Constraints (2012) 17:39–50
DOI 10.1007/s10601-011-9113-8

LETTER

Solving steel mill slab design problems

Stefan Heinz · Thomas Schlechte · Rüdiger Stephan · Michael Winkler

Published online: 18 October 2011

Abstract The steel mill slab design problem from the CSPLIB is a combinatorial optimization problem motivated by an application of the steel industry. It has been widely studied in the constraint programming community. Several methods were proposed to solve this problem. A steel mill slab library was created which contains 380 instances. A closely related binpacking problem called the multiple knapsack problem with color constraints, originated from the same industrial problem, was discussed in the integer programming community. In particular, a simple integer program for this problem has been given by Forrest et al. (INFORMS J Comput 18:129–134, 2006). The aim of this paper is to bring these different studies together. Moreover, we adapt the model of Forrest et al. (INFORMS J Comput 18:129–134, 2006) for the steel mill slab design problem. Using this model and a state-of-the-art integer program solver *all* instances of the steel mill slab library can be solved efficiently to optimality. We improved, thereby, the solution values of 76 instances compared to previous results (Schaus et al., Constraints 16:125–147, 2010). Finally, we consider a recently introduced variant of the steel mill slab design problem, where within all

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

S. Heinz (✉) · T. Schlechte · M. Winkler
Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany
e-mail: heinz@zib.de

T. Schlechte
e-mail: schlechte@zib.de

M. Winkler
e-mail: michael.winkler@zib.de

R. Stephan
Technische Universität Berlin, Institut für Mathematik, Straße des 17. Juni 136,
10623 Berlin, Germany
e-mail: stephan@math.tu-berlin.de

solutions which minimize the leftover one is interested in a solution which requires a minimum number of slabs. For that variant we introduce two approaches and solve all instances of the steel mill slab library with this slightly changed objective function to optimality.

Keywords Steel mill slab design problem · Multiple knapsack problem with color constraints · Integer programming · Set partitioning · Binpacking with side constraints

1 Introduction

The *steel mill slab design problem* is motivated by a real world application from the steel industry. Mathematically, the problem consists of a set of $n \in \mathbb{N}$ 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 $K := \{k_1, \ldots, k_m\} \subset \mathbb{N}$ of $m \in \mathbb{N}$ capacities. 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. Recently, Schaus et al. [14] considered a variation of the problem by adding a second criteria to the objective. Within all solutions, which minimize the leftover, one searches for a solution with a minimal number of used slabs.

The steel mill slab design 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 [15]. 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. 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; the 19 classes are ranging from having 2 to 20 possible capacities. For more details about the generation of these instances and the library we refer to [14].

Outline In the following section, we give a brief overview on different approaches to solve the steel mill slab design problem and related binpacking problems. We recall among others a set packing formulation [4] to the so-called *multiple knapsack problem with color constraints* which can be perceived as a slight generalization of the steel mill slab design problem. In Section 3, we adapt this model to the steel mill slab design problem and the variant where also the number of used slabs is minimized. In Section 4 we report on our computational results. Using a state-of-the-art integer program solver, we solved *all* instances of the steel mill slab library and the original instance to optimality improving the solution value of 76 instances. Moreover, we solved all these instances for the above-mentioned modification of the problem to optimality as well.

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

2 Related work

In the past, several different models have been proposed to solve the steel mill slab design problem. A first set of constraint programming models was presented by Frisch et al. [5] and first computational results for a (small) subset of orders of the original instance were given by the same authors in [6]. Dawande et al. [3] presented an asymptotic polynomial time approximation scheme and two 3-approximation algorithms. Hnich et al. [10] 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 [7] using a large neighborhood search heuristic. Van Hentenryck and Michel [16] introduced a constraint programming model which can be used to solve the original instance using a heuristic approach. Recently, Schaus et al. [14] presented a collection of different constraint-based solving techniques for this problem and introduced the steel mill slab library [15]. All previously used models and solving techniques are not capable of solving all instances of the steel mill slab library.

Kalagnanam et al. [11] and Forrest et al. [4] 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 design 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 *valid* for a slab if the item can be manufactured from it. The goal is to find an assignment such that each slab contains valid 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. [11] presented a compact integer programming formulation, while Forrest et al. [4] gave a set packing formulation and designed a simple column generation approach. Their computational results indicate that this method is superior in practice. They solved an instance with 439 orders, 347 different colors, and 24 slabs (two having the same capacity). Due to the additional assignment restrictions, a slab has a restricted set of items which can be manufactured from it. The number of different colors of the associated valid items is at most 222. This instance is called `mkc` and is part of the MIPLIB2010 [12]. Interpreting,[2] this instance w.r.t. the steel mill slab design problem, it consists of 439 orders, 23 different slab capacities, and 347 colors. Forrest et al. [4] tried to solve an even larger instance called `mkc7`. This instance has 74 slabs, 9,484 orders, and 233 colors within the context of the multiple knapsack problem with color constraints. This boils down to 70 different slab capacities and 642 colors w.r.t. the steel mill slab design problem. See the Appendix of [9] for a more detailed description of the transformation and the particular problem instances in the context of steel mill slab design.

One main reason why these two binpacking problems are hard to solve in practice is that the used models, with the exception of the set packing formulation of Forrest

[2] We ignored the assignment restrictions and allowed an arbitrary number of slabs of each capacity.

et al. [4], are symmetric. In these models, orders are explicitly assigned to slabs, and therefore, symmetry naturally arises by permuting 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. [2]. 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 was pursued by several authors. Van Hentenryck and Michel [16] partly broke symmetry using a customized search routine. Other symmetry breaking techniques are discussed in [6]. The set packing formulation of Forrest et al. [4], however, provides a model that avoids this kind of symmetry, which is one explanation for the performance of their column generation algorithm.

3 Integer programming formulation

Adapting the set packing formulation of Forrest et al. [4], we obtain an integer programming formulation for the steel mill slab design problem. This model does not contain the kind of symmetry mentioned in the previous section.

Let S be the set of all feasible slab designs. A slab design s is an assignment vector $\lambda_s \in \{0,1\}^n$. This vector defines which orders belong to this particular slab design s. This means, order $j \in \{1,\dots,n\}$ belongs to slab design s if $(\lambda_s)_j$ is one. A slab design is *feasible* if the total order size is not greater than the largest available capacity and if s contains orders of at most two different color classes. Each slab design s comes with an unique leftover l_s which is given by

$$l_s = \min\left\{k \in K \mid k \geq \sum_{j=1}^{n} (\lambda_s)_j\right\} - \sum_{j=1}^{n} (\lambda_s)_j.$$

Introducing for each feasible slab design $s \in S$ a binary decision variable x_s which is one if s is used and zero otherwise, we can formulate the steel mill slab design problem as an integer program:

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

This is a set partitioning problem. The objective is to minimize the total leftover. The equalities are set partitioning constraints to ensure that for each order j exactly one slab design s is chosen. Finally, the last conditions state that all variables are binary.

In contrast to the setting of Forrest et al. [4] 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. [4] additionally consider to maximize satisfied orders, i.e., they combine both goals in one objective function. In general we would propose the same solution methodology as Forrest et al. [4] to cope with such formulations. Since the number of columns/slab designs can become quite large, an integer program like

the one above is usually solved with a branch-and-price [2] algorithm. Checking the instances of the steel mill slab library revealed, however, that these instances have between 7,103 and 10,011 feasible slab designs. Therefore, all variables can be generated, i.e., all feasible slab designs can be enumerated, in advance. In [8] a branch-and-price approach for the steel mill slab design problem is briefly discussed.

The set partitioning Model (1) can also be used for the problem of finding within the solutions with minimal leftover one which additionally uses a minimal number of slabs. To cope with this tie breaker rule for different optimal solutions, we can use a simple sequential approach or integrate that tie breaker rule directly in to the first model.

The sequential approach works as follows. First, we solve Model (1) to compute the minimal leftover, say $L^\star$. Then we add a knapsack constraint to limit the total leftover by $L^\star$, and change the objective function to min $\sum_{s\in S} x_s$ which leads to:

$$\begin{aligned} \min \quad & \sum_{s\in S} x_s && && (2)\\ \text{subject to} \quad & \sum_{s\in S} (\lambda_s)_j x_s = 1 && \forall j \in \{1,\ldots,n\} \\ & \sum_{s\in S} l_s x_s \le L^\star \\ & x_s \in \{0,1\} && \forall s \in S. \end{aligned}$$

Solving this integer program, knowing $L^\star$, gives us a solution with minimal leftover which uses the minimum number of slabs. Note that the knapsack constraint can be replaced by an equation. Computational experiments showed that this performed with a similar efficiency as the above knapsack version.

As mentioned above, before one also can integrate the additional optimization criterion directly into the first model. Since n orders require at most n slabs to be served, we can slightly manipulate the objective function by adding $\frac{1}{2n}$ to each coefficient. Then, each used slab design contributes a fixed amount to the objective value independently of the leftover. Therefore, a solution with minimum leftover which uses fewer slabs is cheaper then one which uses more slabs. Using this known technique results in the following integer program:

$$\begin{aligned} \min \quad & \sum_{s\in S} \left(l_s + \tfrac{1}{2n}\right) x_s && && (3)\\ \text{subject to} \quad & \sum_{s\in S} (\lambda_s)_j x_s = 1 && \forall j \in \{1,\ldots,n\} \\ & x_s \in \{0,1\} && \forall s \in S. \end{aligned}$$

In the following section we present computational results for the steel mill slab library in its original formulation and the variant which also minimizes the number of slabs.

4 Computational results

In this section we present our computational studies. We used IBM ILOG 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 IBM ILOG CPLEX. The remaining parameters are kept at their default values. As test set we chose the recently established steel mill slab library [15]. This library contains 380 steel mill slab design instances. We first present the overall results for the original steel mill slab version and its variant. Second, we present more detailed performance results for the different models and selected instances.

4.1 Overall results

For each instance we generated all feasible slab designs in advance. For the instances of the steel mill slab library this took a negligible amount of time (at most 0.02 s). The overall results for the minimization of the leftover are summarized in Table 1. The rows represent capacity classes, each of them consisting of 20 problem instances. The first column, indexed by "$|K|$", states the number of available capacities in this class. The other columns, indexed from 0 to 19, list the *optimal* objective values (leftover) of the 20 (ordered) instances of the corresponding capacity class. Values written in bold italic font indicate an improvement to the previous best known solutions [14]. Overall we improved 76 instances and proved for *all* instances optimality. The running time, which does not included the time for generating all feasible slab designs, for these instances were around one second each, except for five instances of the capacity class 2. Instance 2 required 2.5 s, instance 5 took 101.5 s, instance 6 ran in 174.8 s, instance 8 needed 7.2 s, and instance 15 required 14.6 s (see Table 4). In these cases most of the time is spend for proving optimality.

Table 2 gives the overall results for all instances of the steel mill slab library for which we additionally minimized the number of slabs. The columns of this table are arranged in the same fashion as in the previous table. The values state for each instance the minimum number of used slabs (w.r.t. the minimum leftover). Again, all problems of the steel mill slab library are solved to optimality.

4.2 Detailed results

In what follows we give some more insights on the different models. Table 3 summarizes performance results for each capacity class and model. Thereby, the first column "$|K|$" shows the capacity class by stating the number of available capacities. Followed by pairs of columns for the results of Models (1) and (2), the sequential solving approach (Models (1) + (2)) to find within all solutions with minimal leftover one which uses a minimal number of slabs, and the integrated Model (3) which solves the secondary slab minimization directly. For performance measures we choose the shifted geometric mean[3] for the number of search "Nodes" and for the running

[3]The shifted geometric mean of values $t_1, \ldots, t_n$ is defined as $\left(\prod(t_i + s)\right)^{1/n} - s$ with shift s. We use a shift $s = 10$ for time and $s = 100$ for nodes.

Table 1 Optimal leftover for all instances of the steel mill slab library [15]

	Instance																			
$\|K\|$	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

Table 2 Minimal number of used slabs for all instances of the steel mill slab library [15]

	Instance																			
$\|K\|$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
2	58	51	48	50	50	48	50	52	47	48	52	54	51	56	56	47	50	47	49	55
3	59	55	52	53	52	51	51	53	58	47	55	50	53	52	48	53	53	56	57	59
4	54	53	50	50	47	50	49	51	52	51	51	52	51	48	55	57	55	50	52	52
5	57	56	56	51	51	53	55	49	51	53	49	50	51	54	52	52	52	52	48	51
6	51	50	51	47	47	51	54	48	49	50	52	51	50	52	53	52	52	48	47	53
7	56	48	53	47	50	49	47	50	47	51	54	49	48	51	47	48	50	55	47	47
8	47	50	52	47	51	48	48	47	51	49	51	47	50	49	47	49	48	49	50	47
9	51	50	47	48	47	48	47	49	48	47	50	47	51	47	47	50	49	47	55	48
10	47	47	49	47	48	47	48	47	47	47	51	47	47	47	49	48	49	50	47	47
11	47	47	47	47	47	47	48	51	48	47	48	47	47	49	47	47	47	48	48	47
12	47	48	47	47	47	50	49	47	56	47	47	47	47	47	48	47	47	47	47	47
13	47	47	47	47	47	47	47	50	47	47	47	47	47	47	47	47	47	47	47	47
14	47	47	47	47	47	48	47	47	47	47	47	47	47	47	47	47	47	49	47	47
15	47	47	47	47	47	47	47	47	47	47	47	47	47	47	47	48	47	47	47	47
16	47	47	47	47	47	47	47	47	47	48	47	47	47	47	47	47	47	47	47	47
17	47	47	47	47	47	47	47	47	47	47	47	47	47	47	47	47	47	47	47	47
18	47	47	47	47	47	47	47	47	47	47	47	47	47	47	47	47	47	47	47	47
19	47	47	47	47	47	47	47	47	47	47	47	47	47	47	47	47	48	47	47	47
20	47	47	47	47	47	47	47	47	47	47	47	47	47	47	47	47	47	47	47	47

Table 3 Number of search nodes and running times for all models summarized for the different capacity classes of the steel mill slab library [15]

	Model (1)		Model (2)		Model (1) + (2)		Model (3)	
$\|K\|$	Nodes	Time (s)	Nodes	Time (s)	Nodes	Time (s)	Nodes	Time (s)
2	1,285	5.0	179	2.8	1,656	8.2	2,178	11.1
3	86	0.3	41	0.9	117	1.2	185	0.6
4	43	0.2	49	0.7	91	0.9	325	0.5
5	1	0.1	23	0.3	25	0.4	115	0.3
6	15	0.2	6	0.2	21	0.3	50	0.3
7	1	0.1	2	0.1	3	0.2	71	0.3
8	1	0.1	1	0.1	2	0.2	77	0.3
9	1	0.1	2	0.1	3	0.2	37	0.3
10	1	0.1	2	0.1	3	0.2	78	0.3
11	1	0.1	1	0.1	2	0.2	29	0.2
12	1	0.1	1	0.1	2	0.2	56	0.3
13	1	0.1	1	0.1	2	0.2	62	0.3
14	1	0.1	1	0.1	2	0.2	54	0.3
15	1	0.1	1	0.1	2	0.2	20	0.2
16	1	0.1	1	0.1	2	0.2	58	0.3
17	1	0.1	1	0.1	2	0.2	12	0.2
18	1	0.1	1	0.1	2	0.1	10	0.2
19	1	0.1	1	0.1	2	0.2	33	0.2
20	1	0.1	1	0.1	2	0.2	32	0.3

"Time" in seconds. The *shifted geometric mean* has the advantage that it reduces the influence of outliers. The geometric mean ensures that hard instances are prevented of having a huge impact on the measures. Similar shifting reduces the bias of easy instances, those solved in less than 10 s and/or less than 100 nodes. Note that the measures for the columns related to Models (1) + (2) are computed by first adding the measure values of Models (1) and (2) for each instance and then applying the shifted geometric mean to these values. For a detailed discussion about different measures we refer to [1].

All instances belonging to a class with more than two capacities are easy to solve independently of the chosen model. Regarding the running times all models need less than one second w.r.t. the shifted geometric mean except for the sequential approach (Models (1) + (2)) for capacity class 3 which takes 1.2 s. The number of visited search nodes reveals that for larger capacity classes almost no search is required. That means problems are solved in the root node of the search tree. Concerning the sequential and integrated approaches for additionally minimizing the number of used slabs, it makes (almost) no difference regarding these two methods.

Next, we have a closer look at the results for the capacity class two which are given in Table 4. The columns of this table have almost the same meaning as in Table 3 except that we are stating the real number of search nodes and running times in seconds. The table shows that within this capacity class there are only a few instances which are slightly harder. For the sequential and integrated approaches to minimize, in addition, the number of used slabs, we note that for the instances 5, 8, 14, and 18 the sequential method is superior to the integrated model. In case of the instances 11, 12, 15 and 17, however, the integrated approach slightly dominates the sequential method.

Table 4 Individual results for the instances from capacity class $|K| = 2$ of the steel mill slab library [15] and the steel mill slab version of the mkc instance

	Model (1)		Model (2)		Model (1) + (2)		Model (3)	
Inst.	Nodes	Time (s)	Nodes	Time (s)	Nodes	Time (s)	Nodes	Time (s)
0	150	0.2	1	0.5	151	0.7	1,274	0.4
1	507	0.6	520	1.6	1,027	2.2	555	0.7
2	541	2.5	1	0.6	542	3.1	544	3.0
3	531	1.1	529	5.5	1,060	6.6	534	1.0
4	1	0.1	1	0.4	2	0.5	39	0.1
5	226,348	101.5	1	0.6	226,349	102.1	1,850,052	1149.5
6	367,617	174.8	530	2.3	368,147	177.1	310,277	195.7
7	520	0.8	1	0.7	521	1.5	527	0.8
8	3,421	7.2	1	1.0	3,422	8.2	275,385	402.5
9	367	0.6	1	0.2	368	0.8	1	0.4
10	375	0.3	88	1.0	463	1.3	496	0.7
11	1,024	0.5	536	4.1	1,560	4.6	66	0.3
12	526	0.9	1,183	6.5	1,709	7.4	1,109	1.4
13	713	0.6	1	0.4	714	1.0	565	0.7
14	1,589	1.0	540	3.1	2,129	4.1	55,464	18.2
15	10,569	14.6	7,906	41.2	18,475	55.8	555	2.9
16	530	0.7	157	2.0	687	2.7	1,480	1.7
17	1,012	2.0	1,402	9.7	2,414	11.7	830	2.1
18	517	2.3	1	0.5	518	2.8	21,165	14.9
19	509	0.2	1	0.2	510	0.4	206	0.2
mkc[4]	13,041	119.2	496	136.1	13,537	255.3	207,003	639.7

Finally, we consider the two instances mkc and mkc7 (see [4]). Our approach works for the smaller mkc instance which consists in the steel mill slab context of 439 orders, 23 (different) capacities, 347 colors. There exist 140,223 feasible slab designs which lead in our models to the same number of binary variables. It takes 0.2 s to generate these variables. Table 4 shows in the last line the performance results for this particular instance (excluding the problem generation time). The minimal leftover is 48.32 which requires at least 191 slabs. In case of the much larger instance mkc7 our static approach already failed to generating all feasible slab designs. There exist more than 12 billion slab designs. To cope with that issue a branch-and-price approach using the introduced set partitioning models as basis could be a promising approach, see for example [8].

5 Conclusion

We utilized a standard integer programming model to solve the steel mill slab design problem. An advantage of the proposed model is that the naturally arising symmetries are removed. We solved all instances of the steel mill slab library efficiently. This

[4]Note that in this paper the mkc instance is not equivalent to the correspond instance in the MIPLIB2010 [12]. The MIPLIB2010 version just gave the data input for the steel mill slab design problem which is consider in this paper.

approach is superior to all previous techniques applied to this problem. Furthermore, we showed that the recently introduced variant of that problem, which additionally minimizes the number of used slabs, can be easily incorporated into our integer programming approach. Again, all instances of the steel mill slab library with that secondary objective criteria are solved efficiently to optimality. Besides these instances which all consists of 111 orders we solved the steel mill slab version of `mkc` which contains 439 orders.

All results state that the current steel mill slab library needs an update. Instances with 111 orders can be solved with the introduced integer programming models using a general purposes state-of-the-art solver efficiently.

The introduced models, however, are not scalable. They have the drawback that the number of required variables increases rapidly if there are more orders to be placed. For example, in case of the `mkc7` instance, which has more than 12 billion feasible slab designs, we were not able to create all feasible slab designs in advance. This challenge, however, can be overcome using a branch-and-price approach which generates feasible slab designs only on demand and by reducing the number of required slab designs by considering dominance between one slab design and m others. For the latter one we refer to Prestwich and Beck [13].

Acknowledgements We are grateful to valuable remarks of the editor and the three anonymous reviewers which enhanced this paper.

References

1. Achterberg, T. (2007). *Constraint integer programming*. PhD thesis, Technische Universität Berlin.
2. Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P., & Vance, P. H. (1998). Branch-and-price: Column generation for solving huge integer programs. *Operations Research, 46*, 316–329.
3. Dawande, M., Kalagnanam, J., & Sethuraman, J. (2001). Variable sized bin packing with color constraints. *Electronic Notes in Discrete Mathematics, 7*, 154–157.
4. Forrest, J. J. H., Kalagnanam, J., & Ladányi, L. (2006). A column-generation approach to the multiple knapsack problem with color constraints. *INFORMS Journal on Computing, 18*, 129–134.
5. Frisch, A. M., Miguel, I., & Walsh, T. (2001). Modelling a steel mill slab design problem. In *Proceedings of the IJCAI-01 workshop on modelling and solving problems with constraints* (pp. 39–45).
6. Frisch, A. M., Miguel, I., & Walsh, T. (2001). Symmetry and implied constraints in the steel mill slab design problem. In *Proceedings of CP'01 workshop on modelling and problem formulation* (pp. 8–15).
7. Gargani, A., & Refalo, P. (2007). An efficient model and strategy for the steel mill slab design problem. In C. Bessiere (Ed.), *Principles and practice of Constraint Programming—CP 2007, LNCS* (Vol. 4741, pp. 77–89).
8. Heinz, S., Schlechte, T., & Stephan, R. (2009). *Solving steel mill slab problems with branch-and-price*. ZIB-Report 09-14, Zuse Institute Berlin.
9. Heinz, S., Schlechte, T., Stephan, R., & Winkler, M. (2011). *Solving steel mill slab design problems*. ZIB-Report 11-38, Zuse Institute Berlin.
10. Hnich, B., Kiziltan, Z., Miguel, I., & Walsh, T. (2004). Hybrid modelling for robust solving. *Annals of Operations Research, 130*, 19–39.
11. Kalagnanam, J. R., Dawande, M. W., Trumbo, M., & Lee, H. S. (2000). The surplus inventory matching problem in the process industry. *Operations Research, 48*, 505–516.
12. Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R. E., et al. (2011). MIPLIB 2010. *Mathematical Programming Computation* (Vol. 3).

13. Prestwich, S., & Beck, J. C. (2004). Exploiting dominance in three symmetric problems. In *Fourth international workshop on symmetry and constraint satisfaction problems* (pp. 63–70).
14. Schaus, P., Van Hentenryck, P., Monette, J.-N., Coffrin, C., Michel, L., & Deville, Y. (2010). Solving steel mill slab problems with constraint-based techniques: CP, LNS, and CBLS. *Constraints, 16*, 125–147.
15. Steel mill slab library. http://becool.info.ucl.ac.be/steelmillslab. Accessed Sept 2011.
16. Van Hentenryck, P., & Michel, L. (2008). The steel mill slab design problem revisited. In L. Perron & M. A. Trick (Eds.), *Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2008), LNCS* (Vol. 5015, pp. 377–381).