Quantum speedup of the Travelling Salesman Problem for bounded-degree graphs

Dominic J. Moylett,[1,2,3,*] Noah Linden,[4,†] and Ashley Montanaro[4,‡]

[1] *Quantum Engineering Technology Labs, H. H. Wills Physics Laboratory and Department of Electrical & Electronic Engineering, University of Bristol, BS8 1FD, UK*

[2] *Quantum Engineering Centre for Doctoral Training, H. H. Wills Physics Laboratory and Department of Electrical & Electronic Engineering, University of Bristol, BS8 1FD, UK*

[3] *Heilbronn Institute for Mathematical Research, University of Bristol, BS8 1SN, UK*

[4] *School of Mathematics, University of Bristol, BS8 1TW, UK*

(Dated: December 18, 2016)

The Travelling Salesman Problem is one of the most famous problems in graph theory. However, little is currently known about the extent to which quantum computers could speed up algorithms for the problem. In this paper, we prove a quadratic quantum speedup when the degree of each vertex is at most 3 by applying a quantum backtracking algorithm to a classical algorithm by Xiao and Nagamochi. We then use similar techniques to accelerate a classical algorithm for when the degree of each vertex is at most 4, before speeding up higher-degree graphs via reductions to these instances.

I. INTRODUCTION

A salesman has a map of n cities that they want to visit, including the roads between the cities and how long each road is. Their aim is to start at their home, visit each city and then return home. To avoid wasting time, they want to visit each city exactly once and travel via the shortest route. So what route should the salesman take?

This is an instance of the Travelling Salesman Problem (TSP). More generally, this problem takes an undirected graph $G = (V, E)$ of n vertices connected by m weighted edges and returns the shortest cycle which passes through every vertex exactly once, known as a Hamiltonian cycle, if such a cycle exists. If no Hamiltonian cycle exists, we should report that no Hamiltonian cycle has been found. The length or cost of an edge is given by an $n \times n$ matrix $C = (c_{ij})$ of positive integers, known as a cost matrix. This problem has a number of applications, ranging from route finding as in the story above to circuit board drilling [1].

Unfortunately, the salesman might have to take a long time in order to find the shortest route. The TSP has been shown to be NP-hard [2, Chapter 3], suggesting that even the best algorithms for exactly solving it must take a superpolynomial amount of time. Nevertheless, the importance of the problem has motivated a substantial amount of classical work to develop algorithms for solving it provably more efficiently than the naïve algorithm which checks all $O((n-1)!)$ of the potential Hamiltonian cycles in the graph. Here we consider whether these algorithms can be accelerated using quantum computational techniques.

Grover's famous quantum algorithm [3] for fast unstructured search can be applied to the naïve classical algorithm to achieve a runtime of $O(\sqrt{n!})$, up to polynomial terms in n. However, the best classical algorithms are already substantially faster than this. For many years, the algorithm with the best proven worst-case bounds for the general TSP was the Held-Karp algorithm [4], which runs in $O(n^2 2^n)$ time and uses $O(n2^n)$ space. This algorithm uses the fact that for any shortest path, any subpath visiting a subset of vertices on that path must be the shortest path for visiting those vertices. Held and Karp used this to solve the TSP by computing the length of the optimal route for starting at vertex 1, visiting every vertex in a set $S \subseteq V$ and finishing at a vertex $l \in S$. Denoting the length of this optimal route $D(S, l)$, they showed that this distance could be computed as

$$D(S,l) = \begin{cases} c_{1l} & \text{if } S = \{l\} \\ \min_{m \in S \setminus \{l\}} \left[D(S \setminus \{l\}, m) + c_{ml} \right] & \text{otherwise.} \end{cases}$$

Solving this relation recursively for $S = V$ would result in iterating over all $O((n-1)!)$ Hamiltonian cycles again, but Held and Karp showed that the relation could be solved in $O(n^2 2^n)$ time using dynamic programming. Björklund et al. [5] developed on this result, showing that modifications to the Held-Karp algorithm could yield a runtime of

$$O((2^{k+1} - 2k - 2)^{n/(k+1)} \operatorname{poly}(n)),$$

where k is the largest degree of any vertex in the graph; this bound is strictly less than $O(2^n)$ for all fixed k. Unfortunately, it is not known whether quantum algorithms can accelerate general dynamic programming algorithms. Similarly, it is unclear whether TSP algorithms based around the standard classical techniques of branch-and-bound [6] or branch-and-cut [7] are amenable to quantum speedup.

Here we apply known quantum-algorithmic techniques to accelerate more recent classical TSP algorithms for the important special case of bounded-degree graphs. We say that a graph G is degree-k if the maximal degree of

* Corresponding author: dominic.moylett@bristol.ac.uk
† n.linden@bristol.ac.uk
‡ ashley.montanaro@bristol.ac.uk

any vertex in G is at most k. A recent line of research has produced a sequence of algorithms which improve on the $O^*(2^n)$ runtime of the general Held-Karp algorithm in this setting, where the notation $O^*(c^n)$ omits polynomial factors in n. First, Eppstein presented algorithms which solve the TSP on degree-3 graphs in time $O^*(2^{n/3}) \approx O^*(1.260^n)$, and on degree-4 graphs in time $O^*((27/4)^{n/3}) \approx O^*(1.890^n)$ [8]. The algorithms are based on the standard classical technique of *backtracking*, an approach where a tree of partial solutions is explored to find a complete solution to a problem (see Section II for an introduction to this technique). Following subsequent improvements [9, 10], the best classical runtimes known for algorithms based on this general approach are $O^*(1.232^n)$ for degree-3 graphs [11], and $O^*(1.692^n)$ for degree-4 graphs [12], in each case due to Xiao and Nagamochi. All of these algorithms use polynomial space in n.

An algorithm of Bodlaender et al. [13] achieves a faster runtime of $O^*(1.219^n)$ for solving the TSP in degree-3 graphs, which is the best known; however, this algorithm uses exponential space. Similarly, an algorithm of Cygan et al. [14] solves the TSP in unweighted degree-4 graphs in $O^*(1.588^n)$ time and exponential space. Both of these algorithms use an approach known as cut-and-count, which is based on dynamic programming, so a quantum speedup is not known for either algorithm.

In the case where we have an upper bound L on the maximum edge cost in the graph, Björklund [15] gave a randomised algorithm which solves the TSP on arbitrary graphs in $O^*(1.657^n L)$ time and polynomial space, which is an improvement on the runtime of the Xiao-Nagamochi algorithm for degree-4 graphs when L is subexponential in n. Again, the techniques used in this algorithm do not seem obviously amenable to quantum speedup.

Here we use a recently developed quantum backtracking algorithm [16] to speed up the algorithms of Xiao and Nagamochi in order to find Hamiltonian cycles shorter than a given upper bound, if such cycles do exist. We run this algorithm several times, using binary search to specify what our upper bound should be, in order to find the shortest Hamiltonian cycle and solve the Travelling Salesman Problem. In doing so, we achieve a near-quadratic reduction in the runtimes:

Theorem 1. *There are bounded-error quantum algorithms which solve the TSP on degree-3 graphs in time $O^*(1.110^n \log L \log\log L)$ and on degree-4 graphs in time $O^*(1.301^n \log L \log\log L)$, where L is the maximum edge cost. The algorithms use* $\mathrm{poly}(n)$ *space.*

In this result and elsewhere in the paper, "bounded-error" means that the probability that the algorithm either doesn't find a Hamiltonian cycle when one exists or returns a non-optimal Hamiltonian cycle is at most 1/3. This failure probability can be reduced to δ, for arbitrary $\delta > 0$, by repeating the algorithm $O(\log 1/\delta)$ times. Also here and throughout the paper, log denotes log base 2. Note that the time complexity of our algorithms has some dependence on L, the largest edge cost in the input graph. However, this dependence is quite mild. For any graph whose edge costs are specified by w bits, $L \leq 2^w$. Thus terms of the form $\mathrm{polylog}(L)$ are at most polynomial in the input size.

Next, we show that degree-5 and degree-6 graphs can be dealt with via a randomised reduction to the degree-4 case.

Theorem 2. *There is a bounded-error quantum algorithm which solves the TSP on degree-5 and degree-6 graphs in time $O^*(1.680^n \log L \log\log L)$. The algorithm uses* $\mathrm{poly}(n)$ *space.*

We summarise our results in Table I.

A. Related work

Surprisingly little work has been done on quantum algorithms for the TSP. Dörn [17] proposed a quantum speedup for the TSP for degree-3 graphs by applying amplitude amplification [18] and quantum minimum finding [19] to Eppstein's algorithm, and stated a quadratic reduction in the runtime. However, we were not able to reproduce this result (see Section II below for a discussion).

Very recently, Mandrà, Guerreschi and Aspuru-Guzik [20] developed a quantum algorithm for finding a Hamiltonian cycle in time $O(2^{(k-2)n/4})$ in a graph where *every* vertex has degree k. Their approach reduces the problem to an Occupation problem, which they solve via a backtracking process accelerated by the quantum backtracking algorithm [16]. The bounds obtained from their algorithm are $O(1.189^n)$ for $k = 3$ and $O(1.414^n)$ for $k = 4$, in each case a bit slower than the runtimes of our algorithms; for $k \geq 5$, their algorithm has a slower runtime than Björklund's classical algorithm [15].

Martoňák, Santoro and Tosatti [21] explored the option of using quantum annealing to find approximate solutions for the TSP. Rather than solve the problem purely through quantum annealing, they simplify their Ising Hamiltonian for solving the TSP and use path-integral Monte Carlo [22] to run their model. While no bounds on run time or accuracy were strictly proven, they concluded by comparing their algorithm to simulated annealing via the Metropolis-Hastings algorithm [23] and the Kernighan-Lin algorithm for approximately solving the TSP [24]. Their results showed that ad hoc algorithms could perform better than general simulated or quantum annealing, but quantum annealing could outperform simulated annealing alone. However, they noted that simulated annealing could perform better than in their analysis if combined with local search heuristics [25].

Chen et al. [26] experimentally demonstrated a quantum annealing algorithm for the TSP. Their demonstration used a nuclear-magnetic-resonance quantum simulator to solve the problem for a graph with 4 vertices.

Degree	Quantum	Classical (poly space)	Classical (exp space)
3	$O^*(1.110^n \operatorname{polylog} L)$	$O^*(1.232^n)$ [11]	$O^*(1.219^n)$ [13]
4	$O^*(1.301^n \operatorname{polylog} L)$	$O^*(1.692^n)$ [12], $O^*(1.657^n L)$ [15]	$O^*(1.588^n)$ [14]
5, 6	$O^*(1.680^n \operatorname{polylog} L)$	$O^*(1.657^n L)$ [15]	—

TABLE I. Runtimes of our quantum algorithms for a graph of n vertices with maximum edge cost L, compared with the best classical algorithms known.

B. Organisation

We start by introducing the main technique we use, backtracking, and comparing it with amplitude amplification. Then, in Section III, we describe how this technique can be used to accelerate classical algorithms of Xiao and Nagamochi for graphs of degree at most 4 [11, 12]. In Section IV, we extend this approach to graphs of degree at most 6.

II. BACKTRACKING ALGORITHMS FOR THE TSP

Many of the most efficient classical algorithms known for the TSP are based around a technique known as backtracking. Backtracking is a general process for solving constraint satisfaction problems, where we have v variables and we need to find an assignment to these variables such that they satisfy a number of constraints. A naïve search across all possible assignments will be inefficient, but if we have some local heuristics then we can achieve better performance by skipping assignments that will definitely fail.

Suppose each variable can be assigned one value from $[d] := \{0, \ldots, d-1\}$. We define the set of partial assignments for v variables as $\mathcal{D} := (\{1, \ldots, v\}, [d])^j$, where $j \leq v$, with the first term denoting the variable to assign and the second denoting the value it is assigned. Using this definition for partial assignments, backtracking algorithms have two components. The first is a predicate, $P : \mathcal{D} \to \{\text{true}, \text{false}, \text{indeterminate}\}$, which takes a partial assignment and returns true if this assignment will definitely result in the constraints being satisfied regardless of how everything else is assigned, false if the assignment will definitely result in the constraints being unsatisfied, and indeterminate if we do not yet know. The second is a heuristic, $h : \mathcal{D} \to \{1, \ldots, v\}$, which takes a partial assignment and returns the next variable to assign.

The following simple recursive classical algorithm takes advantage of P and h to solve a constraint satisfaction problem. We take as input a partial assignment (initially, the empty assignment). We run P on this partial assignment; if the result is true then we return the partial assignment, and if it is false then we report that no solutions were found in this recursive call. We then call h on this partial assignment and find out what the next variable to assign is. For every value in $i \in [d]$ we can assign that variable, we recursively call the backtracking algorithm with i assigned to that variable. If one of the recursive calls returns a partial assignment then we return that assignment, otherwise we report that no solutions were found in this call. We can view this algorithm as exploring a tree whose vertices are labelled with partial assignments. The size of the tree determines the worst-case runtime of the algorithm, assuming that there is no assignment that satisfies all the constraints.

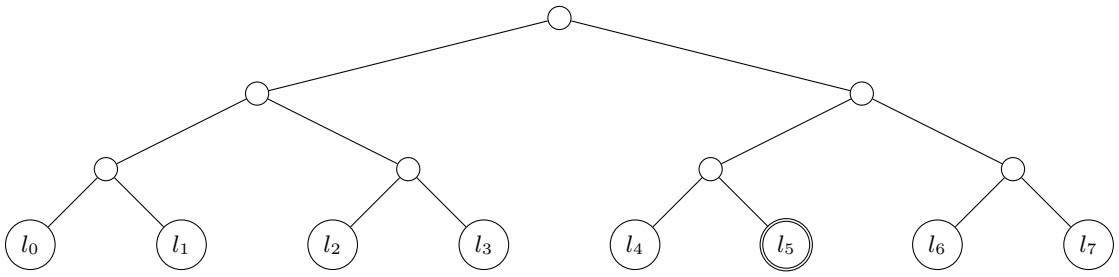
It is known that this backtracking algorithm can be accelerated using quantum techniques:

Theorem 3 (Montanaro [16]). *Let $\mathcal{A}$ be a backtracking algorithm with predicate P and heuristic h that finds a solution to a constraint satisfaction problem on v variables by exploring a tree of at most T vertices. There is a quantum algorithm which finds a solution to the same problem with failure probability δ with $O(\sqrt{T} v^{3/2} \log v \log(1/\delta))$ uses of P and h.*

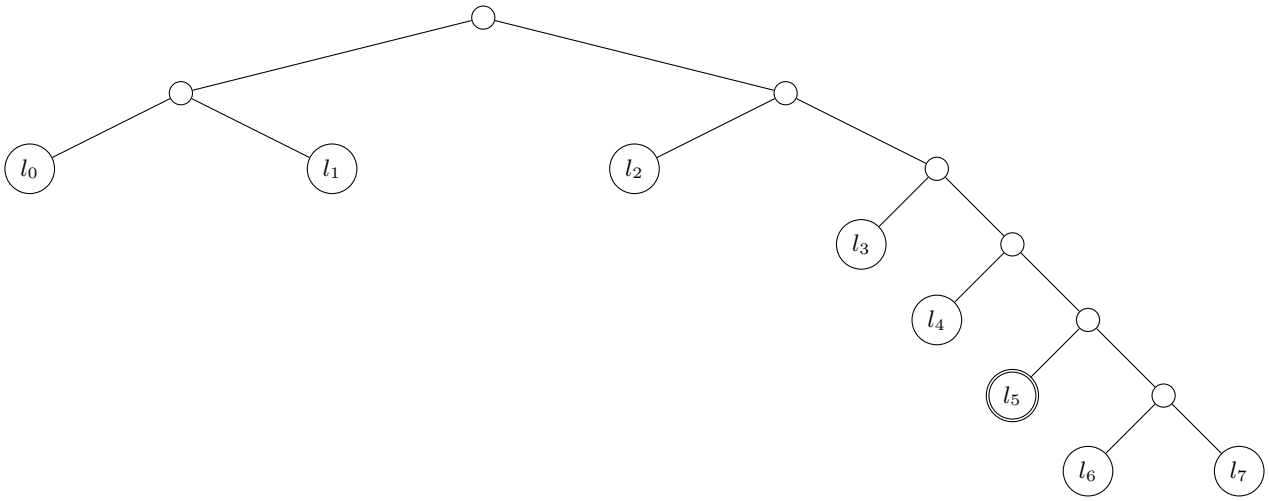
Montanaro's result is based on a previous algorithm by Belovs [27, 28], and works by performing a quantum walk on the backtracking tree to find vertices corresponding to assignments which satisfy the constraints. The reader familiar with [16] may note that the definition of the set of partial assignments $\mathcal{D}$ is different to that given there, in that it incorporates information about the ordering of assignments to variables. However, it is easy to see from inspection of the algorithm of [16] that this change does not affect the stated complexity of the algorithm.

It is worth noting that more standard quantum approaches such as amplitude amplification [18] will not necessarily achieve a quadratic speedup over the classical backtracking algorithm. Amplitude amplification requires access to a function $f : \{0,1\}^k \to \{\text{true}, \text{false}\}$ and a guessing function $\mathcal{G}$. If the probability of $\mathcal{G}$ finding a result $x \in \{0,1\}^k$ such that $f(x) = \text{true}$ is p, then amplitude amplification will succeed after $O(1/\sqrt{p})$ applications of f and $\mathcal{G}$ [18].

To apply amplitude amplification, we would need to access the leaves of the tree, as these are the points where the backtracking algorithm is certain whether or not a solution will be found. Thus, for each integer i, we would need to find a way of determining the i'th leaf l_i in the backtracking tree. In the case of a perfectly balanced tree, such as Fig. 1a, where every vertex in the tree is either a leaf or has exactly d branches descending from it, such a problem is easy: write i in base d and use each



(a) A perfectly balanced backtracking tree.



(b) An unbalanced backtracking tree.

FIG. 1. Example backtracking trees, where l_5 is a leaf corresponding to a solution to a constraint satisfaction problem. In the perfectly balanced case of Fig. 1a, each leaf can be associated with a 3-bit string corresponding to a path to that leaf. But in the unbalanced case of Fig. 1b, specifying a path to a leaf requires 6 bits.

digit of i to decide which branch to explore. But not all backtracking trees are perfectly balanced, such as in Fig. 1b. In these cases, finding leaf l_i is hard as we cannot be certain which branch leads to that leaf. Some heuristic approaches, by performing amplitude amplification on part of the tree, can produce better speedups for certain trees, but do not provide a general speedup on the same level as the quantum backtracking algorithm [16].

It is also worth understanding the limitations of the quantum backtracking algorithm, and why it cannot necessarily speed up all algorithms termed "backtracking algorithms" [16]. First, a requirement for the quantum algorithm is that decisions made in one part of the backtracking tree are independent of results in another part of the tree, which is not true of all classical algorithms, such as constraint recording algorithms [29]. Second, the runtime of the quantum algorithm depends on the size of the entire tree. Thus, to achieve a quadratic speedup over a classical algorithm, the algorithm must explore the whole backtracking tree, instead of stopping after finding the first solution or intelligently skipping branches such as in backjumping [29]. Therefore, it is important to check on a case-by-case basis whether classical backtracking algorithms can actually be accelerated using Theorem 3.

Another limitation of the quantum backtracking algorithm is that often there will be a metric $M : \mathcal{D} \to \mathbb{N}$ we want the backtracking algorithm to minimise while satisfying the other constraints. This is particularly relevant for the TSP, where the aim is to return the shortest Hamiltonian cycle. Classical backtracking algorithms can achieve this by recursively travelling down each branch of the tree to find results $D_1, \ldots, D_d \in \mathcal{D}$ and returning the result that minimises M. The quantum backtracking algorithm cannot perform this; it instead returns a solution selected randomly from the tree that satisfies the constraints. In order to achieve a quantum speedup when finding the result that minimises M, we can modify the original predicate to prune results which are greater than or equal to a given bound. We then repeat the algorithm in a binary search fashion, updating our bound based on whether or not a solution was found. This will find the minimum after repeating the quantum algorithm at most $O(\log M_{max})$ times, where

$$M_{max} = \max\{M(D) : D \in \mathcal{D}, P(D) = \text{true}\}.$$

We describe this binary search approach in more detail in Sec. III B.

The intuition behind why backtracking is a useful technique for solving the TSP is that we can attempt to build up a Hamiltonian cycle by determining for each

edge in the graph whether it should be included in the cycle ("forced"), or deleted from the graph. As we add more edges to the cycle, we may either find a contradiction (e.g. produce a non-Hamiltonian cycle) or reduce the graph to a special case that can be handled efficiently (e.g. a collection of disjoint cycles of four unforced edges). This can sometimes allow us to prune the backtracking tree substantially.

To analyse the performance of backtracking algorithms for the TSP, a problem size measure is often defined that is at least 0 and at most n (e.g. the number of vertices minus the number of forced edges). Note that if there are more than n forced edges then it is impossible to form a Hamiltonian cycle that includes every forced edge, so the number of forced edges is at most n. At the start of the backtracking algorithm, there are no forced edges so the problem size is n. Each step of the backtracking algorithm reduces the problem size until the size is 0, at which point either the n forced edges form a Hamiltonian cycle or a Hamiltonian cycle that includes every forced edge cannot be found. A quasiconvex program can be developed based on how the backtracking algorithm reduces the problem size. Solving this quasiconvex problem produces a runtime in terms of the problem size, which can be re-written in terms of n due to the problem size being at most n.

It was proposed by Dörn [17] that amplitude amplification could be applied to speed up the runtime of Eppstein's algorithm for the TSP on degree-3 graphs [8] from $O^*(2^{n/3})$ to $O^*(2^{n/6})$. Amplitude amplification can be used in this setting by associating a bit-string with each sequence of choices of whether to force or delete an edge, and searching over bit-strings to find the shortest valid Hamiltonian cycle. However, as suggested by the general discussion above, a difficulty with this approach is that some branches of the recursion, as shown in Fig. 2, only reduce the problem size by 2 (as measured by the number of vertices n, minus the number of forced edges). The longest branch of the recursion can, as a result, be more than $n/3$ levels deep. In the worst case, this depth could be as large as $n/2$ levels. Specifying the input to the checking function f could then require up to $n/2$ bits, giving a search space of size $O(2^{n/2})$. Under these conditions, searching for the solution via amplitude amplification could require up to $O^*(2^{n/4})$ time in the worst case. To yield a better runtime, we must take more of an advantage of the structure of our search space to avoid instances which will definitely not succeed.

The same issue with amplitude amplification applies to other classical algorithms for the TSP which are based on backtracking [11, 12]. In the case of the Xiao-Nagamochi algorithm for degree-3 graphs, although the overall runtime bound proven for the problem means that the number of vertices in the tree is $O(2^{3n/10})$, several of the branching vectors used in their analysis have branches that reduce the problem size by less than 10/3, leading to a branch in the tree that could be more than $3n/10$ levels deep.

III. QUANTUM SPEEDUPS FOR THE TRAVELLING SALESMAN PROBLEM ON BOUNDED-DEGREE GRAPHS

Our algorithms are based on applying the quantum algorithm for backtracking (Theorem 3) to Xiao and Nagamochi's algorithm [11]. Before describing our algorithms, we need to introduce some terminology from [11] and describe their original algorithm. The algorithm, and its analysis, are somewhat involved, so we omit details wherever possible.

A. The algorithm of Xiao and Nagamochi

A graph G is k-edge connected if there are k edge-disjoint paths between every pair of vertices. An edge in G is said to be forced if it must be included in the final tour, and unforced otherwise. The set of forced edges is denoted F, and the set of unforced edges is denoted U. An induced subgraph of unforced edges which is maximal and connected is called a U-component. If a U-component is just a single vertex, then that U-component is trivial. A maximal sequence $\mathcal{C}$ of edges in a U-component H is called a circuit if either:

- $\mathcal{C} = \{xy\}$ and there are three edge-disjoint paths from x to y,
- or $\mathcal{C} = \{c_0, c_1, \ldots, c_{m-1}\}$ such that for $0 \leq i < m-1$, there is a subgraph B_i of H such that the only two unforced edges incident to B_i are c_i and c_{i+1}.

A circuit is reducible if subgraph B_i for some i is incident to only two edges. In order for B_i to be reached, both edges incident to B_i need to be forced. Forcing one edge in the circuit then means that the other edges can be either forced or removed. The polynomial time and space process by Xiao and Nagamochi to reduce circuits, by forcing and removing alternating edges in the circuit, is known as the *circuit procedure* [11].

Note that each edge can be in at most one circuit. If two distinct circuits $\mathcal{C}, \mathcal{C}'$ shared an edge e_i, then there are two possibilities. The first is that there is a subgraph B_i incident to unforced edges $e_i \in \mathcal{C} \cap \mathcal{C}', e_{i+1} \in \mathcal{C} - \mathcal{C}', e_j \in \mathcal{C}' - \mathcal{C}$. In this case, B_i is incident to more than two unforced edges, so neither $\mathcal{C}$ nor $\mathcal{C}'$ are circuits, which is a contradiction.

The second is that there is some edge e_i which is incident to distinct subgraphs B_i, B_i' related to $\mathcal{C}, \mathcal{C}'$, respectively. Circuits are maximal sequences, so it cannot be the case that B_i is a subgraph of B_i', otherwise $\mathcal{C}' \subseteq \mathcal{C}$. Now we consider the subgraphs $B_i \cap B_i'$ and $B_i - B_i'$, which must be connected by unforced edges as they are both subgraphs of B_i. These unforced edges are incident to B_i', which is a contradiction as they are not part of $\mathcal{C}'$.

Let X be a subgraph. We define $\mathrm{cut}(X)$ to be the set of edges that connect X to the rest of the graph.

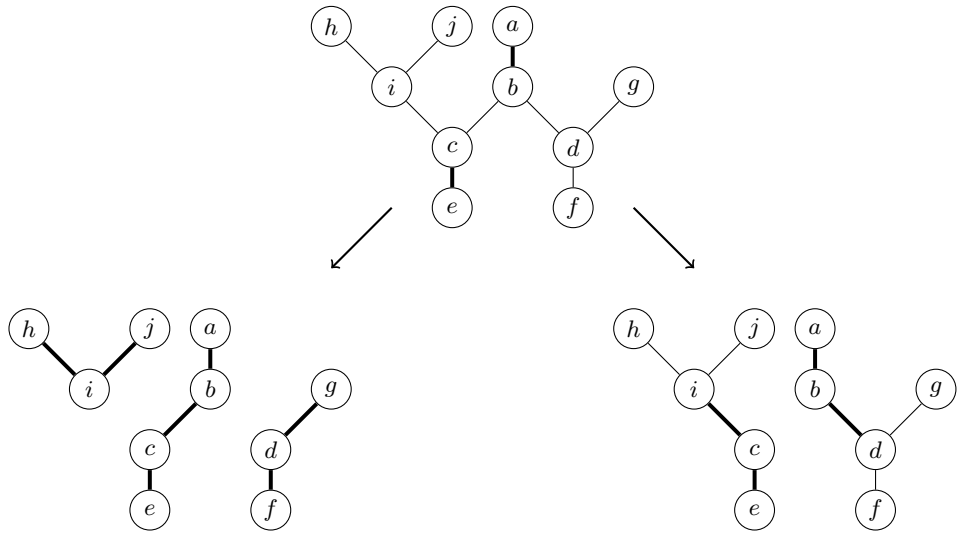


FIG. 2. An instance of the recursive step in Eppstein's backtracking algorithm for the TSP [8] for a subgraph of a larger graph G, with forced edges displayed in bold and branching on edge bc. If we force bc, then b and c are both incident to two forced edges, so bd and ci cannot be part of the Hamiltonian cycle and can be removed from the graph. After these edges are removed, vertices i and d are both of degree 2, so in order to reach those vertices the edges hi, ij, df and dg must also be included in the Hamiltonian cycle. So forcing bc has overall added five edges to the Hamiltonian cycle. On the other hand, if we remove edge bc, we find that b and c are vertices of degree 2, so edges bd and ci must be part of the Hamiltonian cycle. Thus we have only added two more edges to the Hamiltonian cycle.

If $|\text{cut}(X)| = 3$, then we say that X is 3-cut reducible. It was shown by Xiao and Nagamochi [11] that, if X is 3-cut reducible, X can be replaced with a single vertex of degree 3 with outgoing edges weighted such that the length of the shortest Hamiltonian cycle is preserved.

The definition of 4-cut reducible is more complex. Let X be a subgraph such that $\text{cut}(X) \subseteq F$ and $|\text{cut}(X)| = 4$. A solution to the TSP would have to partition X into two disjoint paths such that every vertex in X is in one of the two paths. If x_1, x_2, x_3 and x_4 are the four vertices in X incident to the four edges in $\text{cut}(X)$, then there are three ways these paths could start and end:

- $x_1 \leftrightarrow x_2$ and $x_3 \leftrightarrow x_4$,
- $x_1 \leftrightarrow x_3$ and $x_2 \leftrightarrow x_4$,
- or $x_1 \leftrightarrow x_4$ and $x_2 \leftrightarrow x_3$.

We say that X is 4-cut reducible if for at least one of the above cases it is impossible to create two disjoint paths in X that include all vertices in X. Xiao and Nagamochi defined a polynomial time and space process for applying the above reductions, known as 3/4-*cut reduction* [11].

A set of edges $\{e_i\}$ are *parallel* if they are incident to the same vertices (note that here we implicitly let G be a multigraph; these may be produced in intermediate steps of the algorithm). If there are only two vertices in the graph, then the TSP can be solved directly by forcing the shortest two edges. Otherwise if at least one of the edges is not forced, then we can reduce the problem by removing the longer unforced edges until the vertices are only adjacent via one edge. This is the process Xiao and Nagamochi refer to as *eliminating parallel edges* [11].

Finally, a graph is said to satisfy the parity condition if every U-component is incident to an even number of forced edges and for every circuit $\mathcal{C}$, an even number of the corresponding subgraphs B_i satisfy that $|\text{cut}(B_i) \cap F|$ is odd.

We are now ready to describe Xiao and Nagamochi's algorithm. The algorithm takes as input a graph $G = (V, E)$ and a set of forced edges $F \subseteq E$ and returns the length of the shortest Hamiltonian cycle in G containing all the edges in F, if one exists.

The algorithm is based on four subroutines: *eliminating parallel edges*, the *3/4-cut reduction*, *selecting a good circuit* and the *circuit procedure*, as well as the following lemma:

Lemma 1 (Eppstein [8])**.** *If every U-component in a graph G is trivial or a component of a 4-cycle, then a minimum cost tour can be found in polynomial time.*

We will not define the subroutines here in any detail; for our purposes, it is sufficient to assume that they all run in polynomial time and space. The circuit procedure for a circuit $\mathcal{C}$ begins by either adding an edge $e \in \mathcal{C}$ to F or deleting it from the graph, then performing some other operations. "Branching on a circuit $\mathcal{C}$ at edge $e \in \mathcal{C}$" means generating two new instances from the current instance by applying each of these two variants of the circuit procedure starting with e.

The Xiao-Nagamochi algorithm, named TSP3, proceeds as follows, reproduced from [11]:

1. **If** G is not 2-edge-connected or the instance violates the parity condition, then return ∞;

2. **Elseif** there is a reducible circuit $\mathcal{C}$, then return TSP3(G', F') for an instance (G', F') obtained by applying the circuit procedure on $\mathcal{C}$ started by adding a reducible edge in $\mathcal{C}$ to F;

3. **Elseif** there is a pair of parallel edges, then return TSP3(G', F') for an instance (G', F') obtained by applying the reduction rule of eliminating parallel edges;

4. **Elseif** there is a 3/4-cut reducible subgraph X containing at most eight vertices, then return TSP3(G', F') for an instance (G', F') obtained by applying the 3/4-cut reduction on X;

5. **Elseif** there is a U-component H that is neither trivial nor a 4-cycle, then select a good circuit $\mathcal{C}$ in H and return $\min\{\text{TSP3}(G_1, F_1), \text{TSP3}(G_2, F_2)\}$, where (G_1, F_1) and (G_2, F_2) are the two resulting instances after branching on $\mathcal{C}$;

6. **Else** [each U-component of the graph is trivial or a 4-cycle], solve the problem directly in polynomial time by Lemma 1 and return the cost of an optimal tour.

Step 1 of the algorithm checks that the existence of a Hamiltonian cycle is not ruled out, by ensuring that that there are at least two disjoint paths between any pair of vertices and that the graph satisfies the parity condition. Step 2 reduces any reducible circuit by initially forcing one edge and then alternately removing and forcing edges. Step 3 removes any parallel edges from the graph, and step 4 removes any circuits of three edges as well as setting up circuits of four edges so that all edges incident to them are forced. Step 5 is the recursive step, branching on a good circuit by either forcing or removing an edge in the circuit and then applying the circuit procedure. The algorithm continues these recursive calls until it either finds a Hamiltonian cycle or $G \setminus F$ is a collection of single vertices and cycles of length 4, all of which are disjoint from one another, at which point the problem can be solved in polynomial time via step 6.

Xiao and Nagamochi looked at how the steps of the algorithm, and step 5 in particular as the branching step, reduced the size of the problem for different graph structures. From this they derived a quasiconvex program corresponding to 19 branching vectors, each describing how the problem size is reduced at the branching step in different circumstances. Analysis of this quasiconvex program showed that the algorithm runs in $O^*(2^{3n/10})$ time and polynomial space [11].

B. Quantum speedup of the Xiao-Nagamochi algorithm

Here we describe how we apply the quantum backtracking algorithm to the Xiao-Nagamochi algorithm. It is worth noting that the quantum backtracking algorithm will not necessarily return the shortest Hamiltonian cycle, but instead returns a randomly selected Hamiltonian cycle that it found. Adding constraints on the length of the Hamiltonian cycles to our predicate and running the quantum backtracking algorithm multiple times will allow us to find a solution to the TSP.

The first step towards applying the quantum backtracking algorithm is to define the set of partial assignments. A partial assignment will be a list of edges in G ordered by when they are assigned in the backtracking algorithm and paired with whether the assignment was to force or remove the edge. The assignment is denoted $A \in (\{1, \ldots, m\}, \{\text{force}, \text{remove}\})^j$, where $j \leq m$. We have $m \leq 3n/2$ as G is degree-3.

The quantum approach to backtracking requires us to define a predicate P and heuristic h, each taking as input a partial assignment. Our predicate and heuristic make use of a reduction function, introduced in [11], as a subroutine; this function is described in the next subsection. However it may be worth noting that the algorithm uses the original graph G, and partial assignments of it at each stage.

Firstly, we describe the P function, which takes a partial assignment $A = ((e_1, A_1), \ldots, (e_j, A_j))$ as input:

1. Using the partial assignment A, apply the reduction function to (G, F) to get (G', F').

2. If G' is not 2-edge-connected or fails the parity condition, then return false.

3. If every U-component in G' is either trivial or a 4-cycle, then return true.

4. Return indeterminate.

Step 2 matches step 1 of Xiao and Nagamochi's algorithm. Step 3 is where the same conditions are met as in step 6 of Xiao and Nagamochi's algorithm, where a shortest length Hamiltonian cycle is guaranteed to exist and can be found in polynomial time classically via Lemma 1. Step 4 continues the branching process, which together with how the circuit is picked by h and step 2(c) of the reduction function (qv) matches step 5 of Xiao and Nagamochi.

The h function is described as follows, taking as input a partial assignment $A = ((e_1, A_1), \ldots, (e_j, A_j))$ of the edges of G:

1. Using the partial assignment A, apply the reduction function to (G, F) to get (G', F').

2. Select a U-component in G' that is neither trivial nor a cycle of length 4. Select a circuit $\mathcal{C}$ in

that component that fits the criteria of a "good" circuit [11], then select an edge $e'_i \in \mathcal{C}$.

3. Return an edge in G corresponding to e'_i (if there is more than one, choosing one arbitrarily).

Step 2 applies step 5 of Xiao and Nagamochi's algorithm, by selecting the next circuit to branch on and picking an edge in that circuit. If the reduced version of the graph results in h picking an edge corresponding to multiple edges in the original graph, step 3 ensures that we only return one of these edges to the backtracking algorithm, as step 2(b) of the reduction function will ensure that every edge in the original graph corresponding to an edge in the reduced graph will be consistently forced or removed. The rest of the circuit will be forced or removed by step 2(c) of the reduction function.

We can now apply the backtracking algorithm (Theorem 3) to P and h to find a Hamiltonian cycle. We will later choose its failure probability δ to be sufficiently small that we can assume that it always succeeds, i.e. finds a Hamiltonian cycle if one exists, and otherwise reports that one does not exist. At the end of the algorithm, we will receive either the information that no assignment was found, or a partial assignment. By applying the reduction steps and the partial assignments, we can reconstruct the graph at the moment our quantum algorithm terminated, which will give a graph such that every U-component is either trivial or a 4-cycle. We then construct and return the full Hamiltonian cycle in polynomial time using step 6 of Xiao and Nagamochi's algorithm [11].

To solve the TSP, we need to find the shortest Hamiltonian cycle. This can be done as follows. First, we run the backtracking algorithm. If the backtracking algorithm does not return a Hamiltonian cycle then we report that no Hamiltonian cycle was found. Otherwise after receiving Hamiltonian cycle Γ with length L_Γ, we create variables $\ell \leftarrow 0$ & $u \leftarrow L_\Gamma$ and modify P to return false if

$$\sum_{e_{i,j} \in F} c_{ij} \geq \lceil (\ell + u)/2 \rceil.$$

If no cycle is found after running the algorithm again, we set $\ell \leftarrow \lceil (\ell + u)/2 \rceil$ and repeat. Otherwise, upon receiving Hamiltonian cycle Γ' with total cost $L_{\Gamma'}$, we set $u \leftarrow L_{\Gamma'}$ and repeat. We continue repeating until ℓ and u converge, at which point we return the Hamiltonian cycle found by the algorithm. In the worst case scenario, where the shortest cycle is found during the first run of the backtracking algorithm, this algorithm matches a binary search. So the number of repetitions of the backtracking algorithm required to return the shortest Hamiltonian cycle is at most $O(\log L')$, where

$$L' = \sum_{i=1}^{n} \max\{c_{ij} : j \in \{1, \ldots, n\}\} \tag{1}$$

is an upper bound on the total cost of any Hamiltonian cycle in the graph.

C. The reduction function

Finally, we describe the reduction function, which takes the original graph G and partial assignment A, and applies the partial assignment to this graph in order to reduce it to a smaller graph G' with forced edges F'. This reduction might mean that forcing or removing a single edge in G' would be akin to forcing several edges in G. For example, let X be a 3-reducible subgraph of at most 8 vertices with $\mathrm{cut}(X) = \{ax_1, bx_2, cx_3\}$ for vertices $x_1, x_2, x_3 \in V(X)$. The 3/4-cut reduction reduces X to a single vertex $x \in G'$ with edges ax, bx, cx. If the edges ax and bx are forced, this is equivalent to forcing every edge in $\Pi \cup \{ax_1, bx_2\}$, where Π is the shortest path that starts at x_1, visits every vertex in X exactly once, and ends at x_2. As we need to solve the problem in terms of the overall graph G and not the reduced graph G', our assigned variables need to correspond to edges in G. To do this, our heuristic includes a step where if the edge selected in G' corresponds to multiple edges in G, we simply select one of the corresponding edges in G to return. Likewise, if the next edge in our partial assignment is one of several edges in G corresponding to a single edge in G', we apply the same assignment to all of the other corresponding edges in G.

The reduction function works as follows, using reductions and procedures from Xiao and Nagamochi [11]:

1. Create a copy of the graph $G' \leftarrow G$ and set of forced edges $F' \leftarrow \emptyset$.
2. For each $i = 1, \ldots, j$:
   (a) Repeat until none of the cases apply:
      i. If G' contains a reducible circuit $\mathcal{C}$, then apply the circuit procedure to $\mathcal{C}$.
      ii. If G' contains parallel edges, then apply the reduction rule of eliminating parallel edges.
      iii. If G' contains a subgraph X of at most 8 vertices such that X is 3/4-cut reducible, then apply the 3/4-cut reduction to X.
   (b) Apply assignment (e_i, a_i) to (G', F') by adding edge e_i to F' if $a_i =$ force, or deleting edge e_i from G' if $A_i =$ remove. If edge e_i is part of a set of edges corresponding to a single edge in G', apply the same assignment to all edges in G which correspond to the same edge in G' by adding them all to F' if $a_i =$ force, or deleting them all from G' if $a_i =$ remove.
   (c) Apply the circuit procedure to the rest of the circuit containing edge e_i.
3. Run step 2(a) again.
4. Return (G', F').

Step 2(a)i recreates step 2 from Xiao and Nagamochi's original algorithm by applying the circuit procedure

where possible. Step 2(a)ii recreates step 3 of the original algorithm by applying the reduction of parallel edges. Step 2(a)iii recreates step 4 of the original algorithm via the 3/4-cut reduction. Step 2(b) applies the next step of the branching that has been performed so far, to ensure that the order in which the edges are forced is the same as in the classical algorithm. Step 2(c) corresponds to branching on a circuit at edge e_i. Finally, step 3 checks whether or not the graph can be reduced further by running the reduction steps again.

One might ask if an edge could be part of two circuits, in which case our algorithm would fail as it would not be able to reduce the circuit. However, as discussed in Sec. III A, any edge can only be part of at most one circuit.

D. Analysis

Steps 2(a)i-iii of the reduction algorithm can be completed in polynomial time [11]. All of these steps also reduce the size of a problem by at least a constant amount, so only a polynomial number of these steps are needed. Step 2(b) is constant time and step 2(c) can be run in polynomial time as the circuit is now reducible. All steps are only repeated $O(m)$ times, so the whole reduction algorithm runs in polynomial time in terms of m.

Steps 2 and 3 of the h subroutine run in polynomial time as searching for a good circuit in a component can be done in polynomial time [11]. Likewise, steps 2 and 3 of the P function involve looking for certain structures in the graph that can be found in polynomial time. As a result, the runtimes for the P and h functions are both polynomial in m.

By Theorem 3, the number of calls to P and h we make in order to find a Hamiltonian cycle with failure probability δ is $O(\sqrt{T}\,\mathrm{poly}(m)\log(1/\delta))$, where T is the size of the backtracking tree, which in our case is equal to the number of times the Xiao-Nagamochi algorithm branches on a circuit. P and h both run in polynomial time and as a result can be included in the $\mathrm{poly}(m)$ term of the runtime. Because $m \leq 3n/2$, the polynomial term in this bound is also polynomial in terms of n.

The behaviour of the P and h subroutines is designed to reproduce the behaviour of Xiao and Nagamochi's TSP3 algorithm [11]. It is shown in [11, Theorem 1] that this algorithm is correct, runs in time $O^*(2^{3n/10})$ and uses polynomial space. As the runtime of the TSP3 algorithm is an upper bound on the number of branching steps it makes, the algorithm branches on a circuit $O^*(2^{3n/10})$ times. Therefore, the quantum backtracking algorithm finds a Hamiltonian cycle, if one exists, with failure probability at most δ in time $O^*(2^{3n/20}\log(1/\delta)) \approx O^*(1.110^n\log(1/\delta))$ and polynomial space.

Finding the shortest Hamiltonian cycle requires repeating the algorithm $O(\log L')$ times, where L' is given in Equation 1. By using a union bound over all the runs of the algorithm, to ensure that all runs succeed with high probability it is sufficient for the failure probability δ of each run to be at most $O(1/(\log L'))$. From this we obtain the following result, proving the first part of Theorem 1:

Theorem 4. *There is a bounded-error quantum algorithm which solves the TSP on degree-3 graphs in time $O^*(1.110^n \log L \log\log L)$, where L is the maximum edge cost. The algorithm uses* $\mathrm{poly}(n)$ *space.*

Note that we have used the bound $L' \leq nL$, where the extra factor of n is simply absorbed into the hidden $\mathrm{poly}(n)$ term.

IV. EXTENDING TO HIGHER-DEGREE GRAPHS

We next consider degree-k graphs for $k \geq 4$. We start with degree-4 graphs by applying the quantum backtracking algorithm to another algorithm by Xiao and Nagamochi [12]. We then extend this approach to graphs of higher degree by reducing the problem to degree-4 graphs.

A. Degree-4 graphs

Here we will show the following, which is the second part of Theorem 1:

Theorem 5. *There is a bounded-error quantum algorithm which solves the TSP for degree-4 graphs in time $O^*(1.301^n \log L \log\log L)$, where L is the maximum edge cost. The algorithm uses* $\mathrm{poly}(n)$ *space.*

As the argument is very similar to the degree-3 case, we only sketch the proof.

Proof sketch. Xiao and Nagamochi's algorithm for degree-4 graphs works in a similar way to their algorithm for degree-3 graphs: The graph is reduced in polynomial time by looking for specific structures in the graph and then picking an edge in the graph to branch on. We apply the quantum backtracking algorithm as before, finding a Hamiltonian cycle with failure probability δ in $O^*(1.301^n\log(1/\delta))$ time. We then use binary search to find the shortest Hamiltonian cycle after $O(\log L)$ repetitions of the algorithm, rejecting if the total length of the forced edges is above a given threshold. To achieve overall failure probability 1/3, the algorithm runs in $O^*(1.301^n \log L \log\log L)$ time. □

B. Degree-5 and degree-6 graphs

To deal with degree-5 and degree-6 graphs, we reduce them to the degree-4 case. The complexity of the two cases turns out to be the same; however, for clarity we consider each case separately.

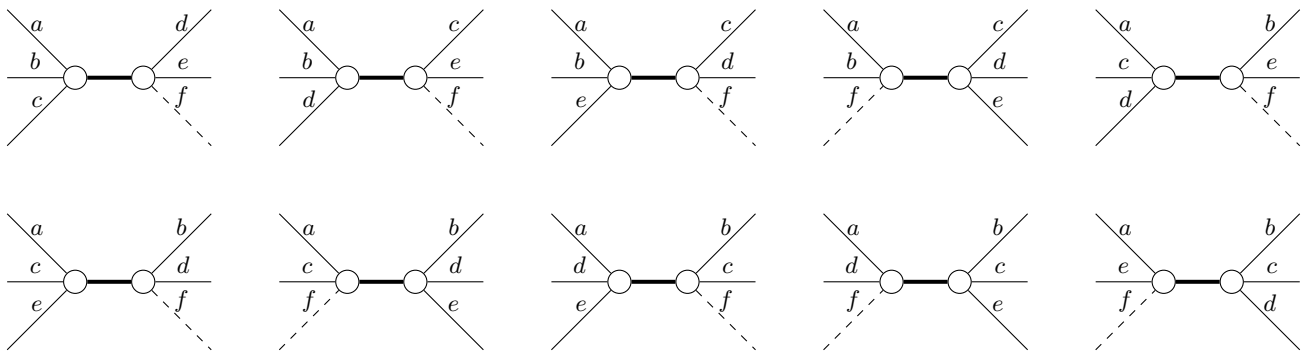


FIG. 3. Breaking a vertex of degree 5 or 6 into two lower-degree vertices. In the degree-5 case, dashed edge f is not present and the vertex is split into one vertex of degree 3 and another of degree 4 connected by a forced edge in bold. In the degree-6 case, dashed edge f is present and the vertex is split into two vertices of degree 4 connected by a forced edge. If edges a and b are included in the original graph's shortest Hamiltonian cycle, then they must not be adjacent to one another in the final graph. This holds in six of the ten ways of splitting the vertex.

Theorem 6. *There is a bounded-error quantum algorithm which solves the TSP for degree-5 graphs in time* $O^*(1.680^n \log L \log\log L)$.

Proof. Our algorithm works by splitting each vertex of degree 5 into one vertex of degree 3 and another of degree 4 connected by a forced edge. The forced edges can be included in our quantum algorithm by modifying step 1 of the reduction function so that F' contains all the forced edges created by splitting a vertex of degree-5 into two vertices connected by a forced edge. Once all degree-5 vertices are split this way, we run the degree-4 algorithm. It is intuitive to think that this splitting of the vertices could increase the runtime complexity of the degree-4 algorithm, due to n being larger. However, the addition of a forced edge incident to every new vertex means that we do not need to create more branches in the backtracking tree in order to include the new vertex in the Hamiltonian cycle. As a result, the time complexity of the degree-4 algorithm will remain the same.

There are 10 unique ways of splitting a vertex of degree 5 into one vertex of degree 3 and another of degree 4 connected by a forced edge. These ten ways of splitting the vertex are shown in Fig. 3 for a vertex incident to edges a, b, c, d, e. Without loss of generality, let a and b be the two edges which are part of the Hamiltonian cycle. In order for a and b to also be part of the Hamiltonian cycle in the degree-4 graph produced, a and b cannot be adjacent to one another. Looking at Fig. 3, the split is successful in six of the ten ways of splitting the vertex.

If there are f vertices of degree 5, then there are 10^f possible ways of splitting all such vertices, of which 6^f will give the correct solution to the TSP. We can apply Dürr and Høyer's quantum algorithm for finding the minimum [19] to find a splitting that leads to a shortest Hamiltonian cycle, or reporting if no cycle exists, after $O((10/6)^{f/2})$ repeated calls to the degree-4 algorithm. To ensure that the failure probability of the whole algorithm is at most 1/3, we need to reduce the failure probability of the degree-4 algorithm to $O((10/6)^{-f/2})$, which can be achieved by repeating it $O(f)$ times and returning the minimum-length tour found. The overall runtime is thus

$$O^*\left(\left(\frac{10}{6}\right)^{\frac{f}{2}} 1.301^n \log L \log\log L\right)$$
$$= O^*(1.680^n \log L \log\log L).$$

□

It is also possible to split a vertex of degree 5 into three vertices of degree 3 connected by two forced edges. There are 15 ways of performing this splitting, of which 6 will succeed. Applying the degree-3 algorithm to these reduced graphs finds a runtime of

$$O^*\left(\left(\frac{15}{6}\right)^{\frac{f}{2}} 1.110^n \log L \log\log L\right)$$
$$= O^*(1.754^n \log L \log\log L)$$

which performs worse than Theorem 6. We next turn to degree-6 graphs, for which the argument is very similar.

Theorem 7. *There is a quantum algorithm which solves the TSP for degree-6 graphs with failure probability* 1/3 *in time* $O^*(1.680^n \log L \log\log L)$.

Proof. We can extend the idea of Theorem 6 to degree-6 graphs by splitting vertices of degree 6 into two vertices of degree 4 connected by a forced edge. Because the degree of both new vertices is 4, there are $\binom{6}{3}/2 = 10$ unique ways of partitioning the edges, of which 4 will fail. We show this in Fig. 3 by including the dashed edge f as the sixth edge. The overall runtime is the same as the degree-5 case. □

C. Degree-7 graphs

We finally considered extending the algorithm to degree-7 graphs by partitioning degree-7 vertices into one of degree 5 and another of degree 4, connected by a forced edge. We can split a vertex of degree 7 into a vertex of degree 4 and another of degree 5 in $\binom{7}{4} = 35$ ways, of which $\binom{7-2}{4-2} + \binom{7-2}{3-2} = 15$ will not preserve the shortest Hamiltonian cycle. We then use the same process as for the degree-5 and degree-6 case, halting after $O((35/20)^{k/2})$ iterations and returning either the shortest Hamiltonian cycle found or reporting if no Hamiltonian cycle exists. From this, our overall runtime is

$$\begin{aligned} &O^*\left(\left(\frac{35}{20}\right)^{k/2} 1.680^n \log L \log\log L\right) \\ =&O^*(2.222^n \log L \log\log L). \end{aligned}$$

This is the point where we no longer see a quantum speedup over the fastest classical algorithms using this approach, as classical algorithms such as those of Held-Karp [4] and Björklund et al. [5] run in $O^*(2^n)$ and $O^*(1.984^n)$ time, respectively.

V. NOTE ADDED

Following the completion of this work, Andris Ambainis informed us of two new related results in this area. First, a quantum backtracking algorithm whose runtime depends only on the number of tree vertices visited by the classical backtracking algorithm, rather than the whole tree [30]. This alleviates one, though not all, of the limitations of the backtracking algorithm discussed in Section II. Second, a new quantum algorithm for the general TSP based on accelerating the dynamic programming algorithm [31]. The algorithm's runtime is somewhat worse than ours for graphs of degree at most 6, and it uses exponential space; but it works for any graph, rather than the special case of bounded-degree graphs considered here.

ACKNOWLEDGMENTS

DJM was supported by the Bristol Quantum Engineering Centre for Doctoral Training, EPSRC grant EP/L015730/1. AM was supported by EPSRC Early Career Fellowship EP/L021005/1. We would like to thank Andris Ambainis for bringing refs. [30, 31] to our attention.

[1] M. Grötschel, M. Jünger, and G. Reinelt, Zeitschrift für Operations Research **35**, 61 (1991).
[2] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization* (Wiley-Interscience Series in Discrete Mathematics, 1985).
[3] L. K. Grover, in *Proc. 28th Annual ACM Symposium on Theory of Computing (STOC'96)* (1996) pp. 212–219.
[4] M. Held and R. M. Karp, Journal of the Society for Industrial and Applied Mathematics **10**, 196 (1962).
[5] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto, in *Proc. 35th International Conference on Automata, Languages and Programming (ICALP'08)* (2008) pp. 198–209.
[6] J. D. C. Little, K. G. Murty, D. W. Sweeney, and C. Karel, Operations Research **11**, 972 (1963).
[7] M. Padberg and G. Rinaldi, SIAM Rev. **33**, 60 (1991).
[8] D. Eppstein, Journal of Graph Algorithms and Applications **11**, 61 (2007).
[9] K. Iwama and T. Nakashima, in *Proc. 13th Annual International Computing and Combinatorics Conference (COCOON'07)* (2007) pp. 108–117.
[10] M. Likiewicz and M. R. Schuster, Journal of Discrete Algorithms **27**, 1 (2014).
[11] M. Xiao and H. Nagamochi, Algorithmica **74**, 713 (2016).
[12] M. Xiao and H. Nagamochi, Theory of Computing Systems **58**, 241 (2016).
[13] H. L. Bodlaender, M. Cygan, S. Kratsch, and J. Nederlof, Information and Computation **243**, 86 (2015).
[14] M. Cygan, J. Nederlof, M. Pilipczuk, M. Pilipczuk, J. van Rooij, and J. Wojtaszczyk, in *Proc. IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS'11)* (2011) pp. 150–159, arXiv:1103.0534.
[15] A. Björklund, SIAM Journal on Computing **43**, 280 (2014).
[16] A. Montanaro, "Quantum walk speedup of backtracking algorithms," (2015), arXiv:1509.02374.
[17] S. Dörn, in *Proc. CIE* (2007) pp. 123–131.
[18] G. Brassard and P. Høyer, "An exact quantum polynomial-time algorithm for Simon's problem," (1997), arXiv:quant-ph/9704027.
[19] C. Dürr and P. Høyer, "A quantum algorithm for finding the minimum," (1996), arXiv:quant-ph/9607014.
[20] S. Mandrà, G. G. Guerreschi, and A. Aspuru-Guzik, New Journal of Physics **18**, 073003 (2016).
[21] R. Martoňák, G. E. Santoro, and E. Tosatti, Phys. Rev. E **70**, 057701 (2004).
[22] J. A. Barker, The Journal of Chemical Physics **70**, 2914 (1979).
[23] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, The Journal of Chemical

Physics **21**, 1087 (1953).
[24] B. W. Kernighan and S. Lin, The Bell System Technical Journal **49**, 291 (1970).
[25] O. C. Martin and S. W. Otto, Annals of Operations Research **63**, 57 (1996).
[26] H. Chen, X. Kong, B. Chong, G. Qin, X. Zhou, X. Peng, and J. Du, Phys. Rev. A **83**, 032314 (2011).
[27] A. Belovs, "Quantum walks and electric networks," (2013), arXiv:1302.3143.
[28] A. Belovs, A. Childs, S. Jeffery, R. Kothari, and F. Magniez, in *Proc. 40th International Conference on Automata, Languages and Programming (ICALP'13)* (2013) pp. 105–122.
[29] R. Dechter, Artificial Intelligence **41**, 273 (1990).
[30] A. Ambainis and M. Kokainis, "Quantum algorithm for tree size estimation, with applications to backtracking and 2-player games," (2016), forthcoming.
[31] A. Ambainis and M. Kokainis, personal communication (2016).