11. Algorithms for groups

The following section is loosely based on the study of algorithms for groups as in the work of Schoup, Schnorr, Buchmann and others. How does one specify a group to a computer? First of all the elements of the group are represented by "words" of some fixed length (bitstrings of a fixed length). Let us denote the set of such words by F. However, since the group does not have order a power of two, not all such words will be elements of the group. Thus we have a membership function $\mu : F \to \{0, 1\}$ that takes the value 1 for elements of the group. We then have the group operations which usually "expand" the word length. In other words we have a superset F^2 containing F which has "double words" (words of twice the size of those in F) and the group multiplication operation usually gives $m : F \times F \to F^2$. There is usually also an inverse operation $i : F^2 \to F^2$ which takes F to itself; more often a "ratio" operation $d : F \times F \to F^2$ is provided instead. Finally, there is a reduction operation $r : F^2 \to F$. The actual group operations are defined by

$$\begin{aligned} x \cdot y &= r(m(x, y)) \\ x \cdot y^{-1} &= r(d(x, y)) \\ x^{-1} &= r(i(x, y)) \end{aligned}$$

This specification should satisfy the condition that these operations, when restricted to the set $G = \mu^{-1}(1)$ satisfy the axioms for an abelian group. We will assume that such a collection of operations has indeed been provided to us as a "black box". We will look at algorithms that study the properties of this group *without* looking into the details of how the maps μ, m, d and r are defined. Some have called this the study of "generic group algorithms".

One way to represent a finite abelian group G and compute its structure is to write it via generators and relations. Suppose we are given a collection $g_1, g_2, \ldots, g_r$ of generators of G and we can find a collection $\rho_1, \rho_2, \ldots, \rho_s$ of relations of the form

$$\rho_j = \prod_i g_i^{a_{ij}}$$

If this collection of relations is sufficient, we obtain an exact sequence

$$\mathbb{Z}^s \xrightarrow{A} \mathbb{Z}^r \xrightarrow{g} G \to 0$$

where A is given by the matrix (a_{ij}) and g the "vector" of generators of G. Such a description reduces the computation of the *abstract* structure of G to matrix manipulations. For example, by reducing the matrix to echelon form we can exhibit an isomorphism *from* a product of cyclic groups *to* the group G. However, it is not so easy(!) to write the inverse of this isomorphism. To do that one must exhibit a (computable) set-theoretic splitting $G \to \mathbb{Z}^r$. One way to compute such a splitting on an element g of G is to find enough relations between g and the collection of the g_i's. It is thus clear that the important problem in the algorithmic analysis of finite abelian groups is that of finding (sufficiently many) relations between elements of the group.

There are essentially three classes of "generic group algorithms" which can be called the Pollard ρ method, Shanks' Baby Step Giant Step method and the Pohlig-Hellman factor method. We illustrate these methods by seeing how they can be used to find a relation between some given elements $g_1, g_2, \ldots, g_r$ of the group.

11.1. **Pollard's ρ.** This method uses the least amount of information about the group and also uses the least amount of storage. We choose a "random" partition of the elements of the group into disjoint sets S_0, S_1, ..., S_r (which are specified by their membership functions μ_i). We now define a collection of maps as follows. Start with a "random" element x_0 of the group. Let $v_0 = (0, \ldots, 0)$ be the origin in the group $\mathbb{Z}^r$. We define an iterated map as follows

$$(x_{i+1}, v_{i+1}) = \Phi(x_i, v_i) = \begin{cases} (x_i^2, 2 \cdot v_i) & x_i \in S_0 \\ (g_j \cdot x_i, e_j + v_i) & x_i \in S_j \end{cases}$$

By the techniques due to Pollard, Brent and Floyd described earlier we can find an integer T such that $(x_{2T}, v_{2T}) = (x_T, v_T)$. It follows that if $v_{2T} - v_T = (a_1, a_2, \ldots, a_r)$, then

$$\prod_i g_i^{a_i} = 1$$

Some aspects of Brent's method can be used to further ensure that v_i does not grow too large (since we are only interested in $v_{2T} - v_T$). Assuming that the choice of S_i is "random" enough we should be close to a relation of the smallest size in the sense that $\sum_i a_i$ is the shortest. That is also an estimate of the number of steps upto a small constant factor.

11.2. **Shanks' Baby step-Giant step.** Now suppose that we know that we are given bounds L_1, L_2, ... L_r with the assurance that there is a relation $\prod_i q_i^{a_i}$ with $0 \leq a_i < L_i$. (We will see below how such a collection of bounds can be constructed inductively given a bound L on the order of G).

Let $h(x)$ be a collision free function for $x \in G$ (for example h is a "hash function"). Let $(a_1, \ldots, a_r)$ be a sequence with a_i an integer not larger than $n_i = \lceil \sqrt{L_i} \rceil$. We compute the terms

$$(\prod_i g_i^{a_i}; a_1, \ldots, a_r; h(\prod_i g_i^{a_i}))$$

for all a_i in this range and store them sorted according the final entry. This allows us to perform a search operation in a time proportional to the logarithm of the length of the list. Now, for each sequence $(b_1, \ldots, b_r)$ where b_i is an integer not larger than $\lfloor \sqrt{L_i} \rfloor + 1$, we compute the expression $h(\prod_i (g_i^{-n_i})^{b_i})$ and try to find it in the given sorted list. Clearly, if it is found then we have a relation of the form

$$\prod_i g_i^{a_i + n_i b_i} = 1$$

By the given assertion on L_i, such relation will eventually be found.

One way to approach this algorithm given only a bound on the size of the group G is to go inductively. First find a relation involving g_1 alone (in other words find the order of g_1) by using $L_1 = L$. For the remainder of the algorithm we put $L_1 = o(g_1)$ as found in this step. In the second iteration we work with g_1 and g_2 with $L_2 = L/L_1$. Iteratively, we would have computed L_i which is the order of g_i in the group $G/ < g_1, \ldots, g_{i-1} >$. We now work with g_1, ..., g_{i+1} and take $L_{i+1} = L/(\prod_{k=1}^{i} L_i)$. Using the above algorithm we obtain the order of g_{i+1} in the group $G/ < g_1, \ldots, g_i >$. For the remainder of the iterations we take L_{i+1} to be this order. Thus, at the end we would have found a *minimal* relation among the g_i's rather than just one relation.

11.3. **Pohlig-Helman factor method.** We now assume that we are given the factorisation of the order n of the group G and we want to find another relation which is smaller than the obvious relation $(n, \ldots, n)$.

Let p be a prime factor of n and k such that $p^k | n$ while $p^{k+1} \not| n$. Replacing g_i by g_i^{n/p^k}, we have effectively replaced the given problem to the case of $G(p)$, the p-adic part of G. By the Chinese Remainder Theorem these solutions can later be combined. Thus we may assume that the order of G is a power p^k of a prime p.

In the case when $k = 1$ we can apply the Baby step-Giant step method to find a relation as above. When $k > 1$, we inductively find a relation $(b_1, \ldots, b_r)$ between the elements g_i^p which lie in the group G^p which has order p^j for some $j < k$. Now, we only need to find (again using the Baby step-Giant step method) a sequence $(a_1, \ldots, a_r)$ with $a_i < p$ such that

$$\prod_i g_i^{a_i} \cdot \prod_i g_i^{pb_i} = 1$$

Note that the same sorted list can be used in each step of this induction.

This algorithm can also be applied in the case when we do not know a factorisation of n but we have some given finite set of "small" primes which are known to factorise n completely.

11.4. **Other problems.** Once sufficiently many relations between a collection of generators of the group G has been found, we have seen above that one can "read off" many of the properties of G such as its isomorphism class. The above algorithms for finding relations can be applied directly to solving other problems or questions regarding the group G.

To find the order of the group G we use the above techniques to find (in succession) the order n_i of a randomly chosen element g_i in the group $G/ < g_1, \ldots, g_{i-1} >$. This probabilistically determines the order of G as the product of the n_i.

Given h and g in G and the fact that h is a power of g we determine this power (the Discrete Log problem) by finding a minimal relation between g and h.

11.5. **Applicability and efficiency.** The fundamental theorem for finite abelian groups states that every abelian group is isomorphic to product of cyclic groups. We have seen that the all-pervading question in algorithmic abelian group theory is "To what extent can we (efficiently) compute this isomorphism and its inverse?" Because of the efficiency aspect it is important to count the number of steps and the amount of storage that our algorithms require. In such measurement numbers of the size of the cardinality of F are considered "large", numbers of the size of the number of bits in the *elements* of F are considered "reasonable" and numbers of the size of the logarithm of the latter number are considered "small" or insignificant.

One can show that the expected running time of the algorithms described above is of the same order as the bounds available. Since these bounds are crudely measured as the order of the group G this makes the algorithms "slow". However, in a given case where not enough information is available or it is suspected that the elements g_i generate a much smaller group one can start with artificially chosen small bounds and run the algorithms; if the assumptions (on the "real" bounds) are valid these will run to completion.

11.6. **Index calculus.** Another useful tool goes by the name "Index Calculus". Suppose that we are given a set of generators $g_1, \ldots, g_r$ of G and *another* "black-box" which computes a function $A : G \times R \to \mathbb{Z}^r$; here R is used to denote random input. The property of A is that given (g, k) (with k random) the output is an r-tuple $(a_1, \ldots, a_r)$, such that with probability greater than $1 - (1/2)^{r+t}$ (for some small integer t) we have the relation

$$g^k = \prod_i g_i^{a_i}$$

(Note that one can quickly check whether this is indeed true). We can use such a black-box A to give a probabilistic solution to the group structure problem. By running the algorithm at most $r + t$ times we will (with high probability) obtain sufficiently many relations to write g in terms of the elements g_i. Thus we can use this to construct a probabilistic algorithm that splits the map $\mathbb{Z}^r \to G$ as required.

Since this algorithm makes use of one additional "black-box" it is not a "generic group algorithm".