Solving Puzzles related to Permutation Groups

Sebastian Egner* Markus Püschel

Institut für Algorithmen und Kognitive Systeme
Universität Karlsruhe
FRG-76128 Karlsruhe
{egner,pueschel}@ira.uka.de

Abstract

Physical puzzles that can be solved with methods for permutation groups are considered and classified according to a number of abstract properties. The approach for solution is based on word stabilizer chains (the transversal elements are factored in words in a given list of generators). New methods are presented to construct word stabilizer chains that take advantage of the special structures present in physical puzzles. In many cases the methods are successful in avoiding the exponential growth of word length that plagues stabilizer chain methods to construct transversal elements. Finally, a classification scheme for puzzles is presented which helps with finding the permutation group related to the puzzle.

1 Introduction

It is a well known fact that Rubik's Cube can be solved with algorithms for permutation groups. But how do you do this efficiently? In fact, it is essential to produce a solution with as few moves as possible. This paper focuses on puzzles where permutation group methods contribute substantially to their solution. In some cases the puzzle has to be looked at in the right way to see the group in it. The puzzles are solved with new methods to construct and use stabilizer chains containing words in the generators as transversal elements. In particular:

- A heuristic method for choosing a good base.
- Optimized procedures to create stabilizer chains containing short words (based on ideas of Minkwitz [13]).
- Improved use of a stabilizer chain (similar to an idea of Osterlund [16]) to factor a given element including a statistical analysis of the word length to be expected.

Our methods yield substantially shorter words than previously known algorithms and are quite successful in using the special structure of real-world puzzles designed for manual solution. The puzzles are classified according to some abstract properties which are relevant for their solution.

*Grant GRK 209/2-96 "Beherrschbarkeit komplexer Systeme"

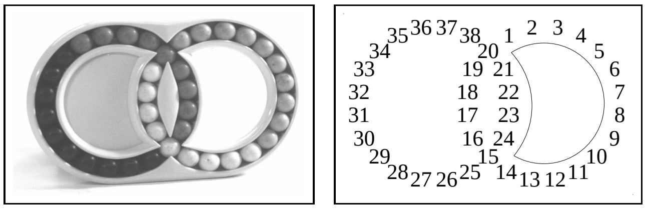


Figure 1: a) The "Pretzel", b) numbering of positions.

2 Factorization in Permutation Groups

In order to formulate the central computational problem treated in this paper some pieces of language are needed. Consider the free group $F = \langle a_1, \ldots, a_r \rangle$ of finite rank r. The identity of F is written id, an element which is either a generator a_i or an inverted generator a_i^{-1} is called a *letter*. A general element $w \in F$ is called a *word*. Any word can be written as a product $x_1 \cdots x_k$, for $k \geq 0$, of letters x_i such that no two adjacent letters cancel. The *length* of the word w is denoted as $|w|$ (k in the representation above). Moreover, we define a total ordering on the words as the lexicographical extension of the following ordering on the letters: $\mathrm{id} < a_1 < a_1^{-1} < a_2 < \ldots < a_r^{-1}$.

The second major ingredient of the formalism is a group homomorphism $w \mapsto \underline{w}$ of F into the symmetric group S_n permuting the finite set $\{1..n\}$ of natural numbers. As usual β^g denotes the image of point β under permutation g. The image group $\langle \underline{a_1}, \ldots, \underline{a_r} \rangle$ will be called G throughout this paper. The main theme of this paper is the following computational problem:

Problem 1 *Given permutations $g_1, \ldots, g_r \in \mathsf{S}_n$ and a subgroup $H \leq G = \langle g_1, \ldots, g_r \rangle$ by generators. Define the homomorphism $\underline{(\cdot)}$ from the free group $F = \langle a_1, \ldots, a_r \rangle$ of rank r into G by $a_i \mapsto \underline{a_i} = g_i$ for all i. Now the task is: For a permutation $g \in G$ find $h \in H$ and $w \in F$ such that $\underline{w} = hg$ and $|w|$ is small.*

In other words, g is factored into a product of the generators $\underline{a_i}$, up to an arbitrary element of H. For trivial H the task is exactly the permutation group element factorization problem. The group H represents a degree of freedom which may be used to find substantially shorter words. This is at best illustrated by an example:

Example Consider Figure 1a) which shows a hardware model of a puzzle which we will call the "Pretzel" (for obvious reasons) in its initial state. The Pretzel consists of two intersecting circular channels containing 38 coloured beads:

10 black, 9 yellow, 10 red and 9 blue. The aim is to render equal coloured beads adjacent by shifting the beads in the channels.

Number the $n = 38$ positions in the channels as in Figure 1b). Then there are $r = 2$ elementary moves defined as $\underline{a_1} = (1, 2, \ldots, 20)$ [turn right channel clockwise] and $\underline{a_2} = (20, \ldots, 24, 15, 25, \ldots, 38)$ [turn left channel clockwise]. The two permutations generate the full symmetric group $G = \mathsf{S}_{38}$. Hence, the innocuous puzzle admits more than $5.2 \cdot 10^{44}$ ways of distributing the beads in the channels! Fortunately, there are just four colours (and not thirty-eight) and this is where the subgroup H comes into play: H contains all permutations of equally coloured beads in the initial state of the Pretzel. Formally, $H = \mathsf{S}_{10} \times \mathsf{S}_9 \times \mathsf{S}_{10} \times \mathsf{S}_9$ is the Young group of the partition (1..10|11..19|25..34|35..38 20..24). Hence, there are just $|G|/|H| \approx 3.0 \cdot 10^{20}$ *observable* states.

Solving the Pretzel now means: Read off a permutation g which takes the initial state into the given (shuffled) state. Then solve Problem 1 to find h and w with $\underline{w} = hg$. Since g maps "initial" into "shuffled" it is the word w^{-1} which represents the sequence of moves to apply. ∎

It should be stressed at this point that producing short words is essential for solving puzzles physically! To our experience it gets embarrassingly difficult to perform more than 200 moves without error. In this sense a number of the physical puzzles we have studied were not solvable with the words produced from previously known methods (for example the Pretzel).

3 Known Methods

A number of different strategies are known for solving Problem 1. We will briefly review them here.

- Define the *short pairs* of G as the sorted list $\mathrm{id} = \mathrm{sp}_1 < \mathrm{sp}_2 < \ldots < \mathrm{sp}_{|G|}$ of the shortest words for the elements in G. (The term "pair" emphasizes that a word w is associated with the permutation $\underline{w}$.) One method to solve Problem 1 is to find $i \in \{1..|G|\}$ such that $g = \underline{\mathrm{sp}_i}$ and to return sp_i. By definition this results in the shortest possible word for g (ignoring H). In particular, the length $|\mathrm{sp}_{|G|}|$ of the last short pair is called the *diameter* of the group G with respect to the given generating list. (Babai and Seress [2] have proved an asymptotically tight upper bound on the diameter as a function of the degree n.) In fact, the short pairs can be enumerated efficiently by a breadth-first enumeration of the words w and testing if $\underline{w}$ has already been generated. (In other words, a minimal spanning tree for the Cayley graph of the generating list $(\underline{a_1}, \ldots, \underline{a_r})$ is being constructed.) It is sufficient to store the last letter of each short pair, which results in a total cost of $|G| \cdot \log_2(2r)$ bits. Even better, Cooperman and Finkelstein [6] have shown that $|G| \cdot \log_2(3)$ bits are sufficient. Clearly, memory is the limiting factor in using the enumeration method. It has been shown by Cooperman, Finkelstein and Sarawagi [7] that the diameter of Rubik's $2 \times 2 \times 2$-Cube ($\approx 3.7 \cdot 10^6$ states) is 11 with respect to the nine generators [turn top, front, left and their inverses and squares]. Since Rubik's $3 \times 3 \times 3$-Cube has $|G| \approx 4.3 \cdot 10^{19}$ states enumeration is out of question with contemporary technology.
- Although the enumeration method for large G runs out of memory it may be possible to use it in two (or more) steps: By magic (or insight) find a subgroup $G_1 < G$ such that G_1 and G/G_1 are both small enough to be enumerated with the enumeration method. Factoring $g \in G$ then consists of finding a short word w_0 such that $g\underline{w_0} \in G_1$ and then finding a short word w_1 such that $g\underline{w_0 w_1} = \underline{\mathrm{id}}$. This method has been applied by Michael Reid to prove that every state of Rubik's Cube can be factored into a word of at most 42 letters in the six generators [turn left, right, top, bottom, front, back]. On the other hand it has been shown by Jerry Bryan [19] that there is a state which does not factor shorter than 24. The stepwise enumeration method results in very short words at the price of enormous tables and extensive searching. We will not consider it in this paper any further.
- The method of choice for permutation groups is the use of stabilizer chains. If the transversal elements are known as words then the stabilizer chain can be used to factor in the permutation group G. (Refer to Butler's book [4] for a nice introduction to stabilizer chains and to the recent paper by Babai, Luks and Seress [1] for the state of the art in handling permutation groups.) The entire problem is how to obtain short words for the transversals? Using the theorem of Schreier (randomized or not) results in an exponential growth of word length (as a function of level). Applied to Rubik's Cube lengths in the 50000s are produced by this approach. The first successful approach known to us is incorporated in the GAP-package `AbStab`, by Osterlund [16, 18] which results in words of length 1500 for Rubik's Cube. A substantial improvement over this is the method of Minkwitz [13] producing words of at most 150 moves with 100 as the average. The method explained below is an improved variant of Minkwitz' approach geared to the special situation of solving puzzles. In particular, a heuristic method is presented to choose a good base automatically and by improved use of the stabilizer chain words of length 60 are reached.

4 Word Stabilizer Chains

Let F, G and the homomorphism $\underline{(\cdot)}$ be defined as in Problem 1. Consider the stabilizer chain

$$G > G_{\beta(1)} > G_{\beta(1),\beta(2)} > \cdots > G_{\beta(1),\ldots,\beta(L)} = E$$

of G with respect to the base $\beta(1), \ldots, \beta(L)$ for some $L \geq 1$. The orbit of the basepoint $\beta(\ell)$ under the action of the stabilizer $G_{\beta(1),\ldots,\beta(\ell-1)}$ is denoted as Γ_ℓ for all levels $\ell \in \{1..L\}$. (Since we will sometimes talk about "going down the stabilizer chain" think of the levels as being arranged vertically, $\ell = 1$ on top, and the orbits extending horizontally to the right.) A collection of words $t(\ell, \gamma) \in F$, for level $\ell \in \{1..L\}$ and basepoint image $\gamma \in \Gamma_\ell$, with the property

$$\gamma^{\underline{t(\ell,\gamma)}} = \beta(\ell) \quad \text{and} \quad \beta(k)^{\underline{t(\ell,\gamma)}} = \beta(k) \text{ for all } k < \ell$$

is called a *word stabilizer chain* (WSC) for F, G, $\underline{(\cdot)}$ with respect to the base $\beta(1), \ldots, \beta(L)$.

Of course, the set $\{\underline{t(\ell,\gamma)} \mid \gamma \in \Gamma_\ell\}$ is an (inverse) transversal of the coset space $G_{\beta(1),\ldots,\beta(\ell-1)} / G_{\beta(1),\ldots,\beta(\ell)}$ and the order of G is $|G| = |\Gamma_1| \cdots |\Gamma_L|$. The following procedure factors the permutation $g \in G$ using the WSC t into a word $\mathtt{factor}(t; g)$ such that $\underline{\mathtt{factor}(t; g)} = g$.

Algorithm factor($t; g$):

$w := \mathrm{id}$;
for $\ell \in \{1..L\}$ **do**
  $w := w \cdot t(\ell, \beta(\ell)^g)$;
  $g := g \cdot \underline{t(\ell, \beta(\ell)^g)}$;
  # loop invariant is $\underline{w} \cdot g^{-1}$, a variant is $g \in G_{\beta(1),\ldots,\beta(\ell)}$
od;
return w^{-1};

A quantitative measure for the quality of a WSC t is the distribution of the word lengths when using t for factorization. We define the *length distribution function* (LDF) of t as the polynomial

$$\mathrm{ldf}(t) = \sum_{g \in G} z^{|\mathtt{factor}(t;g)|}$$

in the formal variable z. Clearly, the z^k-coefficient of $\mathrm{ldf}(t)$ is the number of words of length k in G. An approximation of $\mathrm{ldf}(t)$ that can be computed very fast is

$$\mathrm{ldf}^*(t) = \prod_{\ell=1}^{L} \sum_{\gamma \in \Gamma_\ell} z^{|t(\ell,\gamma)|}.$$

Note that this is not the same as $\mathrm{ldf}(t)$ since cancellation of letters can occur when transversal elements are multiplied in the function `factor`. However, $\mathrm{ldf}^*(t)$ is an upper bound on $\mathrm{ldf}(t)$ in the following sense.

Lemma 1 *Let* $\mathrm{ldf}(t) = \sum_k n_k z^k$ *and* $\mathrm{ldf}^*(t) = \sum_k n_k^* z^k$. *Then for all integers* $K \geq 0$

$$\sum_{k \leq K} n_k \leq \sum_{k \leq K} n_k^*.$$

This means the K-th coefficient of the cumulative distribution function of $\mathrm{ldf}(t)$ *is bounded by the K-th coefficient of the cumulative distribution function of* $\mathrm{ldf}^*(t)$.

Proof Follows from

$$|t(1,\gamma_1) \cdots t(L,\gamma_L)| \leq |t(1,\gamma_1)| + \ldots + |t(L,\gamma_L)|$$

for all $\gamma_\ell \in \Gamma_\ell$, $\ell \in \{1..L\}$ by replacing the summation over G by a summation over $\Gamma_1 \times \cdots \times \Gamma_L$. ∎

To compare ldf, ldf* and the Cayley graph factorization consider Figure 2. The diagram shows word length distributions for the group A_9 (181440 elements) generated by $(1,2,3,4,5)$ and $(5,6,7,8,9)$. The leftmost curve ("Cayley graph") shows the length distribution of the short pairs, which are the words of minimal lengths for the elements. Choosing the base $(7,8,2,3,1,6,9)$ there is a WSC such that every transversal element $t(\ell,\gamma)$ is the shortest possible word. The length distribution function $\mathrm{ldf}(t)$ for this WSC is shown as the middle curve. Finally, the approximation $\mathrm{ldf}^*(t)$ for $\mathrm{ldf}(t)$ is shown as the right curve. Assuming the letters of the transversals are uiid-random variables, cancellation of two adjacent letters at the $L-1$ level interfaces occurs with probability $1/(2r)$ in the free group. Thus, an a priori expectation of the shift of $\mathrm{ldf}^*(t)$ from $\mathrm{ldf}(t)$ is $2(L-1)/(2r) = 3$ letters. This is in correspondence to the shift observed in Figure 2.

There is one important technical point to mention for the application of WSCs to puzzles: In Problem 1 we have introduced a subgroup $H \leq G$ which contains invisible operations of the puzzle. In many cases H contains an entire stabilizer $G_{\beta(1),\ldots,\beta(L_0)}$ in which case the levels $\ell > L_0$ can be discarded. This means that the WSC need only be constructed for levels $\ell \in \{1..L_0\}$, that the procedure `factor` can be stopped at $\ell = L_0$ and that the summation in the LDFs should run over the coset space $G/G_{\beta(1),\ldots,\beta(L_0)}$ instead of G. Lemma 1 remains valid.

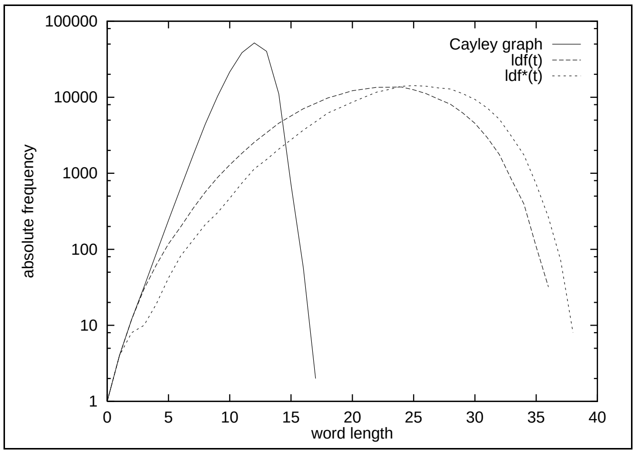


Figure 2: Length distributions for A_9 (in logarithmic scale).

5 Constructing WSCs

The most important assumption underlying our method for constructing a WSC is: *There are very short words $w \in F$ such that the permutation $\underline{w}$ is non-trivial but has many fixed points.* We call these pairs $(w, \underline{w})$ the *lazy pairs.*

Note that the existence of lazy pairs is something special. A random generating set of a group like S_{38} is very unlikely to produce a short lazy pair. However, all puzzles we have encountered are generated such that lazy pairs exist and are found among the first few short pairs. For example, in the Pretzel the 68-th short pair is $\underline{a_1^{-1} a_2^{-1} a_1 a_2} = (1,20,21)(15,25,16)$ which stabilizes 32 of the 38 points! A metaphysical argument in support of the assumption is: If a puzzle did not admit short lazy pairs then it would not be possible to solve the puzzle by hand at all.

The method to construct the word stabilizer chain essentially consists of two steps: First a number of lazy pairs is produced (Section 5.1) then a base adapted to them is chosen (Section 5.2). Once the base has been fixed, a corresponding WSC is constructed by finding transversal elements $t(\ell,\gamma)$ of short word length (Section 5.3).

5.1 Lazy Pairs

Technically, lazy pairs are recruited from the first few thousand short pairs (as defined in section 3) and from the first few thousand elements of the group generated by the commutators of the generators of G. The second source only produces words of exponent sum zero which increases the chance of being lazy, as our experience with real-world puzzles shows. Let the calculated lazy pairs be the list of words $\mathrm{lp} = (\mathrm{lp}_1, \ldots, \mathrm{lp}_{|\mathrm{lp}|})$, sorted by the number of fixed points, such that lp_1 has as many as possible, and sorted by word length with equal number of fixed points. For a permutation g we denote the set of moved points of $g \in G$ as $\mathrm{mp}(g)$.

Hence, $|\mathrm{mp}(g)|$ is the number of moved points of g. By abuse of notation, understand $\mathrm{mp}(w)$ as $\mathrm{mp}(\underline{w})$ for a word $w \in F$.

5.2 Choice of Base

Recall that the central problem in constructing a WSC using Schreier's subgroup theorem lies in the fact that the word length grows exponentially in the length of the base. The longest words are those which lie in the lowest (most stabilizing) levels of the WSC. To avoid this growth, the base is chosen such that some lazy pairs are spread over the levels as much as possible. In combination with the methods to complete the WSC (Section 5.3) this "seed" decreases the word lengths on all levels dramatically.

Technically, the first step is to increase the set of available lazy pairs by constructing the *conjugated pairs* as $\mathrm{lp}_i^{\mathrm{sp}_j}$ for all i, j. However, for every occuring set $\mathrm{mp}(\mathrm{lp}_i^{\mathrm{sp}_j})$ of moved points only the shortest conjugate (in terms of word length) is stored which results in a substantially shorter list. We obtain the list $\mathrm{cp} = \{\mathrm{cp}_1, \ldots, \mathrm{cp}_{|\mathrm{cp}|}\}$. The method described below for stepwise refinement of the base works best using conjugated pairs (and not only lazy pairs).

The base is chosen according to the "strategy of smallest ascending steps". The following algorithm takes the set of conjugated pairs and returns an ordered partition b of the set $\{1..n\}$ in the form of a list of sets. A base can be constructed from this ordered partition by joining the blocks from left to right and specifying the order of points within the blocks. The algorithm works by repeatedly splitting off the smallest possible portion $\mathrm{mp}(u)$ of the leftmost block b_1.

Algorithm `chooseBase`(cp):

$b := (\{1..n\})$;
`while` $\mathrm{cp} \neq \emptyset$ `do`
  u := shortest word of $\{v \in \mathrm{cp} \mid |\mathrm{mp}(v) \cap b_1| = \min\}$;
  b := $(b_1 \cap (\{1..n\} \setminus \mathrm{mp}(u)), b_1 \cap \mathrm{mp}(u), b_2, \ldots, b_{|b|})$;
  $\mathrm{cp} := \{v \in \mathrm{cp} \mid \mathrm{mp}(v) \cap b_1 \neq \emptyset\}$;
  # loop invariant "$v \in \mathrm{cp}$ moves b_1", a variant is $|\mathrm{cp}|$
`od`;
`return` b;

Finally, let us mention that the function `chooseBase` as presented above is not the whole truth. In our implementation of the method two modifications are used: First, it is possible to specify a final piece of the base b by hand. This is simply done by starting at a partition different from $(\{1..n\})$ in function `chooseBase`. The final part of the base is being used to take the symmetry of a puzzle into account — in many cases the group H contains an entire stabilizer so it is important to be able to make the stabilized points the final points of the base as described at the end of section 4.

The second modification concerns the trade-off between word length and laziness: There is a tendency of the base choosing process to get stuck by suboptimal decisions if there are too many conjugated pairs to choose from. (This is an empirical observation from our examples, of course.) Hence, we limit the word length of the conjugated pairs by a constant c and run `chooseBase` on the restricted set $\{v \in \mathrm{cp} \mid |v| \leq c\}$. This results in a partition b which can still be refined by looking at all $v \in \mathrm{cp}$ afterwards. Since c is a magic constant the entire process of constructing a base is performed for different values of c. Then the winner is chosen as the base which spreads the words most evenly among the stabilizer chain.

5.3 Completing the Transversals

After a base has been constructed the word stabilizer chain can be initialized. By storing the words u being chosen by `chooseBase` there is already a small but important supply of transversal elements. Other possible sources of words are the short pairs, the lazy pairs, the conjugated pairs and random words in the generators. All these elements w can be inserted into the stabilizer chain by finding out the maximal level ℓ such that $\beta(1), \ldots, \beta(\ell-1)$ are stabilized by $\underline{w}$. Then w is a possible choice for the transversal element $t(\ell, \beta(\ell)^{\underline{w}^{-1}})$. After the word stabilizer chain has been partially filled it is completed by producing new elements from known ones. Eventually, the shortest word ever seen as a transversal element $t(\ell, \gamma)$ is selected. Useful processes to form new elements are

- "Orbit graph completion": Let w be a word which maps δ into γ. Then $w \cdot t(\ell, \gamma)$ is a transversal element of the form $t(\ell, \delta)$. Note that w may well stabilize the base point $\beta(\ell)$, which means that transversal elements at lower levels may be of use on higher levels. Storing the entire orbit graph in a datastructure makes maximum use of the words being present at a given level.

- "Quotients": Let u and v be transversal elements of the form $t(\ell, \gamma)$. Then $u^{-1}v$ stabilizes the base point $\beta(\ell)$ and hence lies in a level below ℓ. Producing quotients is the most important way to get into lower levels, even though the word length approximately doubles. As Minkwitz [13] has shown, this is particularly useful in combination with random words being inserted and stored on the individual levels.

- "Schreier triples": Let u be any word mapping δ into γ. Then $t(\ell, \delta)^{-1} \cdot u \cdot t(\ell, \gamma)$ stabilizes base point $\beta(\ell)$. If u stabilizes $\beta(1), \ldots, \beta(\ell-1)$ then the newly produced word lies in a level below ℓ. The Schreier triples increase the word length approximately by a factor of three. On the other hand, using Schreier triples is the only known method guaranteed to fill the lower levels ([9], Kap. I, Satz 19.10). For the purposes of solving puzzles we have never been forced to make use of Schreier triples. (In addition, Schreier triples are expensive to use since a transversal of a stabilizer does not necessarily contain a generating set.)

In contrast to the methods above Minkwitz [13] ignores the completion of the orbit graph and the production of Schreier triples and inserts many randomly generated words at a high rate: On each level a fixed number of the shortest words are stored and are occasionally being used to produce quotients. This method is easy to implement and performs very good.

The most delicate issue in the construction of a word stabilizer chain (WSC) is how to combine the individual processes of inserting, storing and producing new words. This greatly affects the time needed to arrive at a good and complete WSC. In fact, at the moment it is not clear to us which strategy will be the best and why.

The method we have implemented into a GAP [18] program first inserts the few words used in `chooseBase` and the few thousand short pairs. Then quotients are formed going from the lowest levels of the WSC to the top (this avoids an avalanche of words). On every level the orbit graph is completed as much as possible. At this point most of the WSC is already filled (in the examples considered!). Then all conjugated pairs are inserted and quotients are formed bottom-up again. If this has not been sufficient then random

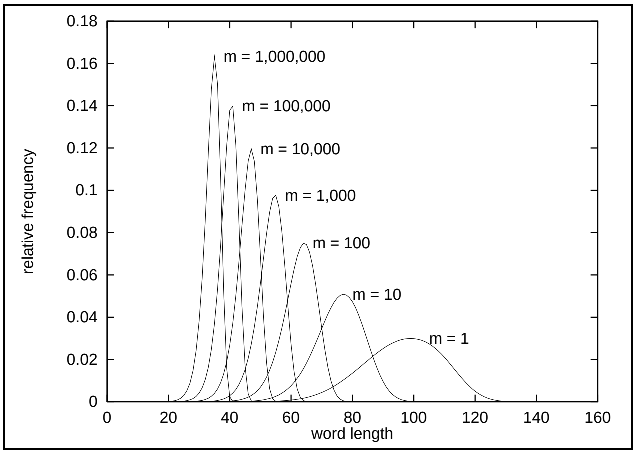


Figure 3: Estimated effect of "trembling" on the word length distribution of a particular WSC for Rubik's Cube.

words are inserted, essentially as in Minkwitz' approach. In contrast to Minkwitz, we insert relatively few words but try to make maximum use of them. This approach exploits the fact that the choice of base matches the information present in the conjugated pairs. For an empirical impression of the performance refer to the gallery at the end of this paper.

It should be stressed at this point that the methods presented here avoid the exponential growth in word length but there is no *guarantee*! Once a base has been chosen there is an optimal WSC in the sense that every $t(\ell, \gamma)$ is the shortest word of all words representing an element of the corresponding coset. This optimal WSC can easily be found by enumerating the entire group and inserting all short pairs into the WSC as described above. The maximum length of any transversal word is bounded by the diameter of the Cayley graph. Hence, short transversal elements always exist! The problem here is: How to construct them without enumerating the group? The presented methods are *one way* to obtain relatively short words within a reasonable amount of time and memory.

6 Improving Word Lengths

In this section we present and analyze a method that we call "trembling", which uses basically the same idea as the procedure `Shrink` in [16]. It represents a more efficient way of using a word stabilizer chain for factorization. As a rule of thumb, "trembling" reduces the word lengths to about 2/3 of the lengths obtained by the procedure `factor` given above. The following procedure factors $g \in G$ using the WSC t by considering the first m short pairs $\mathrm{sp}_1, \ldots, \mathrm{sp}_m$ for improvement.

Algorithm $\mathtt{factorTremble}(t, m; g)$:

$w := \mathtt{factor}(t; g)$;
$\mathtt{for}\ i \in \{1..m\}\ \mathtt{do}$
  $v := \mathrm{sp}_i^{-1} \cdot \mathtt{factor}(t; \underline{\mathrm{sp}_i} \cdot g)$;
  $\mathtt{if}\ |v| < |w|\ \mathtt{then}\ w := v;\ \mathtt{fi}$;
  # loop invariant is $\underline{w} = g$, a variant is $m - i$
$\mathtt{od}$;
$\mathtt{return}\ w$;

The idea of the method is to make a small step (in terms of word length) away from g and hope that this new permutation factors shorter than g itself — doing this repeatedly means "trembling" around g. In contrast to `factorTremble` the procedure `Shrink` uses (recursively) only the generators to "step away" from g. We will now give evidence why this simple strategy decreases the word length considerably.

Assume that $|G|$ is large compared to m. Then the word lengths of the first m short words are small compared to the lengths produced by factoring an arbitrary element of G. This gives us reason to neglect the length of sp_i^{-1} in the analysis of procedure `factorTremble`. As a second assumption

$$\text{estimate} \quad \frac{1}{m}\sum_{i=1}^{m} z^{|\mathtt{factor}(t; \underline{\mathrm{sp}_i} \cdot g)|} \quad \text{by} \quad \frac{1}{|G|}\mathrm{ldf}^*(t).$$

The latter of which can easily be computed. An indication why the length distribution of the trembled elements should be close to that of the entire group is the fact that a random walk on a Cayley graph converges to the uniform distribution (follows from [15], Theorem 6.15). And moreover the Cayley graphs which we study in puzzles are often rapidly mixing so that the convergence is fast.

Lemma 2 *The length distribution function for procedure* `factorTremble` *using the first* $m \geq 1$ *short pairs is defined as*

$$\mathrm{ldf}(t, m) = \sum_{g \in G} z^{|\mathtt{factorTremble}(t, m; g)|}.$$

This is approximately equal to

$$\mathrm{ldf}^*(t, m) = |G| \cdot \sum_{k=0}^{\infty} \left((1 - \sum_{i=0}^{k-1} p_i)^m - (1 - \sum_{i=0}^{k} p_i)^m \right) z^k,$$

where $p_i \cdot |G|$ *is the* z^i*-coefficient of the length distribution function* $\mathrm{ldf}^*(t)$.

Proof By the first assumption above we drop sp_i^{-1} from the words v in procedure `factorTremble`. By the second assumption we treat $|\mathtt{factor}(t, \underline{\mathrm{sp}_1} g)|, \ldots, |\mathtt{factor}(t, \underline{\mathrm{sp}_m} g)|$ as $\mathrm{ldf}^*(t)/|G|$-distributed independent random variables.

Now consider independent identically distributed integer valued random variables $X_1, \ldots, X_m$ for $m \geq 1$. Then the probability $Q_k = \Pr\{\min(X_1, \ldots, X_m) = k\}$ can be computed by enumerating the non-vacuous μ-subsets and considering $X = k$ on the subset and $X > k$ on the complement. This gives

$$Q_k = \sum_{\mu=1}^{m} \binom{m}{\mu} \Pr\{X = k\}^{\mu} \Pr\{X > k\}^{m-\mu}.$$

Completing the binomial sum gives the result. ∎

Obviously, $\mathrm{ldf}(t, 1)$ is the polynomial $\mathrm{ldf}(t)$ as defined in Section 4 and $\mathrm{ldf}(t, m)$ for $m \geq |G|$ is the perfect length distribution as obtained by representing any group element by the shortest possible word.

Consider Figure 3 for an example of using the trembling method. The diagram shows the estimated length distributions $\mathrm{ldf}^*(t, m)/|G|$ for a good word stabilizer chain of the Rubik's Cube for various values of m. As expected, raising m results in shifting the distribution towards lower word lengths and reducing the variance. The 2/3-rule of thumb is reached at $m = 68$. This means: Using the first 68 short pairs for trembling reduces the word lengths to 2/3 of the untrembled word length.

7 Gallery and a Little Taxonomy of Puzzles

After having solved a number of physical puzzles it became clear to us that there are some abstract properties by which the puzzles can be classified. This observation has led us to a classification scheme, a taxonomy, which serves as a "coordinate system" to locate permutation related puzzles in. Of course this classification does not claim to be complete in any sense: Its only purpose is to help finding the permutation group related to a puzzle, if possible. We will now present this taxonomy by introducing typical examples displaying the various properties. Here is the list of the properties together with the examples used for illustration:

- Pure permutation groups (Rubik's Cube)
- Abelian permutation groups (Rubik's Clock)
- Invisible subgroups (Pretzel, Organ)
- Monoids made from groups (Loyd's 15-Puzzle, M_{13})
- Combinatorial modelling (MegaChallenger)
- Microreversibility (Organ)

Some of the puzzles mentioned in this section will be explained more in detail (The Organ, Loyd's 15-Puzzle, MegaChallenger), others only serve as testing objects for our methods (see Table 1).

7.1 Pure Permutation Groups

Certainly, the most famous permutation related puzzle is The Rubik's Cube. (A Java implementation by L. D. Bradly can be found in [3].) The Rubik's Cube is an example of a puzzle that can be modelled by a permutation group in a straightforward way. As mentioned in Section 3 the diameter d of the Cayley graph for the Rubik's Cube, generated by the six quarter turns at the faces, is known to satisfy $24 \leq d \leq 42$. With the methods of Minkwitz, or the ones presented in this paper, a stabilizer chain is obtained which gives word lengths 150 at most with 100 on the average. The "tremble method" presented in Section 6 uses these stabilizer chains to yield words of length 60.

There are other puzzles that can be modelled by a permutation group in a straightforward way: the $2 \times 2 \times 2$ Cube, the $4 \times 4 \times 4$ Cube, the Icosahedron, the Dodecahedron, "Rubik's Clock", a pyramid of 10 four-sided pieces to be assembled, a Pretzel-like machine with 20 counters realizing $\langle (1, \ldots, 20), (1,4)(2,3) \rangle$ (sold by Binary Arts) and a sphere called Masterball with four turnable belts in latitude six planes in longitude and twistable halves.

7.2 Abelian Permutation Groups

If the group is abelian then the problem of finding a shortest word for a given group element is equivalent to the "nearest vector problem" of a finitely generated integer lattice: Let $G = \langle \underline{a}_1, \ldots, \underline{a}_r \rangle$ be an abelian group and $g \in G$ an element to be factored. Clearly, the length of a word $a_1^{e_1} \cdots a_r^{e_r}$ representing g is the ℓ_1-norm $|e_1| + \ldots + |e_r|$ of the exponent vector $e \in \mathbb{Z}^r$. The set $\{e \mid g = \underline{a}_1^{e_1} \cdots \underline{a}_r^{e_r}\}$ of all exponent vectors representing g is of the form $e^{(0)} - R$ where R is the $\mathbb{Z}$-module of relators and $e^{(0)}$ is a particular solution. Hence, the problem of finding a shortest word representing g is the same as finding a lattice element $e^{(r)} \in R$ such that the ℓ_1-norm of $e = e^{(0)} - e^{(r)}$ is minimal. In other words, the element $e^{(r)}$ is a nearest vector to $e^{(0)}$ with respect to the ℓ_1-norm.

Name	r	states	max	trembling		
				1	100	1000
Rubik's Cube	6	$4.3 \cdot 10^{19}$	155	96.3	62.7	53.3
Rubik's 2-Cube	9	$3.7 \cdot 10^{6}$	39	27.7	10.7	[6.3]
Rubik's 4-Cube	12	$5.7 \cdot 10^{48}$	519	344.7	281.3	262.4
Icosahedron	12	$7.1 \cdot 10^{40}$	193	124.4	100.6	93.3
Masterball	5	$4.4 \cdot 10^{26}$	449	277.2	174.3	151.0
Pretzel	2	$3.0 \cdot 10^{20}$	634	410.7	330.0	304.8
A_9 (Figure 2)	2	$1.8 \cdot 10^{5}$	38	24.3	10.2	[5.9]
Nintendo-Barrel	4	$1.7 \cdot 10^{16}$	261	138.7	97.3	84.6
Loyd's Fifteen	3	$1.0 \cdot 10^{13}$	97	59.6	38.2	31.2
Rubik's Fifteen	8	$1.0 \cdot 10^{13}$	90	56.4	34.7	28.3
Conway's M_{13}	54	$1.2 \cdot 10^{6}$	13	8.2	2.9	[1.8]
TripleCross	2	$2.9 \cdot 10^{9}$	112	74.9	45.1	35.3
Organ	6	$2.1 \cdot 10^{12}$	122	83.4	59.2	51.2
MegaChallenger	4	$3.3 \cdot 10^{13}$	130	82.1	47.2	37.5

Table 1: Word lengths ("max" is an upper bound, "trembling" are expectation values from Lemma 2 for values $m \in \{1, 100, 1000\}$) for different puzzles in r group generators. Numbers in brackets are unphysical since the estimation ldf* is used.

Seen algorithmically, the "nearest vector problem" is NP-hard. However, deterministic enumeration methods based on lattice basis reduction techniques are quite efficient and useful in practice. Please refer to the works of H. Ritter and M. Kaib [17, 10] for lattice basis reduction with respect to non-ℓ_2-norms. In addition, for the purpose of solving puzzles it is sufficient to find a short vector (not necessarily shortest). In this case LLL-reduction may be used because $||e||_2 \leq ||e||_1 \leq \sqrt{r}||e||_2$ for all $e \in \mathbb{Z}^r$.

Example A typical example of an abelian puzzle is Rubik's Clock. It consists of 18 clocks (reading in 1..12) arranged in 3×3 arrays on the two sides. All clocks are driven by the clocks at the four corners which are linked with other clocks depending on the state of four buttons. The aim is to reset all clocks.

The set of states is the abelian group $G = \mathsf{Z}_{12}^{14}$ since the back and front clock at the corners are interlocked. There are $r = 30$ generators since some of the $4 \cdot 2^4$ elementary moves have identical effect on the clocks. ■

7.3 Invisible Subgroups

As it has been pointed out already, there are many puzzles admitting a subgroup H of permutations transforming the initial state in a non-observable way. This *invisible subgroup* reduces the number of states to be distinguished. In all cases we have encountered this is realized by reducing the number of available "colours" of the moving pieces. (The "colour" identifies a face on the Rubik's Cube, a bead in the Pretzel or a ball in the Organ.) A good example for the role of the invisible subgroup is the Pretzel presented at the beginning of the paper.

7.4 Monoids Made from Groups

The largest class of interesting puzzles related to permutations cannot be modelled by a group in a straightforward way — they have to be looked at in the right way! A famous example of this kind is Sam Loyd's 15-Puzzle. (A Java implementation by M. Mortazavi can be found in [14].) The 15-Puzzle requires a generalization of the formal modelling from group representations to monoid representations with a special structure. We will now briefly introduce this generalization here without going into details due to space limitations. For an introduction to the theory of monoids refer to the book of Lallement, [11]. An implementation of several monoid theoretic algorithms is available in the AMoRE-system, [12]. Let $\mathcal{T}(\Sigma)$ be the full transformation monoid of mappings $\Sigma \to \Sigma$ with the composition of mappings as multiplication. We write $s \cdot t$ for the image of $s \in \Sigma$ under $t \in \mathcal{T}(\Sigma)$.

Recall that we used to specify the Pretzel by writing down permutations $\underline{a}_1, \ldots, \underline{a}_r \in \mathsf{S}_n$. These permutations define a permutation representation $\underline{(\cdot)}$ of the free group $\langle a_1, \ldots, a_r \rangle$ by mapping $a_i \mapsto \underline{a}_i$ for all i. For the Pretzel $r = 2$ and $n = 38$, so the permutations can easily be specified by their lists of image points.

As a generalization, the 15-Puzzle can be seen as a physical implementation of a transformation representation of a free monoid: Let Σ be the set of all states of the puzzle and define transformations $\underline{a}_1, \ldots, \underline{a}_r \in \mathcal{T}(\Sigma)$ for the $r = 4$ elementary moves [shift the gap Up, Left, Down, Right]. For example, $\underline{a}_1$ maps the state s into the state obtained by shifting the gap upward or into s again if the gap is already in the top row. (Note that $\underline{a}_1$ is neither injective nor surjective and hence it is no permutation.) The transformations define a representation $\underline{(\cdot)}$ of the free monoid $F = \langle a_1, \ldots, a_r \rangle$ into $\mathcal{T}(\Sigma)$ by mapping $a_i \mapsto \underline{a}_i$. Solving the puzzle at state s means finding a (short) word $w \in F$ such that $s \cdot \underline{w}$ is the solution state.

The problem with this modelling of the 15-Puzzle is that the number of states is $n = |\Sigma| = 16 \cdot 15!/2 \approx 10^{13}$. Obviously, the transformations $\underline{a}_i$ cannot be specified by their lists of image points. For this reason it is necessary to define the representation $\underline{(\cdot)}$ in a more structured way:

Model Σ as $\{1..f\} \times \mathsf{S}_p$, where $f = 16$ is the "number of shapes" and $p = 15$ is the "number of pieces". A state where the gap is at position $\varphi \in \{1..f\}$ and the positions of the 15 pieces are given by $\sigma \in \mathsf{S}_p$ is represented by $(\varphi, \sigma) \in \Sigma$. In other words, we assume that the puzzle consists of a set of "pieces" that are rearranged and the pieces are assembled into different "shapes". Moreover, we assume that the action of the elementary moves on the set of states is compatible with the permutation structure. This means that there are transformations $F_i \in \mathcal{T}(\{1..f\})$ and permutations $P_{i,\varphi} \in \mathsf{S}_p$ such that

$$(\varphi, \sigma) \cdot \underline{a}_i = (\varphi \cdot F_i, \sigma \cdot P_{i,\varphi}) \text{ for all } i, \varphi, \sigma.$$

This is a concise formal description of the 15-Puzzle since the $F_i, P_{i,\varphi}$ can be specified easily by their lists of image points.

The solution algorithm for the 15-Puzzle now works as follows: For every position of the gap φ find a word $t_\varphi \in F$ moving the gap into its final position (shape 16, say). Then find generators for the submonoid $F_{16} \leq F$ of all words stabilizing the gap in its final position. In fact, $\underline{w}$ for $w \in F_{16}$ is bijective on the set $\{16\} \times \mathsf{S}_p$ by the special form of the operation. Namely, the image of $(16, \sigma)$ under $\underline{w}$ is $(16, \sigma\Phi(w))$ for some permutation $\Phi(w) \in \mathsf{S}_p$. Hence, Φ is a permutation group representation of F_{16} on the points $\{1..p\}$. It is a low-degree representation of the Schützenberger group (see [11]) of the $\mathcal{D}$-class of minimal rank. The bottom line is: Solving a puzzle defined by a monoid representation of the form stated above has been reduced to a permutation group element factorization problem in a low-degree representation of the Schützenberger group.

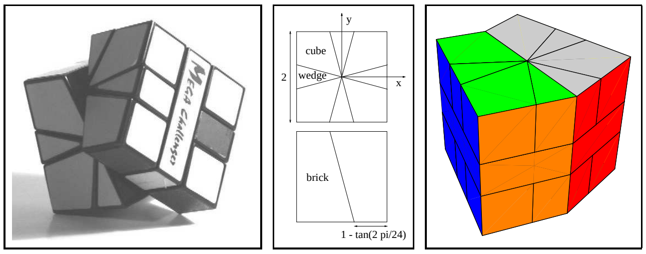


Figure 4: a) "MegaChallenger", b) model, c) twisted view.

Other interesting puzzles with the same type of structure are: The Organ, A globe of six caps containing numbered pieces to be sorted, TripleCross (sold by Binary Arts), The MegaChallenger (explained below), A barrel with balls inside (by Nintendo, 1980), Rubik's Fifteen (a machine with blocking sliders) and BackSpin (two twistable discs containing 35 balls and a hole). A non-hardware puzzle recently proposed by J. H. Conway is M_{13}, [5]. (A Java-realization by S. Egner can be found in [8].) The puzzle is motivated by pure beauty and is based on the projective geometry PG(2, 3). Its state space consists of 13 of the 65520 cosets of the Mathieu group M_{12} in S_{13}.

7.5 Combinatorial Modelling

So far it has been assumed that the generators of a permutation group can be read off from the puzzle in a straightforward way. This need not be the case! For example, in the following puzzle it is already a problem to identify the possible moves in any state.

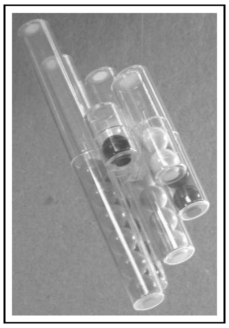
Example The "Mega Challenger" is a machine inspired by the Rubik's Cube. In contrast to the Cube, the moveable parts of the Challenger are non-rectangular: There are eight 30°-wedges, eight extruded cubes and two flat bricks with non-parallel long sides. Refer to Figure 4a) for a photograph of the puzzle and to Figure 4b) for the top view of the upper and middle plane of the MegaChallenger.

In order to find out which rotations are possible in a given state we have implemented a three dimensional geometric model which allows us to find the possible rotation planes and angles automatically. The model is implemented in the programming languages GAP and Mathematica. A perspective rendering of the model is shown in Figure 4c). The modelling is done as for the 15-Puzzle as a union of different shapes and permutation groups stabilizing a given shape. ■

7.6 Microreversibility

A subtle and important property of all puzzles mentioned so far is the inherent reversibility of all elementary moves. Let us call the property that any elementary move can be undone by another single elementary move *microreversibility*. For example, in the 15-Puzzle every shift can be undone by shifting the opposite direction. In contrast, an elementary move of the following puzzle cannot be undone by another elementary move, hence the inverse of any generator cannot be regarded as a word of length 1.

Example Consider the puzzle shown in Figure 5. We call it the "Organ". The Organ is mirror-symmetric consisting of two halves each consisting of six transparent pipes of increasing length arranged in a circle. Within the pipes there are 21 coloured balls: 1 black (bk), 2 white (wt), 3 blue (bl), 4 yellow (ye), 5 green (gn) and 6 red (rd). The two



bk	wt	bl	ye	gn	rd
1	2	4	7	11	16
	3	5	8	12	17
		6	9	13	18
			10	14	19
				15	20
					21

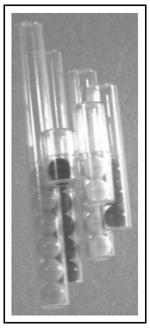


Figure 5: The "Organ", two views and the numbering.

halves are linked in the central vertical axis by a joint which allows the halves to be twisted agains each other. Turning the Organ upside-down after twisting makes the balls roll down the pipes but there may be a congestion since the opposing pipe may be shorter. The balls jutting out can then be redistributed into the lower pipes by twisting the halves again. *Once the balls have been distributed there is no simple way to make them return!* The aim of the puzzle is to sort the balls into their pipes.

To model the Organ as a permutation group we fix a reference position defined as follows (refer to Figure 5): Hold the Organ vertically with all balls in the lower pipes such that the lengths of the upper and lower pipes are equal. (Note that this definition does not specify which of the two halves is the upper one.) In this situation we number the positions within the lower pipes as shown in Figure 5. We now define a move for every colour c by the following three steps: 1. Twist the halves such that the upper c-pipe is over the lower bk-pipe. 2. Turn the Organ upside-down. 3. Twist the (now) upper half in the direction of decreasing pipe lengths until all balls have arrived in the lower half.

Since the moves restore the reference position their action can be described by $r = 6$ permutations on the $n = 21$ positions. E.g. the wt-move is $\underline{a_2} = (1,2,4,7,11,16)(3,5,8,12,17,21)(6,9,13,18,15,20)(10,14,\overline{19})$. It turns out that the group G is the full symmetric group on 21 points and thus it is not necessary to consider any other types of moves. The subgroup H of invisible rearrangements of the balls is the Young-group $H = \mathsf{S}_1 \times \mathsf{S}_2 \times \cdots \times \mathsf{S}_6$ of the partition $(1|2\,3|4\,5\,6|\ldots|16\,17\,18\,19\,20\,21)$ corresponding to the ball colours in the solution state. Hence, the Organ has $|G|/|H| \approx 2.1 \cdot 10^{12}$ observable states. ∎

As explained in the example we have modelled the Organ by a finite permutation group. Hence, every elementary move has an inverse (but it may require a long word of elementary moves). For this reason, the Organ is still *macroreversible*. Note however that macroreversibility is not an issue in the finite puzzles we study: The only way to achieve non-macroreversibility is to introduce "catch-states" into the automaton of the puzzle, and this is not fair. (Exceptional example: The Tamagochi has a catch-state — eventually it is dead.)

8 Acknowledgements

We would like to thank our advisor Thomas Beth and we would like to thank Torsten Minkwitz who has laid the foundations for our work. Most the "puzzle hardware" has been provided by Peer Wichman. Harald Ritter and Susanne Wetzel have provided us with profound information on lattice basis reduction with respect to non-ℓ_2-norms.

References

[1] Babai, L., Luks, E. M., and Seress, A. Fast Management of Permutation Groups. *SIAM Journal of Computing 26* (1997), 1310–1342.

[2] Babai, L., and Seress, A. On the Diameter of Permutation Groups. *Eur. J. Comb. 4* (1992), 231–243.

[3] Bradly, L. D. *Java-applet implementing Rubik's Cube.* `http://www.pha.jhu.edu/~ldb/java/rubik.html`.

[4] Butler, G. *Fundamental Algorithms for Permutation Groups.* Lecture Notes in Computer Science, 559. Springer, 1991.

[5] Conway, J. H. *M_{13}. in Bailey, R. A. (ed.), Surveys in Combinatorics, London Math. Soc. Lect. Note Series 241* (1997).

[6] Cooperman, G., and Finkelstein, L. New Methods for Using Cayley Graphs in Interconnection Networks. *Discrete Applied Mathematics 37/38, special issue on Interconnection Networks* (1992), 95–118.

[7] Cooperman, G., Finkelstein, L., and Sarawagi, N. Applications of Cayley Graphs. *Lecture Notes in Computer Science, AAECC-8 508* (1991), 367–378.

[8] Egner, S. *Java-applet for M_{13}.* `avalon.ira.uka.de/home/egner/m13/m13.html`.

[9] Huppert, B. *Endliche Gruppen*, vol. I. Springer, 1983.

[10] Kaib, M. *Gitterbasenreduktion für beliebige Normen.* PhD thesis, Mathematics Dept., Univ. Frankfurt (Main), 1994.

[11] Lallement, G. *Semigroups and Combinatorial Applications.* J. Wiley & Sons, 1979.

[12] Matz, O., Miller, A., Potthoff, A., Thomas, W., and E., V. Report on the Program AMoRE. Tech. rep., Institut für Informatik und Praktische Mathematik, Universität Kiel, 1995.

[13] Minkwitz, T. An Algorithm for Solving the Factorization Problem in Permutation Groups. *Journal of Symbolic Computation to appear* (1998).

[14] Mortazavi, M. *Loyds's Fifteen.* `math.berkeley.edu/~mortazav/java_test/sq/Sq2.html`.

[15] Motwani, R., and Raghavan, P. *Randomized Algorithms.* Cambridge Univ. Press, 1995.

[16] Osterlund, P. *The shared* GAP*-package* `AbStab`. `http://www-gap.dcs.st-and.ac.uk/~gap/`, 1995.

[17] Ritter, H. *Aufzählung von kurzen Gittervektoren in allgemeiner Norm.* PhD thesis, Mathematics Dept., Univ. Frankfurt (Main), 1997.

[18] Schönert, M., et al. GAP — Groups, Algorithms and Programming, v3.4.3. Tech. rep., Lehrstuhl D für Mathematik, RWTH Aachen, 1995.

[19] The Cube Lover's Archive. `http://www.math.rwth-aachen.de/~Martin.Schoenert/Cube-Lovers/`. Collected E-mails, 1980–1996.