Sublinear Metric Steiner Tree
via
Improved Bounds for Set Cover

Sepideh Mahabadi
(MSR Redmond)

Mohammad Roghani
(Stanford)

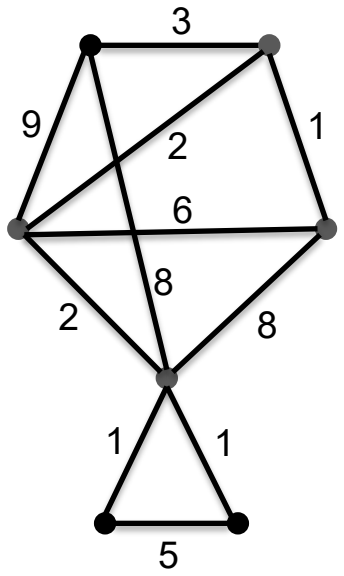
Jakub Tarnawski
(MSR Redmond)

Ali Vakilian
(TTIC)

Steiner Tree

Problem Definition

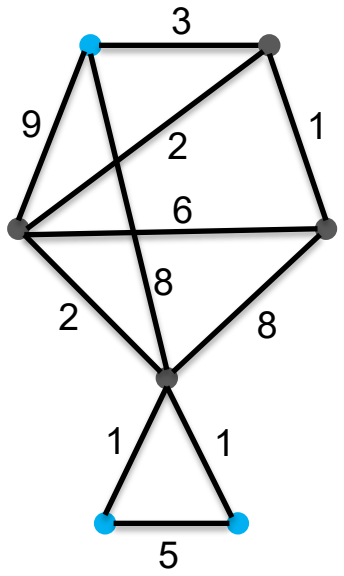
- **Input:** a weighted graph G with n vertices



Steiner Tree

Problem Definition

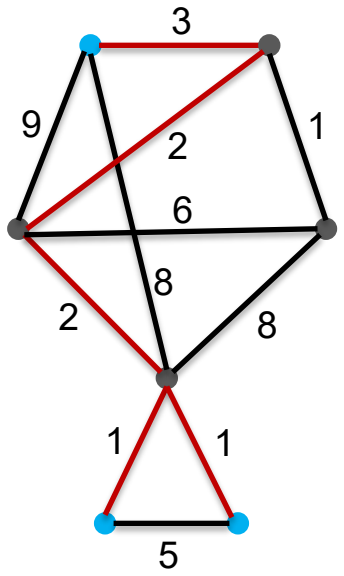
- **Input:** a weighted graph G with n vertices and set T of k terminals.



Steiner Tree

Problem Definition

- **Input:** a weighted graph G with n vertices and set T of k terminals.
- **Output:** a subgraph with minimum costs that connects all the terminals.



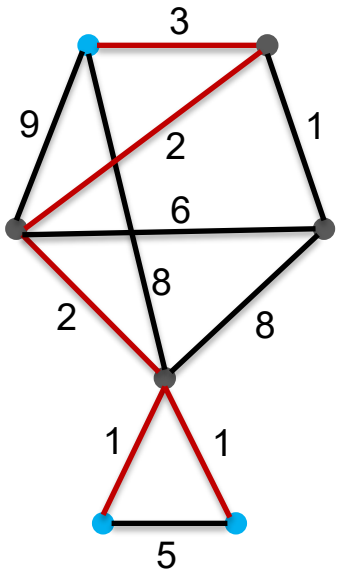
Steiner Tree

Problem Definition

- **Input:** a weighted graph G with n vertices and set T of k terminals.
- **Output:** a subgraph with minimum costs that connects all the terminals.

What is known:

- **Hardness:** $96/95$ APX-hard
  - [Chlebik, Chlebikova'08]
- **Algorithm:** $(\ln 4 + \varepsilon)$-approximation algorithm
  - [Byrka, Grandoni, Rothvos, and Sanita'10]
  - [Goemans, Olver, Rothvos, and Zenklusen'12]
  - [Traub and Zenklusen'21]



Sublinear Metric Steiner Tree

Metric Steiner Tree: the edge weights satisfy the triangle inequality.

- Otherwise no sublinear time algorithm to estimate the cost.

Access model: **Adjacency Matrix Model**

- we have access to the distance matrix of vertices which satisfies the triangle inequality.
- For u and v, what is the distance between them, i.e. $M(u, v)$?

Goal:

- Running time: $n^{2-\Omega(1)}$

What is known

Algorithms:

- MST cost over the metric induced on the terminals is a 2-approximation for the metric Steiner tree. [Gilbert and Pollak'68]
- $(1+\varepsilon)$-approximation for MST in $\tilde{O}(n) \implies (2+\varepsilon)$-approximation for Steiner tree problem in $\tilde{O}(n)$. [Czumaj and Sohler'09]
- $(2-\Omega(1))$-approximation for Steiner tree problem in $\tilde{O}\left(n^{13/7}\right)$. [Chen, Khanna and Tan'23]

Hardness: [Chen, Khanna and Tan'23]

- Outputting the tree itself with factor better than $(2-\varepsilon)$ requires $\Omega(n|T|)$ queries.
- Estimating the cost within $2-\epsilon$ requires requires $\Omega\left(n^{6/5}\right)$.
- Estimating the cost within $5/3-\epsilon$ requires $\Omega\left(n^2\right)$ queries even when the number of terminals is a constant

Our Result

[Chen, Khanna and Tan'23]

- $(2-\Omega(1))$-approximation for Steiner tree problem in $\tilde{O}(n^{13/7})$.
- Estimating the cost within $2-\epsilon$ requires requires $\Omega(n^{6/5})$.

Our Result:

There exists an algorithm that estimates the cost of Steiner tree in $\tilde{O}(n^{5/3})$ time up to a multiplicative factor of $2-\Omega(1)$.

Outline

Outline

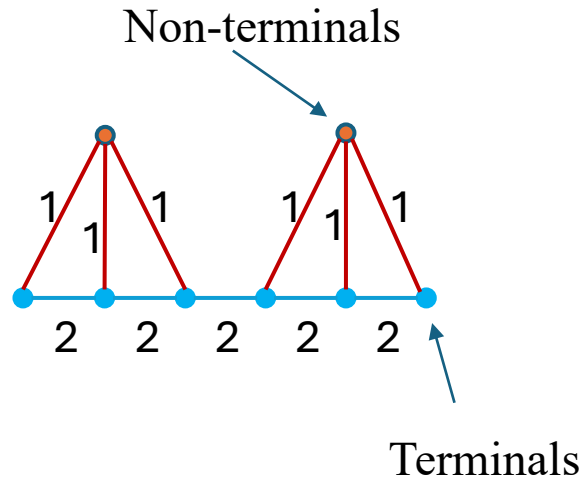
[CKT'23]:	$O(n^{\frac{13}{7}})$	$O(n^{\frac{7}{4}})$
Here:	$O(n^{\frac{5}{3}})$	$O(n^{\frac{5}{3}})$

Outline

Outline

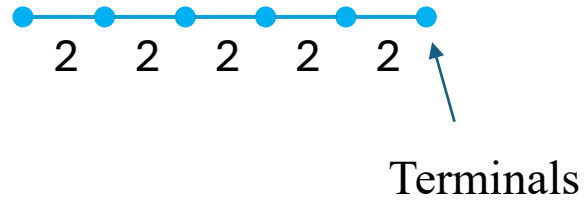
Why Steiner Tree is related to Set Cover

- Suppose that our metric only contains edges of weight 1 or 2.



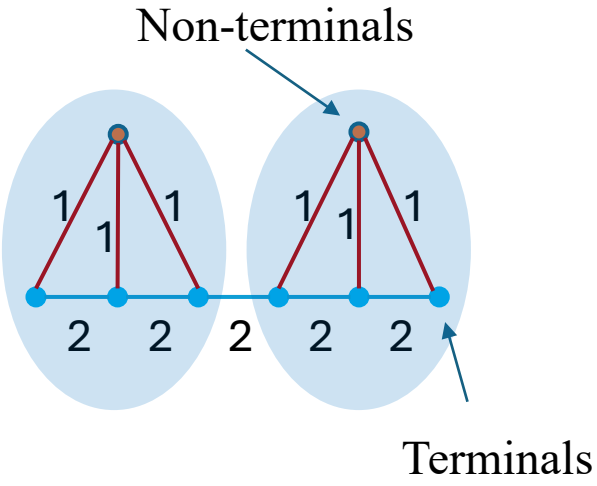
Why Steiner Tree is related to Set Cover

- Suppose that our metric only contains edges of weight 1 or 2 and we start from finding MST of the metric induced on the terminals.
- So up to this point we have 2-approximate solution.



Why Steiner Tree is related to Set Cover

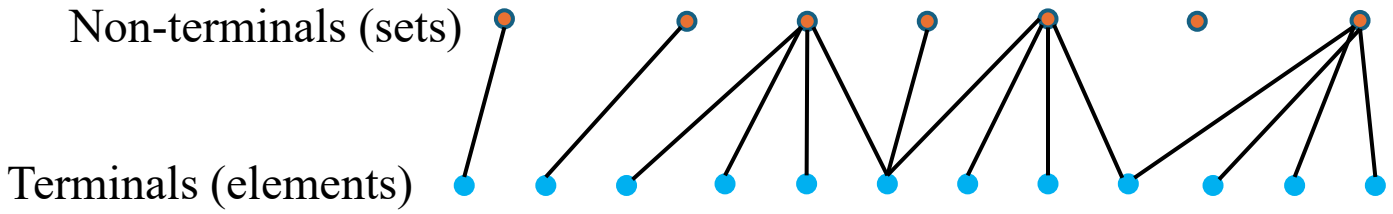
- Suppose that our metric only contains edges of weight 1 or 2 and we start from finding MST of the metric induced on the terminals.
- So up to this point we have 2-approximate solution.
- We try to find local evidence to improve the solution using non-terminals.
- If we remove two edges of weight 2 and add three red edges, we can decrease the cost by 1.



In short

Set Cover Instance:

- **Elements** correspond to terminals
- **Sets** correspond to non-terminal
- A set **contains** an element if the non-terminal has a short edge to the terminal.
- If Set Cover is small, then we can make a lot of improvement over the MST solution.
- If Set Cover is large, then MST itself is already good.



More general metric

1. Compute an MST over terminals using [Czumaj and Sohler'09] which gives a 2 approximation
2. Check weather **improvements over this are possible** using **non-terminal** vertices.
   - **Bucket** the edges based on their weights e.g. $[(1+\epsilon)^i, (1+\epsilon)^{i+1}]$
     - At each weight threshold $\boldsymbol{\tau_i = (1+\epsilon)^i}$ consider the **connected components** of $\boldsymbol{G_\tau}$ with edges upto that threshold: **elements**
     - **Sets** that are **non-terminals**
     - A Set contains an element if the distance between the non-terminal and CC is small
   - **Solution:** Check if the overall contribution of all set cover instances is at least $\eta \cdot W(T)$

➢ Checking if a set contains an element requires checking distance of non-terminal to all terminals in the CC but they compute a net.

- Light and heavy levels, heavy levels are ignorable

Outline

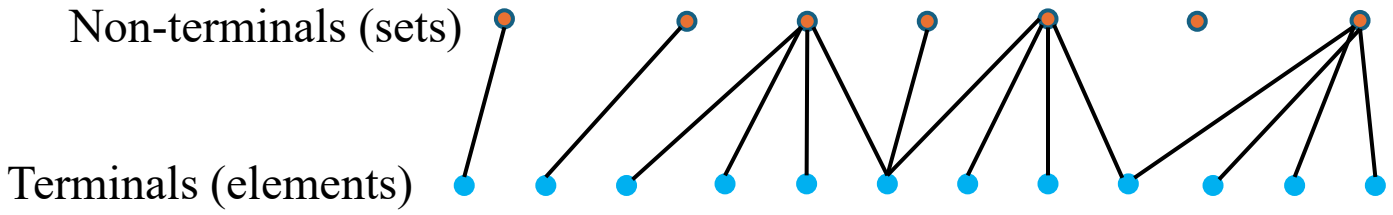
Threshold Set Cover

[Chen, Khanna and Tan'23]: There is a reduction to an instance of set cover problem. (**Threshold Set Cover**)

- **Input:** universe of elements U and collection of sets F.
- **Access Model:** we have access to membership oracle of the instance.
- **Goal:** estimating $|U| - SC(U, F)$, where $SC(U, F)$ is the size of the optimal set cover.

Savings on the solution cost

- Existing literature on sublinear Set Cover but they use adjacency list model, whereas here we have adjacency matrix model



Result

[Chen, Khanna and Tan'23]

- **Theorem.** There exists an algorithm that estimates the cost of threshold set cover in $\tilde{O}(n^{7/4})$ time up to a $(1/4, \varepsilon n)$ multiplicative-additive factor.

Here:

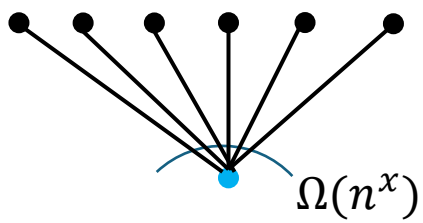
Theorem. There exists an algorithm that estimates the cost of threshold set cover in $\tilde{O}(n^{5/3})$ time up to a $(1/4, \varepsilon n)$ multiplicative-additive factor.

Components

1. Instance sparsification (similar to prior works on sublinear Set Cover ([Indyk, M., Rubinfeld, Vakilian, Yodpinyanee'18], and [CKT'23])
2. Reduction to Matching

Set Sampling Component

- $\boldsymbol{n}$: number of sets $\boldsymbol{k}$: number of elements
- Let $0 < \boldsymbol{x} < 1$ be a parameter that we optimize later.



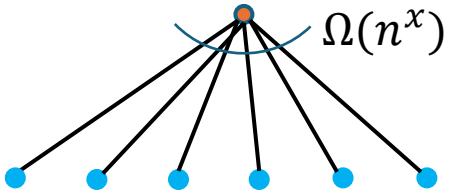
- Sample n^{1-x} sets and add them to the solution (we can afford it because of additive error).

- If an element appears in $\Omega(n^x)$ sets, we expect a large intersection with chosen sets.

➢ We can remove elements with large degrees by spending $O(n^{2-x})$ time.

Element Sampling Component

- $\boldsymbol{n}$: number of sets $\boldsymbol{k}$: number of elements
- Let $0 < \boldsymbol{x} < 1$ be a parameter that we optimize later.



- Iterate over sets one by one and for each, sample n^{1-x} remaining elements.
- If there was a large intersection, add the set to the solution and remove the set and its elements.
- Since each removed set contains $\Omega(n^x)$ elements, this event happens at most $O(n^{1-x})$ time and we can we can afford it because of additive error.

➢ We can remove sets with large degrees by spending $O(n^{2-x})$ time.

Algorithm

- Partitioning terminals based on their degree into T_{low} and T_{high}
- T_{high} can be covered using **set sampling** given the allowed additive error
- Use **element sampling** but iteratively which guarantees that we can add the sets in S_{high} to our solution
  - Only left with S_{low}
- In the end we need to **only** solve Set Cover only for (T_{low}, S_{low})
  - **Degrees** in T_{low} is at most n^x
  - **Degrees** in S_{low} is at most n^x

Outline

Reduction to Matching

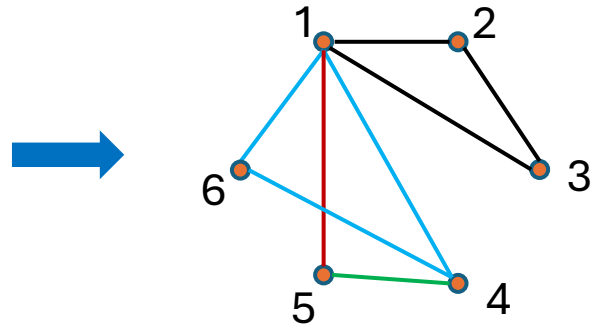
- We construct a graph called $\boldsymbol{H}$, such that:

$S_1 = \{1, 2, 3\}$
$S_2 = \{1, 5\}$
$S_3 = \{4, 5\}$
$S_4 = \{1, 4, 6\}$

Reduction to Matching

- We construct a graph called $\boldsymbol{H}$, such that:
  - Vertices of H are the elements
  - Two vertices are connected if their corresponding elements appear in a set together.

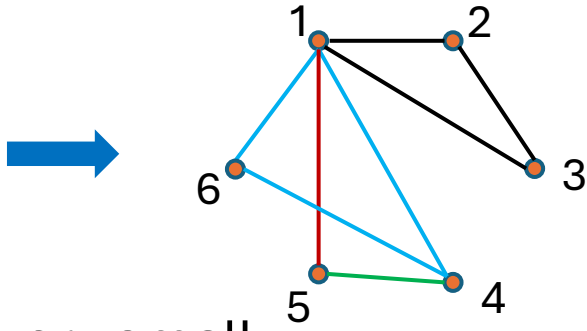
$S_1 = \{1, 2, 3\}$
$S_2 = \{1, 5\}$
$S_3 = \{4, 5\}$
$S_4 = \{1, 4, 6\}$



Reduction to Matching

- We construct a graph called $\boldsymbol{H}$, such that:
  - Vertices of H are the elements
  - Two vertices are connected if their corresponding elements appear in a set together.

$S_1 = \{1, 2, 3\}$
$S_2 = \{1, 5\}$
$S_3 = \{4, 5\}$
$S_4 = \{1, 4, 6\}$



- If $SC(U, F)$ is **very large (roughly n)**, then maximum matching of H is very small.
- If $SC(U, F)$ is small, then maximum matching of H is large.
- We can use the size of approximate maximum matching to estimate $|U| - SC(U, F)$.

Outline

Sublinear Time Matching

- Let d be the average degree of the graph.

- **Theorem** [Behnezhad'21] There exists an algorithm that estimates the size of maximum matching of the graph up to a $(\boldsymbol{1/2, \epsilon n})$ factor in $\tilde{O}(d)$ time given access to adjacency list of the graph.

RGMM Algorithm

[Nguyen Onak’08][Yoshida, Yamamato, Ito’09][Behnezhad’21]

- Pick a random permutation over the edges
- In order to check if an edge is in the matching, we check all the neighbors with a lower rank
  - If any of them in the matching, return NO
  - Otherwise return YES

RGMM Algorithm

[Nguyen Onak'08][Yoshida, Yamamato, Ito'09][Behnezhad'21]

- Pick a random permutation over the edges
- In order to check if an edge is in the matching, we check all the neighbors with a lower rank
  - If any of them in the matching, return NO
  - Otherwise return YES

- **Overall:** Need to sample the above for e.g. $\tilde{O}(1)$ times to get an estimate of the matching size.

RGMM Algorithm

[Nguyen Onak'08][Yoshida, Yamamato, Ito'09][Behnezhad'21]

- Pick a random permutation over the edges
- In order to check if an edge is in the matching, we check all the neighbors with a lower rank
  - If any of them in the matching, return NO
  - Otherwise return YES

- **Need to bound:** the number of queries issued when a vertex is called
- **What is actually bounded:** instead they bound the number of queries made to a vertex.

➢ This gives a bound on the average query time of checking whether an edge is in the matching.

- **Overall:** Need to sample the above for e.g. $\tilde{O}(1)$ times to get an estimate of the matching size.

Sublinear Time Matching

- Let d be the average degree of the graph.

- **Theorem** [Behnezhad'21] There exists an algorithm that estimates the size of maximum matching of the graph up to a $(\mathbf{1/2}, \boldsymbol{\epsilon n})$ factor in $\tilde{O}(d)$ time given access to adjacency list of the graph.

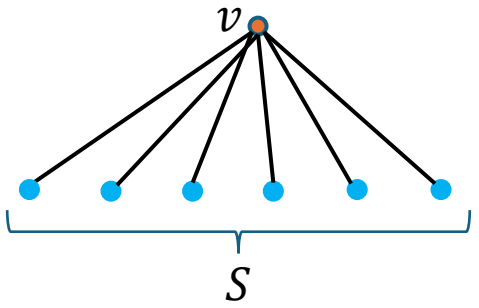
- Average degree of $H = O(n^{2x})$
  - Each element in at most n^x sets
  - Each set has at most n^x elements
- Finding adjacency list of a vertex: $O(n^{1+x})$
- Sparsification cost: $O(n^{2-x})$
- Total cost: $O(\max(n^{2-x}, d \cdot n^{1+x})) = O(\max(n^{2-x}, \ n^{1+3x})) = O(n^{7/4})$
- For $x = 1/4$

Outline

[CKT'23]:	$O(n^{\frac{13}{7}})$	$O(n^{\frac{7}{4}})$
Here:	$O(n^{\frac{5}{3}})$	$O(n^{\frac{5}{3}})$

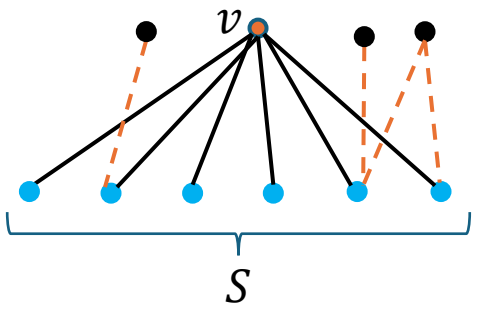
White Box Matching

- **Observation.** The matching algorithm needs a new random neighbor each time in its $\tilde{O}(d)$ explorations and it does not need the entire adjacency list.
- Let v be an element with degree d in H:
- We first query all sets ($O(n)$ time) to find sets S that include v.
- Only $O(n^x)$ sets contain v.



White Box Matching

- **Observation.** The matching algorithm needs a new random neighbor each time in its $\tilde{O}(d)$ explorations and it does not need the entire adjacency list.
- Let v be an element with degree d in H:
- We first query all sets ($O(n)$ time) to find sets S that include v.
- Only $O(n^x)$ sets contain v.
- We randomly query pairs of elements and a set in S.
- We expect to find a random neighbor of v in H after n^{1+x}/d queries.
- So the total cost for seeing a random neighbor is: $O(n + n^{1+x}/d)$



Issue

- A neighbor u of an element/vertex v in H may appear in many sets,
  - {1,2}, {1,2,3}, {1,2,4,5}
- Our sampling is not uniform: it has a bias towards those neighbors that appear is many sets with v
  - 2 is three times more likely to be returned as a neighbor of 1 as opposed to 3,4,5

Generalization of RGMM to Multigraphs

- A neighbor u of an element/vertex v in H may appear in many sets,
  - {1,2}, {1,2,3}, {1,2,4,5}
- Our sampling is not uniform: it has a bias towards those neighbors that appear is many sets with v
  - 2 is three times more likely to be returned as a neighbor of 1 as opposed to 3,4,5
- We let H to be a **multigraph**
  - we can produce a uniform sample of neighbors of a vertex in the multigraph

➢

Generalization of RGMM to Multigraphs

- A neighbor u of an element/vertex v in H may appear in many sets,
  - {1,2}, {1,2,3}, {1,2,4,5}
- Our sampling is not uniform: it has a bias towards those neighbors that appear is many sets with v
  - 2 is three times more likely to be returned as a neighbor of 1 as opposed to 3,4,5
- We let H to be a **multigraph**
  - we can produce a uniform sample of neighbors of a vertex in the multigraph

➢ Extend the analysis of sublinear matching algorithm by [Behnezhad'21] to the multigraphs.

Second Challenge

What we have:

- How to sample a random neighbor of a vertex v in time n^{1+x}/d_v

What we know:

- The average number of queries made to a vertex is d
- If all vertices had degree $d_v = d$, then the query time would have been $d \times \frac{n^{1+x}}{d} \approx n^{1+x}$

Putting everything together

- The sparsification cost: $O(n^{2-x})$
- Exploration size of the matching algorithm: $d = O(n^{2x})$
- The cost of finding a random neighbor in H: $O(n + n^{1+x}/d)$
- Total cost: $O(n^{2-x} + d\left(n + \frac{n^{1+x}}{d}\right)) = O\left(n^{2-x} + dn + n^{1+x}\right) = O(n^{2-x} + n^{1+2x})$
- Setting $x = 1/3$ we have, the total cost will be $O(n^{5/3})$

Outline

[CKT'23]:	$O(n^{\frac{13}{7}})$	$O(n^{\frac{7}{4}})$
Here:	$O(n^{\frac{5}{3}})$	$O(n^{\frac{5}{3}})$

Second Challenge

What we have:

- How to sample a random neighbor of a vertex v in time n^{1+x}/d_v

What we know:

- The average number of queries made to a vertex is d
- If all vertices had degree $d_v = d$, then the query time would have been $d \times \frac{n^{1+x}}{d} \approx n^{1+x}$

Second Challenge

What we have:

- How to sample a random neighbor of a vertex v in time n^{1+x}/d_v

What we know:

- The average number of queries made to a vertex is d
- If all vertices had degree $d_v = d$, then the query time would have been $d \times \frac{n^{1+x}}{d} \approx n^{1+x}$

Potential issue: Vertices with degree far from the average degree.

- Maybe we get to a low degree vertex more often

Second Challenge

What we have:

- How to sample a random neighbor of a vertex v in time n^{1+x}/d_v

What we know:

- The average number of queries made to a vertex is d
- If all vertices had degree $d_v = d$, then the query time would have been $d \times \frac{n^{1+x}}{d} \approx n^{1+x}$

Potential issue: Vertices with degree far from the average degree.

- Maybe we get to a low degree vertex more often

What we show:

- We proved that the expected number of times the matching algorithm visits each vertex is proportional to its degree d_v

Carry everything back to the Steiner Forest

Implementation in ST

- **Sample** a vertex v in H : sample an element in **set cover instance** : sample a **CC** in G_τ
  - Sample a vertex in G
  - Run e.g. BFS-type algorithms to find the connected component CC
  - If size of CC is becoming too large, ignore it (need to show error is small)
  - If the **size** of the CC is z then, then keep it only w.p. $1/z$ so that the sample is uniform

- **Random neighbor** of v in H : Random (element,set) pair in **set cover instance** : distance of a **non-terminal** to **all terminals in CC**
  - Check distance of non-terminal to all terminals: Costly
  - Building a **net** on CC
  - **Ignore** CC with **large net size**
  - Check distance of **non-terminal** to all **net terminals**.

Outline

[CKT'23]:	$O(n^{\frac{13}{7}})$	$O(n^{\frac{7}{4}})$
Here:	$O(n^{\frac{5}{3}})$	$O(n^{\frac{5}{3}})$

Open Questions

- Close the gap ($O(n^{\frac{5}{3}})$ vs $\Omega(n^{\frac{6}{5}})$) in the number of queries to get a $(2-\eta)$ approximation for the Metric Steiner Tree

- Can we get a better approximation (beating 2 by a large constant) in sublinear time?