Sublinear Metric Steiner Tree via Improved Bounds for Set Cover

Sepideh Mahabadi
Microsoft Research, Redmond, WA, USA

Mohammad Roghani
Stanford University, CA, USA

Jakub Tarnawski
Microsoft Research, Redmond, WA, USA

Ali Vakilian
Toyota Technological Institute at Chicago (TTIC), Il, USA

Abstract

We study the metric Steiner tree problem in the sublinear query model. In this problem, for a set of n points V in a metric space given to us by means of query access to an $n \times n$ matrix w, and a set of terminals $T \subseteq V$, the goal is to find the minimum-weight subset of the edges that connects all the terminal vertices.

Recently, Chen, Khanna and Tan [SODA'23] gave an algorithm that uses $\widetilde{O}(n^{13/7})$ queries and outputs a $(2-\eta)$-estimate of the metric Steiner tree weight, where $\eta > 0$ is a universal constant. A key component in their algorithm is a sublinear algorithm for a particular set cover problem where, given a set system $(\mathcal{U}, \mathcal{F})$, the goal is to provide a multiplicative-additive estimate for $|\mathcal{U}| - \mathsf{SC}(\mathcal{U}, \mathcal{F})$. Here $\mathcal{U}$ is the set of elements, $\mathcal{F}$ is the collection of sets, and $\mathsf{SC}(\mathcal{U}, \mathcal{F})$ denotes the optimal set cover size of $(\mathcal{U}, \mathcal{F})$. In particular, their algorithm returns a $(1/4, \varepsilon \cdot |\mathcal{U}|)$-multiplicative-additive estimate for this set cover problem using $\widetilde{O}(|\mathcal{F}|^{7/4})$ membership oracle queries (querying whether a set $S \in \mathcal{S}$ contains an element $e \in \mathcal{U}$), where ε is a fixed constant.

In this work, we improve the query complexity of $(2-\eta)$-estimating the metric Steiner tree weight to $\widetilde{O}(n^{5/3})$ by showing a $(1/2, \varepsilon \cdot |\mathcal{U}|)$-estimate for the above set cover problem using $\widetilde{O}(|\mathcal{F}|^{5/3})$ membership queries. To design our set cover algorithm, we estimate the size of a random greedy maximal matching for an auxiliary multigraph that the algorithm constructs implicitly, without access to its adjacency list or matrix. Previous analyses of random greedy maximal matching have focused on simple graphs, assuming access to their adjacency list or matrix. To address this, we extend the analysis of Behnezhad [FOCS'21] of random greedy maximal matching on simple graphs to multigraphs, and prove additional properties that may be of independent interest.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Sublinear Algorithms, Steiner Tree, Set Cover, Maximum Matching, Approximation Algorithm

Digital Object Identifier 10.4230/LIPIcs.ITCS.2025.74

Related Version *Full Version*: https://arxiv.org/abs/2411.09059

Acknowledgements This work was done while Mohammad Roghani was an intern at Microsoft Research. The work was conducted in part while Sepideh Mahabadi and Ali Vakilian were long-term visitors at the Simons Institute for the Theory of Computing as part of the Sublinear Algorithms program.

1 Introduction

In the Steiner tree problem, we are given an undirected graph $G = (V, E)$, where each edge e has an associated cost $w(e)$, and a specified set of terminal vertices $T \subseteq V$. Then, the objective is to find a minimum-cost subgraph H of G that connects all terminals in T. The Steiner tree problem is one of the most fundamental problems in combinatorial optimization and has been extensively studied by the TCS community since it was included among Karp's 21 NP-Complete problems [34]. The state-of-the-art approximation factor for the Steiner tree problem is $\ln 4 + \varepsilon < 1.39$ [14], and it is known that approximating it to a factor better than 96/95 is NP-hard [17]. This Steiner tree problem has been studied in various domains, including approximation algorithms [40, 14], online algorithms [29, 3, 37, 26], stochastic algorithms [27, 25, 22], and massive data analysis models [15, 18, 16].

▶ **Definition 1** (Sublinear Metric Steiner Tree). *In the metric Steiner tree problem, we are given a set of points V, a set of terminal points $T \subseteq V$, and query access to an oracle $\mathcal{O}$ to the $|V| \times |V|$ distance matrix of a metric space (V, w), where $\mathcal{O}(u, v)$ returns the weight $w(u, v)$ of the edge (u, v).*

Let $\mathsf{ST}(V, T, w)$ denote the weight of a minimum-weight Steiner tree on instance (V, T, w). Then, the goal is to design an algorithm that estimates $\mathsf{ST}(V, T, w)$ using the fewest possible queries to the distance matrix via the oracle $\mathcal{O}$.

Czumaj and Sohler [18] presented the first sublinear query algorithm for the metric Steiner tree problem, showing a $(2 + \varepsilon)$-approximation using $\tilde{O}(k/\varepsilon^{O(1)})$ queries through their improved algorithm for the minimum spanning tree (MST) problem. Specifically, this follows their sublinear $(1 + \varepsilon)$-approximation for MST together with the well-known result by Gilbert and Pollak [23] showing that an α-approximation for MST over the metric induced on the terminals T is a (2α)-approximation for the metric Steiner tree instance with T as the terminal set.

Recently, Chen, Khanna, and Tan [16] studied the design of sublinear algorithms with strictly better-than-2 approximation for the metric Steiner tree problem. On the lower bound side, they showed that for any $\varepsilon > 0$, estimating the Steiner tree cost to within a $(5/3 - \varepsilon)$-factor requires $\Omega(n^2)$ queries, even when the number of terminals $|T|$ is constant. Moreover, they showed that for any $\varepsilon > 0$, estimating the Steiner tree cost to within a $(2 - \varepsilon)$-factor requires $\Omega(n + |T|^{6/5})$ queries. Additionally, they proved that for any $0 < \varepsilon < 1/3$, any algorithm that outputs a $(2 - \varepsilon)$-approximate Steiner tree (not just its cost) requires $\Omega(n|T|)$ queries. On the upper bound side, they showed that it is possible to achieve a better-than-2 estimate of the Steiner tree cost in sublinear time: there exists an algorithm that, with high probability, computes a $(2 - \eta)$-approximation of the Steiner tree cost using $\widetilde{O}(n^{13/7})$ queries, where $\eta > 0$ is a universal constant. At the core of their sublinear algorithm for metric Steiner tree with improved approximation guarantee, they relate the problem of achieving a better-than-2 estimation for the Steiner tree to a variant of set cover problem with a different objective.

▶ **Definition 2** (Threshold Set Cover). *Given a universe of elements $\mathcal{U}$ and a collection $\mathcal{F}$ of subsets of $\mathcal{U}$, in the Threshold Set Cover problem the goal is to estimate $\mathsf{ThSC}(\mathcal{U}, \mathcal{F}) \coloneqq |\mathcal{U}| - \mathsf{SC}(\mathcal{U}, \mathcal{F})$, where $\mathsf{SC}(\mathcal{U}, \mathcal{F})$ denotes the size of an optimal set cover solution for $(\mathcal{U}, \mathcal{F})$, i.e., the minimal number of sets in $\mathcal{F}$ whose union equals $\mathcal{U}$.*

Following the notation of [16] and for simplicity, in our technical sections, we also refer to this problem as set cover.

Specifically, given access to the adjacency matrix of the graph representation of $(\mathcal{U}, \mathcal{F})$, where there is an edge between $e \in \mathcal{U}$ and $S \in \mathcal{F}$ if and only if $e \in S$, Chen, Khanna, and Tan [16] designed an algorithm that, for any constant $0 < \varepsilon < 1$, with high probability, outputs a *multiplicative-additive* $(1/4, \varepsilon|\mathcal{U}|)$-approximation for estimation of $\mathsf{ThSC}(\mathcal{U}, \mathcal{F})$ using $\widetilde{O}_\varepsilon(|\mathcal{F}|^{3/2} + |\mathcal{F}|^{3/4} \cdot |\mathcal{U}|)$ queries to the adjacency matrix (or, membership queries). An estimate SOL for Threshold Set Cover on $(\mathcal{U}, \mathcal{F})$ is a *multiplicative-additive* (γ_1, γ_2)-approximation, if $\gamma_1 \cdot \mathsf{ThSC}(\mathcal{U}, \mathcal{F}) - \gamma_2 \leq \mathrm{SOL} \leq \mathsf{ThSC}(\mathcal{U}, \mathcal{F})$.

More broadly, there has been a large body of work on solving set cover problems in the massive data models of computation over the past decade [41, 19, 28, 20, 2, 30, 1, 5, 31, 24]. In particular the work of [31, 24] consider the set cover problem in the sublinear query model. However their algorithms assumes that it has an access to the adjacency list model as opposed to the adjacency matrix model, and thus cannot be directly employed here.

1.1 Our Results

Our key contribution is an algorithm for Threshold Set Cover, offering improved approximation guarantees and query complexity, as detailed below:

▶ **Theorem 3** (Our Algorithm for Threshold Set Cover). *There exists an algorithm that, given a set system $(\mathcal{U}, \mathcal{F})$ with oracle access to its adjacency matrix (also known as membership queries), outputs a multiplicative-additive $(1/2, \varepsilon \cdot |\mathcal{U}|)$-approximation to Threshold Set Cover, in $\widetilde{O}(|\mathcal{F}|^{5/3})$ time, with high probability.*

Note that both the query complexity and the running time of the algorithm are bounded by $\widetilde{O}(|\mathcal{F}|^{5/3})$, improving upon the algorithm by Chen, Khanna, and Tan [16] for large values of $|\mathcal{U}|$, which uses $\widetilde{O}_\varepsilon(|\mathcal{F}|^{3/2} + |\mathcal{F}|^{3/4} \cdot |\mathcal{U}|)$ membership queries and provides a multiplicative-additive $(1/4, \varepsilon|\mathcal{U}|)$-approximation for the problem. Notably, when $|\mathcal{U}| = \omega(|\mathcal{F}|^{2/3})$, the algorithm becomes sublinear in $|\mathcal{U}| \cdot |\mathcal{F}|$, making it especially relevant for applications in the metric Steiner tree problem. More specifically, our new algorithm for Threshold Set Cover results in the following improved sublinear query algorithm for the metric Steiner tree problem, which we show in Section 5.

▶ **Theorem 4** (Sublinear Algorithm for Metric Steiner Tree). *There exists an algorithm that, given an instance of metric Steiner tree denoted by (V, T, w) with oracle access $\mathcal{O}$ to the distance matrix of (V, w), outputs a $(2 - \eta)$-estimate of $\mathsf{ST}(V, T, w)$ using $\widetilde{O}(n^{5/3})$ queries to $\mathcal{O}$, where $\eta > 0$ is a universal constant, with high probability.*

Notably, the query complexity of our algorithm improves upon the $\widetilde{O}(n^{13/7})$ query complexity of the algorithm of Chen, Khanna, and Tan [16].

For a detailed overview of our technical contribution, see Section 2.

2 Technical Overview

In this section, we provide a brief overview of the technical challenges involved in designing our algorithms. To design a sublinear time algorithm for the Steiner tree problem, we use the framework developed by Chen, Khanna, and Tan [16]. They demonstrated that breaking the 2-approximation barrier for the Steiner tree problem can be reduced to solving an instance of a set cover problem. We refer the reader to Section 4.1 of [16] for details on this reduction.

We denote the variant of the set cover problem as *Threshold Set Cover*. Given a collection of sets $\mathcal{F}$ over a universe of elements $\mathcal{U}$, we aim to estimate the value of $\mathsf{ThSC}(\mathcal{U}, \mathcal{F}) = |\mathcal{U}| - \mathsf{SC}(\mathcal{U}, \mathcal{F})$, where $\mathsf{SC}(\mathcal{U}, \mathcal{F})$ denotes the size of the optimal set cover of the given instance.

To achieve our goal of breaking the 2-approximation barrier for the Steiner tree problem, we need to estimate $\mathsf{ThSC}(\mathcal{U}, \mathcal{F})$ with a $(\gamma, \varepsilon \cdot |\mathcal{U}|)$ multiplicative-additive error, where γ must be a constant and ε is any (small enough) constant. For this problem, we only have access to a membership oracle of the instance, meaning we can query whether a particular element e is in a particular set S or not. Note that this type of access is generally considered more challenging compared to an adjacency list oracle, where the algorithm can access either the ith element of a set S, or the ith set containing an element e. The reason is that if an element is included in only a constant number of sets, the algorithm is required to spend $\Omega(|\mathcal{F}|)$ queries to find just one set that contains the element. Consequently, we cannot use the results from the literature on sublinear set cover [24, 31] because they all rely on an adjacency list access model.

We will now provide an informal, step-by-step description of our algorithm for Threshold Set Cover, highlighting its differences, innovations, and technical challenges in comparison to the algorithm of [16]. For simplicity, in this technical overview we assume that $|\mathcal{F}| = \widetilde{\Theta}(|\mathcal{U}|)$, since this represents the worst-case scenario for the Steiner tree problem. However, our formal proof does not depend on this assumption. We let $n = |\mathcal{F}|$.

First step: sparsification of the Threshold Set Cover instance. The goal of this step is to produce a new instance where each element and each set has a low degree – specifically, where each element is in only a few sets, and each set contains only a few elements. This step is standard in designing sublinear algorithms for the set cover problem for different access models, and a slightly different version of it is also used in the algorithm by [16]. Our slight modification of the sparsification step allows us to relax some constraints in the reduction from the Steiner tree problem to Threshold Set Cover, enabling us to achieve the same query complexity for both problems.

Let $x > 0$ be some constant that we optimize later. Consider a set $S \in \mathcal{F}$ and suppose we randomly sample $\widetilde{O}(n^{1-x})$ elements from the universe and query the membership of all the sampled elements in S. If the size of S is at least $\widetilde{\Omega}(n^x)$, we expect to see a large intersection. Conversely, if the size of S is much smaller, we expect to see a small intersection. If a large intersection exists, we can remove the set S and all its elements from the instance. Since this event occurs at most $\widetilde{O}(n^{1-x})$ times, we can account for the removed elements and sets using the additive error in our estimation. Similarly, we can show that all elements belonging to more than $\widetilde{\Omega}(n^x)$ sets can be covered by a random subcollection of sets of size $\widetilde{O}(n^{1-x})$. Therefore, without loss of generality, by spending $\widetilde{O}(n^{2-x})$ time, we can assume that each set contains at most $\widetilde{O}(n^x)$ elements, and each element is included in at most $\widetilde{O}(n^x)$ sets.

Second step: constructing an auxiliary graph H and estimating the size of its maximum matching. Similar to [16], we construct a graph H with a vertex set where each vertex corresponds to an element of $\mathcal{U}$. We connect two vertices if their corresponding elements appear together in at least one set from $\mathcal{F}$. It is important to note that we do not construct H explicitly, as doing so would be computationally expensive and require $\Omega(n^2)$ time. As shown by [16], if the size of the maximum matching of H is large, it is evident that $\mathsf{ThSC}(\mathcal{U}, \mathcal{F})$ is significantly smaller than $|\mathcal{U}|$. Conversely, if the size of the maximum matching of H is close to zero, then $\mathsf{ThSC}(\mathcal{U}, \mathcal{F})$ is also close to zero. This is sufficient for our purposes, as our goal is to obtain a constant-factor approximation. Intuitively, each matching edge in H indicates that there are two elements that can be covered together, which increases the value of $\mathsf{ThSC}(\mathcal{U}, \mathcal{F})$.

There is extensive literature on estimating the size of maximum matching in sublinear time [4, 6, 8, 7, 9, 12, 13, 33, 36, 38, 39, 43], with significant progress made in recent years. For our application, we use the algorithm of Behnezhad [6] to estimate the size of a random greedy maximal matching (RGMM) of the graph. In summary, this algorithm can estimate the size of the RGMM of a graph in $\widetilde{O}(\bar{d})$ time if given access to the adjacency list of the graph, where $\bar{d}$ denotes the average degree of the graph. We can now use this algorithm as a black box:

- The average degree of H is $\widetilde{O}(n^{2x})$, since each element is in $\widetilde{O}(n^x)$ sets and each set contains $\widetilde{O}(n^x)$ elements.
- Each time the algorithm visits a vertex in H (corresponding to an element), we can spend $\widetilde{O}(n^{1+x})$ time to find its adjacency list in H. This involves first querying all sets that include the element, and then making queries between those sets and all elements.

Therefore, we can simulate the algorithm from [6] in $\widetilde{O}(\bar{d} \cdot n^{1+x}) = \widetilde{O}(n^{1+3x})$ time. By balancing this with the sparsification step, which requires $\widetilde{O}(n^{2-x})$ time, we can set $x = 1/4$ to achieve an algorithm with a running time of $\widetilde{O}(n^{7/4})$. This is essentially the running time of the algorithm by Chen, Khanna, and Tan [16].

Third step: using the algorithm of Behnezhad [6] in a white-box manner. To improve the running time of our algorithm, we need to open up the RGMM algorithm from [6] and utilize its properties to apply it more effectively. The RGMM algorithm is a local algorithm that explores the neighborhood of a given vertex to determine whether it is matched. A key observation is that during each exploration, the algorithm requires a random neighbor of the vertex that has not been explored yet. However, in the previous approach, we constructed the entire adjacency list of the vertex, which is redundant and inefficient. Intuitively, we only need to randomly identify one of the vertex's neighbors in each step.

However, the first challenge we encounter is that we cannot select a neighbor uniformly at random. To illustrate this, consider the following example. Suppose that we have five elements $\mathcal{U} = \{e_1, \ldots, e_5\}$ and three sets: $S_1 = \{e_1, e_2, e_3\}$, $S_2 = \{e_1, e_2, e_4\}$, and $S_3 = \{e_1, e_2, e_5\}$. Suppose that we want to find a random neighbor of e_1 in H. If we first find all sets that include e_1 and then query between those sets and all elements uniformly at random until we find an edge in H, we are likely to see the edge (e_1, e_2) because it appears in all sets. Consequently, the algorithm has a bias towards finding neighbors that appear in more sets with the element.

To overcome this challenge, rather than defining an auxiliary simple graph H, we define an auxiliary multigraph H. In this multigraph, if two elements appear in the same set multiple times, we add an edge for each of those occurrences. Note that the average degree of H remains at most $\widetilde{O}(n^{2x})$. However, the algorithm and analysis for RGMM from [6] are designed for simple graphs. We extend these results to multigraphs and show that we can estimate the size of RGMM for a multigraph, given access to its adjacency list. This extension may be of independent interest and could be useful for tackling other problems in sublinear time. To establish this, we build on the exquisite approach first introduced by Yoshida, Yamamoto, and Ito [43] and further explored in various settings [6, 10, 11]. We employ techniques such as the analysis of the round-complexity of maximal independent sets [21], double-counting arguments to bound the average complexity of RGMM on multigraphs, and others; we encourage the reader to refer to the arXiv version of the paper for further details.

Now, suppose that for each vertex the RGMM algorithm explores in H, we first query all sets to identify those that include the corresponding element. Since the algorithm explores at most $\widetilde{O}(\bar{d})$ vertices in H, this step will cost at most $\widetilde{O}(\bar{d} \cdot n) = \widetilde{O}(n^{1+2x})$ in total. Let v be a vertex that the RGMM algorithm is exploring at the moment. Define $\mathcal{S}_v$ to be the collection

of sets that include element v. Now, if we query uniformly at random between all elements and the collection $\mathcal{S}_v$, each incident edge of v in the multigraph H has an equal probability of being sampled, which resolves the first challenge. For now, assume that the degree of all vertices in H is $\bar{d}$. Since $|S_v| = \widetilde{O}(n^x)$ and there are $\widetilde{O}(n)$ elements in total, we expect to find an element in one of the sets of S_v every $\widetilde{O}(n^{1+x}/\bar{d})$ queries. Thus, to identify a random neighbor of a vertex in H, we need to spend $\widetilde{O}(n^{1+x}/\bar{d})$ time. The RGMM algorithm queries for a random neighbor of a vertex $\widetilde{O}(\bar{d})$ times, since the exploration size is $\widetilde{O}(\bar{d})$; therefore, the total cost is $\widetilde{O}(n^{1+x})$. Combining this with the cost of sparsification, the total cost of the algorithm is $\widetilde{O}(\max(n^{2-x}, n^{1+2x}))$, which is $\widetilde{O}(n^{5/3})$ if we set $x = 1/3$.

The second challenge arises because the RGMM algorithm may predominantly visit vertices with a very low degree in H. For such vertices, finding a random neighbor can be much more time-consuming. Generally, if a vertex v in H has degree $\deg_H(v)$, then each time the algorithm finds a random neighbor of v, it needs to spend $\widetilde{O}(n^{1+x}/\deg_H(v))$ time. Therefore, if the algorithm frequently encounters vertices with constant degree, each query to find a random neighbor can cost $\widetilde{O}(n^{1+x})$. With the exploration size being $\widetilde{O}(\bar{d})$, this can significantly increase the query complexity of the algorithm. As a property of the local RGMM algorithm, we demonstrate that each vertex is visited by the algorithm in proportion to its degree in H. More formally, we prove that RGMM requires $\widetilde{O}(\deg_H(v)/n)$ neighbors of v on average, for a uniformly random permutation of edges. Thus, the degree-dependent factors cancel each other out, and the average cost of this part can be upper-bounded by $\widetilde{O}(n^{1+x})$, which is enough for us to get the $\widetilde{O}(n^{5/3})$ running time for the Threshold Set Cover.

3 Preliminaries

As is common in the literature, we use the term "with high probability" to refer to a probability of at least $1 - n^{-\alpha}$, for a sufficiently large constant $\alpha \geq 2$. Moreover, we use $\widetilde{O}(\cdot)$, $\widetilde{\Theta}(\cdot)$, and $\widetilde{\Omega}(\cdot)$ to hide the dependency on $\text{poly}(\log n)$. For a maximization problem of estimating some value χ, and for $\gamma_1 \in (0, 1]$ and $\gamma_2 > 0$, we say that $\widetilde{\chi}$ is a multiplicative-additive (γ_1, γ_2)-approximation of the value χ if $\gamma_1\chi - \gamma_2 \leq \widetilde{\chi} \leq \chi$.

3.1 Probabilistic Tools

We use the following standard concentration inequalities in our proof.

▶ **Proposition 5** (Chernoff Bound). *Let $X_1, X_2, \ldots, X_n$ be n independent Bernoulli random variables. Let $X = \sum_{i=1}^{n} X_i$. For any $k > 0$, it holds that*

$$\Pr[|X - \mathbf{E}[X]| \geq k] \leq 2\exp\left(-\frac{k^2}{3\,\mathbf{E}[X]}\right).$$

▶ **Definition 6** (Negative Association [32, 35, 42]). *Let $X_1, X_2, \ldots, X_n$ be a set of random variables. We say this set is negatively associated if for any two disjoint index sets $I, J \subseteq [n]$, and two functions f and g, both either monotonically increasing or monotonically decreasing, the following condition is satisfied:*

$$\mathbf{E}[f(X_i : i \in I) \cdot g(X_j : j \in J)] \leq \mathbf{E}[f(X_i : i \in I)] \cdot \mathbf{E}[g(X_j : j \in J)].$$

▶ **Proposition 7** (Chernoff Bound for Negatively Associated Variables). *Let $X_1, X_2, \ldots, X_n$ be a set of negatively associated Bernoulli random variables. Let $X = \sum_{i=1}^{n} X_i$. Then*

$$\Pr\left[X \geq (1+\alpha)\,\mathbf{E}[X]\right] \leq \left(\frac{e^{\alpha}}{(1+\alpha)^{1+\alpha}}\right)^{\mathbf{E}[X]}$$

and

$$\Pr\left[X \leq (1-\alpha)\,\mathbf{E}[X]\right] \leq \left(\frac{e^{-\alpha}}{(1-\alpha)^{1-\alpha}}\right)^{\mathbf{E}[X]}$$

▶ **Proposition 8** (Markov Inequality). *Let X be a non-negative random variable. For any $\alpha > 0$, it holds that*

$$\Pr[X \geq \alpha] \leq \frac{\mathbf{E}[X]}{\alpha}.$$

3.2 Graph Theory

A *multigraph* is a type of graph in which multiple edges, also known as parallel edges, are allowed between any pair of vertices. A *line graph* of a graph G is a graph that represents the adjacencies between the edges of G. More formally, given a graph G, the line graph $L(G)$ is constructed as follows:

- **Vertices**: Each vertex in $L(G)$ corresponds to an edge in G.
- **Edges**: Two vertices in $L(G)$ are connected by an edge if and only if their corresponding edges in G share a common endpoint (i.e., they are incident to the same vertex in G).

For a graph G, we let $\deg_G(v)$ be the degree of vertex v. In the case of multigraphs, $\deg_G(v)$ counts parallel edges multiple times.

Random Greedy Maximal Matching (RGMM). Given a graph $G = (V, E)$, a random greedy maximal matching is constructed by first selecting a random permutation π of the edges E. The algorithm then iterates over the edges in the order specified by π, adding each edge to the matching if neither of its endpoints is already matched (i.e., if none of its adjacent edges have been included in the matching so far). This process continues until all edges have been considered. The term "random" comes from the fact that the permutation π is chosen uniformly at random among all possible permutations of the edges.

Random Greedy Maximal Independent Set (MIS). Given a graph $G = (V, E)$, a random greedy maximal independent set is constructed by first selecting a random permutation π of the vertices V. The algorithm then iterates over the vertices in the order specified by π, adding each vertex to the independent set if none of its neighbors are already in the set (i.e., it does not share an edge with any vertex already included in the independent set). This process continues until all vertices have been considered. The term "random" comes from the fact that the permutation π is chosen uniformly at random among all possible permutations of the vertices.

Parallel Randomized Greedy Maximal Independent Set. Let G be a graph, and let π be a permutation of its vertices. In each iteration, we select all vertices whose rank is lower than that of all their neighbors and then remove these vertices along with their neighbors from the graph. The number of iterations required for G to become empty is referred to as the round complexity, denoted by $\rho(G, \pi)$. It is not hard to see that the MIS produced by the parallel randomized greedy maximal independent set is the same as the output of the random greedy maximal independent set for a fixed permutation π.

4 Sublinear Algorithm for Set Cover

In this section, we formalize and analyze our algorithm for the set cover variant described above. Throughout this section, we assume that $\mathcal{U}$ denotes the universe and $\mathcal{F}$ denotes the collection of sets. We will slightly abuse notation by letting $k = |\mathcal{U}|$ and $n = |\mathcal{F}|$. Without loss of generality, we can assume that $n \geq k$.[1] We use $\mathsf{SC}(\mathcal{U}, \mathcal{F})$ for the size of the minimum set cover of the input instance. Our goal is to design an algorithm that estimates the value of $\chi = k - \mathsf{SC}(\mathcal{U}, \mathcal{F})$ with at most εk additive error and a constant multiplicative factor. For $\gamma_1 \in (0, 1]$ and $\gamma_2 > 0$, we say that $\widetilde{\chi}$ is a multiplicative-additive (γ_1, γ_2)-approximation of the value χ if $\gamma_1 \chi - \gamma_2 \leq \widetilde{\chi} \leq \chi$. Similar to the reduction from the Steiner tree problem to set cover in [16], we need to estimate $k - \mathsf{SC}(\mathcal{U}, \mathcal{F}_{\neq 2})$ where $\mathcal{F}_{\neq 2}$ denotes the collection of all sets in $\mathcal{F}$ except those of size exactly 2. For simplicity, we focus on estimating χ, and in the final step of this section, we will explain how to handle sets of size 2 in our algorithm. For our application, we need γ_1 to be constant and $\gamma_2 = \varepsilon k$ where ε is a small fixed constant.

A High-Level Description of the Algorithm. Our algorithm for estimating the value of χ is formalized in Algorithm 1. Apart from the collection of sets $\mathcal{F}$ and the universe of elements $\mathcal{U}$, the algorithm runs with two parameters, α and β, both of which can be determined based on the values of x and y, which we optimize in the final step of the analysis. The algorithm has three phases: 1) sparsification of the sets, 2) sparsification of the elements, and 3) estimating the size of a maximum matching of the auxiliary graph H (Definition 9). In the following, we first describe each component in words before moving on to the formal proofs.

(Step 1) Set Sparsification. This component of the algorithm is formalized in Algorithm 2. The algorithm maintains a collection of sets $\hat{\mathcal{F}}$ and a universe of elements $\hat{\mathcal{U}}$ that are initially equal to $\mathcal{F}$ and $\mathcal{U}$, respectively. We iterate over all sets in $\mathcal{F}$ one by one and for each set S, we sample $r_1 = |\hat{\mathcal{U}}|/\alpha$ random elements from $\hat{\mathcal{U}}$. Intuitively, if $|S \cap \hat{\mathcal{U}}| \geq \widetilde{\Omega}(\alpha)$, we expect to have an element of S in the r_1 random sampled elements. Having this in mind, if there is a large enough intersection ($\Omega(\log n)$) between the sampled elements and S, we remove set S and all its elements from $\hat{\mathcal{F}}$ and $\hat{\mathcal{U}}$, respectively (we add this set to our solution). Therefore, after the execution of the algorithm, each remaining set in $\hat{\mathcal{F}}$ has at most $\widetilde{O}(\alpha)$ elements. On the other hand, if $|S \cap \hat{\mathcal{U}}|$ is smaller than α, we expect to see a small intersection with the r_1 sampled elements. Consequently, the number of times the algorithm removes a set and its elements from $\hat{\mathcal{F}}$ and $\hat{\mathcal{U}}$ is at most $k/\alpha = o(k)$, which can be accounted for by the additive error in the estimation. Also, if at any point the size of the maintained universe becomes smaller than some threshold (in the algorithm the value of the threshold is $\widetilde{\Theta}(\alpha)$, the algorithm stops processing the rest of the sets (Algorithm 2) since $\hat{\mathcal{U}} = \widetilde{O}(\alpha)$. This step differs from the algorithm in [16] because we sequentially sparsify the sets, whereas their approach is non-adaptive.

(Step 2) Sparsification of Elements. This part of the algorithm is formalized in Algorithm 3. Similar to the previous step, we want to sparsify our instance such that each element in the remaining instance appears in at most $\widetilde{O}(\beta)$ sets. Let $\hat{\mathcal{U}}$ and $\hat{\mathcal{F}}$ be the output of Algorithm 2. We sample $r_2 = |\hat{\mathcal{U}}|/\beta$ random sets from the collection $\hat{\mathcal{F}}$. With the same intuition as the

[1] For the sake of this problem, for each element in the universe we can add a set that only contains the element. The same assumption is also made in [16].

previous step, if some element is in at least $\widetilde{\Omega}(\beta)$ sets of $\hat{\mathcal{F}}$, we expect to see it in many sampled sets. We partition the elements of $\hat{\mathcal{U}}$ into $\mathcal{U}_{low}$ and $\mathcal{U}_{high}$, depending on whether their intersection with the randomly sampled elements is smaller than a given threshold or not. With high probability, each element in $\mathcal{U}_{low}$ appears in at most $\widetilde{O}(\beta)$ sets of $\hat{\mathcal{F}}$, and each element in $\mathcal{U}_{high}$ appears in at least $\widetilde{\Omega}(\beta)$ sets of $\hat{\mathcal{F}}$. This suffices to show that any random subset of $\hat{\mathcal{F}}$ of size $\varepsilon k/2$ can cover all elements of $\mathcal{U}_{high}$, which can be included in the additive error of the estimation. This step is similar to the approach used in [16].

After steps 1 and 2 of the algorithm, we have the property that each set in the remaining instance has at most $\widetilde{O}(\alpha)$ elements, and each element in the remaining instance is in at most $\widetilde{O}(\beta)$ sets.

(Step 3) Estimating the Maximum Matching of Auxiliary Graph H. We construct an auxiliary multigraph H with vertex set $\mathcal{U}_{low}$. For each set $S \in \hat{\mathcal{F}}$, we add an edge in H between every two elements of S. Note that we do not explicitly construct the multigraph H because doing so would require $\Omega(nk)$ time, which is not feasible. We now estimate the size of the maximum matching in H to produce our final estimate. Intuitively, if χ is very small (close to zero), meaning that nearly k sets are required to cover the universe, then the maximum matching in H will also be small. To see this, note that a matching edge implies that its two endpoints can be covered by the same set. Conversely, if χ is large (almost equal to k), then H will have a large matching because each set in the set cover solution covers many new elements, which can be almost paired up in H. Thus, obtaining a constant approximation for the size of the maximum matching in H is sufficient to achieve our goal, and we do this by estimating the size of a random greedy maximal matching in H. To that end, we modify and analyze the algorithm from [6] for multigraphs and adapt it to our access model for the multigraph H, as discussed in detail in the full version of the paper. The algorithm in [16] constructs a similar graph, but it is not a multigraph in the sense that, for any two elements that appear together in multiple sets, only a single edge is added between them in H.

▶ **Definition 9** (Auxiliary Multigraph H). *Let $\hat{\mathcal{F}}$ and $\mathcal{U}_{low}$ be as defined in Algorithm 1, respectively, of Algorithm 1. We construct an auxiliary graph H with vertex set $\mathcal{U}_{low}$ such that for each set S in $\hat{\mathcal{F}}$ and each two different elements $e, e' \in \mathcal{U}_{low}$, we add an edge (e, e') to H. Note that H is a multigraph: multiple sets may contain both elements e and e', and we add an edge for each of these sets.*

Algorithm 1 Sublinear Time Algorithm for Set Cover.

Input: Collection of sets $\mathcal{F}$ and universe of elements $\mathcal{U}$.
Parameter: $\alpha \leftarrow n^x, \quad \beta \leftarrow 10 \max(k/n^{1-y}, 1) \cdot n \log(n)/k$.
Let $\hat{\mathcal{F}}$ and $\hat{\mathcal{U}}$ be the output of Algorithm 2 with input $\mathcal{F}$, $\mathcal{U}$ and α.
Let $\mathcal{U}_{low}$ and $\mathcal{U}_{high}$ be the output of Algorithm 3 with input $\hat{\mathcal{F}}$, $\hat{\mathcal{U}}$, and β.
Let H be the auxiliary multigraph defined in Definition 9. ▷ We do not build H explicitly.
Let $\widetilde{\mu}$ be the estimate of $\mathbf{E}_\pi |\mathrm{RGMM}(H, \pi)|$.
Let $\widetilde{\chi} \leftarrow \widetilde{\mu} + |\mathcal{U} \setminus \mathcal{U}_{low}| - \varepsilon k/2$.
return $\widetilde{\chi}$.

Algorithm 2 Sparsification of Sets.

Input: Collection of sets $\mathcal{F}$, universe of elements $\mathcal{U}$, and parameter α that controls the sparsification ratio.
$\hat{\mathcal{F}} \leftarrow \mathcal{F}, \quad \hat{\mathcal{U}} \leftarrow \mathcal{U}, \quad c \leftarrow 0.$ ▷ We use c only for the analysis.
for $S \in \mathcal{F}$ **do**
  if $|\hat{\mathcal{U}}| < 10\alpha \log n$ **then**
    break
  $r_1 \leftarrow |\hat{\mathcal{U}}|/\alpha.$
  Let $e_1, e_2, \ldots, e_{r_1}$ be r_1 random elements of $\hat{\mathcal{U}}$.
  Make queries between S and elements $e_1, \ldots, e_{r_1}$.
  if $|\{e_1, \ldots, e_{r_1}\} \cap S| \geq 10 \log n$ **then**
    $\hat{\mathcal{F}} \leftarrow \hat{\mathcal{F}} \setminus \{S\}.$
    $c \leftarrow c + 1.$
    for $e \in \hat{\mathcal{U}}$ **do**
      if $e \in S$ **then**
        $\hat{\mathcal{U}} \leftarrow \hat{\mathcal{U}} \setminus \{e\}.$
return $\hat{\mathcal{F}}, \hat{\mathcal{U}}$

Algorithm 3 Sparsification of Elements.

Input: Collection of sets $\hat{\mathcal{F}}$, universe of elements $\hat{\mathcal{U}}$, and parameter β that controls the sparsification ratio.
$r_2 \leftarrow |\hat{\mathcal{U}}|/\beta.$
if $r_2 < 20 \log n/\varepsilon$ **then**
  $\mathcal{U}_{low} \leftarrow \hat{\mathcal{U}}, \quad \mathcal{U}_{high} \leftarrow \emptyset.$
  return $\mathcal{U}_{low}, \mathcal{U}_{high}.$
Let $\{S_1, S_2, \ldots, S_{r_2}\}$ be r_2 random sets from $\hat{\mathcal{F}}$.
Make queries between all elements in $\hat{\mathcal{U}}$ and sets in $\{S_1, S_2, \ldots, S_{r_2}\}$.
Let $\mathcal{U}_{low}$ be the elements that appeared in at most $20 \log n/\varepsilon$ many sets.
$\mathcal{U}_{high} \leftarrow \hat{\mathcal{U}} \setminus \mathcal{U}_{low}.$
return $\mathcal{U}_{low}, \mathcal{U}_{high}.$

4.1 Proof of Correctness

In this section, we prove the correctness of Algorithm 1.

▷ Claim 10. For any set S such that the condition of Algorithm 2 holds during the execution of Algorithm 2, with high probability it holds that $|S \cap \hat{\mathcal{U}}| \geq \alpha$, where $\hat{\mathcal{U}}$ denotes the universe that Algorithm 2 maintains at the time it processes S.

Proof. Suppose that S is a set such that $|S \cap \hat{\mathcal{U}}| < \alpha$ at the time that Algorithm 2 processes this set. Let X_i be the random variable that indicates $e_i \in S$. Thus, we have $\Pr[X_i] \leq |S \cap \hat{\mathcal{U}}|/|\hat{\mathcal{U}}|$. Let $X = \sum_{i=1}^{r_1} X_i$. By linearity of expectation, we have $\mathbf{E}[X] < r_1\alpha/|\hat{\mathcal{U}}| = 1$. Also, note that X_i's are negatively associated random variables. Let $\lambda = (9 \log n)/\mathbf{E}[X]$. Therefore, using the Chernoff bound for negatively associated random variables (Proposition 7) we have

$$\begin{aligned}\Pr[X \geq (1+\lambda)\,\mathbf{E}[X]] &\leq \left(\frac{e^{\lambda}}{(1+\lambda)^{1+\lambda}}\right)^{\mathbf{E}[X]} \\ &\leq \left(\frac{e^{\lambda}}{\lambda^{\lambda}}\right)^{\mathbf{E}[X]} && \text{(Since } \lambda > 1\text{)} \\ &= \left(\frac{e}{\lambda}\right)^{9\log n} && \text{(Since } \lambda = (9\log n)/\,\mathbf{E}[X]\text{)} \\ &\leq \frac{1}{n^9} && \text{(Since } \lambda > e^2\text{)}\end{aligned}$$

which implies that with probability of at least $1 - n^{-9}$,

$$X < (1+\lambda)\,\mathbf{E}[X] = \mathbf{E}[X] + 9\log n < 10\log n.$$

Since we have n sets, using a union bound, with a probability at least $1 - n^{-8}$, for any set such that the condition of Algorithm 2 holds, we have $|S \cap \hat{\mathcal{U}}| \geq \alpha$. ◁

▷ Claim 11. Let c be the variable used in Algorithm 2. With high probability, we have $c \leq k/\alpha$.

Proof. By Claim 10, for every set S such that the condition on Algorithm 2 of Algorithm 2 holds, with high probability we have that $|S \cap \hat{\mathcal{U}}| \geq \alpha$. Hence, each time the algorithm increases c, the size of $\hat{\mathcal{U}}$ decreases by α. Therefore, the total number of times the algorithm increases c is upper-bounded by k/α. ◁

▷ Claim 12. Let $\hat{\mathcal{F}}$ and $\hat{\mathcal{U}}$ be the output of Algorithm 2. Then, each set $S \in \hat{\mathcal{F}}$ has at most $20\alpha\log n$ elements in $\hat{\mathcal{U}}$ with high probability.

Proof. First, note that if the algorithm stops because of the condition of Algorithm 2 and does not process S, it holds that $|\hat{\mathcal{U}}| < 10\alpha\log n$ and the claim trivially holds.

Let S be a set such that at the time that Algorithm 2 processes S, we have $|S \cap \hat{\mathcal{U}}| \geq 20\alpha\log n$. Similar to the proof of Claim 10, let X_i be the random variable that indicates $e_i \in S$ and $X = \sum_{i=1}^{r_1} X_i$. Hence, $\mathbf{E}[X] \geq 20 r_1 \alpha \log n/|\hat{\mathcal{U}}| = 20\log n$. Since X_i's are negatively associated random variables, using Chernoff bound (Proposition 7) for $\lambda = (9\log n)/\,\mathbf{E}[X]$, we have

$$\begin{aligned}\Pr[X \leq (1-\lambda)\,\mathbf{E}[X]] &\leq \left(\frac{e^{\lambda}}{(1+\lambda)^{1+\lambda}}\right)^{\mathbf{E}[X]} \\ &\leq \left(\frac{e^{\lambda}}{\lambda^{\lambda}}\right)^{\mathbf{E}[X]} && \text{(Since } \lambda > 1\text{)} \\ &= \left(\frac{e}{\lambda}\right)^{9\log n} && \text{(Since } \lambda = (9\log n)/\,\mathbf{E}[X]\text{)} \\ &\leq \frac{1}{n^9} && \text{(Since } \lambda > e^2\text{)}\end{aligned}$$

which implies that with a probability of at least $1 - n^{-9}$,

$$X > (1-\lambda)\,\mathbf{E}[X] = \mathbf{E}[X] - 9\log n \geq 20\log n - 9\log n > 10\log n.$$

Therefore, the condition on Algorithm 2 of Algorithm 2 must hold for all such sets with a probability of at least $1 - n^{-8}$ using union bound. ◁

▶ **Lemma 13** (Sets Sparsification Guarantee). *Let $\hat{\mathcal{F}}$ and $\mathcal{U}_{low}$ be outputs of Algorithm 2 and Algorithm 3. Also, let $S \in \hat{\mathcal{F}}$. Then, with high probability, S contains at most $\widetilde{O}(\alpha)$ elements of $\mathcal{U}_{low}$.*

Proof. Note that $\mathcal{U}_{low} \subseteq \hat{\mathcal{U}}$ where $\hat{\mathcal{U}}$ is the output of Algorithm 2. Also, by Claim 12, we have $|S \cap \hat{\mathcal{U}}| \leq 20\alpha \log n$. Thus, with high probability we have $|S \cap \mathcal{U}_{low}| \leq 20\alpha \log n = \widetilde{O}(\alpha)$. ◀

▶ **Lemma 14** (Elements Sparsification Guarantee). *Let $\mathcal{U}_{low}$ be as output by Algorithm 3 and let $e \in \mathcal{U}_{low}$. Then, with high probability, there are at most $\widetilde{O}(\beta)$ sets of $\hat{\mathcal{F}}$ that contain e.*

Proof. Let e be an element of $\hat{\mathcal{U}}$, where $\hat{\mathcal{U}}$ is the output of Algorithm 2. We show that if at least $40\beta \log n/\varepsilon$ sets in $\hat{\mathcal{F}}$ contain e, then $e \in \mathcal{U}_{high}$ in the output of Algorithm 3 with high probability.

Let X_i be an indicator variable for S_i containing e. Thus, we have $\mathbf{E}[X_i] \geq 40\beta \log n/(\varepsilon n)$. Define $X = \sum_{i=1}^{r_2} X_i$. Hence, $\mathbf{E}[X] \geq r_2 \cdot 40\beta \log n/(\varepsilon n) = 40 \log n/\varepsilon$. Since X_i's are negatively associated random variables, using Chernoff bound (Proposition 7) for $\lambda = (9 \log n)/\mathbf{E}[X]$, we have

$$\Pr[X \leq (1-\lambda)\mathbf{E}[X]] \leq \frac{1}{n^9} \qquad \text{(Similar to the proof of Claim 12)}$$

which implies that $X > 20 \log n$ with a probability of at most $1 - n^{-9}$, since

$$X > (1-\lambda)\mathbf{E}[X] = \mathbf{E}[X] - 9\log n \geq 40 \log n/\varepsilon - 9 \log n > 20 \log n/\varepsilon.$$

Using a union bound for all elements in $\hat{\mathcal{U}}$ that are in at least $40\beta \log n/\varepsilon$ sets of $\hat{\mathcal{F}}$, with high probability all of them are going to be included in $\mathcal{U}_{high}$. As a result, for each $e \in \mathcal{U}_{low}$, e is in at most $\widetilde{O}(\beta)$ sets of $\hat{\mathcal{F}}$. ◀

▷ Claim 15. Let $\mathcal{F}'$ be a random collection of $\varepsilon k/5$ sets of $\hat{\mathcal{F}}$. Then, with high probability, every element in $\mathcal{U}_{high}$ is in one of the sets of $\mathcal{F}'$.

Proof. Let $e \in \hat{\mathcal{U}}$ be such that at most $15\beta \log n/\varepsilon$ sets of $\hat{\mathcal{F}}$ contain e. Similar to the proof of Lemma 14, let X be a random variable that denotes the number of sets S_i (for $i = 1, 2, ..., r_2$) that contain e. Using a Chernoff bound, we can prove that with a probability of at least $1 - n^{-2}$, for all such elements e, we have $X < 20 \log n/\varepsilon$. Therefore, all elements in $\mathcal{U}_{high}$ are in at least $15\beta \log n/\varepsilon$ sets of $\hat{\mathcal{F}}$.

Consequently, when $|\mathcal{F}'| \geq \varepsilon k/5$, the expected number of sets in $\mathcal{F}'$ that cover element $e \in \mathcal{U}_{high}$ is at least

$$\frac{15\beta \log n}{\varepsilon} \cdot \frac{1}{n} \cdot \frac{\varepsilon k}{5} \geq \frac{15n \log n}{\varepsilon k} \cdot \frac{1}{n} \cdot \frac{\varepsilon k}{5} = 3 \log n$$

where we used that $\beta \geq n/k$ (see Algorithm 1 in Algorithm 1). Using Chernoff bounds again, we expect all these elements to be covered by at least one set with high probability, which concludes the proof. ◁

▷ Claim 16. Let $\mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}})$ be the optimal set cover size for the universe of elements $\mathcal{U}_{low}$ and the collection of sets $\hat{\mathcal{F}}$ which are the outputs of Algorithm 3 and Algorithm 2, respectively. Let $\widetilde{\mu}$ be the output of Lemma 21 (i.e., a size estimate of a random greedy maximal matching of H) with $\varepsilon k/2$ additive error. Then, it holds that

$$\frac{1}{2}\left(|\mathcal{U}_{low}| - \mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}})\right) - \frac{\varepsilon k}{2} \leq \widetilde{\mu} \leq |\mathcal{U}_{low}| - \mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}}).$$

Proof. Let M be any maximal matching of H. First, we show that

$$\frac{1}{2}\left(|\mathcal{U}_{low}| - \mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}})\right) \leq |M| \leq |\mathcal{U}_{low}| - \mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}}).$$

If for each edge in M, we take the corresponding set, and for the rest of the elements we take a set that only covers that element, we will have covered all elements of $\mathcal{U}_{low}$ with at most $|\mathcal{U}_{low}| - |M|$ sets, which implies that $|M| \leq |\mathcal{U}_{low}| - \mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}})$. On the other hand, no set can cover two elements that are unmatched by M at the same time. Thus, we have $|\mathcal{U}_{low}| - 2|M| \leq \mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}})$.

Since the output of *Lemma* 21 can be smaller than the minimum-sized maximal matching by an additive factor of at most $\varepsilon k/2$, we get the claim. ◁

▷ Claim 17. Conditioning on the high-probability events of Claim 11 and Claim 15, it holds that $\mathsf{SC}(\mathcal{U}, \mathcal{F}) \leq \mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}}) + \varepsilon k/2$.

Proof. All elements of $\mathcal{U} \setminus (\mathcal{U}_{low} \cup \mathcal{U}_{high})$ can be covered using the c sets that are deleted in Algorithm 2. Moreover, by Claim 11, we have $c < o(k)$ since $\alpha > \omega(1)$. On the other hand, the elements of $\mathcal{U}_{high}$ can be covered using $\varepsilon k/5$ sets, as shown in Claim 15. Finally, the elements of $\mathcal{U}_{low}$ can be covered using $\mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}})$ sets. Therefore, the union of these three collections covers all elements, and the size of this solution is at most $\mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}}) + \varepsilon k/2$, which completes the proof. ◁

▶ **Lemma 18.** *Conditioning on the high-probability events of Claim 11 and Claim 15, it holds that* $\chi/2 - \varepsilon k \leq \widetilde{\chi} \leq \chi$. *(Recall that* $\chi = k - \mathsf{SC}(\mathcal{U}, \mathcal{F})$ *and* $\widetilde{\chi}$ *is the output of Algorithm 1, defined on its Algorithm 1.)*

Proof. We have that

$$\begin{aligned}
\widetilde{\chi} &= \widetilde{\mu} + |\mathcal{U} \setminus \mathcal{U}_{low}| - \frac{\varepsilon k}{2} && \text{(Output of Algorithm 1)}\\
&\leq |\mathcal{U}_{low}| - \mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}}) + |\mathcal{U} \setminus \mathcal{U}_{low}| - \frac{\varepsilon k}{2} && \text{(By Claim 16)}\\
&= |\mathcal{U}| - \mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}}) - \frac{\varepsilon k}{2}\\
&\leq |\mathcal{U}| - \mathsf{SC}(\mathcal{U}, \mathcal{F}) && \text{(By Claim 17)}\\
&= \chi.
\end{aligned}$$

On the other hand,

$$\begin{aligned}
2(\widetilde{\chi} + \varepsilon k) &= 2\left(\widetilde{\mu} + |\mathcal{U} \setminus \mathcal{U}_{low}| + \frac{\varepsilon k}{2}\right)\\
&\geq 2\left(\frac{|\mathcal{U}_{low}|}{2} - \frac{\mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}})}{2} + |\mathcal{U} \setminus \mathcal{U}_{low}|\right) && \text{(By Claim 16)}\\
&\geq |\mathcal{U}| - \mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}})\\
&= |\mathcal{U}| - \mathsf{SC}(\mathcal{U}_{low}, \mathcal{F})\\
&\geq |\mathcal{U}| - \mathsf{SC}(\mathcal{U}, \mathcal{F})\\
&= \chi,
\end{aligned}$$

which concludes the proof. ◀

4.2 Time Complexity

▷ Claim 19. Algorithm 2 runs in $O(nk/\alpha)$ time with high probability.

Proof. First note that Algorithm 2 samples at most k/α elements for each set in $\mathcal{F}$ (Algorithm 2). So if we ignore the block of if-condition in Algorithm 2, the runtime is at most $O(nk/\alpha)$. Further, by Claim 11, the condition on Algorithm 2 holds at most k/α times with high probability. Since each time the algorithm enters the if-condition it spends $O(k)$ time to iterate over all elements, the total running time is at most $O(k^2/\alpha + nk/\alpha) = O(nk/\alpha)$. ◁

▷ Claim 20. Algorithm 3 runs in $O(k^2/\beta)$ time.

Proof. The algorithm samples $r_2 = O(k/\beta)$ sets and for each of them makes a membership query between the set and all elements. Hence, the total running time is upper-bounded by $O(k^2/\beta)$. ◁

We defer the proof of the next lemma to the full version of the paper.

▶ **Lemma 21.** *Let H be the multigraph defined in Definition 9. There exists an algorithm with an expected running time of $\widetilde{O}_\varepsilon(k\beta + \alpha\beta n)$ that estimates the value of $E_\pi|\mathrm{RGMM}(H,\pi)|$ with εk additive error with high probability.*

▶ **Lemma 22.** *The total running time of the algorithm is $\widetilde{O}(nk/\alpha + k^2/\beta + n\alpha\beta)$ with high probability.*

Proof. The algorithm runs in expected time $\widetilde{O}(nk/\alpha + n\alpha\beta)$ as shown by Claim 19, Claim 20, Lemma 21, and the fact that $k \le n$. To ensure a high-probability bound on the running time, we execute $\Theta(\log n)$ instances of the algorithm in parallel and use the estimate from the first instance that finishes. Since the expected running time is $\widetilde{O}(kn/\alpha + k^2/\beta + n\alpha\beta)$, the first instance is likely to terminate within $\widetilde{O}(nk/\alpha + k^2/\beta + n\alpha\beta)$ time with probability $1 - 1/\operatorname{poly}(n)$. (We remark that since our approximation guarantees hold with high probability, they also hold with high probability for each of the instances.) ◀

4.3 Putting Everything Together

We combine our ideas to obtain our final theorem for estimating $\chi = k - \mathsf{SC}(\mathcal{U},\mathcal{F})$. Then, we extend our analysis and make slight modifications to the algorithm to estimate $k - \mathsf{SC}(\mathcal{U},\mathcal{F}_{\neq 2})$ which is crucial for designing a sublinear algorithm for the Steiner tree problem.

▶ **Theorem 3** (Our Algorithm for Threshold Set Cover)**.** *There exists an algorithm that, given a set system $(\mathcal{U},\mathcal{F})$ with oracle access to its adjacency matrix (also known as membership queries), outputs a multiplicative-additive $(1/2, \varepsilon\cdot|\mathcal{U}|)$-approximation to Threshold Set Cover, in $\widetilde{O}(|\mathcal{F}|^{5/3})$ time, with high probability.*

Proof. First, if $k \le O(n^{2/3})$, then we can easily query between all elements and sets and compute the estimate in $\widetilde{O}(n^{5/3})$ time since all the steps of the algorithm such as finding the maximal matching can be run in linear time with respect to the size of the input. Now suppose that $k > n^{2/3}$. We use Algorithm 1 to produce the estimate $\widetilde{\chi}$, setting $x = 1/3$ and $y = 1/3$. By Lemma 18, we have that $\widetilde{\chi}$ is a multiplicative-additive $(1/2, \varepsilon k)$-approximation to the value of χ.

Moreover, we have $\alpha = n^{1/3}$ and $\beta = 10\max(k/n^{1-y}, 1)\cdot n\log(n)/k = \widetilde{O}(n^{1/3})$. By Lemma 22, the total runtime of the algorithm is upper-bounded by $\widetilde{O}(kn/\alpha + k^2/\beta + n\alpha\beta) = \widetilde{O}(n^{5/3})$ because of the assumption that $k \le n$. ◀

Now we show how we can estimate $k - \mathsf{SC}(\mathcal{U}, \mathcal{F}_{\neq 2})$ with a slight modification. The only change that is needed in our algorithm is to remove edges of H that are produced by sets of size 2 in $\hat{\mathcal{F}}$. Hence, when the algorithm estimates $\mathbf{E}_\pi[\mathrm{RGMM}(H, \pi)]$ using the vertex oracle, the oracles must not use those edges in the exploration as they do not exist in graph H. Thus, in the implementation of Lemma 21, when the algorithm queries pairs (u, S) where $u \in \mathcal{U}_{low}$ and $S \in \mathcal{F}_v$ to find neighbors of v in H, if $u \in S$, the algorithm starts querying between all elements of $\mathcal{U} \setminus v$ and S until it finds another element in S. If the algorithm finds another element of S, then it accepts the edge (v, u) as a valid edge, and otherwise it continues the random search. So each time the algorithm finds such pair (u, S), it invokes the above procedure to validate the edge. In the next theorem, we demonstrate how to bound the running time of the modified algorithm and discuss its approximation ratio.

▶ **Theorem 23.** *There exists an algorithm that outputs a multiplicative-additive $(1/2, \varepsilon k)$-approximation of value of $\chi = k - \mathsf{SC}(\mathcal{U}, \mathcal{F}_{\neq 2})$ in $\widetilde{O}(|\mathcal{F}|^{5/3})$ time with high probability.*

Proof. We need to show that the new estimation is a multiplicative-additive $(1/2, \varepsilon k)$-approximation of $k - \mathsf{SC}(\mathcal{U}, \mathcal{F}_{\neq 2})$. We follow the same approach as proof of Lemma 18. To follow the same steps, we need analogous claims similar to Claim 16 and Claim 17. The exact same proof of Claim 16 also works to provide a bound on $\mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}}_{\neq 2})$. More specifically, we have

$$\frac{1}{2}\left(|\mathcal{U}_{low}| - \mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}}_{\neq 2})\right) - \frac{\varepsilon k}{2} \leq \widetilde{\mu} \leq |\mathcal{U}_{low}| - \mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}}_{\neq 2}).$$

Further, to show that $\mathsf{SC}(\mathcal{U}, \mathcal{F}_{\neq 2}) \leq \mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}}_{\neq 2}) + \varepsilon k/2$, note that we have $c = o(k)$ (see Algorithm 2 for definition of c) which implies that $\mathcal{U} \setminus (\mathcal{U}_{low} \cup \mathcal{U}_{high})$ can be covered using $o(k)$ sets of size larger than 2 (all sets that are removed in Algorithm 2 have size larger than 2). Additionally, elements of $\mathcal{U}_{high}$ can be covered using $\varepsilon k/5$ sets of $\hat{\mathcal{F}}$ according to Claim 15. Thus, we can cover elements of $\mathcal{U}_{high}$ using $2\varepsilon k/5$ sets of $\hat{\mathcal{F}}_{\neq 2}$ since we can replace each set of size 2 with two sets of size 1 that cover a single element. Finally, elements of $\mathcal{U}_{low}$ can be covered using $\mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}}_{\neq 2})$ sets. The union of all these sets is a valid solution for $\mathsf{SC}(\mathcal{U}, \hat{\mathcal{F}}_{\neq 2})$ which has a size of at most $\mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}}_{\neq 2}) + \varepsilon k/2$. Therefore, the error of our estimation is $(1/2, \varepsilon k)$.

Now it remains to bound the running time of the modified algorithm. We prove that the same bound on the running time as in Lemma 21 holds. Now consider a vertex v that is corresponding to an element in $\mathcal{U}_{low}$. Let $\hat{\mathcal{F}}_v$ be the collection of sets that include v. Also, let $\hat{\mathcal{F}}'_v = \{S | S \in \hat{\mathcal{F}}_v \text{ and } |S| = 2\}$. Let $r = |\hat{\mathcal{F}}_v|$ and $\hat{\mathcal{F}}_v = \{S_1, \ldots, S_r\}$. Also, let $\tau_i = |S_i \cap (\mathcal{U}_{low} \setminus v)|$. Hence, in Lemma 21, when the algorithm queries a pair (u, S) and it returns $u \in S$, the probability that $S = S_i$ is $\tau_i / \sum_{i=1}^{r} \tau_i$. For this set, using a Chernoff bound, the algorithm needs to spend at most $\widetilde{O}(k/\tau_i)$ time to see another element of S_i or explore all elements of $\mathcal{U}$. Therefore, the expected additional time the algorithm needs to spend compared to Lemma 21 is

$$\sum_{i}^{r} \left(\frac{\tau_i}{\sum_{i=1}^{r} \tau_i}\right) \cdot \widetilde{O}\left(\frac{k}{\tau_i}\right) = \widetilde{O}\left(\sum_{i}^{r} \frac{k}{\sum_{i=1}^{r} \tau_i}\right) \leq \widetilde{O}\left(\frac{rk}{\deg_H(v)}\right),$$

where the last inequality follows by the fact that $\deg_H(v) \leq \sum_{i=1}^{r} \tau_i$ since $\deg_H(v) = (\sum_{i=1}^{r} \tau_i) - |\{S | S \in \hat{\mathcal{F}}_v \text{ and } |S| = 2\}|$ after the modification of the graph H. On the other hand, the expected number of times that the oracle calls an adjacent edge of vertex v, is at most $\widetilde{O}(\deg_H(v)/k)$ (for the formal statement and the proof, see the full version of the paper). Also, these two variables have a negative correlation. Therefore, the total cost for all

vertices is at most $\widetilde{O}(rk)$. Combining with the fact that $r = \widetilde{O}(\beta)$ (Lemma 14), the total additional cost is $\widetilde{O}(k\beta)$, which is dominated by other terms in Lemma 22 which implies that the algorithm has the same running time as Theorem 3 if we choose α and β similarly. ◀

5 Connection to Steiner Tree

In this section, we show how our improved algorithm for set cover (from Section 4) implies an improved sublinear algorithm for metric Steiner tree. Formally, we show the following theorem.

▶ **Theorem 4** (Sublinear Algorithm for Metric Steiner Tree). *There exists an algorithm that, given an instance of metric Steiner tree denoted by (V, T, w) with oracle access $\mathcal{O}$ to the distance matrix of (V, w), outputs a $(2-\eta)$-estimate of $\mathsf{ST}(V, T, w)$ using $\widetilde{O}(n^{5/3})$ queries to $\mathcal{O}$, where $\eta > 0$ is a universal constant, with high probability.*

The overall structure of our algorithm is similar to the algorithm of Chen, Khanna and Tan [16]. The main difference in our algorithm compared to [16] is in the set cover component. In the following, we first provide an overview of their algorithm in Section 5.1. We then provide the query-efficient implementation of their algorithm and our modification to it in Section 5.2. We will finally provide the query complexity analysis and the proof of Theorem 4 in Section 5.3. Note that the approximation analysis of our algorithm follows directly from the proof of Theorem 3 in [16].

5.1 Algorithm at a High Level

Step 1: Minimum spanning tree over terminals. The algorithm of [16] as a first step starts with an MST $\mathcal{T}^*$ over the terminals T (whose cost can be estimated in nearly linear time using the sublinear MST algorithm of Czumaj and Sohler [18]). It is known that $w(\mathcal{T}^*)/2 \leq \mathsf{ST}(V, T, w) \leq w(\mathcal{T}^*)$. To get a strictly better-than-2 approximation of $\mathsf{ST}(V, T, w)$, it suffices to detect whether $\mathsf{ST}(V, T, w)$ is closer to $w(\mathcal{T}^*)/2$ or $w(\mathcal{T}^*)$. Hence, the rest of the algorithm is either to provide "significant" local improvements over $w(\mathcal{T}^*)$ using "set cover" like structure (i.e., step 2 in Section 4 of [16]) or "local structure" (i.e., step 3 in Section 4 of [16]), and thus output $(1 - O(\eta))w(\mathcal{T}^*)$ as the estimate of $\mathsf{ST}(V, T, w)$; or conclude that $\mathsf{ST}(V, T, w)$ is closer to $w(\mathcal{T}^*)$ and output it as the estimate of $\mathsf{ST}(V, T, w)$. Then, they show how to implement these steps using sublinear queries to $\mathcal{O}$.

Step 2: Improvement using set cover. First, they partition the edges into $L = O((\log k)/\varepsilon)$ buckets such that the edges of the ith bucket have weights in $[(1+\varepsilon)^{i-1}, (1+\varepsilon)^i)$. Let H_i be the graph built on all the terminals and all the edges upto the ith bucket. They define an instance of set cover corresponding to each level i where *ideally*,

- The elements correspond to the connected components of H_{i-1}.
- The sets correspond to the Steiner vertices.
- A set W_v (corresponding to a Steiner vertex v) contains an element u_S (corresponding to a component S) if the distance between v and some terminal in S is less than a threshold τ (think of it as $\frac{3}{5} \cdot (1+\varepsilon)^i$).

How to use set cover in making a decision about the Steiner tree cost. They show that if one can solve each of these set cover instances approximately, then one can check whether the total contribution of these set cover improvements is more than $O(\eta) \cdot w(\mathcal{T}^*)$. In particular, in that case $\mathsf{ST}(V, T, w)$ is strictly less than $(1 - O(\eta)) \cdot w(\mathcal{T}^*)$. Intuitively, this

is because one can include the Steiner vertices corresponding to the set cover solution and remove a subset of the edges in $\mathcal{T}^*$, while still maintaining a feasible solution to the Steiner tree instance. Hence, this implies that the cost of the constructed solution of the Steiner tree instance is less than $(1 - O(\eta)) \cdot w(\mathcal{T}^*)$.

A challenge and the notion of representatives. The main challenge with the above algorithm is computing the set cover instance. More precisely, in the third bullet point above, in order to check whether a set (corresponding to a Steiner vertex v) contains an element (corresponding to a connected component S), they need to compute the distance of v to all terminals $t \in T$ which could be very costly. Instead, they define a *net* on S which is a maximal subset $\tilde{S} \subseteq S$ such that any pair of terminals in $\tilde{S}$ has distance at least $\varepsilon \cdot (1+\varepsilon)^i$. They call the terminals in $\tilde{S}$ the *representatives*.

Modified set cover instance and the notion of light/heavy levels. Now, to detect if a set contains an element, we only need to check the distance of v to all terminals in $\tilde{S}$. When $|\tilde{S}|$ is small, this can be done efficiently. So, in their algorithm they *only* assign an element to a connected component S if the size of its representatives is *small*. To show that this does not introduce a large error, they define a level i to be *light* if the total sum of the edges of $\mathcal{T}^*$ in bucket i is "small", and define it to be *heavy* otherwise. They show that one can ignore all the levels i that are light, and moreover if a level is heavy, then most of its components have small sets of representatives and thus the error introduced by ignoring the components S with large net size is negligible.

Finally, we note that the notion of representatives will be used in other parts of the overall algorithm such as computing $\mathcal{T}^*$ which we will go over when describing the implementation of this step.

Step 3: Improvement using "local structure". In this step, they consider the hierarchical structure of the connected components. Specifically, they focus on components S that have exactly two child components, S_1 and S_2, where each of these child components also has exactly two child components: S_{11}, S_{12} for S_1 and S_{21}, S_{22} for S_2. Then, they check whether there exists a single Steiner vertex v that can be used to connect components $S_{11}, S_{12}, S_{21}, S_{22}$ and instead remove the corresponding edges in $\mathcal{T}^*$ connecting these components. Similarly to step 2, if the overall advantage of all these 2-level local improvements is more than $O(\eta) \cdot w(\mathcal{T}^*)$, then the algorithm outputs $(1 - O(\eta)) \cdot w(\mathcal{T}^*)$ as its estimate of $\mathsf{ST}(V, T, w)$.

Given that our algorithm does not change this step at all, we refer the reader to [16] for further details. Moreover, the query complexity is exactly the same as in [16].

5.2 Implementation of the Algorithm

Here, we focus on a query-efficient implementation of the algorithm, and particularly highlight where the set cover component was used and how we modify it.

First, we note that one cannot compute H_i exactly, so [16] shows that it suffices to work with an approximate graph H'_i such that $H_i \subseteq H'_i \subseteq H_{i+1}$.

Subroutines. Next, [16] define some useful subroutines for simulating the set cover instance on H'_i:

- $\mathsf{Find}(u, i)$: This receives a terminal u and a level i, and finds some representative terminal u' in the same connected component of H'_i that contains u. Moreover, they show that this subroutine can be implemented using $\widetilde{O}(k)$ queries.

- $\mathsf{BFS}(u, i)$: This subroutine reports all representative terminals that are in the same connected component in H'_i as u, if the total number of such representatives is $\widetilde{O}(L/\varepsilon) = \widetilde{O}(1/\varepsilon)$. Otherwise, the procedure is terminated. This procedure employs Find subroutines and has total query complexity of $\widetilde{O}(k/\varepsilon)$ which is $\widetilde{O}(k)$ given that ε is a constant.

Parameters. The algorithm uses four parameters, whose values will be later set to optimize the query complexity of the algorithm.

- M is a threshold parameter used on the number of components that contain a "small" number of representatives. Note that these are the components to which we assign an element in the universe $\mathcal{U}_i$ of the corresponding set cover instance $(\mathcal{U}_i, \mathcal{F}_i)$.
- R is a threshold parameter defining "low-degree" and "high-degree" elements.
- P is a threshold parameter denoting "low-degree" or "high-degree" sets.
- κ is a threshold parameter on the value of k. At a high level, when $k < \kappa$, we can afford to query all distances between terminals and the Steiner vertices.

Simulation of Step 2 and its query complexity. We now outline the implementation of Step 2 and specify the query complexity of each step, along with potential conditions they impose on the parameters we need to set.

- **The case of small number of terminals: if** $k \leq \kappa$, then we query all distances between terminals and Steiner vertices, which takes $O(n\kappa)$ queries. Then, we estimate $|\mathcal{U}| - \mathsf{SC}(\mathcal{U}, \mathcal{F})$ using our algorithm, but without any further queries.
- **Otherwise,** for each level i, they show that one of the following cases hold:

  Case 1. The total number of representative terminals in all connected components is $\widetilde{O}(M/\varepsilon)$. To detect this case, they use greedy MIS which can be implemented by the BFS and Find subroutines and will take $\widetilde{O}(Mk/\varepsilon)$ queries (for further details, see [43]). If this is the case, then again the set cover instance can easily be computed by querying the distance of all Steiner vertices to all representative terminals which requires $\widetilde{O}(nM/\varepsilon)$ queries. Then, similarly to the case of $k \leq \kappa$, $|\mathcal{U}_i| - \mathsf{SC}(\mathcal{U}_i, \mathcal{F}_i)$ can be estimated using our algorithm, without any further queries.

  Case 2. $|\mathcal{U}_i| \leq M$. In this case they show that level i is in fact light and thus can be ignored. To detect this case, they estimate the size of $|\mathcal{U}_i|$ using calls to BFS starting from $\widetilde{O}(k/M)$ random terminals, which overall takes $\tilde{O}(k^2/M)$ queries. Note that if we are in this case, we take no further action. Also, this step requires $k > M$, which we will ensure in our parameter setup.

  Case 3. The last case is when $|\mathcal{U}_i| \geq M$.
  - **Partitioning of the terminals based on their degree.** First, they partition the terminals into T_{low} and T_{high} based on whether the number of "close-by" Steiner vertices (roughly within distance $(3/5)(1+\varepsilon)^i$) to them is smaller than or larger than R. This partitioning can be computed using $\widetilde{O}(kn/R)$ queries by randomly sampling $\widetilde{O}(n/R)$ Steiner vertices and checking their distance to all the terminals.
  - **Handling high-degree terminals.** Then, by picking $\widetilde{O}(n/R)$ sets uniformly at random, with high probability, all elements corresponding to the components containing at least one terminal in T_{high} are covered. As they can only afford an $\varepsilon|\mathcal{U}_i|$ additive error in their estimate of $|\mathcal{U}_i| - \mathsf{SC}(\mathcal{U}_i, \mathcal{F}_i)$, they require that $n/R < \widetilde{O}(\varepsilon M) = \widetilde{O}(\varepsilon|\mathcal{U}_i|)$.

- **Handling low-degree terminals.** Next, they solve the set cover instance on $\mathcal{U}_{low}$, i.e., the connected components that have no terminal in T_{high}.
  [label=•, leftmargin=*]
  - **Partitioning of the Steiner vertices based on their degree.** They partition the sets of $\mathcal{F}_i$ into $\mathcal{W}_1$ and $\mathcal{W}_2$ based on whether their degree to T_{low} is less than $\Theta(P)$ or higher. This partitioning can be computed using $\widetilde{O}(nk/P)$ queries by randomly sampling k/P terminals from T_{low}. This requires $k > \widetilde{\Omega}(P)$ which we will ensure in our parameter setup. Then, they consider the set cover instances $(\mathcal{U}_{low}, \mathcal{W}_1)$ and $(\mathcal{U}_{low}, \mathcal{W}_2)$ separately, and return the better of the two solutions.

    Our modification to this step: In our set cover algorithm, however, we define $\mathcal{W}_2$ slightly differently, as described in Set Sparsification, see Algorithm 2. More precisely, we iterate over Steiner vertices (i.e., sets in $\mathcal{F}_i$) one by one in an arbitrary order, and at every round $j \leq |\mathcal{F}_i|$, we check whether the degree of the Steiner vertex v_j to T_{low} is more than $\Theta(P)$. If so, we add its corresponding set, W_j, to $\mathcal{W}_2$. Similarly to their test, our test can also be implemented using $\widetilde{O}(nk/P)$ queries. However, each time we add a set W_j to $\mathcal{W}_2$, we find all its "nearby" terminals and remove them from T_{low}, more precisely, $T_{low} \leftarrow T_{low} \setminus W_j$. This step can be simply done by querying the distance of the Steiner vertex v_j and all terminals in T_{low}. Hence, each time a set is added to $\mathcal{W}_2$, we perform an extra $\widetilde{O}(k)$ queries compared to the algorithm of [16]. However, as we can simply bound $|\mathcal{W}_2|$ by k/P, the overall query complexity remains as $\widetilde{O}(nk/P + k^2/P) = \widetilde{O}(nk/P)$.

  - **Handling high-degree Steiner vertices.** To solve $(\mathcal{U}_{low}, \mathcal{W}_2)$, note that by a simple double-counting argument, $|\mathcal{W}_2| \leq kR/P$. So if $kR/P \leq \varepsilon M \leq \varepsilon|\mathcal{U}_i|$, which will be ensured in the parameter setting, we can afford to pick all sets in $\mathcal{W}_2$ and thus, similarly to their argument, we only need to estimate $|\bigcup_{W\in\mathcal{W}_2} W|$ in the set cover instance. This is done by randomly sampling the terminals and using BFS and will take an overall $\widetilde{O}(k^2/M)$ queries.

    Our modification to this step: With the adjusted partitioning of $\mathcal{F}_i$ into $\mathcal{W}_1$ and $\mathcal{W}_2$ in our algorithm, the size of $\mathcal{W}_2$ is at most k/P. Therefore, by setting $k/P \leq \varepsilon M \leq \varepsilon|\mathcal{U}_i|$, we can afford to select all sets in $\mathcal{W}_2$. Notably, this modification relaxes the required condition of [16] from "$kR/P \leq \varepsilon M$" to "$k/P \leq \varepsilon M$".

  - **Handling low-degree Steiner vertices.** Finally, we need to solve $(\mathcal{U}_{low}, \mathcal{W}_1)$. In their approach, this part takes $O(RP \cdot RPk)$ queries, and this is the step where our main improvement comes from.

    Our modification to this step. By our improved bound for set cover (from Section 4), the query complexity of this part reduces to $\widetilde{O}(k^2/M + RP(n+k))$. More precisely, to simulate the algorithm in Lemma 21, we do the following.
    1. To run the RGMM oracle, we need to sample $\widetilde{O}(1)$ elements from $\mathcal{U}_i$ uniformly at random. Note that each element of $\mathcal{U}_i$ corresponds to a small component (a component with a small number of representatives). To find a small component uniformly at random, we first pick a terminal uniformly at random and run a BFS to determine whether it lies in a small component, and if so, whether it is a representative terminal. If the terminal is a representative and lies in a small component, we choose the corresponding connected component with probability $1/z$ where z is the number of representative terminals in that connected component (note that BFS returns this number as well). This approach ensures that each

small connected component has an equal probability of being sampled. Moreover, due to the bound on the number of small connected components, i.e., $|\mathcal{U}_i| \geq M$, we expect to encounter a terminal in a small connected component every $\widetilde{O}(k/M)$ samples. Therefore, the total cost of running all these BFS subroutines is $\widetilde{O}(k^2/M)$, since each BFS takes $\widetilde{O}(k)$ time.

2. For a small component (a vertex in graph H of Lemma 21), we need to identify all the Steiner nodes within a distance of at most τ (which is set roughly as $(3/5)(1+\varepsilon)^i$). This step can be completed in $\widetilde{O}(n)$ time. A similar step with the same time complexity also appears in the proof of Lemma 21.
3. Note that the number of Steiner nodes within this close distance is at most R. Let $\hat{\mathcal{S}}$ be the set of these Steiner nodes. Next, the RGMM algorithm requires a random neighbor of a small component. To get such a neighbor, we keep picking pairs (v,t) in a random order, where $t \in T_{low}$ and $v \in \hat{\mathcal{S}}$, and querying their distance. The first time that we find a pair (v,t) in close distance, we run a BFS from t to check if it is a representative terminal and if it lies in a small component, which takes $\widetilde{O}(k)$ time. If it is not a representative terminal or does not lie in a small connected component, we skip this terminal. Otherwise, we return its small connected component with probability $1/z$, where z is the number of representative terminals in that connected component, ensuring that all neighbors have an equal probability of being selected. We run the above procedure until we find a random neighbor. Therefore, using the running time from Lemma 21 (substituting α and β for R and P), and considering that we run the BFS at most $\widetilde{O}(RP)$ times (which corresponds to the maximum degree of H), we obtain an $\widetilde{O}(RP(n+k))$ time algorithm to estimate the size of the matching.

Simulation of Step 3 and its query complexity. As we are following the exact implementation of [16] for this step, the additional query complexity of this step (compared to the Step 2) is equal to $\widetilde{O}(nk/M)$ for both their algorithm and our algorithm.

5.3 Query Complexity Analysis and Proof of Theorem 4

For completeness, we start with the query complexity analysis of [16].

Analysis of the query complexity of the algorithm of [16]. As computed in Section 5.2, the overall query complexity of their algorithm is bounded by

$$\begin{aligned}&\widetilde{O}(n\kappa + \frac{Mk}{\varepsilon} + \frac{nM}{\varepsilon} + \frac{k^2}{M} + \frac{nk}{R} + \frac{nk}{P} + \frac{k^2}{M} + (RP)^2 k + \frac{nk}{M})\\ &= \widetilde{O}(n\kappa + nM + \frac{nk}{R} + \frac{nk}{P} + (RP)^2 k + \frac{nk}{M}). && \triangleright\ \varepsilon = O(1), k \leq n\end{aligned}$$

Furthermore, the conditions that need to be satisfied are

- $k \leq \kappa$, or
- $k > M$ and $n/R < \widetilde{O}(\varepsilon M)$ and $k > \widetilde{\Omega}(P)$ and $kR/P \leq \varepsilon M$.

The query complexity of their algorithm under the above conditions can be optimized by setting $\kappa = M = n^{6/7}$, $R = n^{1/7}$, and $P = n^{2/7}$, which gives the total query complexity of $\widetilde{O}(n^{13/7})$.

Now, we prove the main theorem of this section.

Proof of Theorem 4. As we are implementing the same algorithm as [16], except replacing their set cover subroutine with a more efficient algorithm, the approximation analysis follows exactly from their proof. It only remains to bound the query complexity of our proposed algorithm for metric Steiner tree using the improved sublinear algorithm for set cover. In Section 5.2, we analyzed the query complexity of the component that is implemented differently in our algorithm. Now, we put the query complexity of all parts together and compute the overall complexity.

Analysis of the query complexity of our algorithm. The overall query complexity of our algorithm is bounded by

$$\begin{aligned}&\widetilde{O}(n\kappa + \frac{Mk}{\varepsilon} + \frac{nM}{\varepsilon} + \frac{k^2}{M} + \frac{nk}{R} + \frac{nk}{P} + \frac{k^2}{M} + RP(k+n) + \frac{nk}{M})\\ &= \widetilde{O}(n\kappa + nM + \frac{nk}{R} + \frac{nk}{P} + RPn + \frac{nk}{M}) \qquad \triangleright\, \varepsilon = O(1), k \le n\end{aligned}$$

Note again that the main difference is that the term P^2R^2k is replaced by RPn. Furthermore, the conditions that need to be satisfied are also slightly more relaxed, as follows:

- $k \le \kappa$, or
- $k > M$ and $n/R < \widetilde{O}(\varepsilon M)$ and $k > \widetilde{\Omega}(P)$ and $k/P \le \varepsilon M$.

Specifically, "$kR/P \le \varepsilon M$" is replaced by "$k/P \le \varepsilon M$". Then, our algorithm can be optimized by setting $\kappa = M = n^{2/3}, R = \widetilde{\Theta}(P) = \widetilde{\Theta}(n^{1/3})$ which gives the total query complexity of $\widetilde{O}(n^{5/3})$. ◀

References

1. Sepehr Assadi. Tight space-approximation tradeoff for the multi-pass streaming set cover problem. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on principles of database systems*, pages 321–335, 2017. doi:10.1145/3034786.3056116.
2. Sepehr Assadi, Sanjeev Khanna, and Yang Li. Tight bounds for single-pass streaming complexity of the set cover problem. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 698–711, 2016. doi:10.1145/2897518.2897576.
3. Baruch Awerbuch, Yossi Azar, and Yair Bartal. On-line generalized steiner problem. *Theoretical Computer Science*, 324(2-3):313–324, 2004. doi:10.1016/J.TCS.2004.05.021.
4. Amir Azarmehr, Soheil Behnezhad, and Mohammad Roghani. Fully dynamic matching: -approximation in polylog update time. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 3040–3061. SIAM, 2024. doi:10.1137/1.9781611977912.109.
5. MohammadHossein Bateni, Hossein Esfandiari, and Vahab Mirrokni. Almost optimal streaming algorithms for coverage problems. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 13–23, 2017. doi:10.1145/3087556.3087585.
6. Soheil Behnezhad. Time-optimal sublinear algorithms for matching and vertex cover. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 873–884, 2021. doi:10.1109/FOCS52979.2021.00089.
7. Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein. Local computation algorithms for maximum matching: New lower bounds. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 2322–2335. IEEE, 2023. doi:10.1109/FOCS57990.2023.00143.
8. Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein. Sublinear time algorithms and complexity of approximate maximum matching. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 267–280. ACM, 2023. doi:10.1145/3564246.3585231.

9 Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein. Approximating maximum matching requires almost quadratic time. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 444–454. ACM, 2024. doi:10.1145/3618260.3649785.

10 Soheil Behnezhad, Mohammad Roghani, Aviad Rubinstein, and Amin Saberi. Beating greedy matching in sublinear time. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*, pages 3900–3945. SIAM, 2023. doi:10.1137/1.9781611977554.CH151.

11 Soheil Behnezhad, Mohammad Roghani, Aviad Rubinstein, and Amin Saberi. Sublinear algorithms for TSP via path covers. In *51st International Colloquium on Automata, Languages, and Programming, ICALP*, volume 297 of *LIPIcs*, pages 19:1–19:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ICALP.2024.19.

12 Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. Dynamic $(1+\varepsilon)$-approximate matching size in truly sublinear update time. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 1563–1588. IEEE, 2023. doi:10.1109/FOCS57990.2023.00095.

13 Sayan Bhattacharya, Peter Kiss, Thatchaphol Saranurak, and David Wajc. Dynamic matching with better-than-2 approximation in polylogarithmic update time. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 100–128. SIAM, 2023. doi:10.1137/1.9781611977554.CH5.

14 Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanita. An improved lp-based approximation for steiner tree. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 583–592, 2010. doi:10.1145/1806689.1806769.

15 Bernard Chazelle, Ronitt Rubinfeld, and Luca Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM Journal on computing*, 34(6):1370–1379, 2005. doi:10.1137/S0097539702403244.

16 Yu Chen, Sanjeev Khanna, and Zihan Tan. Query complexity of the metric steiner tree problem. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4893–4935. SIAM, 2023. doi:10.1137/1.9781611977554.CH179.

17 Miroslav Chlebík and Janka Chlebíková. The steiner tree problem on graphs: Inapproximability results. *Theoretical Computer Science*, 406(3):207–214, 2008. doi:10.1016/J.TCS.2008.06.046.

18 Artur Czumaj and Christian Sohler. Estimating the weight of metric minimum spanning trees in sublinear time. *SIAM Journal on Computing*, 39(3):904–922, 2009. doi:10.1137/060672121.

19 Erik D Demaine, Piotr Indyk, Sepideh Mahabadi, and Ali Vakilian. On streaming and communication complexity of the set cover problem. In *Distributed Computing: 28th International Symposium, DISC 2014, Austin, TX, USA, October 12-15, 2014. Proceedings 28*, pages 484–498. Springer, 2014. doi:10.1007/978-3-662-45174-8_33.

20 Yuval Emek and Adi Rosén. Semi-streaming set cover. *ACM Transactions on Algorithms (TALG)*, 13(1):1–22, 2016. doi:10.1145/2957322.

21 Manuela Fischer and Andreas Noever. Tight analysis of parallel randomized greedy MIS. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 2152–2160. SIAM, 2018. doi:10.1137/1.9781611975031.140.

22 Naveen Garg, Anupam Gupta, Stefano Leonardi, and Piotr Sankowski. Stochastic analyses for online combinatorial optimization problems. In Shang-Hua Teng, editor, *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 942–951, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347185.

23 Edgar N Gilbert and Henry O Pollak. Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 16(1):1–29, 1968.

24 Christoph Grunau, Slobodan Mitrovic, Ronitt Rubinfeld, and Ali Vakilian. Improved local computation algorithm for set cover via sparsification. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2993–3011. SIAM, 2020. doi:10.1137/1.9781611975994.181.

25 Anupam Gupta, MohammadTaghi Hajiaghayi, and Amit Kumar. Stochastic steiner tree with non-uniform inflation. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 134–148. Springer, 2007. doi:10.1007/978-3-540-74208-1_10.

26 Anupam Gupta and Amit Kumar. Online steiner tree with deletions. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 455–467. SIAM, 2014. doi:10.1137/1.9781611973402.34.

27 Anupam Gupta and Martin Pál. Stochastic steiner trees without a root. In *Automata, Languages and Programming: 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005. Proceedings 32*, pages 1051–1063. Springer, 2005. doi:10.1007/11523468_85.

28 Sariel Har-Peled, Piotr Indyk, Sepideh Mahabadi, and Ali Vakilian. Towards tight bounds for the streaming set cover problem. In Tova Milo and Wang-Chiew Tan, editors, *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 371–383. ACM, 2016. doi:10.1145/2902251.2902287.

29 Makoto Imase and Bernard M Waxman. Dynamic steiner tree problem. *SIAM Journal on Discrete Mathematics*, 4(3):369–384, 1991. doi:10.1137/0404033.

30 Piotr Indyk, Sepideh Mahabadi, Ronitt Rubinfeld, Jonathan Ullman, Ali Vakilian, and Anak Yodpinyanee. Fractional set cover in the streaming model. In *20th International Workshop on Approximation Algorithms for Combinatorial Optimization Problem (APPROX 2017)*, 2017.

31 Piotr Indyk, Sepideh Mahabadi, Ronitt Rubinfeld, Ali Vakilian, and Anak Yodpinyanee. Set cover in sub-linear time. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2467–2486. SIAM, 2018. doi:10.1137/1.9781611975031.158.

32 Kumar Joag-Dev and Frank Proschan. Negative association of random variables with applications. *The Annals of Statistics*, 11(1):286–295, 1983.

33 Michael Kapralov, Slobodan Mitrovic, Ashkan Norouzi-Fard, and Jakab Tardos. Space efficient approximation to maximum matching size from uniform edge samples. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1753–1772. SIAM, 2020. doi:10.1137/1.9781611975994.107.

34 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of Symposium on the Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

35 Alam Khursheed and K. M. Lai Saxena. Positive dependence in multivariate distributions. *Communications in Statistics - Theory and Methods*, 10(12):1183–1196, 1981.

36 Reut Levi, Ronitt Rubinfeld, and Anak Yodpinyanee. Brief announcement: Local computation algorithms for graphs of non-constant degrees. In Guy E. Blelloch and Kunal Agrawal, editors, *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2015, Portland, OR, USA, June 13-15, 2015*, pages 59–61. ACM, 2015. doi:10.1145/2755573.2755615.

37 Nicole Megow, Martin Skutella, José Verschae, and Andreas Wiese. The power of recourse for online mst and tsp. *SIAM Journal on Computing*, 45(3):859–880, 2016. doi:10.1137/130917703.

38 Krzysztof Onak, Dana Ron, Michal Rosen, and Ronitt Rubinfeld. A Near-Optimal Sublinear-Time Algorithm for Approximating the Minimum Vertex Cover Size. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1123–1131, 2012. doi:10.1137/1.9781611973099.88.

39 Michal Parnas and Dana Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theor. Comput. Sci.*, 381(1-3):183–196, 2007. doi:10.1016/J.TCS.2007.04.040.

40 Gabriel Robins and Alexander Zelikovsky. Tighter bounds for graph steiner tree approximation. *SIAM Journal on Discrete Mathematics*, 19(1):122–134, 2005. doi:10.1137/S0895480101393155.

41 Barna Saha and Lise Getoor. On maximum coverage in the streaming model & application to multi-topic blog-watch. In *Proceedings of the 2009 siam international conference on data mining*, pages 697–708. SIAM, 2009. doi:10.1137/1.9781611972795.60.

42 David Wajc. Negative association: definition, properties, and applications. *Manuscript, available from https://goo. gl/j2ekqM*, 2017.

43 Yuichi Yoshida, Masaki Yamamoto, and Hiro Ito. An improved constant-time approximation algorithm for maximum matchings. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC*, pages 225–234. ACM, 2009. doi:10.1145/1536414.1536447.