Symmetry in Combinatorial Optimization

Kristina Loeschner

A Thesis

in

The Department

of

Mathematics and Statistics

Presented in Partial Fulfilment of the Requirements for
the Degree of Master of Science at
Concordia University
Montréal, Québec, Canada

May 2002

0-612-72888-9

Abstract

Symmetry in Combinatorial Optimization

Kristina Loeschner

Integer optimization is in the class of NP-hard problems, and it is very time and memory intensive to find optimal solutions. In this thesis an algorithm will be developed to improve the efficiency in solving a linear integer program if there are symmetries in the problem, that is, variables can be permuted without changing the integer program. Using the group of symmetries, the size of the feasible set can be restricted. For the smaller optimization problem, common solution methods will be able to find the optimal solutions faster than for the original problem. The set of all optimal solutions can be generated from the determined ones by applying the symmetry group.

Acknowledgements

This thesis could never have been written without the support of a number of people. I would like to express my gratitude to all of them.

Especially, I want to thank my thesis supervisor, Dr. C. Lam for his time, encouragement and help during the last year.

Also, I am very much indebted to Dr. C. Cummins, without whom my Master's degree might have stalled on the verge of completion because of an administration that took two years to read its own rules and regulations.

Furthermore, my studies in Canada would never have been possible without the scholarship I received from the *Deutscher Akademischer Austauschdienst*.

My special thanks go to my family: my parents who always supported me and helped me survive far away from home with regularly arriving care packages, and Uli and Robert who decided to like me still even though I abandoned them for so long.

Last, but certainly not least: Thank you so much, Matt, for being there for me whenever I needed you!

Contents

List of Tables

List of Figures

List of Symbols

$\mathbb{Z}$	set of integers	[7]
$\mathbb{R}$	real numbers	[7]
$\mathbb{Z}_q$	$= \{0, 1, \ldots, q-1\}$, the set of integers modulo q	p. 40
$\mathbb{Z}^{m \times n}$	set of $m \times n$-matrices with integer entries	
$\|X\|$	cardinality of set X	[7]
$X \times Y$	$\left\{\begin{array}{l}\text{cartesian product of sets } X \text{ and } Y \\ \text{direct product of groups } X \text{ and } Y\end{array}\right.$	[7]
$G \circlearrowleft \Omega$	group G acts on set Ω	p. 12
$\mathrm{stab}_G\{X\}$	setwise stabilizer of set X in G	p. 13
Σ^g	coset of Σ under action of g	p. 13
Σ^G		p. 13
S_n	symmetric group on $\{1, \ldots, n\}$	p. 12
$\mathrm{Sym}\,\Omega$	symmetry group of the set Ω	p. 11
$\mathrm{Sym}\,P$	group of symmetries of the integer program P	p. 17
id	identity element of a group	
$\mathfrak{S}_{opt}(P)$	set of optimal solutions of the integer program P	p. 11
$\alpha \cong \beta$		p. 18
$d(x, y)$	Hamming distance of x and y	p. 40
$K_q(n, r)$	minimum domination number of $\mathbb{F}_q^n$	p. 40
$G(V, E)$	graph with vertex set V and edge set E	p. 3
$\{x, y\}$	edge with endpoints x and y	p. 41
$w(x)$	weight of vertex x	p. 44
$\mathrm{Aut}\,G$	group of automorphisms of graph G	

CHAPTER 1

Introduction

In this thesis, we consider linear integer programming problems with the special property that there exist symmetries in the variables, that is variables can be interchanged without changing the integer program. An algorithm will be developed using the symmetry group to restrict the feasible set, thereby speeding up the computation of optimal solutions with traditional techniques. The set of all optimal solutions can then be generated from the determined ones in the restricted feasible set by applying the symmetry group.

1. Background

The problem to find a solution to an integer optimization problem is non-trivial, in fact it is NP-complete.

In general, it is not hard to find a real value solution for a linear program. The simplex method, developed in the 1950s and much improved since, easily solves large-scale problems with several thousands variables and inequalities (not to mention all the other efficient algorithms that have been developed in the last decades). It uses the fact that the feasible set, i.e. the set of n-tuples satisfying the constraints, represent a polyhedron in $\mathbb{R}^n$. The optimal solution will be either one of its corners, or its facets. Starting at one of the corners, the algorithm moves step by step from one corner to the next in such a way that each time the next feasible solution that is produced is more optimal than the preceding one. Eventually it will reach the optimal solution, at which point it terminates.

Unfortunately, the situation becomes a lot more complicated as soon as we introduce the constraint that the solutions have to be integer valued for some or all of the variables. A classical idea to approach this problem is to drop the condition that the solutions have to be in $\mathbb{Z}$ (this a called a *relaxation* of the integer program), and solve the problem over the real numbers, for example using the simplex method. The problem is that an optimal solution will most likely not coincide with a corner of the real polyhedron in $\mathbb{R}^n$, so the simplex method does not find a feasible solution for the original integer program. Also, there is no easy way to use the obtained solution to determine an integer solution – for example, rounding from the real valued optimal solution to the nearest integers is not necessarily feasible nor optimal. Today's standard solution algorithms use methods like Gomory's Fractional Cuts or Branch-and-Bound Enumeration (see for example [4]), which try to cut off parts of the polyhedron that do not contain integer solutions and hence gradually transform the polyhedron so that the optimal integer solution is a corner. Then relaxation and simplex algorithm will successfully lead to a result. This process is costly, though, and the complexity increases quickly with the number of variables.

With the method proposed in this thesis, the time involved in the optimization process can be decreased for certain integer programs with symmetries. This idea does not seem to be researched so far. By introducing additional constraints, we reduce the size of the feasible set. Classical methods will then be able to find the optimal

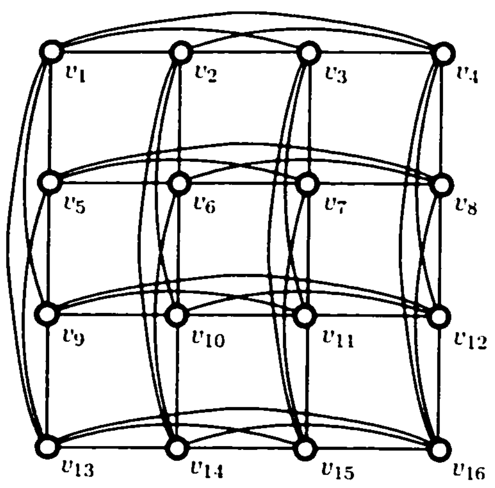

FIGURE 1.1. R_4

solutions faster. As our test cases show, this can lead to a significant improvement of the time the optimization process takes.

The following example shall give an illustration for the relevance of this method.

2. A Motivating Example: Dominating Rooks

Consider the following question: What is the minimum number of rooks it takes to cover or threaten every square of an $m \times m$ chessboard?

This is an example of the dominating set problem, which is studied in graph theory. It can be modelled and solved in terms of an integer optimization problem. A rook placed on a square of the board can attack any other square that is on the same row or column of the board. We can describe this information in the form of a graph $R_m(V, E)$, called the **rooks graph**, where V is the set of vertices and E the set of edges of the graph. The m^2 vertices in V represent the squares of the chessboard. Two vertices are joined by an edge if a rook placed on one of the vertices can attack the other one. For a 4×4 chessboard, we obtain the graph in Figure 1.1.

We want to find a subset D of the vertices such that any vertex in the graph is either in D or is adjacent to an element of D. Then D is called a **dominating set** of the graph R_m, and the **dominating set problem** is to find such a set D of minimal cardinality.

For a graph with finitely many vertices, the dominating set problem can be translated into an optimization problem.

Let $v_1, \dots, v_n$ denote the vertices of R_m. If D is a dominating set of R_m, for $i \in \{1, \dots, n\}$ we define

$$x_i = \begin{cases} 1 & \text{, if } v_i \in D \\ 0 & \text{, otherwise.} \end{cases}$$

In this setup we can write the dominating set problem as a linear program:

$$\text{Minimize } x_1 + \cdots + x_n$$

subject to the constraints

$$x_i + \sum_{\{v_i, v_j\} \in E} x_j \geq 1 \quad \text{for all } v_i \in V,$$
$$x_i \in \{0, 1\}.$$

To minimize the value of the objective function is equivalent to minimizing the size of D, and the constraints guarantee that every vertex is either in D or dominated by an element of D.

We have obtained a linear integer program, i.e. an optimization problem where the objective function and all constraints are linear in the variables $x_1, \dots, x_n$, and all coefficients are integers, as well as the solution for any x_i has to be in a subset of $\mathbb{Z}$. Now we can tackle the problem with optimization methods.

In the case of the 4×4 rooks graph the formulation in an integer program is:

$$\text{Minimize } x_1 + \cdots + x_{16}$$

under the constraints

$$\begin{array}{l}
x_1+x_2+x_3+x_4+x_5+x_9+x_{13} \geq 1\\
x_1+x_2+x_3+x_4+x_6+x_{10}+x_{14} \geq 1\\
x_1+x_2+x_3+x_4+x_7+x_{11}+x_{15} \geq 1\\
x_1+x_2+x_3+x_4+x_8+x_{12}+x_{16} \geq 1\\
x_1+x_5+x_6+x_7+x_8+x_9+x_{13} \geq 1\\
x_2+x_5+x_6+x_7+x_8+x_{10}+x_{14} \geq 1\\
x_3+x_5+x_6+x_7+x_8+x_{11}+x_{15} \geq 1\\
x_4+x_5+x_6+x_7+x_8+x_{12}+x_{16} \geq 1\\
x_1+x_5+x_9+x_{10}+x_{11}+x_{12}+x_{13} \geq 1\\
x_2+x_6+x_9+x_{10}+x_{11}+x_{12}+x_{14} \geq 1\\
x_3+x_7+x_9+x_{10}+x_{11}+x_{12}+x_{15} \geq 1\\
x_4+x_8+x_9+x_{10}+x_{11}+x_{12}+x_{16} \geq 1\\
x_1+x_5+x_9+x_{13}+x_{14}+x_{15}+x_{16} \geq 1\\
x_2+x_6+x_{10}+x_{13}+x_{14}+x_{15}+x_{16} \geq 1\\
x_3+x_7+x_{11}+x_{13}+x_{14}+x_{15}+x_{16} \geq 1\\
x_4+x_8+x_{12}+x_{13}+x_{14}+x_{15}+x_{16} \geq 1
\end{array}$$

$$x_i \in \{0, 1\}$$

We see that this problem grows very fast with the size m of the chessboard. For $m = 4$ we have already 16 variables, for $m = 5$ it will be 25 variables. In the case of a regular board, that is $m = 8$, we have to solve an optimization problem in $8^2 = 64$ variables!

But by looking at the optimization problem, one can see it has a special property. Just as the graph we derived it from has a large number of automorphisms of its vertices, i.e. is very "symmetric", there are symmetries with respect to the variables in the integer program. We can permute the variables $x_1, \ldots, x_{16}$ in certain ways without changing the objective function nor the set of constraints (only possibly transforming one constraint into another). For example, replacing x_1, x_2, x_3, x_4 by x_5, x_6, x_7, x_8, respectively, and vice versa, we will get the same linear program, only with the constraints written in another order: constraints 1, 2, 3 and 4 will take the place of constrains 5, 6, 7 and 8, and vice versa. We say that this permutation of the variables is a symmetry of the linear program. (Note that every automorphism of the rooks graph corresponds to a symmetry of the integer program permuting the respective variables.)

A symmetry in the variables will induce a symmetry in the feasible solutions. We are going to make use of that to restrict the feasible set by adding further constraints, before giving it to linear optimization software, which will then perform more efficiently on the smaller feasible set. Depending on the size of the symmetry group, we cut down on time and memory expenses. After solving this augmented problem, the optimal solutions that we lost by excluding parts of the feasible set can be generated from the obtained solutions using the symmetry group.

In the case of our example, Table 1.1 gives a comparison between the time and memory the optimization software CPLEX [8] used to optimize the original integer program (O) and the augmented system (A). CPLEX is able to handle cases up to $m = 6$ very fast. The table starts from $m = 7$, where the optimization begins to take more time. A clear improvement is visible. It can be seen that the cost of the run

m	time (O)	memory (O)	Cuts (O)	time (A)	memory (A)	Cuts (A)
7	57.3 s	0.90 MB	49	6.7 s	0.10 MB	3
8	668.4 s	7.47 MB	64	59.1 s	0.83 MB	5
9	7581.82 s	70.74 MB	72	707.7 s	9.21 MB	11

TABLE 1.1. CPLEX performance on the rooks graph R_m (O ... original problem, A ... augmented problem)

decreases by about factor 10 in time as well as in memory. The number of Gomory Cuts necessary is greatly reduced in the restricted program.

In Chapter 5 we will look at another dominating set problem, which will be solved in a similar way.

3. Organization of the Thesis

This thesis is structured as follows:

The introduction is followed by Chapter 2, in which the necessary mathematical background will be covered, giving basic preliminaries from integer optimization as well as from group theory.

Chapter 3 will comprise a formal introduction to the problem. The symmetric augmentation algorithm will be motivated and described.

In Chapter 4, we will give details about our implementation of the algorithm.

In the last chapter, results will be presented. The algorithm is applied to optimization problems which were constructed by Ka Leung Ma [13] in his work on the well-known football pool problem but could not be solved by him. A number of new solutions were found for the problems with less than 200 variables, and for the bigger optimization problems several improved lower and upper bounds were established.

The best known upper bound of 73 for the solution of the football pool problem in 6 matches was not improved.

CHAPTER 2

Mathematical Preliminaries

1. Integer Optimization

A **linear (pure) integer programming problem** (IP) in variables $x_1, \dots, x_n$ is an optimization problem $P(x_1, \dots, x_n)$ of the following type:

$$(*) \qquad \text{Minimize} \quad f(x) = c^T x$$

subject to the constraints

$$A_1 x = b_1$$
$$A_2 x \leq b_2$$
$$x_i \in X_i$$

where $A_1 \in \mathbb{Z}^{m_1 \times n}$, $A_2 \in \mathbb{Z}^{m_2 \times n}$, $b_1 \in \mathbb{Z}^{m_1}$, $b_2 \in \mathbb{Z}^{m_2}$ and $x = (x_1, \dots, x_n)^T$. In other words, A_1 is an $m_1 \times n$ matrix with integer coefficients and b_1 is a column vector of length m_1, and so on. The set $X_i \subseteq \mathbb{Z}$ is the domain of x_i for $i \in \{1, \dots, n\}$. Furthermore, $f(x)$ is a linear function in x as $c \in \mathbb{Z}^n$.

If it is clear what is meant, we will refer to $P(x_1, \dots, x_n)$ simply as P.

REMARK. The given form of P does not impose any unnecessary restrictions on the class of problems we consider. Any IP with integer solution can be translated into P:

Every maximization problem is easily transformed into a minimization because finding the maximum of $f(x)$ is equivalent to finding the minimum of $-f(x)$.

Also, all constraints in linear integer programming can be reduced to the given form. Assume we have a constraint of the form

$$a_1x_1 + \cdots + a_nx_n \sim b,$$

where $\sim$ stands for $<$, $>$ or $\geq$. As all operations are conducted over the integers, strict inequalities "$<$" and "$>$" can be rewritten as "$\leq$" and "$\geq$", respectively, by changing the right hand side by 1. Also, when multiplying the whole inequality by (-1), "$\geq$" changes to "$\leq$". Note that we could even restrict the set of constraints to contain only "$\leq$"-constraints by reformulating $Ax = b$ in the form of inequalities as $Ax \leq b$ and $-Ax \leq -b$.

DEFINITION. The **feasible set** F of an IP P is the following subset of its domain:

$$F := \{\alpha := (\alpha_1, \ldots, \alpha_n) \in X_1 \times \cdots \times X_n \mid \alpha \text{ satisfies the constraints of } P(x_1, \ldots, x_n)\}.$$

That means that α is in the feasible set if

$$A_1\alpha = b_1,$$
$$A_2\alpha \leq b_2.$$

An **optimal solution** is a vector $\alpha = (\alpha_1, \ldots, \alpha_n) \in F$ such that we have

$$f(\alpha) \leq f(\beta) \qquad \text{for all } \beta = (\beta_1, \ldots, \beta_n) \in F.$$

That is, α minimizes $f(x)$ on the feasible set.

If α is an optimal solution, then $f(\alpha)$ is called the **optimal value**.

We will denote the set of optimal solutions of P by

$$\mathfrak{S}_{opt}(P) = \{\alpha \in F \mid \alpha \text{ optimal solution of } P\}.$$

For further background on linear programming see [**16**].

LEMMA 1. *Let P be an IP with feasible set F. Assume we create a new IP P' from P by restricting the feasible set to a subset F' of F (for example by adding more constraints). If α is an optimal solution of P (i.e. $\alpha \in \mathfrak{S}_{opt}$) and α' is an optimal solution of P', then $f(\alpha) \leq f(\alpha')$.*

REMARK. The lemma simply states that P' cannot have a "better" optimal value than P.

PROOF. Let α' be an optimal solution of P'. Then $\alpha' \in F' \subseteq F$. By definition $\alpha \in \mathfrak{S}_{opt}(P)$ implies $f(\alpha) \leq f(\beta)$ for all $\beta \in F$, so $f(\alpha) \leq f(\alpha')$. □

2. Group Theoretical Basics

Later, we will want to look at symmetries in the context of an integer optimization problem. For that, we will need some group theory and the notion of group actions. The basic definitions are introduced here. They can also be found in more detail in [**3**].

Let Ω be a nonempty set. A bijection of Ω onto itself is called a **permutation** of Ω. Under composition of mappings, the set of all permutations of Ω forms a group, the **symmetry group** of Ω. We denote this group by **Sym Ω**.

If $\Omega = \{1, \ldots, n\}$, where n a positive integer, $\operatorname{Sym}\Omega$ is called the **symmetric group** on $\{1, \ldots, n\}$ and is denoted by $\boldsymbol{S_n}$. A **permutation group** is a subgroup of a symmetric group.

We will use the usual notation and write the elements of S_n as a product of disjoint cycles. Our convention is to consider permutations as functions acting on the right. This means that a product $\sigma\pi$ of permutations should be read as: first apply σ and then π. For example, (243)(15)(1265)=(1)(24365).

Now we want to look at actions of a group on a set.

DEFINITION. Let G be a group and Ω be a nonempty set. A function

$$\Omega \times G \to \Omega$$

$$(a, \sigma) \mapsto a^{\sigma}$$

is called an **action** of G on Ω if it satisfies:

(1) $a^1 = a$ for all $a \in \Omega$, where 1 denotes the identity element of G,

(2) $(a^{\sigma})^{\pi} = a^{\sigma\pi}$ for all $a \in \Omega$ and all $\sigma, \pi \in G$.

In such a situation, we say G **acts** on Ω, and write $G \circlearrowleft \Omega$. The cardinality of Ω will be referred to as the **degree** of the group action.

The group G is said to act **transitively** on Ω if for any two elements a and b in Ω, there is a $\sigma \in G$ such that $a^{\sigma} = b$, otherwise the action is **intransitive**. If G acts intransitively, then two elements $a, b \in \Omega$ are in the same **orbit** if we can find a $\sigma \in G$ that satisfies $a^{\sigma} = b$. It is easily observed that any two orbits are disjoint, hence the orbits form a partition of Ω.

Now assume we have a group action of G on Ω. Then to every element σ in G we can associate a mapping $a \mapsto a^\sigma$ of Ω into itself. Using properties (1) and (2) it can easily be seen that this is a bijection. Hence we have a homomorphism $\rho : G \to \operatorname{Sym}\Omega$. If Ω has n elements, there is an isomorphism between $\operatorname{Sym}\Omega$ and S_n. Therefore, if the kernel of ρ is trivial, we can identify G with a subgroup of S_n.

Assume we have a subset Σ of Ω and $g \in G$. Then let

$$\Sigma^g := \{a^g \mid a \in \Sigma\}.$$

If H is a subgroup of G, henceforth denoted $H \leq G$, then we define

$$\Sigma^H := \{a^g \mid a \in \Sigma, g \in H\}.$$

We call Σ' a **coset**[1] of Σ under G if there is a $g \in G$ such that

$$\Sigma' = \Sigma^g.$$

It is not difficult to see that the coset relation is symmetric and transitive, i.e. if Σ'_1 and Σ'_2 are cosets of the same set Σ, then they are cosets of one another.

We call

$$\operatorname{stab}_G(\Sigma) = \{g \in G \mid \Sigma^g = \Sigma\}$$

the (setwise) **stabilizer** of Σ in G, and say $g \in \operatorname{stab}_G(\Sigma)$ **fixes** Σ.

[1]This definition of coset does not correspond to the standard notation where cosets are images aH of a subgroup $H \subseteq G$ under the action of a group element $a \in G$.

CHAPTER 3

Symmetric Restriction Algorithm

In this chapter, a formal definition of symmetry in integer programs will be given. The symmetric restriction algorithm, using these symmetries to extend the set of constraints and thereby cut down the size of the feasible set, will be developed.

1. Symmetry of an Integer Programming Problem

Throughout the following chapters, in which we will develop an algorithm constructing and adding new constraints, we will use the following linear integer program for illustration:

EXAMPLE 1. $\tilde{P}(x_1, \ldots, x_n)$:

$$\text{Maximize} \quad x_1 + x_2 + x_3 + x_4$$

subject to the constraints

$$\begin{aligned} x_1 + x_2 \qquad\qquad &\leq 1 \\ x_3 + x_4 &\leq 1 \\ x_i \in \{0, 1\} & \end{aligned}$$

The IP $\tilde{P}$ has the following feasible set:

$$\begin{aligned} F = \{&(0,0,0,0), (0,0,0,1), (0,0,1,0), \\ &(0,1,0,0), (0,1,0,1), (0,1,1,0), \\ &(1,0,0,0), (1,0,0,1), (1,0,1,0)\}. \end{aligned}$$

The notion of symmetry in an integer optimization problem seems quite natural: If we can interchange the x_i in a way such that the IP is not affected by it (i.e., the objective function and the *set* of constraints stay the same), then this permutation of the variables is a symmetry of the IP. We say such a permutation **fixes** P.

In the following example, we will write $x_i \mapsto x_j$ to denote that x_i is replaced by x_j in the IP.

EXAMPLE 2. (continuing Ex. 1) Consider the IP $\tilde{P}$ from above. One can easily see that interchanging x_1 and x_2 fixes $\tilde{P}$, as does interchanging x_3 and x_4.

Consider the map σ: $x_1 \mapsto x_3$, $x_2 \mapsto x_4$, $x_3 \mapsto x_2$, $x_4 \mapsto x_1$. The image of $\tilde{P}(x_1, x_2, x_3, x_4)$ under σ is $\tilde{P}(x_3, x_4, x_2, x_1)$:

$$\begin{aligned} \text{Maximize} \quad & x_3 + x_4 + x_2 + x_1 \\ x_3 + x_4 \qquad & \leq 1 \\ x_1 + x_2 & \leq 1 \\ x_i \in \{0, 1\} & \end{aligned}$$

which is equal to the original IP $\tilde{P}(x_1, x_2, x_3, x_4)$.

REMARK. The domains X_i are part of the constraints. So x_i and x_j are only interchangable if their domains are the same: $X_i = X_j$!

We will consider a permutation of the variables a symmetry of the IP only if we can find a 1-1 correspondance between the constraints before and after applying the permutation and if the objective function is preserved. In other words, after permuting the variables, we have to obtain exactly the same formulation of the IP. Basically, it would be sufficient to require a permutation to fix the feasible set and the objective function. As this is difficult to determine, we will use the weaker requirement of preserving the constraints.

Consider the general IP $P(x_1, \ldots, x_n)$ from page 9.

We want to find a natural way to write a permutation of P as an element of S_n. Instead of writing a symmetry of P in terms of "$x_i \mapsto x_j$", we can as well express it simply as a permutation of the indices: $i \mapsto j$, thereby just looking at permutations of $\{1, \ldots, n\}$.

LEMMA 2. *The group S_n acts on the set $\{x_1, \ldots, x_n\}$ by defining*

$$x_i^\sigma = x_{i^\sigma}$$

for all $i \in \{1, \ldots, n\}$.

EXAMPLE 3. (continuing Ex. 1) $S_4 \circlearrowleft \{x_1, \ldots, x_4\}$:

$$(x_1, x_2, x_3, x_4)^{(1324)} = (x_3, x_4, x_2, x_1)$$

PROOF. We have to show that with this definition, we really have a group action. It is easy to see that the identity fixes each x_i. Let us show that it is consistent under composition:

Let $\sigma, \pi \in S_n$. Then $x_i^{\sigma\pi} = x_{i^{(\sigma\pi)}} = x_{(i^\sigma)^\pi} = x_{i^\sigma}^\pi = (x_i^\sigma)^\pi$. $\square$

Then we can let a permutation σ act on an IP $P(x_1, \ldots, x_n)$ in the obvious way by permuting the variables according to how σ acts on their indices:

DEFINITION. $P(x_1,\dots,x_n)^\sigma := P(x_{1^\sigma},\dots,x_{n^\sigma})$

We are now ready to make the following definition.

DEFINITION. We call $\sigma \in S_n$ a **symmetry** of the integer optimization problem $P(x_1,\dots,x_n)$, if

$$P(x_1,\dots,x_n)^\sigma = P(x_1,\dots,x_n).$$

This means, a symmetry σ of P preserves $f(x)$ and permutes the constraints without changing the set of constraints.

The set of all symmetries of $P(x_1,\dots,x_n)$ forms a group, the symmetry group of $P(x_1,\dots,x_n)$, **Sym *P***. $\operatorname{Sym} P$ is a subgroup of S_n.

EXAMPLE 4. (continuing Ex. 1) $\tilde{P}$ has the following group of symmetries:

$$\operatorname{Sym}\tilde{P} = \{\mathrm{id}, (12), (34), (12)(34), (13)(24), (14)(23), (1324), (1423)\} \subset S_4$$

2. Restriction of the Feasible Set

Now that we have defined symmetry for an IP, we can look at how it extends to the feasible set and the set of optimal solutions of the IP.

A symmetry on $P(x_1,\dots,x_n)$ induces a notion of symmetry on $X_1 \times \cdots \times X_n$ by interchanging the components in a vector: Let $\sigma \in \operatorname{Sym}(P)$, $\alpha = (\alpha_1,\dots,\alpha_n) \in X_1 \times \cdots \times X_n$. Then we define

$$\alpha^\sigma = (\alpha_{1^\sigma},\dots,\alpha_{n^\sigma}).$$

EXAMPLE 5. (continuing Ex. 1) Look at $X_1 \times X_2 \times X_3 \times X_4 = \{0,1\} \times \{0,1\} \times \{0,1\} \times \{0,1\}$.

$$\begin{aligned}
(1,0,0,0) &= (0,1,0,0)^{(12)} = (0,0,1,0)^{(13)(24)} = (0,0,0,1)^{(14)(23)} \\
(1,1,0,0) &= (0,0,1,1)^{(13)(24)} \\
(1,0,1,0) &= (0,1,1,0)^{(12)} = (1,0,0,1)^{(34)} = (0,1,0,1)^{(12)(34)} \\
&\ \vdots
\end{aligned}$$

We get the following orbits in $X_1 \times \cdots \times X_4$:

$$\begin{aligned}
(0,0,0,0)^{\operatorname{Sym} \bar{P}} &= \{(0,0,0,0)\} \\
(1,0,0,0)^{\operatorname{Sym} \bar{P}} &= \{(1,0,0,0), (0,1,0,0), (0,0,1,0), (0,0,0,1)\} \\
(1,1,0,0)^{\operatorname{Sym} \bar{P}} &= \{(1,1,0,0), (0,0,1,1)\} \\
(1,0,1,0)^{\operatorname{Sym} \bar{P}} &= \{(1,0,1,0), (0,1,1,0), (1,0,0,1), (0,1,0,1)\} \\
&\ \vdots
\end{aligned}$$

There is a unique equivalence relation on $X_1 \times \cdots \times X_n$ whose equivalence classes are the orbits of the action of $\operatorname{Sym} P$.

DEFINITION. Let $\alpha = (\alpha_1, \ldots, \alpha_n)$, $\beta = (\beta_1, \ldots, \beta_n) \in X_1 \times \cdots \times X_n$ and let $f(x_1, \ldots, x_n)$ be the objective function of the IP $P(x_1, \ldots, x_n)$. We call α **equivalent** to β, denoted by $\alpha \cong \beta$, if there is a $\sigma \in \operatorname{Sym} P$ such that

$$\alpha^\sigma = \beta.$$

LEMMA 3. *Let $\alpha \in F$ and $\alpha \cong \beta$. Then $\beta \in F$ and $f(\alpha) = f(\beta)$.*

PROOF. Choose $\sigma \in \operatorname{Sym} P$ such that $\alpha^\sigma = \beta$.

We have $P(x_1, \ldots, x_n) = P((x_1, \ldots, x_n)^\sigma)$. Substituting α into $(x_1, \ldots, x_n)$ we get $P(\alpha) = P(\alpha^\sigma) = P(\beta)$. From that we see immediately that if α satisfies the constraints so does β, and that $f(\alpha) = f(\beta)$. $\square$

COROLLARY. *If α is optimal solution, then α^σ is optimal for all $\sigma \in \operatorname{Sym} P$.*

We now want to use Lemma 3 to restrict the feasible set F to a subset F' such that, even though we solve the IP for the (hopefully) much smaller feasible set F' and find a solution (hopefully) much faster, we do not lose any solutions. That means that after finding the optimal solutions of F' we have to be able to recover the set of all solutions in F.

The IP P from page 9 can also be written in the following way:

$$(P) \qquad \text{Minimize} \quad f(x)$$
$$\text{subject to } x \in F.$$

Then by restricting the feasible set we generate a new IP P':

$$(P') \qquad \text{Minimize} \quad f(x)$$
$$\text{subject to } x \in F' \subseteq F.$$

Assume we could find a way to add new constraints thereby restricting F to $F' \subseteq F$ such that F' contains a representative for each equivalence class under $\operatorname{Sym} P$. From Lemma 3 we see that the value of the objective function is constant on any equivalence class in F. It follows that if $\alpha \in F$ is an optimal solution and $\alpha' \in F'$

is a representative of α in F', then α' is certainly an optimal solution of P, and hence of the restricted Problem P', as was stated in Lemma 1. Also, if α' is an optimal solution of P', then it has to be an optimal solution of P. This is because if there was another element $\beta \in F$ such that $f(\beta) < f(\alpha')$, then there would exist a representative $\beta' \in F'$, where $\beta' \cong \beta$, and $f(\beta') < f(\alpha')$. But this contradicts the optimality of α'.

Summarizing, we get the following:

THEOREM 1. *Let P, P', F, and F' be defined as above, such that for all elements $\alpha \in F$ there is an $\alpha' \in F'$ and $\alpha \cong \alpha'$. Then we can recover the set $\mathfrak{S}_{opt}(P)$ of optimal solutions of P from the optimal solutions of P' using the relation*

$$\mathfrak{S}_{opt}(P) = (\mathfrak{S}_{opt}(P'))^{\operatorname{Sym} P}.$$

PROOF. We have $\alpha \in \mathfrak{S}_{opt}(P)$ if and only if there is an $\alpha' \in \mathfrak{S}_{opt}(P')$ such that $\alpha = (\alpha')^\sigma$ for some $\sigma \in \operatorname{Sym} P$. $\square$

The problem now reduces to finding a suitable F' satisfying the conditions of the theorem. Also, we want F' to be as small as possible, because the fewer feasible solutions we have to consider in the optimization process, the better.

Let us discuss how to generate new constraints to add to P in order to restrict F.

The idea is to choose a set $B \subsetneq \{x_1, \ldots, x_n\}$ and to look at the collection $B = B_1$, $B_2, \ldots, B_k$ of all distinct cosets of B under $\operatorname{Sym} P$.

We will call B the **restrictor (set)**, as we want to use it to restrict the feasible set.

THEOREM 2. *If we construct P' by adding the new constraints*

$$\sum_{x_i \in B_1} x_i \leq \sum_{x_i \in B_2} x_i$$

$$\vdots$$

$$\sum_{x_i \in B_1} x_i \leq \sum_{x_i \in B_k} x_i$$

to P. then the restricted feasible set F' of P' will contain a representative of each equivalence class of F.

REMARK. Of course we expect P' to be in the standard form of a linear integer programming problem as defined in Chapter 1. that is with constraints of the form $Ax \leq b$. Therefore we would have to write the new inequalities as

$$\sum_{x_i \in B_1} x_i - \sum_{x_i \in B_j} x_i \leq 0.$$

We will use the other notation for now. because it is more intuitive.

PROOF. We have to show that F' contains a representative for each isomorphism class in F.

Let $\alpha = (\alpha_1, \ldots, \alpha_n) \in F$. By definition F' contains a representative α' for α if there exists some $\sigma \in \mathrm{Sym}(P)$ such that $\alpha' = \alpha^\sigma$ and $\alpha' \in F'$. We will construct such an α'.

Choose some $m \in \{1, \ldots, k\}$ such that

$$\sum_{x_i \in B_m} \alpha_i$$

is minimal. There is a $\sigma \in \mathrm{Sym}\, P$ such that $B^\sigma = B_m$. Let $\alpha' := \alpha^\sigma = (\alpha_{1^\sigma}, \ldots, \alpha_{n^\sigma})$.

Then for each coset B_j, where $j \in \{1, \ldots, k\}$, we have:

$$\begin{aligned}
\sum_{x_i \in B_j} \alpha'_i &= \sum_{x_{i^{\sigma^{-1}}} \in B_j} \alpha'_{i^{\sigma^{-1}}} \\
&= \sum_{x_{i^{\sigma^{-1}}} \in B_j} \alpha_{(i^{\sigma^{-1}})^{\sigma}} \qquad \text{as } \alpha'_l = \alpha_{l^{\sigma}} \\
&= \sum_{x_{i^{\sigma^{-1}}} \in B_j} \alpha_i \\
&= \sum_{x_i \in B_j^{\sigma}} \alpha_i
\end{aligned}$$

That means that every $\sum_{x_i \in B_j} \alpha'_i$ is equal to $\sum_{x_i \in B_h} \alpha_i$ for some h. In particular,

$$\sum_{x_i \in B} \alpha'_i = \sum_{x_i \in B^{\sigma}} \alpha_i = \sum_{x_i \in B_m} \alpha_i$$

is minimal and hence α' is the representative for α we were looking for. $\square$

EXAMPLE 6. Let us apply the theorem to Example 1. Let $B = \{x_1, x_2\}$. There are two cosets of B under $\operatorname{Sym} P$:

$$\begin{aligned}
B_1 &= B &= \{x_1, x_2\} \\
B_2 &= B^{(1423)} &= \{x_3, x_4\}
\end{aligned}$$

According to the theorem we add one new constraint:

$$\begin{array}{ll}
\text{Maximize} & x_1 + x_2 + x_3 + x_4 \\
x_1 + x_2 & \leq 1 \\
\qquad\quad x_3 + x_4 & \leq 1 \\
\boldsymbol{x_1 + x_2} & \boldsymbol{\leq x_3 + x_4}
\end{array}$$

$$x_i \in \{0,1\}$$

As the restricted feasible set we obtain

$$F' = \{(0,0,0,0),(0,0,0,1),(0,0,1,0),(0,1,0,1),(0,1,1,0),(1,0,0,1),(1,0,1,0)\}.$$

We have now found a way to generate a subset of the feasible set of an IP without losing solutions.

There is no reason to stop after one step. If there are still symmetries in the new IP P', then we can use these to add more constraints. This process can be repeated until all symmetries are "used up".

Because of the special form of the new inequalities, in most cases adding new constraints will only cut down the set of symmetries, but not add any. If we assume that we add constraints such that no new symmetries are generated, $\operatorname{Sym} P'$ will be a subgroup of $\operatorname{Sym} P$. In this case, a symmetry of P' will be an element $\sigma \in \operatorname{Sym} P$ that fixes the new constraints in P'.

LEMMA 4. *Let $\sigma \in \operatorname{Sym} P$. We have $(P')^\sigma = P'$ if and only if $B^\sigma = B$, that is σ fixes B setwise.*

PROOF. Clearly, the permutation σ is a bijection of $\{B_1, B_2, \dots, B_k\}$.

($\Rightarrow$) Assume σ does not fix B.

As $\sigma \in \operatorname{Sym} P$, it induces a permutation of the constraints by mapping the set of constraints onto itself. We have $\sigma \in \operatorname{Sym} P'$, if there is a way to extend it to a permutation of the constraints of P' by permuting the "new" constraints with each other.

We know that σ permutes the cosets $B_1 = B, B_2, \ldots, B_k$, so there is an $m \neq 1$ such that $B_m^\sigma = B$. Then σ maps the constraint

$$\sum_{x_i \in B} x_i \leq \sum_{x_i \in B_m} x_i$$

of P' to a constraint of the form

$$\sum_{x_i \in B_j} x_i \leq \sum_{x_i \in B} x_i.$$

where j satisfies $B_j = B^\sigma$. This is not in the set of "new" constraints of P', hence σ cannot fix P'.

($\Leftarrow$) When acting on P', σ maps a constraint of the form

$$\sum_{x_i \in B} x_i \leq \sum_{x_i \in B_m} x_i$$

to another constraint of that form. Also, as $\sigma \in \operatorname{Sym} P$, it fixes the set of "old" constraints from P, as well as the objective function. Hence, $(P')^\sigma = P'$. $\square$

COROLLARY.

$$\operatorname{stab}_{\operatorname{Sym} P}(B) \subseteq \operatorname{Sym} P'$$

COROLLARY. *Assume* $\operatorname{Sym} P' \subseteq \operatorname{Sym} P$. *Then*

$$\operatorname{Sym} P' = \operatorname{stab}_{\operatorname{Sym} P}(B).$$

3. Choosing the Restrictor Set

Of course now the important question is, how do we choose the restrictor B as a subset of $\{x_1, \dots, x_n\}$ such that we get a good extension of the inequalities - and what does "good" mean in this context?

The goal is obviously to restrict the feasible set as far as possible while adding in a minimal number of constraints in order to save memory, and avoid slowing down the construction of the new inequalities and, more importantly, the optimization process.

One restriction to the choice of B we will make right away. As we know, the domains of the variables $x_1, \dots, x_n$ need not be identical. Remember from Chapter 1 that only those x_i can be permuted among each other by a symmetry of the IP that have the same domain. As we will construct inequalities relating these variables, it makes sense to restrict B to contain only elements x_i with the same domain.

We even further want to demand B to contain only elements of a transitive subset of $\{x_1, \dots, x_n\}$ under $\operatorname{Sym} P$. That will automatically take care of the problem of different domains, and will allow us later to make use of the notion of primitivity of a group action, which will be introduced and motivated during this chapter.

A restrictor B will be mapped by elements of $\operatorname{Sym} P$ to various $|B|$-element subsets of $\{x_1, \dots, x_n\}$. We will obtain k cosets, where k is bounded by the total number of $|B|$-subsets, $\binom{n}{|B|}$, and will generate $k-1$ new inequality constraints.

Note the following relation between the choice of B and the size of $\operatorname{Sym} P'$:

LEMMA 5. *Assume there are exactly k distinct cosets $B_1 = B, B_2, \dots, B_k$ of B under* $\operatorname{Sym} P$. *Then*

$$|\operatorname{Sym} P'| \geq \frac{|\operatorname{Sym} P|}{k}.$$

Proof. The group $\operatorname{Sym} P$ acts transitively on the set $\{B_1, \ldots, B_k\}$, which is the orbit of B under the action of $\operatorname{Sym} P$. Then from the orbit-stabilizer theorem we obtain:

$$|\operatorname{Sym} P| = |\operatorname{stab}_{\operatorname{Sym} P}(B)| \cdot |k|.$$

We know from Corollary 2 that $\operatorname{Sym} P'$ contains the stabilizer of B. Therefore $|\operatorname{Sym} P'| \geq |\operatorname{stab}_{\operatorname{Sym} P}(B)|$, and the Lemma follows directly. $\square$

Corollary. *If* $\operatorname{Sym} P' \subseteq \operatorname{Sym} P$, *then*

$$|\operatorname{Sym} P'| = \frac{|\operatorname{Sym} P|}{k}.$$

This means that if k is small, the size of the remaining symmetry group is big, and if k is big, then accordingly we will be left with a smaller number of symmetries.

We want to keep the number of new inequalities in each step small. This seems to be a good decision taking into account the following considerations:

Suppose we have S_n as symmetry group. If we choose B to contain c elements, we obtain $\binom{n}{c}$ cosets and generate $\binom{n}{c} - 1$ new constraints in the first step.

On the other hand, let B consist of just one element, say $B = \{x_1\}$. Then the cosets of B are all the sets $\{x_i\}$, where $2 \leq i \leq n$, and we add the $n - 2$ constraints

$$x_1 \leq x_i, \qquad 2 \leq i \leq n.$$

Our new symmetry group $stab_{S_n}(\{x_1\})$ is the symmetric group of $\{x_2, \ldots, x_n\}$. Again we choose a one element restrictor, say $\{x_2\}$, and produce the constraints

$$x_2 \leq x_i, \qquad 3 \leq i \leq n,$$

and so on. After $n-1$ steps the symmetry group will be reduced to the trivial group $\{\mathrm{id}\}$, and we have generated a total number of new constraints of

$$(n-1)+(n-2)+\ldots+2+1 = \frac{n(n-1)}{2} = \binom{n}{2}$$

But $\binom{n}{2} \leq \binom{n}{c}$ for all $c \in \{2, \ldots, k-2\}$. That means that choosing B as a c-subset with $1 < c < n-1$ makes us generate at least as many inequalities in only the first step, with possibly many more to be added in further ones, than we produce if choosing 1-subsets during all the steps until there are no symmetries left. On the other hand, it will not give us a better restriction on $X_1 \times \cdots \times X_n$.

Generalising this, the number of new constraints can be kept low by making sure that B and its cosets are pairwise disjoint. This leads us to introduce the following definitions:

Let G be a group acting transitively on a set Ω. If we can find a partition of Ω into disjoint subsets $\Sigma_1, \ldots, \Sigma_k$ such that for all $i \in \{1, \ldots, k\}$ and $g \in G$ we have $\Sigma_i^g = \Sigma_j$ for some $j \in \{1, \ldots, k\}$, then Σ_i is called a **block** for G, and we say $\{\Sigma_1, \ldots, \Sigma_k\}$ is an **invariant partition** of Ω under G.

EXAMPLE 7. We are still looking at Example 1. In Example 4 on page 17 we gave the symmetry group Sym $\bar{P}$ of the IP acting on the set of variables $\{x_1, x_2, x_3, x_4\}$.

One can easily check that $\{\{x_1, x_2\}, \{x_3, x_4\}\}$ is an invariant partition under Sym $\tilde{P}$.

For every group action on Ω there is always the partition into singletons $\{\alpha\}$ ($\alpha \in \Omega$) and the partition into one block Ω. These are called **trivial** partitions. Any other partition is called **nontrivial**.

DEFINITION. If the group G acts transitively on the set Ω, then the action of G is said to be **primitive** if there exists no nontrivial invariant partition of Ω under G, **imprimitive** otherwise.

Note that the terms "primitive" and "imprimitive" are only used with reference to a transitive group action.

In fact, most transitive group actions are imprimitive. More details and examples on the topic can be found in [**16**].

If G acts imprimitively on Ω, then we can make the following observation: If $\{\Sigma_1, \ldots, \Sigma_k\}$ is an invariant partition of Ω under G, then all the Σ_i $(1 \leq i \leq k)$ have the same cardinality, say c. This is because for all i, $1 < i \leq k$, there is a $g \in G$ mapping the elements of Σ_1 bijectively onto the elements of Σ_i. Therefore we can characterize a partition by its block size c. We call a nontrivial partition with maximal block size a **maximal** partition.

How do we use all this to pick the restrictor? We have an action of the group Sym P on the set $\{x_1, \ldots, x_n\}$. As Sym $P \neq \{\mathrm{id}\}$ we can find a nontrivial orbit $\bar{X} \subseteq \{x_1, \ldots, x_n\}$. Obviously, Sym P acts on $\bar{X}$ by restricting the action on $\{x_1, \ldots, x_n\}$ to $\bar{X}$. If this action is imprimitive, we choose a maximal invariant partition of $\bar{X}$ under Sym P. Say it has k blocks $\{B_1, \ldots, B_k\}$ of size c. We have $|\bar{X}| = k \cdot c$. If we choose the restrictor to be B_1, then its cosets under $\operatorname{Sym} P$ will be $B_2, \ldots, B_k$, and we generate $k-1$ inequalities, where $k-1$ is strictly less than n. In fact it is strictly less than n/c, where $c > 1$.

Now if the action of $\operatorname{Sym} P$ on $\bar{X}$ is primitive, we will not be able to find a nontrivial invariant partition. But we still want the cosets of B to be pairwise disjoint, so we have to choose the restrictor B to be $\{x_i\}$ for some $x_i \in \bar{X}$. Its cosets will be the other singletons in $\bar{X}$ (so the partition into cosets is nothing but the trivial partition into singletons), and we generate the $|\bar{X}| - 1$ inequalities

$$x_i \leq x_j \qquad \text{for all } x_j \in \bar{X} \setminus \{x_i\}.$$

We may now summarize the algorithm.

4. The Algorithm

Given a linear integer program $P(x_1, \ldots, x_n)$, Figure 3.1 shows the algorithm for stepwise symmetric augmentation of P. It takes the IP P as input and augments it in several executions of a loop, until $\operatorname{Sym} P = \{\mathrm{id}\}$.

To illustrate the algorithm, we want to apply it to the IP from the previous examples:

EXAMPLE 8. Remember that we were looking at the IP $\tilde{P}(x_1, \ldots, x_n)$:

$$\text{Maximize} \quad x_1 + x_2 + x_3 + x_4$$

under the constraints

$$\begin{aligned} x_1 + x_2 \qquad\quad &\leq 1 \\ x_3 + x_4 &\leq 1 \end{aligned}$$

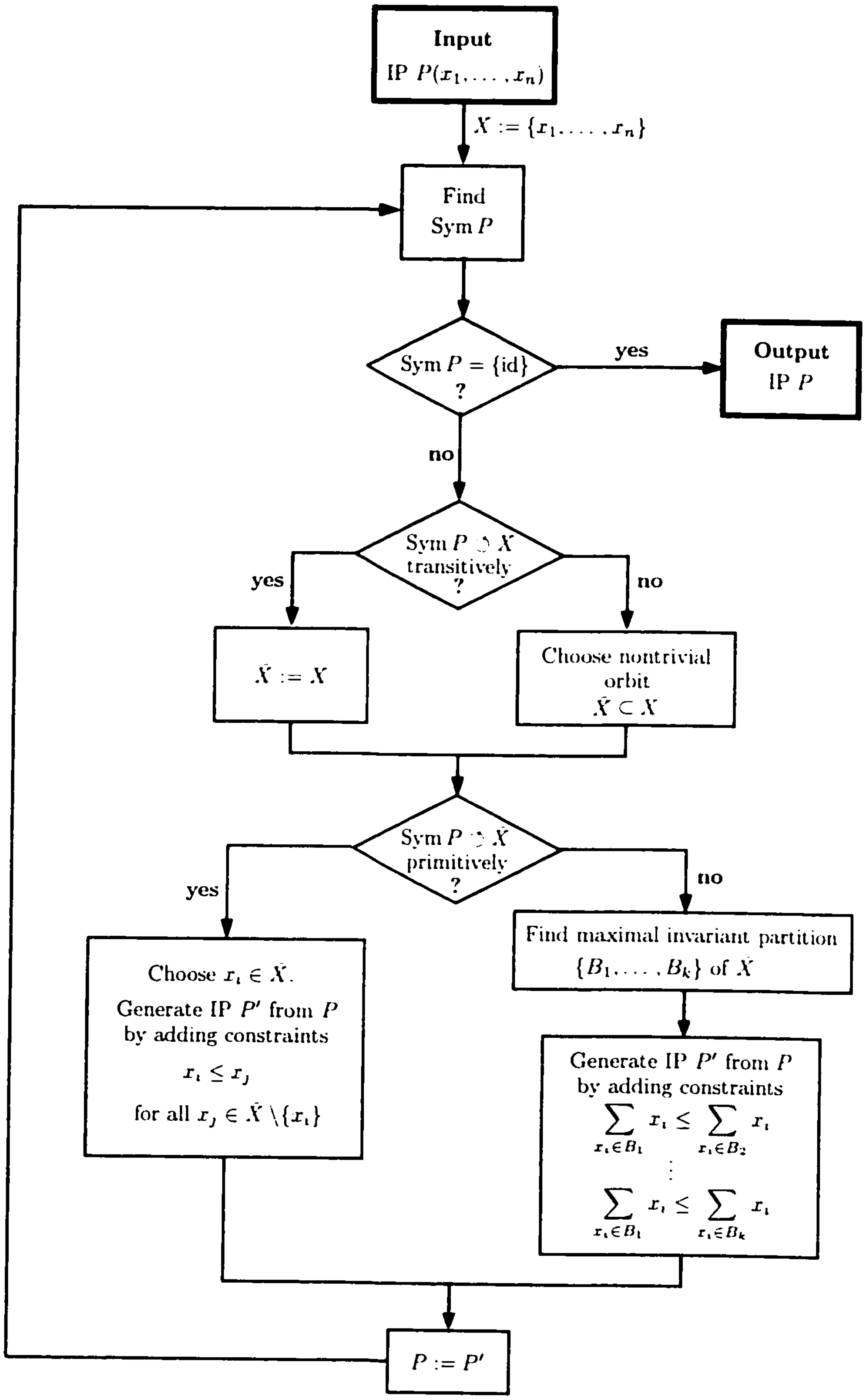
Input
IP $P(x_1,\dots,x_n)$
$X := \{x_1,\dots,x_n\}$
Find Sym P
Sym $P = \{\text{id}\}$?
yes
Output IP P
no
Sym $P \circlearrowright X$ transitively ?
yes
no
$\tilde{X} := X$
Choose nontrivial orbit $\tilde{X} \subset X$
Sym $P \circlearrowright \tilde{X}$ primitively ?
yes
no
Choose $x_i \in \tilde{X}$. Generate IP P' from P by adding constraints $x_i \leq x_j$ for all $x_j \in \tilde{X} \setminus \{x_i\}$
Find maximal invariant partition $\{B_1,\dots,B_k\}$ of $\tilde{X}$
Generate IP P' from P by adding constraints $\sum_{x_i \in B_1} x_i \leq \sum_{x_i \in B_2} x_i$ ⋮ $\sum_{x_i \in B_1} x_i \leq \sum_{x_i \in B_k} x_i$
$P := P'$

FIGURE 3.1. Symmetric Augmentation Algorithm

$$x_i \in \{0, 1\}$$

The symmetry group $\operatorname{Sym} P$ acts transitively on $\{x_1, \dots, x_n\}$. In Chapter 2, Example 6, we had already generated a new constraint. Notice that the choice of the restrictor B in that example satisfies the conditions in the symmetric augmentation algorithm: The cosets of B form a invariant partition for the imprimitive action of $\operatorname{Sym} P$. So in the first step the algorithm picks the partition $\{x_1, x_2\}$, $\{x_3, x_4\}$ and adds the constraint:

$$x_1 + x_2 \leq x_3 + x_4.$$

Denote the augmented IP by $\tilde{P}_1$. The new group of symmetries is $\operatorname{Sym} \tilde{P}_1 = \{\mathrm{id}, (12)(34), (12), (34)\}$. The action is nontransitive, so the orbit $\tilde{X}_1 = \{x_1, x_2\}$ is chosen. As this is a primitive case, we pick $B = \{x_1\}$ and extend $\tilde{P}_1$ by

$$x_1 \leq x_2.$$

The group of symmetries of this new IP $\tilde{P}_2$ has shrunk to $\operatorname{Sym} \tilde{P}_2 = \{\mathrm{id}, (34)\}$. There is only one non-trivial orbit $\tilde{X}_2 = \{x_3, x_4\}$ left, and as this is again primitive, choose $B = \{x_3\}$ and add

$$x_3 \leq x_4.$$

With that third step there are no symmetries left: We have $\operatorname{Sym} \tilde{P}_3 = \{\mathrm{id}\}$. The algorithm terminates.

That means we now want to optimize the following IP $\tilde{P}'$:

$$\begin{array}{llll}
\text{Maximize} & x_1 + x_2 + x_3 + x_4 & & \\
x_1 + x_2 & & \leq 1 & \\
& x_3 + x_4 & \leq 1 & \\
\boldsymbol{x_1 + x_2} & & \leq \boldsymbol{x_3 + x_4} & \\
& \boldsymbol{x_1} & \leq \boldsymbol{x_2} & \\
& \boldsymbol{x_3} & \leq \boldsymbol{x_4} & \\
& x_i \in \{0, 1\} & &
\end{array}$$

By adding the additional constraints we have restricted the feasible set F, given in Example 1 on page 14, to the smaller set

$$F' = \{(0,0,0,0), (0,0,0,1), (0,1,0,1)\}.$$

We see that F' contains exactly one representative from each orbit in the feasible set. F could be reconstructed from F' by applying $\operatorname{Sym} \tilde{P}$ to the elements of F'.

The solution of the optimization is now easily found to be $\mathfrak{S}_{opt}(\tilde{P}') = \{(0,1,0,1)\}$. We obtain $\mathfrak{S}_{opt}(\tilde{P})$ as the orbit of $(0,1,0,1)$ under $\operatorname{Sym} \tilde{P}$:

$$\mathfrak{S}_{opt}(\tilde{P}) = \{(0,1,0,1), (0,1,1,0), (1,0,0,1), (1,0,1,0)\}$$

It can be readily checked, that this is the optimal solution of the integer optimization problem $\tilde{P}$ in Example 1.

CHAPTER 4

Implementation

In this chapter we want to discuss our implementations of the symmetric restriction algorithm which was developed in Chapter 3. It was implemented in a program in the programming language C. The source code is given in the Appendix.

In the first section, we will discuss how to translate the problem into the language of matrices. The program takes as input the integer optimization problem in the form of an integer matrix, and repeatedly determines the group of symmetries and adds new constraints in the form of additional rows to the input matrix, until no more symmetries can be found for the augmented IP.

Finding the symmetry group is the most complex and difficult part of the program. We will show that it reduces to finding a certain automorphism group of an integer matrix. There are software packages available to determine such an automorphism group. We use one of them, ISOM [**12**], which was developped by C. Lam and L. Thiel to provide data structures and routines for efficient handling of permutation groups and isomorphism testing of combinatorial objects. It implements concepts like the permutation group algorithms specified for example in [**1**] or [**10**] and provides an interface for C. We will devote a short section to giving the basic ideas used to find the group of symmetries as the speed of our program greatly depends on this part. Then we will give a brief introduction to our implementation to help understanding the source code.

1. Finding Sym P

Given an integer optimization problem in the form (*) from page 9, it is converted into an integer matrix M of the following form:

$$M = \left(\begin{array}{c|c} c^T & * \\ \hline A_1 & b_1 \\ \hline A_2 & b_2 \end{array}\right)$$

Note that the * in the right upper corner is there to complete the matrix. The element is insignificant and can be substituted by any integer in the range of the other elements in M.

This is a matrix with $n+1$ columns, the first n of which correspond to the coefficients of the variables $x_1, \ldots, x_n$, the rightmost one to the constant right hand side vector, and with $(1+m_1+m_2)$ rows, the first one specifying the objective function, the next m_1 rows the equality constraints, and the last m_2 rows the inequality constraints.

In Chapter 3 we gave a definition of symmetry of an IP. Let us now formulate, what this notion of symmetry means relative to the matrix notation of the IP. The permutation of variables in the IP corresponds to a permutation of the respective columns in M, where the constant column on the right always stays fixed. Then the objective function and the set of constraints are preserved by such a permutation, if we can reorder the set of constraints, such that we get exactly the original IP - or in matrix formulation, if we can permute the rows of the matrix in a way that lets us recover the matrix M. We have to be careful, though, not to mingle objective function, equalities and inequalities with each other in this permutation. In M, this partition into rows and columns that can be permuted is indicated by lines partitioning the matrix. That means that we can interchange rows (columns) with one another that are not separated by one of the lines.

LEMMA 6. *A symmetry of the IP given by matrix M is in one-to-one correspondence to a permutation of the first n columns in M such that there is a permutation of the rows to recover M which respects the row partition.*

In other words, an automorphisms of M under column and row permutations gives rise to a symmetry of the IP by restricting it to its action on the first n columns.

In the input, the initial row partition has to be specified. Note that the distinction between objective function, equality and inequality constraint does not matter at all for the algorithm. To find the symmetries and do augmentations, one only needs to know which rows can be interchanged and which cannot. For more generality, our program requires as input an initial partition of rows and columns (including the original partition of the columns into the first n ones and the right constant vector), which leaves the possibility of encoding additional information if desired.

To find the respective automorphisms of M, we use routines implemented in ISOM. The heart of the algorithm is a combination of invariant analysis and a branch-and-bound search. Partial permutations are generated and extended step by step in a depth-first fashion, while automorphism pruning and branch-and-bound pruning cut down the size of the search. The greatest gain in speed is due to invariant analysis, which reduces the number of possible permutations that have to be considered by looking at invariants of a combinatorial object not permitting certain permutations. The theoretical aspects of the branch-and-bound algorithm applied by ISOM can be found in detail in [**12**, pp.51 ff] and [**2**]. In the literature, the concept of invariant analysis is described for the similar case of graph automorphisms for example in [**15**].

2. Implementation Details

This shall give a brief introduction to some details of the source code of our program to help understanding it.

The program reads in the IP P in the form of the integer matrix M and an initial partition of the rows and columns from an input file. In a loop in the main routine, it then performs several augmentations, adding new rows to the matrix until the symmetry group of the augmented matrix is trivial. To find the new automorphism group after each augmentation step, we assume $\operatorname{Sym} P' \subseteq \operatorname{Sym} P$, so we only have to restrict the symmetry group to a subgroup instead of finding a whole new group. The final augmented matrix corresponds to an IP P' with no symmetries. This IP P' can then be solved using an integer optimization software package (for example CPLEX [**8**]), and if a complete set of optimal solutions $\mathfrak{S}_{opt}(P')$ has been determined, then all solutions for P can be found by applying $\operatorname{Sym} P$ to $\mathfrak{S}_{opt}(P')$.

As already mentioned, the ISOM package will provide us with data structures and routines to efficiently deal with permutations. To store a permutation σ of degree k, it offers the data type *ptr_to_permvect*, which stores σ as a vector $(\sigma(1), \ldots, \sigma(k))$.

Permutation groups are represented by the type *ptr_to_perm_gp*.

The program has to deal with permutations of the columns and rows of a matrix. It addressed those columns and rows by numbering them in the following way

$$\begin{array}{c|c}
 & 1 \quad 2 \quad \ldots \quad n \quad n+1 \\
\begin{array}{l} n+2 \\ n+3 \\ \ldots \\ n+m_1+m_2+2 \end{array} & \left(\begin{array}{c} \\ \\ \ldots \\ \\ \end{array} \right)
\end{array}$$

Let $C_1, \ldots, C_k$ be the initial partition of the columns and $R_1, \ldots, R_l$ the initial partition of the rows. Then the permutation group of the input matrix M is initialized to the direct product group

$$\mathrm{Sym}\, C_1 \times \cdots \times \mathrm{Sym}\, C_k \times \mathrm{Sym}\, R_1 \times \cdots \times \mathrm{Sym}\, R_l.$$

The routine *make_autogp* uses external routines applying invariant analysis and isomorphism testing to return the subgroup of automorphisms of M of this direct product group. Once this group is found, *truncate_group* can be used to restrict it to its action on the first n columns to get the symmetry group of the IP.

The program will then find and add the new constraints to the IP in form of additional rows of the matrix.

After each augmentation of the matrix, *make_autogp* determines the new automorphism group from the previous one by restricting it to those permutations that are still automorphisms of the augmented matrix. Augmentations are performed, until *make_autogp* find a trivial automorphism group, at which point the program terminates.

The routine *Print_CPLEX_output* was added to write the final output IP into a mps-file, which is one of the input formats for the CPLEX optimizer [8].

n	time (s)
< 100	< 1
< 200	< 2
< 500	< 40

TABLE 4.1. Time of symmetric augmentation for an IP with n variables.

The program dealt with all test examples very efficiently. Table 4.1 shows the running times we obtained from examples. It can be easily seen that the time needed for symmetric augmentation is negligible, relative to the time involved in the optimization (see also Table 5.2, 47).

CHAPTER 5

Results

The algorithm that was developed in this thesis was motivated by the work of Ka Leung Ma [**13**, Chapters 5, 6]. There, an attempt was made to solve the football pool problem, which we will introduce below, for 6 and 7 matches by generating a number of integer optimization problems, or at least to improve the known upper bound for the solution. The solution to each of those optimization problems represents a (not necessarily optimal) solution for the football pool. Unfortunately, as integer optimization is very involved (compare Chapter 1), solutions could not be found in all cases. A common feature in all those integer programs is, though, that there are symmetries in the variables due to the large amount of symmetry in the original problem. Our algorithm shall make use of this to solve some of the so far unsolved cases.

1. The Football Pool Problem

The football pool problem is one of the classical problems of combinatorial coding theory. It arises from the following question: One wants to forecast the outcome of n football matches. Each match can end in three possible ways: either a win or a loss for the home team, or there is a tie between the two teams playing. What is the smallest number of bets to be made to, whatever the outcome, have at least in one of them $n-1$ results predicted correctly?

In mathematical language that translates to the following: Consider the set $\mathbb{Z}_3^n$ of n-tuples (or **words**) $(x_1, \ldots, x_n)$ with $x_i \in \mathbb{Z}_3 = \{0, 1, 2\}$. The numbers 0, 1, 2 represent the three outcomes a match can have, so $\mathbb{Z}_3^n$ is the set of all possible results of n matches. The **Hamming distance** $\mathrm{d}(x, y)$ of two words x, $y \in \mathbb{Z}_3^n$ is defined to be the number of coordinates in which x and y differ. A **covering code** C of $\mathbb{Z}_3^n$ is a subset of words such that each element in $\mathbb{Z}_3^n$ has a Hamming distance at most 1 to some word in C. We want to find a covering code of $\mathbb{Z}_3^n$ with a minimal number of elements.

This is a special case of the more general formulation to find a covering code C of $\mathbb{Z}_q^n$, where $\mathbb{Z}_q = \{0, 1, \ldots, q-1\}$, with the property that every element in $\mathbb{Z}_q^n$ is within Hamming distance at most r from at least one codeword in C. Then r is called the **covering radius**. A common notation for the minimum size of a covering code of $\mathbb{Z}_q^n$ is

$$K_q(n, r) = \min\{\quad |C| \quad | \, C \text{ is a covering code with covering radius r of } \mathbb{Z}_q^n\}.$$

In this notation, the football pool problem is to find $K_3(n, 1)$.

In the last decades, the football pool problem has been solved for $n = 1, \ldots, 5$. Also it has been shown, that in the case $n = \frac{1}{2}(3^k - 1)$ there is a minimal covering code of size 3^{n-k} (cf. [**14**], [**20**]). For all other n the size of a minimal covering has not yet been determined. During the years, increasingly better lower and upper bounds have been established for small n. Table 5.1 gives an overview of these results. There, the most common methods that were applied are simulated annealing and tabu search.

n	nb. of words	best lower - upper bounds	reference
1	3	1	
2	9	3	
3	27	5	
4	81	9	
5	243	27	[**9**]
6	729	65 - 73	[**17**], [**11**]
7	2187	153 - 186	[**5**], [**11**]
8	6561	393 - 486	[**11**]
9	19683	1048 - 1356	
10	59049	2814 - 3645	
11	177147	7767 - 9477	[**18**]
12	531441	21395-27702	[**18**]

TABLE 5.1. Best known bounds for $K_3(n, 1)$.

In [**13**], an approach via the dominating set problem like in the rooks example in Chapter 1 is suggested.

2. Dominating Sets and Optimization

In Chapter 1 we have already given an introduction to the dominating set problem. We will discuss now, how the question to find $K_q(n, r)$ can be translated into a problem of that type.

Given a graph $G(V, E)$, the distance of two vertices is the length of the shortest path connecting them. A **dominating set of radius r** is a subset $D \subseteq V$ such that any vertex in G has distance at most r from some element in D.

Let $\Gamma_{n,q}(V, E)$ be the graph, where $V = \mathbb{Z}_q^n$ and two vertices x, y are adjacent if they only differ in one coordinate, that is $E = \{\{x, y\} \mid d(x, y) = 1\}$. This graph is called the **rook domain graph**. Note that the rooks graph R_n in Chapter 1 is a special case of the rook domain graph: $R_q = \Gamma_{2,q}$.

It is not difficult to see that the distance between two vertices x, y in this graph equals the Hamming distance of the words x and y, so we can unambiguously denote both by $d(x, y)$. A covering code of radius r of $\mathbb{Z}_q^n$ corresponds to a dominating set of radius r of the graph $\Gamma_{n,q}(V, E)$.

In this thesis, we want to investigate $\Gamma_{6,3}$, the football pool graph for 6 matches. Theoretically, we could now follow the rooks graph example and transform this dominating set problem into an integer program, to solve it by optimization methods. Unfortunately, though, this would give us an IP in $|\mathbb{Z}_3^6| = 3^6 = 729$ variables, which is by far beyond our optimization capacities, even if we apply symmetric augmentation. [13] suggests a method of using the automorphism group of $\Gamma_{n,q}$ to reduce the problem into a number of smaller ones for graphs with less vertices using orbit graphs. For that, let us discuss the automorphism group of $\Gamma_{n,q}(V, E)$. Just from the construction of the graph we can already guess that it will be very big.

Consider the following operations on $V = \mathbb{Z}_q^n$:

(1) Let $\rho = (\rho_1, \dots, \rho_n) \in S_q^n$, where $\rho_i \in S_q \circlearrowleft \mathbb{Z}_q$, act on an element $x = (x_1, \dots, x_n) \in \mathbb{Z}_q^n$ by componentwise permuting the entries:

$$x^\rho = (x_1^{\rho_1}, \dots, x_n^{\rho_n})$$

(2) Let $\pi \in S_n$ permute the components of an element $x = (x_1, \dots, x_n) \in \mathbb{Z}_q^n$:

$$x^\pi = (x_{1^\pi}, \dots, x_{n^\pi})$$

LEMMA 7 (Ka Leung Ma [13]). *The operations ρ and π are automorphisms of the graph $\Gamma_{n,q}(V, E)$.*

PROOF. We have to show that ρ and π preserve adjacency.

Assume $x = (x_1, \ldots, x_n)$, $y = (y_1, \ldots, y_n) \in \mathbb{Z}_q^n$ are adjacent. That means $d(x, y) = 1$, in other words $x_i = y_i$ for all but exactly one $i \in \{1, \ldots, n\}$, say $x_j \neq y_j$.

(1) We have $x_i^{\rho_i} = y_i^{\rho_i}$ for all $i \neq j$, and $x_j^{\rho_j} \neq y_j^{\rho_j}$, so $d(x^\rho, y^\rho) = 1$, and x^ρ and y^ρ are adjacent.

(2) Similarly, the images x^π and y^π differ only in the j^πth component, so they are adjacent. □

Together, they form a group $G = \{\rho\pi \mid \rho \in S_q^n, \pi \in S_n\}$ of automorphisms of $\Gamma_{n,q}(V, E)$, which acts on the elements of V by first applying ρ, then π. The product of two group elements $\rho_1\pi_1$, $\rho_2\pi_2$ can be defined as follows:

$$(\rho_1\pi_1)(\rho_2\pi_2) = (\rho_1\rho_2^{\pi^{-1}})(\pi_1\pi_2),$$

where S_n acts on S_q^n by permuting the components of the elements of S_q^n in a natural way. This group is called the *wreath product* $S_q wr_{\{1,\ldots,n\}} S_n$ of S_q and S_n. It is not difficult to show that G is of order $(q!)^n n!$.

Let $\operatorname{Aut}\Gamma_{n,q}$ denote the automorphism group of $\Gamma_{n,q}(V, E)$.

LEMMA 8 (Ka Leung Ma [**13**]).

$$\operatorname{Aut}\Gamma_{n,q} = G.$$

PROOF. We have shown that $G \subseteq \operatorname{Aut}\Gamma_{n,q}$. Hence for equality it suffices to show $|\operatorname{Aut}\Gamma_{n,q}| = (q!)^n n!$. See [**13**]. □

3. Orbit Graphs and Weighted Dominating Sets

We will now explain, how to use construct orbit graphs and how to use them to reduce the size of the dominating set problem of a graph.

We have already defined the dominating set problem for general graphs. Now we want to define a similar notion for weighted graphs.

Let a graph $G(V, E)$ and a weight function $w : V \to \mathbb{R}^+$ be given, where w assignes a positive real weight $w(x)$ to each vertex $x \in V$. We define the **weight** of a set D of vertices to be

$$w(D) := \sum_{x \in D} w(x).$$

Then the **weighted dominating set problem** is to find a dominating set D for G that has minimal weight $w(D)$.

Assume a graph $G(V, E)$ and its automorphism group $\operatorname{Aut} G$ are given. Let H be a subgroup of $\operatorname{Aut} G$. If we denote the orbits of V under the action of H by $\mathcal{O}_1$, ..., $\mathcal{O}_m$, then we can define the **orbit graph** of G with respect to H as follows: Let $G_{orb}(V_{orb}, E_{orb})$ be the graph with vertex set $V_{orb} = \{1, \ldots, m\}$ and edge set $E = \{\{i, j\} \mid \{x, y\} \in E \text{ for some } x \in \mathcal{O}_i, y \in \mathcal{O}_j\}$. Each vertex in G_{orb} will be assigned the weight $w(i) = |\mathcal{O}_i|$.

Lemma 9 (Ka Leung Ma [13]). *If D' is a dominating set of G_{orb}, then*

$$D := \bigcup_{i \in D'} \mathcal{O}_i$$

is a dominating set of G, and $|D| = w(D')$. Moreover, H fixes D setwise.

PROOF. We will show the following: If $\{i,j\} \in E_{orb}$, then for each $y \in \mathcal{O}_j$, there is an $x \in \mathcal{O}_i$ such that $\{x,y\} \in E$. From this it is clear that if vertex $j \in V_{orb}$ is dominated by some vertex $i \in D'$, then each $x \in \mathcal{O}_j$ will be dominated by some $y \in \mathcal{O}_i \subseteq D$.

Assume $\{i,j\} \in E_{orb}$ and $y \in \mathcal{O}_j$. By definition of E_{orb}, there is an edge $\{x',y'\} \in E$ with $x' \in \mathcal{O}_i$ and $y' \in \mathcal{O}_j$. Now y and y' are both in $\mathcal{O}_j$, so there must be a $\sigma \in H$ such that $(y')^\sigma = y$. But $H \subseteq \operatorname{Aut} G$, so $\{(x')^\sigma, (y')^\sigma\} = \{x,y\}$ is in E and $(x')^\sigma = x \in \mathcal{O}_i$.

It is clear, that D has size $w(D') = \sum_{i \in D'} |\mathcal{O}_i|$.

As D is the union of orbits under H, it has to be fixed by H. □

That means that by generating orbit graphs of a graph G, we can find dominating sets of G by solving smaller dominating set problems for which it is more likely that a solution can be found with available methods.

Note though, that we can only find dominating sets that are fixed by some subgroup of $\operatorname{Aut} G$. So we will not necessarily be able to find a dominating set of minimum size for G. But still every dominating set that we find using orbit graphs will give us an upper bound on the size of an optimal dominating set of the graph.

To reduce the number of subgroups that have to be taken into consideration for the generation of orbit graphs, it can be shown that it is sufficient to consider subgroups of $\operatorname{Aut} G$ up to conjugation.

Remember that two subgroups H, $K \subseteq \operatorname{Aut} G$ are **conjugate** if there is a $g \in \operatorname{Aut} G$ such that $K = g^{-1}Hg$.

LEMMA 10 (Ka Leung Ma [**13**]). *If the subgroups H, $K \subseteq \operatorname{Aut} G$ are conjugate, then the orbit graph $G_{orb}(H)$ of G with respect to H is isomorphic to the orbit graph $G_{orb}(K)$ with respect to K.*

PROOF. See [**13**]. □

COROLLARY. *The dominating sets of $G_{orb}(H)$ and $G_{orb}(K)$ of a certain size t are in bijective correspondence.*

Note that the bigger the subgroup H, the more elements are contained in the orbit of each vertex, i.e. the smaller the orbit graph with respect to H will be. For smaller orbit graphs, it will be easier to find a dominating set. On the other hand, we stated in Lemma 9, that the corresponding dominating set in the original graph G will be fixed by H, so with a big H the chance to find an optimal dominating set decreases.

4. Orbit Graphs and Weighted Dominating Sets of $\Gamma_{6,3}$

In [**13**], 220 orbit graphs are constructed for $\Gamma_{6,3}$, ranging from 24 vertices to 486 vertices.

As was just discussed, the bigger a subgroup of $\Gamma_{6,3}$ we choose to generate an orbit graph, the less probable it is to find a good dominating set for the original graph through it. For this reason, all orbit graphs for the smallest possible subgroups were generated, the cyclic subgroups. In other words, a representative h of each of the 220 conjugacy classes in $\operatorname{Aut} \Gamma_{6,3}$ was determined, and the orbit graph for $H = \langle h \rangle$ constructed.

TABLE 5.2. CPLEX performance on the orbit graphs of $\Gamma_{6,3}$ with and without symmetric augmentation.

				CPLEX *w/o symm. aug.*		*CPLEX* *with symm. aug.*	
graph	*number variables*	*size* Aut G	*new rows*	*time*	*space*	*time*	*space*
80	72	72	7	23 s		15 s	
91	72	288	13	41 s		10 s	
102	72	72	7	17 s		10 s	
19	81	108	8	17 s		37 s	
90	81	1296	33	62 s		10 s	
174	123	72	8	7 min		5 min 45 s	
29	126	31104	17	115 min	30 MB	17 min	3.5 MB
27	135	31104	17	>140 min	33 MB	13 min	2 MB
35	135	3456	17	>50 min	>14 MB	12 min	4MB
28	162	31104	23	>18 h	>350 MB	126 min	17.5 MB
44	198	864	15			9 h 35 min	45 MB
53	198	432	10			>34 h no solution found	>450 MB

Table 5.3 on page 48 gives a list of those orbit graphs for which no optimal solution could be found in [**13**], together with the results obtained by applying symmetric augmentation. For the generators h of the orbit graphs refer to [**13**]. For all the smaller graphs solutions were determined. In the case of the graphs with a large number of vertices, CPLEX is not necessarily able to find the domination number even if symmetric augmentation is applied. It can be used, though, to establish lower and upper bounds.

So far, we could not improve the upper bound on the size of a minimal dominating set for $\Gamma_{6,3}$ to less than 73.

In Table 5.2 the performance of CPLEX on some of the orbit graphs is shown, comparing time and space expenses of optimization without and with applying symmetric augmentation.

TABLE 5.3. New domination numbers for orbit graphs of $\Gamma_{6,3}$ using CPLEX and symmetric augmentation.

	weight	*weighted dom. set size w*	
graph	*distribution*	[**13**]	*w/ symm. augm.*
1	$5^{144}1^9$	≤ 99	77
10	3^{243}	≤ 81	$w \leq 78$
11	$3^{216}1^{81}$	≤ 95	$^*71 \leq w \leq 80$
12	$3^{240}1^9$	≤ 99	$71 \leq w \leq 81$
18	3^{243}	≤ 96	$72 \leq w \leq 81$
19	9^{81}	≤ 90	81
20	3^{243}	≤ 81	$75 \leq w \leq 81$
21	3^{243}	≤ 81	$75 \leq w \leq 81$
22	3^{243}	≤ 102	$72 \leq w \leq 78$
23	3^{243}	≤ 81	$72 \leq w \leq 81$
24	3^{243}	≤ 96	$72 \leq w \leq 81$
25	3^{243}	≤ 99	$72 \leq w \leq 75$
26	3^{243}	≤ 108	$69 \leq w \leq 81$
27	$6^{108}3^{27}$	≤ 81	81
28	$6^{81}3^{81}$	≤ 81	81
29	$6^{117}3^9$	≤ 81	81
30	$6^{120}3^3$	≤ 81	78
34	$6^{81}3^{81}$	≤ 81	81
35	$6^{108}3^{27}$	≤ 84	81
36	$12^{54}6^93^9$	≤ 81	81
37	$6^{117}3^9$	≤ 81	78
38	$6^{81}3^{81}$	≤ 81	81
39	$6^{108}3^{27}$	≤ 81	78
40	$6^{108}3^{27}$	≤ 81	$78 \leq w \leq 81$
41	$6^{117}3^9$	≤ 81	78
42	$6^{96}3^{24}2^{36}1^9$	≤ 81	80
43	$6^{108}2^{27}1^{27}$	≤ 81	80
44	$6^{72}3^{72}2^{27}1^{27}$	≤ 81	80
45	$6^{108}2^{36}1^9$	≤ 81	80
46	$6^{108}2^{39}1^3$	≤ 85	78
47	$6^{104}3^82^{39}1^3$	≤ 87	79
48	$6^{108}2^{40}1^1$	≤ 86	77
53	$6^{72}3^{72}2^{27}1^{27}$	≤ 102	$79 \leq w \leq 80$
54	$6^{108}2^{36}1^9$	≤ 81	78
55	$6^{96}3^{24}2^{36}1^9$	≤ 79	78
56	$6^{108}2^{39}1^3$	≤ 87	78
57	$6^{120}2^41^1$	≤ 77	77
58	$6^{116}3^82^31^3$	≤ 80	82
60	$6^{116}3^82^31^3$	≤ 84	80

	weight	*weighted dom. set size w*	
graph	*distribution*	[**13**]	*w/ symm. augm.*
77	$6^{108}3^{27}$	≤ 87	81
78	$6^{81}3^{81}$	≤ 90	81
79	$6^{117}3^{9}$	≤ 81	81
80	$12^{54}6^{9}3^{9}$	≤ 81	81
83	$6^{120}3^{3}$	≤ 81	81
84	$6^{108}3^{27}$	≤ 87	87
87	$6^{117}3^{9}$	≤ 90	90
90	$12^{54}3^{27}$	≤ 111	111
91	$12^{54}6^{9}3^{9}$	≤ 105	105
102	$12^{54}6^{9}3^{9}$	≤ 99	99
107	$6^{108}3^{27}$	≤ 87	87
110	$6^{117}3^{9}$	≤ 90	90
112	$12^{54}3^{27}$	≤ 111	111
114	$12^{54}6^{9}3^{9}$	≤ 105	105
120	$12^{54}6^{9}3^{9}$	≤ 105	105
125	$6^{108}3^{27}$	≤ 87	87
128	$12^{54}6^{9}3^{9}$	≤ 99	99
131	$6^{117}3^{9}$	≤ 90	90
137	$6^{108}3^{27}$	≤ 81	81
138	$6^{120}3^{3}$	≤ 81	78
139	$6^{81}3^{81}$	≤ 81	78
140	$6^{117}3^{9}$	≤ 81	78
142	$12^{54}3^{27}$	≤ 99	96
148	$6^{108}3^{27}$	≤ 84	81
149	$12^{54}6^{9}3^{9}$	≤ 84	84
150	$6^{81}3^{81}$	≤ 81	78
151	$6^{81}3^{81}$	≤ 84	81
152	$6^{117}3^{9}$	≤ 81	78
153	$6^{117}3^{9}$	≤ 84	81
154	$6^{108}3^{27}$	≤ 81	78
155	$6^{108}3^{27}$	≤ 87	81
156	$6^{108}3^{27}$	≤ 81	78
157	$6^{81}3^{81}$	≤ 84	81
158	$6^{117}3^{9}$	≤ 81	78
160	$12^{54}6^{9}3^{9}$	≤ 84	84
161	$6^{81}3^{81}$	≤ 81	$75 \leq w \leq 81$
162	$6^{108}3^{27}$	≤ 96	81
163	$6^{120}3^{3}$	≤ 81	78
164	$6^{81}3^{81}$	≤ 81	$78 \leq w \leq 81$
165	$6^{117}3^{9}$	≤ 84	81
166	$6^{108}3^{27}$	≤ 81	78
167	$6^{117}3^{9}$	≤ 78	78

	weight	*weighted dom. set size w*	
graph	*distribution*	**[13]**	*w/ symm. augm.*
168	$6^{121}3^{1}$	≤ 81	78
169	$6^{81}3^{81}$	≤ 81	81
170	$6^{120}3^{3}$	≤ 81	81
173	$6^{81}3^{81}$	≤ 81	78
174	$6^{120}3^{3}$	≤ 78	78
175	$6^{120}3^{3}$	≤ 81	81
176	$6^{81}3^{81}$	≤ 81	81
177	$6^{121}3^{1}$	≤ 81	81
178	$6^{117}3^{9}$	≤ 84	84
179	$6^{108}3^{27}$	≤ 87	81
180	$6^{81}3^{81}$	≤ 84	$78 \leq w \leq 84$
181	$6^{120}3^{3}$	≤ 81	81
182	$2^{360}1^{9}$	≤ 100	$^*w \leq 81$
183	$2^{324}1^{81}$	≤ 94	$^*66 \leq w \leq 87$
184	$2^{364}1^{1}$	≤ 93	
185	$2^{243}1^{243}$	≤ 93	
186	$2^{363}1^{3}$	≤ 105	$^*w \leq 85$
187	$2^{351}1^{27}$	≤ 95	
188	$4^{180}1^{9}$	≤ 87	$78 \leq w \leq 83$
189	$4^{180}2^{4}1^{1}$	≤ 94	$74 \leq w \leq 81$
190	$4^{162}1^{81}$	≤ 88	$80 \leq w \leq 81$
191	$4^{162}2^{40}1^{1}$	≤ 99	$73 \leq w \leq 81$
192	$4^{162}2^{36}1^{9}$	≤ 84	$79 \leq w \leq 81$
193	$4^{162}2^{27}1^{27}$	≤ 84	$78 \leq w \leq 81$
194	$4^{162}2^{39}1^{3}$	≤ 96	$73 \leq w \leq 84$
195	$4^{180}2^{3}1^{3}$	≤ 91	$76 \leq w \leq 83$
196	$4^{182}1^{1}$	≤ 81	$73 \leq w \leq 77$
200	$8^{90}4^{2}1^{1}$	≤ 81	81
201	$4^{162}2^{39}1^{3}$	≤ 89	$73 \leq w \leq 84$
202	$4^{162}2^{27}1^{27}$	≤ 84	$75 \leq w \leq 82$
203	$4^{162}2^{36}1^{9}$	≤ 104	$74 \leq w \leq 84$
204	$4^{162}2^{36}1^{9}$	≤ 110	$73 \leq w \leq 83$
205	$8^{90}2^{3}1^{3}$	≤ 81	81
206	$4^{180}2^{3}1^{3}$	≤ 86	$74 \leq w \leq 83$
207	$2^{243}1^{243}$	≤ 105	$w \leq 86$
208	$2^{363}1^{3}$	≤ 101	$^*w \leq 88$
209	$2^{324}1^{81}$	≤ 96	$^*w \leq 84$
210	$4^{162}2^{39}1^{3}$	≤ 90	$72 \leq w \leq 80$
211	$4^{162}2^{27}1^{27}$	≤ 82	$74 \leq w \leq 78$
212	$4^{162}2^{36}1^{9}$	≤ 73	73
213	$4^{164}2^{36}1^{9}$	≤ 85	$72 \leq w \leq 80$

graph	*weight distribution*	*weighted dom. set size w*	
		[**13**]	*w/ symm. augm.*
214	$2^{360}1^{9}$	≤ 109	$^{\bullet}w \leq 89$
215	$4^{180}2^{3}1^{3}$	≤ 94	$75 \leq w \leq 77$
216	$2^{351}1^{27}$	≤ 102	$^{\bullet}67 \leq w \leq 86$
217	$2^{324}1^{81}$	≤ 101	$^{\bullet}w \leq 89$
218	$2^{351}1^{27}$	≤ 138	$^{\bullet}w \leq 79$
219	$2^{360}1^{9}$	≤ 89	$^{\bullet}w \leq 88$
220	$2^{351}1^{27}$	≤ 94	$^{\bullet}67 \leq w \leq 88$

The entries marked with $^{\bullet}$ were computed using the optimization software OSL.

Bibliography

[1] Gregory Butler, *Fundamental Algorithms for Permutation Groups*, Springer Verlag, Berlin, 1991

[2] G. Butler, C. W. H. Lam, *A General Backtrack Algorithm for the Isomophism Problem of Combinatorial Objects*, Journal of Symbolic Computation, 1, pp. 363-381, 1985

[3] John D. Dixon, Brian Mortimer, *Permutation Groups*, Graduate Texts in Mathematics, No. 163, Springer Verlag, New York, 1996.

[4] L. R. Foulds, *Combinatorial optimization for undergraduates*, Undergraduate Texts in Mathematics, Springer Verlag, New York, 1984

[5] Laurent Habsieger, *A new lower bound for the football pool problem for 7 matches*, Journal de Théorie des Nombres de Bordeaux **8**, pp. 481-484, 1996

[6] Teresa W. Haynes, Stephen T. Hedetniemi, Peter J. Slater, *Fundamentals of Domination in Graphs*, Pure and Applied Mathematics, No. 208, Marcel Dekker, Inc., New York, 1998

[7] Thomas W. Hungerford, *Algebra*, Graduate Texts in Mathematics, No. 73, Springer Verlag, New York, 1974

[8] ILOG CPLEX 7.1 *User's Manual*, March 2001

[9] H. J. L. Kamps, J. H. van Lint, *The Football Pool Problem for 5 Matches*, Journal of Combinatorial Theory 3, pp. 315-325, 1967

[10] Donald L. Kreher, Douglas R. Stinson, *Combinatorial algorithms: generation, enumeration, and search*, CRC Press Series on Discrete Mathematics and its Applications, CRC Press, Boca Raton, 1999

[11] P. J. M. van Laarhoven, E. H. L. Aarts, J. H. van Lint, L. T. Wille, *New Upper Bounds for the Football Pool Problem for 6, 7, and 8 Matches*, Journal of Combinatorial Theory, Series A, **52**, pp. 304-312, 1989

[12] Clement W. H. Lam, *Computational Combinatorics - A Maturing Experimental Approach*, Concordia University, Montreal, 2000

[13] Ka Leung Ma, *Solving the Dominating Set Problem: A Group Theory Approach*, Ph.D. Thesis, Department of Computer Science, Concordia University, Montreal, 1998

[14] J. G. Mauldon, *Covering Theorems for Groups*, Quart. J. Math. Oxford Ser. 2, 1, pp. 284-287, 1950

[15] Brendan D. McKay, *nauty User's Guide (Bersion 1.5)*, Australian National University, Technical Report

[16] George L. Nemhauser, Laurence A. Wolsey, *Integer and Combinatorial Optimization*, Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley & Sons, New York, 1988

[17] Patric R. J. Östergård, Alfred Wassermann, *A New Lower Bound for the Football Pool Problem for 6 Matches*, Preprint

[18] Patric R. J. Östergård, *New Upper Bounds for the Football Pool Problem for 11 and 12 Matches*, Journal of Combinatorial Theory, Series A, **67**, pp. 161-168, 1994

[19] Laurence A. Wolsey, *Integer Programming*, Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley & Sons, New York, 1998

[20] S. K. Zaremba, *Covering Problems concerning Abelian Groups*, J. London Math. Soc. 27, pp. 242-246, 1952

Appendix

In this appendix, the source code of the symmetric augmentation algorithm is given in form of the C program "tsysaug.c".

```
/***/
/* Program to augment the coefficient matrix of an IP using symmetries.
/* Input required:
/*   line 1: num_rows (number of rows of the matrix),
/*           num_cols (number of columns),
/*           maxval   (maximum abs value of matrix elements).
/*   line 2: num_rpart (number of row partitions).
/*   line 3 to (2+num_rpart): list of row numbers in
/*           each partition, zero terminated.
/*   line (3+num_rpart): num_cpart (number of column partitions).
/*   line (4+num_rpart) to (4+num_rpart+num_cpart): list of column numbers in
/*           each partition, zero terminated.
/*   followed by the integer matrix
/*
/* Obligatory first argument: input file name;
/* Second argument (optional): CPLEX output file name (suffix ".mps" will
/*           be added; stdout -> monitor).
/***/

/* #define DEBUG_PART */        /* for debugging */
/* #define DEBUG */
#include <math.h>
#include <sys/times.h>

#include <groupdcl.h>     /* ISOM */
#include <matrixd.h>
#include <nvariant.h>
#include <valdesc.h>
#include <nvilib.h>
#include <blockdcl.h>
#include "tst_mat.h"

typedef struct{
  ptr_to_permvect reps; /* orbit representative for each elem */
  ptr_to_permvect orbs; /* orbits */
  ptr_to_permvect replist; /* list of all diff. representatives in reps, replist
[0]=# of different reps */
}parttype;

typedef struct tms *ptr_to_tms;

/* declarations to link with nvi_lib */
tst_mat_master *tst_mat_context[maxsize];

FILE *in, *outmps;
int num_rows, num_cols, maxVal;
matrix A;
ptr_to_perm_gp autogroup;
ptr_to_permvect rc_orbit;

long start_time, finish_time;
ptr_to_tms sbuffer, fbuffer, ebuffer;

#define HERTZ     60.
#define F_NAME_LENGTH 10


void my_read_matrix(){
/* reads in the matrix A */
  int i,j;

  for (i=1; i<=num_rows; i++)
    for (j=1; j<=num_cols; j++)
      fscanf(in,"%d",&A[i][j]);
}/*my_read_matix*/


void print_A(int start_row){
  /* prints rows start_row to num_rows of matrix A */
```

```
  int i,j;

  for (i=start_row; i<=num_rows; i++){
    printf("%2d:\n", i);
    for (j=1; j<=num_cols; j++){
      printf("%3d",A[i][j]);
      if (j%20==0) printf("\n");
    }
    if (j%20!=0) printf("\n");
  }
}/* print_A */


ptr_to_permvect read_and_create_partn(){
  /* reads in the row and col partitions
  /* and returns them in orbit, where the rows and columns
  /*   are linked in cycle form.
  /* orbit[0]=length of orbit
  */

  int i,start,next,prev,num_part;
  ptr_to_permvect orbit,temp_pv;      /* permutation vectors */

  orbit=Allocatepv(num_rows+num_cols);
  temp_pv=Allocatepv(num_rows+num_cols);    /* Allocatepv(k) allocates
                                               permutation vector of length k */
  for (i=1; i<=num_cols+num_rows; i++)
    temp_pv[i]=1;

  fscanf(in,"%d",&num_part);
  for (i=1; i <= num_part; i++){
    fscanf (in,"%d", &start);
    start+=num_cols;
    if (temp_pv[start]==0) myabort("illegal row partition");
    temp_pv[start]=0;
    prev=start;
    fscanf (in,"%d", &next);
    while (next!=0){
      next+=num_cols;
      if (temp_pv[next]==0) myabort("illegal row partition");
      temp_pv[next]=0;
      orbit[prev]=next;
      prev=next;
      fscanf(in,"%d", &next);
    }
    orbit[prev]=start;
  }

  fscanf(in,"%d",&num_part);
  for (i=1; i <= num_part; i++){
    fscanf(in,"%d", &start);
    if (temp_pv[start]==0) myabort("illegal col partition");
    temp_pv[start]=0;
    prev=start;
    fscanf (in,"%d", &next);
    while (next!=0){
      if (temp_pv[next]==0) myabort("illegal col partition");
      temp_pv[next]=0;
      orbit[prev]=next;
      prev=next;
      fscanf (in,"%d", &next);
    }
    orbit[prev]=start;
  }
  orbit[0]=num_cols+num_rows;
  disposepv(temp_pv);
  return(orbit);
} /* read_and_create_partn*/
```

```c
void initialize(){  /* reads input file, creates start partition in rc_orbit */
  fscanf(in, "%d %d %d", &num_rows, &num_cols, &maxVal);
  if(maxsize<num_rows){
    myabort("Row number bigger than maxsize!");
  }
  if(maxsize<num_cols){
    myabort("Column number bigger than maxsize!");
  }
  rc_orbit=read_and_create_partn();
  my_read_matrix();
  print_A(1);
}


ptr_to_perm_gp make_autogroup(int first_row, int last_row){
/* performs isomorphism testing of the matrix from row first_row+1
/*  to row last_row using the previous automorphism group in
/*  tst_mat_context[count-1] and returns the new automorphism group */

  static ptr_to_perm_gp autogp;
  static ptr_to_permvect temp_pv=NULL;
  ptr_to_permvect tpvect;
  int i;
  static int count=-1;  /* counts number of augmentations */

  if (temp_pv==NULL)
    temp_pv=Allocatepv(num_cols);

  for (i=1; i<=num_cols-1; i++)
    temp_pv[i]=i;
  temp_pv[num_cols]=0;

  if (last_row >=maxsize)
    myabort("too many rows");
  count++;
  if (count == 0){   /* initial loop */
     autogp=NULL;
     tpvect=rc_orbit;
  }
  else {
    autogp = tst_mat_context[count-1]->nvi_context->isom_context->auto_gp;
    changebase(autogp, id_permvect, autogp->permsize);
    tpvect=NULL;
  };

  if (count >= maxsize) myabort ("too many change of context");

  tst_mat_context[count]=init_for_isom_A(&A[first_row], last_row-first_row,
                         num_cols,  -maxVal, maxVal, tpvect, autogp, NULL);
  autogp = copy_gp( find_autogp(&A[first_row], TRUE, FALSE,
           tst_mat_context[count]), NULL);
  /* find_autogp return automorphism group of A */
  truncate_gp(autogp, temp_pv, FALSE);    /* truncates autogp to columns */
  return(autogp);
}/* make_autogp */


void get_replist(parttype p){
  /* updates p.replist according to p.reps, assumes p.replist allocated */
  int i,j;
  boolean cont;

  p.replist[0]=0;
  i=1;
  while(i<=autogroup->permsize){
    cont=FALSE;
    for (j=1;j<=p.replist[0];j++){
       if (p.reps[i]==p.replist[j]) cont=TRUE;
    }
```

```c
    if (!cont) {
      p.replist[0]++;
      p.replist[p.replist[0]]=p.reps[i];
    }
    i++;
  }
}


void dump_part(parttype p){
  /* prints out partition p */
  int i;

  for (i=0; i<=autogroup->permsize;i++){
    printf("%3d",p.reps[i]);
  }
  printf("\n");
  for (i=0; i<=autogroup->permsize;i++)
    printf("%3d",p.orbs[i]);
  printf("\n");
  for (i=0; i<=p.replist[0];i++)
    printf("%3d",p.replist[i]);
  printf("\n");
}


/*****************************************************************************/
/* time handling routines                                                    */

void Init_time_val ()
{ /* initiates timing variables */
  sbuffer = (ptr_to_tms) malloc (sizeof (struct tms));
  fbuffer = (ptr_to_tms) malloc (sizeof (struct tms));
  ebuffer = (ptr_to_tms) malloc (sizeof (struct tms));

  start_time = times(sbuffer);
} /* Init_time_val */


void Calculate_and_print_time_used ()
{ /* calculates and prints the time used */
  finish_time = times(fbuffer);

  /* user time */
  ebuffer->tms_utime = fbuffer->tms_utime - sbuffer->tms_utime;
  /* system time */
  ebuffer->tms_stime = fbuffer->tms_stime - sbuffer->tms_stime;

  printf ("CPU time used ");
  printf ("%f seconds\n",
          (ebuffer->tms_utime + ebuffer->tms_stime)/HERTZ);

} /* Calculate_and_print_time_used */

/*****************************************************************************/


void Print_CPLEX_output (char *arg)
{/* Print out the problem as an mixed integer problem in MPS format
    for CPLEX */
   int i, j; /* use in loop */
   char outmpsname[F_NAME_LENGTH];
   /* char logfilename[F_NAME_LENGTH];*/


   if (!arg) return;
   else if (strcmp(arg, "stdout") == 0) outmps = stdout;
   else{
     sprintf (outmpsname, "%s mps", arg);
```

```
      outmps = fopen(outmpsname, "w");
      printf("Writing %s... ",outmpsname);
    }

    fprintf(outmps, "NAME\n");     /* process name, none is assigned */

    fprintf(outmps, "ROWS\n");     /* the row parameters */

    fprintf(outmps, " N obj\n");  /* objective function */

    /* print all the constraints, the last one is for the weight set,
       if no weight assigned, that means each vertex has a weight of 1 */

    /* G means >=; E means =; L means <= */

    for (i = 1; i < num_rows; i++)
      fprintf(outmps, " G c%d\n", i);

    /* each column */
    fprintf(outmps, "COLUMNS\n");

    for (i = 1; i < num_cols; i++) {
      if(A[1][i])
        fprintf(outmps, "%5c%-9d%-10s%d\n",'x',i,"obj\0",A[1][i]);
      for (j = 2; j <= num_rows; j++) {
        if (A[j][i])
          fprintf(outmps, "%5c%-9d%c%-9d%d\n",'x',i,'c',(j-1),A[j][i]);
      }
      /*     fprintf(outmps, "%5c%-9d%c%-9d%d\n",'x',i,'c',max_level,cell_weight[
i]); */
    }

    /* RHS */
    fprintf(outmps, "RHS\n");
    for (j = 2; j <= num_rows; j++) {
      fprintf(outmps, "%7s%8c%-9d%d\n","rhs\0",'c',(j-1),A[j][num_cols]);
    }

    /*   fprintf(outmps, "%7s%8c%-9d%d\n","rhs\0",'c',max_level,total_content);*/

    /* BOUNDS */
    fprintf(outmps, "BOUNDS\n");
    for (i = 1; i < num_cols; i++) {
      fprintf(outmps, " BV BOUND  x%d\n", i);
    }

    fprintf(outmps, "ENDATA\n");
    if (outmps!=stdout)
      fclose (outmps);
    printf("DONE.\n");

    /*   sprintf (logfilename, "z%dc%dg%d.log", MATCHES, CONTENT, input_graph_num
);

    fprintf (outcplex, "set LOGFILE %s\n", logfilename);
    fprintf (outcplex, "read %s\n", outmpsname);
    fprintf (outcplex, "optimize\n");
    fprintf (outcplex, "display solution - \n");
    fprintf (outcplex, "display slack %d\n", max_level);
    fprintf (outcplex, "\n");*/

} /* Print_CPLEX_output */

/************************************************************************/



int main(int argc, char *argv[]){
```

```
  int size; /* stores #rows of A*/
  int i, j, tmp, temp_int, oldrows=0;
  parttype part;
  ptr_to_permvect orbit, map;
  boolean firstRound;
  double realorder;
  long intorder;
  int onecont,monecont,zerocont;

  /* handling of command line: first argument input file, second (optional)
     argument output file */
  if (argc < 2)
    {
      printf ("\nTo run this program you have to specify at least an input file name.\n");
      exit(0);
    }
  else
    {
      if (argv[1] && (strcmp(argv[1], "stdin") == 0)) in = stdin;
      else
        {
          if ((in = fopen(argv[1], "r")) == NULL){
            printf ("\nI could not open the file \"%s\".\n", argv[1]);
            /* Calculate_and_print_time_used (); */
            exit(0);
          }
        }
    }

  Init_time_val();    /* initialize time */
  initialize();       /* read input */
  firstRound=TRUE;

  oldrows=0;
  while(1){ /* infinite loop */
    autogroup= make_autogroup(oldrows, num_rows);
#ifdef DEBUG
    dump_gp(stdout, autogroup, FALSE);
#endif

  /* & makes pointer to argument; order of autogroup returned in intorder */
    gporder(autogroup,&realorder,&intorder);
    printf("group order = ");
    printf("%3d\n\n",intorder);
    if (intorder==1)  break; /* exit if trivial automorphism group */

    orbit=Allocatepv(num_cols-1);
    map=Allocatepv(autogroup->permsize);
    for (i=1;i<num_cols;i++){
      /* find nontrivial orbit to get transitive subset for partitioning */
      for (j=0;j<num_cols;j++) orbit[j]=0;
      find_orbit(autogroup,i,0,orbit,NULL);
      tmp=0; /* counts orbit length */
      for (j=0;j<num_cols;j++)
        if (orbit[j]!=0){
          tmp++;
          map[tmp]=orbit[j]; /* map stores orbit elems for truncation */
        }
      if (tmp==0) myabort("find_orbit fails!");
      if (tmp>1) break;
    }

    printf("refined orbit: ");
    printf("i="); printf("%3d\n",i);
    printf("orbit:");
    temp_int=0;
    for (j=0;j<num_cols;j++){
      if(orbit[j]!=0){
        printf("%d ",orbit[j]);
      temp_int++;
      if ((temp_int%20) == 0) printf("\n");
      else if ((temp_int%10)==0) printf("  ");
    }
  }
  if ((temp_int%20)!=20) printf("\n");

  if (tmp==autogroup->permsize){   /* autogroup transitive */
    printf("autogroup transitive\n");
    part.reps=Allocatepv(autogroup->permsize);
    part.orbs=Allocatepv(autogroup->permsize);
    part.replist=Allocatepv(autogroup->permsize);
    GreedyMaxPart(autogroup,part.reps,part.orbs);
    get_replist(part);
#ifdef DEBUG_PART
    printf("chosen partition: "); printf("\n");
    dump_part(part);
#endif

    /* check if num_rows of the augmented A is < maxsize */
    if (maxsize <= num_rows+part.replist[0]-1){
      myabort("Row number bigger than maxsize after next augmentation!");
    }
    /* augment A with new inequalities from part */
    for (i=num_rows+1; i<num_rows+part.replist[0]; i++)
      for (j=1; j<=num_cols; j++)
        A[i][j]=0;
    for (i=2;i<=part.replist[0];i++)
      for (j=1;j<=autogroup->permsize;j++){
        if (part.reps[j]==part.replist[1])
          A[num_rows+i-1][j]=1;
        if (part.reps[j]==part.replist[i])
          A[num_rows+i-1][j]=-1;
      }
    oldrows=num_rows; /*because we need it so often to update rc_orbit*/
    num_rows+=part.replist[0]-1;
  }

  else{    /* autogroup intransitive -> truncate to orbit in map */
    printf("autogroup intransitive\n");
    map[tmp+1]=0;
    truncate_gp(autogroup,map,FALSE);
    part.reps=Allocatepv(autogroup->permsize);
    part.orbs=Allocatepv(autogroup->permsize);
    part.replist=Allocatepv(autogroup->permsize);
    GreedyMaxPart(autogroup,part.reps,part.orbs);
    get_replist(part);

#ifdef DEBUG_PART
    printf("chosen partition: "); printf("\n");
    dump_part(part);
#endif

    /* check if num_rows of the augmented A is < maxsize */
    if (maxsize <= num_rows+part.replist[0]-1){
      myabort("Row number bigger than maxsize after next augmentation!");
    }
    /* augment A with new inequalities from part */
    for (i=num_rows+1; i<num_rows+part.replist[0]; i++)
      for (j=1; j<=num_cols; j++)
        A[i][j]=0;
    for (i=2;i<=part.replist[0];i++)
      for (j=1;j<=autogroup->permsize;j++){
        if (part.reps[j]==part.replist[1])
          A[num_rows+i-1][map[j]]=1;
        if (part.reps[j]==part.replist[i])
          A[num_rows+i-1][map[j]]=-1;
      }
    oldrows=num_rows;
```

```
      num_rows+=part.replist[0]-1;
    }

    printf("matrix A augmented by: \n");
    print_A(oldrows+1);
    printf("#lines of the coefficient matrix: %d \n", num_rows);

    if (firstRound){ /*update maxVal*/
      onecont=1;
      monecont=1;
      zerocont=1;
      for (i=1;i<=oldrows;i++)
        for (j=1;j<=num_cols;j++){
          if (A[i][j]==1) onecont=0;
          if (A[i][j]==-1) monecont=0;
          if (A[i][j]==0) zerocont=0;
        }
      maxVal+=onecont+monecont+zerocont;
      printf("maxVal="); printf("%3d",maxVal); printf("\n");
      firstRound=FALSE;
    }

    /* clean up */
    disposepv(orbit); disposepv(map);
    disposepv(part.reps); disposepv(part.orbs); disposepv(part.replist);
    disposepg(autogroup);
  }

  Print_CPLEX_output(argv[2]);
  Calculate_and_print_time_used();
}
```