THE STATE OF THE ART IN COMPUTATIONAL ALGEBRA

Steve Linton

Division of Computer Science, St. Andrews

July 17, 1997

Overview

- Computation and algebra – generalities
- A brief survey of what can be done
- Systems – GAP and MAGMA
- Current hot topics
- An Example – Recognizing $SL_n(q)$ *or* transformation semigroups

GENERALITIES

Why Compute?

- Exploration – to form or dismiss conjectures or gain insight
- Resolve finite problems – sporadic structures, small cases left over by general theorems
- Teaching – allow students to work out non-trivial examples
- As a research area in its own right – algorithms, algorithmics
- As a testbed for Computer Science – type systems, languages, database concepts

Approaching Problems in Computational Algebra

- Charlie Sims, some years ago, divided problems into three classes

  I Problems where many group elements fit into memory

  II Problems where just a few group elements fit

  III Problems where only a fraction of one element fits

- I would generalise this and speak of:

  I Problems that can be solved quickly by entirely automatic means

  II Problems that can be solved using standard low-level tools and some mathematical ingenuity

  III Problems that require special-purpose programming

- These boundaries are both fuzzy and moving
- Frustration arises when you tackle a problem in one class using tools suitable for another

WHAT CAN BE DONE

The Reds and the Greens

- Two schools in CGT (especially pre 1991)
- Green school (Sims, Europeans, Australians, mainly mathematicians)
  - Pragmatists
  - Implementation is important
  - "Good" algorithm performs well on standard or big examples
- Red school (North America, mainly computer scientists)
  - Complexity theorists – started from Graph Isomorphism problem
  - Precise complexity calculation important
  - "Good" algorithm has low asymptotic complexity
- Some fusion since 1991 – Seress, Cooperman
- Result – better theory and better implementations

Finitely Presented Groups

- Most problems unsolvable in general
- Coset enumeration can prove finiteness, or that a subgroup has finite index, up to index 100000 in class I in many cases

  Gives permutation representation on cosets and presentation (large) for subgroup
- Examine quotients – abelian, p, finite, nilpotent, polycyclic, isomorphic to given finite group – `quotpic`
- Can seek confluent rewriting system or automatic structure
- Collection of tricks for proving infiniteness (class II)

Permutation Groups

- Very efficient powerful (and complicated) methods
- Many methods are random, Monte Carlo or Las Vegas
- In nearly-linear Monte Carlo time, given generators, can determine:
  - Size of a permutation group
  - membership test
  - elements uniformly at random
  - composition series
  - Sylow subgroups
- Good performance in practice is achieved for
  - Centralizer
  - Conjugacy of elements
  - Set stabilizer
  - Intersection of subgroups
  - Normalizer
  - Conjugacy of subgroups
- Still hard: double cosets, conjugacy classes

Base and Strong Generating Set

- C.C. Sims
- Given $G \leq S_n$, let $\Omega = \{1, \ldots, n\}$
- Let $B = [b_1, \ldots, b_k]$ be a sequence of elements of Ω
- Define $G_0 = G$, $G_i = Stab_{G_{i-1}}(b_i)$
- B is a *base* for G if $G_k = \{()\}$
- $g \in G$ is characterised by
$$B^g = [b_1^g, \ldots, b_k^g]$$
- $$|G| = \Pi_{i=1}^k |G_{i-1} : G_i| = \Pi_{i=1}^k |b_i^{G_{i-1}}|$$
- $S \subseteq G$ is a strong generating set for G w.r.t. B if
$$\langle S \cap G_i \rangle = G_i$$
- Can write any $g \in G$ efficiently as a reasonably short word in a SGS

Random Algorithms

- Some algorithms make random choices as they run
- Las Vegas algorithms may (if unlucky) fail, but never give a wrong answer
- Monte Carlo algorithms give the wrong answer with some probability $\epsilon < 1/2$. Repeated runs will reduce ϵ as required
- To make a MC algorithm LV simply need a check on the final answer
- MC algorithms for permutation groups are mostly LV once a Base and Strong Generating Set has been found

Finite Soluble Groups

- Basic tool here is a power-commutator presentation:

  $$G = \langle a_1, \dots a_k | a_i^{m_i} = w_i, [a_i, a_j] = w_{ij} \rangle$$

  where each w is in normal form $a_1^{e_1} \dots a_k^{e_k}$
- With one of these, any word can be put into normal form
- If the presentation is nice, then the normal form is unique
- Can get such a presentation from permutation representation, or from another finite presentation (sometimes)
- Some choice of pc-generators, "special" systems are extra nice (Eick, Leedham-Green)
- Many questions about such groups can be answered by working down (or up) a normal or characteristic series and doing linear algebra in each factor
- These and related techniques have also been used to list all groups of order up to 1000 (except 512, 768).
- nilpotent groups (esp. p-groups) are extra nice – very large groups can be studied (Newman, Vaughan-Lee, Sims) – motivated by the Burnside problem

Matrix Representations

- Input: generating matrices for an algebra acting over a finite field
- Questions about irreducibility, indecomposability, submodule structure, etc. can be answered efficiently (Parker, Holt/Rees, Lux)
- For invertible matrices, basic group operations are easy enough
- But obtaining group theoretic information is very hard in general

SYSTEMS

Integrated Systems

- Both GAP and MAGMA are "integrated" systems for computational algebra. They provide:
  - implementations of a wide range of algorithms
  - "glue" to allow algebraic data to be passed between them
  - support features like memory management, large integer arithmetic, finite field arithmetic, lists, records, etc.
  - a programming language with which the user can extend the system, drawing freely on all the existing features
- Why use such a system?
  - The techniques you need may be built in – use as a "desk calculator" – no user programming at all
  - Interpreted language and interactive debugging make it easier to program in GAP or MAGMA than in (say) C, especially small programs
  - Many modern algorithms draw on a huge range of known techniques – avoid the need to reimplement them all
  - Existing system capabilities are there to prepare input and analyse output from your new algorithm
  - Easy for others to build on your work
- Why not?
  - Performance penalty – 3 to 10 times for most things

GAP

- Groups, Algorithms, Programming
- Free software
- Runs under UNIX, MSDOS and MacOS (at least) – easy to port
- Developed over 11 years in Lehrstuhl D für Mathematik, RWTH Aachen, Germany
  - Prof Joachim Neubüser
  - Martin Schönert
  - Cast of thousands, mainly in Aachen, some elsewhere
- Headquarters moving to St Andrews in the next few weeks
- Current version 3.4.4. Version 4 about to enter β-test
- System is mostly written in the GAP language
- Strongest areas
  - Permutation groups (esp in v4)
  - Finite Soluble groups
  - Character tables
  - Programmability
  - A number of powerful contributed share packages
- More info `http://www-gap.dcs.st-and.ac.uk/~gap/`

MAGMA

- Not free (\$1000 – \$2000 for a 3 year license?)
- Successor to Cayley
- Produced by John Cannon in Sydney
- Available for UNIX, DOS, maybe MacOS
- Current version 2.2
- Strong in number theory, representation theory, finitely-presented groups
- Mostly written in C – incorporated many stand-alone C programs
- High performance within the C modules
- Less popular with programmers and extenders

HOT TOPICS

Matrix Group Recognition

- Unlike permutation groups, there is no convenient chain of subgroups in a matrix group, and more sophisticated methods are needed
- Goal is to identify the group G generated by given matrices, and, for example, give its order.
- Main target area $n, q < 100$
- Key is Aschbacher's theorem, classifying maximal subgroups of $SL_n(q)$
- First decide which one(s) G is in
- Use their structure to approach a composition series for G
- Recognise the composition factors against a database
- Substantial progress made – Leedham-Green, O'Brien, Holt/Rees, Pye, Celler *et al*
- GAP share package `matrix`

Black Box Recognition

- First mention is Benson 1982.
- Given "opaque" strings for generating group elements and a way of multiplying, inverting and comparing to the identity
- Problem 1: recognise the group
- Problem 2: find an explicit isomorphism between the group given and some standard version
- Elementary abelian groups are very hard
- Progress with S_n, classical matrix groups.
- Good results here would make the Monte Carlo permutation group algorithms Las Vegas

Infinite Polycyclic Groups

- Algorithms for constructing and computing in finite soluble groups are being extended to the infinite polycyclic case
- Eddie Lo, Gretchen Ostheimer, Werner Nickel

Other algebraic Structures

- Group theoretic algorithms are now far more sophisticated than those for other structures
- Can we apply the same ideas, or at least some of the general lessons in other contexts?
- Recent progress with
  - Lie Algebra (Willem de Graaf)
  - Finitely-presented semigroups (Robertson, Thomas, Ruskuc, *et al*)
  - Transformation Semigroups (SL, Pfeiffer, Ruskuc, Robertson)
  - Finitely-presented associative algebras (SL)
- Interest in
  - Matrix rings and algebras
  - Groupoids
  - Quantum groups
- So far, examples support the general thesis
- GAP 4 is designed to support this work much better than GAP 3