Quantum search with advice

Ashley Montanaro

Department of Computer Science, University of Bristol, UK

arXiv:0908.3066

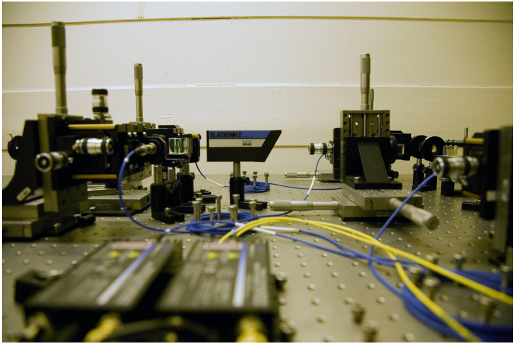
University of BRISTOL

EPSRC Engineering and Physical Sciences Research Council

QICS

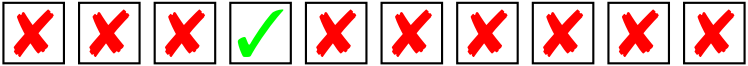
Quantum computing in a nutshell

A quantum computer is a machine which uses quantum physics to achieve a speed-up, or other advantage, over any possible standard ("classical") computer which uses only the laws of classical physics.



Unstructured search

Perhaps the most basic problem in computer science: search of an unstructured list for a single "marked" element.

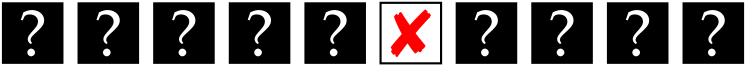


Unstructured search

Perhaps the most basic problem in computer science: search of an unstructured list for a single “marked” element.

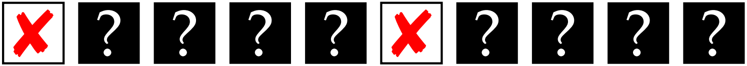
Unstructured search

Perhaps the most basic problem in computer science: search of an unstructured list for a single "marked" element.



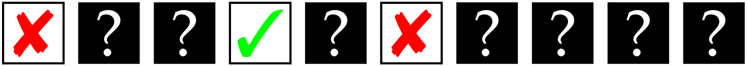
Unstructured search

Perhaps the most basic problem in computer science: search of an unstructured list for a single "marked" element.



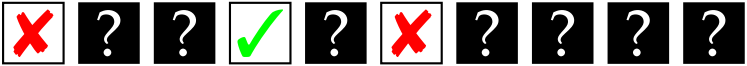
Unstructured search

Perhaps the most basic problem in computer science: search of an unstructured list for a single “marked” element.



Unstructured search

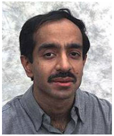
Perhaps the most basic problem in computer science: search of an unstructured list for a single "marked" element.



It's obvious that, in the worst case, any classical algorithm must query the list at least $\Omega(n)$ times (even if we allow a constant probability of error).

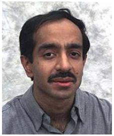
Quantum search

Remarkably, with a quantum computer we can do much better: using Grover's algorithm we can find the marked element using $O(\sqrt{n})$ queries in the worst case.



Quantum search

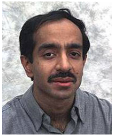
Remarkably, with a quantum computer we can do much better: using Grover's algorithm we can find the marked element using $O(\sqrt{n})$ queries in the worst case.



Some notes on this result:

- In this model, we are only interested in minimising the number of queries used.

Quantum search



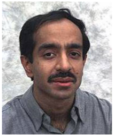
Remarkably, with a quantum computer we can do much better: using Grover's algorithm we can find the marked element using $O(\sqrt{n})$ queries in the worst case.

Some notes on this result:

- In this model, we are only interested in minimising the number of queries used.
- If we are promised that there is exactly one marked item, Grover's algorithm succeeds with certainty.

Quantum search

Remarkably, with a quantum computer we can do much better: using Grover's algorithm we can find the marked element using $O(\sqrt{n})$ queries in the worst case.

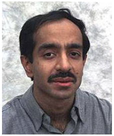


Some notes on this result:

- In this model, we are only interested in minimising the number of queries used.
- If we are promised that there is exactly one marked item, Grover's algorithm succeeds with certainty.
- Grover's algorithm is provably optimal: no quantum algorithm that achieves the same success probability in the worst case can do better by even one query.

Quantum search

Remarkably, with a quantum computer we can do much better: using Grover's algorithm we can find the marked element using $O(\sqrt{n})$ queries in the worst case.



Some notes on this result:

- In this model, we are only interested in minimising the number of queries used.
- If we are promised that there is exactly one marked item, Grover's algorithm succeeds with certainty.
- Grover's algorithm is provably optimal: no quantum algorithm that achieves the same success probability in the worst case can do better by even one query.

So is this all we can say?

Quantum search of structured data

Most databases we want to search in the real world have some kind of structure.

We would like to find a simple model to encapsulate this.

Quantum search of structured data

Most databases we want to search in the real world have some kind of structure.

We would like to find a simple model to encapsulate this.

Some ways we could try to build this in:

- Give the quantum algorithm access to classical heuristics as a black box [Cerf et al '98, Hogg '96, ...].

Quantum search of structured data

Most databases we want to search in the real world have some kind of structure.

We would like to find a simple model to encapsulate this.

Some ways we could try to build this in:

- Give the quantum algorithm access to classical heuristics as a black box [Cerf et al '98, Hogg '96, ...].
- Impose a partial order on the data [AM '09].

Quantum search of structured data

Most databases we want to search in the real world have some kind of structure.

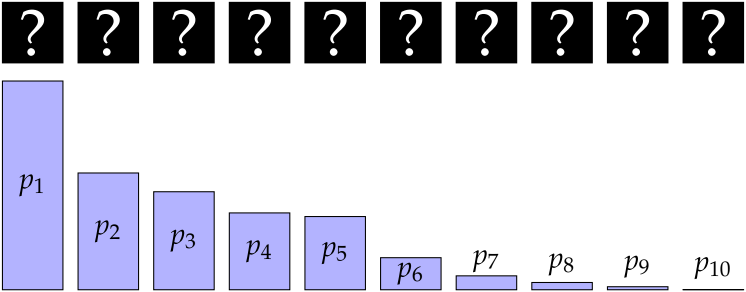
We would like to find a simple model to encapsulate this.

Some ways we could try to build this in:

- Give the quantum algorithm access to classical heuristics as a black box [Cerf et al '98, Hogg '96, ...].
- Impose a partial order on the data [AM '09].
- This talk: say that we are given advice about the database.

Search with advice

As well as the list, we are given access to a probability distribution $\mu = (p_y)$ that hints where the marked element is likely to be.



We have $p_y = \Pr[\text{marked element is at position } y]$.

Formal problem definition

Problem: SEARCH WITH ADVICE

Input: A function $f : \{1, \ldots, n\} \to \{0, 1\}$ that takes the value 1 on precisely one input x, and an "advice" probability distribution $\mu = (p_y)$, $y \in \{1, \ldots, n\}$, where p_y is the probability that $f(y) = 1$.

Output: The marked element x.

Formal problem definition

Problem: Search with Advice

Input: A function $f : \{1, \ldots, n\} \to \{0, 1\}$ that takes the value 1 on precisely one input x, and an "advice" probability distribution $\mu = (p_y)$, $y \in \{1, \ldots, n\}$, where p_y is the probability that $f(y) = 1$.

Output: The marked element x.

- We want to minimise the expected number of queries to find x, under the distribution μ.
- Thus we are solving an average-case search problem.
- Going to an average-case model allows the possibility of exponential speed-ups [Ambainis & de Wolf '01].

The rest of this talk

- A quantum algorithm for Search with Advice
- Proof of optimality of the algorithm
- A different model where advice is expensive
- Application to power law distributions

The model

- For now, we assume that the probability distribution μ is known beforehand, and can be used to design the algorithm.

The model

- For now, we assume that the probability distribution μ is known beforehand, and can be used to design the algorithm.
- Let $T_{\mathcal{A}}(x)$ denote the expected number of queries to f used by an algorithm $\mathcal{A}$, when x is the marked element.

The model

- For now, we assume that the probability distribution μ is known beforehand, and can be used to design the algorithm.
- Let $T_{\mathcal{A}}(x)$ denote the expected number of queries to f used by an algorithm $\mathcal{A}$, when x is the marked element.
- Let $T_{\mathcal{A}}(\mu)$ be the expected number of queries to f used by $\mathcal{A}$ under distribution μ:

$$T_{\mathcal{A}}(\mu) = \sum_{x=1}^{n} p_x T_{\mathcal{A}}(x).$$

The model

- For now, we assume that the probability distribution μ is known beforehand, and can be used to design the algorithm.
- Let $T_{\mathcal{A}}(x)$ denote the expected number of queries to f used by an algorithm $\mathcal{A}$, when x is the marked element.
- Let $T_{\mathcal{A}}(\mu)$ be the expected number of queries to f used by $\mathcal{A}$ under distribution μ:

$$T_{\mathcal{A}}(\mu) = \sum_{x=1}^{n} p_x T_{\mathcal{A}}(x).$$

- Minimising over all algorithms, define the deterministic and quantum (resp.) average-case query complexities of μ:

$$D(\mu) = \min_{\mathcal{A} \text{ classical}} T_{\mathcal{A}}(\mu), \;\; Q(\mu) = \min_{\mathcal{A} \text{ quantum}} T_{\mathcal{A}}(\mu).$$

Classical algorithms

- Assume that p_y is non-increasing with y (so the most likely place for the marked element to be is at the start of the list, etc.).

Classical algorithms

- Assume that p_y is non-increasing with y (so the most likely place for the marked element to be is at the start of the list, etc.).
- Then the optimal classical algorithm to find x is simply to query $f(1)$ through $f(n)$ in turn.

Classical algorithms

- Assume that p_y is non-increasing with y (so the most likely place for the marked element to be is at the start of the list, etc.).
- Then the optimal classical algorithm to find x is simply to query $f(1)$ through $f(n)$ in turn.
- So the classical average-case query complexity can be written down as

$$D(\mu) = \sum_{x=1}^{n} p_x \, x.$$

Classical algorithms

- Assume that p_y is non-increasing with y (so the most likely place for the marked element to be is at the start of the list, etc.).

- Then the optimal classical algorithm to find x is simply to query $f(1)$ through $f(n)$ in turn.

- So the classical average-case query complexity can be written down as

$$D(\mu) = \sum_{x=1}^{n} p_x \, x.$$

- Sometimes much better than naive Grover search – can we do better with a new quantum algorithm?

Algorithm $\mathcal{A}$: search with a known probability distribution

1. Assume the probability distribution is in non-increasing order.

2. Divide the list into blocks that increase in size exponentially (with ratio c, for some constant c).

3. Run Grover search on each block in turn.

4. Stop when the marked item is found.

Algorithm $\mathcal{A}$: search with a known probability distribution

1. Assume the probability distribution is in non-increasing order.
2. Divide the list into blocks that increase in size exponentially (with ratio c, for some constant c).
3. Run Grover search on each block in turn.
4. Stop when the marked item is found.

Example (with $c = 2$):

Algorithm $\mathcal{A}$: search with a known probability distribution

1. Assume the probability distribution is in non-increasing order.
2. Divide the list into blocks that increase in size exponentially (with ratio c, for some constant c).
3. Run Grover search on each block in turn.
4. Stop when the marked item is found.

Example (with $c = 2$):

Algorithm $\mathcal{A}$: search with a known probability distribution

1. Assume the probability distribution is in non-increasing order.
2. Divide the list into blocks that increase in size exponentially (with ratio c, for some constant c).
3. Run Grover search on each block in turn.
4. Stop when the marked item is found.

Example (with $c = 2$):

Algorithm $\mathcal{A}$: search with a known probability distribution

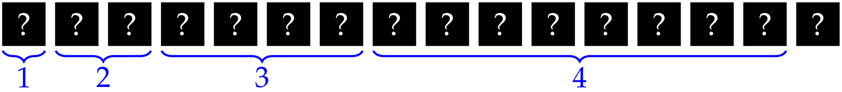
1. Assume the probability distribution is in non-increasing order.
2. Divide the list into blocks that increase in size exponentially (with ratio c, for some constant c).
3. Run Grover search on each block in turn.
4. Stop when the marked item is found.

Example (with $c = 2$):

Algorithm $\mathcal{A}$: search with a known probability distribution

1. Assume the probability distribution is in non-increasing order.
2. Divide the list into blocks that increase in size exponentially (with ratio c, for some constant c).
3. Run Grover search on each block in turn.
4. Stop when the marked item is found.

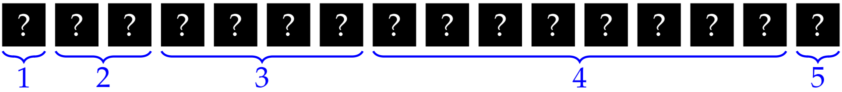
Example (with $c = 2$):



Algorithm $\mathcal{A}$: search with a known probability distribution

1. Assume the probability distribution is in non-increasing order.
2. Divide the list into blocks that increase in size exponentially (with ratio c, for some constant c).
3. Run Grover search on each block in turn.
4. Stop when the marked item is found.

Example (with $c = 2$):



Performance

Proposition

The average number of queries used by Algorithm $\mathcal{A}$, choosing $c = e \approx 2.718$, on an advice distribution $\mu = (p_x)$ is upper bounded by

$$\pi e \sum_{x=1}^{n} p_x \sqrt{x}.$$

Performance

Proposition

The average number of queries used by Algorithm $\mathcal{A}$, choosing $c = e \approx 2.718$, on an advice distribution $\mu = (p_x)$ is upper bounded by

$$\pi e \sum_{x=1}^{n} p_x \sqrt{x}.$$

Proof sketch:

- Searching the m'th block uses $O(c^{m/2})$ queries.
- When x is the marked item, at most $O(\log x)$ blocks are searched by the algorithm.
- So $O(\sqrt{x})$ queries are used on input x.

Optimality (1)

This algorithm is in fact optimal, up to a constant factor. To prove this, we need the following new result:

Proposition

Let $\mathcal{A}$ be a quantum search algorithm such that $T_{\mathcal{A}}(x) \leqslant T$ for all x, for some T. Then

$$T \geqslant \frac{0.206}{\arcsin 1/\sqrt{n}} - 0.316 \geqslant 0.206\sqrt{n} - 1.$$

Optimality (1)

This algorithm is in fact optimal, up to a constant factor. To prove this, we need the following new result:

Proposition

Let $\mathcal{A}$ be a quantum search algorithm such that $T_{\mathcal{A}}(x) \leqslant T$ for all x, for some T. Then

$$T \geqslant \frac{0.206}{\arcsin 1/\sqrt{n}} - 0.316 \geqslant 0.206\sqrt{n} - 1.$$

- This is an average-case variant of known worst-case $\Omega(\sqrt{n})$ lower bounds on quantum search.

Optimality (1)

This algorithm is in fact optimal, up to a constant factor. To prove this, we need the following new result:

Proposition

Let $\mathcal{A}$ be a quantum search algorithm such that $T_{\mathcal{A}}(x) \leqslant T$ for all x, for some T. Then

$$T \geqslant \frac{0.206}{\arcsin 1/\sqrt{n}} - 0.316 \geqslant 0.206\sqrt{n} - 1.$$

- This is an average-case variant of known worst-case $\Omega(\sqrt{n})$ lower bounds on quantum search.
- It's known that one can actually achieve an expected query complexity that is somewhat less than the usual worst-case query complexity guaranteed by Grover's algorithm [Boyer et al '98, Zalka '99].

Optimality (2)

Proposition

Let $\mu = (p_x)$, $x \in [n]$ be an arbitrary non-increasing probability distribution. Then

$$Q(\mu) \geqslant 0.206 \sum_{x=1}^{n} p_x \sqrt{x} - 1.$$

Optimality (2)

Proposition

Let $\mu = (p_x)$, $x \in [n]$ be an arbitrary non-increasing probability distribution. Then

$$Q(\mu) \geqslant 0.206 \sum_{x=1}^{n} p_x \sqrt{x} - 1.$$

Proof sketch:

- By previous proposition, there must exist a y such that $T_{\mathcal{A}}(y) \geqslant 0.206\sqrt{n} - 1$.
- Similarly, there must exist $y' \neq y$ such that $T_{\mathcal{A}}(y') \geqslant 0.206\sqrt{n-1} - 1$.
- Iterating this argument and rearranging gives the result.

Unknown probability distribution

- We can also consider a model where we don't know the advice probability distribution at the start.

Unknown probability distribution

- We can also consider a model where we don't know the advice probability distribution at the start.

- Model: can create the state $|\mu\rangle = \sum_x \sqrt{p_x}|x\rangle$, at the cost of one query. So we can "quantum sample" from μ.

Unknown probability distribution

- We can also consider a model where we don't know the advice probability distribution at the start.

- Model: can create the state $|\mu\rangle = \sum_x \sqrt{p_x}|x\rangle$, at the cost of one query. So we can "quantum sample" from μ.

- In some cases, quantum sampling can be efficient – such as when (p_x) is efficiently integrable [Grover & Rudolph '02].

Unknown probability distribution

- We can also consider a model where we don't know the advice probability distribution at the start.
- Model: can create the state $|\mu\rangle = \sum_x \sqrt{p_x}|x\rangle$, at the cost of one query. So we can "quantum sample" from μ.
- In some cases, quantum sampling can be efficient – such as when (p_x) is efficiently integrable [Grover & Rudolph '02].
- Note that querying in accordance with classical sampling is no better than querying uniformly at random!

Unknown probability distribution

- Let $T^*_{\mathcal{A}}(\mu)$ denote the expected number of queries used by some algorithm $\mathcal{A}$ on distribution μ in this model.

- We present a new quantum algorithm $\mathcal{B}$ that achieves

$$T^*_{\mathcal{B}}(\mu) = K\left(\sum_{x, p_x > 1/n} \sqrt{p_x}\right) + L\sqrt{n}\left(\sum_{x, p_x \leqslant 1/n} p_x\right) + M$$

  for some constants K, L, M.

Unknown probability distribution

- Let $T^*_{\mathcal{A}}(\mu)$ denote the expected number of queries used by some algorithm $\mathcal{A}$ on distribution μ in this model.

- We present a new quantum algorithm $\mathcal{B}$ that achieves

$$T^*_{\mathcal{B}}(\mu) = K\left(\sum_{x,p_x>1/n} \sqrt{p_x}\right) + L\sqrt{n}\left(\sum_{x,p_x\leqslant 1/n} p_x\right) + M$$

  for some constants K, L, M.

- Sometimes significantly better than any classical algorithm (even one that knows μ at the start).

- The new algorithm is based on amplitude amplification.

Amplitude amplification [Brassard et al '02]

Input: Function $f : [n] \to \{0,1\}$ such that f takes the value 1 on precisely one input x; oracle operator $O_\mu : |0\rangle \mapsto |\mu\rangle$; inverse O_μ^{-1}; positive integer k (number of iterations)

Output: The marked element x, or fail

Amplitude amplification [Brassard et al '02]

Input: Function $f : [n] \to \{0,1\}$ such that f takes the value 1 on precisely one input x; oracle operator $O_\mu : |0\rangle \mapsto |\mu\rangle$; inverse O_μ^{-1}; positive integer k (number of iterations)
Output: The marked element x, or fail
create initial state $|\mu\rangle = O_\mu|0\rangle$;
apply operator $-O_\mu I_{|0\rangle} O_\mu^{-1} I_{|x\rangle}$ k times to $|\mu\rangle$;
measure in computational basis, obtaining outcome y;
if *f(y)=1* **then**
    return *y*;
else
    return *fail*;
end

Amplitude amplification [Brassard et al '02]

Input: Function $f : [n] \to \{0,1\}$ such that f takes the value 1 on precisely one input x; oracle operator $O_\mu : |0\rangle \mapsto |\mu\rangle$; inverse O_μ^{-1}; positive integer k (number of iterations)

Output: The marked element x, or fail

create initial state $|\mu\rangle = O_\mu|0\rangle$;
apply operator $-O_\mu I_{|0\rangle} O_\mu^{-1} I_{|x\rangle}$ k times to $|\mu\rangle$;
measure in computational basis, obtaining outcome y;
if *f(y)=1* **then**
    return y;
else
    return *fail*;
end

Lemma

Applying the above algorithm with k iterations returns the location of the marked element with probability $\sin^2((2k+1)\arcsin\sqrt{p_x})$, *using* $k+1$ *queries to* O_μ, k *queries to* O_μ^{-1}, *and* $k+1$ *queries to* f.

Algorithm $\mathcal{B}$: unknown distribution

Input: Function $f : [n] \to \{0, 1\}$ such that f takes the value 1 on precisely one input x; oracle operator $O_\mu : |0\rangle \mapsto |\mu\rangle$; inverse O_μ^{-1}; real $k > 1$

Output: The marked element x

Algorithm $\mathcal{B}$: unknown distribution

Input: Function $f : [n] \to \{0, 1\}$ such that f takes the value 1 on precisely one input x; oracle operator $O_\mu : |0\rangle \mapsto |\mu\rangle$; inverse O_μ^{-1}; real $k > 1$

Output: The marked element x

for $j = 0$ *to* $\lfloor \log_k \sqrt{n} \rfloor$ **do**

  sample from distribution μ;

  if *marked element found* **then**

    return *marked element*;

  end

  pick i uniformly at random from integers $\{0, \ldots, \lfloor k^j \rfloor - 1\}$;

  perform i iterations of amplitude amplification;

  if *marked element found* **then**

    return *marked element*;

  end

end

perform exact Grover search for one marked element on $[n]$;

return *marked element*;

Results (unknown probability distribution)

Proposition

On input x, when called with $k \approx 1.162$, Algorithm $\mathcal{B}$ uses an expected number of at most $\min\{83/\sqrt{p_x} + 4/3, 53\sqrt{n}\}$ *queries to each of f, O_μ, O_μ^{-1}.*

Results (unknown probability distribution)

Proposition

On input x, when called with $k \approx 1.162$, Algorithm $\mathcal{B}$ uses an expected number of at most $\min\{83/\sqrt{p_x} + 4/3, 53\sqrt{n}\}$ *queries to each of f, O_μ, O_μ^{-1}.*

Are there any "natural" advice distributions to which we could apply these results?

Power law distributions

Let $\mu = (p_x)$, $x \in [n]$ be a probability distribution where $p_x \propto x^k$ for some constant $k < 0$. Then

$$D(\mu) = \begin{cases} \Theta(n) & [-1<k<0] \\ \Theta(n/\log n) & [k=-1] \\ \Theta(n^{k+2}) & [-2<k<-1] \\ \Theta(\log n) & [k=-2] \\ \Theta(1) & [k<-2] \end{cases}, \quad Q(\mu) = \begin{cases} \Theta(\sqrt{n}) & [-1<k<0] \\ \Theta(\sqrt{n}/\log n) & [k=-1] \\ \Theta(n^{k+3/2}) & [-3/2<k<-1] \\ \Theta(\log n) & [k=-3/2] \\ \Theta(1) & [k<-3/2] \end{cases}$$

Power law distributions

Let $\mu = (p_x)$, $x \in [n]$ be a probability distribution where $p_x \propto x^k$ for some constant $k < 0$. Then

$$D(\mu) = \begin{cases} \Theta(n) & [-1<k<0] \\ \Theta(n/\log n) & [k=-1] \\ \Theta(n^{k+2}) & [-2<k<-1] \\ \Theta(\log n) & [k=-2] \\ \Theta(1) & [k<-2] \end{cases}, \quad Q(\mu) = \begin{cases} \Theta(\sqrt{n}) & [-1<k<0] \\ \Theta(\sqrt{n}/\log n) & [k=-1] \\ \Theta(n^{k+3/2}) & [-3/2<k<-1] \\ \Theta(\log n) & [k=-3/2] \\ \Theta(1) & [k<-3/2] \end{cases}$$

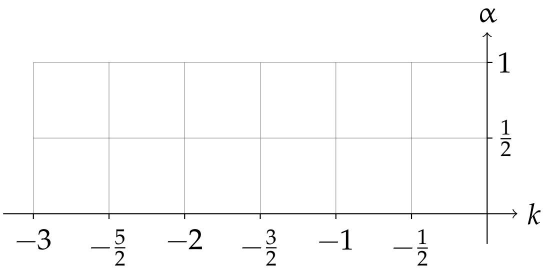
Corollary

There exists a probability distribution μ such that $D(\mu) = \Omega(n^{1/2-\epsilon})$ for arbitrary $\epsilon > 0$, but $Q(\mu) = O(1)$.

A super-exponential average-case query complexity separation!

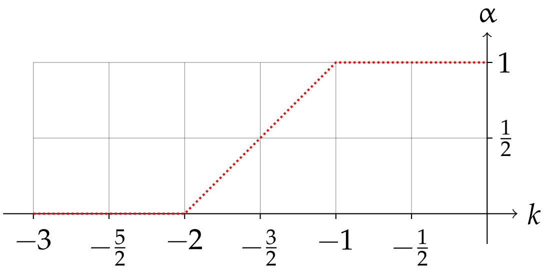
Power law distributions $p_x \propto x^k$

For each k, query complexity is $\Theta(n^\alpha)$ for some α (ignoring log factors). Plotting α against k gives



Power law distributions $p_x \propto x^k$

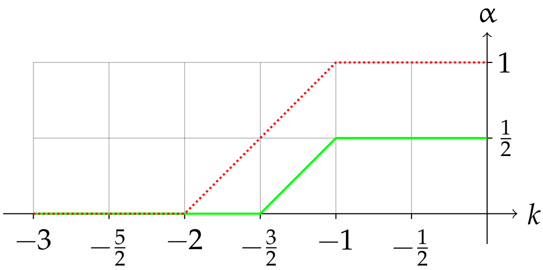
For each k, query complexity is $\Theta(n^\alpha)$ for some α (ignoring log factors). Plotting α against k gives



- Dotted red line: best possible classical algorithm

Power law distributions $p_x \propto x^k$

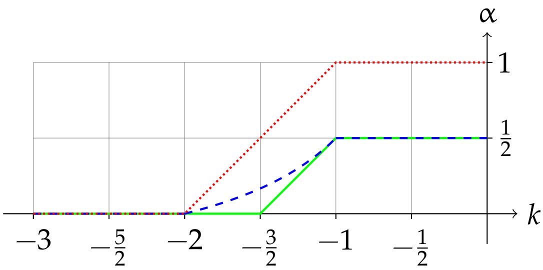
For each k, query complexity is $\Theta(n^\alpha)$ for some α (ignoring log factors). Plotting α against k gives



- Dotted red line: best possible classical algorithm
- Solid green line: quantum, known probability distribution

Power law distributions $p_x \propto x^k$

For each k, query complexity is $\Theta(n^\alpha)$ for some α (ignoring log factors). Plotting α against k gives



- Dotted red line: best possible classical algorithm
- Solid green line: quantum, known probability distribution
- Dashed blue line: quantum, unknown probability distribution

Conclusions

- We've seen that quantum search can dramatically outperform classical search in a model where we're given advice about where to look.
- Moving to an average-case model allows us to obtain (super-)exponential speed-ups.
- These speed-ups are obtained for (fairly) natural advice distributions.
- Applying easy(ish) classical algorithmic techniques to quantum algorithms can lead to significant speed-ups.

Conclusions

- We've seen that quantum search can dramatically outperform classical search in a model where we're given advice about where to look.

- Moving to an average-case model allows us to obtain (super-)exponential speed-ups.

- These speed-ups are obtained for (fairly) natural advice distributions.

- Applying easy(ish) classical algorithmic techniques to quantum algorithms can lead to significant speed-ups.

Applications?

The end

Further reading:

- The paper: arxiv.org/abs/0908.3066
- An introduction to quantum computing for A-level students: www.cs.bris.ac.uk/~montanar/gameshow.pdf
- A more detailed introduction: Richard Jozsa's lecture notes, www.cs.bris.ac.uk/Teaching/Resources/COMSM0214/

The end

Further reading:

- The paper: arxiv.org/abs/0908.3066
- An introduction to quantum computing for A-level students: www.cs.bris.ac.uk/~montanar/gameshow.pdf
- A more detailed introduction: Richard Jozsa's lecture notes, www.cs.bris.ac.uk/Teaching/Resources/COMSM0214/

Thanks for your time!