Quantum walk speedup of backtracking algorithms

Ashley Montanaro

School of Mathematics, University of Bristol

11 January 2016

arXiv:1509.02374

University of Bristol

EPSRC
Engineering and Physical Sciences Research Council

Constraint satisfaction problems

This talk is about a quantum algorithm for solving general constraint satisfaction problems (CSPs).

Constraint satisfaction problems

This talk is about a quantum algorithm for solving general constraint satisfaction problems (CSPs).

- An instance of a CSP on n variables $x_1, \dots, x_n$ is specified by a sequence of constraints, all of which must be satisfied by the variables.

Constraint satisfaction problems

This talk is about a quantum algorithm for solving general constraint satisfaction problems (CSPs).

- An instance of a CSP on n variables $x_1, \dots, x_n$ is specified by a sequence of constraints, all of which must be satisfied by the variables.
- We might want to find one assignment to the variables that satisfies all the constraints, or list all such assignments.

Constraint satisfaction problems

This talk is about a quantum algorithm for solving general constraint satisfaction problems (CSPs).

- An instance of a CSP on n variables $x_1, \dots, x_n$ is specified by a sequence of constraints, all of which must be satisfied by the variables.
- We might want to find one assignment to the variables that satisfies all the constraints, or list all such assignments.
- For many CSPs, the best algorithms known for either task have exponential runtime in n.

Constraint satisfaction problems

This talk is about a quantum algorithm for solving general constraint satisfaction problems (CSPs).

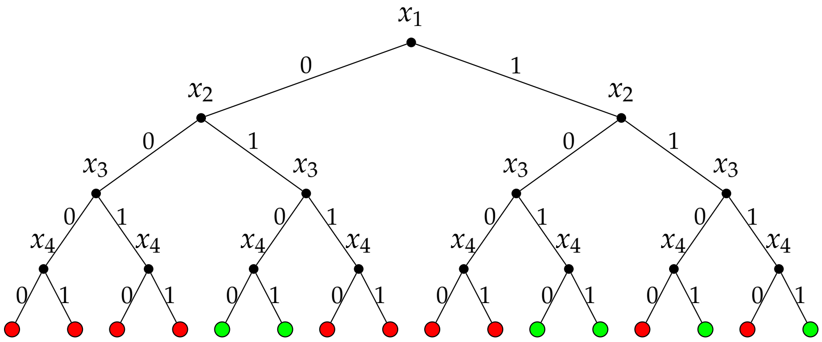
- An instance of a CSP on n variables $x_1, \dots, x_n$ is specified by a sequence of constraints, all of which must be satisfied by the variables.
- We might want to find one assignment to the variables that satisfies all the constraints, or list all such assignments.
- For many CSPs, the best algorithms known for either task have exponential runtime in n.
- A fundamental example: boolean satisfiability with at most 3 variables per clause (3-SAT).

$$(x_1 \vee x_2) \;\wedge\; (x_1 \vee \neg x_3) \;\wedge\; (\neg x_1 \vee \neg x_2 \vee x_4) \;\wedge\; (x_2 \vee x_3)$$

A naïve algorithm

$$(x_1 \vee x_2) \ \wedge \ (x_1 \vee \neg x_3) \ \wedge \ (\neg x_1 \vee \neg x_2 \vee x_4) \ \wedge \ (x_2 \vee x_3)$$

Imagine we want to find all satisfying assignments. One naïve way of doing this is exhaustive search:



A less naïve algorithm

$$(x_1 \vee x_2) \ \wedge \ (x_1 \vee \neg x_3) \ \wedge \ (\neg x_1 \vee \neg x_2 \vee x_4) \ \wedge \ (x_2 \vee x_3)$$

Some paths in this tree are disallowed early on…

- For example, if we set $x_1 = 0$, $x_2 = 0$, we already know the formula is false.

A less naïve algorithm

$$(x_1 \vee x_2) \;\wedge\; (x_1 \vee \neg x_3) \;\wedge\; (\neg x_1 \vee \neg x_2 \vee x_4) \;\wedge\; (x_2 \vee x_3)$$

Some paths in this tree are disallowed early on…

- For example, if we set $x_1 = 0$, $x_2 = 0$, we already know the formula is false.
- We can modify the above algorithm to explore a smaller tree by checking whether the formula is true (or false) at internal nodes in the tree.

A less naïve algorithm

$$(x_1 \vee x_2) \;\wedge\; (x_1 \vee \neg x_3) \;\wedge\; (\neg x_1 \vee \neg x_2 \vee x_4) \;\wedge\; (x_2 \vee x_3)$$

Some paths in this tree are disallowed early on…

- For example, if we set $x_1 = 0$, $x_2 = 0$, we already know the formula is false.
- We can modify the above algorithm to explore a smaller tree by checking whether the formula is true (or false) at internal nodes in the tree.
- Exploring the tree corresponds to substituting variable values into the formula.

A less naïve algorithm

$$(x_1 \vee x_2) \;\wedge\; (x_1 \vee \neg x_3) \;\wedge\; (\neg x_1 \vee \neg x_2 \vee x_4) \;\wedge\; (x_2 \vee x_3)$$

Some paths in this tree are disallowed early on…

- For example, if we set $x_1 = 0, x_2 = 0$, we already know the formula is false.
- We can modify the above algorithm to explore a smaller tree by checking whether the formula is true (or false) at internal nodes in the tree.
- Exploring the tree corresponds to substituting variable values into the formula.
- At each vertex, we determine which variable to choose next using a heuristic.

A less naïve algorithm

$$(x_1 \vee x_2) \ \wedge \ (x_1 \vee \neg x_3) \ \wedge \ (\neg x_1 \vee \neg x_2 \vee x_4) \ \wedge \ (x_2 \vee x_3)$$

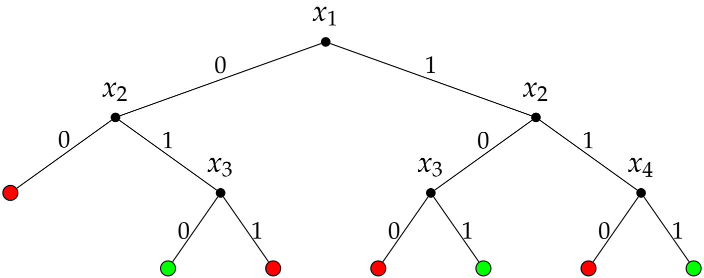
Imagine we use the following heuristic: choose an arbitrary variable in a shortest clause.

A less naïve algorithm

$$(x_1 \vee x_2) \ \wedge \ (x_1 \vee \neg x_3) \ \wedge \ (\neg x_1 \vee \neg x_2 \vee x_4) \ \wedge \ (x_2 \vee x_3)$$

Imagine we use the following heuristic: choose an arbitrary variable in a shortest clause.

Then we can get the following smaller tree:

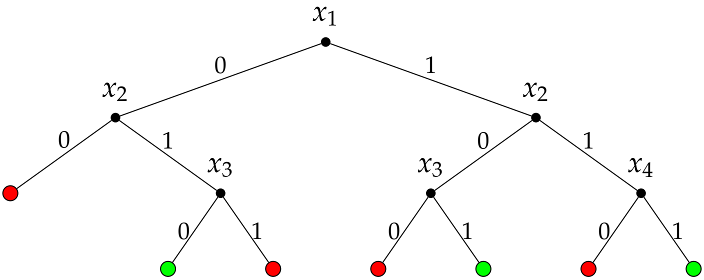


A less naïve algorithm

$$(x_1 \vee x_2) \ \wedge \ (x_1 \vee \neg x_3) \ \wedge \ (\neg x_1 \vee \neg x_2 \vee x_4) \ \wedge \ (x_2 \vee x_3)$$

Imagine we use the following heuristic: choose an arbitrary variable in a shortest clause.

Then we can get the following smaller tree:



This algorithm is a simple variant of the DPLL algorithm, which forms the basis of many of the most efficient SAT solvers used in practice.

General backtracking framework

Suppose we want to solve a constraint satisfaction problem on n variables, each picked from $[d] := \{0, \ldots, d-1\}$.

- Write $\mathcal{D} := ([d] \cup \{*\})^n$, where $*$ means "not assigned yet".

General backtracking framework

Suppose we want to solve a constraint satisfaction problem on n variables, each picked from $[d] := \{0, \ldots, d-1\}$.

- Write $\mathcal{D} := ([d] \cup \{*\})^n$, where $*$ means "not assigned yet".
- Assume we have access to a predicate

  $$P : \mathcal{D} \to \{\text{true}, \text{false}, \text{indeterminate}\}$$

  which tells us the status of a partial assignment.

General backtracking framework

Suppose we want to solve a constraint satisfaction problem on n variables, each picked from $[d] := \{0, \ldots, d-1\}$.

- Write $\mathcal{D} := ([d] \cup \{*\})^n$, where $*$ means "not assigned yet".
- Assume we have access to a predicate

  $$P : \mathcal{D} \to \{\text{true}, \text{false}, \text{indeterminate}\}$$

  which tells us the status of a partial assignment.
- Also assume we have access to a heuristic

  $$h : \mathcal{D} \to \{1, \ldots, n\}$$

  which returns the next index to branch on from a given partial assignment.

General backtracking framework

Suppose we want to solve a constraint satisfaction problem on n variables, each picked from $[d] := \{0, \ldots, d-1\}$.

- Write $\mathcal{D} := ([d] \cup \{*\})^n$, where $*$ means "not assigned yet".
- Assume we have access to a predicate

  $$P : \mathcal{D} \to \{\text{true}, \text{false}, \text{indeterminate}\}$$

  which tells us the status of a partial assignment.
- Also assume we have access to a heuristic

  $$h : \mathcal{D} \to \{1, \ldots, n\}$$

  which returns the next index to branch on from a given partial assignment.
- Also allows randomised heuristics, as distributions over deterministic functions h.

Main result

Theorem

Let T be the number of vertices in the backtracking tree. Then there is a bounded-error quantum algorithm which evaluates P and h $O(\sqrt{T}n^{3/2}\log n)$ times each, and outputs x such that $P(x)$ is true, or "not found" if no such x exists.

Main result

Theorem

Let T be the number of vertices in the backtracking tree. Then there is a bounded-error quantum algorithm which evaluates P and h $O(\sqrt{T}n^{3/2}\log n)$ times each, and outputs x such that $P(x)$ is true, or "not found" if no such x exists.

If we are promised that there exists a unique x_0 such that $P(x_0)$ is true, this is improved to $O(\sqrt{Tn}\log^3 n)$.

In both cases the algorithm uses $\text{poly}(n)$ space and $\text{poly}(n)$ auxiliary quantum gates per use of P and h.

Main result

Theorem

Let T be the number of vertices in the backtracking tree. Then there is a bounded-error quantum algorithm which evaluates P and h $O(\sqrt{T}n^{3/2}\log n)$ times each, and outputs x such that $P(x)$ is true, or "not found" if no such x exists.

If we are promised that there exists a unique x_0 such that $P(x_0)$ is true, this is improved to $O(\sqrt{Tn}\log^3 n)$.

In both cases the algorithm uses $\text{poly}(n)$ space and $\text{poly}(n)$ auxiliary quantum gates per use of P and h.

- The algorithm can be modified to find all solutions by striking out previously seen solutions.

Main result

Theorem

Let T be the number of vertices in the backtracking tree. Then there is a bounded-error quantum algorithm which evaluates P and h $O(\sqrt{T}n^{3/2}\log n)$ times each, and outputs x such that $P(x)$ is true, or "not found" if no such x exists.

If we are promised that there exists a unique x_0 such that $P(x_0)$ is true, this is improved to $O(\sqrt{Tn}\log^3 n)$.

In both cases the algorithm uses poly(n) space and poly(n) auxiliary quantum gates per use of P and h.

- The algorithm can be modified to find all solutions by striking out previously seen solutions.
- We usually think of T as being exponentially large in n. In this regime, this is a near-quadratic separation.

Main result

Theorem

Let T be the number of vertices in the backtracking tree. Then there is a bounded-error quantum algorithm which evaluates P and h $O(\sqrt{T}n^{3/2}\log n)$ times each, and outputs x such that $P(x)$ is true, or "not found" if no such x exists.

If we are promised that there exists a unique x_0 such that $P(x_0)$ is true, this is improved to $O(\sqrt{Tn}\log^3 n)$.

In both cases the algorithm uses $\mathrm{poly}(n)$ space and $\mathrm{poly}(n)$ auxiliary quantum gates per use of P and h.

- The algorithm can be modified to find all solutions by striking out previously seen solutions.
- We usually think of T as being exponentially large in n. In this regime, this is a near-quadratic separation.
- Note that the algorithm does not need to know T.

Previous work

Some previous works have developed quantum algorithms related to backtracking:

- [Cerf, Grover and Williams '00] developed a quantum algorithm for constraint satisfaction problems, based on a nested version of Grover search. This can be seen as a quantum version of one particular backtracking algorithm that runs quadratically faster.

Previous work

Some previous works have developed quantum algorithms related to backtracking:

- [Cerf, Grover and Williams '00] developed a quantum algorithm for constraint satisfaction problems, based on a nested version of Grover search. This can be seen as a quantum version of one particular backtracking algorithm that runs quadratically faster.

- [Farhi and Gutmann '98] used continuous-time quantum walks to find solutions in backtracking trees. They showed that, for some trees, the quantum walk can find a solution exponentially faster than a classical random walk.

Previous work

Some previous works have developed quantum algorithms related to backtracking:

- [Cerf, Grover and Williams '00] developed a quantum algorithm for constraint satisfaction problems, based on a nested version of Grover search. This can be seen as a quantum version of one particular backtracking algorithm that runs quadratically faster.
- [Farhi and Gutmann '98] used continuous-time quantum walks to find solutions in backtracking trees. They showed that, for some trees, the quantum walk can find a solution exponentially faster than a classical random walk.

By contrast, the algorithm presented here achieves a (nearly) quadratic separation for all trees.

Search in the backtracking tree

Idea: Use quantum search to find a marked vertex (i.e. solution) in the tree produced by the backtracking algorithm.

Search in the backtracking tree

Idea: Use quantum search to find a marked vertex (i.e. solution) in the tree produced by the backtracking algorithm.

Many works have studied quantum search in various graphs, e.g. [Szegedy '04], [Aaronson and Ambainis '05], [Magniez et al. '11] ...

Search in the backtracking tree

Idea: Use quantum search to find a marked vertex (i.e. solution) in the tree produced by the backtracking algorithm.

Many works have studied quantum search in various graphs, e.g. [Szegedy '04], [Aaronson and Ambainis '05], [Magniez et al. '11] ...

But here there are some difficulties:

- The graph is not known in advance, and is determined by the backtracking algorithm.

Search in the backtracking tree

Idea: Use quantum search to find a marked vertex (i.e. solution) in the tree produced by the backtracking algorithm.

Many works have studied quantum search in various graphs, e.g. [Szegedy '04], [Aaronson and Ambainis '05], [Magniez et al. '11] . . .

But here there are some difficulties:

- The graph is not known in advance, and is determined by the backtracking algorithm.
- We start at the root of the tree, not in the stationary distribution of a random walk on the graph.

Search in the backtracking tree

Idea: Use quantum search to find a marked vertex (i.e. solution) in the tree produced by the backtracking algorithm.

Many works have studied quantum search in various graphs, e.g. [Szegedy '04], [Aaronson and Ambainis '05], [Magniez et al. '11] ...

But here there are some difficulties:

- The graph is not known in advance, and is determined by the backtracking algorithm.
- We start at the root of the tree, not in the stationary distribution of a random walk on the graph.

These can be overcome using work of [Belovs '13] relating quantum walks to effective resistance in an electrical network.

Quantum walk in a tree

The quantum walk operates on a T-dimensional Hilbert space spanned by $\{|r\rangle\} \cup \{|x\rangle : x \in \{1, \ldots, T-1\}\}$, where r is the root.

Quantum walk in a tree

The quantum walk operates on a T-dimensional Hilbert space spanned by $\{|r\rangle\} \cup \{|x\rangle : x \in \{1, \ldots, T-1\}\}$, where r is the root.

The walk starts in the state $|r\rangle$ and is based on a set of diffusion operators D_x, where D_x acts on the subspace $\mathcal{H}_x$ spanned by $\{|x\rangle\} \cup \{|y\rangle : x \to y\}$:

Quantum walk in a tree

The quantum walk operates on a T-dimensional Hilbert space spanned by $\{|r\rangle\} \cup \{|x\rangle : x \in \{1, \ldots, T-1\}\}$, where r is the root.

The walk starts in the state $|r\rangle$ and is based on a set of diffusion operators D_x, where D_x acts on the subspace $\mathcal{H}_x$ spanned by $\{|x\rangle\} \cup \{|y\rangle : x \to y\}$:

- If x is marked, then D_x is the identity.

Quantum walk in a tree

The quantum walk operates on a T-dimensional Hilbert space spanned by $\{|r\rangle\} \cup \{|x\rangle : x \in \{1, \ldots, T-1\}\}$, where r is the root.

The walk starts in the state $|r\rangle$ and is based on a set of diffusion operators D_x, where D_x acts on the subspace $\mathcal{H}_x$ spanned by $\{|x\rangle\} \cup \{|y\rangle : x \to y\}$:

- If x is marked, then D_x is the identity.
- If x is not marked, and $x \neq r$, then $D_x = I - 2|\psi_x\rangle\langle\psi_x|$, where
$$|\psi_x\rangle \propto |x\rangle + \sum_{y, x \to y} |y\rangle.$$

Quantum walk in a tree

The quantum walk operates on a T-dimensional Hilbert space spanned by $\{|r\rangle\} \cup \{|x\rangle : x \in \{1, \ldots, T-1\}\}$, where r is the root.

The walk starts in the state $|r\rangle$ and is based on a set of diffusion operators D_x, where D_x acts on the subspace $\mathcal{H}_x$ spanned by $\{|x\rangle\} \cup \{|y\rangle : x \to y\}$:

- If x is marked, then D_x is the identity.
- If x is not marked, and $x \neq r$, then $D_x = I - 2|\psi_x\rangle\langle\psi_x|$, where
$$|\psi_x\rangle \propto |x\rangle + \sum_{y, x \to y} |y\rangle.$$
- $D_r = I - 2|\psi_r\rangle\langle\psi_r|$, where
$$|\psi_r\rangle \propto |r\rangle + \sqrt{n} \sum_{y, r \to y} |y\rangle.$$

Quantum walk in a tree

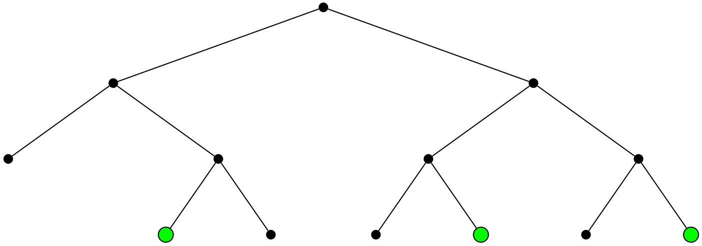
Let A and B be the sets of vertices an even and odd distance from the root, respectively.

Then a step of the walk consists of applying the operator $R_B R_A$, where $R_A = \bigoplus_{x \in A} D_x$ and $R_B = |r\rangle\langle r| + \bigoplus_{x \in B} D_x$.

Quantum walk in a tree

Let A and B be the sets of vertices an even and odd distance from the root, respectively.

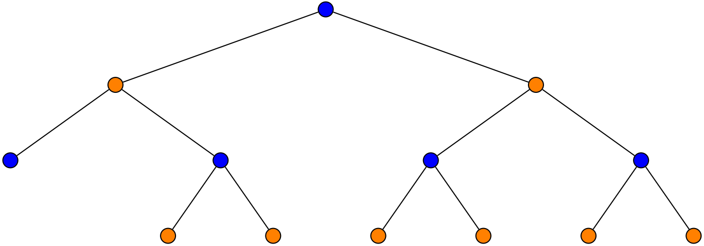
Then a step of the walk consists of applying the operator $R_B R_A$, where $R_A = \bigoplus_{x \in A} D_x$ and $R_B = |r\rangle\langle r| + \bigoplus_{x \in B} D_x$.



Quantum walk in a tree

Let A and B be the sets of vertices an even and odd distance from the root, respectively.

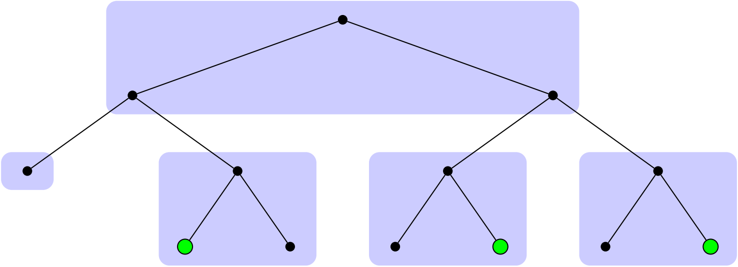
Then a step of the walk consists of applying the operator $R_B R_A$, where $R_A = \bigoplus_{x \in A} D_x$ and $R_B = |r\rangle\langle r| + \bigoplus_{x \in B} D_x$.



Quantum walk in a tree

Let A and B be the sets of vertices an even and odd distance from the root, respectively.

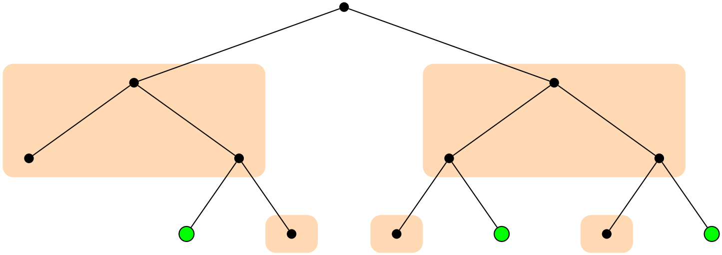
Then a step of the walk consists of applying the operator $R_B R_A$, where $R_A = \bigoplus_{x \in A} D_x$ and $R_B = |r\rangle\langle r| + \bigoplus_{x \in B} D_x$.



Quantum walk in a tree

Let A and B be the sets of vertices an even and odd distance from the root, respectively.

Then a step of the walk consists of applying the operator $R_B R_A$, where $R_A = \bigoplus_{x \in A} D_x$ and $R_B = |r\rangle\langle r| + \bigoplus_{x \in B} D_x$.



Using the walk

We apply phase estimation to $R_B R_A$ on state $|r\rangle$ with precision $O(1/\sqrt{Tn})$, where n is an upper bound on the depth of the tree, and accept if the eigenvalue is 1.

Using the walk

We apply phase estimation to $R_B R_A$ on state $|r\rangle$ with precision $O(1/\sqrt{Tn})$, where n is an upper bound on the depth of the tree, and accept if the eigenvalue is 1.

Claim (special case of [Belovs '13])

- If there is a marked vertex, $R_B R_A$ has a normalised eigenvector with eigenvalue 1 and overlap $\geqslant \frac{1}{2}$ with $|r\rangle$.

Using the walk

We apply phase estimation to $R_B R_A$ on state $|r\rangle$ with precision $O(1/\sqrt{Tn})$, where n is an upper bound on the depth of the tree, and accept if the eigenvalue is 1.

Claim (special case of [Belovs '13])

- If there is a marked vertex, $R_B R_A$ has a normalised eigenvector with eigenvalue 1 and overlap $\geqslant \frac{1}{2}$ with $|r\rangle$.
- If there is no marked vertex, $\|P_\chi |r\rangle\|^2 \leqslant \frac{1}{4}$, where P_χ is the projector onto the space spanned by eigenvectors of $R_B R_A$ with eigenvalue $e^{2i\theta}$, for $|\theta| \leqslant 1/(2\sqrt{Tn})$.

Using the walk

We apply phase estimation to $R_B R_A$ on state $|r\rangle$ with precision $O(1/\sqrt{Tn})$, where n is an upper bound on the depth of the tree, and accept if the eigenvalue is 1.

Claim (special case of [Belovs '13])

- If there is a marked vertex, $R_B R_A$ has a normalised eigenvector with eigenvalue 1 and overlap $\geqslant \frac{1}{2}$ with $|r\rangle$.
- If there is no marked vertex, $\| P_\chi |r\rangle \|^2 \leqslant \frac{1}{4}$, where P_χ is the projector onto the space spanned by eigenvectors of $R_B R_A$ with eigenvalue $e^{2i\theta}$, for $|\theta| \leqslant 1/(2\sqrt{Tn})$.

It follows that we can use the above subroutine to detect a marked vertex with $O(\sqrt{Tn})$ uses of $R_B R_A$.

From detection to search

- We can use the above detection procedure as a subroutine to find marked vertices in the tree, via binary search.

From detection to search

- We can use the above detection procedure as a subroutine to find marked vertices in the tree, via binary search.
- We first apply the procedure to the whole tree. If it outputs “marked vertex exists” we apply it to the subtree rooted at each of the children of the root in turn and repeat.

From detection to search

- We can use the above detection procedure as a subroutine to find marked vertices in the tree, via binary search.

- We first apply the procedure to the whole tree. If it outputs "marked vertex exists" we apply it to the subtree rooted at each of the children of the root in turn and repeat.

- There is a more efficient algorithm if there is exactly one marked vertex, using the fact that the eigenvector with eigenvalue 1 encodes the entire path from the root to the marked vertex.

From quantum tree search to accelerating backtracking

We can now use this search algorithm to speed up the classical backtracking algorithm:

From quantum tree search to accelerating backtracking

We can now use this search algorithm to speed up the classical backtracking algorithm:

- Recall that we have access to P and h.

From quantum tree search to accelerating backtracking

We can now use this search algorithm to speed up the classical backtracking algorithm:

- Recall that we have access to P and h.
- Represent each vertex in the tree by a string $(i_1, v_1), \dots, (i_\ell, v_\ell)$ giving the indices and values of the variables set so far.

From quantum tree search to accelerating backtracking

We can now use this search algorithm to speed up the classical backtracking algorithm:

- Recall that we have access to P and h.
- Represent each vertex in the tree by a string $(i_1, v_1), \ldots, (i_\ell, v_\ell)$ giving the indices and values of the variables set so far.
- Then we can use P and h to determine the neighbours of each vertex. This allows us to implement the D_x operations (efficiently).

Summary and open problems

- If we have a classical backtracking algorithm whose tree has T vertices, there is a quantum algorithm which finds a solution in time $O(\sqrt{T}\,\mathrm{poly}(n))$.

Summary and open problems

- If we have a classical backtracking algorithm whose tree has T vertices, there is a quantum algorithm which finds a solution in time $O(\sqrt{T}\,\mathrm{poly}(n))$.
- This algorithm speeds up DPLL, the basis of many of the fastest SAT solvers used in practice.

Summary and open problems

- If we have a classical backtracking algorithm whose tree has T vertices, there is a quantum algorithm which finds a solution in time $O(\sqrt{T}\,\mathrm{poly}(n))$.
- This algorithm speeds up DPLL, the basis of many of the fastest SAT solvers used in practice.

Open problems:

- What if the classical algorithm is lucky and finds a solution early on?

Summary and open problems

- If we have a classical backtracking algorithm whose tree has T vertices, there is a quantum algorithm which finds a solution in time $O(\sqrt{T}\,\mathrm{poly}(n))$.
- This algorithm speeds up DPLL, the basis of many of the fastest SAT solvers used in practice.

Open problems:

- What if the classical algorithm is lucky and finds a solution early on?
- Can we improve the runtime for finding a solution to the best possible $O(\sqrt{Tn})$?

Summary and open problems

- If we have a classical backtracking algorithm whose tree has T vertices, there is a quantum algorithm which finds a solution in time $O(\sqrt{T}\,\mathrm{poly}(n))$.
- This algorithm speeds up DPLL, the basis of many of the fastest SAT solvers used in practice.

Open problems:

- What if the classical algorithm is lucky and finds a solution early on?
- Can we improve the runtime for finding a solution to the best possible $O(\sqrt{Tn})$?
- If there are k solutions, can we find them all in time $O(\sqrt{Tnk})$?

Summary and open problems

- If we have a classical backtracking algorithm whose tree has T vertices, there is a quantum algorithm which finds a solution in time $O(\sqrt{T}\,\mathrm{poly}(n))$.
- This algorithm speeds up DPLL, the basis of many of the fastest SAT solvers used in practice.

Open problems:

- What if the classical algorithm is lucky and finds a solution early on?
- Can we improve the runtime for finding a solution to the best possible $O(\sqrt{Tn})$?
- If there are k solutions, can we find them all in time $O(\sqrt{Tnk})$?
- What else can we do using the electrical circuit framework of [Belovs '13]?

Thanks!

General backtracking framework

Backtracking algorithm

Return $\texttt{bt}(*^n)$, where $\texttt{bt}$ is the following recursive procedure:

$\texttt{bt}(x)$:

1. If $P(x)$ is true, output x and return.
2. If $P(x)$ is false, return.
3. Set $j = h(x)$.
4. For each $w \in [d]$:
    1. Set y to x with the j'th entry replaced with w.
    2. Call $\texttt{bt}(y)$.

This algorithm runs in time at most $O(d^n)$, but on some instances its runtime can be substantially lower.

Exponentially reduced average runtime

The above algorithm has an instance-dependent runtime: If the classical algorithm uses time T on a given problem instance, the quantum algorithm uses time $O(\sqrt{T}\,\mathrm{poly}(n))$.

Exponentially reduced average runtime

The above algorithm has an instance-dependent runtime: If the classical algorithm uses time T on a given problem instance, the quantum algorithm uses time $O(\sqrt{T}\,\mathrm{poly}(n))$.

- This can be leveraged to obtain exponential reductions in expected runtime.

Exponentially reduced average runtime

The above algorithm has an instance-dependent runtime: If the classical algorithm uses time T on a given problem instance, the quantum algorithm uses time $O(\sqrt{T}\,\mathrm{poly}(n))$.

- This can be leveraged to obtain exponential reductions in expected runtime.
- We consider a setting where the input is picked from some distribution, and we are interested in the average runtime of the algorithm, over the input distribution.

Exponentially reduced average runtime

The above algorithm has an instance-dependent runtime: If the classical algorithm uses time T on a given problem instance, the quantum algorithm uses time $O(\sqrt{T}\,\mathrm{poly}(n))$.

- This can be leveraged to obtain exponential reductions in expected runtime.
- We consider a setting where the input is picked from some distribution, and we are interested in the average runtime of the algorithm, over the input distribution.

Claim

Pick a random 3-SAT instance on n variables by choosing m random clauses, where $\Pr[m = m'] \propto 2^{-Cn^{3/2}/\sqrt{m'}}$.

Then there exists a constant C such that the expected quantum runtime is $\mathrm{poly}(n)$, but a simple backtracking algorithm has expected runtime exponential in n.

From quadratic to exponential speedups?

For example:

- Let $T(X)$ denote the number of vertices in the backtracking tree on input X.

From quadratic to exponential speedups?

For example:

- Let $T(X)$ denote the number of vertices in the backtracking tree on input X.
- Assume $\Pr_X[T(X) = t] \leqslant Ct^{\beta}$ for all t and some C, β.

From quadratic to exponential speedups?

For example:

- Let $T(X)$ denote the number of vertices in the backtracking tree on input X.
- Assume $\Pr_X[T(X) = t] \leqslant Ct^{\beta}$ for all t and some C, β.
- Also assume $\Pr_X[T(X) = t] \geqslant Dt^{\beta}$, for some D, for M different values t, where $M = \exp(O(n))$.

From quadratic to exponential speedups?

For example:

- Let $T(X)$ denote the number of vertices in the backtracking tree on input X.
- Assume $\Pr_X[T(X) = t] \leqslant Ct^{\beta}$ for all t and some C, β.
- Also assume $\Pr_X[T(X) = t] \geqslant Dt^{\beta}$, for some D, for M different values t, where $M = \exp(O(n))$. Then

$$\mathbb{E}_X[T(X)] \geqslant \sum_{t=1}^{M} Dt^{\beta} \cdot t = \Omega(M^{\beta+2}).$$

From quadratic to exponential speedups?

For example:

- Let $T(X)$ denote the number of vertices in the backtracking tree on input X.
- Assume $\Pr_X[T(X) = t] \leqslant Ct^{\beta}$ for all t and some C, β.
- Also assume $\Pr_X[T(X) = t] \geqslant Dt^{\beta}$, for some D, for M different values t, where $M = \exp(O(n))$. Then

$$\mathbb{E}_X[T(X)] \geqslant \sum_{t=1}^{M} Dt^{\beta} \cdot t = \Omega(M^{\beta+2}).$$

- So for $\beta > -2$ the average classical complexity is large.

From quadratic to exponential speedups?

For example:

- Let $T(X)$ denote the number of vertices in the backtracking tree on input X.
- Assume $\Pr_X[T(X) = t] \leqslant Ct^{\beta}$ for all t and some C, β.
- Also assume $\Pr_X[T(X) = t] \geqslant Dt^{\beta}$, for some D, for M different values t, where $M = \exp(O(n))$. Then

$$\mathbb{E}_X[T(X)] \geqslant \sum_{t=1}^{M} Dt^{\beta} \cdot t = \Omega(M^{\beta+2}).$$

- So for $\beta > -2$ the average classical complexity is large.
- But, if $-2 < \beta < -3/2$, the average number of steps used by the quantum backtracking algorithm is

$$\mathbb{E}_X[O(\sqrt{T(X)}\,\mathrm{poly}(n))] \leqslant \sum_{t \geqslant 1} O(\sqrt{t} \cdot t^{\beta}\,\mathrm{poly}(n)) = \mathrm{poly}(n).$$

Proof: marked element case

Claim

Let x_0 be a marked element. Then

$$|\phi\rangle = \sqrt{n}|r\rangle + \sum_{x \neq r, x \rightsquigarrow x_0} (-1)^{\ell(x)} |x\rangle$$

is an eigenvector of $R_B R_A$ with eigenvalue 1, where $\ell(x)$ is the distance of x from the root.

Proof: marked element case

Claim

Let x_0 be a marked element. Then

$$|\phi\rangle = \sqrt{n}|r\rangle + \sum_{x \neq r, x \rightsquigarrow x_0} (-1)^{\ell(x)}|x\rangle$$

is an eigenvector of $R_B R_A$ with eigenvalue 1, where $\ell(x)$ is the distance of x from the root.

Proof:

- Each state $|\psi_x\rangle$ ($x \neq r$, $x \neq x_0$) has uniform support on either 0 or 2 vertices on the path from r to x_0.

Proof: marked element case

Claim

Let x_0 be a marked element. Then

$$|\phi\rangle = \sqrt{n}|r\rangle + \sum_{x \neq r, x \rightsquigarrow x_0} (-1)^{\ell(x)} |x\rangle$$

is an eigenvector of $R_B R_A$ with eigenvalue 1, where $\ell(x)$ is the distance of x from the root.

Proof:

- Each state $|\psi_x\rangle$ ($x \neq r$, $x \neq x_0$) has uniform support on either 0 or 2 vertices on the path from r to x_0.
- So, for all such states, $\langle\phi|\psi_x\rangle = 0$.

Proof: marked element case

Claim

Let x_0 be a marked element. Then

$$|\phi\rangle = \sqrt{n}|r\rangle + \sum_{x \neq r, x \rightsquigarrow x_0} (-1)^{\ell(x)}|x\rangle$$

is an eigenvector of $R_B R_A$ with eigenvalue 1, where $\ell(x)$ is the distance of x from the root.

Proof:

- Each state $|\psi_x\rangle$ ($x \neq r$, $x \neq x_0$) has uniform support on either 0 or 2 vertices on the path from r to x_0.
- So, for all such states, $\langle\phi|\psi_x\rangle = 0$.
- Also,

$$\frac{\langle r|\phi\rangle}{\||\phi\rangle\|} \geqslant \frac{1}{\sqrt{2}}.$$

Proof: no marked element case

Effective spectral gap lemma [Lee et al. '11]

Set $R_A = 2\Pi_A - I$, $R_B = 2\Pi_B - I$. Let P_χ be the projector onto the span of the eigenvectors of $R_B R_A$ with eigenvalues $e^{2i\theta}$ such that $|\theta| \leqslant \chi$. Then, for any $|\psi\rangle$ such that $\Pi_A|\psi\rangle = 0$, we have

$$\|P_\chi \Pi_B |\psi\rangle\| \leqslant \chi \||\psi\rangle\|.$$

Proof: no marked element case

Effective spectral gap lemma [Lee et al. '11]

Set $R_A = 2\Pi_A - I$, $R_B = 2\Pi_B - I$. Let P_χ be the projector onto the span of the eigenvectors of $R_B R_A$ with eigenvalues $e^{2i\theta}$ such that $|\theta| \leqslant \chi$. Then, for any $|\psi\rangle$ such that $\Pi_A|\psi\rangle = 0$, we have

$$\|P_\chi \Pi_B |\psi\rangle\| \leqslant \chi \| |\psi\rangle \|.$$

- Π_A, Π_B project onto the invariant subspaces of R_A and R_B.
- These spaces are spanned by vectors of the form $|\psi_x^\perp\rangle$ for $x \in A$, $x \in B$ respectively.

Proof: no marked element case

Effective spectral gap lemma [Lee et al. '11]

Set $R_A = 2\Pi_A - I$, $R_B = 2\Pi_B - I$. Let P_χ be the projector onto the span of the eigenvectors of $R_B R_A$ with eigenvalues $e^{2i\theta}$ such that $|\theta| \leqslant \chi$. Then, for any $|\psi\rangle$ such that $\Pi_A|\psi\rangle = 0$, we have

$$\|P_\chi \Pi_B |\psi\rangle\| \leqslant \chi \| |\psi\rangle \|.$$

- Π_A, Π_B project onto the invariant subspaces of R_A and R_B.
- These spaces are spanned by vectors of the form $|\psi_x^\perp\rangle$ for $x \in A$, $x \in B$ respectively.
- Here $|\psi_x^\perp\rangle$ is orthogonal to $|\psi_x\rangle$ and has support only on $\{|x\rangle\} \cup \{|y\rangle : x \to y\}$; in addition to $|r\rangle$ in the case of R_B.

Proof: no marked element case

Effective spectral gap lemma [Lee et al. '11]

Set $R_A = 2\Pi_A - I$, $R_B = 2\Pi_B - I$. Let P_χ be the projector onto the span of the eigenvectors of $R_B R_A$ with eigenvalues $e^{2i\theta}$ such that $|\theta| \leqslant \chi$. Then, for any $|\psi\rangle$ such that $\Pi_A|\psi\rangle = 0$, we have

$$\|P_\chi \Pi_B |\psi\rangle\| \leqslant \chi \| |\psi\rangle \|.$$

Consider the vector

$$|\eta\rangle = |r\rangle + \sqrt{n} \sum_{x \neq r} |x\rangle.$$

Proof: no marked element case

Effective spectral gap lemma [Lee et al. ’11]

Set $R_A = 2\Pi_A - I$, $R_B = 2\Pi_B - I$. Let P_χ be the projector onto the span of the eigenvectors of $R_B R_A$ with eigenvalues $e^{2i\theta}$ such that $|\theta| \leqslant \chi$. Then, for any $|\psi\rangle$ such that $\Pi_A|\psi\rangle = 0$, we have

$$\|P_\chi \Pi_B |\psi\rangle\| \leqslant \chi \| |\psi\rangle \|.$$

Consider the vector

$$|\eta\rangle = |r\rangle + \sqrt{n} \sum_{x \neq r} |x\rangle.$$

- On each subspace $\mathcal{H}_x$, $x \in A$, $|\eta\rangle \propto |\psi_x\rangle$, so $\Pi_A|\eta\rangle = 0$. Similarly $\Pi_B|\eta\rangle = |r\rangle$.

Proof: no marked element case

Effective spectral gap lemma [Lee et al. '11]

Set $R_A = 2\Pi_A - I$, $R_B = 2\Pi_B - I$. Let P_χ be the projector onto the span of the eigenvectors of $R_B R_A$ with eigenvalues $e^{2i\theta}$ such that $|\theta| \leqslant \chi$. Then, for any $|\psi\rangle$ such that $\Pi_A|\psi\rangle = 0$, we have

$$\|P_\chi \Pi_B |\psi\rangle\| \leqslant \chi \| |\psi\rangle \|.$$

Consider the vector

$$|\eta\rangle = |r\rangle + \sqrt{n} \sum_{x \neq r} |x\rangle.$$

- On each subspace $\mathcal{H}_x$, $x \in A$, $|\eta\rangle \propto |\psi_x\rangle$, so $\Pi_A|\eta\rangle = 0$. Similarly $\Pi_B|\eta\rangle = |r\rangle$.
- By the effective spectral gap lemma,

$$\|P_\chi |r\rangle\| = \|P_\chi \Pi_B |\eta\rangle\| \leqslant \chi \| |\eta\rangle \| \leqslant \chi \sqrt{Tn}.$$