Symmetry detection in Mixed-Integer Conic Programming

Sven Wiese[1]

MOSEK ApS, Fruebjergvej 3, Symbion Science Park, 2100 Copenhagen, Denmark
sven.wiese@mosek.com

Abstract. We describe how to detect symmetry in Mixed-Integer Conic Programming. We present our framework as an extension to the Mixed-Integer Linear Programming case that results in a relatively small formal overhead. To do so, we introduce the concept of symmetry labelings for cones and study their properties, with a focus on what labelings are useful in practice. We also report on computational experiments enabling and disabling symmetry detection in the optimization software package MOSEK on a series of Mixed-Integer Conic Programming problems.

Keywords: Mixed-Integer Conic Programming, Symmetry

1 Introduction and literature review

Symmetry handling in Mathematical Programming may be broken down into two steps: detection and exploitation. The first step consists, very roughly speaking, in understanding how symmetric solutions (i.e., feasible solutions with the same cost) arise in a given model. Once such knowledge has been established, the goal is to use it in order to speed-up the solution process.

Handling symmetry has proven to be essential for some classes of Mixed-Integer Linear Programming (MILP) problems, but as well in general-purpose MILP solvers some techniques have been found to be useful. Symmetry handling techniques may be categorized as being either static or dynamic [34, 35]. Static techniques try to remove symmetry through better model representations, for instance by adding symmetry-breaking inequalities [44, 23, 26, 43], or by introducing an objective perturbation [12]. Dynamic methods address symmetry directly during the search method applied to a given model, such as Branch-and-Cut. Two prominent lines of research here are isomorphism pruning [28–31] and orbital branching [34, 36], both coming along with versions of so-called orbital fixing. [21, 20, 5, 8] study the concept of orbitopes for performing symmetry handling in the presence of set partitioning, packing or covering constraints. More approaches making use of additional structural knowledge on the underlying problem can be found in [18, 17]. [27, 35] provide more comprehensive surveys on the topic, and [38] a broad computational study comparing various techniques.

Symmetry handling is not confined to MILP. A technique called LP-folding [13] is applied to Linear Programming (LP) problems, and [11, 4, 37] treat symmetric Semidefinite Programming (SDP). For general convex programs, it is well-known how to "project out" symmetries by unifying variables along the same orbit, and the line of research centered around core-points [6, 14, 15, 40, 16] extends these ideas to integer programs. Orbital shrinking [9, 42] is based on similar ideas. As well for Mixed-Integer Nonlinear Programming (MINLP) problems it has been shown that symmetries arise, and symmetry handling has been extended to that case [25, 24]. This paper focuses on symmetry detection for Mixed-Integer Conic Programming (MICP), which may be seen as an extension to MILP, and we put particular emphasis on how the proposed detection procedure is an extension of the one applied in MILP.

The rest of this paper is organized as follows. In Section 2.1 we review how to perform the first symmetry handling step, i.e., the detection step, in MILP, and then show how to do the same in MICP. To that end, we introduce the concept of symmetry labelings for a given cone and therewith define what a formulation symmetry in MICP is in Section 2.2, and then describe the actual detection routine in Section 2.3. We will also give a characterization of symmetry labelings via orbits in Section 2.4, making the whole concept more tangible. We will not introduce new techniques for exploiting already detected symmetry in this paper, but mention how existing techniques from MI(N)LP carry over in Section 2.5. In the same section, we highlight the differences in symmetry detection for MICP and MINLP, respectively. In the recent MOSEK version 10 [33], symmetry handling has been added for both MILP and MICP, and in Section 3 we report on computational experiments assessing its value in the MICP case.

2 Symmetry handling in MICP

We consider Mixed-Integer Conic Programming problems in standard form:

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax = b \\ & x \in \mathcal{K} \cap (\mathbb{Z}^p \times \mathbb{R}^{n-p}) \, . \end{array} \tag{P}$$

Here, $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$, and $\mathcal{K}$ is a proper cone. Denoting the feasible set of (P) by $\mathcal{F}$, a symmetry of (P) is a bijection $s : \mathbb{R}^n \mapsto \mathbb{R}^n$ such that $s(\mathcal{F}) = \mathcal{F}$ and $c^T s(x) = c^T x \ \forall x \in \mathcal{F}$. Since establishing the set of all such bijections, i.e., the symmetry group of (P), is hard both in theory and practice, we take the usual route of resorting to so-called formulation symmetries, a subset of the whole symmetry group.

A formulation symmetry is a permutation of the variable space that leaves the specific description of the feasible set invariant. To that end, let $\mathcal{S}_n$ denote the symmetric group on $[n] := \{1, \dots, n\}$, i.e. the group of all permutations of n elements. Here and in all cases of permutation groups to follow, we assume the group operation to be the composition, i.e., the product $\pi\sigma$ is such that $(\pi\sigma)(i) = (\pi \circ \sigma)(i) = \pi(\sigma(i)) \ \forall i \in [n]$. In order to cast a permutation of $[n]$ as a symmetry of (P), by a slight abuse of notation we let $\pi \in \mathcal{S}_n$ act on $x \in \mathbb{R}^n$ by permuting its components, $\pi(x) = (x_{\pi^{-1}(1)}, \dots, x_{\pi^{-1}(n)})^T$.

Below we will also mention symmetry generators, or generators for short. For any set of permutations $S \subseteq \mathcal{S}_n$, we denote by $\langle S \rangle$ the smallest group containing S. A generating set for some group $G \subseteq \mathcal{S}_n$ is a set $S \subseteq \mathcal{S}_n$ such that $\langle S \rangle = G$, and its members are referred to as generators.

2.1 Detecting formulation symmetries: the linear case

Note that MILP is the special case of (P) where $\mathcal{K} = \mathbb{R}^n_+$, the non-negative orthant. In that case, we have the following definition.

Definition 1 (see, e.g., [38]) *If $\mathcal{K} = \mathbb{R}^n_+$, then $\pi \in \mathcal{S}_n$ is a formulation symmetry of* (P) *iff $\exists\ \sigma \in \mathcal{S}_m$ such that*

$$\pi([p]) = [p] \tag{1}$$

$$\pi(c) = c \tag{2}$$

$$\sigma(b) = b \tag{3}$$

$$A_{\sigma(i),\pi(j)} = A_{ij}. \tag{4}$$

(1) assures that integer variables are permuted with integer variables only, and (2) guarantees that the cost of a permuted solution is not changed. (3)-(4) mean that for every linear constraint, there is another constraint that imposes the same restriction after the variables have been permuted. I.e., the permuted variables have the same coefficients in that constraint as the non-permuted variables have in the original one, and also the right-hand sides are equal. It is thus straightforward to see that in the MILP case, for any formulation symmetry π, $x \in \mathcal{F}$ implies $\pi(x) \in \mathcal{F}$.

The problem of finding formulation symmetries in MILP is usually reduced to the graph automorphism problem, see [41, 39]. From an instance of (P) with $\mathcal{K} = \mathbb{R}^n_+$ we build a colored bipartite graph $(V \cup W, E)$, where the set $V = \{v_1, \ldots, v_n\}$ contains a vertex for each variable x_j, colored with its objective value c_j, and $W = \{w_1, \ldots, w_m\}$ contains one vertex for each constraint, colored with the right-hand side b_i. Also, integer variables, $j \in [p]$, get different colors than continuous variables. We further add an edge $e = \{v_j, w_i\}$ if variable x_j appears in constraint i with a non-zero coefficient a_{ij}, the latter determining the edge color. A graph automorphism is given by a tuple of bijections (π, σ), mapping from V to itself and from W to itself, respectively, such that $\{\pi(v_j), \sigma(w_i)\} \in E$ iff $\{v_j, w_i\} \in E$. One can check that any graph automorphism that keeps vertex and edge colors invariant gives rise to a formulation symmetry, and vice-versa.

Examples of software packages that compute graph automorphisms are Nauty [32], Saucy [7] or Bliss [19]. The algorithms implemented in these software packages are not designed for accounting for edge colors though, so that we need to accommodate this requirement in some other way. One method is to extend the above graph construction to the so-called matrix graph, that we assume to be the method of choice throughout this paper. Therefore, each edge $\{v, w\}$ is replaced with two edges $\{v, a\}$, $\{a, w\}$, introducing an additional vertex a receiving the color of the edge it represents. The number of so introduced additional

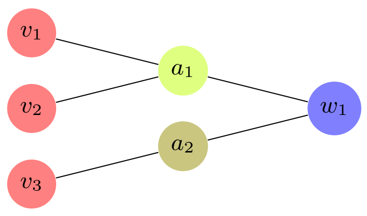

Fig. 1. Matrix graph for a linear constraint

vertices may furthermore be reduced significantly by applying so-called grouping by variables or grouping by constraints, see the following example. We refer to [38] for more details on the matrix and related graph constructions.

Example 1. Consider the constraint $2x_1 + 2x_2 + 3x_3 \leq 4$, and assume for simplicity that all variables have the same objective coefficient and are of the same type. The matrix graph for this (sub-)program would look like in Figure 1. Note that grouping by variables is applied, i.e., we introduce only one intermediate vertex of "color 2", covering both the coefficients of x_1 and x_2 in this constraint.

Graph automorphism algorithms usually return a set $\tilde{S}$ of symmetry generators (π, σ), and the symmetry group to work with is thus $\mathcal{G} = \langle S \rangle$ with $S = \{\pi \mid (\pi, \sigma) \in \tilde{S}\}$.

2.2 The conic case: a cone's symmetry labelings

We will now extend the concepts of the previous section to the conic case, starting with the one essential definition we need.

Definition 2 *We call a function $h : [n] \mapsto \mathbb{N}$ a symmetry labeling w.r.t. a cone $\mathcal{K} \subseteq \mathbb{R}^n$, iff for any $\pi \in \mathcal{S}_n$ the condition $h(\pi(i)) = h(i)\ \forall i \in [n]$ implies $\pi(\mathcal{K}) = \mathcal{K}$.*

Finding a symmetry labeling, in the following just labeling for short, to a given cone just means to assign some value to each variable in such a way that permuting variables with the same value does not affect cone membership. A trivial labeling for every cone is the identity map. We now give some examples of more interesting labelings in the case of commonly used cones.

- $\mathcal{K} = \mathbb{R}^n_+$, the non-negative orthant. Cone membership in $\mathbb{R}^n$ is preserved for arbitrary permutations, meaning that a labeling is any constant function, $h(i) = c\ \forall i \in [n]$.
- $\mathcal{K} = \mathcal{Q}^n = \{x \in \mathbb{R}^n \mid x_1 \geq \|x_{2:n}\|_2\}$, the quadratic cone. A labeling is given by

$$h(i) = \begin{cases} 1, & i = 1 \\ 2, & \textit{otherwise}. \end{cases}$$

  In fact, for any permutation π having 1 as a fixed-point (i.e., $\pi(1) = 1$), clearly $\pi(x) \in \mathcal{Q}^n$ for any $x \in \mathcal{Q}^n$.

- $\mathcal{K} = \mathcal{Q}_r^n = \{x \in \mathbb{R}^n \mid 2x_1x_2 \geq \|x_{3:n}\|_2^2, x_1, x_2 \geq 0\}$, the rotated quadratic cone. A labeling is given by
$$h(i) = \begin{cases} 1, & i \leq 2 \\ 2, & \textit{otherwise.} \end{cases}$$
In fact, permuting either the first two variables, or permuting any subset of the remaining variables does not change cone membership.
- $\mathcal{K} = \mathcal{K}_{exp} = \mathrm{cl}\{x \in \mathbb{R}^3 \mid x_1 \geq x_2 \exp(x_3/x_2), x_2 > 0\}$, the exponential cone. Without any further constraints on x, there is no way to permute variables inside $\mathcal{K}_{exp}$. The only labeling is thus given by the identity.
- $\mathcal{K} = \mathcal{P}_\alpha^n = \{x \in \mathbb{R}^n \mid \prod_{k=1}^{\tilde{n}} x_k^{\alpha_k} \geq \|x_{\tilde{n}+1:n}\|_2, x_1, \dots, x_{\tilde{n}} \geq 0\}$, the power cone with parameter vector $\alpha \in \mathbb{R}^{\tilde{n}}$, $\tilde{n} < n$. In this form $\sum_k \alpha_k = 1$ is required. A labeling is given by
$$h(i) = \begin{cases} i, & i \leq \tilde{n} \\ \tilde{n}+1, & \textit{otherwise.} \end{cases}$$
This is similar to the rotated quadratic cone case.

All the above are examples of primitive cones, i.e., cones that cannot be written as the Cartesian product of two or more lower-dimensional cones. In practice, the cone $\mathcal{K}$ in (P) is often the Cartesian product of primitive cones, and we will now use the concept of labelings to extend the definition of a formulation symmetry to that case.

Definition 3 *Let $\mathcal{K} = \mathcal{K}_1 \times \dots, \times \mathcal{K}_K$ where $\mathcal{K}_k \subseteq \mathbb{R}^{n_k}$, $k \in [K]$, are proper cones, and denote by $N_k = \sum_{l<k} n_l$ the number of variables before $\mathcal{K}_k$, with $N_{K+1} = n$. Then $\pi \in \mathcal{S}_n$ is a formulation symmetry of* (P) *iff $\exists\ \sigma \in \mathcal{S}_m$ such that* (1) - (4)*, and $\exists\ \tau \in \mathcal{S}_K$ and labelings $h_k : [n_k] \mapsto \mathbb{N}$ w.r.t. $\mathcal{K}_k$ such that $\forall k \in [K]$*

$$\mathcal{K}_{\tau(k)} = \mathcal{K}_k \tag{5}$$
$$\pi(\{N_k + 1, \dots, N_{k+1}\}) = \{N_{\tau(k)} + 1, \dots, N_{\tau(k)+1}\} \tag{6}$$
$$h_{\tau(k)}(\pi(i) - N_{\tau(k)}) = h_k(i - N_k)\ \forall i \in \{N_k + 1, \dots, N_{k+1}\}. \tag{7}$$

One can check that a formulation symmetry preserves feasibility also in the MICP case. The additional conditions (5) - (7), similar to the case of a linear constraint, require that for any conic constraint $x' \in \mathcal{K}_k$, where x' is a n_k-dimensional component of x, there is another conic constraint that imposes the same restriction on the permuted variables. (5) corresponds to (3), and (6) - (7) together to (4). (6) states that the variables in the "support" of the original cone are jointly mapped to the new one. In the case of a linear constraint, an analogue condition is implicitly imposed also by (4). (7) then requires that all variables have the same symmetry labeling in the new cone as in the original one. In that light, the analogue of a labeling for a single linear constraint could

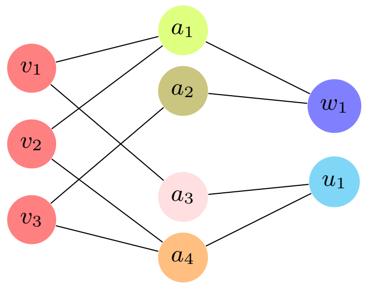

Fig. 2. Matrix graph for linear and conic constraints

be thought of as a function that assigns to each variable its coefficient in that constraint.

As mentioned above, when $K = 1$ and $\mathcal{K} = \mathbb{R}^n_+$ we get the MILP case, so the additional conditions in Definition 3 should not impose any restriction. In fact, (5) - (6) are trivially redundant, and taking into account that a labeling in that case is any constant function, also (7) becomes redundant.

2.3 Extending the matrix graph

Definition 3 can be used to extend the matrix graph construction described in Section 2.1 to the conic case. We introduce a third set of vertices $U = \{u_1, \ldots, u_K\}$, containing one vertex for each cone $\mathcal{K}_k$, colored with the cone type (i.e., quadratic, rotated quadratic, exponential, ...). If x_j appears in K_k, we add an edge $\{v_j, u_k\}$ with color $h(j - N_k)$ for some labeling w.r.t. $\mathcal{K}_k$. Needless to say, for same cone types we use the same labelings. A formulation symmetry is then given by a triple (π, σ, τ) such that in addition to what is required in the linear case, $\{\pi(v_j), \tau(u_k)\} \in E$ iff $\{v_j, u_k\} \in E$. Also here, before applying any algorithm for detecting graph automorphisms, we introduce additional vertexes in order to account for edge colors. Again, we can reduce the number of additional vertexes by applying grouping by variables.

Example 2. Assume that in addition to the setting from Example 1, we also have the conic constraint $(x_1, x_2, x_3) \in \mathcal{Q}^3$. The matrix graph would now look like in Figure 2. Note that grouping by variables is applied: there is only one additional vertex marking the labels of both x_2 and x_3 in the conic constraint.

2.4 Orbit characterization of symmetry labelings

We will now give an alternative characterization of a cone's labelings. While the concepts of the previous section seem more suitable to work with in practice, this section's purpose is to better capture the idea behind labelings, and how a certain such labeling might be preferable to another one. We first need some notation regarding permutations and permutation groups.

For any subgroup $\mathcal{G} \subseteq \mathcal{S}_n$, the orbit of $i \in [n]$ under $\mathcal{G}$ is

$$\operatorname{orb}(\mathcal{G}, i) = \{j \in [n] \mid \exists\ \pi \in \mathcal{G} : j = \pi(i)\}.$$

In other words, the orbit of i is the set of all elements to which i can be mapped by the application of permutations in $\mathcal{G}$. It is a well-known fact that the union of orbits,

$$\mathcal{O}^{\mathcal{G}} := \bigcup_{j \in [n]} \operatorname{orb}(\mathcal{G}, j),$$

forms a partition of $[n]$, the so-called orbital partition.

We further define the orbital partition of a single permutation through the group it generates, i.e., $\mathcal{O}^{\pi} := \mathcal{O}^{\langle\{\pi\}\rangle}$.

Also, for any function $f : X \mapsto Y$, we denote the level sets by

$$L_y(f) = \{x \in X \mid f(x) = y\}.$$

When X is finite, so is $f(X)$ and we can partition X into a finite collection of level sets, denoted by

$$\mathcal{P}^f = \bigcup_{y \in f(X)} L_y(f).$$

Finally, for two partitions $\mathcal{V}, \mathcal{W}$ of the ground set $[n]$, we write $\mathcal{V} \leq \mathcal{W}$ if $\mathcal{V}$ is finer than $\mathcal{W}$, i.e. $\forall V \in \mathcal{V}\ \exists\ W \in \mathcal{W} : V \subseteq W$. $\mathcal{W}$ instead is then called coarser than $\mathcal{V}$. If $\mathcal{V} \leq \mathcal{W}$ and $\mathcal{W} \leq \mathcal{V}$, the two partitions are equal, and we write $\mathcal{V} = \mathcal{W}$.

With all that in mind, we can give an alternative characterization of a labeling, the proof of which is immediate.

Lemma 1. *$h : [n] \mapsto \mathbb{N}$ is a symmetry labeling w.r.t. $\mathcal{K} \subseteq \mathbb{R}^n$, iff for any $\pi \in \mathcal{S}_n$ the condition $\mathcal{O}^{\pi} \leq \mathcal{P}^h$ implies $\pi(\mathcal{K}) = \mathcal{K}$.*

Proof. We show the equivalence $\mathcal{O}^{\pi} \leq \mathcal{P}^h \iff h(\pi(i)) = h(i)\ \forall i \in [n]$ for any $\pi \in \mathcal{S}_n$. First let $\mathcal{O}^{\pi} \leq \mathcal{P}^h$ and $i \in [n]$. By definition of the orbital partition, there is some $o \in \mathcal{O}^{\pi}$ with $i, \pi(i) \in o$. And since $\mathcal{O}^{\pi}$ is finer than $\mathcal{P}^h$, there is some level set $L_y(h) \supseteq o$, and thus $h(\pi(i)) = h(i)$.

The other way round, let $o \in \mathcal{O}$ and fix some $i_0 \in o$. For any other $i \in o$, $i = \pi^r(i_0)$ for some $r \in \mathbb{N}_0$. Now by assumption $h(i) = h(\pi^r(i_0)) = h(i_0)$, and therefore there is again some level set $L_y(h) \supseteq o$. We conclude $\mathcal{O}^{\pi} \leq \mathcal{P}^h$.

This view on labelings tells us that finding one means partitioning the variables in the cone in such a way that permuting inside a partition cell does not affect cone membership. Note that if h is a labeling w.r.t. $\mathcal{K}$ and $h' : [n] \mapsto \mathbb{N}$ is such that $\mathcal{P}^{h'} \leq \mathcal{P}^h$, then also h' is a labeling. In practice it is desirable to work with coarse orbit-preserving functions, implicitly referring to their underlying partitions $\mathcal{P}^h$, and we believe that thinking in terms of orbits helps to better capture this fact. It is also underlined by the following corollary, a straightforward application of Lemma 1 to Definition 3. Note that for ease of exposition, it is stated for the special case $K = 1$, i.e., without assuming that $\mathcal{K}$ can be written as a Cartesian product of primitive cones.

Corollary 1. *Let $\mathcal{K} \subseteq \mathbb{R}^n$ be a proper cone. Then $\pi \in \mathcal{S}_n$ is a formulation symmetry of* (P) *iff $\exists\ \sigma \in \mathcal{S}_m$ such that* (1) - (4), *and $\exists$ a labeling $h : [n] \mapsto \mathbb{N}$ w.r.t. $\mathcal{K}$ such that $\mathcal{O}^\pi \leq \mathcal{P}^h$.*

Lemma 1 could as well be applied to the general setting of Definition 3 to get an analogue but more general result in the product setting, but notation would be more cumbersome. In either case, Corollary 1 highlights that a coarser labeling of the cone $\mathcal{K}$ (coarser labelings of the individual primitive cones in $\mathcal{K}_1 \times \ldots, \times \mathcal{K}_K$) can in principle only lead to the discovery of more formulation symmetries, and is thus more desirable when performing symmetry detection. As an example take the power cone labeling from Section 2.2. If α has some special structure, it might be possible to find coarser labelings. E.g., if $\alpha_{k_1} = \alpha_{k_2}$ for some $k_1, k_2 \leq \tilde{n}$, then x_{k_1} and x_{k_2} may be permuted without affecting cone membership, and we may chose h such that $h(k_1) = h(k_2)$.

Neglecting the product representation of $\mathcal{K}$ for illustration purposes as above is fine, but we now argue that it is in fact crucial to exploit such structural knowledge, if available. Otherwise, it may simply not be easy to specify a good labeling, as the following example shows.

Example 3. Let $\mathcal{K} = \mathcal{Q}^3 \times \mathcal{Q}^3 \subset \mathbb{R}^6$, i.e., $K = 2$, and consider the permutation $\pi_1 = (1,4)(2,5)(3,6)$. Note that here we use cycle notation, i.e., $(i_1, \ldots, i_T)$ is a permutation such that i_2 is the image of i_1, i_3 is the image of i_2 and so forth, and i_1 is the image of i_T. π_1 is the composition of 3 cycles, each of length 2. One can verify that π_1 is a formulation symmetry as per Definition 3. In fact, with $\tau = (1,2) \in \mathcal{S}_2$ and the labeling proposed for Q^n in Section 2.2, (5) - (7) are satisfied. Now it is easy to see that $\mathcal{O}^{\pi_1} = \{\{1,4\},\{2,5\},\{3,6\}\}$, so any labeling h w.r.t $\mathcal{K}$ to capture the valid formulation symmetry π_1 would have to be such that $\mathcal{O}^{\pi_1} \leq \mathcal{P}^h$. However, such a labeling cannot exist, since then also $\mathcal{O}^{\pi_2} \leq \mathcal{P}^h$ for $\pi_2 = (1,4)$. But the point $x = (1,0,0,2,1,1) \in \mathcal{K}$ is such that $\pi_2(x) \notin \mathcal{K}$, disqualifying h as a labeling as per Lemma 1.

The gist of this example is that a labeling cannot capture the variable permutation π and the conic constraint permutation τ at the same time. However, if we exploit a cone's product representation and perform symmetry detection with the matrix graph based on Definition 3, as is described in Section 2.3, we don't even need that.

2.5 Exploiting symmetry and relation to MINLP

Symmetry detection is only the first step of symmetry handling. Once the symmetry group $\mathcal{G}$ is established, various techniques exist to exploit that knowledge in order to speed-up the solution process, as highlighted in Section 1. Most of these techniques have been introduced and studied in the context of MILP. However, one may realize that several of them are solely based on the symmetry group $\mathcal{G}$ defined over the variable space. For example, arguments based on choosing one representative from a set of symmetric solutions exclusively exploit knowledge about $\mathcal{G}$. Or arguments based on the so-called fundamental domain

[10], originally defined not only for polyhedra, but for arbitrary sets. So it is to some extent intuitive why certain techniques would remain valid when going from linear constraints only over to linear and conic constraints. Changing the constraint structure definitely affects the detection step in symmetry handling, but not necessarily the exploitation step. This should be true for, e.g., the addition of symmetry breaking inequalities, isomorphism pruning, and orbital branching and fixing. Care has to be taken instead when a symmetry handling technique explicitly takes into account the constraint structure. One example in this regard is the aggregation of symmetric integer variables [3].

Several of the commonly encountered cones in MICP can be defined through the epigraph of a (convex) function, and in that case the respective problems may be thought of as (convex) MINLP problems. However, this is not true for all cones. And even if it is, the symmetry detection procedure we present in this paper makes use of the representation of a problem in the conic framework. Also [24] uses a graph automorphism algorithm in the MINLP setting, but applied to the expression graph of the underlying problem. Contrary to the matrix graph as in Sections 2.1 and 2.3, expression graphs contain nodes for operators like "+" or "−", in addition to nodes for variables, constraints and coefficients. Although the reformulation of an MINLP into conic form, if possible, may require the introduction of auxiliary constraints, expression graphs are thus expected to have a tendency to be larger than the matrix graph available for MICP. It is sometimes argued that one nice feature of MICP as in (P) is the separation of data and structure. The data of an instance (c, A, b) is contained entirely in the linear part of the problem, while the (non-linear) structure is specified in the conic constraint(s) $x \in \mathcal{K}$. All this comes in handy in our context: the formal overhead introduced when extending symmetry detection from MILP to MICP as in Definition 3 is relatively small, in particular, it is completely independent of the instance data. Not having to deal with data makes the construction of the part of the matrix graph that represents the conic structure relatively easy.

3 Computation

Symmetry handling has been introduced in MOSEK version 10 and is applied to both MILP and MICP problems. MOSEK detects symmetry as described in Section 2, and mainly applies a mixture of certain versions of orbital branching and fixing, and symmetric variable aggregation, see the remarks in Section 2.5. Without going into more detail here, we analyze the effect of symmetry handling and thus show that symmetry does occur in MICP, and what the value in detecting it is. In the following we report on computational results comparing the default version 10.0.15 (row `default` in Tables 1 - 2) against setting the parameter `MSK_IPAR_MIO_SYMMETRY_LEVEL` to 0 (row `sym-0`). The latter completely deactivates any symmetry handling.

All runs were performed in a single thread on isolated nodes of a cluster, each equipped with 32 GB of RAM and 4 cores running at 3.30 GHz. Below we report shifted geometric means of the time to optimality, with a shift of

10 secs. The time limit was set to 7200 secs., and means are computed over all instances that could be solved by at least one setting, counting timed-out instances in a conservative fashion with a value equal to the time limit. We also report how many instances were solved and how often one algorithmic setting was the winner, in the sense that running time was better by at least 5%. We finally report the shifted geometric mean of the explored nodes, with a shift of 10 nodes, and taking into account only those instances that were solved by both configurations.

3.1 Disk covering

The first set of instances arises from the geometric setting in which we have n points $p_1, \dots, p_n \in \mathbb{R}^d$ and want to find a configuration of k balls with centers $c_1, \dots, c_k \in \mathbb{R}^d$ and radii $r_1, \dots, r_k$, respectively, that covers some or all of the points p_i and satisfies some additional constraints. See [1] and references therein for some specific variants and more material. The two variants we considered for the following set of experiments are

- Smallest area k-circle cover: cover all points with k circles of variable diameter, minimizing their compound area
- Maximum k-coverage: cover as many points as possible with k fixed-radius circles

In both variants we have an underlying basic model with variables $x_{ij} \in \{0,1\}$, $i \in [n]$, $j \in [k]$, stating whether point i is covered by ball j or not. We further have variables $c_j \in \mathbb{R}^d$ and $r_j \geq 0$ for the ball centers and radii, respectively, and the big-M constraints $r_i + M(1 - x_{ij}) \geq \|c_j - p_i\|_2$, or equivalently in conic form

$$(r_i + M(1 - x_{ij}), c_j - p_i) \in \mathcal{Q}^{d+1}. \tag{8}$$

Note that a safe value for M can always be deduced from the dispersion of the given points p_i, and that (8) can be brought to the standard form in (P) by introducing linear auxiliary constraints.

- In the smallest-area variant we have an additional conic constraint arising from the minimization of a sum of squares, and require that each point is covered by at least some ball, $\sum_j x_{ij} \geq 1 \ \forall i$. Symmetry arises since interchanging the roles of balls gives rise to symmetric solutions.
- In the maximum-coverage variant, we fix the radii r_j to desired values and count the number of covered points with the additional variables $t_i \in \{0,1\}$ and the constraint $t_i \leq \sum_j x_{ij}$, maximizing the sum of the t_i. If the radii of different balls are fixed to the same value, their roles are again interchangeable and we can detect symmetry in the model. Below we always fixed the radii of all involved balls to the same value.

We created instances for both variants by randomly sampling n points in $[0,1] \times [0,0.3]$, either uniformly or following a Gaussian distribution, with $n \in$

	solved	wins	mean time ($n = 80$)	mean nodes ($n = 66$)
`default`	80	62	65.54	2895
`sym-0`	66	2	167.47	6833

Table 1. Computational results for disk covering instances

$\{10, 20, 30, 40, 50\}$, $k \in \{3, 4, 5\}$, and $r \in \{0.1, 0.2\}$ in case of the maximum-coverage variant. This gives a total of 90 instances. The number of symmetry generators found in a single instance after presolve ranges from 1 to 4. The computational results are shown in Table 1. The advantage of enabling symmetry handling is quite clear here, leading to more solved instances and a reduction in running times and explored nodes by more than a factor of 2.

3.2 F-SPARC

The second set of instances are conic reformulations of the fractional subcarrier and power allocation problem with rate constraints (F-SPARC) arising in OFDMA systems [22]. Roughly speaking, we want to maximize energy efficiency in a data transmission system with n communication channels and k users. We have to assign to each user a number of communication channels so as to assure the user a certain data rate d_j, giving rise to variables x_{ij}, $i \in [n]$, $j \in [k]$, indicating the assignment. In addition, if channel i is assigned to user j, we have to determine the power p_{ij} assigned to the channel when used by user j (i.e., $x_{ij} = 0 \implies p_{ij} = 0$). The data rate requirement for user j can be written as $B \sum_i \log_2(1 + p_{ij}/N) \geq d_j$, where B and N denote the channel's bandwidth and noise level, respectively. Albeit the presence of some more side constraints on, e.g., total power consumption, interchanging the roles of communication channels again gives rise to symmetric solutions, and thus symmetry can be found in these models. More details and how to obtain a conic reformulation using the exponential cone can be found in [2].

We created instances just as described in [22], with

- $(n, k) \in \{(8, 4), (10, 5), (10, 6), (12, 7), (15, 7)\}$ and
- the so-called demand ratio $DR \in \{0.7, 0.8, 0.9\}$, used for randomly sampling user demands.

For each of the resulting 15 combinations we have 5 such random samplings, leading to 75 instances in total. Note that [22] allows for channels with distinct noise levels N_i. Symmetry arises only when at least two channels are identical, which is why we kept N_i fixed over all channels. The number of found symmetry generators after presolve is always $n - 1$. The computational results are summarized in Table 2. The advantage of symmetry handling is even clearer here than in the previous section, leading to twice as many solved instances and savings in the range of 1 -2 orders of magnitude.

	solved	wins	mean time ($n = 62$)	mean nodes ($n = 31$)
`default`	62	62	28.54	352
`sym-0`	31	0	1262.33	28722

Table 2. Computational results for F-SPARC instances

Conclusions

We have seen how symmetry can be detected in MICP, and that the detection procedure is quite similar to the one applied in MILP, making use of how close MICP is to MILP as a problem class. The concept of a cone's symmetry labelings is easily applicable in practice, especially when a product representation through primitive cones is used. We therefore outlined how to get an intuitive grasp on what favorable labelings are by looking at them from different points of view. We further listed two MICP models in which symmetry can be detected, and showed how dramatic the effect of symmetry handling on the solution times in MOSEK can be, i.e., even up to orders of magnitude.

References

1. The MOSEK Notebook Collection: Geometric facility location. https://nbviewer.org/github/MOSEK/Tutorials/blob/master/facility-location/small_disks.ipynb, accessed August-12-2022
2. The MOSEK Notebook Collection: Subcarrier and power allocation. https://nbviewer.org/github/MOSEK/Tutorials/blob/master/f-sparc/fsparc.ipynb, accessed August-12-2022
3. Achterberg, T., Bixby, R.E., Gu, Z., Rothberg, E., Weninger, D.: Presolve reductions in mixed integer programming. INFORMS Journal on Computing **32**(2), 473–506 (2020)
4. Bachoc, C., Gijswijt, D.C., Schrijver, A., Vallentin, F.: Invariant semidefinite programs. In: Anjos, M.F., Lasserre, J.B. (eds.) Handbook on Semidefinite, Conic and Polynomial Optimization, pp. 219–269. Springer US, Boston, MA (2012)
5. Berthold, T., Pfetsch, M.: Detecting orbitopal symmetries. In: Fleischmann, B., Borgwardt, K.H., Klein, R., Tuma, A. (eds.) Operations Research Proceedings 2008. pp. 433–438. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
6. Bödi, R., Herr, K., Joswig, M.: Algorithms for highly symmetric linear and integer programs. Mathematical Programming **137**, 65–90 (2013)
7. Darga, P.T., Liffiton, M.H., Sakallah, K.A., Markov, I.L.: Exploiting structure in symmetry detection for cnf. In: Proceedings of the 41st Annual Design Automation Conference. pp. 530–534 (2004)
8. Faenza, Y., Kaibel, V.: Extended formulations for packing and partitioning orbitopes. Mathematics of Operations Research **34**(3), 686–697 (2009)
9. Fischetti, M., Liberti, L.: Orbital shrinking. In: Mahjoub, A.R., Markakis, V., Milis, I., Paschos, V.T. (eds.) Combinatorial Optimization. pp. 48–58. Springer Berlin Heidelberg (2012)

10. Friedman, E.J.: Fundamental domains for integer programs with symmetries. In: Dress, A., Xu, Y., Zhu, B. (eds.) Combinatorial Optimization and Applications. pp. 146–153. Springer Berlin Heidelberg (2007)
11. Gatermann, K., Parrilo, P.A.: Symmetry groups, semidefinite programs, and sums of squares. Journal of Pure and Applied Algebra **192**(1), 95–128 (2004)
12. Ghoniem, A., Sherali, H.D.: Defeating symmetry in combinatorial optimization via objective perturbations and hierarchical constraints. IIE Transactions **43**(8), 575–588 (2011)
13. Grohe, M., Kersting, K., Mladenov, M., Selman, E.: Dimension reduction via colour refinement. In: Schulz, A.S., Wagner, D. (eds.) Algorithms - ESA 2014. pp. 505–516. Springer Berlin Heidelberg (2014)
14. Herr, K.: Core Sets and Symmetric Convex Optimization. Ph.D. thesis, TU Darmstadt (2013)
15. Herr, K., Rehn, T., Schürmann, A.: Exploiting symmetry in integer convex optimization using core points. Operations Research Letters **41**(3), 298–304 (2013)
16. Herr, K., Rehn, T., Schürmann, A.: On lattice-free orbit polytopes. Discrete Comput Geom **53**, 144–172 (2015)
17. Hojny, C.: Packing, partitioning, and covering symresacks. Discrete Applied Mathematics **283**, 689–717 (2020)
18. Hojny, C., Pfetsch, M.: Polytopes associated with symmetry handling. Mathematical Programming **175**, 197–240 (2019)
19. Junttila, T., Kaski, P.: Engineering an efficient canonical labeling tool for large and sparse graphs. In: Applegate, D., Brodal, G.S., Panario, D., Sedgewick, R. (eds.) Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments and the Fourth Workshop on Analytic Algorithms and Combinatorics. pp. 135–149. SIAM (2007)
20. Kaibel, V., Peinhardt, M., Pfetsch, M.E.: Orbitopal fixing. Discrete Optimization **8**(4), 595–610 (2011)
21. Kaibel, V., Pfetsch, M.: Packing and partitioning orbitopes. Mathematical Programming **114**, 1–36 (2008)
22. Letchford, A.N., Ni, Q., Zhong, Z.: Bi-perspective functions for mixed-integer fractional programs with indicator variables. Mathematical Programming **190**, 39–55 (2021)
23. Liberti, L.: Automatic generation of symmetry-breaking constraints. In: Yang, B., Du, D.Z., Wang, C.A. (eds.) Combinatorial Optimization and Applications. pp. 328–338. Springer Berlin Heidelberg (2008)
24. Liberti, L.: Reformulations in mathematical programming: automatic symmetry detection and exploitation. Mathematical Programming **131**, 273–304 (2012)
25. Liberti, L.: Symmetry in mathematical programming. In: Lee, J., Leyffer, S. (eds.) Mixed Integer Nonlinear Programming. pp. 263–283. Springer New York (2012)
26. Liberti, L., Ostrowski, J.: Stabilizer-based symmetry breaking constraints for mathematical programs. Journal of Global Optimization **60**, 183–194 (2014)
27. Margot, F.: Symmetry in integer linear programming. In: Jünger, M., Liebling, T.M., Naddef, D., Nemhauser, G.L., Pulleyblank, W.R., Reinelt, G., Rinaldi, G., Wolsey, L.A. (eds.) 50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art, pp. 647–686. Springer Berlin Heidelberg (2010)
28. Margot, F.: Pruning by isomorphism in branch-and-cut. Mathematical Programming **94**, 71–90 (2002)
29. Margot, F.: Exploiting orbits in symmetric ILP. Mathematical Programming **98**, 3–21 (2003)

30. Margot, F.: Small covering designs by branch-and-cut. Mathematical Programming **94**, 207–220 (2003)
31. Margot, F.: Symmetric ILP: Coloring and small integers. Discrete Optimization **4**(1), 40–62 (2007)
32. McKay, B.D., Piperno, A.: Practical graph isomorphism, ii. Journal of Symbolic Computation **60**, 94–112 (2014)
33. MOSEK ApS: MOSEK command line tools's documentation. Version 10.0. (2022), https://docs.mosek.com/10.0/cmdtools/index.html
34. Ostrowski, J.: Symmetry in Integer Programming. Ph.D. thesis, Lehigh University, Bethlehem, PA (2008)
35. Ostrowski, J.: Symmetry handling in mixed-integer programming. In: Wiley Encyclopedia of Operations Research and Management Science. John Wiley & Sons, Ltd (2011)
36. Ostrowski, J., Linderoth, J., Rossi, F., Smirglio, S.: Orbital branching. Math. Programming **126**, 147–178 (2011)
37. Permenter, F., Parrillo, P.: Dimension reduction for semidefinite programs via Jordan algebras. Math. Programming **181**, 51–84 (2020)
38. Pfetsch, M., Rehn, T.: A computational comparison of symmetry handling methods for mixed integer programs. Mathematical Programming Computation **11**, 37–93 (2019)
39. Puget, J.F.: Automatic detection of variable and value symmetries. In: van Beek, P. (ed.) Principles and Practice of Constraint Programming - CP 2005. pp. 475–489. Springer Berlin Heidelberg (2005)
40. Rehn, T.: Exploring Core Points for Fun and Profit: A Study Of Lattice-free Orbit Polytopes. Ph.D. thesis, University of Rostock (2014)
41. Salvagnin, D.: A Dominance Procedure for Integer Programming. Master's thesis, University of Padova (2005)
42. Salvagnin, D.: Orbital shrinking: A new tool for hybrid mip/cp methods. In: Gomes, C., Sellmann, M. (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. pp. 204–215. Springer Berlin Heidelberg (2013)
43. Salvagnin, D.: Symmetry breaking inequalities from the schreier-sims table. In: van Hoeve, W.J. (ed.) Integration of Constraint Programming, Artificial Intelligence, and Operations Research. pp. 521–529. Springer International Publishing (2018)
44. Sherali, H.D., Smith, J.C.: Improving discrete model representations via symmetry considerations. Management Science **47**(10), 1396–1407 (2001)