All-Pair Shortest Distance Algorithms Benchmark

CS 323: Course Project

James Bui & Long Pham

December 2025

1 Introduction

Graphs are increasingly used to represent complex systems in modern computing. From social networks to transportation networks, they help us model relationships between individuals, locations, and other entities. A fundamental task on these graphs is computing shortest paths (e.g., the shortest route between two cities). However, as graph datasets grow in size and contain more personal information, they also pose privacy risks. For example, an attacker with access to a social network graph could infer the strength of a relationship between two individuals by combining structural information with publicly available data. If a person is highly connected to people whose school is Denison, it can be inferred that the person goes to Denison.

To mitigate such risks, privacy-enhancing techniques such as secure multiparty computation (SMPC) and differential privacy (DP) have been developed to limit what an adversary can learn from the data or the outputs of graph algorithms. Because graph data is increasingly large and sensitive, developing efficient privacy-preserving algorithms for tasks such as shortest-path computation is essential.

2 Problem statement

Our project benchmarks and analyzes two approaches to differentially private all-pairs shortest path (APSP) algorithms for both directed and undirected graphs. First, we implement a classic APSP algorithm such as Floyd–Warshall and apply differential privacy by injecting noise at different stages of the computation. Second, we implement a differentially private APSP algorithm proposed in prior work by Sealfon (2016) and Chen et al. (2022). This allows us to compare these approaches in terms of runtime and utility under varying numbers of nodes and privacy budgets. Through these benchmarks, we aim to derive insights into the trade-offs between classical-algorithm-with-noise and purpose-built DP APSP algorithms.

3 Methodology

We generate data that models the road networks, where the vertices are the intersections and the edges are the roads. The weight of each edge represents the number of cars on the roads and thus is positive. The benefits of data generation are threefold. First, real data is prevalent in navigation tools such as Google Maps, and protecting user privacy is important as the number of users grows; however, it is often noisy and contains many nodes, which requires a huge amount of computation. In our experiments, we generated graphs using a small but reasonable number of vertices (i.e. $n = 10, 25, 20, 25, 30$)and the results are still comparable to other DP methods we tested. Second, modeling road networks allows us to define sensitivity in a natural way: we define two neighboring graphs as those that differ in one value of one edge weight. In a Network Traffic graph, the weight of an edge is just the number of cars on a road, which means the sensitivity of our query can be fixed to 1, as we only need the count of cars on a road. Third, this type of graph allows us to ignore negative edge weights, since, for example, there cannot be a negative number of cars on a street. One thing to note is that for Chen et al. (2022)'s algorithm, when we add noises to the weights, negative edge weights course appear. but this doesn't affect the algorithm's overall performance.

To measure error, we take the difference between the output matrix (a 2D matrix storing the shortest distances between each pair of nodes) of the baseline (which can be calculated using a non-private classic APSP algorithm) with no privacy protection and the one with differential privacy noise. Additionally, we will measure the difference between the path (with noise) outputted by the algorithm and the same path but with no noise. Since the gap between the length of these two paths can be quite large, comparing them will give us a more complete view of the algorithm's utility, rather than only looking at the average over the whole matrix.

4 Results

For each of the two parameters n (number of graph vertices) and ϵ (privacy budget), we will compare the mean error, the max error, and the runtime between Floyd Warshall with Input Perturbation (DP Input), Floyd Warshall with Output Perturbation (DP Output), both of which are differentially private version of Floyd-Warshall's algorithm, and Chen et al.'s algorithm.

4.1 Privacy Budget (ϵ)

We present results with varying privacy budget from 0.1 to 1 in Figure 1.

The mean and max errors of DP Input and DP Output follow the same trend, where they stay relatively low until a privacy budget around 0.5, and then blows up as the privacy requirement gets tighter.

The algorithm of Chen et al. on the other hand saw a big error right from the start. However, as claimed by the authors, errors grow in a sublinear fashion, which makes it a much better choice for small values of ϵ. Around $\epsilon = 0.16$, DP input's error grows quickly, outpacing Chen et al.'s algorithm, while the latter grows much slower. Output DP does outperform both of the other two in all values of ϵ tested, this is because DP Output only adds noise once all distances are calculated. This might not be effective in real life scenarios because it would require a single computer to takes in every node, compute shortest distances, and release them. If the application calls for splitting the graph and sending back and forth edges connecting graph partitions, DP Output fails to guarantee privacy. On the other hand both Chen et al.'s and DP Input perturbs the edges themselves, which guarantees privacy even in distributed scenarios.

4.2 Number of graph vertices (n)

The mean and max error in Figure 3 and 2 indicate that both DP Input and DP Output's errors rapidly increase as n increases. The output perturbation method consistently achieves the lowest errors, while the one perturbing the input remains slightly higher but follows a similar decreasing trend. In contrast, the Chen et al. (2022) approach shows much larger errors across all n values. This is because Chen et al.'s algorithm adds bigger noise than DP Input from the outset. While DP Input's sensitivity is 1, Chen et al.'s sensitivity is in order of parameters K and L, scaling with n itself.

Another reason for the high error observed for the Chen et al. (2022) algorithm is a consequence of the small scale of our experiment, where the graph size is only ($n = 16$). Their method is designed for large-scale graphs and provides accuracy guarantees that improve as (n) grows. At small (n), the sublinear sampling and sketching steps cannot capture enough structural information about the graph, leading to higher approximation error. With larger graph sizes, we would expect the error of Chen et al.'s approach to decrease significantly and better align with the theoretical performance described in their work. However, increasing (n) also leads to substantially higher runtime and greater computational resource requirements for Chen et al.'s method, which becomes a practical limitation when scaling up as shown in Figure 4.

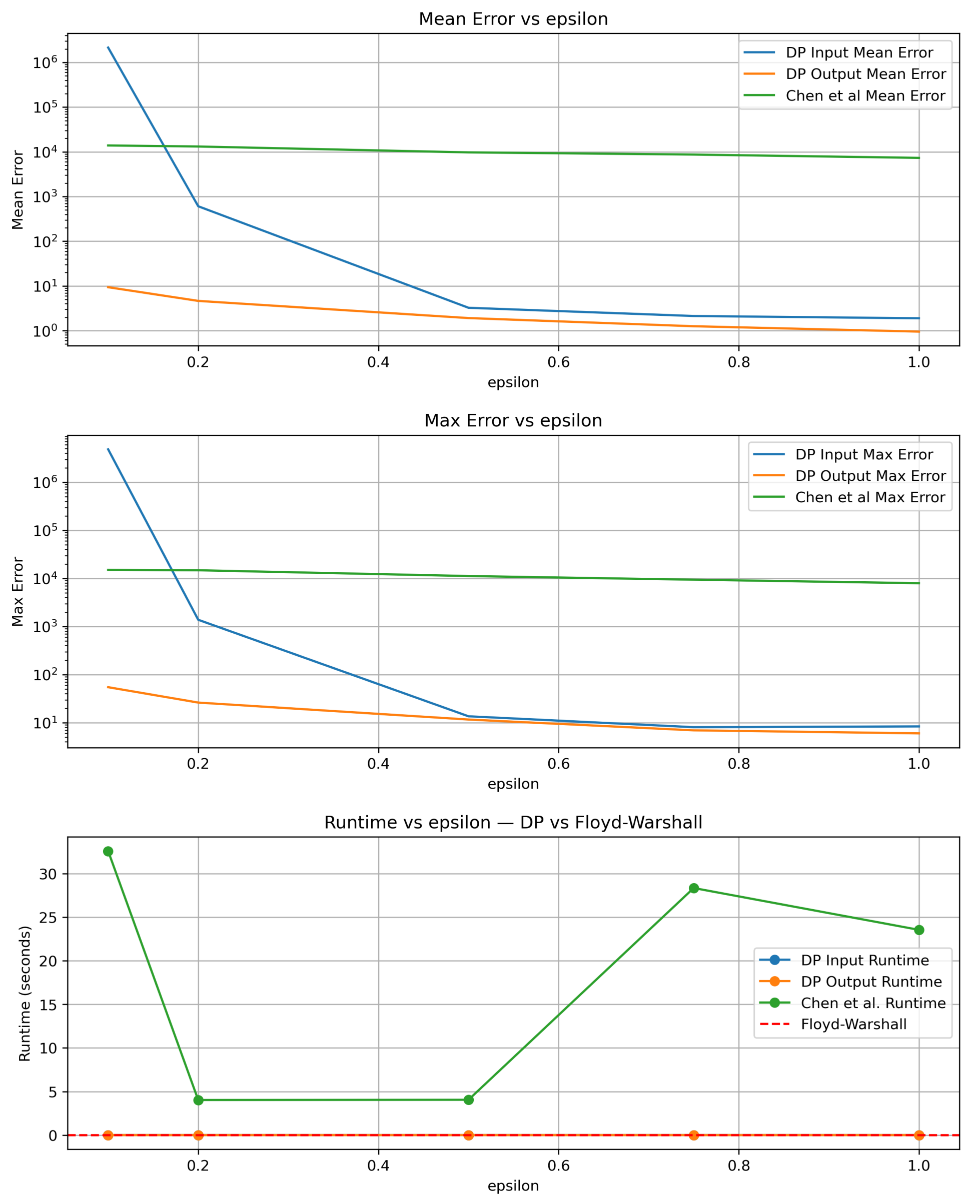

Figure 1: Errors and Runtime vs Epsilon for $n = 16$

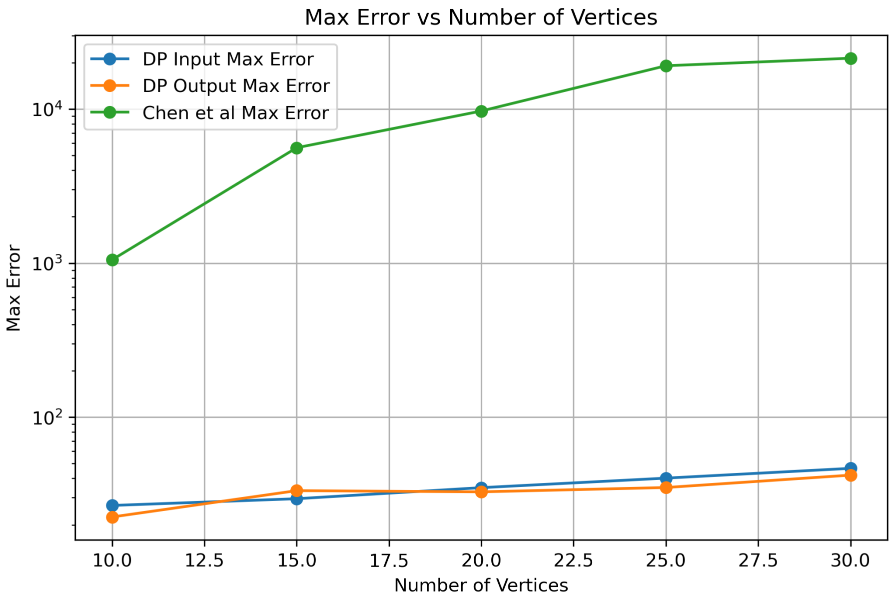

Figure 2: Max Errors vs Number of vertices

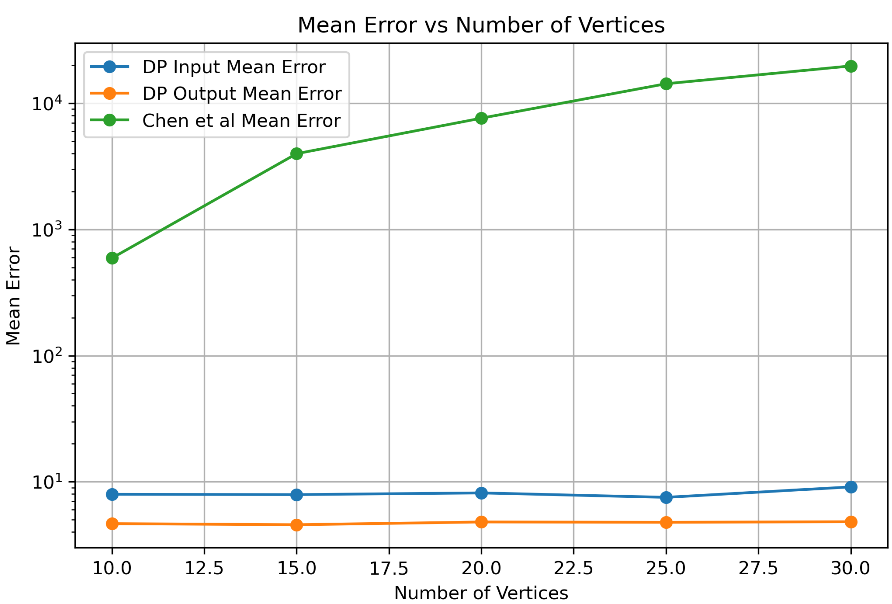

Figure 3: Mean Errors vs Number of vertices

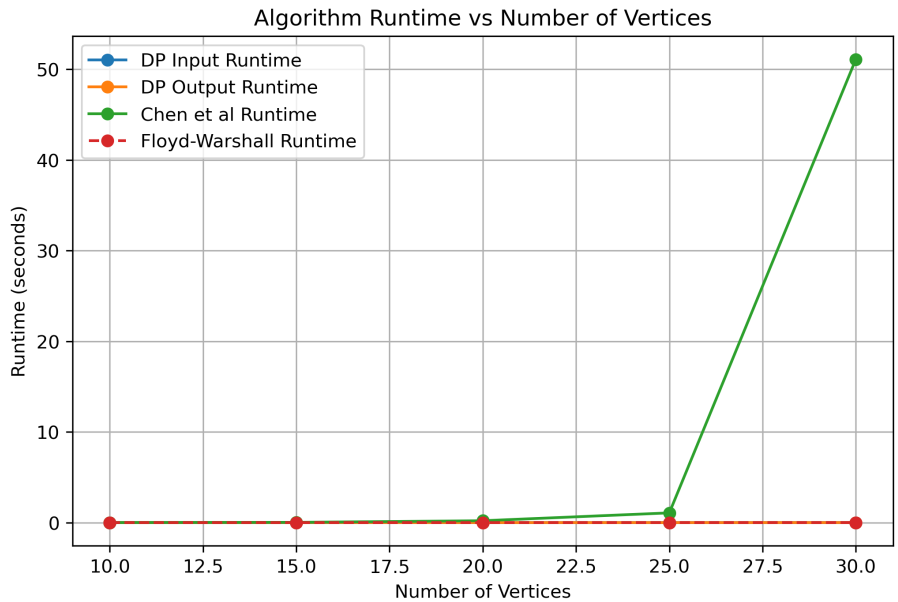

Figure 4: Runtime vs Number of vertices

The runtime comparison shows that both DP Input and DP Output run nearly identically to the standard Floyd–Warshall algorithm. Their perturbations introduce almost no additional computational overhead, making them efficient while maintaining accuracy. Chen et al. (2022) exhibits substantially higher runtime, reflecting the heavier computation required by its sublinear approximation strategy. This result highlights the practical benefit of DP-based perturbation methods, which combine low error with low runtime.

4.3 Limitations

We acknowledge that our experiments operate on a small scale, both in terms of the number of nodes and the relatively small edge weights used in the graphs. These conditions limit the amount of structural information available for the Chen et al. algorithm to exploit, which contributes to its higher observed error. In larger graphs with higher and more varied weights, the algorithm's sampling and sketching procedures would have more meaningful signal to work with, potentially reducing approximation error and aligning more closely with the guarantees reported in the original paper. Exploring this larger-scale regime—using more nodes, richer weight distributions, and heavier graph structure—remains an important direction for future testing.

In Chen et al.'s algorithm, a key parameter (K) controls the sensitivity of the sublinear approximation procedure, influencing how aggressively the method samples and sketches the graph. In the original paper, this parameter is set to a very large value (with a factor of 100) to ensure strong theoretical guarantees on massive graphs. However, such a setting is not practical for our small-scale experiments with $(n = 16)$, as it would require excessive computation without providing meaningful accuracy benefits. To make the algorithm workable in this setting, we scale down this parameter so that the sampling intensity matches the size of our graphs. This adjustment preserves the spirit of the algorithm while preventing unnecessary computational overhead and allowing the method to run within reasonable time on small inputs.

Another parameter that is worth mentioning is L - the size of the hitting set sampled from the set of vertices V, which is used to construct the "blue" edges in the algorithm. L is set to $100 \log n (A\epsilon n)^{\frac{\sqrt{17}-3}{4}}$, where A is the bound on the graph's edge weight, n is the number of vertices. The interplay between this variable means that bigger graphs should have bigger edge weight bound to ensure $L < n$. This complicates our testing because A needs to be changed in order to make the algorithm works at all (if

$L > n$).

Finally, the runtime of Chen et al.'s algorithm can be significantly improved, should a better way of enumerating paths with constraints on the number of edges of different types is found. This is a step towards the returning step of the algorithm, however since the algorithm is recursive and runs $O(logn)$ times (K times), the already factorial path-enumerating step compounds to even higher complexities.

5 Conclusion

This project provides a preliminary analysis of different differentially private algorithms for the All-Pairs Shortest Distance problem. We compare the differentially private Floyd-Warshall variants with input and output perturbation against the sublinear algorithm from Chen et al. (2022). By varying both the privacy budget (ϵ) and the number of vertices, we observe clear trade-offs between accuracy, runtime, and scalability.

Across all tested settings, DP Output achieves the lowest error, benefiting from applying noise only after all distances are computed. DP Input performs similarly, with slightly higher error but still substantially better than Chen et al.'s method under small graph sizes. Chen et al.'s algorithm, while theoretically strong for large graphs, exhibits higher error and runtime in our experiments due to the small scale of the datasets and the inherent overhead of its sampling and sketching components. Its performance is expected to improve for much larger graphs with larger and more varied edge weights, though this comes with increased computational cost.

Overall, our results suggest that perturbation-based adaptations of classical algorithms such as Floyd-Warshall can offer practical and efficient privacy guarantees for small and medium-sized graphs. In contrast, Chen et al.'s algorithm is more suitable for settings involving massive graphs where its sublinear guarantees become meaningful. Extending this work to larger graph sizes, richer weight distributions, and more optimized implementations of Chen et al.'s algorithm is a natural direction for future research and would provide a more complete picture of the strengths and limitations of these privacy-preserving APSP methods.