G. Butler

Fundamental Algorithms for Permutation Groups

Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo
Hong Kong Barcelona
Budapest

Series Editors

Gerhard Goos
Universität Karlsruhe
Postfach 69 80
Vincenz-Priessnitz-Straße 1
W-7500 Karlsruhe, FRG

Juris Hartmanis
Department of Computer Science
Cornell University
5148 Upson Hall
Ithaca, NY 14853, USA

Author

Gregory Butler
CICMA
Department of Computer Science, Concordia University
1455 de Maisonneuve Blvd. West
Montreal, Quebec H3G 1M8, Canada

CR Subject Classification (1991): I.1.2, F.2.2, G.2.1-2

ISBN 3-540-54955-2 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-54955-2 Springer-Verlag New York Berlin Heidelberg

Typesetting: Camera ready by author
Printing and binding: Druckhaus Beltz, Hemsbach/Bergstr.
45/3140-543210 - Printed on acid-free paper

Preface

These notes are derived from an 18-hour lecture course on Symbolic and Algebraic Computation that I have been giving to Honours computer science students at the University of Sydney since 1983, and lectures to mathematics and computer science students at the University of Bayreuth in 1990. Due to the very wide scope of the field, I concentrate on my area of speciality — algorithms for permutation groups. The (short) course at Sydney emphasises Chapters 2, 3, 4, 7, 10, and 11.

The aim of the course is to develop each of the algorithms from scratch, showing how each new piece of group-theoretical information can be used to improve the algorithm. As such, it could be regarded as a course in algorithm development. I assume no background in group theory, so the piecewise introduction of information allows students to become familiar with one concept at a time. Another advantage of this approach is that the correctness of the algorithm is justified as it is developed.

The examples and the exercises assist students to learn the group theory and the working of the algorithms. Occasionally, they show how the algorithms could be further improved, or they develop an alternate algorithm. Access to the computer algebra system Cayley would be beneficial. Students could then easily study other examples and implement the algorithms.

The bibliographical remarks explain the history of the algorithms, and often place them in the broader context.

The algorithms are presented using Pascal control structures with some exceptions. We use the **for**-loop of Pascal to run over sets, even sets that increase during the loop's execution. The meaning of such a loop is that each set member is to be considered precisely once, including those members added during the loop's execution. The order in which they are considered is generally unimportant, but for precision, assume it is the same as the order in which they were added to the set. We do not use **begin** ... **end** to form compound statements. Instead all the control statements have a corresponding closing bracket such as **end if**, **end for**, and **end while** to indicate the extent of the statement. In addition, we occasionally use an **exit**-statement to terminate an algorithm, a **return**-statement to terminate a procedure, and a **result**-statement to terminate a function. The algorithms also use English and mathematical statements and operators. (* Comments in the algorithms are enclosed like this sentence. *)

Some general assumptions are made during the analysis of algorithms. In Chapters 3-6, the analysis of the algorithms is in terms of element multiplications and searches of lists of elements. To this point, the analyses are independent of the element representation. However, we sometimes obtain a single total cost based on multiplications that assumes the elements are permutations. In this case a comparison of elements costs the same as a multiplication, and we assume a (hash) search requires at most two comparisons. In Chapters 7-14, the analyses are concerned with the accesses made to permutations, sets, and Schreier vectors. In essence, we assume that the simple variables of an algorithm will be stored in registers or very fast memory, and therefore the time to access them will be negligible. The cost of accessing one entry, or storing the value in one entry, of a permutation, set, or Schreier vector will be counted as one operation. We are viewing these as one read or write to memory. In any case, these accesses and stores will constitute the bulk of the cost of an algorithm.

The first part of the book, comprising Chapters 2-6, is an introduction to computational group theory for those without a knowledge of group theory. Even so, some mathematical sophistication is a distinct advantage. For example, familiarity with geometry, manipulation of algebraic formulae, and straightforward analysis of algorithms is desirable. Those with a knowledge of group theory, or even a knowledge of computational group theory will find something of interest. We have attempted to develop the algorithms from first principles showing where the particular pieces of group theoretical knowledge have influenced the algorithm, and how they can be improved by using this knowledge, until we reach the state of the art. We have attempted to justify (prove?) the algorithms as we develop them, and to analyse their time and space usage. For many of the algorithms a complete analysis is not known, so we have fallen short on our last goal.

This part is really a slow introduction to group theory and uses of the elementary concepts of group theory in algorithms that handle small groups — that is, groups for which we can store a list of all their elements. It serves also as a first exposure to some techniques that are relevant to large permutation groups, and of course, as a source of information about algorithms for small groups. The problems tackled in this part are: determining a list of elements from a generating set; searching a group for elements with a given property; determining defining relations from a Cayley graph; and determining the lattice of all subgroups.

The first two problems cover the essential fundamental algorithms. The third problem introduces the Cayley graph, which is a useful representation of a group, and provides a link with combinatorial group theory and its algorithms. The last problem, besides being historically significant, leads to a beautifully subtle algorithm that effectively tackles what at first sight appears an impossibly large task.

All our examples are groups of permutations. The analysis of the algorithms is in terms of element multiplications and searches of lists of elements. To this point, the analyses are independent of the element representation. However, we sometimes obtain a single total cost based on multiplications that assumes the elements are permutations. In this case a comparison of elements costs the same as a multiplication, and we assume a (hash) search requires at most two comparisons.

The second part of the book comprises Chapters 7-14 and considers algorithms specific to permutation groups. One benefit of restricting to permutation groups is that the algorithms can handle very large groups, say of order 10^{20}, and of degree up to 10 000. For some problems there are specialist techniques to handle groups of degree up to 10^5, but we will not discuss those here.

The power of these algorithms comes from intimate use of how the permutations act on the points. Think of it as an attempt to make the complexity of the algorithms a function of the degree rather than a function of the group order. The action on points can often be conveniently represented by orbits and Schreier vectors. We will discuss these first, and then present two elementary uses of the information contained in the orbits and Schreier vectors. The two uses are determining whether a group is regular (that is, whether any element of the group fixes a point), and whether a group is imprimitive (that is, whether the group leaves invariant a (non-trivial) partition of the points). In the end, both these uses simply use the information provided by the generators of the group.

A chain of subgroups, where each subgroup fixes at least one more point than the previous one, provides the inductive basis for the remaining algorithms. Such a chain is called a stabiliser chain. Associated with a stabiliser chain is a base and strong generating set — *strong* because it contains generators for each subgroup in the chain, and not just for the whole group. We can represent all the elements of the group, and effectively compute with them, by a base, a strong generating set, and a Schreier vector for each subgroup in the stabiliser chain.

After introducing the concepts of stabiliser chain, base, and strong generating set, we will discuss at length how elements can be represented, and how we could, if necessary, generate all the elements of the group. We delay discussing how one determines a base and strong generating set for a permutation group until the reader has had more experience with the concepts.

A major aim of this part is to discuss searching very large permutation groups. A backtrack algorithm is used. The searching of permutation groups provides a good forum for discussing the various techniques for improving the heuristics of a backtrack algorithm. Several improvements are applicable to all backtrack searches of permutation groups, while others, even though the details rely on what we are searching for, demonstrate the universality of choosing a base appropriate to the problem, and of preprocessing information that is repeatedly required during the search. The improvements applicable to all backtrack searches of permutation groups are just another form of discarding the elements in a coset. We have met this strategy already when searching small groups.

After choosing an appropriate base for the search, we require a corresponding strong generating set. The base change algorithm will provide one from an existing base and strong generating set. This algorithm is presented.

Now we can painlessly present an algorithm that determines a base and strong generating set of a permutation group from a set of generators. We present the Schreier-Sims algorithm. There are many variations of this algorithm. They are generally called Schreier methods or Schreier-Sims methods. We will mention a few variations in passing.

The analyses of this part make some general assumptions. They are concerned with the accesses made to permutations, sets, and Schreier vectors. In essence, we assume that the simple variables of an algorithm will be stored in registers or very fast memory, and therefore the time to access them will be negligible. The cost of accessing one entry, or storing the value in one entry, of a permutation, set, or Schreier vector will be counted as one operation. We are viewing these as one read or write to memory. In any case, these accesses and stores will constitute the bulk of the cost of an algorithm.

The third part of the book comprising Chapters 15-18 looks at the role of homomorphisms in algorithms for permutation groups. In particular, Chapter 16 discusses the computation of Sylow subgroups by using homomorphisms; and Chapter 18 discusses the conversion from a permutation representation to a power-commutator presentation when the group is a p-group or a soluble group, so that these special cases can utilise the efficient algorithms for soluble groups and p-groups. Chapter 17 introduces the notion of a power-commutator presentation and some elementary algorithms based on that representation.

At this stage, the reader will have met the fundamental algorithms for handling permutation groups.

The last chapter briefly discusses what has been omitted from this book and gives pointers to the relevant literature.

I would like to thank all my students. They were the guinea pigs as I experimented with this approach, often pointing out errors and the need for further clarification. In particular, Jowmee Foo and Peter Merel carefully read the first draft, and Bernd Schmalz made detailed notes of the lectures at Bayreuth. Volkmar Felsch helped enormously in clarifying some historical aspects. John Cannon, Volkmar Felsch, Mike Newman, Jim Richardson, and Charles Sims have made numerous comments that have greatly improved the text. They have been conscientious, thorough, and detailed. I thank them all, and accept responsibility for any remaining errors or shortcomings.

Montreal
September 1991

Greg Butler

Contents