Approximating Probabilistic Group Steiner Trees in Graphs

Shuang Yang
Renmin University of China
yangshuangnh@126.com

Yahui Sun*
Renmin University of China
yahuisun@ruc.edu.cn

Jiesong Liu
Renmin University of China
liujiesong@ruc.edu.cn

Xiaokui Xiao
National University of Singapore
xkxiao@nus.edu.sg

Rong-Hua Li
Beijing Institute of Technology
rhli@bit.edu.cn

Zhewei Wei
Renmin University of China
zhewei@ruc.edu.cn

ABSTRACT

Consider an edge-weighted graph, and a number of properties of interests (PoIs). Each vertex has a probability of exhibiting each PoI. The joint probability that a set of vertices exhibits a PoI is the probability that this set contains at least one vertex that exhibits this PoI. The *probabilistic group Steiner tree* problem is to find a tree such that (i) for each PoI, the joint probability that the set of vertices in this tree exhibits this PoI is no smaller than a threshold value, *e.g.,* 0.97; and (ii) the total weight of edges in this tree is the minimum. Solving this problem is useful for mining various graphs with uncertain vertex properties, but is NP-hard. The existing work focuses on certain cases, and cannot perform this task. To meet this challenge, we propose 3 approximation algorithms for solving the above problem. Let $|\Gamma|$ be the number of PoIs, and ξ be an upper bound of the number of vertices for satisfying the threshold value of exhibiting each PoI. Algorithms 1 and 2 have tight approximation guarantees proportional to $|\Gamma|$ and ξ, and exponential time complexities with respect to ξ and $|\Gamma|$, respectively. In comparison, Algorithm 3 has a looser approximation guarantee proportional to, and a polynomial time complexity with respect to, both $|\Gamma|$ and ξ. Experiments on real and large datasets show that the proposed algorithms considerably outperform the state-of-the-art related work for finding probabilistic group Steiner trees in various cases.

PVLDB Reference Format:
Shuang Yang, Yahui Sun, Jiesong Liu, Xiaokui Xiao, Rong-Hua Li, and Zhewei Wei. Approximating Probabilistic Group Steiner Trees in Graphs. PVLDB, 16(2): 343-355, 2022.
doi:10.14778/3565816.3565834

PVLDB Artifact Availability:
Our codes and datasets are at https://github.com/rucdatascience/PGST

Shuang Yang, Yahui Sun, and Jiesong Liu are in the School of Information & Key Laboratory of Data Engineering and Knowledge Engineering of Ministry of Education. Xiaokui Xiao is in the School of Computing. Rong-Hua Li is in the School of CS & Technology. Zhewei Wei is in the Gaoling School of AI.

*Yahui Sun is the corresponding author.

1 INTRODUCTION

Background: Given an edge-weighted graph G and a number of vertex groups, the *classical group Steiner tree* problem [34] is to find a tree in G such that (i) this tree contains at least one vertex in each group; and (ii) the total weight of edges in this tree is the minimum. Despite the fact that this problem was originally studied for the design of very-large-scale integrated circuits (*e.g.*, [19, 20, 34]), a lot of recent work finds this problem useful for various data mining applications, such as information retrieval in relational databases (*e.g.*, [11, 13, 23, 25, 37]), team formation in social networks (*e.g.*, [21, 32, 37, 40]), and pathway identification in metabolic networks (*e.g.*, [16]). We describe an example as follows.

Consider a relational database graph, where vertices and edges represent tuples and foreign key references between tuples, respectively. Each tuple contains some information, including some keywords, *e.g.,* each tuple contains the information of a paper (or a movie), including the topic keywords of this paper (or the genre keywords of this movie). Each edge is associated with a weight representing the distance between two tuples. Consider a user who wants to find papers (or movies) related to an input set Γ of keywords, we can help the user find such papers (or movies) by solving the classical group Steiner tree problem for $|\Gamma|$ groups such that each group is the set of vertices associated with a specific keyword in Γ. Since the solution tree contains at least one vertex in each group, the vertices in this tree correspond to a set of tuples that collectively cover all the keywords in Γ, *i.e.,* these tuples contain the information of the found papers (or movies) related to the input keywords. By minimizing the total weight of edges in this tree, we can reduce the distances between tuples, for ensuring that these tuples correspond to a strong and concise relationship among all the input keywords (*e.g.*, [11, 13, 23, 25, 37]). For instance, in Figure 1, consider a user who wants to find paper(s) related to two keywords: {Database; Algorithm}, if the single tuple v_1 contains both keywords, then, to help this user find paper(s), we could return $\{v_1\}$ as the solution to the classical group Steiner tree problem, in which case v_1 contains the information of the found paper.

Motivation: In the above work, each vertex group corresponds to a property of interest (PoI), and it is implicitly assumed that each vertex in a group exhibits the PoI corresponding to this group with certainty. For instance, in the above work on relational databases, each vertex group corresponds to a keyword, *e.g.,* a topic keyword of papers, and it is implicitly assumed that each vertex, *i.e.,* tuple, in this group contains the information related to this keyword with certainty. This assumption only holds when we are certain on vertex properties, *e.g.,* when we are certain on the topics of papers. However, we often have uncertain or probabilistic vertex properties in practice. A reason is that current artificial intelligence techniques usually label vertices in a statistic and probabilistic way, *e.g.,* the

Proceedings of the VLDB Endowment, Vol. 16, No. 2 ISSN 2150-8097.
doi:10.14778/3565816.3565834

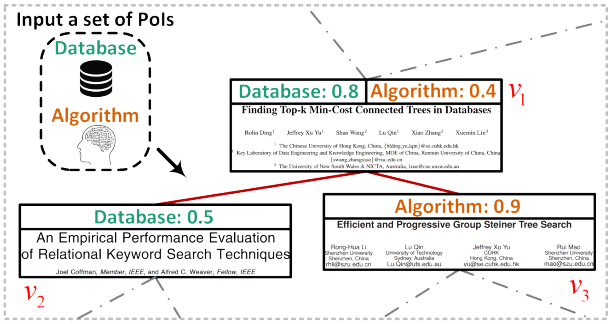


Figure 1: A tree comprising of three vertices v_1, v_2, v_3. Given two properties of interest: PoI_1 (Database) and PoI_2 (Algorithm). The three vertices have the probabilities of $0.8, 0.5, 0.0$ for exhibiting PoI_1, respectively, and the probabilities of $0.4, 0.0, 0.9$ for exhibiting PoI_2, respectively. The tree has the probabilities of $1-(1-0.8)\cdot(1-0.5)=0.9$ and $1-(1-0.4)\cdot(1-0.9)=0.94$ for exhibiting PoI_1 and PoI_2, respectively.

developers of the AMiner data platform [1] use natural language processing techniques to determine the topics (or fields of study) of papers probabilistically [2, 38]. Another reason is that we are uncertain on the qualifications of vertex properties in many cases, *e.g.,* when each tuple contains the information of a movie, including the genre keywords of this movie, we are usually uncertain on whether a movie with a specific keyword can satisfy a user who inserts this keyword, and can only guess that a high rating of the movie indicates a high probability that it satisfies the user.

In such probabilistic cases, for each group g that corresponds to a PoI, we consider that each vertex in g has a positive probability of exhibiting this PoI, while each vertex not in g has a zero probability of exhibiting this PoI. Like a lot of existing work on probabilistic data management (*e.g.,* [8, 14, 42, 43]), we consider that different probability values do not correlate with each other, which suits various cases, *e.g.,* when the topics of a paper are conjectured by extracting topic-related text features from the specific contents of this paper (*e.g.,* [9, 38]), different papers have different and independent probabilities of being in a topic, and a paper has different and independent probabilities of being in different topics.

Then, we consider the probability that a tree exhibits a PoI as the probability that there is at least one vertex in this tree that exhibits this PoI, as shown in Figure 1. Given a threshold value b, *e.g.,* $b=0.9$; and a group that corresponds to a PoI, if the probability that a tree exhibits this PoI is no smaller than b, then we say that this tree or the set of vertices in this tree *satisfactorily covers* this group, *e.g.,* in Figure 1, consider the two groups $\{v_1, v_2\}$ and $\{v_1, v_3\}$ that correspond to PoI_1 and PoI_2, respectively, if $b=0.9$, then the tree $\{(v_1, v_2), (v_1, v_3)\}$ satisfactorily covers both groups. Given a number of groups and a threshold value b, an intuitive problem is to find a minimum-weight tree that satisfactorily covers every group. Solving this problem is meaningful in various cases, *e.g.,* for finding a set of papers that covers every input topic with a high probability (*e.g.,* Figure 1), or for finding a set of movies that satisfies the user for every input genre with a high probability. We refer to this problem as the *probabilistic group Steiner tree* problem, which extends the classical group Steiner tree problem probabilistically.

The existing group Steiner tree algorithms (*e.g.,* [13, 17, 19–21, 25, 34, 37]) focus on finding classical group Steiner trees, which may not satisfactorily cover every group in probabilistic cases. A natural method of adapting these algorithms to find a feasible probabilistic group Steiner tree that satisfactorily covers every group is to (i) find a classical group Steiner tree that contains at least one vertex in each group; and (ii) for each group that this tree does not satisfactorily cover, iteratively merge shortest paths between this tree and nearby vertices in this group into this tree, until this tree satisfactorily covers this group. However, the later experiments show that this method produces solutions with unfavorably large weights, which indicates that the existing group Steiner tree algorithms cannot handle probabilistic cases.

The existing work on probabilistic databases does not address this issue. First, most existing work on probabilistic databases (*e.g.,* [8, 14, 24, 29, 30, 35, 36, 39, 42]) focuses on querying non-graph-structured data, such as querying tuples with high probabilities of containing a keyword (*e.g.,* [14, 24, 29, 35, 39]), and querying worlds, *i.e.,* combinations of tuples, with high probabilities of existence (*e.g.,* [8, 30, 36, 42]). Second, the other work that focuses on graph-structured data (*e.g.,* [31, 33, 43–45]) performs different tasks from finding group Steiner trees, such as finding maximal cliques [45], frequent subgraph patterns [33, 43], trees with high probabilities of being top-k maximum-weight trees [44], and skyline graphs with high probabilities of not being dominated by other graphs based on certain score functions [31]. As a result, new work is required for solving the probabilistic group Steiner tree problem.

Our contributions: The probabilistic group Steiner tree problem is a generalization of the classical group Steiner tree problem. Since the classical group Steiner tree problem is NP-hard [20], the probabilistic group Steiner tree problem is NP-hard as well, which indicates the usefulness of developing approximation algorithms for finding probabilistic group Steiner trees. We develop three such algorithms as follows.

- First, we develop an algorithm: DUAL, that has an approximation guarantee of $\tau\cdot\max\{1,|\Gamma|-1\}$, and a time complexity of
$$O\Big(|\Gamma|\xi^2|V|^{2\xi}\cdot\Big(3^{2\xi}|V|+2^{2\xi}|V|\cdot(2\xi|V|+|E|+|V|\log|V|)\Big)\Big),$$
where $\tau\in[1,\infty)$ is a parameter; $|V|$ and $|E|$ are the numbers of vertices and edges in G, respectively; ξ is the smallest natural number that is larger than or equal to $\log_{(1-p_{min})}(1-b)$, and p_{min} is the minimum positive probability value of vertices.
- To achieve a higher efficiency than DUAL, we develop another algorithm: GRE-TREE, that has an approximation guarantee of $\tau\cdot\xi$, and a time complexity of
$$O\Big(\xi\cdot|g_{min}|\cdot\Big(3^{|\Gamma|}|V|+2^{|\Gamma|}|V|\cdot(|\Gamma||V|+|E|+|V|\log|V|)\Big)\Big),$$
where $|g_{min}|$ is the size of the smallest vertex group.
- GRE-TREE still does not have a sufficiently high efficiency. To further push the limit of algorithmic efficiency, we develop the third algorithm: GRE-PATH, that has an approximation guarantee of $\max\{1,\sum_{g\in\Gamma}\xi_g-1\}$, and a time complexity of
$$O\Big(|g_{min}|\cdot\sum_{g\in\Gamma}\xi_g\cdot L|V|+|E|+|V|\log|V|\Big),$$

where ξ_g is the smallest natural number that is larger than or equal to $\log_{(1-p_{g!min})}(1-b)$, and $p_{g!min}$ is the minimum positive probability value of vertices for covering group $g \in \Gamma$; and L is the average number of labels associated with each vertex in pre-computed hub labels [7, 26, 27] of shortest paths in G.

We conduct experiments using real datasets, and show that (i) DUAL can only be used in tiny graphs with dozens of vertices, and thus is mainly of theoretical interests; (ii) GRE-TREE produces better solutions than the other algorithms in practice, and is efficient when group sizes are small; and (iii) GRE-PATH produces considerably better solutions than baselines built using state-of-the-art classical group Steiner tree algorithms, while scaling well to large graphs, and hence is a more favorable tool than the existing techniques for finding probabilistic group Steiner trees in various cases.

2 PROBLEM FORMULATION

We consider an undirected graph $G(V, E, c)$, where V is the set of vertices, E is the set of edges, and c is a function which maps each edge $e \in E$ to a non-negative value $c(e)$ that we refer to as edge weight. We also consider a set Γ of vertex groups. Each group $g \in \Gamma$ corresponds to a PoI, and is the set of vertices that may exhibit this PoI, and $g \subseteq V$. There is a function p that maps each pair of group $g \in \Gamma$ and vertex $v \in g$ to a value $p_g(v) \in (0, 1]$, which is the probability that v exhibits the PoI corresponding to g. For each vertex $u \notin g$, we consider $p_g(u) = 0$. As discussed in Section 1, we consider that different probability values do not correlate with each other, *e.g.,* for two different vertices v_1 and v_2, $p_g(v_1)$ and $p_g(v_2)$ do not correlate with each other. This suits various scenarios in practice. For example, consider a relational database graph where each of v_1 and v_2 represents a tuple that contains the information of a paper, and g corresponds to a topic keyword. When the topics of a paper are conjectured by analyzing the specific contents of this paper (*e.g.,* [9, 38]), different papers have different and independent probabilities of being in a topic, which means that $p_g(v_1)$ and $p_g(v_2)$ are independent from each other. In such cases, the joint probability that a set of vertices $V' \subseteq V$ exhibits the PoI corresponding to g is

$$p_g(V') = 1 - \prod_{v \in V'} [1 - p_g(v)]. \tag{1}$$

In other words, $p_g(V')$ is the probability that there is at least one vertex in V' that exhibits the PoI corresponding to g. Given a threshold value $b \in (0, 1]$ that is close to or equals 1, *e.g.,* $b = 0.97$. If $p_g(V') \geq b$, then we consider that V' *satisfactorily covers* g.

Like the related work (*e.g.,* [13, 21, 25, 37]), we consider that different groups may overlap with each other, which suits various cases where an entity may exhibit multiple PoIs, *e.g.,* a paper may cover multiple topics or cross multiple fields of study. Then, the sum of probabilities that a vertex exhibits various PoIs that correspond to different groups may be larger than 1, *e.g.,* for vertex v and groups g_1 and g_2, we may have $p_{g_1}(v) + p_{g_2}(v) > 1$.

We define the probabilistic group Steiner tree problem as the problem of finding a minimum-weight tree such that the set of vertices in this tree satisfactorily covers every group in Γ.

Problem 1 (Probabilistic Group Steiner Tree). *Given a graph $G(V, E, c)$; a set of vertex groups Γ; a probability function p; and a threshold value $b \in (0, 1]$, the probabilistic group Steiner tree problem asks for a tree $\Theta(V', E'), V' \subseteq V, E' \subseteq E$ such that (i) $p_g(V') \geq b$ for every $g \in \Gamma$, i.e., V' satisfactorily covers every vertex group; and (ii) the weight of this tree, namely,*

$$c(\Theta) = \sum_{e \in E'} c(e), \tag{2}$$

is the minimum.

Problem 1 degenerates into the classical group Steiner tree problem [34] when $p_g(v) = 1$ for every $v \in g \in \Gamma$. Since the classical group Steiner tree problem is NP-hard [20], Problem 1 is NP-hard as well. This NP-hardness indicates the usefulness of developing approximation algorithms for solving Problem 1. We develop three such algorithms in the following section.

In this paper, we assume that there is at least one feasible solution to Problem 1 in G. Moreover, we assume that G is connected. If G is not connected, then we can solve Problem 1 in G as follows: first, we obtain a solution in each maximal connected component of G that contains at least one feasible solution; then, we evaluate all the obtained solutions, and return the one with the minimum weight.

3 THREE APPROXIMATION ALGORITHMS

In this section, we propose 3 approximation algorithms for solving Problem 1, dubbed DUAL, GRE-TREE and GRE-PATH, respectively.

3.1 The DUAL Algorithm

Here, we develop the dual conquest algorithm, dubbed DUAL. "Dual conquest" refers to the fact that this algorithm iteratively and greedily merges trees that satisfactorily cover two groups.

Core idea of DUAL. We present an original definition as follows.

Definition 1 (Essential Cover). *A subset of vertices $V' \subseteq V$ is an essential cover of a vertex group $g \in \Gamma$ if V' satisfactorily covers g, i.e., $p_g(V') \geq b$, and any proper subset of V' does not satisfactorily cover g, i.e., $p_g(V'') < b$ for any $V'' \subset V'$.*

Consider a vertex group $g_x \in \Gamma$. Let Φ_{g_x} be the set of all essential covers of g_x. Suppose that an optimal solution tree contains an essential cover of g_x: $V' \in \Phi_{g_x}$. Subsequently, consider another vertex group $g_y \in \Gamma \setminus g_x$. Suppose that Θ' is a minimum-weight tree that contains V' and at least one essential cover of g_y. Then, the weight of Θ' is not larger than the weight of an optimal solution tree, since any optimal solution tree contains V' and at least one essential cover of g_y. We can build a feasible solution tree by merging minimum-weight trees that satisfactorily cover two groups, like Θ'. For example, if $\Gamma = \{g_x, g_y, g_z\}$ and there is an optimal solution tree that contains $V' \in \Phi_{g_x}$, then we can build a feasible solution tree through the following two steps. First, we construct a graph by merging (i) a minimum-weight tree that contains V' and at least one essential cover of g_y; and (ii) a minimum-weight tree that contains V' and at least one essential cover of g_z. Then, we find a Minimum Spanning Tree (MST) of the merged graph as a feasible solution tree. In the above process, we merge $|\Gamma| - 1$ trees, and the weight of each merged tree is not larger than the weight of an optimal solution tree. As a result, the weight of the built feasible solution tree is not larger than $|\Gamma| - 1$ times the weight of an optimal solution tree. This idea of iteratively merging trees that satisfactorily cover two groups is the core idea of DUAL.

Algorithm 1 The DUAL algorithm

Input: a graph $G(V, E, c)$, a set of vertex groups Γ, a probability function p, a threshold value b, and a parameter $\tau \in \mathbb{R}$ that $\tau \geq 1$
Output: an approximate solution tree Θ_1

1: Initialize an empty tree $\Theta_1 = \emptyset$, and $c(\Theta_1) = \infty$
2: Find Φ_g for all $g \in \Gamma$, and Φ_{g_x} that $|\Phi_{g_x}| = \min\{|\Phi_g| \mid \forall g \in \Gamma\}$
3: **for** each $V' \in \Phi_{g_x}$ **do**
4: **if** $|\Gamma| = 1$ **then**
5: $\Theta_{V'} = PrunedDP++(G, V', \tau)$
6: **else**
7: Initialize an empty graph $G' = \emptyset$
8: **for** each $g \in \Gamma \setminus g_x$ **do**
9: Initialize an empty tree $\Theta_{ST}(V', \Phi_g) = \emptyset$, and $c(\Theta_{ST}(V', \Phi_g)) = \infty$
10: **for** each $V_j \in \Phi_g$ **do**
11: $\Theta(V', V_j) = PrunedDP++(G, V' \cup V_j, \tau)$
12: $\Theta_{ST}(V', \Phi_g) = \min\{\Theta_{ST}(V', \Phi_g), \Theta(V', V_j)\}$
13: **end for**
14: $G' = G' \cup \Theta_{ST}(V', \Phi_g)$
15: **end for**
16: $\Theta_{V'} = MST(G')$
17: **end if**
18: $\Theta_1 = \min\{\Theta_1, \Theta_{V'}\}$
19: **end for**
20: Return Θ_1

Description of DUAL**.** Algorithm 1 shows the pseudo code of DUAL. Except a graph $G(V, E, c)$, a set of vertex groups Γ, a probability function p and a threshold value b, DUAL also inputs a parameter $\tau \in \mathbb{R}$ such that $\tau \geq 1$. DUAL incorporates a state-of-the-art classical group Steiner tree algorithm: PrunedDP++ [25]. With the input of a graph, a set of vertex groups, and the parameter $\tau \geq 1$, PrunedDP++ outputs a tree such that (i) this tree contains at least one vertex in each group; and (ii) the total edge weight in this tree is no more than τ times the total edge weight in a minimum-weight tree that contains at least one vertex in each group (specifically, we let PrunedDP++ output the first found tree such that the reported approximation ratio of this tree for the classical group Steiner tree problem is no larger than τ). We can also replace PrunedDP++ with some other classical group Steiner tree algorithms, as discussed in the supplement [6].

The algorithm first initializes an empty tree Θ_1, and considers the weight of this tree as infinity (Line 1). Then, for every $g \in \Gamma$, it finds the set of all essential covers of g: Φ_g (Line 2). Subsequently, it finds Φ_{g_x} such that $|\Phi_{g_x}|$ is the minimum, and enumerates and processes every essential cover V' in Φ_{g_x} as follows (Lines 3-19).

It builds a feasible solution tree $\Theta_{V'}$ as follows. If $|\Gamma| = 1$ (Line 4), then it uses a classical group Steiner tree algorithm, PrunedDP++ [25], to approximately find a minimum-weight tree that spans V' as $\Theta_{V'}$, with an approximation guarantee of τ (Line 5). Specifically, with the input of G, V' and τ, PrunedDP++ treats each vertex in V' as a singular vertex group in the classical group Steiner tree problem, and outputs a tree $\Theta_{V'}$ such that (i) $\Theta_{V'}$ spans V'; and (ii) the total edge weight in $\Theta_{V'}$ is no more than τ times the total edge weight in a minimum-weight tree that spans V'.

If $|\Gamma| > 1$, then it builds $\Theta_{V'}$ as follows (Lines 7-16). First, it initializes an empty graph $G' = \emptyset$ (Line 7). For each group $g \in \Gamma \setminus g_x$, it conducts the following steps (Lines 9-14). It initializes an empty tree $\Theta_{ST}(V', \Phi_g)$, and considers the weight of this tree as infinity (Line 9). For each essential cover of g: $V_j \in \Phi_g$ (Line 10), it employs PrunedDP++ to approximately find a minimum-weight tree that spans $V' \cup V_j$ as $\Theta(V', V_j)$, with an approximation guarantee of τ (Line 11), *i.e.,* with the input of G, $V' \cup V_j$ and τ, PrunedDP++ treats each vertex in $V' \cup V_j$ as a singular vertex group in the classical group Steiner tree problem, and outputs a tree $\Theta(V', V_j)$ such that (i) $\Theta(V', V_j)$ spans $V' \cup V_j$; and (ii) the total edge weight in $\Theta(V', V_j)$ is no more than τ times the total edge weight in a minimum-weight tree that spans $V' \cup V_j$. If the weight of $\Theta(V', V_j)$ is smaller than that of $\Theta_{ST}(V', \Phi_g)$, it updates $\Theta_{ST}(V', \Phi_g)$ to be $\Theta(V', V_j)$ (Line 12). After enumerating every essential cover of g, it merges $\Theta_{ST}(V', \Phi_g)$ into G' (Line 14). Then, DUAL finds $\Theta_{V'}$ as a Minimum Spanning Tree (MST) that spans G' (Line 16).

The built $\Theta_{V'}$ is a feasible solution tree. If the weight of $\Theta_{V'}$ is smaller than that of Θ_1, then it updates Θ_1 to $\Theta_{V'}$ (Line 18). After processing every essential cover in Φ_{g_x}, DUAL returns Θ_1 (Line 20).

The novelty of DUAL lies in the fact that it introduces the original concept of "essential covers", and based on the fresh observation that an optimal solution to Problem 1 is a minimum-weight tree that contains at least one essential cover of each group, it achieves the following approximation guarantee by iteratively merging trees that contain essential covers of two groups.

Approximation guarantee of DUAL**.** We prove the approximation guarantee of DUAL as follows.

Theorem 1. DUAL *has a sharp approximation guarantee of*

$$\tau \cdot \max\{1, |\Gamma| - 1\}$$

for solving the probabilistic group Steiner tree problem.

Proof. Suppose that Θ_{opt} is an optimal solution. Θ_{opt} contains at least one essential cover of g_x. Suppose that Θ_{opt} contains $V' \in \Phi_{g_x}$. When $|\Gamma| = 1$, DUAL approximately find a minimum-weight tree that spans V', with an approximation guarantee of τ, in Line 5. Thus, DUAL has an approximation guarantee of τ when $|\Gamma| = 1$. We prove that DUAL has a guarantee of $\tau \cdot (|\Gamma| - 1)$ when $|\Gamma| > 1$ as follows. Let Θ' be a minimum-weight tree that contains V' and satisfactorily covers a vertex group $g \in \Gamma \setminus g_x$. We have

$$c(\Theta') \leq c(\Theta_{opt}). \tag{3}$$

Suppose that Θ' contains an essential cover of g: $V_j \in \Phi_g$. Then,

$$c(\Theta(V', V_j)) \leq \tau \cdot c(\Theta') \leq \tau \cdot c(\Theta_{opt}), \tag{4}$$

where $\Theta(V', V_j)$ is in Line 11 of DUAL. Line 12 guarantees that

$$c(\Theta_{ST}(V', \Phi_g)) \leq c(\Theta(V', V_j)) \leq \tau \cdot c(\Theta_{opt}). \tag{5}$$

Therefore, Lines 16 and 18 further guarantees that

$$\begin{aligned} c(\Theta_1) \quad &\leq c(\Theta_{V'}) \leq c(G') \leq \sum_{g \in \Gamma \setminus g_x} c(\Theta_{ST}(V', \Phi_g)) \\ &\leq \tau \cdot (|\Gamma| - 1) \cdot c(\Theta_{opt}). \end{aligned} \tag{6}$$

Thus, DUAL has an approximation guarantee of $\tau \cdot (|\Gamma| - 1)$ when $|\Gamma| > 1$. We prove the sharpness of $\tau \cdot \max\{1, |\Gamma| - 1\}$ as follows.

First, we prove that τ is sharp when $|\Gamma| = 1$. Consider Figure 2a, where τ is a natural number larger than 1, $\Gamma = \{g_1\}$, and $g_1 =$

$\{v_{g_1,0}, v_{g_1,1}, \cdots, v_{g_1,\tau}\}$. There is an edge between $v_{g_1,0}$ and each of the other vertices, with the weight of 1. Moreover, there is an edge between each pair of vertices in $v_{g_1,1}, \cdots, v_{g_1,\tau}$, with the weight of δ, which is a tiny positive value. Suppose that g_1 is an essential cover of itself. DUAL employs PrunedDP++ to connect $V' = g_1$ in Line 5. PrunedDP++ initializes $v_{g_1,0}$ as a tree rooted at itself. When PrunedDP++ processes this tree in its dynamic programming process (details in [25]), it builds a feasible solution by merging shortest paths between $v_{g_1,0}$ and each of the other vertices. The weight of this feasible solution is τ. Meanwhile, it computes the one-label lower bound in [25] as the weight of the shortest path between $v_{g_1,0}$ and any other vertex, which is 1. It computes a progressive approximation ratio by dividing the weight of the above feasible solution by the weight of the above lower bound. Since this ratio is τ, PrunedDP++ returns the above feasible solution as Θ_1. The optimal solution Θ_{opt} contains 1 edge with the weight of 1 and $\tau - 1$ edges with the weight of δ. The approximation ratio of DUAL for the above instance is

$$\lim_{\delta \to 0} \frac{c(\Theta_1)}{c(\Theta_{opt})} = \frac{\tau}{1 + \delta(\tau - 1)} = \tau. \tag{7}$$

Thus, τ is a sharp approximation guarantee of DUAL when $|\Gamma| = 1$.

Then, we prove that $\tau \cdot (|\Gamma| - 1)$ is sharp when $|\Gamma| > 1$ as follows. Consider Figure 2b, where τ is a natural number larger than 1, $\Gamma = \{g_1, \cdots, g_{|\Gamma|}\}$, $g_1 = \{v_{g_1}\}$, $g_i = \{v_{g_i,1}, \cdots, v_{g_i,\tau}\}$ for $i \in [2, |\Gamma|]$. There is an edge between v_{g_1} and each of the other vertices, with the weight of 1. Moreover, there is an edge between each pair of vertices except v_{g_1}, with the weight of δ. Suppose that g_i is an essential cover of itself for $i \in [1, |\Gamma|]$, and $g_x = g_1$ in Line 2. Thus, $V' = \{v_{g_1}\}$ in Line 3. Consider $V_j = g_i$ for $i \in [2, |\Gamma|]$ in Line 10. DUAL employs PrunedDP++ to connect $V' \cup V_j$ in Line 11. Similar to the above process of PrunedDP++ when $|\Gamma| = 1$, PrunedDP++ returns the set of edges between v_{g_1} and each of the vertices in V_j as $\Theta(V', V_j)$ in Line 11. As a result, DUAL returns the set of edges between v_{g_1} and each of all the other vertices as Θ_1. Θ_{opt} contains 1 edge with the weight of 1 and $\tau \cdot (|\Gamma| - 1) - 1$ edges with the weight of δ. The approximation ratio of DUAL is

$$\lim_{\delta \to 0} \frac{c(\Theta_1)}{c(\Theta_{opt})} = \frac{\tau \cdot (|\Gamma| - 1)}{1 + \delta \cdot (\tau \cdot (|\Gamma| - 1) - 1)} = \tau \cdot (|\Gamma| - 1). \tag{8}$$

Thus, $\tau \cdot (|\Gamma| - 1)$ is sharp when $|\Gamma| > 1$. This theorem holds. □

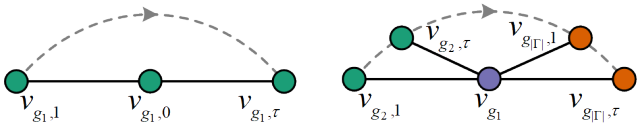
Time complexity of DUAL**.** DUAL has a time complexity of

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

where ξ is the smallest natural number that is larger than or equal to $\log_{(1-p_{min})}(1-b)$, and p_{min} is the minimum value of $p_g(v)$ for every $v \in g \in \Gamma$. The details are in the supplement [6].

3.2 The GRE-TREE Algorithm

The above DUAL has a low efficiency, as it enumerates and computes every essential cover of every group (*e.g.,* Lines 3 and 10 in DUAL). To achieve a higher efficiency, here, we develop the greedy tree concatenating algorithm, dubbed GRE-TREE. "Greedy tree concatenating" refers to the fact that GRE-TREE greedily and iteratively concatenates trees to cover groups that have not been satisfactorily covered yet. Different from DUAL, GRE-TREE does not enumerate essential covers, and has a higher efficiency.



(a) $|\Gamma| = 1$ **(b)** $|\Gamma| > 1$

Figure 2: The sharpness of $\tau \cdot \max\{1, |\Gamma| - 1\}$**.**

Core idea of GRE-TREE**.** We describe the core idea of GRE-TREE as follows. Consider a vertex group $g \in \Gamma$. An optimal solution tree contains at least one vertex in g. Suppose that an optimal solution tree roots at vertex $v \in g$. We can construct a feasible solution tree via the following two steps. First, we initialize a solution tree to be $\{v\}$. Second, we iteratively merge a tree into this solution tree such that the merged tree is a minimum-weight tree that contains v and at least one not-merged-yet vertex in each not-satisfactorily-covered-yet vertex group, until this solution tree becomes feasible, *i.e.,* until it satisfactorily covers every vertex group. Since

$$1 - (1 - p_{min})^{\xi} \geq 1 - (1 - p_{min})^{\log_{(1-p_{min})}(1-b)} \geq b, \tag{9}$$

for each vertex group $g \in \Gamma$, any set of vertices that contains no fewer than ξ vertices in g can satisfactorily cover g. As a result, at most ξ trees are merged in the above second step. Suppose that the trees that are merged in the above second step are iteratively $\Theta_1, \Theta_2, \ldots, \Theta_x$, and the optimal solution that roots at v is Θ_{opt}. For each vertex group g that is not satisfactorily covered by $\Theta_1 \cup \cdots \cup \Theta_{x-1}$, Θ_{opt} must contain at least one vertex that is in g but not in $\Theta_1 \cup \cdots \cup \Theta_{x-1}$, as otherwise Θ_{opt} cannot satisfactorily cover g. On the other hand, Θ_x is a minimum-weight tree that contains v and at least one vertex that is in g but not in $\Theta_1 \cup \cdots \cup \Theta_{x-1}$, for every vertex group g that is not satisfactorily covered by $\Theta_1 \cup \cdots \cup \Theta_{x-1}$. Thus,

$$c(\Theta_x) \leq c(\Theta_{opt}), \tag{10}$$

i.e., the weight of any merged tree is not larger than the weight of an optimal solution tree. Since at most ξ trees are merged, the weight of the constructed feasible solution tree is not larger than ξ times the weight of an optimal solution tree. This is the core idea of GRE-TREE for approximating probabilistic group Steiner trees.

Description of GRE-TREE**.** Algorithm 2 shows the pseudo code of GRE-TREE. It has the same inputs with DUAL. It initializes an empty tree $\Theta_2 = \emptyset$, and considers the weight of this tree as infinite (Line 1). Then, it finds the smallest group g_{min} in Γ (Line 2), and processes each vertex $v \in g_{min}$ as follows (Lines 3-12). First, it initializes a graph $G' = \{v\}$ (Line 4). While G' does not satisfactorily cover all groups, it initializes a set of vertex groups Γ' such that (i) Γ' contains a single-element vertex group $\{v\}$; and (ii) for each group $g \in \Gamma$ that has not been satisfactorily covered by G', Γ' also contains a vertex group that is the set of vertices that are in g but not in G' (Line 6). For example, if $g = \{v_1, v_2, v_3\}$ has not been satisfactorily covered by G' yet, and G' contains v_1, but not v_2 or v_3, then Γ' contains the vertex group $\{v_2, v_3\}$. GRE-TREE employs PrunedDP++ to produce a solution tree to the classical group Steiner tree problem in G for Γ', with an approximation guarantee of τ (Line 7; we can also replace PrunedDP++ with some other classical group Steiner tree algorithms, as discussed in the supplement [6]). In particular, with

Algorithm 2 The GRE-TREE algorithm

Input: a graph $G(V, E, c)$, a set of vertex groups Γ, a probability function p, a threshold value b, and a parameter $\tau \in \mathbb{R}$ that $\tau \geq 1$
Output: an approximate solution tree Θ_2

1: Initialize an empty tree $\Theta_2 = \emptyset$, and $c(\Theta_2) = \infty$
2: Find the smallest vertex group g_{min} in Γ
3: **for** each vertex $v \in g_{min}$ **do**
4: Initialize a graph $G' = \{v\}$
5: **while** G' does not satisfactorily cover all vertex groups **do**
6: Initialize a set of vertex groups Γ' that contains (i) a single-element vertex group $\{v\}$; and (ii) for each $g \in \Gamma$ that is not satisfactorily covered by G', a vertex group that is the set of vertices that are in g but not in G'
7: $\Theta' = PrunedDP++(G, \Gamma', \tau)$
8: $G' = G' \cup \Theta'$
9: **end while**
10: $\Theta_v = MST(G')$
11: $\Theta_2 = \min\{\Theta_2, \Theta_v\}$
12: **end for**
13: Return Θ_2

the input of G, Γ' and τ, PrunedDP++ outputs a tree Θ' such that (i) Θ' contains at least one vertex in each group in Γ'; and (ii) the total edge weight in Θ' is no more than τ times the total edge weight in a minimum-weight tree that contains at least one vertex in each group in Γ'. Notably, Θ' contains v and, for each group g that is not satisfactorily covered by G', also contains at least one vertex that is in g but not in G'. Then, it merges Θ' into G' (Line 8). After the above process, it produces Θ_v as an MST of G' (Line 10), and uses Θ_v to update Θ_2 (Line 11). After enumerating every $v \in g_{min}$, it returns Θ_2 (Line 13).

The novelty of GRE-TREE is that, based on the original observation that no more than ξ vertices in a group are needed for covering this group satisfactorily, it achieves the following approximation guarantee by iteratively merging trees that contain at least one not-merged-yet vertex in each not-satisfactorily-covered-yet group.

Approximation guarantee of GRE-TREE**.** We prove the approximation guarantee of GRE-TREE as follows.

Theorem 2. GRE-TREE *has a sharp approximation guarantee of*

$$\tau \cdot \xi$$

for solving the probabilistic group Steiner tree problem.

Proof. Let Θ_{opt} be an optimal solution. Suppose that Θ_{opt} roots at $v \in g_{min}$. Let Θ_v be the feasible solution produced by GRE-TREE in the loop for v (Lines 3-12). Line 11 guarantees that

$$c(\Theta_2) \leq c(\Theta_v). \tag{11}$$

Suppose that trees that are merged into G' in Line 8 are iteratively $\Theta'_1, \Theta'_2, \ldots, \Theta'_x$, and $\Theta'_0 = \{v\}$, *i.e,* Θ'_0 is the initialized G' in Line 4. For each $i \in [1, x]$ and each group $g \in \Gamma$ that is not satisfactorily covered by $\Theta'_0 \cup \cdots \cup \Theta'_{i-1}$, Θ_{opt} must contains v and at least one vertex in g but not in $\Theta'_0 \cup \cdots \cup \Theta'_{i-1}$, as otherwise Θ_{opt} cannot satisfactorily covers g. Since Θ'_i is a τ-approximation minimum-weight tree that contains v and at least one vertex in g but not in $\Theta'_0 \cup \cdots \cup \Theta'_{i-1}$, we have

$$c(\Theta'_i) \leq \tau \cdot c(\Theta_{opt}). \tag{12}$$

As discussed in the core idea of GRE-TREE, we have $x \leq \xi$. Thus,

$$c(\Theta_2) \leq c(\Theta_v) \leq c(G') \leq \sum_{i \in [1,x]} c(\Theta'_i) \leq \tau \cdot \xi \cdot c(\Theta_{opt}). \tag{13}$$

Thus, $\tau \cdot \xi$ is an approximation guarantee of GRE-TREE. We prove the sharpness of this guarantee as follows. Consider an instance similar to that in Figure 2b: $\Gamma = \{g_1, \cdots, g_{|\Gamma|}\}$, $g_1 = \{v_{g_1}\}$, $g_i = \{v_{g_i,1}, \cdots, v_{g_i,\xi}\}$ for $i \in [2, |\Gamma|]$. There is an edge between v_{g_1} and each of the other vertices, with the weight of 1. Moreover, there is an edge between each pair of vertices except v_{g_1}, with the weight of δ. Suppose that g_i is an essential cover of itself for $i \in [1, |\Gamma|]$. $g_{min} = g_1$ in Line 3. Also suppose that $\tau = |\Gamma| - 1$. The first produced Θ' in Line 7 is a τ-approximation solution of PrunedDP++ for covering v_0 and at least one vertex in g_i for $i \in [2, |\Gamma|]$. Like the discussion in the proof of Theorem 1, Θ' in Line 7 contains an edge between v_{g_1} and a vertex in $\{v_{g_i,1}, \cdots, v_{g_i,\xi}\}$ for each $i \in [2, |\Gamma|]$. After iteratively concatenating such trees, GRE-TREE returns Θ_2 that contains $\tau \cdot \xi$ edges with the weight of 1. The optimal solution Θ_{opt} contains 1 edge with the weight of 1 and $\tau \cdot \xi - 1$ edges with the weight of δ. The approximation ratio of GRE-TREE is

$$\lim_{\delta \to 0} \frac{c(\Theta_2)}{c(\Theta_{opt})} = \frac{\tau \cdot \xi}{1 + \delta \cdot (\tau \cdot \xi - 1)} = \tau \cdot \xi. \tag{14}$$

Thus, $\tau \cdot \xi$ is a sharp guarantee. This theorem holds. □

Time complexity of GRE-TREE**:**

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

The details are in the supplement [6].

3.3 The GRE-PATH Algorithm

The above GRE-TREE utilizes PrunedDP++ $O(\xi \cdot |g_{min}|)$ times. Since PrunedDP++ has an exponential time complexity with respect to $|\Gamma|$, GRE-TREE does not scale well to $|\Gamma|$ or $|g_{min}|$. To address this issue, here, we develop the greedy path concatenating algorithm, dubbed GRE-PATH. "Greedy path concatenating" refers to the fact that GRE-PATH greedily and iteratively concatenates paths to cover groups that have not been satisfactorily covered yet. Different from GRE-TREE, GRE-PATH does not utilize PrunedDP++ in its process.

Core idea of GRE-PATH**.** We describe the core idea of GRE-PATH as follows. First, let ξ_g be the smallest natural number that is larger than or equal to $\log_{(1-p_{g!min})}(1-b)$, where $p_{g!min}$ is the minimum value of $p_g(v)$ for every $v \in g$. Suppose that there is an optimal solution tree Θ_{opt} that roots at vertex v. We can build a tree that roots at v and satisfactorily covers a group $g_x \in \Gamma$ in the following way. First, we initialize an empty tree. Second, we iteratively merge shortest paths between v and vertices in g_x into this tree, in the increasing order of the weights of these paths, until this tree satisfactorily covers g_x. Since any set of vertices that contains ξ_{g_x} vertices in g_x can satisfactorily covers g_x, we merge at most ξ_{g_x} paths in the above process. Suppose that $P_1, \cdots, P_y$ are the merged paths sequentially, and $V_{P_1}, \cdots, V_{P_y}$ are the sets of vertices in these paths. Since $V_{P_1} \cup \cdots \cup V_{P_{y-1}}$ does not satisfactorily cover g_x, an optimal solution tree that roots at v must contain at least one vertex

Algorithm 3 The GRE-PATH algorithm

Input: a graph $G(V, E, c)$, a set of vertex groups Γ, a probability function p, a threshold value b, and hub labels for all pairs of shortest paths in G
Output: an approximate solution tree Θ_3

1: Initialize a tree $\Theta_3 = \emptyset$, and $c(\Theta_3) = \infty$
2: Find the smallest group g_{min} in Γ
3: **for** each vertex $v \in g_{min}$ **do**
4: **for** each group $g \in \Gamma$ **do**
5: Initialize a min Binary heap [41] Q_g that contains each vertex $u \in g$, with the weight of $P(v, u)$ as the priority value
6: **end for**
7: Initialize a tree $\Theta_v = \{v\}$
8: **for** each group $g \in \Gamma$ **do**
9: **while** $p_g(\Theta_v) < b$ **do**
10: Pop u out of Q_g
11: $\Theta_v = \Theta_v \cup P(v, u)$
12: **end while**
13: **end for**
14: $\Theta_3 = \min\{\Theta_3, \Theta_v\}$
15: **end for**
16: Return $\Theta_3 = MST(\Theta_3)$

in g_x but not in $V_{P_1} \cup \cdots \cup V_{P_{y-1}}$. Since we merge paths in the increasing order of the weights of paths, P_y is the shortest path between v and vertices in g_x but not in $V_{P_1} \cup \cdots \cup V_{P_{y-1}}$. As a result,

$$c(P_y) \leq c(\Theta_{opt}), \tag{15}$$

which means that the weight of each merged path is not larger than the weight of an optimal solution tree. We can build a tree that satisfactorily covers all groups by merging shortest paths in a similar way. The number of merged paths is bounded by $\sum_{g\in\Gamma} \xi_g$, and the weight of each merged path is not larger than the weight of an optimal solution tree. This is the core idea of GRE-PATH for approximating probabilistic group Steiner trees.

Description of GRE-PATH**.** Algorithm 3 shows the pseudo code of GRE-PATH. Different from DUAL and GRE-TREE, GRE-PATH does not input the parameter τ, since it does not utilize PrunedDP++. Another difference is that GRE-PATH inputs hub labels (*e.g.,* [7, 12, 26–28]) of shortest paths between all pairs of vertices in the input graph G. By using these labels, we can extract the shortest path between each pair of vertices in G within microseconds. Here, we use a parallel version of the Pruned Landmark Labeling algorithm [7] to prepare these labels, since this parallel version is simple and scales well to large edge-weighted graphs with millions of vertices and edges, and it is also fast to use labels generated by this parallel version to query shortest paths. We can also use the reduction techniques in [26] and the tree decomposition techniques in [27] to generate labels with more complex structures and smaller sizes to record shortest paths. However, since it is often less efficient to query shortest paths using these more complex labels [26, 27], we do not use these more complex labels in this paper.

GRE-PATH initializes an empty tree $\Theta_3 = \emptyset$, and considers the weight of this tree as infinite (Line 1). It finds the smallest group g_{min} in Γ (Line 2), and processes each vertex $v \in g_{min}$ as follows (Lines 3-18). For each $g \in \Gamma$ (Line 4), it initializes a min Binary heap [41] Q_g that contains each vertex $u \in g$, with the weight of $P(v, u)$ as the priority value (Line 5). Subsequently, it initializes a tree $\Theta_v = \{v\}$ (Line 7). For each group $g \in \Gamma$ (Line 8), while Θ_v does not satisfactorily cover g *i.e.,* $p_g(\Theta_v) < b$ (Line 9), it pops out the top element u of Q_g (Line 10), and merges the shortest path between v and u: $P(v, u)$ into Θ_v (Line 11). After the merging process, Θ_v becomes a feasible solution tree. If the weight of Θ_v is smaller than that of Θ_3, then GRE-PATH updates Θ_3 to Θ_v (Line 14). After enumerating every $v \in g_{min}$, GRE-PATH updates and returns Θ_3 as an MST that spans the vertices in Θ_3 (Line 16).

Notably, both GRE-PATH and GRE-TREE iteratively merge subgraphs that contain not-merged-yet vertices in not-satisfactorily-covered-yet groups. The differences between these two algorithms are as follows. GRE-TREE merges $O(\xi)$ trees, and thus has an approximation guarantee proportional to ξ, at the cost of an exponential time complexity with respect to $|\Gamma|$, due to the usage of PrunedDP++ for finding the merged trees. In comparison, GRE-PATH does not utilizes PrunedDP++ to find and merge trees, but merges $O(\sum_{g\in\Gamma} \xi_g)$ paths, and thus achieves a looser approximation guarantee than GRE-TREE, while enjoying a polynomial time complexity, which will be analyzed later. Moreover, different from GRE-TREE that finds the merged tree just before a merge operation, a novelty of GRE-PATH is that it uses hub labeling techniques to precompute all candidate paths that may be merged before the merge operations, for achieving a much higher efficiency than GRE-TREE.

Approximation guarantee of GRE-PATH**.** We show the approximation guarantee of GRE-PATH as follows, the proof of which is in the supplement [6].

Theorem 3. GRE-PATH *has a sharp approximation guarantee of*

$$\max\{1, \sum_{g\in\Gamma} \xi_g - 1\}$$

for solving the probabilistic group Steiner tree problem.

Time complexity of GRE-PATH**:**

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

where L is the average number of hub labels associated with each vertex (details in [7, 12, 28]). The details are in the supplement [6].

4 EXPERIMENTS

In this section, we conduct experiments on a computer with two Intel Xeon Gold 6342 processors and 500 GB RAM[1].

4.1 Datasets

We use three types of real datasets as follows.

Amazon. It is the Amazon product dataset in the Stanford Network Analysis Project [5]. We use it to build a graph, where each vertex represents a product, and each edge between two vertices indicates that there are users who bought these two products at the same time. Each product is associated with the average rating of this product and a set of keywords that describe this product. There are 548,552

[1] **Our codes and datasets are at** https://github.com/rucdatascience/PGST

Table 1: Dataset statistics.

Name	Vertices	Edges	Vertex Groups
Amazon	548,552	987,942	25,958
DBLP	2,497,782	15,759,646	132,337
Movie	62,423	35,323,774	19

vertices, 987,942 edges, and 25,958 keywords in total. For a user who queries $|\Gamma|$ keywords, we consider each group in Γ as the set of vertices associated with a specific queried keyword. We normalize the average ratings of products to the range of $[P_{min}, P_{max}]$, where P_{min} and P_{max} are parameters, and $0 < P_{min} < P_{max} \leq 1$. For group $g \in \Gamma$ and vertex $v \in g$, we set $p_g(v)$ as the normalized average rating of v, which represents the probability that v satisfies the user. In this case, a product has the same probability values of satisfying the user for the associated keywords, *e.g.,* for vertex v and groups g_i and g_j, we have $p_{g_i}(v) = p_{g_j}(v)$. This reflects the fact that a product often either satisfies or does not satisfy the user as an integrated entity.

DBLP. It is the DBLP citation dataset at the AMiner website [2]. We use it to build a graph, where each vertex represents a paper. There is an edge between two vertices if there is a citation relationship between the two papers, which indicates that the contents of the two papers are related to each other, *e.g.,* the fact that paper v cites paper u not only shows that the contents of u are related to v, which is why v cites u, but also shows that the contents of v are related to u, since v is a work built on u. Each paper is associated with some fields of study, and each paper-field pair is further associated with a value, *e.g.,* 0.659, that indicates the probability that this paper is in this field. The developers of the AMiner website produce such values by analyzing the contents of papers using natural language processing techniques [38]. There are 2,497,782 vertices, 15,759,646 edges, and 132,337 fields in total. For a user who queries $|\Gamma|$ fields, we consider each group in Γ as the set of vertices that are associated with a specific queried field. Like Amazon, we normalize the values of paper-field pairs to the range of $[P_{min}, P_{max}]$. For group $g \in \Gamma$ and vertex $v \in g$, we set $p_g(v)$ as the normalized value of the pair of v and g, which represents the probability that v is in field g.

Movie. It is the MovieLens dataset at the GroupLens website [3]. We use it to build a graph, where each vertex represents a movie. Each movie is associated with the average rating of this movie and the genres that this movie belongs to, *e.g.,* comedy. There is an edge between two vertices if there are users who give both movies 5 stars, which indicates that people who like one of these two movies may also like the other one. There are 62,423 vertices, 35,323,774 edges, and 19 genres in total. For a user who queries $|\Gamma|$ genres, we consider each group in Γ as the set of vertices that are associated with a specific queried genre. Like Amazon, we normalize the average ratings of movies to the range of $[P_{min}, P_{max}]$. For group $g \in \Gamma$ and vertex $v \in g$, we set $p_g(v)$ as the normalized average rating of v, which represents the probability that v satisfies the user.

4.2 Experiment Settings

Baseline algorithms. Except the proposed algorithms, we apply four state-of-the-art group Steiner tree algorithms as baselines.

- DPBF [13]: a dynamic programming algorithm that solves the classical group Steiner tree problem to optimality. It is widely used for information retrieval in databases (*e.g.,* [18, 22, 23]).

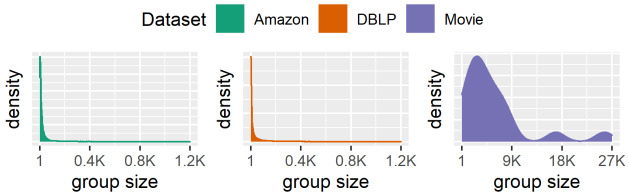


Figure 3: The sizes of candidate vertex groups.

- ENSteiner [21]: a heuristic algorithm that finds sub-optimal solutions to the classical group Steiner tree problem via a transformation from group Steiner trees to Steiner trees [15]. It is widely used for team formation in social networks (*e.g.,* [21, 32, 40]).
- PrunedDP++ [25]: a progressive algorithm that finds optimal or approximate solutions to the classical group Steiner tree problem through a dynamic programming approach. It improves DPBF on both practical efficiency and solution flexibility.
- ImprovAPP [37]: a $(|\Gamma|-1)$-approximation algorithm that solves the classical group Steiner tree problem by greedily and iteratively merging shortest paths between vertices. It improves a previous $(|\Gamma|-1)$-approximation algorithm [20] on both efficiency and practical solution quality.

Each of the above algorithms originally returns a classical group Steiner tree that contains at least one vertex in each group. We use each of these algorithms to iteratively compute $k \in \mathbb{Z}^+$ classical group Steiner trees in the following way: after computing each tree, we update the weights of edges in this tree to be large values, for preferably computing different trees in next iterations. Such computed trees may not satisfactorily cover all groups. To address this issue, for each computed tree, we iteratively merge shortest paths between this tree and nearby vertices in groups that have not been satisfactorily covered by this tree, until this tree satisfactorily covers all groups. We use hub labels of shortest paths to accelerate this process. Then, we replace the merged tree with a minimum spanning tree that spans the vertices in this tree. Notably, we only consider original edge weights in the above process of repairing trees. After that, each of the above algorithms produces k feasible solutions to Problem 1. We let each of these algorithms return the best found solution, and compare the returned solutions with those of the proposed algorithms in the following experiments.

Lower bounds. When $\tau = 1$, the first classical group Steiner tree computed by PrunedDP++ in the above process is an optimal solution to the classical group Steiner tree problem. The weight of this tree is a lower bound of the weight of an optimal solution to Problem 1. We compare this lower bound with the solution weights of the applied algorithms in the following experiments.

Edge weights. There are two major existing methods of setting edge weights for finding group Steiner trees. First, set edge weights to 1 (*e.g.,* [13, 25]). Second, set edge weights to pairwise Jaccard distances (*e.g.,* [21, 37]), *i.e.,* for edge e between vertices u and v, set the weight 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. Due to space limitation, we set edge weights to 1 in the experiments in this paper, and set edge weights to pairwise Jaccard distances in the additional experiments in the supplement [6]. The key observations in the following experiments are consistent with those in the supplement.

Parameters. We vary six parameters as follows.

- $|\Gamma|$: the number of vertex groups. For Amazon, DBLP and Movie, each candidate group is the set of vertices associated with a specific keyword, field of study, and genre, respectively. That is to say, each candidate group corresponds to a PoI. Based on the fact that some PoIs often appear together in practice, we select $|\Gamma|$ candidate groups in the following way. First, we build a graph where vertices are PoIs and two vertices are connected with each other if they appear together at least once, *i.e.,* for Amazon (*resp.* DBLP and Movie), two keywords (*resp.* fields and genres) are associated with the same product (*resp.* paper and movie) at least once. Then, we select a root PoI uniformly at random, and conduct a breadth first search from the root PoI to search nearby PoIs in the above graph, with a maximum search depth of d, which is the minimum value such that at least $|\Gamma| - 1$ nearby PoIs are searched. Subsequently, we select $|\Gamma| - 1$ nearby PoIs uniformly at random from the searched ones. The selected $|\Gamma|$ PoIs, including the root PoI, are often related and may appear together in practice, as shown by some examples in the supplement [6]. We consider the $|\Gamma|$ groups corresponding to the selected PoIs as the selected groups. There may be no feasible solution for some Γ. We regenerate Γ when such a case occurs. We visualize the sizes of candidate groups in Figure 3. For Amazon and DBLP, large groups that contain tens of thousands of vertices have negligibly low densities. As a result, for Amazon and DBLP in Figure 3, we only visualize the densities of group sizes smaller than 1.2K, for clearly showing that most groups contain small numbers of vertices.
- b: the threshold value (see Problem 1).
- τ: the parameterized approximation ratio of PrunedDP++, and $\tau \geq 1$. Since DUAL and GRE-TREE incorporate PrunedDP++, τ is also an input parameter of DUAL and GRE-TREE.
- P_{min}: the minimum positive probability value (see Section 4.1).
- P_{max}: the maximum positive probability value (see Section 4.1).
- k: the number of feasible solutions computed by each baseline algorithm, as discussed above.

Metrics. We evaluate two metrics as follows.

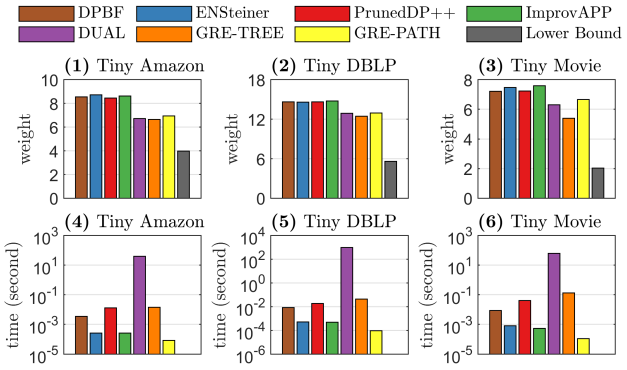
- *weight*: the weight of a solution tree, *i.e.,* $c(\Theta)$ for tree Θ.
- *time*: the running time of an algorithm (unit: second).

Due to space limitation, we show the memory consumption of algorithms in the supplement [6]. Recall that, the lower bounds of optimal solution weights are calculated in the process of PrunedDP++ when $\tau = 1$. To clearly distinguish lower bounds with algorithms, we do not show the running times for calculating lower bounds.

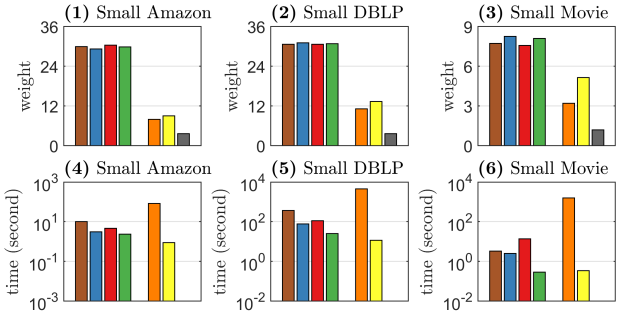
4.3 Quantitative Experiment Results

Here, we show the experiment results. The default values of parameters are: $|\Gamma| = 5$, $b = 0.9$, $\tau = 1$, $P_{min} = 0.5$, $P_{max} = 0.9$, $k = 3$. When we vary one parameter, we set the other parameters to default values. For each set of parameters, we randomly generate 300 instances, and visualize the average metric values.

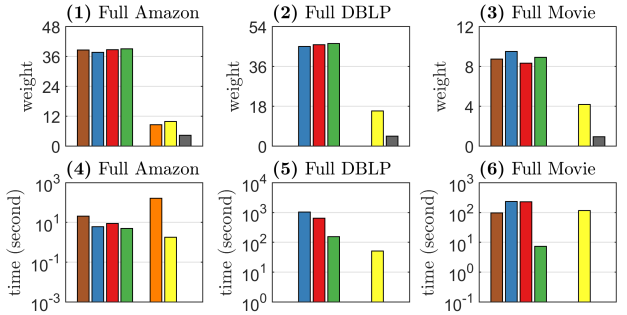
DUAL **is mainly of theoretical interests.** We show that DUAL can only be used in tiny graphs with dozens of vertices in Figure 4a, where "Tiny Amazon" refers to tiny graphs built using the Amazon dataset, *etc.* To build a tiny graph with $|V|$ vertices, we select $|V|$ vertices and the edges between these vertices from the input data in the following way. First, we randomly select a vertex v. Then, we perform a random walk starting from v, and select the first $|V|$ vertices encountered. Since the graph may not be connected, the random walk starting from v may not encounter $|V|$ vertices. In this case, we perform multiple random walks, until $|V|$ vertices are selected. In Figures 4a, $|V| = 45$ for "Tiny Amazon", $|V| = 90$ for "Tiny DBLP", and $|V| = 70$ for "Tiny Movie". In Figures 4a (1-3), DUAL finds lower-weight solutions than the baseline algorithms. By comparing the lower bounds, it can be seen that the solution weights of DUAL are roughly at most 200% of the optimal solution weights for Amazon and DBLP, and 300% for Movie. In Figures 4a (4-6), DUAL is significantly slower than the other algorithms. The reason is that DUAL frequently exploits PrunedDP++ to connect two essential covers optimally, and has a large time complexity. Thus, DUAL can only be used in tiny graphs with dozens of vertices,



(a) DUAL **can only be used in tiny graphs with dozens of vertices.**

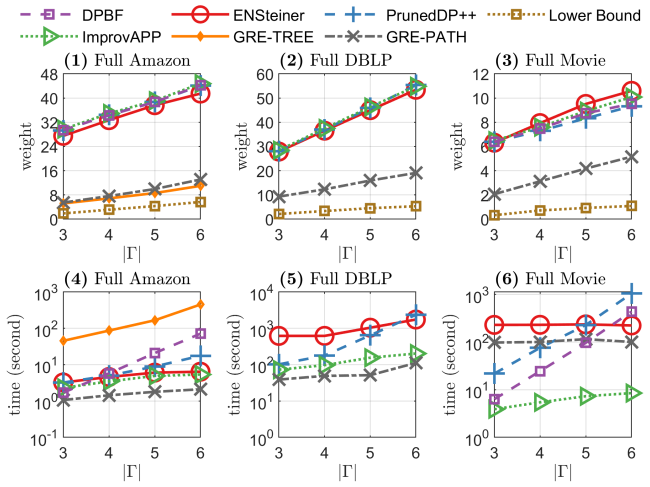


(b) GRE-TREE **does not scale well for small** DBLP **and** Movie.

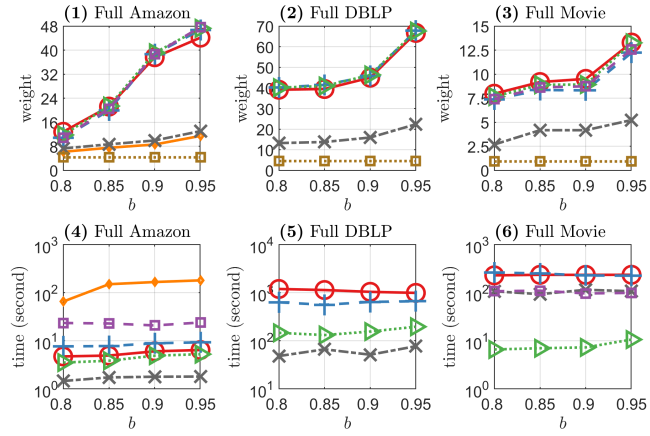


(c) Evaluation of solution quality and speed in full graphs.

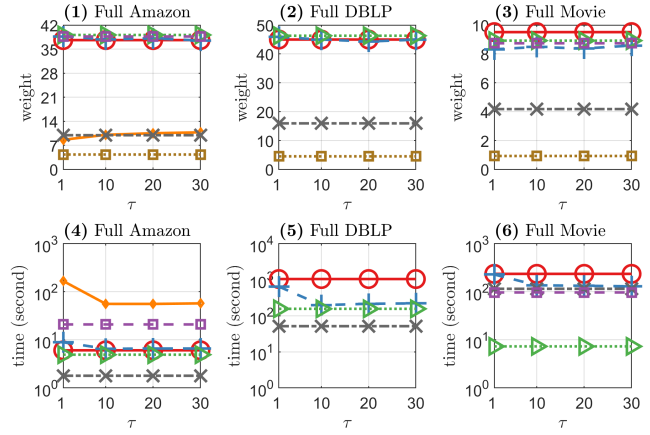
Figure 4: Experiment results in tiny, small, and full graphs.



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

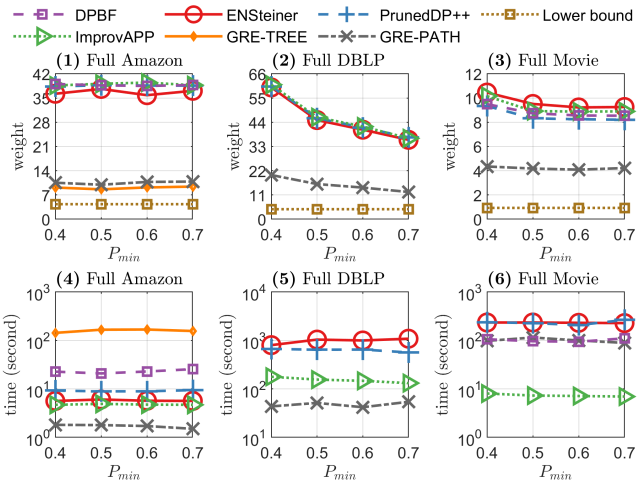


(b) Variation of the threshold value: b.

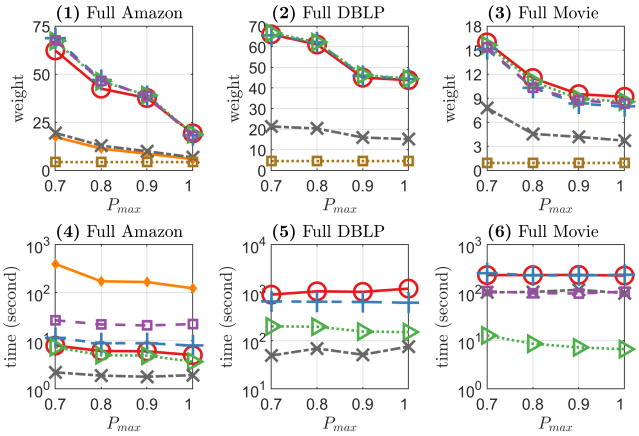


(c) Variation of the approximation parameter: τ.

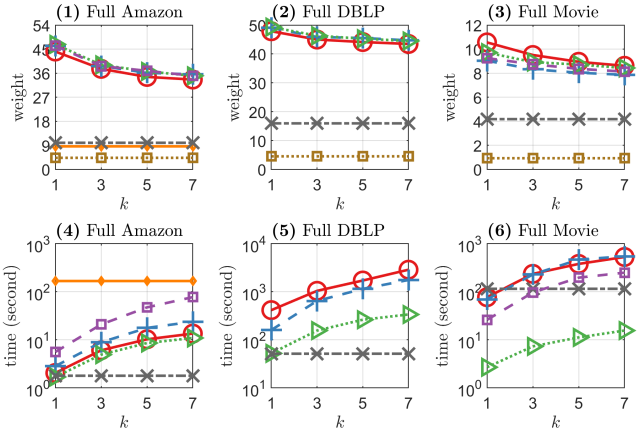
Figure 5: Experiment results of varying $|\Gamma|$, b, and τ.



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



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



(c) Variation of the number of feasible solutions in baselines: k.

Figure 6: Experiment results of varying P_{min}, P_{max} and k.

and is mainly of theoretical interests. As a result, *we do not apply* DUAL *in the following experiments in larger graphs.*

GRE-TREE **is useful when group sizes are small.** We evaluate the performance of GRE-TREE in small graphs built using parts of datasets in Figure 4b, where $|V| = 188,552$ for "Small Amazon", $|V| = 897,782$ for "Small DBLP", and $|V| = 2,423$ for "Small Movie". In Figures 4b (1-3), GRE-TREE produces lower-weight solutions than the other algorithms. By comparing the lower bounds, it can be seen that the solution weights of GRE-TREE are roughly at most 200% of the optimal solution weights for Amazon, and 300% for

DBLP and Movie. For Movie in Figure 4b (6), GRE-TREE is more than two orders of magnitude slower than the other algorithms. In comparison, for Amazon and DBLP in Figures 4b (4-5), GRE-TREE is an order of magnitude slower than baselines. The reason why GRE-TREE is particularly slow for Movie is that the sizes of vertex groups are large for Movie, which means that the size of the smallest group $|g_{min}|$ is often large for Movie, as shown in the supplement [6], while GRE-TREE uses PrunedDP++ $O(\xi \cdot |g_{min}|)$ times. Given that GRE-TREE can produce better solutions than the other algorithms, it may be preferable to use GRE-TREE when group sizes are small and graph sizes are not extremely large, *e.g.,* for Amazon. It is too slow to use GRE-TREE in the full DBLP and Movie graphs. As a result, *we use* GRE-TREE *only in the full* Amazon *graph, but not in the full* DBLP *and* Movie *graphs, in the following experiments.* Moreover, it is too slow to use DPBF in the full DBLP graph. Consequently, *we use* DPBF *only in the full* Amazon *and* Movie *graphs, but not in the full* DBLP *graph, in the following experiments.*

Evaluation of solution quality and speed in full graphs. We evaluate the solution quality and speed of algorithms in full graphs in Figure 4c. In Figures 4c (1-3), the solution weights of GRE-TREE and GRE-PATH are significantly lower than those of the baseline algorithms. This shows the effectiveness of GRE-TREE and GRE-PATH for finding probabilistic group Steiner trees. By comparing the lower bounds, it can be seen that the solution weights of GRE-PATH are roughly at most 200% of the optimal solution weights for Amazon, and 400% for DBLP and Movie. For Amazon and DBLP in Figures 4c (4-5), GRE-PATH is faster than the baselines. For Movie in Figure 4c (6), GRE-PATH is considerably slower than ImprovAPP, and has a similar speed with the other baselines. The reason why GRE-PATH does not have a high efficiency for Movie is that the time complexity of GRE-PATH is in proportion to $|g_{min}|$, and $|g_{min}|$ is large for Movie, as shown in the supplement [6].

Variation of the number of vertex groups: $|\Gamma|$. We vary $|\Gamma|$ in Figure 5a. In Figures 5a (1-3), the solution weights increase with $|\Gamma|$, since larger trees are required to cover more vertex groups. Similarly, the lower bounds increase with $|\Gamma|$. We further observe that the superior solution qualities of GRE-TREE and GRE-PATH over baselines hold well as $|\Gamma|$ varies. In Figures 5a (4-6), DPBF, PrunedDP++ and GRE-TREE do not scale well to $|\Gamma|$. The reason is that these algorithms have exponential time complexities with respect to $|\Gamma|$. In contrast, ENSteiner, ImprovAPP and GRE-PATH often scale well to $|\Gamma|$. Given that GRE-PATH finds high-quality solutions, it is preferable to use GRE-PATH when $|\Gamma|$ is large.

Variation of the threshold value: b. We vary the threshold value b in Figure 5b. The solution weights generally increase with b, since solution trees need to contain more vertices for satisfactorily covering all groups as b increases. The lower bounds do not change with b, since these lower bounds are weights of optimal classical group Steiner trees, which have no relation with b. Similarly, the lower bounds do not change with the following τ, P_{min}, P_{max} and k. In Figure 5b (4), the running time of GRE-TREE may increase with b, since it needs to employ PrunedDP++ to find and concatenate more trees for satisfactorily covering all groups as b increases.

Variation of the approximation parameter: τ. We vary the parameter τ in GRE-TREE and PrunedDP++ in Figure 5c. In Figure 5c (1), the solution weight of GRE-TREE slightly increases with τ. The reason is that τ is the approximation ratio of PrunedDP++ for solving the classical group Steiner tree problem. As a result, the solution weight of PrunedDP++ for a specific classical group Steiner tree instance increases with τ. Since GRE-TREE employs PrunedDP++ to find and concatenate trees, the weights of trees concatenated by GRE-TREE generally increase with τ. In Figures 5c (4-6), the running times of GRE-TREE and PrunedDP++ decrease with τ, since the number of enumerated trees in the dynamic programming process of PrunedDP++ decreases with τ.

Variation of the minimum and maximum positive probability values: P_{min} and P_{max}. We vary P_{min} and P_{max} in Figures 6a and 6b. We observe that the solution weights often decrease with P_{min} and P_{max}, since fewer vertices are required to satisfactorily cover vertex groups as probability values increase. We also observe that the running times of algorithms may decrease with P_{max}, *e.g.,* the running times of GRE-TREE and ImprovAPP in Figure 6b (4) decrease with P_{max}. The reason is that these algorithms may merge fewer trees or paths as probability values increase.

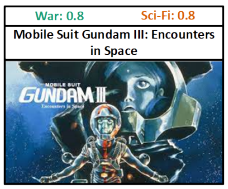
Variation of the number of feasible solutions in baselines: k. We vary k in Figure 6c. In Figures 6c (1-3), the solution weights of baselines decrease with k, since each baseline computes k feasible solutions and returns the best one. For the same reason, in Figures 6c (4-6), the running times of baselines increase with k. Notably, the solution weights of baselines are still considerably larger than those of GRE-TREE and GRE-PATH when $k = 7$. In particular, when $k = 7$, GRE-PATH is often significantly faster than baselines, and produces solutions with considerably smaller weights than baselines, *i.e.,* GRE-PATH still outperforms baselines when $k = 7$.

Key observations in experiments. To help analyze the above experiment results, we summarize the key observations as follows.

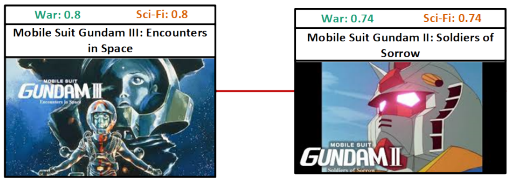
- DUAL can only be used in tiny graphs with dozens of vertices, and thus is mainly of theoretical interests (see Figures 4a (4-6)).
- GRE-TREE produces better solutions than the other algorithms, *e.g.,* Figures 4b (1-3), but does not scale well to group sizes (see Figure 4b (6) and the densities of group sizes in Figure 3). As a result, GRE-TREE is only useful when group sizes are small and graph sizes are not extremely large, *e.g.,* for Amazon.
- GRE-PATH produces considerably better solutions than the baseline algorithms, while scaling well to large graphs, *e.g.,* Figures 4c (1-6), and thus is a favorable tool for finding probabilistic group Steiner trees in various cases.
- The superior solution qualities of GRE-TREE and GRE-PATH over the baseline algorithms hold well when varying the parameters $|\Gamma|$, b, τ, P_{min}, P_{max} and k (see Figures 5 and 6).

4.4 A Case Study

Here, we conduct a case study using the Movie dataset. We also conduct case studies using the Amazon and DBLP datasets in the supplement [6]. For Movie, suppose that a user queries two genres: {War, Sci-Fi}. To help this user find related movies, we input two corresponding vertex groups, and apply the existing ImprovAPP and the proposed GRE-PATH to solve the classical and the probabilistic group Steiner tree problems, respectively. The existing ImprovAPP does not consider the probabilities of vertices for covering groups, and returns the single movie "Mobile Suit Gundam III: Encounters



(a) The retrieved movie via the classical approach.



(b) The retrieved movies via the probabilistic approach.

Figure 7: A case study based on the Movie dataset.

in Space", as shown in Figure 7a. This movie has a probability of 0.8 of satisfying the user for each input genre, which is smaller than the default threshold value $b = 0.9$. In comparison, the proposed GRE-PATH considers the probabilities of vertices for covering groups, and returns two related movies "Mobile Suit Gundam III: Encounters in Space" and "Mobile Suit Gundam II: Soldiers of Sorrow", as shown in Figure 7b. These two movies collectively have a probability of $1-(1-0.8)\cdot(1-0.74) = 0.948$ of satisfying the user for each input genre, which is larger than b. This case shows that, in probabilistic scenarios, we could retrieve information that is more likely to be favorable via the probabilistic group Steiner tree approach than via the classical group Steiner tree approach.

5 RELATED WORK

Reich and Widmayer [34] originally formulated the classical group Steiner tree problem for designing very-large-scale integrated circuits. A lot of recent work finds this problem useful for data mining, as discussed in the beginning of this paper. Three types of algorithms have been developed for finding classical group Steiner trees: *heuristic* (*e.g.*, [21, 34]) and *approximation* (*e.g.*, [10, 17, 19, 20, 37]) algorithms that find sub-optimal solutions without and with worst-case quality guarantees, respectively, and *exact* (*e.g.*, [13, 25]) algorithms that find optimal solutions. The heuristic algorithm in [21], dubbed ENSteiner, finds sub-optimal solutions via a transformation [15] from group Steiner trees to Steiner trees. Wang *et al.* [40] shows that ENSteiner is efficient and often finds high-quality solutions in practice, and thus can be seen as a state-of-the-art heuristic algorithm. Meanwhile, most existing approximation algorithms focus on achieving tight, often poly-logarithmic, approximation guarantees at the cost of large time complexities (*e.g.*, [17, 19]), and are not scalable in practice. Different from these algorithms, the recently developed $(|\Gamma|-1)$-approximation algorithm in [37], dubbed ImprovAPP, improves a previous $(|\Gamma|-1)$-approximation algorithm [20] on both efficiency and practical solution quality, and scales well in practice. Thus, ImprovAPP can be considered as a state-of-the-art scalable approximation algorithm. With respect to exact algorithms, the dynamic programming algorithm in [13], dubbed DPBF, is the first exact algorithm for finding classical group Steiner trees. Recently, Li *et al.* [25] improves DPBF on practical efficiency by incorporating pruning techniques into the dynamic programming process. The improvement, dubbed PrunedDP++, is a state-of-the-art exact algorithm. All the above algorithms do not consider the probabilities of vertices for covering groups, and thus do not suit retrieving information from graph-structured datasets with uncertain vertex properties, like the academic datasets labeled by current artificial intelligence techniques [1, 4], as discussed in Section 1. Moreover, as discussed in Section 1, the existing work on probabilistic databases (*e.g.*, [8, 14, 24, 29–31, 33, 35, 36, 39, 42–45]) focuses on performing different tasks, and cannot address this issue. This motivates us to develop new algorithms for finding probabilistic group Steiner trees in this paper.

6 CONCLUSIONS AND FUTURE WORK

Solving the probabilistic group Steiner tree problem is useful for mining various graph-structured datasets with uncertain vertex properties. To the best of our knowledge, no work has been done to solve this problem to date. We address this issue by proposing three approximation algorithms, dubbed DUAL, GRE-TREE and GRE-PATH, that vary with approximation guarantees and time complexities for solving this problem. Experiments on real datasets show that (i) DUAL can only be used in tiny graphs with dozens of vertices, and thus is mainly of theoretical interests; (ii) GRE-TREE produces better solutions than the other algorithms, and is efficient when group sizes are small; and (iii) GRE-PATH produces considerably better solutions than the state-of-the-art techniques, while scaling well to large graphs.

As discussed in Sections 1 & 2, like a lot of work on probabilistic data management (*e.g.*, [8, 14, 42, 43]), we consider that different probability values are independent from each other. There are also cases where different probability values may correlate with each other, *e.g.*, given two vertices v_1 and v_2, and two groups g_1 and g_2, $p_{g_1}(v_1)$ and $p_{g_2}(v_1)$ may correlate with each other, or $p_{g_1}(v_1)$ and $p_{g_1}(v_2)$ may correlate with each other. Studying such correlated cases is a challenging, but meaningful, future work to do.

Notably, we use weights of optimal classical group Steiner trees as lower bounds in experiments. These lower bounds may not be tight enough to estimate the practical solution qualities of the proposed algorithms well, *e.g.*, in Figures 4c (2-3), the fact that the solution weights of GRE-PATH are around 400% of the lower bounds does not mean that the solution weights of GRE-PATH are around 400% of the optimal solution weights, since these lower bounds may be much smaller than the optimal solution weights. It is preferable to develop tighter lower bounds in the future to better estimate the practical solution qualities of the proposed algorithms.

Further note that, for each vertex $v \in g_{min}$, GRE-PATH merges paths between v and other vertices in an increasing order of the weights of paths. It may be worth exploring new ordering and merging methods in the future, for achieving stronger approximation guarantees, or higher efficiencies and solution qualities in practice.

ACKNOWLEDGMENTS

This work is supported by (i) NSFC Grant 62202472; (ii) a Start Up Grant from National University of Singapore; (iii) NSFC Grants 62072034 and U1809206; and (iv) NSFC Grants 61972401, 61932001, 61832017, and Beijing Natural Science Foundation Grant 4222028, and the major key project of PCL (PCL2021A12). The work is partially done at Peng Cheng Laboratory.

REFERENCES

[1] 2022. AMiner. https://www.aminer.org.
[2] 2022. AMiner: Citation Network Dataset. https://www.aminer.cn/citation.
[3] 2022. GroupLens. https://grouplens.org.
[4] 2022. Microsoft Academic Graph. https://www.microsoft.com/en-us/research/project/microsoft-academic-graph.
[5] 2022. Stanford Network Analysis Project. http://snap.stanford.edu.
[6] 2022. Supplement. https://github.com/rucdatascience/PGST/blob/main/Supplement.pdf.
[7] Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. 2013. 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*. 349–360.
[8] Thomas Bernecker, Hans-Peter Kriegel, Matthias Renz, Florian Verhein, and Andreas Zuefle. 2009. Probabilistic frequent itemset mining in uncertain databases. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. 119–128.
[9] Mark Bukowski, Sandra Geisler, Thomas Schmitz-Rode, and Robert Farkas. 2020. Feasibility of activity-based expert profiling using text mining of scientific publications and patents. *Scientometrics* 123, 2 (2020), 579–620.
[10] Moses Charikar, Chandra Chekuri, Ashish Goel, and Sudipto Guha. 1998. Rounding via trees: deterministic approximation algorithms for group Steiner trees and k-median. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. 114–123.
[11] Joel Coffman and Alfred C Weaver. 2014. An empirical performance evaluation of relational keyword search techniques. *IEEE Transactions on Knowledge and Data Engineering* 26, 1 (2014), 30–42.
[12] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. 2003. Reachability and distance queries via 2-hop labels. *SIAM Journal on Computing* 32, 5 (2003), 1338–1355.
[13] Bolin Ding, Jeffrey Xu Yu, Shan Wang, Lu Qin, Xiao Zhang, and Xuemin Lin. 2007. Finding top-k min-cost connected trees in databases. In *IEEE International Conference on Data Engineering*. IEEE, 836–845.
[14] Huizhong Duan, ChengXiang Zhai, Jinxing Cheng, and Abhishek Gattani. 2013. Supporting keyword search in product database: a probabilistic approach. *Proceedings of the VLDB Endowment* 6, 14 (2013), 1786–1797.
[15] CW Duin, A Volgenant, and Stefan Voß. 2004. Solving group Steiner problems as Steiner problems. *European Journal of Operational Research* 154, 1 (2004), 323–329.
[16] Karoline Faust, Pierre Dupont, Jérôme Callut, and Jacques Van Helden. 2010. Pathway discovery in metabolic networks by subgraph extraction. *Bioinformatics* 26, 9 (2010), 1211–1218.
[17] Naveen Garg, Goran Konjevod, and R Ravi. 2000. A polylogarithmic approximation algorithm for the group Steiner tree problem. *Journal of Algorithms* 37, 1 (2000), 66–84.
[18] Shuo Han, Lei Zou, Jeffery Xu Yu, and Dongyan Zhao. 2017. Keyword search on RDF graphs-a query graph assembly approach. In *Proceedings of the ACM Conference on Information and Knowledge Management*. ACM, 227–236.
[19] Christopher S Helvig, Gabriel Robins, and Alexander Zelikovsky. 2001. An improved approximation scheme for the group Steiner problem. *Networks* 37, 1 (2001), 8–20.
[20] Edmund Ihler. 1990. Bounds on the quality of approximate solutions to the group Steiner problem. In *International Workshop on Graph-Theoretic Concepts in Computer Science*. Springer, 109–118.
[21] Theodoros Lappas, Kun Liu, and Evimaria Terzi. 2009. Finding a team of experts in social networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 467–476.
[22] Guoliang Li, Beng Chin Ooi, Jianhua Feng, Jianyong Wang, and Lizhu Zhou. 2008. EASE: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM, 903–914.
[23] Guoliang Li, Xiaofang Zhou, Jianhua Feng, and Jianyong Wang. 2009. Progressive keyword search in relational databases. In *IEEE International Conference on Data Engineering*. IEEE, 1183–1186.
[24] Jianxin Li, Chengfei Liu, Rui Zhou, and Jeffrey Xu Yu. 2013. Quasi-SLCA based keyword query processing over probabilistic XML data. *IEEE Transactions on Knowledge and Data Engineering* 26, 4 (2013), 957–969.
[25] Rong-Hua Li, Lu Qin, Jeffrey Xu Yu, and Rui Mao. 2016. Efficient and progressive group Steiner tree search. In *Proceedings of the 2016 International Conference on Management of Data*. ACM, 91–106.
[26] Wentao Li, Miao Qiao, Lu Qin, Ying Zhang, Lijun Chang, and Xuemin Lin. 2019. Scaling distance labeling on small-world networks. In *Proceedings of the 2019 International Conference on Management of Data*. 1060–1077.
[27] Wentao Li, Miao Qiao, Lu Qin, Ying Zhang, Lijun Chang, and Xuemin Lin. 2020. Scaling up distance labeling on graphs with core-periphery properties. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1367–1381.
[28] Ye Li, Leong Hou U, Man Lung Yiu, and Ngai Meng Kou. 2017. An experimental study on hub labeling based shortest path algorithms. *Proceedings of the VLDB Endowment* 11, 4 (2017), 445–457.
[29] Xiang Lian and Lei Chen. 2008. Probabilistic ranked queries in uncertain databases. In *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*. 511–522.
[30] Xiang Lian and Lei Chen. 2011. Efficient query answering in probabilistic RDF graphs. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. 157–168.
[31] Xiang Lian, Lei Chen, and Zi Huang. 2015. Keyword search over probabilistic RDF graphs. *IEEE transactions on knowledge and data engineering* 27, 5 (2015), 1246–1260.
[32] Anirban Majumder, Samik Datta, and KVM Naidu. 2012. Capacitated team formation problem on social networks. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1005–1013.
[33] Odysseas Papapetrou, Ekaterini Ioannou, and Dimitrios Skoutas. 2011. Efficient discovery of frequent subgraph patterns in uncertain graph databases. In *Proceedings of the 14th International Conference on Extending Database Technology*. 355–366.
[34] Gabriele Reich and Peter Widmayer. 1989. Beyond Steiner's problem: A VLSI oriented generalization. In *International Workshop on Graph-theoretic Concepts in Computer Science*. Springer, 196–210.
[35] Sarvjeet Singh, Chris Mayfield, Sunil Prabhakar, Rahul Shah, and Susanne Hambrusch. 2007. Indexing uncertain categorical data. In *2007 IEEE 23rd International Conference on Data Engineering*. IEEE, 616–625.
[36] Mohamed A Soliman, Ihab F Ilyas, and Kevin Chen-Chuan Chang. 2007. Top-k query processing in uncertain databases. In *2007 IEEE 23rd International Conference on Data Engineering*. IEEE, 896–905.
[37] Yahui Sun, Xiaokui Xiao, Bin Cui, Saman Halgamuge, Theodoros Lappas, and Jun Luo. 2021. Finding Group Steiner Trees in Graphs with both Vertex and Edge Weights. *Proceedings of the VLDB Endowment* 14, 7 (2021), 1137–1149.
[38] Jie Tang, Limin Yao, Duo Zhang, and Jing Zhang. 2010. A combination approach to web user profiling. *ACM Transactions on Knowledge Discovery from Data* 5, 1 (2010), 1–44.
[39] Yufei Tao, Xiaokui Xiao, and Reynold Cheng. 2007. Range search on multidimensional uncertain data. *ACM Transactions on Database Systems* 32, 3 (2007), 15–es.
[40] Xinyu Wang, Zhou Zhao, and Wilfred Ng. 2016. Ustf: A unified system of team formation. *IEEE Transactions on Big Data* 2, 1 (2016), 70–84.
[41] John William Joseph Williams. 1964. Algorithm 232: heapsort. *Commun. ACM* 7 (1964), 347–348.
[42] Ke Yi, Feifei Li, George Kollios, and Divesh Srivastava. 2008. Efficient processing of top-k queries in uncertain databases with x-relations. *IEEE transactions on knowledge and data engineering* 20, 12 (2008), 1669–1682.
[43] Ye Yuan, Guoren Wang, Lei Chen, and Haixun Wang. 2012. Efficient subgraph similarity search on large probabilistic graph databases. *Proceedings of the VLDB Endowment* 5, 9 (2012), 800–811.
[44] Ye Yuan, Guoren Wang, Lei Chen, and Haixun Wang. 2013. Efficient keyword search on uncertain graph data. *IEEE Transactions on Knowledge and Data Engineering* 25, 12 (2013), 2767–2779.
[45] Zhaonian Zou, Jianzhong Li, Hong Gao, and Shuo Zhang. 2010. Finding top-k maximal cliques in an uncertain graph. In *IEEE 26th International Conference on Data Engineering*. IEEE, 649–652.