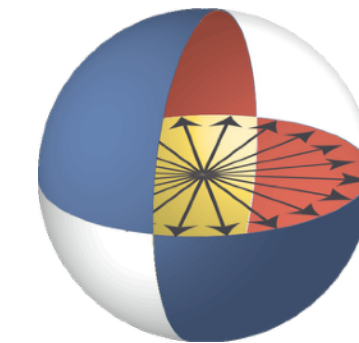


Quantum divide and conquer

Andrew Childs
University of Maryland



UMIACS
University of Maryland
Institute for Advanced
Computer Studies



JOINT CENTER FOR
QUANTUM INFORMATION
AND COMPUTER SCIENCE



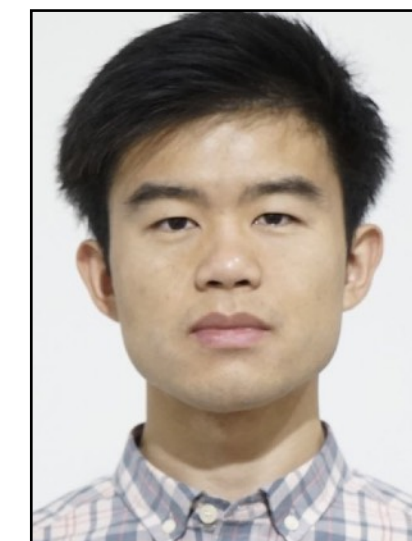
Robin Kothari
Microsoft



Matt Kovacs-Deak
University of Maryland



Aarthi Sundaram
Microsoft



Daochen Wang
University of Maryland

arXiv:2210.06419

The power of quantum computers

Using carefully designed interference between different computational paths, quantum computers can solve some problems dramatically faster than classical computers.

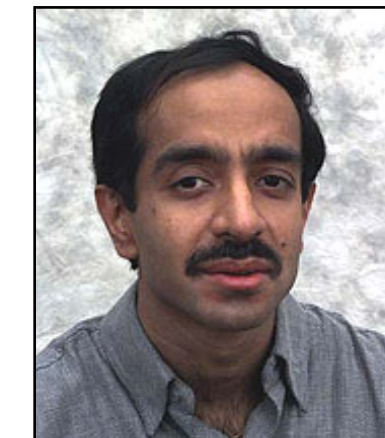
Some problems admit exponential quantum speedup.

Period finding, factoring, discrete log, quantum simulation, quantum linear algebra, Jones polynomial approximation, counting points on curves, graph connectivity with cut queries, ...



Other problems admit polynomial quantum speedup.

Unstructured search, formula evaluation, collision finding, network flows, finding subgraphs, minor-closed graph properties, group commutativity, convex optimization, string problems, ...



What problems can be solved significantly faster by quantum computers than classical ones?

Tools for quantum algorithms

- Fourier sampling
- Grover search/amplitude amplification
- Quantum walk
- Adiabatic optimization/QAOA
- Span programs
- Learning graphs
- Quantum linear systems
- Quantum signal processing
- ...



Divide and conquer

- Divide a problem into subproblems
- Recursively solve the subproblems
- Combine the solutions (with some additional computation) to solve the full problem

Example: Mergesort

- Divide list in half
- Sort left and right halves (recursively)
- Merge the halves in linear time

1	8	6	7	5	3	0	9
---	---	---	---	---	---	---	---

1	8	6	7	5	3	0	9
---	---	---	---	---	---	---	---

1	6	7	8	0	3	5	9
---	---	---	---	---	---	---	---

0	1	3	5	6	7	8	9
---	---	---	---	---	---	---	---

Recurrence for cost $C(n)$: $C(n) = 2C(n/2) + O(n) \Rightarrow C(n) = O(n \log n)$

From classical to quantum divide and conquer

Simple example: $\text{OR}(x) = x_1 \vee x_2 \vee \cdots \vee x_n$

Divide and conquer: $\text{OR}(x) = \text{OR}(\text{OR}(x_{\text{left}}), \text{OR}(x_{\text{right}}))$

Classical: $C(n) \leq 2 C(n/2) \quad \Rightarrow \quad C(n) \leq n$

Quantum: $C_Q(n) \stackrel{?}{\leq} \sqrt{2} C_Q(n/2) \quad \Rightarrow \quad C_Q(n) \stackrel{?}{\leq} \sqrt{n}$

But this is not justified!

- The quantum query complexity of OR is only $O(\sqrt{n})$
- Grover's algorithm has bounded error
- $\sqrt{2}$ is not an integer

From classical to quantum divide and conquer

Typical classical divide-and-conquer recurrence:

Divide an instance of size n into a instances of size n/b

$$C(n) \leq a C(n/b) + C^{\text{aux}}(n)$$

cost of solving auxiliary problem



Corresponding quantum divide-and-conquer recurrence:

$$C_Q(n) \leq \sqrt{a} C_Q(n/b) + C_Q^{\text{aux}}(n)$$

cost of solving auxiliary problem is $O(C_Q^{\text{aux}}(n))$



Query complexity

The model of *query complexity* provides a useful way of exploring the relative power of classical and quantum computers.

Main idea: Input string is described by a black box that can be queried to learn any given character. How many queries are needed to learn some property of the input?

- *Deterministic* query complexity, D : how many queries are needed for a deterministic classical algorithm to produce the correct answer?
- *Randomized* query complexity, R : how many queries are needed for a randomized classical algorithm to produce the correct answer with probability at least $2/3$?
- *Quantum* query complexity, Q : how many queries are needed for a quantum algorithm to produce the correct answer with probability at least $2/3$?

Example: Input describes an n -bit string. Compute the logical OR of the bits.

$$D(\text{OR}) = \Theta(n) \quad R(\text{OR}) = \Theta(n) \quad Q(\text{OR}) = \Theta(n^{1/2})$$

Adversary method

The *quantum adversary method* is a lower bound technique that turns out to be tight.

$$f: S \rightarrow T, \quad S \subseteq \Sigma^n$$

$$\text{Adv}(f) = \max_{\Gamma} \frac{\|\Gamma\|}{\max_{i \in \{1, \dots, n\}} \|\Gamma_i\|} \quad \text{where} \quad \Gamma_{xy} = 0 \text{ if } f(x) = f(y)$$
$$(\Gamma_i)_{xy} = \begin{cases} \Gamma_{xy} & \text{if } x_i \neq y_i \\ 0 & \text{if } x_i = y_i \end{cases}$$

Theorem [Høyer, Lee, Špalek 07; Lee, Mittal, Reichardt, Špalek 10]. $Q(f) = \Theta(\text{Adv}(f))$

Adversary composition

OR composition: Let $g(x, y) = f_1(x) \vee f_2(y)$

$$\text{Then } \text{Adv}(g)^2 \leq \text{Adv}(f_1)^2 + \text{Adv}(f_2)^2$$

Generalizes to arbitrary AND-OR formulas

SWITCH-CASE composition: Let $h(x) = g_{f(x)}(x)$

$$\text{Then } \text{Adv}(h) \leq O(\text{Adv}(f)) + \max_s \text{Adv}(g_s)$$

Quantum divide-and-conquer framework

Suppose f is computed as an AND-OR formula of f_1, \dots, f_a and f^{aux}

$$\text{Then } \text{Adv}(f)^2 \leq \sum_{i=1}^a \text{Adv}(f_i)^2 + O(Q(f^{\text{aux}}))^2$$

Suppose f is computed by first computing $s = f^{\text{aux}}(x)$ and then computing some function g_s

$$\text{Then } \text{Adv}(f) \leq O(Q(f^{\text{aux}})) + \max_s \text{Adv}(g_s)$$

These strategies combine the adversary method (for the term where the constant matters) with the world of quantum algorithms (which are easier to design)

Other strategies are possible using other quantum adversary primitives

Applications

Simpler analysis with slightly improved upper bounds:

- Regular languages: Deciding whether a string over $\{0, 1, 2\}$ contains 20^*2 . This is a key algorithmic result in the query complexity trichotomy for regular languages [Aaronson, Grier, Schaeffer 19].
- String minimality problems: Decision versions of Minimal Length- l Substring, Minimal String Rotation, and Minimal Suffix. Simpler, tighter analysis than [Akmal, Jin 22].

The first nontrivial quantum algorithms for subsequence problems:

- Does x have an increasing subsequence of length k ? $\tilde{O}(\sqrt{n})$
- Do x and y have a common subsequence of length k ? $\tilde{O}(n^{2/3})$

Regular languages


Let $\Sigma = \{0, 1, 2\}$

Problem: Given $x \in \Sigma^n$, is $x \in \Sigma^* 20^* 2 \Sigma^*$?

String x contains 20^*2 iff x_{left} contains 20^*2 or

x_{right} contains 20^*2 or

x_{left} ends in 20^* and x_{right} starts with 0^*2



Can be checked by Grover search in time $O(\sqrt{n})$:

- Search for the last 2 in x_{left} and the first 2 in x_{right}
- Search for a non-0 in the string between them

Recurrence for adversary quantity: $a(n)^2 \leq 2a(n/2)^2 + O(\sqrt{n})^2 \Rightarrow a(n) = O(\sqrt{n \log n})$

This improves the log factors of [Aaronson, Grier, Schaeffer 19] with a simpler proof.

Increasing subsequences

A *subsequence* of a string is obtained by taking a (not necessarily consecutive) subset of the characters, without changing their order.

Longest Increasing Subsequence (LIS): Given $x \in \Sigma^n$ over an ordered alphabet Σ , find a longest increasing subsequence of x

Example: The LIS of 8, 6, 7, 5, 3, 0, 9 is 6, 7, 9

Unfortunately, $Q(\text{LIS}) = \Theta(n)$

k -IS: For fixed k , does $x \in \Sigma^n$ have an increasing subsequence of length k ?

$R(k\text{-IS}) = \Theta(n)$ for $k \geq 2$ (1-IS is trivial)

$Q(2\text{-IS}) = \Theta(\sqrt{n})$ (equivalent to unstructured search)

$Q(k\text{-IS}) = O(n^{k/(k+1)})$ by a generalization of Ambainis's k -distinctness algorithm

Can we do better?

k -Increasing subsequence

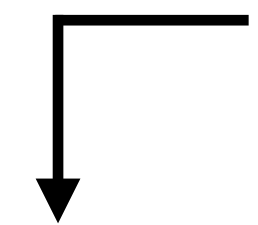
Theorem. For any fixed k , $Q(k\text{-IS}) = O(\sqrt{n} \log^{3(k-1)/2} n)$.

String x contains a k -IS iff

x_{left} contains a k -IS or

x_{right} contains a k -IS or

x contains a k -IS with i elements in x_{left} and $k - i$ elements in x_{right}



Can be checked by computing smallest ending value of an i -IS on left
and largest starting value of a $(k - i)$ -IS on right

These can be computed with $O(\log n)$ computations of j -IS (for $j < k$) and Grover search

Recurrence for adversary quantity: $a_k(n)^2 \leq 2a_k(n/2)^2 + O(a_{k-1}(n)^2 \log^2 n)$

Result follows by induction on k .

Common subsequences

Longest Common Subsequence (LCS): Given $x, y \in \Sigma^n$, find a common subsequence of x and y that is as long as possible

Example: The LCS of QUANTUM** and ALGORITHM is ATM

$$Q(\text{LCS}) = \Omega(n)$$

k -CS: For fixed k , do $x, y \in \Sigma^n$ have a common subsequence of length k ?

$$R(k\text{-CS}) = \Omega(n) \text{ for } k \geq 1$$

$$Q(1\text{-CS}) = \Theta(n^{2/3}) \text{ since this is bipartite element distinctness [Aaronson, Shi 04; Ambainis 03]}$$

$$Q(k\text{-CS}) = O(n^{2k/(2k+1)}), \text{ also using [Ambainis 03]}$$

Can we do better?

k -Common subsequence

Theorem. For any fixed k , $Q(k\text{-CS}) = O(n^{2/3} \log^{3(k-1)/2} n)$.

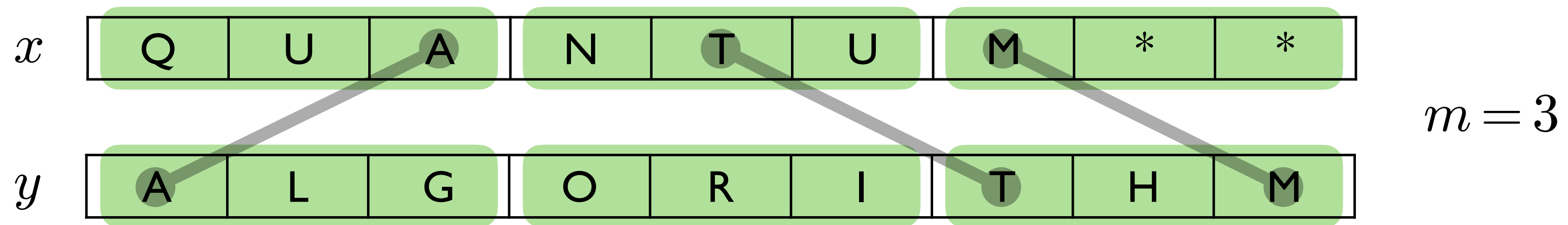
This follows from a quantum divide-and-conquer algorithm with seven parts, but not fewer!

Strategy:

- Divide both x and y into m parts of length n/m
- Determine which parts have collisions
- Consider two cases:
 - “Simple”: a k -CS appears between one part of x and one part of y
Limited number of cases to check
 - “Composite”: everything else
Can be checked by finding j -CSs for $j < k$

Subproblems and signatures

Divide the strings into m parts, giving m^2 subproblems



The *signature* indicates which subproblems contain at least one collision

In the above example, the signature includes $(1, 1)$, $(2, 3)$, $(3, 3)$

Can compute the signature with $O(n^{2/3})$ queries
(m^2 instances of bipartite element distinctness)

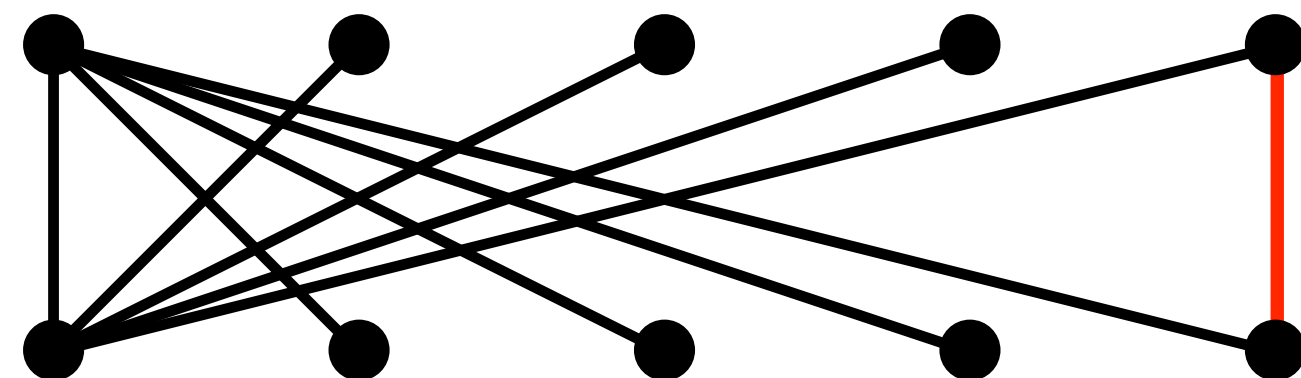
Simple and composite solutions

Definition: A k -CS is *simple* if it is a k -CS of some subproblem; otherwise it is *composite*.

To find composite solutions, it suffices to solve $O(\log n)$ instances of j -CS subproblems for $j < k$, at a cost of $O(a_{k-1}(n) \log n)$.

When considering simple solutions, it suffices to consider *critical* subproblems, those that cannot be combined with a known collision from another part of the signature.

There can be at most $2m - 1$ critical subproblems to consider, at a cost of $\sqrt{2m - 1} a_k(n/m)$.



$$m = 5$$

k -CS complexity analysis

Claim: $a_k(n) = O(n^{2/3} \log^{k-1}(n))$

Let $a_k(n)$ = adversary quantity for k -CS with input length n

Computing the signature: $O(n^{2/3})$

Composite subproblems: $O(a_{k-1}(n) \log n)$

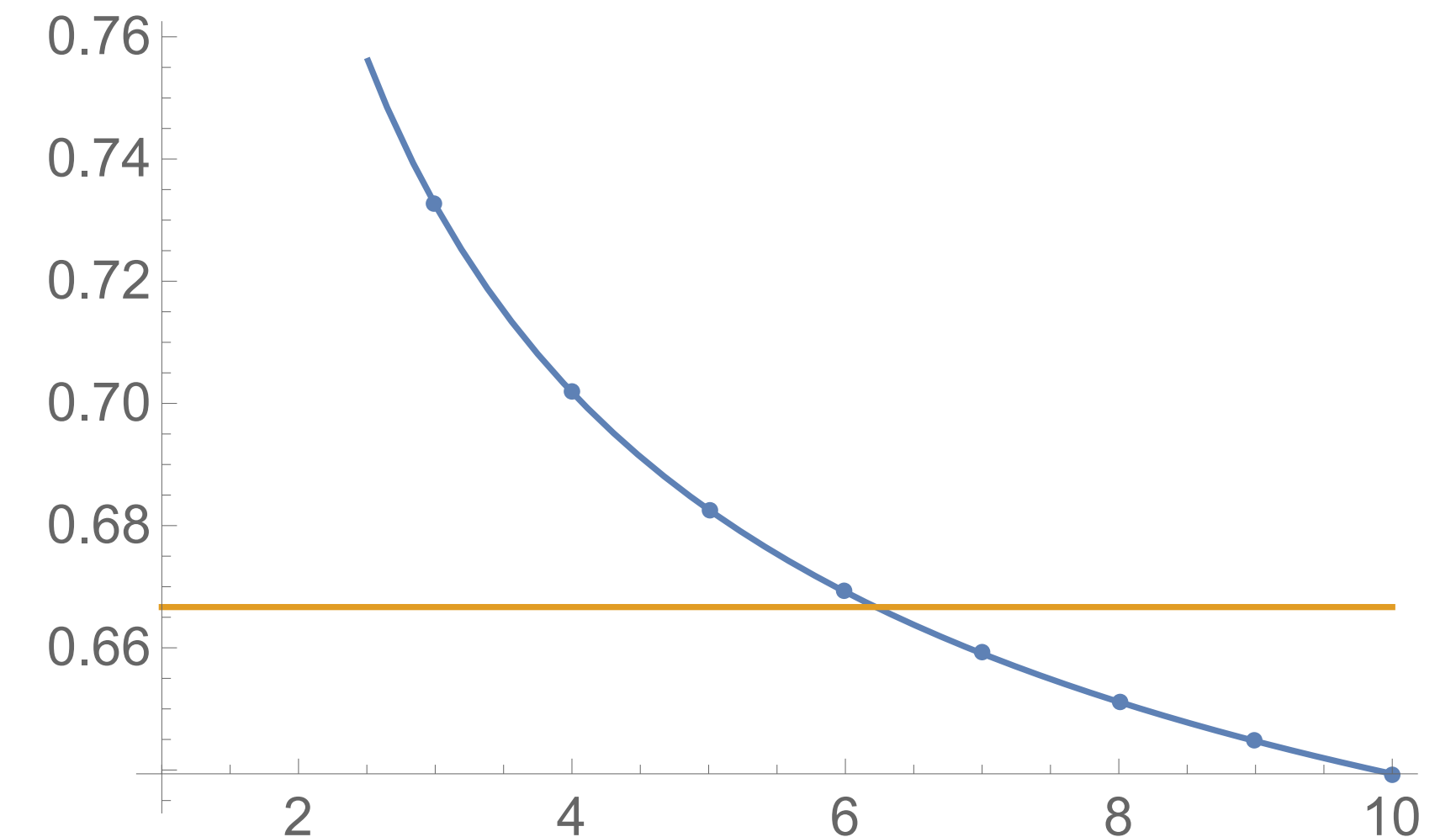
Simple subproblems: $\sqrt{2m-1} a_k(n/m)$

So $a_k(n) \leq O(n^{2/3}) + O(a_{k-1}(n) \log n) + \sqrt{2m-1} a_k(n/m)$

$\leq \sqrt{2m-1} a_k(n/m) + O(n^{2/3} \log^{k-1} n)$ by induction on k

Master Theorem: $a_k(n) = O(n^{2/3} \log^{k-1}(n))$ provided $\log_m(\sqrt{2m-1}) < 2/3$

which is satisfied with $m = 7$



Summary

We have introduced a divide-and-conquer framework for developing quantum algorithms using classical reasoning about division into subproblems, with speedup from quantum combining operations and the use of quantum subroutines.

Applications:

- Simpler analysis for regular language and minimal substring problems with tighter bounds
- $\tilde{O}(\sqrt{n})$ algorithm for k -IS
- $\tilde{O}(n^{2/3})$ algorithm for k -CS

Open problems

- Can we apply quantum divide and conquer to search problems? For example, is there a quantum divide-and-conquer algorithm for minimum finding?
- Can we find applications of quantum divide and conquer using combining functions other than AND-OR formulas and SWITCH-CASE?
- Can we obtain super-quadratic speedups using quantum divide and conquer?