Computing automorphism groups and testing groups for isomorphism

Derek Holt

University of Warwick

Questions, Algorithms, and Computations in Abstract Group Theory, Braunschweig, May 2013

Contents

Search problems in computational group theory

Let G be a finite group with $|G| = n$.

Some computations involve searching through the elements of G, and so their complexity is potentially $\Omega(n)$.

Search problems in computational group theory

Let G be a finite group with $|G| = n$.

Some computations involve searching through the elements of G, and so their complexity is potentially $\Omega(n)$.

Such problems often come in pairs, one of which is to compute a certain subgroup and the other is to find a representative of a coset of the subgroup, which may or may not exist.

Algorithms for the two problems are typically very similar, so they can be considered (and implemented) together.

Examples

1. a) Find the centralizer of an element of G;
   b) Test two elements of G for conjugacy and find a conjugating element if it exists.

Examples

1. a) Find the centralizer of an element of G;
   b) Test two elements of G for conjugacy and find a conjugating element if it exists.

2. a) Find the normalizer of a subgroup of G;
   b) Test two subgroups of G for conjugacy and find a conjugating element if it exists.

Examples

1. a) Find the centralizer of an element of G;
   b) Test two elements of G for conjugacy and find a conjugating element if it exists.

2. a) Find the normalizer of a subgroup of G;
   b) Test two subgroups of G for conjugacy and find a conjugating element if it exists.

3. For $G \leq \mathrm{Sym}(X)$:
   a) Find the stabilizer of a subset of X;
   b) For two subsets $Y, Z \subseteq X$, test for existence of $g \in G$ with $Y^g = Z$ and find g if it exists.

Examples

1. a) Find the centralizer of an element of G;
   b) Test two elements of G for conjugacy and find a conjugating element if it exists.
2. a) Find the normalizer of a subgroup of G;
   b) Test two subgroups of G for conjugacy and find a conjugating element if it exists.
3. For $G \leq \mathrm{Sym}(X)$:
   a) Find the stabilizer of a subset of X;
   b) For two subsets $Y, Z \subseteq X$, test for existence of $g \in G$ with $Y^g = Z$ and find g if it exists.
4. For $G \leq \mathrm{GL}_n(\mathbb{F}_q)$ acting on $V = \mathbb{F}_q^n$:
   a) Find the stabilizer of a subspace W of V;
   b) For two subspaces W, X of V, test for existence of $g \in G$ with $W^g = X$ and find g if it exists.

5
a) Calculate $\mathrm{Aut}(G)$;
b) Test two groups G, H for isomorphism, and find an isomorphism if it exists.

5 a) Calculate $\mathrm{Aut}(G)$;
   b) Test two groups G, H for isomorphism, and find an isomorphism if it exists.

The naive complexity estimate for automorphism group computation and isomorphism testing is $n^{O(\log n)}$, and this is likely to be best possible.

Automorphism groups and isomorphism testing

5 a) Calculate $\mathrm{Aut}(G)$;
  b) Test two groups G, H for isomorphism, and find an isomorphism if it exists.

The naive complexity estimate for automorphism group computation and isomorphism testing is $n^{O(\log n)}$, and this is likely to be best possible.

We find a generating set S of G with $|S| \leq \log_2 n$. A homomorphism $G \to G$ is determined by the images of the elements of S, so we just try all $n^{|S|}$ such images, and test whether they define automorphisms of G.

Automorphism groups and isomorphism testing

5 a) Calculate $\mathrm{Aut}(G)$;
  b) Test two groups G, H for isomorphism, and find an isomorphism if it exists.

The naive complexity estimate for automorphism group computation and isomorphism testing is $n^{O(\log n)}$, and this is likely to be best possible.

We find a generating set S of G with $|S| \leq \log_2 n$. A homomorphism $G \to G$ is determined by the images of the elements of S, so we just try all $n^{|S|}$ such images, and test whether they define automorphisms of G.

For primes p, G. Higman showed how to construct about $p^{2n^3/27}$ non-isomorphic groups of order p^n. These are special p-groups with about $2n/3$ generators. It is unlikely that two such groups can be tested for isomorphism any faster than this.

Automorphism groups and isomorphism testing (ctd)

So all that we can do is to look for algorithms that perform well in practice on small or interesting examples.

Automorphism groups and isomorphism testing (ctd)

So all that we can do is to look for algorithms that perform well in practice on small or interesting examples.

From our naive analysis above, we would expect this to be substantially easier for groups with small numbers of generators, and this turns out to be the case: the minimal generator number is the most significant factor influencing performance of implemented methods.

Automorphism groups and isomorphism testing (ctd)

So all that we can do is to look for algorithms that perform well in practice on small or interesting examples.

From our naive analysis above, we would expect this to be substantially easier for groups with small numbers of generators, and this turns out to be the case: the minimal generator number is the most significant factor influencing performance of implemented methods.

For the remainder of the talk, everything said about computing $\mathrm{Aut}\, G$ applies also to the group isomorphism testing problem.

(In fact group isomorphism testing is one of the facilities that is used most frequently by typical users of computer algebra packages, such as **GAP** and **Magma**.)

Automorphism groups of finite p-groups

The best implemented method is due to Eick, Leedham-Green and O'Brien. As might be expected from the naive complexity analysis above, it performs best for p-groups with a small number of generators.

Automorphism groups of finite p-groups

The best implemented method is due to Eick, Leedham-Green and O'Brien. As might be expected from the naive complexity analysis above, it performs best for p-groups with a small number of generators.

The **lower p-central series** of G is defined by

$$P_0(G) := G; \quad P_i(G) := [P_{i-1}(G), G]P_{i-1}(G)^p \quad (i > 0).$$

We work downwards through the quotients $G/P_i(G)$, computing $\mathrm{Aut}(G/P_i(G))$ for $i = 1, 2, 3, \ldots$.

Automorphism groups of finite p-groups

The best implemented method is due to Eick, Leedham-Green and O'Brien. As might be expected from the naive complexity analysis above, it performs best for p-groups with a small number of generators.

The **lower p-central series** of G is defined by

$$P_0(G) := G; \quad P_i(G) := [P_{i-1}(G), G]P_{i-1}(G)^p \ \ (i > 0).$$

We work downwards through the quotients $G/P_i(G)$, computing $\mathrm{Aut}(G/P_i(G))$ for $i = 1, 2, 3, \ldots$.

The critical step in going from i to $i+1$ is the calculation of the stabilizer of a subspaces in the action of $\mathrm{Aut}(G/P_i(G))$ on the **p-multiplicator** of $G/P_i(G)$, which can be regarded as a vector space over $\mathbb{F}_p$.

An algorithm of Cannon and Holt generalizes a method of Michael Smith for finite solvable groups.

Automorphism groups of general finite groups

An algorithm of Cannon and Holt generalizes a method of Michael Smith for finite solvable groups.

This method belongs to a family of algorithms for computing in finite groups G, which have the following general structure.

Automorphism groups of general finite groups

An algorithm of Cannon and Holt generalizes a method of Michael Smith for finite solvable groups.

This method belongs to a family of algorithms for computing in finite groups G, which have the following general structure.

1. Compute the solvable radical $L := O_\infty(G)$ of G.

Automorphism groups of general finite groups

An algorithm of Cannon and Holt generalizes a method of Michael Smith for finite solvable groups.

This method belongs to a family of algorithms for computing in finite groups G, which have the following general structure.

1. Compute the solvable radical $L := O_\infty(G)$ of G.
2. Solve the problem in G/L using known properties of the nonabelian simple direct factors of $\mathrm{Soc}(G/L)$.

Automorphism groups of general finite groups

An algorithm of Cannon and Holt generalizes a method of Michael Smith for finite solvable groups.

This method belongs to a family of algorithms for computing in finite groups G, which have the following general structure.

1. Compute the solvable radical $L := O_\infty(G)$ of G.
2. Solve the problem in G/L using known properties of the nonabelian simple direct factors of $\mathrm{Soc}(G/L)$.
3. Solve the problem in G using linear algebra to lift the solution through the elementary abelian layers of L.

Automorphism groups of general finite groups (ctd)

For the automorphism group computation, we first find a series

$$G \geq L = N_1 > N_2 > \cdots N_r = 1$$

of characteristic subgroups of G, where $L = N_1 = O\infty(G)$, and each N_i/N_{i+1} is elementary abelian.

Automorphism groups of general finite groups (ctd)

For the automorphism group computation, we first find a series

$$G \geq L = N_1 > N_2 > \cdots N_r = 1$$

of characteristic subgroups of G, where $L = N_1 = O\infty(G)$, and each N_i/N_{i+1} is elementary abelian.

If $G \neq L$ but G/L is moderately small, then $\mathrm{Aut}(G/N_1)$ can be calculated from a knowledge of the automorphism groups of the simple direct factors of $\mathrm{Soc}(G/L)$.

For the automorphism group computation, we first find a series

$$G \geq L = N_1 > N_2 > \cdots N_r = 1$$

of characteristic subgroups of G, where $L = N_1 = O\infty(G)$, and each N_i/N_{i+1} is elementary abelian.

If $G \neq L$ but G/L is moderately small, then $\mathrm{Aut}(G/N_1)$ can be calculated from a knowledge of the automorphism groups of the simple direct factors of $\mathrm{Soc}(G/L)$.

This involves **black box recognition** algorithms for the finite simple groups.

Automorphism groups of general finite groups (ctd)

So we reduce to the **lifting** problem:

Given an elementary abelian p-subgroup N of G, for which $\overline{A} := \mathrm{Aut}(G/N)$ is known, compute $A := \mathrm{Aut}\,G$.

Automorphism groups of general finite groups (ctd)

So we reduce to the **lifting** problem:

Given an elementary abelian p-subgroup N of G, for which $\overline{A} := \mathrm{Aut}(G/N)$ is known, compute $A := \mathrm{Aut}\,G$.

We have a series $1 \leq C \trianglelefteq B \trianglelefteq A$ of normal subgroups of A defined by

$$B := \{\alpha \in A \mid \alpha_{G/N} = 1\}; \qquad C := \{\beta \in B \mid \beta_N = 1\}.$$

So we reduce to the **lifting** problem:

Given an elementary abelian p-subgroup N of G, for which $\overline{A} := \mathrm{Aut}(G/N)$ is known, compute $A := \mathrm{Aut}G$.

We have a series $1 \leq C \trianglelefteq B \trianglelefteq A$ of normal subgroups of A defined by

$$B := \{\alpha \in A \mid \alpha_{G/N} = 1\}; \qquad C := \{\beta \in B \mid \beta_N = 1\}.$$

The subgroups C and B do not depend on $\overline{A}$, and are typically comparatively straightforward to compute:

$$C = H^1(G/N, N) \text{ and } B/C = \mathrm{Aut}N \text{ as } \mathbb{F}_p G/N\text{-module.}$$

Automorphism groups of general finite groups (ctd)

So we reduce to the **lifting** problem:

Given an elementary abelian p-subgroup N of G, for which $\overline{A} := \mathrm{Aut}(G/N)$ is known, compute $A := \mathrm{Aut}\,G$.

We have a series $1 \leq C \trianglelefteq B \trianglelefteq A$ of normal subgroups of A defined by

$$B := \{\alpha \in A \mid \alpha_{G/N} = 1\}; \qquad C := \{\beta \in B \mid \beta_N = 1\}.$$

The subgroups C and B do not depend on $\overline{A}$, and are typically comparatively straightforward to compute:

$$C = H^1(G/N, N) \text{ and } B/C = \mathrm{Aut}N \text{ as } \mathbb{F}_pG/N\text{-module.}$$

The most difficult problem is computing A/B, which is a search problem, where we need to determine which elements of $\overline{A}$ lift to A.

The special methods for p-groups, where we reduce to a subspace stabilizer computation, are not available here.

An alternative approach using automorphisms of p-groups

A recent alternative approach due to David Howden attempts to calculate $\mathrm{Aut}G$ from $\mathrm{Aut}P$ for one or more Sylow subgroups of G. This has been implemented in **Magma**.

A recent alternative approach due to David Howden attempts to calculate $\mathrm{Aut}G$ from $\mathrm{Aut}P$ for one or more Sylow subgroups of G. This has been implemented in **Magma**.

It works best for groups with a Sylow subgroup of relatively small index.

It seems to perform more reliably than existing methods on groups of order up to 2000 (in the **small groups** database).

An alternative approach using automorphisms of p-groups

A recent alternative approach due to David Howden attempts to calculate $\mathrm{Aut}G$ from $\mathrm{Aut}P$ for one or more Sylow subgroups of G. This has been implemented in **Magma**.

It works best for groups with a Sylow subgroup of relatively small index.

It seems to perform more reliably than existing methods on groups of order up to 2000 (in the **small groups** database).

On some types of examples, such as iterated wreath products of S_3 or S_4, it is no good at all, because computing $\mathrm{Aut}P$ for the Sylow subgroups is slower than computing $\mathrm{Aut}G$ with existing methods,

Groups with a large Sylow subgroup

Let $A := \mathrm{Aut}G$. For simplicity, assume that G is solvable.

Lemma

If $O_p(G)$ and $O_q(G)$ are nontrivial for distinct primes p, q, then $G \leq G/O_q(G) \times G/O_p(G)$ and $\mathrm{Aut}G \leq \mathrm{Aut}(G/O_q(G)) \times \mathrm{Aut}(G/O_p(G))$.

Groups with a large Sylow subgroup

Let $A := \operatorname{Aut} G$. For simplicity, assume that G is solvable.

Lemma

If $O_p(G)$ and $O_q(G)$ are nontrivial for distinct primes p, q, then $G \leq G/O_q(G) \times G/O_p(G)$ and $\operatorname{Aut} G \leq \operatorname{Aut}(G/O_q(G)) \times \operatorname{Aut}(G/O_p(G))$.

So we assume that there is a unique prime p with $1 \neq K := O_p(G)$.

Let $G = PQ$ with $P \in \operatorname{Syl}_p(G)$ and Q a complement of P in G. Then:

Lemma

We have $A = A_{P,Q} \operatorname{Inn} G$, where $A_{P,Q} = \{\alpha \in A \mid P^\alpha = P,\ Q^\alpha = Q\}$.

Groups with a large Sylow subgroup (ctd)

The assumption that $O_q(G) = 1$ for all primes $q \neq p$ implies that $C_Q(K) = 1$, and hence Q can be identified with a subgroup $\overline{Q}$ of $\mathrm{Aut}K$.

Groups with a large Sylow subgroup (ctd)

The assumption that $O_q(G) = 1$ for all primes $q \neq p$ implies that $C_Q(K) = 1$, and hence Q can be identified with a subgroup $\overline{Q}$ of $\mathrm{Aut}K$.

The case when $P \trianglelefteq G$ (i.e. $K = P$) is easy.

Proposition

If $K = P$ then $A_{P,Q} \cong \{\alpha \in \mathrm{Aut}K \mid \overline{Q}^{\alpha} = \overline{Q}\}$.

Groups with a large Sylow subgroup (ctd)

The assumption that $O_q(G) = 1$ for all primes $q \neq p$ implies that $C_Q(K) = 1$, and hence Q can be identified with a subgroup $\overline{Q}$ of $\mathrm{Aut}K$.

The case when $P \trianglelefteq G$ (i.e. $K = P$) is easy.

Proposition

If $K = P$ then $A_{P,Q} \cong \{\alpha \in \mathrm{Aut}K \mid \overline{Q}^{\alpha} = \overline{Q}\}$.

More generally, we have

Proposition

$A_{P,Q}$ is isomorphic to a subgroup of

$$B := \{\alpha \in \mathrm{Aut}P \mid K^{\alpha} = K,\ N_P(Q)^{\alpha} = N_P(Q),\ \alpha|_K \in N_{\mathrm{Aut}K}(\overline{Q})\}.$$

Groups with a large Sylow subgroup (ctd)

This subgroup is typically of small index in B, so can be found by a straightforward search.

This subgroup is typically of small index in B, so can be found by a straightforward search.

But finding B itself involves a normalizer computation within $\mathrm{Aut}K$ and the stabilizer in $\mathrm{Aut}P$ of two subgroups of P.

Groups with a large Sylow subgroup (ctd)

This subgroup is typically of small index in B, so can be found by a straightforward search.

But finding B itself involves a normalizer computation within $\mathrm{Aut}K$ and the stabilizer in $\mathrm{Aut}P$ of two subgroups of P.

To perform such computations efficiently, we need

a nice representation of $\mathrm{Aut}P$

This subgroup is typically of small index in B, so can be found by a straightforward search.

But finding B itself involves a normalizer computation within Aut$K$ and the stabilizer in Aut$P$ of two subgroups of P.

To perform such computations efficiently, we need

a nice representation of Aut$P$

We use a permutation representation if one can be found of reasonably small degree.

Representing automorphism groups

For larger p-groups, there may be no suitable permutation representation.

In general, the p-group automorphism algorithm calculates a normal p-subgroup S of $\mathrm{Aut}P$ and a representation of $(\mathrm{Aut}P)/S$ as a subgroup of $\mathrm{GL}_d(p)$, where d is size of a minimal generating set of P.

Representing automorphism groups

For larger p-groups, there may be no suitable permutation representation.

In general, the p-group automorphism algorithm calculates a normal p-subgroup S of $\mathrm{Aut}P$ and a representation of $(\mathrm{Aut}P)/S$ as a subgroup of $\mathrm{GL}_d(p)$, where d is size of a minimal generating set of P.

If $(\mathrm{Aut}P)/S$ is solvable, then we can compute a **PC-presentation** of $\mathrm{Aut}P$.

Representing automorphism groups

For larger p-groups, there may be no suitable permutation representation.

In general, the p-group automorphism algorithm calculates a normal p-subgroup S of $\mathrm{Aut}P$ and a representation of $(\mathrm{Aut}P)/S$ as a subgroup of $\mathrm{GL}_d(p)$, where d is size of a minimal generating set of P.

If $(\mathrm{Aut}P)/S$ is solvable, then we can compute a **PC-presentation** of $\mathrm{Aut}P$.

Otherwise we can treat $\mathrm{Aut}P$ as a **hybrid group**, which we define to be a finite group G with a solvable normal subgroup S, defined by a PC-presentation, together with a nice representation of G/S.

Representing automorphism groups

For larger p-groups, there may be no suitable permutation representation.

In general, the p-group automorphism algorithm calculates a normal p-subgroup S of $\mathrm{Aut}P$ and a representation of $(\mathrm{Aut}P)/S$ as a subgroup of $\mathrm{GL}_d(p)$, where d is size of a minimal generating set of P.

If $(\mathrm{Aut}P)/S$ is solvable, then we can compute a **PC-presentation** of $\mathrm{Aut}P$.

Otherwise we can treat $\mathrm{Aut}P$ as a **hybrid group**, which we define to be a finite group G with a solvable normal subgroup S, defined by a PC-presentation, together with a nice representation of G/S.

Implementations of algorithms for such groups are useful in other contexts, such as computations in large matrix group with nontrivial solvable radical.

Bibliography

J.J. Cannon and D.F. Holt. Automorphism group computation and isomorphism testing in finite groups. *J. Symb. Comput.*, 35, 241–267, 2003.

B. Eick, C.R. Leedham-Green, E.A. O'Brien, Constructing automorphism groups of p-groups. *Comm. Algebra* **30**, 2271–2295, 2002.

D.J.A. Howden *Computing Automorphism Groups and Isomorphism Testing in Finite Groups* PhD Thesis, University of Warwick, 2012.

M.J. Smith, *Computing Automorphisms of Finite Soluble Groups*. PhD Thesis, Australian National University, 1994.