

Finding the Smallest Image of a Set

Steve Linton

Centre for Interdisciplinary Research in Computational Algebra

University of St Andrews

The North Haugh

St Andrews, Fife KY16 9SS, U.K.

sal@dcs.st-and.ac.uk *

ABSTRACT

We describe an algorithm for finding a canonical image of a set of points under the action of a permutation group. Specifically if we order images by sorting them and ordering the resulting sequences lexicographically, we find the first image. This has applications to combinatorial and other search problems, allowing isomorphic results to be eliminated more efficiently.

We give worst-case asymptotic running time estimates and practical results obtained with a GAP implementation.

Keywords

permutation group, algorithm, smallest image

General Terms

Algorithms

Categories and Subject Descriptors

F [2]: 2; G [2]: 1; I [1]: 2

1. INTRODUCTION

Many combinatorial and other problems of interest [10, 1] reduce to finding all inequivalent subsets of the domain of some permutation group with a specified property. Here equivalence of two subsets simply means that there exists an element of group mapping one to the other as a set. Typically such problems are addressed using depth-first search, modified to avoid some parts of the search tree which are known to be equivalent to parts not avoided. Often however, it is not possible, or too expensive, to ensure that the search produces no pairs of equivalent sets. It may therefore be necessary to perform a final pass to reduce the list strictly

*This work was supported by EPSRC grants GR/R29666 and GR/S30580

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 2002 ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

to a set of equivalence class representatives. This process is called *isomorph rejection*.

Effective algorithms exist [3, 4], to test whether two sets are equivalent (although this problem is polynomially equivalent to graph isomorphism [5], so polynomial worst-case running time is not to be expected), however using these to perform isomorph rejection on a collection of n sets involves $O(n^2)$ equivalence tests. The algorithm in this paper determines the minimal set equivalent to a given set, in the ordering given by first sorting the sets into increasing order and then comparing them lexicographically. It seems to run in a time practically comparable to that required for a single equivalence test and allows isomorph rejection on a collection of n sets in using $O(n)$ calls to this algorithm, plus $O(n \log n)$ comparisons of sequences, a dramatic improvement.

The algorithm of this paper has been implemented in GAP [2] and will be included in a forthcoming version of the GRAPE [9] share package. An implementation of it can also be obtained from the deposited contributions section of the GAP web site www.gap-system.org.

2. STANDARD ALGORITHMS

We refer to [6] for standard results and algorithms for computing with permutation groups. We especially use the existence of the stabilizer chain data structure, which represents a chain of subgroups

$$G = G_0 > G_1 > \cdots > G_l = \langle () \rangle$$

where each G_i is the stabilizer in G_{i-1} of a single *base point* b_i . The cosets of G_i in G_{i-1} are thus in bijection with the *basic orbit* $b_i^{G_{i-1}}$ and the product of the lengths of the basic orbits is $|G|$. The data structure also involves a fixed set of permutations which includes generators for all the G_i , the *strong generating set*, and allows one to find very efficiently a word in the strong generators representing an element of the coset $G_i g$ for any $g \in G_{i-1}$, equivalently a word representing an element mapping any point in the i th basic orbit to b_i .

Theorem 4.5.5, of [6], essentially tells us that, provided that we *either* know $|G|$ *a priori* or are prepared to tolerate a small error probability, and that G is given by a reasonably small generating set, then we can construct a stabilizer chain and associated information in time $O(n \log n (\log |G|)^4)$. Furthermore, the words for the coset representatives which are encoded in this structure have length at most $O(\log n)$, and the number of generators constructed for each G_i is also at most $O(\log n)$.

Also important for our purposes in the *change of base* algorithm, described in section 5.4 of [6], which allows one to compute a stabilizer chain corresponding to a different sequence b'_i given a stabilizer chain for b_i . This runs in time $O(n(\log n)^2(\log |G|)^2)$.

Finally, we refer to section 9.2 of [6], for an outline of the partition backtrack algorithm for finding a set stabilizer or testing equivalence of sets under a permutation group, which is described in detail in [4] and [3].

3. THE BASIC ALGORITHM

3.1 Outline

We assume that we are given (generating permutations for) a group G , acting on a finite set Ω of size n and a subset Δ of Ω of size k .

The basic algorithm proceeds iteratively in k iterations.

At iteration i , we determine the initial length i subset M_i of the smallest setwise image of Δ , which, crucially, is simply the lexicographically smallest sequence of length i onto which any sequence of i elements of Δ can be mapped.

In addition, at each iteration, we find *all* length i sequences Γ of elements of Δ which can be mapped onto M_i , which we call *candidates*. The i th iterative step then examines each candidate Γ emerging from the previous step to see what extended sequences $\Gamma :: \delta$ can be mapped to a new *global* minimum image M_i .

This is essentially a breadth-first search technique. This has the advantage for this problem, compared to the more usual depth-first approach, that, at each iteration, we can definitively recognise the global optimum partial solution, and eliminate branches of the search tree which do not lead to it. It has the disadvantage that a great deal of storage may be required to record all the search nodes (candidates) at a given depth, before going on to the next depth. As we shall see below, this does not seem to be a major problem in practice. If necessary, an approach such as *iterative deepening* could be used to reduce storage requirements at some cost in run time.

3.2 The Algorithm in Detail

We fix an ordering of Δ so that $\Delta = [\delta_1, \dots, \delta_k]$ as a sequence. We denote the set $\{1 \dots k\}$ by I . Given any sequence $J = [j_1, j_2, \dots, j_m]$ of elements of I , we write Δ_J for the sequence $[\delta_{j_1}, \dots, \delta_{j_m}]$. We write \bar{J} for $I \setminus J$ when J is a subset of I . We write $S :: x$ for the sequence obtained by appending x to the sequence S .

We define a *candidate record* R as consisting of J_R a duplicate-free sequence of elements of I , and a sequence $D_R = \Delta^g$ where g is some element such that $\Delta_{J_R}^g = M_{|J_R|}$. We write R as an ordered pair (J_R, D_R) . We define the *depth* of a record R to be $|J_R|$.

As an example, consider the cyclic group C_{10} in its natural action on $\Omega = \{1, \dots, 10\}$ and take $\Delta = [1, 7]$ and $i = 1$. Then $M_1 = [1]$ and the set of candidates of depth one is

$$\{([1], [1, 7]), ([2], [5, 1])\}$$

In this, case, this set of candidates is actually unique, because C_{10} is sharply 1-transitive. In general, there will be some ambiguity in the choice of D_R for some or all R , corresponding to choice of g in the definition of D_R . This causes no problem for our algorithm.

Each iterative step of the algorithm will take a list of candidate records of depth $i - 1$ and produce a new list of candidate records of depth i . The step also determines M_i and $G_i = \text{Stab}_G(M_i)$ (where the stabilizer is taken sequentially).

We can now state the basic algorithm.

ALGORITHM 1.

1. Set *Cands* to $[[[]], \Delta]$
2. Set G_0 to G
3. Set M_0 to $[\]$
4. If one is not already known, compute a stabilizer chain for G_0
5. For i from 1 to k do
 - (a) Set $m = n + 1$
 - (b) For each $R \in \text{Cands}$.
 - i. Compute the smallest element m_R in the orbit of any $x \in (D_R)_{\bar{J}_R}$ under G_{i-1} .
 - ii. if $m_R < m$ then set *Pass* to $[R]$, and m to m_R
 - iii. if $m_R = m$ then add R to *Pass*.
 - (c) Set M_i to $M_{i-1} :: m$
 - (d) Change the base of the stabilizer chain of G_{i-1} so that m is the first base point. G_i is therefore the first stabilizer in the chain.
 - (e) Set *Cands* to $[\]$
 - (f) For each R in *Pass*.
 - i. For each $x \in D_R \cap m^{G_{i-1}}$
 - A. Let g be an element of G_{i-1} mapping x to m
 - B. Let j be the index of x in D_R
 - C. Add $(J_R :: j, D_R^g)$ to *Cands*
6. Return M_k .

REMARK 2. There are a number of points to note about this algorithm:

- On the final pass, when $i = k$, we can stop the computation after step 5c, since we will have computed M_k .
- The element g defined in step 5(f)iA need not actually be computed. Using the stabilizer chain data structure for G_{i-1} we can compute a word for g in the generators of G_{i-1} and apply this to D_R as a sequence without actually multiplying permutations.
- All of the base change computations can be done within a single stabilizer chain data structure for G , which will eventually have M_k as an initial subsequence of its basis. No separate permutation groups need to be computed explicitly.
- If at any iteration we find that G_{i-1} is trivial, then M_k will simply be smallest of the D_R for $R \in \text{Cands}$.

3.3 Proof of Correctness

It is clear that the algorithm terminates and returns a set in the orbit of Δ . To prove that this is the lexicographically smallest such set, we need

LEMMA 3. *With the notation established above, let J be a length i sequence of elements of I such that there exists $g \in G$ with $\Delta_J^g = M_i$. After the i th iteration of the outer loop in algorithm 1, $Cands$ will contain a record R with $J_R = J$.*

PROOF. This is proved by induction on i . The base case, $i = 0$ is clear, since $[\]$ is the only possible J .

Otherwise suppose the lemma holds for $i - 1$. Let J be a sequence as in the lemma and g the corresponding group element, and let $J = J' :: j$. Then since $J'^g = M_{i-1}$ there must be an R' in $Cands$ after step $i - 1$ with $J_{R'} = J'$.

Let g' be such that $D_{R'} = \Delta^{g'}$.

Now $g'^{-1}g$ fixes M_{i-1} as a sequence, and so lies in G_{i-1} . But, $\delta_j^{g'} = (D_{R'})_j \in (D_{R'})_{\overline{J}}$ and $(\delta_j^{g'})^{g'^{-1}g} = \delta_j^g = m_i$, so that after R' has been processed in step 5(b)i, m will be set to m_i and R' will be in $Pass$. Furthermore, since m does not change in processing any later candidate, R' will still be in $Pass$ at the completion of step 5b.

Now when we come to step 5(f)i, we see by the same calculation that $\delta_j^{g'}$ is certainly in $D_{R'} \cap m^{G_{i-1}}$ and so there will be a pass through this loop with $x = \delta_j^{g'}$, which will certainly add to $Cands$ a record R with $J_R = J$. This completes the induction and proves the lemma. \square

Now we can prove

THEOREM 4. *Algorithm 1 computes the lexicographically earliest image of Δ under the setwise action of G .*

PROOF. Suppose that there exists $\Gamma = \Delta^g$ which is lexicographically smaller (when sorted) than M_k . Specifically, let $\Gamma = [\gamma_1, \dots, \gamma_k]$ (in increasing order) and $M_k = [m_1, \dots, m_k]$ (also in increasing order by construction). Let π be a permutation of $\{1 \dots k\}$ such that $\gamma_j = \delta_{j\pi}^g$ for each j .

Let i be such that $\gamma_j = m_j$ for all $j < i$ and $\gamma_i < m_i$.

Then, by lemma 3, we know that there will be a candidate record R after pass $i - 1$ such that $J_R = M_{i-1} = [\gamma_1, \dots, \gamma_{i-1}]$. Let $D_R = \Delta^{g'}$. As above, $g'^{-1}g \in G_{i-1}$, $\delta_{i\pi}^{g'} \in (D_R)_{\overline{J_R}}$. Now $(\delta_{i\pi}^{g'})^{g'^{-1}g} = \gamma_i < m_i$. But now, when record R is considered, at step 5(b)i, m will be set to γ_i , and can then never be increased to m_i which is defined to be its final value, a contradiction. \square

EXAMPLE 5. *Let G be S_9 acting on unordered pairs of points, ordered lexicographically. We write ab (where $a \leq b$) for the unordered pair $\{a, b\}$.*

Let Δ be $[36, 25, 34, 55, 67]$.

- *When $i = 1$, $Cands$ is initially $[[\], [36, 25, 34, 55, 67]]$*

The smallest G -image of any point of Δ is 11 which can be reached as an image of 55.

Thus, on this iteration, $m = 11$, $Pass = Cands$, and $G_1 = \text{Stab}_G(1) \cong S_8$

In the second loop, we find that only 55 can be mapped to m , and g can be taken to be $(1, 5)$, thus we set $Cands$ to $[[[4], [36, 12, 34, 11, 67]]]$

- *When $i = 2$, there is just one candidate record R and the smallest image of any element of $(D_R)_{\overline{J_R}}$ is 12*

So, $m = 12$, $Pass = [R]$ and $G_2 = \text{Stab}_G([1, 2]) \cong S_7$

In the second loop, only one element of $(D_R)_{\overline{J_R}}$ can be mapped onto 12, namely 12 itself, say by the identity permutation, so we set $Cands$ to

$$[[[4, 2], [36, 12, 34, 11, 67]]]$$

- *When $i = 3$, there is again just one candidate record R and the smallest image of any element of $(D_R)_{\overline{J_R}}$ is 34*

So, $m = 34$, $Pass = [R]$ and

$$G_3 = \text{Stab}_G([1, 2], \{3, 4\}) \cong S_5 \times S_2$$

In the second loop, however, all three of 34, 36 and 67 can be mapped onto 34, by, say, $(\)$, $(4, 5, 6)$ and $(3, 7, 4, 6)$ respectively. So we set $Cands$ to

$$\begin{aligned} &[[[4, 2, 1], [34, 12, 36, 11, 67]], \\ &[[4, 2, 3], [35, 12, 34, 11, 47]], \\ &[[4, 2, 5], [37, 12, 67, 11, 34]]] \end{aligned}$$

- *When $i = 4$, there are three candidate records, which we denote R_1, R_2 and R_3 .*

In the first loop, we find the minimal image of any point of any $(D_R)_{\overline{J_R}}$ to be 35 and this image is achieved at least once for each R

So, $m = 35$, $Pass = [R_1, R_2, R_3]$ and

$$G_4 = \text{Stab}_G([1, 2, 3, 4, 5]) \cong S_4$$

In the second loop, we find that R_1, R_2 and R_3 contribute 1, 2 and 1 entry to the new $Cands$ respectively, which becomes

$$\begin{aligned} &[[[4, 2, 1, 3], [34, 12, 35, 11, 57]], \\ &[[4, 2, 3, 1], [35, 12, 34, 11, 47]], \\ &[[4, 2, 3, 5], [47, 12, 34, 11, 35]], \\ &[[4, 2, 5, 1], [35, 12, 56, 11, 34]]] \end{aligned}$$

- *Finally, when $i = 5$, we have these four candidate records R_1, \dots, R_4 and in the first loop, examining R_1 we see that 57 can be mapped to 56, which initially becomes m , and R_1 is added to $Pass$. Examining R_2 , however, we find that 47 can be mapped to 46, so we change m and reset $Pass$ to $[R_2]$. Examining R_3 we see that once again 47 can be mapped to 46, so that we add R_3 to $Pass$. Finally R_4 is not added to $Pass$, since 56 cannot be mapped to anything as small as 46.*

So, $m = 46$, and we can stop, concluding that the smallest image of Δ is $\{11, 12, 34, 35, 46\}$.

REMARK 6. *The last step of this example demonstrates a possible optimisation, which we call "pruning". It can happen that two candidate records R and R' have $(D_R)_{\overline{J_R}}$ equal to $(D_{R'})_{\overline{J_{R'}}}$ as sets. In this case R' can be ignored, since any M_k eventually arising from considering R' will also arise by considering R . In practice this optimisation rarely seems to be worth the overhead involved in applying it, except in cases when the advanced algorithm is in any event better.*

3.4 Complexity Analysis

The work of the computation can essentially be divided into four parts

- The initial computation of a stabilizer chain for G . In most applications this will be known anyway, since G will be the overall symmetry group of the problem, and so this work need not be accounted to the smallest image algorithm. When this is not the case, it can be computed in
 - k calls to the base change algorithm to modify the stabilizer chain of G . Each takes time
- In the context of isomorph rejection, the algorithm will often be called twice in succession with sets in the same orbit. In this case, the second call will not need to change the stabilizer chain.
- Computing orbits under G_{i-1} . The calculation can be arranged so that each orbit is computed at most once. Furthermore G_{i-1} has at most $O((\log n)^c)$ generators, so this work thus takes time at most $O(n(\log n)^c)$ on each pass, or $O(kn(\log n)^c)$ in total.
 - Work on the candidates. The processing of each candidate takes time $O(k(\log n)^c)$ since for each candidate, we need to compute the image of a sequence of k points under a word which can be shown to have length at most $O((\log n)^c)$. The other work on each candidate is dominated by this.

The first three of these parts give very satisfactory estimates. This leaves the problem of bounding the number of candidates at each iteration. Three bounds can be easily seen:

- At iteration i there can be at most $k!/i!$ distinct J s
- At iteration i there can be at most $\binom{n}{k-i}$ distinct D s, so provided we use the pruning optimization, this limits the number of candidates
- At any iteration there can be at most $|G|$ candidates.

While it is perfectly possible that each of these will apply at different steps in the same computation, and experiment suggests that the actual number of candidates is much smaller, if you sum each of these over i you obtain:

- A total of $O(k!)$
- A total of $O(n^k)$ or $O(2^n)$ whichever is smaller
- A total of $k|G|$

for the total number of candidates to be considered.

For different assumptions about the relative growth of n , k and $|G|$ these facts can be combined to obtain various asymptotic overall worst-case complexities for the algorithm. In particular

- if $k = O(\log n)$ which is realistic for some families of problems, then $\log(k!) = O(k \log k)$ so running time is $O(n^{c \log \log n})$ which is very close to polynomial.
- Even when k is close to n , the running time is still exponentially bounded.
- Finally if $|G|$ grows polynomially in n then the running time is polynomial in n for any k .

4. ADVANCED ALGORITHM

One situation in which the basic algorithm performs very badly is when the setwise stabilizer of Δ induces a large permutation group on Δ . In this case, if any sequence X of elements of Δ can be mapped onto a sequence M , then so can all sequences in the orbit of X under this group, which thus has an action on the set of candidates after each iteration of algorithm 1. All but one of the candidates in an orbit under this action will be redundant for the purposes of this problem.

We can modify the algorithm to avoid computing some of these redundant candidates, at the cost of some extra group theoretic computation. This gives a big speed-up in some difficult cases, but can slow the algorithm down in simple cases.

We take as an additional input to the algorithm H , a subgroup of the action of the setwise stabilizer G_Δ on Δ . One possibility is to simply compute G_Δ using partition backtrack as part of the initialization of the algorithm, but this is not always a good choice, as the time for this computation can overwhelm the saving made later.

We extend our definition of a candidate record R , adding a field H_R which is a subgroup of the sequence stabilizer $\text{Stab}_H(J_R)$.

ALGORITHM 7. *The advanced algorithm differs from the basic algorithm in just two places:*

- At step 5(f)i, we take a set A of orbit representatives of the action of H_R on \bar{J}_R and loop only over $(D_R)_A \cap m^{G_{i-1}}$.
- In step 5(f)iC add the point stabilizer $\text{Stab}_{H_R}(j)$ as a third field in the new candidate record being constructed

We prove this algorithm correct in the same way as the basic algorithm. In this case, the appropriate algorithm of lemma 3 is:

LEMMA 8. *With the notation established above, let J be a length i sequence of elements of I such that there exists $g \in G$ with $\Delta_j^g = M_i$. After the i th iteration of the outer loop in algorithm 1, $Cands$ will contain a record R such that there exists $h \in H$ such that $J_R^h = J$.*

PROOF. This is proved by induction on i . The base case, $i = 0$ is clear, since $[\]$ is the only possible J and $h = ()$ suffices.

Otherwise suppose the lemma holds for $i - 1$. Let J be a sequence as in the lemma and g the corresponding group element, and let $J = J' :: j$. Then since $J'^g = M_{i-1}$ there must be an R' in $Cands$ after step $i - 1$ and $h \in H$ with $J_{R'}^h = J'$.

Let g' be such that $D_{R'} = \Delta^{g'}$ and let a be an element of g stabilizing Δ as a set and affording the permutation h there.

Now $g'^{-1}ag$ fixes M_{i-1} as a sequence, and so lies in G_{i-1} . But, $\delta_j^{g'} = (D_{R'})_j \in (D_{R'})_{\bar{J}}$ and $(\delta_{j^{h^{-1}}}^{g'})^{g'^{-1}ag} = \delta_j^g = m_i$, so that after R' has been processed in step 5(b)i, m will be set to m_i and R' will be in $Pass$. Furthermore, since m does not change in processing any later candidate, R' will still be in $Pass$ at the completion of step 5b.

Now when we come to step 5(f)i, we see by the same calculation that $\delta_{j^{h^{-1}}}^{g'}$ is certainly in $D_{R'} \cap m^{G_{i-1}}$, and, in

fact, so is any $\delta_{j^{h^{-1}h'}}^{g'}$, for any $h' \in \text{Stab}_H(J_{R'})$. There will thus be a pass through this loop corresponding to the orbit of $j^{h^{-1}}$ under H'_R , which will add an entry R to the new list *Cands*. Now J_R will be $J_{R'} :: j^{h^{-1}h'}$, so $J_R^{h'^{-1}h} = J$. This completes the induction and proves the lemma. \square

Now we can prove:

THEOREM 9. *Algorithm 7 computes the minimal image of Δ under G .*

PROOF. Suppose that there exists $\Gamma = \Delta^g$ which is lexicographically smaller (when sorted) than M_k . Specifically, let $\Gamma = [\gamma_1, \dots, \gamma_k]$ (in increasing order) and $M_k = [m_1, \dots, m_k]$ (also in increasing order by construction). Let π be a permutation of $\{1 \dots k\}$ such that $\gamma_j = \delta_{j\pi}^g$ for each j .

Let i be such that $\gamma_j = m_j$ for all $j < i$ and $\gamma_i < m_i$.

Then, by lemma 3, we know that there will be a candidate record R after pass $i - 1$ and $h \in H$ such that $J_R^h = M_{i-1} = [\gamma_1, \dots, \gamma_{i-1}]$. Let $D_R = \Delta^{g'}$. Let a be an element of G affording h on Δ . As above, $g'^{-1}ag \in G_{i-1}$, $\delta_{i\pi h^{-1}}^{g'} \in (D_R)_{J_R}$. Now $(\delta_{i\pi h^{-1}}^{g'})^{g'^{-1}ag} = \gamma_i < m_i$. But now, when record R is considered, at step 5(b)i, m will be set to γ_i , and can then never be increased to m_i which is defined to be its final value, a contradiction.

\square

5. IMPLEMENTATION ISSUES

The algorithm has been implemented in GAP version 4.3. The standard GAP functions for computing stabilizer chains and doing base changes were used. In step 5(b)i we need to run through a possibly long list of candidates and find the smallest point to which any relevant point of any of them can be mapped under a group. This involves computing orbits under the group, finding the minimal point in each orbit, and being able to recognise quickly which orbit a point is in. To do this, we compute an array of length n which records for each point the number of the orbit it is in, or a special value indicating that that orbit has not been constructed yet. A second array records the smallest point in each orbit, which can be recorded while the orbit being constructed. Using this array, each orbit is constructed at most once, unneeded orbits are never constructed and the minimal element in the orbit containing a point can be identified in constant time.

Like any breadth-first search, the space required to store the search nodes (candidate records in this case) is a significant constraint. We first observe that each node uses only $O(k)$ space (not $O(n)$), as no permutations of Ω are stored in the candidate record. Additionally, the candidate records at each level are constructed and read sequentially, so they could be efficiently stored on secondary storage if necessary. Alternatively, as remarked, an iterative deepening approach could be used, finding the correct value of M_i using a search limited to depth i and then using that to prune a depth-first search to depth $i + 1$ to find m_{i+1} . This would be slower, especially in easy cases, since much work is repeated, but would use vastly less memory.

6. BENCHMARKS

Table 1: Results of Random Examples

n	m	k	T_1	T_2	no prune	prune
20	7	5	10s	34s	1009/524	1055/504
20	7	10	13s	10.5s	1314/857	1430/876
20	7	20	13s	9.5s	1306/1103	1255/1057
20	7	40	12s	10s	1690/1568	1781/1593
25	5	5	9s	58s	705/583	671/442
25	5	10	10s	19s	1119/823	1240/851
25	5	20	9s	9.5s	1342/1230	1605/1236
25	5	40	11s	8s	2533/2431	2818/2611
18	9	5	3s	14s	387/283	420/301
18	9	10	3s	4.5s	697/482	623/475
18	9	20	3.5s	3.5s	686/652	665/639
18	9	40	4s	3.5s	951/790	846/806

6.1 Random Examples

Given integers n , m and k , we take the action of S_n on m -sets (in a random order) and compute minimal images of random sets of k points in this action. It turns out that such sets almost never have non-trivial action of the set stabilizer on the set, so there is no point in using the advanced algorithm. We include here the times to compute the set stabilizer for comparison with this algorithm, and as an indication of the penalty that would be incurred by computing the set stabilizer in order to use the advanced algorithm.

We record in table 1 the average time T_1 to compute an initial stabilizer chain for S_n in this action, the average time T_2 to compute the stabilizer of a size k set Δ , and the average times for the first and second runs (the second run avoids the base change operations), displayed as *time for first run/time for second run* of the two variants of the algorithm:

The averages are taken over 16 runs with different randomly chosen Δ and different orderings of the collection of m -sets. Times are in milliseconds unless marked “s” for seconds and were measured on a 1GHz Pentium III using GAP 4.4.

6.2 A special set of points

In the vector space \mathbb{F}_3^4 there is just one orbit of maximal collections of vectors containing no three vectors summing to zero, each of which contains 20 vectors, (which can be taken as the non-zero norm zero vectors under a quadratic form of minus type). The natural group acting on this problem is the affine general linear group $AGL(4, 3)$, under which such a set of 20 vectors has set stabilizer of order 2880, acting faithfully on the set, but also of importance for the algorithm, there are significant subsets of the set with even larger stabilizers. Considering random sets in the orbit, but always taking the vector space in lexicographic order, we find that computing a stabilizer chain takes 60ms on average, the set stabilizer calculation 40ms, and the basic algorithm about 2.7 seconds (for a first or second run) without pruning, and 2.5 second with pruning. The advanced algorithm takes about 40 ms with or without pruning. Here the advanced algorithm is clearly superior, even if the set stabilizer is not known *a priori*.

Randomly varying the order of the underlying vector space typically prevents the basic algorithm without pruning completing in available memory (512MB). In these cases the basic algorithm with pruning takes on average 21s and the advanced algorithm about 210ms. The stabilizer chain and

set stabilizer times are unchanged.

7. CONCLUDING REMARKS

The algorithm works well in many cases. In the random examples used above, the basic algorithm without pruning seems the best choice. In particular computing the set stabilizer is a lot of work for little benefit in these cases. On the other hand, in a more structured case, but with permutations of much smaller degree, the advanced algorithm is enormously more effective.

The algorithms of this paper, although searching breadth-first rather than depth-first are essentially examples of the type of search described in [8, 7], making central use of a stabilizer chain to structure the search. They do not make use of the more sophisticated partition-based ideas found in [3]. In cases where there actually is a large set stabilizer, the advanced algorithm, which implicitly uses a partition refinement based computation of a set stabilizer works better. It is likely, however that there are cases which, although no large subgroup stabilizes the set, there are still excessive numbers of candidates at intermediate stages. It may be possible to incorporate partition refinement ideas directly into this algorithm to deal more effectively with such cases, but this remains to be explored in future work.

8. REFERENCES

- [1] P. Flener, A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetries in matrix models. In *Proc. 8th International Conference on Principles and Practice of Constraint Programming*, number 2470 in LNCS. Springer, 2002.
- [2] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.3*, 2002. (<http://www.gap-system.org>).
- [3] Jeffrey S. Leon. Permutation group algorithms based on partitions. I. Theory and algorithms. *J. Symbolic Comput.*, 12(4-5):533–583, 1991. Computational group theory, Part 2.
- [4] Jeffrey S. Leon. Partitions, refinements, and permutation group computation. In *Groups and computation, II (New Brunswick, NJ, 1995)*, volume 28 of *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, pages 123–158. Amer. Math. Soc., Providence, RI, 1997.
- [5] Eugene M. Luks. Permutation groups and polynomial-time computation. In *Groups and computation (New Brunswick, NJ, 1991)*, volume 11 of *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, pages 139–175. Amer. Math. Soc., Providence, RI, 1993.
- [6] Ákos Seress. *Permutation group algorithms*, volume 152 of *Cambridge Tracts in Mathematics*. Cambridge University Press, Cambridge, 2003.
- [7] Charles C. Sims. Computation with permutation groups. In *Proc. Second Symposium on Symbolic and Algebraic Manipulation*, pages 23–28. ACM Press, 1971.
- [8] Charles C. Sims. Determining the conjugacy classes of a permutation group. In *Computers in algebra and number theory (Proc. SIAM-AMS Sympos. Appl. Math., New York, 1970)*, pages 191–195. SIAM-AMS Proc., Vol. IV. Amer. Math. Soc., Providence, R.I., 1971.
- [9] Leonard H. Soicher. GRAPE: a system for computing with graphs and groups. In *Groups and computation (New Brunswick, NJ, 1991)*, volume 11 of *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, pages 287–291. Amer. Math. Soc., Providence, RI, 1993.
- [10] Leonard H. Soicher. On the structure and classification of SOMAs: generalizations of mutually orthogonal Latin squares. *Electron. J. Combin.*, 6(1):Research Paper 32, 15 pp. (electronic), 1999.