

Response to Ramesh & Vinay, (2003)

String Matching in $\tilde{O}(\sqrt{n} + \sqrt{m})$ Quantum Time

Matthew Evans, Ariz Siddiqui, Nathan Puskuri

April 27, 2025

Outline

Introduction

Preliminaries

- Grover's Algorithm

- Tight bounds on quantum searching

- Deterministic Sampling

- Minimum Finding Oracle

Quantum String Matching

- The Algorithm

- Concerning Periodicity

- Constant Two-Sided Failure Probability

- Achieving $\tilde{O}(\sqrt{n} + \sqrt{m})$

Conclusion

String Matching in $\tilde{O}(\sqrt{n} + \sqrt{m})$ Quantum Time

H. Ramesh & V. Vinay (IISc Bangalore, 2000)

Problem Statement

Given a text t of length n and a pattern p of length m , decide whether p occurs in t .

- ▶ **Classical bound:** $\Theta(n + m)$ via KMP, Boyer-Moore, etc.
- ▶ **Quantum goal:** Exploit amplitude amplification to beat linear time.
- ▶ **Main result:** A quantum algorithm running in

$$\tilde{O}(\sqrt{n} + \sqrt{m})$$

with constant two-sided error probability.

Conceptual Dependencies

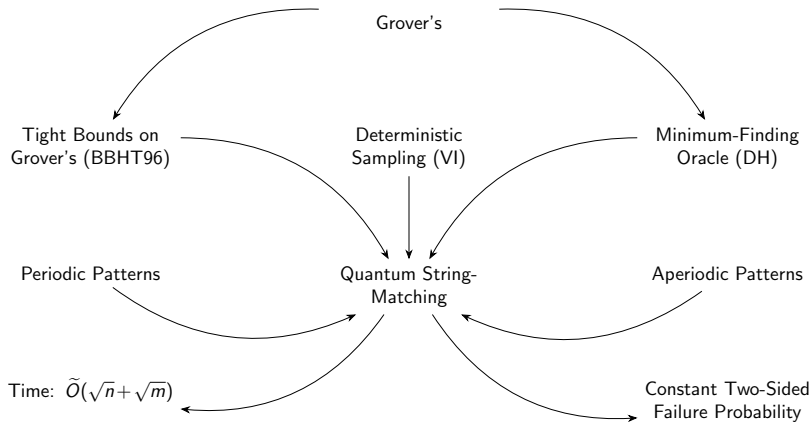
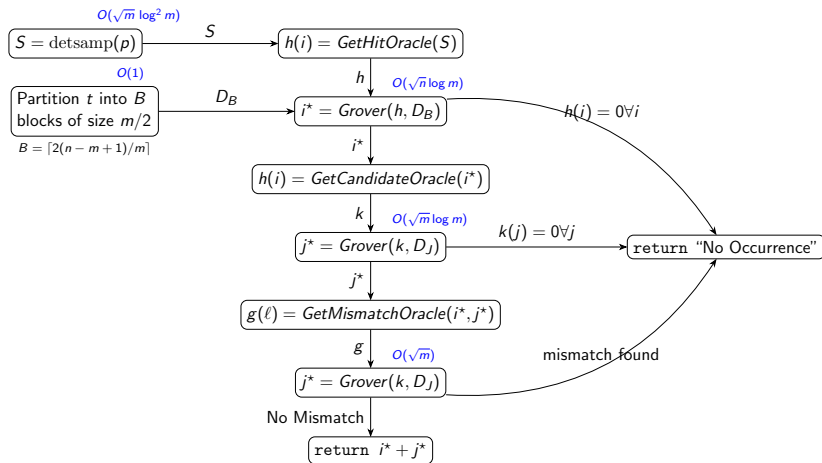


Figure: Conceptual dependencies in the $\tilde{O}(\sqrt{n} + \sqrt{m})$ quantum string-matching algorithm.

The Algorithm



Grover's Algorithm

- ▶ Problem: Given a database of N elements and oracle $f(x) = 1$ for marked items, find an x with $f(x) = 1$.
- ▶ Steps:
 1. Initialize n qubits into uniform superposition over $N = 2^n$ states.
 2. Apply the oracle to flip the phase of marked states.
 3. Perform the diffusion (inversion-about-the-mean) operator.
 4. Repeat oracle + diffusion $\lfloor \frac{\pi}{4} \sqrt{N/t} \rfloor$ times ($t = \#$ marked).
 5. Measure to obtain a marked element with high probability.
- ▶ Time Complexity: $\mathcal{O}(\sqrt{N/t})$.
- ▶ Why it works: Oracle phase-flips mark targets; diffusion amplifies their amplitudes.
- ▶ Significance: Core building block for quantum search algorithms.

BBHT96: Tight bounds on quantum searching

What it is

- ▶ Even when the number of target $t \geq 1$ items in a database is unknown, you can still find a marked item in $\tilde{O}(\sqrt{N/t})$ oracle calls.
- ▶ BBHT96 describes a procedure which, given an oracle flagging at least t marked elements among n candidates, returns one solution in $O(\sqrt{n/t})$ time with constant success probability.

BBHT96: Tight bounds on quantum searching

Why it matters

- ▶ It underpins the claimed time bound in our string-matching algorithm.
- ▶ When checking a text block (size $m/2$) for any matching alignment, t is unknown.
- ▶ BBHT96's procedure lets us invoke Grover's search reliably in this setting.
- ▶ Every oracle in the string-matching pipeline (e.g. f, h) relies on BBHT96's bound to guarantee the $\tilde{O}(\sqrt{n} + \sqrt{m})$ runtime.

Deterministic Sampling

What it is

- ▶ Given an aperiodic pattern p of length m , and a text block of length $m/2$, there are $m/2$ possible alignments.
- ▶ DS picks $O(\log m)$ indices in the pattern so that at most one alignment can match all sampled positions.

Steps

1. Form $m/2$ “copies” of the pattern, each shifted by one position.
2. Find a column where at least two copies differ.
3. Select one of the symbols at that column as the sample and discard copies that don't match.
4. After $O(\log m)$ rounds, one copy remains; its chosen columns form the sample S .

Deterministic Sampling

Why it works

- ▶ Instead of a full pattern check per alignment ($O(m)$ classically, $\tilde{O}(\sqrt{m})$ quantum), only $O(\log m)$ sampled positions are tested.
- ▶ Only the single surviving alignment requires the expensive full check.

Why it matters

- ▶ Enables the overall quantum string matching to run in $\tilde{O}(\sqrt{n} + \sqrt{m})$ by reducing costly \sqrt{m} checks to one per text block.

DH96: Minimum Finding Oracle

What it is

- ▶ Given a database of size n and a comparison oracle that, for any two indices i, j , indicates which element is smaller.
- ▶ Finds the index of the minimum element in $O(\sqrt{n})$ time.
- ▶ Serves as the backbone for all “pick the smallest (or leftmost) index satisfying a condition” steps.

Steps

1. Pick a random starting position k .
2. Use Grover's search to find any index i with $\text{database}[i] < \text{database}[k]$.
3. If such an i is found, set $k \leftarrow i$ and repeat.
4. Otherwise, k is the index of the minimum element.

DH96: Minimum Finding Oracle

Why it matters

- ▶ Building the deterministic sampling set: Repeatedly eliminate half of the $m/2$ pattern copies by finding the leftmost and rightmost survivor via DH96 in $O(\sqrt{m} \log m)$ time.
- ▶ After locating the matching text-block with the $h(i)$ oracle, invoke DH96 over block indices to pinpoint the earliest occurrence, preserving the overall $\tilde{O}(\sqrt{n} + \sqrt{m})$ bound.

The Algorithm (Part 1)

1. Deterministic-Sampling Preprocessing.

- ▶ Run Vishkin's deterministic-sampling on p of length m to obtain an $O(\log m)$ -sized sample set S .
- ▶ Cost: $\tilde{O}(\sqrt{m} \log^2 m)$.

2. Partition the text.

- ▶ Divide the text t into

$$B = \left\lceil \frac{2(n - m + 1)}{m} \right\rceil$$

blocks, each of size $\approx m/2$.

3. Quantum search for a “hit” block.

- ▶ Define oracle $h(i)$: tests if block i has at least one alignment matching on all positions in S in $\tilde{O}(\sqrt{m} \log m)$ time.
- ▶ Use Grover search over $i = 1, \dots, B$ with oracle h ; time $\tilde{O}(\sqrt{n} \log m)$. If none found, conclude “no occurrence.”

The Algorithm (Part 2)

4. Locate surviving alignment in block.

- ▶ Define oracle $k(i^*, j)$: checks alignment at shift j in block i^* on sample S in $O(\log m)$ time.
- ▶ Use Grover search over $j = 0, \dots, \lfloor m/2 \rfloor$ with oracle k ; time $\tilde{O}(\sqrt{m} \log m)$. If none survives, conclude “no occurrence.”

5. Full quantum verification.

- ▶ Run Grover search over $\ell = 1, \dots, m$ with oracle “ $t[i^* + j^* + \ell] \neq p[\ell]$ ” to find any mismatch; time $\tilde{O}(\sqrt{m})$.
- ▶ If no mismatch is found, report occurrence at $i^* + j^*$; otherwise, conclude “no occurrence.”

Concerning Periodicity

Constant Two-Sided Failure Probability

Achieving $\tilde{O}(\sqrt{n} + \sqrt{m})$

Conclusion