Fast Maintenance of 2-hop Labels for Shortest Distance Queries on Fully Dynamic Graphs — Supplemental Materials

The road map of this supplement is as follows.

- In Section S1, we prove the correctness of FastDeM and FastInM.
- In Section S2, we provide detailed discussions on time complexities.
- In Section S3, we prove that CLN can eliminate all redundant indexes.
- In Section S4, we show the efficiency of CLN.

S1. THE CORRECTNESS OF FastDeM AND FastInM

We use the following two theorems to show the correctness of FastDeM and FastInM.

Theorem 1. *Given a graph G, a set L of labels that satisfies the 2-hop cover constraint, and the corresponding PPR, suppose that $w(a,b)$ decreases from w_0 to w_1, then* FastDeM *can maintain L & PPR such that we can use the maintained labels to correctly query the shortest distance between every pair of vertices in the updated graph.*

Proof. Consider a pair of vertices s and t, we use $p(s,t)$ and $p'(s,t)$ to denote a shortest path between s and t before and after the edge weight change, respectively. Let u and u' be the vertices with the highest rank in all shortest paths between s and t before and after the change, respectively, and $u \in p(s,t)$ and $u' \in p'(s,t)$. We prove that we can use the maintained L to correctly query the shortest distance between s and t in the updated graph as follows.

There are two cases: 1) $(a,b) \in p'(s,t)$; and 2) $(a,b) \notin p'(s,t)$.

Consider the first case: $(a,b) \in p'(s,t)$. Without loss of generality, suppose that a is closer to s than b along $p'(s,t)$, and u' is between (a,b) and t, as illustrated in Figure S1 (c). We have $d(t,u') = d'(t,u')$, and u' is the vertex with highest rank in all shortest paths between u' and t both before and after the change. Thus, hub u' spreads from itself to t along $p'(u',t)$ both before and after the change, as otherwise there must be a vertex with a higher rank than u' in a shortest path between u' and t that can prune this spread. Therefore, $(u', d'(t,u')) \in L(t)$ both before and after the change. Similarly, $(u', d'(b,u')) \in L(b)$ both before and after the change. Moreover, FastDeM calls $DIFFUSE$ to re-spread hub u' from b to s along $p'(s,t)$ due to the change. Thus, $(u', d'(s,u')) \in L(s)$ after the maintenance, and we can use the maintained L to correctly query $d'(s,t)$.

Consider the second case: $(a,b) \notin p'(s,t)$. We have $d(s,t) = d'(s,t)$, and u' is the vertex with the highest rank in all shortest paths between s and t both before and after the change. Assume that u' does not spread from u' to t along $p'(u',t)$, and let x be the vertex farthest to u' along $p'(u',t)$ such that u' spreads from u' to x along $p'(u',t)$, and also let y be the neighbor of x along $p'(u',t)$ such that u' does not spread from u' to y along $p'(u',t)$, as shown in Figure S1 (d). To ensure that u' does not spread to y along $p'(u',t)$, there must be a vertex z such that $z \in C(u') \cap C(y)$, $r(z) > r(u')$, and z is in a shortest path between u' and y. This contradicts with the assumption that u' has the highest rank in all shortest paths between s and t. Thus, u' spreads to y, and ultimately to t, along $p'(u',t)$. Similarly, u' also spreads to s along $p'(u',s)$. Therefore, we can use the maintained L to correctly query $d'(s,t)$. Hence, this theorem holds. □

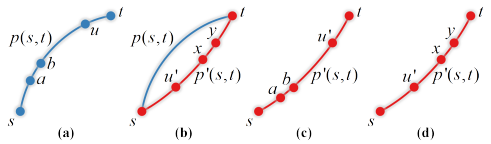

Fig. S1. Some illustrations for the correctness proofs.

Theorem 2. *Given a graph G, a set L of labels that satisfies the 2-hop cover constraint, and the corresponding PPR, suppose that $w(a,b)$ increases from w_0 to w_1, then* FastInM *can maintain L and PPR such that we can use the maintained labels to correctly query the shortest distance between every pair of vertices in the updated graph.*

Proof. Like the above proof, there are two cases: 1) $(a,b) \in p'(s,t)$; and 2) $(a,b) \notin p'(s,t)$.

Consider the first case: $(a,b) \in p'(s,t)$. We have $p(s,t) = p'(s,t)$ and $u = u'$. Without loss of generality, suppose that a is closer to s than b along $p(s,t)$, and u is between (a,b) and t, as shown in Figure S1 (a). Before the change, if $u \notin C(s)$, then there must be a vertex $x \in C(s) \cap C(u)$ such that $r(x) > r(u)$ and x is in a shortest path between s and t. This contradicts with the assumption that u has the highest rank in all shortest paths between s and t. Thus, $u \in C(s)$, and similarly $u \in C(t)$. Specifically, $(u, d(s,u)) \in L(s)$ and $(u, d(t,u)) \in L(t)$ before the maintenance, where $d(s,u)$ and $d(t,u)$ are the shortest distances between s and u, and between t and u, respectively, before the change. Similarly, let $d'(s,u)$ and $d'(t,u)$ be the distance values after the change. Since $d(s,u) < d'(s,u)$, FastInM first calls $SPREAD_1$ and $SPREAD_2$ to deactivate $(u, d(s,u)) \in L(s)$, and then calls $SPREAD_3$ to update this label to $(u, d'(s,u)) \in L(s)$. On the other hand, since $d(t,u) = d'(t,u)$, FastInM keeps $(u, d(t,u)) \in L(t)$. Hence, we can use the maintained L to correctly query $d'(s,t)$ in the updated graph.

Consider the second case: $(a,b) \notin p'(s,t)$. Before the change, assume that hub u' does not spread from u' to t along $p'(u',t)$, and let x be the vertex farthest to u' along $p'(u',t)$ such that hub u' spreads from u' to x along $p'(u',t)$, and also let y be the neighbor of x along $p'(u',t)$ such that u' does not spread from u' to y along $p'(u',t)$, as shown in Figure S1 (b). To ensure that u' does not spread from u' to y along $p'(u',t)$, there is a vertex z such that $z \in C(u') \cap C(y)$, $r(z) > r(u')$, z is in a shortest path between u' and y before the change, and $u' \in PPR[y,z]$ and $y \in PPR[u',z]$. Since u' is the vertex with the highest rank in all shortest paths between u' and y after the change, z cannot prune the spread of hub u' to y along $p'(u',t)$ any more. Since the length of $p'(u',y)$ does not change due to the graph change, either $L(u')[z]$ or $L(y)[z]$ increases after the graph change, otherwise $z \in C(u') \cap C(y)$ still prunes the spread of hub u' to y along $p'(u',t)$. Thus, FastInM must calls $SPREAD_1$ to set either $L(u')[z]$ or $L(y)[z]$ to ∞. In either case, using the above PPR information, FastInM calls $SPREAD_2$ and $SPREAD_3$ to re-spread the hub u' from x to y, and ultimately to t, along $p'(s,t)$. Similarly, after the maintenance, hub u' also spreads from u' to s along $p'(s,t)$. Thus, we can use the maintained L to correctly query $d'(s,t)$ in the updated graph. Hence, this theorem holds. □

S2. THE DISCUSSIONS ON TIME COMPLEXITIES

The time complexity of FastDeM**:** FastDeM has a cost of

$$O\Big(\delta^2 + \Upsilon \cdot (\log \Upsilon + d_a \cdot \delta)\Big).$$

The reason is that populating CL takes $O(\delta^2)$ time, and $DIFFUSE$ takes $O(\Upsilon \cdot (\log \Upsilon + d_a \cdot \delta))$ time, assumed that labels are stored in hashes. Particularly, $DIFFUSE$ pops $O(\Upsilon)$ elements out of the priority queue. Each pop operation takes $O(\log \Upsilon)$ times. After each pop operation, it searches $O(d_a)$ neighbors, and a distance query that costs $O(\delta)$ may be conducted in each of these searches.

The time complexity of FastInM**:** FastInM has a cost of

$$O\Big(\Upsilon \cdot \Big(\log \Upsilon + d_a \cdot \delta + \kappa \cdot (d_a + \delta)\Big)\Big),$$

where κ is the average number of PPR elements of each vertex-hub pair. The details are as follows. First, FastInM pushes labels into AL_1 in $O(\delta)$ time. Then, it performs $SPREAD_1$ in $O(\Upsilon \cdot d_a)$ time, since there are $O(\Upsilon)$ labels deactivated in Line 13, and each deactivation is followed by $O(d_a)$ neighbor searches. Subsequently, it performs $SPREAD_2$ in $O(\Upsilon \cdot \kappa \cdot (d_a + \delta))$ time, since for each of $O(\Upsilon)$ tuples in AL_2 in Line 17, it checks $O(\kappa)$ PPR elements in Line 18, while checking each PPR element takes $O(d_a + \delta)$ time, *e.g.,* it compares $O(d_a)$ values in Line 20 and performs the distance query in $O(\delta)$ time in Line 21 [5]. After that, it performs $SPREAD_3$ in $O(\Upsilon \cdot (\log \Upsilon + d_a \cdot \delta))$ time, just like $DIFFUSE$ in FastDeM.

The time complexity of CLN**:** We eliminate redundant indexes from time to time such that redundant indexes do not become orders of magnitude more than non-redundant ones. Then, we ignore the change of δ in the following analyses. Initially, we explain that a re-generation process, *i.e.,* Algorithm 1, has a time complexity of

$$O\Big(|E| \cdot \delta^2 + |E| \cdot \delta \cdot \log |V|\Big)$$

as follows. For each generated label associated with a vertex $v \in V$, Algorithm 1 inserts $O(deg(v))$ elements into Q. There are $O(|E| \cdot \delta)$ elements in Q in total. For each element in Q, Algorithm 1 takes $O(\log |V|)$ time to pop it out, and also takes $O(\delta)$ time to query a distance. In comparison, CLN has a time complexity of

$$O\Big(|V| \cdot \delta^2 + |E| \cdot \delta \cdot \log |V|\Big).$$

The details are as follows. First, cleaning L in Lines 2-6 takes $O(|V| \cdot \delta^2)$, since there are $O(|V| \cdot \delta)$ labels, and checking whether a label is redundant or not in Line 4 takes $O(\delta)$. Second, generating PPR in Lines 7-16 takes $O(|E| \cdot \delta \cdot \log |V| + |PPR| \cdot \delta)$, since this process is similar to the process of Algorithm 1 with $O(|PPR|)$ distance queries. Given that we generally have $|PPR| \ll |V| \cdot \delta$ in practice, *i.e.,* the number of PPR elements is generally much smaller than the number of 2-hop labels, $O(|PPR| \cdot \delta)$ is covered by $O(|V| \cdot \delta^2)$. As we often have $|E| \gg |V|$ and $\delta \gg \log |V|$, it can be considered that CLN has a smaller time complexity than Algorithm 1.

S3. THE EFFECTIVENESS OF CLN

We show that CLN can eliminate all redundant indexes as follows. First, we present the canonical constraint [1–3] below. For a given rank of vertices, there is only one set of 2-hop labels that satisfies the canonical constraint, *e.g., L* in Figure 1 in the main contents. We refer to a set of 2-hop labels that satisfies the canonical constraint as a canonical set of 2-hop labels, which is minimal in that deleting any label from this set induces that it does not satisfy the 2-hop cover constraint. PLL is a widely-used algorithm for generating a canonical set of 2-hop labels [4, 5].

Definition 1 (Canonical Constraint)**.** *Given a rank of vertices, a set L of 2-hop labels satisfies the canonical constraint if, a vertex v is a hub of $u \in V$, i.e., $v \in C(u)$, if and only if the rank of v is the highest among all vertices in all shortest paths between u and v.*

Suppose that the initial indexes are generated by Algorithm 1 in the main contents. To maintain 2-hop labels after edge weight changes, both the existing solution of IncAsyn + RepairedDecAsyn and the proposed solution of FastIncM + FastDecM generate a new label $L(u)[v]$ only when $r(u) < r(v)$. Let L_m be the maintained set of 2-hop labels by IncAsyn + RepairedDecAsyn or FastIncM + FastDecM after a number of edge weight changes. Further let L_r and PPR_r be the re-generated indexes by Algorithm 1 after these edge weight changes. L_r is a canonical set of 2-hop labels for the updated graph, and PPR_r is the record of the pruning information for generating L_r by PLL. We have the following lemma, which shows that L_m is a super set of the canonical set of 2-hop labels.

Lemma 1. $L_r \subseteq L_m$.

Proof. Consider an arbitrary label $(u', d'_{u's}) \in L_r(s)$. Since L_r is a canonical set of 2-hop labels for the updated graph, u' is the vertex with the highest rank in all shortest paths between s and u' in the updated graph, and $d'_{u's}$ is the distance between s and u' in the updated graph. The proofs of Theorems 1 and 2 show that the vertex with the highest rank in all shortest paths between s and another vertex in the updated graph is a hub of s after each edge weight decrease or increase maintenance. Thus, $(u', d'_{u's}) \in L_m(s)$, and this lemma holds. □

With the input of L_m, let L_c and PPR_c be the cleaned set of 2-hop labels and the cleaned PPR, respectively, by CLN. We further have the following theorem.

Theorem 3. $L_c = L_r$, $PPR_c = PPR_r$.

Proof. Consider an arbitrary label $(v, d_{uv}) \in L_r(u) \subseteq L_m(u)$. When CLN computes d'_{uv} in Line 4, if $d'_{uv} \leq d_{uv}$, 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, since L_c is canonical, 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, $d'_{uv} > d_{uv}$, and CLN inserts (v, d_{uv}) into $L_c(u)$ in Line 6, *i.e.,* $(v, d_{uv}) \in L_c(u)$. On the other hand, consider an arbitrary label $(x, d_{ux}) \in L_m(u) \setminus L_r(u)$. 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, CLN computes $d'_{ux} = d_{ux} = d(u, z) + d(z, x)$, and does not insert (x, d_{ux}) into $L_c(u)$, *i.e.,* $(x, d_{ux}) \notin L_c(u)$. Thus, $L_c = L_r$. Moreover, since PPR_r is the record of the pruning information for generating L_r by PLL; and PPR_c is the record of the pruning information for generating L_c by PLL (notably, the process of generating PPR_c in CLN is essentially the same with the process of generating PPR in Algorithm 1), $PPR_c = PPR_r$. Hence, this theorem holds. □

Therefore, CLN is as effective as a re-generation process for eliminating all redundant indexes.

S4. THE EFFICIENCY OF CLN

The above section proves that CLN is as effective as a re-generation process for eliminating all redundant indexes. We use both CLN and Algorithm 1, *i.e.,* a re-generation process, to eliminate all redundant indexes in the main experiments, first after 1000 edge weight changes, and then after 4000 edge weight changes. It is too slow to apply PLL to large weighted graphs in a single-thread environment. Thus, we use a state-of-the-art parallel strategy of PLL in [6] to implement Algorithm 1, given that the other parallel strategies [7, 8] do not suit weighted cases. To be fair, we also implement CLN parallelly using a thread pool of the same size. We show the running times of CLN and Algorithm 1 in the above experiments in Table S1. It can be seen that CLN is generally around twice faster than Algorithm 1 for eliminating all redundant indexes, given that CLN has a smaller time complexity than Algorithm 1. This shows the usefulness of CLN in dynamic cases.

Table S1. The running times of CLN and Algorithm 1 (unit: seconds).

Name	Jaccard weight type				Random weight type			
	After 1000 changes		After 4000 changes		After 1000 changes		After 4000 changes	
	CLN	Algo. 1	CLN	Algo. 1	CLN	Algo. 1	CLN	Algo. 1
CondMat	1.78	4.35	1.98	4.00	0.71	1.59	0.82	1.62
Gnutella	95.76	180.11	112.21	199.38	32.36	63.17	35.65	67.57
Amazon	2605.54	5101.40	2997.17	5127.14	610.19	1232.76	647.66	1159.18
Book	1332.59	2711.86	1666.87	2948.83	388.84	821.49	425.93	845.68
Hyves	20783.75	27661.649	17121.42	27308.46	1558.92	3006.04	1601.03	2898.71
Skitter	30921.70	56861.41	25585.62	47779.51	2688.64	4713.73	2428.54	4649.20

REFERENCES FOR THE SUPPLEMENT

1. I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck, "Hierarchical hub labelings for shortest paths," in *European Symposium on Algorithms,* (Springer, 2012), pp. 24–35.
2. A. V. Goldberg, I. Razenshteyn, and R. Savchenko, "Separating hierarchical and general hub labelings," in *International Symposium on Mathematical Foundations of Computer Science,* (Springer, 2013), pp. 469–479.
3. M. Jiang, A. W.-C. Fu, R. C.-W. Wong, and Y. Xu, "Hop doubling label indexing for point-to-point distance querying on scale-free networks," Proc. VLDB Endow. **7** (2014).
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.
5. Y. Li, L. H. U, M. L. Yiu, and N. M. Kou, "An experimental study on hub labeling based shortest path algorithms," Proc. VLDB Endow. **11**, 445–457 (2017).
6. K. Lakhotia, R. Kannan, Q. Dong, and V. Prasanna, "Planting trees for scalable and efficient canonical hub labeling," Proc. VLDB Endow. **13** (2019).
7. 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.
8. R. Jin, Z. Peng, W. Wu, F. Dragan, G. Agrawal, and B. Ren, "Parallelizing pruned landmark labeling: dealing with dependencies in graph algorithms," in *Proceedings of the 34th ACM International Conference on Supercomputing,* (2020), pp. 1–13.