EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics and Computer Science
Combinatorial Optimization

Symmetry in the graph coloring problem

Bachelor's Thesis

P.A. Roefs

Supervisor:
Dr. rer. nat. C. Hojny

Eindhoven, 19 June 2023

Abstract

The graph coloring problem is an NP-hard problem [Garey et al., 1974]. Therefore, this problem is often solved by using an integer program. However, various symmetries that occur when solving this integer program result in an unnecessarily long solving time. By applying various symmetry breaking methods the solving time can be reduced. In this report various methods are explained and tested. The foundation of these methods consists of the orbitopal fixing algorithm Kaibel et al. [2011] and the 0-neighbor-sub-symmetry [Bendotti et al., 2020].

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Creating schedules is one of the tasks that need to be performed often within our society. For example, each high school must make a schedule for each of their classes. These schedules must satisfy certain restrictions. The class mathematics and the class physics can not be scheduled in the same time slot if they share a student, since the student must be able to attend both the physics and the mathematics class. Furthermore, we would like to use as few time slots as possible. These kind of problems can be translated into the *graph coloring problem*, which is a fundamental problem in graph theory.

The graph coloring problem is to determine the *chromatic number* of a graph. That is, finding the minimum number of colors needed to color a graph such that two vertices connected by an edge are not colored by the same color.

The high school scheduling problem can be translated into the graph coloring problem by representing the classes as vertices. We connect two vertices with an edge if the classes corresponding to these vertices may not be scheduled in the same time slot. The time slots are represented by the colors. If we now find a coloring with the minimum number of colors, we have solved our scheduling problem. Each class can be scheduled in the time slot that corresponds to the color that is assigned to the vertex that represents the class. Note that in this way we create a schedule with as few time slots as possible.

For small graphs the graph coloring problem can be easily solved by hand. However, for larger graphs this becomes more complex. The graph coloring problem is known to be NP-hard for arbitrary graphs [Garey et al., 1974]. Therefore, (binary) integer programming is used to solve this problem. The introduction of the book by Conforti et al. [2014] gives a clear introduction to integer programming. In short, a binary integer program tries to find the optimal values for a set of variables in $\{0, 1\}$ that must suffice certain constraints, such that a given objective function is optimal. In case of the graph coloring problem the objective is to minimize the number of colors in a graph coloring.

When the binary integer program can not be solved immediately, the problem is split up into two cases by setting a variable to 0 (first case) or to 1 (second case). Then, for these two problems we try to find the solution. This method is called *branch and bound*. After splitting up the problem into two cases, it is still possible that we are not able to solve these sub-problems. Hence, we split up the sub-problems again. By splitting up the problem multiple times we create a branch and bound tree, where each node of the tree represents a (sub-)problem, where some variables might be fixed. In case of the graph coloring problem, in each node of the branch and bound tree certain variables might be fixed, which indicate whether a vertex is with certainty colored or not colored by a color k.

Note that there are multiple ways to color a graph with the minimum number of colors needed. In Figure 1.1 we see a graph for which we need at least 3 colors to color the graph. Figure 1.1 illustrates two possible colorings with 3 colors. However, these graph colorings are not that different from each other. We simply switched the colors around. Green became red, red became orange and orange became green. This is what we call a symmetry.

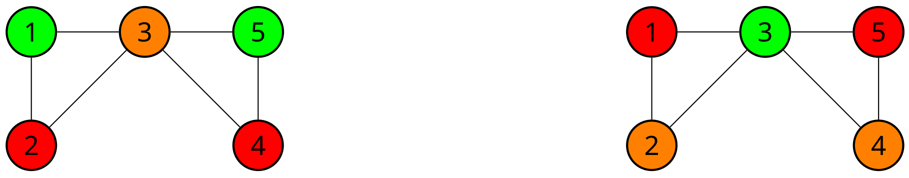

Figure 1.1: Symmetry of a graph coloring

When trying to determine the *chromatic number* of a graph using integer programming, many of such symmetries are encountered. Note that we do not need to find all symmetries of a graph coloring, since these symmetries will have the same number of colors. Therefore, these symmetries do not have an influence on the minimum number of colors in the graph coloring. However, due to these symmetries the branch and bound tree can become unnecessary large which results in a longer solving time of the binary integer program.

Next to the symmetry that was introduced earlier, which was a symmetry of the original problem, we can also encounter sub-symmetries in the sub-problems that we find in the branch and bound tree which are not symmetries of the original problem. One of such symmetries is the *0-neighbor sub-symmetry*, which will be discussed later in detail [Bendotti et al., 2020].

The goal of this project is to create algorithms that find these kinds of (sub)-symmetries in an efficient way. After finding such (sub)-symmetries, we would like to be able to handle these symmetries, such that the solving process of the binary integer program is more efficient. This is done by cutting off nodes of the branch and bound tree that will not influence the optimal value of the original binary integer program.

Chapter 2

Theory & Definitions

In this chapter, necessary theory and definitions will be introduced. We will introduce the integer program that is used to solve the graph coloring problem (Section 2.1). Next to that, we introduce some definitions and lemmas that are useful later on (Section 2.2). Furthermore, we will discuss symmetries of a graph coloring and their impact on the efficiency of solving the integer program (Section 2.3). We will also discuss various methods that tackle these symmetries (Section 2.4).

2.1 Graph coloring problem as integer program

A well known problem within graph theory is the graph coloring problem. If we are given a simple graph $G = (V, E)$, which is a graph with no directed edges and no self loops, than a graph coloring is an assignment of colors to vertices $v \in V$ in such a way that for any two vertices that share an edge, the assigned colors are different. The graph coloring problem is then to find the minimum number of colors needed; this number is also called the chromatic number. Hence, a k-coloring is a map $C : V \to \{1, \dots, k\}$ such that $C(v) \neq C(w)$ for all $\{v, w\} \in E$. The graph coloring problem is known to be NP-hard for arbitrary graphs [Garey et al., 1974].

In order to find the chromatic number of a graph, the following (binary) integer program can be used:

$$\min \sum_{j=1}^{K} y_j \tag{2.1}$$

$$x_{vj} + x_{wj} \leq y_j \quad \{v, w\} \in E, j \in \{1, \dots, K\} \tag{2.2}$$

$$\sum_{j=1}^{K} x_{vj} = 1 \quad v \in V \tag{2.3}$$

$$x_{vj} \in \{0, 1\} \quad v \in V, j \in \{1, \dots, K\} \tag{2.4}$$

$$y_j \in \{0, 1\} \quad j \in \{1, \dots, K\} \tag{2.5}$$

Here K is an upper bound on the chromatic number. The binary variable x_{vj} indicates whether vertex $v \in V$ is colored by color $j \in \{1, \dots, K\}$ and the binary variable y_j indicates whether color $j \in \{1, \dots, K\}$ is used in the graph coloring. Therefore, we want to minimize $\sum_{j=1}^{K} y_j$. Moreover, Equation 2.2 ensures that no two adjacent vertices, are assigned the same color, and that $y_j = 1$ when color $j \in \{1, \dots, K\}$ is used in the graph coloring. Equation 2.3 makes sure that each vertex is assigned exactly 1 color.

An easy upper bound on the chromatic number is of course the number of vertices. However, a stricter upper bound is given by $\Delta(G)+1$. Here $\Delta(G) := \max_{v\in V} |\{w \mid \{v,w\} \in E\}|$ is the maximum degree of the graph. This is equal to the maximum number of neighbors of a vertex in the graph. The proof of this statement can be found in Wilson [2015]. Furthermore, Brooks' theorem states that an upper bound for the chromatic number is given by $\Delta(G)$ if $\Delta(G) \geq 3$. The proof of this can also be found in Wilson [2015]. Hence, for any graph we can obtain an upper bound on the chromatic number which is based only on $\Delta(G)$.

2.2 Definitions

We adhere to the definitions regarding permutations as described in [Martindale and Sterk, 2022]. The most important definitions and lemmas are discussed below.

Definition 1. Let X be a finite set. A bijection $f : X \to X$ is called a *permutation*. The set of all permutations of a set X is denoted by S_X.

Definition 2. Let $\sigma, \tau \in S_X$. The *composition* of σ and τ is denoted by $\sigma \circ \tau$ and equals $\sigma(\tau(x))$. Note that the composition is again a permutation on X.

Definition 3. Let $\sigma \in S_X$. The fixed points of σ are the $x \in X$ such that $\sigma(x) = x$. The set of points fixed by σ is denoted by $\mathrm{fix}(\sigma)$. The complement of $\mathrm{fix}(\sigma)$ is denoted by $\mathrm{supp}(\sigma) := X \setminus \mathrm{fix}(\sigma)$ and is called the support of σ.

Definition 4. Let $\sigma \in S_X$ have support $\mathrm{supp}(\sigma) = \{a_1, \ldots, a_m\}$. We say that σ is an *m-cycle* if, after reordering and relabelling if necessary, we have that $\sigma(a_m) = a_1$ and for every $1 \leq i < m$ we have that $\sigma(a_i) = a_{i+1}$. Furthermore, we will denote this as $\sigma = (a_1, a_2, \ldots, a_m)$. A 2-cycle is called a *transposition*.

Lemma 1. *Consider S_X.*

- *Every permutation is the product of disjoint cycles.*
- *This product is unique up to rearrangement of the factors.*

As a consequence, $\sigma \in X$ can be written as:

$$\sigma = c_1 \cdots c_r,$$

where each c_i is a cycle defined as $(a_{1,i} \cdots a_{m_i,i})$. The *1-cycles* are usually omitted from the notation.

In order to talk about symmetric graph colorings, we will introduce the following definitions. For completeness, we will also give the definition of a graph coloring here. Furthermore, we define: $[n] := \{1, 2, \ldots, n\}$.

Definition 5. Let $G = (V,E)$ be a graph, and let $[K]$ be the set of possible colors. Furthermore, let $C : V \to [K]$ be a function such that $C(v) \neq C(w)$ for all $\{v,w\} \in E$. Then C is a *graph coloring* of G, such that if $C(v) = k$ for $v \in V$ and $k \in [K]$, then v is colored by k.

Definition 6. Let $G = (V,E)$ be a graph. Let $R \subset V$. Then we define:

$$N(R) := \{v \in V \setminus R \mid \exists w \in R \quad \text{such that } \{v,w\} \in E\},$$

to be the *neighbour set* of R.

Definition 7. Let C be a *graph coloring* of the graph $G = (V,E)$. Let $\tau \in S_{[K]}$ be a permutation on the colors in $[K]$, then $\tau \circ C$ is a *graph coloring symmetry* of C.

Lemma 2. *If C^* is a graph coloring symmetry of C, then C^* is again a graph coloring.*

Proof. Let C^* be a *graph coloring symmetry* of C. Then by definition there exists a $\tau \in S_K$, such that $C^* = \tau \circ C$. We must show that for all $\{v, w\} \in E$, we have that $C^*(v) \neq C^*(w)$. Next to that, we must show that $C^* = \tau \circ C : V \to [K]$.

Let $\{v, w\} \in E$. Suppose $\tau \circ C(v) = \tau \circ C(w)$. Since τ is a bijection, this implies that $C(v) = C(w)$. However, by definition of a graph coloring: $C(v) \neq C(w)$. We reach a contradiction. Hence, $\tau \circ C(v) \neq \tau \circ C(w)$. Thus we have shown that $C^*(v) \neq C^*(w)$.

We now show that $C^* = \tau \circ C : V \to [K]$. Note that $\tau : [K] \to [K]$ and $C : V \to [K]$. Hence, $C^* = \tau \circ C : V \to [K]$. We have now shown that indeed C^* is a graph coloring. □

2.3 Symmetries

When solving the graph coloring problem many symmetries are encountered, as was mentioned in Chapter 1. One of such symmetries is visible in Figure 1.1. Another example is given by the graph colorings in Figure 2.1 and Figure 2.2. Again, the color green became red, red became orange and orange became green. In order to talk about these kind of symmetries in a more mathematical way, we will represents these colorings by matrices.

Let $x \in \{0,1\}^{|V| \times K}$ such that x is within the feasible region of the integer program described by the set of equations 2.1, 2.2, 2.3, 2.4 and 2.5. Note that for any given $x \in \{0,1\}^{|V| \times K}$, the value of y is automatically fixed by the integer program. Furthermore, any permutation of the columns of x is again in the feasible region of the integer program, and it has the same objective value as x. To illustrate this, we define $x_1, x_2 \in \{0,1\}^{|V| \times K}$ as follows:

$$x_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \quad x_2 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Note that x_2 can be obtained by permuting the columns of x_1. These are graph colorings of the same graph, where the rows represent the vertices and the columns represent the colors (red, orange and green respectively). The corresponding graph colorings are visible in Figures 2.1 and 2.2.

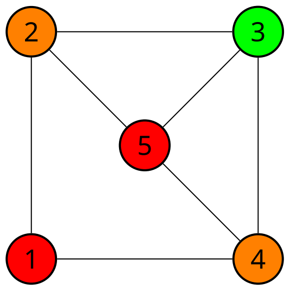

Figure 2.1: Graph coloring corresponding to x_1

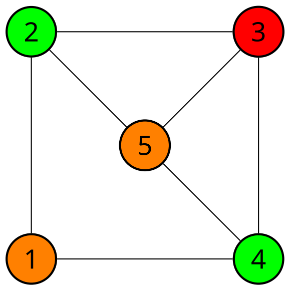

Figure 2.2: Graph coloring corresponding to x_2

Since these symmetries have the same objective value, we would only like to handle one element of each symmetry class. However, the basic strategy to solve the integer program considers solutions that are symmetries of each other. For example, a node in the branch and bound tree could be a sub-problem where vertex 1 is fixed to be colored by red, and vertex 2 is fixed to be colored by green, while another node could be the sub-problem where vertex 1 is fixed to be colored by green, and vertex 2 is fixed to be colored by red. Note that these sub-problems will yield the exact same solutions up to a permutation of the colors. This makes the branch and bound tree unnecessarily large. Therefore, we would like to handle these symmetries in such a way that the branch and bound tree is cut off at certain nodes. Multiple methods have been developed to deal with such symmetries, which we will discuss in the next section.

2.4 Symmetry handling

Multiple methods have been developed to deal with symmetries in solving the graph coloring problem. One way of handling symmetries is to add additional inequalities to the problem, which remove feasible solutions, while ensuring that at least one optimal solution of the original problem remains feasible. For the specific problem of graph coloring, such inequalities have been proposed by Méndez-Díaz and Zabala [2006].

A second method was to reformulate the original integer program as described in Section 2.1. This has been done in [Mehrotra and Trick, 1996] by using a column generation method.

Besides problem specific techniques, also general techniques for handling symmetries in integer programs exist. Margot [2002] studied a branch-and-cut method, that uses methods from computational group theory. He showed how to efficiently prune isomorphic subproblems and generate isomorphism cuts (that do not influence the value of the optimal solution).

In the next section we will discuss the orbitopal fixing method in greater detail since this is the method that forms the foundation of the method introduced in Chapter 4.

2.4.1 Orbitopal fixing

The orbitopal fixing method for symmetry handling is a method that relies on orbitopes, as introduced in [Kaibel and Pfetsch, 2008]. Kaibel et al. [2011] describe a method that utilizes these orbitopes to cut off nodes that will not be lexicographically maximal with respect to the columns of a solution $x \in \{0,1\}^{|V|,K}$ of the integer program. Looking back at the example given in Section 2.3, this would imply that x_1 will be a solution that is found in the branch and bound process, but x_2 is not. This method forms the foundation of the method introduced in Chapter 4. Therefore, we will now give a deeper understanding of the method introduced in [Kaibel et al., 2011].

In [Kaibel and Pfetsch, 2008], orbitopes are introduced. We will state a few important definitions and notations that are introduced in that paper. Let p, q be positive integers. For $x \in \mathbb{R}^{p\times q}$ and $S \subseteq [p]\times[q]$ we define:

$$x(S) := \sum_{(i,j)\in S} x_{ij}$$

Let $\mathcal{M}_{p,q} := \{0,1\}^{p\times q}$ be the set of $0/1$-matrices of size $p \times q$ Furthermore, if we define $\text{row}_i := \{(i,1),(i,2),\dots,(i,q)\}$, where $i \in [p]$, then we are able to define:

$$\mathcal{M}^{=}_{p,q} := \{x \in \mathcal{M}_{p,q} : x(\text{row}_i) = 1 \text{ for all } i\}$$

and,

$$\mathcal{M}^{\leq}_{p,q} := \{x \in \mathcal{M}_{p,q} : x(\text{row}_i) \leq 1 \text{ for all } i\}.$$

So, $\mathcal{M}^{=}_{p,q}$ denotes the set of $0/1$-matrices of size $p \times q$ with exactly one 1 in each row and $\mathcal{M}^{\leq}_{p,q}$ denotes the set of $0/1$-matrices of size $p \times q$ with at most one 1 in each row

Furthermore we introduce the lexicographic ordering $\prec$ of $\mathcal{M}_{p,q}$ with respect to the ordering

$$(1,1) < (1,2) < \cdots < (1,q) < (2,1) < (2,2) < \cdots < (2,q) < \cdots < (p,q)$$

of matrix positions, i.e., $A \prec B$ with $A = (a_{ij}), B = (b_{ij}) \in \mathcal{M}_{p,q}$ if and only if $a_{k\ell} < b_{k\ell}$, where (k,ℓ) is the first position (with respect to the previous ordering) where A and B differ. $S_{[n]}$ is the group of all permutations of $[n]$ and let H be a subgroup of $S_{[q]}$, acting on $\mathcal{M}_{p,q}$ by permuting columns. Let $\mathcal{M}^{\max}_{p,q}(H)$ be the set of matrices of $\mathcal{M}_{p,q}$ that are $\prec$-maximal within their orbits under the group action G. An observation that can be made is that a matrix of $\mathcal{M}_{p,q}$ is contained in $\mathcal{M}^{\max}_{p,q}$ if and only if its columns are in non-increasing lexicographic order (with respect to the order $\prec$ defined earlier).

We can now define orbitopes.

Definition 8. 1. The *full orbitope* associated with the group G is

$$\mathrm{O}_{p,q}(H) := \operatorname{conv} \mathcal{M}^{\max}_{p,q}(H)$$

2. We associate with the group G the following restricted orbitopes:

$$\mathrm{O}^{\leq}_{p,q}(H) := \operatorname{conv}\left(\mathcal{M}^{\max}_{p,q}(H) \cap \mathcal{M}^{\leq}_{p,q}\right) \quad \text{(packing orbitope)}$$
$$\mathrm{O}^{=}_{p,q}(H) := \operatorname{conv}\left(\mathcal{M}^{\max}_{p,q}(H) \cap \mathcal{M}^{=}_{p,q}\right) \quad \text{(partitioning orbitope)}$$

Here $\operatorname{conv}(\cdot)$ denotes the convex hull of a set.

Hence, a packing orbitope is the convex hull of lexicographically maximal matrices where each matrix has at most one 1 in each row. Similarly, a partitioning orbitope is the convex hull of lexicographically maximal matrices where each matrix has exactly one 1 in each row.

Furthermore, in [Kaibel et al., 2011], an algorithm has been introduced that fixes certain entries of a partially known $x \in \mathcal{M}^{=}_{p,q} \cup \mathcal{M}^{\leq}_{p,q}$ given that x must be $\prec$-maximal. This is called the orbitopal fixing algorithm for packing or partitioning orbitopes, depending on whether $x \in \mathcal{M}^{=}_{p,q}$ or $x \in \mathcal{M}^{\leq}_{p,q}$. Note that for $x \in \mathcal{M}^{=}_{p,q} \cup \mathcal{M}^{\leq}_{p,q}$ it is immediately implied that $x_{ij} = 0$ for all $j > i$. Since the columns are sorted with respect to the order $\prec$, and each row contains at most one 1. To illustrate the idea of the algorithm, we give an example. Suppose we are given a partial solution $\tilde{x} \in \mathbb{R}^{3\times3}$ of a binary integer program (in a node of the branch and bound tree), given by:

$$\tilde{x} = \begin{bmatrix} 1 & 0 & 0 \\ * & 0 & 0 \\ * & * & * \end{bmatrix}$$

Here we would like that $\tilde{x} \in \mathrm{O}^{=}_{p,q}$. Note that for $\tilde{x} \in \mathrm{O}^{=}_{p,q} \cup \mathrm{O}^{\leq}_{p,q}$ we must always have the blue 0 entries, for the matrix to be lexicographically maximal. If furthermore $\tilde{x} \in \mathrm{O}^{=}_{p,q}$, we are also able to fix the blue 1 entry. Observe that we can fix a 1 in entry $(2,1)$ since every row must have exactly one 1. Hence:

$$\tilde{x} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ * & * & * \end{bmatrix}$$

Furthermore, we must have that the second column is lexicographically greater or equal than the third column. Note that this is not possible if the entry $(3,3)$ equals 1. Therefore, we can fix it to be equal to 0. This gives us:

$$\tilde{x} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ * & * & 0 \end{bmatrix}$$

Observe, that by fixing these variables to either 0 or 1 we have cut off potential nodes of the branch and bound tree.

Application to the graph coloring problem

Let (x, y), where $x = (x_{ij})$ and $y = (y_j)$, be a solution to the graph coloring problem, then $x \in \mathscr{M}^{=}_{|V|,K}$. A way to not consider all symmetries, is by only considering solutions in $\mathrm{O}^{=}_{p,q}(H)$. In this case H is a subgroup of $S_{[K]}$. This way we do not consider solutions of the graph coloring problem that are the same up to a permutation of the colors (which is the same as a permutation of the columns of x). By applying the orbitopal fixing method for partitioning orbitopes, one can make sure that $x \in \mathrm{O}^{=}_{p,q}(H)$.

Chapter 3

Sub-symmetries

In this Chapter, an elaboration can be found on the sub-symmetry handling method of Bendotti et al. [2020]. This method, together with the orbitopal fixing method by Kaibel et al. [2011] is the foundation of the method that is proposed in Chapter 4. In order to introduce the symmetry handling method of Bendotti et al. [2020], we will first illustrate what a sub-symmetry entails. Next to that, we will introduce some definitions regarding sub-symmetries.

In the previous chapter our focus was on handling symmetries that appear in the original problem of the graph coloring problem. These are the symmetries that arise when permuting the colors. This section will focus on symmetries that are not symmetries of the original problem, but are symmetries of the sub-problems in the branch and bound tree. We denote these kind of symmetries by sub-symmetries. To clarify this notion of a sub-symmetry, we will give an example. Suppose we are given the graph coloring $\tilde{x}_1 \in \mathbb{R}^{5\times 3}$, given by:

$$\tilde{x}_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ * & * & * \\ 0 & 1 & 0 \\ * & * & * \end{bmatrix}$$

The corresponding partial graph coloring can be found in Figure 3.1 Note that in this graph we can interchange the colors red and green in the subset of vertices $R = \{3,5\}$. This is illustrated in Figures 3.2 and 3.3. So Figure 3.3 is a sub-symmetry of 3.2. We can interchange the colors due to the fact that for every $v \in N(R) = \{2,4\}$ we have that v is not colored by red or green. This is called the *0-neighbor-sub-symmetry* [Bendotti et al., 2020]. This statement will be formalized and proven in Section 3.2 for the general case.

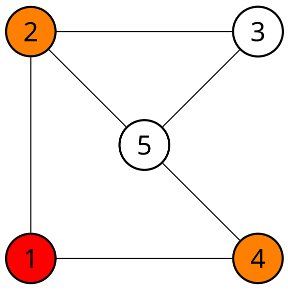

Figure 3.1: Graph coloring corresponding to $\tilde{x}_1$

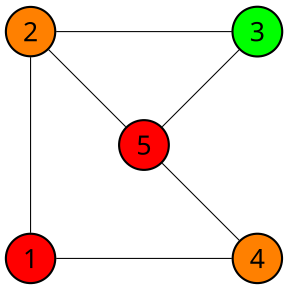

Figure 3.2: Potential completed graph coloring of $\tilde{x}_1$

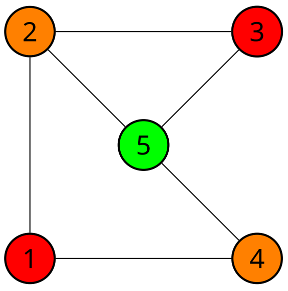

Figure 3.3: Potential completed graph coloring of $\tilde{x}_1$

However, if we are given the partial coloring $\tilde{x}_2 \in \mathbb{R}^{5\times 3}$, given by:

$$\tilde{x}_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ * & * & * \\ 0 & 0 & 1 \\ * & * & * \end{bmatrix},$$

which is visualized in Figure 3.4, we can not switch the colors red and green within $R = \{3,5\}$. In Figure 3.5 we see a valid graph coloring. However, switching the colors red and green within R gives us the invalid coloring that is shown in Figure 3.6. Hence, permuting a set of colors in a subset of V does not have to be a graph coloring symmetry of the original problem.

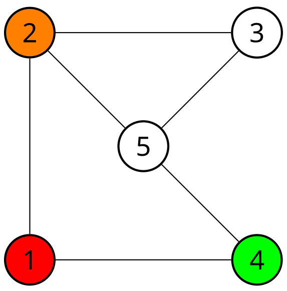

Figure 3.4: Graph coloring corresponding to $\tilde{x}_2$

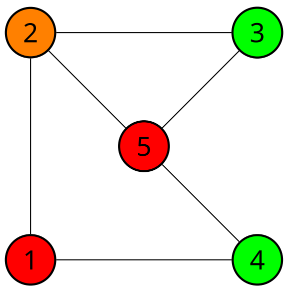

Figure 3.5: Potential completed graph coloring of $\tilde{x}_2$

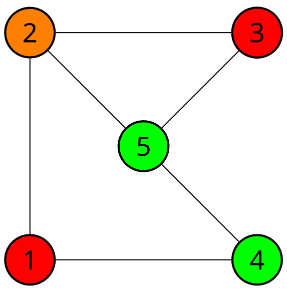

Figure 3.6: Invalid completed graph coloring of $\tilde{x}_2$

3.1 Definition of a sub-symmetry

A sub-symmetry of a graph coloring can be seen as a permutation of the colors on a subset of vertices $R \subset V$. So, if we are dealing with a sub-symmetry only the color assignment of the subset R is changed with respect to the original coloring. The remaining vertices $V \setminus R$ keep the same color assignment. Of course, the sub-symmetry must still be a valid graph coloring. We will now introduce a formal definition of a sub-symmetry.

Definition 9. Let C be a *graph coloring* of the graph $G = (V,E)$. Moreover, let $R \subset V$. Let $\tau \in S_{[K]}$ be a permutation on the colors in $[K]$. We define $C^*: V \to K$ such that $C^*(r) = \tau \circ C(r)$ for all $r \in R$ and $C^*(v) = C(v)$ for all $v \in V \setminus R$. Moreover, $C^*(v) \neq C^*(w)$ for all $\{v,w\} \in E$. Then C^* is a *graph coloring sub-symmetry* of C.

Note that by definition, any *graph coloring sub-symmetry* is again a *graph coloring*.

The next lemma is used to show that the integer program that is used to solve the graph coloring problem, does not need to consider all sub-symmetries of a graph coloring, but just one.

Lemma 3. *Let $C^* \sim_R C$ denote that C^* is a graph coloring sub-symmetry of C, where C is a graph coloring of the graph $G = (V,E)$ and where $R \subset V$. Then $\sim_R$ is an equivalence relation.*

Proof. To show that $\sim_R$ is an equivalence relation it needs to be shown that $\sim_R$ is reflexive, symmetric, and transitive.

We first show the property reflexive. So we must show that $C \sim_R C$. Let C be a *graph coloring* of the graph $G = (V,E)$ and $R \subset V$. Let $\tau = \text{id} \in S_{[K]}$. Then $C^*(v) = \begin{cases} C(v) & v \in V \setminus R \\ \tau \circ C(v) & v \in R \end{cases} = C(v)$.

Note that $C^*(v) = C(v) \neq C(w) = C^*(v) \quad \forall \{v,w\} \in E$ since C is a *graph coloring*, and therefore $C^* = C$ is a *graph coloring sub-symmetry* of C. Therefore, $C \sim_R C$.

Next we show the symmetric property. So we must show that if $C^* \sim_R C$, then also $C \sim_R C^*$ Let C be a *graph coloring* of the graph $G = (V,E)$ and $R \subset V$. Suppose that $C^* \sim_R C$. This implies that there exists a $\tau \in S_{[K]}$ such that

$$C^*(v) = \begin{cases} C(v) & v \in V \setminus R \\ \tau \circ C(v) & v \in R \end{cases}.$$

Now take $\sigma = \tau^{-1} \in S_{[K]}$ then

$$C^{**}(v) = \begin{cases} C^*(v) & v \in V \setminus R \\ \sigma \circ C^*(v) & v \in R \end{cases} = \begin{cases} C(v) & v \in V \setminus R \\ \tau^{-1} \circ \tau \circ C(v) & v \in R \end{cases} = C(v).$$

This implies that $C \sim_R C^*$.

Lastly we show the transitive property. Suppose that $C^{**} \sim_R C^*$ and $C^* \sim_R C$. We now need to show that $C^{**} \sim_R C$.

Since $C^{**} \sim_R C^*$, there exists a $\tau \in S_{[K]}$ such that: $C^{**}(v) = \begin{cases} C^*(v) & v \in V \setminus R \\ \tau \circ C^*(v) & v \in R \end{cases}$

Since $C^* \sim_R C$, there exists a $\sigma \in S_{[K]}$ such that: $C^*(v) = \begin{cases} C(v) & v \in V \setminus R \\ \sigma \circ C(v) & v \in R \end{cases}$

Note that: $C^{**}(v) = \begin{cases} C^*(v) & v \in V \setminus R \\ \tau \circ C^*(v) & v \in R \end{cases} = \begin{cases} C(v) & v \in V \setminus R \\ \tau \circ \sigma \circ C(v) & v \in R \end{cases}$

Since $C^{**} \sim_R C^*$ implies that $C^{**}(v) \neq C^{**}(w) \quad \forall \{v,w\} \in E$, and $\tau \circ \sigma \in S_{[K]}$, we have now shown that $C^{**} \sim_R C$. This completes the proof. □

Note that each equivalence class of $\sim_R$ uses the same number of colors. From the fact that the relation is an equivalence relationship, we know that the equivalence classes partition the set of all possible graph colorings. Hence, we only need to consider one element of each equivalence class.

3.2 0-neighbor-sub-symmetry

As already mentioned in Chapter 3, one of the symmetries that can be considered is the 0-neighbor-sub-symmetry [Bendotti et al., 2020]. Bendotti et al. [2020] considers only 0-neighbor-sub-symmetries in which two colors are being permuted. However, this notion can be extended to any $l \leq K$. As long as $R \subset V$ is chosen such that no neighbor of R is colored by a color in $[l]$, it is okay to permute the colors in $[l]$ within the subset R. This is made formal in Theorem 1.

Theorem 1. *Let C be a graph coloring of the graph $G = (V,E)$ and let $l \leq K$ such that $L = \{c_1, c_2, \ldots, c_l\}$ denotes a subset of $[K]$. Let $R \subset V$ with the condition that for all $v \in N(R)$ we have that $C(v) \notin L$. Let $\tau \in S_L$. Then $C^*(v) = \begin{cases} C(v) & v \in V \setminus R \\ \tau \circ C(v) & v \in R \end{cases}$ is a graph coloring sub-symmetry of C.*

Proof. We assume that for all $x \in N(R)$ we have $C(x) \notin L$. We need to show that for any $\{v,w\} \in E$ we have that $C^*(v) \neq C^*(w)$. Let $\{v,w\} \in E$.

Case 1: $v,w \in V \setminus R$.
In this case we have that $C^*(v) = C(v)$ and $C^*(w) = C(w)$. Since C is a graph coloring we must

have that $C^*(v) = C(v) \neq C(w) = C^*(w)$, which is what we needed.

Case 2: $v, w \in R$.
We know that there exists some $\tau \in S_L$, such that $C^*(v) = \tau \circ C(v)$, and $C^*(w) = \tau \circ C(w)$. Suppose that $\tau \circ C(v) = \tau \circ C(w)$. Since τ is a bijection, this must imply that $C(v) = C(w)$. This is a contradiction because C is a graph coloring. Hence, $C^*(v) = \tau \circ C(v) \neq \tau \circ C(w) = C^*(w)$. This is what we needed to show.

Case 3: $v \in R$ and $w \in V \setminus R$.
There exists a $\tau \in S_L$ such that $C^*(v) = \tau \circ C(v)$. Furthermore, we have that $C^*(w) = C(w)$. Suppose, that $C(v) \notin L$, then $C^*(v) = \tau \circ C(v) = C(v)$. Since C is a graph coloring we have that $C^*(v) = C(v) \neq C(w) = C^*(w)$. If instead $C(v) \in L$, since $\tau \in S_L$ we must have that $\tau \circ C(v) \in L$. So $C^*(v) \in L$. Since $\{v, w\} \in E$, we have that $w \in N(R)$. By assumption we have that $C(w) \notin L$. Hence, $C^*(v) \neq C(w) = C^*(w)$. This completes the proof. □

Algorithm 1 shows how we are able to find multiple subsets R of V in which we can permute the colors of the set $\{c_1, c_2, \ldots, c_l\}$ within the graph coloring.

Algorithm 1 Finding 0-neighbor-sub-symmetry of a graph G

```
Require: $G = (V, E)$, $\{c_1, c_2, \ldots, c_l\}$, and the partial coloring $x \in \mathscr{M}^{=}_{|V|,K}$
Ensure: Subsets of $V$ in which we can permute the colors $\{c_1, c_2, \ldots, c_l\}$
  for each $v \in V$ do
      if $x_{vc_i} = 0$ for all $i \in [l]$ then
          Remove $v$ from $V$
      end if
  end for
  return The connected components of $G$
```

Lemma 4. *Let R be equal to a connected component that is returned by Algorithm 1. For all $v \in N(R)$ we have that $C(v) \notin L$*

Proof. Let R be equal to a component that is outputted by Algorithm 1. Let $v \in N(R)$. Then we have that v is deleted by Algorithm 1. Therefore, v was not colored by any of the colors $\{c_1, c_2, \ldots, c_p\}$. So $C(v) \notin L$ □

The correctness of Algorithm 1 follows from Theorem 1 and Lemma 4.

Chapter 4

Handling sub-symmetries

In this chapter a new method will be introduced. This method considers 0-neighbor sub-symmetries in a dynamical way.
When solving the integer program introduced in 2.1, there arise multiple branches of the branch and bound tree of the IP. We then look at an individual node of this tree, by considering not only global symmetries of the graph (symmetries of the original problem) by using orbitopal fixing (as done in [Kaibel et al., 2011]), but also by considering 0-neighbor-sub-symmetries.

First we will discuss how to handle a 0-neighbor-sub-symmetry, for a given set of colors $\{c_1, c_2, \ldots, c_l\}$. Next it is important to determine which sets of colors are of interest for finding 0-neighbor-sub-symmetries, when we find ourselves in a given node of the branch and bound tree. So we are given a partially known $x \in \mathcal{M}^{=}_{|V|,K}$.

4.1 Handling 0-neighbor-sub-symmetries

As discussed in Section 3.2, we can easily find subsets of vertices in which we can permute the colors of the given set by using Algorithm 1. These sub-symmetries can then be handled by using the orbitopal fixing method for packing orbitopes [Kaibel et al., 2011]. This will be done by selecting the columns corresponding to the color set $\{c_1, c_2, \ldots, c_l\}$ and the rows of $x \in \mathcal{M}^{=}_{|V|,K}$ corresponding to the vertices of a component C that is outputted by Algorithm 1. We will denote this sub-matrix by $\hat{x} \in \mathcal{M}^{\leq}_{|C|,p}$.

Note that in general $\hat{x} \in \mathcal{M}^{\leq}_{|C|,p}$ and not in $\mathcal{M}^{=}_{|C|,p}$, since the vertices (represented by the rows of $\hat{x}$) need not be colored by any of the colors (represented by the columns of $\hat{x}$). Hence, $\hat{x}$ has at most one 1 per row, but not necessarily one 1 per row.

We will give an example to illustrate the method that was just introduced. Suppose we are given the partial graph coloring $x \in \mathcal{M}^{=}_{5,3}$, as shown in Figure 4.1. Then the corresponding (partial) coloring is given in Figure 4.2.

Suppose we would now use Algorithm 1, with the color set $(\texttt{red}, \texttt{blue})$. Then we will be given as output the component consisting of vertices $\{1, 2\}$ and the component consisting of vertices $\{4, 5\}$. Then the sub-matrices $\hat{x}_1, \hat{x}_2 \in \mathcal{M}^{\leq}_{2,2}$ are given in Figure 4.3.

$$x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ * & * & 0 & * \\ * & * & 0 & * \end{bmatrix}$$

Figure 4.1: Partial graph coloring $x \in \mathcal{M}^{=}_{5,3}$

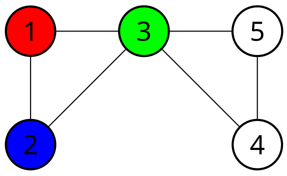

Figure 4.2: Partial graph coloring $x \in \mathcal{M}^{=}_{5,3}$ illustrated as graph

$$\hat{x}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\hat{x}_2 = \begin{bmatrix} * & * \\ * & * \end{bmatrix}$$

Figure 4.3: Sub-matrices $\hat{x}_1$ and $\hat{x}_2$

On these sub-matrices $\hat{x}_1$ and $\hat{x}_2$ the orbitopal fixing algorithm is applied (for packing orbitopes). Note that $\hat{x}_1$ is already lexicographically maximal. Therefore, orbitopal fixing has no effect. However, in the case of $\hat{x}_2$ this algorithm does have an effect. $\hat{x}_2$ becomes $\begin{bmatrix} * & 0 \\ * & * \end{bmatrix}$. Hence,

$$x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ * & 0 & 0 & * \\ * & * & 0 & * \end{bmatrix}.$$

Since we have that $\hat{x}_2 \in \mathcal{M}^{\leq}_{2,2}$ and not necessarily in $\mathcal{M}^{=}_{2,2}$, we can not fix the entry $(1,1)$ of $\hat{x}_2$ to 1.

4.1.1 Choosing color sets

The only thing that is left to determine, is which color sets are useful to consider with respect to the 0-neighbor-sub-symmetry. There are multiple options for these color sets. In this report we will focus on all possible color sets of cardinality p, where $p \in \{2,3,4,K-2,K-3,K-4\}$. We require that $1 < p < K$. The number of such color sets equals $\binom{K}{p}$. This binomial is maximal for $p = \lceil \frac{1}{2}K \rceil$, and minimal for the boundary values $p = 1$ and $p = K$. By choosing $p \in \{2,3,4,K-2,K-3,K-4\}$ the number of possible color sets is kept relatively small. Note that $p = 1$ is not a value that needs to be investigated since we can not switch colors if we are only allowed to switch 1 color. Furthermore, we will not look into $p = K$, since this would mean that we are allowed to switch all the colors. In this case Algorithm 1 will never delete a vertex from a partial coloring $x \in \mathcal{M}^{=}_{|V|,K}$, since a row can not contain only zeros. Hence, the only symmetries that consider all colors are symmetries of the original problem, which have been considered already.

In order to easily refer to these types of color sets, we introduce the following methods:

- **Method 1:** Consider all possible color sets of cardinality 2
- **Method 2:** Consider all possible color sets of cardinality 3
- **Method 3:** Consider all possible color sets of cardinality 4
- **Method 4:** Consider all possible color sets of cardinality $K-2$
- **Method 5:** Consider all possible color sets of cardinality $K-3$
- **Method 6:** Consider all possible color sets of cardinality $K-4$

Note that it is of importance that Algorithm 1 deletes at least 1 vertex, else we are not handling a sub-symmetry, but simply a symmetry of the whole graph. This is unnecessary, since these symmetries are already handled by the orbitopal fixing method (of partitioning orbitopes). Therefore we should only look at a coloring set $(c_1, c_2, \ldots, c_l)$ if there is at least one vertex removed by Algorithm 1.

Observe that this is equivalent to having at least one vertex v, such that $x_{vc_i} = 0$ for all $i \in [l]$ (see Algorithm 1). Since we are only considering lexicographically maximal solutions, we can be sure that the first row of x has on the first entry a 1, and on all other entries a 0. Therefore, for any given color set $\{c_1, c_2, \ldots, c_p\}$ that does not contain color 1 (which is the color represented by the first column of x), Algorithm 1 will delete a vertex. So we should consider all possible color sets that do not contain color 1

However, if the set $\{c_1, c_2, \ldots, c_l\}$ does contain color 1, we must have that there exist a vertex v, such that $x_{v1} = 0$, in order for Algorithm 1 to delete a vertex. Moreover, if such a vertex exist, we must check whether for the remaining colors j in $\{c_1, c_2, \ldots, c_l\}$, we have that $x_{vj} = 0$. The idea of this method is described in Algorithm 2 in case we search for color sets of cardinality 2.

This more dynamic method will be implemented for color sets of cardinality 2, and denoted by **Method d1**. We have chosen not to implement this method for other possible cardinalities of the color set, since results will show that method 1 is the most promising method in comparison to methods 2 to 6 (see Chapter 5).

Algorithm 2 Selecting color sets of cardinality 2 to use when searching for 0-neighbor-sub-symmetries

```
Require: The partial coloring $x \in \mathscr{M}^{=}_{|V|,K}$
Ensure: A set of color sets
  Initialize the set $C$.
  for each color set $\{i, j\}$ that does not contain color $1$ do
      Add $\{i, j\}$ to the set $C$.
  end for
  for each $v \in [|V|] \setminus \{1\}$ do
      if $x_{v,1}$ equals $0$ then
          for each $i \in [K] \setminus \{1\}$ do
              if $x_{v,i}$ equals $0$ then
                  Add $\{1, i\}$ to the set $C$.
              end if
          end for
      end if
  end for
  return C
```

Chapter 5

Results

In this chapter, the various methods introduced in Chapter 4 will be tested on a set of benchmark graphs [Gra]. We have made use of the SCIP optimization suite to be able to implement and test the methods [Bestuzheva et al., 2021]. Next to that, code provided by Hojny et al. [2022] has been used and altered to implement the methods. $K = \Delta(G) + 1$ was used as an upper bound on the chromatic number. Furthermore, the maximum solving time was set to 1 hour: If the instance was not solved after 1 hour, this is denoted by > 3600 in the tables. For a fair comparison of the data, all tests were run on the same hardware.

Table 5.1: Results of graph instance queen5_5. $|V| = 25$, $|E| = 160$ and $K = 17$.

Method	Time (s)
No symmetry handling	1.85
Global symmetry handling	**0.08**
Method 1	0.17
Method d1	0.17
Method 2	0.60
Method 3	1.98
Method 4	0.26
Method 5	1.18
Method 6	4.02

Table 5.2: Results of graph instance queen6_6. $|V| = 36$, $|E| = 290$ and $K = 20$.

Method	Time (s)
No symmetry handling	> 3600
Global symmetry handling	**3.9**
Method 1	4.68
Method d1	4.35
Method 2	11.90
Method 3	44.03
Method 4	6.28
Method 5	22.91
Method 6	81.36

Table 5.3: Results of graph instance queen7_7. $|V| = 49$, $|E| = 476$ and $K = 25$.

Method	Time (s)
No symmetry handling	565.36
Global symmetry handling	65.69
Method 1	18.29
Method d1	**16.52**
Method 2	156.09
Method 3	131.52
Method 4	92.39
Method 5	331.43
Method 6	1611.93

Table 5.4: Results of graph instance myciel3. $|V| = 11$, $|E| = 20$ and $K = 6$.

Method	Time (s)
No symmetry handling	0.37
Global symmetry handling	**0.01**
Method 1	**0.01**
Method d1	**0.01**
Method 2	**0.01**
Method 3	**0.01**
Method 4	**0.01**
Method 5	0.02
Method 6	0.02

Table 5.5: Results of graph instance myciel4. $|V| = 23$, $|E| = 71$ and $K = 12$.

Method	Time (s)
No symmetry handling	1798.06
Global symmetry handling	**0.07**
Method 1	0.45
Method d1	0.46
Method 2	0.56
Method 3	1.01
Method 4	0.13
Method 5	0.75
Method 6	1.13

Table 5.6: Results of graph instance myciel5. $|V| = 47$, $|E| = 236$ and $K = 24$.

Method	Time (s)
No symmetry handling	> 3600
Global symmetry handling	78.42
Method 1	71.74
Method d1	**68.91**
Method 2	225.42
Method 3	1438.40
Method 4	130.34
Method 5	413.47
Method 6	2306.75

Table 5.7: Results of graph instance jean. $|V| = 80$, $|E| = 254$ and $K = 37$.

Method	Time (s)
No symmetry handling	306.11
Global symmetry handling	**8.86**
Method 1	18.39
Method d1	17.63
Method 2	165.79
Method 3	1710.10
Method 4	56.34
Method 5	350.88
Method 6	> 3600

Table 5.8: Results of graph instance huck. $|V| = 74$, $|E| = 301$ and $K = 54$.

Method	Time (s)
No symmetry handling	> 3600
Global symmetry handling	**9.88**
Method 1	30.72
Method d1	30.50
Method 2	549.08
Method 3	> 3600
Method 4	123.86
Method 5	3314.96
Method 6	> 3600

Table 5.9: Results of graph instance david. $|V| = 87$, $|E| = 406$ and $K = 83$.

Method	Time (s)
No symmetry handling	> 3600
Global symmetry handling	620.77
Method 1	**109.23**
Method d1	111.96
Method 2	> 3600
Method 3	> 3600
Method 4	1545.93
Method 5	> 3600
Method 6	> 3600

Table 5.10: Results of graph instance anna. $|V| = 138$, $|E| = 493$ and $K = 72$.

Method	Time (s)
No symmetry handling	> 3600
Global symmetry handling	**108.08**
Method 1	1241.96
Method d1	1348.76
Method 2	> 3600
Method 3	> 3600
Method 4	1363.51
Method 5	> 3600
Method 6	> 3600

Table 5.11: Results of graph instance zeroin.i.1. $|V| = 211$, $|E| = 4100$ and $K = 112$.

Method	Time (s)
No symmetry handling	> 3600
Global symmetry handling	**758.14**
Method 1	> 3600
Method d1	> 3600
Method 2	> 3600
Method 3	> 3600
Method 4	> 3600
Method 5	> 3600
Method 6	> 3600

Table 5.12: Results of graph instance games120. $|V| = 120$, $|E| = 638$ and $K = 14$.

Method	Time (s)
No symmetry handling	**1.22**
Global symmetry handling	6.81
Method 1	11.96
Method d1	11.89
Method 2	14.18
Method 3	30.11
Method 4	12.59
Method 5	19.35
Method 6	45.81

Chapter 6

Conclusions

The results (see Chapter 5) show that depending on the graph instance different methods can perform best. In most of the test cases we see that the global symmetry handling performs best. For some other instances we see that method 1 or method d1 performs best. Note furthermore, that the performances of the method 1 and d1 are much alike. In the majority of the cases we see that methods 2-6 perform worse than the other methods (except for the method where we perform no symmetry handling at all). Next to that, we see one instance (games120) where no symmetry handling performs better than any of the other methods.

Chapter 7

Discussion

As mentioned in the conclusion, no method performs best in all instances. Depending on the graph, different methods perform best. This can be explained by the lack of certain symmetries in some graphs. In a graph instances where method 1 or d1 performs best we expect that 0-neighbor-sub-symmetries, which consider color sets of cardinality 2, occur either early on in the branch and bound tree or often in the branch and bound tree. This keeps the branch and bound tree smaller in comparison with the other methods. For instances where the global symmetry handling method performed best we expect a lack of 0-neighbor-sub-symmetries of any kind.

Note furthermore that in most instances method 1 performs better than methods 2 to 6. Intuitively this can be explained as follows: If we were looking at 0-neighbor-sub-symmetries considering the color set {red, orange, green} (method 2) we are simply looking at a submatrices consisting of three columns, corresponding to red, orange and green, which must be lexicographically maximal (column 1 > column 2 > column 3). However, we could also look at the 0-neighbor-sub-symmetry considering the color sets {red, orange} and {orange, green} (method 1), which also implies column 1 > column 2 > column 3 for the found submatrices. This seems to show that method 1 also finds all the symmetries of method 2. A similar argument can be given for the other methods. Note however, that these submatrices of the two methods do not necessarily have the same number of rows which is why it is not necessarily true that method 1 implies the other methods.

Moreover, it is interesting that in the instance of myciel4, we see a rare occasion where method 4 outperforms $1,2,3,5$ and 6. This could be explained due to an occurring subsymmetry of $K-2$ colors, that is not implied by any of the other methods. Based on the results, one expects that a 0-neighbor-sub-symmetry of $K-2$ colors does not occur often in an arbitrary graph.

In further research, one could look for heuristics to determine beforehand, which method is optimal for a given graph. It would be interesting to find a heuristic that depends on the values for $|V|,|E|,K$ and $\Delta(G)$, since these values are easy to obtain for any arbitrary graph.

References

Graph coloring instances. https://mat.tepper.cmu.edu/COLOR/instances.html. Accessed: 17-06-2023. 17

Pascale Bendotti, Pierre Fouilhoux, and Cécile Rottner. Symmetry-breaking inequalities for ilp with structured sub-symmetry. *Mathematical Programming*, 183:61–103, 2020. iii, 2, 9, 11

Ksenia Bestuzheva, Mathieu Besançon, Wei-Kun Chen, Antonia Chmiela, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Oliver Gaul, Gerald Gamrath, Ambros Gleixner, Leona Gottwald, Christoph Graczyk, Katrin Halbig, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Thorsten Koch, Marco Lübbecke, Stephen J. Maher, Frederic Matter, Erik Mühmer, Benjamin Müller, Marc E. Pfetsch, Daniel Rehfeldt, Steffan Schlein, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Boro Sofranac, Mark Turner, Stefan Vigerske, Fabian Wegscheider, Philipp Wellner, Dieter Weninger, and Jakob Witzig. The SCIP Optimization Suite 8.0. Technical report, Optimization Online, December 2021. URL http://www.optimization-online.org/DB_HTML/2021/12/8728.html. 17

Michele Conforti, Gerard Cornuejols, and Giacomo Zambelli. *Integer Programming*. Springer International Publishing, 2014. 1

M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified np-complete problems. In *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, STOC '74, page 47–63, New York, NY, USA, 1974. Association for Computing Machinery. ISBN 9781450374231. 10.1145/800119.803884. URL https://doi.org/10.1145/800119.803884. iii, 1, 3

Christopher Hojny, Tom Verhoeff, and Sten Wessel. Handling sub-symmetry in integer programming using activation handlers, 2022. 17

Volker Kaibel and Marc Pfetsch. Packing and partitioning orbitopes. *Mathematical Programming*, 114:1–36, 7 2008. ISSN 00255610. 10.1007/S10107-006-0081-5/METRICS. URL https://link.springer.com/article/10.1007/s10107-006-0081-5. 6

Volker Kaibel, Matthias Peinhardt, and Marc E. Pfetsch. Orbitopal fixing. *Discrete Optimization*, 8(4):595–610, 2011. ISSN 1572-5286. https://doi.org/10.1016/j.disopt.2011.07.001. URL https://www.sciencedirect.com/science/article/pii/S1572528611000430. iii, 6, 7, 9, 13

François Margot. Pruning by isomorphism in branch-and-cut. *Mathematical Programming*, 94:71–90, 2002. 6

Chloe Martindale and Hans Sterk. Algebra and discrete mathematics (2wf50), January 2022. 4

Anuj Mehrotra and Michael A. Trick. A column generation approach for graph coloring. *INFORMS Journal on Computing*, 8(4):344–354, 1996. 10.1287/ijoc.8.4.344. 6

Isabel Méndez-Díaz and Paula Zabala. A branch-and-cut algorithm for graph coloring. *Discrete Applied Mathematics*, 154(5):826–847, 2006. ISSN 0166-218X. https://doi.org/10.1016/j.dam.2005.05.022. URL `https://www.sciencedirect.com/science/article/pii/S0166218X05003094`. IV ALIO/EURO Workshop on Applied Combinatorial Optimization. 6

Robin J. Wilson. *Introduction to graph theory*. Prentice Hall, 2015. 4