Finding Group Steiner Trees in Graphs with both Vertex and Edge Weights: Some Supplemental Materials

Road map. In Section 1, we prove the transformation from group Steiner trees to Steiner trees. In Section 2, we prove the approximation guarantee of LANCET. In Section 3, we prove the approximation guarantee of exIhlerA. In Section 4, we prove the approximation guarantee of FastAPP. In Section 5, we prove the approximation guarantee of ImprovAPP. In Section 6, we prove the approximation guarantee of PartialOPT. In Section 7, we show that PrunedDP and PrunedDP++ in [2] rely on techniques that do not hold in graphs with vertex weights. In Section 8, we show the memory consumption results. In Section 9, we refine the solutions of exENSteiner, exIhlerA and FastAPP.

1 THE TRANSFORMATION

Theorem 1. *Let $G(V, E, w, c)$ be a connected undirected graph, and Γ be a set of vertex groups. Let $G_t(V_t, E_t, w_t, c_t)$ be a connected undirected graph, and $T_t \subseteq V_t$ be a set of compulsory vertices. Based on G and Γ, we construct G_t and T_t in the following way:*

(1) Initialize $V_t = V$, $E_t = E$, $T_t = \emptyset$, $w_t = (1-\lambda)w$, and $c_t = \lambda c$.

(2) For each vertex group $g \in \Gamma$, (i) add a dummy vertex v_g into T_t and V_t, such that $w_t(v_g) = 0$, and (ii) add dummy edges (v_g, j) for all $j \in g$ into E_t, such that $c_t(v_g, j) = M$, and M is a constant satisfying

$$M > (1-\lambda)\sum_{v\in V} w(v) + \lambda \sum_{e\in E_{MST}} c(e), \tag{1}$$

and E_{MST} is the set of edges in a Minimum Spanning Tree of G.

Let Θ_{G_t} be an optimal solution to the vertex- and edge-weighted Steiner tree problem in G_t, and $\Theta^{non}_{G_t}$ be the non-dummy part of Θ_{G_t}. Then, there is an optimal solution to the vertex- and edge-weighted group Steiner tree problem in G, namely, Θ_G, that has the same sets of vertices and edges with $\Theta^{non}_{G_t}$.

Proof. Since dummy vertices only connect non-dummy vertices, there are at least $|\Gamma|$ dummy edges in Θ_{G_t}. If $c_\lambda(\Theta_G) < c(\Theta^{non}_{G_t})$, then there is a feasible solution to the vertex- and edge-weighted Steiner tree problem in G_t: Θ'_{G_t} such that

$$c(\Theta'_{G_t}) = c_\lambda(\Theta_G) + M|\Gamma| < c(\Theta_{G_t}), \tag{2}$$

which is not possible. Thus, we have $c_\lambda(\Theta_G) \geq c(\Theta^{non}_{G_t})$. Let Θ''_{G_t} be a tree in G_t such that (i) every dummy vertex v_g is a leaf of Θ''_{G_t}; and (ii) the non-dummy part of Θ''_{G_t}, namely, $\Theta^{non''}_{G_t}$, is in a Minimum Spanning Tree of G. Suppose that there is a dummy vertex v_g in Θ_{G_t} that is not a leaf. Since $c(\Theta^{non''}_{G_t}) < M$, we have

$$c(\Theta_{G_t}) \geq c(\Theta^{non}_{G_t}) + M(|\Gamma|+1) > c(\Theta''_{G_t}) = c(\Theta^{non''}_{G_t}) + M|\Gamma|, \tag{3}$$

which is not possible. Thus, every dummy compulsory vertex v_g is a leaf of Θ_{G_t}. As a result, $\Theta^{non}_{G_t}$ is connected and shares the same sets of vertices and edges with a feasible solution to the vertex- and edge-weighted group Steiner tree problem in G, which means that $c_\lambda(\Theta_G) \leq c(\Theta^{non}_{G_t})$. Therefore, $c_\lambda(\Theta_G) = c(\Theta^{non}_{G_t})$. Hence, this theorem holds. □

2 THE APPROXIMATION GUARANTEE OF LANCET

LANCET can be regarded as the vertex- and edge-weighted version of the algorithm in [3], which achieves an approximation guarantee of $2(1 - 1/|T_t|)$ for solving the vertex-unweighted Steiner tree problem. This approximation guarantee relies on the following deduction (*i.e.,* Lemma 1 in [3]): since a pre-order traversal of a tree traverses every edge in this tree exactly twice, in a graph with only edge weights, if we perform a pre-order traversal of an optimal solution tree and sum up every weight that we encounter (including duplicates), then the result is exactly twice the weight of an optimal solution tree. However, in a graph with both vertex and edge weights, summing up the weights that we encounter during this traversal does not always result in twice the weight of an optimal solution tree, since (i) an optimal solution tree may contain non-compulsory vertices with positive weights; and (ii) a pre-order traversal of an optimal solution tree may visit such a vertex more than twice (specifically, the number of times that a pre-order traversal of an optimal solution tree visits such a vertex equals the degree of this vertex in this optimal solution tree). Thus, the above approximation guarantee of $2(1 - 1/|T_t|)$ does not hold for LANCET. In what follows, we establish the approximation guarantee of LANCET.

Theorem 2. LANCET *has a sharp approximation guarantee of $|T_t| - 1$ for solving the vertex- and edge-weighted Steiner tree problem.*

Proof. LANCET merges $|T_t| - 1$ LWPs to connect all compulsory vertices together. Suppose that the highest-weight one of these LWPs is LWP', and Θ_{opt} is an optimal solution. Since $c(LWP')$ is smaller than or equal to the weight of the LWP between a pair of compulsory vertices, we have

$$c(\Theta_{opt}) \geq c(LWP'). \tag{4}$$

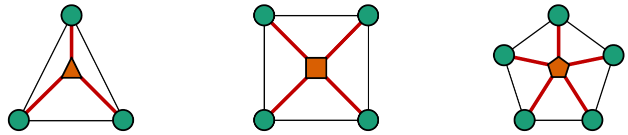


Figure 1: Touching the approximation guarantee of $|T_t|-1$.

Since there are $|T_t|-1$ LWPs that have been merged, we have

$$(|T_t|-1)c(\Theta_{opt}) \geq (|T_t|-1)c(LWP') \geq c(\Theta). \tag{5}$$

Therefore, LANCET has an approximation guarantee of $|T_t|-1$. We further show that $|T_t|-1$ is the sharp approximation guarantee of LANCET. Consider a regular polygon composed of $|T_t|$ compulsory vertices, and a non-compulsory vertex that is in the middle of this polygon and connects $|T_t|$ compulsory vertices (see Figure 1). The weight of each edge between compulsory vertices is x, the weight of each edge between a compulsory vertex and the middle non-compulsory vertex is 0, the weight of each compulsory vertex is 0, and the weight of the middle non-compulsory vertex is z. Suppose that $x=z-\delta$, where δ is a tiny positive value; and $z<(|T_t|-1)x$. Since $x<z$, Θ contains $|T_t|-1$ edges between compulsory vertices, and $c(\Theta)=(|T_t|-1)x$. Since $z<(|T_t|-1)x$, Θ_{opt} contains all the edges between compulsory vertices and the middle non-compulsory vertex, and $c(\Theta_{opt})=z$. We have

$$\lim_{\delta\to 0}\frac{c(\Theta)}{c(\Theta_{opt})}=\frac{(|T_t|-1)(z-\delta)}{z}=|T_t|-1. \tag{6}$$

Hence, $|T_t|-1$ is the sharp approximation guarantee of LANCET. □

3 THE APPROXIMATION GUARANTEE OF exIhlerA

Theorem 3. exIhlerA *has a sharp approximation guarantee of* $|\Gamma|-1$ *for solving the vertex- and edge-weighted group Steiner tree problem.*

Proof. Suppose that $\Theta_{OPT}(V_{OPT},E_{OPT})$ is an optimal solution. Let $\Gamma=\{g_1,\ldots,g_{|\Gamma|}\}$. There is a tuple $(v_1,\ldots,v_{|\Gamma|})$ such that $v_i\in V_{OPT}\cap g_i$ for all $i\in\{1,\ldots,|\Gamma|\}$. Without loss of generality, assume that $g_{min}=g_1$. For every $i\in\{2,\ldots,|\Gamma|\}$, there is exactly one simple path between v_1 and v_i in Θ_{OPT}, which we refer to as $P_{v_1v_i}$. We have

$$c_\lambda(P_{v_1v_i})\leq c_\lambda(\Theta_{OPT}), \tag{7}$$

$$\sum_{g\in\Gamma\setminus g_1}c_\lambda(LWP_{\lambda v_1 g})\leq\sum_{i\in\{2,\ldots,|\Gamma|\}}c_\lambda(P_{v_1v_i}). \tag{8}$$

Thus,

$$c_\lambda(\Theta)\leq c_\lambda(G_{v_1})\leq\sum_{g\in\Gamma\setminus g_1}c_\lambda(LWP_{\lambda v_1 g})\leq\sum_{i\in\{2,\ldots,|\Gamma|\}}c_\lambda(P_{v_1v_i})\leq(|\Gamma|-1)c_\lambda(\Theta_{OPT}). \tag{9}$$

Hence, exIhlerA has an approximation guarantee of $|\Gamma|-1$. We note that this guarantee is sharp. To explain, consider the graph $G(V,E,w,c)$ in Figure 2, where $V=\{v_0,v_1,\ldots,v_{|\Gamma|}\}$, $E=\{(v_{|\Gamma|},v_0),(v_{|\Gamma|},v_1),\ldots,(v_{|\Gamma|},v_{|\Gamma|-1})\}$, $w(i)=0$ for all $i\in V$, $c(v_{|\Gamma|},v_1)=\ldots=c(v_{|\Gamma|},v_{|\Gamma|-1})=1$, and $c(v_{|\Gamma|},v_0)=1+\delta$, where δ is a tiny positive value. In addition, $\Gamma=\{v_0,v_1\}\cup\ldots\cup\{v_0,v_{|\Gamma|-1}\}\cup\{v_{|\Gamma|}\}$. Let $\lambda=1$. Since $g_{min}=\{v_{|\Gamma|}\}$, exIhlerA produces the solution $\Theta=\{(v_{|\Gamma|},v_1),\ldots,(v_{|\Gamma|},v_{|\Gamma|-1})\}$, and $c_\lambda(\Theta)=|\Gamma|-1$. When $|\Gamma|=2$, Θ is the optimal solution, *i.e.*, the approximation ratio is $|\Gamma|-1=1$. When $|\Gamma|>2$, we have $\Theta_{OPT}=\{(v_{|\Gamma|},v_0)\}$, and

$$\lim_{\delta\to 0}\frac{c_\lambda(\Theta)}{c_\lambda(\Theta_{OPT})}=\frac{|\Gamma|-1}{1+\delta}=|\Gamma|-1. \tag{10}$$

Hence, the best possible approximation guarantee of exIhlerA is $|\Gamma|-1$. □

4 THE APPROXIMATION GUARANTEE OF FastAPP

Theorem 4. FastAPP *has a sharp approximation guarantee of* $|\Gamma|-1$ *for solving the vertex- and edge-weighted group Steiner tree problem.*

Proof. Let $\Theta_{OPT}(V_{OPT},E_{OPT})$ be an optimal solution, and $\Gamma=\{g_1,\ldots,g_{|\Gamma|}\}$. There is a tuple $(v_1,\ldots,v_{|\Gamma|})$ such that $v_i\in V_{OPT}\cap g_i$ for all $i\in\{1,\ldots,|\Gamma|\}$. Without loss of generality, suppose that $g_{min}=g_1$. Let $g_x\in\Gamma\setminus g_1$ be such a vertex group that

$$c_\lambda(LWP_{\lambda v_1 g_x})=\max\{c_\lambda(LWP_{\lambda v_1 g})\mid\forall g\in\Gamma\setminus g_1\}. \tag{11}$$

Since $LWP_{\lambda v_1 g_x}$ links fewer groups to v_1 than Θ_{OPT}, we have

$$c_\lambda(LWP_{\lambda v_1 g_x})\leq c_\lambda(\Theta_{OPT}). \tag{12}$$

Lines 5-8 in FastAPP guarantee that

$$\max\{c_\lambda(LWP_{\lambda i_{min} g})\mid\forall g\in\Gamma\setminus g_1\}\leq c_\lambda(LWP_{\lambda v_1 g_x}). \tag{13}$$

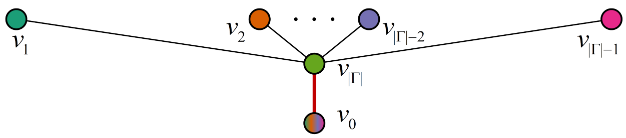


Figure 2: Touching the approximation guarantee of $|\Gamma|-1$.

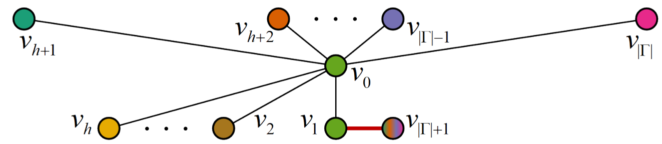


Figure 3: Touching the approximation guarantee of $|\Gamma|-h+1$.

By Lines 10-11 in FastAPP, we have

$$c_\lambda(\Theta) \leq (|\Gamma|-1)\cdot \max\{c_\lambda(LWP_{\lambda i_{min} g}) \mid \forall g \in \Gamma \setminus g_1\}. \tag{14}$$

Thus,

$$c_\lambda(\Theta) \leq (|\Gamma|-1)c_\lambda(LWP_{\lambda v_1 g_x}) \leq (|\Gamma|-1)c_\lambda(\Theta_{OPT}). \tag{15}$$

Hence, FastAPP has an approximation guarantee of $|\Gamma|-1$. The sharpness of this guarantee can be seen from the example in Section 3, *i.e.,* Figure 2. Hence, this theorem holds. □

5 THE APPROXIMATION GUARANTEE OF ImprovAPP

Theorem 5. ImprovAPP *has a sharp approximation guarantee of* $|\Gamma|-1$ *for solving the vertex- and edge-weighted group Steiner tree problem.*

Proof. Let $\Theta_{OPT}(V_{OPT}, E_{OPT})$ be an optimal solution. Let $\Gamma = \{g_1, \ldots, g_{|\Gamma|}\}$. There is a tuple $(v_1, \ldots, v_{|\Gamma|})$ such that $v_i \in V_{OPT} \cap g_i$ for all $i \in \{1, \ldots, |\Gamma|\}$. Without loss of generality, suppose that $g_{min} = g_1$. When ImprovAPP processes v_1 in the for loop (Lines 5-14), it concatenates $|\Gamma|-1$ lowest weight paths that link $\{g_2, \ldots, g_{|\Gamma|}\}$, respectively, to build Θ_{v_1}. Let $LWP_{\lambda v_x g_y}$ be one of these paths that has the largest regulated weight, and links $g_y \in \{g_2, \ldots, g_{|\Gamma|}\}$. Then,

$$c_\lambda(\Theta_{v_1}) \leq (|\Gamma|-1)c_\lambda(LWP_{\lambda v_x g_y}). \tag{16}$$

Let $LWP_{\lambda v_1 g_y}$ be the lowest weight path between v_1 and g_y. Since $LWP_{\lambda v_1 g_y}$ has been pushed into Q initially (Line 6) and has (possibly) been updated to $LWP_{\lambda v_x g_y}$ (Line 12), we have

$$c_\lambda(LWP_{\lambda v_x g_y}) \leq c_\lambda(LWP_{\lambda v_1 g_y}). \tag{17}$$

Since $LWP_{\lambda v_1 g_y}$ links fewer groups to v_1 than Θ_{OPT}, we have

$$c_\lambda(LWP_{\lambda v_1 g_y}) \leq c_\lambda(\Theta_{OPT}). \tag{18}$$

Thus,

$$c_\lambda(\Theta) \leq c_\lambda(\Theta_{v_1}) \leq (|\Gamma|-1)c_\lambda(LWP_{\lambda v_x g_y}) \leq (|\Gamma|-1)c_\lambda(LWP_{\lambda v_1 g_y}) \leq (|\Gamma|-1)c_\lambda(\Theta_{OPT}). \tag{19}$$

Therefore, ImprovAPP has an approximation guarantee of $|\Gamma|-1$. The sharpness of this guarantee can be seen from the example in Section 3, *i.e.,* Figure 2. Thus, this theorem holds. □

6 THE APPROXIMATION GUARANTEE OF PartialOPT

Theorem 6. PartialOPT *has a sharp approximation guarantee of* $|\Gamma|-h+1$ *for solving the vertex- and edge-weighted group Steiner tree problem.*

Proof. Suppose that $\Theta_{OPT}(V_{OPT}, E_{OPT})$ is an optimal solution, and $\Gamma = \{g_1, \ldots, g_{|\Gamma|}\}$. There is a tuple $(v_1, \ldots, v_{|\Gamma|})$ such that $v_i \in V_{OPT} \cap g_i$ for all $i \in \{1, \ldots, |\Gamma|\}$. Without loss of generality, let $g_{min} = g_1$. For $v_1 \in g_{min}$, $\Theta^h_{v_1}$ is optimal for $\Gamma_1 = \{\{v_1\}, g_2, \ldots, g_h\}$. Since $\Theta^h_{v_1}$ connects fewer vertex groups to v_1, we have

$$c_\lambda(\Theta^h_{v_1}) \leq c_\lambda(\Theta_{OPT}). \tag{20}$$

If $\Gamma_2 = \{\{v_1\}\}$, then $\Theta^{|\Gamma|}_{v_1} = \{v_1\}$. Otherwise, we implement exIhlerA to produce $\Theta^{|\Gamma|}_{v_1}$ for $\Gamma_2 = \{\{v_1\}, g_{h+1}, \ldots, g_{|\Gamma|}\}$. Suppose that $\Theta^{|\Gamma|}_{OPT}$ is an optimal solution for Γ_2. The proof of Theorem 3 shows that

$$c_\lambda(\Theta^{|\Gamma|}_{v_1}) \leq (|\Gamma|-h)c_\lambda(\Theta^{|\Gamma|}_{OPT}). \tag{21}$$

Since $\Theta^{|\Gamma|}_{OPT}$ connects fewer vertex groups to v_1, we have

$$c_\lambda(\Theta^{|\Gamma|}_{OPT}) \leq c_\lambda(\Theta_{OPT}). \tag{22}$$

Thus,

$$c_\lambda(\Theta) \leq c_\lambda(G_{v_1}) = c_\lambda(\Theta^h_{v_1} \cup \Theta^{|\Gamma|}_{v_1}) \leq c_\lambda(\Theta^h_{v_1}) + c_\lambda(\Theta^{|\Gamma|}_{v_1}) \leq (|\Gamma|-h+1)c_\lambda(\Theta_{OPT}). \tag{23}$$

Therefore, PartialOPT has an approximation guarantee of $|\Gamma| - h + 1$. We show that this guarantee is sharp. Consider the graph $G(V, E, w, c)$ in Figure 3, where $V = \{v_0, v_1, \ldots, v_{|\Gamma|+1}\}$, $E = \{(v_0, v_1), (v_0, v_2), \ldots, (v_0, v_{|\Gamma|}), (v_1, v_{|\Gamma|+1})\}$, $w(i) = 0$ for every $i \in \{v_0, \ldots, v_{h-1}\}$, $w(i) = 1$ for every $i \in \{v_h, \ldots, v_{|\Gamma|+1}\}$, $c(v_0, v_1) = \delta_1$, $c(v_1, v_{|\Gamma|+1}) = \delta_2$, where δ_1 and δ_2 are two tiny positive values, and $\delta_1 < \delta_2$, and all the other edge weights are zero. In addition, $\Gamma = \{g_1, \ldots, g_{|\Gamma|}\} = \{v_0, v_1\} \cup \{v_2, v_{|\Gamma|+1}\} \cup \ldots \cup \{v_{|\Gamma|}, v_{|\Gamma|+1}\}$. Let $\lambda = 0.5$, *i.e.,* vertex and edge weights are regulated equally. PartialOPT enumerates two vertices in g_{min}: v_0 and v_1. For v_0, PartialOPT produces $\Theta^h_{v_0} = \{(v_0, v_2), \ldots, (v_0, v_h)\}$, and $\Theta^{|\Gamma|}_{v_0} = \{(v_0, v_{h+1}), \ldots, (v_0, v_{|\Gamma|})\}$. Thus, $\Theta_{v_0} = \{(v_0, v_2), \ldots, (v_0, v_{|\Gamma|})\}$. Similarly, for v_1, since $\delta_1 < \delta_2$, PartialOPT produces $\Theta_{v_1} = \{(v_0, v_1), \ldots, (v_0, v_{|\Gamma|})\}$. We have $\Theta = \Theta_{v_0}$. When $|\Gamma| = h$, Θ is the optimal solution, *i.e.*, the approximation ratio is $|\Gamma| - h + 1 = 1$. When $|\Gamma| > h$, we have $\Theta_{OPT} = \{(v_1, v_{|\Gamma|+1})\}$, and

$$\lim_{\delta_2 \to 0} \frac{c_\lambda(\Theta)}{c_\lambda(\Theta_{OPT})} = \frac{|\Gamma| - h + 1}{1 + \delta_2} = |\Gamma| - h + 1. \tag{24}$$

Hence, $|\Gamma| - h + 1$ is the best possible approximation guarantee of PartialOPT. This theorem holds. □

7 THE RECENT WORK ON ENHANCING DPBF

The PrunedDP and PrunedDP++ algorithms in [2] enhance the DPBF algorithm in [1] for finding optimal vertex-unweighted group Steiner trees. The main idea of this enhancement is to incorporate pruning techniques into the process of DPBF. Here, we show that PrunedDP and PrunedDP++ rely on pruning techniques that do not hold in graphs with vertex weights. In the following, we use $T(v, \Gamma)$ to signify the minimum-weight tree that roots at vertex v and covers all vertex groups in Γ.

Theorem 2 in [2] does not hold in graphs with vertex weights. Theorem 2 in [2] is the core pruning technique in PrunedDP, and is also an important pruning technique in PrunedDP++. This theorem does not hold in graphs with vertex weights. To explain, we first briefly describe the dynamic programming process of DPBF through an example in Figure 4. Understanding this process is necessary for understanding the reason why Theorem 2 in [2] does not hold in graphs with vertex weights.

In Figure 4, there are three vertex groups $g_1 = \{v_1\}$, $g_2 = \{v_2\}$ and $g_3 = \{v_3\}$. The weight of u is 1, and each of the other vertex and edge weights is δ, and δ is a tiny positive value. The optimal solution tree is the whole graph, and the weight of this tree is $1 + 6\delta$ (*i.e.,* the sum of vertex and edge weights). To find this tree, DPBF first initializes $T(v_1, \{g_1\})$ as the single vertex v_1; $T(v_2, \{g_2\})$ as the single vertex v_2; and $T(v_3, \{g_3\})$ as the single vertex v_3. Then, DPBF grows $T(v_1, \{g_1\})$, $T(v_2, \{g_2\})$ and $T(v_3, \{g_3\})$ to vertex u, and produces $T(u, \{g_1\})$ as the edge (u, v_1); $T(u, \{g_2\})$ as the edge (u, v_2); and $T(u, \{g_3\})$ as the edge (u, v_3). Subsequently, it merges $T(u, \{g_1\})$ and $T(u, \{g_2\})$ as $T(u, \{g_1, g_2\}) = \{(u, v_1), (u, v_2)\}$. At last, it merges $T(u, \{g_3\})$ and $T(u, \{g_1, g_2\})$, and produces the optimal solution tree.

Theorem 2 in [2] is that: in DPBF, we can merge two subtrees $T(u, \Gamma')$ and $T(u, \Gamma'')$ for $\Gamma'' \subset \Gamma \setminus \Gamma'$ only when the total weight of these two subtrees is not larger than $\frac{2}{3}$ of the weight of an optimal solution tree. This theorem is true when all vertex weights are zero. For example, if all vertex weights are zero in the above instance, then the weight of the optimal solution tree is 3δ. When we merge $T(u, \{g_1\})$ and $T(u, \{g_2\})$ as $T(u, \{g_1, g_2\})$ in the above process, the total weight of $T(u, \{g_1\})$ and $T(u, \{g_2\})$ is 2δ, which is not larger than $\frac{2}{3}$ of the weight of an optimal solution tree. By Theorem 2 in [2], merging these two subtrees may help produce the optimal solution tree. If the total weight of $T(u, \{g_1\})$ and $T(u, \{g_2\})$ is larger than $\frac{2}{3}$ of the weight of an optimal solution tree, then merging these two subtrees does not help produce the optimal solution tree, and thus this merge can be avoided. However, this is true only when all vertex weights are zero. For example, if we consider the vertex weights in the above instance, then the total weight of $T(u, \{g_1\})$ and $T(u, \{g_2\})$ is $2 + 4\delta$, which is larger than $\frac{2}{3}$ of the weight of an optimal solution tree: $1 + 6\delta$ (notably, even the weight of $T(u, \{g_1, g_2\}) = \{(u, v_1), (u, v_2)\}$, which is $1 + 4\delta$, is larger than $\frac{2}{3}$ of the weight of an optimal solution tree). As a result, if we use Theorem 2 in [2] in the above instance with vertex weights, then the optimal solution tree will never be found. That is to say, Theorem 2 in [2] does not hold in graphs with vertex weights.

We point out the specific place in the proof of Theorem 2 in [2] that does not hold in graphs with vertex weights as follows. In the beginning of the proof of Theorem 2 in [2], an optimal solution is assumed to be a tree rooted at vertex u with k subtrees, $T_1, \ldots, T_k$. Each subtree T_i roots at v_i, and the weight of each subtree is smaller than half of the weight of an optimal solution tree (*e.g.,* in Figure 4, T_i is the single vertex v_i). Let $\overline{T_i}$ be the the edge-grown subtree that is grown by T_i with an edge (v_i, u) (*e.g.,* in Figure 4, $\overline{T_i}$ is the edge (v_i, u)). The proof of Theorem 2 in [2] claims that: there are three different cases: (1) the weight of each $\overline{T_i}$ is smaller than half of the weight of an optimal solution tree; (2) there is only one edge-grown subtree $\overline{T_i}$ that has a weight no smaller than half of the weight of an optimal solution tree; and (3) there are two edge-grown subtrees and the weight of each one is half of the weight of an optimal solution tree. This claim is not true in vertex-weighted scenarios, where there is a fourth case: there are more than two edge-grown subtrees such that the weight of each one is large than half of the weight of an optimal solution tree. For example, in Figure 4, if we consider vertex weights, then the weight of each $\overline{T_i}$ is $1 + 2\delta$, which is larger than half of the weight of an optimal solution tree.

Lemmas 2 and 3 in [2] do not hold in graphs with vertex weights. Except Theorem 2 in [2], another important pruning technique in PrunedDP++ is the tour-based lower bounds construction method for A^*-search. There are two types of tour-based lower bounds, which are based on Lemmas 2 and 3 in [2], respectively. We show that these two lemmas do not hold in vertex-weighted scenarios as follows.

First, we briefly introduce the label-enhanced graph in [2], which is constructed by adding dummy vertices and edges into the graph as follows. For each group $g_i \in \Gamma$, we add a dummy vertex $\widetilde{v_i}$, and also add a dummy edge $(\widetilde{v_i}, u)$ with zero weight for every $u \in g_i$. For example, in Figure 5, the graph contains two vertices v and u, and one edge (v, u), and there are three vertex groups $g_1 = g_2 = g_3 = \{u\}$.

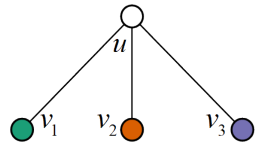


Figure 4: An example for showing Theorem 2 in [2].

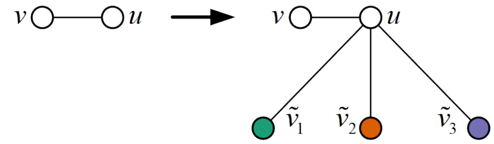


Figure 5: An example for showing Lemmas 2 and 3 in [2].

We add three dummy vertices $\widetilde{v_1}, \ldots, \widetilde{v_3}$ and three dummy edges $(\widetilde{v_1}, u), \ldots, (\widetilde{v_3}, u)$ for creating the label-enhanced graph. Then, [2] uses $W(\widetilde{v_i}, \widetilde{v_j}, \Gamma')$ to refer to the weight of the minimum-weight route that starts from $\widetilde{v_i}$, ends at $\widetilde{v_j}$, and passes through all dummy vertices that corresponds to vertex groups in Γ'. Moreover, [2] uses $d(v, \widetilde{v_i})$ to refer to the weight of the minimum-weight path between non-dummy vertex v and dummy vertex $\widetilde{v_i}$.

Lemma 2 in [2] is that: for every pair of vertex v and a subset of vertex groups $\Gamma' \subseteq \Gamma$, the weight of $T(v, \Gamma')$ is not smaller than $lb_1 = \frac{\min_{g_i, g_j \in \Gamma'} \{d(v,\widetilde{v_i}) + W(\widetilde{v_i}, \widetilde{v_j}, \Gamma') + d(\widetilde{v_j}, v)\}}{2}$. This lemma is true when all vertex weights are zero. For example, in Figure 5, let the weight of edge (v, u) be δ, which is a tiny positive value, and all vertex weights be zero, and $\Gamma' = \{g_1, g_2\}$. Then, $d(v, \widetilde{v_1}) = d(\widetilde{v_2}, v) = \delta$, $W(\widetilde{v_1}, \widetilde{v_2}, \Gamma') = 0$, and $T(v, \Gamma')$ is the edge (v, u). As a result, $lb_1 = \delta$, which equals the weight of $T(v, \Gamma')$. Thus, Lemma 2 in [2] holds. This lemma is proven in [2] by first doubling every edge in $T(v, \Gamma')$ to obtain an Euler tour that starts from v and also ends at v, and then employing the fact that the total edge weight (including duplicates) that we encounter in this Euler tour is twice the total edge weight in $T(v, \Gamma')$. Nevertheless, Lemma 2 in [2] does not hold in vertex-weighted scenarios. For example, in the above instance, let the weights of v and u be 0 and 1, respectively. Then, $d(v, \widetilde{v_1}) = d(\widetilde{v_2}, v) = 1 + \delta$, and $W(\widetilde{v_1}, \widetilde{v_2}, \Gamma') = 1$. As a result, $lb_1 = \frac{3+2\delta}{2}$, which is larger than the weight of $T(v, \Gamma')$: $1 + \delta$. Thus, Lemma 2 in [2] does not hold any more. The reason is that the weight of u is counted multiple times, since the tour that starts from v and ends at v encounters u multiple times. Generally speaking, the total vertex and edge weight that we encounter in the Euler tour in the proof of Lemma 2 in [2] may be more than twice the weight of $T(v, \Gamma')$, since an Euler tour of a tree may visit a vertex in this tree more than twice. As shown in Section 2 in this supplement, for a similar reason, LANCET does not have a guarantee of 2.

Also for a similar reason, Lemma 3 in [2] does not hold in graphs with vertex weights. The details are as follows. [2] uses $W(\widetilde{v_i}, \Gamma')$ to refer to the weight of the minimum-weight route that starts from $\widetilde{v_i}$, and passes through all dummy vertices that corresponds to vertex groups in Γ'. Lemma 3 in [2] is that: the weight of $T(v, \Gamma')$ is not smaller than $lb_2 = \frac{\max_{g_i \in \Gamma'} \{d(v,\widetilde{v_i}) + W(\widetilde{v_i}, \Gamma') + \min_{g_j \in \Gamma'} \{d(\widetilde{v_j}, v)\}\}}{2}$. This lemma is true when all vertex weights are zero. Consider the above instance. If all vertex weights are zero, then $d(v, \widetilde{v_1}) = d(\widetilde{v_2}, v) = \delta$, $W(\widetilde{v_1}, \Gamma') = W(\widetilde{v_2}, \Gamma') = 0$, $\min_{g_j \in \Gamma'} \{d(\widetilde{v_j}, v)\} = \delta$, and the weight of $T(v, \Gamma')$ is δ. Consequently, $lb_2 = \delta$, which equals the weight of $T(v, \Gamma')$. Thus, Lemma 3 in [2] holds. Like Lemma 2, Lemma 3 is proven in [2] by doubling every edge in $T(v, \Gamma')$ to obtain an Euler tour. Also like Lemma 2, since an Euler tour of a tree may visit a vertex in this tree more than twice, Lemma 3 in [2] does not hold in graphs with vertex weights. For example, suppose that, in Figure 5, the weights of v and u are 0 and 1, respectively. Then, $d(v, \widetilde{v_1}) = d(\widetilde{v_2}, v) = 1 + \delta$, $W(\widetilde{v_1}, \Gamma') = W(\widetilde{v_2}, \Gamma') = 1$, $\min_{g_j \in \Gamma'} \{d(\widetilde{v_j}, v)\} = 1 + \delta$, and the weight of $T(v, \Gamma')$ is $1 + \delta$. As a result, $lb_2 = \frac{3+2\delta}{2}$, which is larger than the weight of $T(v, \Gamma')$. Thus, Lemma 3 in [2] does not hold any more.

Recall that (i) Theorem 2 in [2] is the core pruning technique in PrunedDP, and is also an important pruning technique in PrunedDP++; and (ii) another important pruning technique in PrunedDP++ is the tour-based lower bounds construction method for A^*-search, and there are two types of tour-based lower bounds, which are based on Lemmas 2 and 3 in [2], respectively. Since Theorem 2 and Lemmas 2 and 3 in [2] do hold in graphs with vertex weights, we do not implement PrunedDP and PrunedDP++ in our paper.

8 MEMORY CONSUMPTION IN EXPERIMENTS

Here, we report the memory consumption results in the experiments in our paper. The reported memory consumption of each algorithm contains the memory consumed by each input of this algorithm (*e.g.,* G and Γ) as well as any other memory consumed in the process of this algorithm. We use adjacency lists based on hashes to store graphs. Adjacency lists based on hashes consume more memories than adjacency lists based on arrays. Our purpose of using adjacency lists based on hashes is to fully optimize the time complexities of algorithms.

Our extensions. We report the memory consumption of ENSteiner, IhlerA, exENSteiner and exIhlerA in Figure 6, where vertex groups are selected via the uniform approach, and the parameter settings are: for Toronto, $|V| = 46073$, $|\Gamma| = 8$, $\lambda = 0.33$; for DBLP, $|V| = 2497782$, $|\Gamma| = 6$, $\lambda = 0.33$; for MovieLens, $|V| = 2423$, $|\Gamma| = 5$, $\lambda = 0.33$ (this corresponds to the experiments in Figure 2 in our paper). We observe that exENSteiner and exIhlerA consume similar amounts of memory with ENSteiner and IhlerA respectively.

Comparing DPBF **with** Basic**.** We report the memory consumption of DPBF and Basic in Figure 7, where vertex groups are selected via the uniform and non-uniform approaches in the left and right sub-figures, respectively. For Toronto, $|V| = 46073$, $|\Gamma| = 8$, $\lambda = 0.33$; for DBLP, $|V| = 107782$, $|\Gamma| = 5$, $\lambda = 0.33$; for MovieLens, $|V| = 10423$, $|\Gamma| = 5$, $\lambda = 0.33$ (this corresponds to the experiments in Figure 3 in our paper). For Toronto and DBLP in the left sub-figure and DBLP in the right sub-figure, Basic consumes slightly more memory than DPBF. There are two reasons. First, Basic stores the lowest weight paths between vertices and vertex groups, while DPBF does not store this. Second, both Basic and DPBF iteratively pop trees out of a min priority queue. Basic records trees that have been popped out (details in [2]), while DPBF does not record this. In comparison, for Toronto in the right sub-figure, Basic consumes slightly less memory than DPBF. This shows that the pruning technique in Basic is more effective here, and as a result Basic enumerates fewer trees than DPBF, even though Basic is slightly slower than DPBF here, as shown in our paper.

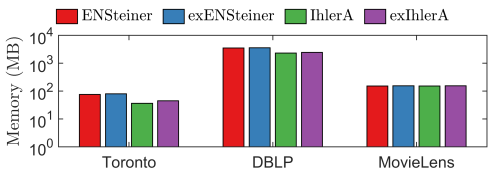


Figure 6: Our extensions.

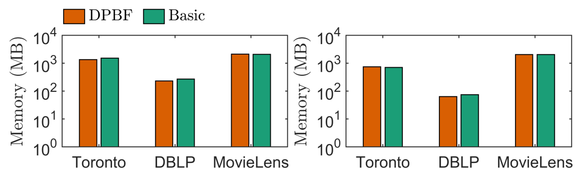
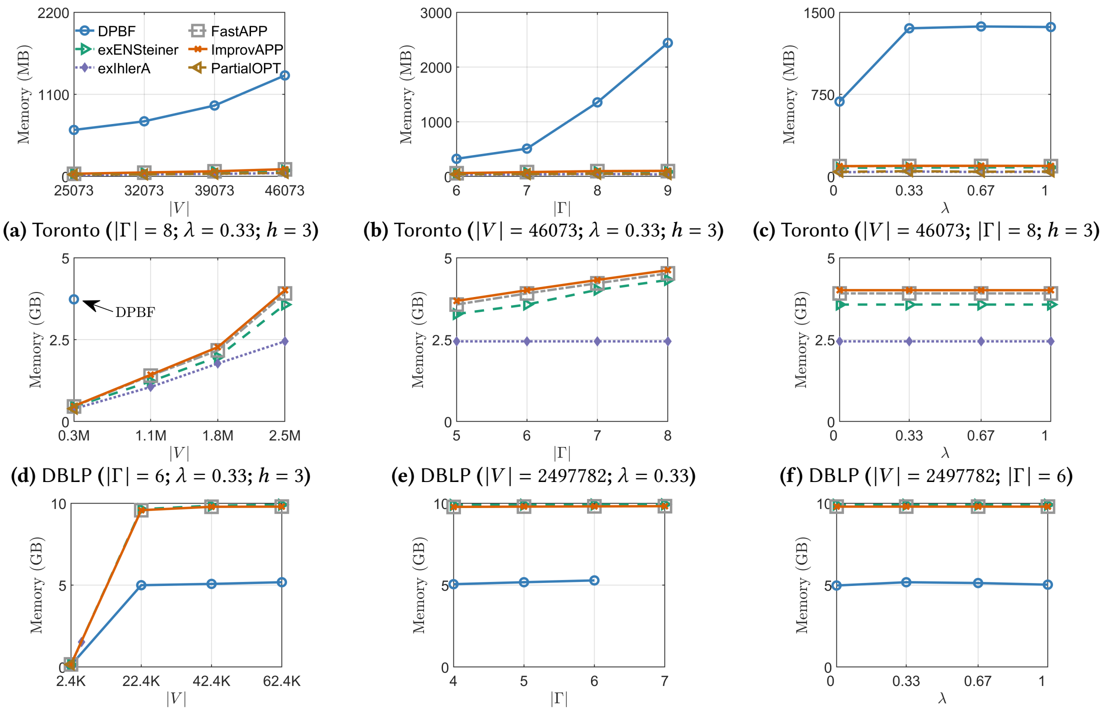


Figure 7: Comparing DPBF with Basic.



(a) Toronto ($|\Gamma| = 8$; $\lambda = 0.33$; $h = 3$) **(b)** Toronto ($|V| = 46073$; $\lambda = 0.33$; $h = 3$) **(c)** Toronto ($|V| = 46073$; $|\Gamma| = 8$; $h = 3$)

(d) DBLP ($|\Gamma| = 6$; $\lambda = 0.33$; $h = 3$) **(e)** DBLP ($|V| = 2497782$; $\lambda = 0.33$) **(f)** DBLP ($|V| = 2497782$; $|\Gamma| = 6$)

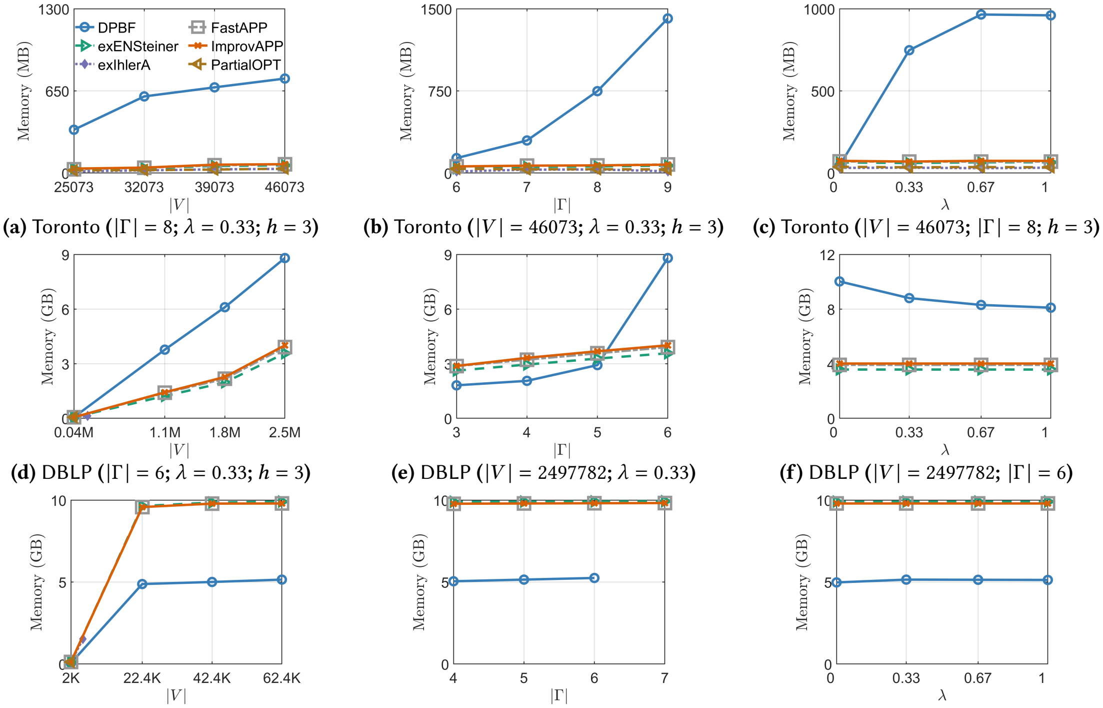
(g) MovieLens ($|\Gamma| = 5$; $\lambda = 0.33$; $h = 3$) **(h)** MovieLens ($|V| = 62423$; $\lambda = 0.33$) **(i)** MovieLens ($|V| = 62423$; $|\Gamma| = 5$)

Figure 8: The memory consumption in the main experiments where vertex groups are selected uniformly.

The main experiments. We report the memory consumption results in the main experiments in our paper in Figures 8 and 9. We observe that the memory consumption of DPBF grows exponentially with $|\Gamma|$ for Toronto and DBLP (*e.g.*, Figures 9b and 9e). The reason is that the memory consumed in the dynamic process of DPBF has a complexity of $O(2^{|\Gamma|}|V|)$. In comparison, the memory consumption of DPBF does not grow much with $|\Gamma|$ for MovieLens (*e.g.,* Figure 8h), since the MovieLens graph is dense, and as a result the memory consumed by the MovieLens graph dominates the $O(2^{|\Gamma|}|V|)$ memory consumed in the dynamic process of DPBF. Notably, for MovieLens, the memories consumed by exENSteiner, FastAPP and ImprovAPP are roughly twice the memory consumed by DPBF. The reason is that exENSteiner, FastAPP and ImprovAPP build an additional graph that has (i) the same set of edges with the input graph G and (ii) newly defined edge weights for finding lowest weight paths in G (see Section 3.2 in our paper on how to find lowest weight paths; further note that, even though ENSteiner does not build such an additional graph, ENSteiner consumes a similar amount of memory with exENSteiner for MovieLens in Figure 6, since the MovieLens graph in Figure 6 is small and sparse, and as a result the memory consumed by the additional graph built by exENSteiner is dominated by the other consumed memory). We further observe that DPBF consumes more memory when vertex groups are selected uniformly. For example, in Figure 8b, DPBF consumes around 2.5GB when $|\Gamma| = 9$, while in Figure 9b, DPBF consumes around 1.5GB when $|\Gamma| = 9$. The reason is that DPBF often enumerates more trees when vertex groups are selected uniformly.

Varying h in PartialOPT**.** We report the memory consumption of PartialOPT with respect to h in Figure 10, where the Toronto data is used, vertex groups are selected via the uniform approach, $|V| = 46073$, $|\Gamma| = 6$, $\lambda = 0.33$ (this corresponds to the experiments in Figure 7 in our paper). We observe that the memory consumed by PartialOPT grows exponentially with h. The reason is that PartialOPT employs DPBF to connect h vertex groups optimally, and the space complexity of this process is $O(2^h|V|)$.

Comparing LANCET **with** GKA**.** We compare the memory consumption of LANCET and GKA in Figure 11, where the Toronto data is used, vertex groups are selected via the uniform approach, $\lambda = 0.33$, $|\Gamma| = |T_t| = 6$ (this corresponds to the experiments in Figure 8 in our paper). We observe that the memory consumption of GKA increases quickly with $|V|$. The reason is that GKA stores the lowest weight paths between all pairs of vertices, which has a space complexity of $O(|V_t|^2)$, where $|V_t| = |V| + |\Gamma|$. In comparison, LANCET only stores the lowest weight paths from every compulsory vertex to the other vertices, which has a space complexity of $O(|T_t||V_t|)$ (see Line 3 of LANCET).



(a) Toronto ($|\Gamma| = 8$; $\lambda = 0.33$; $h = 3$) **(b)** Toronto ($|V| = 46073$; $\lambda = 0.33$; $h = 3$) **(c)** Toronto ($|V| = 46073$; $|\Gamma| = 8$; $h = 3$)

(d) DBLP ($|\Gamma| = 6$; $\lambda = 0.33$; $h = 3$) **(e)** DBLP ($|V| = 2497782$; $\lambda = 0.33$) **(f)** DBLP ($|V| = 2497782$; $|\Gamma| = 6$)

(g) MovieLens ($|\Gamma| = 5$; $\lambda = 0.33$; $h = 3$) **(h)** MovieLens ($|V| = 62423$; $\lambda = 0.33$) **(i)** MovieLens ($|V| = 62423$; $|\Gamma| = 5$)

Figure 9: The memory consumption in the main experiments where vertex groups are selected non-uniformly.

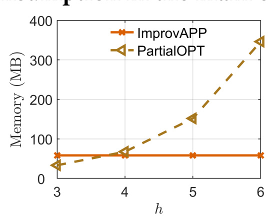


Figure 10: Varying h.

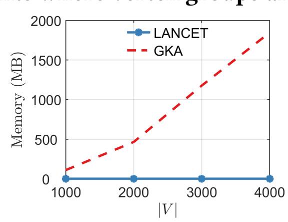


Figure 11: LANCET **versus** GKA**.**

9 REFINING THE SOLUTIONS OF exENSteiner, exIhlerA AND FastAPP

There is a solution refinement process in ImprovAPP, *i.e.,* Lines 17-29 in ImprovAPP. This refinement process refines a sub-optimal solution by removing non-unique-group leaves from this solution. Here, we use this process to refine the solutions of exENSteiner, exIhlerA and FastAPP. Notably, this process has already been incorporated into PartialOPT (*i.e.,* Line 13 in PartialOPT). Thus, we do not refine the solutions of PartialOPT here. We report the refinement results in Figures 12 and 13, where exENSteiner+R, exIhlerA+R and FastAPP+R are the refinements of exENSteiner, exIhlerA and FastAPP, respectively.

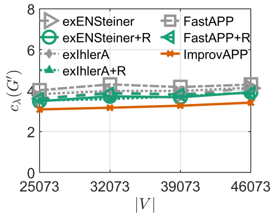
Suppose that there is a feasible solution tree $\Theta(V_\Theta, E_\Theta)$. Then, the time complexity of refining this solution is $O(|\Gamma||V_\Theta| + |V_\Theta| \log |V_\Theta|)$ (details in Section 4.3 in our paper). Since we generally have $|V_\Theta| \ll |V|$ in practice, the running times of refinement are negligible when comparing to the running times of our algorithms. For example, each of our algorithms takes around 100s to produce a feasible solution in the full DBLP graph, while it only takes around 2ms to refine this solution. Thus, we only evaluate the solution qualities in Figures 12 and 13, and do not evaluate the running times of refinement. We observe that ImprovAPP dominates exENSteiner+R, exIhlerA+R and FastAPP+R on solution qualities. We also observe that the refinement is often more effective when vertex groups are selected non-uniformly. For example, the refinement is more effective in Figure 13d than in Figure 12d. The reason is that, when vertex groups are selected non-uniformly, the sizes of the selected vertex groups are often larger, and as a result the leaves in the feasible solutions produced by exENSteiner, exIhlerA and FastAPP are more likely to be non-unique-group leaves. Nevertheless, we consider the refinement process as useful no matter vertex groups are selected uniformly or non-uniformly, given that the running times of the refinement are negligible.

REFERENCES

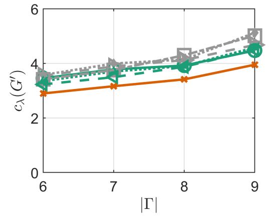
[1] Bolin Ding, Jeffrey Xu Yu, Shan Wang, Lu Qin, Xiao Zhang, and Xuemin Lin. 2007. Finding top-k min-cost connected trees in databases. In *IEEE International Conference on Data Engineering*. IEEE, 836–845.

[2] Rong-Hua Li, Lu Qin, Jeffrey Xu Yu, and Rui Mao. 2016. Efficient and progressive group Steiner tree search. In *Proceedings of the 2016 International Conference on Management of Data*. ACM, 91–106.

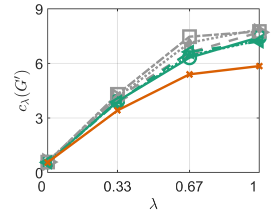
[3] Hiromitsu Takahashi and Akira Matsuyama. 1980. An approximate solution for the Steiner problem in graphs. *Math. Japonica* 24, 6 (1980), 573–577.



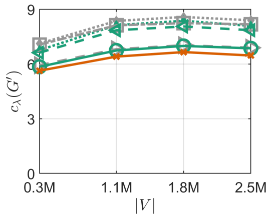
(a) Toronto ($|\Gamma| = 8$; $\lambda = 0.33$; $h = 3$)



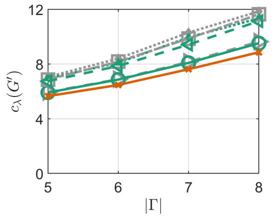
(b) Toronto ($|V| = 46073$; $\lambda = 0.33$; $h = 3$)



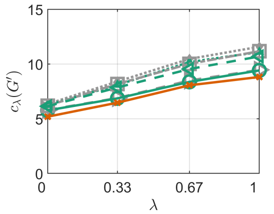
(c) Toronto ($|V| = 46073$; $|\Gamma| = 8$; $h = 3$)



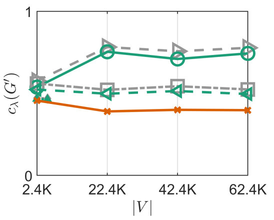
(d) DBLP ($|\Gamma| = 6$; $\lambda = 0.33$; $h = 3$)



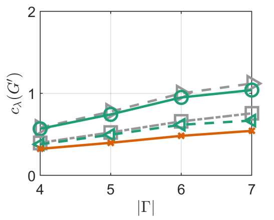
(e) DBLP ($|V| = 2497782$; $\lambda = 0.33$)



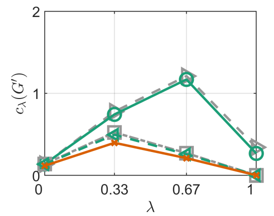
(f) DBLP ($|V| = 2497782$; $|\Gamma| = 6$)



(g) MovieLens ($|\Gamma| = 5$; $\lambda = 0.33$; $h = 3$)

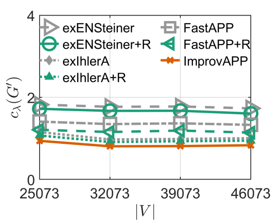


(h) MovieLens ($|V| = 62423$; $\lambda = 0.33$)

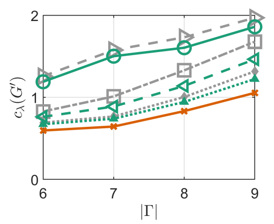


(i) MovieLens ($|V| = 62423$; $|\Gamma| = 5$)

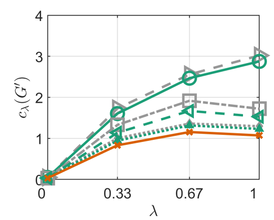
Figure 12: The refinement results in the main experiments where vertex groups are selected uniformly.



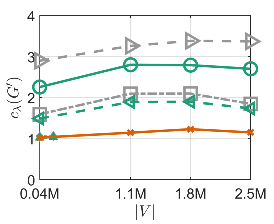
(a) Toronto ($|\Gamma| = 8$; $\lambda = 0.33$; $h = 3$)



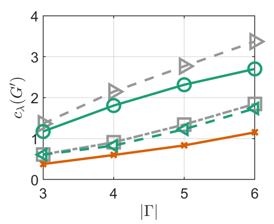
(b) Toronto ($|V| = 46073$; $\lambda = 0.33$; $h = 3$)



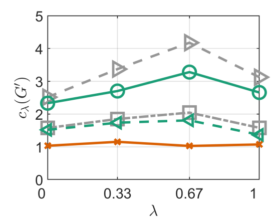
(c) Toronto ($|V| = 46073$; $|\Gamma| = 8$; $h = 3$)



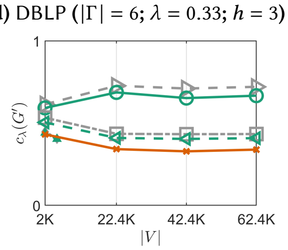
(d) DBLP ($|\Gamma| = 6$; $\lambda = 0.33$; $h = 3$)



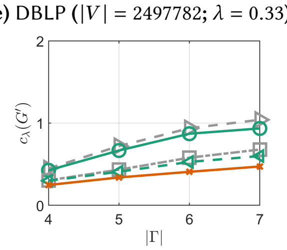
(e) DBLP ($|V| = 2497782$; $\lambda = 0.33$)



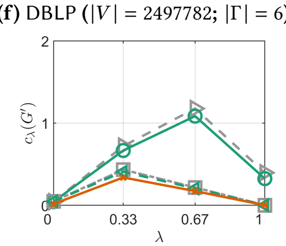
(f) DBLP ($|V| = 2497782$; $|\Gamma| = 6$)



(g) MovieLens ($|\Gamma| = 5$; $\lambda = 0.33$; $h = 3$)



(h) MovieLens ($|V| = 62423$; $\lambda = 0.33$)



(i) MovieLens ($|V| = 62423$; $|\Gamma| = 5$)

Figure 13: The refinement results in the main experiments where vertex groups are selected non-uniformly.