Quantum Speedups for Treewidth

Vladislavs Kļevickis
Centre for Quantum Computer Science, Faculty of Computing, University of Latvia, Riga, Latvia

Krišjānis Prūsis
Centre for Quantum Computer Science, Faculty of Computing, University of Latvia, Riga, Latvia

Jevgēnijs Vihrovs
Centre for Quantum Computer Science, Faculty of Computing, University of Latvia, Riga, Latvia

Abstract

In this paper, we study quantum algorithms for computing the exact value of the treewidth of a graph. Our algorithms are based on the classical algorithm by Fomin and Villanger (Combinatorica 32, 2012) that uses $O(2.616^n)$ time and polynomial space. We show three quantum algorithms with the following complexity, using QRAM in both exponential space algorithms:

- $O(1.618^n)$ time and polynomial space;
- $O(1.554^n)$ time and $O(1.452^n)$ space;
- $O(1.538^n)$ time and space.

In contrast, the fastest known classical algorithm for treewidth uses $O(1.755^n)$ time and space. The first two speed-ups are obtained in a fairly straightforward way. The first version uses additionally only Grover's search and provides a quadratic speedup. The second speedup is more time-efficient and uses both Grover's search and the quantum exponential dynamic programming by Ambainis et al. (SODA '19). The third version uses the specific properties of the classical algorithm and treewidth, with a modified version of the quantum dynamic programming on the hypercube. As a small side result, we give a new classical time-space tradeoff for computing treewidth in $O^*(2^n)$ time and $O^*(\sqrt{2^n})$ space.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Quantum computation theory

Keywords and phrases Quantum computation, Treewidth, Exact algorithms, Dynamic programming

Digital Object Identifier 10.4230/LIPIcs.TQC.2022.11

Related Version *Preprint*: `https://arxiv.org/abs/2202.08186`

Funding Supported by the project "Quantum algorithms: from complexity theory to experiment" funded under ERDF programme 1.1.1.5.

1 Introduction

For many NP-complete problems, the exact solution can be found much faster than a brute-force search over the possible solutions; it is not so rare that the best currently known algorithms are exponential [9]. Perhaps one of the most famous examples is the travelling salesman problem, where a naive brute-force requires $O^*(n!)$ computational time, but a dynamic programming algorithm solves it exactly only in $O^*(2^n)$ time [4, 14]. Such algorithms are studied also because they can reveal much about the mathematical structure of the problem and because sometimes in practice they can be more efficient than subexponential algorithms with a large constant factor in their complexity.

With the advent of quantum computing, it is curious how quantum procedures can be used to speed up such algorithms. A clear example is illustrated by the SAT problem: while iterating over all possible assignments to the Boolean formula on n variables gives $O^*(2^n)$ time, Grover's search [13] can speed this up quadratically, resulting in $O^*(\sqrt{2^n})$ time. Grover's search can also speed up exponential dynamic programming: recently Ambainis et al. [1] have shown how to apply Grover's search recursively together with classical

precalculation to speed up the $O^*(2^n)$ dynamic programming introduced by Bellman, Held and Karp [4, 14] to a $O(1.817^n)$ quantum algorithm. For some problems like the travelling salesman problem and minimum set cover, the authors also gave a more efficient $O(1.728^n)$ time quantum algorithm by combining Grover's search with both divide & conquer and dynamic programming techniques. Their approach has been subsequently applied to find a speedup for more NP-complete problems, including graph coloring [19], minimum Steiner tree [18] and optimal OBDD ordering [20].

In this paper, we focus on the NP-complete problem of finding the treewidth of a graph. Informally, the treewidth is a value that describes how close the graph is to a tree; for example, the treewidth is 1 when the graph is a tree, while the treewidth of a complete graph on n vertices is $n-1$. This quantity is prominently used in parameterized algorithms, as many problems are efficiently solvable when treewidth is small, such as vertex cover, independent set, dominating set, Hamiltonian cycle, graph coloring, etc. [3]. The applications of treewidth, both theoretical and practical, are numerous, see [5] for a survey. If the treewidth is at most k, it can be computed exactly in $O(n^{k+2})$ time [2]; 2-approximated in parameterized linear time $2^{O(k)}n$ [17]; $O(\sqrt{\log k})$-approximated in polynomial time [8]; k-approximated in $O(k^7 n \log n)$ time [10].

As for exact exponential time treewidth algorithms, both currently most time and space efficient algorithms were proposed by Fomin and Villanger in [11]: the first uses $O(1.755^n)$ time and space and the second requires $O(2.616^n)$ time and polynomial space. The crucial ingredient of these algorithms is a combinatorial lemma that upper bounds the number of connected subsets with fixed neighborhood size (Lemma 7), as well as gives an algorithm that lists such sets.

Our main motivation for tackling these algorithms is that although the $O(1.817^n)$ quantum algorithm from [1] is applicable to treewidth, it is still less efficient than Fomin's and Villanger's. In this paper we show that their techniques are also amenable to quantum search procedures. In particular, we focus on their polynomial space algorithm. This algorithm has two nested procedures: the first procedure uses Lemma 7 to search through specific subsets of vertices S to fix as a bag of the tree decomposition; the second procedure finds the optimal width of the tree decomposition with S as a bag.

We find that Grover's search can be applied to the listing procedure of Lemma 7, thus speeding up the first procedure quadratically. For the second procedure, classically one can use either the $O^*(2^n)$ time and space dynamic programming algorithm or the $O^*(4^n)$ time and polynomial space divide & conquer algorithm (Fomin and Villanger use the latter), which both were introduced in [6]. The divide & conquer algorithm we can also speed up using Grover's search. Thus, we obtain a quadratic speedup for the polynomial space algorithm:

▶ **Theorem 1.** *There is a bounded-error quantum algorithm that finds the exact treewidth of a graph on n vertices in $O(1.61713^n)$ time and polynomial space.*

Next, using the fact that the $O^*(2^n)$ dynamic programming algorithm can be sped up to an $O^*(1.817^n)$ quantum algorithm together with the quadratic speedup of Lemma 7, we obtain our second quantum algorithm:

▶ **Theorem 2.** *Assuming the QRAM data structure, there is a bounded-error quantum algorithm that finds the exact treewidth of a graph on n vertices in $O(1.55374^n)$ time and $O(1.45195^n)$ space.*

The last theorem suggests a possibility for an even more efficient algorithm by trading some space for time. We achieve this by proving a treewidth property which essentially states that we can precalculate some values of dynamic programming for the original graph, and reuse these values in the dynamic programming for its subgraphs (Lemma 22). This

allows us a global precalculation, which can be used in the second procedure of the treewidth algorithm. To do that, we have to modify the $O(1.817^n)$ algorithm of [1]. We refer to it as the asymmetric quantum exponential dynamic programming. This gives us the following algorithm:

▶ **Theorem 3.** *Assuming the QRAM data structure, there is a bounded-error quantum algorithm that finds the exact treewidth of a graph on n vertices in $O(1.53793^n)$ time and space.*

Lastly, we observe that replacing the $O^*(4^n)$ divide & conquer algorithm in the classical $O(2.616^n)$ polynomial space algorithm by the $O^*(2^n)$ dynamic programming only lowers the time complexity to $O^*(2^n)$. However, the interesting consequence is that the space requirement drops down to $O^*(\sqrt{2^n})$. Hence, we obtain a *classical* time-space tradeoff:

▶ **Theorem 4.** *The treewidth of a graph on n vertices can be computed in $O^*(2^n)$ time and $O^*\left(\sqrt{2^n}\right)$ space.*

Time-wise, this is more efficient than the $O(2.616^n)$ time polynomial space algorithm, and space-wise, this is more efficient than the $O(1.755^n)$ time and space algorithm. It also fully subsumes the time-space tradeoffs for permutation problems proposed in [16] applied to treewidth.

2 Preliminaries

We denote the set of integers from 1 to n by $[n]$. For a set S, denote the set of all its subsets by 2^S. We call a permutation of a set of vertices $S \subseteq V$ a bijection $\pi : S \to [|S|]$. We denote the set of permutations of S by $\Pi(S)$. For a permutation $\pi \in \Pi(S)$, let $\pi_{<v} = \{w \in S \mid \pi(w) < \pi(v)\}$ and $\pi_{>v} = \{w \in S \mid \pi(w) > \pi(v)\}$. We say that a subset T is a *prefix* of $\pi : S \to [|S|]$ if $\{\pi(t) \mid t \in T\} = \{1, \dots, i\}$ for some $i \in \{1, \dots, |S|\}$ or T is an empty set. Similarly, we say that T is a *suffix* of $\pi : S \to [|S|]$ if $\{\pi(t) \mid t \in T\} = \{i, \dots, |S|\}$ for some $i \in \{1, \dots, n\}$ or T is an empty set.

We write $O(f(n)) = \text{poly}(n)$ if $f(n) = O(n^c)$ for some constant c. Let $O(\text{poly}(n)f(m)) = O^*(f(m))$. This is useful since our subprocedures will often have some running time $f(m)$ times some function that depends on the size of the input graph G on n vertices. In this paper, we are primarily concerned with the exponential complexity of the algorithms, hence, we are interested in the $f(m)$ value of an $O^*(f(m))$ complexity.

Graph notation

For a graph $G = (V, E)$ and a subset of vertices $S \subseteq V$, denote $G[S]$ as the graph induced in G on S. For a subset of vertices $S \subseteq V$, let $N(S) = \{v \in V - S \mid u \in S, \{u, v\} \in E\}$ be its *neighborhood*. We call a subset $S \subseteq V$ connected if $G[S]$ is connected, and $C \subseteq V$ a clique if $G[C]$ is a complete graph.

For completeness, we also describe the notions of *potential maximum cliques* and *minimal separators*, which are specific subsets of V. Our quantum algorithms don't additionally rely on them more than the algorithm of Fomin and Villanger; for their further properties, see e.g. [11].

A graph is called *chordal* if every cycle of length at least 4 contains an edge that connects non-consecutive vertices of the cycle. A *triangulation* of a graph $G = (V, E)$ is a chordal graph $H = (V, E')$ such that $E \subseteq E'$. A triangulation H is *minimal* if every graph obtained from H by removing any edge is not a triangulation. A set of vertices $\Omega \subseteq V$ of a graph $G = (V, E)$ is a *potential maximum clique* if there is a minimal triangulation H of G such that Ω is a maximal clique of H.

For two non-adjacent vertices u and v in a graph G, a subset of vertices $S \subseteq V$ is a u, v-separator if u and v are in different connected components of $G[V - S]$. A u, v-separator is *minimal* if none of its proper subsets is a u, v-separator. A set S is called a *minimal separator*, if there exist two vertices $u, v \in G$ such that S is a minimal u, v-separator.

Treewidth

A *tree decomposition* of a graph $G = (V, E)$ is a pair (X, T), where $T = (V_T, E_T)$ is a tree and $X = \{\chi_i \mid i \in V_T\} \subseteq 2^V$ such that:

- $\bigcup_{\chi \in X} \chi = V$;
- for each edge $\{u, v\} \in E$, there exists $\chi \in X$ such that $u, v \in \chi$;
- for any vertex $v \in V$ in G, the set of vertices $\{\chi \mid v \in \chi\}$ forms a connected subtree of T.

We call the subsets $\chi \in X$ *bags* and the vertices of T *nodes*. The *width* of (X, T) is defined as the minimum size of $\chi \in X$ minus 1. The *treewidth* of G is defined as the minimum width of a tree decomposition of G and we denote it by $\mathbf{tw}(G)$. We also consider optimal tree decompositions given that some subset $\chi \in V$ is a bag of the tree. We denote the smallest width of a tree decomposition of G among those that contain χ as a bag by $\mathbf{tw}(G, \chi)$.

Approximations

For the binomial coefficients, we use the following well-known approximation:

▶ **Theorem 5** (Entropy approximation). *For any $k \in [0, 1]$, we have $\binom{n}{k} \leq 2^{\mathrm{H}\left(\frac{k}{n}\right) \cdot n}$, where $\mathrm{H}(\epsilon) = -(\epsilon \log_2(\epsilon) + (1 - \epsilon) \log_2(1 - \epsilon))$ is the binary entropy function.*

Quantum subroutines

Our algorithms use a well-known variation of Grover's search, quantum minimum finding:

▶ **Theorem 6** (Theorem 1 in [7]). *Let $\mathcal{A} : N \to [n]$ be an exact quantum algorithm with running time T. Then there is a bounded-error quantum algorithm that computes $\min_{i \in [N]} \mathcal{A}(i)$ in $O^*(T\sqrt{N})$ time.*

Two of our algorithms use the QRAM data structure [12]. This structure stores N memory entries and, given a superposition of memory indices together with an empty data register $\sum_{i \in [N]} \alpha \left|i\right\rangle \left|\mathbf{0}\right\rangle$, it produces the state $\sum_{i \in [N]} \alpha \left|i\right\rangle \left|\mathrm{data}_i\right\rangle$ in $O(\log N)$ time. In our algorithms, N will always be exponential in n, which means that a QRAM operation is going to be polynomial in n. Thus, this factor will not affect the exponential complexity, which we are interested in.

In our algorithms, we will often have a quantum algorithm that takes exact subprocedures (like in Theorem 6), and give it bounded-error subprocedures. Since we always going to take $O(\exp(n))$ number of inputs, this issue can be easily solved by repeating the subprocedures $\mathrm{poly}(n)$ times to boost the probability of correct answer to $1 - O(1/\exp(n))$: it can be then shown that the branch in which all the procedures have correct answers has constant amplitude. The final bounded-error algorithm incurs only a polynomial factor, and does not affect the exponential complexity. We also note that on a deeper perspective, all our quantum subroutines are based on the primitive of Grover's search [13]; an implementation of Grover's search with bounded-error inputs that does not incur additional factors in the complexity has been shown in [15].

We also are going to encounter an issue that sometimes we have some real parameter $\alpha \in [0, 1]$ and we are examining $\binom{n}{\alpha n}$. Since αn is not integer, this value is not defined; however, we can take this to be any value between $\binom{n}{\lfloor \alpha n \rfloor}$ or $\binom{n}{\lceil \alpha n \rceil}$, as they differ only by a factor of n. Thus, this does not produce an issue for the exponential complexity analysis. Henceforward we abuse the notation and simply write $\binom{n}{\alpha n}$.

3 Combinatorial lemma

In this section we describe how the main combinatorial lemma of [11] can be sped up quantumly qudratically using Grover's search.

▶ **Lemma 7** (Lemmas 3.1. and 3.2. in [11]). *Let $G = (V, E)$ be a graph. For every $v \in V$ and $b, f \geq 0$, the number of connected subsets $B \subseteq V$ such that (1) $v \in B$, (2) $|B| = b + 1$, and (3) $|N(B)| = f$ is at most $\binom{b+f}{b}$. There also exists an algorithm that lists all such sets in $O^*(\binom{b+f}{b})$ time and polynomial space.*

Informally, this lemma is used in the treewidth algorithm to search for a set, such that, if fixed as a bag of the tree decomposition, the remaining graph breaks down into connected components of bounded size; then, the optimal width of the tree decomposition with this bag fixed can be solved using algorithms from Section 4. The lemma upper bounds the number of sets to consider.

Their proof of this lemma (see Appendix A) essentially gives a branching algorithm that splits the problem into several problems of the same type, and solves them recursively. The idea for applying Grover's search to such a branching algorithm is simple. The algorithm that generates all sets can be turned into a procedure that, given a number $i \in [\binom{b+f}{b}]$ of the set we need to generate, generates this set in polynomial time. Then, we can run Grover's search over all integers in $[\binom{b+f}{b}]$ on this procedure.

▶ **Lemma 8.** *Let $G = (V, E)$ be a graph on n vertices, and $\mathcal{A} : 2^V \to [n]$ be an exact quantum algorithm with running time T. For every $v \in V$ and $b, f \geq 0$, let $\mathcal{B}_{v,b,f}$ be the set of connected subsets $B \subseteq V$ satisfying the conditions of Lemma 7. Then there is a bounded-error quantum algorithm that computes $\min_{B \in \mathcal{B}_{v,b,f}} \mathcal{A}(B)$ in time $O^*\left(T\sqrt{\binom{b+f}{b}}\right)$.*

4 Fixed bag treewidth algorithms

In this section we describe algorithms that calculate the optimal treewidth of a graph with the condition that a subset of its vertices is fixed as a bag of the tree decomposition. We then show ways to speed them up quantumly. Both approaches were given by Bodlaender et al. [6].

4.1 Treewidth as a linear ordering

Both of these algorithms use the fact that treewidth can be seen as a graph linear ordering problem. For a detailed description, see Section 2.2 of [6], from where we also borrow a lot of notation. We will also use the properties of this formulation in our improved quantum algorithm.

A linear ordering of a graph $G = (V, E)$ is a permutation $\pi \in \Pi(V)$. The task of a linear ordering problem is finding $\min_{\pi \in \Pi(V)} f_G(\pi)$, for some known function f_G.

For two vertices $v, w \in V$, define a predicate $P^G_\pi(v, w)$ to be true iff there is a path from v to w in G such that all internal vertices in that path are before v and w in π. Then define $R^G_\pi(v)$ to be the number of vertices w such that $\pi(w) > \pi(v)$ and $P^G_\pi(v, w)$ holds. The following proposition gives a description of treewidth as a linear ordering problem:

▶ **Proposition 9** (Proposition 3 in [6]). *Let $G = (V, E)$ be a graph, and k a non-negative integer. The treewidth of G is at most k iff there is a linear ordering π of G such that for each $v \in V$, we have $R^G_\pi(v) \leq k$.*

For a set of vertices $S \subseteq V$ and a vertex $v \notin S$, define $Q_G(S, v) = \{w \in V - S - \{v\} \mid v \text{ and } w \text{ are connected by a path in } G[S \cup \{v, w\}]\}$. Note that $R_\pi^G(v) = |Q_G(\pi_{<v}, v)|$, and $|Q_G(S, v)|$ can be computed in $\text{poly}(n)$ time using, for example, depth-first search.

Then define the quantities $\text{TWR}_G(L, S) = \min_{\substack{\pi \in \Pi(V) \\ L \text{ is a prefix of } \pi}} \max_{v \in S} |Q_G(L \cup \pi_{<v}, v)|$ and $\text{TW}_G(S) = \min_{\pi \in \Pi(V)} \max_{v \in S} |Q_G(\pi_{<v}, v)|$. These notations are connected by the relation $\text{TW}_G(G) = \text{TWR}_G(\varnothing, S)$. Note that $\mathbf{tw}(G)$ is equal to $\min_{\pi \in \Pi(V)} \max_{v \in V} R_\pi^G(v) = \text{TW}_G(V)$.

The following lemma gives a way to find optimal fixed bag tree decompositions using the algorithms for finding the optimal linear arrangements:

▶ **Lemma 10.** *Let $G = (V, E)$ be a graph, and $\chi \subseteq V$ a subset of its vertices. Then $\mathbf{tw}(G, \chi) = \max(\text{TW}_G(V - \chi), |\chi| - 1)$.*

In the final treewidth algorithms, we will also use the following fact:

▶ **Lemma 11.** *Let $G = (V, E)$ be a graph and $\chi \subseteq V$ a subset of its vertices. Let $\mathcal{C}$ be the set of connected components of $G[V - \chi]$. Then $\mathbf{tw}(G, \chi) = \max_{C \in \mathcal{C}} \mathbf{tw}(G[C \cup \chi], \chi)$.*

Proofs of these lemmas are given in Appendix B.

4.2 Divide & Conquer

The first algorithm is based on the following property:

▶ **Lemma 12** (Lemma 7 in [6])**.** *Let $G = (V, E)$ be a graph, $S \subseteq V$, $|S| \geq 2$, $L \subseteq V$, $L \cap S = \varnothing$, $1 \leq k < |S|$. Then*

$$\text{TWR}_G(L, S) = \min_{\substack{S' \subseteq S \\ |S'| = k}} \max(\text{TWR}_G(L, S'), \text{TWR}_G(L \cup S', S - S')).$$

Note that $\text{TWR}_G(L, \{v\}) = |Q_G(L, v)|$ can be calculated in polynomial time. The value we wish to calculate is $\text{TWR}_G(\varnothing, V - \chi)$. Picking $k = |S|/2$ in Lemma 12 and applying Lemma 10, we obtain a $\text{poly}(|V|)4^{|V|-|\chi|}$ deterministic algorithm with polynomial space:

▶ **Theorem 13** (Theorem 8 in [6])**.** *Let $G = (V, E)$ be a graph on n vertices and $\chi \subseteq V$ a subset of its vertices. There is an algorithm that calculates $\mathbf{tw}(G, \chi)$ in $O^*(4^{n-|\chi|})$ time and polynomial space.*

Immediately we can prove a quadratic quantum speedup using Grover's search:

▶ **Theorem 14.** *Let $G = (V, E)$ be a graph on n vertices and $\chi \subseteq V$ a subset of its vertices. There is a bounded-error quantum algorithm that calculates $\mathbf{tw}(G, \chi)$ in $O^*(2^{n-|\chi|})$ time and polynomial space.*

Proof. We can apply quantum minimum finding to check sets S' in Lemma 12, in order to obtain a quadratic speedup over Theorem 13. To avoid the accumulation of error in the recursion, we can use the Grover's search implementation with bounded-error inputs [15]. ◀

4.3 Dynamic programming

The second algorithm is based on the following recurrence:

▶ **Lemma 15** (Lemma 5 in [6])**.** *Let $G = (V, E)$ be a graph and $S \subseteq V$, $S \neq \varnothing$. Then*

$$\text{TW}_G(S) = \min_{v \in S} \max(\text{TW}_G(S - \{v\}), |Q_G(S - \{v\}, v)|).$$

Note that in fact Lemma 15 is a special case of Lemma 12 with $L = \varnothing$ and $k = |S| - 1$. This lemma together with Lemma 10 and a dynamic programming technique by Bellman, Held and Karp [4, 14] gives the following algorithm:

▶ **Theorem 16** (Theorem 6 in [6]). *Let $G = (V, E)$ be a graph on n vertices and $\chi \subseteq V$ a subset of its vertices. There is an algorithm that calculates $\mathbf{tw}(G, \chi)$ in $O^*(2^{n-|\chi|})$ time and space.*

This algorithm calculates the values of $\mathrm{TW}_G(S)$ for all sets S in order of increasing size of the sets, and also stores them all in memory. Such dynamic programming can be sped up quantumly: Ambainis et al. [1] have shown a $O(1.817^n)$ time and space quantum algorithm with QRAM for such problems. Therefore, this gives the following quantum algorithm:

▶ **Theorem 17.** *Let $G = (V, E)$ be a graph on n vertices and $\chi \subseteq V$ a subset of its vertices. Assuming the QRAM data structure, there is a bounded-error quantum algorithm that calculates $\mathbf{tw}(G, \chi)$ in $O^*(1.816905^{n-|\chi|})$ time and space.*

Note that this algorithm can be used to calculate $\mathrm{TWR}_G(L, S)$. Firstly, $\mathrm{TWR}_G(L, \varnothing) = 0$ and

$$\mathrm{TWR}_G(L, S) = \min_{v \in S} \max(\mathrm{TWR}_G(L, S - \{v\}), |Q_G(L \cup (S - \{v\}), v)|)$$

by Lemma 12. As already mentioned earlier, the value $|Q_G(L \cup (S - \{v\}), v)|$ can be calculated in polynomial time. Hence this recurrence is of the same form as Lemma 15.

▶ **Theorem 18.** *Let $G = (V, E)$ be a graph on n vertices and $L, S \subseteq V$ be disjoint subsets of vertices. Assuming the QRAM data structure, there is a bounded-error quantum algorithm that calculates $\mathrm{TWR}_G(L, S)$ in $O^*(1.81691^{|S|})$ time and space.*

5 Fomin's and Villanger's algorithm

In this section, we first describe the polynomial space treewidth algorithm of [11]. Afterwards, we summarize the time complexity for the classical algorithm and then for the same algorithm sped up by the quantum tools presented above. The proofs for the theorem in this Section are given in Appendix B.

The algorithm relies on the following, shown implicitly in the proof of Theorem 7.3. of [11].

▶ **Lemma 19.** *Let $G = (V, E)$ be a graph, and $\beta \in [0, 1]$. There exists an optimal tree decomposition (X, T) of G so that at least one of the following holds:*

(a) *There exists a bag $\Omega \in X$ such that Ω is a potential maximum clique and there exists a connected component C of $G[V - \Omega]$ such that $|C| \leq \beta n$.*

(b) *There exists a bag $S \in X$ such that S is a minimal separator and there exist two disjoint connected components C_1, C_2 of $G[V - S]$ such that $N(C_1) = N(C_2) = S$ and $|C_2| \geq |C_1| \geq \beta n$.*

The idea of the algorithm then is to try out all possible potential maximum cliques and minimal separators that conform to the conditions of this lemma, and for each of these sets, to find an optimal tree decomposition of G given that the examined set is a bag of the decomposition using the algorithm from Theorem 13. The treewidth of G then is the minimum width of all examined decompositions.

The potential maximum clique generation is based on the following lemma.

▶ **Lemma 20** (Lemma 7.1. in [11]). *Let $G = (V, E)$ be a graph. The number of maximum potential cliques Ω of size p such that there exists a connected component C of $G[V - \Omega]$ of size c is at most $n\binom{n-c}{p-1}$.*[1] *The set of all these cliques can also be generated in time $O^*(\binom{n-c}{p-1})$.*

For the minimal separators, suppose that the size of S is fixed, denote it by s. Note that since C_1 in Lemma 19 is a connected component such that $N(C_1) = S$, then instead of generating minimal separators, we can generate the sets of vertices C with neighborhood size equal to s. The set generated in this way contains in their neighbourhoods all of the minimal separators of size s that we are interested in, and for those sets that do not contain such a separator, the fixed-bag treewidth algorithm will still find some tree decomposition of the graph, albeit not an optimal one. The generation is done using Lemma 7: for a fixed size c of C, the number of such C with exactly s neighbors is at most $n\binom{c+s}{c}$ (the factor of n comes from trying each of n vertices as the fixed vertex $v \in B$). The algorithm generating all such C requires time $O^*\left(\binom{c+s}{c}\right)$. For a set C, we then find an optimal tree decomposition of G containing $N(C)$ as a fixed bag using the algorithm from Theorem 13. In this way we work through all c from βn to $n - s - |C_2| \leq (1 - \beta)n - s$.

Algorithm 1 The polynomial space algorithm for treewidth.

1. For c from 0 to βn and p from 1 to $n - c$ generate the set of potential maximal cliques Ω of size p with a connected component of $G[V - \Omega]$ of size c using Lemma 20. For each Ω, find $\mathbf{tw}(G, \Omega)$ using Theorem 13.
2. For s from 1 to $(1 - 2\beta)n$ and for c from βn to $(1 - \beta)n - s$ generate the set of subsets C such that $|C| = c$ and $N(C) = s$ using Lemma 7. Let $S = N(C)$; then $\mathbf{tw}(G, S)$ is equal to the maximum of $\mathbf{tw}(G[S \cup C], S)$ and $\mathbf{tw}(G[V - C], S)$ by Lemma 11. Use the algorithm from Theorem 13 to compute these values.
3. Output the minimum width of all examined tree decompositions.

▶ **Theorem 21** (Theorem 7.3. in [11]). *Algorithm 1 computes the treewidth of a graph with n vertices in $O^*(2.61508^n)$ time and polynomial space.*

5.1 A time-space tradeoff

One might ask whether replacing the $O^*(4^n)$ divide & conquer algorithm from Theorem 13 with the $O^*(2^n)$ dynamic programming algorithm from Theorem 16 in Algorithm 1 can give any interesting complexity. Indeed, we can show the following previously unexamined classical time-space tradeoff.

▶ **Theorem 4.** *The treewidth of a graph on n vertices can be computed in $O^*(2^n)$ time and $O^*\left(\sqrt{2^n}\right)$ space.*

We can compare this to the existing treewidth algorithms. The most time-efficient treewidth algorithm runs in time and space $O^*(1.7549^n)$ [11], which is more than $O^*\left(\sqrt{2^n}\right)$. The polynomial space $O^*(2.6151^n)$ algorithm, of course, is slower than $O^*(2^n)$. The time-space tradeoffs for permutation problems from [16] give $TS \gtrsim 3.93$, where $T \geq 2$ and

[1] The original lemma gives an upper bound if the size of Ω is not fixed, but our statement follows from their proof. We need to fix $|\Omega|$ because in the quantum algorithms, Grover's search will be called for fixed $|\Omega|$ and $|C|$.

$\sqrt{2} \leq S \leq 2$ are the time and space complexities (bases of the exponent to the power of n) of the algorithm. Here, $TS = 2^{\frac{3}{2}} \approx 2.83$, $T = 2$ and $S = \sqrt{2}$. Therefore, Theorem 4 fully subsumes their tradeoff for treewidth. We also note that this tradeoff cannot be "tuned" directly for less time and more space, since the first stage requires $\Theta^*(2^n)$ time for any β.

5.2 Quantum complexity

Now we are ready to examine the quantum versions of the algorithm. First, we consider the analogue of Algorithm 1 sped up quadratically using Grover's search using Lemma 8 and Theorem 14.

▶ **Theorem 1.** *There is a bounded-error quantum algorithm that finds the exact treewidth of a graph on n vertices in $O(1.61713^n)$ time and polynomial space.*

Similarly, we can replace the algorithm from Theorem 16 with the quantum dynamic programming algorithm from Theorem 17:

▶ **Theorem 2.** *Assuming the QRAM data structure, there is a bounded-error quantum algorithm that finds the exact treewidth of a graph on n vertices in $O(1.55374^n)$ time and $O(1.45195^n)$ space.*

6 Improved quantum algorithm

We can see that in Theorem 2 we still have some room for improvement by trading space for time. This can be done using an additional technique. The main idea is to make a global precalculation for $\mathrm{TW}_G(S)$ for all subsets $S \subseteq V$ of size at most αn, for some constant parameter α. Then, as we will see later, these values can be used in all calls of the quantum dynamic programming because of the properties of treewidth. For many such calls, this reduces the $O^*(1.817^d)$ running time to something smaller, which in turn reduces the overall time complexity.

6.1 Asymmetric quantum dynamic programming on the hypercube

We describe our modification to the quantum dynamic programming algorithm by Ambainis et al. [1]. First, we prove the following lemma that allows us to reutilize the precalculated DP values on the original graph G in the DP calculation in the subgraphs examined by our algorithms.

▶ **Lemma 22.** *Let $G = (V, E)$ be a graph, and $\chi \subseteq V$ a subset of its vertices. Suppose that C is a union of some number of connected components of $G[V - \chi]$. Then for any $S \subseteq C$, we have $\mathrm{TW}_{G[C \cup \chi]}(S) = \mathrm{TW}_G(S)$.*

Proof. Examine the permutations π achieving

$$\mathrm{TW}_{G[C\cup\chi]}(S) = \min_{\pi \in \Pi(C \cup \chi)} \max_{v \in S} |Q_G(\pi_{<v}, v)|.$$

As a direct consequence of Lemma 15, there exists such a permutation π with the property that S is its prefix. Now let $\pi' \in \Pi(V)$ be a permutation obtained by adding the vertices of $V - C - \chi$ to π in any order. Examine any vertex $u \in V - C - \chi$ and any $v \in S$. Since u and v are located in different connected components of $G[C - \chi]$, any path from u to v in G passes through some vertex of χ. However, $\pi_{<v} \cap \chi = \varnothing$, as $\pi_{<v} \subset S$. Then we can conclude that $Q_{G[C\cup\chi]}(\pi_{<v}, v) = Q_G(\pi'_{<v}, v)$, as u cannot contribute to Q. Therefore, $\mathrm{TW}_G(S) \leq \mathrm{TW}_{G[C\cup\chi]}(S)$. On the other hand, $\mathrm{TW}_G(S) \geq \mathrm{TW}_{G[C\cup\chi]}(S)$, as additional vertices cannot decrease TW. ◀

Now we are ready to describe our quantum dynamic programming procedure. Suppose that all values of $\mathrm{TW}_G(S)$ for sets with $|S| \leq \alpha n$ are known and stored in QRAM, where $\alpha \in \left[0, \frac{1}{2}\right]$ is some fixed parameter. Suppose that we have fixed a subset $\chi \subseteq V$, and our task is to calculate $\mathbf{tw}(G[C \cup \chi], \chi)$ for a union C of some connected components of $G[V - \chi]$. By Lemma 10, it is equal to $\max(\mathrm{TW}_{G[C\cup\chi]}(C), |\chi| - 1)$. Since $|\chi|$ is known, our goal is to compute $\mathrm{TW}_{G[C\cup\chi]}(C)$.

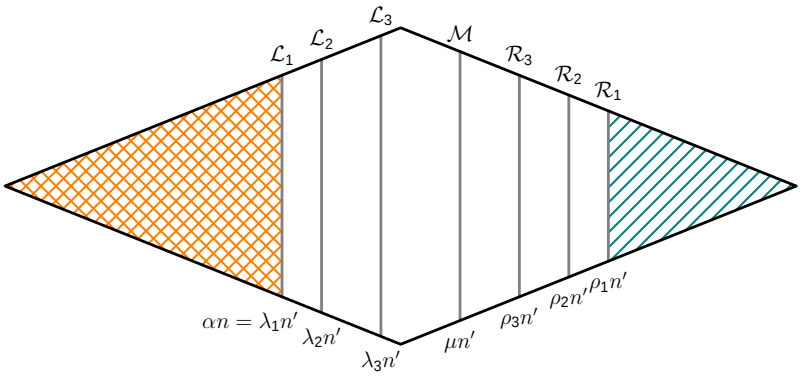

Figure 1 A schematic representation of layers in the Boolean hypercube with $k = 3$.

Let $n = |V|$ and $n' = |C|$. If $n' \leq \alpha n$, then $\mathrm{TW}_{G[C\cup\chi]}(C) = \mathrm{TW}_G(C)$ by Lemma 22 and is known from the precalculated values. Hence, assume that $n' > \alpha n$. Pick some natural k, we will call this *the number of layers*. Let $\lambda_1 = \frac{\alpha n}{n'}$, and pick constants $\lambda_2 < \ldots < \lambda_k < \mu < \rho_k < \ldots < \rho_1 < 1$, such that $\lambda_1 < \lambda_2$. Then define collections $L_i = \{S \subseteq C \mid |S| = \lambda_i n'\}$, $\mathcal{M} = \{S \subseteq C \mid |S| = \mu n'\}$ and $\mathcal{R}_i = \{S \subseteq C \mid |S| = \rho_i n'\}$. We call these collections *layers*: we can represent subsets $S \subseteq C$ as vertices on the hypercube of dimension n'; then these layers are defined as the subsets of vertices with some fixed Hamming weight, see Figure 1. For all sets S corresponding to the vertices in the crosshatched area (such that $|S| \leq \alpha n$), the value of $\mathrm{TW}_{G[C\cup\chi]}(S) = \mathrm{TW}_G(S)$ is known from the assumed precalculation.

Now we will describe the quantum procedure. Denote $G' = G[C \cup \chi]$. Also denote $\mathrm{TW}'_{G'}(S) = \mathrm{TWR}_{G'}(S, C - S)$ and note that $\mathrm{TW}_{G'}(S) = \mathrm{TWR}_{G'}(\varnothing, S)$. Informally, calculating $\mathrm{TW}_{G'}(S)$ means finding the best ordering for the vertices S as a prefix of the permutation, and $\mathrm{TW}'_{G'}(S)$ means finding the best ordering for the vertices $C - S$, where $C - S$ is in the middle of permutation, followed by some ordering of χ.

Algorithm 2 is exactly the algorithm of [1], with the exception that the precalculation is performed only for suffixes (and the precalculation for prefixes comes "for free"). Informally, the idea of the algorithm is to find the optimal path between the vertices s and t with the smallest and highest Hamming weight in the hypercube. First, we use Grover's search over the vertex v_{k+1} in the middle layer $\mathcal{M}$. Then we search independently for the best path from s to v_{k+1} and from v_{k+1} to t; the optimal path from s to t is their concatenation. To find the best path from s to v_{k+1}, we use Grover's search over the vertex v_k on the layer $\mathcal{L}_k$ such that there exists a path from v_k to v_{k+1}. Then we find the best path from v_k to v_{k+1} by recursively using the $O^*(1.817^{n'})$ algorithm (where n' is the dimension of the hypercube with v_k and v_{k+1} being the smallest and largest weight vertices, respectively). We combine it with the best path from s to v_k, which we find in the similar way (fixing v_{k-1}, …, v_1). The value of the optimal path from s to v_1 is known from the global precalculation we assumed

took place before the algorithm. The optimal path from v_{k+1} to t is found analogously; only to know the value of the best path from vertices in $\mathcal{R}_1$ to t, we have to precalculate these values "from the back" using DP in the beginning of the algorithm.

Algorithm 2 Asymmetric quantum dynamic programming algorithm.

1. For all $S \in \mathcal{R}_1$, calculate and store in QRAM the values $\mathrm{TW}'_{G'}(S)$ using the recurrence

$$\mathrm{TW}'_{G'}(S) = \min_{v \in C-S} \max\big(\mathrm{TW}'_{G'}(S \cup \{v\}), |Q_{G'}(S,v)|\big).$$

   This follows from Lemma 12 with $k = 1$.
2. Use quantum minimum finding over sets $S \in \mathcal{M}$ to find the answer,

$$\mathrm{TW}_{G'}(C) = \min_{S \in \mathcal{M}} \max\big(\mathrm{TW}_{G'}(S), \mathrm{TW}'_{G'}(S)\big).$$

   This also follows from Lemma 12 with $k = \mu n'$.

   - To find $\mathrm{TW}_{G'}(S)$, we use the recursive procedure $\textsc{BestPrefix}_i(G', S)$. Its value is equal to $\mathrm{TW}_{G'}(S)$, and it requires $S \in \mathcal{L}_i$ (if $i = k+1$, then $S \in \mathcal{M}$). The needed value is then given by $\textsc{BestPrefix}_{k+1}(G', S)$. The description of $\textsc{BestPrefix}_i(G', S)$:
     - If $i = 1$, return $\mathrm{TW}_{G'}(S) = \mathrm{TW}_G(S)$ that is stored in QRAM.
     - If $1 < i \le k+1$, then use quantum minimum finding over the sets $T \in \mathcal{L}_{i-1}$ to find

$$\mathrm{TW}_{G'}(S) = \min_{\substack{T \in \mathcal{L}_{i-1} \\ T \subset S}} \max(\textsc{BestPrefix}_{i-1}(G', T), \mathrm{TWR}_{G'}(T, S-T)).$$

       Again, this recurrence follows from Lemma 12 with $k = \lambda_{i-1} n'$. The value of $\mathrm{TWR}_{G'}(T, S-T)$ is calculated by the quantum dynamic programming from Theorem 18 and requires $O^*(1.817^{|S|-|T|})$ time and QRAM space.
   - To find $\mathrm{TW}'_{G'}(S)$, we similarly use the recursive procedure $\textsc{BestSuffix}_i(G', S)$. Its value is equal to $\mathrm{TW}'_{G'}(S)$, and it requires $S \in \mathcal{R}_i$ (if $i = k+1$, then $S \in \mathcal{M}$). The needed value is then given by $\textsc{BestSuffix}_{k+1}(G', S)$. The description of $\textsc{BestSuffix}_i(G', S)$:
     - If $i = 1$, return $\mathrm{TW}'_{G'}(S)$ stored in QRAM from the precalculation in Step 1.
     - If $1 < i \le k+1$, then use quantum minimum finding over the sets $T \in \mathcal{R}_{i-1}$ to find

$$\mathrm{TW}'_{G'}(S) = \min_{\substack{T \in \mathcal{R}_{i-1} \\ S \subset T}} \max(\mathrm{TWR}_{G'}(S, T-S), \textsc{BestSuffix}_{i-1}(G', T)).$$

       Again, this recurrence follows from Lemma 12 with $k = \rho_{i-1} n' - \rho_i n'$. The value of $\mathrm{TWR}_{G'}(S, T-S)$ is calculated by the quantum dynamic programming from Theorem 18 and requires $O^*(1.817^{|T|-|S|})$ time and QRAM space.

We will estimate the time complexity of Algorithm 2. The space complexity will not be necessary, because for the final treewidth algorithm it will be dominated by the global precalculation, as we will see later.

- The time of the precalculation Step 1 is dominated by the size of the layer $\mathcal{R}_1$. It is equal to $O^*(|\mathcal{R}_1|) = O^*(\binom{n'}{\rho_1 n'})$, which by Lemma 5 is

$$O^*\Big(2^{\mathrm{H}(\rho_1)n'}\Big).$$

- Let the time of a call of $\textsc{BestPrefix}_i(G', S)$ be T_i, it can be calculated as follows. If $i = 1$,

$$T_1 = O^*(1),$$

as all we need to do is to fetch the corresponding value $\mathrm{TW}_G(S)$ from QRAM. If $i > 1$, then quantum minimum finding examines all $T \in \mathcal{L}_{i-1}$ such that $T \subset S$. The number of such T is $\binom{|S|}{|T|} = \binom{\lambda_i n'}{\lambda_{i-1} n'}$ (for generality, denote $\lambda_{k+1} = \mu$). Again, by Lemma 5, this is at most $2^{\mathrm{H}(\lambda_{i-1}/\lambda_i)\cdot\lambda_i n'}$. The call to $\textsc{BestPrefix}_{i-1}(G', T)$ requires time T_{i-1} and calculating $\mathrm{TWR}_{G'}(T, S-T)$ with the algorithm from Theorem 18 requires time $O^*(1.817^{|S|-|T|}) = O^*(1.817^{(\lambda_i-\lambda_{i-1})n'})$. Putting these estimates together, we get that for $i > 1$,

$$T_i = O^*\left(\sqrt{2^{\mathrm{H}\left(\frac{\lambda_{i-1}}{\lambda_i}\right)\cdot\lambda_i n'}} \cdot \max\Big(T_{i-1}, 1.817^{(\lambda_i-\lambda_{i-1})n'}\Big)\right).$$

- The time T'_i for $\textsc{BestSuffix}_i(G', S)$ is calculated analogously. We can check the precalculated values from Step 1 in

$$T'_1 = O^*(1)$$

and (taking $\rho_{k+1} = \mu$) for $i > 1$,

$$T'_i = O^*\left(\sqrt{2^{\mathrm{H}\left(\frac{1-\rho_{i-1}}{1-\rho_i}\right)\cdot(1-\rho_i)n'}} \cdot \max\Big(T'_{i-1}, 1.817^{(\rho_{i-1}-\rho_i)n'}\Big)\right).$$

- Lastly, the number of sets examined in the first quantum minimum finding in Step 2 is equal to the size of $\mathcal{M}$, which is $\binom{n'}{\mu n'} = 2^{\mathrm{H}(\mu)n'}$ by Lemma 5. Therefore, Step 2 requires time

$$O^*\Big(\sqrt{2^{\mathrm{H}(\mu)n'}} \cdot \max\big(T_{k+1}, T'_{k+1}\big)\Big).$$

For any of the complexities $\mathcal{T}$ examined here, let's look at $\log_2(\mathcal{T})/n'$; since we are interested in the exponential complexity, we need to investigate only the constant c in $O^*(2^{cn})$. Also note that $\log_2(1.817) \approx 0.862$. This results in the following optimization program

$$\begin{aligned}
\text{minimize} \quad & T(\lambda_1) = \max\left(\mathrm{H}(\rho_1), \frac{\mathrm{H}(\mu)}{2} + \max\big(t_{k+1}, t'_{k+1}\big)\right) \\
\text{s.t.} \quad & \lambda_1 < \ldots < \lambda_k < \lambda_{k+1} = \mu = \rho_{k+1} < \rho_k < \ldots < \rho_1 < 1 \\
& t_i = \mathrm{H}\left(\frac{\lambda_{i-1}}{\lambda_i}\right) \cdot \lambda_i + \max(t_{i-1}, 0.862(\lambda_i - \lambda_{i-1})) && \forall i \in [2, k+1] \\
& t_1 = 0 \\
& t'_i = \mathrm{H}\left(\frac{1-\rho_{i-1}}{1-\rho_i}\right) \cdot (1-\rho_i) + \max\big(t'_{i-1}, 0.862(\rho_{i-1} - \rho_i)\big) && \forall i \in [2, k+1] \\
& t'_1 = 0
\end{aligned}$$

We can solve this program numerically and find the time complexity, depending on the value of λ_1. Note that for $\lambda_1 \leq 0.28448$ the $O(1.817^{n'})$ symmetric quantum dynamic programming is more efficient, so we don't have to calculate the complexity in that case. Figure 2 shows the time complexity $T(\lambda_1)$ for $k = 0, 1, 2, 3$. We can see that the advantage of adding additional layers quickly becomes negligible.

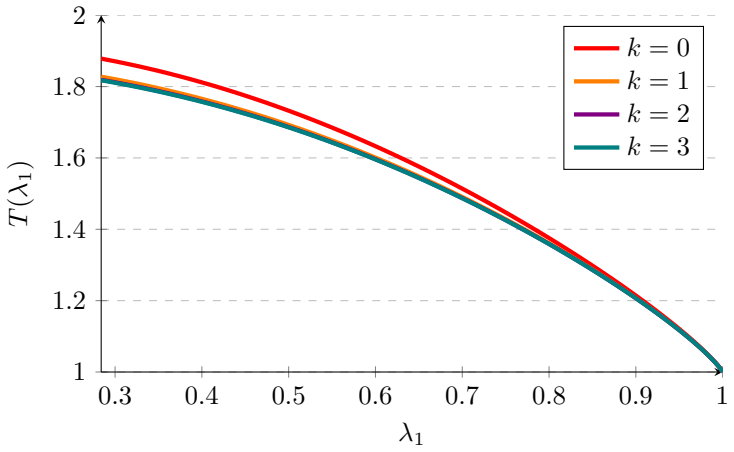

Figure 2 Running time of the asymmetric quantum dynamic programming algorithm.

Note that with $\lambda_1 \approx 0.28448$, $T(\lambda_1)$ becomes $O^*(1.817^{n'})$, as this is the same parameter for the precalculation layer as in [1]. Thus if it happens that $\alpha n < 0.28448n'$, the asymmetric version of the algorithm will have time complexity larger than $O^*(1.817^{n'})$, so in that case it is better to call the algorithm from Theorem 17. Our procedure for calculating $\mathrm{TW}_{G[C\cup\chi]}(C)$ is given in Algorithm 3.

Algorithm 3 Quantum algorithm calculating $\mathbf{tw}(G[C \cup \chi], \chi)$ assuming global precalculation.

Assume that $\mathrm{TW}_G(S)$ are stored in QRAM for all $|S| \leq \alpha n$.
- If $n' \leq \alpha n$, fetch $\mathrm{TW}_{G[C\cup\chi]}(C) = \mathrm{TW}_G(C)$ from the global precalculation.
- Else if $\alpha n \leq 0.28448n'$, find $\mathrm{TW}_{G[C\cup\chi]}(C)$ using the $O(1.817^{n'})$ algorithm of Theorem 17.
- Else calculate $\mathrm{TW}_{G[C\cup\chi]}(C)$ using Algorithm 2.

Return $\mathbf{tw}(G[C \cup \chi], \chi) = \max\big(\mathrm{TW}_{G[C\cup\chi]}(C), |\chi| - 1\big)$.

6.2 Final quantum algorithm

Now we can give the improved quantum dynamic programming algorithm for treewidth, see Algorithm 4. It requires two constant parameters: $\alpha, \beta \in \left[0, \frac{1}{2}\right]$. The value αn gives the limit for the global precalculation, and βn is the cutoff point for the two stages as in Algorithm 1.

Algorithm 4 Improved quantum algorithm for treewidth.

1. Calculate $\mathrm{TW}_G(S)$ for all subsets S such that $|S| \leq \alpha n$ and store them in QRAM.
2. For c from 0 to βn and p from 1 to $n - c$ examine the set of potential maximal cliques Ω of size p with a connected component of size c. Apply Lemma 8 to Lemma 20 to find the minimum of $\mathbf{tw}(G, \Omega)$ in $O^*\left(\sqrt{\binom{n-c}{p-1}}\right)$ iterations. Calculate the value of $\mathbf{tw}(G, \Omega)$ using Algorithm 3.
3. For s from 1 to $(1 - 2\beta)n$ and for c from βn to $(1 - \beta)n - s$ examine the set of subsets C such that $|C| = c$ and $|N(C)| = s$. Let $S = N(C)$; then $\mathbf{tw}(G, S)$ is equal to the maximum of $\mathbf{tw}(G[S \cup C], S)$ and $\mathbf{tw}(G[V - C], S)$ by Lemma 11. Find the minimum of $\mathbf{tw}(G, S)$ using Lemma 8 in $O^*\left(\sqrt{\binom{c+s}{s}}\right)$ iterations. Calculate the values of $\mathbf{tw}(G[S \cup C], S)$ and $\mathbf{tw}(G[V - C], S)$ using Algorithm 3.
4. Return the minimum width of all examined tree decompositions.

▶ **Theorem 3.** *Assuming the QRAM data structure, there is a bounded-error quantum algorithm that finds the exact treewidth of a graph on n vertices in $O(1.53793^n)$ time and space.*

We can calculate the complexity similarly as in Theorem 2.

Proof. First, we choose α such that it balances the time complexity of the global precalculation (Step 1) and the rest of the algorithm (Steps 2–4). The space complexity of this step asymptotically is equal to its time complexity. Therefore, the space complexity of this algorithm is equal to

$$O^*\left(\binom{n}{\alpha n}\right).$$

Denote the time complexity of Algorithm 3 with $|C| = n'$ and some chosen λ_1 by $O^*(\mathcal{T}(\lambda_1)^{n'})$. For fixed α and n', λ_1 is calculated as $\alpha n/n'$. Then

$$\mathcal{T}(\lambda_1) = \begin{cases} 1, & \text{if } \lambda_1 \geq 1, \\ 1.81691, & \text{if } \lambda_1 \leq 0.28448, \\ T(\lambda_1), & \text{otherwise.} \end{cases}$$

Similarly as we have obtained Equations (1, 2) in the proof of Theorem 21, we can also calculate the time complexity here. The time complexity of Step 2 now is equal to

$$O^*\left(\sum_{c=0}^{\beta n} \sqrt{2^{n-c}} \cdot \mathcal{T}\left(\frac{\alpha n}{c}\right)^c\right).$$

The running time of Step 3 is given by

$$O^*\left(\max_{d=\beta n}^{(1-\beta)n} \sqrt{\binom{n-d}{\beta n}} \cdot \mathcal{T}\left(\frac{\alpha n}{d}\right)^d\right).$$

We can numerically find that $\alpha \approx 0.154468$ and $\beta \approx 0.386401$ balance these complexities, which then are all $O(1.53793^n)$. In our numerical calculation, we have used $k = 3$ for Algorithm 3. ◀

References

1 Andris Ambainis, Kaspars Balodis, Jānis Iraids, Martins Kokainis, Krišjānis Prūsis, and Jevgēnijs Vihrovs. Quantum speedups for exponential-time dynamic programming algorithms. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '19, pages 1783–1793, USA, 2019. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611975482.107.

2 Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987. doi:10.1137/0608024.

3 Stefan Arnborg and Andrzej Proskurowski. Linear time algorithms for np-hard problems restricted to partial k-trees. *Discrete Appl. Math.*, 23(1):11–24, April 1989. doi:10.1016/0166-218X(89)90031-0.

4 Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *J. ACM*, 9(1):61–63, 1962. doi:10.1145/321105.321111.

5 Hans L. Bodlaender. Discovering treewidth. In Peter Vojtáš, Mária Bieliková, Bernadette Charron-Bost, and Ondrej Sýkora, editors, *SOFSEM 2005: Theory and Practice of Computer Science*, pages 1–16, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. doi:10.1007/978-3-540-30577-4_1.

6 Hans L. Bodlaender, Fedor V. Fomin, Arie M. C. A. Koster, Dieter Kratsch, and Dimitrios M. Thilikos. On exact algorithms for treewidth. *ACM Trans. Algorithms*, 9(1), 2012. doi:10.1145/2390176.2390188.

7 Christoph Dürr and Peter Høyer. A quantum algorithm for finding the minimum, 1996. arXiv:quant-ph/9607014.

8 Uriel Feige, Mohammad Taghi Hajiaghayi, and James R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM Journal on Computing*, 38(2):629–657, 2008. doi:10.1137/05064299X.

9 Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Springer Science & Business Media, 2010.

10 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, Michał Pilipczuk, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. *ACM Trans. Algorithms*, 14(3), June 2018. doi:10.1145/3186898.

11 Fedor V. Fomin and Yngve Villanger. Treewidth computation and extremal combinatorics. *Combinatorica*, 32:289–308, 2012. doi:10.1007/s00493-012-2536-z.

12 Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum random access memory. *Phys. Rev. Lett.*, 100:160501, 2008. doi:10.1103/PhysRevLett.100.160501.

13 Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 212–219, New York, NY, USA, 1996. Association for Computing Machinery. doi:10.1145/237814.237866.

14 Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. *Journal of SIAM*, 10(1):196–210, 1962. doi:10.1145/800029.808532.

15 Peter Høyer, Michele Mosca, and Ronald de Wolf. Quantum search on bounded-error inputs. In *Automata, Languages and Programming*, ICALP'03, pages 291–299, Berlin, Heidelberg, 2003. Springer-Verlag. doi:10.1007/3-540-45061-0_25.

16 Mikko Koivisto and Pekka Parviainen. *A Space–Time Tradeoff for Permutation Problems*, pages 484–492. SODA '10. Society for Industrial and Applied Mathematics, USA, 2010. doi:10.1137/1.9781611973075.41.

17 Tuukka Korhonen. A single-exponential time 2-approximation algorithm for treewidth, 2021. arXiv:2104.07463.

18 Masayuki Miyamoto, Masakazu Iwamura, Koichi Kise, and François Le Gall. Quantum speedup for the minimum steiner tree problem. In Donghyun Kim, R. N. Uma, Zhipeng Cai, and Dong Hoon Lee, editors, *Computing and Combinatorics*, pages 234–245, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-58150-3_19.

19 Kazuya Shimizu and Ryuhei Mori. Exponential-time quantum algorithms for graph coloring problems. In Yoshiharu Kohayakawa and Flávio Keidi Miyazawa, editors, *LATIN 2020: Theoretical Informatics*, pages 387–398, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-61792-9_31.

20 Seiichiro Tani. Quantum Algorithm for Finding the Optimal Variable Ordering for Binary Decision Diagrams. In Susanne Albers, editor, *17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020)*, volume 162 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 36:1–36:19, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SWAT.2020.36.

A Quantum speedup of the combinatorial lemma

The approach described in Section 3 was formalized by Shimizu and Mori:

▶ **Lemma 23** (Lemma 4 in [19]). *Let P be a decision problem with parameters $n_1, \dots, n_\ell$. Suppose that there is a branching rule $b(P)$ that reduces P to $m_{b(P)}$ problems $P_1, \dots, P_{m_{b(P)}}$ of the same class. Here, P_i has parameters $f_j^{b(P),i}(n_j)$ for $j \in [\ell]$, where $f_j^{b(P),i} \le n_j$. At least one of the parameters of P_i must be strictly smaller than the corresponding parameter of P. The solution for P is equal to the minimum of the solutions for P_1, …, $P_{m_b(P)}$.*

Let $U(n_1, \dots, n_\ell)$ be an upper bound on the number of leaves in the computational tree. Assume that the running time of computing $b(P)$, P_i, $f_j^{b(P),i}$ and $U(n_1, \dots, n_\ell)$ is polynomial w.r.t. n_1, …, n_ℓ. Suppose that $U(n_1, \dots, n_\ell) \ge \sum_{i=1}^{m_b(P)} U(f_1^{b(P),i}(n_1), \dots, f_\ell^{b(P),i}(n_\ell))$. Also suppose that T is the running time for the computation at each of the leaves in the computational tree. Then there is a bounded-error quantum algorithm that computes P and has running time $\mathrm{poly}(n_1, \dots, n_\ell)\sqrt{U(n_1, \dots, n_\ell)}T$.

We apply this to the combinatorial lemma:

Proof of Lemma 7. According to the proof of Lemma 7 in [11], we have that

- $\ell = 2$, $n_1 = b$, $n_2 = f$.
- $b(P)$ splits the problem into $m_{b(P)} = f + b$ problems.
- P_i has parameters $f_1^{b(P),i}(b) = b - 1$ and $f_2^{b(P),i}(f) = f - i + 1$.
- $U(b, f) = \binom{b+f}{f}$.
- $\sum_{i=1}^{m_b(P)} U(f_1^{b(P),i}(b), f_1^{b(P),i}(f)) = \sum_{i=1}^{f+b} \binom{f+b-i}{b-1} = \sum_{i=0}^{f+b-1} \binom{f+b-1-i}{b-1} = \binom{b+f}{f} = U(b, f)$.
- Computing $b(P)$, $f_1^{b(P),i}$ and $U(b, f)$ takes time polynomial in b and f; computing P_i involves contracting two vertices in the graph and can be done in $\mathrm{poly}(n)$ time. ◀

B Proofs of the theorems

To prove Lemma 10, we use the following:

▶ **Lemma 24** (Lemma 11 in [6]). *Let $C \subseteq V$ induce a clique in a graph $G = (V, E)$. The treewidth of G equals* $\max(\mathrm{TW}_G(V - C), |C| - 1)$.

Proof of Lemma 10. Completing a bag of a tree decomposition into a clique does not change the width of the tree decomposition. The claim then follows from Lemma 24. ◀

Proof of Lemma 11. Let (X, T) be a tree decomposition with the smallest width w that contains χ as a bag. For a connected component $C \in \mathcal{C}$, examine the tree decomposition (X_C, T_C) obtained from (X, T) by removing all vertices not in χ or C from all bags. Clearly,

this is a tree decomposition of $G[C \cup \chi]$ with χ as a bag; as we only have possibly removed some vertices, its width is at most w. Now, examine the tree decomposition obtained by taking all (X_C, T_C) and making χ its common bag. This is a valid tree decomposition, since no two vertices in distinct connected components of $\mathcal{C}$ are connected by an edge. Its width is the maximal width of (X_C, T_C), therefore at most w. ◀

Proof of Theorem 21. The algorithms from Lemma 7 and Theorem 13 both require polynomial space, hence it holds also for Algorithm 1.

Now we analyze the time complexity; Stage 1 of the algorithm requires time

$$O^*\left(\sum_{c=0}^{\beta n}\sum_{p=1}^{n-c}\binom{n-c}{p-1}4^{n-p}\right) = O^*\left(\sum_{c=0}^{\beta n} 2^{n-c}4^c\right) = O^*\left(\max_{c=0}^{\beta n} 2^{n+c}\right) = O^*\left(2^{(1+\beta)n}\right). \quad (1)$$

Stage 2 of the algorithm requires time

$$O^*\left(\sum_{s=1}^{(1-2\beta)n}\sum_{c=\beta n}^{(1-\beta)n-s}\binom{c+s}{c}\max\left(4^c, 4^{n-c-s}\right)\right).$$

Note that we can assume that $C = C_1$ and $V - C - S$ contains C_2 (we can check this in polynomial time by finding the connected components of $G[V - S]$); since $|C_2| \geq |C_1|$, we can assume that $n - c - s \geq c$. Hence the complexity becomes

$$O^*\left(\sum_{s=1}^{(1-2\beta)n}\sum_{c=\beta n}^{(1-\beta)n-s}\binom{c+s}{c}4^{n-c-s}\right) = O^*\left(\max_{s=1}^{(1-2\beta)n}\max_{c=\beta n}^{(1-\beta)n-s}\binom{c+s}{c}4^{n-c-s}\right).$$

Now denote $d = n - c - s$, then $c + s = n - d$ and we can rewrite the complexity as

$$O^*\left(\max_{d=\beta n}^{(1-\beta)n}\max_{c=\beta n}^{n-d}\binom{n-d}{c}4^d\right).$$

For any d, the maximum of $\binom{n-d}{c}$ over $c \geq \beta n$ can be one of two cases: if $\beta n \leq \frac{n-d}{2}$, it is equal to $\Theta^*(2^{n-d})$; otherwise it is equal to $\binom{n-d}{\beta n}$. In the first case, for the interval $c \in [\beta n, \frac{n-d}{2}]$, the function being maximized becomes $2^{n-d}4^d = 2^{n+d}$. Since this function is increasing in d, its maximum is covered by the second case with the smallest c such that $c = \frac{n-d}{2}$ (in case $\beta n \leq \frac{n-d}{2}$). Therefore, the complexity of Stage 2 of the algorithm becomes

$$O^*\left(\max_{d=\beta n}^{(1-\beta)n}\binom{n-d}{\beta n}4^d\right). \quad (2)$$

Now we are searching for the optimal $\beta \in \left[0, \frac{1}{2}\right]$ that balances the complexities (1) and (2). We solve it numerically and obtain $\beta \approx 0.38685$, giving complexity $O^*(2.61508^n)$. ◀

Proof of Theorem 4. First, we look at the time complexity. The time complexity of Stage 1 now is equal to

$$O^*\left(\sum_{c=0}^{\beta n} 2^{n-c}2^c\right) = O^*(2^n).$$

The time complexity of Stage 2 is equal to

$$O^*\left(\max_{d=\beta n}^{(1-\beta)n}\binom{n-d}{\beta n}2^d\right) = O^*\left(2^{n-d}2^d\right) = O^*(2^n).$$

The space complexity of Stage 1 is equal to

$$O^*\left(\max_{c=0}^{\beta n} 2^c\right) = O^*\left(2^{\beta n}\right).$$

The space complexity of Stage 2 is equal to

$$O^*\left(\max_{d=\beta n}^{(1-\beta)n} 2^d\right) = O^*\left(2^{(1-\beta)n}\right).$$

Therefore, the time complexity of this algorithm is $O^*(2^n)$ and, taking $\beta = \frac{1}{2}$, the space complexity is equal to $O^*\left(\sqrt{2^n}\right)$. ◀

Proof of Theorem 1. In Algorithm 1, we replace the algorithms from Lemmas 7 and 20 with the quantum algorithm from Lemma 8; the algorithm from Theorem 13 is replaced with the algorithm from Theorem 14. Since all exponential subprocedures now are sped up quadratically, the time complexity becomes

$$O\left(\sqrt{2.61508^n}\right) = O(1.61713^n).$$

The space complexity is still polynomial, as Grover's search additionally uses only polynomial space. ◀

Proof of Theorem 2. The time complexity of the first stage is now equal to

$$O^*\left(\sum_{c=0}^{\beta n} \sqrt{2^{n-c}} \cdot 1.816905^c\right) = O^*\left(\sqrt{2^n} \cdot 1.28475^{\beta n}\right).$$

For the second stage, the time is given by

$$O^*\left(\max_{d=\beta n}^{(1-\beta)n} \sqrt{\binom{n-d}{\beta n}} \cdot 1.816905^d\right).$$

We can numerically find that $\beta \approx 0.3755$ balances these complexities, which then are both $O(1.55374^n)$. The space complexity is

$$O^*\left(1.816905^{\max(\beta n,(1-\beta)n)}\right) = O^*\left(1.816905^{(1-\beta)n}\right) = O(1.45195^n).$$ ◀