Approximating Probabilistic Group Steiner Trees in Graphs: Supplemental Materials

This supplement is available online [1]. The road map of this supplement is as follows.

- In Section S1, we discuss that the existing dynamic programming methods for solving the classical group Steiner tree problem do not suit finding optimal solutions to the probabilistic group Steiner tree problem.
- In Section S2, we prove that the probabilistic group Steiner tree problem is NP-hard when $|\Gamma| = 1$ and G is a tree.
- In Section S3, we prove the approximation guarantee of GRE-PATH.
- In Sections S4-S6, we discuss the time complexities of DUAL, GRE-TREE and GRE-PATH, respectively.
- In Section S7, we explain why we use Fibonacci heap [2] in Line 5 of GRE-PATH.
- In Section S8, we incorporate ImprovAPP with DUAL and GRE-TREE.
- In Section S9, we conduct additional case studies.
- In Section S10, we discuss the methods of selecting vertex groups in experiments.
- In Section S11, we report the memory consumption of algorithms in experiments.
- In Section S12, we show the sizes of $|V_g|$ and $|g_{min}|$ in experiments.
- In Section S13, we conduct additional experiments where edge weights are defined as pairwise Jaccard distances.

S1. SOME DISCUSSIONS ON THE DYNAMIC PROGRAMMING APPROACH TO FINDING GROUP STEINER TREES

The experiments in the main content indicates that two dynamic programming algorithms, DPBF in [3] and PrunedDP++ in [4], 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 efficiently solve the probabilistic group Steiner tree problem to optimality. Here, we show that, since the probabilistic group Steiner tree problem has a harsher NP-hard condition than the classical problem, 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-weight 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 [3]).

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

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 weight 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 weight of the combined tree is minimal. We show an example in Figure S1, where $\Gamma = \{g_1, g_2\}$, $g_1 = \{v_1, v_4\}$, $g_2 = \{v_2, v_3\}$, all edge weights 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-weight tree that roots at itself and contains at least one vertex in this group. For example, we initialize $T(v_1, \mathbf{p}_1) = \{v_1\}$ and $T(v_2, \mathbf{p}_2) = \{v_2\}$. Then, we grow $T_g(v_2, \mathbf{p}_1) = (v_1, v_2) \cup T(v_1, \mathbf{p}_1) = (v_1, v_2)$. 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.

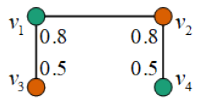


Fig. S1. An illustration of the dynamic programming approach to finding group Steiner trees.

Then, 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-weight 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-weight tree that roots at this vertex and satisfactorily covers this group. However, Theorem 1 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 such trees in the classical case, since the classical problem is trivial when $|\Gamma| = 1$ (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 harsher NP-hard condition of the probabilistic group Steiner tree problem, we cannot extend the above dynamic programming model to efficiently solve the probabilistic group Steiner tree problem to optimality.

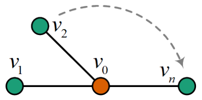


Fig. S2. An illustration for proving Theorem 1.

S2. THE PROOF OF NP-HARDNESS

Theorem 1. *Problem 1 is NP-hard when* $|\Gamma| = 1$ *and* G *is a tree.*

Proof. We prove this theorem by reducing the well-known knapsack problem [5] to a special case of Problem 1 where $|\Gamma| = 1$ and G is a tree in polynomial time. We briefly introduce the knapsack problem: given a set $X = \{v_1, \ldots, v_n\}$ of items, each item $v \in X$ is associated with a weight value $w(v)$ and a profit value $o(v)$, along with a maximum weight capacity W, the objective is to find a subset $X' \subseteq X$ of items such that (i) $\sum_{v \in X'} w(v) \leq W$; and (ii) $\sum_{v \in X'} o(v)$ is maximized. We reduce the knapsack problem to a special case of Problem 1 as follows.

Given an instance of the knapsack problem, we build the tree in Figure S2, where the sets of vertices and edges are $V = \{v_0, v_1, \ldots, v_n\}$ and $E = \{(v_0, v_1), \ldots, (v_0, v_n)\}$, respectively. The weight of edge (v_0, v_i) is $o(v_i)$ for every $i \in [1, n]$. Let $o(v_0) = 0$. Let $V' \subseteq V$ be a subset of vertices such that $v_0 \in V'$. The knapsack problem is to find a subset $X' = V \setminus V' \subseteq X$ of items such that (i) $\sum_{v \in V \setminus V'} w(v) \leq W$; and (ii) $\sum_{v \in V'} o(v)$ is minimized (notably, minimizing $\sum_{v \in V'} o(v)$ is equivalent to maximizing $\sum_{v \in V \setminus V'} o(v)$). Consider an instance of Problem 1 in the above tree. Let $\Gamma = \{g\}$, $g = \{v_1, \ldots, v_n\}$, and $p_g(v) = w(v)$ for all $v \in g$ be such small values that $p_g(v_i) \cdot p_g(v_j)$ is negligible for every pair of $i, j \in [1, n]$ (notably, given any instance of the knapsack problem, we can make $w(v)$ such small values by dividing $w(v)$ and W by a constant large value, without changing the solution). Note that, $w(v_0) = p_g(v_0) = 0$. In the knapsack problem, we have

$$W < \sum_{v \in V} w(v) = \sum_{v \in V} p_g(v), \tag{S4}$$

as otherwise the knapsack problem is trivial to solve. In the above instance of Problem 1, since $w(v) = p_g(v)$ for all $v \in g$ are small values and $b \in (0, 1]$, we let

$$b = \sum_{v \in V} p_g(v) - W \tag{S5}$$

without loss of generality. The condition that $p_g(V') \geq b$ in the above instance of Problem 1 is

$$p_g(V') = 1 - \prod_{v \in V'} [1 - p_g(v)] \approx \sum_{v \in V'} p_g(v) \geq b, \tag{S6}$$

i.e.,

$$\sum_{v \in V \setminus V'} p_g(v) \leq \sum_{v \in V} p_g(v) - b = W, \tag{S7}$$

which is equivalent to the condition that $\sum_{v \in V \setminus V'} w(v) \leq W$. The above instance of Problem 1 is to find V' such that (i) $p_g(V') \geq b$; and (ii) $\sum_{v \in V'} o(v)$ is minimized. Therefore, an optimal solution to the above instance of Problem 1 corresponds to an optimal solution to the given instance of the knapsack problem, and vice versa. Thus, via the above approach, we reduce the knapsack problem to a special case of Problem 1 where $|\Gamma| = 1$ and G is a tree in polynomial time. Since the knapsack problem is NP-hard [5], this theorem holds. □

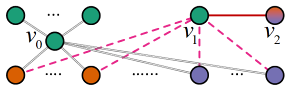


Fig. S3. The sharpness of $\max\{1, \sum_{g \in \Gamma} \xi_g - 1\}$.

S3. THE APPROXIMATION GUARANTEE OF GRE-PATH

We prove the approximation guarantee of GRE-PATH as follows.

Theorem 4. 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.

Proof. Let Θ_{opt} be an optimal solution. Suppose that Θ_{opt} contains $v \in g_{min}$. Let Θ_v be the feasible solution produced by GRE-PATH in the loop for v (Lines 3-18). Line 17 guarantees that

$$c(\Theta_3) \leq c(\Theta_v). \tag{S8}$$

To satisfactorily cover each vertex group g, GRE-PATH merges at most ξ_g paths from v to vertices in g. Suppose that GRE-PATH merges $P(v, u_1), P(v, u_2), \cdots, P(v, u_x)$ for satisfactorily covering g, *i.e,* $u_1, u_2, \ldots, u_x \in g$, and

$$c(P(v, u_1)) \leq c(P(v, u_2)) \leq \cdots \leq c(P(v, u_x)). \tag{S9}$$

Since Θ_{opt} contains v and satisfactorily covers g, there are two possible scenarios: (i) Θ_{opt} contains $u_1, u_2, \ldots, u_x$, and does not contain any other vertex in g; (ii) Θ_{opt} contains $u_y \in g$ such that $u_y \notin \{u_1, u_2, \ldots, u_x\}$. Since GRE-PATH merges shortest paths in an increasing order of the weights of these paths, in both scenarios,

$$c(P(v, u_x)) \leq c(\Theta_{opt}). \tag{S10}$$

Thus, the weight of each path that is merged into Θ_v is not larger than $c(\Theta_{opt})$. To satisfactorily cover every vertex group, GRE-PATH merges at most $\sum_{g \in \Gamma} \xi_g$ paths into Θ_v. Moreover, the single-vertex path $P(v, v)$ can be seen as merged into Θ_v for satisfactorily covering g_{min}. Since $c(P(v, v)) = 0$, we have

$$c(\Theta_v) \leq \Big(\sum_{g \in \Gamma} \xi_g - 1\Big) \cdot c(\Theta_{opt}). \tag{S11}$$

Therefore, $\sum_{g \in \Gamma} \xi_g - 1$ is an approximation guarantee of GRE-PATH. We prove the sharpness of this guarantee via the instance in Figure S3, where g_{min} is the set of green vertices. There are three special vertices: $v_0 \in g_{min}$, $v_1 \in g_{min}$ and $v_2 \notin g_{min}$. There is an edge between v_0 and any other vertex except v_1 and v_2, and the weight of each of these edges (*i.e.*, each gray edge) is 1. There is an edge between v_1 and any other vertex that is not in g_{min}, and the weight of each of these edges except edge (v_1, v_2) (*i.e.*, each pink edge) is $1 + \delta$, while the weight of (v_1, v_2) (*i.e.*, the red edge) is $1 + 2\delta$, where δ is a positive value. There is no other edge in this graph. $p_{g_{min}}(v)$ are equal small values for every $v \in g_{min} \setminus v_1$, while $p_{g_{min}}(v_1) = 1$. v_2 is in every vertex group $g \in \Gamma \setminus g_{min}$ (*e.g.*, the set of orange vertices and v_2 is a group, and the set of purple vertices and v_2 is another group).

$p_g(v_2) = 1$ for every $g \in \Gamma \setminus g_{min}$. Furthermore, the set of gray edges corresponds to a feasible solution. In the loop of Lines 3-18, GRE-PATH produces a feasible solution Θ_{v_0}, which contains and only contains all the gray edges, and

$$c(\Theta_{v_0}) = \sum_{g\in\Gamma} \xi_g - 1. \tag{S12}$$

Moreover, GRE-PATH also produces a feasible solution Θ_{v_1}, which contains and only contains all the pink edges, and

$$c(\Theta_{v_1}) = \sum_{g\in\Gamma\setminus g_{min}} \xi_g \cdot (1+\delta). \tag{S13}$$

Suppose that $c(\Theta_{v_0}) \leq c(\Theta_{v_1})$, which means that

$$\sum_{g\in\Gamma} \xi_g - 1 \leq \sum_{g\in\Gamma\setminus g_{min}} \xi_g + \sum_{g\in\Gamma\setminus g_{min}} \xi_g \cdot \delta, \tag{S14}$$

$$\delta \geq \frac{\xi_{g_{min}} - 1}{\sum_{g\in\Gamma\setminus g_{min}} \xi_g}. \tag{S15}$$

Thus, when $|\Gamma|$ is large, δ can be a tiny value. When δ is tiny, the optimal solution is $\Theta_{opt} = (v_1, v_2)$, and

$$c(\Theta_{opt}) = 1 + 2\delta. \tag{S16}$$

In the above case where $c(\Theta_{v_0}) \leq c(\Theta_{v_1})$, the solution of GRE-PATH is $\Theta_3 = \Theta_{v_0}$. The approximation ratio of GRE-PATH is

$$\lim_{\delta\to 0} \frac{c(\Theta_3)}{c(\Theta_{opt})} = \frac{\sum_{g\in\Gamma} \xi_g - 1}{1+2\delta} = \sum_{g\in\Gamma} \xi_g - 1. \tag{S17}$$

Moreover, in scenarios where $\Gamma = \{g\}$ and $\xi_g = 1$, GRE-PATH can find optimal solutions, *i.e.*, the approximation ratio of GRE-PATH is 1. Hence, $\max\{1, \sum_{g\in\Gamma} \xi_g - 1\}$ is a sharp approximation guarantee of GRE-PATH. This theorem holds. □

S4. THE 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 as follows. DUAL initializes a tree (line 1) in $O(1)$ time. It finds Φ_g for every $g \in \Gamma$ (Line 2) in $O(|\Gamma| \cdot \xi^2 \cdot |V|^{\xi})$ time. The reason is as follows. For any group $g \in \Gamma$, let V_g be an essential cover of g. Since

$$1 - (1-p_{min})^{\xi} \geq 1 - (1-p_{min})^{\log_{(1-p_{min})}(1-b)} \geq b, \tag{S18}$$

we have

$$|V_g| \leq \min\{|g|, \xi\}. \tag{S19}$$

That is to say, the size of any essential cover of any vertex group g is not larger than $\min\{|g|, \xi\}$. To find Φ_g, we find every set of k vertices in g for every $k \in [1, \min\{|g|, \xi\}]$, and check whether these sets of vertices are essential covers of g. Since checking whether a set of vertices V' is an essential cover of g takes $O(|V'|)$ time and enumerating every set of k vertices in g takes $O(|g|^k)$ time, DUAL finds Φ_g for every $g \in \Gamma$ (Line 2) in

$$O\Big(|\Gamma| \cdot (|V|^1 1 + \cdots + |V|^{\xi}\xi)\Big) = O\Big(|\Gamma| \cdot \xi^2 \cdot |V|^{\xi}\Big) \tag{S20}$$

time. It finds Φ_{g_x} in $O(|\Gamma|)$ time. Since $|V_{g_x}| \le \min\{|g_x|, \xi\}$,

$$O\Big(|\Phi_{g_x}|\Big) = O\Big(\sum_{j \in [1, \min\{|g_x|, \xi\}]} \binom{|g_x|}{j}\Big) = O\Big(\xi \cdot |V|^{\xi}\Big). \tag{S21}$$

It processes each V' in Φ_{g_x} as follows (Lines 4-18). If $|\Gamma| = 1$ (Line 4), it uses PrunedDP++ to find $\Theta_{V'}$ (Line 5) in

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

time, where $O(3^{\xi}|V|)$ corresponds to a tree merging process in PrunedDP++, and $O(2^{\xi}|V| \cdot (\xi|V| + |E| + |V|\log|V|))$ corresponds to merging $O(\xi)$ shortest paths together and finding an MST of the merged graph for producing a feasible solution tree $O(2^{\xi}|V|)$ times (details in [4]; notably, PrunedDP++ improves the previous DPBF algorithm [3] on practical efficiency, but has a larger time complexity than DPBF). If $|\Gamma| > 1$, it builds $\Theta_{V'}$ as follows (Lines 7-16). It initializes G' in $O(1)$ time (Line 7). For each $g \in \Gamma \setminus g_x$ and each $V_j \in \Phi_g$, it finds $\Theta(V', V_j)$ (Line 11) in

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

time. Thus, the cost of Line 11 throughout the loop of Lines 8-15 is

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

It uses each computed $\Theta(V', V_j)$ to update $\Theta_{ST}(V', \Phi_g)$ (Line 12), and each update takes $O(|V|)$ time. For each $g \in \Gamma \setminus g_x$, it merges $\Theta_{ST}(V', \Phi_g)$ into G' (Line 14) in $O(|V|)$ time. After the loop of Lines 8-15, it finds an MST as $\Theta_{V'}$ (Line 16) in $O(|E| + |V|\log|V|)$ time. It updates Θ_1 (Line 18) in $O(|V|)$ time. At last, it returns Θ_1 (Line 20) in $O(|V|)$ time.

S5. 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 group or not (Line 5), we record the probability that G' does not satisfactorily cover each 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 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

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

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 $O(|V|)$.

S6. THE TIME COMPLEXITY OF GRE-PATH

GRE-PATH has a time complexity of

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

where L is the average number of hub labels associated with each vertex (details in [6–8]). 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 $O\Big(|\Gamma| \cdot |V| \cdot (L + \log|V|)\Big)$ time, since the time complexity of checking the cost of $P(v, u)$ in Line 7 is $O(L)$, and the time complexity of inserting an element into a Binary heap is $O(\log|V|)$. Subsequently, it initialize Θ_v in $O(1)$ time (Line 10). To efficiently check the probability that Θ_v satisfactorily covers each group, we record the probability that Θ_v does not satisfactorily cover each 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, and merge a path into Θ_v (Line 14) in $O(L|V|)$ time. 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 an MST that spans the vertices in Θ_3 (Line 19) in $O(|E| + |V|\log|V|)$ time.

S7. THE CHOICE OF HEAPS IN GRE-PATH

GRE-PATH frequently inserts elements into heaps in Line 7 and pops elements out of heaps in Line 13. In Figure S4, we compare the speeds of Binary [9], Fibonacci [2] and Pairing [10] heaps in the C++ Boost Library (https://www.boost.org) for these operations. The experiment results show that Binary heap is the fastest. Thus, we use Binary heap in Line 5 of GRE-PATH.

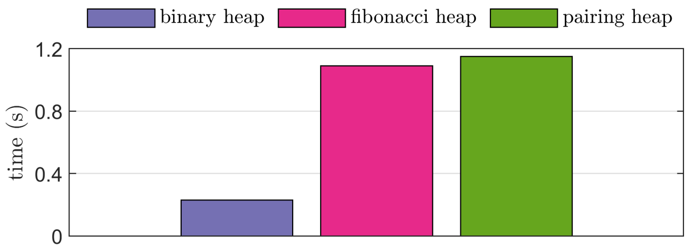


Fig. S4. The running times of three different heaps for inserting one million elements with random priority values and then popping out these elements.

S8. INCORPORATING IMPROVAPP WITH DUAL AND GRE-TREE

DUAL and GRE-TREE incorporate a state-of-the-art classical group Steiner tree algorithm: PrunedDP++ [4], which is based on a dynamic programming approach. We can also replace PrunedDP++ in DUAL and GRE-TREE with another state-of-the-art classical group Steiner tree algorithm: ImprovAPP [11], which is based on a greedy concatenation approach. Specifically, we can replace PrunedDP++ with ImprovAPP in Lines 5 and 11 of DUAL, as well as in Line 7 of GRE-TREE. We refer to the two algorithms after the replacement as DUAL-ImprovAPP and GRE-TREE-ImprovAPP, respectively.

In Figure S5, we compare DUAL-ImprovAPP with GRE-PATH, where $|V| = 300$ for "Tiny Amazon", "Tiny DBLP" and "Tiny Movie", and the other parameter settings are the same with those in the main experiments. It can be seen that the solution weights of DUAL-ImprovAPP are smaller than those of GRE-PATH, at the cost of a significantly lower speed than GRE-PATH, since a large number of essential covers are enumerated in Lines 3 and 10 of Algorithm 1. In Figure S6, we compare GRE-TREE-ImprovAPP with GRE-PATH, where $|V| = 548,552$ for "Amazon", $|V| = 897,782$ for "Small DBLP" and $|V| = 4,000$ for "Small Movie". Like the above experiment results, the solution weights of GRE-TREE-ImprovAPP are smaller than those of GRE-PATH, at the cost of a significantly lower speed than GRE-PATH, since, for each vertex $v \in g_{min}$, GRE-TREE-ImprovAPP implements ImprovAPP multiple times in the while loop in Lines 5-9 of Algorithm 2.

Since ImprovAPP have an approximation guarantee of $\max\{1, |\Gamma| - 1\}$ for solving the classical group Steiner tree problem, DUAL-ImprovAPP has an approximation guarantee of $(\max\{1, |\Gamma| - 1\})^2$ (by changing τ in the approximation guarantee of DUAL to $\max\{1, |\Gamma| - 1\}$), while GRE-TREE-ImprovAPP has an approximation guarantee of $\max\{1, |\Gamma| - 1\} \cdot \xi$ (by changing τ in the approximation guarantee of GRE-TREE to $\max\{1, |\Gamma| - 1\}$). Therefore, DUAL and GRE-TREE can achieve tighter guarantees than DUAL-ImprovAPP and GRE-TREE-ImprovAPP, respectively, by setting $\tau \geq 1$ smaller than $\max\{1, |\Gamma| - 1\}$ when $|\Gamma| > 2$.

Given that, in comparison with GRE-PATH, DUAL-ImprovAPP and GRE-TREE-ImprovAPP have a significantly low efficiency and thus mainly of theoretical interests, we choose to make these theoretical interests as strong as possible. As a result, we incorporate PrunedDP++, but not ImprovAPP, with DUAL and GRE-TREE, for achieving tight and tunable approximation guarantees.

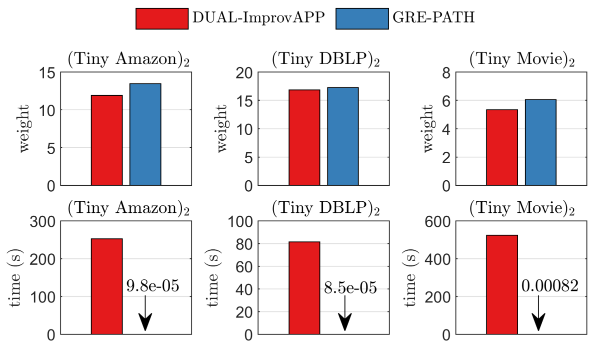


Fig. S5. DUAL-ImprovAPP is significantly slower than GRE-PATH.

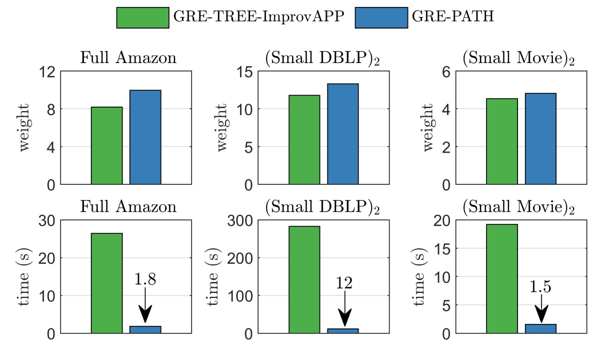


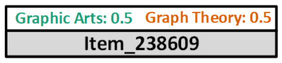
Fig. S6. GRE-TREE-ImprovAPP is significantly slower than GRE-PATH.

S9. ADDITIONAL CASE STUDIES

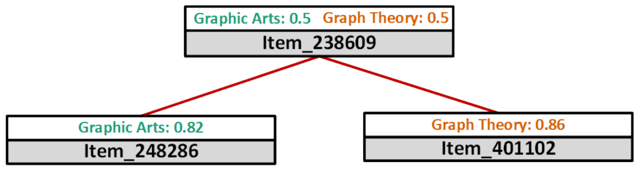
In the main contents, we conduct a case study using the Movie dataset. In addition, here, we conduct similar case studies using the Amazon and DBLP datasets.

For Amazon, suppose that a user queries two keywords: {Graphic Arts, Graph Theory}. To help this user find related products, 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 product "Item_238609" in Figure S7a. This product has a probability of 0.5 of satisfying the user for each input keyword. This probability 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 three related products in Figure S7b. These three products collectively have a probability of $1 - (1 - 0.82) \cdot (1 - 0.5) = 0.91$ of satisfying the user for the keyword "Graphic Arts", and a probability of $1 - (1 - 0.86) \cdot (1 - 0.5) = 0.93$ of satisfying the user for the keyword "Graph Theory". These two probabilities are larger than b. Thus, GRE-PATH retrieves closely related products that are more likely to satisfy the user.

For DBLP, suppose that a user queries two topics: {Distributed design patterns, Separation of concerns}. To help this user find related papers, 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 returns the single paper "On the Influence of Scale in a Distributed System" in Figure S8a. This paper has a probability of 0.75 of being in the field of "Distributed design patterns", and a probability of 0.72 of being in the field of "Separation of concerns". These two probabilities are smaller than b. In comparison, the proposed GRE-PATH returns three related papers in Figure S8b. These three paper collectively have a probability of $1 - (1 - 0.76) \cdot (1 - 0.74) = 0.9376$ of being in the field of "Distributed design patterns", and a probability of $1 - (1 - 0.72) \cdot (1 - 0.72) = 0.9216$ of being in

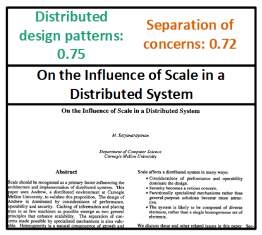


(a) The retrieved product via the classical approach.

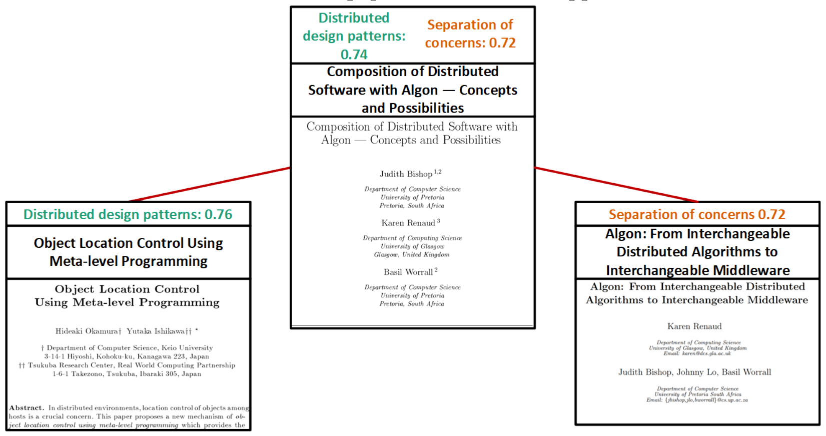


(b) The retrieved products via the probabilistic approach.

Fig. S7. A case study based on the Amazon dataset.


Distributed design patterns: 0.75
Separation of concerns: 0.72
On the Influence of Scale in a Distributed System

(a) The retrieved paper via the classical approach.


Distributed design patterns: 0.74
Separation of concerns: 0.72
Composition of Distributed Software with Algon — Concepts and Possibilities
Distributed design patterns: 0.76
Object Location Control Using Meta-level Programming
Separation of concerns 0.72
Algon: From Interchangeable Distributed Algorithms to Interchangeable Middleware

(b) The retrieved papers via the probabilistic approach.

Fig. S8. A case study based on the DBLP dataset.

the field of "Separation of concerns". These two probabilities are larger than b. Thus, GRE-PATH retrieves closely related papers that are more likely to be in the queried fields.

Like the Movie case study in the main contents, the above Amazon and DBLP case studies show 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.

S10. THE METHODS OF SELECTING VERTEX GROUPS IN EXPERIMENTS

In the main experiments, we select $|\Gamma|$ groups in such a way that the PoIs corresponding to the selected groups are often related and may appear together in practice. Since a Breadth First Search is conducted in this selection method, we refer to this selection method as the BFS selection method. A naive comparison of this method is the Simple Random selection method, *i.e.*, selecting $|\Gamma|$ vertex groups uniformly at random. Notably, each group corresponds to a PoI. Here, we show some examples of the selected PoIs using the BFS and the Simple Random selection methods.

Specifically, in Tables S1 and S2, we show some examples of the selected PoIs using the BFS and the Simple Random selection methods, respectively, for Amazon. It can be seen from Table S1 that, when using the BFS selection method, the selected PoIs are often related and may occur together in practice, such as the five PoIs in the No.8 row in Table S1: "Thiebaud, Wayne | Realism | Painting | Art | Arts & Photography" (notably, Wayne Thiebaud is an American painter; https://en.wikipedia.org/wiki/Wayne_Thiebaud). Comparatively, it can be seen from Table S2 that, when using the Simple Random selection method, the selected PoIs are often unrelated and may rarely occur together in practice, *e.g.,* the two PoIs "Iranian Cinema" and "Composition & Creative Writing" in the No.9 row in Table S2 are barely related and rarely occur together in practice. We can get similar comparison results by analyzing the examples of the selected PoIs for DBLP in Tables S3-S4. Such comparison results are not so obvious for Movie in Tables S5-S6, due to the reason that there are only 19 PoIs in the Movie dataset, and most pairs of these PoIs may occur together in practice. Nevertheless, the above examples of the selected PoIs indicate that the BFS selection method is more practical than the Simple Random selection method, which is the reason why we use the BFS selection method in the experiments in the main contents.

S11. THE MEMORY CONSUMPTION OF ALGORITHMS IN EXPERIMENTS

In Figures S9-S10, we report the memory consumption of algorithms in the main experiments. The reported memory consumption does not contain the memory consumed by the common inputs of algorithms, *i.e.,* the input graph G and the input set of vertex groups Γ, but contains all the other memory consumed in the computing process.

First, we report the memory consumption of algorithms in graphs with different sizes in Figure S9, which corresponds to Figure 4 in the main contents. We also report the memory consumption of hub labels of shortest paths in graphs in Figure S9. Both GRE-PATH and the baseline algorithms use these hub labels to retrieve the shortest paths to be merged. In the full Amazon and DBLP graphs, the memories consumed by hub labels are much higher than the memories consumed by different algorithms, while in the full Movie graph, the memory consumed by hub labels is slighter smaller than that consumed by ENSteiner. The reason is that the Movie graph is much denser than the Amazon and DBLP graphs, and contains a large number of edges. These edges do not enlarge the size of hub labels significantly, due to the pruned landmark nature of the applied Pruned Landmark Labeling algorithm [7] for generating these labels. In comparison, these edges enlarge the memories consumed by ENSteiner significantly, since ENSteiner builds and records a graph by adding dummy vertices and edges into the input graph G in the computing process. Notably, as discussed in the main contents, we can also use different methods [12, 13] to generate hub labels with smaller sizes, at the cost of lower efficiencies of querying shortest paths using hub labels. We refer to [7, 12, 13] for more details of generating hub labels, and [14, 15] for the methods of updating hub labels for querying shortest paths in dynamic graphs.

We further observe that the memories consumed by DUAL and GRE-TREE are often larger than the memories consumed by PrunedDP++. This indicates that the implementations of PrunedDP++ in Line 11 of DUAL and Line 7 of GRE-TREE often consume larger amounts of memories than the baseline implementation of PrunedDP++, since PrunedDP++ is often used in Line 11 of DUAL and Line 7 of GRE-TREE to connect more groups, *e.g.,* in the initial implementation of PrunedDP++ in Line 7 of GRE-TREE, $|\Gamma'|$ often equals $|\Gamma| + 1$ and thus is often larger than $|\Gamma|$.

We also observe that, the memories consumed by GRE-PATH are always smaller than the memories consumed by the other algorithms, since it has a simple process of pushing $O(|\Gamma||V|)$ values into priority queues and then popping out these values. In comparison, the other algorithms has more complex processes. For example, DPBF, PrunedDP++, DUAL and GRE-TREE incorporate dynamic programming processes to obtain classical group Steiner trees, and may record an exponential number of trees in the dynamic programming processes, and thus consume more memories than GRE-PATH. We further note that ENSteiner may consume more memories than the other algorithms in some cases, *e.g.,* in Figure S9c (3). The reason is that, different from the other algorithms, ENSteiner builds and records a graph by adding dummy vertices and edges into the input graph G. When G is large, such as the case in Figure S9c (3), the built graph in ENSteiner consumes a lot of memory.

Subsequently, we report the memory consumption of algorithms when varying parameters in Figure S10 (notably, the memory consumption of hub labels of shortest paths in graphs do not change with parameters; thus, we do not report the memory consumption of hub labels here). We observe that the memories consumed by DPBF and PrunedDP++ may increase significantly with $|\Gamma|$, *e.g.,* in Figure S10a (3). The reason is that the number of recorded trees in the dynamic programming processes of these two algorithms are exponential to $|\Gamma|$. Nevertheless, in Figure S10a (1), the memories consumed by PrunedDP++ and GRE-TREE, which incorporates PrunedDP++, do not increase with $|\Gamma|$ as significantly as DPBF. This shows that the branch and bound techniques in PrunedDP++ effectively accelerate the dynamic programming process. On the other hand, we note that the memories consumed by different algorithms do not change with b, τ, P_{min} and P_{max}. Moreover, we observe that the memories consumed by DPBF may increase with k, since more feasible solutions are computed with the increase of k.

S12. THE SIZES OF $|V_G|$ AND $|G_{MIN}|$ IN EXPERIMENTS

We visualize the sizes of $|V_g|$ and $|g_{min}|$ in the main experiments in Figures S11-S12. In the full graphs in Figure S11c, $|V_g|$ is an order of magnitude smaller than $|V|$ for Amazon and DBLP, and is similar to $|V|$ for Movie ($|V|$ is 548552, 2497782 and 62423 for Amazon, DBLP and Movie, respectively). The reason is that the sizes of candidate groups are often small for Amazon and DBLP, but are often large for Movie, as shown in Figure 3 in the main contents. For a similar reason, $|g_{min}|$ is larger for Movie than for Amazon and DBLP. On the other hand, in Figure S12, $|V_g|$ and $|g_{min}|$ do not change much with b, τ, P_{min}, P_{max} and k. In comparison, in Figure S12a, $|V_g|$ increases with $|\Gamma|$, while $|g_{min}|$ decreases with $|\Gamma|$. The reason is that, as $|\Gamma|$ increases, more vertices are in the selected groups, and smaller groups are more likely to be selected.

S13. ADDITIONAL EXPERIMENT RESULTS WITH PAIRWISE JACCARD DISTANCES

In the main experiments, we set edge weights to 1. Here, we conduct additional experiments by setting edge weights to pairwise Jaccard distances (*e.g.,* [11, 16]), *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. We show that the key observations in the main experiments still hold here.

DUAL **is mainly of theoretical interests.** First, we show that DUAL can only be used in tiny graphs with dozens of vertices in Figure S13a, where $|V| = 45$ for "Tiny Amazon", $|V| = 90$ for "Tiny DBLP", and $|V| = 70$ for "Tiny Movie". We observe that, in Figures S13a (4-6), DUAL is significantly slower than the other algorithms. These experiments show that DUAL can only be used in tiny graphs with dozens of vertices, and is mainly of theoretical interests.

GRE-TREE **is useful when group sizes are small.** We evaluate the performance of GRE-TREE in Figure S13b, 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 is significantly slower than the other algorithms for Movie, since group sizes are large for Movie. Like the main experiments, GRE-TREE can produce better solutions than the other algorithms. Thus, it may be preferable to use GRE-TREE when group sizes are small and graph sizes are not extremely large, *e.g.,* for Amazon.

Experiment results in full graphs. We evaluate the solution quality and speed of algorithms using the full datasets in Figure S13c. We observe that, in Figures S13c (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.

In conclusion, the key observations in the main experiments, *i.e.,* (i) DUAL can only be used in tiny graphs with dozens of vertices; (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 and scales well to large graphs, still hold here.

REFERENCES FOR THE SUPPLEMENT

1. "Supplement," (2022). https://github.com/rucdatascience/PGST/blob/main/Supplement.pdf.
2. M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," Journal of the ACM **34**, 596–615 (1987).
3. 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.
4. 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.
5. R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations,* (Springer, 1972), pp. 85–103.
6. E. Cohen, E. Halperin, H. Kaplan, and U. Zwick, "Reachability and distance queries via 2-hop labels," SIAM Journal on Computing **32**, 1338–1355 (2003).
7. 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.
8. 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).
9. J. W. J. Williams, "Algorithm 232: heapsort," Commun. ACM **7**, 347–348 (1964).
10. M. L. Fredman, R. Sedgewick, D. D. Sleator, and R. E. Tarjan, "The pairing heap: A new form of self-adjusting heap," Algorithmica **1**, 111–129 (1986).
11. 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).
12. W. Li, M. Qiao, L. Qin, Y. Zhang, L. Chang, and X. Lin, "Scaling distance labeling on small-world networks," in *Proceedings of the 2019 International Conference on Management of Data,* (2019), pp. 1060–1077.

13. W. Li, M. Qiao, L. Qin, Y. Zhang, L. Chang, and X. Lin, "Scaling up distance labeling on graphs with core-periphery properties," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, (2020), pp. 1367–1381.
14. D. Ouyang, L. Yuan, L. Qin, L. Chang, Y. Zhang, and X. Lin, "Efficient shortest path index maintenance on dynamic road networks with theoretical guarantees," Proc. VLDB Endow. **13**, 602–615 (2020).
15. M. Zhang, L. Li, W. Hua, and X. Zhou, "Efficient 2-hop labeling maintenance in dynamic small-world networks," in *2021 IEEE 37th International Conference on Data Engineering*, (IEEE, 2021), pp. 133–144.
16. 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.

Table S1. Some examples of the selected PoIs (Amazon; the BFS selection method).

No.	The selected PoIs ($\vert\Gamma\vert = 5$)
1	Middle Eastern ∣ Ancient ∣ Figure Drawing ∣ History of Religion ∣ Bible
2	Toymaking ∣ Projects ∣ Crafts & Hobbies ∣ Europe ∣ Grandparenting
3	Pop Culture ∣ South Atlantic ∣ Virgin Islands ∣ Boating ∣ Hiking
4	Pulver, Liselotte ∣ Satire ∣ Specialty Stores ∣ Wilder, Billy ∣ Billy Wilder
5	Karaoke ∣ Girl Groups ∣ Music Video & Concerts ∣ General Christmas ∣ Music Outlet
6	Insurance Law ∣ Administration & Policy ∣ Business & Investing Books ∣ Business ∣ Nutshell
7	Rukeyser, Muriel ∣ Millay, Edna St Vincent ∣ Poetry, Drama & Short Stories ∣ Poets, A-Z ∣ Books on Tape
8	Thiebaud, Wayne ∣ Realism ∣ Painting ∣ Art ∣ Arts & Photography
9	Home Care ∣ Medical ∣ Hospice Care ∣ Pharmacy ∣ Spirituality
10	Rio de Janeiro ∣ Travel ∣ History ∣ Social Sciences ∣ Latin America

Table S2. Some examples of the selected PoIs (Amazon; the Simple Random selection method).

No.	The selected PoIs ($\vert\Gamma\vert = 5$)
1	Superman ∣ Johnston, Aaron Kim ∣ Stamper, J. B. ∣ Hillenbrand, Will ∣ Languages & Tools
2	Epistemology ∣ McGuire, Christine ∣ Mcquary, Chuck ∣ Kober, Jeff ∣ Gilded Age
3	Equipment ∣ Balaban, Bob ∣ Chicago Symphony Orchestra ∣ Sirtis, Marina ∣ Solti, Georg
4	MacDonald, Betty ∣ Brooke, Hillary ∣ Anatomy ∣ Winston, Don ∣ Parent Participation
5	Dorsey, Jimmy ∣ Graduate Preparation ∣ Lowry, Morton ∣ Schrader, Paul ∣ Data Structures
6	For Seniors ∣ Finch, Jon ∣ Languages & Tools ∣ Sutras ∣ Crawford, Broderick
7	Networking & System Administration ∣ Yourcenar, Marguerite ∣ Talbot, Nita ∣ Kaiser, Walter ∣ Overman, Lynne
8	Database Storage & Design ∣ Yanne, Jean ∣ Clásicos ∣ Test Guides - High School ∣ Millay, Edna St Vincent
9	Iranian Cinema ∣ Composition & Creative Writing ∣ Redding, Otis ∣ Mutter, Scott ∣ Browne, Roscoe Lee
10	Fertility Books ∣ Ziglar, Zig ∣ Morwood, Peter ∣ Smith, Dan Warry ∣ Torres, Liz

Table S3. Some examples of the selected PoIs (DBLP; the BFS selection method).

No.	The selected PoIs ($\vert\Gamma\vert = 5$)
1	Prefix ∣ Binary prefix ∣ Binary search algorithm ∣ Encoding (memory) ∣ Parallel computing
2	Solar battery ∣ Distance-vector routing protocol ∣ High voltage ∣ Collinear antenna array ∣ Renewable energy
3	Writing style ∣ Personal development ∣ Comprehension ∣ Structure (mathematical logic) ∣ Sentence
4	Fading ∣ Electromagnetic reverberation chamber ∣ Channel capacity ∣ Reverberation room ∣ Antenna (radio)
5	Psychological intervention ∣ Forensic psychiatry ∣ Anger ∣ Computer security ∣ Artificial intelligence
6	Good clinical practice ∣ Health care ∣ Medicine ∣ Terminology ∣ Torture
7	Unimodular lattice ∣ Ring of integers ∣ Lattice (group) ∣ Leech lattice ∣ Mass formula
8	Performance indicator ∣ Self-organizing network ∣ Integer programming ∣ Supply chain network ∣ Maintainability
9	Hardware obfuscation ∣ Gate array ∣ Identifier ∣ Matrix decomposition ∣ Lock (computer science)
10	Dissolved organic carbon ∣ Downwelling ∣ Biochemical oxygen demand ∣ Satellite imagery ∣ Artificial neural network

Table S4. Some examples of the selected PoIs (DBLP; the Simple Random selection method).

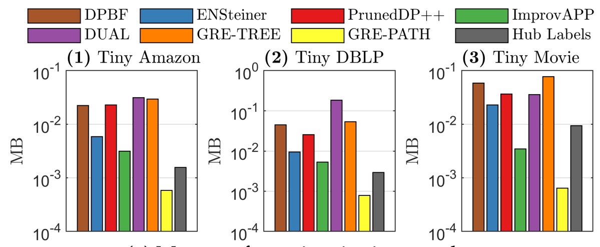
No.	The selected PoIs ($\vert\Gamma\vert = 5$)
1	Flexural strength ∣ Documentary evidence ∣ Radiation mode ∣ Voltage compensation ∣ Spectrum auction
2	Polytropic process ∣ Guanidine ∣ Environmental full-cost accounting ∣ The Imaginary ∣ Selective sweep
3	Food group ∣ Frontal lobe ∣ Psychological contract ∣ Tax reform ∣ Ammonia
4	Financialization ∣ Virtual work ∣ Organic search ∣ Fibre optic gyroscope ∣ Prime number theorem
5	Threading (protein sequence) ∣ Junior school ∣ Exome ∣ Ultra-large-scale systems ∣ Fuzzy classification
6	Hybrid security ∣ Zonal polynomial ∣ Pharmacology ∣ Gas chromatography ∣ Fermi level
7	CAAT box ∣ Color filter array ∣ Weather and climate ∣ Individual mobility ∣ Germanium
8	Inertial measurement unit ∣ Streaking ∣ Traditional education ∣ Bispectrum ∣ Honeycomb structure
9	Clicker ∣ Japanese chess ∣ Token passing ∣ T/TCP ∣ Data center network architectures
10	Toothbrush ∣ Managed care ∣ Audio power ∣ Contraction mapping ∣ Index set

Table S5. Some examples of the selected PoIs (Movie; the BFS selection method).

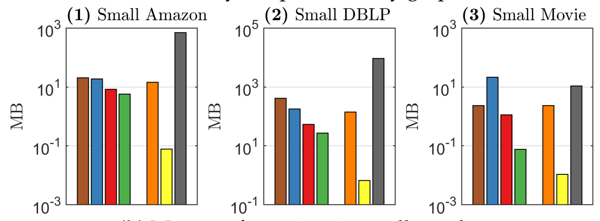
No.	The selected PoIs ($\lvert\Gamma\rvert = 5$)
1	Romance ∣ Thriller ∣ Western ∣ Musical ∣ Drama
2	Action ∣ Comedy ∣ Drama ∣ Animation ∣ Crime
3	Film-Noir ∣ Action ∣ Animation ∣ Thriller ∣ Drama
4	Musical ∣ Drama ∣ Romance ∣ Action ∣ Western
5	Crime ∣ Comedy ∣ Thriller ∣ Children ∣ Action
6	Film-Noir ∣ Action ∣ Musical ∣ Crime ∣ Western
7	Drama ∣ Comedy ∣ Western ∣ Film-Noir ∣ Crime
8	Animation ∣ Thriller ∣ Comedy ∣ Romance ∣ Drama
9	Crime ∣ Drama ∣ Film-Noir ∣ Romance ∣ Animation
10	Children ∣ Horror ∣ Action ∣ Drama ∣ Comedy

Table S6. Some examples of the selected PoIs (Movie; the Simple Random selection method).

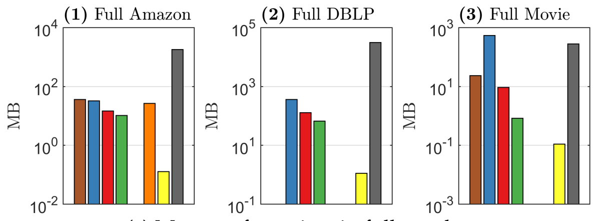
No.	The selected PoIs ($\lvert\Gamma\rvert = 5$)
1	Crime ∣ Thriller ∣ Animation ∣ Western ∣ Children
2	Drama ∣ Horror ∣ Film-Noir ∣ Crime ∣ Western
3	Musical ∣ Western ∣ Action ∣ Children ∣ Horror
4	Horror ∣ Thriller ∣ Animation ∣ Western ∣ Drama
5	Thriller ∣ Western ∣ Comedy ∣ Crime ∣ Children
6	Horror ∣ Film-Noir ∣ Comedy ∣ Drama ∣ Thriller
7	Drama ∣ Animation ∣ Thriller ∣ Film-Noir ∣ Horror
8	Thriller ∣ Animation ∣ Western ∣ Romance ∣ Action
9	Children ∣ Horror ∣ Animation ∣ Thriller ∣ Crime
10	Action ∣ Comedy ∣ Romance ∣ Musical ∣ Animation



(a) Memory footprints in tiny graphs.

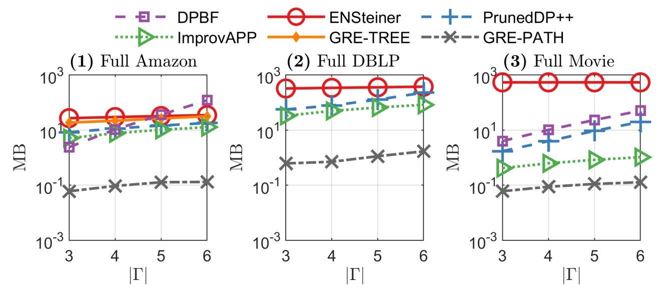


(b) Memory footprints in small graphs.

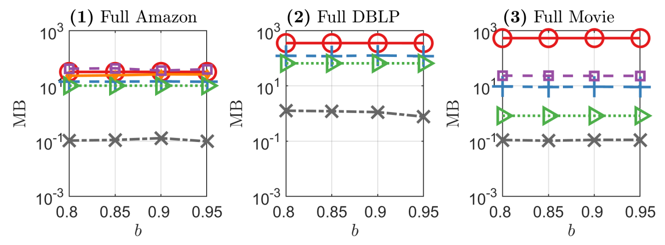


(c) Memory footprints in full graphs.

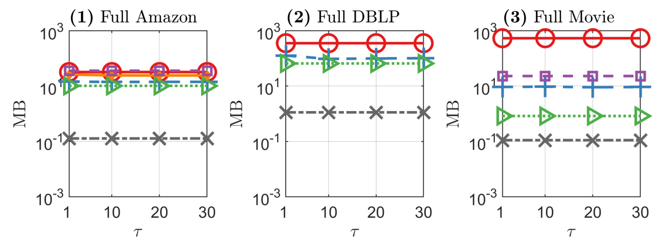
Fig. S9. Memory footprints in graphs with different sizes.



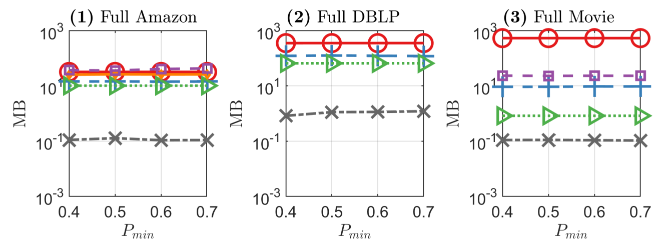
(a) Variation of the number of vertex groups: $|\Gamma|$.



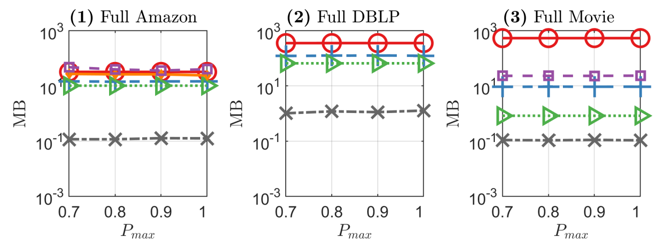
(b) Variation of the threshold value: b.



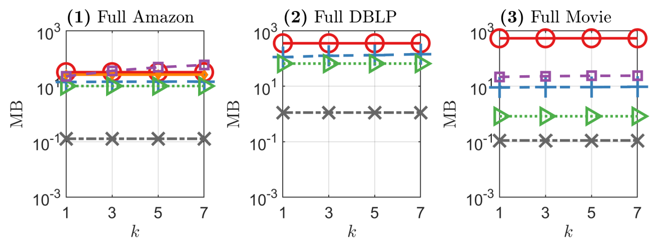
(c) Variation of the approximation parameter: τ.



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

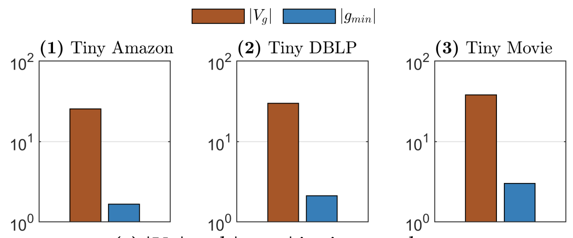


(e) Variation of the maximum positive probability value: P_{max}.

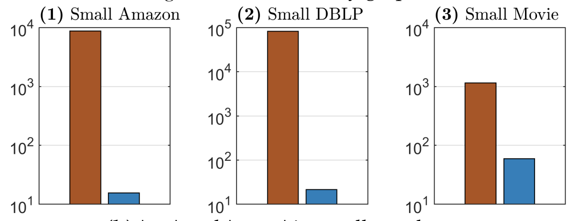


(f) Variation of the number of feasible solutions in baselines: k.

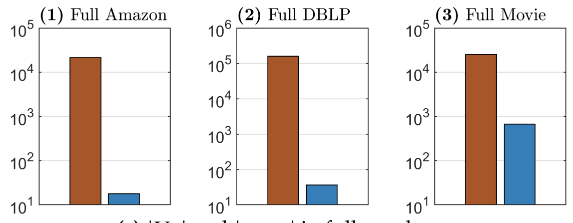
Fig. S10. Memory footprints of varying parameters.



(a) $|V_g|$ and $|g_{min}|$ in tiny graphs.

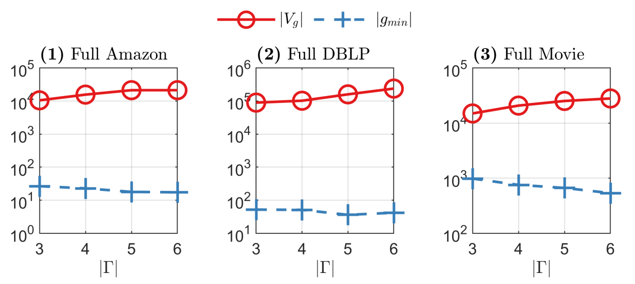


(b) $|V_g|$ and $|g_{min}|$ in small graphs.

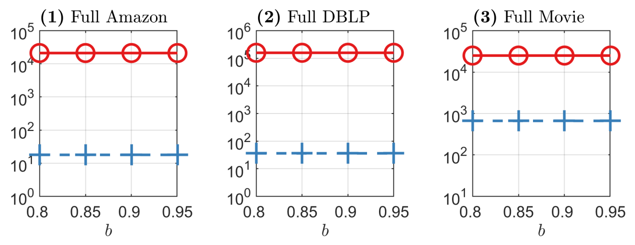


(c) $|V_g|$ and $|g_{min}|$ in full graphs.

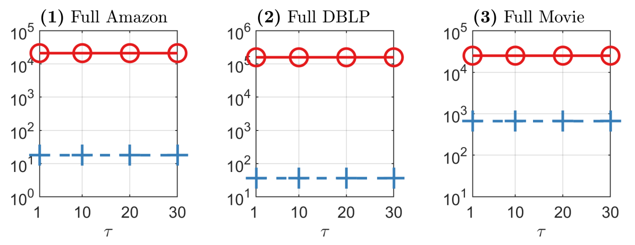
Fig. S11. $|V_g|$ and $|g_{min}|$ in graphs with different sizes.



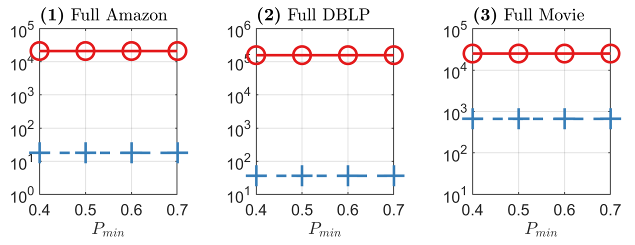
(a) Variation of the number of vertex groups: $|\Gamma|$.



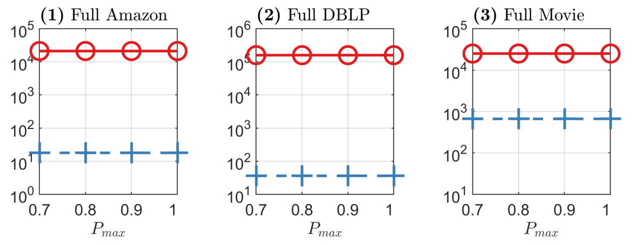
(b) Variation of the threshold value: b.



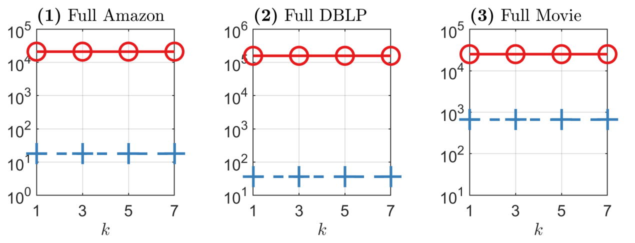
(c) Variation of the approximation parameter: τ.



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

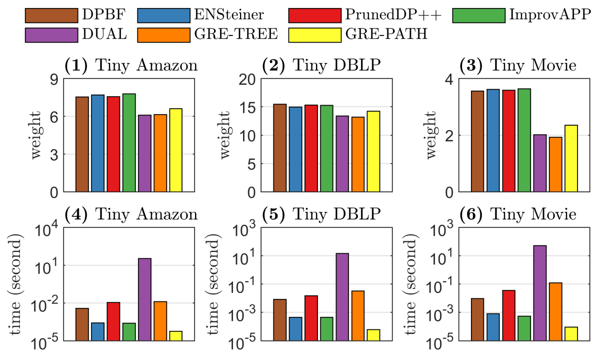


(e) Variation of the maximum positive probability value: P_{max}.

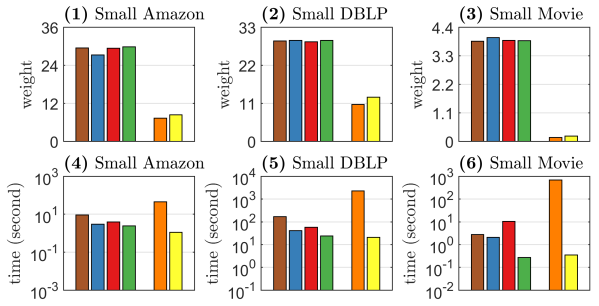


(f) Variation of the number of feasible solutions in baselines: k.

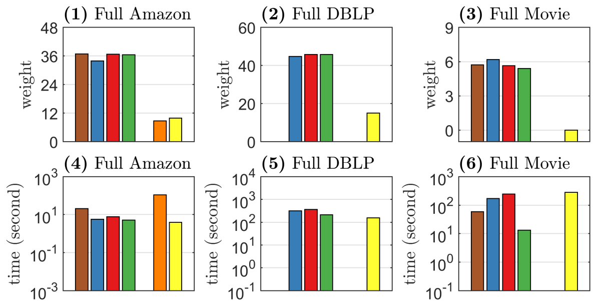
Fig. S12. $|V_g|$ and $|g_{min}|$ of varying parameters.



(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) Experiment results in full graphs.

Fig. S13. Experiment results in graphs with different sizes (pairwise Jaccard distances).