T.C. Brouwer

Solving an arbitrary permutation puzzle

Bachelor thesis, June 18, 2016

Supervisor: Dr. R.M. van Luijk

Mathematisch Instituut, Universiteit Leiden

Contents

1 Introduction

This Bachelor thesis is inspired by Rubik's cube, a famous permutation puzzle on the solving of which some mathematical research has been conducted, for example see [4].

The initial set-up was to consider a four-dimensional Rubik's cube and writing a general solution for it. However in the process, we have shifted to a more general problem setting. As we make more precise in the next section, in this paper we suggest an algorithm for writing a general solution for an arbitrary permutation puzzle. Our result is summarized in section 12.

2 Mathematical formulation

Notation. Throughout this paper $\mathrm{Sym}(n)$ will denote the permutation group on n elements.

Definition 2.1. Let T be a subset of $\mathrm{Sym}(n)$. We define the **inverse closure** of T as $\hat{T} = T \bigcup \{t^{-1} : t \in T\}$. A subset T of $\mathrm{Sym}(n)$ is called *inversely closed* if $T = \hat{T}$.

Definition 2.2. Let T be a subset of $\mathrm{Sym}(n)$. The **word group** of T is the free group on $\hat{T}$.

Any element t in the word group of T can be represented by a sequence $t = (t_1, q_1, t_2, q_2, \ldots, t_k, q_k)$ with t_i in $\hat{T}$ and q_i in $\{-1, 1\}$. Let s be in $\mathrm{Sym}(n)$. An element of the word group of T is called a **word** in T. For a word $t = (t_1, q_1, t_2, q_2, \ldots, t_k, q_k)$ in the word group of T with $\prod_{i=1}^{k} t_i^{q_i} = s$, we say s can be written as $t_1^{q_1} t_2^{q_2} \cdots t_k^{q_k}$ and that t writes s. The $t_i^{q_i}$ are called letters. If $t = (t_1, q_1, \ldots, t_m, q_m)$ and $u = (u_1, p_1, \ldots, u_n, p_n)$ are two words in T, then tu is the word

$$(t_1, q_1, \ldots, t_m, q_m, u_1, p_1, \ldots, u_n, p_n)$$

In a permutation puzzle we are given a subset T of $\mathrm{Sym}(n)$ and an $s \in \langle T \rangle$. The goal of the permutation puzzle is find a word for s in T that is as short as possible. In this paper we will do the following:

- In sections 3 through 5 we will examine what the minimum possible length for writing an arbitrary $s \in \langle T \rangle$ as a word in T for arbitrary subsets T of $\mathrm{Sym}(n)$ is.
- In sections 6 through 11 we will construct an algorithm that, for an arbitrary subset T of $\mathrm{Sym}(n)$, enables us to write an arbitrary $s \in \langle T \rangle$ as a word in T.

3 Lower limit in n

In this section we will show that writing an arbitrary $s \in \langle T \rangle$ as a word in T for an arbitrary subset T of $\mathrm{Sym}(n)$, cannot be done in polynomial length in n.

Theorem 3.1. *There exists a sequence* $(T_1, T_2, \ldots, T_n, \ldots)$ *where, for each* n, T_n *is a subset of* $\mathrm{Sym}(n)$*, and a sequence of elements* $(s_1, s_2, \ldots, s_n, \ldots)$*, where each* s_n *is in* $\langle T_n \rangle$*, such that* s_n *can not be written as a word in* T_n *of length bounded by a polynomial in* n*.*

Proof. Let $p_1 < p_2 < p_3 < \ldots$ be the primes. Let i_n be the largest natural number such that $\sum_{j=1}^{i_n} p_j \leq n$.

Let q_n in $\mathrm{Sym}(n)$ be the product of disjoint cycles of length $p_1, p_2, \ldots, p_{i_n}$. We take $T_n = \{q_n\}$. We choose $s_n = (q_n)^{\frac{|\langle T_n \rangle|}{2}}$.

To write s_n as word in T_n we need at least $\frac{|\langle T_n \rangle|}{2}$ letters from T_n. The order of q_n is $|\langle T_n \rangle|$, which is equal to $\prod_{j=1}^{i_n} p_j$. For all i we have $p_i \geq 2$, therefore we have that the order of q_n is larger than 2^{i_n}.

We will now prove that p_{i_n} is at least the largest prime p such that $p \leq \sqrt{n}$. Suppose p_{i_n} is smaller than $p \leq \sqrt{n}$. Then i_n is not maximal, because $p_{i_n+1} \leq \sqrt{n}$ and therefore also $i_n + 1 \leq \sqrt{n}$, thus $\sum_{j=1}^{i_n+1} p_j \leq \sum_{j=1}^{\lfloor \sqrt{n} \rfloor} \sqrt{n} \leq n$. We conclude that $p_{i_n} \geq p$ and thus that i_n is larger than or equal to the number of primes smaller than or equal to $\sqrt{n}$.

By the prime number theorem we know that for n large enough $i_n \geq \frac{1}{2} \frac{\sqrt{n}}{\log(\sqrt{n})}$, see page 9 of [1]. Therefore $|\langle T_n \rangle| \geq 2^{\frac{1}{2} \frac{\sqrt{n}}{\log(\sqrt{n})}}$, and we need at least $2^{\frac{1}{2} \frac{\sqrt{n}}{\log(\sqrt{n})}}$ letters from T_n to write s_n, which is not bounded by a polynomial in n. □

4 Upper limit in group order

In this section we will prove an upper limit in terms of the group order. We will use the concept of the Cayley-graph of a group as described in paragraph 3.1 of [2].

Definition 4.1. The **Cayley-graph** $W_T = (\langle T \rangle, E)$ of a subset T of $\mathrm{Sym}(n)$ is an undirected graph with the set of vertices equal to the set of elements of $\langle T \rangle$. Two vertices g_1 and g_2 are connected if and only if there exists an element t in $\hat{T}$ such that $g_1 t = g_2$.

The following lemma is deduced from theory on the Cayley-graph, and will help us to put an upper limit on the word length in terms of the group order.

Lemma 4.1. *Given an inversely closed subset* T *of* $\mathrm{Sym}(n)$*, any element* s *of* $\langle T \rangle$ *can be written as a word in* T *of length at most* $\frac{\#\langle T \rangle}{2}$*.*

Proof. Without loss of generality we may assume $1 \notin T$. If T is empty then the empty word suffices to show the statement is true.

If $\#T$ is 1 then the element in T is has to be of order 2 since T is inversely closed. Now the only non-trivial element in $\langle T \rangle$ can be written as a word of one letter and the statement is true.

Now assume $\#T \geq 2$. Then the degree of the Cayley-graph is at least 2. Let d denote the degree of the Cayley-graph. From Cayley-graph theory we know

the vertex-connectivity of the Cayley-graph is at least $\frac{2(d+1)}{3}$, see Theorem 3.7 of [2], so in this case at least 2. Let W_T^i denote the set of vertices of W_T that are connected to the vertex corresponding to the identity by a path of length at most i. Equivalently, W_T^i contains those elements of $\langle T \rangle$ which can be written as a word in T of length at most i. We will prove by induction that $\#W_T^i \geq 2i+1$ for $i < \frac{\#\langle T \rangle}{2}$.

We have $\#W_T^0 = 1$. Now for a natural number i with $i+1 < \frac{\langle T \rangle}{2}$ assume $\#W_T^i \geq 2i+1$. If $W_T^i = \langle T \rangle$ there is nothing to prove. If $\#W_T^i = \#\langle T \rangle - 1$ the missing vertex is in $W_T^{i+1} = \langle T \rangle$ and the statement holds. So assume there are at least two vertices outside of W_T^i. Since W_T^i is connected to at least one of the vertices, say v_1, this v_1 in W_T^{i+1}. Because the vertex-connectivity of W_T is at least 2, also W_T with v_1 removed is connected, so there is another vertex v_2 outside W_T^i in W_T^{i+1} and hence $\#W_T^{i+1} \geq \#W_T^i + 2 \geq 2(i+1)+1$.

We now discern two cases.

- $\langle T \rangle$ is divisible by 2. For $j = \frac{\langle T \rangle}{2} - 1$ it holds that $\#W_T^j \geq \#\langle T \rangle - 1$. So there is at most one element in $\langle T \rangle$ not in W_T^j. But if this element exists, because the Cayley-graph is connected, it can be written using one more letter. So we see that any element of $\langle T \rangle$ can be written as a word in T of length at most $\frac{\langle T \rangle}{2}$, so the statement holds.

- $\langle T \rangle$ is not divisible by 2. Now let k be the integer $\frac{\langle T \rangle - 1}{2}$. It holds that $\#W_T^k = \langle T \rangle$. So the statement holds.

This concludes the proof. □

Concluding this section we will show this is the lowest upper limit possible in terms of group order.

Theorem 4.2. *Let T be a subset of $\mathrm{Sym}(n)$. Any element of $\langle T \rangle$ can be written as a word in T of length at most $\frac{|\langle T \rangle|}{2}$. Also for every n there exists a subset T_n of $\mathrm{Sym}(n)$ with $|\langle T_n \rangle| \geq n$ and an element s in $\langle T_n \rangle$ such that s can not be written as a word in T_n of length less than $\frac{|\langle T_n \rangle|}{2} - 1$.*

Proof. The first statement is exactly Lemma 4.1. To prove the second statement take $s = (1 \cdots n) \in \mathrm{Sym}(n)$ and $T_n = \{s\}$. We have that $|\langle T_n \rangle| = n$, and writing the element $s^{\lfloor \frac{n}{2} \rfloor}$ takes at least $\frac{n}{2} - 1$ letters. □

5 An example for $\mathrm{Sym}(n)$

In this section we will give an example for writing elements of $\mathrm{Sym}(n)$ as words in $T = \{(12), (1 \cdots n)\}$.

We need the following definition.

Definition 5.1. Given a permutation $s \in \mathrm{Sym}(n)$, for each $m \in \{1, \ldots, n\}$ we define the distance to the original position of m to be

$$\min(|s(m) - m|, n - |s(m) - m|).$$

The **total disposition** of s, denoted by $d(s)$, is the sum of the distances to original position, i.e.

$$d(s) = \sum_{m=1}^{n} \min(|s(m) - m|, n - |s(m) - m|).$$

Let t be the permutation $(1 \cdots n)$. The **cyclic disposition** of s is $\min_{j \in \mathbf{Z}}(d(t^j s))$.

In the next proof we use the following fact. If x, t and n are natural numbers with $0 \leq x < n$ and $0 \leq t < n$, then it holds that

$$\min(|x - t|, n - |x - t|) = \min(x - t \mod n, n - (x - t) \mod n)$$

where $a \mod n$ is the representative in $\{0, \ldots, n-1\}$ of the residue class of a in $\mathbf{Z}/n\mathbf{Z}$.

Lemma 5.1. *Given a permutation $s \in \mathrm{Sym}(n)$ the cyclic disposition of s is at most $\frac{n^2}{4}$. Also for every n with n odd, there exists an $s \in \mathrm{Sym}(n)$ such that the cyclic disposition of s is at least $\frac{n^2-1}{4}$.*

Proof. We write

$$\begin{aligned}
\sum_{j=1}^{n} d(t^j s) &= \sum_{j=1}^{n} \sum_{m=1}^{n} \min(|t^j s(m) - m|, n - |t^j s(m) - m|) \\
&= \sum_{m=1}^{n} \sum_{j=1}^{n} \min(|t^j s(m) - m|, n - |t^j s(m) - m|) \\
&= \sum_{m=1}^{n} \sum_{i=1}^{n} \min(i, n - i)
\end{aligned}$$

This last sum is $\frac{n^3-n}{4}$ if n is odd, and $\frac{n^3}{4}$ if n is even. It follows that there exists a j such that $d(t^j s)$ is at most $\frac{n^2}{4}$.

Let n be odd and consider the function $f : \mathbf{Z}/n\mathbf{Z} \longrightarrow \mathbf{Z}/n\mathbf{Z}$ defined by $f(m) = 2m - 1 \mod n$. Since n is odd, multiplication by 2 is a bijection and so is the composition with the map $x \mapsto x - 1$.

Because f is bijective it defines a permutation s_f in $\mathrm{Sym}(n)$. The permuta-

tion s_f maps m to $f(m)$. We now calculate the total disposition of s_f.

$$\begin{aligned} d(s_f) &= \sum_{m=1}^{n} \min(|s_f(m) - m|, n - |s_f(m) - m|) \\ &= \sum_{m=1}^{n} \min(|f(m) - m|, n - |f(m) - m|) \\ &= \sum_{m=1}^{n} \min(|((2m-1) \mod n) - m|, n - |((2m-1) \mod n) - m|) \\ &= \sum_{m=1}^{n} \min(m-1, n-(m-1)) \\ &= \frac{n^2-1}{4} \end{aligned}$$

We will now show that the cyclic disposition of s_f is equal to the total disposition of s_f. Let j be in $\mathbf{Z}$. Let t be the permutation $(1 \cdots n)$. The permutation $t^j s_f$ is characterized by the map f_j which maps m to $f_j(m) = f(m) + j \mod n = 2m - 1 + j \mod n$. We now calculate the total disposition of $t^j s$.

$$\begin{aligned} d(t^j s) &= \sum_{m=1}^{n} \min(|s_{f_j}(m) - m|, n - |s_{f_j}(m) - m|) \\ &= \sum_{m=1}^{n} \min((|(2m-1+j) \mod n) - m|, n - |((2m-1+j) \mod n) - m|) \\ &= \sum_{m=1}^{n} \min(|m-1+j| \mod n, n - (|m-1+j| \mod n)) \\ &= \sum_{m=1}^{n} \min(m, n-m) \\ &= \frac{n^2-1}{4} \end{aligned}$$

Thus we see that for every n with n odd, there exists an $s \in \mathrm{Sym}(n)$ such that the cyclic disposition of s is at least $\frac{n^2-1}{4}$. □

Definition 5.2. For a subset T of $\mathrm{Sym}(n)$ the minimum word length function γ_T is the function on $\langle T \rangle$ such that for an s in $\langle T \rangle$ we have that $\gamma_T(s)$ is the minimum length of a word in T that writes s.

Definition 5.3. Let f and g be two functions from a domain D to the natural numbers. The function f is said to be $\Omega(g)$ if for some constant c it holds that $c \cdot f(d) > g(d)$ for all $d \in D$.

Theorem 5.2. *Given the sequence $(T_2, T_3 \ldots, T_n, \ldots)$ with $T_n = \{(12), (1 \ldots n)\} \subset \mathrm{Sym}(n)$, there exists a sequence $(s_2, s_3, \ldots, s_n, \ldots)$ with $s_n \in \mathrm{Sym}(n)$ such that*

the function

$$\Gamma : \{3, 4, \ldots\} \to \boldsymbol{N}$$

$$n \mapsto \gamma_{T_n}(s_n)$$

is $\Omega(n^2)$.

Proof. By lemma 5.1 for every n with n odd, there exists an $s_n \in \mathrm{Sym}(n)$ such that the cyclic disposition of s_n is at least $\frac{n^2-1}{4}$. We consider the sequence $(s_3, \ldots, s_{2n+1}, \ldots)$. Let n be in $\mathbf{N}$. Let $(t_1, \ldots, t_k)$ be a word for s_n in T_n. Define $d_i = \prod_{j=1}^{i} t_i$. Consider the sequence $(d_1, d_2, \ldots, d_k)$. Let a_i be the cyclic disposition of d_i. Since $d_k = s_n$ we have that $a_k \geq \frac{n^2-1}{4}$. If $t_{i+1} = (1 \cdots n)$ or its inverse, then $a_{i+1} = a_i$. If $t_{i+1} = (12)$ then, since only two elements are moved and their distance to their original position increases only by one at most, $a_{i+1} \leq a_i + 2$. If both t_i and t_{i+1} are equal to (12) then $a_{i+2} = a_i$. Therefore $a_{i+2} \leq a_i + 2$. Since $a_0 = 0$ and $a_1 \leq 2$ we have that a_k at most $k+1$, and thus k is at least $\frac{n^2-1}{4} - 1$.

For $n \geq 4$ with n even we have that $s_{n-1} \in \mathrm{Sym}(n)$ and since $n-1$ is odd by the above a word for s_{n-1} has length at least $\frac{(n-1)^2}{4} - 1 = \frac{n^2-2n}{4} - \frac{3}{4}$. Thus Γ is $\Omega(n^2)$. □

Theorem 5.3. *Let $n > 1$ be in $\mathbf{N}$ and let T be the set $\{(12), (1 \ldots n)\} \subset \mathrm{Sym}(n)$. Now every $s \in \mathrm{Sym}(n)$ can be written as a word in T using less than $\frac{3}{2}n^2$ letters.*

Before giving the proof of this theorem, we will give a sketch of an algorithm which writes $s \in \mathrm{Sym}(n)$ as a word in T using the stated number of letters. The algorithm starts with s and then sorts 2,3,...,n relative to 1. At each stage the algorithm outputs the permutations used to sort, and after the sorting has finished, the inverses of the elements in the output, in reverse order, are a word for s in T.

Essential to the algorithm is that it sorts relative to 1. So starting with 2 the algorithm outputs letters that will move 2 next to 1. Continuing the algorithm outputs letters that will move 3 next to 2 etcetera. When n is moved next to $n-1$ all numbers are sorted relative to 1. Depending on the position of 1 a sequence consisting only of the letter $(1 \ldots n)$ will move all numbers to their original position and the algorithm finishes.

We will now show how a single number i is sorted relative to 1. By definition of the algorithm we assume that all numbers smaller than i are already sorted relative to 1. First the algorithm outputs the letter $(1 \ldots n)$ until i is at the orginal position of 2. Now the algorithm outputs alternatingly the letters (12) and $(1 \ldots n)$ until i is sorted relative to 1. The letters smaller than i all have not been in the original position of 1 and 2 when a letter (12) was applied, so all numbers smaller than i are still sorted relative to 1, so now all numbers smaller then $i+1$ are sorted relative to 1.

The complexity of the algorithm is given in the proof of Theorem 5.3 below.

Proof. The algorithm consists of two parts we will analyse separately.

1. Sorting all numbers 2, ..., n relative to 1.
2. Returning 1, and thus all numbers, to their original position.

The two parts are analysed below.

1. For sorting a number i with $1 \leq i < n$ the algorithm does the following. By outputting up to $\frac{n}{2}$ times either $(1 \dots n)$ or $(n \dots 1)$ the number i is moved to the original position of 2. The algorithm now outputs up to $n - i$ times the combination of the letters (12) and $(1 \dots n)$. For sorting i the algorithms thus outputs at most $\frac{n}{2} + 2(n - i)$ letters. For sorting all numbers up to n relative to 1 the algorithm outputs at most

$$\Sigma_{i=2}^{n}(\frac{n}{2} + 2n - 2i) \leq \frac{5n^2}{2} - n^2 - n = \frac{3n^2}{2} - n$$

letters.

2. For returning 1 and thus all numbers to their original position the algorithm outputs up to $\frac{n}{2}$ times either $(1 \dots n)$ or $(n \dots 1)$.

Combing the two parts we see that the algorithm outputs less than $\frac{3n^2}{2}$ letters. □

Remark 5.1. The algorithm as described above is not optimal.

6 Basic concepts of Schreier-Sims

The next two sections are based on the Schreier-Sims algorithm as described in paragraphs 1 and 2 in chapter 4 of [3]. Some of the proofs given here come from the excellent book of Seress, and only minor adaptations have been made. For example, we allow an element of the base to be any subset of $\{1, \dots, n\}$, whereas Seress allows only singletons. Further optimization of the Schreier-Sims algorithm is described in chapter 4 paragraphs 3 and onwards of the book.

In this section we will introduce the basic concepts of the Schreier-Sims algorithm. Later on we will use the Schreier-Sims algorithm in an **extended Schreier-Sims algorithm**. This last algorithm will create a data structure that enables us to use an **extended membership testing** algorithm. This extended membership testing algorithm will decide whether a given permutation can be written as a word in a given subset T of $\mathrm{Sym}(n)$, and if so will output a word in T for the permutation.

We will consider a group $G \subset \mathrm{Sym}(n)$.

Definition 6.1. A **base** for G is a sequence $B = (\beta_1, \cdots, \beta_m)$, with β_i a subset of $\{1, \dots, n\}$ such that the only element of G that fixes B is the identity.

Let $B = (\beta_1, \cdots, \beta_m)$ be a base. Let $G^{[i]} := G_{(\beta_1, \cdots, \beta_{i-1})}$ be the pointwise stabilizer of $(\beta_1, \cdots, \beta_{i-1})$. Now B defines a subgroup chain:

$$G = G^{[1]} \geq G^{[2]} \geq \cdots \geq G^{[m]} \geq G^{[m+1]} = 1.$$

Definition 6.2. A base B is called **non-redundant** if all subgroups in the above defined subgroup chain are proper subgroups of their predecessor.

Definition 6.3. A **strong generating set** for G relative to a base $B = (\beta_1, \cdots, \beta_m)$ is a generating set S for G such that for $1 \geq i \geq m+1$ we have:

$$\langle S \cap G^{[i]} \rangle = G^{[i]}. \tag{1}$$

Definition 6.4. Let H be a subgroup of G. A (left) **transversal** R for G mod H is a subset of G containing exactly one element of each left coset of H. We also require that 1 is in R. For any g in G we denote the element in $gH \cap R$ as $\overline{g}$.

Definition 6.5. A **directed rooted labeled tree** is a triple (T, S, f); here T is a directed graph for which the underlying undirected graph contains no cycles and such that there exists a vertex from which each other vertex can be reached. It follows that this vertex is unique and this vertex is called the **root** of the tree. Also f is a map from the set of edges of T to S. We call S the label set of the directed rooted labeled tree.

Let $B = (\beta_1, \ldots, \beta_m)$ be a base for G. Let i be a natural number with $1 \leq i \leq m$ and let s be in $G_{(\beta_1,\ldots,\beta_{i-1})}$ and let v be a left coset of $G_{(\beta_1,\ldots,\beta_i)}$ in $G_{(\beta_1,\ldots,\beta_{i-1})}$. The natural action of $G_{(\beta_1,\ldots,\beta_{i-1})}$ on the left cosets of $G_{(\beta_1,\ldots,\beta_i)}$ is defined by $s(v) = sv$, where the multipliction is the multiplication induced by the group law of G.

Definition 6.6. Given a subset T of $\mathrm{Sym}(n)$, a **Schreier tree data structure** Δ for T consist of

- a base $B = (\beta_1, \ldots, \beta_m)$ for $\langle T \rangle$, and
- m directed rooted labeled trees (T_i, S_i, f_i) with $i \in \{1 \cdots m\}$,

such that

- S_i is a subset of $G_{(\beta_1,\ldots,\beta_{i-1})}$,
- each T_i is a directed tree with as vertices the left cosets of $G_{(\beta_1,\ldots,\beta_i)}$ in $G_{(\beta_1,\ldots,\beta_{i-1})}$.
- the vertex $G_{(\beta_1,\cdots,\beta_i)}$ is the root of the tree,
- every edge e in T_i pointing from a vertex γ to a vertex δ and with $f(e) = s$ has the property that $s(\gamma) = \delta$.

The directed rooted labeled trees in a Schreier tree data structure are called **Schreier trees**.

Theorem 6.1. *Let G be a subgroup of* $\mathrm{Sym}(n)$ *and let $B = (\beta_1, \beta_2, \ldots, \beta_m)$ be a base for G. Let i be a natural number with $1 \leq i \leq m$. Let C be the set of left cosets of $G_{(\beta_1,\ldots,\beta_i)}$ in $G_{(\beta_1,\ldots,\beta_{i-1})}$. Let A be the orbit of β_i under $G_{(\beta_1,\ldots,\beta_{i-1})}$. For each g in C and g_1 be in g we define the map f by $f(g) = g_1(\beta_i)$. Then f is a well-defined bijective map f from C to A.*

Proof. We will first show that f is well defined. Let g_2 also be in g. Then there exists an h_1 in $G_{(\beta_1,\ldots,\beta_i)}$ such that $g_1 h_1 = g_2$. We see that

$$g_2(\beta_i) = g_1 h_1(\beta_i) = g_1(h_1(\beta_i)) = g_1(\beta_i).$$

For every element $\alpha \in A$ in the orbit of β_i in $G_{(\beta_1,\ldots,\beta_{i-1})}$ there exists a g_3 in $G_{(\beta_1,\ldots,\beta_{i-1})}$ such that $g_3(\beta_i) = \alpha$. By definition f maps the coset of $G_{(\beta_1,\ldots,\beta_i)}$ in $G_{(\beta_1,\ldots,\beta_{i-1})}$ containing g_3 to α, and therefore f is surjective.

Let g' be in C such that $f(g') = f(g) = g_1(\beta_i)$. Let g'_1 be in g'. Then $g'_1(\beta_i) = g_1(\beta_i)$, so they are in the same left coset of $G_{(\beta_1,\ldots,\beta_i)}$ in $G_{(\beta_1,\ldots,\beta_{i-1})}$. Therefore $g' = g$. This shows f is injective.

We conclude f is bijective.

□

Remark 6.1. By the above theorem we will identify the cosets of $G_{(\beta_1,\ldots,\beta_i)}$ in $G_{(\beta_1,\ldots,\beta_{i-1})}$ with the elements in the orbit of β_i under $G_{(\beta_1,\ldots,\beta_{i-1})}$.

Lemma 6.2. *Let T be a subset of* $\mathrm{Sym}(n)$ *and Δ be a Schreier tree data structure for T. Let $|T_i|$ denote the number of vertices in T_i. Then $\prod_{i=1}^{m} |T_i| = |\langle T \rangle|$.*

Proof. The number of vertices in T_i is equal to the number of cosets of $G_{(\beta_1,\ldots,\beta_i)}$ in $G_{(\beta_1,\ldots,\beta_{i-1})}$. This is equal to the index of $G_{(\beta_1,\ldots,\beta_i)}$ in $G_{(\beta_1,\ldots,\beta_{i-1})}$. So we have iteratively

$$|\langle T \rangle| = |T_1| \cdot |G_{(\beta_1)}| = |T_1| \cdot |T_2| \cdot |G_{(\beta_1,\beta_2)}| = \ldots = \prod_{i=1}^{m} |T_i|.$$

□

Definition 6.7. For a Schreier tree data structure Δ with base $B = (\beta_1, \ldots, \beta_m)$, the **cardinality** of Δ is the largest cardinality of the β_i in B.

Let $B = (\beta_1, \beta_2, \ldots, \beta_m)$ be a base for G. If we have a strong generating set S for G relative to B, we can compute the orbits $G^{[i]}\beta_i$ and the transversals for $G^{[i]}$ mod $G^{[i+1]}$ as follows. We will store the data in a Schreier tree data structure. Let S_i denote $S \cap G^{[i]}$. For i with $1 \leq i \leq m$, the computation is as follows:

1. We start with T_i empty and add a vertex β_i.
2. For each newly added vertex t in T_i and for every s in S_i we compute $s(t)$, and if $s(t)$ is not in T_i we add it to T_i and we add an edge from t to $s(t)$ labeled s.

3. If no new vertices were added, we stop. Otherwise we repeat step 2.

Note that each vertex γ corresponds to the left coset of $G^{[i+1]}$ in $G^{[i]}$ consisting of elements of $G^{[i]}$ that move β_i to γ. Now let γ be such a vertex in T_i. There is a unique path from β_i to γ. If $(s_1, s_2, \ldots, s_t)$ are the labels along this path starting from β_i, then $s_t s_{t-1} \cdots s_1$ is an element of $G^{[i]}$, that by definition of the Schreier tree moves β_i to γ. We construct the transversal R_i for $G^{[i]}$ mod $G^{[i+1]}$ by taking this element as the unique element in R_i that is contained in the left coset corresponding to γ. By doing this for all vertices, we obtain a complete transversal R_i.

Algorithm 6.1. Using the Schreier tree data structure Δ as created above we can write every g in G as $g = r_1 r_2 \cdots r_m$ with r_i in R_i. This procedure is called **sifting through** Δ and is done as follows:

1. Set $i = 1$ and $g_1 = g$.

2. There is a unique path from β_i to $g_i(\beta_i)$. If $(s_1, s_2, \ldots, s_t)$ are the labels along this path starting from β_i, then we set $r_i = s_t s_{t-1} \cdots s_1$.

3. If i is m we stop. Otherwise we set $g_{i+1} = r_i^{-1} g_i$. Note that g_{i+1} is in $G_{(\beta_1,\ldots,\beta_i)}$. Add one to i and repeat step 2.

Now $r_m^{-1} g_m$ is in $G_{(\beta_1,\ldots,\beta_m)} = \{1\}$, and therefore $r_1 r_2 \cdots r_m = g$. To illustrate why this algorithm works, consider the following. The element r_1 moves β_1 to $g(\beta_1)$ and the element r_2 moves β_2 to $g_2(\beta_2) = r_1^{-1} g(\beta_2)$ without moving β_1. Hence the element $r_1 r_2$ moves β_1 and β_2 to $g(\beta_1)$ and $g(\beta_2)$ respectively.

Remark 6.2. During the sifting process we construct an r_i for $1 \leq i \leq m$. We construct r_i by multiplying labels of edges in the Schreier trees of a Schreier tree data structure.

Remark 6.3. We can also use sifting to test g in Sym(n) for membership of G. In this case we apply the above procedure. The element g is not in G if and only if either for some i we have $g_i(\beta_i)$ is not a vertex in T_i, or $g^{-1} r_1 r_2 \cdots r_m$ is not the identity.

Definition 6.8. For g in Sym(n) the g_i that we can calculate and that has the highest index i among the indices we can calculate is called the **siftee** of g.

For the Schreier-Sims Algorithm, we will need the following two lemmas.

Lemma 6.3. *Let $H \leq G = \langle S \rangle$ and let R be a left transversal for G* mod *H, with* 1 *in R. Now the set*

$$T = \{(\overline{sr})^{-1} sr | s \in S, r \in R\}$$

generates H. The elements of T are called the Schreier generators *of H.*

Proof. By definition the elements of T are in H, so it is enough to show that the set $T \cup T^{-1}$ generates H. Note that $T^{-1} = \{(\overline{sr})^{-1} sr | s \in S^{-1}, r \in R\}$. Let $h \in H$ be arbitrary. Since $H \leq G$, h can be written in the form $h = s_k s_{k-1} \cdots s_1$ with $s_i \in S \cup S^{-1}$. We define a sequence $h_0, h_1, \ldots, h_k$ of group elements such that

$$h_j = s_k \cdots s_{j+2} s_{j+1} r_{j+1} t_j \cdots t_2 t_1$$

with $t_i \in T \cup T^{-1}$, $r_{j+1} \in R$ and $h_j = h$. We set h_0 to be $s_k \cdots s_2 s_1 r_1$ with $r_1 = 1$. Recursively, if h_j is already defined then let t_{j+1} be $(\overline{s_{j+1} r_{j+1}})^{-1} s_{j+1} r_{j+1}$ and set r_{j+2} to be $\overline{s_{j+1} r_{j+1}}$. Clearly $h_{j+1} = h_j = h$, and it has the required form.

We have $h = h_k = r_{k+1} t_k \cdots t_2 t_1$. Since $h \in H$ and $t_k \cdots t_2 t_1 \in \langle T \rangle \leq H$, we must have $r_{k+1} \in H \cap R = \{1\}$. Hence $h \in \langle T \rangle$. □

Lemma 6.4. *Let $(\beta_1, \ldots, \beta_k)$ be a sequence of subsets of $\{1, \ldots, n\}$, and G a subgroup of* Sym(n)*. For $1 \leq j \leq k+1$ let S_j be a subset of the stabilizer $G_{(\beta_1, \ldots, \beta_{j-1})}$ such that we have $\langle S_j \rangle \geq \langle S_{j+1} \rangle$ for all $j \leq k$. If $G = \langle S_1 \rangle$ and $S_{k+1} = \emptyset$ and*

$$\langle S_j \rangle_{\beta_j} \quad = \quad \langle S_{j+1} \rangle \tag{2}$$

for all $1 \leq j \leq k$, then $B = (\beta_1, \ldots, \beta_k)$ is a base for G and $S = \bigcup_{1 \leq j \leq k} S_j$ is a strong generating set for G relative to B.

Proof. We use induction on k. Our inductive hypothesis is that $S^* = \cup_{2 \leq j \leq k} S_j$ is a strong generating set for $\langle S_2 \rangle$, relative to the base $B^* = (\beta_2, \ldots, \beta_k)$. Let $G^{[i]}$ denote $G_{(\beta_1, \ldots, \beta_{i-1})}$. By definition (1) holds for $i = 1$. We have to check that (1) holds for $2 \leq i \leq k+1$. For $i = 2$ we have that (1) holds since applying (2) with $j = 1$, we obtain $G_{\beta_1} = \langle S_2 \rangle \leq \langle S \cap G_{\beta_1} \rangle$. The reverse containment is obvious. For $i > 2$ we have that (1) follows from the fact that $S^* \cap G_{(\beta_1, \ldots, \beta_{i-1})}$ generates $\langle S_2 \rangle_{(\beta_2, \ldots, \beta_{i-1})}$by the inductive hypothesis, and so $G^{[i]} \geq \langle S \cap G_{(\beta_1, \ldots, \beta_{i-1})} \rangle \geq \langle S^* \cap G_{(\beta_1, \ldots, \beta_{i-1})} \rangle = \langle S_2 \rangle_{(\beta_2, \ldots, \beta_{i-1})} = (G_{\beta_1})_{(\beta_2, \ldots, \beta_{i-1})} = G^{[i]}$. □

7 The Schreier-Sims Algorithm

In this section we will explain the Schreier-Sims algorithm. The input of the algorithm will be a subset T of Sym(n) for some n. The output will be a Schreier tree data structure Δ.

Using lemma 6.3 we can create a set of generators for each $G^{[i+1]} \leq G^{[i]}$ and thus a strong generating set for G relative to B. However we might be using more generators than necessary. Therefore when adding a new generator, we will first test wether this generator is redundant. This can be done by sifting as described below. Note that to sift in a group we need a strong generating set for that group.

Given is a subset T of Sym(n). Let G be $\langle T \rangle$. We maintain a list $B = (\beta_1, \ldots, \beta_{m'})$ of already known elements of a non-redundant base for G. We also maintain a list $(S_1, S_2, \ldots, S_i, \ldots S_{m'})$, where each S_i is an approximation

for a generator set of the stabilizer $G_{(\beta_1,\ldots,\beta_{i-1})}$ for $1 \leq i \leq m'$. We always maintain for $1 \leq i \leq m'$ that:

$$\langle S_{i+1} \rangle \leq \langle S_j \rangle_{\beta_i} \leq \langle S_j \rangle$$

We say the data structure is *up to date below level* k if equation (2) holds for all j with $k < j \leq m'$.

The following diagram sketches the situation;

$$\begin{array}{ccccc}
\langle S_1 \rangle & \subset & G & & \\
\cup & & \cup & & \\
\langle S_2 \rangle & \subset & G_{(\beta_1)} & & \\
\cup & & \cup & & \\
\vdots & \vdots & \vdots & & \\
\langle S_j \rangle & \subset & G_{(\beta_1,\ldots,\beta_{j-1})} & = & G^{[j]} \\
\cup & & \cup & & \cup \\
\langle S_{j+1} \rangle & \subset & G_{(\beta_1,\ldots,\beta_j)} & = & G^{[j+1]}
\end{array}$$

Remark 7.1. During the algorithm we will choose new base points. In this section we will not elaborate on how to choose these base points. However, we do note one can choose a base such that the resulting Schreier tree data structure has cardinality $r = 1$.

Remark 7.2. The base we construct in the algorithm will be non-redundant.

We will now describe the algorithm.

Given is a subset T of $\mathrm{Sym}(n)$ and a positive integer r. Let G be $\langle T \rangle$. We maintain a list $B = (\beta_1, \ldots, \beta_{m'})$ of already known elements of a non-redundant base for G and an approximation S_i for a generator set of the stabilizer $G_{(\beta_1,\ldots,\beta_{i-1})}$ for $1 \leq i \leq m'$. We always maintain the property that for all i we have $\langle S_i \rangle \geq \langle S_{i+1} \rangle$. We say the data structure is *up to date below level* j if equation (2) holds for all i with $j < i \leq m'$.

We execute the Schreier-Sims algorithm as follows:

1. We set $S_1 = T$ and for $j \geq 2$ we set $S_j = \emptyset$. We start the algorithm by choosing a subset β_1 of $\{1, \cdots, n\}$ of cardinality at most r that is moved by a generator in T, and we set $m' = 1$. Now, the data structure is up to date below level $j = 1$.

2. If the data structure is up to date below level j, we compute the Schreier tree (T_j, S_j, f_j) for $\langle S_j \rangle$ mod $\langle S_j \rangle_{\beta_j}$.

3. We test whether equation (2) in Lemma 6.4 holds for j. This can be done by sifting the Schreier generators (see Lemma 6.3) obtained using the elements in the transversal encoded by the Schreier tree (T_j, S_j, f_j) and the elements in S_j, and applying Lemma 6.3 yielding generators for $\langle S_j \rangle_{\beta_j}$. We sift in the group $\langle S_{j+1} \rangle$. This is possible because the data structure is up to date below level j and thus by 6.4 we have a strong generating set for $\langle S_{j+1} \rangle$.

4. We now discern two cases.
   - If equation (2) holds for j, the data structure is up to date below level $j-1$.
   - Otherwise there is a Schreier generator s that has a non-trivial siftee. We add this Schreier generator to S_{j+1}. If $j = m'$ we also choose β_{j+1} to be a new subset of $\{1, \cdots, n\}$ of cardinality at most r that is moved by s and we increase m' by one. The data structure is now up to date below level $j+1$.
5. If the data structure is up to date below level 0, we are done. Lemma 6.4 implies correctness. Otherwise we go to step 2.

Remark 7.3. In the implementation of the algorithm, we do not recompute the entire Schreier tree (T_j, S_j, f_j) in step 2. Instead, we store the already computed Schreier trees and for a new element in S_j, we apply this element to each of the vertices in the previous Schreier tree. This yields an updated Schreier tree (T_j, S_j, f_j).

8 Complexity of the Schreier-Sims algorithm

We will now analyse the complexity of the Schreier-Sims algorithm as described above. This analysis is analogous to [3], but we have introduced the cardinality of the base, r, which we will use in our analysis.

Notation. Throughout this paper log will denote the log in base 2.

Theorem 8.1. *The Schreier-Sims algorithm taking input T and a natural number $r \geq 1$ constructs a Schreier tree data structure Δ of cardinality at most r in $O(n\binom{n}{r}^2 \log^3 |G| + n\binom{n}{r}^2 |T| \log |G|)$ time using $O(n \log^2 |G| + \binom{n}{r} \log |G| + |T|n)$ memory.*

Proof. The length of the base is at most $\log |G|$. In computing the Schreier tree (T_1, S_1, f_1) for step 2 as above, we apply each element of T to each of at most $\binom{n}{r}$ vertices in T_1. This takes $O(\binom{n}{r}|T|)$. For a fixed β_k in the base with $k > 1$, the set S_k changes at most $\log |G|$ times during the algorithm, since the group $\langle S_k \rangle$ increases every time we add an element to it. After a change of $\langle S_k \rangle$ we have to update the Schreier tree (T_k, S_k, f_k). There are at most $\binom{n}{r}$ vertices in T_k. When an element s is added to $\langle S_k \rangle$, to update the Schreier tree in step 2 as above, we have to calculate the image of each vertex in T_k under s only once. Calculating the image of a point under a permutation is $O(1)$. This calculation thus costs us $O(\binom{n}{r}|B|) = O(\binom{n}{r} \log |G|)$. So for all base points together constructing all Schreier trees as in step 2 as above takes $O(\binom{n}{r}|B| \log |G| + \binom{n}{r}|T|)$, which is

$$O(\binom{n}{r} \log^2 |G| + \binom{n}{r}|T|). \tag{3}$$

Now we estimate the time it takes to sift all Schreier generators for step 3 as above. Every element of the transversal R_k has to be combined with an element

of S_k only once. So the total number of Schreier generators is $\sum_k |R_k||S_k|$, where $|R_k|$ and $|S_k|$ are considered after the algorithm finishes. Then we have for any k that $|R_k| = O(\binom{n}{r})$. We have that $S_1 = T$ and for $k > 1$ we have that $|S_k| = O(\log |G|)$. Therefore it holds that

$$\sum_k |R_k||S_k| = O(\tbinom{n}{r} \log^2 |G| + \tbinom{n}{r}|T|).$$

To retrieve an element of R_k from the Schreier tree, we have to multiply all labels along a path in a Schreier tree; this means at most $\binom{n}{r}$ permutation multiplications costing $O(n)$ each. We have to retrieve at most $\log |G|$ of these elements, so the cost of one sift is

$$O(n\tbinom{n}{r} \log |G|).$$

The total cost for sifting all Schreier generators is the cost of one sift, multiplied by the number of Schreier generators. This is

$$O(n\tbinom{n}{r}^2 \log^3 |G| + n\tbinom{n}{r}^2 |T| \log |G|). \tag{4}$$

By (3) and (4) we conclude the total time cost of the algorithm is

$$O(n\tbinom{n}{r}^2 \log^3 |G| + n\tbinom{n}{r}^2 |T| \log |G|).$$

We have to store $\sum_k |S_k|$ strong generators which is at most $O(\log^2 |G|)$, therefore requiring $O(n \log^2 |G|)$ memory. Storing the Schreier trees requires $O(\binom{n}{r} \log |G|)$ memory. Storing the base costs mr which is $O(r \log |G|)$, but since $r \leq n$ this is dominated by the $O(n \log^2 |G|)$ above. Counting the $|T|n$ memory used to store the initial generators, we conclude there is a total memory cost of

$$O(n \log^2 |G| + \tbinom{n}{r} \log |G| + |T|n).$$

□

9 The extended Schreier-Sims algorithm and the extended membership testing algorithm

In this section we will describe an adaptation of the Schreier-Sims algorithm that creates a data structure with which an extended memberschip testing algorithm, given as input a permutation, will output a word for this permutation.

Definition 9.1. Let T be a subset of $\mathrm{Sym}(n)$. Given a Schreier tree data structure Δ for T with base $B = (\beta_1, \ldots, \beta_m)$. We define three types of **Schreier pointers** for Δ:

- Type 1: a sequence $p = (p_1, q_1, 1, \tilde{p})$ where p_1 is in T, $q_1 \in \{1, -1\}$ and $\tilde{p} = p_1^{q_1}$.

- Type 2: a sequence $p = (p_1, q_1, \ldots, p_c, q_c, i, \tilde{p})$ with $2 \leq i \leq m$ and such that p_k is another Schreier pointer, either $p'_k = (p'_1, q'_1 \ldots, p'_{c'}, q'_{c'}, i', j')$ or $p'_k = (p'_1, q'_1 \ldots, p'_{c'}, q'_{c'}, i', \tilde{p'})$, with $i' < i$ and $j' \in G^{[i]}\beta_i$ and q_k is in $\{-1, 1\}$ for all k and $\tilde{p} = \prod_{k=1}^{c} \tilde{p_k}^{q_k}$.
- Type 3: either
  - a sequence $p = (p_1, q_1, i, j)$ where p_1 is in T, $q_1 \in \{1, -1\}$, $1 \leq i \leq m$ and $j \in G^{[i]}\beta_i$ or
  - a sequence $p = (p_1, q_1, \ldots, p_c, q_c, i, j)$ with $2 \leq i \leq m$ and such that p_k is another Schreier pointer, either $p'_k = (p'_1, q'_1 \ldots, p'_{c'}, q'_{c'}, i', j')$ or $p'_k = (p'_1, q'_1 \ldots, p'_{c'}, q'_{c'}, i')$, with $i' < i$ and such that $j \in G^{[i]}\beta_i$. Also q_k is in $\{-1, 1\}$ for all k.

  For a type 3 Schreier pointer we define $\tilde{p} = \prod_{k=1}^{c} \tilde{p_k}^{q_k}$.

A type 1 Schreier pointer and a type 3 Schreier pointer of the first form is said to be of length 1; a type 2 Schreier pointer and a type 3 Schreier pointer of the second form is said to be of length c. A Schreier pointer is said to point at each of its p_i's. A Schreier pointer $p = (p_1, q_1, \ldots, p_c, q_c, i, j)$ or $p = (p_1, q_1, \ldots, p_c, q_c, i, \tilde{p})$ is said to be in the i-th Schreier tree.

Definition 9.2. Given a Schreier tree data structure Δ, a **pointer structure** for Δ is a set P of Schreier pointers such that the map from

$$\{p \in P | p \text{ is a type 3 Schreier pointer}\} \to \{(i, j) : 1 \leq i \leq m,\ j \in G^{[i]}\beta_i\ ,\ j \neq \beta_i\}$$

$$(p_1, q_1, \ldots, p_c, q_c, i, j) \mapsto (i, j)$$

is bijective and such that for every p in P we have that $\overline{p}$ is equal to the label of the unique last edge in the path from the root of T_i to the vertex j. The Schreier pointer p is said to correspond to this unique last edge.

The **extended Schreier-Sims algorithm** takes as input a subset T of $\mathrm{Sym}(n)$ and an integer r and outputs both a Schreier tree data structure Δ of cardinality at most r and a pointer structure P for Δ. The algorithm executes the Schreier-Sims algorithm. The following Schreier pointers are added to P:

- At initialization we add a type 1 Schreier pointer $(t, 1, 1, t)$ to P for each $t \in T$.
- In step 4 of the Schreier-Sims algorithm, if a new Schreier generator s is added to S_{j+1} we add a type 2 Schreier pointer to P. The Schreier generator s has been constructed as $(\overline{s'r})^{-1}s'r$ with $s' \in S_j$ and $r \in R_j$. The added Schreier pointer $p = (p_1, q_1, \ldots, p_c, q_c, i, \tilde{p})$ is constructed as follows.
  - There is a unique path in the j-th Schreier tree from the root to $\overline{s'r}(\beta_i)$ with Schreier pointers $(a_1, \ldots, a_n)$ corresponding to the edges of this path such that $\tilde{a_n} \cdots \tilde{a_1} = \overline{s'r}$ and thus $\tilde{a_1}^{-1} \cdots \tilde{a_n}^{-1} = (\overline{s'r})^{-1}$.

- There is a Schreier pointer b in the j-th Schreier tree with $\tilde{b} = s'$.
- There is a unique path in the j-th Schreier tree from the root to $r(\beta_i)$ with Schreier pointers $(c_1, \ldots, c_m)$ corresponding to the edges of this path such that $\tilde{c_m} \cdots \tilde{c_1} = r$.
- We set the Schreier pointer we add to

$$p = (a_1, -1, \ldots, a_n, -1, b, 1, c_m, 1, \ldots c_1, 1, j+1, \tilde{p}).$$

  It holds that $\tilde{p} = (\overline{s'r})^{-1} s' r = s$

- In step 2 of the Schreier-Sims algorithm we update the i-th Schreier tree. For each Schreier generator s that is newly added to S_i we apply this s to each vertex in the existing Schreier tree. For each vertex v for which applying s results in a new vertex v' in the updated Schreier tree, the extended Schreier-Sims algorithm adds a type 3 Schreier pointer p to P.

  If $i = 1$ then s is equal to a $t \in T$. We set the Schreier pointer we add to $(t, 1, 1, v')$.

  If $i \geq 2$ then we have added a type 2 Schreier pointer $p = (p_1, q_1, \ldots, p_c, q_c, i, \tilde{p})$ with $\tilde{p} = s$ to the i-th Schreier tree. We add the Schreier pointer $p' = (p_1, q_1, \ldots, p_c, q_c, i, v')$.

Note that in the implementation we can maintain a list of type 2 Schreier pointers instead of a list of the strong generators since the type 2 Schreier pointers contain the strong generators.

The **extended membership testing algorithm** for a Schreier tree data structure Δ for a subset T of $\mathrm{Sym}(n)$ and a pointer structure P for Δ takes as input a permutation $g \in \mathrm{Sym}(n)$. If s is in $\langle T \rangle$ it will output a word in T for g. This is done by sifting g through Δ. By Remark 6.2 we write g during the sifting process by multiplying labels of edges in the Schreier trees of Δ. The pointer stucture P by definition encodes a word for every label of every edge in the Schreier trees of Δ.

When sifting through the i-th Schreier tree there is a unique path from β_i to $g_i(\beta_i)$ with edges $(e_1, \ldots, e_n)$. We first output the word encoded by the Schreier pointer corresponding to e_n, then the word encoded by the Schreier pointer corresponding to e_{n-1} etcetera. Algorithm 6.1 shows that the resulting output is a word for g in T.

10 Output word length

Notation. Given a Schreier tree T_i we define the height of a Schreier tree as the longest path from the root towards a vertex of the tree. For example a tree with one vertex has height 0. We will denote the height of the tree as $h(T_i)$. We define $h(T_0) = 0$.

In this section we will prove an upper limit for the word length for words that are output of the extended membership algorithm. We will use the following lemma.

Lemma 10.1. *In a pointer structure constructed by the extended Schreier-Sims algorithm, a Schreier pointer for the i-th Schreier tree is of length at most*

$$2 \cdot h(T_{i-1}) + 1$$

for $i \geq 2$ and exactly 1 for $i = 1$.

Proof. A Schreier pointer in the i-th tree consists of three parts. The length of the parts corresponding to $(\overline{sr})^{-1}$ and r can be at most equal to the longest path in T_{i-1} i.e. at most $h(T_{i-1})$. The part corresponding to s is of length 1. Thus the Schreier pointer is of length at most $2 \cdot h(T_{i-1}) + 1$. The length of each Schreier pointer in the first Schreier tree is 1 by definition. □

Theorem 10.2. *Given input $T \subset \mathrm{Sym}(n)$, $s \in \langle T \rangle = G$ and a Schreier tree data structure Δ and a pointer structure P for Δ, constructed by the extended Schreier-Sims algorithm. Then the extended membership testing algorithm will output a word for s with letters in T that is no longer than $|G|^2$ letters.*

Proof. Let m denote the length of the base used. By Lemma 10.1 we know that the maximal length of a Schreier pointer in the i-th Schreier tree is

$$2 \cdot h(T_{i-1}) + 1.$$

In the extended membership testing algorithm we sift s in Δ. When sifting through the i-th Schreier tree, at most $h(T_i)$ Schreier pointers are used to output the word for s. the Schreier pointer only points to pointers in $(i-1)$-th Schreier tree. We output a letter if the Schreier pointer is in the first Schreier tree. So to write a Schreier pointer corresponding to an edge in the i-th Schreier tree explicitly at most

$$\prod_{k=1}^{i-1} (2 \cdot h(T_k) + 1)$$

letters are output. Therefore during the sifting through the i-th Schreier tree at most

$$h(T_i) \prod_{k=1}^{i-1} (2 \cdot h(T_k) + 1)$$

letters are output. During the whole sifting process, there are at most

$$\sum_{i=1}^{m} \left[h(T_i) \prod_{k=1}^{i-1} (2 \cdot h(T_k) + 1) \right]$$

letters output. This is smaller than

$$\sum_{i=1}^{m} \left[2^{i-1} \cdot \prod_{k=1}^{i} (h(T_k) + 1) \right].$$

Now $h(T_k) + 1$ is at most the number of vertices in T_k. By Lemma 6.2 we have that $\prod_i |T_i| = |G|$. So this is at most

$$\sum_{i=1}^{m} 2^{i-1} \cdot |G| < 2^m |G|.$$

The length of the base m is at most $\log |G|$, therefore the length of the word we output is at most

$$|G|^2.$$

□

11 Complexity analysis

We will now calculate the complexity of the extended Schreier-Sims algorithm and the extended membership testing algorithm.

The cost of storing a permutation in $\mathrm{Sym}(n)$ is n.

Theorem 11.1. *Given a subset T of $\mathrm{Sym}(n)$ and a natural number $r \geq 1$, the extended Schreier-Sims algorithm will output a Schreier tree data structure Δ for T and a pointer structure P for Δ of cardinality r requiring*

$$O(\tbinom{n}{r} \log^2 |G| + \tbinom{n}{r}^2 \log |G|) + |T|n)$$

memory, and in

$$O(n\tbinom{n}{r}^2 \log^3 |G| + n\tbinom{n}{r}^2 |T| \log |G|)$$

time.

Proof. By Theorem 8.1 the Schreier-Sims algorithm has a memory requirement of

$$O(n \log^2 |G| + \tbinom{n}{r} \log |G| + |T|n).$$

In the extended Schreier-Sims algorithm, for each edge in each Schreier tree we also store a Schreier pointer. Also for each strong generator we store a Schreier pointer.

By Lemma 10.1 we know that the length of a Schreier pointer in the i-th Schreier tree is at most $2 \cdot h(T_{i-1}) + 1$. The height of the Schreier trees is at most $\binom{n}{r}$. Thus the length of a Schreier pointer is at most $2 \cdot \binom{n}{r} + 1$. If $i \geq 2$ storing the Schreier pointer has a memory requirement equal to the length of the Schreier pointer. For $i = 1$ the cost for storing a Schreier pointer is equal to the cost of storing a permutation of $\mathrm{Sym}(n)$ which is $n \leq \binom{n}{r} < 2 \cdot \binom{n}{r} + 1$.

A Schreier tree contains at most $\binom{n}{r}$ vertices. There are at most $\log |G|$ Schreier trees. As we have seen in the proof of Theorem 8.1 we store $O(\log^2 |G|)$ strong generators. Therefore the cost of storing all Schreier pointers is at most

$$O((2 \cdot \tbinom{n}{r} + 1)(\log^2 |G| + \tbinom{n}{r} \log |G|)) = O(\tbinom{n}{r} \log^2 |G| + \tbinom{n}{r}^2 \log |G|)$$

Total memory requirement is

$$O(n \log^2 |G| + \binom{n}{r} \log |G| + |T|n + \binom{n}{r} \log^2 |G| + \binom{n}{r}^2 \log |G|)$$

which is

$$O(\binom{n}{r} \log^2 |G| + \binom{n}{r}^2 \log |G| + |T|n).$$

By Theorem 8.1 the Schreier-Sims algorithm has a time requirement of

$$O(n\binom{n}{r}^2 \log^3 |G| + n\binom{n}{r}^2 |T| \log |G|).$$

Adding a Schreier pointer has a time requirement equal to the length of that Schreier pointer. Therefore, analogous to above, adding the Schreier pointers takes

$$O(\binom{n}{r} \log^2 |G| + \binom{n}{r}^2 \log |G|)$$

We conclude there is a total time requirement of

$$O(n\binom{n}{r}^2 \log^3 |G| + n\binom{n}{r}^2 |T| \log |G| + \binom{n}{r} \log^2 |G| + \binom{n}{r}^2 \log |G|)$$

which is equal to

$$O(n\binom{n}{r}^2 \log^3 |G| + n\binom{n}{r}^2 |T| \log |G|).$$

□

Theorem 11.2. *Let T be a subset of* $\mathrm{Sym}(n)$ *and let Δ and P be a Schreier tree data structure for T and a pointer structure for Δ, both created by the extended Schreier-Sims algorithm. The extended membership algorithm, given input $s \in \mathrm{Sym}(n)$, outputs a word for s in T requiring at most*

$$O(n \cdot |G|^2 + n\binom{n}{r} \log |G|)$$

time.

Proof. As we have seen in the proof of Theorem 8.1, the cost of one sift in the Schreier-Sims algorithm is $O(n\binom{n}{r} \log |G|)$

The extended membership testing algorithm also uses the pointer structure. For every edge label used in membership testing, we have to explicitly write the Schreier pointer corresponding to that edge. By Lemma 10.1 a pointer in the i-th Schreier tree has length at most

$$2 \cdot h(T_{i-1}) + 1.$$

Such a pointer points only to pointers in i-th Schreier tree. Writing a Schreier pointer in the first Schreier tree explicitly costs n. Therefore writing such a pointer explicitly will cost at most

$$n \cdot \prod_{k=1}^{i-1}(2 \cdot h(T_k) + 1)$$

time. Therefore the sifting through the i-th Schreier tree takes at most

$$n \cdot h(T_i) \prod_{k=1}^{i-1}(2 \cdot h(T_k) + 1).$$

Sifting through all Schreier trees takes at most

$$n \cdot \sum_{i=1}^{m} \big[h(T_i) \prod_{k=1}^{i-1}(2 \cdot h(T_k) + 1)\big].$$

This is smaller than

$$n \cdot \sum_{i=1}^{m} \big[2^{i-1} \cdot \prod_{k=1}^{i}(h(T_k) + 1)\big].$$

Now $h(T_k) + 1$ is at most the number of vertices in T_k. By Lemma 6.2 we have that $\prod_i |T_i| = |G|$. So this is at most

$$n \cdot \sum_{i=1}^{m} 2^{i-1} \cdot |G| < n \cdot 2^m |G|.$$

The length of the base m is at most $\log |G|$, therefore the added time requirement for sifting is

$$O(n \cdot |G|^2).$$

The total time requirement becomes

$$O(n \cdot |G|^2 + n(\tbinom{n}{r}) \log |G|).$$

□

12 Concluding remarks

In this paper we set out to describe an algorithm for writing a general solution for an arbitrary permutation puzzle. In section 3 we saw that the length of the output of such an algorithm can not be polynomial in the number of elements that are permuted. In section 4 we saw that the length of the output can be linear in the group order.

As we have seen in Theorem 11.1 the algorithm we have suggested has, for a fixed size of the elements of the base and taking the length of the input into account, a polynomial time and memory requirement for setup (the extended Schreier-Sims algorithm) and using this setup to solve the permutation puzzle

(the extended membership testing algorithm) is has a polynomial time requirement in the group order, as we saw in Theorem 11.2. In Theorem 10.2 we proved that the length of the output is quadratic in the group order.

This leaves the challenge to find an algorithm with polynomial setup requirements, that has a linear output length.

References

[1] Apostol, Tom M. Introduction to Analytic Number Theory. New York: Springer-Verlag, 1976.

[2] Babai, László. "Chapter 27: Automorphism groups, isomorphism, reconstruction". In Graham, R. L.; Grötschel, M.; Lovász, L.. Handbook of Combinatorics. Amsterdam: Elsevier, 1995. pp, 1447-1540.

[3] Seress, Ákos. Permutation Group Algorithms. 1st ed. Cambridge: Cambridge University Press, 2003.

[4] Turner, Edward C.; Gold, Karen F. "Rubik's Groups" in The American Mathematical Monthly, Vol 92, No 9 (Nov., 1985), pp. 617-629.