Indexing Shortest Paths Between Vertices and Vertex Groups in Weighted Knowledge Graphs — Supplemental Materials

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

- In Section S1, we present the proofs that are omitted in the main contents.
- In Section S2, we present the details of the extended Core-Tree-Decomposition process.
- In Section S3, we discuss the feasibility of extending the proposed techniques to directed graphs.
- In Section S4, we conduct additional experiments with different thread pool size and d.

S1. PROOFS

Theorem 1. *L^{EPSL} satisfies the 2-hop cover constraint.*

Proof. In EPSL, a hub $x \in C^{temp}_{>0}(u)$ (Line 17) is propagated from $v \in N(u)$ via edge (u, v) (Lines 6-7). Let s and t be an arbitrary pair of vertices. Let $x \in V$ be the vertex with the highest rank in all shortest paths between s and t. When EPSL propagates the hub x from itself to s along a shortest path between x and s, suppose that it meets $u \in V$. We have $r(x) > r(u)$, which means that the pruning condition in Line 8 is not satisfied. Since x is propagated along a shortest path to u, $d^{temp} = d(x, u)$ in Line 11. If there is $y \in C^{temp}_{<k}(u) \cup C^{temp}_{<k}(x)$ (Line 12) such that

$$d'_{min}(u, y) + d'(x, y) \leq d^{temp}, \tag{S1}$$

then y must be in a shortest path between u and x. However, since x is the vertex with the highest rank in all shortest paths between u and x, we have $r(y) < r(x)$, which means that y cannot be a hub of x due to the pruning condition in Line 8. Thus, there is no such y that meets pruning condition in Line 13. Therefore, EPSL can successfully propagates the hub x from itself to s along a shortest path between x and s, with a distance value of $d(x, s)$, and similarly, to t, with a distance value of $d(x, t)$. Since the distance values in L^{temp} are no smaller than shortest distance values, the hub x will be inserted into both $L^{EPSL}(s)$ and $L^{EPSL}(t)$ in Line 25, with the distance value of $d(x, s)$ and $d(x, t)$, respectively. Thus, in the end of EPSL, there is a vertex $x \in C^{EPSL}(s) \cap C^{EPSL}(t)$ that is in a shortest path between s and t, and we can use L^{EPSL} to query the shortest distance or path between s and t. Hence, this theorem holds. □

Theorem 2. *In a weighted graph $G(V, E, w)$, for a pair of vertices s and t such that $s \neq t$, we have*

$$d(s,t) = \begin{cases} d(f(s), f(t)), & if\ f(s) \neq f(t) \\ \min_{v_i \in N(s)}\{w(s,t), 2w(s,v_i)\}, & if\ f(s) = f(t), \end{cases} \tag{S2}$$

where $w(s, t) = \infty$ when $(s, t) \notin E$.

Proof. There are three cases as follows.
Case 1: $s, t \in V_3$. We have $f(s) = s$, $f(t) = t$, $d(s, t) = d(f(s), f(t))$. Since $s \neq t$, $f(s) \neq f(t)$. This theorem holds.
Case 2: $s \in V_3$ and $t \in V \setminus V_3$ (symmetrically, $t \in V_3$ and $s \in V \setminus V_3$). We have $f(s) = s$, $f(s) \neq t$, and $f(s) \neq f(t)$. $t \in V_1$ or V_2 means that $t \simeq_1 f(t)$ or $t \simeq_2 f(t)$, which further means that $d(v_i, t) = d(v_i, f(t))$ for each v_i that is not t or $f(t)$. Since $f(s) = s$ is not t or $f(t)$, $d(s, t) = d(f(s), f(t))$. This theorem holds.
Case 3: $s, t \in V \setminus V_3$. There are two sub-cases. Sub-case 1: $f(s) = f(t)$. We have $s \simeq_1 t$ or $s \simeq_2 t$. If $s \simeq_1 t$, then $P_s(s, t)$ contains two edges (s, v) and (v, t) such that $w(s, v) = w(v, t) = \min_{v_i \in N(s)}\{w(s, v_i)\}$. If $s \simeq_2 t$, then $P_s(s, t)$ either contains two edges (s, v) and (v, t) or contains a single edge (s, t). This theorem holds. Sub-case 2: $f(s) \neq f(t)$. We have s is not t or $f(t)$, and t is not s or $f(s)$. Since $d(v_i, t) = d(v_i, f(t))$ for each v_i that is not t or $f(t)$; and symmetrically $d(v_i, s) = d(v_i, f(s))$ for each v_i that is not s or $f(s)$, we have $d(s, t) = d(f(s), f(t))$. This theorem holds. □

Lemma 1. $L^{can} \subseteq L^{EPSL}$.

Proof. For a vertex $u \in V$, consider a label $(x, d(u,x), p_{ux}) \in L^{can}(u)$. Suppose that this label corresponds to a shortest path $\{x, v_1, \cdots, v_k, u\}$. For each vertex in this path, the rank of x is the highest among all vertices in all shortest paths between this vertex and x. We explain that $(x, d(x,v_1), p_{v_1 x})$ is inserted into $L_1^{temp}(v_1)$ when $d = 1$ in the while loop of EPSL as follows. When EPSL tries to insert $(x, d(x,v_1), p_{v_1 x})$ into $L_1^{temp}(v_1)$ in the while loop, since $r(x) > r(v_1)$, the pruning condition in Line 8 of EPSL is not met. If the pruning condition in Line 13 of EPSL is met, then there is a vertex y that is in a shortest path between x and v_1, and y is a hub of both x and v_1 in L^{temp}, which means that $r(y) > \max\{r(x), r(v_1)\}$, due to the pruning condition in Line 8. However, this contradicts with the assumption that the rank of x is the highest among all vertices in all shortest paths between v_1 and x. Thus, the pruning condition in Line 13 of EPSL is not met when EPSL tries to insert $(x, d(x,v_1), p_{v_1 x})$ into $L_1^{temp}(v_1)$ in the while loop. As a result, EPSL successfully inserts $(x, d(x,v_1), p_{v_1 x})$ into $L_1^{temp}(v_1)$. Similarly and iteratively, EPSL inserts $(x, d(u,x), p_{ux})$ into $L_d^{temp}(u)$. Given that $d(u,x)$ is the shortest distance between x and u, EPSL also inserts $(x, d(u,x), p_{ux})$ into $L^{EPSL}(u)$ in Line 25. Hence, each label in L^{can} is also in L^{EPSL}. This lemma holds. □

Lemma 2. *Given a graph $G(V,E,w)$ and any set L of 2-hop labels such that $L^{can} \subseteq L$, after performing the procedure $L' = CanonicalFix(L)$, we have $L' = L^{can}$.*

Proof. For a vertex $u \in V$, consider a label $(v, d(u,v), p_{uv}) \in L^{can}(u) \subseteq L(u)$. When the $CanonicalFix$ procedure computes $d'(u,v)$ in Line 6 of CPSL, if $d'(u,v) \leq d(u,v)$, then there is a vertex $y \in C_{>r(v)}(u) \cap C(v)$ that is in a shortest path between u and v, and $r(y) > r(v)$. However, this contradicts with the fact that the rank of v is the highest among all vertices in all shortest paths between u and v. Thus, the procedure inserts $(v, d(u,v), p_{uv})$ into $L'(u)$ in Line 8 of CPSL. On the other hand, consider another label $(x, d(u,x), p_{ux}) \in L(u) \setminus L^{can}(u)$. Let $z \in V$ be the vertex with the highest rank among all vertices in all shortest paths between u and x. We have $r(z) > r(x)$, and $z \in C_{>r(x)}(u) \cap C(x)$. As a result, the procedure computes $d'(u,x) = d(u,x) = d(u,z) + d(z,x)$, and does not insert $(x, d(u,x), p_{ux})$ into $L'(u)$. Hence, this lemma holds. □

Lemma 3. $L^{CL} = L^{can}$.

Proof. Suppose that the customized pruning condition in Line 13 has been removed. If we can prove that $L^{can} \subseteq L^{temp}$ in Line 22, then, by Lemma 2, this lemma holds. We prove that $L^{can} \subseteq L^{temp}$ in Line 22 as follows. Consider a label $(u, d(v), p_v) \in L^{can}(v)$. We have $r(u) > r(v)$. In the Dijkstra-style search from u in CL, v will be popped out of Q with the priority value of $d(v)$ and the predecessor value of p_v. If $Query(u, v, L^{temp}) \leq d(v)$ in Line 7, then there is a vertex $y \in C^{temp}(u) \cap C^{temp}(v)$ that is in a shortest path between u and v and $r(y) > \max\{r(u), r(v)\}$, which contradicts with the assumption that u has the highest rank among all vertices in all shortest paths between u and v. Thus, we have $Query(u, v, L^{temp}) > d(v)$ in Line 7, and the label $(u, d(v), p_v)$ is inserted into $L^{temp}(v)$ in Line 10. Hence, we have $L^{can} \subseteq L^{temp}$ in Line 22, and then by Lemma 2, this lemma holds. □

Theorem 4. *L^{P-CL} satisfies the enhanced 2-hop cover constraint.*

Proof. Lemma 3 shows that L^{CL} meets the 2-hop cover constraint. For every pair of vertices v_i and v_j such that $|\{v_i, v_j\} \cap D| < 2$, there is a vertex $u \in C^{CL}(v_i) \cap C^{CL}(v_j)$ that is in a shortest path between v_i and v_j, *i.e.*, there are two labels $(u, d(u,v_i), p_{v_i u}) \in L^{CL}(v_i)$ and $(u, d(u,v_j), p_{v_j u}) \in L^{CL}(v_j)$. Since $\max\{d(u,v_i), d(u,v_j)\} < 2M$, these two labels are not pruned by the customized pruning condition in Line 13 of P-CL, and are in L^{P-CL}. Hence, this theorem holds. □

Theorem 5. *L^{P-CL} satisfies the enhanced canonical constraint.*

Proof. For a pair of vertices u and v, if $|\{u,v\} \cap D| = 2$, then $d(u,v) \geq 2M$, and $v \notin C^{P-CL}(u)$. Subsequently, if $|\{u,v\} \cap D| < 2$, then since $d(u,v) < 2M$, $v \in C^{P-CL}(u)$ if and only if $v \in C^{CL}(u)$. Lemma 3 shows that $v \in C^{P-CL}(u)$ if and only if the rank of v is the highest among all vertices in all shortest paths between u and v. Thus, when $|\{u,v\} \cap D| < 2$, $v \in C^{P-CL}(u)$ if and only if the rank of v is the highest among all vertices in all shortest paths between u and v. This theorem holds. □

Algorithm S1. The extended Core-Tree-Decomposition procedure

Input: a transformed graph $G(V, E, w)$, a parameter d
Output: a set I_T of tree indexes, a core graph $G_{\lambda+1}$

1: $I_T = \emptyset$, $G_0 = G$; $\overrightarrow{Z(x,y)} = y$, $\overrightarrow{Z(y,x)} = x$ for each $(x,y) \in E$
2: **for** i from 1 to n **do**
3: $\quad u_i \leftarrow$ the vertex with the minimum degree in G_{i-1}
4: $\quad N_i \leftarrow$ the neighbor set of u_i in G_{i-1}
5: $\quad B_i = \{u_i\} \cup N_i$
6: $\quad \delta_i^-(x) = w(u_i, x)$ for each $x \in N_i$ // $w(u_i, x)$ is an edge weight in G_{i-1}
7: $\quad G_i = G_{i-1}$
8: $\quad$ **if** $|N_i| \geq d$ **then**
9: $\quad\quad \lambda = i - 1$ // $G_{\lambda+1}$ is the above G_i
10: $\quad\quad B^c = V \setminus \{u_1, \cdots, u_\lambda\}$ // $u_{\lambda+1}$ is in the core
11: $\quad\quad$ Break
12: $\quad V_i = V_{i-1} \setminus \{u_i\}$ // remove u_i and its adjacent edges from G_i
13: $\quad$ **for** each pair of vertices x and y in N_i **do**
14: $\quad\quad$ **if** $(x,y) \notin E_i$ **then** // E_i is the set of edges in G_i
15: $\quad\quad\quad$ Insert (x,y) into E_i with the weight of $w(x,u_i) + w(y,u_i)$
16: $\quad\quad\quad \overrightarrow{Z(x,y)} = \overrightarrow{Z(x,u_i)}$, $\overrightarrow{Z(y,x)} = \overrightarrow{Z(y,u_i)}$
17: $\quad\quad$ **else if** $w(x,u_i) + w(y,u_i) < w(x,y)$ **then**
18: $\quad\quad\quad$ Update $w(x,y) = w(x,u_i) + w(y,u_i)$ in G_i
19: $\quad\quad\quad \overrightarrow{Z(x,y)} = \overrightarrow{Z(x,u_i)}$, $\overrightarrow{Z(y,x)} = \overrightarrow{Z(y,u_i)}$
20: **for** i from $\lambda + 2$ to n **do**
21: $\quad u_i \leftarrow$ the vertex with the $(i-\lambda)^{th}$ smallest degree in $G_{\lambda+1}$
22: **for** i from λ downto 1 **do**
23: $\quad p(i) = \min_{u_j \in N_i} j$
24: $\quad$ **if** $p(i) > \lambda$ or $N_i = \emptyset$ **then**
25: $\quad\quad r_i = i$
26: $\quad\quad p(i) = -1$
27: $\quad\quad \delta^T(u_i, x) = \delta_i^-(x)$ for each $x \in N_i$
28: $\quad\quad$ Insert $\{x, \delta^T(u_i,x), \overrightarrow{Z(u_i,x)}\}$ into $I_T(u_i)$ for each $x \in N_i$
29: $\quad$ **else**
30: $\quad\quad r_i = r_{p(i)}$
31: $\quad\quad A = \emptyset$
32: $\quad\quad t = i$
33: $\quad\quad$ **while** $p(t)! = -1$ **do**
34: $\quad\quad\quad A = A \cup \{u_{p(t)}\}$
35: $\quad\quad\quad t = p(t)$
36: $\quad\quad A = A \cup \{u_{p(t)}\}$
37: $\quad\quad N_i^T = N_i \cap A$ // equivalent to $N_i^T = N_i \setminus B^c$
38: $\quad\quad$ **for** each $x \in A \cup N_{r_i}$ **do**
39: $\quad\quad\quad \delta^T(u_i,x) = \min\{\delta_i^-(x), \min_{u_j \in N_i^T} \delta_i^-(u_j) + \delta^T(u_j,x)\}$ // $\delta_i^-(x) = \infty$ for each $x \notin N_i$
40: $\quad\quad\quad$ **if** $\delta_i^-(x) > \min_{u_j \in N_i^T} \delta_i^-(u_j) + \delta^T(u_j,x)$ **then**
41: $\quad\quad\quad\quad \overrightarrow{Z(u_i,x)} = \overrightarrow{Z(u_i,u_j)}$
42: $\quad\quad\quad$ Insert $\{x, \delta^T(u_i,x), \overrightarrow{Z(u_i,x)}\}$ into $I_T(u_i)$ for each $x \in A \cup N_{r_i}$
43: Return $\{I_T, G_{\lambda+1}\}$

S2. THE EXTENDED CORE-TREE-DECOMPOSITION PROCESS

Here, we extend the Core-Tree decomposition method in [3]. The only difference is that we record the predecessor information in tree indexes. We present the extend method as Algorithm S1. Given a transformed graph $G(V, E, w)$ and a parameter d, the algorithm returns a set I_T of tree indexes and a core graph $G_{\lambda+1}$. First, it initializes $I_T = \emptyset$, $G_0 = G$, and $\overrightarrow{Z(x,y)} = y$, $\overrightarrow{Z(y,x)} = x$ for each $(x,y) \in E$ (Line 1), where $\overrightarrow{Z(x,y)}$ is the predecessor of x to y via edge (x,y), which is originally y and may be updated later.

Then, for i from 1 to n (Line 2), it lets u_i be the vertex with the minimum degree in G_{i-1} (Line 3), N_i be the neighbor set of u_i in G_{i-1} (Line 4), $B_i = \{u_i\} \cup N_i$ (Line 5), and $\delta_i^-(x) = w(u_i, x)$ for each $x \in N_i$ (Line 6). Subsequently, it lets $G_i = G_{i-1}$ (Line 7). If $|N_i| \geq d$ (Line 8), it lets $\lambda = i - 1$ (Line 9), which means that $G_{\lambda+1}$ is the above G_i, and lets $B^c = V \setminus \{u_1, \cdots, u_\lambda\}$, and breaks the for loop (Line 11). Otherwise, it removes u_i and its adjacent edges from G_i (Line 12). For each pair of vertices x and y in N_i (Line 13), if $(x,y) \notin E_i$ (Line 14), then it inserts (x,y) into E_i with the weight of $w(x,u_i) + w(y,u_i)$ (Line 15), or if $(x,y) \in E_i$ and $w(x,u_i) + w(y,u_i) < w(x,y)$ (Line 17), it updates $w(x,y) = w(x,u_i) + w(y,u_i)$ in G_i (Line 18). In the above two cases, it also updates $\overrightarrow{Z(x,y)} = \overrightarrow{Z(x,u_i)}$, $\overrightarrow{Z(y,x)} = \overrightarrow{Z(y,u_i)}$ (Lines 16 and 19).

For i from $\lambda+2$ to n (Line 20), it lets u_i be the vertex with the $(i-\lambda)^{th}$ smallest degree in $G_{\lambda+1}$, *e.g.*, $u_{\lambda+2}$ is the vertex with the second smallest degree in $G_{\lambda+1}$ ($u_{\lambda+1}$ is the vertex with the smallest degree in $G_{\lambda+1}$). Subsequently, for i from λ downto 1 (Line 22), it lets $p(i) = \min_{u_j \in N_i} j$ (Line 23), *i.e.*, the ID of the parent of B_i in the decomposed tree. If $p(i) > \lambda$ or $N_i = \emptyset$ (Line 24), which means that B_i is the root of a decomposed tree, then it lets $r_i = i$ (Line 25), $p(i) = -1$ (Line 26), $\delta^T(u_i, x) = \delta_i^-(x)$ for each $x \in N_i$ (Line 27). It further generates the set $I_T(u_i)$ of tree indexes, which contains a hub label $\{x, \delta^T(u_i, x), \overrightarrow{Z(u_i, x)}\}$ for each $x \in N_i$ (Line 28). If B_i is not a root (Line 29), then it lets $r_i = r_{p(i)}$ (Line 30), $A = \emptyset$ (Line 31). It computes A as the set of ancestors of u_i in the decomposed tree as follows. It lets $t = i$ (Line 32). While $p(t)! = -1$ (Line 33), it lets $A = A \cup \{u_{p(t)}\}$ (Line 34) and $t = p(t)$ (Line 35). After the while loop, $u_{p(t)}$ is the root of u_i. It lets $A = A \cup \{u_{p(t)}\}$ (Line 36). After generating the set A of ancestors of u_i, it lets $N_i^T = N_i \cap A$ (Line 37), which is equivalent to $N_i^T = N_i \setminus B^c$. For each $x \in A \cup N_{r_i}$ (Line 38), it computes $\delta^T(u_i, x) = \min\{\delta_i^-(x), \min_{u_j \in N_i^T} \delta_i^-(u_j) + \delta^T(u_j, x)\}$ (Line 39), where $\delta_i^-(x) = \infty$ for each $x \notin N_i$. If $\delta_i^-(x) > \min_{u_j \in N_i^T} \delta_i^-(u_j) + \delta^T(u_j, x)$ (Line 40), it updates $\overrightarrow{Z(u_i, x)} = \overrightarrow{Z(u_i, u_j)}$ (Line 41). Then, it generates the set $I_T(u_i)$ of tree indexes, which contains a hub label $\{x, \delta^T(u_i, x), \overrightarrow{Z(u_i, x)}\}$ for each $x \in A \cup N_{r_i}$ (Line 42). After generating the set I_T of tree indexes, it returns $\{I_T, G_{\lambda+1}\}$.

S3. EXTENDING THE PROPOSED TECHNIQUES TO DIRECTED GRAPHS

Like the related work (*e.g.*, [2, 3]), we can extend the proposed techniques to directed cases. First, note that the dummy transformation method can be used in directed graphs. For example, to find the directed shortest path from $v_1 \in V$ to $\{v_2, v_3\}$, we can add a dummy vertex v_d and directed dummy edges (v_2, v_d) and (v_3, v_d) with the same weight, and then find the directed shortest path from v_1 to v_d, which contains the directed shortest path from v_1 to $\{v_2, v_3\}$.

As discussed in [2, 4], we can extend 2-hop labels to directed cases in the following way. We associate each vertex $v \in V$ with a set of in-direction labels $L_{IN}(v)$ and a set of out-direction labels $L_{OUT}(v)$ such that each hub in $L_{IN}(v)$ can reach v, and v can reach each hub in $L_{OUT}(v)$. Then, we use the following equation to query the directed shortest distance from s to t.

$$d(s,t) = \min_{u \in C_{OUT}(s) \cap C_{IN}(t)} d(s,u) + d(u,t). \tag{S3}$$

The canonical hub labeling idea works in directed graphs as well. Like the method of extending PSL to directed graphs [2], we can extend EPSL to generate directed hub labels in the following way: in each iteration of the while loop of EPSL, we compute $L_{OUT\ k}^{temp}(u)$ and $L_{IN\ k}^{temp}(u)$, while pruning labels based on the above equation. The extended reduction techniques can also be further extended to directed cases. In particular, we consider edge directions when checking whether there is an equivalence relation between u and v for the equivalence relation reduction, as well as when querying shortest distances and paths with reductions. Moreover, as discussed in [3], the Core-Tree decomposition method can also be extended to directed cases. Furthermore, the $CanonicalFix$ procedure can prune non-canonical directed hub labels. Note that, we use the above equation to query a distance when checking whether a directed label should be pruned or not. In the end, like the method of extending PLL to directed graphs [4], we can extend P-CL to generate directed hub labels as follows. For each $v \in V$, perform two Dijkstra-style searches from v, once on the directed G, and once on the reverse of G, where edge directions are the reverses of those in G, for generating L_{IN}^{temp} and L_{OUT}^{temp}, respectively.

S4. ADDITIONAL EXPERIMENT RESULTS

In the main experiments, we set the size of thread pool to 80 (the computer has 96 threads in total), and set d to 20. Here, we conduct additional experiments where these two parameters are set to different values.

First, we decrease the size of thread pool to 50 (while keeping d to 20), and present the additional experiment results in Figure S1. We observe that, in comparison with the main experiment results, the times of generating indexes are slightly larger in Figure S1. For example, CT-CPSL* takes less than 60s to index the Musae graph with random edge weights in the main experiments, while it takes more than 60s to do so in Figure S1.

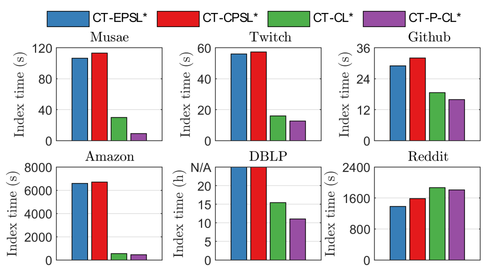
Second, we increase d to 50 (while keeping the size of thread pool to 80), and present the additional experiment results in Figure S2. We observe that, in comparison with the main experiment results, the times of generating indexes are slightly larger in Figure S2. For example, the four algorithms take less than 18s to index the Github graph with random edge weights in the main experiments, while these algorithms take more than 30s to do so in Figure S2. Moreover, in comparison with the main experiment results, the times of querying shortest paths are significantly larger in Figure S2. For example, the four algorithms take less than 2.4ms to query shortest paths in the Amazon graph with Jacard edge weights in the main experiments, while these algorithms take more than 5ms to do so in Figure S2. The reason is that a larger d induces a large part of tree indexes in Core-Tree indexes, and it is generally slow to query shortest distances or paths using tree indexes [3].

Nevertheless, the key experiment observations in the main experiments also hold in the above additional experiments. The details are as follows.

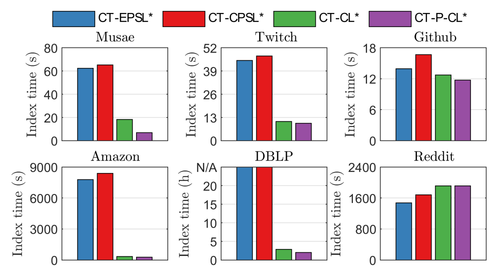
- In comparison with the baseline CT-EPSL*, the proposed CT-P-CL* can be up to an order of magnitude faster to generate indexes (*e.g.*, for Amazon in Figure S1a), save up to a half of memory spaces (*e.g.*, for Musae in Figure S1d), and be generally two or three times faster to query shortest paths between vertices and vertex groups (*e.g.*, for Github in Figure S1f). An exception is that CT-P-CL* is slower than CT-EPSL* to generate indexes for Reddit in Figures S1a-S1b, which indicates that the lengths of paths are generally proportional to the numbers of edges in these paths for Reddit, as discussed before.
- The improved baseline CT-CPSL* obtains similar performances with CT-P-CL* in some cases, *e.g.*, for Reddit in Figures S1-S2, CT-CPSL* is slightly faster than CT-P-CL* to generate indexes, while consuming similar amounts of memory and having similar query speeds with CT-P-CL*.

REFERENCES FOR THE SUPPLEMENT

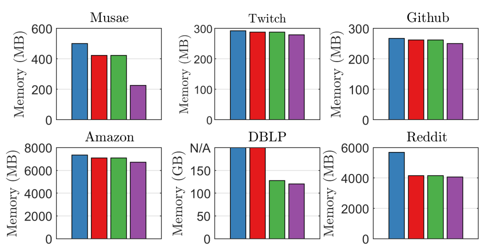
1. "The supplement," (2021). https://github.com/anonym-opensource/SP/blob/main/Supplement.pdf.
2. W. Li, M. Qiao, L. Qin, Y. Zhang, L. Chang, and X. Lin, "Scaling distance labeling on small-world networks," in *Proceedings of the 2019 International Conference on Management of Data,* (2019), pp. 1060–1077.
3. W. Li, M. Qiao, L. Qin, Y. Zhang, L. Chang, and X. Lin, "Scaling up distance labeling on graphs with core-periphery properties," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data,* (2020), pp. 1367–1381.
4. T. Akiba, Y. Iwata, and Y. Yoshida, "Fast exact shortest-path distance queries on large networks by pruned landmark labeling," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data,* (2013), pp. 349–360.



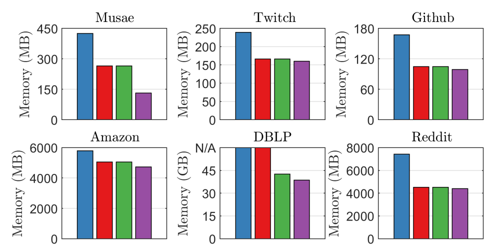
(a) The times of generating indexes (Jacard).



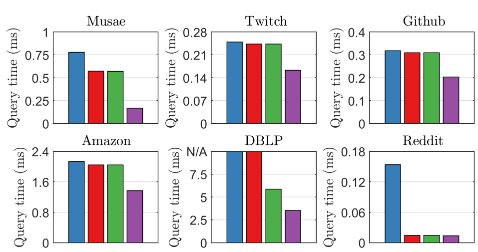
(b) The times of generating indexes (Random).



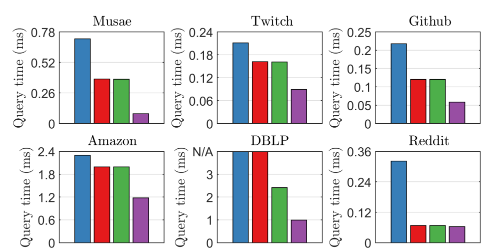
(c) The memory consumption of indexes (Jacard).



(d) The memory consumption of indexes (Random).

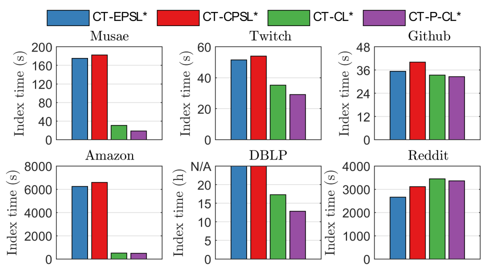


(e) The times of querying shortest paths (Jacard).

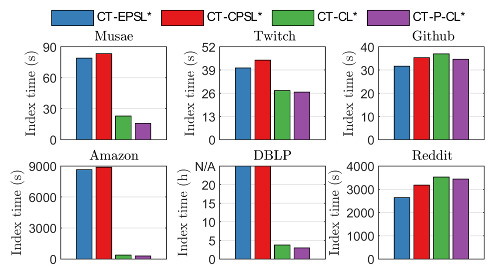


(f) The times of querying shortest paths (Random).

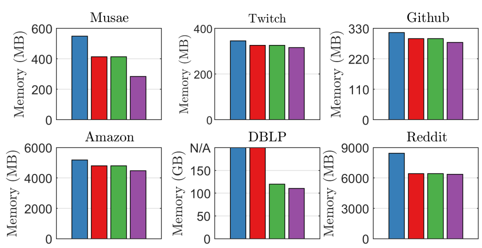
Fig. S1. Additional experiment results of 50 threads.



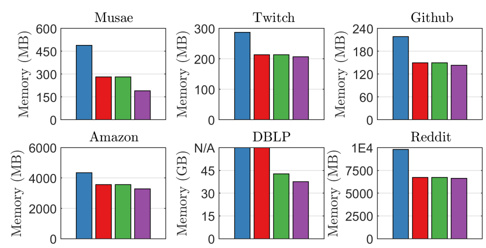
(a) The times of generating indexes (Jacard).



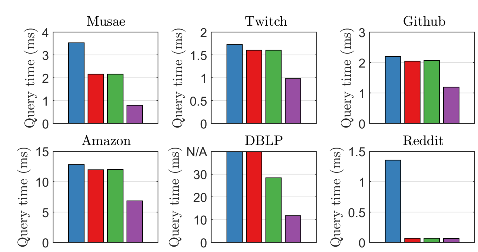
(b) The times of generating indexes (Random).



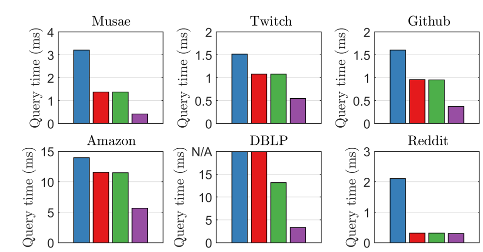
(c) The memory consumption of indexes (Jacard).



(d) The memory consumption of indexes (Random).



(e) The times of querying shortest paths (Jacard).



(f) The times of querying shortest paths (Random).

Fig. S2. Additional experiment results of $d = 50$.