S-arithmetic groups in quantum computing

MFO mini-workshop

Sebastian Schönnenbeck
joint with Alex Bocharov and Vadym Kliuchnikov

November 2016

Experimental and constructive algebra

RWTH AACHEN UNIVERSITY

Quantum computing

History

Quantum computing is an alternative to classical computers that exploits certain properties of quantum systems to obtain exponential speedup on certain problems. The concept goes back to Paul Benioff, Yuri Manin (1980), Richard Feynman (1982) and David Deutsch (1985).

Classical and quantum computers

Classical

- Memory unit: Bit, 0 or 1
- Think: Element of $\mathbb{F}_2$
- State of an n-bit computer: Element of $\mathbb{F}_2^n$
- Computation step: Applying $g \in \mathbb{F}_2^{n\times n}$

Quantum

- Memory unit: Qubit, "superposition" of 0 and 1
- Think: Element of $\mathbb{C}^2 \cong \mathbb{C}[(\mathbb{F}_2,+)]$ (with norm 1)
- State of an n-qubit computer: Element of $(\mathbb{C}^2)^{\otimes n} \cong \mathbb{C}^{2^n}$
- Computation step: Applying $g \in \mathrm{U}_{2^n}(\mathbb{C})$ (*gate*)
- Consequence: Computations on a quantum device are necessarily reversible.

Example application

Shor's algorithm

Used for factoring large numbers and roughly works as follows:

1. We want to factor n and choose random $2 \leq x \leq n-1$.
2. Compute the order r of $x \mod n$.
3. If r is even compute $\gcd(x^{r/2}+1, n), \gcd(x^{r/2}-1, n)$.

Fact sheet

- If $n = p \cdot q$ with primes p and q the algorithm finds a nontrivial factor with probability $\frac{1}{4}$ each time it reaches step (3).
- Every step except for (2) can be done efficiently on a classical computer.
- Quantum computer can perform step (2) in polynomial time by efficiently computing the period of the function $i \mapsto x^i \mod n$.
- Due to the large constants in the run time Shor's algorithm only beats classical algorithms for very large numbers. Current (optimistic) estimate: ~ 2 kbit (RSA key size).

State of the art

Device development (some milestones)

- 1998: First experimental demonstration of a quantum algorithm (2 qubits).
- 2000: First working 5- and 7-qubit devices.
- 2001: First execution of Shor's algorithm.
- 2006: First 12-qubit device.
- 2007: First demonstration of certain important gates; D-Wave claims to have a 28-qubit device.
- 2015: D-Wave announces to have $1000+$-qubit device.

Development on factoring

- 2001: Factoring 15 on an 8-qubit device. This remained the record until 2012.
- 2012: Factoring 21 on a 10-qubit device. Both these results rely on prior knowledge of the solution.
- 2012: Factoring 143 on a 4-qubit device. Using a different algorithm (without prior knowledge but only applicable to products of twin primes).
- 2012/14: Factoring 56153 on a 4 qubit device. Using the same algorithm as for 143 (in fact: the same computation). Current record.

Our basic setup

Let's assume we have a quantum computer

Our model

- Our quantum computer has n qubits, its states are elements of $\mathbb{C}^{2^n}$.
- Quantum algorithms we want to execute are unitary matrices $g \in \mathrm{U}_{2^n}(\mathbb{C})$.
- The architecture of our device gives us a finite gate set $\mathcal{G}$ which we are allowed to use as building blocks (each gate comes with an associated cost).
- The algorithms we can (exactly) execute on our device are the elements of $G := \langle \mathcal{G} \rangle \lneq \mathrm{U}_{2^n}(\mathbb{C})$.

Remark

Every algorithm can be executed approximately with arbitrary precision if and only if G is dense in $\mathrm{U}_{2^n}(\mathbb{C})$. In this case $\mathcal{G}$ is called a *universal gate set*.

Problems we care about

Exact synthesis (Compiling an algorithm)

- Input: A unitary g that can be expressed as a product of our gates $\mathcal{G}$ (i.e. $g \in G$).
- Task: Find a sequence $(g_1, ..., g_d)$ of elements of $\mathcal{G}$ with $g = g_1 ... g_d$.
- Constraint: Find such a sequence with low(est) cost (without brute forcing this).

State preparation (Preparing the quantum computer)

- Input: A norm-1 vector v in the G-orbit of $|0\rangle = e_1$.
- Task: Find a sequence $(g_1, ..., g_d)$ of elements of $\mathcal{G}$ with $v = g_1 ... g_d |0\rangle$.
- Constraint: Same as for exact synthesis.

Problem

For arbitrary/ “random” choices of $\mathcal{G}$ either problem is virtually impossible to solve without brute forcing.

Problems we (essentially) ignore

Actually building a quantum computer

Lots of physics/ engineering problems, e.g.:

- Quantum decoherence (any interaction of the device with the outside world may collapse your computation)
- Temperature constraints (current devices mostly work at less than 1 Kelvin)

Approximate synthesis/ state preparation

An arbitrary unitary g is usually not expressible as a word in $\mathcal{G}$.
$\rightsquigarrow$ Approximation step needed to find $g' \in G$ close to g (possibly expressible with low cost).

Designing quantum algorithms

Turning a real world problem into a quantum algorithm requires us to find a *reversible* implementation. Sometimes this is straightforward, sometimes very hard (e.g. quantum implementation of $\arcsin$).

Gate sets

What would constitute a “good” gate set?

Constraints

1. G needs to be dense in $\mathrm{U}_{2^n}(\mathbb{C})$.
2. $\mathcal{G}$ needs to be easy to work with on a (classical) computer.
3. $\mathcal{G}$ needs to be realizable in the real world.

Examples of realizable gates

Hadamard gate $H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$, T-gate $T = \mathrm{diag}(e^{i\pi/4}, 1)$,
CNOT $\mathrm{diag}\left(1, 1, \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\right)$.

Observation

All of these gates have entries in some algebraic number field and thus at least fulfill condition (2).

Paulis, Cliffords and T-gates

The Clifford+T gate set is the most widely used gate set in quantum computing.

The Pauli matrices

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

$$\mathcal{P}_1 := \langle X, Y, Z \rangle, \mathcal{P}_n := \mathcal{P}_1^{\otimes n}$$

the Pauli group on n qubits.

The Clifford group

$\mathcal{U} := \mathrm{U}_{2^n}(\mathbb{Q}[\zeta_8])$, $\mathcal{C}_n := \mathrm{N}_G(\mathcal{P}_n)$ (up to global phases i.e. scalars) the n-qubit Clifford group.

T-Gates

T-gates on n qubits: $\exp(\frac{i\pi}{8}(P + I)) \in \mathrm{U}_{2^n}(\mathbb{Q}[\zeta_8]),\ P \in \mathcal{P}_n$
Classical T-gate: $\mathrm{diag}(e^{i\pi/4}, 1) = \mathrm{diag}(\zeta_8, 1) = \exp(\frac{i\pi}{8}(Z + I))$.

Our gate set

The Clifford+T gate set is given by $\mathcal{G}_n = \mathcal{C}_n \cup T$-gates.

Cost function

In experiments T-gates are ~ 40 times as expensive as Cliffords. We simplify this and assign T-gates cost 1 and assume Cliffords are free. The corresponding cost function is called the *T-count*.

Exactly executable unitaries

$$G_n := \langle \mathcal{G}_n \rangle = \mathrm{U}_{2^n}\left(\mathbb{Z}\left[\zeta_8, \frac{1}{1+\zeta_8}\right]\right) = \mathrm{U}_{2^n}\left(\mathbb{Z}\left[i, \frac{1}{\sqrt{2}}\right]\right)$$

up to a constraint on the determinant.

Notation

Set $K = \mathbb{Q}(\zeta_8)$ and $\mathfrak{p} := \langle 1 + \zeta_8 \rangle \lhd \mathcal{O}_K$ the unique prime ideal containing 2.
In this notation G_n is the S-arithmetic subgroup $\mathrm{U}_{2^n}(\mathcal{O}_{K,S})$ (with $S = \{\mathfrak{p}\}$), i.e. the group of all elements of $\mathrm{U}(\mathbb{Q}(\zeta_8))$ whose entries are integral “away from $\mathfrak{p}$”.

Task

Perform exact synthesis in $\mathcal{G}_n$ for small n with low/ minimal T-count.

General idea

Observation

U_{2^n} is a connected reductive linear algebraic group (over $\mathbb{Q}(\sqrt{2})$).

- The S-arithmetic group $G_n = \mathrm{U}_{2^n}(\mathcal{O}_{K,\{\mathfrak{p}\}})$ acts on the affine Bruhat-Tits building of U_{2^n} at $\mathfrak{p}$.
- Stabilizers of simplices in this action are arithmetic subgroups and hence finite.

$\rightsquigarrow$ Can use certain substructures of the Bruhat-Tits building to find generators and do exact synthesis.

Problem(s)

- Already for small n the decomposition of the very structured local objects into global ones is highly complicated.
- Even if this works out we don't necessarily get an algorithm to do exact synthesis in the generators we started with (i.e. $\mathcal{G}_n$).

The 1-qubit case

Optimal exact synthesis and state preparation

In the 1-qubit case it is known how to do exact synthesis and state preparation optimally.

- The corresponding Bruhat-Tits building is a tree.
- In particular shortest lengths paths are unique.
- There is a vertex v whose stabilizer is exactly $\mathcal{C}_n$.
- Tv is a neighbours of v and $\mathcal{C}_n$ acts transitively on the neighbours.

$\rightsquigarrow$ We can do exact synthesis by traversing the graph and shortest paths automatically lead to optimal T-count.
State preparation can be performed using a similar idea.

Qutrits

The properties of the 1-qubit case generalize to other architectures as well. For example one can do (provably) optimal exact synthesis and state preparation in the Qutrit-Clifford+R gate set. This works in a different architecture dealing with qutrits (i.e. elements of $\mathbb{C}^3$) instead of qubits.

The setup

Reminder: $K = \mathbb{Q}(\zeta_8)$ with ring of integers $\mathcal{O}_K = \mathbb{Z}[\zeta_8]$, $\mathfrak{p} := \langle 1+\zeta_8 \rangle \supset \langle 2 \rangle = \mathfrak{p}^4$.
Idea: Perform exact synthesis in a gate set closely relatied to $\mathcal{G}_n$ using the affine building but without computing all of it.

Setup

- There is an even, unimodular, hermitian $\mathcal{O}_K$-lattice $L \subset K^{2^n}$ such that
$$\mathcal{C}_n = \mathrm{U}(L) = \{g \in \mathrm{U}_{2^n}(K) \mid gL = L\}.$$
L is called the complex generalized Barnes-Wall lattice.
- For $g \in \mathrm{U}_{2^n}\left(\mathbb{Z}\left[\frac{1}{\sqrt{2}}, i\right]\right)$ we have
$$L/(L \cap gL) \cong gL/(L \cap gL) \cong \mathcal{O}_K/\mathfrak{p}^{d_1} \oplus \ldots \mathcal{O}_K/\mathfrak{p}^{d_{2^n-1}}, d_1 \geq \ldots \geq d_{2^n-1} \geq 0.$$
Set $\mathrm{ED}(L, gL) := x_1^{d_1} \ldots x_{2^n-1}^{d_{2^n-1}}$ and let $\preceq$ be the graded-lexicographical order.
- In a precomputation step we compute a set $\mathcal{T}' \subset \mathrm{U}_{2^n}(K)$ consisting of finitely many (n^2 to be more precise) $\mathcal{C}_n$-conjugacy classes.
- This precomputation step has to be performed only once.

The algorithm

Algorithm 1 Exact synthesis in Clifford+$\mathcal{T}'$

```
function ExactSynthesis
    Input: $g \in \mathrm{U}_{2^n}\left(\mathbb{Z}\left[\frac{1}{\sqrt{2}}, i\right]\right)$
    Output: A sequence $(g_1, \ldots, g_r) \in \mathcal{T}'^r$ such that $g^{-1} g_1 \cdot \ldots \cdot g_r \in \mathcal{C}_n$
    $M \leftarrow gL$
    $w \leftarrow [\,]$
    while $M \neq L$ do
        $h \leftarrow \mathrm{argmin}_{x \in \mathcal{T}'} \mathrm{ED}(xL, M)$
        $M \leftarrow h^{-1} M$
        Append $h$ to $w$
    return $w$
```

Fact sheet

Some facts about the algorithm

- Algorithm was implemented and successfully tested for $n = 2, 3, 4$.
- Correctness only proven for $n = 2$ at this point (computation heavy).
- Algorithm neatly generalizes to other (similarly constructed) gate sets.
- We can (at least for $n = 2$) rewrite the result from the new gate set to $\mathcal{G}_n$.
- In some case (in fact in the majority of experiments) the algorithm comes with a certificate that we have achieved optimal T-count.
- In general we obtain (experimentally) near optimal T-count.
- The approach gives some insight on approximate synthesis.
- A closely related algorithm solves the state preparation problem in the same gate set.

Performance

The Giles-Selinger algorithm

Thus far the only general purpose algorithm for exact synthesis in $\mathcal{G}$ was the Giles-Selinger algorithm reducing the occuring denominators in a unitary column-wise.

Example (Hadamard gate)

Let $H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ and $g := H \otimes H$. Giles-Selinger finds an implementation of g using 24 T-gates, while our algorithm uses 0 T-gates.

Example (Giles-Selinger paper)

Let

$$g := \frac{1}{\sqrt{2}^3}\begin{pmatrix} \zeta_8^2 - \zeta_8 + 1 & \zeta_8^3 + \zeta_8^2 + \zeta_8 & \zeta_8^3 & -\zeta_8^2 \\ \zeta_8^2 + \zeta_8 & -\zeta_8^3 + \zeta_8^2 & -\zeta_8^2 - 1 & \zeta_8^3 + \zeta_8 \\ \zeta_8^3 + \zeta_8^2 & -\zeta_8^3 - 1 & 2\zeta_8^2 & 0 \\ -1 & \zeta_8 & 1 & -\zeta_8^3 + 2\zeta_8 \end{pmatrix}$$

Giles-Selinger implements g using 68 T-gates, while our algorithm returns either 11 or 13 T-gates.

Performance (cont.)

Hidden 1-qubit gate

- $g_1 = \exp(\frac{i\pi}{8}(I_2 + X))\exp(\frac{i\pi}{8}(I_2 + Y))$ a 1-qubit gate of T-count 2 and infinite order.
- $g = g_1 \otimes I_2$ is a 2-qubit gate of T-count 2 and infinite order.
- g^k has T-count $2k$ for $k \in \mathbb{Z}_{\geq 0}$.

k	1	2	3	4	5	6	7
T-count G-S	56	72	88	104	120	136	152
T-count new alg.	2	4	6	8	10	12	14

Future work

Open tasks

- Prove the correctness of the algorithm for $n > 2$.
- Find a way to make the precomputation step more efficient.
- Find a tiebreaker that may lead to optimal synthesis.
- Generalize the algorithm to other gate sets.
- Study the runtime of the algorithm (polynomial time?).