Noname manuscript No.
(will be inserted by the editor)

Orbital Shrinking: Theory and Applications

Matteo Fischetti[1] · **Leo Liberti**[2] ·
Domenico Salvagnin[3] · **Toby Walsh**[4]

Received: date / Accepted: date

Abstract Symmetry plays an important role in optimization. The usual approach to cope with symmetry in discrete optimization is to try to eliminate it by introducing artificial symmetry-breaking conditions into the problem, and/or by using an ad-hoc search strategy. This is the common approach in both the mixed-integer programming (MIP) and constraint programming (CP) communities. In this paper we argue that symmetry is instead a beneficial feature that we should preserve and exploit as much as possible, breaking it only as a last resort. To this end, we outline a new approach, that we call orbital shrinking, where additional integer variables expressing variable sums within each symmetry orbit are introduced and used to "encapsulate" model symmetry. This leads to a discrete relaxation of the original problem, whose solution yields a bound on its optimal value. Then, we show that orbital shrinking can be turned into an exact method for solving arbitrary symmetric MIP instances. The proposed method naturally provides a new way for devising hybrid MIP/CP decompositions. Finally, we report computational results on two specific applications of the method, namely the multi-activity shift scheduling and the multiple knapsack problem, showing that the resulting method can be orders of magnitude faster than pure MIP or CP approaches.

Matteo Fischetti
DEI, University of Padova, Italy
E-mail: matteo.fischetti@unipd.it

Leo Liberti
IBM "T.J. Watson" Research Center, Yorktown Heights, 10598 NY, USA, *and*
LIX, École Polytechnique, 91128 Palaiseau, France
E-mail: leoliberti@us.ibm.com, liberti@lix.polytechnique.fr

Domenico Salvagnin
DEI, University of Padova, Italy
E-mail: domenico.salvagnin@unipd.it

Toby Walsh
NICTA and UNSW, Sydney, Australia
E-mail: toby.walsh@nicta.com.au

Keywords Mathematical programming · constraint programming · discrete optimization · symmetry · relaxation · MINLP

.

1 Introduction

Symmetry has long been recognized as a curse for the traditional enumeration approaches used in both the MIP and CP communities—we refer to [22, 9] for recent surveys on the subject. The usual approach to cope with this redundancy source is to destroy symmetry by introducing artificial conditions into the problem, or by using a clever branching strategy such as isomorphism pruning [20,21] or orbital branching [27]. We will outline a different approach, that we call *orbital shrinking*, where additional integer variables expressing variable sums within each orbit are introduced and used to "encapsulate" model symmetry. This leads to a discrete relaxation of the original problem, whose solution yields a bound on its optimal value. The underlying idea here is that we see symmetry as a positive feature of our model, so we want to preserve it as much as possible, breaking it only as a last resort.

While orbital shrinking can, in some cases, provide an exact (and symmetry free) reformulation of the original model, in general the shrunken reformulation yields only a relaxation. Thus, we devise a general decomposition framework to turn orbital shrinking into an exact method for arbitrary symmetric MIPs. The proposed method naturally provides a new way for designing hybrid MIP/CP decompositions, although pure MIP/MIP decompositions are allowed as well.

Finally, we specialize the general framework above to two applications, namely multi-activity shift scheduling and multiple knapsack problems. Although this paper is squarely in the mathematical programming field, following a relatively recent trend we exploit a degree of integration between mathematical and constraint programming [25,12] in the applications of the orbital shrinking theory. Computational results show that the resulting method can be orders of magnitude faster than pure MIP or CP approaches.

The outline of the paper is as follows. In Section 2, we review some main results on symmetry groups in the context of optimization problems. Then, in Section 3, we present orbital shrinking, and show that it yields a relaxation of the original problem. In Section 4 we discuss some features that help mining the best formulation subgroup to apply orbital shrinking with, and show their effect on a set of symmetric MILPs. In Section 5 we describe a general decomposition framework based on orbital shrinking, while in Sections 6 and 7 we specialize the general framework to multi-activity shift scheduling and multiple knapsack problems, also reporting computational results. Conclusions are finally drawn in Section 8.

We assume the reader is familiar with mixed-integer programming, constraint programming and basic group theory. The present paper extends and is based on the preliminary results presented in [4,37,36], by the same authors.

2 Optimization under Symmetry

Let P be an arbitrary mixed-integer nonlinear program of the form

$$\min f(x) \tag{1}$$
$$g_i(x) \leq 0 \quad \forall i \in C \tag{2}$$
$$x_j \in \mathbb{Z} \quad \forall j \in J \tag{3}$$

where $J \subseteq [n] = \{1, \ldots, n\}$ is the subset of integer variables. Without loss of generality, the objective function $f(x)$ is assumed to be convex.

Let $\mathcal{F}(P)$ be the feasible set of P, and $\mathcal{G}(P)$ be the set of global optima of P. In general, deciding whether a permutation of the variables of P fixes $\mathcal{G}(P)$ requires knowledge of $\mathcal{G}(P)$, which is the end goal of solving P anyhow.

We therefore restrict our attention to those variable permutations which fix the formulation of P. Thus, we define the symmetry group G_P of P as the set of permutations $\pi \in S_n$ that leave the formulation of P unchanged, except for a possible reordering of the constraints. Since the latter bears no influence on $\mathcal{F}(P)$ and $\mathcal{G}(P)$, the formulation group is clearly a group of symmetries of P. This definition is easy to implement for linear formulations, be they Linear Programs (LP), Mixed-Integer Linear Programs (MILP), since a permutation of the variables directly translates to a permutation of the columns of the data matrices. For Nonlinear Programs (NLP) and Mixed-Integer Nonlinear Programs (MINLP), we restrict our attention to classes of problems whose formulations involve functions represented by finite mathematical expressions, which are sequences of operators (e.g. sum, difference, product, fraction, power, exponential, logarithm and so on) applied to constants and variables. These expressions are easily represented by trees, and whole NLP/MINLP formulations can be represented by suitable Directed Acyclic Graphs (DAG). G_P is then obtained as a restriction of the automorphism group of this DAG to the set of variable indices of P, see [17] for more details. G_P can be computed by means of any graph isomorphism package such as Nauty [24] or Saucy [16], which perform satisfactorily in practice.

Note that any subgroup G of G_P (or G_P itself) partitions the set of variables into equivalence classes, called *orbits* (i.e., variables that are mapped one to the other by some permutation in G). We denote with Ω_G this orbital partition of $[n]$. Note that, by definition, integer and continuous variables cannot be permuted with each other, so each orbit contains only integer or only continuous variables. Note also that if the formulation group G_P is used, constraints of P are themselves partitioned into equivalence classes, called *constraint orbits*: in particular, two constraints are in the same orbit if and only if one is mapped into the other (because of reordering) when some variable permutation $\pi \in G_P$ is applied. Finally, given a subset $I \subseteq [n]$, we define the subgroup of G_P that keeps any $i \in I$ fixed, i.e., such that $\pi(i) = i$ for all $i \in I$; this is called the *point-wise stabilizer* of G_P w.r.t. I, and denoted as $G_P[I]$.

3 Orbital Shrinking Relaxation

The main objective of this work is to discuss a new relaxation for symmetric problems. Instead of trying to break symmetries, as is standard in the current literature [20–22,27,17], our relaxation can be described as the "formulation modulo a group". We basically replace the original integer variables x by integer variables assigned to each orbit of the action of (a subgroup of) the formulation group on the variable index set. In this section, we will describe how to construct the orbital shrinking reformulation (OSR) of a given optimization problem P, and show that this is indeed a relaxation of the original problem.

Let us consider a partition (V_1, V_2) and (C_1, C_2, C_3) of the variables and constraints of P, respectively, that satisfies the following conditions:

- All constraints in C_1 are convex w.r.t. the variables in V_1. In other words, constraints in C_1 are convex once the variables in V_2 are fixed.
- All constraints in C_2 are functions of sums of variables along the orbits defined by the group $G_P[V_2]$, the point-wise stabilizer of G_P w.r.t. V_2.

The meaning of the constraint partition will become clear in the following; a remark is given after Corollary 2. In what follows, for notational simplicity, we will denote $G_P[V_2]$ by G and $\Omega_{G_P[V_2]}$ by Ω.

Given a problem P and such a partition $(V_1, V_2, C_1, C_2, C_3)$, we define the orbital shrinking reformulation P_{OSR} as the model obtained by the following procedure:

- for each $\omega \in \Omega$, define a variable z_ω. If the orbit ω is made of integer variables, z_ω is integer as well, if not it is continuous. Let $z = (z_\omega \mid \omega \in \Omega)$.
- for each constraint $g_i(x) \leq 0$ in $C_1 \cup C_2$, define a new constraint $\overline{g}_i(z) \leq 0$, obtained from $g_i(x) \leq 0$ through the formal substitution:

$$x_j \to \frac{z_\omega}{|\omega|} \tag{4}$$

  where $j \in \omega$.
- define the objective function $\overline{f}(z)$ applying to $f(x)$ the same formal substitution (4).
- ignore constraints in C_3.

We now show that P_{OSR} is a relaxation of P. We start with a few lemmata.

Lemma 1 *Let ω be an orbit of the action of G on $[n]$ and f any function with* $\operatorname{dom} f = [n]$. *Then*

$$\sum_{\pi \in G} f(\pi(j)) = \frac{|G|}{|\omega|} \sum_{l \in \omega} f(l) \quad \forall j \in \omega. \tag{5}$$

Proof Let $T_{jl} = \{\pi \in G : \pi(j) = l\}$. It is easy to show that $|T_{jl}| = |G[j]|$ for all $l \in \omega$. Indeed, given an arbitrary $\pi \in T_{jl}$, we can define the map $f : G[j] \to T_{jl}$

as $\sigma \to \pi\sigma$. This map is a bijection, with inverse $f^{-1} : \sigma \to \pi^{-1}\sigma$, hence the two sets have the same cardinality. Finally

$$\sum_{\pi \in G} f(\pi(j)) = \sum_{l \in \omega} \sum_{\pi \in T_{jl}} f(\pi(j)) = \sum_{l \in \omega} |T_{jl}| f(l) = \frac{|G|}{|\omega|} \sum_{l \in \omega} f(l),$$

where the last equality is justified by the orbit-stabilizer theorem. □

Lemma 2 *Let x^* be an arbitrary feasible solution of P, and consider the convex combination $\overline{x}$ defined as*

$$\overline{x} = \frac{1}{|G|} \sum_{\pi \in G} \pi(x^*) \tag{6}$$

Then, for each $j \in \omega$, we have

$$\overline{x}_j = \frac{1}{|\omega|} \sum_{l \in \omega} x^*_l. \tag{7}$$

Proof Define a function $X : [n] \to \mathbb{R}$ as $X(j) = x^*_j$. Applying Lemma 1 we get:

$$\overline{x}_j = \frac{1}{|G|} \sum_{\pi \in G} \pi(x^*) = \frac{1}{|G|} \sum_{\pi \in G} X(\pi(j)) = \frac{1}{|\omega|} \sum_{l \in \omega} X(l) = \frac{1}{|\omega|} \sum_{l \in \omega} x^*_l.$$

□

Lemma 3 *Let x^* be an arbitrary feasible solution of P. Then, for any $\pi \in G$ and for any $\omega \in \Omega$*

$$\sum_{j \in \omega} x^*_j = \sum_{j \in \omega} \pi(x^*)_j = \sum_{j \in \omega} \overline{x}_j.$$

Proof By definition, all permutations in G map variables in ω to other variables in ω, so the sums of the variables in a given orbit is invariant to permutations in G. This proves the first equality. The second equality then follows from (6). □

Theorem 1 *P_{OSR} is a relaxation of P.*

Proof Let x^* be an arbitrary feasible solution of P. We will show that there always exists a point z^* feasible for P_{OSR} and such that $o(z^*) \leq f(x^*)$, hence the claim. Given x^*, let us construct the two points $\overline{x}$ and z^* as

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

and

$$z^*_\omega = \sum_{j \in \omega} x^*_j.$$

For each constraint in C_1, $g_i(\overline{x}) \leq 0$ because $\overline{x}[V_2] = x^*[V_2]$ and g_i is convex in V_1. Similarly, for each constraint in C_2, we have $g_i(\overline{x}) = g_i(x^*) \leq 0$ because of Lemma 3. So, all constraint in $C_1 \cup C_2$ are satisfied by $\overline{x}$.

Now let us consider z^*. The integrality requirements on z are automatically satisfied, as x^* is a feasible solution of P, and thus sums of integer values within an orbit yield an integer result. In addition, for each constraint in $C_1 \cup C_2$, we have by definition and by Lemma 2

$$\overline{g}_i(z^*) = g_i(\overline{x}) \leq 0$$

since x^* itself is feasible for those constraints. Thus, z^* is feasible for P_{OSR}. As far as the objective function is concerned, we have $\overline{f}(z^*) = f(\overline{x}) \leq f(x^*)$ where the equality is by definition of $o(z)$ and Lemma 2, while the inequality is by convexity of $f(x)$ and because $\overline{x}$ is a convex combination of solutions with the same cost (because of symmetry). □

Corollary 1 *If P is a convex optimization problem, then P_{OSR} is an exact reformulation of P.*

Proof In the convex case, we have $J = C_2 = C_3 = V_2 = \emptyset$. Given an optimal solution z^* of P_{OSR}, we can construct a point x^* as

$$x^*_j = \frac{z^*_\omega}{|\omega|}$$

which, by convexity of constraints, is feasible for P and has the same objective value as z^*. The result easily follows. □

The result of Corollary 1 was already proved in [7].

Corollary 2 *If there exists an optimal solution x^* of P such that $|\omega|$ divides $\sum_{j\in\omega} x^*_j$ for all orbits ω associated to integer variables, and $C_3 = \emptyset$, then P_{OSR} is an exact reformulation of P.*

Proof In this case, the point x^* constructed as in the previous corollary is also integer, and satisfies all the constraints of P. It is then feasible for P and thus optimal. □

Note that the set C_3 of constraints is provided to give freedom in the choice of which variables to put into V_2. Intuitively, for a non-convex constraint we have a choice between ignoring it completely in the orbital shrinking relaxation and stabilizing its variables, which is clearly stronger but may reduce the shrinking possibilities considerably. In addition, we note that the partition (C_1, C_2, C_3) is by definition consistent with the constraint orbits of P and that all constraints in the same orbit will be mapped to the same constraint in P_{OSR}, so in practice P_{OSR} has one variable for each variable orbit and one constraint for each constraint orbit in P associated with a constraint in $C_1 \cup C_2$.

Finally, note that convexity is crucial for the above arguments. Indeed, given an arbitrary MINLP, a direct formal substitution according to (4) does not yield a relaxation in general, as shown in the following example.

Example 1 Let the feasible set of P be defined as

$$\{(x_1, x_2) \mid (x_1 - x_2)(x_2 - x_1) \leq -1\}$$

This set is not empty and the two variables are clearly symmetric. However, with the formal substitution $x_i \to z/2$ we obtain the set

$$\{z \mid 0 \leq -1\}$$

which is empty. □

4 Mining for the best subgroup: features and MILP results

Theorem 1 remains obviously valid if a subgroup G' of G is used (instead of G) in the construction of P_{OSR}. Different choices of G' lead to different relaxations and hence to different bounds. If G' is the trivial group induced by the identity permutation, no shrinking at all is performed and the relaxation coincides with the original problem. As G' grows in size, it generates longer orbits and the relaxation becomes more compact and easier to solve (also because more symmetry is encapsulated into the relaxation), but the lower bound quality decreases.

Ideally, we wish the relaxation to be (a) as tight as possible and (b) as efficient as possible with respect to the CPU time taken to solve it. In this section we discuss and computationally evaluate a few ideas for generating subgroups G' which should intuitively yield "good" relaxations in a MILP context. All experiments were conducted on a 1.4GHz Intel Core 2 Duo 64bit with 3GB of RAM. The MILP solver of choice is IBM ILOG CPLEX 12.2.

4.1 Automatic generation of the whole symmetry group

The formulation group is detected automatically using the techniques discussed in [17]: the MILP is transformed into a Directed Acyclic Graph (DAG) encoding the incidence of variables in objective and constraints, and a graph automorphism software (`nauty` [24]) is then called on the DAG. The orbital-shrinking relaxation is constructed automatically using a mixture of `bash` scripting, GAP [6], AMPL [5], and ROSE [18].

4.2 The instance set

We considered the following 39 symmetric MILP instances (in their minimization form):

```
ca36243 ca57245 ca77247 clique9 cod105 cod105r cod83 cod83r cod93 cod93r cov1053
cov1054 cov1075 cov1076 cov1174 cov954 flosn52 flosn60 flosn84 jgt18 jgt30
mered 04_35 oa25332 oa26332 oa36243 oa57245 oa77247 of5_14_7 of7_18_9 ofsub9
pa36243 pa57245 pa77247 sts135 sts27 sts45 sts63 sts81
```

all taken from F. Margot's website `http://wpweb2.tepper.cmu.edu/fmargot/index.html`.

4.3 Generator ranking

Using orbital shrinking with the whole symmetry group G has the merit of yielding the most compact relaxation. On our test set, however, this approach yields a relaxation bound which is not better than the LP bound 31 times out of 39, and for the remaining 8 times it is not better than the root-node CPLEX's bound (i.e., LP plus root node cuts)—although this will not necessarily be the case for other symmetric instances (e.g., for instances with small symmetry groups).

We observe that the final OSR relaxation only depends on the orbits of G' rather than on G' itself. If G' is trivial, then there are n orbits of size 1. If G' is the full symmetry group, then there is only 1 orbit of size n (since G' is transitive for these instances): so, in general, the smaller G' is, the more (and/or shorter) orbits it yields. We therefore consider the idea of testing subgroups with orbits of varying size, from small to large. Since testing all subgroups of G is impractical, we look at its generator list $\Pi = (\pi_0, \dots, \pi_k)$ (including the identity permutation). For any permutation π we let $\mathrm{fix}(\pi)$ be the subset of $[n]$ fixed by π, i.e., containing those i such that $\pi(i) = i$. We then reorder our list Π so that

$$|\,\mathrm{fix}(\pi_0)| \geq \cdots \geq |\,\mathrm{fix}(\pi_k)|$$

and for all $\ell \leq k$ we define G_ℓ as the subgroup of G induced by the sublist $(\pi_0, \cdots, \pi_\ell)$. This leads to a subgroup chain

$$G_0, G_1, \cdots, G_k = G$$

with increasing number of generators and hence larger and larger orbits (G_0 being the trivial group induced by the identity permutation). In our view, the first generators in the list are the most attractive ones in terms of bound quality—having a large $\mathrm{fix}(\pi)$ implies that the generated subgroup is likely to remain valid for (P) even when several variables are fixed by branching.

For each instance in our test set, we generated the relaxations corresponding to each G_ℓ and recorded bound values and CPU times, plotting the results against ℓ. We set a maximum user CPU time of 1800s, as we deemed a relaxation useless if it takes too long to solve. The typical behavior of the relaxation in terms of bound value and CPU time was observed to be mostly monotonically decreasing in function of the number ℓ of involved generators. Figure 1 shows an example of these results on the `sts81` instance.

4.4 Choosing a good set of generators

Our generator ranking provides a "dial" to trade bound quality versus CPU time. We now consider the question of how to set this dial automatically, i.e., how to choose a value of $\ell \in [k]$ leading to a good subgroup G_ℓ.

Out of the 39 instances in our test set, 16 yields the same bound independently of ℓ, and were hence discarded from this experiment. The remaining 23 instances:

	sts81	
ℓ/k	obj	CPU
1/14	45	3.60
2/14	45	1.51
3/14	45	1.18
4/14	45	1.13
5/14	33	0.01
6/14	33	0.01
7/14	33	0.02
8/14	33	0.00
9/14	29	0.02
10/14	29	0.00
11/14	29	0.01
12/14	28	0.01
13/14	28	0.00
14/14	27	0.00

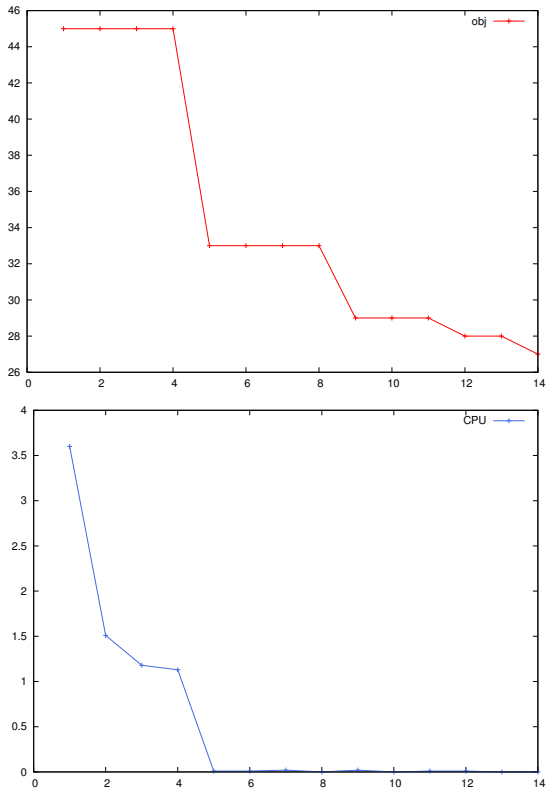

Fig. 1 Bound values and CPU times against the number ℓ of generators for instance `sts81`.

`ca36243 clique9 cod105 cod105r cod83 cod83r cod93 cod93r cov1075 cov1076 cov954 mered 04_35 oa36243 oa77247 of5_14_7 of7_18_9 pa36243 sts135 sts27 sts45 sts63 sts81`

yield a nonzero decrease in bound value as ℓ increases, so they are of interest for our test.

Having generated and solved relaxations for all $\ell \leq k$, we hand-picked good values of ℓ for each instance, based on these prioritized criteria:

1. bound provided by G_ℓ strictly tighter than LP bound;
2. minimize CPU time, with strong penalty for choices of ℓ leading to excess of 10 seconds;
3. on lack of other priorities, choose ℓ leading to bounds around midrange in $[\mathrm{bnd}(G_k), \mathrm{bnd}(G_1)]$, where $\mathrm{bnd}(G')$ denotes the bound value obtained by solving the OSR based on the subgroup G'.

This choice led to the first three columns of Table 1 (the fourth will be explained later).

Instance	G_ℓ	LP	CPU	$\frac{\text{inc}(G_\ell)}{\text{inc}(G)}$
ca36243	49	48	0.07	0.50
clique9	∞	36	0.06	0.87
cod105	-16	-18	4.91	0.99
cod105r	-13	-15	0.25	0.99
cod83	-26	-28	0.12	0.98
cod83r	-22	-25	4.44	0.88
cod93	-48	-51	3.07	0.98
cod93r	-46	-47	2.74	0.97
cov1075	19	18	3.03	0.86
cov1076	44	43	185.83	0.73
cov954	28	26	0.45	0.79
mered	∞	140	0.12	0.92
04_35	∞	70	0.07	0.75
oa36243	∞	48	0.75	0.50
oa77247	∞	112	0.00	0.98
of5_14_7	∞	35	0.13	0.62
of7_18_9	∞	63	0.04	0.91
pa36243	-44	-48	1.26	0.50
sts135	60	45	0.05	0.88
sts27	12	9	0.01	0.88
sts45	24	15	0.39	0.66
sts63	27	21	0.00	1.00
sts81	33	27	0.00	0.88

Table 1 Hand-picked choice of the subgroup G_ℓ.

Next we looked for a feature of the solution data over all $\ell \leq k$ and over all instances, whose average value corresponds to values of ℓ that are close to the hand-picked ones in Table 1. Again, intuition led our choice for this feature. Our reasoning is as follows. We observe that, given any orbit ω, the OSR relaxation replaces $\sum_{j\in\omega} x_j$ with a single variable z_ω. Suppose now that a constraint $\sum_{j\in\omega} x_j \leq b_i$ happens to exist in the MILP formulation (P): this is simply reformulated to a bound constraint $z_\omega \leq b_i$, thus replacing a $|\omega|$-ary original relation on the decision variables x with a unary relation on the decision variables z. Intuitively, this will over-simplify the problem and will likely yield a poor relaxation. Instead, we would like to deal with orbits that are somehow "orthogonal" to the problem constraints.

To this aim, consider the i-th (out of, say, r) MILP constraint, namely $\sum_{j\in[n]} a_{ij}x_j \leq b_i$, and define the *incidence* of an orbit ω with respect to the support of this constraint as $|\omega \cap \{j \in [n] : a_{ij} \neq 0\}|$. Intuitively, the lower the incidence of an orbit, the farther we are from the situation where problem constraints become over-simplified range constraints in the relaxation. Lower incidence orbits should yield tighter relaxations, albeit perhaps harder to solve. Given a subgroup G' with orbits $\Omega' = \{\omega'_1, \cdots, \omega'_{m'}\}$, we then extend the incidence notion to G'

$$\text{inc}(G', i) := \left| \bigcup_{\omega'\in\Omega'} \omega' \cap \{j \in [n] : a_{ij} \neq 0\} \right|, \tag{8}$$

and finally to the whole MILP formulation

$$\operatorname{inc}(G') = \sum_{i \in [r]} \operatorname{inc}(G', i). \tag{9}$$

The rightmost column of Table 1 reports the relative incidence of G_ℓ, computed as $\operatorname{inc}(G_\ell)/\operatorname{inc}(G)$, for those ℓ that were hand-chosen to be "best" according to the prioritized criteria listed above. Its average is 0.82 with standard deviation 0.17. This value allows us to generate a relaxation which is hopefully "good", by automatically selecting the value of ℓ such that $\operatorname{inc}(G_\ell)/\operatorname{inc}(G)$ is as close to 0.82 as possible.

4.5 Bound strength

The quality of the OSR relaxation we obtain with the method of Section 4.4 is reported in Table 2 whose columns include: the instance name, the automatically determined value of ℓ and the total number k of generators, the best-known optimal objective function value for the instance (starred values correspond to guaranteed optima), the bound given by G_1 which provides the tightest non-trivial OSR bound, the bound given by G_ℓ and the associated CPU time, the CPU time "cpx_t" spent by CPLEX 12.2 (default settings) on the original formulation to get the same bound as OSR (only reported when the OSR bound is strictly better than the LP bound), and the LP bound. Entry *limit* marks an exceeded time limit of 1800 sec.s, while boldface highlights the best results for each instance.

The results show that the "fix" and "inc" features we chose are meaningful: our bound is often stronger than the LP bound, whilst often taking only a fraction of a second to solve. The effect of orbital shrinking can be noticed by looking at the "cpx_t" column, where it is evident that the original formulation takes significantly longer to reach the bound given by our relaxation.

5 A general Orbital Shrinking based decomposition method

Let P be a MINLP as in the previous sections and let G be the chosen symmetry group for P. Using G, we can construct the orbital shrinking reformulation P_{OSR} of P, which will act as the master problem in our decomposition scheme, much like in a traditional Benders decomposition scheme. Indeed, the scheme is akin to a logic-based Benders decomposition [13], although the decomposition is not based on a traditional variable splitting, but on aggregation, and the OSR master works with the aggregated variables z. A similar approach, although problem specific, was also used in [19].

Instance	ℓ/k	best	G_1	G_ℓ	CPU	cpx_t	LP
ca36243	3/6	49*	49	48	0.02		48
clique9	5/15	∞^*	∞	∞	**0.06**	0.17	36
cod105	3/11	-12*	*limit*	**-14.09**†	*limit*		-18
cod105r	3/10	-11*	-11	**-11**	**24.12**	28.36	-15
cod83	3/9	-20*	-21	**-24**	16.78	**9.54**	-28
cod83r	3/7	-19*	-21	**-22**	**4.44**	7.85	-25
cod93	3/10	-40		**-46.11**†	*limit*		-51
cod93r	3/8	-38	-39	**-44**	**271.74**	446.48	-47
cov1075	3/9	20*	20	**19**	**3.03**	79.79	18
cov1076	3/9	45	44	43	2.78		43
cov954	3/8	30*	28	26	0.11		26
mered	21/31	∞^*	∞	∞	**0.15**	3.37	140
O4_35	3/9	∞^*	∞	70	0.00		70
oa36243	3/6	∞^*	∞	48	0.01		48
oa77247	3/7	∞^*	∞	∞	**0.10**	265.92	112
of5_14_7	7/9	∞^*	∞	35	0.00		35
of7_18_9	7/16	∞^*	∞	∞	**0.09**	0.15	63
pa36243	3/6	-44*	-44	-48	0.01		-48
sts135	3/8	106	75	**60**	**0.11**	109.81	45
sts27	4/8	18*	14	**12**	0.01	1.05	9
sts45	2/5	30*	24	15	0.00		15
sts63	4/9	45*	36	**27**	**0.02**	1.99	21
sts81	5/14	61	45	**33**	**0.01**	3.92	27

Table 2 OSR performance (in the G_ℓ/CPU columns). Entries marked * denote guaranteed optimal values; those marked † denote the best lower bound at the time limit. In sts27, CPLEX closes the gap at the root node. Values for cpx_t are only present when the OSR bound is integer and better than the LP bound.

For each feasible solution z^* of P_{OSR}, we can then define the following (slave) feasibility check problem $R(z^*)$

$$g_i(x) \leq 0 \quad \forall i \in C \tag{10}$$

$$\sum_{j \in \omega} x_j = z^*_\omega \quad \forall \omega \in \Omega \tag{11}$$

$$x_j \in \mathbb{Z} \quad \forall j \in J \tag{12}$$

If $R(z^*)$ is feasible, then the aggregated solution z^* can be *disaggregated* into a feasible solution x^* of P, with the same cost. Otherwise, z^* must be removed from P_{OSR}, in either of the following two ways:

1. Generate a *nogood* cut that forbids the assignment z^* to the z variables. As in logic-based Benders decomposition, an ad-hoc study of the problem is needed to derive (strong) nogood cuts.
2. Branching. As z^* is integer, branching on non-fractional z variables is needed, and z^* 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 search tree has finite depth, assuming that z variables are bounded. Note that in this case the method may repeatedly

check for feasibility the same aggregated solution z^*: in practice, this can easily be 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 $R(z^*)$ has the same symmetry group of P, so symmetry may still be an issue while solving $R(z^*)$. This issue is usually solvable because (i) linking constraints (11) may make the model much easier to solve, and (ii) the easier structure of $R(z^*)$ may allow for more effective symmetry breaking techniques. Note also that $R(z^*)$ is a pure feasibility problem, so a CP solver may be a better choice than a MINLP solver.

The above decomposition strategy is well suited for pure integer problems, but is not very convenient when continuous variables are present in the model because in the mixed-integer case one should enumerate, in the master, all possible values also for the continuous variables, which makes the method impractical. However, the method can be modified to deal with continuous variables more effectively. In particular, we can:

- zero out the objective coefficients of the z_ω variables in P_{OSR};
- reintroduce the objective coefficients of the continuous variables in $R(z^*)$;
- remove the linking constraints (11) associated to orbits of continuous variables.

The advantage of the above modification is that only the partial assignments over the aggregated integer variables need to be checked, at the expense of turning the feasibility check into an optimization problem itself. Such extended method has been used in [26] to solve a very challenging instance of 3-dimensional quadratic assignment problem. Interestingly, the role of MIP and CP was swapped in [26]: a CP solver was used to enumerate all feasible solutions of the master problem, while a MIP solver was used to solve the optimization slaves.

6 Application to Shift Scheduling

A shift scheduling problem assigns a feasible working shift to a set of employees, in order to satisfy the demands for a given set of activities at each period in a given time horizon. The set of feasible shifts that can be assigned to employees is often defined by a complex set of work regulation agreements and other rules. Assigning a shift to an employee means specify an activity for each period, which may be a working activity or a rest activity (e.g., lunch). The objective is usually to minimize the cost of the schedule, which is usually a linear combination of working costs plus some penalties for undercovering/overcovering the demands of the activities in each time period. If the set of working activities W is made by a single activity, we talk of *single-activity* shift scheduling, while if there are several working activities we talk of *multi-activity* shift scheduling. In what follows we consider the latter case, with the additional constraint that all employees are identical.

In particular, suppose we are given a planning horizon divided into a set of periods T, a set of activities A, a subset $W \subset A$ of working activities, and a set of employees E. For each period $t \in T$ and for each working activity $a \in W$, we are given a demand d_{at}, an assignment cost c_{at}, an undercovering cost c^-_{at} and an overcovering cost c^+_{at}. Introducing the set of integer variables y_{at}, which count the number of employees assigned to activity a at period t, and integer variables s^-_{at}, s^+_{at} that count the appropriate under/over covering, we can formulate the problem as:

$$\min \sum_a \sum_t c_{at} y_{at} + \sum_a \sum_t c^+_{at} s^+_{at} + \sum_a \sum_t c^-_{at} s^-_{at} \tag{13}$$

$$y_{at} - s^+_{at} + s^-_{at} = d_{at} \quad \forall a \in W, \forall t \in T \tag{14}$$

$$\sum_e x_{eat} = y_{at} \quad \forall a \in W, \forall t \in T \tag{15}$$

$$\langle x \text{ defines a feasible shift } \forall e \in E \rangle \tag{16}$$

$$y_{at}, s^+_{at}, s^-_{at} \in \mathbb{Z}^+ \tag{17}$$

$$x_{eat} \in \{0, 1\} \tag{18}$$

where x_{eat} are binary variables, each of which is equal to 1 if employee e is assigned to activity a in period t. Depending on how we formulate constraints (16), we may end up with very different models. A convenient way to define the set of feasible shifts that can be assigned to a given employee is to use a regular or a context-free language, i.e., the set of feasible shifts can be viewed as the words accepted by a finite automaton or, more generally, by a push-down automaton. It has been shown in [29,1] that it is possible to derive a polyhedron that describes a given regular/context-free language. Such representations are compact (in an appropriate extended space, i.e., introducing additional variables) and thus lead directly to a MIP formulation of the problem. In particular, the extended formulation for a regular language is essentially a network flow formulation based on the expanded graph associated with the accepting automaton (see [28,1] for details). The extended formulation for the context-free language, on the other hand, is based on an and-or graph built by the standard CYK parser [14] for the corresponding grammar [29,33].

Note that it is not necessary to describe completely the set of feasible shifts by a regular/context-free language. The formal language may capture only some of the constraints defining a feasible shift, with the remaining ones described as linear inequalities. This may simplify the corresponding automaton considerably (for example, regular languages are notoriously bad at handling counting arguments). However, describing the set of feasible shifts with formal languages alone has some important implications. First of all, it has been proven for both the regular and context-free languages that the derived polyhedron is integral [29], and thus, if the are no other constraints, it is possible to optimize a linear function over the set of feasible shifts by solving just a

linear program. Even more importantly, these results have been extended also to polyhedra describing sets of feasible shifts [2]. It is then possible to consider an aggregated (*implicit*) model and reconstruct an optimal solution of the original one with a polynomial post-processing phase. From the OSR point of view, this means that P_{OSR} is in this case an exact reformulation. Consider for example the regular polytope in its extended form: the optimal solution is always a flow of integral value, say k, and basic network flow theory guarantees that it can be decomposed into k paths of unitary flow (and since each path in the expanded graph corresponds to a word in the language, this is a feasible solution for the original explicit problem). Similar reasoning applies to the grammar polytope (although it is not a flow model), as successfully shown in [2]. It is interesting to note that this gives the current state-of-the-art for solving multi-activity shift scheduling problems.

Unfortunately, it is not always reasonable to describe the set of feasible shifts completely with a formal language. While it is true that formal languages can be extended without changing the complexity of the corresponding MIP encoding (this is particularly true for context-free languages [33]), still some cardinality constraints may be very awkward to express, see [37] for examples. As such, we assume in the following that the formal language captures the constraints that define the set of feasible shifts only partially, and thus we need to apply the decomposition framework of Section 5 in order to turn OSR into an exact procedure.

6.1 MIP model

The MIP model that we use is a simple modification of the general model (13)-(18). The main difference is that we partition the set of feasible shifts Ω into k subsets Ω_k, each of which is described by a potentially different deterministic finite automaton (DFA) and by cardinality constraints. This partition can simplify a lot the structure of the DFAs, and in general makes the implicit model more accurate, since the cardinality constraints are aggregated only within employees of the same "kind". This of course increases the size of the relaxation, but since the aggregated model is quite compact, this is usually well worth it. For each shift type Ω_k, the MIP model decides how many employees are assigned a shift in Ω_k, and then computes an aggregated integer flow of the same value. In details:

$$\min \sum_k \sum_a \sum_t c_{at} y^k_{at} + \sum_a \sum_t c^+_{at} s^+_{at} + \sum_a \sum_t c^-_{at} s^-_{at} \tag{19}$$

$$\sum_k y^k_{at} - s^+_{at} + s^-_{at} = d_{at} \quad \forall a \in W, \forall t \in T \tag{20}$$

$$\text{regular}(y^k, w^k, \text{DFA}^k) \quad \forall k \in K \tag{21}$$

$$\langle \text{cardinality constraints for } y^k \rangle \quad \forall k \in K \tag{22}$$

$$\sum_k w^k \leq E \tag{23}$$

$$w^k, y^k_{at}, s^+_{at}, s^-_{at} \in \mathbb{Z}^+ \tag{24}$$

Note that we use the notation of constraint (21) to refer to the extended MIP formulation of the regular constraint involving flow variables. The constraint ensures that variables y^k can be decomposed into w^k words accepted by the automaton DFA^k. Constraints (22) refers to the cardinality constraints expressed as linear constraints that complete the description of sets Ω_k. Finally, if an upper bound E is given on the number of employees that can scheduled, it can be imposed in constraint (23).

6.2 CP checker

The decision to partition the set of feasible shifts into k subsets Ω_k has an important consequence on the structure of the CP checker: the model actually decomposes into k separate CP models, one for each type of shift. Given an index k, suppose the master (MIP) model assigns $\overline{w}^k$ employees, with their aggregated shifts described by $\overline{y}_{at}$; then the corresponding CP model can easily be formulated by using several *global constraints* [35]. Global constraints are used within a CP solver to represent general purpose and common substructures, for which efficient and effective constraint propagators are known. While a global constraint is typically semantically redundant, in the sense that the same effect can be obtained as the conjunction of several simpler constraints, it is very convenient as a shorthand for expressing frequently recurring patterns. Less obviously, global constraints also make the underlying CP solver more efficient, as propagators can better exploit the substructures of the problem at hand. In our case, the corresponding CP model, which is similar to the one proposed in [3], reads:

$$\texttt{gcc}(x^e, \sigma^e, A) \quad \forall e \in 1, \ldots, \overline{w}^k \tag{25}$$

$$\tau^e = \sum_{a \in W} \sigma_a^e \quad \forall e \in 1, \ldots, \overline{w}^k \tag{26}$$

$$\langle \text{cardinality constraints for } \sigma^e, \tau^e \rangle \quad \forall e \in 1, \ldots, \overline{w}^k \tag{27}$$

$$\texttt{regular}(x^e, \text{DFA}^k) \quad \forall e \in 1, \ldots, \overline{w}^k \tag{28}$$

$$\texttt{gcc}(x_t, \overline{y}_t, A) \quad \forall t \in T \tag{29}$$

$$x^e \preceq x^{e+1} \quad \forall e \in 1, \ldots, \overline{w}^k - 1 \tag{30}$$

Variables x_t^e denote the activity assigned to employee e at time t. Variables σ_a^e count the number of periods assigned to each activity for employee e, while τ^e gives the sum over all working activities. Both are needed to specify the cardinality constraints (27). Variables σ_a^e are linked to variables x_t^e through global cardinality constraints (25). We recall that a global cardinality constraint $\texttt{gcc}(x, y, S)$ is satisfied if y (which can be either variable or constant) counts the number of occurrences of values in S in the set of values assigned to variables x [34]. Global constraints based on regular languages are used (28) to complete the description of the sets of feasible shifts. We recall that a regular constraint $\texttt{regular}(x, DFA)$ is satisfied if the values assigned to the variables x, in the given order, constitute a word accepted by the automaton DFA [28]. In (29) global cardinality constraints are used again to link the variables in the CP model to the master solution $\overline{y}_{at}$. Finally, we impose a lexicographic order among the shifts of the employees with constraints (30).

The CP model above is usually extremely fast in proving whether the aggregated solution can be turned into a solution of the original model. However, as the number of activities and employees increases, it can occasionally become very time consuming. The main reason for this behavior is the weak interaction between the cardinality constraints (29) and the symmetry breaking constraints (30). To overcome this issue, we implemented an ad-hoc propagator that implements a custom symmetry breaking strategy based on the cardinality constraints, see [37] for details.

Another issue with the CP model above is that the minimum/maximum length of a working shift (i.e., the number of periods, breaks included, between the first and last working period) is constrained only implicitly by the regular constraints. Again, we implemented a custom propagator that deals with that. The combined effect of these propagators is very significant: we often observed reductions of $2-3$ orders of magnitude in both the number of nodes and the running times on hard instances. Sometimes, we observed even higher savings. For example, on one instance with 9 full-time employees and 3 activities, we reduced the running times from 6 minutes to 10^{-5} seconds, with a number of nodes dropping from 765,026 to 1.

6.3 Computational Results

We tested our method on the multi-activity instances used in [1,2,32]. This testbed is derived from a real-world store, and contains instances with 1 to 10 working activities (each class has 10 instances). A basic description of the problem is as follows:

- The planning horizon of 1 day is divided into 96 slots of 15 minutes.
- A part-time employee must work a minimum of 3 hours and less than 6 hours, and is entitled to one break of 15 minutes.
- A full-time employee can work between 6 and 8 hours, and is entitled to have two breaks of 15 minutes plus a lunch break of 1 hour (in any order).
- When an employee starts working on one activity, it must do it for at least 1 hour. In addition, a break/lunch is needed before changing activity.
- A break cannot be scheduled at the beginning/end of the shift.
- At specific times of the day (e.g., when the store is closed), no employee is allowed to work.
- Overcovering/undercovering is allowed, with an associated cost.
- The cost of a shift is the sum of the costs of all working activities performed in the shift.

We implemented our method in C++, using IBM ILOG CPLEX 12.2 [15] as black box MIP solver, and Gecode 3.7.3 [8] 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 core was used by each process). Every method was given a time limit of 1 hour per instance. Concerning the set of feasible shifts Ω, we simply partitioned it into full-time and part-time shifts. We could have partitioned the full-time shifts further (depending on the relative order of breaks and lunch), but it seemed overkill because all full-time shifts share the same cardinality constraints (this was confirmed by some preliminary tests). In general, disaggregating shifts depending on the cardinality constraints seems to work well in practice.

From the implementation point of view, our hybrid method is made of the following phases:

- First, the aggregated model is solved with CPLEX, using the default settings. The outcome of this (usually fast) first phase is a dual bound potentially stronger than the LP bound, and the set of aggregated solutions collected by the MIP solver during the solution process (not necessarily feasible for the original model).
- We apply an ad-hoc MIP repair/improve heuristic (see [37] for details) to each aggregated solution which is within 20% of the aggregated model optimal solution. The outcome of this phase is always a feasible solution for the original model, thus a primal bound. Note that if the gap between the two is already below the 1% threshold, we are done.
- We solve the aggregated model again, this time implementing the hybrid MIP/CP approach. This means that we disable dual reductions (other-

wise the decomposition would not be correct) and use CPLEX callbacks framework to implement the decomposition.

Here is a more detailed description of the last phase. Whenever the MIP solver finds an integer solution, either with its own heuristics or because the LP relaxation happens to be integer, we build the corresponding CP models and solve them with Gecode DFS algorithm. As far as the branching strategy of the CP solver is concerned, after some trial-and-error we found that ranking the variables by increasing time period was the most successful policy. If the check is successful, then we update the incumbent, otherwise the solution is rejected. In both cases, we apply the MIP repair/improve heuristic on it to try to find a new incumbent. If the solution was the optimal solution of an LP relaxation, then we force a branching on a integer variable and keep going. As to branching inside the MIP solver, we let CPLEX apply its own powerful strategies whenever the relaxation has some fractional variables. If this is not the case, we branch first on the w variables and then, if all w variables are already fixed, on the y, again ranking them by increasing time period. The rationale behind this strategy is that if the w variables are not fixed to some value, then we cannot even formulate the CP checking model, so the sooner we fix them the better. Note that as soon as the w variables are fixed, we can build a CP model akin to (25)-(30) where the y variables are not necessarily fixed but just take the domains of the current node. In this case, we let the CP solver run with a strict fail limit (1000 in our code) and, if it detects infeasibility, then we prune the node.

Table 3 reports a comparison between the proposed method and others in the literature, for a number of activities from 1 to 10. As far as the number of employees is concerned, we put an upper bound of 12 for instances with up to 2 activities, of 24 for instances with 3 to 8 activities and of 30 for instances with 9 or 10 activities. `cpx-reg` refers to the explicit model based on the regular constraint in [1], while `grammar` refers to the implicit model based on the grammar constraint in [2]. Note that for `grammar` we are reporting the results from [2], which were obtained on a different machine and, more importantly, with an older version of CPLEX, so the numbers are meant to give just a reference. All methods were run to solve the instances to near-optimality, stopping when the final integrality gap dropped below 1%.

According to Table 3, `hybrid` outperforms significantly the explicit model `cpx-reg`, which scales very poorly because of symmetry issues and slow LPs. When compared to `grammar`, `hybrid` is very competitive only for up to 2 activities, while after that threshold `grammar` clearly takes the lead. This is somehow expected: the set of feasible shifts in these instances can indeed be described without too much effort with an extended grammar, and it is no surprise that the pure implicit MIP model outperforms our decomposition approach. However, `hybrid` is likely to be the best approach if the extended grammar is not a viable option.

Table 4 reports a closer comparison between `cpx-reg` and `hybrid`, reporting the average final gap, average number of variables in the model and average

Table 3 Average computing times between the different methods to solve to near-optimality (gap $\leq 1\%$) the instances with up to 10 activities.

	# solved (10)			time(s)		
# act.	`cpx-reg`	`hybrid`	`grammar`	`cpx-reg`	`hybrid`	`grammar`
1	10	10	10	41.3	9.1	283.7
2	9	10	9	707.9	194.5	379.9
3	4	5	9	2957.3	1996.4	205.4
4	3	6	10	2970.2	1827.9	300.5
5	0	8	10	3600.0	1438.4	146.2
6	1	4	10	3530.6	2340.6	213.8
7	1	6	10	3438.7	2399.0	230.9
8	0	5	10	3600.0	2201.5	257.1
9	0	4	10	3600.0	2444.0	289.1
10	0	2	10	3600.0	3275.6	516.7

node throughput for each category. According to the table, `hybrid` consistently yields very small gaps (always below 3% on average), while for `cpx-reg` is always above 60% with more than 4 activities. As far as the number of variables of the models is concerned, `hybrid` needs approximately 1/10 of the number of variables of `cpx-reg`, which promptly turns into a much faster node throughput: `hybrid` is more than two order of magnitude faster in exploring nodes than `cpx-reg`. Note that, according to [2], `grammar` models range from 70,000 variables for instances with 1 activity to 96,000 for instances with 10 activities, so the hybrid model based on regular languages is significantly smaller.

Table 4 Comparison of average final gap between `cpx-reg and hybrid`.

	gap(%)		#vars		node/sec	
# act.	`cpx-reg`	`hybrid`	`cpx-reg`	`hybrid`	`cpx-reg`	`hybrid`
1	0.72	0.24	9,956	1,908	21.99	10.36
2	0.78	0.61	13,608	2,925	3.52	20.60
3	3.74	3.00	34,903	4,152	0.59	8.42
4	25.18	1.39	43,005	5,291	0.20	3.92
5	62.55	1.01	52,979	6,828	0.05	3.32
6	75.89	1.59	62,442	8,364	0.03	1.86
7	90.00	0.90	73,693	9,936	0.01	1.64
8	100.00	1.92	78,809	10,603	0.01	1.22
9	100.00	1.52	104,561	11,509	0.01	1.05
10	100.00	2.76	120,049	13,302	0.01	0.86

Finally, Table 5 shows the gap just before the beginning of the last phase (but after the aggregated model has been solved and its solutions have been used to feed the MIP repair/improve heuristic). On almost all categories the average final gap is below 10%, with an average running time of 1 minute. This heuristic alone significantly outperforms `cpx-reg` for a number of activities greater than 3. It is also clear from the table that solving the OSR relaxations

with a black box MIP solver is usually very fast. Interestingly, solving these MIPs turn out to be often faster than solving the LP relaxations of the original models, while providing better or equal dual bounds. For example, on one instance with 1 activity, the LP relaxation of the original model takes 0.26 seconds to solve, yielding a dual bound of 142.48, while the OSR MIP takes 0.12 seconds and yields a dual bound of 182.54 (in this case, equal to the value of the optimal solution). On another instance with 10 activities, the LP relaxation takes 269.55 seconds, while the OSR MIP takes only 52.77 seconds, both yielding the same dual bound in this case.

Table 5 MIP repair/improve heuristics standalone results.

# act.	time(s)	gap(%)
1	6.2	1.5
2	46.5	6.5
3	24.7	20.3
4	30.3	7.1
5	34.5	5.9
6	33.5	10.5
7	63.2	7.1
8	69.3	7.7
9	89.8	6.7
10	65.9	8.0

7 Application to the Multiple Knapsack Problem

In the present section, we specialize the general framework of the Section 5 to the multiple knapsack problem (MKP) [38,30]. This a natural generalization of the traditional knapsack problem [23], 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

$$\max \sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij} \tag{31}$$

$$\sum_{j=1}^{n} w_j x_{ij} \leq C_i \quad \forall i = 1, \ldots, m \tag{32}$$

$$\sum_{i=1}^{m} x_{ij} \leq 1 \quad \forall j = 1, \ldots, n \tag{33}$$

$$x \in \{0,1\}^{m \times n} \tag{34}$$

where binary variable x_{ij} is set to 1 if and only if item j is loaded into knapsack i. We 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 P_{OSR} reads

$$\max \sum_{k=1}^{K} p_k y_k \tag{35}$$

$$\sum_{k=1}^{K} w_k y_k \leq mC \tag{36}$$

$$0 \leq y_k \leq |V_k| \quad \forall k = 1, \ldots, K \tag{37}$$

$$y \in \mathbb{Z}_+^K \tag{38}$$

Intuitively, in P_{OSR} 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 [39] and exploiting the CDBF [10] 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 $\{0, \ldots, C\}$. With this choice of variables, the model reads:

$$\text{binpacking}(b, l, s) \tag{39}$$

$$b_{j-1} \leq b_j \quad \text{if } s_{j-1} = s_j \tag{40}$$

where (40) 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 [11]. 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

$$\sum_{p \in P} a_{pk} x_p = y_k^* \tag{41}$$

$$\sum_{p \in P} x_p = m \tag{42}$$

$$x_p \in \mathbb{Z}_+ \tag{43}$$

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.

7.1 Computational Experiments

We implemented our codes in C++, using IBM ILOG CPLEX 12.4 [15] as black box MIP solver and Gecode 3.7.3 [8] 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 [31]. According to [31], difficult instances can be obtained introducing some correlation between profits and weights. Among the hardest instances presented in [31] 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 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 $C = (\sum_{i=1}^{n} w_i)/8$.

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 authors 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 (31)−(34), denoted as `cpxorig`. The other is obtained by aggregating the binary variables corresponding to identical items. The model, denoted as `cpx`, reads

$$\max \sum_{i=1}^{m} \sum_{k=1}^{K} p_j z_{ik} \tag{44}$$

$$\sum_{k=1}^{K} w_j z_{ik} \leq C \quad \forall i = 1, \ldots, m \tag{45}$$

$$\sum_{i=1}^{m} z_{ik} \leq U_k \quad \forall k = 1, \ldots, K \tag{46}$$

$$z \in \mathbb{Z}_{+}^{m \times K} \tag{47}$$

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 6. 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.

Table 6 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

Then, we compared three variants of the hybrid MIP/CP procedure described in Section 7, that differs on the models used for the feasibility check. The first variant, denoted by `BPstd`, is based on the compact model (39)−(40). The second and the third variants are both based on the extended model

Table 7 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

(41)−(43), but differs on the solver used: a CP solver for `BPcgCP` and a MIP solver for `BPcgMIP`. All variants use model (35)−(38) as a master problem, which is fed to CPLEX and solved with dual reductions disabled, to ensure correctness of the method. CPLEX callbacks are used to implement the decomposition. A comparison of the three methods is given in Table 7. 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 (40) 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 (41)−(43), 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 8 shows the average gap closed by the OSR 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.

Table 8 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

8 Conclusions

We discussed a new methodology for deriving a relaxation of symmetric discrete optimization problems, based on variable aggregation within orbits. The idea is that we view symmetry as a positive feature that we want to exploit—as opposed to breaking, as commonly done in discrete optimization.

The approach, called orbital shrinking, sometimes leads to an exact and symmetry-free reformulation of a given problem. This happens, for instance, in case a standard ILP model for the asymmetric Traveling Salesman Problem (TSP) is solved and the input arc costs happen to be symmetric. In this setting, the symmetry group has an orbit $\{(i,j), (j,i)\}$ for each node pair $\{i,j\}$, and orbital shrinking automatically produces the symmetric TSP formulation of the problem—which is of course a much better way to model it when costs are symmetric. In this context, orbital shrinking can be seen as an automatic preprocessing step to produce a more effective model for the actual input data, capable of fixing some kind of modeling errors.

In other cases, orbital shrinking produces just a relaxation of the original problem, so it needs to be embedded in a more general solution scheme. We have described a master-slave framework akin to Benders' decomposition, where orbital shrinking acts as the master problem and generates a sequence of aggregated solutions to be checked for feasibility by a suitable slave subproblem—possibly based on Constraint Programming. Although the framework itself is not entirely new, a novelty of our approach is that it is driven by the automatically-detected symmetry group of the formulation at hand. Computational results on two specific applications prove the effectiveness of the scheme.

Future work should be devoted to the study of sufficient conditions under which orbital shrinking produces an exact reformulation. Practical applications of orbital shrinking decomposition to other symmetric problems are also worth investigating.

Acknowledgements The first and third authors were supported by the "PRIN2012 Project" of MiUR, Italy. The second author was partially supported by grants Digiteo Chair 2009-14D "RMNCCO" and Digiteo 2009-55D "ARM". We thank Mauro Diligenti who asked the question that originated the present work —Why do you discrete optimization guys hate symmetry and want to destroy it?

References

1. M.-C. Côté, B. Gendron, C.-G. Quimper, and L.-M. Rousseau. Formal languages for integer programming modeling of shift scheduling problems. *Constraints*, 16(1):54–76, 2011.
2. M.-C. Côté, B. Gendron, and L.-M. Rousseau. Grammar-based integer programming models for multiactivity shift scheduling. *Management Science*, 57(1):151–163, 2011.
3. S. Demassey, G. Pesant, and L.-M. Rousseau. A cost-regular based hybrid column generation approach. *Constraints*, 11(4):315–333, 2006.
4. M. Fischetti and L. Liberti. Orbital shrinking. In *Proceedings of ISCO*, 2012.
5. R. Fourer and D. Gay. *The AMPL Book*. Duxbury Press, Pacific Grove, 2002.
6. The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.4.10*, 2007.
7. K. Gatermann and P. Parrilo. Symmetry groups, semidefinite programs and sums of squares. *Journal of Pure and Applied Algebra*, 192:95–128, 2004.
8. Gecode Team. Gecode: Generic constraint development environment, 2012. Available at `http://www.gecode.org`.
9. I. P. Gent, K. E. Petrie, and J.-F. Puget. Symmetry in constraint programming. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, pages 329–376. Elsevier, 2006.
10. I. P. Gent and T. Walsh. From approximate to optimal solutions: Constructing pruning and propagation rules. In *IJCAI*, pages 1396–1401. Morgan Kaufmann, 1997.
11. P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9:849–859, 1961.
12. J. N. Hooker. *Integrated Methods for Optimization*. Springer, 2006.
13. J. N. Hooker and G. Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 96(1):33–60, 2003.
14. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison Wesley, 1979.
15. IBM ILOG. *CPLEX 12.4 User's Manual*, 2012.

16. H. Katebi, K. A. Sakallah, and I. L. Markov. 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, pages 113–127, 2010.
17. L. Liberti. Reformulations in mathematical programming: Automatic symmetry detection and exploitation. *Mathematical Programming A*, 131:273–304, 2012.
18. L. Liberti, S. Cafieri, and D. Savourey. Reformulation optimization software engine. In K. Fukuda, J. van der Hoeven, M. Joswig, and N. Takayama, editors, *Mathematical Software*, volume 6327 of *LNCS*, pages 303–314, New York, 2010. Springer.
19. J. Linderoth, F. Margot, and G. Thain. Improving bounds on the football pool problem by integer programming and high-throughput computing. *INFORMS Journal on Computing*, 21(3):445–457, 2009.
20. F. Margot. Pruning by isomorphism in branch-and-cut. *Mathematical Programming*, 94:71–90, 2002.
21. F. Margot. Exploiting orbits in symmetric ILP. *Mathematical Programming B*, 98:3–21, 2003.
22. F. Margot. Symmetry in integer linear programming. In M. Jünger, T. Liebling, D. Naddef, G. Nemhauser, W. Pulleyblank, G. Reinelt, G. Rinaldi, and L. Wolsey, editors, *50 Years of Integer Programming 1958-2008*, pages 647–686. Springer Berlin Heidelberg, 2010.
23. S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. Wiley, 1990.
24. B. D. McKay. Practical graph isomorphism, 1981.
25. M. Milano. *Constraint and Integer Programming: Toward a Unified Methodology*. Kluwer Academic Publishers, 2003.
26. H. D. Mittelmann and D. Salvagnin. On solving a hard quadratic 3-dimensional assignment problem. Technical report, DEI, University of Padova, 2013.
27. J. Ostrowski, J. Linderoth, F. Rossi, and S. Smriglio. Orbital branching. *Mathematical Programming*, 126:147–178, 2011.
28. G. Pesant. A regular language membership constraint for finite sequences of variables. In *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings*, volume 3258 of *Lecture Notes in Computer Science*, pages 482–495. Springer, 2004.
29. G. Pesant, C.-G. Quimper, L.-M. Rousseau, and M. Sellmann. The polytope of context-free grammar constraints. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 223–232, 2009.
30. D. Pisinger. An exact algorithm for large multiple knapsack problems. *European Journal of Operational Research*, 114(3):528–541, 1999.
31. D. Pisinger. Where are the hard knapsack problems? *Computers & Operations Research*, 32:2271–2284, 2005.
32. C.-G. Quimper and L.-M. Rousseau. A large neighbourhood search approach to the multi-activity shift scheduling problem. *Journal of Heuristics*, 16:373–392, 2010.
33. C.-G. Quimper and T. Walsh. Decomposing global grammar constraints. In *13th International Conference on Principles and Practices of Const raint Programming (CP-2007)*. Springer-Verlag, 2007.
34. J-C. Régin. Generalized arc consistency for global cardinality constraint. In *AAAI*, volume 1, pages 209–215, 1996.
35. F. Rossi, P. van Beek, and T. Walsh, editors. *The Handbook of Constraint Programming*. Elsevier, 2006.
36. D. Salvagnin. Orbital shrinking: a new tool for hybrid mip/cp methods. In *CPAIOR*, pages 204–215, 2013.
37. D. Salvagnin and T. Walsh. A hybrid mip/cp approach for multi-activity shift scheduling. In *CP*, pages 633–646, 2012.
38. A. Scholl, R. Klein, and C. Jürgens. Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & OR*, 24(7):627–645, 1997.
39. P. Shaw. A constraint for bin packing. In Mark Wallace, editor, *CP*, volume 3258 of *Lecture Notes in Computer Science*, pages 648–662. Springer, 2004.