Differentially Private All-Pairs Shortest Distances for Low Tree-Width Graphs

1st Javad B. Ebrahimi
Sharif University of Technology
Institute for Research in Fundamental Sciences (IPM)
Tehran, Iran
javad.ebrahimi@sharif.edu

2nd Alireza Tofighi Mohammadi
Sharif University of Technology
Tehran, Iran
a.tofighi77@sharif.edu

3rd Fatemeh Kermani
Sharif University of Technology
Tehran, Iran
f.kermani@sharif.edu

***Abstract*—In this paper, we present a polynomial time algorithm for the problem of differentially private all pair shortest distances over the class of low tree-width graphs. Our result generalizes the result of Sealfon [9] for the case of trees to a much larger family of graphs. Furthermore, if we restrict to the class of low tree-width graphs, the additive error of our algorithm is significantly smaller than that of the best known algorithm for this problem, proposed by Chen et. al. in [2].**

***Index Terms*—differential privacy, algorithms, shortest path, graph theory, tree-width.**

I. Introduction

A. Differential Privacy

Privacy-preserving data analysis is a way of learning about population while keeping confidential information about individuals private. *Differential privacy* introduced in the work of Dwork et. al. [4] is a definition and clarification of this concept.

There are two main ingredients to make an algorithm privacy preserving. The first one is to answer queries in a randomized way. That is, to output a random element from the set of all possible outcomes. Equivalently, the output can be modeled as a probability distribution over the set of possible outcomes. The second point is to make sure that adding or removing any individual from the dataset does not significantly change the output distribution. Datasets which are only different in one individual, are called neighboring datasets. To generalize *neighboring* definition in a more abstract context, we can consider the case where the domain of the algorithm is a metric space $(\mathcal{X}, d)$. In this case, we say $x, y \in \mathcal{X}$ are neighbors if $d(x, y) \leq 1$. As an example, when $\mathcal{X}$ is the set of integral vectors and d is the ℓ_1-metric, neighboring elements are of the form $x, y \in \mathcal{X}$ such that x, y differ in exactly one co-ordinate and in that co-ordinate, they are equal to two consecutive numbers.

One may observe that if a mechanism outputs the same distribution regardless of the input, it will be completely private. However, the main challenge in differential privacy is to design differential private mechanism with high level of privacy, while, approximating a target function reasonably accurate. More precisely, for a target function f on a metric space $(\mathcal{X}, d)$, $\mathcal{M}$ is a DP mechanism if for any close point $x_1, x_2 \in \mathcal{X}$, $\mathcal{M}(x_1), \mathcal{M}(x_2)$ are *close* distributions and $\mathcal{M}(x)$ is a good approximation of $f(x)$. In section IV we make this definition mathematically precise.

B. Sensitivity

The ℓ_1 sensitivity of a function f is the maximum value that the change of a single individual's data in input can change the function f's output. Therefore, intuitively, sensitivity measures the minimum amount of uncertainty we must have in the response in order to guarantee keeping an individual's data private.

The output of functions with higher sensitivity must be perturbed more to preserve a certain extent of privacy, and for functions with lower sensitivity, we can add lower noise to guarantee differential privacy with the same amount of privacy leak.

By choosing the noise from a family of the distributions, called Laplace distributions, we can guarantee that the mechanism is private and the error is small. For more details about Laplace noise and the proof of this statement, see [5] and for the formal definitions of the ε-differential privacy, error and Laplace mechanism see section IV.

C. Differential privacy on graphs

One of the canonical problems in graph theory and computer science is to compute the shortest path between a pair of vertices of a given graph. This problem is well-explored and efficient algorithms have been proposed in the literature. However, we are interested in the differential private version of this problem. This problem was first proposed by Sealfon in [9]. Now, we explain the model of the problem.

Let $G = (V, E)$ be a graph on the vertex set V and the edge set E and let w be the edge weight function that assigns non-negative weights to the edges of the underlying graph $G = (V, E)$. Suppose that G is publicly known but w is private. Each dataset is a weight function w and two datasets w_1, w_2 are neighbors when $\|w_1 - w_2\|_1 := \sum_{e \in E} |w_1(e) - w_2(e)| \leq 1$.

In the all pairs shortest distances problem, the target function is $f : \mathcal{X} \to \mathbb{R}^{\binom{n}{2}}$. Where $\mathcal{X}$ is the set of all possible weights on edges of G and for a $w \in \mathcal{X}$, $f(w)$ is a vector of shortest distances of all pairs of vertices.

The goal is to efficiently construct a differential private mechanism to approximate f with minimum ℓ_∞ error. We call this error the additive error of the algorithm.

The rest of this paper is organized as follows. In the next section, we review the history of differentally private APSD problem and the previous results on this problem. In Section III, we try to explain the main challenges to generalizing the Sealfon's approach from trees to low tree-width graphs. Then we describe our solution to overcome these challenges in several stages. Also, we explain our algorithm for differentially private APSD problem informally. A high level justification of the correctness of the algorithm is also provided in that section. Next, we overview some preliminaries and introduce basic definitions and notations. Section V consists of the formal description of the algorithm, the proof of its correctness and analysing the time complexity of the algorithm. Finally, in the last section we conclude our paper.

II. Related Works

Differential privacy has been applied to graph problems, including the all-pairs shortest distances (APSD for short) problem. Sealfon [9] was the first to formally study APSD with privacy. The paper introduces a model for differentially private analysis of the shortest distances in a weighted graphs in which the graph topology is assumed to be publicly known and the private information consists only of the edge weights. In the DP framework, he required that the algorithm be (ε, δ)-DP, where neighboring data sets (i.e. inputs) correspond to those whose weight vectors w, w' which differs by at most 1 in the ℓ_1-distance. Sealfon gave an $O(n \log n/\varepsilon)$-error "input-perturbation" algorithm for APSD, which adds Laplace noise to all edge weights and computes the shortest path in this resulting graph with noisy weights.

When the underlying graph is a tree, Sealfon developed a DP mechanism with significantly smaller error. His main idea is to employ a standard technique in graph algorithm known as "Centroid decomposition" of trees. He first finds the unique path from the root to all the vertices and then, for every pair of vertices, with 3 queries, he can compute the distances between the two.

The idea is to split the tree into subtrees of at most half the size of the original tree. As long as we can release the distance from the root to each subtree with small error, we can then, recurse on the subtrees. Sealfon showed that there is an algorithm that is ε-differentially private on T such that on the input $w : E \to \mathbb{R}^+$, outputs approximate distances between all pairs of vertices. Also, with probability at least $1-\gamma$, the additive error on each output distance is $O(\log^{2.5} V \cdot \log(1/\gamma)/\varepsilon)$ for any $\gamma \in (0, 1)$.

Fan and Li [6] revisited the problem of privately releasing approximate distances between all pairs of vertices in a graph. They proposed improved algorithms with smaller error term to that problem for grid graphs and trees.

Fan et al. [7] also generalized Sealfon's approach for trees to graphs that with removing few nodes become acyclic. The subset of vertices that removing make graph without any cycles called feedback vertex set.

Chen et al. in [2] also studied this problem. They gave an ε-DP algorithm with additive error $\tilde{O}(n^{2/3}/\varepsilon)$ and an (ε, δ)-DP algorithm with additive error $\tilde{O}(\sqrt{n}/\varepsilon)$ where n denotes the number of vertices. This is the best known additive error for arbitrary graphs.

III. Technical Overview

Two of the commonly used ideas of differential privacy, are to add random noise to the input (input perturbation), or to add random noise to the output of the algorithm (output perturbation). In both cases, the output of the algorithm is a random function which estimates the desired function and has some error.

For instance, if we add *i.i.d.* Laplace noises to the input, (i.e. edge weights), and then, compute the shortest path according to noisy weights, the magnitude of error to achieve ε-DP is $1/\varepsilon$. [1] Since every path in a graph has at most $n-1$ edges, we may need to use $\Theta(n)$ noisy values to compute each shortest distance. Thus, the error is proportional to $\Theta(n \cdot 1/\varepsilon)$ (see Algorithm 3 of [9]). Alternatively, if we add Laplace noise to the output, i.e. the weights are exact and after computing the shortest distances we add the noise to the result. Since there are $\Theta(n^2)$ pairs of shortest distances, in order to achieve ε-DP, the magnitude of the noise must be $\Theta(n^2/\varepsilon)$. Thus, the error is proportional to $\Theta(n^2/\varepsilon)$ (see Section 4 of [9]).

Note that in the both cases, the error is proportional to the product of the magnitude of noise and the number of needed noisy values to release each shortest distance. Therefore, roughly speaking, in order to reduce the magnitude of error, we must reduce either the magnitude of the noises or the number of added noises or both.

For trees, Sealfon in Section 4.1 of [9] invented an elegant idea to decrease the magnitude of noise and the number of noisy values needed to compute each of the shortest distances.

The algorithm constructs an intermediate graph, which has two important properties: First, if we change the weights of the input graph by at most 1, only $O(\log n)$ of the weights in the intermediate graph will change and they change no more than 1. Secondly, for all $v, u \in V$, the weight of an $O(\log n)$-hop shortest path between v and u in the intermediate graph, is equal to the shortest distance between v and u in the input graph. Thus, we can use $O(\log n/\varepsilon)$-magnitude Laplace noises and we only need $O(\log n)$ noisy values to compute each shortest distance. Hence, the magnitude of the error is roughly $O(\log^2 n/\varepsilon)$. [2]

Example 1 (Sealfon's construction of the intermediate graph for a path): Figure 1 gives an example of the graph made by Sealfon's algorithm. The ℓ_1 sensitivity of the graph is at most

[1] In order to achieve ε-DP, we can add a Laplace noise with parameter $1/\varepsilon$. In Section IV we define the Laplace distribution and see that the magnitude of such noise will be proportional to $1/\varepsilon$ with high probability.

[2] The exact value is $O(\log^{2.5} n/\varepsilon)$. The extra log factor is because this property with high probability.

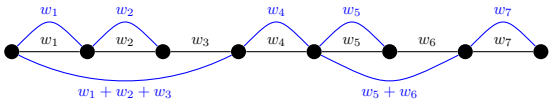

Fig. 1. Making an intermediate graph for a path. black edges are the base graph and the blue edges are the intermediate graph.

2, i.e. changing the weight of a black edge affects at most 2 of the blue edges

In this paper, we aim to extend the Sealfon's idea beyond trees. We address several challenges due to inherent differences between trees and low tree-width graphs. These challenges include:

1) The edges of the shortest path between two specific vertices in a tree, only depend on the publicly known topology of the graph and they are unique. So the desired algorithm, only needs to privately release the summation of private weights of the edges of the unique paths. Clearly, this property is not true for non-tree graphs.
2) In trees, due to uniqueness of shortest paths, it is possible to calculate the all pairs shortest distances using only the shortest distances between the root and other vertices. This can be achieved by computing the lowest common ancestor for each pairs of vertices and calculating the distances from the root to every vertex. This property of trees allows for efficient calculation of all pairs shortest distances using only the root to all distances.
3) Trees have a vertex called the centroid. The centroid is a vertex that, removing it, separates the tree into connected components which have sizes no greater than half the size of the original tree.

This paper introduces Algorithm 3 that calculates all pairs shortest distances with differential privacy on low tree-width graphs and overcome the challenges above. The algorithm takes $G = (V, E)$ and T as input. Where T is tree decomposition of G and T has width p. Also assume that $|V| = n$. The algorithm consists of three stages. In the first stage, we construct an intermediate graph with the properties that mentioned above, in the second stage, we ensure the differential privacy by adding noises to the intermediate graph and in the last one, the algorithm estimates all pairs shortest distances.

Stage 1: Algorithm 2, uses a divide and conquer approach that constructs an intermediate graph G' by adding edges to the original graph. By proposition 2, the algorithm exploits T and finds a subset of vertices, called separator, which removing it partitions graph into components with half size. Thus, the depth of recursion call of the algorithm will be $O(\log n)$. We use separator instead of centroid to address challenge 3.

The edges of intermediate graph are computed by the ComputeEdges recursive function in Algorithm 1. This function takes as input, a weighted graph $G = (V, E)$, a tree decomposition T of G, and a subset of vertices V_0 of G. The union of V_0 and separator act like root in tree. we call the union, V_0'. We deal challenge 2 using V_0'. In each recursion call of ComputeEdges, lemma 4 shows that the shortest distances between all $v \in V_0'$ and V is equals to $O(\log n)$-hop shortest distance between v and u in the returned graph.
We need to extend the result to all pairs shortest distances. To do so, we need a wise selection of smaller problems and V_0. We construct smaller versions of the problem by combining the connected components after removing the separator and the separator itself. We also propagate V_0' for V_0 of smaller versions of the problem in the divide and conquer approach. In Lemma 5, we show that with this selection, we can extend the result from V_0' to V for all pairs shortest distances.

Stage 2: Lemma 3 shows that the sensitivity of the intermediate graph is low. Thus, the algorithm then adds Laplace noise to the edge weights of G' to provide differential privacy. The scale of the Laplace noise is $O(p^2 \log^2 n/\varepsilon)$, where p is the tree-width of the graph.

Stage 3: In the description of stage 1, we mentioned that for each $v, u \in V$, the shortest distance between v and u is equal to an $O(\log n)$-hop shortest distance in the intermediate graph. The algorithm uses post-processing Proposition 4 to return all-pairs shortest path distances for paths with at most $O(\log n)$ hops in the graph G' with noisy edge weights. The minimization of the weight of all $O(\log n)$-hop paths addresses challenge 1.

We uses Laplace noises, thus, with high probability, the magnitude of each noise is proportional to $O(\log n \cdot p^2 \log^2 n/\varepsilon)$. We minimize over $O(\log n)$-hop paths, thus we use only $O(\log n)$ noises to estimate each shortest distance. Thus, with high probability, the magnitude of the additive error of each shortest distance is at most $O(p^2 \log^4 n/\varepsilon)$. This result outperforms previous results in ε-DP when $p = o(n^{1/3})$.

In section VII, we provide a detailed description of the technical aspects of the algorithm and its proof.

IV. Preliminaries

In this section we describe the main concepts of graph theory and differential privacy, that are used in this paper.

A. Graph Theory

Let $G = (V, E)$ denote an undirected graph with vertex set V and edge set E. we also show V and E by $V(G)$ and $E(G)$ respectively and let $w : E \to \mathbb{R}^+$ be a weight function. Let $|V|$ and $|E|$ be the number of vertices and edges, respectively. For $X, Y \subseteq V$, let donate $E[X, Y]$ be all edges between X and Y.

Let $\mathcal{P}_{xy}^G$ denote the set of paths between a pair of vertices $x, y \in V(G)$. For any path $P \in \mathcal{P}_{xy}^G$, the weight $w(P)$ is the sum $\sum_{e \in P} w(e)$ of the weights of the edges of P. The distance $d_{G,w}(x, y)$ from x to y denotes the weighted shortest distance $\min_{P \in \mathcal{P}_{xy}^G} w(P)$. We will denote the hop length $\ell(P)$ of path $P = (v_0, \ldots, v_\ell)$ is the number ℓ of edges on the path. Also the k-hop distance $d_{G,w}^k(x, y)$ between x and y denotes as the minimum weight between at most k-hop paths $d_{G,w}^k = \min_{P \in \mathcal{P}_{xy}^G, \ell(P) \leq k} w(P)$.

Definition 1 (Tree decomposition): [1], [3] A tree decomposition of a graph $G(V, E)$ is a labeled tree T, where each node i of T is labeled by a subset (bag) $B_i \subset V$ of vertices of G, each edge of G is in a subgraph induced by at least one of the B_i, and the nodes of T labeled by any vertex $v \in V$ are connected in T. The width of T is maximum of cardinality of bags of T minus 1.

Definition 2 (Tree-width): [1], [3] The tree-width of G is the minimum integer p such that there exists a tree decomposition G with width p.

It is known that a tree has tree-width 1, a series-parallel graph has tree-width 2, a k-clique has tree-width $k-1$, and an n by n grid has tree-width $\Theta(n)$.

Proposition 1 (Reduced tree decomposition): Let $G = (V, E)$ be a graph of tree-width k, and let $H = (V', E')$ be a subgraph of G. If T a width-k tree decomposition of G, then, by removing nodes that are not in V' from the bags of T and removing any bags that become subsets of other bags, we obtain a reduced tree decomposition T' of H with width k, and no bag of T' being a subset of another.

Proposition 2 (Tree-width implies separator): [8] Let $G(V, E)$ be a graph of tree-width p. Then, there is a set S of at most $p+1$ such that every connected component of $G \setminus S$ has no more than half of vertices. We call this set the separator of graph and we can find it with $O(|V|^2)$ time complexity by tree decomposition of graph.

B. Differential Privacy

We now formally define differential privacy in the private edge weight model.

Definition 3 (Neighboring weights): For any edge set E, two weight functions $w, w' : E \to \mathbb{R}^+$ are neighboring, denoted $w \sim w'$, if

$$\|w - w'\|_1 = \sum_{e \in E} |w(e) - w'(e)| \leq 1$$

Definition 4 (Mechanism): A randomized algorithm $\mathcal{M}$ with domain A and discrete range B is associated with a mapping $\mathcal{M} : A \to \Delta(B)$. On input $a \in A$, the algorithm $\mathcal{M}$ outputs $\mathcal{M}(a) = b$ with probability $(\mathcal{M}(a))_b$ for each $b \in B$. The probability space is over the coin flips of the algorithm $\mathcal{M}$.

Definition 5 (Differential privacy, [4]): A randomized algorithm $\mathcal{M} : A \to R$ is (ε, δ)-differentially private if for all $\mathcal{S} \subseteq R$ and for all $x, y \in R$ such that $x \sim y$:

$$\Pr[\mathcal{M}(x) \in \mathcal{S}] \leq \exp(\varepsilon) \Pr[\mathcal{M}(y) \in \mathcal{S}] + \delta$$

where the probability space is over the coin flips of the mechanism $\mathcal{M}$.

If $\delta = 0$, we use ϵ-differential privacy or ϵ-DP as ϵ-differential privacy.

Definition 6 (Sensitivity of a function): The ℓ_1-sensitivity of a function $f : A \to \mathbb{R}^k$ is:

$$\Delta_1 f = \max_{x,y \in A, x \sim y} \|f(x) - f(y)\|_1$$

The Laplace distribution with scale parameter $b > 0$ is defined by the probability density function

$$f(x) = \frac{1}{2b} \exp\left(-\frac{|x|}{b}\right).$$

For a random variable $X \sim \text{Lap}(b)$, the probability that $|X| \geq t \cdot b$ for some $t > 0$ is given by $\Pr(|X| \geq t \cdot b) = 2\exp(-t)$.

Definition 7 (Laplace mechanism): Given any function $f : A \to \mathbb{R}^k$, the Laplace mechanism is defined as:

$$\mathcal{M}_L(x, f(\cdot), \varepsilon) = f(x) + (Y_1, \ldots, Y_k)$$

where Y_i are *i.i.d.* random variables drawn from $\text{Lap}(\Delta_1 f/\varepsilon)$.

In order to be able to use the Laplace mechanism, we should know that it preserve the privacy of the algorithm. Theorem 3.6 of [5] states as bellow.

Proposition 3 (Laplace mechanism is ε-DP [5]): The Laplace mechanism $\mathcal{M}_L(x, f(\cdot), \varepsilon)$ is ε-DP.

Proposition 4 (Post-Processing): Let $\mathcal{M} : \mathbb{R}^{|\mathcal{X}|} \to R$ be a randomized algorithm that is (ε, δ)-differentially private. Let $f : R \to \mathbb{R}'$ be an arbitrary randomized mapping. Then, $f \circ \mathcal{M} : \mathbb{R}^{|\mathcal{X}|} \to \mathbb{R}'$ is $(\varepsilon, \delta)-$ differentially private.

V. Main result

In Section III, we described the intuition behind the algorithm and its properties. In this section, we will present the details of the algorithm implementation in pseudo-code form. Next, we will present the technical aspects of its properties and correctness.

The following theorem, states that Algorithm 3 has the desired properties.

Theorem 1 (Main theorem): Let $G = (V, E)$ be a graph of tree-width p. Then, there exists an ε-differentially private algorithm that takes as input a weight function $w : E \to \mathbb{R}^+$ and releases the all-pairs shortest path distances of G with weight w. For any $\gamma \in (0, 1)$, the error is bounded by $O(\log(1/\gamma) \cdot p^2 \cdot \log^4(|V|)/\varepsilon)$ with probability at most $1 - \gamma$.

To prove that Algorithm 3 satisfies the conditions of our main theorem (Theorem 1), we need to establish several properties of Algorithms 1, 2, and 3.

We present these properties in the form of following lemmas. The proofs can be found in Section VII.

The first property we will show is that the recursion depth of Algorithm 1 is not too large.

Lemma 1 (Recursion depth of algorithm is logarithmic): Let $G = (V, E)$ be a graph of tree-width p and let T be its corresponding tree-decomposition. Then, the recursion depth of Algorithm 1 on input G and T is at most $O(\log |V|)$.

In the previous lemma, we demonstrated that the depth of the Algorithm 1 is $O(\log |V|)$. Since the size of the starting set V_0 increases by at most $O(p)$ at each depth, there is an upper bound for the size of the starting set in all recursive steps. The following lemma presents this condition in a formal format.

Lemma 2 (Upper bound on starting set size): When running the ComputeEdges(G, w, T, V_0) function on a graph $G = (V, E)$ with tree-width p and a set of starting vertices $V_0 \subseteq V$,

the size of the starting vertex set in all recursive steps of the function is bounded by $O(|V_0| + p \cdot \log |V|)$.

Now, it's time to find an upper bound for the sensitivity of the function.

Lemma 3 (Sensitivity of algorithm): Let $G = (V, E)$ be a graph of tree-width p and let T be its corresponding tree-decomposition. Let $V_0 \subseteq V$ be an arbitrary set of starting vertices. If $f(w) = \text{CONSTRUCTGRAPH}(G, w, T, V_0)$, then, the ℓ_1 sensitivity of f is $O\left((|V_0| + p) \cdot p \cdot \log^2 |V| + p^2\right)$.

By Lemma 3, we know that to guarantee ε-DP, the magnitude of each noise can be bounded by $O(p^2 \log^2 |V|/\varepsilon)$. The next two lemmas state that we can also involve only a small number of shortcuts to compute the shortest distances.

Algorithm 1 Determining Shortcut Edges for Addition to The Intermediate Graph

1: **function** COMPUTEEDGES($G = (V, E), w : E \to \mathbb{R}^+, T, V_0 \subseteq V$)
Input: G is the graph with weight function w. T is the tree decomposition of G with maximum bag size $p + 1$, and V_0 is a helper argument called the starting set.
Output: A list of triples (v, u, x) where $v, u \in V$ and $x \in \mathbb{R}^+$ that should be added to the intermediate graph for computing distances with few hops
2: $R \leftarrow \emptyset$ ▷ Array R is a list of weighted edges
3: **if** $|V| \leq 6(p + 1)$ **then**
4: **for all** $v \in V$ **do**
5: **for all** $u \in V$ **do**
6: Insert $(v, u, d_{G,w}(v, u))$ into R
7: **else** ▷ Partition the graph into smaller components using a bag from the tree decomposition
8: Let S be the bag of T that partitions G into components of size at most $|V|/2$ by proposition 2
9: $V_0' \leftarrow V_0 \cup S$
10: Let the connected components of $G \setminus S$ be $C_1, \ldots, C_l$
▷ Add edges between vertices in the starting set and separator
11: **for all** $v \in V_0'$ **do**
12: **for all** $u \in S$ **do**
13: Insert $(v, u, d_{G,w}(v, u))$ into R
▷ Recursively compute edges for each component
14: **for all** $i \in [l]$ **do**
15: $H_i \leftarrow (V(C_i) \cup S, E(C_i) \cup E[C_i, S])$
16: $V_i \leftarrow (V(C_i) \cap V_0')$
17: Let T_i be the reduced tree decomposition T for H_i using Proposition 1
18: Let w_i be the restriction of function w to edges in H_i
19: $R_i \leftarrow$COMPUTEEDGES(H_i, w_i, T_i, V_i)
20: **for all** $(v, u, x) \in R_i$ **do**
21: Insert (v, u, x) into R
22: **return** R
23: **end function**

The following lemma states that we can compute the shortest distance between each pair $v \in V_0 \cup S$ and $u \in V$ using only $O(\log |V|)$-hop paths in the intermediate graph.

Lemma 4 (Distance preservation of intermediate graph: Some pair case): Let $G = (V, E)$ be a weighted graph of tree-width p, T be its corresponding tree-decomposition and $w : E \to \mathbb{R}^+$ be the weight function. Let $V_0 \subseteq V$ be an arbitrary subset of vertices. Let (G', w') be the output of $\text{CONSTRUCTGRAPH}(G, w, T, V_0)$ and S be the bag selected on first call of COMPUTEEDGES on line 8 in Algorithm 1. Then, for all $v \in V_0 \cup S$ and $u \in V$, we have $d_{G,w}(v, u) = d^l_{G',w'}(v, u)$ where $l = \max(2, \log_{1.5} |V|)$.

Now, it's time to improve upon the result of the last lemma. In the next lemma, we will prove that the result is also correct for all pairs shortest distances.

Lemma 5 (Distance preservation of intermediate graph: All pair case): Let $G = (V, E)$ be a weighted graph of tree-width p and the weight function $w : E \to \mathbb{R}^+$. Let T be a tree-decomposition of G, and let $(G', w') = \text{CONSTRUCTGRAPH}(G, w, T, \emptyset)$. Let S be the bag selected on the first call of COMPUTEEDGES on line 8 in Algorithm 1. Then, for all v and $u \in V$, we have $d_{G,w}(v, u) = d^l_{G',w'}(v, u)$ where $l = 2 \cdot \max(2, \log_{1.5} |V|)$.

Proof of Theorem 1: Let $\mathcal{M}(G, T, p, w, c, \varepsilon)$ be the following mechanism. Run Algorithm 3. Lemma 3 asserts that the ℓ_1 sensitivity of the algorithm after the first stage is $O\left((0 + p) \cdot p \cdot \log^2 |V| + p^2\right)$. Then, there exists some c_1 such that the ℓ_1 sensitivity after the first stage is at most $c_1 \cdot \left(p^2 \cdot \log^2(|V|)\right)$. This allows us to use the Laplace mechanism. By the Laplace mechanism 3, after the second stage, the algorithm is ε-DP for all $c \geq c_1$. So, by post-processing 4, $\mathcal{M}(G, T, p, w, c, \varepsilon)$ is ε-DP for all $c \geq c_1$.

Algorithm 2 Constructing a Low ℓ_1-Sensitivity Intermediate Graph for Efficient APSD Computation

1: **function** CONSTRUCTGRAPH($G = (V, E), w : E \to \mathbb{R}^+, T, V_0 \subseteq V$)
Input: Graph G with weight function w, tree decomposition T of G with maximum bag size $p+1$, and starting set V_0
Output: A graph in which all-pair shortest path distances of G can be computed using only $O(\log |V|)$-hop paths
2: $R \leftarrow$ COMPUTEEDGES(G, V_0)
3: Create a copy of G and call it G'
4: Create a copy of w and call it w'
▷ Add the computed shortcut edges to the new graph
5: **for all** $(v, u, x) \in R$ **do**
6: **if** there is no edge between v and u **then**
7: Add edge between v and u in G'
8: Set $w'(v, u) \leftarrow x$
9: **else if** $x < w'(v, u)$ **then**
10: Set $w'(v, u) \leftarrow x$
11: **return** (G', w')
12: **end function**

Algorithm 3 Differentially Private All-Pairs Shortest Path Distances for Low Tree-Width Graphs

Input: Graph G with weight function w, tree decomposition T of G with maximum bag size $p+1$ and a constant number c and privacy parameter ε

Output: An estimate of all-pairs shortest distances with low error

▷ First stage: construct the shortcut graph

$(G', w') \leftarrow$ ConstructGraph$(G, w, T, \emptyset)$

▷ Second stage: add Laplace noise to the edge weights to provide differential privacy

for all $e \in E(G')$ **do**

  $X_e \sim \mathrm{Lap}(c \cdot (p+1)^2 \log(|V|)/\varepsilon)$

  $w'(e) \leftarrow w'(e) + X_e$

▷ Third stage: use post-processing to return all-pairs shortest path distances

return All-pairs shortest at most $c \cdot \log|V|$-hop distances for G' with weight function w'

For all $\gamma \in (0,1)$, from the Laplace distribution, we know that for all $e \in E(G')$, $\Pr(|X_e| \geq \log(|E(G')|/\gamma) \cdot c \cdot p^2 \cdot \log^2(|V|)/\varepsilon) = \gamma/|E(G')|$, and we also know that $|E(G')| \leq |V|^2$. So by the union bound, with probability $1-\gamma$, no X_e has magnitude greater than $2c \cdot \log(|V|/\gamma) \cdot p^2 \cdot \log^2(|V|)/\varepsilon$. In the last stage, we return at most $c\log|V|$-hop shortest distances. These paths contain at most $c\log|V|$ edges and so with probability $1-\gamma$, the magnitude of error in each released distance is at most $2c^2 \cdot log(|V|/\gamma)p^2log^3(|V|)/\varepsilon$ between the returned value and the result if all X_es are equal to zero. From lemma 5, for $c > 2/\log(1.5)$, the result when X_es are equal to zero is APSD for G with weight w. Thus, for $c > max(c_1, 2/\log(1.5))$, $\mathcal{M}(G,T,p,w,c,\varepsilon)$ is an ε-DP mechanism and with probability $1-\gamma$, has an error less than $2c^2log(|V|/\gamma)p^2log^3(|V|)/\varepsilon$, which is $O(log(1/\gamma)p^2log^4(|V|)/\varepsilon)$.

Note that in the proof of Theorem 1, we took the mechanism $\mathcal{M}(G,T,p,w,c,\varepsilon)$ as Algorithm 3 on the input graph G, weight function w, tree decomposition T, constant c, and privacy parameter ε, and returns its result.

A. Time Complexity

In this sub-section, we analyze the time complexity of Algorithm 3 and show that it has a polynomial-time implementation. Lemma 6 shows that the first stage of the algorithm is polynomial-time and Theorem 2 shows that our main algorithm is a polynomial-time algorithm.

Lemma 6: The time complexity of Algorithm 2 on an input graph $G = (V, E)$ is $O(|V|^3 \log|V|)$.

The proof of Lemma 6 is deferred to the appendix.

Theorem 2: The time complexity of Algorithm 3 on input graph $G = (V, E)$ is $O(|V|^3 \log|V|)$.

Proof. Lemma 6 shows that the first stage of the algorithm has $O(|V|^3 \log|V|)$ time complexity. In the second stage, we run a for loop on every edge of the intermediate graph, so its time complexity is $O(|V|^2)$. The third stage of the algorithm can be implemented using dynamic programming. Let $d(v,u,k)$ be the k-hop shortest distance on G' between v and u. We can fill it if $k=0$ or $k=1$ in $O(|V|^2)$ time. Then, for every $v,u \in V, 1 \leq k \leq c \cdot \log|V|$, $d(v,u,k)$ is equal to the minimum of $d(v,z,k-1) + w'(z,u)$ for every $z \in V$. So we can compute it in $O(|V|^3 \log|V| \cdot c)$ time. We assume c is constant, so the running time of the algorithm is $O(|V|^3 \log|V|)$. □

VI. Conclusion

We proposed a polynomial time differentially private algorithm for the problem of all pair shortest distance problem for the class of low tree-width graphs. This class of graphs contains trees as a subclass. Thus, our founding generalizes the result of Sealfon in [9]. Despite the fact that general low tree-width graphs are fundamentally more complex than trees, we manage to achieve essentially the same additive error term as that of the work of Sealfon for trees.

On the other hand, while the algorithm of the work of [2] provides a differentially private algorithm for the all pair shortest distance problem for general graphs, when restrict to the class of low tree-width graphs (i.e tree-width of order $o(n^{1/3})$), our algorithm benefits from significantly lower additive error. i.e. $O(p^2 \operatorname{polylog}(n)/\varepsilon)$ versus $O(n^{2/3} \operatorname{polylog}(n)/\varepsilon)$.

References

[1] H. L. Bodlaender, "A tourist guide through treewidth," *Acta Cybernetica*, vol. 11, pp. 1–21, 1998.

[2] J. Y. Chen, B. Ghazi, R. Kumar, P. Manurangsi, S. Narayanan, J. Nelson, and Y. Xu, "Differentially private all-pairs shortest path distances: Improved algorithms and lower bounds," in *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2023, pp. 5040–5067.

[3] R. Diestel, *Graph Theory*, 4th ed. Springer-Verlag, 2010.

[4] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Theory of Cryptography*, S. Halevi and T. Rabin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 265–284.

[5] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Foundations and Trends in Theoretical Computer Science*, vol. 9, 01 2013.

[6] C. Fan and P. Li, "Distances release with differential privacy in tree and grid graph," in *2022 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2022, pp. 2190–2195.

[7] C. Fan, P. Li, and X. Li, "Breaking the linear error barrier in differentially private graph distance release," *arXiv preprint arXiv:2204.14247*, 2022.

[8] U. Feige, "Treewidth and graph minors," https://www.wisdom.weizmann.ac.il/ robi/teaching/2012a-AdvancedAlgorithms/Lecture9+10-Treewidth.pdf, 2012.

[9] A. Sealfon, "Shortest paths and distances with differential privacy," in *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, ser. PODS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 29–41. [Online]. Available: https://doi.org/10.1145/2902251.2902291

VII. Appendix

A. Proof of Lemma 1

Let $d(n)$ denote the maximum recursion depth of the ComputeEdges function on graph inputs with n vertices with tree-width p, we have:

$$d(n) \leq \begin{cases} 1 & n \leq 6 \cdot (p+1) \\ d(n/2+p+1)+1 & Otherwise \end{cases}$$

Thus, we claim that for $C \geq \frac{1}{\log 1.5}$, $d(n) \leq C \log n$ we show the result by induction.

For $n \leq 6(p+1)$, we have $d(n) = 1 = O(\log n)$. Otherwise, we have:

$$\begin{aligned} \mathrm{d}(n) &= 1 + d(n/2 + p + 1) \\ &\leq 1 + C(\log(n/2 + p + 1)) \\ &\leq 1 + C(\log(n/c)) && (3n/2 > p + 1 + n/2) \\ &\leq 1 - C \log 1.5 + C \log(n) \\ &\leq C \log(n) && (C \geq 1/\log 1.5) \end{aligned}$$

Hence, $d(n) = O(\log n)$ and by induction, the statement is proved.

B. Proof of Lemma 2

During the execution of the algorithm, the size of the starting vertex set increases by at most $p+1$ in each recursive step. According to Lemma 1, the maximum recursion depth is $O(\log |V|)$. Therefore, the size of the starting vertex set is bounded by $O(|V_0| + p \cdot \log |V|)$.

C. Proof of Lemma 3

During each recursive call of the ConstructGraph function (except for the base case), with a starting set of size k, $(k + p) \cdot p$ edges are added to the returned list. By Lemma 2, $k = O(|V_0| + p \cdot \log |V|)$. For each edge, we add two public vertices that are calculated without using private data and a shortest distance. Since the ℓ_1 sensitivity of each shortest distance is 1, returning all $(k + p) \cdot p$ edges has an ℓ_1 sensitivity of $O(p \cdot (|V_0| + p) + p^2 \cdot \log |V|)$, which is $O((|V_0| + p) \cdot p \cdot \log |V|)$.

In each recursive call, we call some graphs for the next level which are edge-disjoint. So at each recursion depth level, the graphs in each call are edge-disjoint from one another, so the ℓ_1 sensitivity of the result at each recursion depth remains $O((|V_0| + p) \cdot p \cdot \log |V|)$. At the last recursion depth, the number of vertices in the graph is less than $6p^2$ and we return all-pairs shortest distances, so the ℓ_1 sensitivity at this depth level is $O(p^2)$. On the other hand, by Lemma 1, there are at most $O(\log |V|)$ recursion depth levels. Therefore, the ℓ_1 sensitivity of f is $O((|V_0| + p) \cdot p \cdot \log^2 |V| + p^2)$.

D. Proof of Lemma 4

In algorithm 2, the weights of all added edges are the weights of some path in G, so $d_{G,w}(v, u) \leq d^l_{G',w'}(v, u)$ for all $v, u \in V$ and $l \in [|V|]$.

We will use induction on $|V|$ to prove we have:

$$d^t_{G',w'}(v, u) \leq d_{G,w}(v, u) \tag{1}$$

for $t = \max(2, \log_{1.5} |V|)$ and for all $v \in V_0 \cup S$ and $u \in V$. If $|V| \leq 6(p+1)$, then, $w'(v, u) = d_w(v, u)$ which concludes the result for this case.

Now assume that the claim holds for graphs of tree-width at most p and fewer vertices than n. Consider a weighted graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}^+$ and $|V| = n$. For any $v \in V_0 \cup S$ and $u \in V$, let P be $v = v_0, v_1, \ldots, v_t = u$ shortest path in G. If $u \in S$, we have $d^1_{G',w'}(v, u) = w(P)$ which makes the result trivial. Otherwise, if $u \in G \setminus S$, let $C_1, \ldots, C_l$ be the connected components of $G \setminus S$ and let $U_1, \ldots U_t$ be the subgraphs that are constructed on line 15 using the sets C_i. Also define the sets V_i, weight functions w_i, and tree decompositions T_i following the algorithm. Thus there exists an index $i \in [l]$ such that $u \in C_i$. Let graph G_i with weight function w_i be the return of ConstructGraph(U_i, w_i, T_i, V_i).

If the path P is completely contained in U_i, according to induction hypothesis and since $|V(U_i)| \leq n$, 1 holds for G_i and weight w_i. Since this graph a subgraph of G' and all edge weights of G' is less than edge weights of G_i, we can conclude the result in this case. Otherwise, if P is not in fully on U_i, then, since $C_i \subseteq U_i$, P is not completely contained in C_i. Because $\{C_1, \ldots, C_t\}$ are the connected components of $G \setminus S$, thus there exists an index $j \in [t]$ such that the vertex v_j lies in S. Without loss of generality we can assume that j is the largest index such that vertex v_j lies in S. Then, the subpath P' consisting of vertices $v_j, v_{j+1}, \ldots, v_t$ lies entirely within U_i. The induction hypothesis implies that $d^{\max(2,\log_{1.5}|V_i|)}_{G_i,w_i}(v_j, v_t) \leq w(P')$.

Because $n > 6(p+1)$, we have $|V_i \leq n/1.5$. Therefore, we have $\log_{1.5} |V_i| \leq \log_{1.5} |V| - 1$.

For the last step note that because all edge weights in G' is less than or equal to those in G_i and G_i is a subgraph of G', we have $d^{\log_{1.5}(|V|)-1}_{G',w'}(v_j, v_t) \leq w(P')$. Since $v_j \in S$, there is an edge between v_0 and v_j in G' with weight $d_{G,w}(v_0, v_j) = w(v_0, v_1, \ldots v_j)$. Thus for $l = \log_{1.5}(|V|)$, we have $d^l_{G',w'}(v_0, v_k) \leq w(v_0, v_1, \ldots v_j) + w(v_j, v_{j+1}, \ldots v_t) = w(P)$. Because $n > 6(p+1)$, thus $\log_{1.5}(|V|) > 2$, so the claim is true and thus, the lemma is true.

E. Proof of Lemma 5

We prove the lemma by induction. If $|V| \leq 6(p+1)$, the result follows trivially. Now assume that the result holds true for all graphs of tree-width at most p and less than n vertices. Let G be a graph with n vertices with weight $w : E \to \mathbb{R}^+$ and tree-width at most p and T its corresponding tree decomposition. Let $v, u \in V$ be two arbitrary vertices and let path $P = v = v_0, v_1, \ldots v_t = u$ be the shortest path between v and u in G with weight function w. If there exists some $i \in \{0, 1, \ldots, t\}$ such that $v_i \in S$, then, Lemma 4 implies that $d_{G,w}(v, v_i) = d^{\max(2,\log_{1.5}|V|)}_{G',w'}(v, v_i)$ and $d_{G,w}(v_i, u) = d^{\max(2,\log_{1.5}|V|)}_{G',w'}(v_i, u)$. Since v_i is on the shortest path between v and u, we have $d_{G,w}(v, u) = d_{G,w}(v, v_i) + d_{G,w}(v_i, u)$. We also can deduce that $d^{2\cdot\max(2,\log_{1.5}|V|)}_{G',w'}(v, u) \leq d^{\max(2,\log_{1.5}|V|)}_{G',w'}(v, v_i) + d^{\max(2,\log_{1.5}|V|)}_{G',w'}(v_i, u)$ which implies that $d^{2\cdot\max(2,\log_{1.5}|V|)}_{G',w'}(v, u) \leq d_{G,w}(v, u)$. In proof of Lemma 4, we also showed that $d^{2\cdot\max(2,\log_{1.5}|V|)}_{G',w'}(v, u) \geq d_{G,w}(v, u)$ so the result implies in this case. Otherwise assume that there is no i such that $v_i \in S$. Then, path P is completely contained on one of connected components of $G \setminus S$ which implies the result by induction hypothesis.

F. Proof of Lemma 6

The algorithm consists of two steps. At first, it runs the COMPUTEEDGES$(\cdot)$ function and then constructs the graph. The second step runs a for loop on every item of the result of the first step, so if the time complexity of the first stage be $T(|V|)$, the time complexity of the second stage is $O(T(|V|)+ |V|^2)$, $O(|V|^2)$ for initializing w' and $O(T(|V|)$ for iterating R. Now, we prove that the $T(|V|) = O(|V^3| \log |V|)$. The COMPUTEEDGES function is a recursive function. Lemma 1 shows that the depth of it is $O(\log |V|)$. The recursion call of each instance of the function is based on connected components created when removing a bag from input, thus, in each depth-level of recursion call, at most $|V|$ instances of the function can be executed simultaneously. If $|V| \leq 6(p+1)$, the running time of function is $O(|V|^2)$. Otherwise, computing S and constructing H_is, V_i's and T_is also has $O(|V|^2)$ time complexity. Thus, the time complexity of each instance of COMPUTEEDGES without recursion calls is $O(|V|^2)$.

Thus, the time complexity of COMPUTEEDGES on input graph $G = (V, E)$ is $O(|V|^2 \cdot |V| \cdot \log |V|) = O(|V|^3 \log |V|)$ and the lemma is proved.