# On "Testing group commutativity" by F.Magniez and A.Nayak

Laura Mancinska

University of Waterloo,
Department of C&O

April 3, 2008

# Introduction

# Black box groups

Black box group model

- Elements of the group are encoded as words over a finite alphabet

# Black box groups

Black box group model

- Elements of the group are encoded as words over a finite alphabet
- Group operation is performed by a black box containing oracles $O_G$ and $O_G^{-1}$

$$O_G \left|g, h\right\rangle = \left|g, gh\right\rangle$$
$$O_G^{-1} \left|g, h\right\rangle = \left|g, g^{-1}h\right\rangle$$

# Black box groups

Black box group model

- Elements of the group are encoded as words over a finite alphabet
- Group operation is performed by a black box containing oracles $O_G$ and $O_G^{-1}$

$$O_G \left|g, h\right\rangle = \left|g, gh\right\rangle$$
$$O_G^{-1} \left|g, h\right\rangle = \left|g, g^{-1}h\right\rangle$$

When do we use black box groups?

# Group Commutativity

*Problem*

*Input:* Generators $g_1, \ldots, g_k$ of $G$ (specified as $n-$bit strings)

# Group Commutativity

*Problem*

*Input:* Generators $g_1, \ldots, g_k$ of $G$ (specified as $n-$bit strings)

*Black box:* Oracles $O_G$ and $O_G^{-1}$

# Group Commutativity

*Problem*

*Input:* Generators $g_1, \ldots, g_k$ of $G$ (specified as $n-$bit strings)

*Black box:* Oracles $O_G$ and $O_G^{-1}$

*Task:* Determine whether $G$ is abelian

# Classical algorithms for Group commutativity

- Naive algorithm with query complexity $\Theta(k^2)$. This is optimal *deterministic* algorithm up to a constant [I.Pak, 2000].

# Classical algorithms for Group commutativity

- Naive algorithm with query complexity $\Theta(k^2)$. This is optimal *deterministic* algorithm up to a constant [I.Pak, 2000].
- Randomized algorithm with query complexity $\Theta(k)$ [I.Pak, 2000]. This is optimal *randomized* algorithm up to a constant [F.Magniez, A.Nayak, 2005]

# Randomized algorithm for group commutativity

*Definition.* Define random subproduct as

$$h = g_1^{a_1} \dots g_k^{a_k},$$

where $a_i \in \{0, 1\}$ are determined by independent tosses of a fair coin.

# Randomized algorithm for group commutativity

*Definition.* Define random subproduct as

$$h = g_1^{a_1} \dots g_k^{a_k},$$

where $a_i \in \{0, 1\}$ are determined by independent tosses of a fair coin.

*Algorithm:*

1. Take two random subproducts $h_1, h_2$

# Randomized algorithm for group commutativity

*Definition.* Define random subproduct as

$$h = g_1^{a_1} \dots g_k^{a_k},$$

where $a_i \in \{0, 1\}$ are determined by independent tosses of a fair coin.

*Algorithm:*

1. Take two random subproducts $h_1, h_2$
2. Test whether $h_1 h_2 = h_2 h_1$

# Randomized algorithm for group commutativity

*Definition.* Define random subproduct as

$$h = g_1^{a_1} \dots g_k^{a_k},$$

where $a_i \in \{0, 1\}$ are determined by independent tosses of a fair coin.

*Algorithm:*

1. Take two random subproducts $h_1, h_2$
2. Test whether $h_1 h_2 = h_2 h_1$
3. Repeat steps 1,2 for $c$ times (to give correct answer with probability at least $1 - \left(\frac{3}{4}\right)^c$)

# Randomized algorithm for group commutativity

*Definition.* Define random subproduct as

$$h = g_1^{a_1} \dots g_k^{a_k},$$

where $a_i \in \{0,1\}$ are determined by independent tosses of a fair coin.

*Algorithm:*

1. Take two random subproducts $h_1, h_2$
2. Test whether $h_1 h_2 = h_2 h_1$
3. Repeat steps 1,2 for $c$ times (to give correct answer with probability at least $1 - \left(\frac{3}{4}\right)^c$)
4. Answer that G is abelian if the tested subproducts commuted

# Randomized algorithm for group commutativity

*Definition.* Define random subproduct as

$$h = g_1^{a_1} \dots g_k^{a_k},$$

where $a_i \in \{0, 1\}$ are determined by independent tosses of a fair coin.

*Algorithm:*

1. Take two random subproducts $h_1, h_2$ ($\leq 2k$ queries)
2. Test whether $h_1 h_2 = h_2 h_1$
3. Repeat steps 1,2 for $c$ times (to give correct answer with probability at least $1 - \left(\frac{3}{4}\right)^c$)
4. Answer that G is abelian if the tested subproducts commuted

# Randomized algorithm for group commutativity

*Definition.* Define random subproduct as

$$h = g_1^{a_1} \dots g_k^{a_k},$$

where $a_i \in \{0, 1\}$ are determined by independent tosses of a fair coin.

*Algorithm:*

1. Take two random subproducts $h_1, h_2$ ($\leq 2k$ queries)
2. Test whether $h_1 h_2 = h_2 h_1$ (2 queries)
3. Repeat steps 1,2 for $c$ times (to give correct answer with probability at least $1 - \left(\frac{3}{4}\right)^c$)
4. Answer that G is abelian if the tested subproducts commuted

# Quantum algorithm

# Main steps

- Construct a random walk on a graph

# Main steps

- Construct a random walk on a graph
- Quantize the random walk using Szegedy's approach

# Main steps

- Construct a random walk on a graph
- Quantize the random walk using Szegedy's approach
- Evaluate the quantities in

$$S + \frac{1}{\sqrt{\delta\varepsilon}}(U + C)$$

# Constructing random walk

$S_l$ – the set of all $l$-tuples of distinct elements from $\{1, \ldots, k\}$

# Constructing random walk

$S_l$ – the set of all $l$-tuples of distinct elements from $\{1, \ldots, k\}$
$g_u := g_{u_1} \ldots g_{u_l}$, where $u = (u_1, \ldots, u_l) \in S_l$

# Constructing random walk

$S_l$ – the set of all $l$-tuples of distinct elements from $\{1, \ldots, k\}$

$g_u := g_{u_1} \ldots g_{u_l}$, where $u = (u_1, \ldots, u_l) \in S_l$

$t_u$ – balanced binary tree with generators $g_{u_1}, \ldots, g_{u_l}$ as leaves

# Constructing random walk

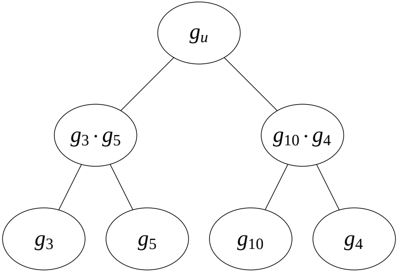
$S_l$ – the set of all $l$-tuples of distinct elements from $\{1, \ldots, k\}$
$g_u := g_{u_1} \ldots g_{u_l}$, where $u = (u_1, \ldots, u_l) \in S_l$
$t_u$ – balanced binary tree with generators $g_{u_1}, \ldots, g_{u_l}$ as leaves

*Example.* Let $l = 4, k = 20, u = \{3, 5, 10, 4\} \in S_4$.

# Constructing random walk

$S_l$ – the set of all $l$-tuples of distinct elements from $\{1, \ldots, k\}$
$g_u := g_{u_1} \ldots g_{u_l}$, where $u = (u_1, \ldots, u_l) \in S_l$
$t_u$ – balanced binary tree with generators $g_{u_1}, \ldots, g_{u_l}$ as leaves

*Example.* Let $l = 4, k = 20, u = \{3, 5, 10, 4\} \in S_4$. Then
$g_u = g_3 \cdot g_5 \cdot g_{10} \cdot g_4$ and $t_u$ looks as follows



# Constructing random walk

*Random walk on* $S_l$

- States are trees $t_u$, $u \in S_l$

# Constructing random walk

*Random walk on $S_l$*

- States are trees $t_u$, $u \in S_l$
- Transitions from each $t_u$ are as follows
    - With probability $1/2$ stay at $t_u$

# Constructing random walk

*Random walk on $S_l$*

- States are trees $t_u$, $u \in S_l$
- Transitions from each $t_u$ are as follows
  - With probability $1/2$ stay at $t_u$
  - With probability $1/2$ do
    1. Pick a random leave position $i \in \{1, \cdots, l\}$ and a random generator index $j \in \{1, \cdots, k\}$

# Constructing random walk

*Random walk on $S_l$*

- States are trees $t_u$, $u \in S_l$
- Transitions from each $t_u$ are as follows
  - With probability $1/2$ stay at $t_u$
  - With probability $1/2$ do
    1. Pick a random leave position $i \in \{1, \cdots, l\}$ and a random generator index $j \in \{1, \cdots, k\}$
    2. If $j = u_m$ for some $m$, exchange $u_i$ and $u_m$, else set $u_i = j$

# Constructing random walk

*Random walk on* $S_l$

- States are trees $t_u$, $u \in S_l$
- Transitions from each $t_u$ are as follows
  - With probability $1/2$ stay at $t_u$
  - With probability $1/2$ do
    1. Pick a random leave position $i \in \{1, \cdots, l\}$ and a random generator index $j \in \{1, \cdots, k\}$
    2. If $j = u_m$ for some $m$, exchange $u_i$ and $u_m$, else set $u_i = j$
    3. Update tree $t_u$

# Constructing quantum walk

We quantize a random walk consisting of two independent random walks on $S_l$

# Constructing quantum walk

We quantize a random walk consisting of two independent random walks on $S_l$

- States are pairs of trees $(t_u, t_v)$, where $u, v \in S_l$

# Constructing quantum walk

We quantize a random walk consisting of two independent random walks on $S_l$

- States are pairs of trees $(t_u, t_v)$, where $u, v \in S_l$
- If transition matrix of the walk on $S_l$ was $P$, then the new transition matrix is $P \otimes P$

# Constructing quantum walk

We quantize a random walk consisting of two independent random walks on $S_l$

- States are pairs of trees $(t_u, t_v)$, where $u, v \in S_l$
- If transition matrix of the walk on $S_l$ was $P$, then the new transition matrix is $P \otimes P$

Vertex $(t_u, t_v)$ is marked iff $g_u g_v \neq g_v g_u$.

# Evaluating parameters – the fraction of marked vertices

*Lemma.* If $G$ is not abelian and $l = o(k)$ then

$$\mathrm{Pr}_{u,v \in S_l}[g_u g_v \neq g_v g_u] \geq const \cdot \left(\frac{l}{k}\right)^2$$

# Evaluating parameters – the fraction of marked vertices

*Lemma.* If $G$ is not abelian and $l = o(k)$ then

$$\mathrm{Pr}_{u,v \in S_l}[g_u g_v \neq g_v g_u] \geq const \cdot \left(\frac{l}{k}\right)^2$$

Thus, fraction of marked vertices, $\varepsilon = \Omega\left(\left(\frac{l}{k}\right)^2\right)$

# Evaluating parameters – the spectral gap

*Lemma.* If $l \leq \frac{k}{2}$, then the spectral gap for the walk on $S_l$ is at least $\frac{1}{8e\ l \log l}$.

# Evaluating parameters – the spectral gap

*Lemma.* If $l \leq \frac{k}{2}$, then the spectral gap for the walk on $S_l$ is at least $\frac{1}{8e\ l \log l}$.

Thus, the spectral gap, $\delta = \Omega\left(\frac{1}{l \log l}\right)$

# Estimating parameters – setup, update and checking cost

- Setup cost, $S = \Theta(l)$

# Estimating parameters – setup, update and checking cost

- Setup cost, $S = \Theta(l)$
- Update cost, $U = \Theta(\log(l))$

# Estimating parameters – setup, update and checking cost

- Setup cost, $S = \Theta(l)$
- Update cost, $U = \Theta(\log(l))$
- Checking cost $C = O(1)$

# Query complexity of the quantum algorithm

| |
|---|
| $\varepsilon = \Omega\left(\left(\frac{l}{k}\right)^2\right)$ |
| $\delta = \Omega(\frac{1}{l \log l})$ |
| $S = \Theta(l)$ |
| $U = \Theta(\log l)$ |
| $C = \Theta(1)$ |

# Query complexity of the quantum algorithm

| |
|---|
| $\varepsilon = \Omega\left(\left(\frac{l}{k}\right)^2\right)$ |
| $\delta = \Omega(\frac{1}{l \log l})$ |
| $S = \Theta(l)$ |
| $U = \Theta(\log l)$ |
| $C = \Theta(1)$ |

$$S + \frac{1}{\sqrt{\delta\varepsilon}}(U + C)$$

# Query complexity of the quantum algorithm

| |
|---|
| $\varepsilon = \Omega\left(\left(\frac{l}{k}\right)^2\right)$ |
| $\delta = \Omega(\frac{1}{l \log l})$ |
| $S = \Theta(l)$ |
| $U = \Theta(\log l)$ |
| $C = \Theta(1)$ |

$$S + \frac{1}{\sqrt{\delta\varepsilon}}(U + C) = O\left(l + \frac{k \log^{3/2} l}{\sqrt{l}}\right)$$

# Query complexity of the quantum algorithm

| |
|---|
| $\varepsilon = \Omega\left(\left(\frac{l}{k}\right)^2\right)$ |
| $\delta = \Omega(\frac{1}{l \log l})$ |
| $S = \Theta(l)$ |
| $U = \Theta(\log l)$ |
| $C = \Theta(1)$ |

$$S + \frac{1}{\sqrt{\delta\varepsilon}}(U + C) = O\left(l + \frac{k \log^{3/2} l}{\sqrt{l}}\right)$$

# Query complexity of the quantum algorithm

| |
|---|
| $\varepsilon = \Omega\left(\left(\frac{l}{k}\right)^2\right)$ |
| $\delta = \Omega(\frac{1}{l \log l})$ |
| $S = \Theta(l)$ |
| $U = \Theta(\log l)$ |
| $C = \Theta(1)$ |

$$S + \frac{1}{\sqrt{\delta\varepsilon}}(U + C) = O\left(l + \frac{k \log^{3/2} l}{\sqrt{l}}\right)$$

To minimize quantum query complexity we set $l = k^{2/3}$ and get

$$O(k^{2/3} \log k)$$

# Lower bounds

## Unique collision

*Black box:* Function $f : \{1, \ldots, k\} \to \{1, \ldots, k\}$

## Unique collision

*Black box:* Function $f : \{1, \ldots, k\} \to \{1, \ldots, k\}$
*Input:* Value of $k$

### Unique collision

*Black box:* Function $f : \{1, \ldots, k\} \to \{1, \ldots, k\}$
*Input:* Value of $k$
*Task:* Output YES if there exists a unique pair $x \neq y \in \{1, \ldots, k\}$ such that $f(x) = f(y)$. Output NO if $f$ is a permutation.

### Unique collision

*Black box:* Function $f : \{1, \ldots, k\} \to \{1, \ldots, k\}$
*Input:* Value of $k$
*Task:* Output YES if there exists a unique pair $x \neq y \in \{1, \ldots, k\}$ such that $f(x) = f(y)$. Output NO if $f$ is a permutation.

### Unique split collision

Output YES if there exists a unique pair $x, y$ such that $x \in \{1, \ldots, \frac{k}{2}\}, y \in \{\frac{k}{2} + 1, \ldots, k\}$ such that $f(x) = f(y)$.

### Theorem

The quantum query complexity of unique split collision is $\Omega(k^{2/3})$.

### Unique collision

*Black box:* Function $f : \{1, \ldots, k\} \rightarrow \{1, \ldots, k\}$
*Input:* Value of $k$
*Task:* Output YES if there exists a unique pair $x \neq y \in \{1, \ldots, k\}$ such that $f(x) = f(y)$. Output NO if $f$ is a permutation.

### Unique split collision

Output YES if there exists a unique pair $x, y$ such that $x \in \{1, \ldots, \frac{k}{2}\}, y \in \{\frac{k}{2} + 1, \ldots, k\}$ such that $f(x) = f(y)$.

### Theorem

The quantum query complexity of unique split collision is $\Omega(k^{2/3})$.

**Theorem**

The quantum query complexity of group commutativity is $\Omega(k^{2/3})$.

**Theorem**

The quantum query complexity of group commutativity is $\Omega(k^{2/3})$.

Idea: Reduce *unique split collision* to group commutativity by constructing a group that is commutative iff function $f$ has a unique split collision.

## Summary

*Problem.* Decide whether group specified by $k$ generators is abelian.

- Classical query complexity is $\Theta(k)$.

# Summary

*Problem.* Decide whether group specified by $k$ generators is abelian.

- Classical query complexity is $\Theta(k)$.
- Quantum query complexity is upper bounded by $O(k^{2/3} \log k)$ (algorithm based on Q-walk) and lower bounded by $\Omega(k^{2/3})$.