Chapter 4. Searching Small Groups

This chapter develops algorithms for searching small groups. Many algorithms in computational group theory involve searching a group for an element (or elements) with specific properties. We will look at the general principles involved in these searches, and illustrate them by finding elements that commute with (or centralize) a given element. While we consider these principles in the context of small groups, they do apply to large permutation groups as well.

We will consider elements that satisfy a given property P. Of course, we must be able to determine whether an element satisfies the property or not. If the element g satisfies the property P, we write $P(g)$.

We begin with algorithms that find one element with the specified property. Then we consider algorithms for finding all the elements with a specified property, in the special case where the set of all such elements forms a subgroup.

Linear Search of the List of Elements

Suppose we are searching a group G for an element that satisfies a given property P. For small groups we have a list of elements, so our search could just run through this list. This is the approach taken in Algorithm 1.

Algorithm 1 : Search List of Elements for One Element

Input : a list of elements of a group G;
a property P;

Output : an element g with property P, if one exists;

```
begin
  for each element g of G do
    if P(g) then
      exit with result( g );
    end if;
  end for;
end;
```

Let $c(P)$ be the (average) cost of determining whether an element satisfies property P. Then the worst case cost of Algorithm 1 is

$$|G| \times c(P).$$

For a concrete example, let z be a fixed element of the group G. Let the property P be

$$P : g \in G,\ g \notin \langle z \rangle,\ g \times z = z \times g.$$

The elements that satisfy P are those that commute with (or centralize) z. Assuming that no elements in the cyclic group $<z>$ are actually tested, the cost of determining the property is

2 multiplications, and

1 comparison of elements.

Searching as a Discarding Process

Let us view the linear search of the list of elements in another way. This alternate view allows us to generalize the algorithm, and improve the search. Instead of running through the list of elements, let us regard the search as a process for decreasing a set of possible solutions by discarding non-solutions. Initially, the whole group G will be the set of possible solutions. The process will end when one element in the set is found that has the specified property. With this view, Algorithm 1 for finding an element that commutes with z is transformed into Algortihm 2.

Algorithm 2 : Discarding elements to find one element

Input : a list of elements for a group G;
a property P;

Output : an element g with property P, if one exists;

```
begin
  Γ := G - <z>;
  while Γ is not empty do
    choose g ∈ Γ;
    if P(g) then
      exit with result(g);
    else
      Γ := Γ - {g};
    end if;
  end while;
end;
```

As noted in Chapter 2, an effective way to represent the subset Γ is as a bitstring of length $|G|$.

One approach to improving the search algorithm is to discard more than one element of the group at a time. This attempts to avoid tests for the property P. Hopefully, the cost of discarding the extra elements will be less than the cost of the tests avoided.

Just how much can we discard? Let $Ded_P(g)$ be a set of elements y of G for which we can "easily deduce" not $P(y)$ from not $P(g)$. By this we mean that the set can be constructed from the property P and the element g, without global knowledge of the group G, (that is, without running through the list of elements of G). For example, from not $P(g)$ in the case of commuting elements, we can easily deduce

not $P(g^{-1})$,
not $P(z \times g)$,
not $P(g \times z)$,
not $P(z^m \times g)$, for any integer m,
not $P(g \times z^m)$, for any integer m, and
not $P(z^m \times g \times z^n)$, for any integers n and m.

but we could not easily deduce

not $P(y)$, for $y^m = g$, for some integer m

even though it is deducible from not $P(g)$, since the only practical way of finding such y's would be to run through the group G taking powers.

The set $Ded_P(g)$ is in some sense the largest set of elements we could discard from the set of possible solutions. In practice, we require a subset $D_P(g)$ of $Ded_P(g)$ that is cheap to compute - at least more cheaply than performing the tests. Typically a coset of some subgroup is used as the subset. For our particular property P, one could use $D_P(g) = <z> \times g$ as the discard subset. The cost of forming a coset is one multiplication per element of the coset (and in this instance, one search per element to create the subset as a bitstring). This is the same as the cost of one test per element, (under our usual assumptions about the cost of a search).

The search algorithm is now Algorithm 3.

Algorithm 3 : Discarding cosets to find one element

Input : a list of elements of a group G;
a property P;

Output : an element g with property P, if one exists;

begin
$\Gamma := G - <z>$;
while Γ is not empty **do**
choose $g \in \Gamma$;
if $P(g)$ **then**
exit with result(g);
else
$D_P(g) := <z> \times g$; $\Gamma := \Gamma - D_P(g)$;
end if;
end while;
end;

An analysis of the worst case of Algorithm 3 follows. The worst case occurs when no element satisfies the property. In this case, all the elements of the group are eventually discarded from the set of possible solutions. The cost of forming $D_P(g)$ as a bitstring is $|z|-1$ multiplications and searches. That is, one multiplication and search for each element in the set that was not tested. The number of elements tested is $|G|/|z|$, with the remaining elements being discarded as part of $D_P(g) - \{g\}$, for some g. The cost of each test is one comparison and two multiplications. Thus the worst case cost is

$\frac{|G|}{|z|}$ comparisons,

$\frac{|G|}{|z|} \times 2 + \left\lceil |G| - \frac{|G|}{|z|} \right\rceil$ multiplications, and

$|G| - \frac{|G|}{|z|}$ searches.

Taking a comparison to be equivalent to a multiplication, and a search to be equivalent to two comparisons, gives a total worst case cost of

$3 \times |G|$ multiplications.

This is also the worst case cost of Algorithm 1, because for this example the cost of discarding an element is the same as performing a test of the property.

Another Example

An example where the cost of discarding an element is less than the cost of performing a test of the property occurs when one is searching for an element g that normalizes a given subgroup U. Let S be a set of generators for U. Then the property is

$$P : g \in G,\ g \notin U,\ g^{-1} \times s \times g \in U, \text{ for } each\ s \in S.$$

To test the property requires $|S| \times 2$ multiplications and $|S|$ searches. (This assumes the inverse of the element g is known, or that the cost of a "division" of permutations is the same as a multiplication.) We can take the discard set $D_P(g)$ to be the coset $U \times g$. Then the cost of discarding an element is one multiplication and one search.

In the worst case where no element normalizes U, there are $|G|/|U|$ tests, and the remaining elements are discarded. This gives a cost of

$$2 \times |S| \times \frac{|G|}{|U|} + \left\lceil |G| - \frac{|G|}{|U|} \right\rceil \text{ multiplications, and}$$

$$|S| \times \frac{|G|}{|U|} + \left\lceil |G| - \frac{|G|}{|U|} \right\rceil \text{ searches.}$$

This gives a total equivalent to

$$\frac{|G|}{|U|} \times \left[4 \times |S| + 3 \times |U| - 3\right]$$

multiplications.

When the subgroup U has two generators, this represents a saving of $5 \times \left\lceil 1 - \frac{1}{|U|} \right\rceil \times |G|$ multiplications over Algorithm 1, which requires $4 \times |S| \times |G|$ multiplications.

Finding All Elements

Often the purpose of a search is to find all the elements with a specific property knowing that these elements form a subgroup. For example, the set of elements that commute with a given element z forms a subgroup called the centralizer of z in G. In this section, we look at adapting our search algorithm for one element to find all elements.

The search for a single element attempts to discard as many elements as possible from the set of possible solutions. However, it only does this for elements it knows do not have the specified property. Once it finds an element with the property it stops. The search for all elements has the possibility of discarding elements that it has already included in the subgroup. Thus the set of possible solutions is reduced in two ways.

Let H be the set of elements found so far to have property P. The set will be a subgroup. Let $D_P(g)$ be the discard set for the element g that does not satisfy P. Then our algorithm is Algorithm 4.

Algorithm 4 : Subgroup of elements with a given property

Input : a list of elements of a group G;
a property P such that the elements satisfying P form a subgroup;

Output : the subgroup H of G of elements that satisfy the property;

```
begin
  H := { id }; Γ := G - H;
  while Γ is not empty do
    choose g ∈ Γ;
    if P(g) then (* add to H those with the property *)
      H := < H, g >;  Γ := Γ - H;
    else (* discard others without the property *)
      Γ := Γ - D_P(g);
    end if;
  end while;
end;
```

Since the set of all elements satisfying the property P forms a subgroup, it follows from $P(g)$ that all elements of $< H, g >$ satisfy P. Hence they are added to H and discarded from Γ. Forming the new subgroup H is just the inductive step of Dimino's algorithm modified to work with bitstrings.

Furthermore, we may take, for any such property P, the coset $H \times g$ as the discard set $D_P(g)$. This is so because all the elements of H satisfy P while g does not satisfy P.

Analysis

The analysis of Algorithm 4 is not straightforward, and is still incomplete. We will cover the initial ground work in this section to provide a crude upper bound on the cost. In the next section we provide a more detailed analysis.

The algorithm, in essence, finds generators

$$id = s_0, s_1, s_2, \ldots, s_t$$

for the final subgroup H. The values taken by the variable H in the algorithm are $H_i = < s_1, s_2, \ldots, s_i >$, for i from 0 to t. Let us first consider the number of tests performed during the algorithm. Suppose the elements tested were the sequence $\{g_i\}$, and that i_k elements were tested between the finding of s_{k-1} and s_k. For convenience, let g_0 be the identity, and let i_{t+1} be one more than the number of tests performed after finding the last generator. Then

$$\begin{aligned} g_{i_1} &= s_1, \\ g_{i_1 + i_2} &= s_2, \\ &\vdots \\ g\left[\sum_{j=1}^{k} i_j\right] &= s_k, \\ &\vdots \end{aligned}$$

While the value of H is H_k we form $i_{k+1} - 1$ discard sets, and perform at most one inductive step of Dimino's algorithm. If $D_P(g)$ is the coset $H \times g$ then the cost of forming the discard set is one multiplication and one search per element (different from g). Let the constant c denote this cost per element. Then the total cost of the discards is

$$\sum_{k=0}^{t} \left[c \times \left[i_{k+1} - 1 \right] \times |H_k| \right].$$

The total effort of forming the subgroups is equivalent to the complete Dimino's algorithm for $H = < s_1, s_2, \ldots, s_t >$. This cost was analysed in the third chapter. The adjustments to that analysis due to the use of bitstrings are minor. It is just the additional cost of one search per element of the final subgroup H_t in order to obtain their positions in the list of elements of G.

Let us obtain an upper bound for the values i_k. This will give us an upper bound on the cost of the algorithm.

The value of $i_{t+1} - 1$ is bounded by the number of cosets of H_t in G minus one, since each discard set is a coset of H_t distinct from H_t.

For an upper bound on i_k, recall that the value of H will change once we find any element of the final subgroup H_t. There are $|G|/|H_{k-1}|$ cosets of H_{k-1} that could possibly be considered for discarding, but of these $|H_t|/|H_{k-1}|$ give an element that satisfies the property. Thus $i_k - 1$, the number of cosets discarded, is bounded by

$$\frac{|G|-|H_t|}{|H_{k-1}|}.$$

Thus

$$c \times \left[i_{k+1}-1\right] \times |H_k| \quad \leq \quad c \times \left[|G| - |H_t|\right]$$

and the cost of discarding is bounded by

$$c \times (t+1) \times \left[|G| - |H_t|\right].$$

Multiple Discarding

The above upper bound assumes that the values of i_k are independent of each other. This is not the case, so we should expect to be able to refine the bound.

The above upper bound says that each element not in the subgroup is discarded the maximum number of times; that being $(t+1)$. While this figure is not exact, multiple discarding of elements does occur. There are $|H_k{:}H_{k-1}|$ cosets of H_{k-1} in one coset of H_k, and many of these could be discarded while the value of the variable H is H_{k-1} without all of them being discarded. Therefore, there is the possibility of locating an element that causes the coset of H_k to be discarded. This may discard some cosets of H_{k-1} again. Hence the occurrence of multiple discarding.

There are $|G|-|H_t|$ elements without the property P, and $(i_k-1) \times |H_{k-1}|$ of these have been discarded in cosets of H_{k-1}. Thus there are

$$|G|-|H_t| - \left[i_k-1\right] \times |H_{k-1}|$$

elements without the property that could be selected as the element g in the algorithm.

There are

$$\frac{|G|-|H_t|}{|H_k|}$$

cosets of H_k containing elements without the property. It only requires one element of a coset to be still in the set Γ and the whole coset could be discarded. So we are only guaranteed that a complete coset of H_k has been discarded as a union of cosets of H_{k-1} if the number of elements still in Γ is less than the number of cosets of H_k without the property P. Hence, the number of complete cosets of H_k discarded as cosets of H_{k-1} is at least

$$M = \max\left[0, \frac{|G|-|H_t|}{|H_k|} - \left[|G|-|H_t|-\left[i_{k-1}-1\right] \times |H_{k-1}|\right]\right].$$

Thus $i_{k+1}-1$ is bounded by

$$\frac{|G|-|H_t|}{|H_k|} - M \;=\; \min\left[\frac{|G|-|H_t|}{|H_k|},\ |G|-|H_t|-\left[i_k-1\right] \times |H_{k-1}|\right].$$

Let

$$j_k = c \times \left\lfloor i_{k+1} - 1 \right\rfloor \times |H_k|, \text{ and}$$
$$K = c \times \left\lfloor \frac{|G|}{|H_t|} \right\rfloor$$

then, for the worst case analysis, we wish to maximise $\sum_{k=0}^{t} j_k$ subject to the constraints

$$j_{-1} = 0, \text{ and}$$

$$0 \le j_k \le \min \left\lfloor K, \left\lfloor K - j_{k-1} \right\rfloor \times |H_k| \right\rfloor, \; k = 0,1,\ldots,t.$$

A solution to this problem has not been found to date.

Summary

Searching for subgroups and elements is an integral part of many other algorithms, as well as being essential in the study of groups. Viewing it as a discarding process is a good paradigm for improvements, and for analogy with search algorithms for large permutation groups.

Typically the search algorithms discard cosets (of some subgroup). The analysis of these algorithms is not straightforward and multiple discarding is not yet fully analysed. Other choices of discard sets have rarely been investigated, and their analysis is more incomplete.

Exercises

(1/Easy) In the symmetric group of degree 4, execute Algorithm 3 to find a centralizing element of z=(1,2,3,4).

(2/Easy) In the symmetric group of degree 4, execute Algorithm 4 to construct the normalizer of $H = < (1,2,3,4) >$. Use $D_P(g) = H \times g$. Do the same for $H = < (1,2)(3,4), (1,3)(2,4) >$.

(3/Very Difficult) Extend the analysis of multiple discarding to account for all the previous values of H.

Bibliographical Remarks

The search algorithm was developed to compute the normalizer of a subgroup. This information was required by the lattice of subgroups algorithm (in Chapter 6). Algorithms to compute the normalizer in this context have been suggested and implemented by various people. The references are V. Felsch and J. Neubüser, "*Ein Programm zur Berechnung des Untergruppenverbandes einer endlichen Gruppe*", Mitteilungen des Rheinish-Westfälischen Institutes für Instrumentelle Mathematik (Bonn) **2** (1963) 39-74; V. Felsch, **Ein Programm zur Berechnung des Untergruppenverbandes und der Automorphismengruppe einer endlichen Gruppe**, Diplomarbeit, Kiel, 1963; J. J. Cannon, **Computation in Finite Algebraic Structures**, Ph. D. Thesis, Sydney, 1969; and J.J. Cannon and J. Richardson, "*The GROUP system for investigating the structure of groups*", Technical Report 8, Computer-Aided Mathematics Project, Department of Pure Mathematics, University of Sydney, 1971. The Felsch algorithm is essentially the algorithm of this chapter. It differs in that it retains information about the representatives of the cosets discarded. The author first saw Felsch's algorithm towards the end of 1975 in the implementation of the subgroup lattice algorithm in

GROUP at that time. The exposition of this chapter is a simplified presentation of that algorithm.

Algorithm 4 is suggested in P. Dreyer, **Ein Programm zur Berechnung der auflösbaren Untergruppen von Permutationsgruppen**, Diplomarbeit, Kiel, 1970, however, Dreyer implemented the Felsch/Neubüser algorithm.

Another choice of discard set when searching for a subgroup, namely the double coset $H \times g \times H$, is investigated in G. Butler, "*Double cosets and searching small groups*", **SYMSAC'81** (Proceedings of 1981 ACM Symposium on Symbolic and Algebraic Computation, Snowbird, Utah, August 5-7, 1981), (P. Wang, ed.), ACM, 1981, 182-187. In general the cost of computing double cosets is too expensive. This paper is the initial source of our analysis. In particular, it introduces the study of multiple discarding.