Orbital Shrinking: Theory and Applications

Matteo Fischetti[a], Leo Liberti[b,c], Domenico Salvagnin[a], Toby Walsh[d]

[a] *DEI, Università di Padova, Italy*
[b] *IBM TJ Watson Research Center, Yorktown Heights, NY 10598, USA*
[c] *LIX, Ecole Polytechnique, 91128 Palaiseau, France*
[d] *NICTA and UNSW, Sydney, Australia*

Abstract

We present a method, based on formulation symmetry, for generating Mixed-Integer Linear Programming (MILP) relaxations with fewer variables than the original symmetric MILP. Our technique also extends to convex MINLP, and some nonconvex MINLP with a special structure. We consider an appropriate subgroup of the formulation group, and replace each orbit with a single variable. By means of the orbit barycenter, we are able to prove that the new MILP formulation is a relaxation of the original one. We showcase the effectiveness of our relaxation both on a library of symmetric MILP, and as part of a decomposition method applied to two important applications (multi-activity shift scheduling and multiple knapsack problem), showing that it can improve CPU times by several orders of magnitude compared to pure MIP or CP approaches.

Keywords: Mathematical programming, constraint programming, discrete optimization, symmetry, relaxation, MINLP

1. Introduction

Branch-and-Bound (BB) type methods often become very slow when the solution set is symmetric Gent et al. (2006); Margot (2010), due to the explorations of many symmetric subtrees. Given a Mathematical Programming (MP) formulation, we distinguish the automorphism group of its solution set (called the *solution group*) and the group of variable symmetries fixing the formulation (called the *formulation group*). The latter is usually defined as the group of variable index permutations which keep the objective function invariant, the right-hand-side constraint vector invariant, and permutes the order of the constraints Cohen et al. (2005); Margot (2002). It is very easy to show that the formulation group is a subgroup of the solution group.

Email addresses: matteo.fischetti@unipd.it (Matteo Fischetti), leoliberti@us.ibm.com (Leo Liberti), domenico.salvagnin@unipd.it (Domenico Salvagnin), toby.walsh@nicta.com.au (Toby Walsh)

Finding a universal technique for determining the solution group automatically would imply knowing the solution set *a priori*, which would make the optimization problem moot. On the other hand, various techniques for finding the formulation group of a Constraint Satisfaction Program (CSP) and of a MP have been proposed in the literature (see e.g. Bödi et al. (2013); Liberti (2008, 2012); Puget (2005)). The most efficient methods reduce to the graph isomorphism problem, which can be solved in practice using tools such as `nauty` McKay (2007).

Once some symmetries are known, they can be exploited in a variety of ways. In Constraint Programming (CP) and Mixed-Integer Programming (MIP), a common technique consists in trying to make some of the symmetric solutions infeasible by:

- adjoining Symmetry-Breaking Constraints (SBC) to the original formulation Liberti (2012); Liberti and Ostrowski (accepted); Smith (2010);
- using a clever branching strategy in constraint propagation Gent and Smith (2000) or in BB (e.g. isomorphism pruning Margot (2002, 2003) or orbital branching Ostrowski et al. (2008, 2011)).

In Semidefinite Programming (SDP), due to the fact that decision variables are matrices, symmetry can be exploited very naturally through group representation theory. This yields formulations with fewer variables (in fact, the variable matrix becomes block-diagonal) but having the same optimum Gatermann and Parrilo (2004).

A different approach is proposed in Bödi et al. (2013), where solving an Integer Linear Program (ILP) with a highly transitive solution group is essentially reduced to a line search in a lattice. The proposed method is very innovative, but most practically occurring ILPs have groups that are very far from being transitive. A generalization of this approach which aims to relax the requirement for high transitivity is given in Herr et al. (2013). Although applicability remains limited, this technique was used in solving the instance `toll-like` in the MIPLIB2010 library Koch et al. (2011), which was previously unsolved.

We propose another approach, called *Orbital Shrinking* (OS), for exploiting symmetry in MILP and certain subclasses of Mixed-Integer Nonlinear Programming (MINLP). Orbital shrinking is a relaxation technique: given a MIP P and a subgroup G of its formulation (or solution) group, it replaces each orbit of variables by a single variable. Therefore, OS produces compact MIP relaxations. If G is transitive, and hence has only one orbit, the resulting MIP is trivial, because it has only one variable. At the other extreme, if G is the trivial group, then there are as many orbits as there are variables, and the relaxation is the same as the original MIP.

To solve problems exactly, we employ the OS relaxation (OSR) in a general purpose decomposition framework, which we apply to two real-life applications: multi-activity shift scheduling and multiple knapsack problems. OS decomposition naturally provides a new way for designing hybrid MIP/CP decompositions: our 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 1.1, we review some main results on symmetry groups in the context of optimization problems. Then, in Section 2, we present orbital shrinking, and show that it yields a relaxation of the original problem. In Section 3 we define the OSR hierarchy. In Section 4 we analyze differences and similarities between OS and orbital branching, symmetric SDP block-diagonal reformulations, and core point algorithms. In Section 5 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 6 we describe a general decomposition framework based on orbital shrinking, while in Sections 7 and 8 we specialize the general framework to multi-activity shift scheduling and multiple knapsack problems, also reporting computational results. Conclusions are finally drawn in Section 9.

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 Fischetti and Liberti (2012); Salvagnin (2013); Salvagnin and Walsh (2012), by the same authors.

1.1. Some notation and terminology

Let P be an arbitrary MINLP of the form

$$\min f(x) \tag{1}$$

$$\forall i \in C \qquad g_i(x) \leq 0 \tag{2}$$

$$\forall j \in J \qquad x_j \in \mathbb{Z} \tag{3}$$

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.

We consider the formulation group G_P of P, containing the set of permutations $\pi \in S_n$ that leave the formulation of P unchanged, except for a possible reordering of the constraints. The practical applicability of this definition extends to Linear Programs (LP) and MILPs. With MINLPs, we restrict our attention to functional forms which are closed with respect to the usual operators $(+, -, \times, \div, (\cdot)^a)$ and unary functions $(\log, \exp)$. These expressions are easily represented by trees, and whole MINLP formulations can be represented by suitable Directed Acyclic Graphs (DAG) (see e.g. Belotti et al. (2009)). G_P is then obtained as a restriction of the automorphism group of this DAG to the set of variable indices of P (see e.g. Liberti (2012)). G_P can be computed by means of any graph isomorphism package such as Nauty McKay (1981) or Saucy Katebi et al. (2010): these both implement backtracking algorithms which are expontial-time in the worst case, but which are sufficiently fast in practice to be of use.

Any subgroup G of G_P partitions the set of variables into equivalence classes called *orbits* via its natural action: two variable indices i, j are in the same class if there is $g \in G$ such that $g(i) = j$. We denote by Ω_G the *orbital partition* of the action of G on $[n]$. We remark that, by definition, integer and continuous variables cannot be permuted with each other, so each orbit contains only integer

or only continuous variables. 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 $\pi \in G$ is applied. Finally, given a subset $I \subseteq [n]$, the *point-wise stabilizer* $G[I]$ of G with respect to I is the subgroup of G consisting of permutations π such that $\pi(i) = i$ for all $i \in I$.

2. Orbital Shrinking Relaxation

The OS relaxation with respect to a subgroup G of its formulation group could be best described as "formulation modulo G", as it replaces entire orbits by single variables. In this section, we will describe how to construct the OSR of a given optimization problem P, and show that this is indeed a relaxation of the original problem.

The first step is to classify variables and constraints according to their incidence and (non)linearity/convexity. Specifically, the subgroup G defining the OSR will be taken with respect to a certain subgroup of G_P rather than the whole of G_P, as defined below. Let us consider a partition (V_1, V_2) and (C_1, C_2, C_3) of the variables and constraints of P, respectively, which satisfies the following conditions:

- all constraints in C_1 are convex w.r.t. the variables in V_1 (in other words, 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;
- C_3 consists of all constraints not in C_1 or C_2.

The meaning of the constraint partition will become clear in the following; a remark is given after Corollary 2.6. Also note that, by definition, $G_P[V_2]$ is a group of variable permutations acting on V_1. Now let G be a subgroup of $G_P[V_2]$ and $\Omega = \Omega_G$ be the orbital partition of $[n]$ induced by G.

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 MP formulation obtained by the following procedure:

- for each $\omega \in \Omega$, define a variable z_ω, constrained to be integer if and only if the orbit ω consists of integer variables; 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 $\bar{g}_i(z) \leq 0$, obtained from $g_i(x) \leq 0$ through the formal substitution:

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

  for all $j \in \omega$;
- define the objective function $\bar{f}(z)$ as the result of applying to $f(x)$ the same formal substitution (4);

- ignore constraints in C_3.

Next, we show that P_{OSR} is a relaxation of P.

Lemma 2.1 is a basic generalization of Burnside's Lemma: we suspect it exists in the group theory literature, but we were not able to find it. Since the applicability to *any* function ψ makes the lemma interesting in its own right, we decided to provide a proof.

2.1 Lemma
Let $\omega \in \Omega$ and ψ be any function with $\operatorname{dom}\psi = [n]$. Then

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

Proof. For any $l \in \omega$, let $T_{jl} = \{\pi \in G : \pi(j) = l\}$. It is easy to show that $|T_{jl}| = |G[j]|$ (recall $G[j]$ is the point-wise stabilizer of j) for all $l \in \omega$: given an arbitrary $\pi \in T_{jl}$, we can define the map $\phi : G[j] \to T_{jl}$ as $\sigma \to \pi\sigma$ for any $\sigma \in G[j]$. This map is a bijection, with inverse $\phi^{-1} : \sigma \to \pi^{-1}\sigma$, so the two sets have the same cardinality. Hence

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

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

Lemma 2.2 is an application of Lemma 2.1 to the *barycenter* Bödi et al. (2013) of the group action, also called group average Gatermann and Parrilo (2004) or Reynolds operator Sturmfels (2008).

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

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

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

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

Proof. Define a function $X : [n] \to \mathbb{R}$ as $X(j) = x^*_j$. Applying Lemma 2.1 we get:

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

□

Lemma 2.3 is well known, and basically states that the barycenter is an invariant of the group action (see Sturmfels (2008)). We provide a proof for completeness, since it is very short.

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

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

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 follows by definition of $\bar{x}$ (see Eq. (6)). □

Finally, we prove that P_{OSR} is a relaxation. We recall that the P_{OSR} formulation involves an objective function $\bar{f}(z)$ and constraints $\bar{g}(z) \leq 0$.

2.4 Theorem
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 $\bar{f}(z^*) \leq f(x^*)$, hence the claim. Given x^*, let us construct the two points $\bar{x}$ and z^* as

$$\bar{x} = \frac{1}{|G|} \sum_{\pi \in G} \pi(x^*) \tag{8}$$

$$\forall \omega \in \Omega \qquad z^*_\omega = \sum_{j \in \omega} x^*_j. \tag{9}$$

For each constraint in C_1, $g_i(\bar{x}) \leq 0$ because $\bar{x}[V_2] = x^*[V_2]$ and g_i is convex in V_1. Similarly, for each constraint in C_2, we have $g_i(\bar{x}) = g_i(x^*) \leq 0$ because of Lemma 2.3. So, all constraint in $C_1 \cup C_2$ are satisfied by $\bar{x}$.

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 2.2

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

since x^* itself is feasible for those constraints. Thus, z^* is feasible for C_3, which implies it is feasible for P_{OSR}. As far as the objective function is concerned, we have $\bar{f}(z^*) = f(\bar{x}) \leq f(x^*)$ where the equality is by definition of $\bar{f}(z)$ and Lemma 2.2, while the inequality is by convexity of $f(x)$ and because $\bar{x}$ is a convex combination of solutions with the same cost (by symmetry). □

Corollary 2.5 essentially follows because the barycenter is a group invariant, and is at the basis of the ideas developed in Bödi et al. (2013); Gatermann and Parrilo (2004); Herr et al. (2013).

2.5 Corollary
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. □

Corollary 2.6 shows the easiest case where P_{OSR} is an exact reformulation.

2.6 Corollary
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.

□

2.1. Remarks

- Cor. 2.6 applies, for instance, in case a standard ILP model for the asymmetric Traveling Salesman Problem (TSP) is solved on symmetric input arc costs. In this setting, the group action induces 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.

- Cor. 2.6 provides a sufficient condition for P_{OSR} to be a reformulation, but this condition is not necessary. In fact, the OS based decomposition in Section 6 aims to iteratively refine an OSR "master" problem until it becomes possible to find a feasible solution in the original x variables which matches the solution in the OSR z variables.

- 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 $C_1 \cup C_2$.

- The convexity of the constraints in C_1 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.

  2.7 Example
  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.

- The constraints in C_2 can be convex or nonconvex, as long as their arguments are sums of original variables over the group orbits: essentially, the carry over to the OSR unchanged, aside from the replacement of orbital sums by the corresponding z variable.

3. The OSR hierarchy

As mentioned above, for every subgroup G of $G_P[V_2]$ there is a different OSR; when G is transitive and has only one orbit the OSR is trivial, whereas when G is trivial the OSR is the same as the original formulation. This yields a hierarchy $\mathcal{H}$ of relaxations, one for each element of the subgroup lattice $\mathcal{L}$ of $G_P[V_2]$. For any MP formulation P, let $v(P) = v_P(x^*)$ be the objective function value of a global optimum x^* of P; we assume without loss of generality that the optimization direction is minimization.

3.1 Proposition
Let $G \leq H \leq G_P[V_2]$ and P^G_{OSR}, P^H_{OSR} be the corresponding OSRs. Then $v(P^G_{\mathsf{OSR}}) \geq v(P^H_{\mathsf{OSR}})$.

Proof. Let k be the number of orbits of the action of G, and h be the number of orbits of the action of H. Since $G \leq H$, we have $k \geq h$, and, in particular, each orbit of H is partitioned into subsets of the orbits of G. If $k = h$ then $P^G_{\mathsf{OSR}} = P^H_{\mathsf{OSR}}$ and the result follows, so assume $k > h$. Let z^* be an optimum of P^G_{OSR}, ω_i be any orbit of the action of H, and $\{\theta_{ip} \mid p \leq \ell_i\}$ be the set of orbits of G which partition ω_i. We show that there is a feasible solution y^* of P^H_{OSR} with objective function value $v_{P^G_{\mathsf{OSR}}}(z^*)$. We index the components of z^* as z^*_{ip}, where $i \in \{1, \ldots, h\}$ and $p \in \{1, \ldots, \ell_i\}$, and define $y^*_i = \sum_{p \leq \ell_i} z^*_{ip}$. Consider any $i \leq h$. Since $z^*_{ip} = \sum_{j \in \theta_{ip}} x^*_j$ for some feasible solution x^* of P (by Eq. (9)), we have

$$y^*_i = \sum_{\substack{p \leq \ell_i \\ j \in \theta_{ip}}} x^*_j = \sum_{j \in \omega_i} x^*_j,$$

where the last equality follows because the θ_{ip}'s are a partition of ω_i. By construction, there is a feasible point y' of P^H_{OSR} such that $y'_i = \sum_{j\in\omega_i} x^*_j$, which implies that $y'_i = y^*_i$ for each $i \leq h$. Hence $y^* = y'$, which means that y^* is feasible in P^H_{OSR} as claimed. □

This means that $\mathcal{H}$ contains an order-isomorphic copy of $\mathcal{L}$.

3.2 Corollary
The subgroup lattice $\mathcal{L}$ of $G_P[V_2]$ can be order-embedded in the the lattice $\mathcal{H}$ partially ordered by $v(P^G_{\mathsf{OSR}}) \geq v(P^H_{\mathsf{OSR}})$ for each $G \leq H$ in $\mathcal{L}$.

Proof. Consider the partial order on $\mathcal{H}$ defined by $v(P^G_{\mathsf{OSR}}) \geq v(P^H_{\mathsf{OSR}})$ and $G \leq H$. □

4. The role of the barycenter

In this section we discuss the relationships between OSR and some other existing methods for exploiting symmetry in MP, namely orbital branching Ostrowski et al. (2011), the symmetry-based block-diagonal SDP reformulation Gatermann and Parrilo (2004) and core point algorithms Bödi et al. (2013); Herr et al. (2013). All of these methods are based on the concept of the *barycenter* of the group action, defined in Eq. (6).

4.1. Orbital branching

Orbital branching Ostrowski et al. (2011) is a particular branching technique which yields size reductions in BB trees when applied to symmetric MILPs involving binary variables. It is based on the observation that the orbital disjunction:

$$\left(\bigvee_{j\in\omega} x_j = 1\right) \vee \sum_{j\in\omega} x_j = 0$$

can be reduced to

$$x_h = 1 \vee \sum_{j\in\omega} x_j = 0$$

for any $h \in \omega$. Although the barycenter does not appear explicitly, by Lemma 2.2 we know that orbital sums are actually the barycenter in disguise. Since orbital branching is a branching technique and orbital shrinking is a reformulation technique, obviously the two are very different. The only similarity is that every right branch subproblem has a constraint $\sum_{j\in\omega} x_j = 0$, which, in OSR terms, can be subsumed by fixing $z_\omega = 0$.

4.2. Semidefinite programming

Symmetry invariance in SDPs is enforced in Gatermann and Parrilo (2004) as a constraint $X = \sigma X$ for all $\sigma \in G_{\mathsf{SDP}}$. The action of G_{SDP} on decision matrices permutes their rows and columns, and is induced by a conjugation action $\sigma X = \rho^{-1} X \rho$ given by a linear orthogonal representation $\rho(\sigma)$. By orthogonality, $\rho^{-1} = \rho^{\top}$, which implies that $X = \sigma X$ can be written as $X = \rho^{\top} X \rho$, which in turn yields $\rho^{\top} X = X \rho$. This commutativity of the group action induces, by Schur's lemma, a block diagonal form on X, which is the reason why the SDP can be written in a more compact form, i.e. only using the nonzero elements on the block-diagonalized decision matrix.

Here are the main differences between Gatermann and Parrilo (2004) and the OSR:

1. block-diagonalization yields exact reformulations of SDPs, whereas OSR yields a hierarchy of relaxations of MILPs and certain MINLPs;

2. block-diagonalization is based on the conjugation action of a linear orthogonal representation of G_{SDP}, whereas OSR is based on the standard action of a subgroup of S_n over $[n]$.

The only similarity is that the theories behind the two methods both employ the barycenter in the proofs. In particular, the barycenter is used in the proof of (Gatermann and Parrilo, 2004, Thm. 3.3) to show that symmetric SDPs having a feasible decision matrix X_1 also have another feasible X_σ, with the same objective function value, which is invariant with respect to the SDP group G_{SDP}. This is the first appearance in the optimization literature of Cor. 2.5.

The following example shows that the effect of block-diagonalization and OSR are completely different even on a toy example taken from Gatermann and Parrilo (2004).

4.1 Example

Example 4.3 in Gatermann and Parrilo (2004) considers the SDP $\min\langle C, X\rangle$, *where* $X = (x_{ij})$ *and* $C = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$, *subject to* $x_{ij} = x_{ji}$ *for all* $i < j \leq 3$, $x_{12} = x_{13}$ *and* $X \succeq 0$. *The formulation group* G_{SDP} *is given by simultaneous permutations of the last two rows and columns of* C *and* X, *and is isomorphic to the cyclic group of order 2. According to Gatermann and Parrilo (2004), The corresponding block-diagonalization yields the reduced SDP* $\min(\langle C_1, X_1\rangle + y)$, *with* $C_1 = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$, $X_1 = \begin{pmatrix} x_{11} & \sqrt{2}x_{12} \\ \sqrt{2}x_{12} & x_{22}+x_{23} \end{pmatrix}$, $X_1 \succeq 0$ *and* $y = x_{22} - x_{23}$, *which has 4 variables. By contrast, if we relax* $X \succeq 0$ *by* $X \geq 0$ *and write out the corresponding LP* P *we get* $\min(x_{22} + x_{33})$ *subject to* $x_{ij} = x_{ji}$ *for all* $i < l \leq 3$ *and* $x_{12} = x_{13}$, *which has formulation group* $G_P = \langle (x_{22}, x_{33})\rangle$, *with a unique nontrivial orbit* $\{x_{22}, x_{33}\}$. *Consequently, the OSR, which in this case is also an exact reformulation by Cor.* (2.5), *has 7 variables, underlining the difference between the two techniques.*

4.3. Core point algorithms

The papers Bödi et al. (2013); Herr et al. (2013) discuss a new approach to optimize symmetric MILP (or convex MINLP). The first paper Bödi et al. (2013) requires G_P to be at least transitive on $[n]$ (i.e. its action on $[n]$ consists of a single orbit). This requirement is partly relaxed the second paper Bödi et al. (2013): G_P is only assumed to have a direct product of (possibly trivial) symmetric groups as a subgroup.[1]

Let P be the ILP $\max\{c^\top x \mid Ax \leq b \wedge x \in \mathbb{Z}^n\}$, and let $G \leq G_P$ have a transitive action on $[n]$. The papers Bödi et al. (2013); Herr et al. (2013) consider a decomposition of the feasible region $\mathcal{F}$ of P in the *fixed subset* $F_G = \{x \in \mathcal{F} \mid \forall g \in G\ (gx = x)\}$ (its span is called the *fixed subspace*) and the affine subspaces $H_c^k = \{x \in \mathbb{R}^n \mid c^\top x = k\}$, where $k \in \mathbb{Z}$, which contain the level sets. Without loss of generality, c is assumed to be a coprime integral vector, i.e. one whose components have unit greatest common divisor. It is shown in Bödi et al. (2013) that $\{H_c^k \mid k \in \mathbb{Z}\}$ is a partition of $\mathbb{Z}^n$ for each coprime c.

In Bödi et al. (2013), G is assumed to be transitive: by definition, this implies that for each $i \neq j \in [n]$ there is $\pi \in G$ mapping c_i to c_j, yielding c to be a scaling of the all-one vector $\mathbf{1}$. Enters the barycenter: since it is invariant with respect to the action of G, it spans the fixed subspace. By definition, the barycenter is a scaled sum of all the orbit elements (see Eq. (7)), and hence, by the same reasoning as the one carried out for c, it is also a scaled version of $\mathbf{1}$. In particular, the span of c is equal to the fixed subspace; and, moreover, each affine subspace H_c^k is orthogonal to the fixed subspace.

The fact that F_G aligns precisely with the objective function direction is obviously as rare as full transitivity of G, but it pays off handsomely: if y is an optimal solution of the continuous relaxation, then the barycenter $\zeta\mathbf{1}$ (where $\zeta = \frac{1}{n}\sum_j y_j$) is also optimal, so $\lfloor\zeta\rfloor\mathbf{1}$ is a feasible integer point; hence the ILP optimal value must be at least as large as that of $\lfloor\zeta\rfloor\mathbf{1}$, namely $\lfloor\zeta\rfloor n$. Trivially, we also have $\lfloor\zeta n\rfloor$ as an upper bound. Hence it suffices to find an integer feasible point in H_c^k for the largest possible k in $K = \{\lfloor\zeta\rfloor n, \ldots, \lfloor\zeta n\rfloor\}$. In Bödi et al. (2013), Alg. A suggests a direct search for decreasing values of k (an obvious $\log(|K|)$ improvement could be given by using bisection on K).

In general, the subproblem of (Bödi et al., 2013, Alg. A), which consists of finding an integer feasible point in each H_c^k, is hard. However, if G is μ-transitive (with $\mu \geq \lfloor\frac{n}{2}\rfloor + 1$), then the integer feasible points in H_c^k that are closest to the fixed subspace, called *core points*, can be found in polynomial time (Bödi et al., 2013, Alg. B). Transitivity rarely occurs in practical problems; and μ-transitivity, requiring the existence of permutations mapping any μ-tuple to any other, is even rarer.

The paper Herr et al. (2013), subsequent to Bödi et al. (2013), relaxes the transitivity requirements, generalizes the definition of a core point, and proposes

[1] We remark that the instance library data given in Liberti (2012) shows that many formulation groups are *isomorphic* to products of symmetric groups. Further analysis, however, shows that in most cases formulation groups are *not* themselves products of symmetric groups.

two algorithms for solving symmetric MILPs and convex MINLPs. The first algorithm is a generalization of the core point algorithm of Bödi et al. (2013): to address the fact that $c = \mathbf{1}$ is no longer the unique generator of the fixed subspace, (Herr et al., 2013, Alg. A) still solves integer feasibility subproblems on H_c^k but also checks their objective function values. The second algorithm is based on a smart parametrization of generalized core points, and the fact that it is sufficient to find an optimal core point. It turns out that this parametrization results in a reformulation of the original problem. As mentioned above, the requirement on G for these algorithms to work is that it should be a direct product of symmetric group $S_{k_1} \times \cdots \times S_{k_d}$. The reformulation adds new sets of variables: $t_i \in \mathbb{Z}$ for each $i \leq d$, and $s_{ij} \in \{0, 1\}$ for each $i \leq d$ and $j < k_i$; and new sets of constraints:

$$\begin{aligned} &\forall i \leq d \qquad && x_{ij} = t_i \mathbf{1}_{k_i} + \sum_{j<k_i} s_{ij} \mathbf{c}_j \\ &\forall i \leq d \qquad && \sum_{j<k_i} s_{ij} \leq 1, \end{aligned}$$

where $\mathbf{1}_{k_i}$ is the all-one vector of size k_i, $\mathbf{c}_j = \sum_{h \leq j} e_h$ (with e_h the h-th unit vector of the standard basis of $\mathbb{R}^n$) is a representative of the core set of H_c^j, and the components of x have been appropriately re-indexed using i, j. The original variables x can then be eliminated using the equivalent expressions in the t and s variables.

Here are the main differences between OSR and core point algorithms:

- OSR does not make *any* assumption on the structure of G, other than it should be nontrivial, whereas core points algorithms make strong assumptions on G;
- OSR is a relaxation method, i.e. it acts on the formulation, whereas the core point algorithms in Bödi et al. (2013) do not;
- the reformulation derived in the second core point algorithm in Herr et al. (2013) is exact, and, in particular, different from the OS relaxation (see Example 4.2 below).

The only similarity between the OSR and the results in Bödi et al. (2013) is the same as for the SDP techniques discussed above: the fixed subspace LP reformulation (Bödi et al., 2013, Eq. (3)) is the same as the OS relaxation whenever there are no integer variables (see Cor. 2.5). The only similarity with Herr et al. (2013) is that the barycenter is the point of departure to derive a reformulation of the original MIP.

4.2 Example

In this example, we show that the OSR is different from the core point reformulation. Consider the ILP $\max x_1 + x_2 + 2(x_3 + x_4 + x_5)$ *subject to* $x_1 + x_2 = 1$, $x_3 + x_4 + x_5 = 2$ *and* $x \in \{0, 1\}^5$. *We take* $G = G_P = \langle (1, 2), (3, 4, 5) \rangle = S_2 \times S_3$,

which has the two orbits $\{1,2\}$ *and* $\{3,4,5\}$*. After reindexing the* x *variables, the ILP becomes:*

$$\begin{array}{lllll} \max\limits_{x\in\{0,1\}^5} & x_{11}+x_{12} & +2x_{21}+2x_{22}+2x_{23} & & \\ & x_{11}+x_{12} & & = & 1 \\ & & x_{21}+x_{22}+x_{23} & = & 2. \end{array}$$

Now the core point equations read:

$$\begin{aligned} \begin{pmatrix} x_{11} \\ x_{12} \end{pmatrix} &= \begin{pmatrix} 1 \\ 1 \end{pmatrix} t_1 + \begin{pmatrix} 1 \\ 0 \end{pmatrix} s_{11} + \begin{pmatrix} 1 \\ 1 \end{pmatrix} s_{12} \\ \begin{pmatrix} x_{21} \\ x_{22} \\ x_{23} \end{pmatrix} &= \begin{pmatrix} 1 \\ 1 \end{pmatrix} t_2 + \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} s_{21} + \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} s_{22} + \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} s_{23}, \end{aligned}$$

and after replacement, the core point reformulation is:

$$\begin{array}{rcl} \max 2t_1 + s_{11} + 2s_{12} + 6t_2 + 2s_{21} + 4s_{22} + 6s_{23} & & \\ 2t_1 + s_{11} + 2s_{12} & = & 1 \\ 3t_2 + s_{21} + 2s_{22} + 3s_{23} & = & 2 \\ s_{11} + s_{12} & \leq & 1 \\ s_{21} + s_{22} + s_{23} & \leq & 1 \\ t_1, t_2 & \in & \mathbb{Z} \\ s_{11}, s_{12}, s_{21}, s_{22}, s_{23} & \in & \{0,1\}, \end{array}$$

whereas the OSR is:

$$\begin{aligned} \max z_1 + 2z_2 \\ z_1 = 1 \\ z_2 = 2 \\ z_1 \in \{0,\dots,2\} \\ z_2 \in \{0,\dots,3\}, \end{aligned}$$

marking the difference between the two approaches.

For specific examples, it might happen that OSR and the core point reformulation turn out to be similar: take e.g. Example 4.2, and change the constraint $x_3 + x_4 + x_5 = 2$ to $x_3 + x_4 + x_5 = 1$. Then the assignment constraints make it possible to immediately infer $t = 0$ and $s_{12} = s_{22} = s_{23} = 0$ in the core point reformulation, yielding $\max s_{11} + s_{21}$ subject to $s_{11} = 1$ and $s_{21} = 1$, which are the same objective function and constraints of the OSR. But whereas s are binary variables, z are general integer variables: this difference is a token of the fact that whereas OSR variables encapsulate whole orbits, the core point reformulation describes a single orbit representative.

5. Mining for the best subgroup: features and MILP results

In this section we discuss the choice of the formulation subgroup G with respect to which the OSR is constructed. As remarked previously, if G is the trivial group, 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, 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 some 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.

5.1. Automatic generation of the whole symmetry group

The formulation group is detected automatically using the techniques discussed in Liberti (2012): 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` McKay (1981)) is then called on the DAG. The orbital-shrinking relaxation is constructed automatically using a mixture of `bash` scripting, GAP gap4 (2007), AMPL Fourer and Gay (2002), and ROSE Liberti et al. (2010).

5.2. 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 O4_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/`.

5.3. Generator ranking

The OSR based on $G = G_P[V_2]$ 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 OSR 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 transitive, there is only 1 orbit of size n: 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 $G_P[V_2]$ is impractical, we look at its generator list $\Pi = (\pi_0, \ldots, \pi_k)$ (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 the generator 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 $G_P[V_2]$ induced by the sublist $(\pi_0, \cdots, \pi_\ell)$. This leads to a subgroup chain

$$G_0 \leq G_1 \leq \cdots \leq G_k = G_P[V_2]$$

in the subgroup lattice $\mathcal{L}$ (see Section 3) 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 in terms of bound quality — having a large $\mathrm{fix}(\pi)$ implies that the generated subgroup is likely to remain valid 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 deem 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.

5.4. Choosing a good set of generators

The proposed 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 yield the same bound independently of ℓ, and were hence discarded from this experiment. The remaining 23 instances:

```
ca36243 clique9 cod105 cod105r cod83 cod83r cod93 cod93r cov1075
cov1076 cov954 mered O4_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;

	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

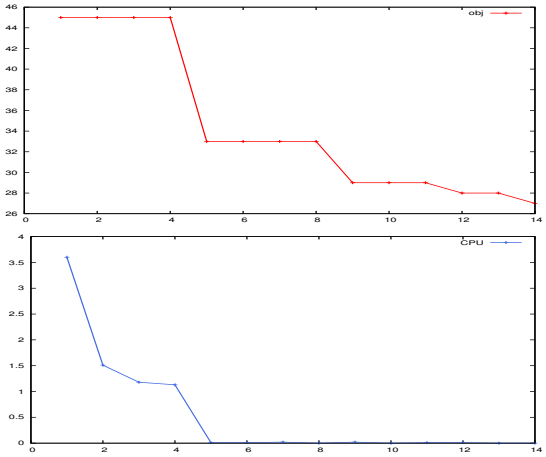

Figure 1: Bound values and CPU times against the number ℓ of generators for instance `sts81`.

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).

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

Instance	G_ℓ	LP	CPU	$\frac{\text{inc}(G_\ell)}{\text{inc}(G_P[V_2])}$
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.

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'

$$\text{inc}(G', i) := \left| \bigcup_{\omega' \in \Omega'} \omega' \cap \{j \in [n] : a_{ij} \neq 0\} \right|, \tag{10}$$

and finally to the whole MILP formulation

$$\text{inc}(G') = \sum_{i \in [r]} \text{inc}(G', i). \tag{11}$$

The rightmost column of Table 1 reports the relative incidence of G_ℓ, computed as $\text{inc}(G_\ell)/\,\text{inc}(G_P[V_2])$, 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 $\text{inc}(G_\ell)/\,\text{inc}(G_P[V_2])$ is as close to 0.82 as possible.

5.5. Bound strength

The quality of the OSR relaxation we obtain with the method of Section 5.4 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.

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
`04_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.

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.

6. Orbital Shrinking decomposition

Let P be a MINLP as in the previous sections and let G be the chosen formulation subgroup for P. Using G, we can construct the OSR 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 Hooker and Ottosson (2003), 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 Linderoth et al. (2009).

6.1. The slave feasibility problem

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

$$\begin{aligned} \forall i \in C \quad g_i(x) &\leq 0 && (12)\\ \forall \omega \in \Omega \quad \sum_{j \in \omega} x_j &= z^*_\omega && (13)\\ \forall j \in J \quad x_j &\in \mathbb{Z} && (14) \end{aligned}$$

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.

6.2. Symmetry of the slave problem

It is important to note that, by construction, problem $R(z^*)$ may also have a nontrivial formulation group, so symmetry may still be an issue while solving $R(z^*)$.

6.1 Lemma
The group G used to generate P_{OSR} is a subgroup of the formulation group G_R of $R(z^*)$.

Proof. Let P be the MP formulation (1)-(3), and $\pi \in G \leq G_P$. Then π stabilizes the constraints (2) of P, which appear in $R(z^*)$ as (12). Since constraints (13) are generated by means of the action of G, π fixes each orbit of this action, and hence π also stabilizes (13). Thus $\pi \in G_R$ as claimed. □

On the other hand, G_R could be a subgroup of G_P or vice versa, or be such that $G_R = G_P$, depending on the value of z^*, on the action of G and on the objective function of P.

6.2 Example
Let P be the problem $\max\{\sum_{i=1}^{5} x_i \mid \sum_{i=1}^{5} x_i \leq 1 \wedge \forall i \leq 5\ x_i \in \{0,1\}\}$*: we have $G_P \cong S_5$. Consider the subgroup $G = \langle (1,2),(3,4),(3,5)\rangle \cong C_2 \times S_3$, having the two orbits $\{1,2\}$ and $\{3,4,5\}$. Then $R(z^*)$ is:*

$$\begin{aligned} x_1 + x_2 + x_3 + x_4 + x_5 &\leq 1 \\ x_1 + x_2 &= z_1^* \\ x_3 + x_4 + x_5 &= z_2^*, \end{aligned}$$

which, for any feasible z^*, has formulation group $G_R = G$, a proper subgroup of G_P. Obviously, by modifying P so that the objective is $x_1 + x_2 + 2(x_3 + x_4 + x_5)$ we have $G_P = G$ and hence $G_R = G_P$.

6.3 Example
Let P be the problem

$$\max\left\{\sum_{i\leq 3} x_i + 2\sum_{4\leq i\leq 6} x_i \mid \sum_{i\leq 6} x_i \leq 6 \wedge \forall i \leq 6\ x_i \in \{0,1\}\right\},$$

and take $x^* = (1,1,1,1,1,1)$ as the optimal solution, with corresponding P_{OSR} solution z^*. We have $G_P = \langle (1,2),(1,3),(4,5),(4,6)\rangle \cong S_3 \times S_3$. Consider $G = G_P$, having two orbits $\{1,2,3\}$ and $\{4,5,6\}$. Then $z^* = (3,3)$ and $R(z^*)$ is:

$$\begin{aligned} x_1 + x_2 + x_3 + x_4 + x_5 + x_6 &\leq 6 \\ x_1 + x_2 + x_3 &= 3 \\ x_4 + x_5 + x_6 &= 3. \end{aligned}$$

The formulation group G_R certainly contains all of the permutations of $G_P = G$ (as per Lemma 6.1), and also the permutation $(1,4)(2,5)(3,6)$, which is not in G_P, and makes $G_R \cong (S_3 \times S_3) \rtimes C_2$. So in this case G_P is a proper subgroup of G_R.

The issues arising from G_R being nontrivial are usually solvable because (i) linking constraints (13) 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. In addition, the method intuitively allows for symmetry to exploited twice: once in the (symmetry-free) master, where P_{OSR} effectively enumerates equivalence classes of solutions of P, and once in each slave, where more traditional symmetry breaking techniques, such as orbital branching or isomorphism pruning can be used.

6.3. Dealing with continuous variables

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 (13) 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 Mittelmann and Salvagnin (2013) to solve a very challenging instance of 3-dimensional quadratic assignment problem. Interestingly, the role of MIP and CP was swapped in Mittelmann and Salvagnin (2013): a CP solver was used to enumerate all feasible solutions of the master problem, while a MIP solver was used solve the optimization slaves.

7. 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.

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:

$$\min \sum_{a\in W}\sum_{t\in T} c_{at}y_{at} + \sum_{a\in W}\sum_{t\in t} c^+_{at}s^+_{at} + \sum_{a\in W}\sum_{t\in T} c^-_{at}s^-_{at} \tag{15}$$

$$\forall a \in W, t \in T \quad y_{at} - s^+_{at} + s^-_{at} = d_{at} \tag{16}$$

$$\forall a \in W, t \in T \quad \sum_{e\in E} x_{eat} = y_{at} \tag{17}$$

$$\forall e \in E \quad \langle x \text{ defines a feasible shift } \rangle \tag{18}$$

$$\forall a \in W, t \in T \quad y_{at}, s^+_{at}, s^-_{at} \in \mathbb{Z}^+ \tag{19}$$

$$\forall e \in E, a \in W, t \in T \quad x_{eat} \in \{0,1\} \tag{20}$$

where x_{eat} are binary variables, each of which is equal to 1 if employee e is assigned to activity a in period t. Model (15)-(20) is symmetric because employees are assumed to be identical, so with $|E|$ employees we have a symmetry group of order $|E|!$. As for orbits, each orbit contains all the variables for all employees associated with a given entity (for example, for each activity a and time period t we have an orbit containing the variables x_{*at}).

Depending on how we formulate constraints (18), 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 Côté et al. (2011a); Pesant et al. (2009) 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 Côté et al. (2011a); Pesant (2004) 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 Hopcroft and Ullman (1979) for the corresponding grammar Pesant et al. (2009); Quimper and Walsh (2007).

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

Pesant et al. (2009), 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 Côté et al. (2011b). 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 Côté et al. (2011b). 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 Quimper and Walsh (2007)), still some cardinality constraints may be very awkward to express, see Salvagnin and Walsh (2012) 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 6 in order to turn OSR into an exact procedure.

7.1. MIP model

The MIP model that we use is a simple modification of the general model (15)-(20). 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 model, but since the corresponding aggregated model is quite compact, this is usually well worth it.

For each shift type Ω_k, the aggregated MIP model, i.e., the orbital shrinking relaxation that acts as our master problem, decides how many employees are assigned a shift in Ω_k, and then computes an aggregated integer flow of the

same value. In details:

$$\min \sum_{k\in K}\sum_{a\in W}\sum_{t\in T} c_{at}y^k_{at} + \sum_{a\in W}\sum_{t\in T} c^+_{at}s^+_{at} + \sum_{a\in W}\sum_{t\in T} c^-_{at}s^-_{at} \tag{21}$$

$$\forall a \in W, t \in T \quad \sum_{k\in K} y^k_{at} - s^+_{at} + s^-_{at} = d_{at} \tag{22}$$

$$\forall k \in K \quad \mathsf{regular}(y^k, w^k, \mathrm{DFA}^k) \tag{23}$$

$$\forall k \in K \quad \langle \text{cardinality constraints for } y^k \rangle \tag{24}$$

$$\sum_{k\in K} w^k \leq E \tag{25}$$

$$\forall k \in K, a \in W, t \in T \quad w^k, y^k_{at}, s^+_{at}, s^-_{at} \in \mathbb{Z}^+ \tag{26}$$

Note that we use the notation of constraint (23) 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 (24) 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 be scheduled, it can be imposed in constraint (25).

7.2. 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 $\bar{w}^k$ employees, with their aggregated shifts described by $\bar{y}_{at}$; then the corresponding CP model can easily be formulated by using several *global constraints* Rossi et al. (2006). Global constraints are used within a CP solver to represent general purpose and common substructures, for which efficient and effective constraint propagators are known. In our case, it turned out that, using standard global constraints from the literature and implementing some specific propagators for the model at hand (see Salvagnin and Walsh (2012) for details), the CP model was always extremely fast in proving whether the aggregated solution can be turned into a solution of the original model.

7.3. Computational Results

We tested our method on the multi-activity instances used in Côté et al. (2011a,b); Quimper and Rousseau (2010). This testbed is derived from a real-world store, and contains instances with 1 to 10 working activities (each class has 10 instances).

We implemented our method in C++, using IBM ILOG CPLEX 12.2 cplex (2012) as black box MIP solver, and Gecode 3.7.3 Gecode Team (2012) 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.

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 Salvagnin and Walsh (2012) 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 described in Section 6. This means that we disable dual reductions (otherwise the decomposition would not be correct) and use CPLEX callbacks framework to implement the decomposition.

Table 3 reports a comparison between the proposed method and others in the literature, for a number of activities from 1 to 10. `cpx-reg` refers to the explicit model based on the regular constraint in Côté et al. (2011a), while `grammar` refers to the implicit model based on the grammar constraint in Côté et al. (2011b). Note that for `grammar` we are reporting the results from Côté et al. (2011b), 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.

Finally, Table 4 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

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

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.

Table 4: 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

8. Application to the Multiple Knapsack Problem

In the present section, we specialize the general framework of Section 6 to the multiple knapsack problem (MKP) Pisinger (1999); Scholl et al. (1997). This a natural generalization of the traditional knapsack problem Martello and

Toth (1990), 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

$$\max \sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij} \tag{27}$$

$$\forall i \in \{1, \ldots, m\} \quad \sum_{j=1}^{n} w_j x_{ij} \leq C_i \tag{28}$$

$$\forall j \in \{1, \ldots, n\} \quad \sum_{i=1}^{m} x_{ij} \leq 1 \tag{29}$$

$$x \in \{0,1\}^{m \times n} \tag{30}$$

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 Q reads

$$\max \sum_{k=1}^{K} p_k y_k \tag{31}$$

$$\sum_{k=1}^{K} w_k y_k \leq mC \tag{32}$$

$$\forall k \in \{1, \ldots, K\} \quad 0 \leq y_k \leq |V_k| \tag{33}$$

$$y \in \mathbb{Z}_+^K \tag{34}$$

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 Shaw (2004) and exploiting the CDBF Gent and Walsh (1997) 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:

$$\mathsf{binpacking}(b, l, s) \tag{35}$$

$$b_{j-1} \leq b_j \quad \text{if } s_{j-1} = s_j \tag{36}$$

where (36) 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 Gilmore and Gomory (1961). 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

$$\sum_{p \in P} a_{pk} x_p = y_k^* \tag{37}$$

$$\sum_{p \in P} x_p = m \tag{38}$$

$$x_p \in \mathbb{Z}_+ \tag{39}$$

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.

8.1. Computational Experiments

We implemented our codes in C++, using IBM ILOG CPLEX 12.4 cplex (2012) as black box MIP solver and Gecode 3.7.3 Gecode Team (2012) 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 Pisinger (2005). More details about our instance generation procedure can be found in Salvagnin (2013) 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 mentioned 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 (27)−(30), denoted as `cpxorig`. The other is obtained by aggregating the binary variables corresponding to identical items. The model,

denoted as `cpx`, reads

$$\max \sum_{i=1}^{m} \sum_{k=1}^{K} p_j z_{ik} \tag{40}$$

$$\forall i \in \{1, \ldots, m\} \quad \sum_{k=1}^{K} w_j z_{ik} \leq C \tag{41}$$

$$\forall k \in \{1, \ldots, K\} \quad \sum_{i=1}^{m} z_{ik} \leq U_k \tag{42}$$

$$z \in \mathbb{Z}_{+}^{m \times K} \tag{43}$$

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 5. 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 5: 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 8, that differs on the models used for the feasibility check. The first variant, denoted by `BPstd`, is based on the compact model (35)−(36). The second and the third variants are both based on the extended model (37)−(39),

Table 6: 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

but differs on the solver used: a CP solver for `BPcgCP` and a MIP solver for `BPcgMIP`. All variants use model (31)−(34) 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 6. 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 (36) 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. `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 7 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 7: 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

9. Conclusions

We discussed a new methodology for deriving a relaxation of symmetric discrete optimization problems, based on variable aggregation within orbits.

The approach, called orbital shrinking, sometimes leads to an exact and symmetry-free reformulation of a given problem. 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 new, a novelty of our approach is that it is driven by the automatically-detected formulation group 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.

Acknowledgments

The first and third authors were supported by the University of Padova (Progetto di Ateneo "Exploiting randomness in Mixed Integer Linear Programming"), and by MiUR, Italy (PRIN project "Mixed-Integer Nonlinear Optimization: Approaches and Applications") The second author was partially supported by grants Digiteo Chair 2009-14D "RMNCCO" and Digiteo 2009-55D "ARM".

Belotti, P., Lee, J., Liberti, L., Margot, F., Wächter, A., 2009. Branching and bounds tightening techniques for non-convex MINLP. Optimization Methods and Software 24 (4), 597–634.

Bödi, R., Herr, K., Joswig, M., 2013. Algorithms for highly symmetric linear and integer programs. Mathematical Programming 137, 65–90.

Cohen, D., Jeavons, P., Jefferson, C., Petrie, K., Smith, B., 2005. Symmetry definitions for constraint satisfaction problems. In: van Beek, P. (Ed.), Constraint Programming. Vol. 3709 of LNCS. Springer, pp. 17–31.

Côté, M.-C., Gendron, B., Quimper, C.-G., Rousseau, L.-M., 2011a. Formal languages for integer programming modeling of shift scheduling problems. Constraints 16 (1), 54–76.

Côté, M.-C., Gendron, B., Rousseau, L.-M., 2011b. Grammar-based integer programming models for multiactivity shift scheduling. Management Science 57 (1), 151–163.

cplex, 2012. CPLEX 12.4 User's Manual. IBM ILOG.

Fischetti, M., Liberti, L., 2012. Orbital shrinking. In: Proceedings of ISCO.

Fourer, R., Gay, D., 2002. The AMPL Book. Duxbury Press, Pacific Grove.

gap4, 2007. GAP – Groups, Algorithms, and Programming, Version 4.4.10. The GAP Group.
URL `http://www.gap-system.org`

Gatermann, K., Parrilo, P., 2004. Symmetry groups, semidefinite programs and sums of squares. Journal of Pure and Applied Algebra 192, 95–128.

Gecode Team, 2012. Gecode: Generic constraint development environment. Available at `http://www.gecode.org`.

Gent, I., Smith, B., 2000. Symmetry breaking in constraint programming. In: Horn, W. (Ed.), Proceedings of the 14th European Conference on Artificial Intelligence. Vol. 14 of ECAI. IOS Press, Amsterdam, pp. 599–603.

Gent, I. P., Petrie, K. E., Puget, J.-F., 2006. Symmetry in constraint programming. In: Rossi, F., van Beek, P., Walsh, T. (Eds.), Handbook of Constraint Programming. Elsevier, pp. 329–376.

Gent, I. P., Walsh, T., 1997. From approximate to optimal solutions: Constructing pruning and propagation rules. In: IJCAI. Morgan Kaufmann, pp. 1396–1401.

Gilmore, P. C., Gomory, R. E., 1961. A linear programming approach to the cutting-stock problem. Operations Research 9, 849–859.

Herr, K., Rehn, T., Schürmann, A., 2013. Exploiting symmetry in integer convex optimization using core points. Operations Research Letters 41, 298–304.

Hooker, J. N., Ottosson, G., 2003. Logic-based Benders decomposition. Mathematical Programming 96 (1), 33–60.

Hopcroft, J. E., Ullman, J. D., 1979. Introduction to Automata Theory, Languages and Computation. Addison Wesley.

Katebi, H., Sakallah, K. A., Markov, I. L., 2010. 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. Vol. 6175. pp. 113–127.

Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R., Danna, E., Gamrath, G., Gleixner, A., Heinz, S., Lodi, A., Mittelmann, H., Ralphs, T., Salvagnin, D., Steffy, D., Wolter, K., 2011. Miplib 2010. Mathematical Programming Computation 3 (2), 103–163.

Liberti, L., 2008. Automatic generation of symmetry-breaking constraints. In: Yang, B., Du, D.-Z., Wang, C. (Eds.), Combinatorial Optimization, Constraints and Applications (COCOA08). Vol. 5165 of LNCS. Springer, Berlin, pp. 328–338.

Liberti, L., 2012. Reformulations in mathematical programming: Automatic symmetry detection and exploitation. Mathematical Programming A 131, 273–304.

Liberti, L., Cafieri, S., Savourey, D., 2010. Reformulation optimization software engine. In: Fukuda, K., van der Hoeven, J., Joswig, M., Takayama, N. (Eds.), Mathematical Software. Vol. 6327 of LNCS. Springer, New York, pp. 303–314.

Liberti, L., Ostrowski, J., accepted. Stabilizer-based symmetry breaking constraints for mathematical programs. Journal of Global Optimization.

Linderoth, J., Margot, F., Thain, G., 2009. Improving bounds on the football pool problem by integer programming and high-throughput computing. INFORMS Journal on Computing 21 (3), 445–457.

Margot, F., 2002. Pruning by isomorphism in branch-and-cut. Mathematical Programming 94, 71–90.

Margot, F., 2003. Exploiting orbits in symmetric ILP. Mathematical Programming B 98, 3–21.

Margot, F., 2010. 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, pp. 647–686.

Martello, S., Toth, P., 1990. Knapsack Problems: Algorithms and Computer Implementations. Wiley.

McKay, B., 2007. *nauty* User's Guide (Version 2.4). Computer Science Dept. , Australian National University.

McKay, B. D., 1981. Practical graph isomorphism.

Mittelmann, H. D., Salvagnin, D., 2013. On solving a hard quadratic 3-dimensional assignment problem. Tech. rep., DEI, University of Padova.

Ostrowski, J., Linderoth, J., Rossi, F., Smriglio, S., 2008. Constraint orbital branching. In: Lodi, A., Panconesi, A., Rinaldi, G. (Eds.), IPCO. Vol. 5035 of LNCS. Springer, pp. 225–239.

Ostrowski, J., Linderoth, J., Rossi, F., Smriglio, S., 2011. Orbital branching. Mathematical Programming 126, 147–178.

Pesant, G., 2004. 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. Vol. 3258 of Lecture Notes in Computer Science. Springer, pp. 482–495.

Pesant, G., Quimper, C.-G., Rousseau, L.-M., Sellmann, M., 2009. The polytope of context-free grammar constraints. Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 223–232.

Pisinger, D., 1999. An exact algorithm for large multiple knapsack problems. European Journal of Operational Research 114 (3), 528–541.

Pisinger, D., 2005. Where are the hard knapsack problems? Computers & Operations Research 32, 2271–2284.

Puget, J.-F., 2005. Automatic detection of variable and value symmetries. In: van Beek, P. (Ed.), Constraint Programming. Vol. 3709 of LNCS. Springer, New York, pp. 475–489.

Quimper, C.-G., Rousseau, L.-M., 2010. A large neighbourhood search approach to the multi-activity shift scheduling problem. Journal of Heuristics 16, 373–392.

Quimper, C.-G., Walsh, T., 2007. Decomposing global grammar constraints. In: 13th International Conference on Principles and Practices of Const raint Programming (CP-2007). Springer-Verlag.

Rossi, F., van Beek, P., Walsh, T. (Eds.), 2006. The Handbook of Constraint Programming. Elsevier.

Salvagnin, D., 2013. Orbital shrinking: a new tool for hybrid mip/cp methods. In: CPAIOR. pp. 204–215.

Salvagnin, D., Walsh, T., 2012. A hybrid mip/cp approach for multi-activity shift scheduling. In: CP. pp. 633–646.

Scholl, A., Klein, R., Jürgens, C., 1997. Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. Computers & OR 24 (7), 627–645.

Shaw, P., 2004. A constraint for bin packing. In: Wallace, M. (Ed.), CP. Vol. 3258 of Lecture Notes in Computer Science. Springer, pp. 648–662.

Smith, B., 2010. Symmetry breaking constraints in constraint programming. In: Kaibel, V., Liberti, L., Schürmann, A., Sotirov, R. (Eds.), Exploiting Symmetry in Optimization. Vol. 7 of Oberwolfach Reports. European Mathematical Society, Zürich, p. 2258.

Sturmfels, B., 2008. Algorithms in Invariant Theory, 2nd Edition. Springer, Wien.