Data Structures for Representing Symmetry in Quadratically Constrained Quadratic Programs

Georgia Kouyialis · Ruth Misener
Department of Computing, Imperial College London, SW7 2AZ, UK 
{g.kouyialis14,r.misener}@imperial.ac.uk

Abstract. Symmetry in mathematical programming may lead to a multiplicity of solutions. In nonconvex optimisation, it can negatively affect the performance of the Branch and Bound algorithm. Symmetry may induce large search trees with multiple equivalent solutions, i.e. with the same optimal value. Dealing with symmetry requires detecting and classifying it first. This paper develops several methods for detecting symmetry in quadratically constrained quadratic optimisation problems via adjacency matrices. Using graph theory, we transform these matrices into binary layered graphs and enter them into the software package nauty. Nauty generates important symmetric properties of the original problem.

Keywords: quadratic programs, symmetry, binary layered graph, automorphism group

1 Introduction

Several geometry problems are mathematically formulated as quadratically constrained quadratic programs. The occurrence of symmetry in these problems results in many equivalent feasible and optimal solutions that each can be technically generated from the other. Identifying and classifying problem symmetries is an important step towards exploiting tree-based algorithms such as branch-and-cut. This subsequently allows state-of-the-art solver softwares to omit symmetric solutions. In this Section we initially study quadratically constrained quadratic programs and the McCormick relaxation method applied on the bilinear terms presented in this formulation. We also provide basic preliminaries on group theory.

1.1 Mathematical Preliminaries

1.1.1 Quadratically Constrained Quadratic Programs The basic formulation (PQ):

Definition 1.

\[
\min_{x \in \mathbb{R}^n} f_0(x)
\]
\[s.t. \quad f_k(x) \leq 0 \quad \forall k = 1, \ldots, m
\]
\[x_i \in [x^L_i, x^U_i] \quad \forall i = 1, \ldots, n
\]

where

\[
f_k(x) = \sum_{i=1}^n \sum_{j=1}^n x_i \alpha_{ij}^k x_j + \sum_{i=1}^n \alpha_{i0}^k x_i + \alpha_{00}^k \forall k = 0, \ldots, m
\]

with coefficients \(\alpha_{ij}^k \in \mathbb{R}\) for \(i \in \{1, \ldots, n\}, j \in \{1, \ldots, n\}\) and \(k \in \{0, 1, \ldots, m\}\). We assume finite bounds on every variable \(x_i \in [x^L_i, x^U_i], i \in \{1, \ldots, n\}\).
The property of nonconvexity that arise in these problems, imposes more complexities. It causes the existence of multiple local optima and we seek for a global solution that gives the best optimal value. Mccormick [30] achieves a convex relaxation of such problems by adding inequality constraints generated on new auxiliary variables which combine the given ones. More precisely a Reformulation Linearisation Technique (RLT) is the Mccormick convex and concave relaxation for bilinear terms. In this part, we follow Anstreicher [21] and derive convex relaxation of the original [PQ] in Definition 1. For each bilinear term set $X_{ij} = x_i x_j$, the Mccormick hull forms under and overestimator constraints, related to the variable bounds $x_i^L \leq x_i \leq x_i^U$, $x_j^L \leq x_j \leq x_j^U$

\[
\begin{align*}
X_{ij} &\geq x_i^L x_j^L + x_i^L x_j^U + x_i^U x_j^L - x_i^U x_j^U \\
X_{ij} &\geq x_i^L x_j^L + x_i^L x_j^U + x_i^U x_j^L - x_i^U x_j^U \\
X_{ij} &\leq x_i^U x_j^U + x_i^U x_j^U - x_i^U x_j^U \\
X_{ij} &\leq x_i^L x_j^L + x_i^U x_j^U - x_i^U x_j^U 
\end{align*}
\]

**Statement 1.** Any [PQ] can be linearised by using the reformulation linearisation technique.

To derive the Mccormick [30] convex and concave relaxation for bilinear terms, consider any quadratic equation of the form $f_k(x) = x^T Q_k x + p_k^T x + r_k \leq 0 \forall k = \{0, \ldots, m\}$ and define:

\[
X = xx^T = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} (x_1 \cdots x_n) = \begin{pmatrix} x_1 x_1 & x_1 x_2 & \cdots & x_1 x_n \\ x_2 x_1 & \ddots & \cdots & \vdots \\ \vdots & \cdots & \ddots & x_n x_1 \\ x_n x_1 & \cdots & \cdots & x_n x_n \end{pmatrix}
\]

Rewrite each quadratic expression using the inner product [32]:

\[
x^T Q_k x = \sum_{i=1}^{n} \sum_{j=1}^{n} Q_{ki} x_i x_j = Q_k \bullet X = \sum_{i=1}^{n} \sum_{j=1}^{n} Q_{kj} x_{ij}
\]

Now use the variable bounds of $x_i, x_j$ to obtain the constraints of the following linearised optimisation problem. Note that for $i \neq j$, $X_{ij} = X_{ji}$ as the matrices are symmetric above their diagonal, and $X = X^T$. Hence we can define the linear form of [PQ] as

**Definition 2.**

\[
\begin{align*}
\max & \quad Q_0 \bullet X + p_0^T x + r_0 \\
\text{s.t.} & \quad Q_k \bullet X + p_k^T x + r_k \leq 0 \quad \forall k = \{1, \ldots, m\} \\
& \quad X - x^L x^T - x(x^L)^T \geq -x^T (x^L)^T \\
& \quad X - x^L x^T - x(x^L)^T \leq -x^T (x^L)^T \\
& \quad X - x^L x^T - x(x^U)^T \geq -x^T (x^U)^T \\
& \quad X - x^L x^T - x(x^U)^T \leq -x^T (x^U)^T \\
& \quad X = X^T \\
& \quad x \in [x^L, x^U]
\end{align*}
\]

where $x \in \mathbb{R}^n$, $Q_0, \ldots, Q_m \in \mathbb{R}^{n \times n}$, are $n$ by $n$ matrices and $p_k \in \mathbb{R}^n$, $r_k \in \mathbb{R}$.

### 1.1.2 Group Theory

Basic definitions and notation of group theory are provided as a supportive material on the illustration of our idea and the analysis of the results [6]. A group $(W, \cdot)$ is a nonempty set $W$ with a binary operation $\cdot$ on $W$ satisfying the following properties:
A permutation of a set $Y = \{1, \ldots, n\}$ is a bijective function $\pi : Y \rightarrow Y$. For permutations $\pi \in \Pi^n$, $\sigma \in \Pi^n$, $A(\pi, \sigma)$ is a matrix obtained by permuting the columns of $A$ by $\pi$ and the rows of $A$ by $\sigma$. The set $S_n : Y \rightarrow Y$ is the symmetric group of all permutations on $n$ symbols. Its group operation is the composition of such permutation operations.

A subgroup $Z$ of group $W$ is a nonempty subset of $W$ that forms a group itself under the operations induced by $W$.

Two groups $W$, $Z$ are isomorphic if there exists a bijective function $\phi : W \rightarrow Z$ that satisfies:

1. $\phi(I) = I$;
2. $\phi(g^{-1}) = \phi(g)^{-1}$, $\forall g \in W$;
3. $\phi(gz) = \phi(g)\phi(z)$, $\forall g, z \in W$.

The automorphism group or in other words the symmetry group is an isomorphism of a group to itself; maps a group to itself while preserving all of its structure. A main part of this paper evolves around the symmetry in the original nonconvex problem and how it is affected after relaxing the problem which leads to an ordinary linear program $P^{LQ}$ problem and its dual $P^{LQ}$. For permutations $\pi \in \Pi^n$, $\sigma \in \Pi^n$, $A(\pi, \sigma)$ is a matrix obtained by permuting the columns of $A$ by $\pi$ and the rows of $A$ by $\sigma$. The set $S_n : Y \rightarrow Y$ is the symmetric group of all permutations on $n$ symbols. Its group operation is the composition of such permutation operations.

A subgroup $Z$ of group $W$ is a nonempty subset of $W$ that forms a group itself under the operations induced by $W$.

Two groups $W$, $Z$ are isomorphic if there exists a bijective function $\phi : W \rightarrow Z$ that satisfies:

1. $\phi(I) = I$;
2. $\phi(g^{-1}) = \phi(g)^{-1}$, $\forall g \in W$;
3. $\phi(gz) = \phi(g)\phi(z)$, $\forall g, z \in W$.

The automorphism group or in other words the symmetry group is an isomorphism of a group to itself; maps a group to itself while preserving all of its structure. A main part of this paper evolves around the symmetry in the original nonconvex problem and how it is affected after relaxing the problem which leads to an ordinary linear program $P^{LQ}$ problem and its dual $P^{LQ}$.

Section 1.2 provides a specific geometry problem which motivates the study of symmetry and surveys the relevant literature. Section 2 formally defines symmetry in quadratic optimisation problems and identifies the role of integrality and nonconvexity in such cases. Section 3 introduces a novel symmetry detection methodology; Initially we associate quadratic and linear optimisation problem with matrices. Then we construct binary layered graphs that encode information from the matrices and capture the structure of the original problem. We employ the software package nauty with these graphs which generates important symmetric information of the original problem. The algorithm implementation provided in the software package nauty by McKay [31] is associated with a search tree and determines the automorphism group of a problem and whether two graphs are isomorphic. The generator of this group can be projected to the variables and the constraints of the given mathematical program. A computational case that corroborates the proposed methods and the conclusion are discussed in Sections 4, 5.

1.2 Background

1.2.1 Motivation Visually consider a problem of locating two identical circles $(c_1, c_2)$ with centre coordinates $(x_1, x_2), (x'_1, x'_2)$ in a unit square. The optimisation problem is to make the circles as large as possible without overlapping. There are four ways to locate these circles and they are related by rotations and reflections. Mathematically speaking, there are four sets of feasible (approximated) solutions which give the same objective value; distance between their centre coordinates.

---

Table 1: Table of Notation.

| Symbol | Description | Symbol | Description |
|--------|-------------|--------|-------------|
| $x_i$  | Variables   | $I$    | Identity element |
| $x$    | Vectors of variables | $\pi, \sigma$ | Permutations |
| $\alpha$ | Coefficient | $\Pi^n$ | Set of all permutations |
| $c, b, p$ | Vectors of parameters | $S_n$ | Symmetric group order $n$ |
| $A, Q$ | Matrices of parameters | $Y$ | Sets |
| $X$ | Matrix of auxiliary variables | $f, h, \phi$ | Functions |
| $M, IM, JM, KM$ | Sparse representations of matrices | $G, H$ | Graphs |
| $\mathcal{F}$ | Set of feasible solutions | $E, V$ | Set of edges, vertices |
| $\mathcal{G}, \mathcal{G}'$ | Symmetry groups | $e$ | Edges in the graph |
| $W, Z$ | Groups | $u, v$ | Nodes in the graph |
As shown in Figure 1.2.1, 1 = \{(0.293, 0.293), (0.707, 0.707)\}, 2 = \{(0.293, 0.707), (0.707, 0.293)\}, 3 = \{(0.707, 0.707), (0.293, 0.293)\}, 4 = \{(0.707, 0.293), (0.293, 0.707)\}. However only one is studied as unique as all the others can be obtained by permuting the variables of the problem. Consider solution 1 and permute variables \((x_3, x'_2)\) to get solution 2 = \{(0.293, 0.707), (0.707, 0.293)\}. \((x_1, x'_1)\) to get 4 = \{(0.707, 0.293), (0.293, 0.707)\}, apply both permutations \((x_2, x'_2)(x_1, x'_1)\) to get solution 3 = \{(0.707, 0.707), (0.293, 0.293)\}. Permutations \((x_1, x_2)\) and/or \((x'_1, x'_2)\) take solution 1 to itself \([7]\). The exchange of the variables of the problem which leaves the set of feasible solutions and the objective function value unaffected is the symmetry in an optimisation problems.

A widely used algorithm for solving mathematical programming problems is the Branch and Bound (B&B) algorithm \([42]\). A tree search strategy that divides the problem into subproblems and solves them recursively by forming upper and lower bounds for each one. The algorithm converges as the gap between the original nonconvex problem and the convex relaxation decreases as the size of variable domain is reduced and a better solution can not be obtained. Hence symmetry can cause unexpectedly large trees which immediately affects the time that is taken for the problem to be solved. In other instances, some highly symmetric problems can remain unsolvable if their symmetry is not exploited \([17]\). So exploiting symmetry, e.g. via advanced branching strategies, may offer an important advantage for branch-and-cut \([38]\). A first step towards that is to detect and represent symmetry which consists the novel contribution of this work.

1.2.2 Literature Review Margot \([29]\) defines symmetry in Integer Linear Programming (ILP) of the form \(P^L = \min_{x \in \mathbb{Z}^n} \{c^T x \mid Ax \geq b\}\) where \(A \in \mathbb{R}^{m \times n}\) and vectors \(c \in \mathbb{R}^n, b \in \mathbb{R}^m\) as the set of variable permutations under which any feasible solutions remains feasible and the objective function value is invariant. Let \(\mathcal{F}\) be the set of feasible solutions of any problem \(P^L\). The symmetry group is:

\[
\mathcal{G}(P^L) = \{ \pi \in \Pi^n \mid \forall \; \hat{x} \in \mathcal{F}, \; \pi(\hat{x}) \in \mathcal{F} \quad \text{and} \quad c^T \pi(\hat{x}) = c^T \hat{x} \}
\]

Liberti \([22]\) studies and extends the definition of symmetry to mixed-integer nonlinear optimisation problems.

Symmetric structure in optimisation may be viewed through the lens of group theory for \([20]\ \[4\ \[13\]. In many situations though, it is hard to detect the symmetry of the original problem and its polyhedron representation \([5]\) since the problem needs to be solved which is not always possible neither easy. A subgroup of the symmetry group; the formulation group of each problem reflects the symmetric properties of its variables and constraints on which the symmetry group is based. Hence symmetry handling approaches present methodologies to associate optimisation problems with graph representations from which the graph automorphism is generated by using software tools \([3\ \[3\ [19\ \[23\ \[29\ \[39\].

Many researchers exploit the above information and the insights of several problems; covering problems \([26]\ \[28\], scheduling and packing problems \([7]\ \[35\] and engineering problems as the unit commitment problem and heat exchanger network synthesis \([1]\ \[20\ \[37\]. They identify the presence of symmetry and in some cases propose symmetry
handling approaches for problems with known symmetric structure. The improved performance of the solvers validates
the efficiency of these techniques [38]. However, they are problem specific and cannot be generalised to other problems.

There are several methods to exploit symmetry which are categorised as static and dynamic methods. Static meth-
ods adjoin new constraints to the formulation in order to make some symmetric optima infeasible. Sherali and co-
workers add symmetry breaking constraints or perturb the objective function [44, 41]. Other researchers investigate
the orbitopes of a problem [8, 9]: convex hull of 0-1 matrices that represent possible solutions to packing and par-
titioning constraints. The new constraints yield to a reformulation which is guaranteed to keep at least one symmetric
optimum feasible. Orbitopes have additionally been considered for cutting planes [11, 16]. Liberti [21] automatically
generates symmetry handling inequalities, whereas other works study inequalities which exploit multiple variable or-
bits [8, 24]; the groups of variables that can be sent to each other under some actions (permutations in the group) which
are equivalent with respect to symmetry of the problem.

In the second category fall approaches which modify the solution method i.e. the search tree algorithm to recognis-
and exploit symmetry dynamically as it goes along. For example, constraints can be derived for each node in the tree
to forbid the isomorphic nodes [13, 12, 20]. Exploiting symmetry also advances from restrictions and directions on
pruning and branching via isomorphism pruning [25, 26, 27], and orbital/constrained orbital branching [34, 36]. By
introducing artificial variables, Fischetti et al. [10] reformulate the problem to a reduced problem which considers only
variables of symmetry orbits instead of all variables, so-called orbital shrinking.

While most of these works consider the symmetry representation as a step enclosed by the scope of handling
symmetry, Liberti is the first who state the importance of a practical and general representation of symmetry. He uses
expression trees to explicitly capture the structure of an optimisation problem and develops the ROSE reformulation
software engine that produces a file representation of the problem as Directed Acyclic Graphs (DAG).

The work introduced in this paper concerns with the improvements on symmetry detection, which is the first phase
of symmetry handling techniques. Symmetry representation is an elementary process given to the software package
data, on which all the following steps to break symmetry depend. Hence it is very essential to guarantee and increase
its correctness and efficiency.

2 Symmetry in Quadratic Optimisation Problems

After surveying the available sources for detecting symmetry we contemplate the explanation of symmetry given by
Margot [29] which also presented by Liberti [22] on different problems. Under a set of permutations of the variables
of the problem, each feasible solution can be mapped to another solution having the same value and the whole set of
feasible solutions \( \mathcal{F} \) can be mapped to itself.

Modifying this definition to the case of quadratic problems we define symmetry in \( P^Q \)

\[
\mathcal{G}(P^Q) = \{ \pi \in \Pi^n \mid \forall \hat{x} \in \mathcal{F}, \quad \pi(\hat{x}) \in \mathcal{F} \quad \text{and} \quad (\pi(\hat{x}))^T Q_0 \pi(\hat{x}) + p_0^T \pi(\hat{x}) = \hat{x}^T Q_0 \hat{x} + p_0^T \hat{x} \}
\]

This work discern the symmetry in nonlinear problems \( P^Q \) and after applying RLT technique the symmetry in
its linearised form \( P^{LQ} \). While relaxing the problem more and more constraints need to be added and especially on
duplications caused by symmetry they are unnecessary. This might lead to forms of the original problem in the tree
which are even harder to be solved at the end. We evaluate and state how the integrality and nonlinearities affect the
symmetry group of the original problem. A Mixed-Integer Quadratically Constrained Quadratic program \( P^{MIO} \) is a
generalised form of \( P^{LQ} \) where a subset of the variables \( x_i \) can also take discrete values.

**Theorem 1.** The symmetry in the mixed-integer quadratic problem \( P^{MIO} \) implies the symmetry in \( P^{LQ} \)

**Proof.** When relaxing the integrality constraints of \( P^{MP} \) to get a \( P^{LQ} \) problem the feasible region of possible solutions
becomes wider and \( \mathcal{F}(P^{MIO}) \subseteq \mathcal{F}(P^{LQ}) \). Hence for any \( \pi \in \mathcal{G}(P^{MIO}) \Rightarrow \pi \in \mathcal{G}(P^{LQ}) \) and \( \mathcal{G}(P^{MIO}) \subseteq \mathcal{G}(P^{LQ}) \). 

Suppose that \( P^{LQ} \) is a linearised program and definitions of formulation group follows definition of the quadratic
case.
Theorem 2. The symmetry in the original quadratic problem \( P \) implies the symmetry in the linearised problem \( P_{LQ} \).

Proof. Using McCormick relaxations we get the convex and concave envelopes of the bilinear set. Hence the feasible region of the original problem is not affected and the feasible set of solutions \( \mathcal{F}(P) \subseteq \mathcal{F}(P_{LQ}) \). Let \( \pi \in \mathcal{G} \) and suppose that \( \exists \bar{x} \in \mathcal{F}(P) \) s.t. \( \pi(\bar{x}) \in \mathcal{F}(P_{LQ}) \subseteq \mathcal{F}(P_{LQ}) \) and as any solution under \( \pi \) is still feasible in \( P_{LQ} \) the objective function value of \( P_{LQ} \) remains unaffected, hence \( \pi \in \mathcal{G}(P_{LQ}) \). Hence \( \mathcal{G}(P_{LQ}) \subseteq \mathcal{G}(P) \). \( \square \)

2.1 Novel Structures to Represent Symmetry

It is critical to point out that the symmetry group is based on the feasible set of solutions of an optimisation problem. Deriving this set is the ultimate aim when solving a problem which is impractical in our work. Hence the scope of this paper is to efficiently associate data structures with optimisation problems which can generate the formulation group; a set of permutations that fix the problem formulation, a subset of the symmetry group. In the following section we propose and evaluate structures to detect the formulation group which captures the symmetric nature of a given linear and nonlinear programming problem.

2.2 Matrix Representation

In this part we suggest two different methods of forming a problem as an adjacency matrix. We use these matrices to define the formulation group of a problem and then transform it into a graph for detecting and classifying the automorphism group which reveals the symmetry. The presence of linear and bilinear terms in quadratic problems though indicates their difficulty. It is important to mention that we initially considered to construct a matrix where the columns would represent the linear and bilinear terms of a problem and the rows the objective function and the given constraints. However after working on that case we observed that we could not get the required automorphism group. On the graph representation we realised that these has been caused due to the fact that each bilinear term though indicates their difficulty. It is important to mention that we initially considered to construct a matrix where the columns would represent the linear and bilinear terms of a problem and the rows the objective function and the given constraints. However after working on that case we observed that we could not get the required automorphism group. On the graph representation we realised that these has been caused due to the fact that each bilinear term

Consider the problem \( P_{LQ} \) which incorporates the constraints of the original problem and the RLT constraints formed by McCormick relaxation for each nonlinear term as presented in Section 1. Let \( \tilde{m} = (1 + m + (# \text{ of non linear terms}) \times 4) \) and \( \tilde{n} = (1 + n + # \text{ of non linear terms}) \).

Method 1. Create a 2 dimensional matrix: \( A^{LQ} \in \mathbb{R}^{\tilde{m} \times \tilde{n}} \), with entries the coefficients of each term in \( P_{LQ} \)

The number of columns set as \( \tilde{n} \) consist of an identity elements, each variable and the auxiliary variables introduced for nonlinear terms and the number of rows say \( \tilde{m} \) for the objective function and all the constraints of the problem. Note that the maximum number of nonlinear terms is: \( \frac{n(n+1)}{2} \).

Method 2. Create a tensor: \( A^{Q} \in \mathbb{R}^{(n+1) \times (n+1) \times (m+1)} \), with entries the coefficients of each term in \( P_{LQ} \)

Each matrix corresponds to an equation of the problem and the rows and columns to an identity element and the variables of the problem, capturing the relations between the bilinear term.

Comparing these two methods and regarding the next step which is the transformation of the adjacency matrix into a graph we are currently working with Method 2. If we think of the case where a problem has \( n \) variables, as already stated for Method 1 for each nonlinear term 4 new constraint need to be added. Then in the worst case scenario we might have up to \((1+n)(1+\frac{n}{2})(1+m+2n^2+2n)\) entries in contrast to the Method 2 where we have up to \((1+n)(1+n)(1+m)\). In most cases, Method 2 has fewer entries than Method 1. There exist pathological cases, e.g. fully dense formulations where \( m > 3n^2 + 6n + 3 \), where Method 1 has fewer entries. The graph transformation is based on the number of entries of these matrices. Hence, dealing with smaller graphs reduces their complexity and the procedure and time taken to generate their symmetric properties and to compare them.
A = \begin{bmatrix}
\text{identity} & \text{Variables} \\
\cdots & \cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots & \cdots 
\end{bmatrix}

\{ \text{Objective Function} \}

\{ \text{Constraints} \}

RLT constraints

Fig. 2: Method 1

Fig. 3: Method 2

Since many of the values in the matrix are going to be 0, a sparse matrix representation $A = A(I,J,K)$ size $I,J,K: n+1,n+1,m+1$ is used to reduce space in memory and time accessing all the coefficient of the problem. This representation stores only nonzero values of $A$, as elements indexed by the two variables in row, column and the stage for the relevant constraint.

To support Method 1, Method 2 on how a problem can be represented as a matrix we provide the following mathematical work. Note that the indices of the position (rows, columns) of each entry start from 0.

For $P^{LQ}$ the associated matrix $A^{LQ} = (a_{kj}) \in \mathbb{R}^{n \times n}$ is a matrix with entries the coefficients of $P^{LQ}: a_{kj}$, for $k = \{0,1,\ldots,n\}, j = \{0,1,\ldots,n\}$.

For $P^{Q}$ the associated tensor $A^{Q} = (a^{k}_{ij}) \in \mathbb{R}^{(n+1) \times (n+1) \times m}$ consists on the coefficients of $P^{Q} \alpha^{k}_{ij}$, for $i = \{0,1,\ldots,n\}, j = \{0,1,\ldots,n\}, k = \{0,1,\ldots,m\}$.

Next, we define the formulation group of a matrix; necessary for detecting symmetry.

The formulation group $G$ of the matrix $A^{LQ}$

$$G(A^{LQ}) = \{\pi \in \Pi^{n} | \exists \sigma \in \Pi^{m} \text{ such that } A(\sigma, \pi) = A\}$$

Is the set of permutations of the columns of $A^{LQ}$ such that there is a corresponding permutation of the rows that when applied yields the original matrix.

Same for $A^{Q}$ is defined:

$$G(A^{Q}) = \{\pi \in \Pi^{n} | \exists \sigma \in \Pi^{m} \text{ such that } A(\pi, \pi, \sigma) = A\}$$

The set of permutations of the columns and rows of each matrix in $A^{Q}$ such that there is a corresponding permutation between the matrices that when is applied yields the original tensor. The same permutation $\pi$ acts both on rows and columns of the matrix $A^{Q}$ which represent the same number and type of variables.

### 2.3 Binary Layered Graphs

To detect symmetry we convert these adjacency matrices into graphs and then use software nauty [31] for generating the required symmetric information of the problem. In $P^{Q}$ problems, except of the nonlinear terms it is also important to consider how to include the different coefficients of the variables on the graph representation. Ideally each variable could be a vertex in the graph and each coefficient a label of an edge connecting the vertices involved. However, nauty [31] accepts only unweighted graphs. A structure that we propose and can handle this situation is a binary layered directed graph representation with loops as described below. Initially this section provides preliminary definitions on graph theory and then the description on how to illustrate different mathematical problems as graphs with the relevant mathematical work on the graph construction.
2.4 Graph Theory

A graph is a tuple $G = (V, E)$ where $V$ is a (finite) set of vertices and $E$ is a finite collection of edges. An edge $e \in E$ from a vertex to itself $e = \{u, u\}$ is said to be a loop.

Two simple graphs $G$ and $H$, are isomorphic, denoted $G \cong H$, if $\exists f : V_G \rightarrow V_H$ such that: \{u, v\} $\in E(G) \iff \{f(u), f(v)\} \in E(H)$. In a graph any set of adjacent vertices as in $E(V) = \{\{u, v\} | u, v \in V, u \neq v\}$ is still adjacent under this map. An automorphism is an isomorphism of a graph to itself. Given a graph $G$, a permutation $\pi$ of $V(G)$ is an automorphism of $G$, if $\forall u, v \in V(G)$ such that: $\{u, v\} \in E(G) \iff \{\pi(u), \pi(v)\} \in E(G)$.

A partitioning of $G$ into $u$ parts is a collection of nonempty disjoint subsets $V_0, \ldots, V_{u-1}$ for $k \in \mathbb{Z}$ whose union is $V$, i.e., $V = V_0 \cup \ldots \cup V_{u-1} \forall u$. In this paper, the notion of vertex colouring is used to partition the set of vertices of a graph into subsets of vertices with the same colour. A graph $G = (V, E)$ is coloured if it is associated with a function $c_v : V(G) \rightarrow \{0, 1, \ldots, u-1\}$, $\forall v$. For any $v \in V$, vertex $v$ is assigned colour $c_v(v)$. According to this definition $k$ is considered as the number of colour classes of $V$, and $G(V, E)$ is called a $u$-layered graph.

The main graph structure that we are using to visualise the mathematical problems in this report is the binary layered graph.

**Definition 3.** Binary layered graph: Is a vertex coloured graph $G = (V, E)$ where each colour $i \in \{0, 1, \ldots, u-1\}$ is associated with a binary representation and the number of layers is derived from:

$$l = \lceil \log_2 (u+1) \rceil \quad \text{for} \quad u \in \mathbb{Z}$$

**Fig. 4:** General form of a Binary Layered Graph.

Hence for $s = 0, 1, \ldots, l-1$ each partition $V_s = \{v_0^{(s)}, \ldots, v_{n-1}^{(s)}\}, \forall n \in \mathbb{Z}

The form of this graph is:

The automorphism group of a vertex coloured graph is defined as:

$$Aut(G(V, E), c_v) = \{\pi \in \Pi^{|v|} | \pi \in G(V, E), c_v(\pi(v)) = c_v(v), \forall v \in V\}$$

Note that, in literature vertices are also called nodes or points and edges lines. Next we describe the different graphic illustrations of problems with a finite number of algebraic expressions.

2.5 Graph Representation

Subsection 2.2 presents two methods on how to associate matrices with optimisation problems $P^{LQ}$ and $P^Q$

For $i = \{1, \ldots, n\}, k = \{1, \ldots, m\}, k \in \mathbb{Z}$ where $n$ is the number of variables, $m$ is the number of constraints and $u$ the number of unique coefficients in each problem. The following graph representations skeletons are presented for $G = (V, E)$:
Graph 1. is compatible with linear problems (originally or after applying RLT) and matrix representation in Method 1. The number of layers \( L = \lceil \log_2 (u+1) \rceil + 1 \), where on (layer 0) vertices correspond to the objective function and the constraints of the problem. On every other layer there are copies of these nodes as shown by the vertical lines. Then on the top layer there is one vertex for an identity element and vertices for each variable of the problem.

The total number of vertices is: \(|V| = (\hat{n} + 1) \times (L - 1) + \hat{m} + 1\).

From nodes in (layer 0) and its copies, we add edges with endpoints the nodes on the top layer, based on which variable is included on each constraint and what is the coefficient in front of this variable.

```
1: procedure G=(V,E)
2: \( V \leftarrow \emptyset \), \( V_i \subset V \leftarrow \emptyset \), \( E \leftarrow \emptyset \)
3: \( L = \lceil \log_2 (|U|+1) \rceil + 1 \) \( \triangleright \) Define \( L \in \mathbb{Z}^+ \) the number of layers
4: for \( s = 0 \rightarrow L - 2 \) do \( \triangleright \) Partition of vertices
5: \( \text{for } k = 0 \rightarrow \hat{m} \) do
6: \( V_s \leftarrow V_s \cup \{v_s^{(k)}\} \) \( \triangleright \) Copies of vertices representing the constraints on (layer 0)
7: \( V_s \leftarrow V_s \cup V_s \)
8: for \( j = 0 \rightarrow \hat{n} \) do
9: \( V_{L-1} \leftarrow V_{L-1} \cup \{v_j^{(L-1)}\} \) \( \triangleright \) Vertices representing the variables
10: return \( G \)
```

Note that vertices on each layer are classified in the same partition since they can be exchanged between them.

3 Detect Symmetry

In this Section we explain how to construct a graph from a mathematical optimisation problem that consists of a finite number of algebraic expressions and how to detect symmetry. Moreover we compare the proposed method with existing methods in literature.

3.1 Transformation to Binary Layered Graphs

The following part outlines the high-level reduction of the problem to a binary layered graph by using the matrices in Methods 1, 2 and a set of rules on how to add edges on each different case.

Given \( X = \{x_i\} \) for \( i = \{0, \ldots, n\} \), \( x_0 = I \) is an identity element with \( x_0 x_i = x_i \) \( \forall i \), and sparse representation of vectors \( M, IM, JM, KM \) with maximum number of entries \( N \in \mathbb{Z} \).
The following example incorporates all the steps and the algorithms proposed in this paper. We construct the binary labelled graph and then enter it into nauty through dreadnaut command lines which compute the formulation group of the original problem.

Fig. 6: Illustration of Graph 2 and the relevant algorithm on how to construct the vertex set.

The number of layers consists on the number of unique elements \(|U| \in \mathbb{Z}\) in vector \(M\) and each one is stored in a vector \(\tilde{U} \in \mathbb{R}^p\). Moreover, nauty reads only positive integer values and symmetry consists on the relations of the variables and the input data on the coefficients of the problem. Then assigning positive integer values on the entries of \(M\) will not affect the symmetric properties of the problem as long as position arrays remain unaffected. We introduce the function: \(h: M \rightarrow M \times \mathbb{Z}^+\). Binary layered graph benefits from a binary representation of each coefficient which is generated from function: \(M(t) = c_{t-1}2^{L-1} + c_{L-2}2^{L-2} + \ldots + c_02^0\), for \(c_t \in \{0,1\}\). For nonzero \(c_t\) the powers of 2 i.e. \(s = \{0,\ldots,L-1\}\) reveal which layers encode that value.

3.2 Graph Construction

Using the above information we can now construct a graph \(G = (V,E)\) for each of the three cases. Initially set \(V = \emptyset, E = \emptyset\). It is worth explaining how to create the edge set which presents the linear and bilinear relations and the coefficients of the problems. Given a sparse matrix representation \(M, IM, JM, KM \in \mathbb{R}^{N}\).

**Statement 2.** Matrix \(A^L \in \mathbb{R}^{(n \times (n+1))^L}\) is associated to the Graph 1 representation.

Algorithm 1 shows how to construct the edge set which consists on the vertical edges that connect copies of vertex \(j\) on (layer 0) to the vertex \(j\) on the next layer (see lines 3-5). Moreover the edges from (layer \(L-1\)) to any other layer are based on which variable is included in each constraint and its coefficient value. For any \(s = \{0,\ldots,L-1\}\), if \(c_s = 1\) then \((s)\) indicates the required layers and the graph construction follows lines 8-11.

**Statement 3.** Tensor \(A^Q \in \mathbb{R}^{(n+1) \times (n+1) \times m}\) is associated to the Graph 2 representation.

In the same way as the previous statement, we are given a sparse matrix representation: \(M, IM, JM, KM \in \mathbb{R}^{N}\). The edge set consists on three parts presented in Algorithm 2. The vertical edges between the copies of vertices represent the constraints of the problem starting from (layer 0 - see lines 4-6) and the vertical edges between copies of vertices represent the variables of the problem as derived in lines 9-10. The algorithms encodes the relation of bilinear terms in one partition in lines 12-22. After expressing the entries of \(M\) in binary form, for any \(s\), for \(KM(F) = k, k = \{0,\ldots,m\}, F = \{0,\ldots,N-1\}\), lines 23 -33 show how to connect each constraint to the terms included in it with combining the coefficient as well.

4 Computational Case

The following example incorporates all the steps and the algorithms proposed in this paper. We construct the binary labelled graph and then enter it into nauty through dreadnaut command lines which compute the formulation group of the original problem.
4.1 Numerical example

In this part we consider a simple example of a Problem 1 and apply the two different graph representations as described in Subsection 2.5. Problem 2 is the relaxed form of the original Problem 1 after applying convex relaxation.

**Problem 1.**

\[
\begin{align*}
\text{max} & \quad 3x_1 + 3x_4 + 2x_2x_3 \\
\text{s.t.} & \quad x_2 + x_1^2 + 1 \leq 0 \\
& \quad x_3 + x_4^2 + 1 \leq 0 \\
& \quad x_2 + x_3 + 1 \leq 0 \\
& \quad x_1, x_2, x_3, x_4 \in [0, 1]
\end{align*}
\]

by introducing the auxiliary variables, \( X_{23} = x_2x_3 \), \( X_{11} = x_1^2 \), \( X_{44} = x_4^2 \) and add the McCormick relaxation constraints.

**Problem 2.**

\[
\begin{align*}
\text{max} & \quad 3x_1 + 2X_{23} + 3x_4 \\
\text{s.t.} & \quad X_{11} + x_2 + 1 \leq 0 \\
& \quad x_3 + X_{44} + 1 \leq 0 \\
& \quad x_2 + x_3 + 1 \leq 0 \\
& \quad x_2 + x_3 - X_{23} - 1 \leq 0 \\
& \quad X_{23} - x_2 \leq 0 \\
& \quad X_{23} - x_3 \leq 0 \\
& \quad 2x_1 - X_{11} - 1 \leq 0 \\
& \quad X_{11} - x_1 \leq 0 \\
& \quad 2x_4 - X_{44} - 1 \leq 0 \\
& \quad X_{44} - x_4 \leq 0 \\
& \quad X_{23}, X_{11}, X_{44} \geq 0 \\
& \quad x_1, x_2, x_3, x_4 \in [0, 1]
\end{align*}
\]

The graph representation of Problem 1 has the structure as in Figure 7. Nauty software generates the following permutations: \( \pi = (1\ 2)(5\ 6)(9\ 12)(10\ 11)(14\ 17)(15\ 16) \); the automorphism group of the graph under which it remains invariant. The relevant enumeration distinguishes which permutations are applied on the constraints and which on the variables of the problem. We then reflect these information on the original problem and explain its symmetric properties.
which is the main purpose of this paper. Permutations \((1\,2)(5\,6)\) as shown in Figure 7 permute the constraints \(c_1, c_2\) of Problem 1. Permutations \((9\,12)(10\,11)\) are associated to the variables \(x_1, x_4\) and \(x_2, x_3\) with \((14\,17)(15, 16)\) their copies. Hence the formulation group of Problem 1 is \(G = (x_1x_4)(x_2x_3)\).

On Problem 2, we apply Method 1 described in Section 2 and construct the graph as shown in Figure 8. \textit{Nauty} generates \((1\,2)(5\,6)(7\,9)(8\,10)(12\,13)(16\,17)(18\,20)(19\,21)(23\,24)(27\,28)(29\,31)(30\,32)(34\,37)(35\,36)(39\,40)\) with specific permutations \((34\,37)(35\,36)(39\,40)\) to reveal the symmetric relations of variables \((x_1x_4)(x_2x_3)(X_{11}X_{44})\) the formulation group \(G\) of Problem 2. The above results validate both Method 1, 2 in Section 2 for representing an optimisation problem as a graph and then generate its symmetric properties.

### 4.2 Comparison with current methods

We evaluate the trade-offs among the graph constructions in this paper and different graph constructions already exist in the literature. Regarding the graph transformation and its significant role in dealing with symmetry, Ostrowski et al. \cite{34}, introduce the method “Orbital Branching” for combating symmetry. They illustrate each problem and its subproblems on each node of the tree as graphs. Then they call \textit{nauty} to compute the automorphism group and the orbits of the graph. The graphs that are used are simple graphs of problems with 0−1 coefficients and linear terms which are known to be highly symmetric. The presence of many coefficients in a problem expand the difficulty of identifying its symmetric properties. Liberti \cite{23} states that any mathematical expression with finite number of operators, variables and constants can be illustrated as an expression tree. The vertex set consist of sets of root nodes corresponding to objective function and to the constraints. Additionally there are set of nodes for the operator and the constant nodes. Hence a transformation to a unique canonical tree is required as a pre-processing face, leading to \textit{Directed Acyclic Graphs} from where he automatically generates the formulation group of a problem. A major advance of both methods is that they are easy to implement and \(DAG\) can capture the structure of mathematical problems in the most generalised form of \(P^{MQ}\). The trade-offs of our idea on how to construct the graphs which is where symmetry is detected fall into two categories. The function of assigning integer values to the coefficient enables us to work not only with non 0−1 coefficient but with any other type and value. Comparing it to the other methods suggested, the fact that we are using a logarithmic number of layers can significantly reduce the number of nodes in the graph. We do not need to include a new layer of vertices for each new coefficient but as opposed even for a large number of coefficients we can keep a fairly small number of nodes. For example, we are able to present 60 different coefficients in a graph with 6 layers. Another advantage of our method is that we are able to capture the relation of bilinear terms in a way that the mathematical operations presented in the problem need not to be included in different extra nodes in the problem. Addition is the main operation on which the structure of the graph is based, multiplication is only presented with edges and loops. Subtraction is treated as a new coefficient together with the number that follows. The original form of \(DAG\) graphs without any simplification proposed, can provide us with informations on the exact formulation of the original problem something which is not clear with our method since we focus on the general symmetric structure and not the problem itself. \(BLG\) may be associated with problems that have the exact same symmetric structure but different formulation. As long as we are not dealing with solving the problems on this paper it does not seem to cause any problem.

### 5 Conclusion

This work appraise the presence and significance of symmetry in optimisation problems. Symmetry representation and detection are the fundamental steps towards exploiting symmetry. We propose novel graph structures that we claim to consistently capture the symmetric properties of a problem in a coherent size. Further computational results are required to validate the robustness of this method which is the extension of this work.

We acknowledge support from EPSRC, a DTP to G.K. (EP/P008739/1) and a Research Fellowship (EP/P016871/1) to R.M.
**Algorithm 1 Construct Graph 1 - Edge Set**

1: `procedure G=(V,E)
2: V ← V, E ← ∅
3: for s = 0 → L - 3 do
4: for k = 0 → m do
5: E = E ∪ (v_k^s, v_k^{s+1})
6: for F = 0…N do
7: for k = 0…m do
8: if IM(F) = k then
9: E = E ∪ \{(v_k^{L-1}, v_{JM(F)}^s)\}
10: else
11: E = E ∪ ∅
12: return G

**Algorithm 2 Construct Graph 2 - Edge Set**

1: `procedure G=(V,E)
2: V ← V
3: E ← ∅
4: for s = 0 → L - 4 do
5: for k = 0…n do
6: E = E ∪ (v_k^s, v_k^{s+1})
7: for j = 0…n do
8: E = E ∪ (v_j^{L-2}, v_j^{L-1})
9: for F = 0…N - 1 do
10: if IM(F) = JM(F) then
11: E = E ∪ (v_{IM(F)}^{L-1}, v_{JM(F)}^{L-1})
12: else
13: if IM(F) < JM(F) then
14: E = E ∪ (v_{IM(F)}^{L-1}, v_{JM(F)}^{L-1})
15: else
16: E = E ∪ ∅
17: for F = 0…N - 1 do
18: for k = 0…m do
19: if KM(F) = k then
20: if IM(F) = 0 then
21: E = E ∪ (v_0 v_{JM(F)}^{L-2})
22: else
23: E = E ∪ (v_{L-1} v_{JM(F)}^{L-1})
24: return G

> Edges to connect each constraint with the variables and their coefficient

> Vertical edges between copies of vertices

> Copies of vertices representing the constraints

> Vertical edges between copies of vertices

> Copies of vertices representing the variables

> Bilinear terms

> Add a loop

> Add an edge
Bibliography

[1] J. Alemany, P. Komarnicki, J. Linand, and F. Magnago. Exploiting symmetry in unit commitment solutions for a large-scale electricity market. In Electric Power Systems Research, volume 140, 2016.

[2] K. M. Anstreicher. Semidefinite programming versus the reformulation-linearization technique for nonconvex quadratically constrained quadratic programming. J. Glob. Optim., 43(2):471–484, 2009.

[3] T. Berthold and M.E. Pfetsch. Detecting orbitopal symmetries. In Proceedings of the Annual International Conference of the German Operations Research Society (GOR), pages 433–438. Springer Berlin Heidelberg, 2009.

[4] R. Bödi, K. Herr, and M. Joswig. Algorithms for highly symmetric linear and integer programs. Math. Program., 137(1):65–90, 2013.

[5] D. Bremner, M. S. Sikirić, D. V. Pasechnik, T. Rehn, and A. Schürmann. Computing symmetry groups of polyhedra. LMS J. Comp. Math, 17(1):565–581, 2014.

[6] A. Clark. Elements of Abstract Algebra. Dover Books in Mathematics. Courier Corporation, 1984.

[7] A. Costa, P. Hansen, and L. Liberti. On the impact of symmetry-breaking constraints on spatial branch-and-bound for circle packing in a square. Disc. Appl. Math., 161(1):96–106, 2013.

[8] G. Dias and L. Liberti. Orbital independence in symmetric mathematical programs. In Proceedings of the Combinatorial Optimization and Applications: 9th International Conference, COCOA 2015, TX, USA, 2015., pages 467–480. Springer International Publishing, 2015.

[9] Y. Faenza and V. Kaibel. Extended formulations for packing and partitioning orbitopes. Math. Oper. Res., 34(3):686–697, 2009.

[10] M. Fischetti, L. Liberti, D. Salvagnin, and T. Walsh. Orbital shrinking: Theory and applications. Disc. Appl. Math., 222:109–123, 2017.

[11] E. J. Friedman. Fundamental domains for integer programs with symmetries. In A. W. M. Dress, V. Xu, and B. Zhu, editors, COCOA, Lecture Notes in Computer Science, pages 146–153. Springer, 2007.

[12] I. P. Gent and B. M. Smith. Symmetry breaking in constraint programming. In Proceedings of the ECAI-2000,14th European Conference on Artificial Intelligence, 2000, pages 599–603. IOS Press, 2000.

[13] I. P. Gent, T. Kelsey, S. A. Linton, I. McDonald, I. Miguel, and B. M. Smith. Conditional symmetry breaking. In P. van Beek, editor, Proceedings of the Principles and Practice of Constraint Programming - CP 2005: 11th International Conference, Spain, 2005. Springer Berlin Heidelberg, 2005.

[14] A. Ghoniem and H. D. Sherali. Defeating symmetry in combinatorial optimization via objective perturbations and hierarchical constraints. IIE Transactions, 43(8):575–588, 2011.

[15] K. Herr, T. Rehn, and A. Schürmann. Exploiting symmetry in integer convex optimization using core points. Oper. Res. Lett., 41(3):298–304, 2013.

[16] C. Höjny and M. E. Pfetsch. Symmetry handling via symmetry breaking polytopes. In 13th Cologne Twente Workshop on Graphs and Combinatorial Optimization, Istanbul, Turkey, 2015. proceedings, pages 63–66, 2015.

[17] R. G. Jeroslow. Trivial integer problems unsolvable by branch-and-bound. Math. Program., 6(1):105 – 109, 1974.

[18] V. Kaibel, M. Peinhardt, and M. E. Pfetsch. Orbital fixing. Disc. Optim., 8(4):595–610, 2011.

[19] B. Kneven, J. Ostrowski, and S. Pokutta. Detecting almost symmetries of graphs. Math. Program. Comp., 2017.

[20] G. Kouyiialis and R. Misener. Detecting symmetry in designing heat exchanger networks. In Proceedings of the International Conference of Foundations of Computer-Aided Process Operations - FOCAPO/CPC 2017. Arizona, 2017.

[21] L. Liberti. Automatic generation of symmetry-breaking constraints. In Proceedings of the Combinatorial Optimization and Applications: 2nd International Conference, COCOA 2008, NL, Canada, 2008., pages 328–338. Springer Berlin Heidelberg, 2008.

[22] L. Liberti. Symmetry in mathematical programming. pages 263–283. Springer New York, 2012.

[23] L. Liberti. Reformulations in mathematical programming: automatic symmetry detection and exploitation. Math. Program., 131(1):273–304, 2012.

[24] L. Liberti and J. Ostrowski. Stabilizer-based symmetry breaking constraints for mathematical programs. J. Glob. Optim., 60 (2):183–194, 2014.

[25] F. Margot. Pruning by isomorphism in branch-and-cut. Math. Program., 94(1):71–90, 2002.

[26] F. Margot. Small covering designs by branch-and-cut. Math. Program., 94(2):207–220, 2003.
[27] F. Margot. Exploiting orbits in symmetric ILP. *Math. Program.*, 98(1):3–21, 2003.
[28] F. Margot. Symmetric ILP: Coloring and small integers. *Disc. Optim.*, 4(1):40–62, 2007. Mixed Integer Programming.
[29] F. Margot. Symmetry in integer linear programming. In *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*, pages 647–686. Springer Berlin Heidelberg, 2010.
[30] G. P. McCormick. Computability of global solutions to factorable nonconvex programs: Part i – convex underestimating problems. *Math. Program.*, 10(1):147–175, 1976.
[31] B. D. McKay and A. Piperno. Practical graph isomorphism, ii. *J. Symb. Comp.*, 60:94–112, 2014.
[32] R. Misener and C. A. Floudas. Global optimization of mixed-integer quadratically-constrained quadratic programs (miqcp) through piecewise-linear and edge-concave relaxations. *Math. Program.*, 136(1):155–182, 2012.
[33] R. Misener and C. A. Floudas. GloMIQO: Global mixed-integer quadratic optimizer. *J. Glob. Optim.*, 57(1):3–50, 2013.
[34] J. Ostrowski, J. Linderoth, F. Rossi, and S. Smiriglio. Constraint orbital branching. In *Proceedings of the Integer Programming and Combinatorial Optimization: 13th International Conference, IPCO 2008 Bertinoro, Italy, 2008.*, pages 225–239. Springer-Verlag, 2008.
[35] J. Ostrowski, A. Vannelli, and M. F. Anjos. Symmetry in scheduling problems. In *Cahier du GERAD G-2010-69, GERAD, QC, Canada*, 2010.
[36] J. Ostrowski, J. Linderoth, F. Rossi, and S. Smiriglio. Orbital branching. *Math. Program.*, 126(1):147–178, 2011.
[37] J. Ostrowski, M. F. Anjos, and A. Vannelli. Modified orbital branching for structured symmetry with an application to unit commitment. *Math. Program.*, 150(1):99–129, 2015.
[38] M. E. Pfetsch and T. Rehn. A computational comparison of symmetry handling methods for mixed integer programs. technical report. Technical report, Optimization Online, 2015.
[39] J. F. Puget. Automatic detection of variable and value symmetries. In P. V. Beek, editor, *Proceedings of the Principles and Practice of Constraint Programming - CP 2005: 11th International Conference, Spain, 2005.*, pages 475–489. Springer Berlin Heidelberg, 2005.
[40] A. Ramani and I. L. Markov. Automatically exploiting symmetries in constraint programming. In *Proceedings of the Recent Advances in Constraints: Joint ERCIM/CoLogNet International Workshop on Constraint Solving and Constraint Logic Programming, CSCLP 2004, Switzerland, 2004.*, pages 98–112. Springer-Verlag, 2005.
[41] H. D. Sherali and J. C. Smith. Improving discrete model representations via symmetry considerations. *Manag. Sci.*, 47(10):1396–1407, 2001.
[42] E. M. B. Smith and C. C. Pantelides. A symbolic reformulation/spatial branch-and-bound algorithm for the global optimisation of nonconvex MINLPs. *Comput. Chem. Eng.*, 23(4):457–478, 1999.
[43] E. Specht. [http://www.packomania.com/](http://www.packomania.com/)