Order Computations in Generic Groups
Thesis Defense

Andrew V. Sutherland

Massachusetts Institute of Technology

April 27, 2007

Outline

Why generic groups?

Complexity Results

Strong lower bounds.
(Babai & Szémeredi 1984, Shoup 1997, Babai & Beals 1997)

Generality

Algorithms reusable and widely applicable.
Computational algebra, number theory, cryptography.
(ATLAS, Magma, GAP, Mathematica, LiDIA, Pari/GP)

Puzzle Appeal

What's inside the black box?

Why generic groups?

Complexity Results

Strong lower bounds.
(Babai & Szémeredi 1984, Shoup 1997, Babai & Beals 1997)

Generality

Algorithms reusable and widely applicable.
Computational algebra, number theory, cryptography.
(ATLAS, Magma, GAP, Mathematica, LiDIA, Pari/GP)

Puzzle Appeal

What's inside the black box?

Why generic groups?

Complexity Results

Strong lower bounds.
(Babai & Szémeredi 1984, Shoup 1997, Babai & Beals 1997)

Generality

Algorithms reusable and widely applicable.
Computational algebra, number theory, cryptography.
(ATLAS, Magma, GAP, Mathematica, LiDIA, Pari/GP)

Puzzle Appeal

What's inside the black box?

Computational Model

Black Box Groups

Black box $\mathcal{B}(G)$ supports: $\mathcal{B}_{mult}(\alpha, \beta)$, $\mathcal{B}_{inv}(\alpha)$, $\mathcal{B}_{id}()$.
(Babai and Szémeredi 1984)

Unique Identification

Bijective identification map $\mathcal{B} : G \leftrightarrow \mathcal{I}$ hides representation.
(Shoup 1997)

Random Group Elements

$\mathcal{B}_{rand}()$ returns a uniformly random $\alpha \in G$.
(CLMNO 1995, Babai 1997, Pak 2000)

Generic Group Algorithms

Generic Group Functions/Relations

Defined for any finite group. Invariant under isomorphisms.
Examples: α^k, $|\alpha|$, $DL(\alpha, \beta)$, $isAbelian()$, $Generators()$.

Complexity Metrics

Count group operations and group identifiers stored.

Correctness

Must be correct for every group G and every black box $\mathcal{B}(G)$.

Why order computation?

Fundamental Problem

Essential component of many generic algorithms (order oracle).

Hard Problem

Exponential lower bounds (Babai 1999, Sutherland 2007).
Strictly harder than factoring integers.
As hard as $DL(\alpha, \beta)$? $(\alpha^k = 1_G$ vs. $\alpha^k = \beta)$.

Easy Problem

Given a factored exponent of α (a multiple of $|\alpha|$), linear or near-linear upper bounds (CL 1997, Sutherland 2006).

Why order computation?

Fundamental Problem

Essential component of many generic algorithms (order oracle).

Hard Problem

Exponential lower bounds (Babai 1999, Sutherland 2007).
Strictly harder than factoring integers.
As hard as $DL(\alpha, \beta)$? $(\alpha^k = 1_G$ vs. $\alpha^k = \beta)$.

Easy Problem

Given a factored exponent of α (a multiple of $|\alpha|$), linear or near-linear upper bounds (CL 1997, Sutherland 2006).

Why order computation?

Fundamental Problem

Essential component of many generic algorithms (order oracle).

Hard Problem

Exponential lower bounds (Babai 1999, Sutherland 2007).
Strictly harder than factoring integers.
As hard as $DL(\alpha, \beta)$? $(\alpha^k = 1_G$ vs. $\alpha^k = \beta)$.

Easy Problem

Given a factored exponent of α (a multiple of $|\alpha|$), linear or near-linear upper bounds (CL 1997, Sutherland 2006).

Why order computation?

Fundamental Problem

Essential component of many generic algorithms (order oracle).

Hard Problem

Exponential lower bounds (Babai 1999, Sutherland 2007).
Strictly harder than factoring integers.
As hard as $DL(\alpha, \beta)$? $(\alpha^k = 1_G$ vs. $\alpha^k = \beta)$.

Easy Problem

Given a factored exponent of α (a multiple of $|\alpha|$), linear or near-linear upper bounds (CL 1997, Sutherland 2006).

Why order computation?

Fundamental Problem

Essential component of many generic algorithms (order oracle).

Hard Problem

Exponential lower bounds (Babai 1999, Sutherland 2007).
Strictly harder than factoring integers.
As hard as $DL(\alpha, \beta)$? $(\alpha^k = 1_G$ vs. $\alpha^k = \beta)$.

Easy Problem

Given a factored exponent of α (a multiple of $|\alpha|$), linear or near-linear upper bounds (CL 1997, Sutherland 2006).

Order Computation

Problem

- Find the least positive N such that $\alpha^N = 1_G$.
- No upper bound on N.
- $\alpha^k = \alpha^j \iff k \equiv j \mod N$.

Solutions

- Birthday paradox suggests $\approx \sqrt{N}$.
- Pollard rho method $\sqrt{2\pi N}$ (Teske 1998, 2001).
- Shanks baby-steps giant-steps $2\sqrt{2N}$ (Terr 2000).

Order Computation

Problem

- Find the least positive N such that $\alpha^N = 1_G$.
- No upper bound on N.
- $\alpha^k = \alpha^j \iff k \equiv j \mod N$.

Solutions

- Birthday paradox suggests $\approx \sqrt{N}$.
- Pollard rho method $\sqrt{2\pi N}$ (Teske 1998, 2001).
- Shanks baby-steps giant-steps $2\sqrt{2N}$ (Terr 2000).

Lower Bounds?

Babai

Exponential lower bound in black-box groups.

Shoup

$\Omega(\sqrt{N})$ lower bound for discrete logarithm in generic groups.

Terr

$\sqrt{2N}$ lower bound on addition chains.

Birthday Paradox

$\sqrt{(2\log 2)N}$ lower bound for a random algorithm.

Outline

Main Results

New Generic Order Algorithm

Always $o\left(N^{1/2}\right)$, usually near $O\left(N^{1/3}\right)$.
Occasionally subexponential.

Order Computation Theorem

Many order computations for the cost of one.

Abelian Group Structure Algorithm

$O\left(M^{1/4}\right)$ in almost all cases, given $M \geq |G|$ and $\lambda(G)$.

The Basic Idea

Modified Baby-steps Giant-steps

What if we knew $|\alpha|$ were odd?

What if we knew $|\alpha| \perp 6$?

What if we knew $|\alpha| \perp \prod_{p \le L} p$?

Key Fact: Orders Can Be Divided

For any $\beta = \alpha^d$:

$$|\beta| = N_1 \quad \text{and} \quad |\alpha^{N_1}| = N_2 \qquad \Longrightarrow \qquad |\alpha| = N_1 N_2.$$

Note that $N_1 = |\alpha| / \gcd(d, |\alpha|)$ and $N_2 = \gcd(d, |\alpha|)$.

The Basic Idea

Modified Baby-steps Giant-steps

What if we knew $|\alpha|$ were odd?

What if we knew $|\alpha| \perp 6$?

What if we knew $|\alpha| \perp \prod_{p \leq L} p$?

Key Fact: Orders Can Be Divided

For any $\beta = \alpha^d$:

$$|\beta| = N_1 \quad \text{and} \quad |\alpha^{N_1}| = N_2 \qquad \Longrightarrow \qquad |\alpha| = N_1 N_2.$$

Note that $N_1 = |\alpha| / \gcd(d, |\alpha|)$ and $N_2 = \gcd(d, |\alpha|)$.

The Basic Idea

Modified Baby-steps Giant-steps

What if we knew $|\alpha|$ were odd?

What if we knew $|\alpha| \perp 6$?

What if we knew $|\alpha| \perp \prod_{p \leq L} p$?

Key Fact: Orders Can Be Divided

For any $\beta = \alpha^d$:

$$|\beta| = N_1 \quad \text{and} \quad |\alpha^{N_1}| = N_2 \qquad \Longrightarrow \qquad |\alpha| = N_1 N_2.$$

Note that $N_1 = |\alpha| / \gcd(d, |\alpha|)$ and $N_2 = \gcd(d, |\alpha|)$.

The Basic Idea

Modified Baby-steps Giant-steps

What if we knew $|\alpha|$ were odd?

What if we knew $|\alpha| \perp 6$?

What if we knew $|\alpha| \perp \prod_{p \leq L} p$?

Key Fact: Orders Can Be Divided

For any $\beta = \alpha^d$:

$$|\beta| = N_1 \quad \text{and} \quad |\alpha^{N_1}| = N_2 \qquad \Longrightarrow \qquad |\alpha| = N_1 N_2.$$

Note that $N_1 = |\alpha| / \gcd(d, |\alpha|)$ and $N_2 = \gcd(d, |\alpha|)$.

Primorial Steps Algorithm

1. Let $E = \prod p^h$ and $P = \prod p \quad$ (for $p \leq L$ with $p^{h+1} > M$).
2. Compute $\beta = \alpha^E$.
3. Use baby-steps $\perp P$ and giant-step multiples of P to find $N_1 = |\beta|$.
4. Use a fast order algorithm to find $N_2 = |\alpha^{N_1}|$ given E.
5. Return $N_1 N_2$.

Primorials

w	p_w	P_w	$\phi(P_w)$	$\phi(P_w)/P_w$	$P_w/\phi(P_w)$
1	2	2	1	0.5000	2.0000
2	3	6	2	0.3333	3.0000
3	5	30	8	0.2667	3.7500
4	7	210	48	0.2286	4.3450
5	11	2310	480	0.2078	4.8125
6	13	30030	5760	0.1918	5.2135
7	17	510510	92160	0.1805	5.5394
8	19	9699690	1658880	0.1710	5.8471
9	23	223092870	36495360	0.1636	6.1129
10	29	6469693230	1021870080	0.1579	6.3312

Table: The First Ten Primorials

Complexity

Worst Case

$$O\left(\sqrt{N/\log\log N}\right)$$

Best Case

$$O(\pi(L)\lg M)$$

Typical Case

Let's try it.

Complexity

Worst Case

$$O\left(\sqrt{N/\log\log N}\right)$$

Best Case

$$O(\pi(L)\lg M)$$

Typical Case

Let's try it.

Complexity

Worst Case

$$O\left(\sqrt{N/\log\log N}\right)$$

Best Case

$$O(\pi(L)\lg M)$$

Typical Case

Let's try it.

The Multi-Stage Sieve

Factoring in the Dark

Problem: We don't know any factors until we find them all.

Play the Odds

Solution: Alternate sieving and searching until we do.

Reap the Benefits

Result: Complexity depends on $q_*(N) = \max(\sqrt{p_1}, p_2)$.

The Multi-Stage Sieve

Factoring in the Dark

Problem: We don't know any factors until we find them all.

Play the Odds

Solution: Alternate sieving and searching until we do.

Reap the Benefits

Result: Complexity depends on $q_*(N) = \max(\sqrt{p_1}, p_2)$.

The Multi-Stage Sieve

Factoring in the Dark

Problem: We don't know any factors until we find them all.

Play the Odds

Solution: Alternate sieving and searching until we do.

Reap the Benefits

Result: Complexity depends on $q_*(N) = \max(\sqrt{p_1}, p_2)$.

Complexity

Median Complexity

$O(N^{0.344})$ for uniform distribution on $N = |\alpha|$. Often better.

More generally...

$$Pr\left[T(N) \leq cN^{1/u}\right] \geq G(1/u, 2/u)$$

Subexponential Result

Choosing appropriate u gives $L[1/2, \sqrt{2}]$ algorithm for solving one of a sequence of random problems.

Complexity

Median Complexity

$O(N^{0.344})$ for uniform distribution on $N = |\alpha|$. Often better.

More generally...

$$Pr\left[T(N) \leq cN^{1/u}\right] \geq G(1/u, 2/u)$$

Subexponential Result

Choosing appropriate u gives $L[1/2, \sqrt{2}]$ algorithm for solving one of a sequence of random problems.

Complexity

Median Complexity

$O(N^{0.344})$ for uniform distribution on $N = |\alpha|$. Often better.

More generally...

$$Pr\left[T(N) \leq cN^{1/u}\right] \geq G(1/u, 2/u)$$

Subexponential Result

Choosing appropriate u gives $L[1/2, \sqrt{2}]$ algorithm for solving one of a sequence of random problems.

Semismooth and Smooth Probabilities

u	$G(1/u, 2/u)$	$\rho(u)$
2.2	0.8958	0.2203
2.5	0.7302	0.1303
2.9	0.5038	0.0598
3.0	0.4473	0.0486
4.0	0.0963	0.0049
5.0	0.0124	0.0003
6.0	1.092e-03	1.964e-05
8.0	3.662e-06	3.232e-08

The Group Exponent

Definition of $\lambda(G)$

$\lambda(G)$ is the least E such that $\alpha^E = 1_G$ for all $\alpha \in G$.
Equivalently, $\lambda(G) = \text{lcm}(|\alpha|)$ over $\alpha \in G$.

The Universal Exponent

Given factored $\lambda(G)$, all order computations are fast.

Generalization

For any subset $S \subseteq G$, $\lambda(S)$ is defined similarly.

Computing $\lambda(S)$ via Order Computations

Set Order Algorithm

Let $E = 1$.

For $\alpha \in S$:

1. Compute $e \leftarrow |\alpha^E|$ using a general order algorithm.
2. Factor e and set $E \leftarrow eE$.
3. Compute $|\alpha|$ using a fast order algorithm given E.

Output $\lambda(S) = E$.

Order Computation Theorem

Complexity of Set Order Algorithm

Exponentiation: $|S| O(\lg E)$

General Order: $T_1(e_1) + \cdots + T_1(e_k) \leq T_1(e_1 \cdots e_k) = T_1(E)$

Fast Order: $|S| T_2(\lg E)$

Order Computation Theorem

Let S be any subset of G. Computing $|\alpha|$ for all $\alpha \in S$ costs

$$(1 + o(1)) T_1(\lambda(S)) + |S| T_2(\lg \lambda(S))$$

group operations.

The Structure of an Abelian Group

Structure Theorem for Finite Abelian Groups

For any finite abelian group G:

1. $G \cong C_{d_1} \otimes \cdots \otimes C_{d_k}$ with $d_1 | \cdots | d_k$.
2. $G \cong C_{p^r} \otimes \cdots \otimes C_{q^s}$ with $p, \ldots, q$ prime.

The Problem

Find generators with known order for each cyclic group.
In other words, compute a *basis* for G.

Computing the Structure of an Abelian Group

Main Idea

Use $\lambda(G)$ to process p-Sylow subgroups H_p separately.
Compute $\alpha^{\lambda(G)/p^h}$ for random $\alpha \in G$ to sample H_p.

Basic Algorithm

Let $\vec{\alpha} = \emptyset$.

1. Try to find a random $\beta \in H_p$ not spanned by $\vec{\alpha}$.
2. Determine a minimal relation on $\vec{\alpha} \circ \beta$.
3. Reduce $\vec{\alpha} \circ \beta$ to a basis, update $\vec{\alpha}$, and repeat.

Computing the Structure of an Abelian Group

Benefits of using p-Sylow subgroups

Greatly simplifies basis reduction (avoids SNF).
Big savings when $|G|$ contains multiple primes.

Helpful Hint

Use $M = O\left(|G|^{\delta}\right)$ to avoid expensive discrete logs.
Big savings when $|G|$ contains a prime $p > \sqrt{M}$.

Net Result

Complexity is $O\left(M^{1/4}\right) = O\left(|G|^{\delta/4}\right)$ once $\lambda(G)$ is known (in almost all cases).

Performance Comparisons

Reference Problem - Ideal Class Groups

Compute the ideal class group of $\mathbb{Q}[\sqrt{D}]$ for negative D.

Comparison to Generic Algorithms: $D = -4(10^{30} + 1)$

(Teske 1998): 250 million gops, 15 days ($\approx$ 2-6 hours)
Multi-stage sieve: 250,000 gops, 6 seconds.

Comparison to Non-Generic Algorithms: $D = -4(10^{54} + 1)$

(Buchmann MPQS 1999): 9 hours ($\approx$ 10-30 minutes)
Existing generic: 3×10^{14} gops, 200 years.
Multi-stage sieve: 800,000 gops, 17 seconds

Performance Comparisons

Reference Problem - Ideal Class Groups

Compute the ideal class group of $\mathbb{Q}[\sqrt{D}]$ for negative D.

Comparison to Generic Algorithms: $D = -4(10^{30} + 1)$

(Teske 1998): 250 million gops, 15 days ($\approx$ 2-6 hours)
Multi-stage sieve: 250,000 gops, 6 seconds.

Comparison to Non-Generic Algorithms: $D = -4(10^{54} + 1)$

(Buchmann MPQS 1999): 9 hours ($\approx$ 10-30 minutes)
Existing generic: 3×10^{14} gops, 200 years.
Multi-stage sieve: 800,000 gops, 17 seconds

Performance Comparisons

Reference Problem - Ideal Class Groups

Compute the ideal class group of $\mathbb{Q}[\sqrt{D}]$ for negative D.

Comparison to Generic Algorithms: $D = -4(10^{30} + 1)$

(Teske 1998): 250 million gops, 15 days ($\approx$ 2-6 hours)
Multi-stage sieve: 250,000 gops, 6 seconds.

Comparison to Non-Generic Algorithms: $D = -4(10^{54} + 1)$

(Buchmann MPQS 1999): 9 hours ($\approx$ 10-30 minutes)
Existing generic: 3×10^{14} gops, 200 years.
Multi-stage sieve: 800,000 gops, 17 seconds

Recipe for Subexponential Algorithms

Subexponential Approach

Choose u so that $cN^{1/u}G(1/u, 2/u) \approx 1$.
Running time is “aysmptotically” $L(1/2, \sqrt{2})$ or $L(1/2, 1)$.

Example: $D = -(10^{80} + 1387)$

Primorial steps: 10^9 gops, 8 hours (u = 7).
Existing generic: $\approx 10^{21}$ gops, many millenia.
Best non-generic: a few days.

Generic Solution

Works for any problem that can be reduced to random order computations.

Outline

Main Results

New Generic Order Algorithm

Always $o\left(N^{1/2}\right)$, usually near $O\left(N^{1/3}\right)$.
Occasionally subexponential.

Order Computation Theorem

Many order computations for the cost of one.

Abelian Group Structure Algorithm

$O\left(M^{1/4}\right)$ in almost all cases, given $M \geq |G|$ and $\lambda(G)$.

Future Work - Specific Questions

What is the right bound for order computation?

$$O\left(\sqrt{N/logN}\right)? \qquad \Omega\left(\sqrt{N}/\log N\right)?$$

Space efficient worst case?

$o\left(\sqrt{N}\right)$ algorithm using polylogarithmic space?

Future Work - The Bigger Picture

Applications of the Order Computation Theorem

Which generic algorithms could be redesigned to take better advantage of these results?

Subexponential Applications

Which problems reduce to random order computations?