Orbital Shrinking: a new tool for hybrid MIP/CP methods

Domenico Salvagnin

DEI, University of Padova, salvagni@dei.unipd.it

Abstract. Orbital shrinking is a newly developed technique in the MIP community to deal with symmetry issues, which is based on aggregation rather than on symmetry breaking. In a recent work, a hybrid MIP/CP scheme based on orbital shrinking was developed for the multi-activity shift scheduling problem, showing significant improvements over previous pure MIP approaches. In the present paper we show that the scheme above can be extended to a general framework for solving arbitrary symmetric MIP instances. This framework naturally provides a new way for devising hybrid MIP/CP decompositions. Finally, we specialize the above framework to the multiple knapsack problem. Computational results show that the resulting method can be orders of magnitude faster than pure MIP approaches on hard symmetric instances.

1 Introduction

We consider a integer linear optimization problem P of the form

$$\min cx \tag{1}$$
$$Ax \geq b \tag{2}$$
$$x \in \mathbb{Z}^n_+ \tag{3}$$

For ease of explanation, we assume that the feasible set $f(P)$ of P is bounded and non-empty. Let Π^n be the set of all permutations π of the ground set $I^n = \{1, \ldots, n\}$. The symmetry group G of P is the set of all permutations π_i such that if x is a feasible solution of P then $\pi_i(x)$ is again a feasible solution of P of the same cost. Clearly, G is a permutation group of I^n, i.e., a subgroup of I^n. In addition, G naturally induces a partition $\Omega = \{V_1, \ldots, V_K\}$ of the set of variables of P, called *orbital partition*. Intuitively, two variables x_i and x_j are in the same orbit V_k if and only if there exists $\pi \in G$ such that $\pi(i) = j$. Integer programs with large symmetry groups occur naturally when formulating many combinatorial optimization problems, such as graph coloring, scheduling, packing and covering design.

Symmetry has long been recognized as a curse for the traditional enumeration approaches used in both the MIP and CP communities—we refer to [1,2] for recent surveys on the subject. The reason is that many subproblems in the enumeration tree are isomorphic, with a clear waste of computational resources.

Various techniques for dealing with symmetric problems have been studied by different research communities and the usual approach to deal with symmetry is to try to eliminate it by introducing artificial symmetry-breaking constraints and/or by using ad-hoc search strategies.

In [3], a new technique called *orbital shrinking* for dealing with symmetric problems was presented, which is based on aggregation rather than on symmetry breaking. Let $G = \{\pi^1, \ldots, \pi^M\}$ be the symmetry group of P. Given an arbitrary feasible point $x \in f(P)$, we can construct the average point $\overline{x}$

$$\overline{x} = \frac{1}{M} \sum_{\pi^i \in G} \pi^i(x)$$

Trivially, $c\overline{x} = cx$. It is also easy to prove that $\overline{x}$ must have $\overline{x}_j$ constant within each orbit V_k, and that it can be efficiently computed by taking averages within each orbit, i.e.

$$\overline{x}_j = \frac{1}{|V_k|} \sum_{i \in V_k} x_i \quad \text{where } j \in V_k$$

If P were a convex optimization problem (e.g. a linear program), then $\overline{x}$ would be feasible for P, as it is a convex combination of feasible points of P. Thus, if we wanted to optimize over P, the only unknowns would be the averages within each orbit or, equivalently, their sums $y_k = \sum_{j \in V_k} x_j$, and we could derive an *equivalent* shrunken reformulation Q by

i) introducing sum variables y_k
ii) replacing x_j, $j \in V_k$ with $y_k/|V_k|$ in each constraint and in the objective function.

However, Q is not in general an equivalent reformulation of P when P is an arbitrary integer program, since the average point $\overline{x}$ may not satisfy the integrality requirements. However, it is still possible to prove (see [3] for details) that, if we impose the integrality requirements on the aggregated variables y_k, Q is an equivalent reformulation of the problem obtained from P by relaxing the integrality constraints on x with the surrogate integrality constraints on the sums over the orbits, and thus Q itself is a relaxation of P. Note that the LP relaxation of Q is equivalent to the LP relaxation of P and thus Q cannot be weaker than the standard LP relaxation of P, and can be quite stronger.

Example 1. Let us consider the very tiny Steiner Triple System (STS) instance of size 7

$$\begin{aligned}\min x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 \\ x_1 + x_2 + x_4 \geq 1 \\ x_2 + x_3 + x_5 \geq 1 \\ x_3 + x_4 + x_6 \geq 1 \\ x_4 + x_5 + x_7 \geq 1 \\ x_5 + x_6 + x_1 \geq 1 \\ x_6 + x_7 + x_2 \geq 1 \\ x_7 + x_1 + x_3 \geq 1 \\ x \in \{0,1\}^7\end{aligned}$$

It is easy to see that all variables belong to the same orbit, and thus, after introducing the sum variable $y = \sum_{j=1}^{7} x_j$, the orbital shrinking model reads

$$\begin{aligned}\min y \\ 3/7y \geq 1 \\ y \leq 7 \\ y \in \mathbb{Z}_+\end{aligned}$$

Its linear programming relaxation has value 7/3, which is of course the same value as the LP relaxation of the original model. However, imposing the integrality requirement on y, we can increase the value of the relaxation to 3. In this case, this is also the value of an optimal integral solution of the original model, so orbital shrinking closes 100% of the integrality gap. Unfortunately, this is not always the case, even for instances from the same class. Indeed, if we move to the STS instance of size 9, orbital shrinking is not able to improve over the LP bound of 3, while the optimum is 5. □

The STS example above shows that the orbital shrinking relaxation Q might be a oversimplified approximation of P (in the worst case, reducing to a trivial integer program with only one variable), thus providing no useful information for solving P. This is however not always the case. In some particular cases, Q is indeed an exact reformulation of P, even if P is an integer program. For example, consider a knapsack problem with identical items: an orbital shrinking relaxation would replace the binary variables x_j associated with the identical items with a general integer variable y_k that counts how many items of type k need to be taken in the solution, and this is clearly equivalent to the original formulation (but symmetry free). In other cases, Q, although not a reformulation, still retains enough structure from P such that that solving Q provides useful insights for solving P. Of course the question is how to exploit this information to obtain a sound and complete method for solving P.

A partial answer to this question was given in [4], where a hybrid MIP/CP scheme based on orbital shrinking was developed for the multi-activity shift scheduling problem, which is the problem of covering the demands of a finite set of activities over a time horizon by assigning them to a set of employees. In real-world applications, the set of feasible shifts (i.e., the set of feasible sequences of activity assignments to a single employee) is defined by many regulation constraints. In this case, many constraints of the problem (in particular those formulated with the aid of formal languages, such regular expressions or context-free grammars) are preserved by the shrinking process, while some others are not. In particular, cardinality constraints (e.g., the number of allowed working hours for a single employee in a single day) are replaced by surrogate versions in Q. Still, Q provides a very strong dual bound on P and its aggregated solutions can often be turned into feasible solutions for P. In order to get a complete method, the following strategy was proposed in [4]: solve the orbital shrinking model Q with a black box MIP solver and, whenever an (aggregated) integer feasible solution y^* is found, check with a CP solver if it can be turned into a feasible solution x^* for P. The scheme is akin to a logic-based Benders decomposition [5], although the decomposition is not based on a traditional variable splitting, but on aggregation.

In the present paper we show that the scheme above can be extended to a general framework for solving arbitrary symmetric MIP instances. This framework, described in Section 2, naturally provides a new way for devising hybrid MIP/CP decompositions. Then, in Section 3, we specialize the above framework to the multiple knapsack problem. Computational results in Section 4 show that the resulting method can be orders of magnitude faster than pure MIP approaches on hard symmetric instances. Conclusions are finally drawn in Section 5.

2 A general Orbital Shrinking based decomposition method

Let P be an integer linear program as in the previous section and let G be the symmetry group of P. Note that if G is unknown to the modeler then the whole scheme can be applied starting from a subgroup G' of G, such as, for example, the symmetry group G_{LP} of the formulation, which is defined as

$$G_{LP} = \{\pi \in \Pi^n | \pi(c) = c \wedge \exists \sigma \in \Pi^m \ s.t. \ \sigma(b) = b, A(\pi, \sigma) = A\}$$

where $A(\pi, \sigma)$ is the matrix obtained by A permuting the columns with π and the rows with σ. Intuitively, a variable permutation π defines a symmetry if there exists a constraint permutation σ such that the two together leave the formulation unchanged. Note that if the permutation group G_{LP} is used, also constraints of P are partitioned into *constraint orbits*: in this case, two constraints are in the same orbit if and only if there exists a σ (as defined above) mapping one to the other. G_{LP} can be computed with any graph isomorphism package such as Nauty [6] or Saucy [7], which perform satisfactorily in practice.

Using G (or G_{LP}) we can compute the orbital partition Ω of P and construct the shrunken model Q

$$\min dy \tag{4}$$

$$By \geq r \tag{5}$$

$$y \in \mathbb{Z}_{+}^{K} \tag{6}$$

where $y_k = \sum_{j \in V_k} x_j$ and constraints (5) are obtained from the constraints (2) by replacing each occurrence of variable x_j with $y_k/|V_k|$, where k is the index of the orbit to which x_j belongs. It is easy to show that all constraints in the same orbit will be mapped to the same constraint in Q, so in practice Q has one variable for each variable orbit and one constraint for each constraint orbit in P. In the small STS example of the previous section, all constraints are in the same constraint orbit, and indeed they are all are mapped to $3/7y \geq 1$ in the orbital shrinking reformulation (the other constraint, $y \leq 7$, is derived from the upper bounds of the binary variables). Note that model Q acts like a master problem in a traditional Benders decomposition scheme.

For each integer feasible solution y^* of Q, we can then define the following (slave) feasibility check problem $R(y^*)$

$$Ax \geq b \tag{7}$$

$$\sum_{j \in V_k} x_j = y_k^* \quad \forall k \in K \tag{8}$$

$$x \in \mathbb{Z}_{+}^{n} \tag{9}$$

If $R(y^*)$ is feasible, then the aggregated solution y^* can be *disaggregated* into a feasible solution x^* of P, with the same cost. Otherwise, y^* must be rejected, in either of the following two ways:

1. Generate a *nogood* cut that forbids the assignment y^* to the y variables. As in logic-based Benders decomposition, an ad-hoc study of the problem is needed to derive stronger nogood cuts.
2. Branching. Note that in the (likely) event that the solution y^* is the integral LP relaxation of a node, then branching on non-fractional y variables is needed, and y^* will still be a feasible solution in one of the two child nodes. However, the method would still converge, because the number of variables is finite and the tree has a finite depth. Note that in this case the method may repeatedly check for feasibility the same aggregated solution: in practice, this can be easily avoided by keeping a list (cache) of recently checked aggregated solutions with the corresponding feasibility status.

It is important to note that, by construction, problem $R(y^*)$ has the same symmetry group of P, so symmetry may still be an issue while solving $R(y^*)$. This issue is usually solvable because (i) the linking constraints (8) may make the model much easier to solve and (ii) the (easier) structure of the problem may allow for more effective symmetry breaking techniques. Note also that $R(y^*)$ is a pure feasibility problem, so a CP solver may be a better choice than a MIP solver.

3 Application to the Multiple Knapsack Problem

In the present section, we specialize the general framework of the previous section to the multiple knapsack problem (MKP) [8,9]. This a natural generalization of the traditional knapsack problem [10], where multiple knapsack are available. Given a set of n items with weights w_j and profits p_j, and m knapsacks with capacity C_i, MKP reads

$$\max \sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij} \tag{10}$$

$$\sum_{j=1}^{n} w_j x_{ij} \leq C_i \quad \forall i = 1, \ldots, m \tag{11}$$

$$\sum_{i=1}^{m} x_{ij} \leq 1 \quad \forall j = 1, \ldots, n \tag{12}$$

$$x \in \{0,1\}^{m \times n} \tag{13}$$

where binary variable x_{ij} is set to 1 if and only if item j is loaded into knapsack i. Since we are interested in symmetric instances, we will assume that all m knapsacks are identical and have the same capacity C, and that also some items are identical.

When applied to problem MKP, the orbital shrinking reformulation Q reads

$$\max \sum_{k=1}^{K} p_k y_k \tag{14}$$

$$\sum_{k=1}^{K} w_k y_k \leq mC \tag{15}$$

$$0 \leq y_k \leq |V_k| \quad \forall k = 1, \ldots, K \tag{16}$$

$$y \in \mathbb{Z}_+^K \tag{17}$$

Intuitively, in Q we have a general integer variable y_k for each set of identical items and a single knapsack with capacity mC. Given a solution y^*, the corresponding $R(y^*)$ is thus a one dimensional bin packing instance, whose task is to check whether the selected items can indeed be packed into m bins of capacity C.

To solve the bin-packing problem above, we propose two different approaches. The first approach is to deploy a standard compact CP model based on the global `binpacking` constraint [11] and exploiting the CDBF [12] branching scheme for search and symmetry breaking. Given an aggregated solution y^*, we construct a vector s with the sizes of the items picked by y^*, and sort it in non-decreasing order. Then we introduce a vector of variables b, one for each item: the value of b_j is the index of the bin where item j is placed. Finally, we introduce a variable

l_i for each bin, whose value is the load of bin i. The domain of variables l_i is $[0..C]$. With this choice of variables, the model reads:

$$\text{binpacking}(b, l, s) \tag{18}$$

$$b_{j-1} \leq b_j \quad \text{if } s_{j-1} = s_j \tag{19}$$

where (19) are symmetry breaking constraints.

The second approach is to consider an extended model, akin to the well known Gilmore and Gomory column generation approach for the cutting stock problem [13]. Given the objects in y^*, we generate all feasible packings p of a single bin of capacity C. Let P denote the set of all feasible packings and, given packing p, let a_{pk} denote the number of items of type k picked. The corresponding model is

$$\sum_{p \in P} a_{pk} x_p = y^*_k \tag{20}$$

$$\sum_{p \in P} x_p = m \tag{21}$$

$$x_p \in \mathbb{Z}_+ \tag{22}$$

where integer variables x_p count how many bins are filled according to packing p. In the following, we will denote this model with `BPcg`. Model `BPcg` is completely symmetry free, but it needs an exponential number of columns in the worst case.

4 Computational Experiments

We implemented our codes in C++, using IBM ILOG Cplex 12.4 [14] as black box MIP solver and Gecode 3.7.3 [15] as CP solver. All tests have been performed on a PC with an Intel Core i5 CPU running at 2.66GHz, with 8GB of RAM (only one CPU was used by each process). Each method was given a time limit of 1 hour per instance.

In order to generate hard MKP instances, we followed the systematic study in [16]. According to [16], difficult instances can be obtained introducing some correlation between profits and weights. Among the hardest instances presented in [16] are the so-called *almost strongly correlated* instances, in which weights w_j are distributed—say uniformly—in the range $[1, R]$ and the profits p_j are distributed in $[w_j + R/10 - R/500, w_j + R/10 + R/500]$. These instances correspond to real-life situations where the profit is proportional to the weight plus some fixed charge value and some noise. Given this procedure, a possibility for generating hard-enough instances is to construct instances where the coefficients are of moderate size, but where all currently used upper bounds have a bad performance. Among these difficult classes, we consider the *spanner instances*: these instances are constructed such that all items are multiples of a quite small set of items—the so-called spanner set. The spanner instances $\text{span}(v, l)$ are characterized by the following three parameters: v is the size of the spanner set, l is the multiplier limit, and we may have any distribution of the items in the spanner set. More formally, the instances are generated as follows: a set of v items is generated with weights in the interval $[1, R]$, with $R = 1000$, and profits according to the distribution. The items (p_k, w_k) in the spanner set are normalized by dividing the profits and weights by $l + 1$, with $l = 10$. The n items are then constructed by repeatedly choosing an item (p_k, w_k) from the spanner set, and a multiplier a randomly generated in the interval $[1, l]$. The constructed item has profit and weight (ap_k, aw_k). Capacities are computed as $C = \frac{\sum_{i=1}^{n} w_i}{8}$.

In order to have a reasonable test set, we considered instances with a number of items n in $\{30, 40, 50\}$ and number of knapsacks m in $\{3, 4, 5, 6\}$. For each pair of (n, m) values, we generated 10 random instances following the procedure described above, for a total of 120 instances. All the instances are available from the author upon request. For each set of instances, we report aggregate results comparing the shifted geometric means of the number of branch-and-cut nodes and the computation times of the different methods. Note that we did not use specialized solvers, such as ad-hoc codes for knapsack or bin packing problems, because the overall scheme is very general and using the same (standard) optimization packages in all the methods allows for a clearer comparison of the different approaches.

As a first step, we compared 2 different pure MIP formulations. One is the natural formulation (10)−(13), denoted as `cpxorig`. The other is obtained by aggregating the binary variables corresponding to identical items. The model, denoted as `cpx`, reads

$$\max \sum_{i=1}^{m} \sum_{k=1}^{K} p_j z_{ik} \tag{23}$$

$$\sum_{k=1}^{K} w_j z_{ik} \leq C \quad \forall i = 1, \ldots, m \tag{24}$$

$$\sum_{i=1}^{m} z_{ik} \leq U_k \quad \forall k = 1, \ldots, K \tag{25}$$

$$z \in \mathbb{Z}_{+}^{m \times K} \tag{26}$$

where U_k is the number of items of type k. Note that `cpx` would be obtained automatically from formulation `cpxorig` by applying the orbital shrinking procedure if the capacities of the knapsacks were different. While one could argue that `cpxorig` is a modeling mistake, the current state-of-the-art in preprocessing is not able to derive `cpx` automatically, while orbital shrinking would. A comparison of the two formulations is shown in Table 1. As expected, `cpx` clearly outperforms `cpxorig`, solving 82 instances (out of 120) instead of 65. However, `cpx` performance is rapidly dropping as the number of items and knapsacks increases.

Then, we compared three variants of the hybrid MIP/CP procedure described in Section 3, that differs on the models used for the feasibility check. The first

Table 1. Comparison between `cpxorig` and `cpx`.

		# solved		time (s)		nodes	
n	m	`cpxorig`	`cpx`	`cpxorig`	`cpx`	`cpxorig`	`cpx`
30	3	10	10	1.16	0.26	3,857	1,280
30	4	9	10	12.28	3.42	65,374	16,961
30	5	6	8	291.75	79.82	2,765,978	1,045,128
30	6	7	7	108.83	48.05	248,222	164,825
40	3	9	10	19.48	2.72	103,372	9,117
40	4	8	8	351.07	35.56	3,476,180	421,551
40	5	2	3	2,905.70	1,460.95	25,349,383	23,897,899
40	6	3	5	308.29	234.19	626,717	805,007
50	3	6	9	70.73	12.44	259,099	32,310
50	4	2	7	1,574.34	254.58	8,181,128	4,434,707
50	5	0	2	3,600.00	700.69	26,017,660	4,200,977
50	6	3	3	308.29	307.98	586,400	1,025,907

Table 2. Comparison between hybrid methods.

		# solved			time (s)			nodes		
n	m	`BPstd`	`BPcgCP`	`BPcgMIP`	`BPstd`	`BPcgCP`	`BPcgMIP`	`BPstd`	`BPcgCP`	`BPcgMIP`
30	3	10	10	10	0.07	0.05	0.05	245	270	270
30	4	10	10	10	0.18	0.12	0.08	157	160	160
30	5	10	10	10	1.28	0.26	0.14	90	88	88
30	6	10	10	10	1.24	0.25	0.13	42	40	40
40	3	10	10	10	0.64	0.42	0.17	502	540	540
40	4	10	10	10	0.54	0.20	0.17	225	224	224
40	5	9	10	10	8.63	1.20	0.62	202	225	225
40	6	8	10	10	17.96	1.65	0.46	48	60	60
50	3	10	10	10	1.59	0.93	0.44	837	914	914
50	4	10	10	10	4.06	1.11	0.60	337	335	335
50	5	6	8	10	137.52	23.97	3.58	172	245	335
50	6	7	7	10	17.15	12.73	2.85	17	16	140

variant, denoted by `BPstd`, is based on the compact model (18)−(19). The second and the third variants are both based on the extended model (20)−(22), but differs on the solver used: a CP solver for `BPcgCP` and a MIP solver for `BPcgMIP`. All variants use model (14)−(17) as a master problem, which is feed to Cplex and solved with dual reductions disabled, to ensure the correctness of the method. Cplex callbacks are used to implement the decomposition. A comparison of the three methods is given in Table 2. Note that the number of nodes reported for hybrid methods refers to the master only—the nodes processed to solve the feasibility checks are not added to the count, since they are not easily comparable, in particular when a CP solver is used. Of course the computation times refer to the whole solving process (slaves included). According to the table, even the simplest model `BPstd` clearly outperforms `cpx`, solving 110 instances (28 more) and with speedups up to two orders of magnitude. However, as the number of knapsacks increases, symmetry can still be an issue for this compact model, even though symmetry breaking is enforced by constraints (19) and by CDBF. Replacing the compact model with the extended model, while keeping the same solver, shows some definite improvement, increasing the number of solved instances from 110 to 115 and further reducing the running times. Note that for the instances in our testbed, the number of feasible packings was always manageable (at most a few thousands) and could always be generated by Gecode in a fraction of a second. Still, on some instances, the CP solver was not very effective in solving the feasibility model. The issue is well known in the column generation community: branching on variables x_p yields highly unbalanced trees, because fixing a variable x_p to a positive integer value triggers a lot of propagations, while fixing it to zero has hardly any effect. In our particular case, replacing the CP solver with a MIP solver did the trick. Indeed, just solving the LP relaxation was sufficient in most cases to detect infeasibility. Note that if infeasibility is detected by the LP relaxation of model (20)−(22), then standard LP duality can be used to derive a (Benders) nogood cut violated by the current aggregated solution y^*, without any ad-hoc study. In our implementation, however, we did not take advantage of this possibility, and just stuck to the simpler strategy of branching on integer variables. `BPcgMIP` is able to solve all 120 instances, in less than four seconds (on average) in the worst case. The reduction in the number of nodes is particularly significant: while `cpx` requires millions of nodes for some classes, `BPcgMIP` is always solving the instances in fewer than 1,000 nodes.

Finally, Table 3 shows the average gap closed by the orbital shrinking relaxation with respect to the initial integrality gap, and the corresponding running times (obtained by solving the orbital shrinking relaxation with a black box MIP solver, without the machinery developed in this section). According to the table, orbital shrinking yields a much tighter relaxation than standard linear programming, while still being very cheap to compute.

5 Conclusions

In this paper we presented a general framework for deploying hybrid MIP/CP decomposition methods for symmetric optimization problems. This framework is similar in spirit to logic-based Benders decomposition schemes, but it is based on aggregation rather than on the usual variable splitting argument, thus being applicable to a completely different class of problems. The overall scheme can be obtained as a generalization of a recent approach developed for the multi-activity shift scheduling problem, where it showed significant improvements over previous pure MIP approaches. In order to further test its effectiveness, we specialized the general scheme to the multiple knapsack problem, giving a clear example

Table 3. Average gap closed by orbital shrinking and corresponding time.

n	m	gap closed	time (s)
30	3	45.3%	0.007
30	4	46.6%	0.004
30	5	42.8%	0.004
30	6	54.4%	0.002
40	3	48.4%	0.013
40	4	67.2%	0.007
40	5	55.3%	0.005
40	6	58.6%	0.003
50	3	52.7%	0.031
50	4	64.5%	0.030
50	5	61.1%	0.006
50	6	76.7%	0.003

on how to apply the method in practice and some recommendations on how to solve the possible pitfalls of the approach. Computational results confirmed that the resulting method can be orders of magnitude faster than standard pure MIP approaches on hard symmetric instances.

References

1. Margot, F.: Symmetry in integer linear programming. In Jünger, M., Liebling, T., Naddef, D., Nemhauser, G., Pulleyblank, W., Reinelt, G., Rinaldi, G., Wolsey, L., eds.: 50 Years of Integer Programming 1958-2008. Springer Berlin Heidelberg (2010) 647–686
2. Gent, I.P., Petrie, K.E., Puget, J.F.: Symmetry in constraint programming. In Rossi, F., van Beek, P., Walsh, T., eds.: Handbook of Constraint Programming. Elsevier (2006) 329–376
3. Fischetti, M., Liberti, L.: Orbital shrinking. In Mahjoub, A.R., Markakis, V., Milis, I., Paschos, V.T., eds.: ISCO. Volume 7422 of Lecture Notes in Computer Science., Springer (2012) 48–58
4. Salvagnin, D., Walsh, T.: A hybrid mip/cp approach for multi-activity shift scheduling. In: CP. (2012) 633–646
5. Hooker, J.N., Ottosson, G.: Logic-based Benders decomposition. Mathematical Programming **96**(1) (2003) 33–60
6. McKay, B.D.: Practical graph isomorphism (1981)
7. Katebi, H., Sakallah, K.A., Markov, I.L.: Symmetry and satisfiability: An update. In: Theory and Applications of Satisfiability Testing - SAT 2010, 13th International Conference, SAT 2010, Edinburgh, UK, July 11-14, 2010. Proceedings. Volume 6175. (2010) 113–127
8. Scholl, A., Klein, R., Jürgens, C.: Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. Computers & OR **24**(7) (1997) 627–645
9. Pisinger, D.: An exact algorithm for large multiple knapsack problems. European Journal of Operational Research **114**(3) (1999) 528–541

10. Martello, S., Toth, P.: Knapsack Problems: Algorithms and Computer Implementations. Wiley (1990)
11. Shaw, P.: A constraint for bin packing. In Wallace, M., ed.: CP. Volume 3258 of Lecture Notes in Computer Science., Springer (2004) 648–662
12. Gent, I.P., Walsh, T.: From approximate to optimal solutions: Constructing pruning and propagation rules. In: IJCAI, Morgan Kaufmann (1997) 1396–1401
13. Gilmore, P.C., Gomory, R.E.: A linear programming approach to the cutting-stock problem. Operations Research **9** (1961) 849–859
14. IBM ILOG: CPLEX 12.4 User's Manual. (2011)
15. Gecode Team: Gecode: Generic constraint development environment (2012) Available at `http://www.gecode.org`.
16. Pisinger, D.: Where are the hard knapsack problems? Computers & Operations Research **32** (2005) 2271–2284