Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

Orbital shrinking: Theory and applications

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

[a] *DEI, Università di Padova, Italy*
[b] *CNRS LIX, Ecole Polytechnique, 91128 Palaiseau, France*
[c] *NICTA and UNSW, Sydney, Australia*

A R T I C L E   I N F O

Article history:
Received 27 January 2016
Received in revised form 8 January 2017
Accepted 20 January 2017
Available online 21 February 2017

Keywords:
Mathematical programming
Constraint programming
Discrete optimization
Symmetry
Relaxation
MINLP

A B S T R A C T

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 showcase the effectiveness of our relaxation when embedded in 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.

1. Introduction

Branch-and-Bound (BB) type methods often become very slow when the solution set is symmetric [11,25], due to the exploration 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 keeping the objective function invariant and permuting the order of the constraints [3,23]. It is very easy to show that the formulation group is a subgroup of the solution group.

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 [2,19,20,37]. The most efficient methods reduce to the graph isomorphism problem, which can be solved in practice using tools such as `nauty` [27].

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 [20,21,46];
- using a clever branching strategy in constraint propagation [10] or in BB (e.g. isomorphism pruning [23,24] or orbital branching [30,31]).

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 [8].

* Corresponding author.
E-mail addresses: matteo.fischetti@unipd.it (M. Fischetti), liberti@lix.polytechnique.fr (L. Liberti), domenico.salvagnin@unipd.it (D. Salvagnin), toby.walsh@nicta.com.au (T. Walsh).

http://dx.doi.org/10.1016/j.dam.2017.01.015
0166-218X/© 2017 Elsevier B.V. All rights reserved.

A different approach is proposed in [2], 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 [14]. Although applicability remains limited, this technique was used in solving the instance `toll-like` in the MIPLIB2010 library [18], 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 analyze differences and similarities between OS and core point algorithms. In Section 4 we describe a general decomposition framework based on orbital shrinking, while in Sections 5 and 6 we specialize the general framework to multi-activity shift scheduling and multiple knapsack problems, also reporting computational results. Conclusions are finally drawn in Section 7.

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 [7,42,43], by the same authors.

1.1. Some notation and terminology

Let P be an arbitrary MINLP of the form

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

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

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

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. For a point $x' \in \mathbb{R}^n$ and a subset $V \subseteq [n]$, we let $x'[V]$ be the subsequence of x' indexed by V.

We consider the formulation group G_P of P, containing the set of permutations $\pi \in S_n$ (the symmetric group of order n, which acts naturally on the variable indices) 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) [1]. G_P is then obtained as a restriction of the automorphism group of this DAG to the set of variable indices of P [20]. G_P can be computed by means of any graph isomorphism package such as Nauty [28] or Saucy [17]: these both implement backtracking algorithms which are exponential-time in the worst case, but which are sufficiently fast in practice to be of use.

Any subgroup G of S_n 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 OSR with respect to a subgroup G of S_n 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 group G defining the OSR will be taken with respect to partitions (V_1, V_2) and (C_1, C_2, C_3) of the variables and constraints of P respectively, that satisfy the following conditions:

- all constraints in C_1 are convex w.r.t. the variables in V_1 (in other words, when the variables in V_2 are fixed);
- all constraints in C_2 have the property that all variables in V_1 can only appear through their orbit sums $\sum_{\ell\in\omega} x_\ell$;
- C_3 consists of all constraints not in C_1 or C_2.

We consider the relaxation P' of P obtained by removing constraints in C_3 from P. Let G be any subgroup of $G_{P'}[V_2]$, and $\Omega = \Omega_G$ be the orbital partition of $[n]$ induced by G. We can now define the OSR P_{OSR} as the MP formulation obtained by the following procedure:

1. for each $\omega \in \Omega$, introduce a variable z_ω, constrained to be general integer if and only if the orbit ω consists of integer variables; let $z = (z_\omega \mid \omega \in \Omega)$;
2. for each constraint $g_i(x) \le 0$ in $C_1 \cup C_2$, define a new constraint $\bar{g}_i(z) \le 0$, obtained from $g_i(x) \le 0$ through the formal substitution:
$$x_j \to \frac{z_\omega}{|\omega|} \tag{4}$$
for all $j \in \omega$;
3. define the objective function $\bar{f}(z)$ as the result of applying to $f(x)$ the same formal substitution (4);
4. drop constraints in C_3 from P_{OSR}.

Example 2.1. Consider the MIP formulation P:

$$\min_{x\in\{0,1\}^6} x_1 + 2x_2 + 3x_3 + x_4 + 2x_5 + 3x_6 \tag{5}$$

$$x_1 + x_2 + x_3 \ge 1 \tag{6}$$

$$x_4 + x_5 + x_6 \ge 1 \tag{7}$$

$$x_1 + x_4 \le 1 \tag{8}$$

$$x_2 + x_5 \le 1 \tag{9}$$

$$x_3 + x_6 \le 1. \tag{10}$$

It is easy to show that $G_P = \langle(1,4),(2,5),(3,6)\rangle$. The action of G_P on the variable index set $[n]$ yields the orbit set $\Omega = \{\omega_1,\omega_2,\omega_3\}$ with $\omega_1 = \{1,4\}$, $\omega_2 = \{2,5\}$, $\omega_3 = \{3,6\}$. We remark that the action induced by G_P on the constraint index set $\mathcal{C}(P) = \{(6),\ldots,(10)\}$ yields four orbits: $\{(6),(7)\}$ and the three trivial orbits $\{(8)\}$, $\{(9)\}$, $\{(10)\}$. Since all of the constraints are convex, we can derive an OSR using partitions $V_1 = [n]$, $C_1 = \mathcal{C}(P)$ and $V_2 = C_2 = C_3 = \varnothing$. The resulting P_{OSR} is:

$$\begin{aligned}\min_{z\in\{0,1,2\}^3}\ & z_1 + 2z_2 + 3z_3\\ & z_1 + z_2 + z_3 \ge 2\\ & z_1 + z_2 + z_3 \ge 2\\ & z_1 \le 1\\ & z_2 \le 1\\ & z_3 \le 1,\end{aligned}$$

which we can more conveniently write as

$$\begin{aligned}\min_{z\in\{0,1\}^3}\ & z_1 + 2z_2 + 3z_3\\ & z_1 + z_2 + z_3 \ge 2.\end{aligned}$$

Now suppose we add the constraint

$$(x_1 + x_4)(x_2 + x_5) = 0 \tag{11}$$

to P: since it is not convex in V_1, it cannot belong to C_1. On the other hand, it depends on sums of variables in each of their orbits, so it qualifies as a member of C_2, which is now equal to the singleton set $\{(11)\}$, and is reformulated in the OSR as

$$z_1 z_2 = 0. \tag{12}$$

Suppose now we add the constraint

$$2x_3 + 3x_6 \le 5 \tag{13}$$

to P. This breaks the symmetry $(3, 6)$, which implies that the formulation group G_P becomes the trivial group. We remark, however, that the formulation P' consisting of P without the constraints in Eqs. (6), (7), (10) and (13), has formulation group $G_{P'} = \langle(1, 4), (2, 5)\rangle$. We can therefore define $G = G_{P'}$ and take $V_1 = \{1, 2, 4, 5\}$, $V_2 = \{3, 6\}$, $C_1 = \{(8), (9)\}$, $C_2 = \{(11)\}$ and $C_3 = \{(6), (7), (10), (13)\}$. We obtain the OSR:

$$\min_{x,z\in\{0,1\}} z_1 + 2z_2 + 3x_3 + 3x_6 \tag{14}$$

$$z_1z_2 = 0. \tag{15}$$

We can, however, consider a different partition: $V_1 = [n]$, $V_2 = \varnothing$, $C_1 = \{(6), (7), (8), (9), (10)\}$, $C_2 = \{(11)\}$ and $C_3 = \{(13)\}$. This allows us to choose the larger group $G = \langle(1, 4), (2, 5), (3, 6)\rangle$, yielding the OSR

$$\begin{aligned}\min_{z\in\{0,1\}^3} &\; z_1 + 2z_2 + 3z_3\\ &z_1 + z_2 + z_3 \geq 2\\ &z_1z_2 = 0,\end{aligned}$$

which provides an OSR with five variables instead of six. $\square$

Next, we show that P_{OSR} is a relaxation of P. Lemma 2.2 is a basic generalization of Burnside's Lemma [41]: 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.

Lemma 2.2. *Let $\omega \in \Omega$ and ψ be any function with* dom $\psi = [n]$. *Then*

$$\sum_{\pi\in G} \psi(\pi(j)) = \frac{|G|}{|\omega|}\sum_{l\in\omega}\psi(l) \quad \forall j \in \omega. \tag{16}$$

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. $\square$

Corollary 2.3 is an application of Lemma 2.2 to the *barycenter* [2] of the group action, also called group average [8] or Reynolds operator [47].

Corollary 2.3. *Let x^* be an arbitrary feasible solution of P, and consider the convex combination $\bar{x}$ defined as*

$$\bar{x} = \frac{1}{|G|}\sum_{\pi\in G}\pi(x^*). \tag{17}$$

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

$$\bar{x}_j = \frac{1}{|\omega|}\sum_{l\in\omega} x^*_l. \tag{18}$$

Proof. Define a function $X : [n] \to \mathbb{R}$ as $X(j) = x^*_j$. Applying Lemma 2.2 we get:

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

Lemma 2.4 is well known, and basically states that the barycenter is an invariant of the group action [47]. We provide a proof for completeness, since it is very short.

Lemma 2.4. *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 are invariant to permutations in G. This proves the first equality. The second equality follows by definition of $\bar{x}$ (see Eq. (17)). $\square$

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 $\bar{g}(z) \le 0$.

Theorem 2.5. *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 $\bar{f}(z^*) \le f(x^*)$, hence the claim. Given x^*, let us construct the two points $\bar{x}$ and z^* as

$$\bar{x} = \frac{1}{|G|} \sum_{\pi \in G} \pi(x^*) \tag{19}$$

$$\forall \omega \in \Omega \qquad z^*_\omega = \sum_{j \in \omega} x^*_j. \tag{20}$$

For each constraint i in C_1, we claim $g_i(\bar{x}) \le 0$: note that $\bar{x}[V_2] = x^*[V_2]$ and that g_i is convex in each variable in V_1. By Corollary 2.3, Eq. (18) holds. Hence, for all variable indices but j fixed, we have

$$g_i(\bar{x}_j) = g_i\left(\frac{1}{|\omega|} \sum_{l \in \omega} x^*_l\right) \le \frac{1}{|\omega|} \sum_{l \in \omega} g_i(x^*_l) \le 0.$$

For each constraint i in C_2, we have $g_i(\bar{x}) = g_i(x^*) \le 0$ because of Lemma 2.4. 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 Corollary 2.3

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

since x^* itself is feasible for those constraints. Finally, since we drop the constraints in C_3 from P_{OSR}, we do not need to check their feasibility. As far as the objective function is concerned, we have $\bar{f}(z^*) = f(\bar{x}) \le f(x^*)$ where the equality is by definition of $\bar{f}(z)$ and Corollary 2.3, 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 2.6 essentially follows because the barycenter is a group invariant, and is at the basis of the ideas developed in [2,8,14].

Corollary 2.6. *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 2.7 shows the easiest case where P_{OSR} is an exact reformulation.

Corollary 2.7. *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

1. Corollary 2.7 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.
2. Since P' (P without the constraints in C_3) is a relaxation of P, G_P ends up being a subgroup of $G_{P'}$. This means that the OSR can be derived from a group G which is not actually a subgroup G_P, but might instead have G_p as a subgroup. The set C_3 of constraints is provided to give freedom in the choice of which variables to put into V_2.

3. We note that the partition (C_1, C_2, C_3) is 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}. Hence P_{OSR} has one variable for each variable orbit and one constraint for each constraint orbit in P associated with a constraint in $C_1 \cup C_2$.
4. 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 Example 2.8.
5. 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, they carry over to the OSR unchanged, aside from the replacement of orbital sums by the corresponding z variable.

Example 2.8. Let the feasible set of P be defined as

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

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

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

which is empty. $\square$

3. The role of the barycenter

In this section we discuss the relationships between OSR and other existing methods for exploiting symmetry in MP based on core points. All of these methods are based on the concept of the *barycenter* of the group action, defined in Eq. (17).

The papers [2,14] discuss a new approach to optimize symmetric MILP (or convex MINLP). The first paper [2] 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 in the second paper [2]: G_P is only assumed to have a direct product of (possibly trivial) symmetric groups as a subgroup.[1]

Let P be the ILP $\max\{c^\top x \mid Ax \le b \wedge x \in \mathbb{Z}^n\}$, and let $G \le G_P$ have a transitive action on $[n]$. The papers [2,14] 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 [2] that $\{H_c^k \mid k \in \mathbb{Z}\}$ is a partition of $\mathbb{Z}^n$ for each coprime c.

In [2], G is assumed to be transitive: by definition, this implies that for each $i \ne j \in [n]$ 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. (18)), 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 [2], 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 [2, Alg. A], which consists of finding an integer feasible point in each H_c^k, is hard. However, if G is μ-transitive (with $\mu \ge \lfloor\frac{n}{2}\rfloor + 1$), 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 [2, 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 [14], subsequent to [2], relaxes the transitivity requirements, generalizes the definition of a core point, and proposes two algorithms for solving symmetric MILPs and convex MINLPs. The first algorithm is a generalization of the core point algorithm of [2]: to address the fact that $c = \mathbf{1}$ is no longer the unique generator of the fixed subspace, [14, 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 $i \le d$, and $s_{ij} \in \{0, 1\}$ for each $i \le d$ and $j < k_i$; and new sets of constraints:

$$\forall i \le d \qquad x_{ij} = t_i \mathbf{1}_{k_i} + \sum_{j<k_i} s_{ij}\mathbf{c}_j$$

[1] We remark that the instance library data given in [20] 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.

$$\forall i \le d \quad \sum_{j<k_i} s_{ij} \le 1,$$

where $\mathbf{1}_{k_i}$ is the all-one vector of size k_i, $\mathbf{c}_j = \sum_{h\le j} e_h$ (with e_h the hth 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 [2] do not;
- the reformulation derived in the second core point algorithm in [14] is exact, and, in particular, different from the OSR.

The only similarity between the OSR and the results in [2] is that the fixed subspace LP reformulation [2, Eq. (3)] is the same as the OSR whenever there are no integer variables (see Corollary 2.6). The only similarity with [14] is that the barycenter is the point of departure to derive a reformulation of the original MIP.

4. 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 [15], 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 [22,32]. Note that the technique is general as a framework, not as a black-box algorithm that can be readily applied to an arbitrary instance, in the same way as logic-based Benders is: in both cases some problem-specific knowledge must be exploited for the method to succeed.

4.1. The slave feasibility problem

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

$$\forall i \in C \quad g_i(x) \le 0 \tag{21}$$

$$\forall \omega \in \Omega \quad \sum_{j\in\omega} x_j = z^*_\omega \tag{22}$$

$$\forall j \in J \quad x_j \in \mathbb{Z}. \tag{23}$$

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.

4.2. Symmetry of the slave problem

It is important to note that, by construction, problem $R(z^*)$ may also have a nontrivial formulation group.

Lemma 4.1. *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 $\pi \in G \le G_P$. Then π stabilizes the constraints (2) of P, which appear in $R(z^*)$ as (21). Since constraints (22) are generated by means of the action of G, π fixes each orbit of this action, and hence π also stabilizes (22). 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.

Example 4.2. 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 $G_R = G_P$.

Example 4.3. 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 4.1), 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.

Having a nontrivial symmetry group in $R(z^*)$ does not necessarily make the problem harder to solve, as it can be used to further simplify the problem or solving it more efficiently. In addition (i) linking constraints (22) may make the model much easier to solve, and (ii) the easier structure of $R(z^*)$ may allow for more effective symmetry breaking techniques. Still, it must be taken into account when devising an effective orbital shrinking decomposition.

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 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.

4.3. 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 (22) 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 [29] to solve a very challenging instance of 3-dimensional quadratic assignment problem. Interestingly, the role of MIP and CP was swapped in [29]: 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.

5. 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:

$$\min \sum_{a\in W}\sum_{t\in T} c_{at}y_{at} + \sum_{a\in W}\sum_{t\in t} c^+_{at}s^+_{at} + \sum_{a\in W}\sum_{t\in T} c^-_{at}s^-_{at} \tag{24}$$

$$\forall a \in W, t \in T \quad y_{at} - s^+_{at} + s^-_{at} = d_{at} \tag{25}$$

$$\forall a \in W, t \in T \quad \sum_{e\in E} x_{eat} = y_{at} \tag{26}$$

$$\forall e \in E \quad \langle x \text{ defines a feasible shift } \rangle \tag{27}$$

$$\forall a \in W, t \in T \quad y_{at}, s^+_{at}, s^-_{at} \in \mathbb{Z}^+ \tag{28}$$

$$\forall e \in E, a \in W, t \in T \quad x_{eat} \in \{0, 1\} \tag{29}$$

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 (24)–(29) 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}).

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 [4,34] 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 [4,33]. 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 [16] for the corresponding grammar [34,39].

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 [34], 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 [5]. 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 [5]. 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 [39]), still some cardinality constraints may be very awkward to express, as shown in the following example taken from [43].

Example 5.1.

Let us consider a time horizon of 18 h, divided into 18 periods. A feasible shift is a word of length 18 build from the alphabet $\Sigma = \{a, b, r\}$ (where a denotes the only working activity, r is a rest period, while b is a break period) that follows the pattern *rest–work–break–work–break–work–break–work–rest*. Suppose that the breaks are constrained to be one period long, and the number of working periods must be between 6 and 8. Then, a very simple grammar encoding the set of feasible shifts, ignoring the cardinality constraint, is:

$$S \to RFR \quad F \to PBP \quad P \to WBW$$

$$R \to Rr|r \quad W \to Wa|a \quad B \to b.$$

In this particular case, since the number of breaks in the shift is fixed (3), it is very easy to extend the grammar to deal with the cardinality constraint by restricting the production rule $F \to PBP$ to be applied only with substring of length between 9 and 11. This can be handled very well by the CYK parser, and thus the cardinality constraint can be added essentially at no cost.

However, let us consider a slightly more complicated case. The pattern of a feasible shift is the same, but now the length of breaks is *not* fixed to one. In particular, the number of break periods is constrained to be between 4 and 6. The best we can do keeping approximately the same grammar as before is the following (we use the notation of [5] to indicate restrictions on production rules):

$$S \to RFR \quad F_{[10,14]} \to PBP \quad P_{[3,10]} \to WBW$$

$$R \to Rr|r \quad W \to Wa|a \quad B \to Bb|b.$$

It is easy to see that the restrictions cannot be tightened any further, otherwise we may lose feasible shifts. However, the grammar also accepts the substring *rrababab bbbbbbaarr*, which violates both cardinality constraints.

The nature of the issues of the previous example is not exclusively theoretical: as the set of feasible shifts is finite, there always exists a regular/context-free language that describes that set. However, the corresponding automaton may be unreasonably large in practice. Note also that a very small change in the set of rules defining a feasible shift (like having a fixed or flexible amount of breaks, as in the example) can make the difference between being able to encode everything with a formal language or not.

In the following, we assume in the 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 4 in order to turn OSR into an exact procedure.

5.1. MIP model

The MIP model that we use is a simple modification of the general model (24)–(29). 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”.

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.

5.2. 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* [40]. 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 [43] 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.

5.3. Computational results

We tested our method on the multi-activity instances used in [4,5,38]. 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 [6] as black box MIP solver, and Gecode 3.7.3 [9] as CP solver. All tests have been performed on a PC with an Intel Core i5 CPU running at 2.66 GHz, with 8GB of RAM (only one core was used by each process). Every method was given a time limit of 1 h 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 [43] to each aggregated solution which is within 20%[2] 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 4. This means that we disable dual reductions (otherwise the decomposition would not be correct) and use CPLEX callbacks framework to implement the decomposition.

2 This threshold is used to avoid calling the repair/improve heuristic too many times.

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

# Act.	# Solved (10)			Time (s)		
	`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

Table 2
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

Table 1 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 [4], while `grammar` refers to the implicit model based on the grammar constraint in [5]. Note that for `grammar` we are reporting the results from [5], 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 1, `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 2 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 min. 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 turns out to be often faster than solving the LP relaxations of the original models, while providing better or equal dual bounds.

6. Application to the multiple knapsack problem

In the present section, we specialize the general framework of Section 4 to the multiple knapsack problem (MKP) [35,44]. This a natural generalization of the traditional knapsack problem [26], where multiple knapsacks 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

$$\max \sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij} \tag{30}$$

$$\forall i \in \{1, \ldots, m\} \quad \sum_{j=1}^{n} w_j x_{ij} \leq C_i \tag{31}$$

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

$$x \in \{0, 1\}^{m \times n} \tag{33}$$

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 we allow some items to also be identical, although this is not strictly required.

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

$$\max \sum_{k=1}^{K} p_k y_k \tag{34}$$

$$\sum_{k=1}^{K} w_k y_k \le mC \tag{35}$$

$$\forall k \in \{1, \dots, K\} \quad 0 \le y_k \le |V_k| \tag{36}$$

$$y \in \mathbb{Z}_+^K. \tag{37}$$

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 [45] and exploiting the CDBF [12] branching scheme for search and symmetry breaking. Given an aggregated solution y^*, we construct a vector s with the sizes of the items picked by y^*, and sort it in non-decreasing order. Then we introduce a vector of variables b, one for each item: the value of b_j is the index of the bin where item j is placed. Finally, we introduce a variable l_i for each bin, whose value is the load of bin i. The domain of variables l_i is $\{0, \dots, C\}$. With this choice of variables, the model reads:

$$\text{binpacking}(b, l, s) \tag{38}$$

$$b_{j-1} \le b_j \quad \text{if } s_{j-1} = s_j \tag{39}$$

where (39) are symmetry breaking constraints.

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

$$\sum_{p \in P} a_{pk} x_p = y_k^* \quad \forall k \tag{40}$$

$$\sum_{p \in P} x_p = m \tag{41}$$

$$x_p \in \mathbb{Z}_+ \tag{42}$$

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.

6.1. Computational experiments

We implemented our codes in C++, using the same hardware and software of Section 5.

In order to generate hard MKP instances, we followed the systematic study in [36]. More details about our instance generation procedure can be found in [42]. 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 (30)–(33), denoted as `cpxorig`. The other is obtained by aggregating the binary variables corresponding to identical items. The model, denoted as `cpx`, reads

$$\max \sum_{i=1}^{m} \sum_{k=1}^{K} p_j z_{ik} \tag{43}$$

Table 3
Comparison between `cpxorig` and `cpx`.

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

Table 4
Comparison between hybrid methods.

n	m	# Solved			Time (s)			Nodes		
		`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

$$\forall i \in \{1, \ldots, m\} \quad \sum_{k=1}^{K} w_j z_{ik} \leq C \tag{44}$$

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

$$z \in \mathbb{Z}_{+}^{m \times K} \tag{46}$$

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 3. As expected, `cpx` clearly outperforms `cpxorig`, solving 82 instances (out of 120) instead of 65. However, `cpx` performance is rapidly dropping as the number of items and knapsacks increases.

Then, we compared three variants of the hybrid MIP/CP procedure described in Section 6, that differs on the models used for the feasibility check. The first variant, denoted by `BPstd`, is based on the compact model (38)–(39). The second and the third variants are both based on the extended model (40)–(42), but differs on the solver used: a CP solver for `BPcgCP` and a MIP solver for `BPcgMIP`. All variants use model (34)–(37) 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 4. 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 (39) 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

Table 5
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

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 1000 nodes.

Finally, Table 5 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.

7. 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. 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".

References

[1] P. Belotti, J. Lee, L. Liberti, F. Margot, A. Wächter, Branching and bounds tightening techniques for non-convex MINLP, Optim. Methods Softw. 24 (4) (2009) 597–634.
[2] R. Bödi, K. Herr, M. Joswig, Algorithms for highly symmetric linear and integer programs, Math. Program. 137 (2013) 65–90.
[3] D. Cohen, P. Jeavons, C. Jefferson, K. Petrie, B. Smith, Symmetry definitions for constraint satisfaction problems, in: P. van Beek (Ed.), Constraint Programming, in: LNCS, vol. 3709, Springer, 2005, pp. 17–31.
[4] M.-C. Côté, B. Gendron, C.-G. Quimper, L.-M. Rousseau, Formal languages for integer programming modeling of shift scheduling problems, Constraints 16 (1) (2011) 54–76.
[5] M.-C. Côté, B. Gendron, L.-M. Rousseau, Grammar-based integer programming models for multiactivity shift scheduling, Manage. Sci. 57 (1) (2011) 151–163.
[6] CPLEX, CPLEX 12.4 User's Manual, IBM ILOG, 2012.
[7] M. Fischetti, L. Liberti, Orbital shrinking, in: Proceedings of ISCO, 2012, pp. 48–58.

[8] K. Gatermann, P. Parrilo, Symmetry groups, semidefinite programs and sums of squares, J. Pure Appl. Algebra 192 (2004) 95–128.
[9] Gecode Team, Gecode: Generic constraint development environment, 2012. Available at http://www.gecode.org.
[10] I. Gent, B. Smith, Symmetry breaking in constraint programming, in: W. Horn (Ed.), Proceedings of the 14th European Conference on Artificial Intelligence, in: ECAI, vol. 14, IOS Press, Amsterdam, 2000, pp. 599–603.
[11] I.P. Gent, K.E. Petrie, J.-F. Puget, Symmetry in constraint programming, in: F. Rossi, P. van Beek, T. Walsh (Eds.), Handbook of Constraint Programming, Elsevier, 2006, pp. 329–376.
[12] I.P. Gent, T. Walsh, From approximate to optimal solutions: Constructing pruning and propagation rules, in: IJCAI, Morgan Kaufmann, 1997, pp. 1396–1401.
[13] P.C. Gilmore, R.E. Gomory, A linear programming approach to the cutting-stock problem, Oper. Res. 9 (1961) 849–859.
[14] K. Herr, T. Rehn, A. Schürmann, Exploiting symmetry in integer convex optimization using core points, Oper. Res. Lett. 41 (2013) 298–304.
[15] J.N. Hooker, G. Ottosson, Logic-based benders decomposition, Math. Program. 96 (1) (2003) 33–60.
[16] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Languages and Computation, Addison Wesley, 1979.
[17] H. Katebi, K.A. Sakallah, 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, vol. 6175, 2010, pp. 113–127.
[18] T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R. Bixby, E. Danna, G. Gamrath, A. Gleixner, S. Heinz, A. Lodi, H. Mittelmann, T. Ralphs, D. Salvagnin, D. Steffy, K. Wolter, Miplib 2010, Math. Program. Comput. 3 (2) (2011) 103–163.
[19] L. Liberti, Automatic generation of symmetry-breaking constraints, in: B. Yang, D.-Z. Du, C. Wang (Eds.), Combinatorial Optimization, Constraints and Applications, COCOA08, in: LNCS, vol. 5165, Springer, Berlin, 2008, pp. 328–338.
[20] L. Liberti, Reformulations in mathematical programming: Automatic symmetry detection and exploitation, Math. Program. A 131 (2012) 273–304.
[21] L. Liberti, J. Ostrowski, Stabilizer-based symmetry breaking constraints for mathematical programs, J. Global Optim. 60 (2) (2014) 183–194.
[22] J. Linderoth, F. Margot, G. Thain, Improving bounds on the football pool problem by integer programming and high-throughput computing, INFORMS J. Comput. 21 (3) (2009) 445–457.
[23] F. Margot, Pruning by isomorphism in branch-and-cut, Math. Program. 94 (2002) 71–90.
[24] F. Margot, Exploiting orbits in symmetric ILP, Math. Program. B 98 (2003) 3–21.
[25] F. Margot, Symmetry in integer linear programming, in: M. Jünger, T. Liebling, D. Naddef, G. Nemhauser, W. Pulleyblank, G. Reinelt, G. Rinaldi, L. Wolsey (Eds.), 50 Years of Integer Programming 1958-2008, Springer Berlin Heidelberg, 2010, pp. 647–686.
[26] S. Martello, P. Toth, Knapsack Problems: Algorithms and Computer Implementations, Wiley, 1990.
[27] B. McKay, *nauty* User's Guide (Version 2.4), Computer Science Dept., Australian National University, 2007.
[28] B.D. McKay, Practical graph isomorphism, 1981.
[29] H.D. Mittelmann, D. Salvagnin, On solving a hard quadratic 3-dimensional assignment problem, Math. Program. Comput. 7 (2015) 219–234.
[30] J. Ostrowski, J. Linderoth, F. Rossi, S. Smriglio, Constraint orbital branching, in: A. Lodi, A. Panconesi, G. Rinaldi (Eds.), IPCO, in: LNCS, vol. 5035, Springer, 2008, pp. 225–239.
[31] J. Ostrowski, J. Linderoth, F. Rossi, S. Smriglio, Orbital branching, Math. Program. 126 (2011) 147–178.
[32] J. Ostrowski, J. Linderoth, F. Rossi, S. Smriglio, Solving large Steiner triple covering problems, Oper. Res. Lett. 39 (2) (2011) 127–131.
[33] 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, in: Lecture Notes in Computer Science, vol. 3258, Springer, 2004, pp. 482–495.
[34] G. Pesant, C.-G. Quimper, L.-M. Rousseau, M. Sellmann, The polytope of context-free grammar constraints, Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (2009) 223–232.
[35] D. Pisinger, An exact algorithm for large multiple knapsack problems, European J. Oper. Res. 114 (3) (1999) 528–541.
[36] D. Pisinger, Where are the hard knapsack problems? Comput. Oper. Res. 32 (2005) 2271–2284.
[37] J.-F. Puget, Automatic detection of variable and value symmetries, in: P. van Beek (Ed.), Constraint Programming, in: LNCS, vol. 3709, Springer, New York, 2005, pp. 475–489.
[38] C.-G. Quimper, L.-M. Rousseau, A large neighbourhood search approach to the multi-activity shift scheduling problem, J. Heuristics 16 (2010) 373–392.
[39] C.-G. Quimper, T. Walsh, Decomposing global grammar constraints, in: 13th International Conference on Principles and Practices of Const Raint Programming (CP-2007), Springer-Verlag, 2007, pp. 590–604.
[40] F. Rossi, P. van Beek, T. Walsh (Eds.), The Handbook of Constraint Programming, Elsevier, 2006.
[41] J.J. Rotman, Advanced Modern Algebra, Prentice Hall, 2002.
[42] D. Salvagnin, Orbital shrinking: A new tool for hybrid MIP/CP methods, in: CPAIOR, 2013, pp. 204–215.
[43] D. Salvagnin, T. Walsh, A hybrid MIP/CP approach for multi-activity shift scheduling, in: CP, 2012, pp. 633–646.
[44] A. Scholl, R. Klein, C. Jürgens, Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem, Comput. Oper. Res. 24 (7) (1997) 627–645.
[45] P. Shaw, A constraint for bin packing, in: M. Wallace (Ed.), CP, in: Lecture Notes in Computer Science, vol. 3258, Springer, ISBN: 3-540-23241-9, 2004, pp. 648–662.
[46] B. Smith, Symmetry breaking constraints in constraint programming, in: V. Kaibel, L. Liberti, A. Schürmann, R. Sotirov (Eds.), Exploiting Symmetry in Optimization, in: Oberwolfach Reports, vol. 7, European Mathematical Society, Zürich, 2010, p. 2258.
[47] B. Sturmfels, Algorithms in Invariant Theory, 2nd ed., Springer, Wien, 2008.