. This approach is superior to the previous techniques applied to this problem.

Table 1. Results for the steel mill slab instances [11].

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
2	*22*	*54*	*100*	*34*	*15*	*36*	*40*	*42*	531	76	*66*	*64*	*19*	*78*	*44*	296	*56*	*155*	*36*	*36*
3	*5*	*15*	*10*	*14*	*7*	35	*11*	*39*	*63*	*155*	*39*	*14*	*6*	*19*	*15*	45	*35*	*8*	*22*	*17*
4	*32*	*18*	*10*	*7*	8	6	*6*	3	1	*12*	*13*	*8*	*1*	*19*	1	*11*	*15*	0	5	*12*
5	0	*21*	*5*	*1*	*9*	*8*	0	0	*1*	*2*	*7*	*5*	*17*	*7*	2	*10*	5	*11*	*15*	0
6	0	*19*	*0*	0	0	*1*	*0*	0	0	1	*0*	7	0	*12*	*2*	*3*	0	0	0	*0*
7	0	0	*1*	0	*1*	*2*	0	*1*	0	0	7	0	2	4	0	0	0	1	0	1
8	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	*3*	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

References

1. C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. Savelsbergh, and P. H. Vance, *Branch-and-price: Column generation for solving huge integer programs.*, Operations Research, 46 (1998), pp. 316–329.
2. M. Dawande, J. Kalagnanam, and J. Sethuraman, *Variable sized bin packing with color constraints*, Electronic Notes in Discrete Mathematics, 7 (2001), pp. 154–157.
3. J. J. H. Forrest, J. Kalagnanam, and L. Ladányi, *A column-generation approach to the multiple knapsack problem with color constraints*, INFORMS Journal on Computing, 18 (2006), pp. 129–134.
4. A. M. Frisch, I. Miguel, and T. Walsh, *Modelling a steel mill slab design problem*, in Proceedings of the IJCAI-01 Workshop on Modelling and Solving Problems with Constraints, 2001, pp. 39–45.
5. ———, *Symmetry and implied constraints in the steel mill slab design problem*, in Proceedings of CP'01 Workshop on Modelling and Problem Formulation, 2001, pp. 8–15.
6. A. Gargani and P. Refalo, *An efficient model and strategy for the steel mill slab design problem*, in Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings, 2007, pp. 77–89.
7. P. V. Hentenryck and L. Michel, *The steel mill slab design problem revisited*, in Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 5th International Conference, CPAIOR 2008, Paris, France, May 20-23, 2008, Proceedings, vol. 5015 of LNCS, 2008, pp. 377–381.
8. B. Hnich, Z. Kiziltan, I. Miguel, and T. Walsh, *Hybrid modelling for robust solving*, Annals of Operations Research, 130 (2004), pp. 19–39.
9. J. R. Kalagnanam, M. W. Dawande, M. Trumbo, and H. S. Lee, *The surplus inventory matching problem in the process industry*, Operations Research, 48 (2000), pp. 505–516.
10. S. Martello, D. Pisinger, and P. Toth, *Dynamic programming and strong bounds for the 0-1 knapsack problem*, Management Science, 45 (1999), pp. 414–424.
11. *Steel mill slab library*. `http://becool.info.ucl.ac.be/steelmillslab`.