Quantum Time-Space Tradeoffs for Deciding Systems of Linear Inequalities

Robert Špalek

sr@cwi.nl

joint work with Andris Ambainis and Ronald de Wolf

quant-ph/0511200

Time-Space Tradeoffs

- A relation between the running time and space complexity

The more memory is available,
the faster the algorithm can possibly run.

Time-Space Tradeoffs

- A relation between the running time and space complexity

The more memory is available,
the faster the algorithm can possibly run.

- Example: sorting of N numbers

$$TS = N^2$$

Systems of Linear Inequalities

- Let A be a fixed $N \times N$ Boolean matrix
  Let x, b be integer input vectors of length N
- The task is to output for each row whether

$$Ax \geq b$$

Systems of Linear Inequalities

- Let A be a fixed $N \times N$ Boolean matrix
  Let x, b be integer input vectors of length N
- The task is to output for each row whether

$$Ax \geq b$$

- We study the query complexity with bounded error
  - Classically

$$TS = N^2$$

Systems of Linear Inequalities

- Let A be a fixed $N \times N$ Boolean matrix
  Let x, b be integer input vectors of length N
- The task is to output for each row whether

$$Ax \geq b$$

- We study the query complexity with bounded error
  - Classically

$$TS = N^2$$

  - Quantumly

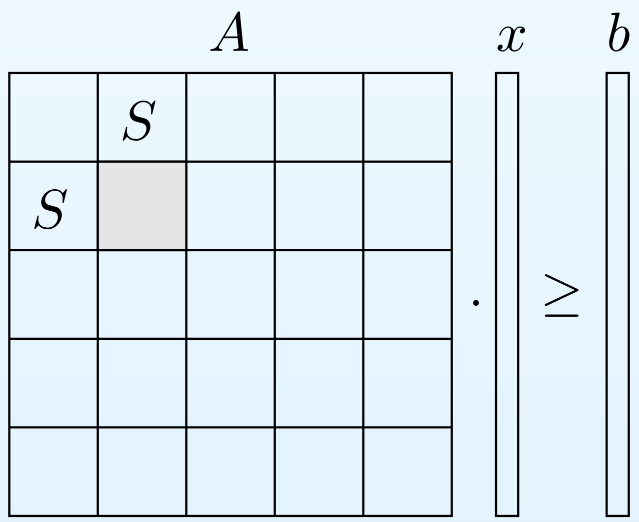
$$T^2 S = N^3 t, \quad S \leq N/t$$
$$TS = N^2, \quad S > N/t$$

    if numbers in b are *at most t*
- Omitting log-factors in the upper bounds

Upper Bound

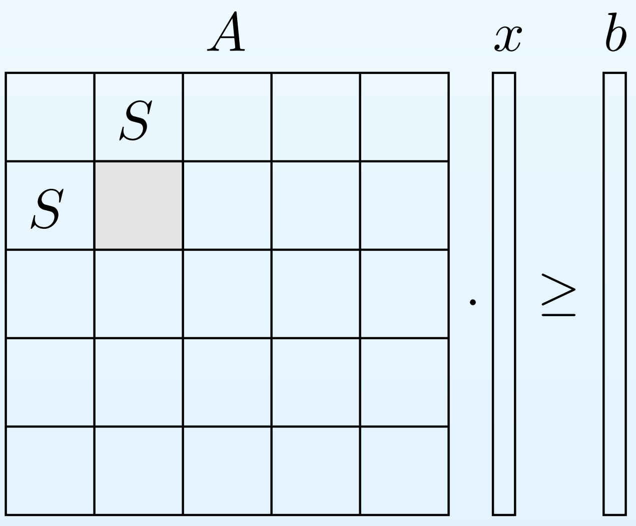
Classical Algorithm

- Split the matrix into $(N/S)^2$ blocks of size $S \times S$



Classical Algorithm

- Split the matrix into $(N/S)^2$ blocks of size $S \times S$
- Evaluate the output row-wise
  - maintain S counters at the same time
  - in each of the N/S blocks, read S inputs and update all counters using the *fixed* matrix A

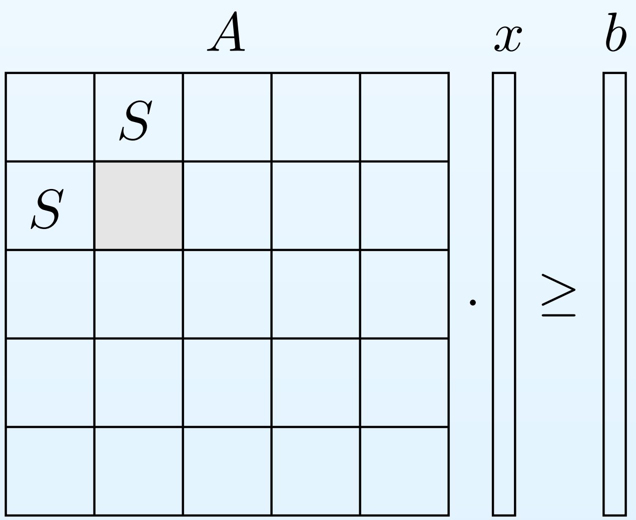


Classical Algorithm

- Split the matrix into $(N/S)^2$ blocks of size $S \times S$
- Evaluate the output row-wise
  - maintain S counters at the same time
  - in each of the N/S blocks, read S inputs and update all counters using the *fixed* matrix A
- The query complexity is

$$T = \frac{N}{S} \cdot \left(\frac{N}{S} \cdot S \right) = \frac{N^2}{S}$$

when the space is S



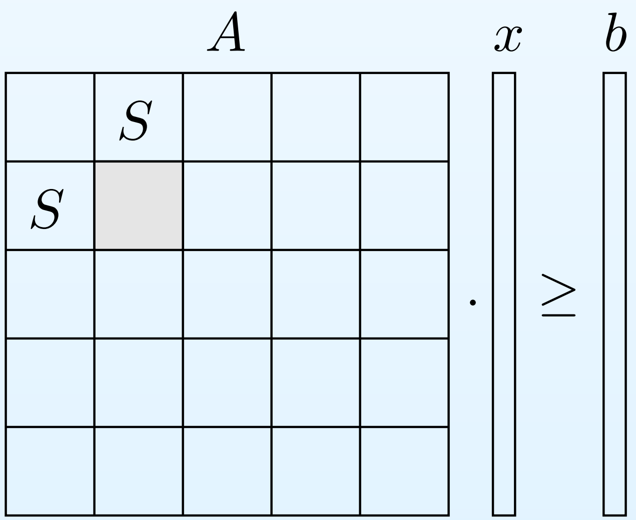
Classical Algorithm

- Split the matrix into $(N/S)^2$ blocks of size $S \times S$
- Evaluate the output row-wise
  - maintain S counters at the same time
  - in each of the N/S blocks, read S inputs and update all counters using the *fixed* matrix A
- The query complexity is

$$T = \frac{N}{S} \cdot \left(\frac{N}{S} \cdot S \right) = \frac{N^2}{S}$$

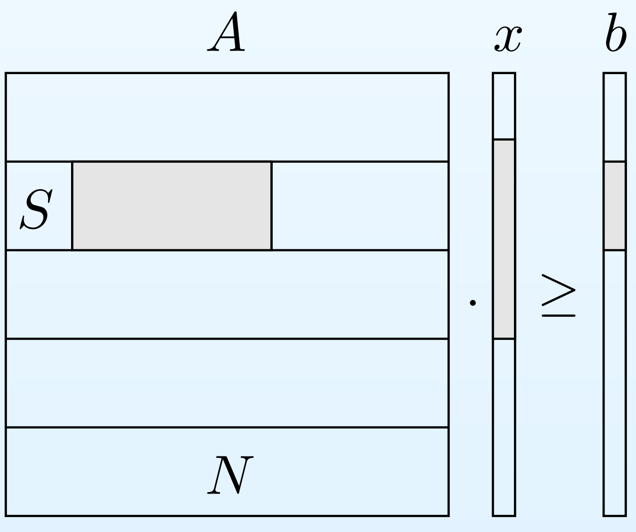
when the space is S

$TS \leq N^2$



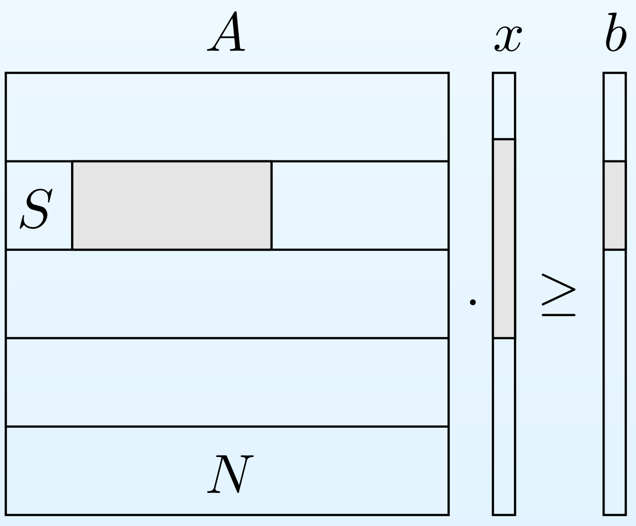
Quantum Algorithm

- Split the matrix into N/S row blocks of height S



Quantum Algorithm

- Split the matrix into N/S row blocks of height S
- Evaluate the output row-wise
  - maintain S counters at the same time
  - use *quantum counting* and *Grover search* to find non-zero inputs
  - the speedup is $N \to \sqrt{NSt}$ per row block

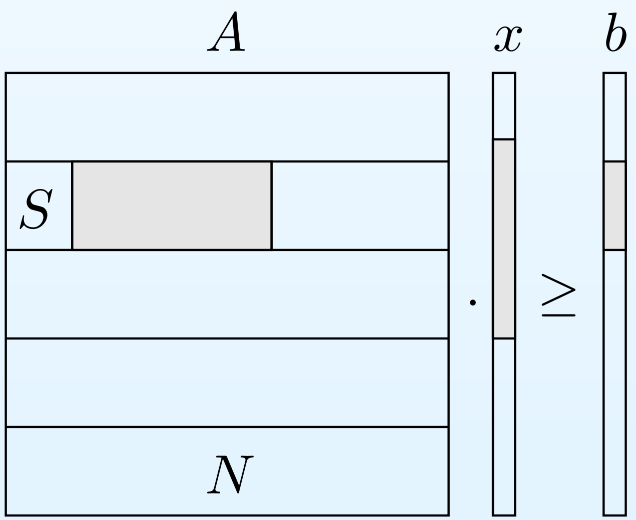


Quantum Algorithm

- Split the matrix into N/S row blocks of height S
- Evaluate the output row-wise
  - maintain S counters at the same time
  - use *quantum counting* and *Grover search* to find non-zero inputs
  - the speedup is $N \to \sqrt{NSt}$ per row block
- The query complexity is

$$T = \frac{N}{S} \cdot \sqrt{NSt} = N^{3/2}\sqrt{\frac{t}{S}}$$

when the space is S



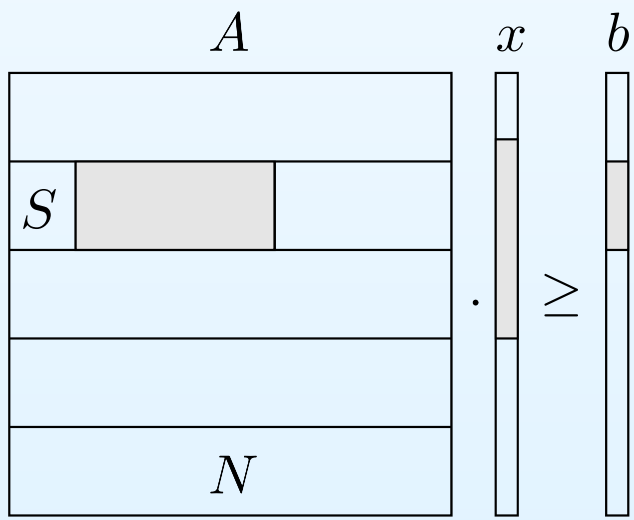
Quantum Algorithm

- Split the matrix into N/S row blocks of height S
- Evaluate the output row-wise
  - maintain S counters at the same time
  - use *quantum counting* and *Grover search* to find non-zero inputs
  - the speedup is $N \to \sqrt{NSt}$ per row block
- The query complexity is

$$T = \frac{N}{S} \cdot \sqrt{NSt} = N^{3/2}\sqrt{\frac{t}{S}}$$

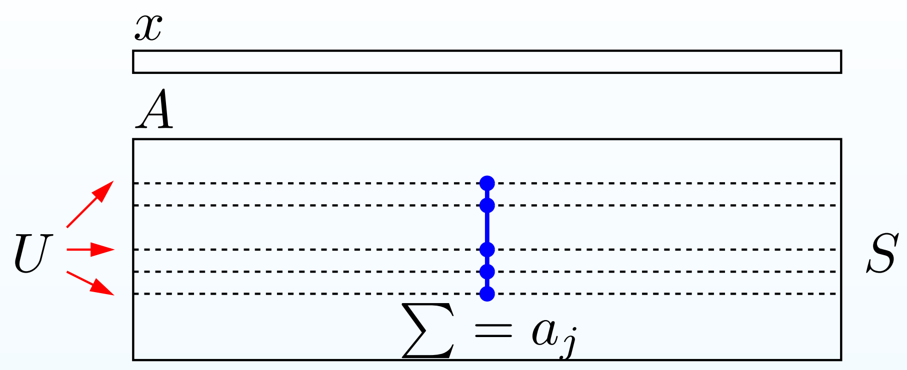
when the space is S

$$T^2 S \leq N^3 t$$



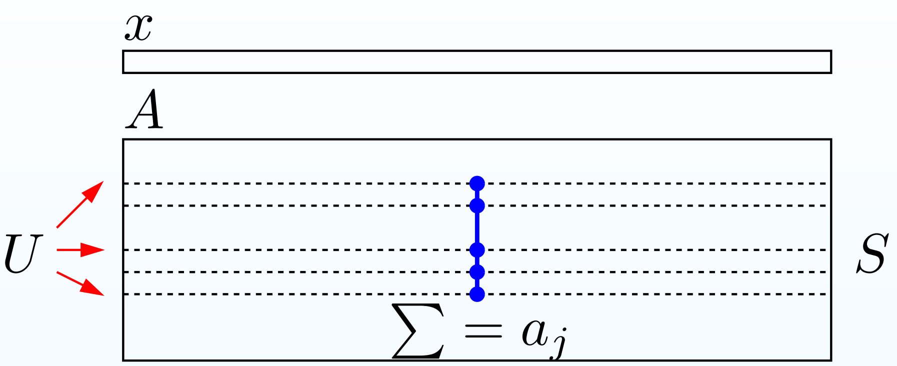
Quantum Algorithm (cont.)

y	vector of counters
U	set of *open rows* with $y_i < b_i$
a	column sum of A over the rows from U



Quantum Algorithm (cont.)

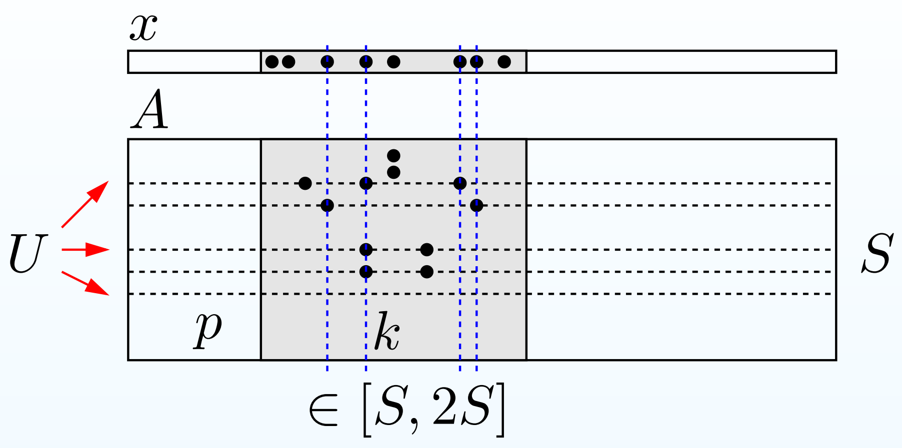
y	vector of counters
U	set of *open rows* with $y_i < b_i$
a	column sum of A over the rows from U



Start at position $p \leftarrow 1$ and with $U \leftarrow [1, S]$.

Quantum Algorithm (cont.)

y	vector of counters
U	set of *open rows* with $y_i < b_i$
a	column sum of A over the rows from U



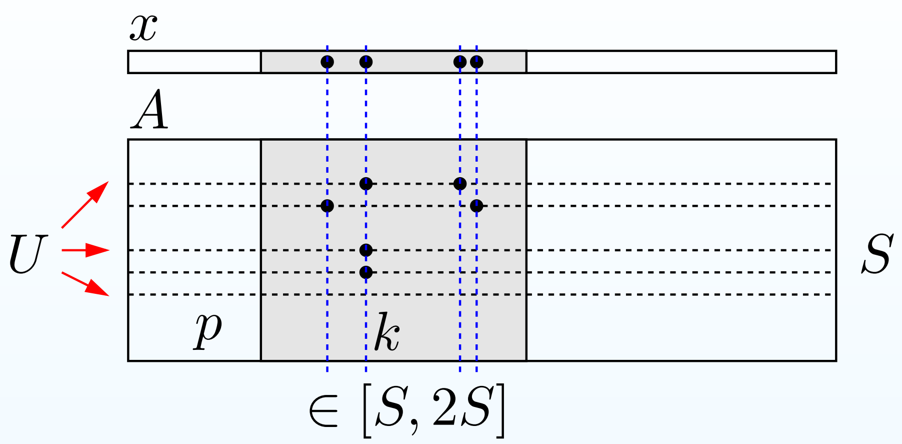
While $p \le N$ and $U \neq \emptyset$, do

- Find by binary search some k such that

$$S \le \sum_{j=p}^{p+k-1} a_j x_j \le 2S \qquad \text{. . . quantum counting}$$

Quantum Algorithm (cont.)

y	vector of counters
U	set of *open rows* with $y_i < b_i$
a	column sum of A over the rows from U



While $p \leq N$ and $U \neq \emptyset$, do

- Find by binary search some k such that

$$S \leq \sum_{j=p}^{p+k-1} a_j x_j \leq 2S$$

...quantum counting

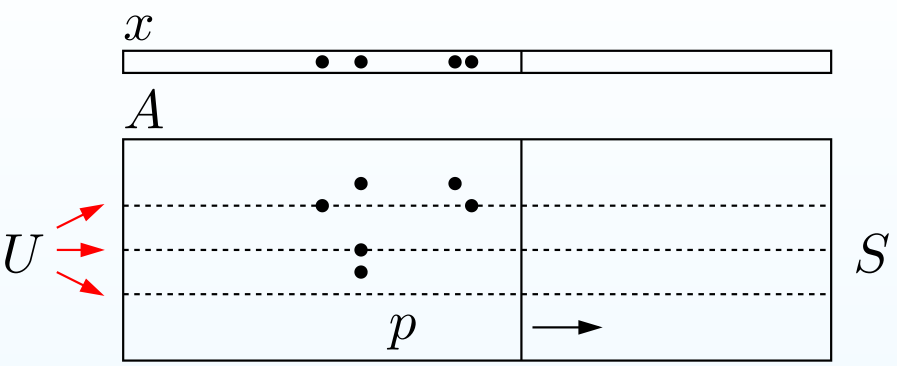
- Find all positions j inside $[p, p+k-1]$ such that

$$a_j x_j > 0$$

...quantum search

Quantum Algorithm (cont.)

y	vector of counters
U	set of *open rows* with $y_i < b_i$
a	column sum of A over the rows from U



While $p \leq N$ and $U \neq \emptyset$, do

- Find by binary search some k such that
$$S \leq \sum_{j=p}^{p+k-1} a_j x_j \leq 2S$$
...quantum counting
- Find all positions j inside $[p, p+k-1]$ such that
$$a_j x_j > 0$$
...quantum search
- Update the counters y, remove from U the rows that have been closed in this iteration, and set $p \leftarrow p + k$

Complexity of the Algorithm

$i:$	1	2	3	4	5

In the i-th iteration of length k_i,

- cost of quantum counting with $\sqrt{k_i}$ queries is negligible
- quantum search costs $\sqrt{k_i r_i t} + \sqrt{k_i s_i}$, where
  - r_i is the number of closed rows
  - s_i is the total number added to counters in this iteration

Complexity of the Algorithm

$i:$	1	2	3	4	5

In the i-th iteration of length k_i,

- cost of quantum counting with $\sqrt{k_i}$ queries is negligible
- quantum search costs $\sqrt{k_i r_i t} + \sqrt{k_i s_i}$, where
  - r_i is the number of closed rows
  - s_i is the total number added to counters in this iteration

By Cauchy-Schwarz,

$$
\begin{aligned}
T &= \sum_i \left(\sqrt{k_i r_i t} + \sqrt{k_i s_i} \right) \\
&\leq \sqrt{\sum k_i} \sqrt{t \sum r_i} + \sqrt{\sum k_i} \sqrt{\sum s_i} \\
&\leq \sqrt{NSt}
\end{aligned}
$$

Lower Bound

Direct Product Theorems

- Suppose we need $T(f)$ queries to compute f with small error. How hard is it to compute k independent instances $f(x_1), \ldots, f(x_k)$?

Direct Product Theorems

- Suppose we need $T(f)$ queries to compute f with small error. How hard is it to compute k independent instances $f(x_1), \ldots, f(x_k)$?
- Relation between total number of queries T and overall success probability σ:

$$T \leq \alpha k \cdot T(f) \Rightarrow \sigma \leq 2^{-\gamma k}$$

α, γ are small positive constants

Direct Product Theorems

- Suppose we need $T(f)$ queries to compute f with small error. How hard is it to compute k independent instances $f(x_1), \ldots, f(x_k)$?
- Relation between total number of queries T and overall success probability σ:

$$T \leq \alpha k \cdot T(f) \Rightarrow \sigma \leq 2^{-\gamma k}$$

  α, γ are small positive constants

- It is not known, whether the DPT holds in general!

  [Shaltiel, 2001]
  Counterexample for average-case complexity.
  However, DPT plausible for worst-case complexity.

Symmetric Functions

A function f is symmetric iff
it only depends on the Hamming weight of the input

Symmetric Functions

A function f is symmetric iff
it only depends on the Hamming weight of the input

- *Implicit threshold* is the minimal t such that f is constant on $[t, n-t]$. Example:
  - OR and AND have $t = 1$
  - parity and majority have $t = \frac{n}{2}$
  - a-threshold function with $a \leq \frac{n}{2}$ has $t = a$

Symmetric Functions

A function f is symmetric iff
it only depends on the Hamming weight of the input

- *Implicit threshold* is the minimal t such that f is constant on $[t, n-t]$. Example:
  - OR and AND have $t = 1$
  - parity and majority have $t = \frac{n}{2}$
  - a-threshold function with $a \leq \frac{n}{2}$ has $t = a$
- Bounded-error quantum query complexity of a symmetric function is

$$Q_2(f) = \Theta(\sqrt{tn})$$

Quantum Query DPT

- [Klauck, Š, de Wolf, FOCS 2004]
  DPT for k instances of the OR function

$$T \leq \alpha k \sqrt{n} \Rightarrow \sigma \leq 2^{-\gamma k}$$

  using the polynomial method

Quantum Query DPT

- [Klauck, Š, de Wolf, FOCS 2004]
  DPT for k instances of the OR function

  $$T \leq \alpha k \sqrt{n} \Rightarrow \sigma \leq 2^{-\gamma k}$$

  using the polynomial method
- [Ambainis, 2005]
  Reproves [KŠW] using adversary arguments.

Quantum Query DPT

- [Klauck, Š, de Wolf, FOCS 2004]
  DPT for k instances of the OR function

  $$T \leq \alpha k \sqrt{n} \Rightarrow \sigma \leq 2^{-\gamma k}$$

  using the polynomial method
- [Ambainis, 2005]
  Reproves [KŠW] using adversary arguments.
- [Ambainis, Š, de Wolf, 2005]
  Generalize [KŠW, Amb] to all symmetric functions f.

  $$T \leq \alpha k \sqrt{tn} \Rightarrow \sigma \leq 2^{-\gamma k}$$

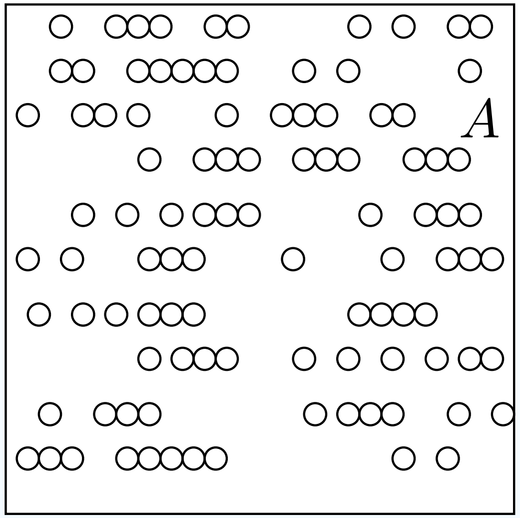
  t is the implicit threshold of f

Constructing a Hard Matrix

[Klauck, Š, de Wolf, 2004]
Using probabilistic method,

- For every $k = o(N/\log N)$,

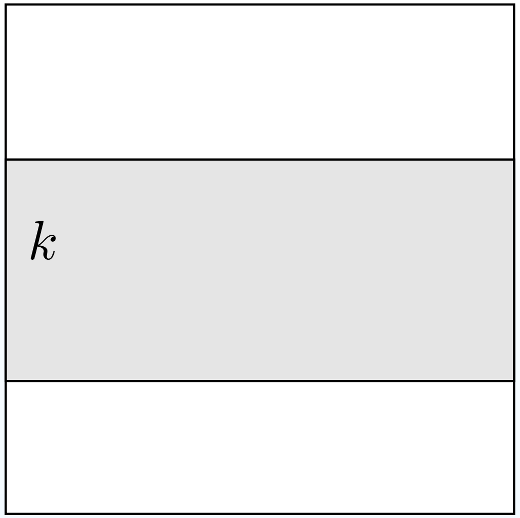
Constructing a Hard Matrix



[Klauck, Š, de Wolf, 2004]
Using probabilistic method,

- For every $k = o(N/\log N)$,
- there is an $N \times N$ Boolean matrix A such that all rows of A have weight $N/2k$,

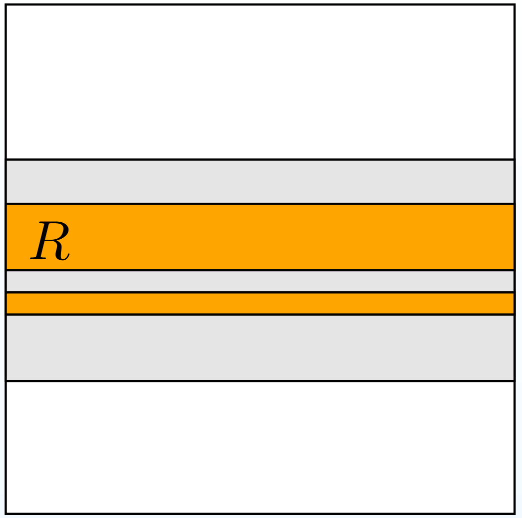
Constructing a Hard Matrix



[Klauck, Š, de Wolf, 2004]
Using probabilistic method,

- For every $k = o(N/\log N)$,
- there is an $N \times N$ Boolean matrix A such that all rows of A have weight $N/2k$,
- and every set of k rows of A

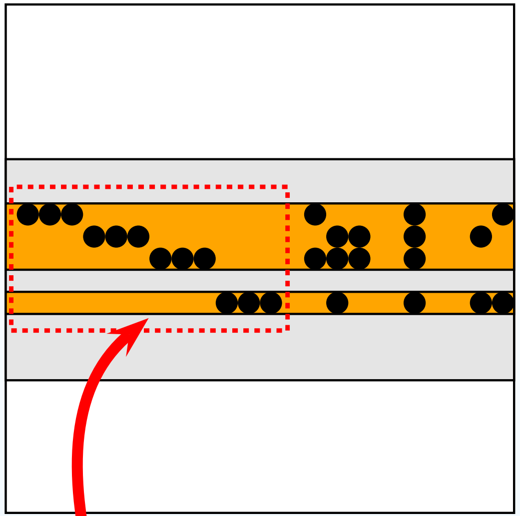
Constructing a Hard Matrix



[Klauck, Š, de Wolf, 2004]
Using probabilistic method,

- For every $k = o(N/\log N)$,
- there is an $N \times N$ Boolean matrix A such that all rows of A have weight $N/2k$,
- and every set of k rows of A
- contains a set R of $k/2$ rows with the following property:

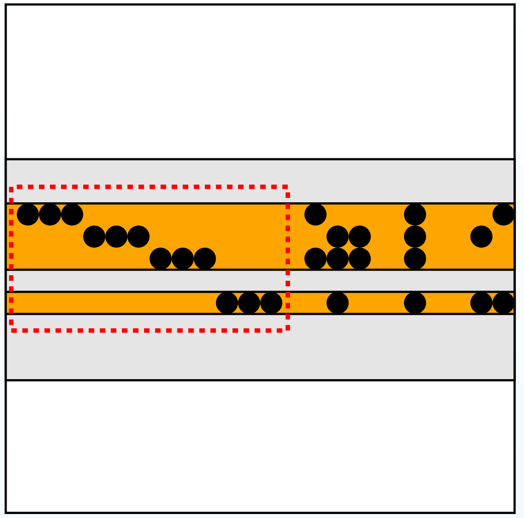
Constructing a Hard Matrix



[Klauck, Š, de Wolf, 2004]
Using probabilistic method,

- For every $k = o(N/\log N)$,
- there is an $N \times N$ Boolean matrix A such that all rows of A have weight $N/2k$,
- and every set of k rows of A
- contains a set R of $k/2$ rows with the following property:
- each row in R contains at least $N/6k$ ones that occur in no other row of R.

Constructing a Hard Matrix



[Klauck, Š, de Wolf, 2004]
Using probabilistic method,

- For every $k = o(N/\log N)$,
- there is an $N \times N$ Boolean matrix A such that all rows of A have weight $N/2k$,
- and every set of k rows of A
- contains a set R of $k/2$ rows with the following property:
- each row in R contains at least $N/6k$ ones that occur in no other row of R.

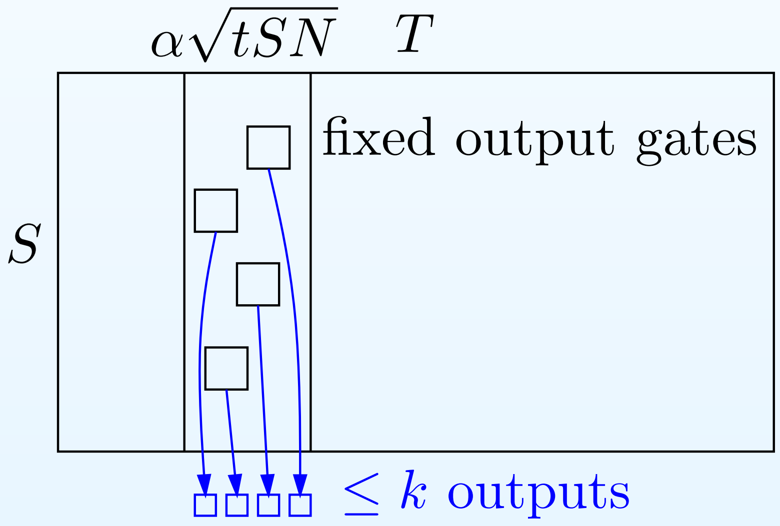
Proof: pick $N/2k$ ones at random in each row.

Lower Bound for the System of Linear Inequalities

- Slice the circuit into $\frac{T}{\alpha\sqrt{tSN}}$ slices, each containing $\alpha\sqrt{tSN}$ queries

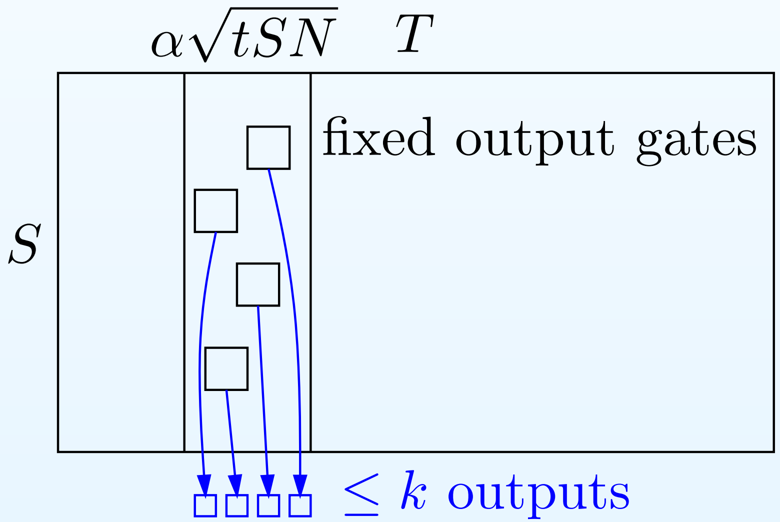
Lower Bound for the System of Linear Inequalities

- Slice the circuit into $\frac{T}{\alpha\sqrt{tSN}}$ slices, each containing $\alpha\sqrt{tSN}$ queries
- Let k be the maximal number of output gates in a slice



Lower Bound for the System of Linear Inequalities

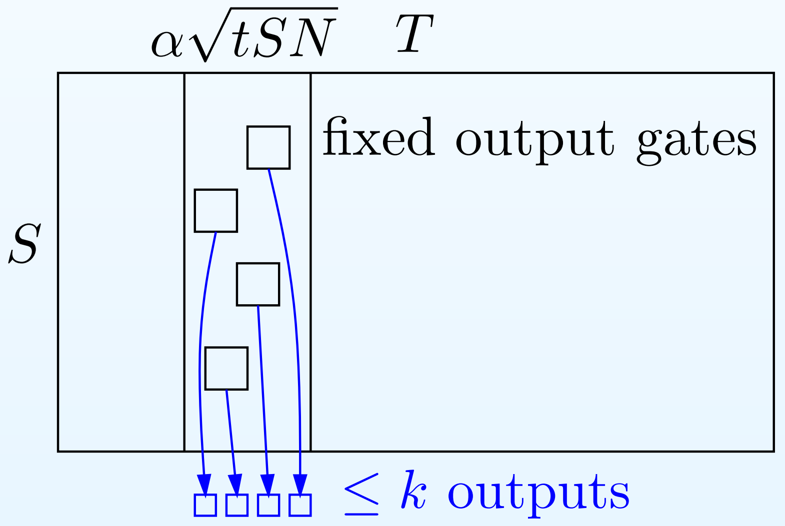
- Slice the circuit into $\frac{T}{\alpha\sqrt{tSN}}$ slices, each containing $\alpha\sqrt{tSN}$ queries
- Let k be the maximal number of output gates in a slice



- We show that $k = O(S)$ due to the DPT

Lower Bound for the System of Linear Inequalities

- Slice the circuit into $\frac{T}{\alpha\sqrt{tSN}}$ slices, each containing $\alpha\sqrt{tSN}$ queries
- Let k be the maximal number of output gates in a slice



- We show that $k = O(S)$ due to the DPT
- $N \leq$ # slices $\cdot\, k = O\left(\frac{T\sqrt{S}}{\alpha\sqrt{tN}}\right)$, hence $T^2 S = \Omega(N^3 t)$

Each Slice Has Only Few Output Gates

If $k < S$, then certainly $k = O(S)$, so assume $k \geq S$

Each Slice Has Only Few Output Gates

If $k < S$, then certainly $k = O(S)$, so assume $k \geq S$

- Within the maximal slice, the circuit outputs whether $(Ax)_i \geq b_i$ for k distinct rows i with overall probability $\geq 2/3$

Each Slice Has Only Few Output Gates

If $k < S$, then certainly $k = O(S)$, so assume $k \geq S$

- Within the maximal slice, the circuit outputs whether $(Ax)_i \geq b_i$ for k distinct rows i with overall probability $\geq 2/3$
- Use the hard matrix A with many disjoint ones. The algorithm computes $k/2$ independent t-threshold functions with $n = N/6k$ bits each.

Each Slice Has Only Few Output Gates

If $k < S$, then certainly $k = O(S)$, so assume $k \geq S$

- Within the maximal slice, the circuit outputs whether $(Ax)_i \geq b_i$ for k distinct rows i with overall probability $\geq 2/3$
- Use the hard matrix A with many disjoint ones. The algorithm computes $k/2$ independent t-threshold functions with $n = N/6k$ bits each.
- Replace S-qubit starting state by completely mixed state; overlap with correct state is 2^{-S}, hence we get a circuit for $\text{Threshold}_{n,t}^{(k/2)}$ with probability $\sigma \geq \frac{2}{3} \cdot 2^{-S}$

Each Slice Has Only Few Output Gates

If $k < S$, then certainly $k = O(S)$, so assume $k \geq S$

- Within the maximal slice, the circuit outputs whether $(Ax)_i \geq b_i$ for k distinct rows i with overall probability $\geq 2/3$
- Use the hard matrix A with many disjoint ones. The algorithm computes $k/2$ independent t-threshold functions with $n = N/6k$ bits each.
- Replace S-qubit starting state by completely mixed state; overlap with correct state is 2^{-S}, hence we get a circuit for $\text{Threshold}_{n,t}^{(k/2)}$ with probability $\sigma \geq \frac{2}{3} \cdot 2^{-S}$
- However the number of queries $T = \alpha\sqrt{tSN} \leq \alpha\sqrt{tkN} = \alpha k\sqrt{tn}$, hence by DPT $\sigma \leq 2^{-\gamma k}$

Each Slice Has Only Few Output Gates

If $k < S$, then certainly $k = O(S)$, so assume $k \geq S$

- Within the maximal slice, the circuit outputs whether $(Ax)_i \geq b_i$ for k distinct rows i with overall probability $\geq 2/3$
- Use the hard matrix A with many disjoint ones. The algorithm computes $k/2$ independent t-threshold functions with $n = N/6k$ bits each.
- Replace S-qubit starting state by completely mixed state; overlap with correct state is 2^{-S}, hence we get a circuit for $\text{Threshold}_{n,t}^{(k/2)}$ with probability $\sigma \geq \frac{2}{3} \cdot 2^{-S}$
- However the number of queries $T = \alpha\sqrt{tSN} \leq \alpha\sqrt{tkN} = \alpha k\sqrt{tn}$, hence by DPT $\sigma \leq 2^{-\gamma k}$

Conclude that $k = O(S)$

Each Slice Has Only Few Output Gates

If $k < S$, then certainly $k = O(S)$, so assume $k \geq S$

- Within the maximal slice, the circuit outputs whether $(Ax)_i \geq b_i$ for k distinct rows i with overall probability $\geq 2/3$
- Use the hard matrix A with many disjoint ones. The algorithm computes $k/2$ independent t-threshold functions with $n = N/6k$ bits each. $\Longleftarrow$ we need $t \leq n/2 = O(N/S)$
- Replace S-qubit starting state by completely mixed state; overlap with correct state is 2^{-S}, hence we get a circuit for $\text{Threshold}_{n,t}^{(k/2)}$ with probability $\sigma \geq \frac{2}{3} \cdot 2^{-S}$
- However the number of queries $T = \alpha\sqrt{tSN} \leq \alpha\sqrt{tkN} = \alpha k\sqrt{tn}$, hence by DPT $\sigma \leq 2^{-\gamma k}$

Conclude that $k = O(S)$

Lower bound for one-sided error

- Stronger direct product theorem for threshold functions for one-sided error algorithms that never say $|x| \geq t$ in any instance if it is not true

$$T \leq \alpha k \sqrt{tn} \Rightarrow \sigma \leq 2^{-\gamma k t}$$

Lower bound for one-sided error

- Stronger direct product theorem for threshold functions for one-sided error algorithms that never say $|x| \geq t$ in any instance if it is not true

$$T \leq \alpha k \sqrt{tn} \Rightarrow \sigma \leq 2^{-\gamma k t}$$

$$\left(\text{two-sided symmetric:} \quad T \leq \alpha k \sqrt{tn} \Rightarrow \sigma \leq 2^{-\gamma k}\right)$$

Lower bound for one-sided error

- Stronger direct product theorem for threshold functions for one-sided error algorithms that never say $|x| \geq t$ in any instance if it is not true

$$T \leq \alpha k \sqrt{tn} \Rightarrow \sigma \leq 2^{-\gamma k t}$$

$$\Big(\text{two-sided symmetric:} \quad T \leq \alpha k \sqrt{tn} \Rightarrow \sigma \leq 2^{-\gamma k}\Big)$$

- The same slicing approach (with different slice-size) gives

$$\begin{aligned} T^2 S &\geq N^3 t^2, \quad & t \leq S \leq N/t^2 \\ TS &= N^2, & S > N/t^2 \end{aligned}$$

Lower bound for one-sided error

- Stronger direct product theorem for threshold functions for one-sided error algorithms that never say $|x| \geq t$ in any instance if it is not true

$$T \leq \alpha k \sqrt{tn} \Rightarrow \sigma \leq 2^{-\gamma k t}$$

$$\Big(\text{two-sided symmetric:} \quad T \leq \alpha k \sqrt{tn} \Rightarrow \sigma \leq 2^{-\gamma k}\Big)$$

- The same slicing approach (with different slice-size) gives

$$\begin{aligned} T^2 S &\geq N^3 t^2, \quad & t \leq S \leq N/t^2 \\ TS &= N^2, & S > N/t^2 \end{aligned}$$

- We do not have a matching upper bound, and we conjecture that the lower bound is not tight

Conclusion

- Quantum search speeds up the evaluation of a system of linear inequalities when the space is small

$$T^2 S \leq N^3 t \text{ for } S \leq N/t$$

Conclusion

- Quantum search speeds up the evaluation of a system of linear inequalities when the space is small

$$T^2 S \le N^3 t \text{ for } S \le N/t$$

- If space is big, then quantum computers offer no speedup over classical computers

$$TS \le N^2 \text{ for } S > N/t$$

Conclusion

- Quantum search speeds up the evaluation of a system of linear inequalities when the space is small

$$T^2 S \leq N^3 t \text{ for } S \leq N/t$$

- If space is big, then quantum computers offer no speedup over classical computers

$$TS \leq N^2 \text{ for } S > N/t$$

- A matching lower bound proved using direct product theorems

Conclusion

- Quantum search speeds up the evaluation of a system of linear inequalities when the space is small

$$T^2 S \leq N^3 t \text{ for } S \leq N/t$$

- If space is big, then quantum computers offer no speedup over classical computers

$$TS \leq N^2 \text{ for } S > N/t$$

- A matching lower bound proved using direct product theorems
- For one-sided error algorithms the lower bound is stronger