

Symmetry Detection for Quadratic Optimization Using Binary Layered Graphs

Authors:

Georgia Kouyialis, Xiaoyu Wang, Ruth Misener

Date Submitted: 2019-12-16

Keywords: quadratically-constrained quadratic optimization, quadratic optimization, symmetry

Abstract:

Symmetry in mathematical optimization may create multiple, equivalent solutions. In nonconvex optimization, symmetry can negatively affect algorithm performance, e.g., of branch-and-bound when symmetry induces many equivalent branches. This paper develops detection methods for symmetry groups in quadratically-constrained quadratic optimization problems. Representing the optimization problem with adjacency matrices, we use graph theory to transform the adjacency matrices into binary layered graphs. We enter the binary layered graphs into the software package nauty that generates important symmetric properties of the original problem. Symmetry pattern knowledge motivates a discretization pattern that we use to reduce computation time for an approximation of the point packing problem. This paper highlights the importance of detecting and classifying symmetry and shows that knowledge of this symmetry enables quick approximation of a highly symmetric optimization problem.

Record Type: Published Article

Submitted To: LAPSE (Living Archive for Process Systems Engineering)

Citation (overall record, always the latest version):

LAPSE:2019.1634

Citation (this specific file, latest version):

LAPSE:2019.1634-1

Citation (this specific file, this version):


LAPSE:2019.1634-1v1

DOI of Published Version: <https://doi.org/10.3390/pr7110838>

License: Creative Commons Attribution 4.0 International (CC BY 4.0)

Article

Symmetry Detection for Quadratic Optimization Using Binary Layered Graphs

Georgia Kouyialis [†], Xiaoyu Wang and Ruth Misener ^{* }Department of Computing, Imperial College London, London SW7 2AZ, UK;
g.kouyialis14@imperial.ac.uk (G.K.); xiao.wang10@imperial.ac.uk (X.W.)^{*} Correspondence: r.misener@imperial.ac.uk; Tel.: +44-207-594-8315[†] Current address: Schlumberger Cambridge Research.

Received: 16 May 2019; Accepted: 5 November 2019; Published: 9 November 2019



Abstract: Symmetry in mathematical optimization may create multiple, equivalent solutions. In nonconvex optimization, symmetry can negatively affect algorithm performance, e.g., of branch-and-bound when symmetry induces many equivalent branches. This paper develops detection methods for symmetry groups in quadratically-constrained quadratic optimization problems. Representing the optimization problem with adjacency matrices, we use graph theory to transform the adjacency matrices into binary layered graphs. We enter the binary layered graphs into the software package *nauty* that generates important symmetric properties of the original problem. Symmetry pattern knowledge motivates a discretization pattern that we use to reduce computation time for an approximation of the point packing problem. This paper highlights the importance of detecting and classifying symmetry and shows that knowledge of this symmetry enables quick approximation of a highly symmetric optimization problem.

Keywords: symmetry; quadratic optimization; quadratically-constrained quadratic optimization

1. Introduction

When the optimization variables can be permuted without changing the structure of the underlying optimization problem, we say that the *formulation group* of an optimization problem is *symmetric* [1,2]. For motivation, consider the circle packing problem illustrated in Figure 1 [3]. Given an integer $n > 0$, the circle packing problem asks: what is the largest radius r for which n non-overlapping circles can be placed in the unit square? Costa et al. [3] show that the formulation group, i.e., a subgroup of symmetry group generated by permuting variables and constraints, is isomorphic to a symmetry group created by permuting the variable indices and switching the two coordinates in a unit square ($C_2 \times S_n$). Solution methods for nonconvex optimization problems lacking symmetry-aware formulations and/or solution procedures may end up exploring all of these equivalent solutions. In other words, symmetry may cause classical optimization methods such as branch-and-bound to explore many unnecessary subtrees.

More generally, a number of authors have considered a range of symmetry detection methods, e.g., for constraint programming [4], integer programming [1,5–8], and mixed-integer nonlinear optimization [2]. These automatic symmetry detection methods can then be used to mitigate the computational difficulties caused by symmetries, e.g., with symmetry-breaking constraints [9–11], objective perturbation [12], specialized branching strategies [13,14], cutting planes [15,16], and extended formulations [17]. The recent computational comparison of Pfetsch and Rehn [18] indicates that these state-of-the-art symmetry handling methods expedite the solution process for the MIPLIB 2010 instances and additionally enable more instances to be solved in a time limit.

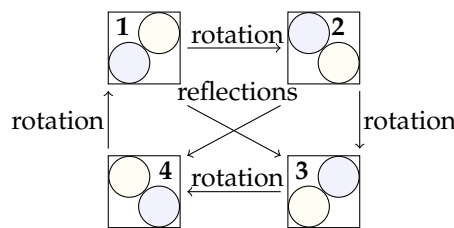


Figure 1. Given an integer $n > 0$, the circle packing problem asks: what is the largest radius r for which n non-overlapping circles can be placed in the unit square? Already for $n = 2$, there are four equivalent solutions [3]; these solutions are related to one another via rotations and reflections.

Researchers have also developed symmetry-handling methods for specific applications including covering design [19], circle packing [3,20], scheduling [21], transmission switching [22], unit commitment [23–26], and heat exchanger network synthesis [27,28]. As a concrete example of the type of contributions researchers have made, consider a job shop scheduling problem that minimizes makespan on two identical machines. Good scheduling formulations and/or solution procedures, e.g., Maravelias and Grossmann [29], Maravelias [30], and Mistry et al. [31], will implicitly exclude two of the three equivalent solutions diagrammed in Figure 2.

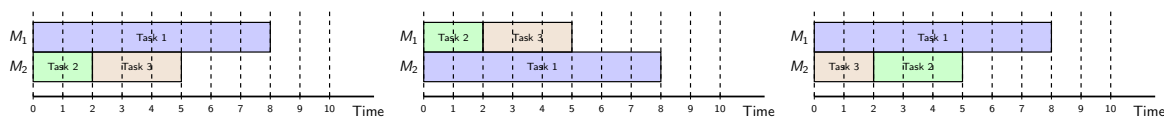


Figure 2. To observe symmetries that may arise in scheduling, consider a job shop scheduling problem that minimizes makespan on two identical machines. Lacking symmetry-aware formulations and/or solution procedures, a solution procedure may end up exploring all three of these equivalent solutions.

This paper develops detection methods for symmetry groups in quadratically-constrained quadratic optimization problems. Representing the optimization problem with adjacency matrices, we use graph theory to transform the adjacency matrices into binary layered graphs. We enter the binary layered graphs into the software package nauty that generates important symmetric properties of the original problem. Symmetry pattern knowledge motivates a discretization pattern that we use to reduce computation time for an approximation of the point packing problem.

2. Formulation Symmetry for Quadratically-Constrained Quadratic Optimization Problems

Consider the quadratically-constrained quadratic optimization problem (QCQP):

$$\begin{aligned}
 & \min_{x \in \mathbb{R}^n} f_0(x) \\
 & \text{s.t. } f_k(x) \leq 0 \quad \forall k = 1, \dots, m \\
 & \quad x_i \in [x_i^L, x_i^U] \quad \forall i = 1, \dots, n,
 \end{aligned} \tag{QCQP}$$

where:

$$f_k(x) = \sum_{i=1}^n \sum_{j=1}^n a_{ij}^k x_i x_j + \sum_{i=1}^n a_{i0}^k x_i + a_{00}^k \quad \forall k = 0, \dots, m, \tag{1}$$

with finite variable bounds $x_i^L, x_i^U \in \mathbb{R}, \forall i$ and coefficients $a_{ij}^k \in \mathbb{R}$ for $i, j \in \{0, \dots, n\}, k \in \{0, \dots, m\}$.

To represent symmetry in the QCQP formulation, consider S_n , the symmetric group of order n formed by the $n!$ possible permutation operations. The *formulation group* of QCQP, or \mathcal{G}_{QCQP} , is the set of variable index permutations that preserve the objective and constraint structure [2]. For a variable index permutation $\pi \in S_n$, we seek the constraint index permutations $\sigma \in S_n$ that maintain both the

objective value and the constraint structure on the feasible domain $dom(f)$ where $f = [f_1, \dots, f_m]$. More formally:

Definition 1 (Formulation group of QCQP).

$$\mathcal{G}_{QCQP} = \{ \pi \in S_n \mid \forall x \in dom(f_0) f_0(\pi x) = f_0(x) \wedge \forall x \in dom(f) \exists \sigma \in S_m (\sigma f(\pi x) = f(x)) \}.$$

Because $dom(f)$ may be nonconvex and difficult to compute, this paper considers a \mathcal{G}_{QCQP} restriction that enforces symmetry on the entire box bounds, i.e., we assume that $dom(f) = [x^L, x^U]$ for the purpose of computing \mathcal{G}_{QCQP} . The next subsections represent formulation symmetry in two ways: (i) expression graphs in Section 2.1 and (ii) tensors in Section 2.2. Representing formulation symmetry using expression graphs is due to Liberti [2] and the tensor representation is new to this paper.

2.1. Symmetry Detection with Expression Graphs

One option to compare two functions is to compare their expression trees, i.e., a directed acyclic graph representation of each function that incorporates the relevant operations, constants, and variables [2]. These expression tree models were first developed for mixed integer nonlinear optimization (MINLP) by Smith and Pantelides [32] and are common in most global MINLP solvers [33–39] and other MINLP-related software [40–42]. Figure 3 illustrates a simple example of an expression tree for $3x_1 + 2x_4^2 + 2x_2x_3$. A tree comparison algorithm may recursively compare two trees to determine equivalence [2]. More advanced implementations may detect equivalent but differently-formulated expressions, e.g., $(x_1 + x_2)^2$ versus $x_1^2 + 2x_1x_2 + x_2^2$.

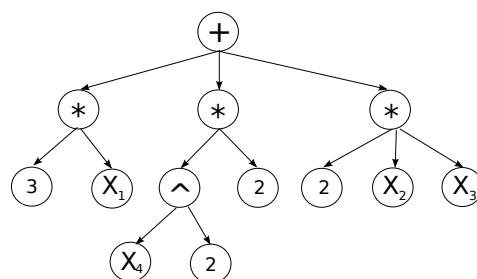


Figure 3. Example of an expression tree for $3x_1 + 2x_4^2 + 2x_2x_3$.

With a directed acyclic graph representation, Liberti [2] computes the formulation symmetry group using the graph isomorphism problem, i.e., a problem that can be solved using off-the-shelf software nauty [43]. Liberti [2] also proves how to map the automorphism group of a directed acyclic graph to the formulation group of the original MINLP.

2.2. Symmetry Detection with Tensors

As an alternative to the expression tree representation, Figure 4 illustrates that QCQP can be represented as a tensor: $A_{QCQP} \in \mathbb{R}^{(n+1) \times (n+1) \times (m+2n)}$. Each of the two dimensions $(n + 1)$ corresponds to a constant term and the variables. Each two-dimensional slice of the tensor corresponds to the constant, linear, and quadratic terms in a constraint. The first m slices correspond to Equation (1) and have entries a_{ij}^k . The next $2n$ slices correspond to the box constraints, i.e., $x_i \geq x_i^L$ and $x_i \leq x_i^U, \forall i$.

The formulation group of this representation is:

$$\mathcal{G}_{QCQP,T} = \{ \pi \in S_n \mid \forall x \in dom(f_0) f_0(\pi x) = f_0(x) \wedge \forall x \in dom(f) \exists \sigma \in S_m (A_{QCQP}(\pi, \pi, \sigma) = A_{QCQP}) \}.$$

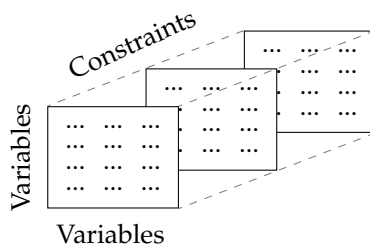


Figure 4. Tensor representation of the symmetry.

2.2.1. Sparse Tensor Representation

For a given tensor A_{QCQP} , consider a sparse representation, illustrated in Figure 5, that reduces the memory required to store the tensor. Instead of storing the entire tensor, we store arrays of length s where s is the number of nonzero entries in QCQP. The first array, $M = (M_1, \dots, M_s)$ stores all nonzero entries α_{ij}^k of QCQP. The next three arrays, $I = (I_1, \dots, I_s), J = (J_1, \dots, J_s), K = (K_1, \dots, K_s)$ represent the indices corresponding to the nonzero α_{ij}^k entries. The maximum size of s is $(n + 1)^2(m + 2n)$, but, in practice, most arrays will be significantly shorter.

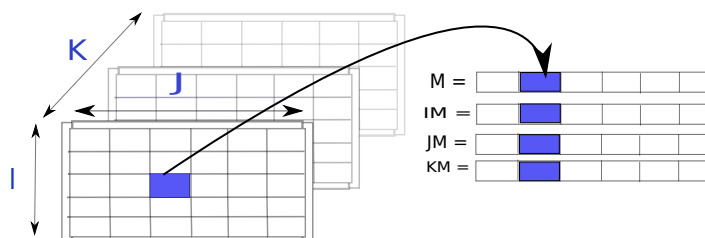


Figure 5. Sparse tensor representation models A_{QCQP} as 4 arrays with the nonzero entry α_{ij}^k in M and arrays I, J, K holding the index.

2.2.2. Converting Matrices to Edge-Labeled, Vertex-Colored Graphs

We convert the sparse tensor representation of A_{QCQP} into an edge-labeled, vertex-colored graph. Given the edge-labeled, vertex-colored graph, generating graph automorphisms to the original problem symmetries is well-known [1,5,44,45]. To construct the edge-labeled, vertex-colored graph, consider a graph $G = (V, E, c)$ corresponding to an instance M, I, J, K . The function $c : E \mapsto r$, for $r \in \{0, \dots, \ell - 1\}$ is an edge coloring where $\ell \in \mathbb{Z}^+$ is the number of different coefficients in M . Each unique element in M is stored in a vector $U \in R^\ell$. We also partition (color) the vertex set into four subsets: a set V_F representing the objective function, V_C nodes for the constraints, a constant node V_S , and V_R variable nodes. The automorphism definition prevents vertices from being mapped onto a vertex of a different color, so these colors prevent, for example variables becoming constraints. The equivalence relation is [2]:

$$\forall u, v \in V_p \ u \sim v \implies (u, v \in V_F \wedge \ell(u) = \ell(v)) \vee (u, v \in V_C \wedge \ell(u) = \ell(v)) \vee (u, v \in V_S \wedge \ell(u) = \ell(v)) \vee (u, v \in V_R \wedge \ell(u) = \ell(v)).$$

Figure 6 illustrates the edge-labeled, vertex-colored graph. Initially, the edge set is empty $E = \emptyset$. For $i = \{0, \dots, s\}$ where $s = |M|$, add an edge $v_{I_i}^{(r)}$ to $v_{K_i}^{(r)}$, i.e., from a vertex in the set that represents the constant element / variables to a vertex in the set of the objective function / constraints, with the relevant color. The graph construction incorporates edges between variable nodes V_R for the quadratic bilinear terms. For $i = \{0, \dots, s\}$:

- If $I_i = J_i$, i.e., a quadratic term, then $E = E \cup \{(v_{I_i}, v_{K_i})^r \cap \{(v_{I_i})^r\}\}$.
- else for bilinear term, $I_i \neq J_i$, then $E = E \cup \{(v_{I_i}, v_{K_i})^r \cap \{(v_{J_i}, v_{K_i})^r \cap \{(v_{I_i}, v_{J_i})^r\}\}$.

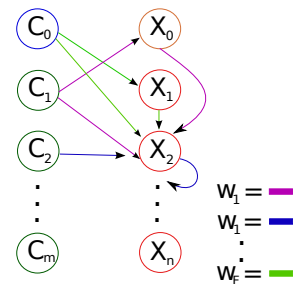


Figure 6. The tensor A_{QCQP} as an edge-labeled, vertex-colored graph.

3. Formulation Symmetry Detection via Binary Layered Graphs

The software nauty [43], which detects symmetry, accepts vertex-colored graphs but does not accept the Section 2.2.2 edge-labeled, vertex-colored graphs. Thus, we associate edge colors with layers in a graph and transform the edge-labeled, vertex-colored graph into a vertex-colored graph. Since the transformation from an edge-labeled, vertex-colored graph to a vertex-colored graph is isomorphic [43], the transformation does not lose anything. Using the resulting *binary layered graphs*, we generate the automorphism group and find symmetry in the original QCQP.

To convert a graph $G = (V, E, c)$ with ℓ colors into an ℓ -layered graph [43], we replace each vertex $v_j \in V$ with a fixed connected graph of ℓ vertices $v_j^{(0)}, \dots, v_j^{(\ell-1)}$. If an edge $(v_j, v_{j'})$ has color r , add an edge from $v_j^{(r)}$ to $v_{j'}^{(r)}$. Finally, we partition the vertices by the superscripts, $V_r = \{v_0^{(r)}, \dots, v_{n-1}^{(r)}\}$. Alternatively, a binary representation avoids too many layers in G when the number of colors is large.

Definition 2 (Binary Layered Graph). Let $\ell \in \mathbb{Z}^+$ be the number of edge labels of G . A binary layered graph is a vertex-colored graph where the number of layers $L = \lceil \log_2(\ell + 1) \rceil$ matches a binary representation.

We assign a unique positive integer $\mu(z)$ to each $z \in \mathbf{U}$ and map edge labels $\mu(z)$ to a binary representation that switches on/off parameters c_t to represent the edge colors as layers. If $c_t = 1$, add a new edge from v_i^t to v_j^t for every $c_t \in \{c_1, \dots, c_{L-1}\}$:

$$\mu(z) = 2^{L-1} \cdot c_{L-1}(z) + 2^{L-2} \cdot c_{L-2}(z) + \dots + 2^0 \cdot c_0(z), \text{ for } c_t \in \{0, 1\}, t = \{0, \dots, L-1\}. \quad (2)$$

Figure 7 illustrates the resulting binary labeled graph with its $L = \lceil \log_2(\ell + 1) \rceil + 2$ layers. There are vertices for the objective function and each constraint and layers of copies of these constraints (connected with vertical edges). The horizontal edges encode the problem coefficients. On the upper part of Figure 7, there are vertices for a constant element and each variable and a layer of variable copies (connected with vertical edges). Here, the horizontal edges and loops distinguish the linear and bilinear terms. Algorithms 1 and 2 summarize computing the vertex and edge sets, respectively.

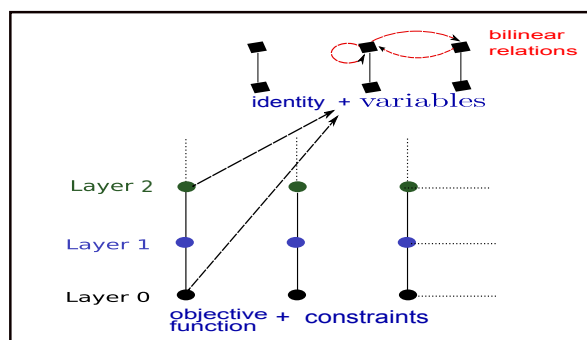


Figure 7. Binary layered graph representation of QCQP using the tensor representation A_{QCQP} .

Algorithm 1 Algorithm constructing the vertex set

```

1: procedure G=(V, E)
2:   V ← ∅, Vs ⊂ V ← ∅ ∀ s, E ← ∅
3:   L ∈ ℤ, L = ⌈log2(|U| + 1)⌉ + 2
4:   for s = 0 → L - 3 do
5:     for k = 0 → m do
6:       Vs ← Vs ∪ {vk(s)}
7:     end for
8:     V ← V ∪ Vs
9:   end for
10:  for s = L - 2, L - 1 do
11:    for i = 0 → n do
12:      Vs ← Vs ∪ {vi(s)}
13:    end for
14:    V ← V ∪ Vs
15:  end for
16:  return G
17: end procedure

```

▷ Define $L \in \mathbb{Z}^+$ the number of layers
 ▷ Partition of vertices representing the constraints
 ▷ Copies of vertices representing the constraints
 ▷ Partition of vertices representing the variables
 ▷ Copies of vertices representing the variables

Algorithm 2 Algorithm constructing the edge set

```

1: procedure G=(V, E)
2:   V ← V
3:   E ← ∅
4:   for s = 0 → L - 4 do
5:     for k = 0...m do
6:       E = E ∪ E ∪ (vk(s), vk(s+1))
7:     end for
8:   end for
9:
10:  for j = 0...n do
11:    E = E ∪ (vj(L-2), vj(L-1))
12:  end for
13:  for F = 0, ... N - 1 do
14:    if IM(F) = JM(F) then
15:      E = E ∪ (vIM(F)}(L-1), vIM(F)}(L-1))
16:    else
17:      if IM(F) < JM(F) then
18:        E = E ∪ (vIM(F)}(L-1), vJM(F)}(L-1))
19:      else
20:        E = E ∪ ∅
21:      end if
22:    end if
23:  end for
24:  for F = 0...N - 1 do
25:    for k = 0...m do
26:      if KM(F) = k then
27:        if IM(F) = 0 then
28:          E = E ∪ (vk(s), vJM(F)}(L-2))
29:        else
30:          E = E ∪ (vIM(F)}(L-1), vJM(F)}(L-1))
31:        end if
32:      end if
33:    end for
34:  end for
35:  return G
36: end procedure

```

▷ 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

4. Numerical Discussion and Comparison to the State-of-the-Art

The following example incorporates the algorithms proposed in this paper. We construct the binary labeled graph and then enter it into nauty through the dreadnaut command line interface:

$$\begin{aligned} \max_{x_1, x_2, x_3, x_4 \in [0,1]} \quad & 3x_1 + 3x_4 + 2x_2x_3 && (c_0), \\ & x_2 + x_1^2 + 1 \leq 0 && (c_1), \\ & x_3 + x_4^2 + 1 \leq 0 && (c_2), \\ & x_2 + x_3 + 1 \leq 0 && (c_3). \end{aligned}$$

The optimization problem has sparse matrix representation: $M = (3, 3, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1)$, $I = (0, 0, 2, 0, 0, 1, 0, 0, 4, 0, 0, 0)$, $J = (1, 4, 3, 0, 2, 1, 0, 3, 4, 0, 2, 3)$, $K = (0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3)$, vector of unique elements $U = [1, 2, 3]$, and $L = \lceil \log_2 4 \rceil = 2$ layers.

Equation (2) computes the binary representation of each unique element, e.g., $3 = 2^1 + 2^0$ indicates that there is an edge between vertices on layer zero and another edge between the same vertices on layer 1. The graph consists of four layers and $|V| = 18$, one associated with a constant element and one with the objective function and the rest for the problem variables and constraints. The left-hand side of Figure 8 illustrates the graph representation. Nauty generates permutations: $\pi = (1, 2)(5, 6)(9, 12)(10, 11)(14, 17)(15, 16)$. To see how these Nauty-generated permutations usefully explain the symmetry properties of the entire problem, observe: (i) Permutations $(1, 2)(5, 6)$, as shown in Figure 8, permute the constraints c_1, c_2 and (ii) Permutations $(9, 12)(10, 11)$ are associated with the variables x_1, x_4 and x_2, x_3 with $(14, 17)(15, 16)$ their copies. These permutations therefore allow us to automatically calculate the formulation group $\mathcal{G} = (x_1x_4)(x_2x_3)$.

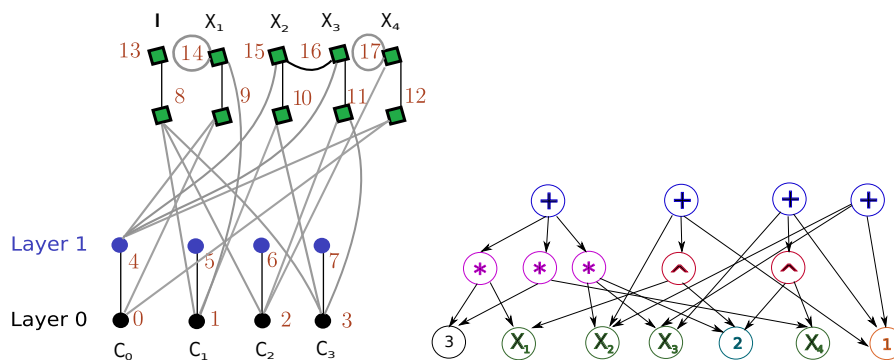


Figure 8. Illustration of the example problem using the binary labeled graph representation (left) and the directed acyclic graph representation (right).

The right-hand side of Figure 8 uses Section 2.1 to develop a directed acyclic graph representation for the same problem. The graph colors represent the vertex partitioning that enables node exchanges. In this case, the directed acyclic graph representation uses a smaller number of vertices and edges than the tensor-based representation. However, the representations generate the same formulation group.

Comparison. To evaluate the trade-offs between the tensor and directed acyclic graph representations, observe that both methods will search for the same formulation group symmetries. However, the tensor representation may be especially useful when working with problems with many differently-valued coefficients, i.e., the logarithmic number of layers may reduce the number of nodes. The function assigning integer values to the problem coefficients lets us work not only with 0–1 coefficients, but also with any other value.

5. Exploiting Symmetry in the Point Packing Problem

Once symmetry has been detected, we can use our knowledge of the symmetry to mitigate the computational difficulties caused by symmetry. Here, we focus on solving the point packing problem [46–49]. The point packing problem concerns packing n points to a unit square. The aim is to maximize the in-between distance between any two points:

$$\begin{aligned} \max \quad & \theta \\ \text{subject to} \quad & (x_i - x_j)^2 + (y_i - y_j)^2 \geq \theta^2 \quad 1 \leq i < j \leq n, \\ & 0 \leq x_i \leq 1, 0 \leq y_i \leq 1 \quad 1 \leq i \leq n, \end{aligned}$$

where θ denotes the minimum distance between any two points. To approximate this problem, consider a grid approach that approximates the optimal solution by adding grid lines to the unit square and forcing the points to be placed only to the vertices generated by these grid lines. Note that the point packing problem has significant applications, e.g., in placing mobile phone towers.

For n points, there will be at most a $k^* \times k^*$ grid, where k^* is the smallest number whose square is the least integer that is greater than n , i.e., $(k^* - 1)^2 < n$ and $k^{*2} \geq n$. In other words, we add at most $2k^*$ grid lines. On this $k^* \times k^*$ grid, points will occupy most of the vertexes and the unit length of spacing has been maximized. This approach, unfortunately, has the potential of missing the optimal solution. Consider fitting six points to the square. The most fitting grid is 3×3 and the optimal θ we achieve is 0.5. However, the optimal solution of 0.6009 is achieved by the arrangement shown in Figure 9, which is not available on a 3×3 grid. Although an approximation, k^* is still a useful pruning tool, e.g., points need to be at least $\frac{1}{k^*-1}$ away from any other point.

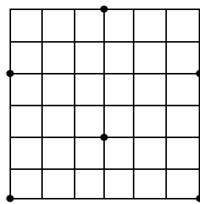


Figure 9. Optimal arrangement of six points.

Exhaustive Search and 2D Symmetry Removal

First, consider Algorithm 3, an exhaustive search method. To break the symmetries, start by calculating how many grids can possibly have points and the upper limit on the number of points on any occupied grid lines. Once calculated, the complete set of all possible combinations on the x -axis is calculated. For example, on a 5×5 grid, one possible x -coordinate setup is $(2, 0, 1, 0, 2)$. If we label the five horizontal grid lines from 1 to 5, this setup means that there are two points on grid 1, no points on grid 2, one point on grid 3, no points on grid 4 and two points on grid 5. To remove x -axis symmetries, we consider reflections as a duplication, i.e., only one of $(1, 0, 2, 0, 2)$ and $(2, 0, 2, 0, 1)$ are considered.

Algorithm 3 Algorithm 1—Exhaustive Search

Input: number of points n , number of grids k , number of occupied grids m .

Output: The optimal solution d^*

- 1: **Step1** Calculate the upper bound of number of points on each occupied grid u
 - 2: **Step2** Generate complete set of combinations of x -coordinate
 - 3: **Step3** Symmetries removal on x -coordinates
 - 4: **Step4** Generate all y -coordinate sets based on binomial coefficient
 - 5: **Step5** Y -coordinates pruning
-

Algorithm 3 considers the unique x -axis combinations. The y -axis, i.e., the horizontal grid lines, should preserve the same characteristics. Thus, in the improved Algorithm 4, we generate the set O of possible point arrangement such as $(2, 0, 1, 0, 2)$, and this is applied to both horizontal and vertical grid lines; in other words, we now consider the product $O \times O$ to give the exact coordinates of all n points.

Algorithm 4 Algorithm 2—2D Symmetry Removal

Input: number of points n , number of grids k , number of occupied grids m .

Output: The optimal solution d^*

- 1: **Step1** Calculate the upper bound of number of points on each occupied grid u
 - 2: **Step2** Generate complete set of combinations of x -coordinate
 - 3: **Step3** Symmetries removal on x -coordinates
 - 4: **Step4** Use the x -coordinate set to determine y -coordinate
 - 5: **Step5** Y -coordinates pruning
-

Example 1. Consider for five points on a 5×5 grid where we have O as $\{O_1 = (1, 0, 2, 0, 2), O_2 = (1, 2, 0, 0, 2), O_3 = (1, 0, 0, 2, 2), O_4 = (2, 1, 0, 0, 2), O_5 = (2, 0, 1, 0, 2)\}$. The finalized set of full coordinates would contain 25 elements where the product of O with itself is taken. One particular point setup generated by this approach is for example, $O_3 \times O_4$. If we call the vertical grids as V_1 to V_5 , respectively, and the horizontal grids as H_1 to H_5 , respectively, $O_3 \times O_4$ would mean that we have the following point locations:

1. One point on V_1 , 2 points on V_4 and 2 points on V_5 . This is from O_3 .
2. Two points on H_1 , 1 point on H_2 and 2 points on H_5 . This is from O_4 .

For $O_3 \times O_4$, we would have five complete point setups, as shown in Figure 10. In the last setup, we have also added the labels for grids to match what we defined earlier. This diagram contains all possible arrangements of points under this particular orbit partitioning.

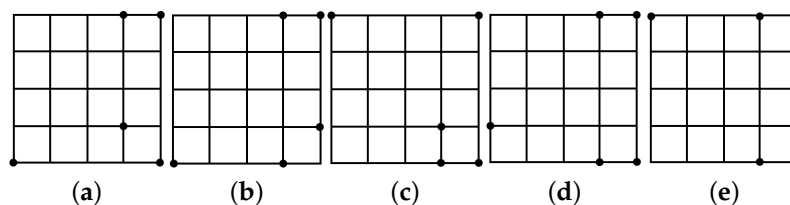


Figure 10. Five point setups for $O_3 \times O_4$. (a) point setup1; (b) point setup2; (c) point setup3; (d) point setup4; (e) point setup5.

To improve further, we can implement pruning mechanisms such as using the bound $\frac{1}{k^x-1}$ to prune the non-optimal setups. In this particular example, we would have pruned all five setups as none of them satisfy the 0.5 bound from the most fitting grid. This means that $O_3 \times O_4$ is not the optimal point setup for five points on a 5×5 grid.

6. Results and Comparisons

Both algorithms have been implemented and run on the same devices (HP EliteDesk 800 G2 TWR Intel Core i7-6700 3.4 GHz) to provide effective comparisons. Figure 11 shows the experimental results of 2D Symmetry Removal for packing 6 points. We eliminated the result from Exhaustive Search as it is clear that 2D Symmetry Removal outperforms Exhaustive Search in terms of run-time. We notice that m , the number of occupied grids (in the diagram, this corresponds to the occupied vertical grids), has an impact on the run-time. To a reasonable extent, the larger the m , the more choices we have regarding where we place the points so it in general takes longer time to compute. Although our strategies effectively convert the point packing QCQP into a mixed-integer linear optimization problems, we could have alternatively designed a branch-and-bound algorithm that is symmetry aware.

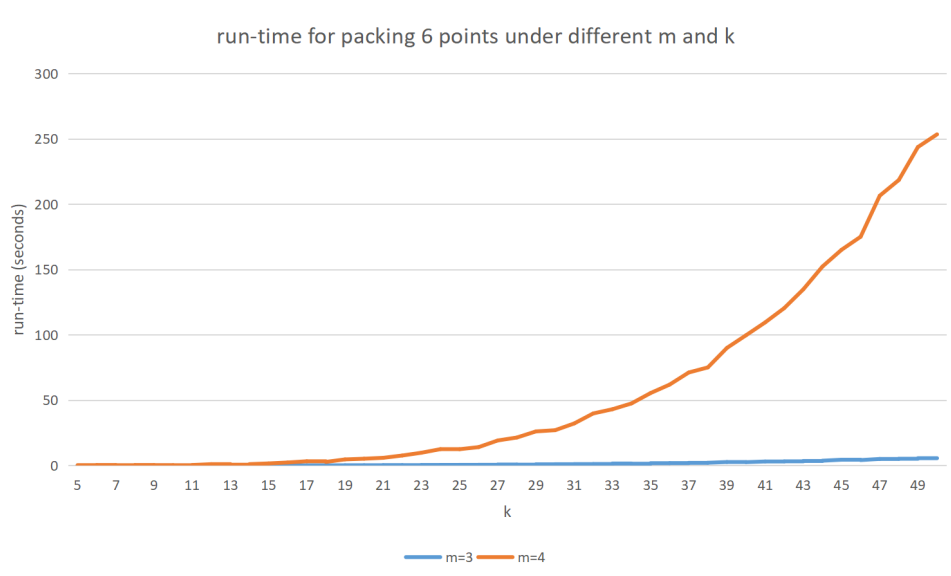


Figure 11. Run-time for six points on different grids with different number of occupied grids. The line $m = 4$ is above the line $m = 3$.

7. Conclusions

This paper has explored alternative representations for finding symmetry in formulation groups of a quadratically-constrained optimization problem. We also show that, after knowing the symmetry, we can design significantly better methods to solve the optimization problems. The contributions in this paper are relevant to industrial problems that contain a point packing element [50–52].

Author Contributions: Conceptualization, G.K. and R.M.; methodology, G.K.; validation, X.W.; writing, G.K., X.W., and R.M.; supervision, R.M.

Funding: This work was funded by an Engineering & Physical Sciences Research Council (EPSRC) Research Fellowship to R.M. (Grant No. EP/P016871/1) and an EPSRC DTP to G.K.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

MINLP Mixed-integer nonlinear optimization
 QCQP Quadratically-constrained quadratic optimization

References

1. Margot, F. Symmetry in Integer Linear Programming. In *50 Years of Integer Programming 1958–2008: From the Early Years to the State-of-the-Art*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 647–686.
2. Liberti, L. Reformulations in mathematical programming: Automatic symmetry detection and exploitation. *Math. Program.* **2012**, *131*, 273–304. [[CrossRef](#)]
3. Costa, A.; Hansen, P.; Liberti, L. On the impact of symmetry-breaking constraints on spatial Branch-and-Bound for circle packing in a square. *Discret. Appl. Math.* **2013**, *161*, 96–106. [[CrossRef](#)]
4. Puget, J.F. Automatic Detection of Variable and Value Symmetries. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming—CP 2005*, Sitges, Spain, 1–5 October 2005; van Beek, P., Ed.; Springer: Berlin/Heidelberg, Germany, 2005; pp. 475–489.
5. Salvagnin, D. A Dominance Procedure for Integer Programming. Master’s Thesis, University of Padua, Padua, Italy, 2005.

6. Berthold, T.; Pfetsch, M. Detecting Orbitopal Symmetries. In Proceedings of the Annual International Conference of the German Operations Research Society (GOR), Augsburg, Germany, 3–5 September 2008; Springer: Berlin/Heidelberg, Germany, 2009; pp. 433–438.
7. Bremner, D.; Dutour Sikirić, M.; Pasechnik, D.V.; Rehn, T.; Schürmann, A. Computing symmetry groups of polyhedra. *LMS J. Comput. Math.* **2014**, *17*, 565–581. [[CrossRef](#)]
8. Knueven, B.; Ostrowski, J.; Pokutta, S. Detecting almost symmetries of graphs. *Math. Program. Comput.* **2018**, *10*, 143–185. [[CrossRef](#)]
9. Sherali, H.D.; Smith, J.C. Improving Discrete Model Representations via Symmetry Considerations. *Manag. Sci.* **2001**, *47*, 1396–1407. [[CrossRef](#)]
10. Liberti, L. Automatic Generation of Symmetry-Breaking Constraints. In *Combinatorial Optimization and Applications*; Yang, B., Du, D.Z., Wang, C.A., Eds.; Springer: Berlin/Heidelberg, Germany, 2008; pp. 328–338.
11. Liberti, L.; Ostrowski, J. Stabilizer-based symmetry breaking constraints for mathematical programs. *J. Glob. Optim.* **2014**, *60*, 183–194. [[CrossRef](#)]
12. Ghoniem, A.; Sherali, H.D. Defeating symmetry in combinatorial optimization via objective perturbations and hierarchical constraints. *IIE Trans.* **2011**, *43*, 575–588. [[CrossRef](#)]
13. Ostrowski, J.; Linderoth, J.; Rossi, F.; Smriglio, S. Constraint Orbital Branching. In Proceedings of the 13th International Conference on Integer Programming and Combinatorial Optimization IPCO, Bertinoro, Italy, 26–28 May 2008; pp. 225–239.
14. Ostrowski, J.; Linderoth, J.; Rossi, F.; Smiriglio, S. Orbital branching. *Math. Program.* **2011**, *126*, 147–178. [[CrossRef](#)]
15. Margot, F. Pruning by isomorphism in branch-and-cut. *Math. Program.* **2002**, *94*, 71–90. [[CrossRef](#)]
16. Kaibel, V.; Peinhardt, M.; Pfetsch, M.E. Orbitopal fixing. *Discret. Optim.* **2011**, *8*, 595–610. [[CrossRef](#)]
17. Faenza, Y.; Kaibel, V. Extended Formulations for Packing and Partitioning Orbitopes. *Math. Oper. Res.* **2009**, *34*, 686–697. [[CrossRef](#)]
18. Pfetsch, M.E.; Rehn, T. A computational comparison of symmetry handling methods for mixed integer programs. *Math. Program. Comput.* **2019**, *11*, 37–93. [[CrossRef](#)]
19. Margot, F. Small covering designs by branch-and-cut. *Math. Program.* **2003**, *94*, 207–220. [[CrossRef](#)]
20. Costa, A.; Liberti, L.; Hansen, P. Formulation symmetries in circle packing. *Electron. Notes Discret. Math.* **2010**, *36*, 1303–1310. [[CrossRef](#)]
21. Ostrowski, J.; Vannelli, A.; Anjos, M.F. *Symmetry in Scheduling Problems*; Cahier du GERAD G-2010-69; GERAD: Montreal, QC, Canada, 2010.
22. Ostrowski, J.; Wang, J.; Liu, C. Exploiting Symmetry in Transmission Lines for Transmission Switching. *IEEE Trans. Power Syst.* **2012**, *27*, 1708–1709. [[CrossRef](#)]
23. Ostrowski, J.; Wang, J. Network reduction in the Transmission-Constrained Unit Commitment problem. *Comput. Ind. Eng.* **2012**, *63*, 702–707. [[CrossRef](#)]
24. Ostrowski, J.; Anjos, M.F.; Vannelli, A. Modified orbital branching for structured symmetry with an application to unit commitment. *Math. Program.* **2015**, *150*, 99–129. [[CrossRef](#)]
25. Lima, R.M.; Novais, A.Q. Symmetry breaking in MILP formulations for Unit Commitment problems. *Comput. Chem. Eng.* **2016**, *85*, 162–176. [[CrossRef](#)]
26. Knueven, B.; Ostrowski, J.; Wang, J. The Ramping Polytope and Cut Generation for the Unit Commitment Problem. *INFORMS J. Comput.* **2018**, *30*, 739–749. [[CrossRef](#)]
27. Kouyialis, G.; Misener, R. Detecting Symmetry in Designing Heat Exchanger Networks. In Proceedings of the International Conference of Foundations of Computer-Aided Process Operations-FOCAPO/CPC, Tucson, AZ, USA, 8–12 January 2017.
28. Letsios, D.; Kouyialis, G.; Misener, R. Heuristics with performance guarantees for the minimum number of matches problem in heat recovery network design. *Comput. Chem. Eng.* **2018**, *113*, 57–85. [[CrossRef](#)]
29. Maravelias, C.T.; Grossmann, I.E. A hybrid MILP/CP decomposition approach for the continuous time scheduling of multipurpose batch plants. *Comput. Chem. Eng.* **2004**, *28*, 1921–1949. [[CrossRef](#)]
30. Maravelias, C.T. Mixed-Time Representation for State-Task Network Models. *Ind. Eng. Chem. Res.* **2005**, *44*, 9129–9145. [[CrossRef](#)]
31. Mistry, M.; Callia D’Iddio, A.; Huth, M.; Misener, R. Satisfiability modulo theories for process systems engineering. *Comput. Chem. Eng.* **2018**, *113*, 98–114. [[CrossRef](#)]

32. Smith, E.M.B.; Pantelides, C.C. A symbolic reformulation/spatial branch-and-bound algorithm for the global optimisation of nonconvex MINLPs. *Comput. Chem. Eng.* **1999**, *23*, 457–478. [[CrossRef](#)]
33. Tawarmalani, M.; Sahinidis, N.V. A polyhedral branch-and-cut approach to global optimization. *Math. Program.* **2005**, *103*, 225–249. [[CrossRef](#)]
34. Belotti, P.; Lee, J.; Liberti, L.; Margot, F.; Wachter, A. Branching and Bounds Tightening techniques for Non-convex MINLP. *Optim. Methods Softw.* **2009**, *24*, 597–634. [[CrossRef](#)]
35. Youdong, L.; Linus, S. The global solver in the LINDO API. *Optim. Methods Softw.* **2009**, *24*, 657–668.
36. Misener, R.; Floudas, C.A. ANTIGONE: Algorithms for coNTinuous/Integer Global Optimization of Nonlinear Equations. *J. Glob. Optim.* **2014**, *59*, 503–526. [[CrossRef](#)]
37. Vigerske, S. Decomposition in Multistage Stochastic Programming and a Constraint Integer Programming Approach to Mixed-Integer Nonlinear Programming. Ph.D. Thesis, Humboldt-Universität zu Berlin, Berlin, Germany, 2013.
38. Mahajan, A.; Leyffer, S.; Linderoth, J.; Luedtke, J.; Munson, T. Minotaur: A mixed-integer nonlinear optimization toolkit. *Optim. Online* **2017**, 6275.
39. Boukouvala, F.; Misener, R.; Floudas, C.A. Global optimization advances in Mixed-Integer Nonlinear Programming, MINLP, and Constrained Derivative-Free Optimization, CDFO. *Eur. J. Oper. Res.* **2016**, *252*, 701–727. [[CrossRef](#)]
40. Fourer, R.; Maheshwari, C.; Neumaier, A.; Orban, D.; Schichl, H. Convexity and concavity detection in computational graphs: Tree walks for convexity assessment. *INFORMS J. Comput.* **2010**, *22*, 26–43. [[CrossRef](#)]
41. Hart, W.E.; Laird, C.; Watson, J.; Woodruff, D.L. Pyomo: Modeling and solving mathematical programs in python. *Math. Program. Comput.* **2011**, *3*, 219–260. [[CrossRef](#)]
42. Cecon, F.; Siirola, J.D.; Misener, R. SUSPECT: MINLP special structure detector for Pyomo. *Optim. Lett.* **2019**. [[CrossRef](#)]
43. McKay, B.D.; Piperno, A. Practical graph isomorphism, II. *J. Symb. Comput.* **2014**, *60*, 94–112. [[CrossRef](#)]
44. Ramani, A.; Aloul, F.; Markov, I.; Sakallah, K.A. Breaking instance-independent symmetries in exact graph coloring. *J. Artif. Intell. Res.* **2006**, *1*, 324–329. [[CrossRef](#)]
45. Ramani, A.; Markov, I.L. Automatically Exploiting Symmetries in Constraint Programming. In *Recent Advances in Constraints*; Faltings, B.V., Petcu, A., Fages, F., Rossi, F., Eds.; Springer: Berlin/Heidelberg, Germany, 2005; pp. 98–112.
46. Anstreicher, K.M. Semidefinite programming versus the reformulationlinearization technique for nonconvex quadratically constrained quadratic programming. *J. Glob. Optim.* **2009**, *43*, 471–484. [[CrossRef](#)]
47. Misener, R.; Floudas, C.A. Global optimization of mixed-integer quadratically-constrained quadratic programs (MIQCQP) through piecewise-linear and edge-concave relaxations. *Math. Program.* **2012**, *136*, 155–182. [[CrossRef](#)]
48. Misener, R.; Floudas, C.A. GloMIQO: Global mixed-integer quadratic optimizer. *J. Glob. Optim.* **2013**, *57*, 3–50. [[CrossRef](#)]
49. Furini, F.; Traversi, E.; Belotti, P.; Frangioni, A.; Gleixner, A.; Gould, N.; Liberti, L.; Lodi, A.; Misener, R.; Mittelman, H. QPLIB: A library of quadratic programming instances. *Math. Program. Comput.* **2019**, *11*, 237–265. [[CrossRef](#)]
50. Jones, D.R. A fully general, exact algorithm for nesting irregular shapes. *J. Glob. Optim.* **2014**, *59*, 367–404. [[CrossRef](#)]
51. Misener, R.; Smadbeck, J.B.; Floudas, C.A. Dynamically generated cutting planes for mixed-integer quadratically constrained quadratic programs and their incorporation into GloMIQO 2. *Optim. Methods Softw.* **2015**, *30*, 215–249. [[CrossRef](#)]
52. Wang, A.; Hanselman, C.L.; Gounaris, C.E. A customized branch-and-bound approach for irregular shape nesting. *J. Glob. Optim.* **2018**, *71*, 935–955. [[CrossRef](#)]

