Approximating Probabilistic Group Steiner Trees in Graphs: Supplemental Materials

This supplement is at https://github.com/YahuiSun/PGST/blob/main/Supplement.pdf. The road map of this supplement is as follows. In Sections S1 and S2, we show the details of the time complexities of GRE-TREE and GRE-PATH, respectively. In Section S3, we discuss that the efficient dynamic programming methods for the classical group Steiner tree problem do not suit finding optimal solutions to the probabilistic group Steiner tree problem. In Section S4, we conduct additional experiment results where edge costs are defined as pairwise Jaccard distances.

S1. THE TIME COMPLEXITY OF GRE-TREE

GRE-TREE has a time complexity of

$$O\Big(\xi \cdot |g_{min}| \cdot \Big(3^{|\Gamma|}|V| + 2^{|\Gamma|}|V| \cdot (|\Gamma||V| + |E| + |V|\log|V|)\Big)\Big).$$

The details are as follows. The algorithm initializes Θ_2 (Line 1) in $O(1)$ time. Then, it finds g_{min} (Line 2) in $O(|\Gamma|)$ time, and processes each vertex $v \in g_{min}$ as follows (Lines 3-12). First, it initialize $G' = \{v\}$ (Line 4) in $O(1)$ time. It iteratively concatenates trees into G' through a loop (Lines 5-9). To efficiently check whether G' satisfactorily covers every vertex group or not (Line 5), we record the probability that G' does not cover each vertex group, and update these probabilities whenever a new vertex is added into G' in Line 8. By doing this, the cost of checking whether G' satisfactorily covers every vertex group or not (Line 5) throughout the loop of Lines 5-9 is $O(|\Gamma||V|)$. We also record Γ' in Line 6, and update Γ' whenever a new vertex is added into G' in Line 8. By doing this, the cost of constructing Γ' in Line 6 throughout the loop of Lines 5-9 is also $O(|\Gamma||V|)$. There are $O(\xi)$ iterations in the above loop. In each iteration, it uses PrunedDP++ to produce Θ' (Line 7) at a cost of

$$O\Big(3^{|\Gamma|}|V| + 2^{|\Gamma|}|V|(|\Gamma||V| + |E| + |V|\log|V|)\Big).$$

Then, it merges Θ' into G' (Line 8) in $O(|V|)$ time. After the above loop, it finds an MST as Θ_v (Line 10) in $O(|E| + |V|\log|V|)$ time, and uses Θ_v to update Θ_2 (Line 11) at a cost of $O(|V|)$. After enumerating every $v \in g_{min}$, it returns Θ_2 (Line 13) at a cost of $O(|V|)$.

S2. THE TIME COMPLEXITY OF GRE-PATH

GRE-PATH has a time complexity of

$$O\Big(|g_{min}| \cdot \Big(L|\Gamma||V| + \sum_{g \in \Gamma} \xi_g \cdot \log|V|\Big) + |E| + |V|\log|V|\Big),$$

where L is the average number of labels associated with each vertex in the input hub labels (details in [1–3]). The details are as follows. First, the algorithm initializes an empty tree in $O(1)$ time (Line 1). Then, it finds g_{min} at a cost of $O(|\Gamma|)$ (Line 2), and processes each vertex $v \in g_{min}$ as follows (Lines 3-18). It initializes and popularizes $|\Gamma|$ priority queues (Lines 4-9) in $O(L|\Gamma||V|)$ time, since the time complexity of check the cost of $P(v,u)$ in Line 7 is $O(L)$. Subsequently, it initialize Θ_v in $O(1)$ time (Line 10). To efficiently check the probability that Θ_v satisfactorily covers each vertex group, we record the probability that Θ_v does not satisfactorily cover each vertex group, and update these probabilities whenever a new vertex is added into Θ_v in Line 14. There are $O(|V|)$ vertices added into Θ_v. Whenever a new vertex is added into Θ_v, we update the recorded probabilities in $O(|\Gamma|)$ time. Thus, the cost of updating the recorded probabilities throughout the loop of Lines 11-16 is $O(|\Gamma||V|)$. The cost of checking the condition in Line 12 using the recorded probabilities is $O(1)$. There are $O(\sum_{g \in \Gamma} \xi_g)$ while iterations (Lines 13-14). In each iteration, it pops out the top element of Q_g (Line 13) in $O(\log|V|)$ time. The cost of merging paths into Θ_v (Line 14) throughout the loop of Lines 11-16 is $O(L|V|)$. After the loop, it updates Θ_3 using Θ_v (Line 17) in $O(|V|)$ time. In the end, it updates Θ_3 to be the MST that spans the vertices in Θ_3 (Line 19) in $O(|E| + |V|\log|V|)$ time.

S3. SOME DISCUSSION ON THE DYNAMIC PROGRAMMING APPROACH TO FINDING GROUP STEINER TREES

The experiments in the main content indicates that two dynamic programming algorithms, DPBF in [4] and PrunedDP++ in [5], can solve the classical group Steiner tree problem to optimality within reasonable amounts of time. With this in mind, one may wonder whether we can extend these dynamic programming algorithms to solve the probabilistic group Steiner tree problem to optimality efficiently. Here, we show that, since the probabilistic group Steiner tree problem is NP-hard even when $|\Gamma| = 1$, we are unable to do this. The details are as follows.

We describe the dynamic programming model behind DPBF and PrunedDP++ for solving the classical group Steiner tree problem as follows. Let $\mathbf{p}$ be a set of vertex groups. Let $T(v, \mathbf{p})$ be a minimum-cost tree that roots at vertex v and contains at least one vertex in each group in $\mathbf{p}$. The dynamic programming model behind DPBF and PrunedDP++ is as follows (details in [4]).

$$T(v, \mathbf{p}) = \min(T_g(v, \mathbf{p}), T_m(v, \mathbf{p})), \tag{S1}$$

where

$$T_g(v, \mathbf{p}) = \min_{u \in N(v)} \{(v, u) \cup T(u, \mathbf{p})\}, \tag{S2}$$

$$T_m(v, \mathbf{p}_1 \cup \mathbf{p}_2) = \min_{\mathbf{p}_1 \cap \mathbf{p}_2 = \emptyset} \{T(v, \mathbf{p}_1) \cup T(v, \mathbf{p}_2)\}, \tag{S3}$$

and $N(v)$ is the set of adjacent vertices of v. In Equation (S1), $T_g(v, \mathbf{p})$ is a tree generated via a grow process, while $T_m(v, \mathbf{p})$ is a tree generated via a merge process. Equation (S2) describes the grow process: $T_g(v, \mathbf{p})$ is generated by combining edge (v, u) with $T(u, \mathbf{p})$ for such $u \in N(v)$ that the cost of the combined tree is minimal. Equation (S3) describes the merge process: $T_m(v, \mathbf{p}_1 \cup \mathbf{p}_2)$ is generated by combining $T(v, \mathbf{p}_1)$ and $T(v, \mathbf{p}_2)$ for such $\mathbf{p}_1 \cap \mathbf{p}_2 = \emptyset$ that the cost of the combined tree is minimal. We show an example via Figure S1, where $\Gamma = \{g_1, g_2\}$, $g_1 = \{v_1, v_4\}$, $g_2 = \{v_2, v_3\}$, all edge costs are 1, and edge (v_1, v_2) is the optimal solution to the classical group Steiner tree problem. Let $\mathbf{p}_1 = \{g_1\}$, $\mathbf{p}_2 = \{g_2\}$, $\mathbf{p} = \{g_1, g_2\}$. In DPBF or PrunedDP++, for each vertex in a group, we initialize this single vertex as the minimum-cost tree that roots at itself and contains at least one vertex in the group. For example, we initialize $T(v_1, \mathbf{p}_1) = \{v_1\}$ and $T(v_2, \mathbf{p}_2) = \{v_2\}$. Then, we $T_g(v_2, \mathbf{p}_1) = (v_1, v_2) \cup T(v_1, \mathbf{p}_1) = (v_1, v_2)$. We merge $T(v_2, \mathbf{p}) = T_g(v_2, \mathbf{p}_1) \cup T_g(v_2, \mathbf{p}_2) = (v_1, v_2)$, which is the found optimal solution to the classical group Steiner tree problem.

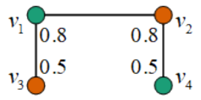
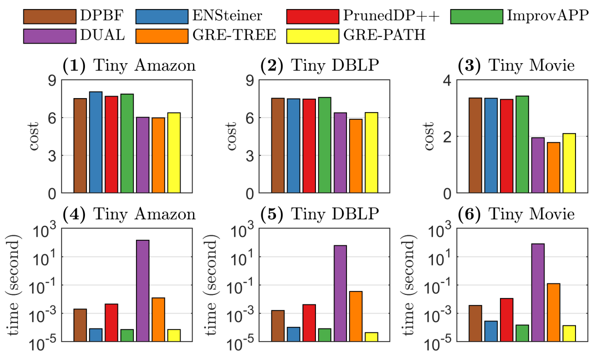


Fig. S1. An illustration of the dynamic programming approach to finding group Steiner trees.

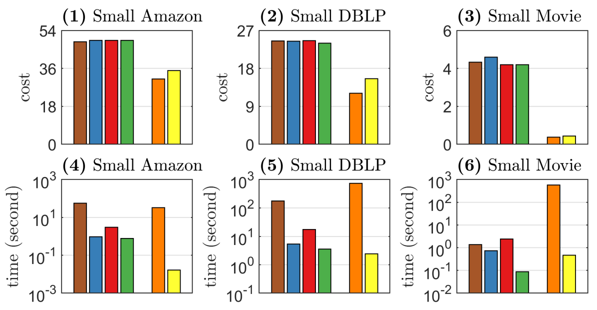
Subsequently, let us consider the probabilistic case. Let $p_{g_1}(v_1) = p_{g_2}(v_2) = 0.8$, $p_{g_2}(v_3) = p_{g_1}(v_4) = 0.5$, and $b = 0.9$. The whole input graph G in Figure S1 is the optimal solution to the probabilistic group Steiner tree problem. Intuitively, the extension of the above dynamic programming model to the probabilistic case would be to let $T(v, \mathbf{p})$ be a minimum-cost tree that roots at vertex v and satisfactorily covers every group in $\mathbf{p}$. Then, for each vertex in a group, we need to initialize the minimum-cost tree that roots at this vertex and satisfactorily covers the group. However, Theorem 1 in the main content shows that the probabilistic group Steiner tree problem is NP-hard even when $|\Gamma| = 1$, which means that it is NP-hard to initialize each of the above trees in the probabilistic case, *e.g.,* it is NP-hard to initialize $T(v_1, \mathbf{p}_1)$ and $T(v_2, \mathbf{p}_2)$ in Figure S1. In comparison, it is trivial to initialize trees in the classical case (notably, in the probabilistic case, each initialized tree may contain multiple vertices, while in the classical case, each initialized tree only contains a single vertex). As a result, it is too slow to conduct the initialization process in the probabilistic case. Therefore, due to the stricter NP-hardness of the probabilistic group Steiner tree problem, we cannot extend the above dynamic programming model to solve the probabilistic group Steiner tree problem to optimality efficiently.

S4. ADDITIONAL EXPERIMENT RESULTS

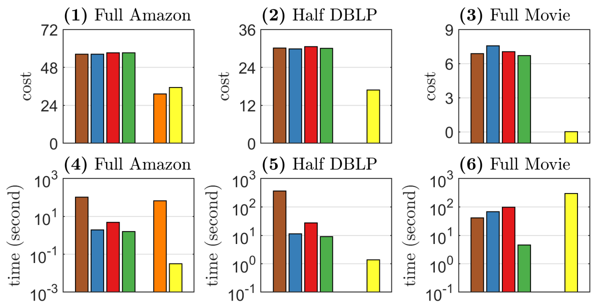
In the experiments in the main content, we set edge costs to 1. Here, we conduct additional experiments by setting edge costs to pairwise Jaccard distances (*e.g.,* [6, 7]), *i.e.,* for edge e between



(a) DUAL can only be used in tiny graphs with a few vertices.



(b) GRE-TREE does not scale well for small DBLP and Movie.



(c) Experiment results in large graphs.

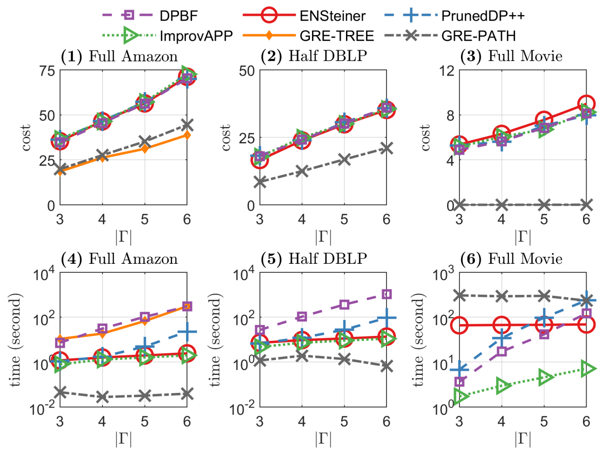
Fig. S2. Experiment results in graphs with different sizes.

vertices u and v, set the cost of e as $c(e) = 1 - \frac{|V_u \cap V_v|}{|V_u \cup V_v|}$, where V_u and V_v are the sets of vertices adjacent to u and v, respectively. We show that the experiment conclusions in the main content also hold for the following additional experiments.

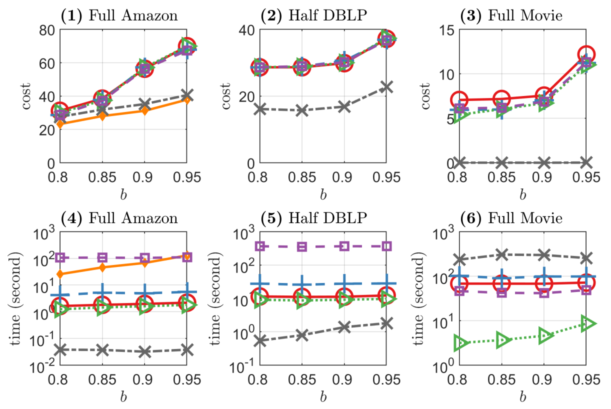
DUAL **is mainly of theoretical interests.** First, we show that DUAL can only be used in tiny graphs with a few vertices in Figure S2a, where $|V| = 30$ for "Tiny Amazon", and $|V| = 60$ for "Tiny DBLP" and "Tiny Movie". We observe that, in Figures S2a (4-6), DUAL is significantly slower than the other algorithms. These experiments show that DUAL can only be used in tiny graphs with a few vertices, and is mainly of theoretical interests. Due to this reason, we do not apply DUAL in the following experiments.

GRE-TREE **is useful when group sizes are small.** We evaluate the performance of GRE-TREE in Figure S2b, where $|V| = 188,552$ for "Small Amazon", $|V| = 448,891$ for "Small DBLP", and $|V| = 2,423$ for "Small Movie". We observe that GRE-TREE does not scale well for DBLP and Movie. Specifically, GRE-TREE consumes nearly a thousand seconds in the small DBLP and Movie graphs. Due to the slowness of GRE-TREE for DBLP and Movie, we only implement GRE-TREE for Amazon, but not for DBLP and Movie, in the following experiments.

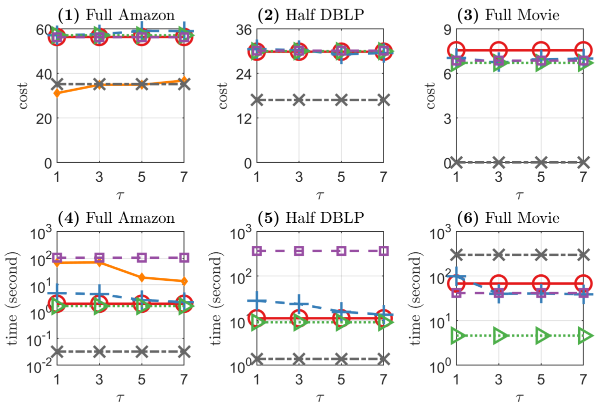
In the following experiments in Figures S3 and S4, we apply the full Amazon and Movie graphs, but only apply the half DBLP graph, *i.e.*, we set $|V| = 1,248,891$ for DBLP. The reason why we do not apply the full DBLP graph is as follows. The proposed GRE-PATH inputs hub labels (*e.g.*, [1–3]) for all pairs of shortest paths in G. The pruned landmark labeling method developed by Akiba *et al.* [2] is a state-of-the-art hub labeling method. We use this method to produce hub labels



(a) Variation of the number of vertex groups: $|\Gamma|$.



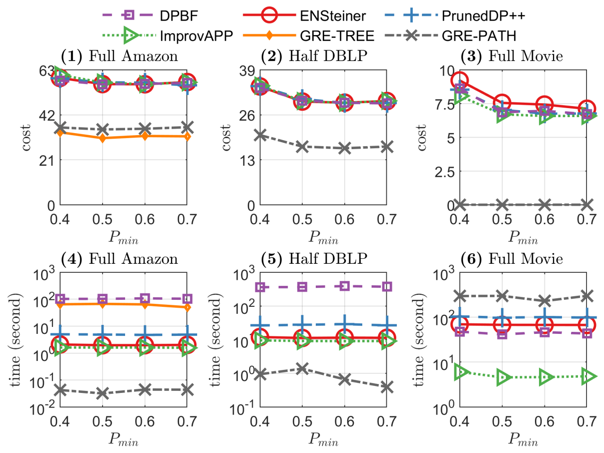
(b) Variation of the threshold value: b.



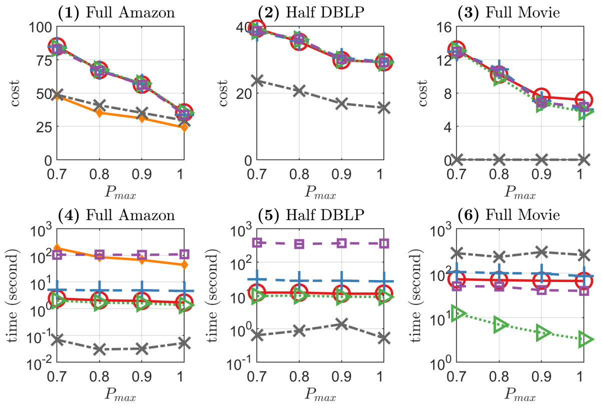
(c) Variation of the approximation parameter: τ.

Fig. S3. Experiment results of varying $|\Gamma|$, b, and τ.

of shortest paths. When all edge costs are equal, like the experiments in the main content, this method produces hub labels by implementing breadth first searches $O(|V|)$ times, which induces a time complexity of $O(|V|(|V|+|E|))$. When edge costs are not equal, like the experiments in this supplement, this method produces hub labels by implementing Dijkstra's algorithm $O(|V|)$ times, which induces a time complexity of $O(|V|(|V|\log|V|+|E|))$. Thus, it is slower the produce hub labels when edge costs are not equal. In particular, it takes 17,465 seconds to produce hub labels for the full DBLP graph in the main content, where all edge costs are equal, while it takes 106,078 seconds to produce hub labels for the half DBLP graph in this supplement, where edge costs are not equal. It is too slow to produce hub labels for the full DBLP graph in this supplement. Due to this reason, we apply the half DBLP graph in the following experiments in Figures S3 and S4.



(a) Variation of the minimum positive probability value: P_{min}.



(b) Variation of the maximum positive probability value: P_{max}.

Fig. S4. Experiment results of varying P_{min} and P_{max}.

Experiment results in large graphs. We evaluate the solution quality and speed of algorithms using the full Amazon and Movie datasets and a half DBLP dataset in Figure S2c. We observe that, in Figures S2c (1-3), the solution costs of GRE-TREE and GRE-PATH are significantly lower than those of the baseline algorithms. This shows that the effectiveness of GRE-TREE and GRE-PATH for finding probabilistic group Steiner trees. Like the experiments in the main content, GRE-PATH has a higher efficiency than the baseline algorithms for Amazon and DBLP, but a lower efficiency than the baseline algorithms for Movie.

Variation of the number of vertex groups: $|\Gamma|$. We vary the number of vertex groups: $|\Gamma|$ in Figure S3a. Like the experiments in the main content, in Figures S3a (1-3), the solution costs increase with $|\Gamma|$, and the superior solution quality of the proposed GRE-TREE and GRE-PATH over the baseline algorithms holds well as $|\Gamma|$ varies. Also like the experiments in the main content, in Figures S3a (4-6), DPBF, PrunedDP++ and GRE-TREE do not scale well to $|\Gamma|$, while ENSteiner, ImprovAPP and GRE-PATH have stronger scalabilities to $|\Gamma|$.

Variation of the threshold value: b. We vary the threshold value: b in Figure S3b. Like the experiments in the main content, the solution costs generally increase with b. Also like the experiments in the main content,, in Figures S3b (4-6), the running times of baseline algorithms do not change much with b, while the running times of GRE-TREE and GRE-PATH may increases with b, *e.g.,* GRE-TREE in Figure S3b (4) and GRE-PATH in Figure S3b (5). We further observe that the running time of ImprovAPP also increases with b in Figure S3b (6). The reason is that it first finds a classical group Steiner tree without considering b, and then merging shortest paths into this tree for satisfactorily covering all vertex groups. Since the running time of finding a classical group Steiner tree is small for Movie, while the running time of the merging process increases with b, the running time of ImprovAPP also increases with b for Movie.

Variation of the approximation parameter: τ. We vary the parameter τ in GRE-TREE and PrunedDP++ in Figure S3c. Like the experiments in the main content, in Figures S3c (4-6). The running times of GRE-TREE and PrunedDP++ decrease with τ. Also like the experiments in the main content, in Figures S3c (1-3), the practical solution qualities of GRE-TREE and PrunedDP++ do not change much with τ.

Variation of the minimum and maximum positive probability values: P_{min} and P_{max}. We vary the minimum and maximum positive probability values: P_{min} and P_{max} in Figure S4. Like the experiments in the main content, in Figures S4a (1-3) and S4b (1-3), the solution costs often decreases with P_{min} and P_{max}. In Figures S4a (4-6), the running times of algorithms do not change much with P_{min}. In figures S4b (4-6), the running times of GRE-TREE and ImprovAPP may decrease with P_{max}, as these two algorithms merge fewer trees or paths as probability values increase.

REFERENCES FOR THE SUPPLEMENT

1. E. Cohen, E. Halperin, H. Kaplan, and U. Zwick, "Reachability and distance queries via 2-hop labels," SIAM J. on Comput. **32**, 1338–1355 (2003).
2. T. Akiba, Y. Iwata, and Y. Yoshida, "Fast exact shortest-path distance queries on large networks by pruned landmark labeling," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data,* (2013), pp. 349–360.
3. Y. Li, L. H. U, M. L. Yiu, and N. M. Kou, "An experimental study on hub labeling based shortest path algorithms," Proc. VLDB Endow. **11**, 445–457 (2017).
4. B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin, "Finding top-k min-cost connected trees in databases," in *IEEE International Conference on Data Engineering,* (IEEE, 2007), pp. 836–845.
5. R.-H. Li, L. Qin, J. X. Yu, and R. Mao, "Efficient and progressive group Steiner tree search," in *Proceedings of the 2016 International Conference on Management of Data,* (ACM, 2016), pp. 91–106.
6. T. Lappas, K. Liu, and E. Terzi, "Finding a team of experts in social networks," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining,* (ACM, 2009), pp. 467–476.
7. Y. Sun, X. Xiao, B. Cui, S. Halgamuge, T. Lappas, and J. Luo, "Finding group steiner trees in graphs with both vertex and edge weights," Proc. VLDB Endow. **14**, 1137–1149 (2021).