Quantum Verification of Matrix Products

Robert Špalek

`sr@cwi.nl`

joint work with Harry Buhrman

Centre for Mathematics and Computer Science

Amsterdam, The Netherlands

Matrix multiplication

Given $n \times n$ matrices A and B, compute $C = AB$.

- School algorithm: time $O(n^3)$

Matrix multiplication

Given $n \times n$ matrices A and B, compute $C = AB$.

- School algorithm: time $O(n^3)$
- [Strassen, 1969]
  Divide and conquer method: time $O(n^{2.807})$
- [Coppersmith & Winograd, 1987]
  Arithmetic progression: time $O(n^{2.376})$

Matrix multiplication

Given $n \times n$ matrices A and B, compute $C = AB$.

- School algorithm: time $O(n^3)$
- [Strassen, 1969]
  Divide and conquer method: time $O(n^{2.807})$
- [Coppersmith & Winograd, 1987]
  Arithmetic progression: time $O(n^{2.376})$
- Best known lower bound is only $\Omega(n^2)$
  The actual complexity is open

Matrix verification

Given $n \times n$ matrices A, B, and C, decide whether $C = AB$.

Matrix verification

Given $n \times n$ matrices A, B, and C, decide whether $C = AB$.

- [Freivalds, 1979]
  Classical algorithm with time $O(n^2)$
  1. Pick a random vector x
  2. Compute $y = Cx$ and $y' = A(Bx)$
  3. Compare y with y'
- Matrix-vector products take time $O(n^2)$
- Constant success probability

Quantum computing

Computers based on laws of quantum physics

- quantum state is a *superposition* of classical states

$$|\psi\rangle = \sum_{x=0}^{2^n-1} \alpha_x |x\rangle, \quad \text{where } \alpha_x \in \mathbb{C} \text{ and } \sum_x |\alpha_x|^2 = 1$$

- computational step is defined by

$$|\psi\rangle \to U|\psi\rangle$$

  for a *unitary* (i.e. norm-preserving) operator U

- outcome is observed by a *measurement*
  the probability of seeing x is $|\alpha_x|^2$

$$\Pr[\Psi = x] = |\alpha_x|^2$$

Quantum algorithms for matrix verification

- [Grover, 1996]
  Searching an unsorted database in time $O(\sqrt{n})$
- [Ambainis, Buhrman, Høyer, Karpinski & Kurur, 2002]
  Matrix verification in time $O(n^{7/4})$ using Freivalds's trick with a random vector, and Grover's search

Quantum algorithms for matrix verification

- [Grover, 1996]
  Searching an unsorted database in time $O(\sqrt{n})$
- [Ambainis, Buhrman, Høyer, Karpinski & Kurur, 2002]
  Matrix verification in time $O(n^{7/4})$ using Freivalds's trick with a random vector, and Grover's search
- [our paper]

Matrix verification in time $O(n^{5/3})$
using two random vectors
and quantum random walks

Quantum random walks

Similar to classical random walks, but with quantum coin flips instead of random coin flips.

Quantum random walks

Similar to classical random walks, but with quantum coin flips instead of random coin flips.

- [Ambainis, 2004] used quantum walks to solve element distinctness (i.e. deciding whether all n input numbers are distinct) in time $O(n^{2/3})$

Quantum random walks

Similar to classical random walks, but with quantum coin flips instead of random coin flips.

- [Ambainis, 2004] used quantum walks to solve element distinctness (i.e. deciding whether all n input numbers are distinct) in time $O(n^{2/3})$
- [Szegedy, 2004] generalized his technique to the problem of finding a marked vertex in an undirected graph G in time

$$O\left(T_{\text{init}} + \frac{1}{\sqrt{\delta\varepsilon}} \cdot (T_{\text{test}} + T_{\text{walk}})\right) ,$$

  - T_{init} is time of picking a uniform superposition of vertices
  - T_{test} is time of testing whether a vertex is marked
  - T_{walk} is time of walking one step over G

Quantum random walks

Similar to classical random walks, but with quantum coin flips instead of random coin flips.

- [Ambainis, 2004] used quantum walks to solve element distinctness (i.e. deciding whether all n input numbers are distinct) in time $O(n^{2/3})$
- [Szegedy, 2004] generalized his technique to the problem of finding a marked vertex in an undirected graph G in time

$$O\left(T_{\text{init}} + \frac{1}{\sqrt{\delta\varepsilon}} \cdot (T_{\text{test}} + T_{\text{walk}})\right) \quad ,$$

  - δ is the spectral gap of G
  - ε is the fraction of marked vertices

Quantum random walks

Similar to classical random walks, but with quantum coin flips instead of random coin flips.

- [Ambainis, 2004] used quantum walks to solve element distinctness (i.e. deciding whether all n input numbers are distinct) in time $O(n^{2/3})$
- [Szegedy, 2004] generalized his technique to the problem of finding a marked vertex in an undirected graph G in time

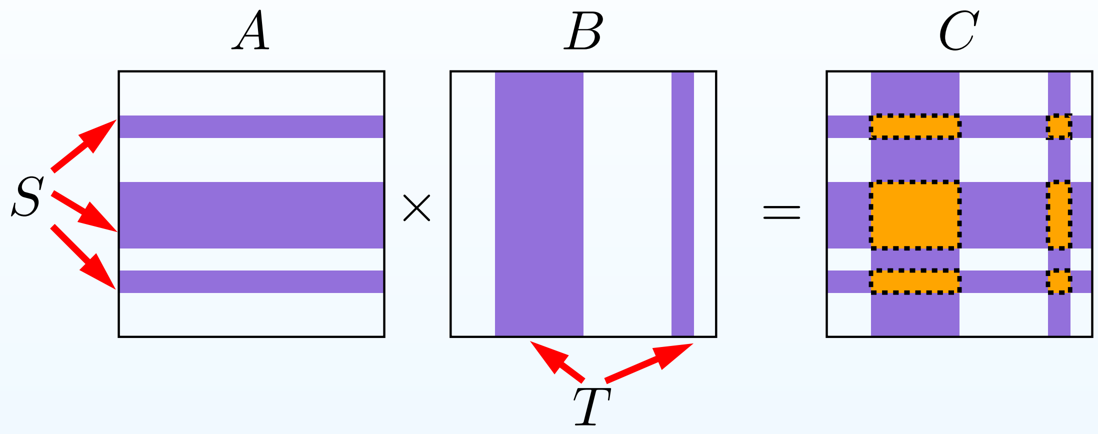
$$O\left(T_{\text{init}} + \frac{1}{\sqrt{\delta\varepsilon}} \cdot (T_{\text{test}} + T_{\text{walk}})\right) \quad ,$$

- Classical random walks converge in time proportional to

$$\frac{1}{\delta\varepsilon}$$

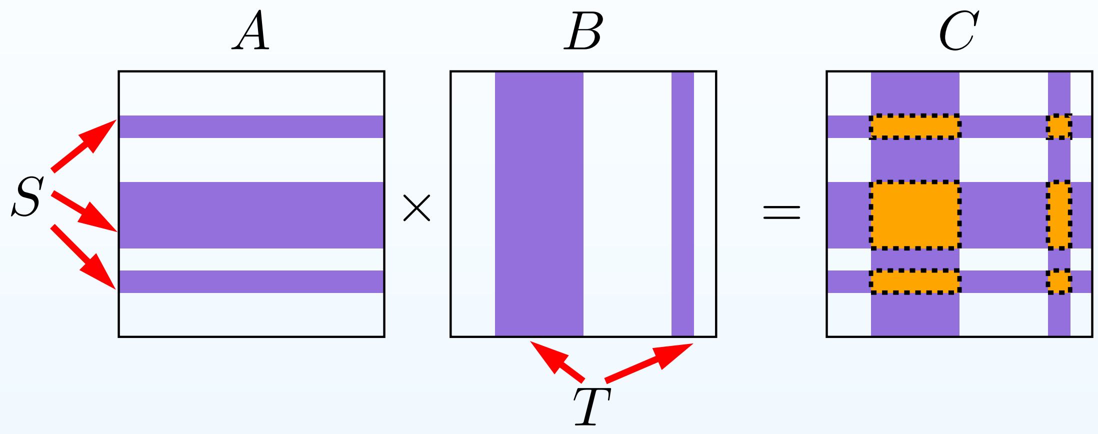
Quantum algorithm for matrix verification

Verification of matrix product $AB = C$



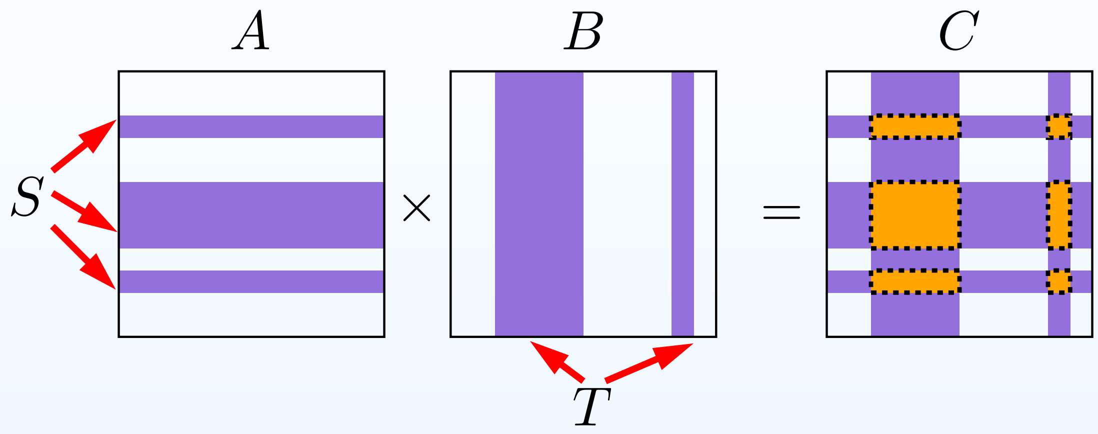
1. *Init* a superposition of subsets $S, T \subseteq [n]$ of size $k = n^{2/3}$.
   Read the rows of A and columns of B specified by S, T.

Verification of matrix product $AB = C$



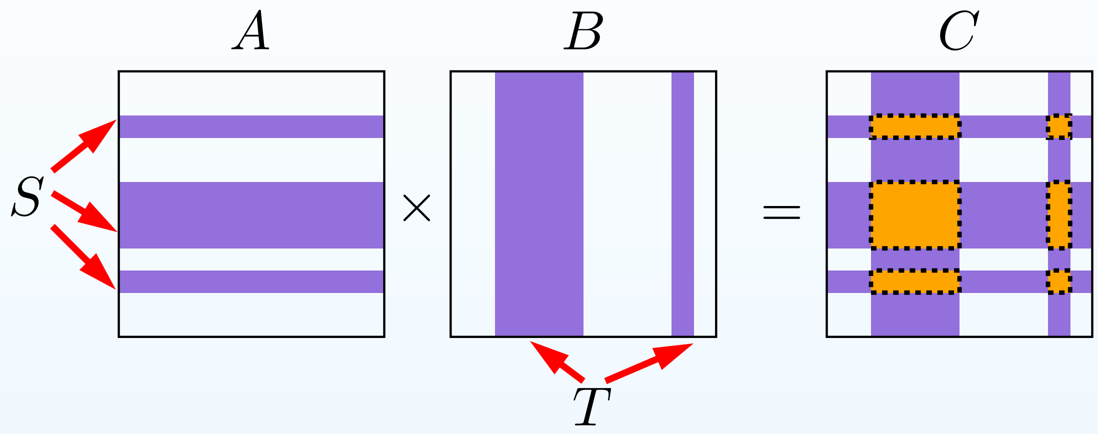
1. *Init* a superposition of subsets $S, T \subseteq [n]$ of size $k = n^{2/3}$.
   Read the rows of A and columns of B specified by S, T.
2. Repeat $n/\sqrt{k}$ times the following:

Verification of matrix product $AB = C$



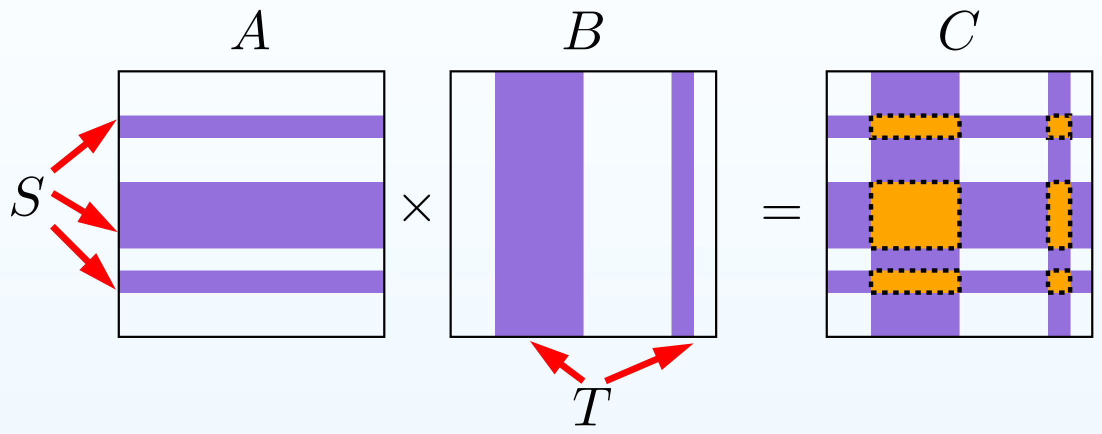
1. *Init* a superposition of subsets $S, T \subseteq [n]$ of size $k = n^{2/3}$.
   Read the rows of A and columns of B specified by S, T.
2. Repeat $n/\sqrt{k}$ times the following:
   (a) *Test* the matrix product restricted to $S \times T$,
   and flip the quantum phase if a wrong entry is found.

Verification of matrix product $AB = C$



1. *Init* a superposition of subsets $S, T \subseteq [n]$ of size $k = n^{2/3}$.
   Read the rows of A and columns of B specified by S, T.
2. Repeat $n/\sqrt{k}$ times the following:
   (a) *Test* the matrix product restricted to $S \times T$,
   and flip the quantum phase if a wrong entry is found.
   (b) *Walk* with S, T by replacing one row and one column.

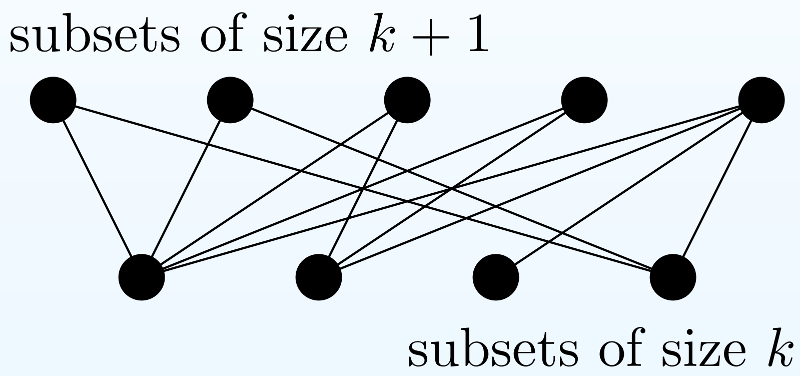
Verification of matrix product $AB = C$



1. *Init* a superposition of subsets $S, T \subseteq [n]$ of size $k = n^{2/3}$.
   Read the rows of A and columns of B specified by S, T.
2. Repeat $n/\sqrt{k}$ times the following:
   (a) *Test* the matrix product restricted to $S \times T$,
   and flip the quantum phase if a wrong entry is found.
   (b) *Walk* with S, T by replacing one row and one column.
3. Measure S, T, and the submatrices,
   and verify classically the restricted matrix product.

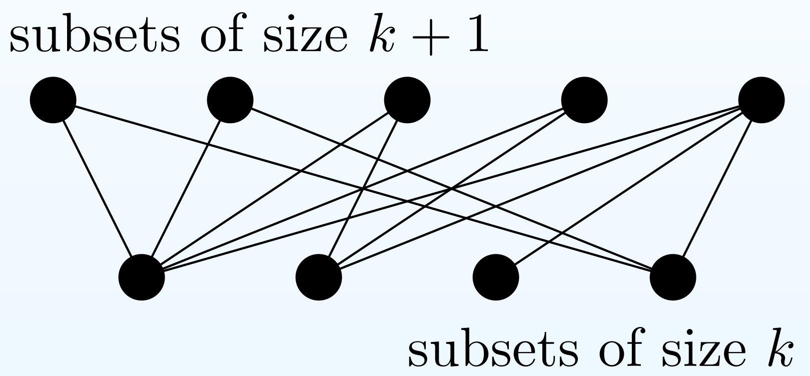
Graph used in the algorithm

- *Johnson graph* $J(n,k)$ has vertices $\binom{[n]}{k} \cup \binom{[n]}{k+1}$ and edges between sets that differ in exactly one item



Graph used in the algorithm

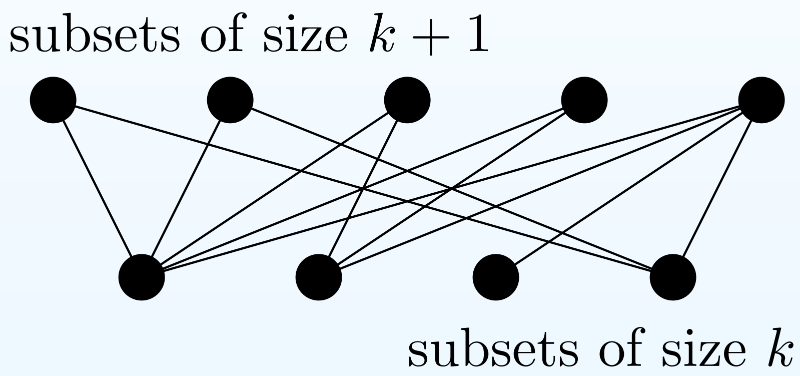
- *Johnson graph* $J(n,k)$ has vertices $\binom{[n]}{k} \cup \binom{[n]}{k+1}$ and edges between sets that differ in exactly one item



- The spectral gap of $J(n,k)$ is $\delta = \Theta(\frac{1}{k})$

Graph used in the algorithm

- *Johnson graph* $J(n,k)$ has vertices $\binom{[n]}{k} \cup \binom{[n]}{k+1}$ and edges between sets that differ in exactly one item



- The spectral gap of $J(n,k)$ is $\delta = \Theta(\frac{1}{k})$
- Our algorithm walks on the *strong product graph* $J(n,k) \times J(n,k)$, which has the same gap

Query complexity of the algorithm

The fraction of subsets S, T containing a wrong entry is $\varepsilon \geq \frac{k^2}{n^2}$; the worst case is exactly one entry.

Query complexity of the algorithm

The fraction of subsets S, T containing a wrong entry is $\varepsilon \geq \frac{k^2}{n^2}$; the worst case is exactly one entry. Hence we need

$$\frac{1}{\sqrt{\delta\varepsilon}} \leq \frac{1}{\sqrt{\frac{1}{k} \cdot \frac{k^2}{n^2}}} = \frac{n}{\sqrt{k}}$$

iterations of the quantum walk of [Szegedy, 2004]

Query complexity of the algorithm

The fraction of subsets S, T containing a wrong entry is $\varepsilon \geq \frac{k^2}{n^2}$; the worst case is exactly one entry. Hence we need

$$\frac{1}{\sqrt{\delta\varepsilon}} \leq \frac{1}{\sqrt{\frac{1}{k} \cdot \frac{k^2}{n^2}}} = \frac{n}{\sqrt{k}}$$

iterations of the quantum walk of [Szegedy, 2004]

- Init: $2kn + k^2$

Query complexity of the algorithm

The fraction of subsets S, T containing a wrong entry is $\varepsilon \geq \frac{k^2}{n^2}$; the worst case is exactly one entry. Hence we need

$$\frac{1}{\sqrt{\delta\varepsilon}} \leq \frac{1}{\sqrt{\frac{1}{k} \cdot \frac{k^2}{n^2}}} = \frac{n}{\sqrt{k}}$$

iterations of the quantum walk of [Szegedy, 2004]

- Init: $2kn + k^2$
- Repeat $n/\sqrt{k}$ times:

Query complexity of the algorithm

The fraction of subsets S, T containing a wrong entry is $\varepsilon \geq \frac{k^2}{n^2}$; the worst case is exactly one entry. Hence we need

$$\frac{1}{\sqrt{\delta\varepsilon}} \leq \frac{1}{\sqrt{\frac{1}{k} \cdot \frac{k^2}{n^2}}} = \frac{n}{\sqrt{k}}$$

iterations of the quantum walk of [Szegedy, 2004]

- Init: $2kn + k^2$
- Repeat $n/\sqrt{k}$ times:
  - Test: 0

Query complexity of the algorithm

The fraction of subsets S, T containing a wrong entry is $\varepsilon \geq \frac{k^2}{n^2}$; the worst case is exactly one entry. Hence we need

$$\frac{1}{\sqrt{\delta\varepsilon}} \leq \frac{1}{\sqrt{\frac{1}{k} \cdot \frac{k^2}{n^2}}} = \frac{n}{\sqrt{k}}$$

iterations of the quantum walk of [Szegedy, 2004]

- Init: $2kn + k^2$
- Repeat $n/\sqrt{k}$ times:
  - Test: 0
  - Walk: $2n + 2k$

Query complexity of the algorithm

The fraction of subsets S, T containing a wrong entry is $\varepsilon \geq \frac{k^2}{n^2}$; the worst case is exactly one entry. Hence we need

$$\frac{1}{\sqrt{\delta\varepsilon}} \leq \frac{1}{\sqrt{\frac{1}{k} \cdot \frac{k^2}{n^2}}} = \frac{n}{\sqrt{k}}$$

iterations of the quantum walk of [Szegedy, 2004]

- Init: $2kn + k^2$
- Repeat $n/\sqrt{k}$ times:
  - Test: 0
  - Walk: $2n + 2k$
- Verify: 0

Query complexity of the algorithm

The fraction of subsets S, T containing a wrong entry is $\varepsilon \geq \frac{k^2}{n^2}$; the worst case is exactly one entry. Hence we need

$$\frac{1}{\sqrt{\delta\varepsilon}} \leq \frac{1}{\sqrt{\frac{1}{k} \cdot \frac{k^2}{n^2}}} = \frac{n}{\sqrt{k}}$$

iterations of the quantum walk of [Szegedy, 2004]

- Init: $2kn + k^2$
- Repeat $n/\sqrt{k}$ times:
  - Test: 0
  - Walk: $2n + 2k$
- Verify: 0

Since $k \leq n$, the query complexity is

$$Q = O(kn + n^2/\sqrt{k})$$

Query complexity of the algorithm

The fraction of subsets S, T containing a wrong entry is $\varepsilon \geq \frac{k^2}{n^2}$; the worst case is exactly one entry. Hence we need

$$\frac{1}{\sqrt{\delta\varepsilon}} \leq \frac{1}{\sqrt{\frac{1}{k} \cdot \frac{k^2}{n^2}}} = \frac{n}{\sqrt{k}}$$

iterations of the quantum walk of [Szegedy, 2004]

- Init: $2kn + k^2$
- Repeat $n/\sqrt{k}$ times:
  - Test: 0
  - Walk: $2n + 2k$
- Verify: 0

Since $k \leq n$, the query complexity is $Q = O(kn + n^2/\sqrt{k})$

Q is minimal for $k = n^{2/3}$, and then it is $O(n^{5/3})$

Improving the running time

The original *running time* is higher than the number of queries due to multiplications of submatrices.

Improving the running time

The original *running time* is higher than the number of queries due to multiplications of submatrices. To fix it,

- We multiply both sides of the equation $AB = C$ by random vectors p, q from left and right. We thus verify $pABq = pCq$.

Improving the running time

The original *running time* is higher than the number of queries due to multiplications of submatrices. To fix it,

- We multiply both sides of the equation $AB = C$ by random vectors p, q from left and right. We thus verify $pABq = pCq$.
- This complicates analysis, because errors can cancel out, e.g. in $\mathbb{GF}(2)$. However, this can be handled.

Improving the running time

The original *running time* is higher than the number of queries due to multiplications of submatrices. To fix it,

- We multiply both sides of the equation $AB = C$ by random vectors p, q from left and right. We thus verify $pABq = pCq$.
- This complicates analysis, because errors can cancel out, e.g. in $\mathbb{GF}(2)$. However, this can be handled.
- Instead of keeping the submatrices of A and B in the memory, we update matrix-vector products pA and Bq, and the number pCq.

Improving the running time

The original *running time* is higher than the number of queries due to multiplications of submatrices. To fix it,

- We multiply both sides of the equation $AB = C$ by random vectors p, q from left and right. We thus verify $pABq = pCq$.
- This complicates analysis, because errors can cancel out, e.g. in $\mathbb{GF}(2)$. However, this can be handled.
- Instead of keeping the submatrices of A and B in the memory, we update matrix-vector products pA and Bq, and the number pCq.
- This decreases both
  1. running time of *testing* $(pA)(Bq) = pCq$

Improving the running time

The original *running time* is higher than the number of queries due to multiplications of submatrices. To fix it,

- We multiply both sides of the equation $AB = C$ by random vectors p, q from left and right. We thus verify $pABq = pCq$.
- This complicates analysis, because errors can cancel out, e.g. in $\mathbb{GF}(2)$. However, this can be handled.
- Instead of keeping the submatrices of A and B in the memory, we update matrix-vector products pA and Bq, and the number pCq.
- This decreases both
  1. running time of *testing* $(pA)(Bq) = pCq$
  2. space complexity

Matrix multiplication

- With $1 < w \leq \sqrt{n}$ wrong entries, the running time is faster $O(n^{5/3}/w^{1/3})$.

Matrix multiplication

- With $1 < w \leq \sqrt{n}$ wrong entries, the running time is faster $O(n^{5/3}/w^{1/3})$.
- Let $AB = C$ contain w non-zero entries.
  Compute C as follows:
  1. Set C to a zero matrix.
  2. Run verification until it outputs "correct", recomputing wrong entries.

Matrix multiplication

- With $1 < w \leq \sqrt{n}$ wrong entries, the running time is faster $O(n^{5/3}/w^{1/3})$.
- Let $AB = C$ contain w non-zero entries.
  Compute C as follows:
  1. Set C to a zero matrix.
  2. Run verification until it outputs “correct”, recomputing wrong entries.
- Total running time is

$$\sum_{\ell=1}^{w} \frac{n^{5/3}}{\ell^{1/3}} = O(n^{5/3} w^{2/3}) = O(n^2) \text{ for } w \leq \sqrt{n} \ .$$

Matrix multiplication

- With $1 < w \leq \sqrt{n}$ wrong entries, the running time is faster $O(n^{5/3}/w^{1/3})$.
- Let $AB = C$ contain w non-zero entries.
  Compute C as follows:
  1. Set C to a zero matrix.
  2. Run verification until it outputs "correct", recomputing wrong entries.
- Total running time is

$$\sum_{\ell=1}^{w} \frac{n^{5/3}}{\ell^{1/3}} = O(n^{5/3}w^{2/3}) = O(n^2) \text{ for } w \leq \sqrt{n} \ .$$

- However, for $w > \sqrt{n}$, the speedup of verification is smaller, and the classical algorithm by [Indyk, 2005] with time $O(n^2 + nw)$ takes over.

Summary and open problems

- Matrix verification in $O(n^{5/3})$ queries using quantum walks
- Improved running time and space complexity using two random vectors
- Verification is faster with many wrong entries
- Fast matrix multiplication

Summary and open problems

- Matrix verification in $O(n^{5/3})$ queries using quantum walks
- Improved running time and space complexity using two random vectors
- Verification is faster with many wrong entries
- Fast matrix multiplication

- Can we multiply dense matrices faster than classically?
- Matching lower bound?