# *Quantum Algorithms for the Steiner Tree Problem*

Gerald Huang

2023, Term 3

# Abstract

The STEINERTREE problem is a well-studied computational problem that is known to be NP-complete. Roughly speaking, given a subset $S \subseteq V$ of vertices called the *terminal vertices*, the problem is to compute the minimum sized tree that connects all of the terminal vertices. Its fastest known classical algorithm runs in $O^*(2^k)$ [11], parameterised by the size of the subset $S$. In this thesis, we give a quantum speedup by appealing to the classical algorithm only for small subsets. For large subsets, we use Grover's search and a tree decomposition to decompose the terminal vertex set into approximately even-sized subsets. We show that this matches the quantum algorithm established in [1] for SETCOVER, and conclude with a few comments about its relationship with the SETCOVER problem.

# Acknowledgements

Throughout this journey, my undying gratitude goes to my supervisor Prof. Serge Gaspers for his continuing support and discussions about the topic. I would like to also extend my gratitudes to Dr. Mashbat Suzuki for the further discussions during our meetings. This journey has been an endless wave fueled by curiosity and a propensity for learning, culminating in a thesis project that I am forever grateful in participating in. The knowledge and experience that I have gained from this project has been, and will be, a driving force for my future works.

I would also like to thank my friends and family for their unwavering support throughout this research year. Without them to cheer me on in the background, I would not have the encouragement and motivation to keep working through the reports. In particular, I would like to thank Liam Heng for his words of wisdom, Stephanie Tong for constantly keeping me in check, Olivia Kosasih for her academic advice throughout the year, Ethan Brown for the many satirically hilarious messages over the past year, and many more people whom I have befriended over the past year or so.

I would also like to thank my assessor Dr. Abdallah Saffidine for his comments and constructive criticisms throughout my thesis. They have undoubtedly helped me in writing more succinctly in both, my presentations and thesis.

– Gerald Huang, z5209342

# Contents

# Chapter 1

# Introduction

The *Steiner tree* problem is a generalisation of the shortest path and minimum spanning tree problem. We define the problem formally.

**Definition 1.** Let $G$ be an undirected and unweighted graph, and let $S \subseteq V$ be a subset of vertices. A *Steiner tree* is a subtree of $G$ that connects all of the vertices in $S$.

The natural question that we may ask is: *what is the smallest number of edges required to form a Steiner tree?*. We will study the decision variant, which we call STEINERTREE as follows.

| STEINERTREE |
| --- |
| **Instance**: An undirected and unweighted graph $G = (V, E)$ and an integer $\ell$. |
| **Task**: Does $G$ have a Steiner tree with at most $\ell$ edges? |

The decision problem is known to be NP-complete, so we do not expect any efficient algorithm to solve the problem exactly. Therefore, all of the algorithms that we will study in this thesis will have an exponential-time analysis.

We make a few simplification observations on the structure of the graph before assuming these structures for the rest of the paper.

**Observation 1.** Let $G = (V, E)$ be an undirected and edge-weighted graph. If $G$ is disconnected and any vertex in $S$ is disconnected from any other vertex in $S$, then no such Steiner tree exists.

That is, if $u, v \in S$ such that there exist no path from $u$ to $v$, then we know immediately that no such Steiner tree can exist. This is easy to check in polynomial-time via a breadth-first search. Therefore, such a graph must at least connect all of the terminal vertices.

**Observation 2.** Let $G = (V, E)$ be an undirected and edge-weighted graph. If $G$ is disconnected, then any vertex that is not connected to any terminal vertex may be ignored.

These vertices will never form part of our Steiner tree; therefore, it is enough to ignore these vertices altogether. For the rest of the paper, we may assume that $G$ is connected.

## 1.1 Special Cases

We first examine special cases which admit polynomial-time algorithms before exploring the more general problem.

**Observation 3.** If $|S| = 1$, then the *Steiner Tree* problem is trivial.

The vertex $v \in S$ forms a Steiner tree.

**Observation 4.** If $|S| = 2$, then the *Steiner Tree* problem reduces to a shortest path problem.

Note that the *Steiner Tree* problem is to compute the minimum cost tree such that all vertices in $S$ are connected. However, if $|S| = 2$, say $u, v \in S$, this reduces to computing the minimum cost tree such that $u$ and $v$ are connected, which is the shortest path.

**Observation 5.** If $S = V$, then the *Steiner Tree* problem reduces to a minimum spanning tree problem.

# Chapter 2

# Preliminaries

## 2.1 Nomenclature and Notation

We begin with the standard notation of $O^*$. $O^*$ acts like $O$, except it hides polynomial factors. Similarly, $\widetilde{O}$ hides polylogarithmic factors. We will also extensively make use of Stirling's approximation, which we state but not prove.

**Proposition 1** (Stirling's approximation)**.**

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n.$$

This allows us to write binomial coefficients in terms of $n^n$. More specifically, we have that

$$\begin{aligned}\binom{n}{k} &= \frac{n!}{k!(n-k)!} \\ &\sim \frac{\sqrt{2\pi n}(n/e)^n}{\sqrt{2\pi k}(k/e)^k \cdot \sqrt{2\pi(n-k)}((n-k)/e)^{n-k}} \\ &\leq O^*\left(\frac{n^n}{k^k(n-k)^{n-k}}\right).\end{aligned}$$

In other words, we can bound $\binom{n}{k}$ by the above expression in our asymptotic time complexity analysis.

## 2.2 Quantum algorithms

We will make use of the quantum algorithms for searching in an unsorted database and finding the minimum. We state, but do not prove, these results.

### 2.2.1 Grover's search and Quantum Minimum Finding

**Theorem 1** ([9, 2]). *Let $A : \{1, 2, \ldots, N\} \to \{0, 1\}$ be a bounded-error quantum algorithm with running time $T$. Then there exist a bounded-error quantum algorithm that computes*

$$\bigvee_{x \in \{1,\ldots,N\}} A(x)$$

*with running time $\widetilde{O}(\sqrt{N}T)$.*

**Theorem 2** ([5]). *Let $A : \{1, 2, \ldots, N\} \to \{0, 1\}$ be a bounded-error quantum algorithm with running time $T$. Then there exist a bounded-error quantum algorithm that computes*

$$\min_{x \in \{1,\ldots,N\}} A(x)$$

*with running time $\widetilde{O}(\sqrt{N}T)$.*

## 2.3 Graph Contraction

For a graph $G = (V, E)$ and a subset $A \subseteq V$ of vertices, a *graph contraction* is a graph obtained by removing all of the vertices in $A$ and all of its incident edges, replacing the vertices by a new vertex $v_A$, and then adding all of the incident edges back to join $v_A$. We refer to the new graph by $G/A$.

## 2.4 Previous Works

### 2.4.1 The Dreyfus-Wagner Algorithm

In this section, we restrict our attention towards the first well-known exact classical algorithm which solves the *Steiner Tree* problem, the *Dreyfus-Wagner* algorithm. This algorithm exploits the overlapping subproblem property of the problem: if there is an intermediate Steiner tree for a subset $D \cup \{a\}$ for some $D \subset S$ and $a \in S$, then necessarily there exist Steiner trees $S_1$ and $S_2$ connecting $D \cup \{a\}$ and $(S - D - \{a\}) \cup \{a\}$ [4]. Therefore, the original problem can be decomposed into smaller subproblems with each subproblem solved recursively.

We present the algorithm and discuss its overall running time. Since the algorithm utilises shortest paths throughout the algorithm, we shall find the shortest paths for all pairs of vertices in the graph. This has running time $\Theta(n^3)$ via the *Floyd-Warshall* algorithm [6]. For simplicity, denote $d(i, j)$ to be the shortest distance between vertex $i$ and vertex $j$.

To perform the algorithm, we first define a few notations. For a subset $D \subseteq S$ and for each vertex $k \in V$, denote $f_k(D)$ to be the minimum distance by partitioning $D$ into two proper subsets $E, F \subset D$, adding the minimum distance from some vertex $w \in E$ to $k$ and adding the minimum distance from some vertex $w' \in F$ to $k$. $f_k(D)$ is, then, the minimum distance over all distinct choices of sets for $E$ and $F$. We carefully note that $f_k(D)$ is not necessarily the *minimum* distance of a Steiner tree that connects all vertices of $D$ and $k$.

Therefore, having computed $f_k(D)$ for a given subset $D$ and all vertices $k \in V$, we now explore the original problem, this time introducing a vertex $m \in S - D$ not already explored in $D$. Let $f(D, m)$ denote the subproblem of computing the minimum distance Steiner tree that connects all of the vertices in $D \cup \{m\}$. Since $f_k(D)$ does not require information about vertex $m$, this algorithm extends completely to all vertices $m \in V$ of the original graph. To construct the tree that connects $m$ to the currently computed Steiner tree, we examine all trees $f_k(D)$ and connect $m$ to $k$ via the shortest path with length $d(m, k)$. Therefore, the Steiner tree that solves the subproblem $f(D, m)$ should have distance

$$f(D, m) = \min_{k \in V} \{d(m, k) + f_k(D)\}$$

with the final solution being exactly $\min_{k \in S} f(S - k, k)$. Naturally, we solve each subproblem in increasing size of our subsets $D$. To obtain the actual tree, we may store the intermediate vertices $k$. Notice that storing such vertices allows us to iteratively build the tree as we increase the sizes of $D$.

To examine the running time of the algorithm, we observe that we have to compute all subsets of size $i$ for each $i$ from 1 to $|S|$. However, to compute $D$, we have to consider all proper subsets of $D$. For each subset $D$ and vertex $m$, we also have $O(n)$ work, where $n$ is the number of vertices, to check through all vertices $k$. Therefore, we obtain a rough running time estimation of

$$\sum_{i=1}^{|S|} \binom{|S|}{i} 2^i n = O^*(3^{|S|}),$$

where $O^*(\cdot)$ suppresses the polynomial. This gives a general fixed-parameter tractable algorithm for *Steiner tree* with the parameter being the size of $S$. Due to Ambainis et al.[1], many dynamic programming solutions can have a speed up in its running time. In particular, replacing parts of the

dynamic programming aspect with Grover's search leads to an overall quantum improvement in the running time.

### 2.4.2 Existing Quantum Algorithms

[10] shows that the Minimum Steiner tree problem can be solved with a bounded-error quantum algorithm in time $O^*(1.812^k)$, using techniques from [1] and the tree decomposition which we discuss in Section 3.1.3. In my thesis, we use the same techniques but apply it on unweighted graphs, or equivalently on graphs with unit weight. This allows for a faster preprocessing algorithm, utilising the inclusion-exclusion formulation of STEINERTREE.

# Chapter 3

# The Algorithm

Our algorithm comes in two stages: for trees with small number of terminal vertices, we can compute it using Yates' algorithm. For trees with a large number of terminal vertices, we apply Grover's search.

## 3.1 Preprocessing Small Terminal Sets

### 3.1.1 The Inclusion-Exclusion Algorithm

For *small* enough terminal vertices, we can afford to run a classical algorithm. We define this more formally in this section. We build on the work developed by Nederlof [11]. Let $\alpha \in (0, 1/2)$, and call a subset $X \subseteq S$ *$\alpha$-small* if $|X| \leq \alpha|S|$. We now describe a general algorithm to determine if a given graph $G$ has a Steiner tree with at most $k$ edges.

We firstly appeal to the generalised *inclusion-exclusion principle*.

**Proposition 2** (Folklore)**.** *Let $\mathcal{U}$ denote some universal set, and suppose that $A_1, \ldots, A_n \subseteq \mathcal{U}$. Then*

$$\left|\bigcap_{i=1}^{n} A_i\right| = \sum_{I \subseteq \{1,\ldots,n\}} (-1)^{|I|} \cdot \left|\bigcap_{i \in I} \overline{A_i}\right|,$$

*where we define $\bigcap_{i \in \emptyset} A_i = \mathcal{U}$ by convention.*

We show how this can be used to solve the Steiner tree problem.

**Definition 2.** A tree $T$ is *rooted* if there exist a vertex $v \in V(T)$ has been assigned as the "root" of the

tree. An *ordered* tree is a rooted tree when an ordering is specified for the children of each vertex.

Given an ordered tree (and thus, a rooted tree), we have the following definition.

**Definition 3** (Branching walks)**.** Let $G = (V(G), E(G))$ be a graph and $H = (V(H), E(H))$ an ordered tree. Let $f : V(H) \to V(G)$ be a homomorphism from $H$ to $G$; that is, if $(u, v) \in E(H)$, then $(f(u), f(v)) \in E(G)$. We denote the tuple $(H, f)$ to be a branching walk.

We denote the *length* of a branching walk $(H, f)$ by the number of edges $E(H)$ in $H$, and say that $H$ *starts from* $s \in V(G)$ if $f(v) = s$, where $v$ is the root of $H$. Finally, a branching walk *visits* a vertex $v \in V(G)$ if there exist some vertex $w \in V(H)$ such that $f(w) = v$.

We also make it clear that two branching walks $(H_1, f_1)$ and $(H_2, f_2)$ are distinct if there does not exist an isomorphism $\psi : H_1 \to H_2$ such that $f_1(v) = f_2(\psi(v))$ for each $v \in V(H_1)$. We are now ready to reformulate the Steiner tree problem.

For a terminal vertex $s \in V(G)$, it turns out that a branching walk starting at $s$ of length at most $\ell$ visiting every terminal vertex in $G$ is equivalent to the existence of a Steiner tree of length at most $\ell$. Therefore, we may instead compute the number of such branching walks of length at most $\ell$. If the number of such branching walks is not zero, then a Steiner tree exists. We prove this result with the following lemma.

**Lemma 1.** *Let $s \in V(G)$ be a terminal vertex. There exists a branching walk starting at $s$ of length at most $\ell$ that visits all of the terminal vertices in $G$ if and only if there exists a Steiner tree with at most $\ell$ edges.*

*Proof.* We prove both directions.

- ( $\implies$ ) Let $(H, f)$ be a branching walk of length $\ell$ that visits every terminal vertex in $G$. This corresponds to a connected subgraph of $G$. If we let $T$ be a spanning tree of $(f(V(T)), f(E(T)))$, then this will have all of the properties of a Steiner tree with at most $\ell$ edges.

- ( $\impliedby$ ) Let $T$ be a Steiner tree with at most $\ell$ edges. Let a branching walk be $B = (T, f)$ where $f : V(T) \to V(T)$ is the identity function. For an arbitrary terminal vertex $s \in V(G)$, let $s$ be the root of $B$. Then $B$ is a valid branching walk of length at most $\ell$ edges visiting every terminal vertex.

□

We now consider the inclusion-exclusion formulation. Let $s \in S$ be an arbitrarily chosen terminal

vertex, and denote $\mathcal{U}$ to be the universe set of *branching walks* $(T, f)$ of length $\ell$ starting from $s$. For each vertex $v \in S$, define $\mathcal{A}_v \subseteq \mathcal{U}$ to be the set of branching walks that visit vertex $v$. It then follows that

$$\bigcap_{v \in S} \mathcal{A}_v$$

is the set of branching walks that visit *every* vertex in $S$ of length at most $\ell$. Thus, if

$$\left| \bigcap_{v \in S} \mathcal{A}_v \right| > 0,$$

then a Steiner tree with at most $\ell$ edges exist by Lemma 1.

By Proposition 2, we have that

$$\left| \bigcap_{v \in S} \mathcal{A}_v \right| = \sum_{X \subseteq S} (-1)^{|X|} \cdot \left| \bigcap_{v \in X} \overline{\mathcal{A}_v} \right|$$

and so, it suffices to compute the right hand side expression. We show that $\left|\bigcap_{v \in X} \overline{\mathcal{A}_v}\right|$ can be solved in polynomial time.

**Theorem 3.** $\left| \bigcap_{v \in X} \overline{\mathcal{A}_v} \right|$ *can be computed in polynomial time.*

To prove this theorem, we first state a few results about ordered trees.

**Lemma 2.** *Let $\mathcal{T}_n$ be the set of all distinct ordered trees on $n$ edges. Then*

$$|\mathcal{T}_n| = \sum_{i=0}^{n-1} |\mathcal{T}_i| \cdot |\mathcal{T}_{n-i-1}| .$$

Let $\mathcal{B}_X(s, \ell)$ be the number of branching walks of length at most $\ell$ starting at $s \in G[V \setminus X]$ in the subgraph $G[V \setminus X]$. From Lemma 2, it follows that

$$\mathcal{B}_X(s, \ell) = \sum_{t \in N(s) \setminus X} \sum_{i=0}^{\ell-1} \mathcal{B}_X(s, i) \cdot \mathcal{B}_X(t, \ell - i - 1).$$

The inner sum is a direct application of the lemma, while the outer sum just enumerates over all neighbourhood vertices. Each distinct neighbourhood vertex acting as the root constructs a distinct pair of branching walks and so, we do not overcount. These results imply Theorem 3.

*Proof of Theorem 3.* By definition of $\mathcal{A}_v$, $\overline{\mathcal{A}_v}$ is the set of all branching walks of length at most $\ell$ which

avoids vertex $v$. Therefore, $\bigcap_{v \in X} \overline{\mathcal{A}_v}$ is the set of all branching walks of length at most $\ell$ which avoids the set $X$. This is equivalent to computing the set of branching walks of length at most $\ell$ in the subgraph $G[V \setminus X]$.

Let $\mathcal{B}_X(s, \ell)$ denote the number of branching walks of length at most $\ell$ starting from vertex $s \in (V \setminus X) \cap S$. From Lemma 2 and its corollary, we have that

$$\mathcal{B}_X(s, \ell) = \sum_{t \in N(s) \setminus X} \sum_{i=0}^{\ell-1} \mathcal{B}_X(s, i) \cdot \mathcal{B}_X(t, \ell - i - 1).$$

The base case occurs with $\mathcal{B}_X(s, 0) = 1$ and $\mathcal{B}_X(s, i) = 0$ for all $i < 0$. Computing $\mathcal{B}_X(s, \ell)$ is clearly a polynomial-time dynamic programming algorithm: each subproblem can be stored and accessed in $O(1)$ time and there are only polynomial many calls to the look-up table. Therefore, $\mathcal{B}_X(s, \ell)$ can be computed in polynomial-time.

We observe that the simplified problem of computing $\mathcal{B}_X(s, \ell)$ is equivalent to computing the number of branching walks of length at most $\ell$ starting from vertex $s$ in the graph $G[V \setminus X]$, which is equivalent to computing the number of branching walks that avoid $X$. In other words, we have that

$$\left| \bigcap_{v \in X} \overline{\mathcal{A}_v} \right| = \mathcal{B}_X(s, \ell).$$

Thus, $\left| \bigcap_{v \in X} \overline{\mathcal{A}_v} \right|$ can be computed in polynomial-time. □

With this inclusion-exclusion formulation, $\left| \bigcap_{v \in X} \mathcal{A}_v \right|$ can be computed in $O^*(2^k)$ time. In the next section, we will describe a method to effectively solve

### 3.1.2 Yates' Algorithm

**Definition 4.** Suppose that $\alpha \in (0, 1/2)$ is fixed. Let $U$ be some universe set and $R$ an algebraic ring, and let $f : \mathcal{P}(U) \to R$ where $\mathcal{P}(U)$ denotes the power set of $U$. For each $\alpha$-small subset $X \subseteq U$, we denote the *$\alpha$-small zeta transform* as

$$f\zeta_\alpha = \sum_{Y \subseteq X} f(Y),$$

where $|X| \leq \alpha |U|$.

**Theorem 4.** *The $\alpha$-small zeta transform $f\zeta_\alpha$ can be computed in time $O^*\left(\binom{|U|}{\alpha|U|}\right)$.*

*Proof.* Our algorithm follows very closely to Yates' algorithm [12]. We firstly fix some ordering of $U = \{u_1, \ldots, u_n\}$ and then consider each element in rounds, increasing in cardinality. Consider some $\alpha$-small subset $X \subseteq U$.

In the first round, we set $g_0(X) = f(X)$. We then iterate over the elements of $U$, one at a time in the order it appears in $U$. In particular, when processing $u_i$, we let

$$g_i(X) = g_{i-1}(X) + \begin{cases} 0 & \text{if } u_i \notin X, \\ g_{i-1}(X \setminus \{u_i\}) & \text{if } u_i \in X. \end{cases}$$

Our final solution arrives by setting $f\zeta_\alpha(X) = g_n(X)$.

Correctness follows from induction, which can additionally be found in [8]. □

We will find the optimal $\alpha$ in Section 3.3 but for now, we will keep $\alpha$ general throughout the algorithm.

### 3.1.3 A Tree Decomposition

To effectively use Yates' algorithm as part of the algorithm, we need a way to subdivide a larger Steiner tree into subtrees or subforests of strictly smaller terminal size. To this end, we introduce a decomposition developed by Fuchs et al. [7]

**Definition 5.** Let $T$ be a tree. A 2-*split* of $T$ is a partition

$$T = T_1 \cup T_2$$

such that $T_1$ and $T$ are both connected.

We use this decomposition on minimum Steiner trees. In particular, if we let $T$ be a *minimum Steiner tree* for the terminal set $X \subseteq S$, then $T_1$ is a subtree and $T_2$ is a subforest, consisting of a collection of components. Additionally, we define the following sets

$$A \coloneqq V(T_1) \cap V(T_2)$$
$$X_1 \coloneqq X \cap V(T_1) \setminus A;$$
$$X_2 \coloneqq X \cap V(T_2) \setminus A.$$

We call $A$ the set of *split vertices*, and these are good candidates to connect smaller Steiner trees that we

either split or compute classically. This kind of decomposition turns out to be useful because being able to split the terminal vertex set into two sets allows us to more easily apply the quantum Grover search algorithm more effectively. If $|A|$ is *small enough*, then $X_1$ and $X_2$ becomes a reasonably good partition of the terminal vertex set $X$. Therefore, we would like to eventually bound the size $|A|$.

Moreover, this decomposition maintains the *minimum Steiner tree* property on smaller subsets as long as a few conditions are met. We state, but do not prove, these results below.

**Lemma 3.** *Let $T$ be a minimum Steiner tree for $X \subseteq S$, and let $T = T_1 \cup T_2$ be a 2-split. Then the following properties hold.*

- *The tree $T_1$ is a minimum Steiner tree for $X_1 \cup A$.*
- *The tree $T_2/A$ is a minimum Steiner tree for $X_2 \cup \{v_A\}$,*

*where $T_2/A$ denotes the graph generated by contracting the vertex set $A$ and replacing the vertices of $A$ with the vertex $v_A$.*

This allows us to define a suitable recurrence to compute the weight of a minimum Steiner tree. Here, the weight of a minimum Steiner tree refers to the number of edges in such a tree.

The next theorem shows us the existence of such a 2-split decomposition.

**Theorem 5.** *For any $\epsilon > 0$ and $0 < \beta < 1$, any Steiner tree $T$ for $X \subseteq S$ with $k = |X|$ sufficiently large admits a 2-split $T = T_1 \cup T_2$ such that:*

- $(\beta - \epsilon)k \leq |X'| = |X \cap V(T_1)| \leq (\beta + \epsilon)k$.
- $|A| \leq \lceil \log \frac{1}{\epsilon} \rceil$.

For each fixed $\epsilon$, the size of $A$ is bounded by a constant; therefore, we can compute each such set $A$ in polynomial-time. The useful property in Theorem 5 is that such a 2-split is *guaranteed* to exist given the conditions, so we obtain the following recurrence.

Let $W_G(X)$ be the number of edges in a minimum Steiner tree in $G$ for the terminal vertex set $X$. Then for any $\epsilon > 0$ and $0 < \beta \leq 1/2$, Lemma 3 and Theorem 5 gives us the following recurrence

$$W_G(X) = \min_{\substack{X' \subseteq X, \\ (\beta-\epsilon)k \leq |X'| \leq (\beta+\epsilon)k}} \min_{\substack{A \subseteq V(T), \\ |A| \leq \lceil \log \frac{1}{\epsilon} \rceil}} \left\{ W_G(X_1 \cup A) + W_{G/A}(X_2 \cup \{v_A\}) \right\}. \tag{1}$$

In our final solution, we see that if $W_G(S) \leq \ell$, then a Steiner tree with at most $\ell$ edges exist. On the other hand, if $W_G(X) > \ell$, then the minimum Steiner tree must have more than $\ell$ edges. Therefore, computing $W_G(S)$ solves STEINERTREE.

### 3.1.4 The Classical Algorithm: A Summary

To not lose sight of the forest among the trees, we summarise the classical algorithm in this section. We begin with the inclusion-exclusion formulation. A Steiner tree of length at most $\ell$ exists if and only if there exist a branching walk of length at most $\ell$ that starts from a terminal vertex $s \in S$ that visits every terminal vertex. Defining $\mathcal{A}_v$ to be the set of all branching walks of length at most $\ell$ that visit vertex $v$, we see that the Steiner tree problem is equivalent to determining

$$\left| \bigcap_{v \in S} \mathcal{A}_v \right| > 0.$$

By the inclusion-exclusion principle, we see that this is equivalent to determining whether

$$\sum_{X \subseteq S} (-1)^{|X|} \cdot \left| \bigcap_{v \in S} \overline{\mathcal{A}_v} \right| > 0.$$

Computing this summation is classically done in $O^*(2^k)$ with $k = |S|$. Therefore, our aim is to compute this classically for all $|X| \leq \alpha k$ instead. This has running time $O^*\left(\binom{k}{\alpha k}\right)$.

We now explore the quantum part of the algorithm. To this end, we introduced the tree decomposition by Fuchs et al. Indeed, this allows us to split a tree with terminal vertex $X$ into two sets $X_1$ and $X_2$. The 2-split ensures that the size of $X_1$ and $X_2$ are *approximately* equal. In what follows, we will run Grover's search on the recurrence derived in (1).

## 3.2 The quantum part

For Steiner trees with a large enough terminal vertex set, we run Grover's search to decompose the tree. To this end, we use the same approach from Ambainis et al.

**Theorem 6.** *There exist a bounded-error quantum algorithm that solves* STEINERTREE *in time* $O^*(1.728^k)$.

We extensively use the tree decomposition in Section 3.1.3 to decompose the terminal vertex set into *approximately* two halves. We run a Grover's search over the two halves three times, the last of which we can directly use the preprocessed sets.

Let $T$ be a minimum Steiner tree. In Section 3.1.3, the terminal vertex set $X$ of a minimum Steiner tree can be decomposed into $X_1 \cup A$ in $G$ and $X_2 \cup \{v_A\}$ in $G/A$ satisfying the condition that:

- $|A| \leq \lceil \log \frac{1}{\epsilon} \rceil$.

Therefore, whenever we run Grover's search, we additionally search over all subsets for $A$ such that $|A| \leq \lceil \log \frac{1}{\epsilon} \rceil$. This overhead cost is relatively minor whenever $\epsilon$ is fixed. If at any point we get $|X| \leq \alpha k$, we can simply access the preprocessed sets instead.

**Algorithm 1** Quantum Algorithm for STEINERTREE

1. For all $|X| \leq ((1-\alpha)/4 + 15\epsilon)k$ and all $A \subseteq V$ such that $|A| \leq \lceil \log \frac{1}{\epsilon} \rceil$, compute $W_G(X \cup A)$ and $W_{G/A}(X \cup \{v_A\})$ using the Inclusion-Exclusion Algorithm.
2. Run quantum minimum finding over all subsets $X \subseteq S$ such that $|X| \leq k/2$ and all $A \subseteq V$ such that $|A| \leq \lceil \log \frac{1}{\epsilon} \rceil$ to find the answer to the equation in (1).
   - To compute $W_G(X \cup A)$ and $W_{G/A}(X \cup \{v_A\})$ for $|X| \leq k/2$, run quantum minimum finding for Equation (1) with $|X'| \leq k/4$ and $X' \subseteq X$ and over all $A' \subseteq A$ such that $|A'| \leq \lceil \log \frac{1}{\epsilon} \rceil$.
   - To compute $W_G(X \cup A)$ and $W_{G/A}(X \cup \{v_A\})$ for $|X| \leq k/4$, run quantum minimum finding for Equation (1) with $|X'| \leq \alpha k/4$ and $X' \subseteq X$ and over all $A' \subseteq A$ such that $|A'| \leq \lceil \log \frac{1}{\epsilon} \rceil$. For any set $X$ such that $|X| \leq \alpha k/4$ or $|X| \leq (1-\alpha)k/4$, we know $W_G(X \cup A)$ and $W_{G/A}(X \cup \{v_A\})$ from the classical preprocessing step.

## 3.3 Analysing the running time

Since $\epsilon$ is a small constant, we will omit its presence in the running time complexity. We will also define $\hat{O}$ to hide all polynomial factors in $n$ and $k$, and all factors of the form $2^{O(\epsilon k)}$.

The classical preprocessing time is

$$\hat{O}\left(\binom{k}{\leq (1-\alpha)k/4}\right).$$

The quantum complexity is

$$\hat{O}\left(\sqrt{\binom{k}{k/2}\binom{k/2}{k/4}\binom{k/4}{\alpha k/4}}\right).$$

The time complexity is minimised when the two parts are equal. Using Stirling's approximation, numerical analysis shows that the optimal choice for $\alpha$ is approximately 0.055362. The running time is then $O^*\left(1.728^k\right)$.

# Chapter 4

# Applications

In this final section, we relate the STEINERTREE problem to other problems commonly studied and showed that other problems also admit a quantum improvement. In particular, we discuss its relation to the SETCOVER problem which has a known lower bound conjecture.

Our first known result is a linear reduction from SETCOVER to STEINERTREE. We first describe the SETCOVER problem.

**Definition 6.** Let $U$ be a universe set, and let $A_1, \ldots, A_k \subseteq U$ be a collection of $k$ sets. We say that $A_1, \ldots, A_k$ *covers* $U$ if every element of $U$ is contained in at least one set $A_i$ for some $i = 1, \ldots, k$. The size of a covering is the number of sets in the collection.

The SETCOVER problem, then, is to determine if there exist a covering of $U$ of size at most $k$.

| SETCOVER |
| --- |
| **Instance**: A set $U = \{1, \ldots, n\}$ called the *universe set*, a collection of sets $A_1, \ldots, A_t \subseteq U$, and an integer $k$. |
| **Task**: Is there a covering of $U$ of size at most $k$? |

**Theorem 7.** SETCOVER *reduces to* STEINERTREE.

Consider an instance $(U, \mathcal{F}, k)$ of SETCOVER. We construct an instance of STEINERTREE as follows:

- Let $G = (V, E)$ with $V = U \cup \mathcal{F} \cup \{t_0\}$ and $E = \{\{u, F : u \in F,\ F \in \mathcal{F}\}$.
- $S = U \cup \{t_0\}$.

Then the instance $(U, \mathcal{F}, k)$ of SETCOVER translates to the instance $(G = (V, E), S, k)$ of STEINERTREE.

We observe that the parameter of the problems reduce linearly: $|U| \to |U| + 1$. Therefore, a quantum algorithm for STEINERTREE gives a quantum algorithm for SETCOVER with the same time complexity. Interestingly, the time complexity of the quantum algorithm established by [1] matches exactly with the time complexity of our algorithm. Additionally, a faster quantum algorithm for STEINERTREE would imply a faster quantum algorithm for SETCOVER.

The next result is the *set cover conjecture*, a conjecture established by Cygan et al. [3] Approximately speaking, the conjecture posits that there is no algorithm that can solve SETCOVER faster than $O^*(2^n)$. By the reduction from Theorem 7, this would imply that STEINERTREE cannot be solved faster than $O^*(2^k)$ with $k = |S|$.

In the original formulation of the conjecture, we define a family of special cases. Let $\Delta$-SETCOVER denote the SETCOVER problem where each set $A_i$ is restricted to having size at most $\Delta > 0$. Therefore, SETCOVER is the unrestricted case.

**Conjecture 1** (Set Cover Conjecture)**.** For every fixed $\epsilon > 0$, there exist some $\Delta(\epsilon) > 0$ such that no algorithm (even randomised) solves $\Delta$-SETCOVER in time $O^*(2^{(1-\epsilon)n})$.

Under this hardness assumption, several other lower bounds have been proposed.

**Theorem 8.** *Assume that, for all $\epsilon < 1$, there exist some $t$ such that* SETCOVER *with sets of size at most $t$ cannot be computed in time $2^{\epsilon n}poly(n)$. Then, for all $\epsilon < 1$, we have that*

- STEINERTREE *cannot be computed in time* $2^{\epsilon k}\,\mathrm{poly}(n)$.
- CONNECTEDVERTEXCOVER *cannot be computed in time* $2^{\epsilon k}\,\mathrm{poly}(n)$.
- SETPARTITIONING *cannot be computed in time* $2^{\epsilon n}\,\mathrm{poly}(m)$.
- SUBSETSUM *cannot be computed in time* $t^{\epsilon}\,\mathrm{poly}(n)$

## 4.1 Summary and Open Problems

Since there are close relationships between the STEINERTREE and the SETCOVER problems, one direction is to prove that the $O^*(1.728^k)$ bound is tight which would imply that our algorithm is optimal. Indeed, this would also have implications on the lower bound for many other problems such as CONNECTEDVERTEXCOVER and SETPARTITIONING. For problems which admit a linear parameterised reduction from and to SETCOVER, it is convincing that these problems also admit a quantum algorithm with running time $O^*(1.728^{\mathrm{param}})$.

It is presently open whether the set cover conjecture holds: if it holds, then, indeed, the preprocessing algorithm should hold optimally. This gives greater hope that the quantum algorithm is, indeed, optimal.

# Bibliography


[1] Andris Ambainis et al. "Quantum speedups for exponential-time dynamic programming algorithms". In: *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms* (2019), pp. 1783–1793. DOI: 10.1137/1.9781611975482.107.

[2] Michel Boyer et al. "Tight Bounds on Quantum Searching". In: *Fortschritte der Physik* 46.4–5 (1999), pp. 493–505.

[3] Marek Cygan et al. "On Problems as Hard as CNF-SAT". In: *ACM Transactions on Algorithms* 12.3 (2016), pp. 1–24. DOI: 10.1145/2925416.

[4] S.E. Dreyfus and R.A. Wagner. "The Steiner Problem in Graphs". In: *Networks* 1.3 (1971), pp. 195–207. DOI: 10.1002/net.3230010302.

[5] Christoph Dürr and Peter Høyer. "A Quantum Algorithm for Finding the Minimum". In: (1996).

[6] Robert E. Floyd. "Algorithm 97: Shortest path". In: *Communications of the ACM* 5.6 (1962). DOI: 10.1145/367766.368168.

[7] Bernard Fuchs, Walter Kern, and Xinhui Wang. "Speeding up the Dreyfus-Wagner algorithm for minimum Steiner trees". In: *Discrete Mathematics and Mathematical Programming* (2007).

[8] Serge Gaspers and Jerry Zirui Li. "Quantum Algorithms for Graph Coloring and other Partitioning, Covering, and Packing Problems". In: (2023).

[9] Lov K. Grover. "A fast quantum mechanical algorithm for database search". In: *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing* (1996). DOI: 10.1145/237814.237866.

[10] Masayuki Miyamoto et al. "Quantum Speedup for the Minimum Steiner Tree Problem". In: (2020). DOI: 10.48550/arXiv.1904.03581.

[11] Jesper Nederlof. "Fast Polynomial-Space Algorithms Using Mobius Inversion: Improving on Steiner Tree and Related Problems". In: *Automata, Languages and Programming* (2009).

[12] Frank Yates. "The design and analysis of factorial experiments". In: *Harpenden Imperial Bureau of Soil Science* 5 (1937).