Linear Optimization over Permutation Groups [⋆]

Christoph Buchheim

Istituto di Analisi dei Sistemi ed Informatica "Antonio Ruberti", Viale Manzoni, 30, 00185 Roma, Italy

Michael Jünger

Universität zu Köln, Institut für Informatik, Pohligstr. 1, 50969 Köln, Germany

Abstract

For a permutation group given by a set of generators, the problem of finding "special" group members is NP-hard in many cases. E.g., this is true for the problem of finding a permutation with a minimum number of fixed points or a permutation with a minimal Hamming distance from a given permutation. Many of these problems can be modeled as linear optimization problems over permutation groups. We develop a polyhedral approach to this general problem and derive an exact and practically fast algorithm based on the branch & cut-technique.

Key words: permutation groups, fixed point minimization, Hamming distance
1991 MSC: 20B40, 90C10

1 Introduction

Permutation groups can have exponential size in the number of elements of the domain. For computational matters, they are usually specified by a small set of generators. In fact, for every permutation group on $n > 3$ elements, there exists a set of at most $\lfloor n/2 \rfloor$ generators [11]. Many natural tasks like testing

[⋆] This work was partially supported by the IST Programme under contract number IST-1999-14186 (ALCOM-FT) and by the Marie Curie RTN ADONET 504438, both funded by the EU.

Email address: `buchheim@iasi.cnr.it` (Christoph Buchheim).

membership and computing the order of the group can be performed in polynomial time for such a representation. An algorithm for solving both problems was presented by Sims using the concept of strong generating sets [13], but only much later it was shown to run in polynomial time [6]. The fastest algorithm devised so far runs in $O(n^4 \log^c n + mn^2)$ time, where m denotes the number of given generators and c is some constant [1]. Additional membership tests only take $O(n^2)$ runtime.

On the other hand, finding "special" permutations in a permutation group given by generators is often computationally hard. To give an example, it is an NP-complete problem to decide the natural question whether the group contains a fixed-point-free permutation. In the subsequent sections, we discuss the much more general problem of linear optimization over permutation groups: we desire to minimize or maximize an arbitrary linear objective function over all permutations in the given group. Here, we identify permutations with permutation matrices, so that a linear objective function is given by a cost coefficient for each matrix entry.

We present a promising approach for solving this general problem that is based on integer programming techniques. For this, we examined the polytope spanned by all permutation matrices corresponding to elements of the given permutation group. We present a complete description of this polytope by linear constraints derived from the given set of generators. Using this description, we designed a heuristic cutting plane algorithm and embedded it into a branch & cut-framework. We tested our algorithm in numerous experiments; the results were very satisfactory in general.

There are many practical applications for this algorithm. Our main motivation for studying linear optimization over permutation groups was the desire to find geometric symmetries in graphs for the purpose of automatic graph drawing [5]. In this area of research, the aim is to create nice two- or three-dimensional layouts of abstractly given graphs automatically. An important criterion for measuring the layout quality is whether possible symmetric structure of the graph is revealed. This leads to the problem of finding an automorphism of the graph with a minimum number of fixed points that satisfies certain additional conditions imposed by the restriction of the number of dimensions. Unfortunately, these conditions cannot be modeled by a linear objective function, but we could solve the problem by adding new linear constraints to the permutation polytope corresponding to the automorphism group of the graph. Generators of this group were computed by `nauty` [12].

In general, if a permutation group is given as an automorphism group of some combinatorial object, it is usually easier to compute a set of generators of this group than to find an automorphism optimizing a given linear objective function. This is even true in terms of theoretical complexity unless the graph-isomorphism problem is NP-complete. Thus it makes sense to combine our algorithm with `nauty` in general.

This paper is organized as follows. In Sect. 2, we give a precise definition of our basic problem along with some special cases and a proof of its NP-hardness. In Sect. 3, we define and investigate the permutation polytope corresponding to a permutation group. Finally, in Sect. 4, we describe a branch & cut-algorithm that is based on this investigation and report experimental results.

2 The Basic Problem

In the following, we examine the problem of linear optimization over a permutation group that is given by a set generators:

Problem 1 *Given a finite set X, permutations $\pi_1, \ldots, \pi_m$ of X, and any cost function $c\colon X^2 \to \mathbf{R}$, find a permutation $\pi \in (\pi_1, \ldots, \pi_m)$ minimizing $\sum_{i \in X} c(i, \pi(i))$.*

Here $(\pi_1, \ldots, \pi_m)$ denotes the subgroup of the symmetric group over X that is generated by $\pi_1, \ldots, \pi_m$, i.e., the group of all compositions of these permutations. Problem 1 is very general. Even if we only consider binary cost functions, i.e., if we require $c(X^2) \in \{0, 1\}$, there are still a lot of interesting special cases arising from special cost functions. Some of them are listed in the following.

Problem 2 *Given a finite set X, permutations $\pi_1, \ldots, \pi_m$ of X, and a subset R of X^2, find a permutation $\pi \in (\pi_1, \ldots, \pi_m)$ using a minimum number of pairs in R, i.e., minimizing $\#\{(i,j) \in R \mid \pi(i) = j\}$.*

Problem 3 *Given a finite set X and permutations $\pi_1, \ldots, \pi_m$ of X, find a permutation in $(\pi_1, \ldots, \pi_m)$ with a minimum number of fixed points.*

Problem 4 *Given a finite set X and permutations $\pi_1, \ldots, \pi_m$ and $\overline{\pi}$ of X, find a permutation $\pi \in (\pi_1, \ldots, \pi_m)$ agreeing with $\overline{\pi}$ on as many points as possible, i.e., maximizing $\#\{i \in X \mid \pi(i) = \overline{\pi}(i)\}$.*

Problem 5 *Given a finite set X, permutations $\pi_1, \ldots, \pi_m$ of X, and any two colorings $\varphi_1, \varphi_2\colon X \to \mathbf{N}$, find a permutation $\pi \in (\pi_1, \ldots, \pi_m)$ transferring φ_1 to φ_2 as well as possible, i.e., maximizing $\#\{i \in X \mid \varphi_1(i) = \varphi_2 \circ \pi(i)\}$.*

Observe that Problem 2 is the general problem with binary cost function, while Problems 3, 4, and 5 are special cases. Problem 4 searches for the closest permutation to $\overline{\pi}$ with respect to the Hamming distance. This is also a special case of Problem 5 by setting $\varphi_1 = \overline{\pi}$ and $\varphi_2 = \mathrm{id}_X$ and identifying the finite set X with a subset of $\mathbf{N}$.

In the remainder of this section, we show the NP-hardness of Problem 1. For this, it obviously suffices to show that the following problem is NP-complete:

Problem 6 *Given a finite set X and permutations $\pi_1, \ldots, \pi_m$ of X, decide whether $(\pi_1, \ldots, \pi_m)$ contains a fixed-point-free permutation.*

This problem is similar to the problem of deciding whether a given graph admits a fixed-point-free automorphism, which was shown to be NP-complete by Lubiw [9]. The NP-completeness of Problem 6 does not follow from this result, as computing generators of the automorphism group of a graph is an isomorphism-complete problem and hence possibly NP-complete. However, essentially the same reduction as in the proof of Lubiw can be used for our problem, except that we do not need edges and labels so that in our case the proof is technically more simple. For this reason, and for the sake of completeness, we include a full proof here.

Theorem 7 *Problem 6 is NP-complete.*

PROOF. The problem is in NP. For showing completeness, we use a reduction from 3SAT. Given an instance of 3SAT, we denote the variables by $U = \{u_1, \ldots, u_p\}$ and the clauses by $C = \{c_1, \ldots, c_q\}$. Furthermore, for $r \in \{0, \ldots, 7\}$ and $k \in \{1, 2, 3\}$, let $r^{(k)}$ denote the number r with the k-th bit in the binary representation changed. An instance of Problem 6 is constructed as follows: we define the domain X by

$$X = \bigcup_{i=1}^{p} \{u_i(0), u_i(1)\} \cup \bigcup_{j=1}^{q} \{c_j(0), \ldots, c_j(7)\} .$$

For every variable u_i, we define two permutations $\pi_i(t)$ and $\pi_i(f)$; both are involutions. The permutation $\pi_i(t)$ fixes all points except that $u_i(0) \leftrightarrow u_i(1)$ and, for every clause c_j containing the variable u_i without negation as the k-th literal,

$$c_j(r) \leftrightarrow c_j(r^{(k)}) \quad \text{for all } r = 0, \ldots, 7 .$$

The permutation $\pi_i(f)$ is defined analogously as fixing all points except that $u_i(0) \leftrightarrow u_i(1)$ and, for every clause c_j containing the negated variable $\overline{u}_i$ as the k-th literal,

$$c_j(r) \leftrightarrow c_j(r^{(k)}) \quad \text{for all } r = 0, \ldots, 7 .$$

Now consider $G = (\pi_1(t), \pi_1(f), \ldots, \pi_p(t), \pi_p(f))$. We claim that C is satisfiable if and only if G contains a fixed-point-free permutation of X.

For any satisfying truth assignment $T\colon U \to \{t, f\}$, it is readily checked that the composition of all $\pi_i(T(u_i))$ for $i = 1, \ldots, p$ is a fixed-point-free permutation: on the points $u_i(0)$ and $u_i(1)$, this is obvious, as only the permutation $\pi_i(T(u_i))$ acts on these points in a non-trivial way. As any clause c_j contains at least one satisfied literal u_i or $\overline{u}_i$, the corresponding permutation $\pi_i(T(u_i))$ induces a fixed-point-free permutation on $\{c_j(0), \dots, c_j(7)\}$. Only the permutations corresponding to the two other literals in c_j also act on $\{c_j(0), \dots, c_j(7)\}$ in a non-trivial way, but these change other bits in the binary representation and hence cannot induce a fixed point.

Now let π be any fixed-point-free permutation in G. As G is Abelian and all generators are involutions, we may assume that every generator appears at most once in a chosen composition of π. Additionally, for every $i = 1, \dots, p$, we must have either $\pi_i(t)$ or $\pi_i(f)$ in this composition, as otherwise the points $u_i(0)$ and $u_i(1)$ would be fixed by π. For the same reason, we cannot have both. Hence a well-defined truth assignment $T \colon U \to \{t, f\}$ is given by

$$T(u_i) = \begin{cases} t & \text{if } \pi_i(t) \text{ appears in the chosen composition of } \pi \text{, and} \\ f & \text{if } \pi_i(f) \text{ appears in the chosen composition of } \pi . \end{cases}$$

We claim that T satisfies all clauses in C. Indeed, if any clause c_j would not be satisfied by T, i.e., all literals of c_j would be false, then by construction every point in $\{c_j(0), \dots, c_j(7)\}$ was fixed by π, contradicting our assumption. □

As shown by this proof, Problem 6 remains NP-complete even if restricted to permutation groups of exponent two, i.e., permutation groups containing only involutions. Problems 1 to 3 are straightforward generalizations of Problem 6, hence they are all NP-hard already for this special class of permutation groups. The same can be shown for Problems 4 and 5 by similar proofs, where for Problem 5 we may even require $\varphi_1(X) = \varphi_2(X) = \{0, 1\}$ without loosing NP-hardness.

3 The Permutation Polytope

Let $G = (\pi_1, \dots, \pi_m)$ be a permutation group over a finite set X. Let n denote the number of elements of X. As a first step towards a polyhedral approach to Problem 1, we define and describe a polytope modeling G in the following. Consider

$$M \colon G \to \mathbf{R}^{X^2}, \quad M(\pi)_{ij} = \begin{cases} 1 & \text{if } \pi(i) = j \\ 0 & \text{otherwise .} \end{cases}$$

This is the usual way of representing permutations of X by $n \times n$ permutation matrices. Notice that M yields a group monomorphism of G into the general linear group $\mathrm{GL}_n\mathbf{R}$. The *permutation polytope* P_G corresponding to the group G is defined as the convex hull of $M(G)$ in $\mathbf{R}^{X^2}$. A similar but much more general class of polytopes related to real representations of finite groups was examined by Barvinok [3].

Theorem 8 *The automorphism group of P_G contains G as a subgroup. It acts transitively on the vertices of P_G.*

PROOF. For $\pi \in G$, define an automorphism $\varphi'(\pi)\colon \mathbf{R}^{X^2} \to \mathbf{R}^{X^2}$ by $e_{ij} \mapsto e_{i\pi(j)}$ for all $i, j \in X$, where e_{ij} is the unit vector corresponding to the dimension (i, j). The map $\varphi'(\pi)$ induces a permutation of the vertices of P_G, as for $\psi \in G$ we have $\varphi'(\pi)(M(\psi)) = M(\pi \circ \psi)$. Thus $\varphi(\pi) = \varphi'(\pi)|_{P_G} \in \text{Aut } P_G$. This defines a group monomorphism $\varphi\colon G \to \text{Aut } P_G$. For the second statement, let $\psi_1, \psi_2 \in G$. Then $\varphi(\psi_2 \circ \psi_1^{-1})$ maps $M(\psi_1)$ to $M(\psi_2)$. $\square$

Corollary 9 *All cones of vertices of P_G are isomorphic.*

Observe that the automorphism group of P_G can be much larger than G. For an example, consider the cyclic group generated by $\pi = (1\ 2\ \ldots\ n)$. In this case, the polytope P_G is spanned by the vectors $M(\pi^i)$, $i = 1, \ldots, n$, which all use different dimensions. Thus Aut P_G is the symmetric group over G containing $n!$ elements, while G only contains n elements.

Our next task is to give a complete description of P_G in terms of linear constraints derived from the set of generators $\{\pi_1, \ldots, \pi_m\}$. This generalizes the corresponding results for the automorphism polytope [5]. We use x_{ij} to denote the variable for the dimension (i, j), i.e., we have $x_{ij} = M(\pi)_{ij}$ if (x_{ij}) models $\pi \in G$.

First consider the special case that G is the complete symmetric group over X. In this case, the permutation polytope P_G is the well-known assignment polytope for X. The latter is fully described by the constraints

$$\sum_{j \in X} x_{ij} = 1 \quad \text{for all } i \in X \tag{1}$$

$$\sum_{i \in X} x_{ij} = 1 \quad \text{for all } j \in X \tag{2}$$

and by non-negativity of all variables [4]. Its dimension is $(n-1)^2$ and the number of its facets is n^2 [2].

In the general case, the polytope P_G is obviously a subpolytope of the assignment polytope. In particular, the constraints (1) and (2) are still valid for P_G. However, this is not a complete description any more, hence we have to find further valid constraints. As our group G is specified by a set of generators, we have to derive the new constraints from these generators. For this, fix any positive integer t and define a relation on X^t by

$$(i_1, \ldots, i_t) \sim (j_1, \ldots, j_t) \quad \Longleftrightarrow \quad \exists\, \pi \in G\colon \forall s \in \{1, \ldots, t\}\colon \pi(i_s) = j_s\ .$$

Since G is a group, this defines an equivalence relation on X^t and hence a partitioning of X^t. We call this partitioning the *t-partitioning* of X with respect to G.

The t-partitioning can easily be computed from a set of generators using disjoint dynamic sets: for this, start with sets containing single elements. For all $(i_1, \ldots, i_t) \in X^t$ and all given generators π, merge the sets containing $(i_1, \ldots, i_t)$ and $(\pi(i_1), \ldots, \pi(i_t))$, respectively. Clearly, the resulting partitioning is the t-partitioning with respect to G.

Now let I be a multiset of elements of X^2. Suppose that

$$\{(i_1, j_1), \ldots, (i_t, j_t)\} \subseteq I \quad \Longrightarrow \quad (i_1, \ldots, i_t) \not\sim (j_1, \ldots, j_t)\ . \tag{3}$$

By definition, the *homomorphism constraint*

$$H_{I,t}\colon \sum_{(i,j)\in I} x_{ij} \le t - 1 \tag{4}$$

is a valid inequality for the polytope P_G. The following statement is a straightforward generalization of Theorem 1 in [5].

Theorem 10 *In the affine subspace given by the equations (1), each rational inequality valid for P_G is induced by a homomorphism constraint.*

PROOF. Consider an arbitrary rational inequality

$$H\colon \sum_{(i,j)\in X^2} a_{ij} x_{ij} \le t - 1$$

and assume that it is valid for P_G. We may assume $a_{ij} \in \mathbf{Z}$ for all $i, j \in X$ and $t \in \mathbf{Z}$. For every $(i, j) \in X^2$ with $a_{ij} < 0$, we can replace x_{ij} by

$$1 - \sum_{j' \in X\setminus\{j\}} x_{ij'}$$

using (1), thereby increasing the coefficient of each $x_{ij'}$ by $-a_{ij} > 0$. After these replacements, all coefficients on the left hand side are non-negative, hence we may assume $a_{ij} \ge 0$ for all $(i, j) \in X^2$. Since $M(\mathrm{id}_X) \in P_G$, we derive $\sum_{i\in X} a_{ii} \le t - 1$ and hence $t \ge 1$. Let I be the multiset of elements of X^2 containing the pair (i,j) exactly a_{ij} times, for all $(i,j) \in X^2$. As H is valid for P_G, condition (3) holds for I and t. Obviously, we have $H = H_{I,t}$. □

Corollary 11 *The permutation polytope P_G is completely described by the linear constraints (1) and (4).*

Corollary 11 provides an interesting and useful link between the group structure of G and the linear structure of P_G. This link is crucial for developing an algorithm for Problem 1 using polyhedral methods, as explained in the following section.

4 The Branch & Cut-Algorithm

In the following, we give a more detailed overview of our integer programming approach for solving Problem 1. The proposed branch & cut-algorithm mainly relies on Corollary 11 of the preceding section. Thereafter, we present results of an experimental evaluation. For our implementation, we used ABACUS 2.4 [8] in combination with CPLEX 7.1 [14].

Our starting point is the linear program modeling the assignment problem, i.e., the linear program describing all permutations of X, equipped with the objective function of Problem 1:

$$\begin{array}{lrll} \min & \multicolumn{3}{l}{\sum_{(i,j)\in X^2} c(i,j)x_{ij}} \\[1ex] \text{s.t.} & \sum_{j\in X} x_{ij} = 1 & \text{for all } i \in X & \\ & \sum_{i\in X} x_{ij} = 1 & \text{for all } j \in X & \qquad (5) \\ & x_{ij} \geq 0 & \text{for all } (i,j) \in X^2\ . & \end{array}$$

The number of variables in this linear program is n^2, but often many can be left out; see Sect. 4.1. Having computed an optimal solution $\overline{x}_{ij}$ of (5), we check whether this solution is feasible, i.e., whether the corresponding permutation is a member of G; see Sect. 4.2. If it is, we have an optimal solution of Problem 1.

Otherwise, we try to separate $\overline{x}_{ij}$ from the permutation polytope, i.e., we try to find linear constraints that are valid for the polytope P_G but violated by the given solution $\overline{x}_{ij}$. By Corollary 11, we only have to separate homomorphism constraints. In fact, we only consider homomorphism constraints for $t = 2$ in our implementation; see Sect. 4.1 again.

If we find such violated constraints, we add them to (5) and reoptimize. Again, if the optimal solution is integer, we test feasibility and proceed as above. However, we may also get a fractional solution now. In this case, we first try to find feasible but not necessarily optimal solutions of Problem 1 by applying a primal heuristic guided by the fractional values of the variables; see Sect. 4.3. Afterwards, we separate and proceed as above.

If we do not find any violated constraint in some separation phase, we have to branch, i.e., we have to split up the problem into two subproblems by choosing a variable and setting this variable to zero in one subproblem and to one in the other subproblem. In Sect. 4.4, we explain how to select the branching variable and the next open subproblem to be processed.

This algorithm yields an optimal solution of Problem 1 after finite time. In general, the runtime is exponential in n and m, however, most practical instances can be solved quickly—see Sect. 4.5 for the results of an experimental evaluation.

4.1 Separation

The core of any branch & cut-approach is the algorithm used for separation. In our case, we have to solve the following

Problem 12 *Given a permutation group $G = (\pi_1, \ldots, \pi_m)$ over a finite set X and a vector $\overline{x} \in \mathbf{R}^{X^2}$, decide whether $\overline{x} \in P_G$. If the answer is negative, find a cutting plane for $\overline{x}$, i.e., a linear inequality valid for P_G but violated by $\overline{x}$.*

By a general result [7], this separation problem is polynomial time equivalent to the corresponding optimization problem. As the latter is Problem 1, we derive from Theorem 7 that Problem 12 is NP-hard. By Corollary 11, this is already true for separating homomorphism constraints. In fact, as long as t is not bounded by a very small constant, we cannot deal with homomorphism constraints anyway, as the t-partitioning, which has to be computed before, is defined on a set of cardinality n^t. In order to implement a practically fast branch & cut-algorithm, Problem 12 thus has to be approached heuristically.

In our experiments, we observed that the best runtimes were achieved by using only homomorphism constraints with $t = 1$ or $t = 2$. When we also tried to separate homomorphism constraints with $t = 3$ or even higher values of t, the number of LPs and subproblems we had to solve on average only decreased slightly, while the time spent for separation grew significantly. Only for very few instances the runtime could be improved by considering homomorphism constraints for $t > 2$.

Hence we only consider $t = 1$ and $t = 2$ in our implementation. In both cases, we may assume that the index set I defining the constraint contains each element of X^2 at most once. The case $t = 1$ is trivial: a subset I of X^2 defines a valid homomorphism constraint with right hand side zero if and only if for all $(i, j) \in I$ there is no $\pi \in G$ mapping i to j. This can be checked easily using the 1-partitioning. In the affirmative case, the corresponding variable x_{ij} can be omitted from the beginning. Observe that by this the number of variables in our LPs can often be decreased significantly.

For $t = 2$, the situation is much more complicated: a subset I of X^2 defines a valid homomorphism constraint with right hand side one if and only if it defines an independent set in the conflict graph $H = (X^2, E)$, where $((i, j), (i', j')) \in E$ if and only if $(i, i') \sim (j, j')$ with respect to the 2-partitioning. We do not know whether separating homomorphism constraints for $t = 2$ can be done in polynomial time, but we conjecture that this problem is NP-hard. In our implementation, we separate using a fast greedy independent set heuristic in H.

4.2 Feasibility

The problem of deciding feasibility for an integer solution of the current LP-relaxation is equivalent to the membership test for the group G: as we use the constraints (1) and (2) in all our LP-relaxations, any integer solution $\overline{x}_{ij}$ gives rise to a permutation π of X by setting $\pi(i) = j$ if and only if $\overline{x}_{ij} = 1$.

Thus, for a given permutation π of X, we must decide whether it is contained in G, i.e., generated by the permutations $\pi_1, \ldots, \pi_m$. It has long been an open question whether this problem can be solved in polynomial time. In 1971, an algorithm for testing membership has been devised by Sims [13]; only in 1980, Furst *et al.* [6] could show that a version of this algorithm runs in $O(n^6 + mn^2)$ time. For many years, the best-known algorithm needed $O(n^5 + mn^2)$ time. The currently fastest algorithm was presented by Babai *et al.* [1] in 1997, reducing runtime to $O(n^4 \log^c n + mn^2)$, where c is some constant.

This algorithm and its predecessors share the useful property that subsequent membership tests can be carried out much more efficiently: most of the time is needed for constructing a strong generating set with respect to some subgroup chain of G. Once having computed this, every membership test can be performed in $O(n^2)$ time. In our context, this fact is very important, as we have to check many permutations in general, not only those corresponding to integer solutions of the LP-relaxation but also those arising from primal heuristics; see Sect. 4.3. As in general even the number of variables in our LP is quadratic in n, a quadratic runtime for each membership test is definitely acceptable.

Whenever a permutation π corresponding to an integer LP-solution turns out not to be a member of G, we add the constraint $\sum_{i \in X} x_{i\pi(i)} \leq |X| - 2$ to our LP. This constraint is valid for the polytope P_G as any two different permutations of X must differ on at least two points of the domain. Hence no permutation of X can violate this constraint except for π.

4.3 Primal Heuristics

In our implementation, we use the following simple but very effective primal heuristic: we traverse all variables x_{ij} in descending order according to their current LP-value. If $\pi(i)$ and $\pi^{-1}(j)$ are undefined up to now, we set $\pi(i) = j$ and $\pi^{-1}(j) = i$. It is easy to see that after having traversed all variables we are left with a well-defined permutation π of X. Then we check this permutation for membership in G; see Sect. 4.2. If successful, we compare the objective function value of π to the one of the currently best feasible solution, and save π if better.

An even simpler but often effective method to find good primal solutions is to have a look at the given generators $\pi_1, \ldots, \pi_m$ or at certain compositions of them. These are feasible by definition. In our implementation, we check π_i and $\pi_i \circ \ldots \circ \pi_1$ for $i = 1, \ldots, m$. The permutations of the second type are particularly useful when minimizing the number of fixed points.

4.4 Branching and Enumeration

According to our evaluation, the following branching and enumeration strategy clearly outperforms all standard techniques implemented in ABACUS: as branching variable, we always choose the one with the largest fractional LP-value; the subproblem considered next in the enumeration tree is the one with most variables set to one.

The aim of this strategy is to set as many variables as possible as soon as possible. For this, notice that setting a variable x_{ij} to one implies setting all variables $x_{i'j}$ with $i' \neq i$ and all variables $x_{ij'}$ with $j' \neq j$ to zero by (1) and (2). More generally, it allows to set to zero any variable $x_{i'j'}$ such that $(i, i') \not\sim (j, j')$ with respect to the 2-partitioning. On the other hand, setting a variable to zero doesn't necessarily allow to set further variables.

Our enumeration strategy is thus designed to find feasible solutions quickly. This is also true for the depth first method. In fact, the results for depth first enumeration were similar to those for our strategy. On the other hand, breadth first or combinations of breadth and depth first resulted in significantly longer runtimes.

Our branching strategy also aims at finding feasible solutions quickly, which is done by choosing variables with an LP-value already close to one. According to our enumeration strategy, in the one of the two resulting subproblems considered first this variable will be set to one. For our problem, this method turned out to be much more successful than choosing a variable with an LP-value close to one half.

For reducing the size of the enumeration trees further, we also plan to implement the isomorphism pruning technique devised by Margot [10]. The idea of this technique is to prevent that isomorphic subproblems in this tree are solved more than once. This method is profitable whenever the linear model under consideration has a large symmetry group. In our case, it is easy to see that the group G itself is a subgroup of this symmetry group—as long as we do not take the objective function into account. The most useful way to embed G is to define the symmetry induced by π as mapping x_{ij} to $x_{\pi(i)\pi(j)}$ for all $(i, j) \in X^2$.

As the symmetry group must also fix the objective function, it is usually trivial for random instances. In these cases, isomorphism pruning is pointless. For some special problems, however, the objective function is compatible with G. E.g., this is true for minimizing the number of fixed points. In these cases, we can apply isomorphism pruning with respect to G. The full symmetry group of our model may be larger than G; nevertheless, it is preferable to use G as we do not have to spend time for computing the symmetry group then, as necessary in other applications, but get it for free as part of the problem instance. Furthermore, we can make use of the group theoretic algorithms needed for the membership tests anyway.

4.5 Experimental Results

In the following, we give a short summary of experimental results obtained for our implementation of the branch & cut-algorithm presented here. All relevant parts of this implementation have been described above; we didn't use any further improvements or refinements. Unless stated otherwise, we sticked to the standard parameter setting of ABACUS. The experiments were carried out on an Intel Pentium 4 processor with 2.80 GHz. All runtime figures are given in CPU-seconds; they do not contain the time needed for the membership test preprocessing, as our focus is on the branch & cut-algorithm (and our implementation of the group theoretic algorithms is surely not state of the art). In all experiments, we restricted the total runtime to five CPU-minutes per instance.

We are not aware of any other existing algorithm for linear optimization over permutation groups. Thus we cannot present any comparison here but have to state our results independently. Nor can we refer to any existing test set. Instead, we had to create our own test instances, which was a delicate task as on one hand it was hard to decide which instances were appropriate and on the other hand the performance of the algorithm strongly depended on the type of the chosen instances.

4.5.1 Test sets

Our aim was to create test instances randomly and automatically. The hardest instances we could construct in this way were produced as follows: given the domain size n, we create a permutation π over $X = \{1, \ldots, n\}$. For every point $i \in X$, we fix $\pi(i) = i$ with a probability of $1 - \frac{1}{\sqrt{n}}$. The remaining points in X are permuted randomly by π. The reason for choosing so many fixed points for each generator was that otherwise too many created instances turned out to be full symmetric groups.

We produced two classes of instances: *small* instances were generated by $\lfloor \frac{1}{2}\sqrt{n} \rfloor$ permutations, each one created as just explained, while *large* instances were generated by $\lfloor 2\sqrt{n} \rfloor$ such permutations. Creating instances in this way, we often encountered permutation groups being full symmetric groups when restricted to their orbits. Being trivial for our approach, such instances were rejected. We created 100 instances for each $n = 20, 40, 60, 80, 100$ and for each of the two classes of instances.

When dealing with permutation groups, the order of G usually has a stronger impact on runtime than the size n of the domain and the number m of generators. The group orders of our test instances are shown in Fig. 1.

4.5.2 Objective functions

We report experimental results for two types of objective functions leading to extremely different runtime figures. For the first type, coefficients were chosen randomly from $\{0, \ldots, n-1\}$. For the second type, we set $c(i, j) = 1$ if $i = j$ and $c(i, j) = 0$ otherwise; this models the problem of minimizing the number of fixed points arising in our original application [5]. We also experimented with other objective functions, yielding results lying between those for the two reported cases.

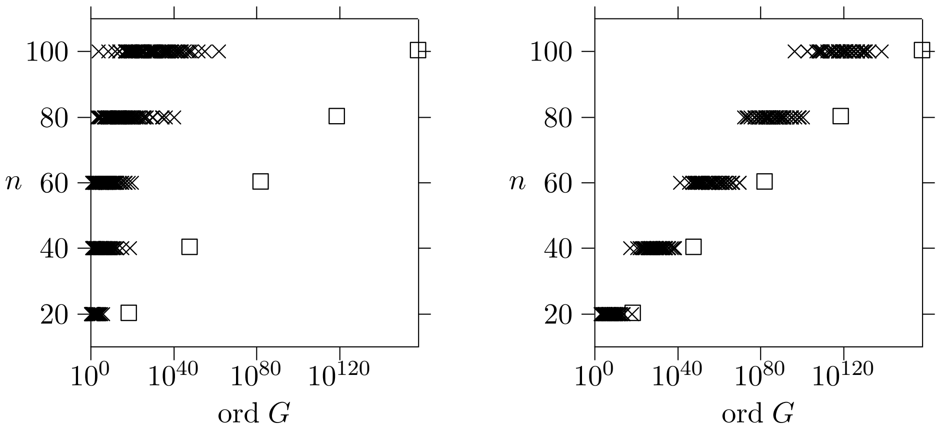

Fig. 1. Group orders for small (left) and large (right) instances. Every cross represents a single instance. The boxes indicate the maximum size $n!$ of a permutation group on n elements

4.5.3 Results for random objective functions

The runtime results for random objective functions are given in Table 1 for small instances and Table 2 for large instances. For each domain size, we list average and maximal figures for the total runtime, for the time needed to find an optimal solution—not necessarily knowing its optimality at this point—, for the number of subproblems considered in the enumeration tree, including the root problem, and for the number of LP-relaxations solved during the optimization process.

Table 1
Results for small instances, random objective functions

	runtime		opttime		#subprobs		#LPs		gap	
n	avg	max	avg	max	avg	max	avg	max	avg	max
20	0.00	0.04	0.00	0	2.0	17	3.3	22		
40	0.08	3.63	0.00	0	18.7	929	29.5	1365		
60	1.50	77.23	0.18	6	158.9	7485	234.8	11230		
80	8.09	4 %	1.79	130	410.4	10175	655.6	14920	0.1 %	4.2 %
100	20.22	28 %	1.99	91	678.0	11015	1014.6	16092	0.5 %	8.8 %

Recall that we restricted runtime to five CPU-minutes per instance. The figures reported in Table 1 and Table 2 only refer to those instances that could be solved within this limit. Where this was not the case for all 100 instances represented by a row of the table, we state the percentage of unsolved instances instead of the maximal runtime. In the last column, we then note the average and maximal gap over all 100 instances, i.e., the quality guarantee for

Table 2
Results for large instances, random objective functions

	runtime		opttime		#subprobs		#LPs		gap	
n	avg	max	avg	max	avg	max	avg	max	avg	max
20	0.03	1.57	0.01	1	11.9	571	14.8	667		
40	1.91	116.82	0.22	14	139.4	9165	186.3	12427		
60	3.03	5 %	0.03	3	50.2	4211	83.6	7044	0.1 %	3.0 %
80	2.96	15 %	0.96	81	25.9	1031	31.2	1440	0.5 %	7.0 %
100	1.39	25 %	0.05	1	3.5	65	6.2	130	1.2 %	12.6 %

the best found solution relative to the unknown optimal solution.

First of all, the results show that runtime varies strongly from one instance to the other, even within the same test set. In fact, up to 28% of the instances could not be solved to proven optimality within the time limit of five CPU-minutes, while the remaining instances were solved within a few CPU-seconds on average. This fact becomes obvious in Fig. 2, showing that nearly all large instances were either solved very quickly or not solved at all. For larger groups, the percentage of the latter increases. However, hard instances also occur for smaller groups. For the small instances, the general picture was the same.

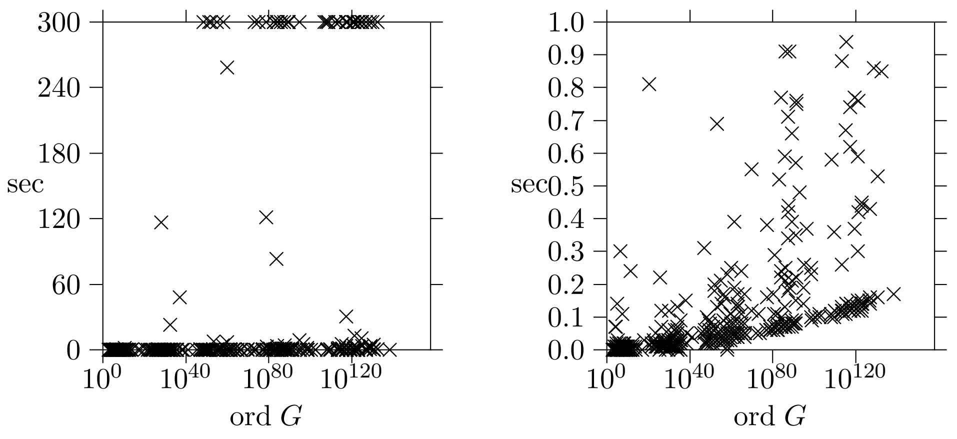

Fig. 2. Runtime results for large instances by the size of the permutation group: on the left, all instances; on the right, instances needing at most one CPU-second

Looking at the gaps, it is remarkable that even for the unsolved instances the algorithm in general yields a feasible solution of reasonable quality. In many cases, this solution should even be optimal—this hope is underpinned by the fact that for the solved instances the optimal solution was usually found much earlier than its optimality was proven: on average, this happened after 13.8% of the total runtime.

Comparing the results for small and large instances, it is not obvious that one of the two classes is significantly harder: while for the latter the percentage of unsolved instances is higher in most cases, the solved instances need less time in general. In summary, the algorithm behaves more capricious for large instances. Most figures in Table 2 decrease for larger n because more of the hard instances reach the time limit then and are hence not included any more.

4.5.4 Fixed point minimization

Up to now, all reported results referred to random objective functions. In our main application, we desire to minimize the number of fixed points. For this special objective function, our algorithm performed dramatically better; see Table 3 for small groups and Table 4 for large groups. A possible explanation is that the probability of a feasible solution to be (provably) optimal is much higher here. This is true in particular if there exist fixed-point-free permutations in G. Consequently, large instances needed much less subproblems and LPs to be solved than small ones in general. Yet runtime for large instances is longer, as the number of variables in the LPs that can be omitted is smaller in this case.

Table 3
Results for small instances, minimizing the number of fixed points

	runtime		opttime		#subprobs		#LPs		gap	
n	avg	max	avg	max	avg	max	avg	max	avg	max
20	0.00	0.01	0.00	0	1.1	7	1.0	5		
40	0.00	0.05	0.00	0	1.3	19	1.2	11		
60	0.01	0.09	0.00	0	1.2	19	1.1	13		
80	0.02	1.58	0.01	1	2.6	161	2.6	164		
100	0.07	1 %	0.04	4	4.8	333	4.1	287	0.0 %	1.6 %

Table 4
Results for large instances, minimizing the number of fixed points

	runtime		opttime		#subprobs		#LPs		gap	
n	avg	max	avg	max	avg	max	avg	max	avg	max
20	0.00	0.09	0.00	0	1.9	63	1.9	75		
40	0.02	0.42	0.00	0	2.1	57	1.6	32		
60	0.04	0.06	0.00	0	1.0	1	1.0	1		
80	0.07	0.13	0.00	0	1.0	1	1.0	1		
100	0.13	1 %	0.00	0	1.0	1	1.0	1	0.1 %	10.0 %

5 Conclusion

We presented a branch & cut-algorithm for linear optimization over permutation groups, based on an investigation of the polytope spanned by all corresponding permutation matrices. This polytope is hard to examine in a traditional way, as we do not even know how to compute its dimension in polynomial time. In particular, searching for general classes of facet-inducing inequalities is a hopeless task. Nonetheless, we could find a complete linear description of this polytope including a class of cutting planes that we can separate in a fast heuristic way.

Together with other ingredients, this leads to a promising algorithm. The runtime of this algorithm strongly depends both on the structure of the group and the objective function, but according to our experience instances from practice are much easier to solve in general than the instances we created for our evaluation; see e.g. [5].

Future improvement can be expected from further examination of the class of homomorphism constraints, in particular, from finding more comprehensive subclasses that can still be separated quickly, even if only heuristically.

References

[1] L. Babai, E. M. Luks, and Á. Seress. Fast management of permutation groups I. *SIAM J. Comput.*, 26(5):1310–1342, 1997.

[2] M. Balinski and A. Russakoff. On the assignment polytope. *SIAM Rev.*, 16:516–525, 1974.

[3] A. I. Barvinok. Combinatorial complexity of orbits in representations of the symmetric group. *Adv. Sov. Math.*, 9:161–182, 1992.

[4] G. Birkhoff. Tres observaciones sobre el algebra lineal. *Revista Facultad de Ciencias Exactas, Puras y Applicadas, Universidad Nacional de Tucuman, Serie A (Matematicas y Fisica Teoretica)*, 5:147–151, 1946.

[5] C. Buchheim and M. Jünger. Detecting symmetries by branch & cut. *Math. Prog., Ser. B*, 98:369–384, 2003.

[6] M. L. Furst, J. Hopcroft, and E. M. Luks. Polynomial time algorithms for permutation groups. In *Proc. 21st IEEE Foundations of Computer Science*, pages 36–41, 1980.

[7] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization.* Springer-Verlag, 1988.

[8] M. Jünger and S. Thienel. The ABACUS system for branch-and-cut-and-price-algorithms in integer programming and combinatorial optimization. *Softw. Pract. Exper.*, 30(11):1325–1352, 2000.

[9] A. Lubiw. Some NP-complete problems similar to graph isomorphism. *SIAM J. Comput.*, 10(1):11–21, 1981.

[10] F. Margot. Exploiting orbits in symmetric ILP. *Math. Prog., Ser. B*, 98:3–21, 2003.

[11] A. McIver and P. M. Neumann. Enumerating finite groups. *Quart. J. Math. Oxford*, 2(38):473–488, 1987.

[12] B. D. McKay. `nauty` user's guide (version 1.5). Technical Report TR-CS-90-02, Computer Science Department, Australian National University, 1990.

[13] C. C. Sims. Computation with permutation groups. In S. R. Petrick, editor, *Proc. 2nd Symp. on Symbolic and Algebraic Manipulation*, pages 23–28, 1971.

[14] CPLEX 7.1. `www.ilog.com/products/cplex`, 2000.