Quantum search of partially ordered sets

Ashley Montanaro[1]

[1]Department of Computer Science
University of Bristol
Bristol, UK

13th March 2008

Introduction

One of the most significant quantum algorithms developed so far is Grover's algorithm for unstructured search [Grover '97].

? ? ? ? ? ? ?

Introduction

One of the most significant quantum algorithms developed so far is Grover's algorithm for unstructured search [Grover '97].

Introduction

One of the most significant quantum algorithms developed so far is Grover's algorithm for unstructured search [Grover '97].

0	**0**	**0**	**0**	**0**	**1**	**0**

- Given an arbitrary non-zero function $f : [n] \mapsto \{0, 1\}$, Grover's quantum algorithm finds an x such that $f(x) = 1$, with constant probability, using only $O(\sqrt{n})$ queries to f.

- Important extension to amplitude amplification: given a probabilistic algorithm A that succeeds with probability p, and the ability recognise a correct solution, can output a correct solution with probability $O(1)$ using only $O(1/\sqrt{p})$ uses of A [Brassard et al '00].

More complicated search

Grover's algorithm is already a useful primitive to speed up more complicated classical algorithms. For example, we can:

- Find the minimum element in a set of n integers in $O(\sqrt{n})$ time [Dürr, Høyer '96],
- Find a collision in a $2 \rightarrow 1$ function $f : [2n] \mapsto [n]$ in $O(n^{1/3})$ time [Brassard et al '97],
- Find a spanning tree in an n-vertex graph in $O(n^{3/2})$ time (adjacency matrix model) [Dürr et al '04],
- ...

All these applications work by finding a part of the problem in question that's essentially unstructured, and running Grover search on this.

More structured search?

Could we speed up a search problem that has some kind of recursive structure?

(NB: already known that quantum search of an ordered n-element list requires $\Omega(\log n)$ time [Høyer et al '01])

More structured search?

Consider the problem of (classical) search of an abstract "database" D_n, parametrised by a problem size n, with the following characteristics:

- If $n \leqslant n_0$ for some constant n_0: can find the element in time $T(n) \leqslant t_0$, for some constant t_0.

More structured search?

Consider the problem of (classical) search of an abstract "database" D_n, parametrised by a problem size n, with the following characteristics:

- If $n \leqslant n_0$ for some constant n_0: can find the element in time $T(n) \leqslant t_0$, for some constant t_0.
- If $n > n_0$: the database can be divided into k sub-databases of size at most $\lceil n/k \rceil$, for some constant $k > 1$.

More structured search?

Consider the problem of (classical) search of an abstract "database" D_n, parametrised by a problem size n, with the following characteristics:

- If $n \leqslant n_0$ for some constant n_0: can find the element in time $T(n) \leqslant t_0$, for some constant t_0.
- If $n > n_0$: the database can be divided into k sub-databases of size at most $\lceil n/k \rceil$, for some constant $k > 1$.
- If the element is in the original database, then it is in exactly one of these sub-databases.

More structured search?

Consider the problem of (classical) search of an abstract "database" D_n, parametrised by a problem size n, with the following characteristics:

- If $n \leqslant n_0$ for some constant n_0: can find the element in time $T(n) \leqslant t_0$, for some constant t_0.
- If $n > n_0$: the database can be divided into k sub-databases of size at most $\lceil n/k \rceil$, for some constant $k > 1$.
- If the element is in the original database, then it is in exactly one of these sub-databases.
- Each division into sub-databases uses time $f(n)$, where $f(n) = O(n^{1-\epsilon})$ for some $\epsilon > 0$.

What is the time $T(n)$ to find an element in D_n?

Recursive quantum search?

This is easy to solve by the recurrence

$$T(n) = k\,T(n/k) + O(n^{1-\epsilon})$$

Recursive quantum search?

This is easy to solve by the recurrence

$$T(n) = k\,T(n/k) + O(n^{1-\epsilon}) = O(n)$$

We would like to find a quantum version of this recurrence. Can we get a speed-up by searching the sub-databases in quantum parallel?

Recursive quantum search theorem

Consider the problem of searching an abstract "database" D_n, parametrised by a problem size n, with the following characteristics:

- If $n \leqslant n_0$ for some constant n_0: can find the element in time $T(n) \leqslant t_0$, for some constant t_0.

Recursive quantum search theorem

Consider the problem of searching an abstract "database" D_n, parametrised by a problem size n, with the following characteristics:

- If $n \leqslant n_0$ for some constant n_0: can find the element in time $T(n) \leqslant t_0$, for some constant t_0.
- If $n > n_0$: the database can be divided into k sub-databases of size at most $\lceil n/k \rceil$, for some constant $k > 1$.

Recursive quantum search theorem

Consider the problem of searching an abstract "database" D_n, parametrised by a problem size n, with the following characteristics:

- If $n \leqslant n_0$ for some constant n_0: can find the element in time $T(n) \leqslant t_0$, for some constant t_0.
- If $n > n_0$: the database can be divided into k sub-databases of size at most $\lceil n/k \rceil$, for some constant $k > 1$.
- If the element is in the original database, then it is in exactly one of these sub-databases.

Recursive quantum search theorem

Consider the problem of searching an abstract "database" D_n, parametrised by a problem size n, with the following characteristics:

- If $n \leqslant n_0$ for some constant n_0: can find the element in time $T(n) \leqslant t_0$, for some constant t_0.
- If $n > n_0$: the database can be divided into k sub-databases of size at most $\lceil n/k \rceil$, for some constant $k > 1$.
- If the element is in the original database, then it is in exactly one of these sub-databases.
- Each division into sub-databases uses time $f(n)$, where $f(n) = O(n^{1/2-\epsilon})$ for some $\epsilon > 0$.

Then there is a quantum algorithm that finds an element in D_n with constant probability in time $T(n) = O(\sqrt{n})$.

Finding the intersection of two sorted lists

Problem

Given monotone functions $f : [n] \mapsto \mathbb{Z}$, $g : [n] \mapsto \mathbb{Z}$, output a y such that $f(x) = g(x') = y$ for some x, x', if one exists.

1	**2**	**4**	**4**	**8**	**9**	**10**

3	**4**	**5**	**6**	**6**	**7**	**8**

Finding the intersection of two sorted lists

Problem

Given monotone functions $f : [n] \mapsto \mathbb{Z}$, $g : [n] \mapsto \mathbb{Z}$, output a y such that $f(x) = g(x') = y$ for some x, x', if one exists.

1	**2**	**4**	**4**	**8**	**9**	**10**

3	**4**	**5**	**6**	**6**	**7**	**8**

- Obvious classical lower bound is $2n$ queries (have to read all the input in)
- "Obvious" quantum algorithm uses $O(\sqrt{n}\log n)$ queries
- [Buhrman et al '05] gave an ingenious $O(\sqrt{n}c^{\log^* n})$ algorithm
- Lower bound is $\Omega(\sqrt{n})$ queries

We give an algorithm matching this lower bound.

A recursive classical algorithm

Idea: reduce the problem to searching in a 2d array sorted along rows and columns.

- Consider a notional $n \times n$ array T where $T(x, y) = f(x) - g(n+1-y)$.
- Then finding a zero in T finds a match in the two lists.

	8	**7**	**6**	**6**	**5**	**4**	**3**
1	-7	-6	-5	-5	-4	-3	-2
2	-6	-5	-4	-4	-3	-2	-1
4	-4	-3	-2	-2	-1	0	1
4	-4	-3	-2	-2	-1	0	1
8	0	1	2	2	3	4	5
9	1	2	3	3	4	5	6
10	2	3	4	4	5	6	7

A recursive classical algorithm

Idea: write down an asymptotically optimal recursive classical algorithm for this task, then use the recursive quantum search theorem.

A recursive classical algorithm

Idea: write down an asymptotically optimal recursive classical algorithm for this task, then use the recursive quantum search theorem.

Given an $n \times n$ array A:

1. Perform binary search on the middle row/column of A.
2. After binary search, can eliminate two subarrays of A containing about half the elements in A.
3. We're left with two subarrays which might contain the target element: recurse on these subarrays.

A recursive classical algorithm

Idea: write down an asymptotically optimal recursive classical algorithm for this task, then use the recursive quantum search theorem.

Given an $n \times n$ array A:

1. Perform binary search on the middle row/column of A.
2. After binary search, can eliminate two subarrays of A containing about half the elements in A.
3. We're left with two subarrays which might contain the target element: recurse on these subarrays.

Can show $T(n) \leqslant O(\log n) + 2T(n/2) = O(n)$.

(a different optimal classical algorithm was already known [Linial, Saks '85], but seems harder to "make quantum")

Example

1	3	5	10	13
2	4	7	11	14
6	8	9	15	21
12	16	17	20	24
18	19	22	23	25

(green: integer to search for, yellow: not searched yet, blue: currently being searched, red: discarded)

Example

1	3	5	10	13
2	4	7	11	14
6	8	9	15	21
12	16	17	20	24
18	19	22	23	25

(green: integer to search for, yellow: not searched yet, blue: currently being searched, red: discarded)

Example

1	3	5	10	13
2	4	7	11	14
6	8	9	15	21
12	16	17	20	24
18	19	22	23	25

(green: integer to search for, yellow: not searched yet, blue: currently being searched, red: discarded)

Applying the recursive quantum search theorem

Let's check that we can apply the theorem.

Applying the recursive quantum search theorem

Let's check that we can apply the theorem.

- If $n \leqslant n_0$ for some constant n_0: can find the element in time $T(n) \leqslant t_0$, for some constant t_0. ✓

Applying the recursive quantum search theorem

Let's check that we can apply the theorem.

- If $n \leqslant n_0$ for some constant n_0: can find the element in time $T(n) \leqslant t_0$, for some constant t_0. ✓

- If $n > n_0$: the database can be divided into k sub-databases of size at most $\lceil n/k \rceil$, for some constant $k > 1$. ✓

Applying the recursive quantum search theorem

Let's check that we can apply the theorem.

- If $n \leqslant n_0$ for some constant n_0: can find the element in time $T(n) \leqslant t_0$, for some constant t_0. ✓
- If $n > n_0$: the database can be divided into k sub-databases of size at most $\lceil n/k \rceil$, for some constant $k > 1$. ✓
- Each division into sub-databases uses time $f(n)$, where $f(n) = O(n^{1/2-\epsilon})$ for some $\epsilon > 0$. ✓

Applying the recursive quantum search theorem

Let's check that we can apply the theorem.

- If $n \leqslant n_0$ for some constant n_0: can find the element in time $T(n) \leqslant t_0$, for some constant t_0. ✓
- If $n > n_0$: the database can be divided into k sub-databases of size at most $\lceil n/k \rceil$, for some constant $k > 1$. ✓
- Each division into sub-databases uses time $f(n)$, where $f(n) = O(n^{1/2-\epsilon})$ for some $\epsilon > 0$. ✓
- If the element is in the original database, then it is in exactly one of these sub-databases. ✘

We might have more than one zero in the array.

Finishing off the algorithm

Problem: The recursive quantum search algorithm can only cope with at most one marked element.

Solution:

- Note that the zeroes only occur in rectangular blocks, with at most one block per row and column
- If there's only one such "zero block", can modify the search algorithm to pretend that the block contains one element
- If not, to reduce to the single-block case, repeatedly throw away random rows and columns over several rounds
- Can show that with constant probability, one round will have only one zero block remaining
- Can also show that the asymptotic query complexity isn't hurt by doing this

Proof idea of the recursive quantum search theorem

- Idea: perform the recursive search of the k sub-databases in quantum parallel.
- Want to end up with a recurrence like $T(n) \leqslant O(n^{1/2-\epsilon}) + \sqrt{k}\,T(n/k) = O(\sqrt{n})$.
- Immediate from amplitude amplification?

Proof idea of the recursive quantum search theorem

- Idea: perform the recursive search of the k sub-databases in quantum parallel.
- Want to end up with a recurrence like $T(n) \leqslant O(n^{1/2-\epsilon}) + \sqrt{k}\,T(n/k) = O(\sqrt{n})$.
- Immediate from amplitude amplification?

Not so fast!

- What happens if we perform l levels of recursion then use amplitude amplication on the resulting k^l sub-databases?
- Seems to give $T(n) \leqslant k^l O(n^{1/2-\epsilon}) + O(k^{l/2})\,T(n/k^l) = O(n^{1/2+c})$
  - for some constant c that turns up because of the hidden constant in the big-O notation

Proof idea of the recursive quantum search theorem

- Idea: perform the recursive search of the k sub-databases in quantum parallel.
- Want to end up with a recurrence like $T(n) \leqslant O(n^{1/2-\epsilon}) + \sqrt{k}\, T(n/k) = O(\sqrt{n})$.
- Immediate from amplitude amplification?

Not so fast!

- What happens if we perform l levels of recursion then use amplitude amplication on the resulting k^l sub-databases?
- Seems to give $T(n) \leqslant k^l O(n^{1/2-\epsilon}) + O(k^{l/2})\, T(n/k^l) = O(n^{1/2+c})$
  - for some constant c that turns up because of the hidden constant in the big-O notation

Moral: We have to be very careful about constants in this recursive algorithm!

A general recursive quantum search algorithm

- We extend a powerful result of [Aaronson and Ambainis '05] on quantum search of spatial regions.
- Idea: it's more efficient to do fewer iterations of amplitude amplification

A general recursive quantum search algorithm

- We extend a powerful result of [Aaronson and Ambainis '05] on quantum search of spatial regions.
- Idea: it's more efficient to do fewer iterations of amplitude amplification
- So our recursive algorithm performs "a small amount of" amplitude amplification on an algorithm that consists of:
  - Divide the database into some number of sub-databases
  - Pick one of these sub-databases at random
  - Call yourself on that sub-database
- Then it does "lots" of amplitude amplification at the end.
- Importantly, can find exact bounds on the time required to achieve a certain success probability!

Extensions

- The quantum algorithm for finding an integer in a $n \times n$ array of distinct integers immediately extends to a d-dimensional $n \times n \times \cdots \times n$ array sorted in each dimension (complexity is $O(n^{(d-1)/2})$)
- This is a special case of a more general problem: quantum search of partially ordered sets (posets).
- One can show general upper and lower bounds for this task (summary: quantum computers can achieve at most a quadratic speed-up (approx) for any poset, and barely any speed-up at all for some posets).

Summary and further work

- We have outlined a general approach for achieving a quantum speed-up from recursive classical search algorithms.
- This gives a quantum algorithm that finds the intersection of two sorted n-element lists in $O(\sqrt{n})$ time.

Summary and further work

- We have outlined a general approach for achieving a quantum speed-up from recursive classical search algorithms.
- This gives a quantum algorithm that finds the intersection of two sorted n-element lists in $O(\sqrt{n})$ time.

Future work?

- Extend the recursive quantum search theorem to finding multiple marked elements?
- Further applications? Finding problems where the speed-up is more dramatic?

Summary and further work

- We have outlined a general approach for achieving a quantum speed-up from recursive classical search algorithms.
- This gives a quantum algorithm that finds the intersection of two sorted n-element lists in $O(\sqrt{n})$ time.

Future work?

- Extend the recursive quantum search theorem to finding multiple marked elements?
- Further applications? Finding problems where the speed-up is more dramatic?

Further reading: "Quantum search of partially ordered sets", http://arxiv.org/abs/quant-ph/0702196

Advertisement

Bristol Summer School on Probabilistic Techniques in Computer Science

6-11 July 2008

- Keynote speaker: Bela Bollobás.
- Topics include: randomised algorithms, communication complexity, concentration of measure, data stream algorithms, ...

http://www.cs.bris.ac.uk/probtcs08/