THEORY OF COMPUTING, Volume 14 (15), 2018, pp. 1–24
www.theoryofcomputing.org

Quantum-Walk Speedup of Backtracking Algorithms

Ashley Montanaro*

Received December 2, 2016; Revised May 4, 2017; Published December 3, 2018

Abstract: We describe a general method to obtain quantum speedups of classical algorithms which are based on the technique of backtracking, a standard approach for solving constraint satisfaction problems (CSPs). Backtracking algorithms explore a tree whose vertices are partial solutions to a CSP in an attempt to find a complete solution. Assume there is a classical backtracking algorithm which finds a solution to a CSP on n variables, or outputs that none exists, and whose corresponding tree contains T vertices, each vertex corresponding to a test of a partial solution. Then we show that there is a bounded-error quantum algorithm which completes the same task using $O(\sqrt{T}n^{3/2}\log n)$ tests. In particular, this quantum algorithm can be used to speed up the DPLL algorithm, which is the basis of many of the most efficient SAT solvers used in practice. The quantum algorithm is based on the use of a quantum walk algorithm of Belovs to search in the backtracking tree.

1 Introduction

Grover's quantum search algorithm [33] is one of the great success stories of quantum computation. One important domain to which the algorithm can be applied is the solution of constraint satisfaction problems (CSPs). Consider a constraint satisfaction problem (CSP) expressed as a predicate $P : [d]^n \to \{\text{true}, \text{false}\}$,

*Supported by EPSRC Early Career Fellowship EP/L021005/1.

ACM Classification: F.1.2, G.1.6

AMS Classification: 81P68, 68Q25

Key words and phrases: quantum computing, quantum query complexity, quantum walk

DOI: 10.4086/toc.2018.v014a015

where $[d] = \{0, \dots, d-1\}$. We would like to find an assignment x to the n variables such that $P(x)$ is true, or output "not found" if no such x exists. This framework encompasses many important problems such as Boolean satisfiability and graph colouring. Grover's algorithm solves such a CSP using $O(\sqrt{d^n})$ evaluations of P, whereas with no further information about P, finding an x such that $P(x)$ is true requires $\Omega(d^n)$ evaluations classically in the worst case. However, when we are faced with an instance of a CSP in practice, we usually have some additional information about its structure. For example, P may be defined as the conjunction of smaller constraints of a particular type, as in the case of graph colouring. This information often allows classical algorithms to solve the CSP significantly more efficiently than the above bound would suggest, throwing some doubt on whether straightforward use of Grover's algorithm will really be used to solve CSPs in practice.

One of the most important and most general classical tools to take advantage of problem structure, both in theory and in practice, is backtracking [9]. This technique can be used when we have the ability to recognise whether partial solutions to a problem can be extended to full solutions. We assume that the predicate P allows us to pass it a partial assignment x of the form $x : S \to [d]$, where $S \subseteq \{1, \dots, n\}$, which specifies the values assigned to the variables in the set S. We can equivalently think of x as an element of $\mathcal{D} := ([d] \cup \{*\})^n$, where the $*$'s represent the positions which are as yet unassigned values. We say that x is *complete* if it contains no $*$'s. Then P returns:

- "true" if x is a solution to the CSP (that is, for any complete assignment consistent with x, that assignment is a valid solution);
- "false" if it is clear that x cannot be extended to a solution to the CSP (that is, no complete assignment consistent with x is a valid solution);
- "indeterminate" otherwise.

We say that a partial assignment x is valid if $P(x)$ is true or indeterminate, and invalid if $P(x)$ is false.

Algorithm 1 below describes a generic way to use this information classically. The algorithm assumes access to P and a heuristic $h(x)$ which determines how to extend a given partial assignment x, by selecting the next variable to assign a value. One simple example of a heuristic would be to order the variables arbitrarily, and then return the lowest index of a variable not yet assigned a value. However, one could also consider more complicated heuristics based on other properties of x. We think of P and h as black boxes ("oracles"). The basic idea behind Algorithm 1 is to fail early: if we know that a partial assignment cannot be extended to a solution, we should give up on it and try a different one. We can think of the algorithm as exploring a tree, whose internal vertices are partial solutions to P, and whose leaves are solutions to P or certificates that the partial solution cannot be extended to a complete solution. This tree is of size at most $O(d^n)$, but for some problem instances could be substantially smaller.

A canonical example of a powerful backtracking algorithm which fits into the framework of Algorithm 1 is the DPLL (Davis-Putnam-Logemann-Loveland) algorithm [23, 22] for k-SAT. This algorithm forms the basis of many of the most successful SAT solvers used in practice [25, 40, 32]. For many practically relevant problem instances, the algorithm runs more quickly than worst-case upper bounds would suggest. Another appealing aspect of this algorithm is that, unlike "local search" methods based on random walks or similar ideas, it can sometimes produce efficient proofs of unsatisfiability, corresponding to small backtracking trees.

```
Assume that we are given access to a predicate $P : \mathcal{D} \to \{\text{true}, \text{false}, \text{indeterminate}\}$, and a heuristic
$h : \mathcal{D} \to \{1, \dots, n\}$ which returns the next index to branch on from a given partial assignment.
Return bt($*^n$), where bt is the following recursive procedure:
bt($x$):
  1. If $P(x)$ is true, output $x$ and return.
  2. If $P(x)$ is false, or $x$ is a complete assignment, return.
  3. Set $j = h(x)$.
  4. For each $w \in [d]$:
      (a) Set $y$ to $x$ with the $j$'th entry replaced with $w$.
      (b) Call bt($y$).
```

Algorithm 1: General classical backtracking algorithm.

We now sketch how one version of DPLL can be understood as an instance of Algorithm 1. Assume we have a k-SAT formula ϕ on n variables, and we want to list its satisfying assignments. The predicate P behaves as follows, given a partial assignment x as input. Substitute the assigned values in x into ϕ. If ϕ then evaluates to true or false, return that value. Otherwise, for each clause in ϕ that contains only one variable, assign a value to that variable such that ϕ does not evaluate to false. Next, for each variable that only occurs negated or unnegated in ϕ, assign false or true (respectively) to that variable. Repeat this procedure until ϕ no longer changes. At the end, if ϕ is neither true or false, return indeterminate. One simple heuristic h that can be used is to select the first variable still present in the simplified formula. However, a variety of more complicated heuristics h can also be used, and the choice of heuristic can substantially affect the search time in practice; see [44] for a discussion and experimental results.

Note that Algorithm 1 outputs all solutions x such that $P(x)$ is true, though in practice the algorithm might be modified to terminate when the first solution is found. We assume that P and h can both be evaluated in time $\text{poly}(n)$, so the most important contribution to the complexity of Algorithm 1 is usually the number of vertices in the tree, which can often be exponential in n. To simplify the complexity bounds, we also assume throughout that $d = O(1)$; this is effectively without loss of generality as any predicate with local domain size d can be replaced with one which uses $O(\log d)$ bits to encode each variable.

1.1 Results

We show here that there is a quantum equivalent of Algorithm 1 which can be substantially faster.

Theorem 1.1. *Let T be an upper bound on the number of vertices in the tree explored by Algorithm 1. Then for any $0 < \delta < 1$ there is a quantum algorithm which, given T, evaluates P and h $O(\sqrt{Tn}\log(1/\delta))$ times each, outputs true if there exists x such that $P(x)$ is true, and outputs false otherwise. The algorithm uses* $\text{poly}(n)$ *space, $O(1)$ auxiliary operations per use of P and h, and fails with probability at most δ.*

We usually think of T as being exponential in n; in this regime this complexity is a near-quadratic speedup over the classical algorithm, assuming that the classical algorithm explores the whole tree. In a black-box setting, a query complexity lower bound on $\Omega(\sqrt{Tn})$ queries to P and h follows from a bound of Aaronson and Ambainis [1] on the complexity of local search in graphs (see Section 4 for a discussion), so the complexity bound of Theorem 1.1 is optimal for $\delta = \Omega(1)$. The algorithm can be modified to find a solution, rather than just detect the existence of one, with a small penalty in the running time.

Theorem 1.2. *Let T be the number of vertices in the tree explored by Algorithm 1. Then for any $0 < \delta < 1$ there is a quantum algorithm which makes $O(\sqrt{T}n^{3/2}\log n \log(1/\delta))$ evaluations of each of P and h, 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, there is a quantum algorithm which outputs x_0 making $O(\sqrt{Tn}\log^3 n \log(1/\delta))$ evaluations of each of P and h. In both cases the algorithm uses* poly(n) *space, $O(1)$ auxiliary operations per use of P and h, and fails with probability at most δ.*

We stress that these results can be applied to any backtracking algorithm which fits into the framework of Algorithm 1, whatever the predicate P or the choice of the heuristic h. In particular, they can be applied to the DPLL algorithm as discussed above.

Theorems 1.1 and 1.2 can also be applied to backtracking algorithms which make use of randomness in the heuristic h, by interpreting these algorithms as first fixing a random seed, then using this seed as input to a deterministic heuristic h. Note that in this case, depending on the model one assumes, the algorithm could be query-efficient but not time-efficient. This is because writing down the random seed itself would require time $\Theta(T)$. In practice, "randomised" heuristics are often based on the use of efficient pseudorandom number generators. In this case we would effectively have a deterministic heuristic, and would hence retain time-efficiency.

Observe that the bound on the running time in Theorem 1.2 is instance-dependent and, to use it, we do not need to know an upper bound on the running time T of the underlying classical backtracking algorithm. For instances on which the classical algorithm runs quickly, the quantum algorithm also runs quickly. However, also observe that the running time of the quantum algorithm for detecting the existence of a solution scales like $O(\sqrt{Tn})$, regardless of how many solutions there are and where they are located. If the classical algorithm is asked only to determine the existence of a solution, rather than to find all of them, its running time might be substantially less than $O(T)$. In particular, if there are substantially more than $\sqrt{T}$ solutions and they are uniformly distributed throughout the tree, the classical running time required to find a solution may be less than the quantum running time; another case where this can happen is if the classical algorithm is lucky and finds a solution "early on" in the tree. These limitations have been removed by subsequent work of Ambainis and Kokainis [7], who have described a quantum algorithm which finds a solution in time $\widetilde{O}(\sqrt{T'}n^{3/2})$, where T' is the number of vertices actually visited by the classical algorithm.

The quantum algorithms can be leveraged to obtain an *exponential* separation between average quantum and classical running times. The speedup for any given instance is approximately quadratic. However, given the right distribution on the input instances, this can be amplified to an exponential average running time separation. This phenomenon was first noted in the context of quantum query complexity by Ambainis and de Wolf [6], who gave several examples of superpolynomial separations in expected running time for the computation of total functions. While a striking effect, it is not clear whether this has any meaningful consequences, given that the speedup on each instance is just polynomial. We therefore defer a more detailed discussion to the arXiv version of this work [45].

1.2 Techniques

The algorithms which achieve the bounds of Theorems 1.1 and 1.2 are based on the use of a discrete-time quantum walk to find a marked vertex within the tree produced by the classical backtracking algorithm, corresponding to a partial solution x such that $P(x)$ is true. Quantum walks have become a basic tool in quantum algorithm design [19, 4, 53, 42]. In particular, they have been applied in several contexts to solve search problems on graphs [51, 53, 42, 38], sometimes achieving up to a quadratic speedup over classical algorithms. However, in prior work it is usually assumed that the input graph is known in advance, and moreover that the initial state of the quantum walk is the stationary distribution of the corresponding random walk. Aaronson and Ambainis [1] described a different approach to spatial search on graphs; this does not use a quantum walk, but also assumes the input graph is known in advance.

Here we would like to use quantum walks in a context where the input graph is defined implicitly by the backtracking algorithm and hence is not known in advance, and where the walk starts at the root of the tree. One of the few cases where such walks have been studied is beautiful work of Belovs [10, 11]. The main result of that work relates the complexity of detecting a marked vertex by quantum walk on a graph to the *effective resistance* of the graph. Informally, this quantity is determined by thinking of the graph as an electrical circuit and calculating the resistance between the initial vertex and the set of marked vertices. Belovs' result can be seen as a quantum variant of previous classical work characterising properties of random walks on graphs (such as the commute time and cover time) in terms of effective resistance [17].

The main quantum subroutine used here is just the special case of Belovs' result where the underlying graph is a tree, for which we include a slightly more concise correctness proof. We are also able to extend Belovs' work to give an algorithm for finding a marked vertex in a tree, rather than just detecting one. This can easily be achieved using binary search; in the case where there is promised to be a unique marked element, we give a more efficient algorithm based on analysing eigenvectors of the quantum walk operator.

Once we have the quantum search algorithm, all that remains is to check the claim that the P and h functions can indeed be used to implement the required quantum walk operations, namely mixing across the neighbours of a vertex in the tree, dependent on whether the vertex is marked. To do this one has to be careful to ensure that the quantum walk steps are implemented efficiently.

1.3 Other prior work

This paper is connected to prior work in a number of different areas: classical and quantum algorithms for backtracking, other quantum techniques for solving CSPs, and quantum walk algorithms. Backtracking is a fundamental technique in computer science and has been studied since at least the 1960s. The classical literature on this topic is too vast to summarise here; see [37, 29, 9] for introductions to the topic and historical overviews. We now discuss relevant prior results within the field of quantum computation.

First, Cerf, Grover and Williams attempted to find a direct quantum speedup of backtracking algorithms [16]. The algorithm of [16] is based on a nested version of Grover search. The complete tree of partial assignments is expanded to a certain depth, then quantum search is performed within the subset of partial assignments which have not yet been ruled out. The complexity of the algorithm depends on the number of valid partial assignments at this depth. It is argued in [16] that, for some reasonable distributions on random CSPs, the average complexity of the quantum algorithm (over the distribution on instances) will be smaller than would be obtained from Grover search. By contrast, the bounds of Theorems 1.1 and 1.2 hold in the worst case and are applicable to arbitrary backtracking algorithms: if a faster backtracking algorithm is found, we immediately obtain a faster quantum algorithm.

The algorithm used here can be seen as an extreme version of the nested search strategy of [16]. The diffusion operation used in the quantum walk can be viewed as applying Grover search within a subspace spanned by a vertex in the tree and its children. The algorithm repeatedly performs these searches across many vertices and levels simultaneously. On the other hand, the algorithm of [16] can be seen as accelerating a restricted classical backtracking algorithm which uses a predicate P which is only capable of detecting whether partial assignments at a particular level are false.

Similarly to the present work, Farhi and Gutmann [28] have studied the use of quantum walks to speed up classical backtracking algorithms by searching within the backtracking tree.[1] These authors showed that there are some trees for which continuous-time quantum walks can be used to find a marked vertex exponentially faster than a classical random walk. The special structure of these trees leads to interference effects which enable the quantum walk to penetrate the tree more quickly than the random walk. However, for the examples presented in [28] where there is an exponential speedup of this form, the structure of the tree enables an alternative classical algorithm to also find a marked vertex efficiently. Here, we seek to accelerate classical search in arbitrary trees, with no prior assumptions about the structure of the tree.

A related, but different, approach towards quantum speedup of recursive classical algorithms was proposed by Fürer [30]. Imagine we have a constraint satisfaction problem for which we can put a non-trivial upper bound L on the number of leaves in the computation tree of a recursive classical algorithm for solving the problem. The idea of [30] was to apply Grover search over the leaves of the computation tree to find a solution in time $O(\sqrt{L}\operatorname{poly}(n))$. This approach relies on knowing, in advance, an efficiently computable mapping associating each integer between 1 and L with a leaf. For many more complicated recursive algorithms we may not know such a mapping. Indeed, it is unlikely to be possible to compute such a mapping for general backtracking algorithms in polynomial time. Counting the number of vertices in a backtracking tree is known to be #P-complete [52], but the ability to determine the identity of the ℓ'th leaf, for arbitrary ℓ, would allow the number of vertices in the tree to be calculated efficiently via an exponential doubling procedure. The quantum algorithm presented here, on the other hand, can be applied to any classical backtracking algorithm, even if we do not know a bound on L in advance.

A somewhat similar idea to Fürer's was previously used by Angelsmark, Dahllöf and Jonsson [8] to obtain quantum speedups for CSPs. These authors observed that, for certain CSPs, one can construct a set of d^{cn} easily checked certificates, for some $c < 1$, such that the existence of a solution to the CSP is certified by at least one certificate. Then Grover search can be used to find a certificate, if one exists, in time $O(d^{cn/2}\operatorname{poly}(n))$.

An alternative, and simpler, approach to find quantum speedups of classical algorithms for CSPs is the use of amplitude amplification [14]. This can be applied to any classical algorithm which can be

[1]Their paper uses somewhat different terminology, e. g., "decision tree" rather than "backtracking tree", but the basic setting is the same.

expressed as repeatedly running a randomised subroutine which runs in time $\mathrm{poly}(n)$ and finds a solution with probability p. The corresponding quantum algorithm has a running time of $O((1/\sqrt{p})\,\mathrm{poly}(n))$, a near-quadratic improvement on the classical $O((1/p)\,\mathrm{poly}(n))$ if p is small. For example, it was observed by Ambainis [3] that Schöning's efficient randomised algorithm for k-SAT [50] can be accelerated in this way; Dantsin, Kreinovich and Wolpert [21] gave several other examples. Deterministic backtracking algorithms are, of course, not amenable to this approach.

Finally, a completely different technique for solving CSPs is the quantum adiabatic algorithm [27]. Although there is some numerical evidence that this algorithm may outperform classical algorithms for CSPs [26], the adiabatic algorithm's running time is hard to analyse for large input sizes and there is as yet no analytical proof of its superiority over classical algorithms.

Quantum walks on trees have been used previously in a quite distinct context, to obtain a near-quadratic speedup for evaluation of AND-OR formulae [5]. In that algorithm the structure of the formula (which is known in advance) defines the tree on which the walk takes place. It is interesting to note that the quantum walk used in [5] is similar to the quantum walk used here, but has apparently quite different properties. In that work, an eigenvalue of the quantum walk operator determines whether an AND-OR formula evaluates to 1, whereas here it determines whether the tree contains a marked vertex. Another case where the concept of effective resistance was used in quantum computing is work by Wang, which gave an efficient quantum algorithm for approximating effective resistances [55]. This uses some similar ideas to the present work but does not seem directly applicable. Subsequent work by Jeffery and Kimmel [35] has used effective resistance to relate the quantum query complexity of evaluating a NAND tree to a complexity measure of the tree.

1.4 Subsequent applications

Following the completion of an initial version of this work, several papers have applied the quantum algorithm given here to speed up classical backtracking algorithms.

Alkim et al. [2] and del Pino, Lyubashevsky and Pointcheval [24] suggested that the quantum backtracking algorithm could be used to obtain a quadratic speedup for solving the shortest vector problem in lattices, by accelerating classical lattice-enumeration algorithms based on the BKZ algorithm of Schnorr and Euchner [49, 31, 18]. The enumeration subroutine of the BKZ algorithm as usually described uses information from vertices previously visited in the backtracking tree (such as the length of the shortest vector found so far) to prune the tree, so it does not quite fit the requirements of the quantum backtracking algorithm. However, efficient variants of the algorithm are known which do not use this type of optimisation (e. g., [31, Algorithm 2]), and these should allow a quantum speedup.

Mandrà, Guerreschi and Aspuru-Guzik [43] combined the quantum backtracking algorithm with a novel reduction to obtain quantum speedups for solving exact satisfiability problems. Finally, Moylett, Linden and the author [46] gave near-quadratic quantum speedups over the most efficient classical algorithms known for solving the Travelling Salesman Problem on bounded-degree graphs. The result of [46] is based on applying the quantum backtracking algorithm to accelerate classical algorithms of Xiao and Nagamochi [56, 57].

As discussed in Section 1.1, subsequent work of Ambainis and Kokainis [7] has removed the limitation that the running time of the quantum backtracking algorithm depends on the size of the entire tree. Their algorithm finds a solution in time $\widetilde{O}(\sqrt{T'}n^{3/2})$, where T' is the number of vertices actually visited by the classical algorithm.

1.5 Organisation

We begin in Section 2 by describing the main underlying quantum ingredient, the use of a quantum walk to detect a marked vertex in a tree. This algorithm is a special case of an algorithm described by Belovs [10]. We then go on in Sections 2.3 and 2.4 to describe extensions to this algorithm to allow finding a marked vertex, and a faster running time in the case where we know there is a unique marked vertex. Section 3 shows that the algorithm can be applied to accelerate backtracking algorithms for CSPs. Section 4 concludes with a discussion of some ways in which the algorithm could be improved, and barriers to doing so.

2 Quantum walks on trees

2.1 Preliminaries

We will need the following tools, which have been used many times elsewhere in quantum algorithm design.

Lemma 2.1 (Effective spectral gap lemma [39]). *Let Π_A and Π_B be projectors on the same Hilbert space, and 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 $\mathrm{e}^{2i\theta}$ such that $|\theta| \le \chi$. Then, for any vector $|\psi\rangle$ such that $\Pi_A|\psi\rangle = 0$, we have*

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

Theorem 2.2 (Phase estimation [20, 36]). *For every integer $s \ge 1$, and every unitary U on m qubits, there exists a uniformly generated quantum circuit C that acts on $m+s$ qubits and has the following properties.*

1. *C uses the controlled-U operator $O(2^s)$ times, and contains $O(s^2)$ other gates.*
2. *For every eigenvector $|\psi\rangle$ of U with eigenvalue 1, $C|\psi\rangle|0^s\rangle = |\psi\rangle|0^s\rangle$.*
3. *If $U|\psi\rangle = \mathrm{e}^{2i\theta}|\psi\rangle$, where $\theta \in (0,\pi)$, then $C|\psi\rangle|0^s\rangle = |\psi\rangle|\omega\rangle$, where $|\omega\rangle$ satisfies*
$$|\langle \omega | 0^s \rangle|^2 = \sin^2(2^s\theta)/(2^{2s}\sin^2\theta) .$$
4. *For any $|\phi\rangle \in (\mathbb{C}^2)^{\otimes m}$, expanded as $|\phi\rangle = \sum_k \lambda_k |\psi_k\rangle$, where $|\psi_k\rangle$ is an eigenvector of U with eigenvalue $\mathrm{e}^{2i\theta_k}$, then*
$$C|\phi\rangle|0^s\rangle = \sum_k \lambda_k |\psi_k\rangle|\omega_k\rangle ,$$
where $\sum_{k:\theta_k \ge \varepsilon} |\langle \omega_k | 0^s\rangle|^2 = O(1/(2^s\varepsilon))$ for any $\varepsilon > 0$.

Call 2^{-s} the precision of the circuit.

Phase estimation is normally used to estimate eigenvalues of U (hence its name); here, however, similarly to [42] we will only need to apply it to distinguish the eigenvalue 1 from other eigenvalues. If the smallest nonzero phase is ε, this can be done with $O(1/\varepsilon)$ uses of controlled-U.

Fact 2.3 (Close states and measurement outcomes, see, e. g., [13])**.** *Let* $|\psi_1\rangle$, $|\psi_2\rangle$ *be quantum states satisfying* $\||\psi_1\rangle - |\psi_2\rangle\| = \varepsilon$. *Then the total variation distance between the two distributions on measurement outcomes obtained by measuring each state in the computational basis is at most* ε.

(This fact is usually presented with ε replaced with 4ε [13]; the tighter constant stated here can easily be obtained by relating the fidelity of $|\psi_1\rangle$ and $|\psi_2\rangle$ to their trace distance, for example.)

2.2 The quantum walk algorithm

We now describe a quantum algorithm for detecting a marked vertex in a tree. The algorithm is a special case of a beautiful connection between quantum walks and electrical circuits due to Belovs [10] (see also [11]), which is a quantum analogue of a similar connection between random walks and electrical circuits [17]. This is conceptually elegant and leads to a very concise proof of a previous result of Szegedy [53] on detecting marked elements using a quantum walk. Here we only use these ideas for the special case of trees and a quantum walk starting at the root. This will enable us to simplify some notation and, hopefully, make the algorithm more intuitive.

Consider a rooted tree with T vertices, labelled $r, 1, \dots, T-1$, with vertex r being the root, where the distance from the root to any leaf is at most n. Assume for simplicity in what follows that the root is promised not to be marked. For each vertex x, let $\ell(x)$ be the distance of x from the root. We assume throughout that, although we do not necessarily know the structure of the tree in advance, we can determine $\ell(x)$ for any x. Let A be the set of vertices an even distance from the root (including the root itself), and let B be the set of vertices at an odd distance from the root. We write $x \to y$ to mean that y is a child of x in the tree. For each x, let d_x be the degree of x as a vertex in an undirected graph. Thus, for all $x \neq r$, $d_x = |\{y : x \to y\}| + 1$; and $d_r = |\{y : r \to y\}|$.

The quantum walk operates on the Hilbert space $\mathcal{H}$ spanned by $\{|r\rangle\} \cup \{|x\rangle : x \in \{1, \dots, T-1\}\}$, and starts in the state $|r\rangle$. Unlike many discrete-time quantum walk algorithms, it does not use a separate "coin" space. The walk is based on a set of diffusion operators D_x, where D_x acts on the subspace $\mathcal{H}_x$ spanned by $\{|x\rangle\} \cup \{|y\rangle : x \to y\}$. The diffusion operators are defined as follows.

- 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
$$|\psi_x\rangle = \frac{1}{\sqrt{d_x}}\left(|x\rangle + \sum_{y, x \to y} |y\rangle\right).$$
  Note that if x is a leaf vertex, then $|\psi_x\rangle = |x\rangle$.
- $D_r = I - 2|\psi_r\rangle\langle\psi_r|$, where
$$|\psi_r\rangle = \frac{1}{\sqrt{1 + d_r n}}\left(|r\rangle + \sqrt{n} \sum_{y, r \to y} |y\rangle\right).$$

```
Input: Operators $R_A$, $R_B$, a failure probability $\delta$, upper bounds on the depth $n$ and the number of
vertices $T$. Let $\beta, \gamma > 0$ be universal constants to be determined.

  1. Repeat the following subroutine $K = \lceil \gamma \log(1/\delta) \rceil$ times:
       (a) Apply phase estimation to the operator $R_B R_A$ on $|r\rangle$ with precision $\beta/\sqrt{Tn}$.
       (b) If the eigenvalue is 1, accept; otherwise, reject.
  2. If the number of acceptances is at least $3K/8$, return "marked vertex exists"; otherwise, return
     "no marked vertex."
```

Algorithm 2: Detecting a marked vertex.

Observe that D_x can be implemented with only local knowledge, i. e., based only on whether x is marked and the neighbourhood structure of x. A step of the walk consists of applying the operator $R_B R_A$, where

$$R_A = \bigoplus_{x \in A} D_x \qquad \text{and} \qquad R_B = |r\rangle\langle r| + \bigoplus_{x \in B} D_x .$$

An alternative way of viewing this process is as a quantum walk on the graph given by the *edges* of the tree, where we identify each vertex with the edge from its parent in the tree, and add an additional "input" edge into the root. Also note that the quantum walk is similar to the "staggered" quantum walks considered in [48].

The algorithm for detecting a marked vertex is presented as Algorithm 2.

The next lemma is a special case of a result of Belovs [10, Theorem 4].

Lemma 2.4 (Belovs). *Algorithm 2 makes $O(\sqrt{Tn}\log(1/\delta))$ uses of each of R_A and R_B. There exist universal constants β, γ such that it fails with probability at most δ.*

Proof. The complexity bound is immediate from Theorem 2.2. For the correctness proof, we first show that, if there is a marked vertex, then $|r\rangle$ is quite close to (a normalised version of) an eigenvector $|\phi\rangle$ of $R_B R_A$ with eigenvalue 1. Let x_0 be a marked vertex and set

$$|\phi\rangle = \sqrt{n}|r\rangle + \sum_{x \neq r, x \rightsquigarrow x_0} (-1)^{\ell(x)} |x\rangle . \tag{2.1}$$

Here $x \rightsquigarrow x_0$ denotes the vertices x on the unique path from the root to x_0, including x_0 itself. To see that $|\phi\rangle$ is invariant under $R_B R_A$, first note that $|\phi\rangle$ is orthogonal to all states $|\psi_x\rangle$, where $x \neq r$ and x is not marked. Indeed, any such state $|\psi_x\rangle$ either has uniform support on exactly 2 consecutive vertices v in the path from r to x_0, or is not supported on any vertices in this path. $|\phi\rangle$ is also orthogonal to $|\psi_r\rangle$ by direct calculation. We have

$$\| |\phi\rangle \|^2 = n + \ell(x_0) \leq 2n .$$

Thus

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

Therefore, phase estimation returns the eigenvalue 1 with probability at least $1/2$. On the other hand, if there are no marked vertices, we consider the vector

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

Let Π_A and Π_B be projectors onto the invariant subspaces of R_A and R_B, i. e., $\Pi_A = (I+R_A)/2$, $\Pi_B = (I+R_B)/2$. These spaces are spanned by vectors of the form $|\psi_x^\perp\rangle$ for $x \in A$, $x \in B$ respectively, where $|\psi_x^\perp\rangle$ 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. On each subspace $\mathcal{H}_x$, $x \in A$, $|\eta\rangle$ is proportional to $|\psi_x\rangle$, so $\Pi_A|\eta\rangle = 0$. Similarly $\Pi_B|\eta\rangle = |r\rangle$. Recall that P_χ is the projector onto the span of the eigenvectors of R_BR_A with eigenvalues $\mathrm{e}^{2i\theta}$ such that $|\theta| \leq \chi$. By the effective spectral gap lemma (Lemma 2.1), $\|P_\chi|r\rangle\| = \|P_\chi\Pi_B|\eta\rangle\| \leq \chi\||\eta\rangle\| \leq \chi\sqrt{Tn}$. For small enough $\chi = \Omega(1/\sqrt{Tn})$, this is upper-bounded by $1/2$. By Theorem 2.2, there exists β such that applying phase estimation to R_BR_A with precision $\beta/\sqrt{Tn}$ returns the eigenvalue 1 with probability at most $1/4$.

Using a Chernoff bound, there exists γ such that, by repeating the subroutine $\lceil\gamma\log(1/\delta)\rceil$ times and returning "marked vertex exists" if the fraction of acceptances is greater than $3/8$, and "no marked vertex" otherwise, we obtain that the overall algorithm fails with probability at most δ. □

2.3 Finding a marked vertex

From now on, we assume that the degree of every vertex in the tree is $O(1)$; this is not a significant restriction for the application to backtracking. For trees obeying this restriction we can use the detection algorithm as a subroutine to *find* a marked vertex efficiently, via binary search.

To find a marked vertex, we start by applying Algorithm 2 to the entire tree. If it outputs "marked vertex exists," we apply the algorithm to the subtrees rooted at each child of the root in turn, to detect marked vertices within each subtree. Assuming the algorithm did not fail at any point, there must be a marked vertex in at least one subtree. We pick the root of one such subtree and check whether it is marked. If it is marked, we output its label and terminate; if it is not marked, we apply Algorithm 2 to each of its children and repeat. This process continues until we have found a marked vertex. As there are at most $O(n)$ repetitions to reach a leaf and $O(1)$ subtrees are checked at each repetition, the time complexity of the algorithm is multiplied by a factor of $O(n)$. Note that, when we apply the algorithm to subtrees, we must leave the parameter T unchanged; this is because the tree could be quite unbalanced, and a given subtree could contain many vertices.

We have thus far assumed that we know an upper bound on T in advance. If we do not, we can repeat the whole search algorithm $O(\log T) = O(n)$ times, doubling a guess for T each time (starting with $T = 1$) until we either find a marked vertex, or the algorithm returns "no marked vertex." This exponential doubling does not affect the asymptotic running time. If our guess for T is too low, the correctness proof of Algorithm 2 no longer holds, so the detection algorithm may claim that there is a marked vertex in a situation where there is actually no marked vertex. This may lead to the above binary search procedure returning an incorrect result. But we can deal with this situation by checking the final vertex returned by the search algorithm, and only terminating if it is marked; if it is not, we know that the search has failed, and continue doubling our guess for T. On the other hand, one can see from inspecting the proof of Lemma 2.4 that, if there is a marked vertex, the phase estimation subroutine in Algorithm 2 will accept with probability at least $1/2$ whether or not our guess for T is large enough. Therefore, if there is a marked vertex, Algorithm 2 will output that a marked vertex exists with probability at least $1-\delta$, for δ of our choice.

Using this procedure the total number of uses of Algorithm 2 (with differing values of T) is $O(n^2)$, so in order for the whole algorithm to succeed with probability, say, $2/3$, it is sufficient to reduce the failure probability of each use of Algorithm 2 to $O(1/n^2)$. This costs an additional time factor of $O(\log n)$ per use of the algorithm, giving a total running time of $O(\sqrt{T}n^{3/2}\log n)$. This can in turn be improved to an arbitrary failure probability $\delta > 0$ by taking $O(\log 1/\delta)$ repetitions, leading to an overall bound of time $O(\sqrt{T}n^{3/2}\log n \log(1/\delta))$.

Finally, we can find *all* marked vertices by simply repeating the algorithm, modifying the underlying oracle operator to strike out previously seen marked elements. If there are k marked elements, the overall running time is $O(k\sqrt{T}n^{3/2}\log n \log(k/\delta))$.

2.4 Search with a unique marked element

If we are promised that there exists a unique marked element in the tree, we can improve the above bounds by a factor of almost n. In general this improvement is not particularly large, as we usually have $T \gg n$; however, for some "tall and thin" trees it can be relatively significant. In particular, following this improvement we see that the complexity of the quantum algorithm for the search problem is never worse than the classical tree size $O(T)$, up to logarithmic factors.

We assume that there is a unique marked vertex x_0 and that $\ell(x_0) = n$. This second assumption is without loss of generality. We can determine $\ell(x_0)$ at the start of the algorithm by applying Algorithm 2 to the subtree rooted at r and of depth i, for differing values of i. That is, we only expand the tree up to depth i, and use binary search on $i \in \{1,\dots,n\}$ to find the minimal i such that the tree of depth i contains x_0. This needs $O(\log n)$ repetitions, so the complexity of this part is $O(\sqrt{Tn}\log n \log\log n)$, where the log log term comes from reducing the failure probability of Algorithm 2 to $O(1/(\log n))$. Once $\ell(x_0)$ is determined, we henceforth only search within the tree of depth $\ell(x_0)$.

Let $|\phi'\rangle = |\phi\rangle/\||\phi\rangle\|$, where the eigenvector $|\phi\rangle$ is defined in (2.1), i. e.,

$$|\phi'\rangle = \frac{1}{\sqrt{2}}|r\rangle + \frac{1}{\sqrt{2n}} \sum_{x \neq r, x \rightsquigarrow x_0} (-1)^{\ell(x)} |x\rangle \,. \tag{2.2}$$

The starting point for the search algorithm is the observation[2] that $|\phi'\rangle$ encodes the entire path from r to x_0. If we measure $|\phi'\rangle$, and do not receive outcome r, we receive a measurement outcome y which is uniformly distributed on the path from r to x_0. We can then repeat the algorithm on the subtree rooted at y, obtaining a new state of the form of $|\phi'\rangle$ for a smaller value of n. The expected number of measurements we would need to make to find x_0 is logarithmic in n (rather than the bound of $O(n)$ which follows from the previous binary search algorithm).

We first bound the total number of quantum walk steps used to find x_0, given access to states of the form of $|\phi'\rangle$ for various subtrees. Let C be chosen such that there is an algorithm which produces $|\phi'\rangle$ for an arbitrary subtree of the overall tree using $C\sqrt{Tn}$ steps (i. e., an algorithm which produces a state of the form (2.2) for different choices of r). Given that $\ell(x_0) = n$, measuring a copy of $|\phi'\rangle$ will give a

[2]A similar observation was used in [55] to approximate effective resistances.

"good" outcome (which is not r) with probability $1/2$. The distance from the root of such an outcome is uniformly distributed. Considering only the good outcomes, the expected total number of steps S_n to find x_0, given that $\ell(x_0) = n$, therefore satisfies

$$S_n \leq \frac{1}{n}\sum_{i=0}^{n-1} S_i + C\sqrt{Tn}\,.$$

We claim that $S_n \leq 4C\sqrt{Tn}$. The proof is by induction. First, $S_0 = 0$ as no quantum walk steps are made. Assume $S_i \leq 4C\sqrt{Ti}$ for all $i < n$. Then

$$S_n \leq \frac{4C}{n}\sum_{i=0}^{n-1}\sqrt{Ti} + C\sqrt{Tn} \leq 4C\sqrt{\frac{1}{n}\sum_{i=0}^{n-1} Ti} + C\sqrt{Tn} = \frac{4C}{\sqrt{2}}\sqrt{T}\sqrt{n-1} + C\sqrt{Tn} \leq 4C\sqrt{Tn}\,,$$

where the second inequality is Jensen's inequality. As on average half the outcomes are good, the expected total number of steps is thus at most $8C\sqrt{Tn}$.

We can approximately produce $|\phi'\rangle$ by applying phase estimation to the operator $R_B R_A$, with input state $|r\rangle$. If we write

$$|r\rangle = \frac{1}{\sqrt{2}}|\phi'\rangle + \frac{1}{\sqrt{2}}|\phi^{\perp}\rangle\,,$$

where $|\phi^{\perp}\rangle$ is normalised and orthogonal to $|\phi\rangle$, the result of applying phase estimation on $|r\rangle$ with s ancilla qubits is a state of the form

$$\frac{1}{\sqrt{2}}|\phi'\rangle|0^s\rangle + \frac{1}{\sqrt{2}}\sum_{k,\theta_k>0}\lambda_k|\psi_k\rangle|\omega_k\rangle,$$

where $|\psi_k\rangle$ is an eigenvector of $R_B R_A$ with eigenvalue $\mathrm{e}^{2i\theta_k}$. Write each $|\omega_k\rangle$ as $|\omega_k\rangle = \mu_k|0^s\rangle + |\omega'_k\rangle$ for some subnormalised vectors $|\omega'_k\rangle$ orthogonal to $|0^s\rangle$. If we obtain outcome $|0^s\rangle$ when we measure the second register, which will occur with probability at least $1/2$, the first register collapses to

$$|\widetilde{\phi'}\rangle = \frac{1}{\sqrt{1+\sum_{k,\theta_k>0}|\lambda_k\mu_k|^2}}\left(|\phi'\rangle + \sum_{k,\theta_k>0}\lambda_k\mu_k|\psi_k\rangle\right).$$

To bound the distance between $|\widetilde{\phi'}\rangle$ and the desired state $|\phi'\rangle$, we split the sum into two parts. For any $\varepsilon > 0$, via Theorem 2.2 we have

$$\sum_{k,\theta_k\geq\varepsilon}|\lambda_k\mu_k|^2 \leq \sum_{k,\theta_k\geq\varepsilon}|\mu_k|^2 = O(1/(2^s\varepsilon))\,.$$

On the other hand, we prove the following technical claim in Section 2.5 below. Recall that P_ε is the projector onto the span of the eigenvectors of $R_B R_A$ with eigenvalues $\mathrm{e}^{2i\theta}$ such that $|\theta| \leq \varepsilon$.

Lemma 2.5. $\|P_\varepsilon|\phi^{\perp}\rangle\| = O(\varepsilon\sqrt{Tn})$.

Given Lemma 2.5, we have

$$\sum_{k,0<\theta_k\le\varepsilon} |\lambda_k\mu_k|^2 \le \sum_{k,0<\theta_k\le\varepsilon} |\lambda_k|^2 = \|P_\varepsilon|\phi^\perp\rangle\|^2 = O(\varepsilon^2 Tn)\,.$$

Fixing an accuracy δ and taking $\varepsilon = \Theta(\delta/\sqrt{Tn})$, $2^s = O(\sqrt{Tn}/\delta^3)$, we have $\||\widetilde{\phi'}\rangle - |\phi'\rangle\| = O(\delta)$. By Fact 2.3, measuring $|\widetilde{\phi'}\rangle$ in the computational basis is indistinguishable from measuring $|\phi'\rangle$, except with probability $O(\delta)$. As the algorithm uses $O(\log n)$ copies of $|\phi'\rangle$ in total, the overall total variation distance between the distribution on measurement outcomes obtained by using copies of $|\widetilde{\phi'}\rangle$ versus that using copies of $|\phi'\rangle$ can be bounded by an arbitrarily small constant by taking $\delta = O(1/\log n)$. This corresponds to the failure probability of the algorithm being bounded by an arbitrarily small constant. The overall complexity of the algorithm is therefore $O(\sqrt{Tn}\log^3 n)$.[3] As before, the failure probability can be made arbitrarily small via repetition.

One may wonder whether the same approach could work when the number of marked elements is greater than 1. In this case, the subspace spanned by eigenvectors of $R_B R_A$ with eigenvalue 1 will include states of the form (2.2) for different marked elements x_0. Dependent on the position of these marked elements in the tree, the projection of $|r\rangle$ onto this subspace could be a state with high weight on levels of the tree near the root, unlike the case of a single marked element, where the weight of $|\phi'\rangle$ is uniform across the entire path to x_0. So the search procedure above will be less efficient, as fewer levels of the tree will be traversed at each iteration.

In Section 4 we discuss some other barriers to improving the complexity and applicability of these algorithms.

2.5 Proof of technical claim for search with one marked element

In this subsection, we prove the following claim from Section 2.4.

Lemma 2.5 (restated). $\|P_\chi|\phi^\perp\rangle\| = O(\chi\sqrt{Tn})$.

Let x_0 be the unique marked vertex, assuming for simplicity in the proof (as justified in Section 2.4) that $\ell(x_0) = n$, and hence that x_0 is a leaf in the tree. We can write

$$\begin{aligned}|\phi^\perp\rangle &= \sqrt{2}|r\rangle - |\phi'\rangle = \sqrt{2}|r\rangle - \frac{1}{\sqrt{2n}}\left(\sqrt{n}|r\rangle + \sum_{x\neq r, x\rightsquigarrow x_0} (-1)^{\ell(x)}|x\rangle\right) \\ &= \frac{1}{\sqrt{2}}|r\rangle - \frac{1}{\sqrt{2n}}\sum_{x\neq r, x\rightsquigarrow x_0}(-1)^{\ell(x)}|x\rangle\,.\end{aligned}$$

Recall that Π_A and Π_B are projectors onto the invariant subspaces of R_A and R_B. The invariant subspace of R_A is spanned by vectors of the form $|\psi_x^\perp\rangle$ for each vertex $x \in A$, and if $x_0 \in A$, in addition the vector $|\psi_{x_0}\rangle$. The invariant subspace of R_B is similar (replacing A with B) but also contains $|r\rangle$. Here $\langle\psi_x|\psi_x^\perp\rangle = 0$ and $|\psi_x^\perp\rangle$ has support only on $\{|x\rangle\} \cup \{|y\rangle : x \to y\}$. In order to apply the effective spectral gap lemma, we determine a vector $|\xi\rangle$ such that $\Pi_A|\xi\rangle = 0$ and $\Pi_B|\xi\rangle = |\phi^\perp\rangle$.

[3] One way to improve the polylogarithmic factors in this complexity could be to reweight the tree such that the eigenvector of $R_B R_A$ with eigenvalue 1 has more weight on x_0 (Alexander Belov, personal communication).

First assume $x_0 \in B$. We will take $|\xi\rangle$ to be a linear combination of vectors $|\psi_x\rangle$ for $x \in A$. Then the first of these two constraints is immediately satisfied. The second will be satisfied if, for a set of vectors $|\zeta\rangle$ which span the invariant subspace of R_B, i. e.,

$$|\zeta\rangle \in \{|r\rangle, |\psi_{x_0}\rangle\} \cup \{|\psi_x^\perp\rangle : \langle\psi_x^\perp|\psi_x\rangle = 0, x \in B\},$$

we have $\langle\zeta|\xi\rangle = \langle\zeta|\phi^\perp\rangle$. To compute the required inner products, first observe that $|\psi_x^\perp\rangle$ only has support on x and its children, so for all x not on the path from r to x_0, $\langle\psi_x^\perp|\phi^\perp\rangle = 0$. On the other hand, for each $x \in B$ such that $x \rightsquigarrow x_0$, define a basis for the space $\operatorname{span}\{|\psi_x^\perp\rangle : \langle\psi_x^\perp|\psi_x\rangle = 0\}$ by fixing the vectors

$$|\psi_{x,i}^\perp\rangle = -|x\rangle + (d_x - 1)|N_i(x)\rangle - \sum_{j \neq i} |N_j(x)\rangle,$$

where $N_i(x)$ denotes the i'th child of x, recalling that d_x denotes the degree of x. We have

$$\langle\psi_{x,i}^\perp|\phi^\perp\rangle = \begin{cases} -d_x/\sqrt{2n} & \text{if } i = i_0, \\ 0 & \text{otherwise.} \end{cases} \tag{2.3}$$

where i_0 denotes the unique child of x on the path to x_0.

We now find a vector $|\xi\rangle = \sum_x \alpha_x |x\rangle$ satisfying the above constraints. First, we require $\alpha_r = \langle r|\xi\rangle = \langle r|\phi^\perp\rangle = 1/\sqrt{2}$ and $\alpha_{x_0} = \langle x_0|\phi^\perp\rangle = 1/\sqrt{2n}$. For each x, let x' denote the parent of x in the tree. For $|\xi\rangle$ to be a linear combination of vectors $|\psi_x\rangle$, $x \in A$, it is necessary and sufficient that $\alpha_x = \alpha_{x'}$ for all $x \in B$; except in the case $\ell(x) = 1$, where we require $\alpha_x = \sqrt{n}\,\alpha_r$. We in addition need $\alpha_x = \alpha_{x'}$ for all $x \neq r \in A$ such that x' is not on the path to x_0, in order that $\langle\psi_{x'}^\perp|\xi\rangle = \langle\psi_{x'}^\perp|\phi^\perp\rangle = 0$. For each child y of $x \in B$, set $\alpha_y = \gamma$ if $y \not\rightsquigarrow x_0$, and $\alpha_y = \delta$ if $y \rightsquigarrow x_0$. Then from (2.3) we have the final constraints that

$$-\alpha_x + 2\gamma - \delta = 0 \text{ if } y \not\rightsquigarrow x_0,$$

$$-\alpha_x - (d_x - 2)\gamma + (d_x - 1)\delta = -\frac{d_x}{\sqrt{2n}} \text{ if } y \rightsquigarrow x_0.$$

These equations have unique solution $\gamma = \alpha_x - 1/\sqrt{2n}$, $\delta = \alpha_x - \sqrt{2/n}$. This now uniquely defines all coefficients α_x. In particular, observe that

$$\alpha_{x_0} = \sqrt{\frac{n}{2}} - \left(\frac{n-1}{2}\right)\sqrt{\frac{2}{n}} = \frac{1}{\sqrt{2n}}$$

as required.

This constructs $|\xi\rangle$ in the case where $x_0 \in B$. If instead $x_0 \in A$, the procedure is similar. Now $|\psi_{x_0}\rangle$ is not in the invariant subspace of R_B (which only makes it easier to satisfy the inner product constraints), but also $|\xi\rangle$ must be a linear combination of vectors $|\psi_x\rangle$ corresponding only to *unmarked* vertices $x \in A$. This new constraint implies that now $\alpha_{x_0} = 0$. But following the above procedure now gives

$$\alpha_{x_0} = \sqrt{\frac{n}{2}} - \frac{n}{2}\sqrt{\frac{2}{n}} = 0$$

as required. In either case, for all x, we have $|\alpha_x| \leq \sqrt{n/2}$. So $\||\xi\rangle\| = O(\sqrt{Tn})$ and hence, by the effective spectral gap lemma (Lemma 2.1), $\|P_\chi|\phi^\perp\rangle\| = O(\chi\sqrt{Tn})$.

3 From quantum walks on trees to accelerating backtracking

To complete the proofs of Theorems 1.1 and 1.2, we now verify that Algorithm 2 can be applied to search in the tree defined by a backtracking algorithm. In order to do this, it is sufficient to define a suitable efficient mapping between partial assignments and vertices in a tree, and to implement the operators R_A and R_B appropriately and efficiently. As the quantum walk subroutines assume that the root of the tree is not marked, the first step of the algorithm is to check whether $P(*^n)$ is true. If so, the algorithm immediately returns "true"; if not, it runs Algorithm 2 on a graph defined as follows.

The current state of the backtracking algorithm is represented by a vertex in a rooted tree labelled with a sequence of the form $(i_1,v_1),\dots,(i_\ell,v_\ell)$, for $1 \le \ell \le n$. The sequence corresponds to a partial assignment $x \in \mathcal{D}$ where we assign $x_{i_k} = v_k$ for $k = 1,\dots,\ell$, and $x_j = *$ for all other indices j. The tree only contains vertices corresponding to valid partial assignments. Each vertex except for the root (which is labelled with the empty sequence) is connected to its parent, the vertex labelled with $(i_1,v_1),\dots,(i_{\ell-1},v_{\ell-1})$. It is also connected to all vertices of the form $(i_1,v_1),\dots,(i_\ell,v_\ell),(j,w)$, where $j = h((i_1,v_1),\dots,(i_\ell,v_\ell))$, $w \in [d]$, and $P((i_1,v_1),\dots,(i_\ell,v_\ell),(j,w))$ is not false. That is, all vertices corresponding to valid partial assignments which extend the current partial assignment by assigning a value to the variable whose index is given by h. It is convenient to assume that the predicate P and the heuristic h take as input a string of (index, value) pairs which describe value assignments to variables, rather than an element of $\mathcal{D}$; if not, converting between these representations can be done in time $O(n)$. We will also assume that, for all complete assignments, the predicate returns either true or false (as it should do).

The algorithm takes place within the Hilbert space $\mathcal{H}^{(n)} = \mathbb{C}^{n+1} \otimes (\mathbb{C}^{n+1} \otimes \mathbb{C}^{d+1})^{\otimes n}$ together with an ancilla space. Each basis vector within $\mathcal{H}^{(n)}$ represents a partial assignment described by a sequence as above. The first register stores a level ℓ between 0 and n, representing the length of the sequence (the number of non-$*$'s in the assignment). Each of the next ℓ registers stores a pair (i_k,v_k) giving the index of a variable (an integer between 1 and n) and the assignment to that variable (an integer between 0 and $d-1$). Except during updates to the state, the remaining $n-\ell$ registers all contain the pair $(0,*)$. The algorithm can easily be modified to use qubits if desired, rather than systems with dimension $n+1$ and $d+1$, by encoding each subsystem in $O(\log n + \log d)$ qubits.

Let $U_{\alpha,S}$, for $S \subseteq [d]$ and $\alpha \in \mathbb{R}$, act on $\mathbb{C}^{d+1}$ with basis $\{|*\rangle, |0\rangle, \dots, |d-1\rangle\}$ by mapping $|*\rangle \mapsto |\phi_{\alpha,S}\rangle$, where

$$|\phi_{\alpha,S}\rangle := \frac{1}{\sqrt{\alpha|S|+1}} \left(|*\rangle + \sqrt{\alpha} \sum_{i \in S} |i\rangle \right).$$

We assume that, for any subset $S \subseteq [d]$ and any fixed $\alpha \in \mathbb{R}$, we can perform $U_{\alpha,S}$ and its inverse in time $O(1)$ each. Dependent on the gate set being used, we may not be able to exactly implement these operators within this running time bound. But as we assume that $d = O(1)$, they can always be approximately implemented up to accuracy $1-\varepsilon$ in time $\operatorname{poly}\log(1/\varepsilon)$ using the Solovay-Kitaev theorem [47], which would multiply the running time of the overall algorithm by at most a polylogarithmic factor. The running time of the Solovay-Kitaev approach has a polynomial dependence on d, but $U_{\alpha,S}$ could also be implemented in time $\operatorname{poly}(\log d, \log 1/\varepsilon)$ via the use of a quantum Fourier transform (see, e. g., [15, Appendix D]); we omit the details. By applying $U_{\alpha,S}$ and its inverse we can perform the operation $I - 2|\phi_{\alpha,S}\rangle\langle\phi_{\alpha,S}|$.

In order to use Algorithm 2, we need to implement the operators R_A and R_B. The implementation

```
Input: A basis state $|\ell\rangle|(i_1,v_1)\rangle\dots|(i_n,v_n)\rangle \in \mathcal{H}^{(n)}$ corresponding to a partial assignment $x_{i_1} = v_1,\dots,x_{i_\ell} = v_\ell$. Ancilla registers $\mathcal{H}_{\text{anc}}$, $\mathcal{H}_{\text{next}}$, $\mathcal{H}_{\text{children}}$, storing a tuple $(a,j,S)$, where $a \in \{*\}\cup[d]$, $j \in \{0,\dots,n\}$, $S \subseteq [d]$, initialised to $a = *$, $j = 0$, $S = \emptyset$.

1. If $P(x)$ is true, return.
2. If $\ell$ is odd, subtract $h((i_1,v_1),\dots,(i_{\ell-1},v_{\ell-1}))$ from $i_\ell$ and swap $a$ with $v_\ell$.
3. If $a \neq *$, subtract 1 from $\ell$. (Now $\ell$ is even and $(i_{\ell+1},v_{\ell+1}) = (0,*)$.)
4. Add $h((i_1,v_1),\dots,(i_\ell,v_\ell))$ to $j$.
5. For each $w \in [d]$:
    (a) If $P((i_1,v_1),\dots,(i_\ell,v_\ell),(j,w))$ is not false, set $S = S \cup \{w\}$.
6. If $\ell = 0$, perform the operation $I - 2|\phi_{n,S}\rangle\langle\phi_{n,S}|$ on $\mathcal{H}_{\text{anc}}$. Otherwise, perform the operation $I - 2|\phi_{1,S}\rangle\langle\phi_{1,S}|$ on $\mathcal{H}_{\text{anc}}$.
7. Uncompute $S$ and $j$ by reversing steps 5 and 4.
8. If $a \neq *$, add 1 to $\ell$. If $\ell$ is now odd, add $h((i_1,v_1),\dots,(i_{\ell-1},v_{\ell-1}))$ to $i_\ell$ and swap $v_\ell$ with $a$. (Now $a = *$ again.)
```

Algorithm 3: Implementation of the operator R_A.

of R_A using $I - 2|\phi_{\alpha,S}\rangle\langle\phi_{\alpha,S}|$, P and h is described in Algorithm 3. R_B is similar, except that: step 1 is replaced with the check "If $P(x)$ is true or $\ell = 0$, return"; "odd" is replaced with "even" in steps 2 and 8; and the check "If $\ell = 0$" is removed from step 6. The first of these changes is because R_B should leave the root of the tree invariant; and the last is because ℓ is always odd at that point in the modified algorithm, so the check is unnecessary.

We now argue that Algorithm 3 correctly implements R_A. Write $x = (i_1,v_1),\dots,(i_\ell,v_\ell)$ for the partial assignment passed to the algorithm, and write $x' = (i_1,v_1),\dots,(i_{\ell-1},v_{\ell-1})$ for the parent partial assignment in the tree. The goal of the algorithm is to implement the operator $\bigoplus_{x\in A} D_x$ defined in Section 2. For each $x \in A$, D_x only acts on the subspace corresponding to x and its children. To implement D_x, it is therefore sufficient to map the basis state corresponding to $(i_1,v_1),\dots,(i_\ell,v_\ell)$, and all the basis states corresponding to $(i_1,v_1),\dots,(i_\ell,v_\ell),(j,w)$ for $w \in [d]$, where $j = h((i_1,v_1),\dots,(i_\ell,v_\ell))$ and ℓ is even, to a $(d+1)$-dimensional subspace on which $U_{\alpha,S}$ can be implemented, and then returning to the original subspace. This is precisely what Algorithm 3 does.

In more detail, the algorithm performs the following steps. First, it does nothing when x is marked, corresponding to the definition of D_x. If x is not marked, the behaviour depends on whether ℓ is even (corresponding to $x \in A$) or ℓ is odd (corresponding to $x \in B$). Define y by setting $y = x$ if $x \in A$, and $y = x'$ if $x \in B$. Then the algorithm implements an inversion about $|\psi_y\rangle$, which is split into 4 subparts as follows.

- Steps 2-3: Perform a map of the form $|x\rangle \mapsto |y\rangle|*\rangle$ for $x \in A$, and $|x\rangle \mapsto |y\rangle|w\rangle$ for $x \in B$, where w is the value of x at the $h(x')$'th position, i. e., the most recent variable assignment that was made by the backtracking algorithm.

- Steps 4-5: Determine the children of y.

- Step 6: Perform the operation $I - 2|\psi_y\rangle\langle\psi_y|$ using the knowledge of the children of y.

- Steps 7-8: Uncompute junk and reverse the first map.

It can be verified that the algorithm implements the desired behaviour for all basis state inputs of the form $|\ell\rangle|(i_1,v_1)\rangle\dots|(i_n,v_n)\rangle$ such that $(i_1,v_1),\dots,(i_\ell,v_\ell)$ is a valid path in the backtracking tree; we omit the routine details. As the algorithm implements the operation $R_A = \bigoplus_{x\in A} D_x$ unitarily for all basis states $|x\rangle$, it also implements R_A correctly for all superpositions of basis states. Together with the similar implementation of R_B, this is enough to implement Algorithm 2. For each use of R_A and R_B the algorithm uses $O(1)$ auxiliary operations as claimed.

4 Improving the quantum walk algorithm?

We finish by addressing the question of how tight the bounds are which we have obtained on quantum search in trees. It is clear that, given a tree with T vertices, we must have a lower bound of the form $\Omega(\sqrt{T})$ for finding a marked vertex (otherwise, we could use the algorithm to solve the unstructured search problem on T elements using $o(\sqrt{T})$ quantum queries, which is impossible [12]). There are several plausible ways in which the complexity of the algorithm presented here could be improved to get closer to this bound. However, there appear to be some challenges to doing so in each of these cases.

1. Reduction of the dependence on the depth n. It is easy to see that, if we would like to apply the quantum backtracking algorithm to general trees, there must be some dependence on the depth in the running time. Indeed, consider a path on T vertices, which has depth $T-1$. Then, if the marked vertex is the last one in the path, we require $\Omega(T)$ steps to find it. More generally, it was shown by Aaronson and Ambainis [1] that for each pair T and n, there is a tree containing T vertices and with depth $O(n)$ such that determining the existence of a marked vertex requires $\Omega(\sqrt{Tn})$ queries. This holds even if we know the tree in advance and are allowed to perform arbitrary "local" operations to search within it.

2. Reduction of the overhead for searching with multiple marked vertices. It would be interesting to determine whether the search algorithm in Section 2.4 could be generalised to work with a similar efficiency for an arbitrary number of marked vertices. The question of when one can convert a quantum walk speedup for detecting a marked element to a speedup for finding a marked element has been studied extensively. One recent example is work of Høyer and Komeili [34], which describes an improved algorithm for quantum walk search on the torus; see [34] for many further references.

   But while it was shown by Szegedy [53] that the time to detect a marked element using a quantum walk is at most the square root of the classical hitting time, it is not known whether the time to find a marked element has the same scaling in general. Indeed, Krovi et al. [38] have described a way (generalising previous results of [54, 41]) to modify the original quantum walk approach of Szegedy to obtain a quadratic speedup for the search problem in the case where there is a unique marked element. However, if there is more than one marked element, the running time of their algorithm scales with a quantity they call the extended hitting time, which may be larger than the hitting time. In any case, all of these algorithms assume that the graph is known in advance and the initial state of the quantum walk algorithm corresponds to the stationary distribution of the random walk. Neither of these assumptions applies here.

3. Reduction of the dependence on k to find one, or all, of k marked vertices. For the unstructured search problem with k marked elements out of T, Grover's algorithm can find a marked element using $O(\sqrt{T/k})$ queries, which implies an algorithm which finds all marked elements in $O(\sqrt{Tk})$ queries. It would be natural to hope for a bound of a similar form for quantum search on trees, e. g., $O(\sqrt{Tn/k})$ to find a marked vertex and $O(\sqrt{Tnk})$ to find all k of them. Unfortunately, it is far from clear that this can be achieved.

   Indeed, consider the following argument due to Alexander Belov. Imagine we have access to an algorithm $\mathcal{A}$ which finds one of $k > 1$ marked vertices using $o(\sqrt{Tn})$ queries, and consider an arbitrary tree containing one marked leaf ℓ_0. Modify the tree by attaching a subtree of depth $O(\log k)$ below that leaf containing k vertices, all of which are marked and are labelled such that ℓ_0 can be determined from their labels. Then, using $\mathcal{A}$, we can find one of these vertices using $o(\sqrt{Tn})$ queries. Finding such a vertex enables us to find ℓ_0 with no additional queries, contradicting the aforementioned $\Omega(\sqrt{Tn})$ lower bound [1]. However, this argument does not rule out the possibility that some other approach could find all k marked vertices in, for example, $O(\sqrt{Tnk})$ time.

Acknowledgements

I would like to thank Alexander Belov, Aram Harrow, Oded Regev, and several anonymous referees for helpful comments on previous versions of this paper.

References

[1] SCOTT AARONSON AND ANDRIS AMBAINIS: Quantum search of spatial regions. *Theory of Computing*, 1(4):47–79, 2005. Preliminary version in FOCS'03. [doi:10.4086/toc.2005.v001a004, arXiv:quant-ph/0303041] 4, 5, 18, 19

[2] ERDEM ALKIM, LÉO DUCAS, THOMAS PÖPPELMANN, AND PETER SCHWABE: Post-quantum key exchange – a new hope. In *Proc. 25th USENIX Security Symp. (USENIX Security'16)*, pp. 327–343. USENIX Association, 2016. Available at IACR. 7

[3] ANDRIS AMBAINIS: Quantum search algorithms. *ACM SIGACT News*, 35(2):22–35, 2004. [doi:10.1145/992287.992296, arXiv:quant-ph/0504012] 7

[4] ANDRIS AMBAINIS: Quantum walk algorithm for element distinctness. *SIAM J. Comput.*, 37(1):210–239, 2007. Preliminary version in FOCS'04. [doi:10.1137/S0097539705447311, arXiv:quant-ph/0311001] 5

[5] ANDRIS AMBAINIS, ANDREW M. CHILDS, BEN W. REICHARDT, ROBERT ŠPALEK, AND SHENGYU ZHANG: Any AND-OR formula of size N can be evaluated in time $N^{1/2+o(1)}$ on a quantum computer. *SIAM J. Comput.*, 39(6):2513–2530, 2010. Preliminary version in FOCS'07. [doi:10.1137/080712167, arXiv:quant-ph/0703015] 7

[6] ANDRIS AMBAINIS AND RONALD DE WOLF: Average-case quantum query complexity. *J. Phys. A: Math. Gen.*, 34(35):6741–6754, 2001. Preliminary version in STACS'00. [doi:10.1088/0305-4470/34/35/302, arXiv:quant-ph/9904079] 4

[7] ANDRIS AMBAINIS AND MARTINS KOKAINIS: Quantum algorithm for tree size estimation, with applications to backtracking and 2-player games. In *Proc. 49th STOC*, pp. 989–1002. ACM Press, 2017. [doi:10.1145/3055399.3055444, arXiv:1704.06774] 4, 7

[8] OLA ANGELSMARK, VILHELM DAHLLÖF, AND PETER JONSSON: Finite domain constraint satisfaction using quantum computation. In *Proc. 27th Internat. Symp. Mathemat. Found. Comput. Sci. (MFCS'02)*, pp. 93–103. Springer, 2002. [doi:10.1007/3-540-45687-2_7] 6

[9] PETER VAN BEEK: Backtracking search algorithms. In *Handbook of Constraint Programming*. Elsevier, 2006. [doi:10.1016/S1574-6526(06)80008-8] 2, 5

[10] ALEKSANDRS BELOVS: Quantum walks and electric networks, 2013. [arXiv:1302.3143] 5, 8, 9, 10

[11] ALEKSANDRS BELOVS, ANDREW M. CHILDS, STACEY JEFFERY, ROBIN KOTHARI, AND FRÉDÉRIC MAGNIEZ: Time-efficient quantum walks for 3-distinctness. In *Proc. 40th Internat. Colloq. on Automata, Languages and Programming (ICALP'13)*, pp. 105–122. Springer, 2013. [doi:10.1007/978-3-642-39206-1_10, arXiv:1302.7316] 5, 9

[12] CHARLES H. BENNETT, ETHAN BERNSTEIN, GILLES BRASSARD, AND UMESH V. VAZIRANI: Strengths and weaknesses of quantum computing. *SIAM J. Comput.*, 26(5):1510–1523, 1997. [doi:10.1137/S0097539796300933, arXiv:quant-ph/9701001] 18

[13] ETHAN BERNSTEIN AND UMESH V. VAZIRANI: Quantum complexity theory. *SIAM J. Comput.*, 26(5):1411–1473, 1997. Preliminary version in STOC'93. [doi:10.1137/S0097539796300921] 9

[14] GILLES BRASSARD, PETER HØYER, MICHELE MOSCA, AND ALAIN TAPP: Quantum amplitude amplification and estimation. In SAMUEL J. LOMONACO, JR. AND HOWARD E. BRANDT, editors, *Quantum Computation and Information*, volume 305 of *Contemporary Mathematics*, pp. 53–74. AMS, 2002. [doi:10.1090/conm/305/05215, arXiv:quant-ph/0005055] 6

[15] CHRIS CADE, ASHLEY MONTANARO, AND ALEKSANDRS BELOVS: Time and space efficient quantum algorithms for detecting cycles and testing bipartiteness. *Quantum Inf. Comput.*, 18(1 & 2):18–50, 2018. [arXiv:1610.00581] 16

[16] NICOLAS J. CERF, LOV K. GROVER, AND COLIN P. WILLIAMS: Nested quantum search and structured problems. *Phys. Rev. A*, 61(3):032303:1–14, 2000. [doi:10.1103/PhysRevA.61.032303, arXiv:quant-ph/9806078] 5, 6

[17] ASHOK K. CHANDRA, PRABHAKAR RAGHAVAN, WALTER L. RUZZO, ROMAN SMOLENSKY, AND PRASOON TIWARI: The electrical resistance of a graph captures its commute and cover times. *Comput. Complexity*, 6(4):312–340, 1996. Preliminary version in STOC'89. [doi:10.1007/BF01270385] 5, 9

[18] YUANMI CHEN AND PHONG Q. NGUYEN: BKZ 2.0: Better lattice security estimates. In *Proc. 17th Internat. Conf. on the Theory and Application of Cryptology and Information Security (ASIACRYPT'11)*, pp. 1–20. Springer, 2011. [doi:10.1007/978-3-642-25385-0_1] 7

[19] ANDREW M. CHILDS, RICHARD CLEVE, ENRICO DEOTTO, EDWARD FARHI, SAM GUTMANN, AND DANIEL A. SPIELMAN: Exponential algorithmic speedup by a quantum walk. In *Proc. 35th STOC*, pp. 59–68. ACM Press, 2003. [doi:10.1145/780542.780552, arXiv:quant-ph/0209131] 5

[20] RICHARD CLEVE, ARTUR EKERT, CHIARA MACCHIAVELLO, AND MICHELE MOSCA: Quantum algorithms revisited. *Proc. Royal Soc. A*, 454(1969):339–354, 1998. [doi:10.1098/rspa.1998.0164, arXiv:quant-ph/9708016] 8

[21] EVGENY DANTSIN, VLADIK KREINOVICH, AND ALEXANDER WOLPERT: On quantum versions of record-breaking algorithms for SAT. *ACM SIGACT News*, 36(4):103–108, 2005. [doi:10.1145/1107523.1107524] 7

[22] MARTIN DAVIS, GEORGE LOGEMANN, AND DONALD LOVELAND: A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962. [doi:10.1145/368273.368557] 2

[23] MARTIN DAVIS AND HILARY PUTNAM: A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960. [doi:10.1145/321033.321034] 2

[24] RAFAËL DEL PINO, VADIM LYUBASHEVSKY, AND DAVID POINTCHEVAL: The whole is less than the sum of its parts: Constructing more efficient lattice-based AKEs. In *Proc. 10th Internat. Conf. on Security and Cryptography for Networks (SCN'16)*, pp. 273 – 291. Springer, 2016. [doi:10.1007/978-3-319-44618-9_15] 7

[25] NIKLAS EÉN AND NIKLAS SÖRENSSON: An extensible SAT-solver. In *Proc. 6th Internat. Conf. on Theory and Applications of Satisfiability Testing (SAT'03)*, pp. 502–518. Springer, 2003. [doi:10.1007/978-3-540-24605-3_37] 2

[26] EDWARD FARHI, JEFFERY GOLDSTONE, SAM GUTMANN, JOSHUA LAPAN, ANDREW LUNDGREN, AND DANIEL PREDA: A quantum adiabatic evolution algorithm applied to random instances of an NP-complete problem. *Science*, 292(5516):472–475, 2001. [doi:10.1126/science.1057726, arXiv:quant-ph/0104129] 7

[27] EDWARD FARHI, JEFFERY GOLDSTONE, SAM GUTMANN, AND MICHAEL SIPSER: Quantum computation by adiabatic evolution. Technical report, MIT, 2000. [arXiv:quant-ph/0001106] 7

[28] EDWARD FARHI AND SAM GUTMANN: Quantum computation and decision trees. *Phys. Rev. A*, 58(2):915–928, 1998. [doi:10.1103/PhysRevA.58.915, arXiv:quant-ph/9706062] 6

[29] EUGENE C. FREUDER AND ALAN K. MACKWORTH: Constraint satisfaction: An emerging paradigm. In *Handbook of Constraint Programming*, pp. 13–27. Elsevier, 2006. [doi:10.1016/S1574-6526(06)80006-4] 5

[30] MARTIN FÜRER: Solving NP-complete problems with quantum search. In *Proc. 10th Latin Amer. Symp. on Theoretical Informatics (LATIN'08)*, pp. 784–792. Springer, 2008. [doi:10.1007/978-3-540-78773-0_67] 6

[31] NICOLAS GAMA, PHONG Q. NGUYEN, AND ODED REGEV: Lattice enumeration using extreme pruning. In *Proc. 29th Internat. Conf. on the Theory and Application of Cryptographic Techniques (EUROCRYPT'10)*, pp. 257–278. Springer, 2010. [doi:10.1007/978-3-642-13190-5_13] 7

[32] CARLA P. GOMES, HENRY KAUTZ, ASHISH SABHARWAL, AND BART SELMAN: Satisfiability solvers. In *Handbook of Knowledge Representation*, pp. 89–134. Elsevier, 2008. [doi:10.1016/S1574-6526(07)03002-7] 2

[33] LOV K. GROVER: Quantum mechanics helps in searching for a needle in a haystack. *Phys. Rev. Lett.*, 79(2):325–328, 1997. [doi:10.1103/PhysRevLett.79.325, arXiv:quant-ph/9706033] 1

[34] PETER HØYER AND MOJTABA KOMEILI: Efficient quantum walk on the grid with multiple marked elements. In *Proc. 34th Symp. Theoret. Aspects of Computer Sci. (STACS'17)*, pp. 42:1–42:14. DROPS, 2017. [doi:10.4230/LIPIcs.STACS.2017.42, arXiv:1612.08958] 18

[35] STACEY JEFFERY AND SHELBY KIMMEL: NAND-trees, average choice complexity, and effective resistance, 2015. [arXiv:1511.02235] 7

[36] ALEXEI KITAEV: Quantum measurements and the abelian stabilizer problem, 1996. [arXiv:quant-ph/9511026] 8

[37] DONALD E. KNUTH: Estimating the efficiency of backtrack programs. *Mathematics of Computation*, 29(129), 1975. [doi:10.1090/S0025-5718-1975-0373371-6] 5

[38] HARI KROVI, FRÉDÉRIC MAGNIEZ, MARIS OZOLS, AND JÉRÉMIE ROLAND: Quantum walks can find a marked element on any graph. *Algorithmica*, 74(2):851–907, 2016. Preliminary version in ICALP'10. [doi:10.1007/s00453-015-9979-8, arXiv:1002.2419] 5, 19

[39] TROY LEE, RAJAT MITTAL, BEN W. REICHARDT, ROBERT ŠPALEK, AND MARIO SZEGEDY: Quantum query complexity of state conversion. In *Proc. 52nd FOCS*, pp. 344–353. IEEE Comp. Soc. Press, 2011. [doi:10.1109/FOCS.2011.75, arXiv:1011.3020] 8

[40] INÊS LYNCE AND JOÃO P. MARQUES-SILVA: An overview of backtrack search satisfiability algorithms. *Annals of Mathematics and Artificial Intelligence*, 37(3):307–326, 2003. [doi:10.1023/A:1021264516079] 2

[41] FRÉDÉRIC MAGNIEZ, ASHWIN NAYAK, PETER C. RICHTER, AND MIKLOS SANTHA: On the hitting times of quantum versus random walks. *Algorithmica*, 63(1–2):91–116, 2012. Preliminary version in SODA'09. [doi:10.1007/s00453-011-9521-6, arXiv:0808.0084] 19

[42] FRÉDÉRIC MAGNIEZ, ASHWIN NAYAK, JÉRÉMIE ROLAND, AND MIKLOS SANTHA: Search via quantum walk. *SIAM J. Comput.*, 40(1):142–164, 2011. Preliminary version in STOC'07. [doi:10.1137/090745854, arXiv:quant-ph/0608026] 5, 9

[43] SALVATORE MANDRÀ, GIAN GIACOMO GUERRESCHI, AND ALÁN ASPURU-GUZIK: Faster than classical quantum algorithm for dense formulas of exact satisfiability and occupation problems. *New J. Phys.*, 18(7):073003, 2016. [doi:10.1088/1367-2630/18/7/073003, arXiv:1512.00859] 7

[44] JOÃO MARQUES-SILVA: The impact of branching heuristics in propositional satisfiability algorithms. In *Proc. 9th Portuguese Conf. on Artificial Intelligence: Progress in Artificial Intelligence (EPIA'99)*, pp. 62–74. Springer, 1999. [doi:10.1007/3-540-48159-1_5] 3

[45] ASHLEY MONTANARO: Quantum walk speedup of backtracking algorithms, 2015. [arXiv:1509.02374] 5

[46] DOMINIC J. MOYLETT, NOAH LINDEN, AND ASHLEY MONTANARO: Quantum speedup of the traveling-salesman problem for bounded-degree graphs. *Phys. Rev. A*, 95(3):032323:1–10, 2017. [doi:10.1103/PhysRevA.95.032323, arXiv:1612.06203] 7

[47] MICHAEL A. NIELSEN AND ISAAC L. CHUANG: *Quantum Computation and Quantum Information*. Cambridge University Press, 2000. 16

[48] RENATO PORTUGAL, RAQUELINE AZEVEDO M. SANTOS, T. D. FERNANDES, AND DEMERSON N. GONÇALVES: The staggered quantum walk model. *Quantum Information Processing*, 15(1):85–101, 2016. [doi:10.1007/s11128-015-1149-z, arXiv:1505.04761] 10

[49] CLAUS-PETER SCHNORR AND MARTIN EUCHNER: Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Math. Program.*, 66(1–3):181–199, 1994. Preliminary version in FCT'91. [doi:10.1007/BF01581144] 7

[50] UWE SCHÖNING: A probabilistic algorithm for *k*-SAT and constraint satisfaction problems. In *Proc. 40th FOCS*, pp. 410–414. IEEE Comp. Soc. Press, 1999. [doi:10.1109/SFFCS.1999.814612] 7

[51] NEIL SHENVI, JULIA KEMPE, AND K. BIRGITTA WHALEY: Quantum random-walk search algorithm. *Phys. Rev. A*, 67(5):052307:1–11, 2003. [doi:10.1103/PhysRevA.67.052307, arXiv:quant-ph/0210064] 5

[52] LARRY STOCKMEYER: On approximation algorithms for #P. *SIAM J. Comput.*, 14(4):849–861, 1985. [doi:10.1137/0214060] 6

[53] MARIO SZEGEDY: Quantum speed-up of Markov chain based algorithms. In *Proc. 45th FOCS*, pp. 32–41. IEEE Comp. Soc. Press, 2004. [doi:10.1109/FOCS.2004.53, arXiv:quant-ph/0401053] 5, 9, 18

[54] AVATAR TULSI: Faster quantum walk algorithm for the two dimensional spatial search. *Phys. Rev. A*, 78(1):012310:1–6, 2008. [doi:10.1103/PhysRevA.78.012310, arXiv:0801.0497] 19

[55] GUOMING WANG: Efficient quantum algorithms for analyzing large sparse electrical networks. *Quantum Inf. Comput.*, 17(11 & 12):987–1026, 2017. [arXiv:1311.1851] 7, 12

[56] MINGYU XIAO AND HIROSHI NAGAMOCHI: An exact algorithm for TSP in degree-3 graphs via circuit procedure and amortization on connectivity structure. *Algorithmica*, 74(2):713–741, 2016. Preliminary version in TAMC'13. [doi:10.1007/s00453-015-9970-4, arXiv:1212.6831] 7

[57] MINGYU XIAO AND HIROSHI NAGAMOCHI: An improved exact algorithm for TSP in graphs of maximum degree 4. *Theory Comput. Syst.*, 58(2):241–272, 2016. Preliminary version in COCOON'12. [doi:10.1007/s00224-015-9612-x] 7

AUTHOR

Ashley Montanaro
Reader in Quantum Computation
University of Bristol
Bristol, UK
ashley.montanaro@bristol.ac.uk
`http://people.maths.bris.ac.uk/~csxam/`

ABOUT THE AUTHOR

ASHLEY MONTANARO graduated from the University of Bristol in 2008; his advisor was Richard Jozsa. His academic interests include many aspects of quantum computing and quantum information theory, with a particular focus on quantum algorithms and quantum computational complexity. Outside of work, he enjoys writing self-referential biographical entries.