A Practical Sublinear Approximation for Group Steiner Tree

Yuxuan Yang
Nanjing University
State Key Laboratory of Novel Software Technology
Nanjing, Jiangsu, China
yxyang@smail.nju.edu.cn

Sirui Chen
Nanjing University
State Key Laboratory of Novel Software Technology
Nanjing, Jiangsu, China
siruichen@smail.nju.edu.cn

Zhuolin He
Nanjing University
State Key Laboratory of Novel Software Technology
Nanjing, Jiangsu, China
zhuolin.he@smail.nju.edu.cn

Gong Cheng*
Nanjing University
State Key Laboratory of Novel Software Technology
Nanjing, Jiangsu, China
gcheng@nju.edu.cn

ABSTRACT

The Group Steiner Tree Problem (GSTP) is widely used in graph data management and mining, yet existing algorithms trade off practical efficiency against approximation quality: efficient methods offer only linear guarantees, while those with sublinear guarantees fail to scale to large graphs. In this paper, we present MonoGST+, a novel algorithm for GSTP that breaks this trade-off by achieving a sublinear approximation while matching the running time of state-of-the-art linear-approximation solvers. Our approach extends a 2-star-based reduction to weighted set cover with a suspendable search and a monotonicity- and unimodality-aware pruning strategy to eliminate redundant computation. Experiments on multiple real-world datasets demonstrate the effectiveness and efficiency of MonoGST+, providing a practical, high-quality solution for GSTP applications.

PVLDB Reference Format:
Yuxuan Yang, Sirui Chen, Zhuolin He, and Gong Cheng. A Practical Sublinear Approximation for Group Steiner Tree. PVLDB, 19(10): 2630 - 2642, 2026.
doi:10.14778/3828612.3828620

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/nju-websoft/MonoGST.

1 INTRODUCTION

The Group Steiner Tree Problem (GSTP) is a classical NP-hard optimization problem. Given a graph with n vertices, m weighted edges, and a query consisting of g groups—each group is a subset of vertices, the goal is to find a tree that contains at least one vertex from each group while minimizing the total edge weight.

Application Scenarios. GSTP finds applications in the following representative scenarios in graph data management and mining.

Team Formation. In social networks, vertices represent experts annotated with skills, edges encode acquaintance or collaboration, and smaller edge weights indicate closer relationships. Expert team formation [20, 31] requires the selection of a team of experts whose collective skills cover a specified set while maintaining strong connectivity. In GSTP-based solutions [23, 28], each required skill corresponds to a group of experts who possess that skill, and a small-weight GST corresponds to a cohesive team. See Figure 1a.

Keyword-Based Exploration. In knowledge graphs (KGs) [15], entities are vertices, and relations are edges; relational databases are converted into graphs where tuples are vertices and foreign keys are edges, often weighted. Keyword-based exploration [21, 26] helps users extract relevant structures. GSTP-based solutions [7, 25, 32] map each input keyword to a group of matching vertices and return a minimum-weight tree covering all groups. See Figure 1b.

Knowledge Graph Summarization. To reuse a KG, its summary [21, 29] provides users with an overview of its core content. The state-of-the-art Pattern-Coverage Snippet Generation (PCSG) method [30] builds an Entity-Link Graph (ELG) from a KG and uses Entity Description Pattern (EDP) and Link Pattern (LP) to classify entities and relations into groups, respectively. This GSTP-based solution induces an optimal summary with the greatest compression from a smallest subgraph of the ELG showing all patterns. See Figure 1c.

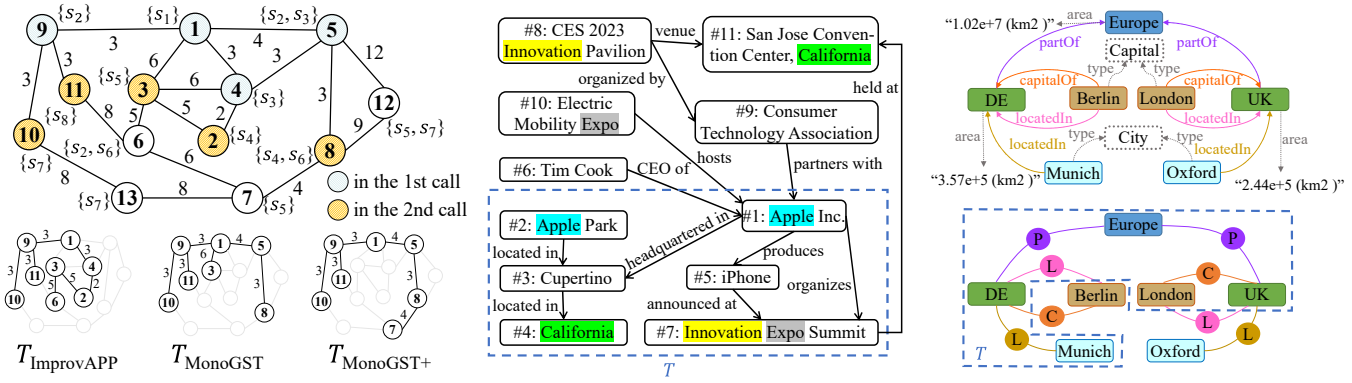
Very-Large-Scale Integration. In integrated circuit design, min-cost routing [33] connects multiple equivalent pins. GSTP-based solutions [2, 14] run on the Hanan grid reduction of a circuit net.

Motivation. GSTP is NP-hard [17]. Exact algorithms such as DPBF [7] and PrunedDP++ [22] have an exponential running time in the number of groups g and are difficult to scale to large graphs. This motivates *practical approximation algorithms*. The state of the art includes ImprovAPP [28] and the index-based KeyKG$^+$ [27]. They guarantee a $(g-1)$ approximation, i.e., an approximation ratio linear in g. Earlier algorithms such as BANKS [16], BANKS-II [19], and BLINKS [12] also provide $O(g)$-approximation guarantees. All these approaches are inspired by the so-called 1-star structure [18].

For a *more accurate approximation*, PartialOPT [28] achieves a $(g - h + 1)$ approximation with a parameter h, but its running time grows exponentially with h. There are also algorithms with polylogarithmic approximation ratios [5, 9], embedding the graph into a tree metric to solve. However, they are primarily of theoretical interest. Besides, while using the d-star structure has the potential to produce a ratio sublinear in g, experimental validation is lacking for

*Corresponding author.

Proceedings of the VLDB Endowment, Vol. 19, No. 10 ISSN 2150-8097.
doi:10.14778/3828612.3828620



(a) The top part is a social network of experts $\{1, 2, \ldots, 13\}$; each vertex is annotated with the corresponding expert's skill set, and edge weights measure relationship weakness. Each skill induces a group of experts, e.g., $s_1 \mapsto \{1\}$ and $s_2 \mapsto \{5, 6, 9\}$. Given a set of required skills as a query $Q = \{s_1, \ldots, s_8\}$, the three GSTs in the bottom part represent feasible teams with weights 24, 22, and 20 computed by ImprovAPP, MonoGST, and MonoGST+, respectively.

(b) Consider a technology-themed KG, along with an illustrative query "Apple Innovation Expo California." For this query, each keyword is mapped to a group of entity vertices in the KG: *Apple* $\mapsto \{\#1, \#2\}$, *Innovation* $\mapsto \{\#7, \#8\}$, *Expo* $\mapsto \{\#7, \#10\}$, and *California* $\mapsto \{\#4, \#11\}$. The dashed-boxed subgraph T is a feasible answer to this query; this GST simultaneously covers and connects all the four keyword-induced groups.

(c) The top part is a KG; vertices (or edges) with the same color have the same EDP (or LP) and form a group. Subdividing the colored edges by inserting vertices and removing classes (dotted boxes) and literals (quoted) yields an ELG in the bottom part. The dashed-boxed GST T covers all pattern-induced groups. Extending it back to a subgraph of the original KG by restoring the removed classes and literals, we will obtain a feasible pattern-coverage summary.

Figure 1: Application scenarios of GSTP.

large graphs, e.g., the 2-star heuristic [3, 13] requiring expensive all-pairs shortest path computation, and the reduction to the Directed Steiner Tree Problem (DSTP) [4].

In summary, existing practical methods typically have mediocre (i.e., linear) approximation guarantees, while methods with stronger (i.e., sublinear) guarantees still lack scalability validation. Table 1 summarizes their time complexities and approximation ratios.

Our Work. To bridge this performance trade-off, our goal is to develop a practical GST algorithm that is *both effective*, providing a sublinear approximation guarantee, *and efficient*, with a speed on par with the fastest index-free linear-approximation algorithms.

Specifically, we first propose a basic two-step algorithm MonoGST, which greedily approximates via 2-star based on distances calculated by Dijkstra's algorithm and non-trivially achieves an approximation ratio $O(\sqrt{g})$. Our improved algorithm MonoGST+ incorporates a *suspendable Dijkstra's algorithm* to calculate distances as needed, leverages the *monotonicity and unimodality of the greedy function* to prune the search space, and constructs a GST from a 2-star through *a greedy re-selection of paths* to enhance the approximation. Our technical contributions are as follows.

- For MonoGST, which greedily solves a weighted set cover problem to approximate an optimum 2-star and converts it into a GST, we perform a holistic analysis of this two-step approximation, producing a guaranteed ratio of $O(\sqrt{g})$.
- In MonoGST+, we embed the set cover procedure within Dijkstra's algorithm to establish a dynamic bounding mechanism and exploit the monotonicity and unimodality of the greedy function, substantially pruning the search space.

Experiments on five diverse real-world datasets demonstrate that MonoGST+ attains better solution quality than existing approximation algorithms while running as fast as the state-of-the-art index-free linear-approximation solver, thereby bridging the long-standing effectiveness-efficiency trade-off in GSTP research.

Table 1: Comparison between GST algorithms.

Algorithm	Time Complexity	Approx. Ratio
DPBF	$O(3^g n + 2^g((g + \log n)n + m))$	1
PrunedDP++	$O(3^g n + 2^g((g + \log n)n + m))$	1
Reduction to tree metric	polynomial	$O(\log^3 n \log g)$
MonoGST (ours)	$O(n(m + ng^2 + n \log n))$	$O(\sqrt{g})$
MonoGST+ (ours)	$O(n(m + (n + g)g + n \log n))$	$O(\sqrt{g})$
Reduction to DSTP	$O(n(mg + ng \log n))$	$O(\sqrt{g})$
2-star heuristic	$O(n^3 + n^2 g^2)$	$O(\sqrt{g} \ln g)$
PartialOPT	$O(n(gn + 3^h n + 2^h(m + hn + n \log n)))$	$g - h + 1$
KeyKG$^+$	$O(n^2 g + ng^3)$	$g - 1$
ImprovAPP	$O(g(m + n \log n + n^2 + n \log g))$	$g - 1$
BANKS	$O(n^2 \log n + nm)$	$O(g)$
BANKS-II	$O(n^2 \log n + nm)$	$O(g)$
BLINKS	dependent on partition	$O(g)$

2 PRELIMINARIES

2.1 Notation

In a simple undirected graph $G = \langle V, E, w \rangle$, V is a set of n vertices, $E \subseteq V \times V$ is a set of m edges, and $w : E \to R^{0+}$ is an edge weighting function. A path is an alternating sequence of vertices and edges, and its length is the total weight of its edges. The distance between two vertices u and v, $\texttt{dist}(u, v)$, is the length of a shortest u-v path $\texttt{SP}(u, v)$. We lightly abuse $\texttt{dist}(u, K)$ to represent the distance between a vertex u and a set K of vertices, i.e., the minimum distance between u and any vertex in K. Similarly, $\texttt{SP}(u, K)$ represents any shortest path between u and any vertex in K.

In a query $Q = \{K_1, K_2, \ldots, K_g\}$, each $K_i \in Q$ is a set of vertices called terminals. A Group Steiner Tree (GST) corresponding to

this query is a tree $T = \langle V_T, E_T \rangle$ containing at least one vertex in each K_i, i.e., $V_T \cap K_i \neq \emptyset$ for each K_i. The weight of this tree is $\mathtt{w}(T) = \sum_{e \in E_T} \mathtt{w}(e)$. Given G and Q, the Group Steiner Tree Problem (GSTP) is to find a minimum-weight GST. This optimization problem is NP-hard with an inapproximability of $O(\log^{2-\epsilon} g)$ [10].

2.2 2-star

A d-star is a tree composed of $d+1$ levels of vertices from the original graph [3, 13]. The first level is a root vertex r, the $(d+1)$-th level contains g terminals from distinct groups (or $g-1$ if r is chosen from a group), and the other levels contain intermediate vertices. Each edge (u, v) is weighted by $\mathtt{dist}(u, v)$ calculated in the original graph. The weight $\mathtt{w}'(R)$ of a d-star R is the sum of its edge weights.

A d-star can be converted into a GST by replacing each edge (u, v) with a shortest u-v path from the original graph. The GST converted from an optimum (i.e., minimum-$\mathtt{w}'$) d-star rooted in r yields a $2d(g/2)^{\frac{1}{d}}$ approximation to an optimum (i.e., minimum-$\mathtt{w}$) GST that contains r [13]. By computing an optimum d-star rooted in each vertex r in a particular (e.g., the smallest) group and converting all such d-stars into GSTs, their minimum weight has a guaranteed approximation ratio. Specifically, when $d = 2$, there is only one level of intermediate vertices, and an optimum 2-star approximates an optimum GST in the ratio $O(\sqrt{g})$.

2.3 Weighted Set Cover Problem

Recall the weighted set cover problem. Let $U_0 = \{a_1, a_2, \ldots, a_g\}$ be the universe and let $\mathcal{S} = \{S_1, S_2, \ldots\}$ be a family of subsets where each $S_i \subseteq U_0$ has a nonnegative weight. A feasible solution is a subfamily $\mathcal{C} \subseteq \mathcal{S}$ whose union covers U_0, i.e., $\bigcup_{S \in \mathcal{C}} S = U_0$. An optimum solution minimizes the total weight of the subsets in $\mathcal{C}$.

A classic greedy algorithm that repeatedly chooses the subset minimizing the cost-effectiveness—the ratio of its weight to the number of uncovered elements it contains—achieves an $O(\ln g)$ approximation ratio $H_g = \sum_{k=1}^{g} \frac{1}{k} \leq \ln g + 1$.

3 MonoGST: A BASIC ALGORITHM

3.1 Basic Idea

Our basic algorithm MonoGST efficiently implements the approximation scheme described in Section 2.2. In the first step, for each vertex r in the smallest group, we *greedily solve a weighted set cover problem* to approximate an optimum 2-star rooted in r with a ratio $O(\ln g)$, where each iteration *explores a pruned search space*. Combined with the ratio $O(\sqrt{g})$ to approximate an optimum GST with an optimum 2-star in the second step, an overall ratio $O(\sqrt{g} \ln g)$ might be expected. However, *our holistic analysis reduces it to* $O(\sqrt{g})$.

3.2 Algorithm Design

Algorithm 1 (MonoGST). Let K_1 be the smallest group (line 1); we will enumerate its vertices as the root r of a 2-star. We compute single-source shortest paths (SSSP) from every other group and retain the resulting shortest-path trees (line 2). Based on these distances, for each vertex $c \in V$, we construct a sorted list GI_c of group indices $\{2, 3, \ldots, g\}$ in non-decreasing order of the distance between c and each group (lines 3–6); when we cover groups through an intermediate vertex c of a 2-star, *we will only enumerate*

Algorithm 1: MonoGST

Input: Graph $G = \langle V, E, \mathtt{w} \rangle$; Query $Q = \{K_1, K_2, \ldots, K_g\}$.
Output: GST $T = \langle V_T, E_T \rangle$.

1 Swap K_1 with the smallest $K_i \in Q$;
2 Compute a shortest-path tree from each $K_i \in \{K_2, K_3, \ldots, K_g\}$;
3 **foreach** $c \in V$ **do**
4 **for** $i \leftarrow 2$ **to** g **do**
5 $GI_{c,i} \leftarrow i$;
6 Sort GI_c by $\mathtt{dist}(c, K_{GI_{c,i}})$ in non-decreasing order;
7 **foreach** $r \in K_1$ **do**
8 $T_r \leftarrow \langle \{r\}, \emptyset \rangle$;
9 $U \leftarrow \{2, 3, \ldots, g\}$;
10 Compute a shortest-path tree from r;
11 **while** $U \neq \emptyset$ **do**
12 $c_0 \leftarrow null, p_0 \leftarrow 1, S_0 \leftarrow \emptyset, f_0 \leftarrow \infty$;
13 **foreach** $c \in V$ **do**
14 **for** $p \leftarrow 2$ **to** g **do**
15 $S \leftarrow \{GI_{c,2}, GI_{c,3}, \ldots, GI_{c,p}\}$;
16 $f \leftarrow \mathtt{Cost}(r, c, p)$;
17 **if** $f < f_0$ **then**
18 $c_0 \leftarrow c, p_0 \leftarrow p, S_0 \leftarrow S, f_0 \leftarrow f$;
19 Add the vertices and edges of $\mathtt{SP}(r, c_0)$ to T_r;
20 **foreach** $j \in (S_0 \cap U)$ **do**
21 Add the vertices and edges of $\mathtt{SP}(c_0, K_j)$ to T_r;
22 $U \leftarrow U \setminus S_0$;
23 $T \leftarrow \arg\min_{r \in K_1} \mathtt{w}(T_r)$;
24 **return** T

the prefixes—rather than all subsets—of GI_c *to prune the search space.* Specifically, for each root $r \in K_1$ (line 7), we initialize a GST T_r with r, the set U of all remaining group indices as the universe to be covered, and a shortest-path tree rooted in r (lines 8–10). In each iteration until U is covered (line 11), we enumerate each intermediate vertex $c \in V$ and each prefix p of GI_c corresponding to a subset of group indices $S = \{GI_{c,2}, GI_{c,3}, \ldots, GI_{c,p}\}$, i.e., to cover $S \cap U$ through c (lines 13–15), and calculate the *cost-effectiveness* f of the pair $\langle c, p \rangle$ rooted in r as our greedy function (line 16):

$$\mathtt{Cost}(r, c, p) = \begin{cases} \infty & \text{if } \{GI_{c,2}, GI_{c,3}, \ldots, GI_{c,p}\} \cap U = \emptyset, \\ \frac{\mathtt{dist}(r,c) + \sum_{j \in (\{GI_{c,2}, GI_{c,3}, \ldots, GI_{c,p}\} \cap U)} \mathtt{dist}(c, K_j)}{|\{GI_{c,2}, GI_{c,3}, \ldots, GI_{c,p}\} \cap U|} & \text{otherwise.} \end{cases} \quad (1)$$

Note that more generally, for any subset S of group indices that may not be a prefix of GI_c, we analogously define the cost-effectiveness of the pair $\langle c, S \rangle$ rooted in r, which will be used in our proofs:

$$\mathtt{Cost}(r, c, S) = \begin{cases} \infty & \text{if } S \cap U = \emptyset, \\ \frac{\mathtt{dist}(r,c) + \sum_{j \in (S \cap U)} \mathtt{dist}(c, K_j)}{|S \cap U|} & \text{otherwise.} \end{cases} \quad (2)$$

We greedily select the best pair $\langle c_0, p_0 \rangle$ with the smallest cost-effectiveness f_0, which corresponds to a subset of group indices $S_0 = \{GI_{c_0,2}, GI_{c_0,3}, \ldots, GI_{c_0,p_0}\}$ and covers $S_0 \cap U$ (lines 12, 17–18), likely leading to a low-weight 2-star. We expand the current GST T_r by converting and merging this subtree of 2-star as described in Section 2.2: we add to T_r an r-c_0 shortest path and a shortest path

from c_0 to each group in $S_0 \cap U$ (lines 19–21); then update U (line 22). After constructing $|K_1|$ GSTs using each root in K_1, the algorithm outputs the minimum-weight GST among them (lines 23–24).

Running Example. In Figure 1a, for $Q = \{\{1\}, \{5,6,9\}, \{4,5\}, \{2,8\}, \{3,7,12\}, \{6,8\}, \{10,12,13\}, \{11\}\}$, we start with $r = 1 \in K_1$. The 1st iteration selects $c_0 = 5$ with $GI_{c_0} = \langle 2,3,4,6,5,7,8\rangle$, $p_0 = 3$, $S_0 = \{2,3\}$, $f_0 = \frac{\texttt{dist}(1,5)+\texttt{dist}(5,5)+\texttt{dist}(5,5)}{|\{2,3\}\cap\{2,3,4,5,6,7,8\}|} = 2$, adds paths $1-5$, 5, 5, and updates $U = \{4,5,6,7,8\}$. The 2nd iteration selects $c_0 = 8$ with $GI_{c_0} = \langle 4,6,2,3,5,7,8\rangle$, $p_0 = 5$, $S_0 = \{4,6,2,3\}$, $f_0 = \frac{\texttt{dist}(1,8)+\texttt{dist}(8,8)+\texttt{dist}(8,8)}{|\{4,6,2,3\}\cap\{4,5,6,7,8\}|} = 3.5$, adds paths $1-5-8$, 8, 8, and updates $U = \{5,7,8\}$. The 3rd iteration selects $c_0 = 9$ with $GI_{c_0} = \langle 2,7,8,3,4,5,6\rangle$, $p_0 = 4$, $S_0 = \{2,7,8\}$, $f_0 = \frac{\texttt{dist}(1,9)+\texttt{dist}(9,10)+\texttt{dist}(9,11)}{|\{2,7,8\}\cap\{5,7,8\}|} = 4.5$, adds paths $1-9$, $9-10$, $9-11$, and updates $U = \{5\}$. The 4th iteration selects $c_0 = 1$ with $GI_{c_0} = \langle 2,3,4,5,7,8,6\rangle$, $p_0 = 5$, $S_0 = \{2,3,4,5\}$, $f_0 = \frac{\texttt{dist}(1,1)+\texttt{dist}(1,3)}{|\{2,3,4,5\}\cap\{5\}|} = 6$, adds paths 1, $1-3$, and updates $U = \emptyset$, completing set cover.

3.3 Algorithm Analysis

It is straightforward that MonoGST produces a feasible GST. In the following, we analyze its approximation ratio and time complexity.

3.3.1 Approximation Ratio. Finding an optimum 2-star rooted in r is NP-hard [3] and can be approximated in the ratio $O(\ln g)$ using the classic greedy algorithm to solve a weighted set cover problem with $U = \{2,3,\ldots,g\}$ as the universe and a family of subsets where each corresponding to a pair $\langle c, S\rangle \in V \times 2^U$ covers S through an intermediate vertex c and has weight $\texttt{dist}(r,c) + \sum_{j\in S} \texttt{dist}(c, K_j)$. In MonoGST, we essentially prune the search space of S from 2^U to the prefixes of GI_c (lines 14–15). The following lemma shows that *this pruning maintains the quality of the 2-star found.*

Lemma 1. *Pruning the search space of S from 2^U to the prefixes of GI_c maintains the weight of the 2-star found greedily.*

Proof. Since GI_c is sorted by $\texttt{dist}(c, K_{GI_{c,i}})$ in non-decreasing order, the cost-effectiveness f of S in MonoGST (lines 15–16) is not greater than that of any $S' \in 2^U$ subject to $|S' \cap U| = |S \cap U|$. Then it suffices to enumerate all prefixes of GI_c instead of 2^U. □

Therefore, *in the following analysis we will ignore this pruning.* Given a query Q, let T^*_Q be an optimum GST and $R^*_{Q,r}$ an optimum 2-star rooted in r. Let T_Q be the GST returned by MonoGST which is converted from the 2-star R_Q. The conversion process indicates $\texttt{w}(T_Q) \le \texttt{w}'(R_Q)$. The following lemma is well-known [13].

Lemma 2. *If any T^*_Q contains r, then $\texttt{w}'(R^*_{Q,r}) \le 2\sqrt{2g} \cdot \texttt{w}(T^*_Q)$.*

Combining this ratio $O(\sqrt{g})$ with $O(\ln g)$ for approximating an optimum 2-star yields $O(\sqrt{g}\ln g)$. However, this result is loose. Motivated by the view that MonoGST solves $|K_1|$ *reduced* GSTP instances, each with a distinct $\{r\} \subseteq K_1$ as its own K_1, we prove the following lemma that *leads to a better approximation ratio $O(\sqrt{g})$.* The key is to *charge each iteration recursively with a decreased number of uncovered groups* (i.e., i in the following analysis).

Lemma 3. *If $K_1 = \{r\}$, then $\texttt{w}'(R_{\{\{r\}\}\cup Q}) \le 4\sum_{i=1}^{|Q|} i^{-\frac{1}{2}} \cdot \texttt{w}(T^*_{\{\{r\}\}\cup Q})$.*

Proof. Given the query $\{\{r\}\}\cup Q$, let $\langle c_0, p_0\rangle$ be the pair selected in the first iteration of MonoGST that covers $S_0 \cap U = S_0$, that is,

$$\texttt{w}'\big(R_{\{\{r\}\}\cup Q}\big) = |S_0|\texttt{Cost}(r, c_0, S_0) + \texttt{w}'\big(R_{\{\{r\}\}\cup(Q\setminus\{K_j | j\in S_0\})}\big)\,. \quad (3)$$

As the first greedy choice, $\texttt{Cost}(r, c_0, S_0)$ also serves as a lower bound on the weight per covered group needed for a 2-star rooted in r to cover the groups in Q. Therefore,

$$\texttt{Cost}(r, c_0, S_0) \le \frac{\texttt{w}'\big(R^*_{\{\{r\}\}\cup Q, r}\big)}{|Q|} \le \frac{2\sqrt{2(|Q|+1)}\texttt{w}\big(T^*_{\{\{r\}\}\cup Q}\big)}{|Q|}\,, \quad (4)$$

where the second inequality follows from Lemma 2, so

$$\begin{aligned} |S_0|\texttt{Cost}(r, c_0, S_0) &\le 2\sqrt{2}\texttt{w}\big(T^*_{\{\{r\}\}\cup Q}\big)\frac{\sqrt{|Q|+1}}{|Q|}|S_0| \\ &\le 2\sqrt{2}\texttt{w}\big(T^*_{\{\{r\}\}\cup Q}\big)\sum_{i=|Q|-|S_0|+1}^{|Q|}\frac{\sqrt{i+1}}{i}\,. \end{aligned} \quad (5)$$

Expanding the recurrence of Equation (3) and substituting Equation (5) into it, we obtain

$$\texttt{w}'\big(R_{\{\{r\}\}\cup Q}\big) \le 2\sqrt{2}\texttt{w}\big(T^*_{\{\{r\}\}\cup Q}\big)\sum_{i=1}^{|Q|}\frac{\sqrt{i+1}}{i} \le 4\texttt{w}\big(T^*_{\{\{r\}\}\cup Q}\big)\sum_{i=1}^{|Q|} i^{-\frac{1}{2}}\,. \quad (6)$$

□

Theorem 1. *MonoGST has an approximation ratio $O(\sqrt{g})$.*

Proof. Let R'_Q be the smallest-weight 2-star found in MonoGST. Since MonoGST can be viewed as solving $|K_1|$ reduced versions of the problem, each taking a distinct $r \in K_1$, with $\{\{r\}\} \cup (Q \setminus \{K_1\})$ as its own Q and $\{r\}$ as its own K_1, we have

$$\texttt{w}'(R'_Q) = \min_{r\in K_1} \texttt{w}'\big(R_{\{\{r\}\}\cup(Q\setminus\{K_1\})}\big)\,. \quad (7)$$

Let T'_Q be the GST converted from R'_Q. We have

$$\texttt{w}(T_Q) \le \texttt{w}(T'_Q) \le \texttt{w}'(R'_Q)\,. \quad (8)$$

Combining Equations (8)(7) and Lemma 3, we obtain

$$\frac{\texttt{w}(T_Q)}{\texttt{w}(T^*_Q)} \le \frac{\min_{r\in K_1} 4\left(\sum_{i=1}^{g-1} i^{-\frac{1}{2}}\right)\texttt{w}\big(T^*_{\{\{r\}\}\cup(Q\setminus\{K_1\})}\big)}{\min_{r\in K_1}\texttt{w}\big(T^*_{\{\{r\}\}\cup(Q\setminus\{K_1\})}\big)} = 4\left(\sum_{i=1}^{g-1} i^{-\frac{1}{2}}\right) \in O(\sqrt{g})\,. \quad (9)$$

□

3.3.2 Time Complexity. Let $|K_1| = n_1$. There are $(g-1+n_1)$ SSSP computations (lines 2, 10), each taking $O(m + n\log n)$ time using Dijkstra's algorithm with a Fibonacci heap. The construction and sorting of GI_c for all $c \in V$ (lines 3–6) takes $O(ng\log g)$ time. Cost (line 16) is calculated $O(n_1 gng)$ times, each taking a constant time calculated incrementally using distances from SSSP. The construction of T_r for all $r \in K_1$ (lines 19–21) takes $O(n_1 ng)$ time.

Overall, the time complexity of MonoGST is in $O((m+n\log n)(g+n_1) + nn_1g^2)$. Since n_1 and g are often very small in practical applications, MonoGST scales almost linear in graph size. When n_1 is close to n, the time complexity becomes $O(n(m + ng^2 + n\log n))$.

Note that in multi-query scenarios, intermediate results such as shortest-path trees computed by SSSP can be cached and reused to reduce the time cost of subsequent queries.

4 MonoGST+: AN IMPROVED ALGORITHM

4.1 Basic Idea

Based on MonoGST, our improved algorithm MonoGST+ is designed to increase efficiency and effectiveness in three aspects.

First, MonoGST performs $|K_1|$ SSSP computations (line 10) using Dijkstra's algorithm. To accelerate them, MonoGST+ interleaves each SSSP computation with its subsequent set cover procedure through *a suspendable Dijkstra's algorithm that proceeds in a lazy manner* (Section 4.3), driven by the needs of the set cover stage to prune unneeded distance calculations; in each iteration of the set cover procedure, Dijkstra's algorithm is resumed and then suspended once the next distance reaches an upper bound that is dynamically derived from previous iterations, ensuring that distances exceeding this bound are not needed for the current iteration.

Second, we prove that in MonoGST, for each intermediate vertex c (line 13), *the prefix p that minimizes* $\texttt{Cost}(r, c, p)$ *is monotonically non-decreasing over iterations of the set cover procedure* (line 11), and *in each iteration,* $\texttt{Cost}(r, c, p)$ *has only one local optimum as p increases* (Section 4.4). Therefore, unlike MonoGST which repeatedly enumerates all possible prefixes (line 14), MonoGST+ prunes the prefixes and reduces time complexity by a factor of g.

Third, MonoGST converts a 2-star into a GST as described in Section 2.2, replacing each edge of the 2-star with a shortest path from the original graph (lines 19–21). For overlapping paths, their total edge weight is overestimated by the weight of 2-star, leading to suboptimal path selection. To address it, MonoGST+ *connects the selected intermediate vertices to terminals through greedily re-selected paths* (Section 4.5), which could result in a smaller total weight.

4.2 Algorithm Design

Algorithm 2 (MonoGST+). As in MonoGST, we identify the smallest group K_1 (line 1), compute SSSP (line 2), and construct a sorted list GI_c for each $c \in V$ (lines 3–6). Further, for each $1 \le k \le g-1$, we calculate the minimum sum D_k^{Sum} of distances corresponding to the first k entries among all such lists (lines 7–8), used to calculate the distance upper bound to suspend Dijkstra's algorithm. For each root $r \in K_1$ of a 2-star (line 9), we initialize the set U of all remaining group indices to be covered (line 10), the *dynamic distance upper bound* $\overline{d}$ (line 11), and an empty set CP of $\langle c, p\rangle$ pairs (line 11)—at most one pair for each intermediate vertex c—such that $\langle c, p\rangle$ has the current smallest cost-effectiveness for c defined by Equation (1) given the distances already calculated by the suspendable Dijkstra's algorithm. We also initialize an empty set C_0 of current intermediate vertices in the 2-star (line 11), used to construct a GST. Then the algorithm *alternates between iterations of Dijkstra's algorithm and the set cover procedure.* Specifically, we initialize a min heap of distance-vertex pairs to support Dijkstra's algorithm (line 12). In each iteration until U is covered (line 13), we call SuspendableDijkstra (detailed in Section 4.3) to *resume Dijkstra's algorithm* to expand CP with CP^{New}, a set of $\langle c, p\rangle$ pairs where $\texttt{dist}(r, c)$ is newly calculated, and possibly tighten $\overline{d}$ (lines 14–15). After *suspending Dijkstra's algorithm* by $\overline{d}$, we greedily select the best pair $\langle c_0, p_0\rangle \in CP$ that has the smallest cost-effectiveness f_0 (lines 16–21); *we will define $\overline{d}$ in a way that unvisited vertices cannot contribute any smaller cost-effectiveness.* We add c_0 to C_0 and update U (lines 22–23). Since U is involved in calculating the cost-effectiveness by Equation (1) and this set has changed, to maintain the property of CP, for each $\langle c, p\rangle \in CP$ we call UpdatePrefix (detailed in Section 4.4) to update p to have the current smallest cost-effectiveness f for c (lines 24–26). We use their smallest value f_0 (line 27) and D^{Sum} to possibly relax $\overline{d}$ (line 28, detailed in Section 4.3) to resume Dijkstra's algorithm in the next iteration. When U is covered, we call ConstructTree (detailed in Section 4.5) to construct a GST with the root r and intermediate vertices C_0 (line 29). After constructing $|K_1|$ GSTs using each root in K_1, the algorithm outputs the minimum-weight GST among them (lines 30–31).

Algorithm 2: MonoGST+

Input: Graph $G = \langle V, E, \texttt{w}\rangle$; Query $Q = \{K_1, K_2, \ldots, K_g\}$.
Output: GST $T = \langle V_T, E_T\rangle$.

1 Swap K_1 with the smallest $K_i \in Q$;
2 Compute a shortest-path tree from each $K_i \in \{K_2, K_3, \ldots, K_g\}$;
3 **foreach** $c \in V$ **do**
4 **for** $i \leftarrow 2$ **to** g **do**
5 $GI_{c,i} \leftarrow i$;
6 Sort GI_c by $\texttt{dist}(c, K_{GI_{c,i}})$ in non-decreasing order;
7 **foreach** $k \leftarrow 1$ **to** $g-1$ **do**
8 $D_k^{\text{Sum}} \leftarrow \min_{c \in V} \sum_{i=2}^{k+1} \texttt{dist}(c, K_{GI_{c,i}})$;
9 **foreach** $r \in K_1$ **do**
10 $U \leftarrow \{2, 3, \ldots, g\}$;
11 $\overline{d} \leftarrow \infty, CP \leftarrow \emptyset, C_0 \leftarrow \emptyset$;
12 $pq \leftarrow$ a min heap with $\{\langle 0, r\rangle\} \cup \{\langle \infty, v\rangle \mid v \in (V \setminus \{r\})\}$;
13 **while** $U \neq \emptyset$ **do**
14 $CP^{\text{New}}, \overline{d} \leftarrow$ SuspendableDijkstra$(pq, \overline{d}, r, g, U, GI, D^{\text{Sum}})$;
15 $CP \leftarrow CP \cup CP^{\text{New}}$;
16 $c_0 \leftarrow null, p_0 \leftarrow 1, S_0 \leftarrow \emptyset, f_0 \leftarrow \infty$;
17 **foreach** $\langle c, p\rangle \in CP$ **do**
18 $S \leftarrow \{GI_{c,2}, GI_{c,3}, \ldots, GI_{c,p}\}$;
19 $f \leftarrow \texttt{Cost}(r, c, p)$;
20 **if** $f < f_0$ **then**
21 $c_0 \leftarrow c, p_0 \leftarrow p, S_0 \leftarrow S, f_0 \leftarrow f$;
22 $C_0 \leftarrow C_0 \cup \{c_0\}, f_0 \leftarrow \infty$;
23 $U \leftarrow U \setminus S_0$;
24 **foreach** $\langle c, p^{\text{Old}}\rangle \in CP$ **do**
25 $p, f \leftarrow$ UpdatePrefix$(c, p^{\text{Old}}, r, g, U, GI)$;
26 $CP \leftarrow (CP \setminus \{\langle c, p^{\text{Old}}\rangle\}) \cup \{\langle c, p\rangle\})$;
27 $f_0 \leftarrow \min\{f_0, f\}$;
28 $\overline{d} \leftarrow \max_{k=1}^{|U|}(kf_0 - D_k^{\text{Sum}})$;
29 $T_r \leftarrow$ ConstructTree(r, C_0, Q);
30 $T \leftarrow \arg\min_{r \in K_1} \texttt{w}(T_r)$;
31 **return** T

Running Example. Reconsider the GSTP instance in Section 3.2. With $D^{\text{Sum}} = \langle 0, 0, 3, 6, 10, 19, 32\rangle$, we start with $r = 1 \in K_1$. In the 1st iteration, CP is expanded with $CP^{\text{New}} = \{\langle 1, 3\rangle, \langle 4, 3\rangle, \langle 9, 4\rangle, \langle 5, 3\rangle\}$

and $\overline{d}$ is tightened to 4. We add $c_0 = 5$ to C_0, update $U = \{4, 5, 6, 7, 8\}$, update CP to $\{\langle 1, 4\rangle, \langle 4, 4\rangle, \langle 9, 4\rangle, \langle 5, 5\rangle\}$, and relax $\overline{d}$ to 12.5. In the 2nd iteration, CP is expanded with $CP^{\mathsf{New}} = \{\langle 2, 5\rangle, \langle 3, 5\rangle, \langle 10, 4\rangle, \langle 11, 4\rangle, \langle 8, 3\rangle\}$ and $\overline{d}$ is tightened to 8. We add $c_0 = 8$ to C_0, update $U = \{5, 7, 8\}$, update CP to $\{\langle 1, 7\rangle, \langle 4, 8\rangle, \langle 9, 4\rangle, \langle 5, 8\rangle, \langle 2, 6\rangle, \langle 3, 6\rangle, \langle 10, 4\rangle, \langle 11, 4\rangle, \langle 8, 7\rangle\}$, and relax $\overline{d}$ to 10.5. In the 3rd iteration, CP and $\overline{d}$ remain. We add $c_0 = 9$ to C_0, update $U = \{5\}$, update CP to $\{\langle 1, 8\rangle, \langle 4, 8\rangle, \langle 9, 8\rangle, \langle 5, 8\rangle, \langle 2, 6\rangle, \langle 3, 6\rangle, \langle 10, 8\rangle, \langle 11, 8\rangle, \langle 8, 7\rangle\}$, and update $\overline{d} = 6$. In the 4th iteration, CP and $\overline{d}$ remain. We add $c_0 = 1$ to C_0 and update $U = \emptyset$, so the set cover procedure is completed. The resulting GST is constructed with $r = 1$ and $C_0 = \{5, 8, 9, 1\}$.

4.3 Suspendable Dijkstra's Algorithm

Dijkstra's algorithm calculates distances in non-decreasing order. In MonoGST+, each call to SuspendableDijkstra (line 14) resumes Dijkstra's algorithm, which will be suspended once the next distance reaches $\overline{d}$. We describe SuspendableDijkstra in Algorithm 3 and will later elaborate on the definition and computation of $\overline{d}$.

Algorithm 3 (SuspendableDijkstra). Building on Dijkstra's algorithm (lines 2–3, 6, 10), we suspend it when the next distance d reaches the upper bound $\overline{d}$ (lines 4–5). For each vertex c that is newly visited since the previous suspension, we call UpdatePrefix (detailed in Section 4.4) to find the prefix p such that $\langle c, p\rangle$ has the current smallest cost-effectiveness f for c (line 7). We return all such $\langle c, p\rangle$ pairs CP^{New} (lines 1, 8, 11) and an updated $\overline{d}$ possibly tightened based on each f using D^{Sum} (line 9, detailed later).

Running Example. Within the execution of MonoGST+ in Section 4.2, we elaborate on its first call to SuspendableDijkstra with an initial pq, $\overline{d} = \infty$, and $r = 1$. The 1st visited vertex is 1 with $d = 0$, $p = 3$, and $f = 3$, tightening $\overline{d}$ to 6. The 2nd visited vertex is 4 with $d = 3$, $p = 3$, and $f = 2.5$, tightening $\overline{d}$ to 5. The 3rd visited vertex is 9 with $d = 3$, $p = 4$, and $f = 3$, not further tightening $\overline{d}$. The 4th visited vertex is 5 with $d = 4$, $p = 3$, and $f = 2$, tightening $\overline{d}$ to 4. The next vertex is 2 with $d = 5$ exceeding the current $\overline{d}$, so Dijkstra's algorithm is suspended, returning $CP^{\mathsf{New}} = \{\langle 1, 3\rangle, \langle 4, 3\rangle, \langle 9, 4\rangle, \langle 5, 3\rangle\}$ and $\overline{d} = 4$. The vertices visited in the 2nd call to SuspendableDijkstra are annotated in Figure 1a. No vertex is visited in the 3rd and 4th calls.

Definition of $\overline{d}$. In MonoGST+, let $\langle c_0, p_0\rangle$ be the best pair that has the smallest cost-effectiveness f_0 selected from the current CP (lines 16–21). To ensure the correctness of using the suspendable Dijkstra's algorithm, we need to define $\overline{d}$ such that unvisited vertices cannot contribute any smaller cost-effectiveness. In other words, *every unvisited vertex c must not satisfy*:

$$\min_{\emptyset \subset S \subseteq U} \mathtt{Cost}(r, c, S) < f_0 \,. \tag{10}$$

The following lemma derives a necessary condition for this.

Lemma 4. *A vertex $c \in V$ satisfies Equation* (10) *only if*

$$\mathtt{dist}(r, c) < \max_{D \subseteq D_c \text{ s.t. } |D| \le |U|} \sum_{d \in D} (f_0 - d) \,, \tag{11}$$

where D_c is a multiset:

$$D_c = \{\mathtt{dist}(c, K_i) \mid K_i \in \{K_2, K_3, \ldots, K_g\}\} \,. \tag{12}$$

Algorithm 3: SuspendableDijkstra

Input: Min heap pq of distance-vertex pairs; Distance upper bound $\overline{d}$; Root r; Number g of groups; Set U of all remaining group indices; Sorted lists GI of group indices; List D^{Sum} of sum of distances.
Output: Set CP^{New} of vertex-prefix pairs; Updated distance upper bound $\overline{d}$.

1 $CP^{\mathsf{New}} \leftarrow \emptyset$;
2 **while** $pq \neq \emptyset$ **do**
3 $\quad \langle d, c\rangle \leftarrow pq$.top();
4 $\quad$ **if** $d \ge \overline{d}$ **then**
5 $\quad\quad$ **break**;
6 $\quad pq$.pop();
7 $\quad p, f \leftarrow \mathtt{UpdatePrefix}(c, 2, r, g, U, GI)$;
8 $\quad CP^{\mathsf{New}} \leftarrow CP^{\mathsf{New}} \cup \{\langle c, p\rangle\}$;
9 $\quad \overline{d} \leftarrow \min\{\overline{d}, \max_{k=1}^{|U|}(kf - D_k^{\mathsf{Sum}})\}$;
10 $\quad$ Relax c's incident edges and update pq;
11 **return** $CP^{\mathsf{New}}, \overline{d}$;

Proof. We prove by contradiction. Suppose Equation (11) does not hold for some $c \in V$ that satisfies Equation (10), so we choose $S' = \arg\min_{\emptyset \subset S \subseteq U} \mathtt{Cost}(r, c, S)$ and have $\mathtt{Cost}(r, c, S') < f_0$. We define a multiset $D' = \{\mathtt{dist}(c, K_j) \mid j \in S'\}$. Therefore, we obtain

$$\begin{aligned} \mathtt{dist}(r, c) &\ge \max_{D \subseteq D_c \text{ s.t. } |D| \le |U|} \sum_{d \in D} (f_0 - d) \ge \sum_{d \in D'} (f_0 - d) \\ &= \sum_{j \in S'} (f_0 - \mathtt{dist}(c, K_j)) = |S'| f_0 - \sum_{j \in S'} \mathtt{dist}(c, K_j) \,. \end{aligned} \tag{13}$$

By manipulating this equation, we obtain

$$f_0 \le \frac{\mathtt{dist}(r, c) + \sum_{j \in S'} \mathtt{dist}(c, K_j)}{|S'|} = \mathtt{Cost}(r, c, S') \,, \tag{14}$$

which contradicts $\mathtt{Cost}(r, c, S') < f_0$. □

With this necessary condition, we define

$$\overline{d} = \max_{c \in V} \max_{D \subseteq D_c \text{ s.t. } |D| \le |U|} \sum_{d \in D} (f_0 - d) \,. \tag{15}$$

Any unvisited vertex c satisfies $\mathtt{dist}(r, c) \ge \overline{d}$, so Equation (11) does not hold and Equation (10) is not satisfied, ensuring the correctness of using SuspendableDijkstra in MonoGST+.

Efficient Computation of $\overline{d}$. We need to update $\overline{d}$ when CP is expanded so that the current smallest cost-effectiveness f_0 may change. This occurs in MonoGST+ where CP is expanded with CP^{New} (line 15) and in SuspendableDijkstra where CP^{New} is expanded (line 8) and should be incorporated into CP. To calculate $\overline{d}$, we transform Equation (15) into an efficiently computable form.

Specifically, in MonoGST+, we calculate $\overline{d}$ using D^{Sum} (line 28). The following theorem proves that this transformation is equivalent.

Theorem 2. *For any cost-effectiveness f_0, we have*

$$\max_{c \in V} \max_{D \subseteq D_c \text{ s.t. } |D| \le |U|} \sum_{d \in D} (f_0 - d) = \max_{k=1}^{|U|} (k f_0 - D_k^{\mathsf{Sum}}) \,. \tag{16}$$

Proof. We enumerate all possible values of $|D|$ and then use the definitions of GI and D^{Sum}:

$$\begin{aligned}
&\max_{c\in V}\max_{D\subseteq D_c \text{ s.t. } |D|\le|U|}\sum_{d\in D}(f_0-d)\\
=&\max_{k=1}^{|U|}\max_{c\in V}\max_{D\subseteq D_c \text{ s.t. } |D|=k}\sum_{d\in D}(f_0-d)\\
=&\max_{k=1}^{|U|}\left(kf_0-\min_{c\in V}\min_{D\subseteq D_c \text{ s.t. } |D|=k}\sum_{d\in D}d\right)\\
=&\max_{k=1}^{|U|}\left(kf_0-\min_{c\in V}\sum_{i=2}^{k+1}\mathtt{dist}(c,K_{GI_{c,i}})\right)=\max_{k=1}^{|U|}\left(kf_0-D_k^{\mathsf{Sum}}\right).
\end{aligned}\tag{17}$$

□

Note $\overline{d}$ increases monotonically in f_0. In `SuspendableDijkstra`, when we visit a new vertex c and find the current smallest cost-effectiveness f for c (line 7), f may be smaller than the smallest cost-effectiveness from the current CP, so we use it to possibly tighten $\overline{d}$ (line 9) and suspend Dijkstra's algorithm earlier.

4.4 Prefix Pruning

In `MonoGST+` (line 25) and `SuspendableDijkstra` (line 7), each call to `UpdatePrefix` finds the prefix p for a given vertex c such that $\langle c,p\rangle$ has the current smallest cost-effectiveness f for c. In `UpdatePrefix`, we exploit the monotonicity and unimodality of the greedy function `Cost` to prune the prefixes that need to be enumerated. We describe `UpdatePrefix` in Algorithm 4 and will later prove the monotonicity and unimodality of `Cost`.

Algorithm 4 (`UpdatePrefix`). When called in `MonoGST+`, we receive a vertex c and a prefix p^{Old} such that $\langle c,p^{\text{Old}}\rangle$ had—before U changes—the smallest cost-effectiveness for c; when called in `SuspendableDijkstra`, we receive a newly visited vertex c and a trivially initialized $p^{\text{Old}}=2$. We will prove the *monotonicity* of `Cost`, i.e., *the prefix p that minimizes $\mathtt{Cost}(r,c,p)$ is monotonically non-decreasing over iterations of the set cover procedure*, so instead of repeatedly enumerating p from 2 as in `MonoGST`, we enumerate from p^{Old} (lines 1–2, 6–7). We will also prove the *unimodality* of `Cost`, i.e., *$\mathtt{Cost}(r,c,p)$ has only one local optimum as p increases within each iteration of the set cover procedure*, so instead of enumerating p to g as in `MonoGST`, we will terminate the enumeration and return the current p and the corresponding f if a larger cost-effectiveness is found for the next prefix $p+1$ (lines 3–5, 8).

Running Example. Within the execution of `MonoGST+` in Section 4.2, consider $r=1$ and $c=5$. As U shrinks over iterations from $\{2,3,4,5,6,7,8\}$ to $\{4,5,6,7,8\}$, $\{5,7,8\}$, and finally $\{5\}$, the prefix p that minimizes $\mathtt{Cost}(1,5,p)$ monotonically increases from 3 to 5 and finally 8. When $U=\{2,3,4,5,6,7,8\}$, as p increases, f decreases from 4 to the local optimum 2 and then increases to 2.33.

Unimodality of `Cost`. The following theorem proves the (weak) unimodality of $\mathtt{Cost}(r,c,p)$: it has only one local optimum as p increases, i.e., it is first monotonically non-increasing and then monotonically non-decreasing.

Theorem 3. *For any fixed r,c,U, there exists a prefix p^* such that*

$$\begin{aligned}
&\mathtt{Cost}(r,c,2)\ge\mathtt{Cost}(r,c,3)\ge\cdots\ge\mathtt{Cost}(r,c,p^*),\\
&\mathtt{Cost}(r,c,p^*)\le\mathtt{Cost}(r,c,p^*+1)\le\cdots\le\mathtt{Cost}(r,c,g).
\end{aligned}\tag{18}$$

Algorithm 4: `UpdatePrefix`

Input: Vertex c; Prefix p^{Old} such that $\langle c,p^{\text{Old}}\rangle$ had the smallest cost-effectiveness for c; Root r; Number g of groups; Set U of all remaining group indices; Sorted lists GI of group indices.
Output: Updated prefix p; Corresponding cost-effectiveness f.

```
1 p ← p^Old;
2 f ← Cost(r, c, p);
3 while p < g do
4     if Cost(r, c, p + 1) > f then
5         break;
6     p ← p + 1;
7     f ← Cost(r, c, p);
8 return p, f;
```

Proof. If $GI_{c,p+1}\notin U$, we will have $\mathtt{Cost}(r,c,p)=\mathtt{Cost}(r,c,p+1)$ by Equation (1); the proof is trivial over such intervals. Therefore, without loss of generality, we assume $GI_{c,p}\in U$ for all $2\le p\le g$.

Define $\Delta\mathtt{Cost}(p)=\mathtt{Cost}(r,c,p+1)-\mathtt{Cost}(r,c,p)$. It suffices to prove that $\Delta\mathtt{Cost}(p)$ changes sign at most once (from negative to positive) as p increases from 2 to $g-1$. In fact, with Equation (1) and the above assumption, we obtain

$$\begin{aligned}
\Delta\mathtt{Cost}(p)&=\mathtt{Cost}(r,c,p+1)-\mathtt{Cost}(r,c,p)\\
&=\frac{\mathtt{dist}(r,c)+\sum_{j=2}^{p+1}\mathtt{dist}(c,K_{GI_{c,j}})}{p}-\frac{\mathtt{dist}(r,c)+\sum_{j=2}^{p}\mathtt{dist}(c,K_{GI_{c,j}})}{p-1}\\
&=\frac{\sum_{j=2}^{p}(\mathtt{dist}(c,K_{GI_{c,p+1}})-\mathtt{dist}(c,K_{GI_{c,j}}))-\mathtt{dist}(r,c)}{p(p-1)}.
\end{aligned}\tag{19}$$

The denominator is positive. By the definition of GI_c, since $j<p+1$, we have $\mathtt{dist}(c,K_{GI_{c,p+1}})-\mathtt{dist}(c,K_{GI_{c,j}})\ge 0$, so as p increases, the numerator is non-decreasing and hence changes sign at most once (from negative to positive), which completes our proof. □

Monotonicity of `Cost`. Rewriting $\mathtt{Cost}(r,c,p)$ as $\mathtt{Cost}_U(r,c,p)$, we explicitly indicate the involvement of U in calculating the cost-effectiveness by Equation (1). The following theorem proves the monotonicity of $\mathtt{Cost}_U(r,c,p)$: the prefix p that minimizes $\mathtt{Cost}_U(r,c,p)$, i.e., $p^*_U=\arg\min_{2\le p\le g}\mathtt{Cost}_U(r,c,p)$ with rightmost tie-breaking, is monotonically non-decreasing as U shrinks.

Theorem 4. *For any fixed r,c, we have $p^*_{U'}\ge p^*_U$ for $U'\subset U$.*

Proof. Define $\Delta\mathtt{Cost}_U(p)=\mathtt{Cost}_U(r,c,p+1)-\mathtt{Cost}_U(r,c,p)$. It suffices to prove that for all $2\le p<p^*_U$, $\Delta\mathtt{Cost}_{U'}(p)\le 0$.

For each $2\le p<p^*_U$, there are two cases to consider.

Case 1: $GI_{c,p+1}\notin U$, so $GI_{c,p+1}\notin U'$. By Equation (1), we have $\mathtt{Cost}_{U'}(r,c,p)=\mathtt{Cost}_{U'}(r,c,p+1)$, i.e., $\Delta\mathtt{Cost}_{U'}(p)=0$.

Case 2: $GI_{c,p+1}\in U$. Let $S_{U,p}=\{GI_{c,2},\ldots,GI_{c,p}\}\cap U$, so $S_{U,p+1}\neq\emptyset$. There are two sub-cases to consider.

Case 2.1: $S_{U,p}=\emptyset$, so $S_{U',p}=\emptyset$. By Equation (1), we have $\mathtt{Cost}_{U'}(r,c,p)=\infty\ge\mathtt{Cost}_{U'}(r,c,p+1)$, so $\Delta\mathtt{Cost}_{U'}(p)\le 0$.

Case 2.2: $S_{U,p}\neq\emptyset$, so $|S_{U,p+1}|\ge 2$. By Equation (1), we have

$$\begin{aligned}
\Delta\mathtt{Cost}_U(p)&=\mathtt{Cost}_U(r,c,p+1)-\mathtt{Cost}_U(r,c,p)\\
&=\frac{\mathtt{dist}(r,c)+\sum_{j\in S_{U,p+1}}\mathtt{dist}(c,K_j)}{|S_{U,p+1}|}-\frac{\mathtt{dist}(r,c)+\sum_{j\in S_{U,p}}\mathtt{dist}(c,K_j)}{|S_{U,p}|}\\
&=\frac{\sum_{j\in S_{U,p+1}}(\mathtt{dist}(c,K_{GI_{c,p+1}})-\mathtt{dist}(c,K_j))-\mathtt{dist}(r,c)}{|S_{U,p+1}|(|S_{U,p+1}|-1)}.
\end{aligned}\tag{20}$$

Algorithm 5: `ConstructTree`

Input: Root r; Set C_0 of intermediate vertices;
Query $Q = \{K_1, K_2, \ldots, K_g\}$.
Output: GST $T_r = \langle V_r, E_r \rangle$.

```
1 T_r = ⟨V_r, E_r⟩ ← ⟨{r}, ∅⟩;
2 foreach c_0 ∈ C_0 do
3 |   Add the vertices and edges of SP(r, c_0) to T_r;
4 while Q ≠ ∅ do
5 |   ⟨v, K⟩ ← arg min_{⟨v_j,K_i⟩∈(V_r×Q)} dist(v_j, K_i);
6 |   Add the vertices and edges of SP(v, K) to T_r;
7 |   Q ← Q \ {K};
8 return T_r;
```

The denominator is positive. By the definition of GI_c, we have $\texttt{dist}(c, K_{GI_{c,p+1}}) - \texttt{dist}(c, K_j) \geq 0$, so if $\Delta\texttt{Cost}_U(p) \leq 0$, we will also have $\Delta\texttt{Cost}_{U'}(p) \leq 0$. Theorem 3 ensures $\Delta\texttt{Cost}_U(p) \leq 0$. □

4.5 Path Re-Selection

In `MonoGST+`, each call to `ConstructTree` (line 29) constructs a GST with a given root r and intermediate vertices C_0, connecting C_0 to the terminals in the query Q via greedily re-selected paths.

Algorithm 5 (`ConstructTree`). As in `MonoGST`, we initialize a GST T_r with the root r (line 1) and add to T_r one r-c_0 shortest path for each intermediate vertex $c_0 \in C_0$ (lines 2–3). Inspired by Prim's algorithm, we iteratively grow and return T_r until it covers all groups in Q (lines 4, 8). Each iteration adds to T_r a shortest path between the current T_r and any uncovered group $K \in Q$ (line 5–7).

4.6 Algorithm Analysis

We analyze `MonoGST+`'s approximation ratio and time complexity.

4.6.1 Approximation Ratio. `MonoGST+` differs from `MonoGST` in the following aspects: pruning the distances to be calculated, pruning the prefixes to be enumerated, and re-selecting paths to convert a 2-star into a GST. Our analysis in Sections 4.3 and 4.4 ensures that the two pruning methods maintain the optimality of the greedy choice in solving the weighted set cover problem to approximate an optimum 2-star. To prove that `MonoGST+` has the same approximation ratio as `MonoGST`, we only need to show that the path re-selection in Section 4.5 maintains the weight of the GST constructed.

Following the notation in Section 3.3, let $R_{\{\{r\}\cup(Q\setminus K_1)}$ be the 2-star rooted in $r \in K_1$ found in `MonoGST+`. Let $T_{\{\{r\}\cup(Q\setminus K_1)}$ be the GST rooted in r returned by `ConstructTree` in `MonoGST+`. The following lemma bounds the weight of $T_{\{\{r\}\cup(Q\setminus K_1)}$.

Lemma 5. $\texttt{w}(T_{\{\{r\}\cup(Q\setminus K_1)}) \leq \texttt{w}'(R_{\{\{r\}\cup(Q\setminus K_1)})$.

Proof. Let C_0 be the set of intermediate vertices of $R_{\{\{r\}\cup(Q\setminus K_1)}$. We separately consider the connections between r and C_0 and the connections between C_0 and the terminals in Q.

Between r and C_0, `ConstructTree` adds to $T_{\{\{r\}\cup(Q\setminus K_1)}$ one r-c_0 shortest path for each intermediate vertex $c_0 \in C_0$ (lines 2–3). The total length of these paths is not greater than the total weight of their corresponding (r, c_0) edges in $R_{\{\{r\}\cup(Q\setminus K_1)}$.

Table 2: Statistics of datasets.

Dataset	n	m	g	# of Queries
Toronto	46,073	68,353	5–8	160
MovieLens	62,423	35,323,774	5–8	160
DBLP	2,497,782	12,786,329	5–8	160
LinkedMDB	1,326,784	2,132,796	1–10	200
DBpedia	5,887,296	18,338,729	1–10	438

Between C_0 and Q, `ConstructTree` adds to $T_{\{\{r\}\cup(Q\setminus K_1)}$ one shortest path between the current $T_{\{\{r\}\cup(Q\setminus K_1)}$ and each group $K \in Q$ (lines 5–6). The length of such a path is not greater than the weight of the corresponding edge in $R_{\{\{r\}\cup(Q\setminus K_1)}$ which represents the length of a shortest path between a particular intermediate vertex in C_0 and K. □

Theorem 5. `MonoGST+` *has an approximation ratio* $O(\sqrt{g})$.

Proof. This proof resembles our proof of Theorem 1.

Specifically, let T_Q be the GST returned by `MonoGST+`. We have

$$\texttt{w}(T_Q) = \min_{r\in K_1} \texttt{w}(T_{\{\{r\}\cup(Q\setminus K_1)}) \leq \min_{r\in K_1} \texttt{w}'(R_{\{\{r\}\cup(Q\setminus K_1)}), \quad (21)$$

where the last inequality follows from Lemma 5.

Combining Equation (21) and Lemma 3, we obtain Equation (9). □

4.6.2 Time Complexity. Let $|K_1| = n_1$. There are $(g - 1 + n_1)$ SSSP computations (lines 2, 14), each taking $O(m+n\log n)$ time. However, since `SuspendableDijkstra` is often terminated early in practical applications, the actual running time of the n_1 SSSP computations is almost negligible. The construction and sorting of GI_c for all $c \in V$ (lines 3–6) takes $O(ng\log g)$ time, and $D^{\texttt{Sum}}$ is calculated in $O(ng)$ time (lines 7–8). Since $|CP| \leq n$, `Cost` is calculated $O(n_1 gn)$ times (line 19), and indirectly (within `UpdatePrefix`) calculated $O(n_1 gn)$ times (lines 14, 25) thanks to its monotonicity, reduced by a factor of g compared to `MonoGST`, each taking a constant time. The distance upper bound $\overline{d}$ is updated $O(n_1 g)$ times (line 28), and indirectly (within `SuspendableDijkstra`) updated $O(n_1 n)$ times (line 14), each taking $O(g)$ time. The construction of T_r for all $r \in K_1$ (line 29) takes $O(n_1(n + g)g)$ time by maintaining the distance between the current T_r and each group in `ConstructTree`.

Overall, `MonoGST+` has a time complexity of $O((m + n\log n)(g + n_1) + ng\log g + nn_1 g + n_1 g^2)$. Since n_1 and g are often very small in practical applications, `MonoGST+` scales almost linear in graph size. When n_1 is close to n, the time complexity becomes $O(n(m + (n + g)g + n\log n))$, outperforming `MonoGST` by a factor of g or n.

5 EXPERIMENTS

5.1 Experimental Setup

5.1.1 Datasets. We reused five datasets that were used to evaluate our main baselines [27, 28]. Table 2 summarizes their statistics.

Toronto, **MovieLens**, and **DBLP** were used in [28].[1] Toronto is a road network where road junctions are grouped according to their nearby facility types; GSTP is used to formulate property search

[1] https://github.com/YahuiSun/GroupSteinerTree/tree/main/Datasets (accessed June 9, 2026)

for a region close to certain facility types. MovieLens is a network of co-rated movies where movies are grouped according to their genres; GSTP is used to formulate movie recommendation related to certain genres. DBLP is a co-authorship network where experts are grouped according to their skills (i.e., research topics); GSTP is used to formulate team formation for experts having certain skills. For each graph, we followed [22] to generate 160 queries by randomly sampling groups under varying parameters: the number of groups $g \in \{5, 6, 7, 8\}$ and the average group size $f \in \{200, 400, 800, 1600\}$, with 10 queries generated for each $\langle g, f \rangle$ setting.

LinkedMDB[2] and **DBpedia**[3] were used in [27]. LinkedMDB is a movie KG and DBpedia is an encyclopedic KG, where entities are grouped according to the keywords in their names (`rdfs:label`) and edges are weighted inversely by the informativeness of their relation types [6]; GSTP is used to formulate keyword-based KG exploration. We followed [27] to generate keyword queries: for LinkedMDB, we sampled 200 natural language questions from [24] and removed punctuation and stopwords; for DBpedia, we used 438 multi-word queries from [11].

5.1.2 Evaluation Metrics. We evaluated each algorithm's efficiency by measuring its **running time** on each GSTP instance. A time limit of 500 seconds was set per run, after which it was marked as a timeout. When reporting the mean running time over all GSTP instances, we treated timeout runs as 500 seconds.

Task-specific effectiveness metrics in the literature generally aim to minimize the weight of the computed GSTs. We therefore evaluated each algorithm's effectiveness using normalized weight (**NWeight**), defined as the ratio of the weight of its computed GST to that of a minimum-weight GST found by an exact solver [22]. When reporting the maximum and mean NWeights over all GSTP instances, we excluded timeout runs and infeasible instances (e.g., groups lying in different connected components).

5.1.3 Implementation Details. All experiments were conducted on a server equipped with an Intel Xeon E5-2643 v4 CPU and 629 GB of RAM, running Ubuntu 24.04.2 LTS. The source code was compiled with `g++` 13.3.0 under C++20 using the `-O2` optimization flag.

5.2 Comparison with Baselines

5.2.1 Baselines. We compared our algorithms with the following state-of-the-art exact and approximate GST algorithms.

Exact Algorithms. **PrunedDP++** [22] is the fastest exact GST algorithm, despite its exponential running time. DPBF [7], less efficient than PrunedDP++, was excluded from our empirical comparison.

Sublinear-Approximation Algorithms. **Reduction to DSTP** [4] converts GSTP into DSTP; our optimized implementation enumerates the vertices in K_1 as the root of a DST and replaces the DST solver used in [4] with a state-of-the-art one [34], providing a $O(\sqrt{g})$ approximation that matches the guarantee of our algorithms. We also compared with **2-star heuristic**, the first 2-star-based sublinear-approximation GST algorithm [3], which guaranties a $O(\sqrt{g} \ln g)$ approximation but requires an expensive computation of all-pairs shortest paths. Polylogarithmic-approximation algorithms [5, 9] were excluded from our empirical comparison due to their theoretical emphasis and limited scalability on large graphs.

Linear-Approximation Algorithms. **PartialOPT** [28] is a parameterized GST algorithm that achieves a $(g - h + 1)$ approximation by paying a cost of exponential running time in $h \in [2, g]$; we followed [28] to set $h = 3$. **KeyKG$^+$** [27] and **ImprovAPP** [28] represented, respectively, the fastest index-based and index-free approximate GST algorithms, both guaranteeing a $(g - 1)$ approximation; note that the distance index used in KeyKG$^+$ greatly accelerates its execution in exchange for per-graph preprocessing that incurs significant time and memory overheads. Earlier, less efficient linear-approximation algorithms (e.g., BANKS-II [19]) were excluded from our empirical comparison.

5.2.2 Experimental Results. Table 3 shows, for each algorithm on each dataset, its number of timeout runs, mean running time, and maximum and mean NWeights over all GSTP instances.

Efficiency Results (Running Time). KeyKG$^+$ is the fastest algorithm on most datasets, which is expected as it leverages a precomputed index [1] for fast distance and shortest-path lookup. However, its index construction may not scale to very large graphs. On the DBLP dataset, index building is not complete within 24 hours; the performance of KeyKG$^+$ on this dataset is not available.

Among index-free solutions, ImprovAPP and our `MonoGST+` are the only algorithms that complete all runs without timeouts. Their running time is comparable, within the same order of magnitude. Specifically, ImprovAPP runs faster on Toronto, MovieLens, and DBLP, while `MonoGST+` outperforms on LinkedMDB and DBpedia.

Our `MonoGST`, the basic variant of `MonoGST+`, shows a slowdown of 1–2 orders of magnitude on Toronto, MovieLens, and DBLP, underscoring the effectiveness of our optimizations for `MonoGST+`. Nevertheless, `MonoGST` remains substantially faster than PartialOPT across all datasets, but they encounter timeouts on most or all GSTP instances in the DBLP dataset. The exact algorithm PrunedDP++ also outperforms PartialOPT on most datasets but suffers from timeouts on the three largest graphs: DBLP, LinkedMDB, and DBpedia. Finally, the reduction-to-DSTP approach and the 2-star heuristic show clear scalability limitations. On every dataset, a large proportion or even all of their runs result in timeouts.

Effectiveness Results (NWeight). With the exact algorithm PrunedDP++ as a baseline, which consistently returns a minimum-weight GST (i.e., NWeight = 1), our `MonoGST+` outputs satisfying results. Across all datasets, its mean NWeight is at most 1.040, and its largest NWeight on any GSTP instance is only 1.417. Owing to its extensive search space based on 2-star, `MonoGST+` outperforms all other approximation algorithms in terms of mean NWeight.

Among the remaining sublinear-approximation algorithms, the reduction-to-DSTP approach and our `MonoGST` deliver comparable solution quality. The performance of the 2-star heuristic cannot be reported due to its consistent timeouts on all GSTP instances.

Linear-approximation algorithms exhibit larger deviations from the optimum. The maximum NWeight reached by ImprovAPP is consistently larger than that of `MonoGST+`, with their gap more pronounced on larger graphs, up to 40% on DBpedia. PartialOPT and KeyKG$^+$ show even larger deviations, with their maximum NWeights exceeding 2 on LinkedMDB and Toronto, respectively.

[2] https://www.cs.toronto.edu/~oktie/linkedmdb/linkedmdb-latest-dump.zip (accessed June 9, 2026)
[3] https://downloads.dbpedia.org/2016-10/core-i18n/en/mappingbased_objects_en.ttl.bz2 (accessed June 9, 2026)

Table 3: Comparison with baselines.

Algorithm	# of Timeouts ↓	Mean Time (s) ↓	NWeight ↓	
			Max	Mean
Experimental Results on Toronto				
PrunedDP++	0	0.932 ± 0.888	1.000	1.000 ± 0.000
MonoGST	0	11.838 ± 9.169	1.417	1.056 ± 0.069
MonoGST+	0	0.085 ± 0.041	1.417	1.026 ± 0.051
Reduction to DSTP	38	267.258 ± 162.425	1.417	1.045 ± 0.063
2-star heuristic	160	-	-	-
PartialOPT	80	390.843 ± 129.659	1.735	1.143 ± 0.139
KeyKG$^+$ (index-based)	0	0.013 ± 0.009	2.900	1.188 ± 0.276
ImprovAPP	0	0.047 ± 0.010	1.424	1.040 ± 0.073
Experimental Results on MovieLens				
PrunedDP++	0	10.819 ± 11.723	1.000	1.000 ± 0.000
MonoGST	0	51.990 ± 37.250	1.192	1.025 ± 0.042
MonoGST+	0	1.599 ± 0.316	1.192	1.015 ± 0.035
Reduction to DSTP	160	-	-	-
2-star heuristic	160	-	-	-
PartialOPT	160	-	-	-
KeyKG$^+$ (index-based)	0	0.012 ± 0.009	1.481	1.039 ± 0.077
ImprovAPP	0	0.967 ± 0.197	1.293	1.016 ± 0.039
Experimental Results on DBLP				
PrunedDP++	10	134.338 ± 147.632	1.000	1.000 ± 0.000
MonoGST	149	495.881 ± 18.761	1.134	1.062 ± 0.043
MonoGST+	0	15.876 ± 8.242	1.209	1.032 ± 0.042
Reduction to DSTP	160	-	-	-
2-star heuristic	160	-	-	-
PartialOPT	160	-	-	-
KeyKG$^+$ (index-based)	-	-	-	-
ImprovAPP	0	9.088 ± 2.254	1.358	1.082 ± 0.074
Experimental Results on LinkedMDB				
PrunedDP++	6	67.647 ± 128.349	1.000	1.000 ± 0.000
MonoGST	0	1.644 ± 1.775	1.261	1.054 ± 0.069
MonoGST+	0	0.756 ± 0.671	1.223	1.040 ± 0.057
Reduction to DSTP	5	47.806 ± 96.860	1.313	1.054 ± 0.074
2-star heuristic	200	-	-	-
PartialOPT	0	8.355 ± 10.148	2.386	1.131 ± 0.185
KeyKG$^+$ (index-based)	0	0.001 ± 0.003	1.676	1.092 ± 0.122
ImprovAPP	0	1.434 ± 0.908	1.618	1.089 ± 0.112
Experimental Results on DBpedia				
PrunedDP++	8	41.934 ± 84.946	1.000	1.000 ± 0.000
MonoGST	3	38.881 ± 71.790	1.214	1.016 ± 0.033
MonoGST+	0	7.717 ± 3.538	1.204	1.012 ± 0.029
Reduction to DSTP	120	285.426 ± 155.837	1.360	1.036 ± 0.052
2-star heuristic	438	-	-	-
PartialOPT	44	121.205 ± 148.101	1.640	1.098 ± 0.139
KeyKG$^+$ (index-based)	0	0.006 ± 0.013	1.586	1.064 ± 0.097
ImprovAPP	0	8.723 ± 5.043	1.682	1.068 ± 0.099

Figure 1a presents a case study. With root $r = 1$, to cover the group $\{3, 7, 12\}$, MonoGST follows the 2-star to select the path $1-3$, while MonoGST+ re-selects $8-7$ which is shorter. This case also illustrates that 1-star-based algorithms are prone to local optima; to cover the group $\{4, 5\}$, ImprovAPP greedily selects 4 which is closer to the root but the subsequent edges have larger weights than those through 5 selected by our 2-star-based algorithms.

Concluding Remarks. Our MonoGST+ delivers the most accurate solutions among all approximation algorithms while remaining one of the fastest index-free methods, making it a highly practical GST solver with substantial application potential.

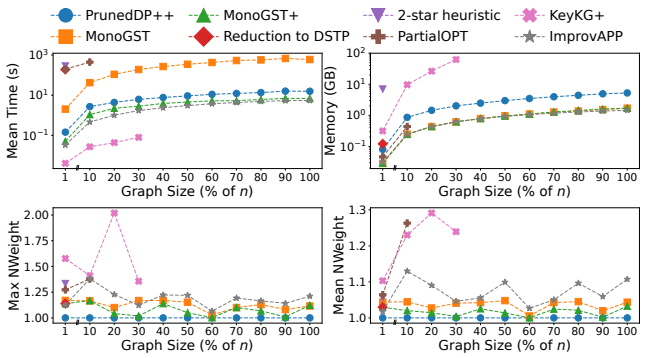


Figure 2: Scalability analysis.

5.3 Scalability Analysis

We systematically analyzed how the performance of our algorithms and the baselines varies with the graph size n.

5.3.1 Graph Settings. On DBLP, we extracted 10 subgraphs containing different proportions of vertices from the original graph: varying in $\{1\%, 10\%, 20\%, \ldots, 90\%\}$. Specifically, given each percentage, we started with random vertices and performed breadth-first search until the required number of vertices were visited. Then we extracted the subgraph induced by the visited vertices.

5.3.2 Experimental Results. Figure 2 shows, for each algorithm on each graph, its mean running time, memory used, and maximum and mean NWeights over all GSTP instances.

Efficiency Results (Running Time and Memory Use). ImprovAPP and our MonoGST+ are the fastest index-free algorithms and have the lowest memory use. Crucially, their time and space increase slowly with n. Other algorithms that exhibit comparable scalability though slower include PrunedDP++ and our MonoGST.

The remaining algorithms show scalability limitations and their curves terminate early. The index building in KeyKG$^+$ uses a vast volume of memory and is complete within 24 hours only for the smallest four graphs. The reduction-to-DSTP approach, the 2-star heuristic, and PartialOPT encounter timeouts on all GSTP instances on all but the smallest one or two graphs. The 2-star heuristic also has the highest memory use, more than 7GB for the smallest graph.

Effectiveness Results (NWeight). For all approximation algorithms, their NWeights fluctuate when graph expands. On almost all graphs, our 2-star-based algorithms output better results than ImprovAPP and KeyKG$^+$ based on 1-star.

Concluding Remarks. Our MonoGST+ is among the highest scalable algorithms and its finer solution quality remains consistent across different graph sizes, strengthening its practicality.

5.4 Robustness Analysis

We analyzed how the number of groups g and the average group size f affect the performance of our MonoGST+.

5.4.1 Parameter Settings. On DBLP, we followed the query generation scheme described in Section 5.1.1 to generate more queries under extended parameter ranges: varying $g \in \{5, 10, \ldots, 50\}$ with $f = 200$, and varying $f \in \{200, 400, \ldots, 2000\}$ with $g = 5$.

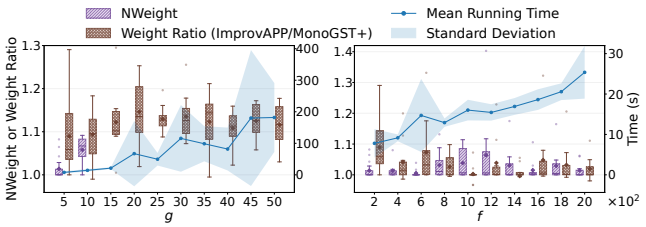


Figure 3: Robustness analysis of MonoGST+.

5.4.2 Experimental Results. Figure 3 plots the distributions of running time and NWeight of MonoGST+ over all GSTP instances in each parameter setting. For $g > 10$, the exact solver could not find a minimum-weight GST in reasonable time since its time complexity is exponential in g, so NWeight became unavailable; compensating for that, we also calculated the ratio of the weight of the GST computed by our main baseline ImprovAPP to that by MonoGST+.

Efficiency Results (Running Time). In general, the mean running time increases near proportionally to g and f, from 7.80 to 181.85 seconds when g increases from 5 to 50, and from 7.80 to 25.37 seconds when f increases from 200 to 2,000.

Effectiveness Results (NWeight). The mean NWeight fluctuates between 1.006 and 1.064 when f changes from 200 to 2,000. It is also small when $g \leq 10$. For larger values of g such that NWeight is not calculable, the mean weight ratio of ImprovAPP to MonoGST+ is kept between 1.108 and 1.146; it is also mostly above 1 when f changes, showing that MonoGST+ robustly delivers high-quality solutions and outperforms ImprovAPP.

Concluding Remarks. The high accuracy of MonoGST+ is insensitive to g and f. Its running time is empirically linear in these parameters, conforming to our theoretical analysis in Section 4.6.2.

5.5 Ablation Study

5.5.1 Variant Settings. We conducted an ablation study by implementing and comparing five variants of MonoGST+, aiming to isolate and quantify the effects of its key components.

The variant **w/o suspension** replaces our suspendable Dijkstra's algorithm (Section 4.3) with the standard Dijkstra's algorithm to compute SSSP from each root $r \in K_1$ in a manner identical to that in MonoGST (line 10). The variant **w/o suspension & pruning** further disables our prefix pruning (Section 4.4) and enumerates each prefix p from 2 to g in a manner identical to that in MonoGST (line 14). The variant **w/o re-selection** replaces our path re-selection (Section 4.5) with the naive method for converting a 2-star into a GST described in Section 2.2 in a manner identical to that in MonoGST (lines 19–21). Note that suspension and prefix pruning mainly optimize running time, while path re-selection focuses on solution quality, addressing different aspects of performance.

MonoGST+ enumerates the vertices in the smallest group as the root of a 2-star (line 1). Differently, the variants **median group** and **largest group** choose a group of median size and the largest group as K_1 to enumerate, respectively.

5.5.2 Experimental Results. Table 4 shows, for each variant on each dataset, its mean running time, and maximum and mean NWeights over all GSTP instances. For precisely reflecting performance difference, we removed per-run time limit from this ablation study.

Table 4: Ablation study of MonoGST+.

Algorithm	Mean Time (s) ↓	NWeight ↓	
		Max	Mean
MonoGST+ *on Toronto*	0.085 ± 0.041	1.417	1.026 ± 0.051
w/o suspension	7.884 ± 5.873	1.417	1.025 ± 0.050
w/o suspension & pruning	11.819 ± 9.734	1.417	1.025 ± 0.049
w/o re-selection	0.088 ± 0.043	1.417	1.056 ± 0.069
median group	0.101 ± 0.041	1.417	1.026 ± 0.051
largest group	0.101 ± 0.041	1.417	1.026 ± 0.051
MonoGST+ *on MovieLens*	1.599 ± 0.316	1.192	1.015 ± 0.035
w/o suspension	49.986 ± 35.506	1.192	1.015 ± 0.035
w/o suspension & pruning	51.145 ± 36.323	1.192	1.015 ± 0.035
w/o re-selection	1.649 ± 0.274	1.192	1.025 ± 0.042
median group	1.653 ± 0.303	1.192	1.015 ± 0.035
largest group	1.600 ± 0.269	1.192	1.015 ± 0.035
MonoGST+ *on DBLP*	15.876 ± 8.242	1.209	1.032 ± 0.042
w/o suspension	1700.880 ± 1247.202	1.209	1.031 ± 0.041
w/o suspension & pruning	2394.501 ± 1834.009	1.209	1.031 ± 0.041
w/o re-selection	16.669 ± 10.110	1.241	1.058 ± 0.053
median group	16.261 ± 8.354	1.209	1.032 ± 0.042
largest group	16.275 ± 8.524	1.209	1.032 ± 0.042
MonoGST+ *on LinkedMDB*	0.756 ± 0.671	1.223	1.040 ± 0.057
w/o suspension	1.544 ± 1.684	1.223	1.038 ± 0.055
w/o suspension & pruning	1.572 ± 1.732	1.223	1.038 ± 0.055
w/o re-selection	0.753 ± 0.667	1.261	1.053 ± 0.068
median group	0.763 ± 0.678	1.246	1.034 ± 0.055
largest group	0.761 ± 0.677	1.246	1.034 ± 0.055
MonoGST+ *on DBpedia*	7.717 ± 3.538	1.204	1.012 ± 0.029
w/o suspension	39.992 ± 107.420	1.204	1.012 ± 0.029
w/o suspension & pruning	41.697 ± 110.882	1.204	1.012 ± 0.028
w/o re-selection	7.560 ± 3.447	1.204	1.016 ± 0.032
median group	7.853 ± 3.566	1.204	1.012 ± 0.029
largest group	7.797 ± 3.560	1.204	1.012 ± 0.029

Efficiency Results (Running Time). The suspendability mechanism is the key driver behind the efficiency of MonoGST+. It enables speedups of 1–2 orders of magnitude on Toronto, MovieLens, and DBLP, and 2×–5× on LinkedMDB and DBpedia. With suspendability on different datasets, only an average of 0.02%–20% of all vertices are visited when Dijkstra's algorithm is terminated.

Prefix pruning provides both theoretical and empirical benefits: it reduces the time complexity by a factor of g, and produces consistent speedups across all datasets, up to 29%–33% on Toronto and DBLP. The benefit becomes more pronounced as g increases, e.g., from 21% to 42% as g increases from 5 to 8 on Toronto.

Switching from the smallest group to the median or largest group leads to consistent increases in running time, up to 19% on Toronto.

Effectiveness Results (NWeight). Path re-selection yields consistent, though modest, quality improvements on all datasets, up to 2%–3% reduction in mean NWeight on Toronto and DBLP. Incurring negligible time cost, it represents a net-positive trade-off between effectiveness and efficiency.

Enumerating the smallest, median, or largest group has little effect on solution quality on most datasets.

Concluding Remarks. In summary, all three optimizations integrated into MonoGST+ collectively establish its superiority over MonoGST in terms of efficiency and effectiveness. These novel techniques enable MonoGST+ to achieve state-of-the-art performance among approximate GST algorithms. It is also cost-effective to enumerate the vertices in the smallest group as the root of a 2-star.

6 RELATED WORK

6.1 Sublinear-Approximation Algorithms

The 2-star concept is first introduced in [3] to solve GSTP, where a heuristic two-step algorithm achieves an $O(\sqrt{g}\ln g)$ approximation, and is later generalized to d-star [13]. *Our holistic analysis improves this ratio to* $O(\sqrt{g})$. Besides, the algorithms in [3, 13] incur the high cost of computing all-pairs shortest paths. *Our suspendable Dijkstra's algorithm calculates distances lazily and supports early termination.* Although both [3] and our approach employ greedy set cover, *we define a novel greedy function whose monotonicity and unimodality enable efficient pruning*. This combination of pay-as-you-go distance calculation and a pruning-friendly greedy function makes our MonoGST+ a practical solution for large GSTP instances.

GSTP can be reduced to DSTP [4] where the d-star-based algorithm achieves an $O(\sqrt{g})$ approximation in $O(n^d g^{2d})$ time, not efficient on large graphs even for $d = 2$. Using a more scalable DST algorithm [34] can reduce the time complexity of solving DSTP to $O(mg + ng\log n)$. Nevertheless, the approach still requires processing up to n DSTP instances, each involving g SSSP computations, amounting to a total of gn computations. By contrast, *our algorithms perform at most* $(g + n)$ *SSSP computations,* accounting for their orders-of-magnitude speedup in our experiments.

Polylogarithmic-approximation algorithms [5, 9] relax GSTP to its variant on a tree with stretch $O(\log n)$, and invoke heavy subroutines such as solving linear programs with exponentially many constraints. As noted in their papers, they incur large hidden constants and accumulated polylogarithmic factors, hard to perform efficiently in practice. GSTP can also be reduced to the Steiner Tree Problem (STP) [8], but an approximation ratio is guaranteed only if using an exact, exponential-time STP solver. *Consequently, the algorithms discussed above are more oriented towards theoretical analysis than practical deployment,* thus omitted from our experiments.

6.2 Linear-Approximation Algorithms

KeyKG$^+$ [27] as an extension of 1-star achieves a $(g - 1)$ approximation. To accelerate the search for the closest uncovered group to merge into the current GST, it uses a distance index [1]. Building this index is computationally expensive and may not scale to very large graphs, such as the DBLP dataset in our experiments.

Among index-free methods that guarantee a $(g - 1)$ approximation, ImprovAPP [28] performs SSSP computations and uses a priority queue to track the uncovered group closest to each vertex in the current GST. PartialOPT [28] improves the approximation to $(g - h + 1)$ by introducing a parameter $h \in [2, g]$ and invoking an exact GST solver [7] on h groups. Its time complexity grows exponentially with h; in practice, a small h (e.g., $h = 3$) must be used, resulting in marginal quality gains.

Earlier methods also achieve an $O(g)$-approximation, such as BANKS [16], its improved variant BANKS-II [19] that employs bidirectional search, and BLINKS [12] that partitions the graph into blocks and builds a block-level index for search pruning.

The algorithms discussed above are predominantly 1-star-based, which trades off a linear approximation ratio in g for efficiency. By contrast, building on the 2-star framework, *our* MonoGST+ *guarantees a strictly better approximation ratio while maintaining its running time comparable to the state-of-the-art index-free linear-approximation algorithms.* Therefore, our work bridges a long-standing gap that has traditionally forced a choice between high-quality approximation and scalable, efficient computation.

6.3 Exact Algorithms

Prominent exact GST solvers include DPBF [7], which employs best-first dynamic programming, and its successor PrunedDP++ [22] based on A* search. Both of them exhibit exponential time complexity and limited scalability. *This body of work is largely orthogonal to ours, as they pursue different aims and adopt disparate methodologies.* In our experimental setup, PrunedDP++ serves a specific role: it provides the optimum GST weights needed to compute NWeight for quantifying the empirical approximation ratios of our algorithms.

7 CONCLUSION

Our work reconciles the long-standing trade-off between sublinear approximation and practical running time for GSTP through: (i) MonoGST, which, via a holistic analysis of its two-step greedy approximation, achieves an $O(\sqrt{g})$ ratio; and (ii) MonoGST+, which further incorporates a suspendable search mechanism and a pruning strategy that exploits the monotonicity and unimodality of the greedy function to match the efficiency of state-of-the-art linear-approximation solvers. This combined effectiveness and efficiency is validated by an extensive evaluation across five real-world datasets spanning tasks such as property search in road networks, team formation in social networks, and keyword-based exploration of KGs, where MonoGST+ produces the highest-quality GSTs among approximation algorithms without compromising on speed.

While these results are promising, several important limitations point to clear avenues for advancement. First, regarding efficiency, our method is on par with the fastest existing index-free baselines but has not advanced beyond; its reliance on g full SSSP computations remains a notable limitation at scale. Second, concerning approximation quality, our work is currently limited to the $d = 2$ case of the d-star framework, despite the fact that larger d generally yields higher quality. Future directions, therefore, involve: (i) developing lightweight, updatable indices (rather than the heavy, static one in KeyKG$^+$) to accelerate distance queries; and (ii) deeply exploring the d-star structure to efficiently expand the search space.

ACKNOWLEDGMENTS

This work was supported by the Noncommunicable Chronic Diseases-National Science and Technology Major Project (2025ZD0544900), the Fundamental Research Funds for the Central Universities (No. 2026300429), the Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China (No. JYB2025XDXM118), and the "111 Center" (No. B26023).

REFERENCES

[1] 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* (New York, New York, USA) *(SIGMOD '13)*. Association for Computing Machinery, New York, NY, USA, 349–360. https://doi.org/10.1145/2463676.2465315

[2] C. Douglass Bateman, C. S. Helvig, Gabriel Robins, and Alexander Zelikovsky. 1997. Provably good routing tree construction with multi-port terminals. In *Proceedings of the 1997 International Symposium on Physical Design* (Napa Valley, California, USA) *(ISPD '97)*. Association for Computing Machinery, New York, NY, USA, 96–102. https://doi.org/10.1145/267665.267697

[3] C. Douglass Bateman, C. S. Helvig, Gabriel Robins, and Alexander Zelikovsky. 1997. Provably good routing tree construction with multi-port terminals. In *Proceedings of the 1997 International Symposium on Physical Design* (Napa Valley, California, USA) *(ISPD '97)*. Association for Computing Machinery, New York, NY, USA, 96–102. https://doi.org/10.1145/267665.267697

[4] Moses Charikar, Chandra Chekuri, To yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. 1999. Approximation Algorithms for Directed Steiner Problems. *Journal of Algorithms* 33, 1 (1999), 73–91. https://doi.org/10.1006/jagm.1999.1042

[5] Chandra Chekuri, Guy Even, and Guy Kortsarz. 2006. A greedy approximation algorithm for the group Steiner problem. *Discrete Applied Mathematics* 154, 1 (2006), 15–34. https://doi.org/10.1016/j.dam.2005.07.010

[6] Gong Cheng, Fei Shao, and Yuzhong Qu. 2017. An Empirical Evaluation of Techniques for Ranking Semantic Associations. *IEEE Trans. on Knowl. and Data Eng.* 29, 11 (Nov. 2017), 2388–2401. https://doi.org/10.1109/TKDE.2017.2735970

[7] 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 *2007 IEEE 23rd International Conference on Data Engineering*. 836–845. https://doi.org/10.1109/ICDE.2007.367929

[8] C.W Duin, A Volgenant, and S Voß. 2004. Solving group Steiner problems as Steiner problems. *European Journal of Operational Research* 154, 1 (2004), 323–329. https://doi.org/10.1016/S0377-2217(02)00707-5

[9] Naveen Garg, Goran Konjevod, and Ramamoorthi Ravi. 2000. A polylogarithmic approximation algorithm for the group Steiner tree problem. *Journal of Algorithms* 37, 1 (2000), 66–84.

[10] Eran Halperin and Robert Krauthgamer. 2003. Polylogarithmic inapproximability. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing* (San Diego, CA, USA) *(STOC '03)*. Association for Computing Machinery, New York, NY, USA, 585–594. https://doi.org/10.1145/780542.780628

[11] Faegheh Hasibi, Fedor Nikolaev, Chenyan Xiong, Krisztian Balog, Svein Erik Bratsberg, Alexander Kotov, and Jamie Callan. 2017. DBpedia-Entity v2: A Test Collection for Entity Search. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Shinjuku, Tokyo, Japan) *(SIGIR '17)*. Association for Computing Machinery, New York, NY, USA, 1265–1268. https://doi.org/10.1145/3077136.3080751

[12] Hao He, Haixun Wang, Jun Yang, and Philip S. Yu. 2007. BLINKS: ranked keyword searches on graphs. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data* (Beijing, China) *(SIGMOD '07)*. Association for Computing Machinery, New York, NY, USA, 305–316. https://doi.org/10.1145/1247480.1247516

[13] C. S. Helvig, G. Robins, and A. Zelikovsky. 1998. Improved approximation bounds for the group Steiner problem. In *Proceedings of the Conference on Design, Automation and Test in Europe* (Le Palais des Congrés de Paris, France) *(DATE '98)*. IEEE Computer Society, USA, 406–413.

[14] Christopher S Helvig, Gabriel Robins, and Alexander Zelikovsky. 2001. An improved approximation scheme for the group steiner problem. *Networks: An International Journal* 37, 1 (2001), 8–20.

[15] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia D'amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. 2021. Knowledge Graphs. *ACM Comput. Surv.* 54, 4, Article 71 (July 2021), 37 pages. https://doi.org/10.1145/3447772

[16] Arvind Hulgeri and Charuta Nakhe. 2002. Keyword Searching and Browsing in Databases using BANKS. In *Proceedings of the 18th International Conference on Data Engineering (ICDE '02)*. IEEE Computer Society, USA, 431.

[17] Edmund Ihler. 1990. Bounds on the quality of approximate solutions to the Group Steiner Problem. In *Proceedings of the 16rd International Workshop on Graph-Theoretic Concepts in Computer Science (WG '90)*. Springer-Verlag, Berlin, Heidelberg, 109–118.

[18] Edmund Ihler. 1991. Bounds on the quality of approximate solutions to the group Steiner problem. In *Graph-Theoretic Concepts in Computer Science*, Rolf H. Möhring (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 109–118.

[19] Varun Kacholia, Shashank Pandit, Soumen Chakrabarti, S. Sudarshan, Rushi Desai, and Hrishikesh Karambelkar. 2005. Bidirectional expansion for keyword search on graph databases. In *Proceedings of the 31st International Conference on Very Large Data Bases* (Trondheim, Norway) *(VLDB '05)*. VLDB Endowment, 505–516.

[20] 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* (Paris, France) *(KDD '09)*. Association for Computing Machinery, New York, NY, USA, 467–476. https://doi.org/10.1145/1557019.1557074

[21] Wangchao Le, Feifei Li, Anastasios Kementsietsidis, and Songyun Duan. 2014. Scalable Keyword Search on Large RDF Data. *IEEE Transactions on Knowledge and Data Engineering* 26, 11 (2014), 2774–2788. https://doi.org/10.1109/TKDE.2014.2302294

[22] 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* (San Francisco, California, USA) *(SIGMOD '16)*. Association for Computing Machinery, New York, NY, USA, 91–106. https://doi.org/10.1145/2882903.2915217

[23] Anirban Majumder, Samik Datta, and K.V.M. Naidu. 2012. Capacitated team formation problem on social networks. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Beijing, China) *(KDD '12)*. Association for Computing Machinery, New York, NY, USA, 1005–1013. https://doi.org/10.1145/2339530.2339690

[24] Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. 2016. Key-Value Memory Networks for Directly Reading Documents. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, Jian Su, Kevin Duh, and Xavier Carreras (Eds.). Association for Computational Linguistics, Austin, Texas, 1400–1409. https://doi.org/10.18653/v1/D16-1147

[25] Soumajit Pramanik, Jesujoba Alabi, Rishiraj Saha Roy, and Gerhard Weikum. 2024. Uniqorn: unified question answering over rdf knowledge graphs and natural language text. *Journal of Web Semantics* 83 (2024), 100833.

[26] Lu Qin, Jeffrey Xu Yu, and Lijun Chang. 2009. Keyword search in databases: the power of RDBMS. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data* (Providence, Rhode Island, USA) *(SIGMOD '09)*. Association for Computing Machinery, New York, NY, USA, 681–694. https://doi.org/10.1145/1559845.1559917

[27] Yuxuan Shi, Gong Cheng, and Evgeny Kharlamov. 2020. Keyword Search over Knowledge Graphs via Static and Dynamic Hub Labelings. In *Proceedings of The Web Conference 2020* (Taipei, Taiwan) *(WWW '20)*. Association for Computing Machinery, New York, NY, USA, 235–245. https://doi.org/10.1145/3366423.3380110

[28] Yahui Sun, Xiaokui Xiao, Bin Cui, Saman K. Halgamuge, Theodoros Lappas, and Jun Luo. 2021. Finding Group Steiner Trees in Graphs with both Vertex and Edge Weights. *Proc. VLDB Endow.* 14, 7 (2021), 1137–1149. https://doi.org/10.14778/3450980.3450982

[29] Šejla Čebirić, François Goasdoué, Haridimos Kondylakis, Dimitris Kotzinos, Ioana Manolescu, Georgia Troullinou, and Mussab Zneika. 2019. Summarizing semantic graphs: a survey. *The VLDB Journal* 28, 3 (June 2019), 295–327. https://doi.org/10.1007/s00778-018-0528-3

[30] Xiaxia Wang, Gong Cheng, Tengteng Lin, Jing Xu, Jeff Z. Pan, Evgeny Kharlamov, and Yuzhong Qu. 2021. PCSG: Pattern-Coverage Snippet Generation for RDF Datasets. In *The Semantic Web – ISWC 2021*, Andreas Hotho, Eva Blomqvist, Stefan Dietze, Achille Fokoue, Ying Ding, Payam Barnaghi, Armin Haller, Mauro Dragoni, and Harith Alani (Eds.). Springer International Publishing, Cham, 3–20.

[31] 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. https://doi.org/10.1109/TBDATA.2016.2546303

[32] Jianye Yang, Wu Yao, and Wenjie Zhang. 2021. Keyword search on large graphs: A survey. *Data Science and Engineering* 6, 2 (2021), 142–162.

[33] Martin Zachariasen and Andre Rohe. 2003. Rectilinear group Steiner trees and applications in VLSI design. *Mathematical Programming* 94, 2 (2003), 407–433.

[34] Guangyi Zhang, Nikolaj Tatti, and Aristides Gionis. 2025. SpeedSteiner: A Fast O(k1/2)-Approximation Algorithm for Directed Steiner Tree. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management* (Seoul, Republic of Korea) *(CIKM '25)*. Association for Computing Machinery, New York, NY, USA, 4096–4105. https://doi.org/10.1145/3746252.3761415