World Wide Web (2023) 26:2427–2452
https://doi.org/10.1007/s11280-023-01146-2

Efficient maintenance of highway cover labelling for distance queries on large dynamic graphs

Muhammad Farhan[1] · **Qing Wang**[1]

Received: 26 August 2022 / Revised: 14 December 2022 / Accepted: 29 January 2023 /
Published online: 22 March 2023

Abstract
Graphs in real-world applications are typically dynamic which undergo rapid changes in their topological structure over time by either adding or deleting edges or vertices. However, it is challenging to design algorithms capable of supporting updates efficiently on dynamic graphs. In this article, we devise a parallel fully dynamic labelling method to reflect rapid changes on graphs when answering shortest-path distance queries, a fundamental problem in graph theory. At its core, our solution accelerates query processing through a fully dynamic distance labelling of a limited size, which provides a good approximation to bound online searches on dynamic graphs. Our parallel fully dynamic labelling method leverages two sources of efficiency gains: landmark parallelism and anchor parallelism. Furthermore, it can handle both incremental and decremental updates efficiently using a unified search approach and a bounded repairing inference mechanism. We theoretically analyze the correctness, labelling minimality, and time complexity of our method, and also conduct extensive experiments to empirically verify its efficiency and scalability on 10 real-world large networks.

Keywords Graph algorithms · Highway cover · Shortest-path distance queries · Distance labelling · Dynamic graphs

1 Introduction

Given a graph G, a *distance query* on G is to answer the distance between any two vertices in the graph G. As a fundamental primitive, distance queries are widely applied in modern network-oriented systems, such as communication networks, context-aware search

This article belongs to the Topical Collection: *Special Issue on Knowledge-Graph-Enabled Methods and Applications for the Future Web*
Guest Editors: Xin Wang, Jeff Pan, Qingpeng Zhang, Yuan-Fang Li

✉ Muhammad Farhan
muhammad.farhan@anu.edu.au

Qing Wang
qing.wang@anu.edu.au

[1] School of Computing, Australian National University, Canberra, Australia

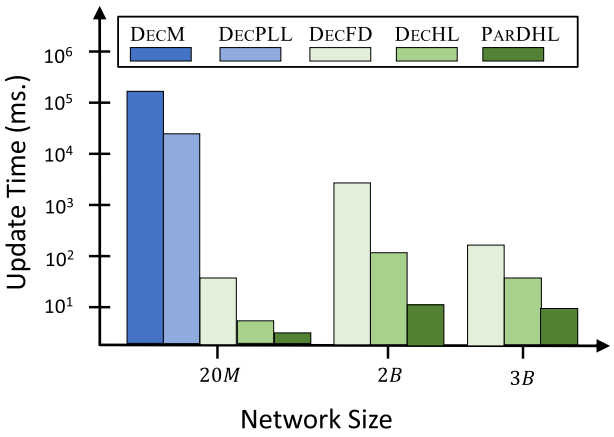

Figure 1 Performance overview of our proposed method ParDHL and the state-of-the-art methods DecM [8], DecPLL [9], DecFD [10] and DecHL [11], where the update time is calculated by processing 1,000 edge deletions over complex networks with sizes varying from 20 millions of edges to 3 billions of edges

in web graphs [1, 2], social network analysis [3, 4], route-planning in road networks [5, 6], management of resources in computer networks [7], and so on.

Traditionally, a distance query can be answered using Dijkstra's algorithm [12] on non-negative weighted graphs or breadth-first search (BFS) algorithm on unweighted graphs. However, these algorithms may end up traversing the entire network when two vertices are far apart from each other, thus becoming too slow for applications that require low latency. To speed up query response time, a plethora of methods have been proposed in the past years [5, 10, 13–21]. Among these methods, precomputing a *distance labelling* is typically considered as a promising solution. However, most of existing distance labelling methods were designed for static networks.

Networks in the real-world are typically dynamic which undergo rapid changes, i.e. edge additions/deletions in their topological structure over time. For example, people become friend/unfriend or follow/unfollow others in social networks, web links become valid/invalid in web graphs, and communication networks may have faults being detected and recovered [7, 22, 23]. It is imperative to design dynamic algorithms that can efficiently update distance labelling to reflect graph changes for fast and accurate responses to distance queries. So far, only limited attempts have been made on maintaining a distance labelling for dynamic graphs [8, 10, 24–26]. Among them, the methods considering incremental updates (i.e. edge additions) [10, 24, 25] are relatively efficient, e.g., an incremental update can be processed on graphs with billions of vertices in less than one second [25]. Unfortunately, the methods considering decremental updates still suffer from long update time of a distance labelling [8–10]. As shown in Figure 1, the average update time of one edge deletion on graphs of size around 20 million edges is 135 seconds for DecM [8] and 19 seconds for DecPLL [9], which are very inefficient. Moreover, these methods all consider the single-update setting, i.e., performing one single edge insertion or edge deletion at a time. Unlike existing works, in this article, we aim to explore the following research questions:

> Q1: *Is it possible to design a dynamic labelling algorithm which can efficiently reflect both incremental and decremental updates on graphs for fast and accurate distance computation?*

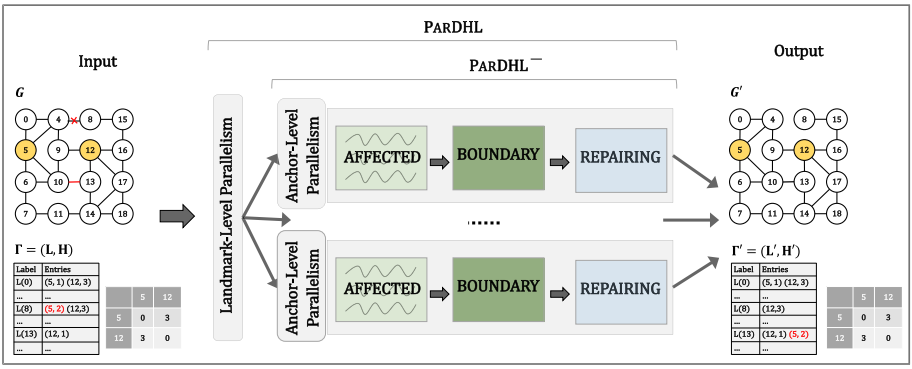

Figure 2 An illustration of our parallel framework, which dynamically maintains highway cover labelling for graphs undergoing rapid updates

Q2: *Can such a dynamic labelling algorithm handle multiple updates in parallel in order to offer performance gains over existing dynamic labelling algorithms in the single-update setting?*

To answer these research questions, we propose a parallel solution for answering distance queries on dynamic graphs undergoing rapid changes in their topological structure. Our method is efficient both in time and space, and can scale to large graphs with billions of edges. There are several design considerations. First and foremost, we combine offline labelling and online searching in our proposed solution so as to leverage the advantages from both sides - accelerating query processing through a distance labelling that has a limited size but provides a good approximation to bound online searches. Then, we proceed to design a fully dynamic distance labelling algorithm, which dynamizes a distance labelling to efficiently reflect updates on the underlying graph. This algorithm consists of three stages: (1) *Finding affected vertices* - to precisely identify vertices that are affected by updates; (2) *Finding boundary vertices* - to bound the traversal space that is needed for repairing; (3) *Repairing affected vertices* - to change the labels of affected vertices via an inference process based on their new distances. Figure 2 illustrates the high-level overview of our solution. At its core, we abide by the following principles:

Parallel searches: We exploit interactions between updates and design a novel parallel approach to find affected vertices, which involves both landmark parallelism and anchor parallelism.
Bounded space: We bound search spaces for updates to only small portions of graphs that are affected, which are achieved by identifying boundary vertices with respect to updates.
Repair inference: We develop a repairing approach that can efficiently infer the new distances of affected vertices to repair their labels through a level-by-level computation from boundary vertices.

1.1 Contributions

To summarise, the main contributions of this work are as follows:

(a) We study the problem of answering exact distance queries on dynamic graphs by proposing an efficient solution comprising of a fully dynamic distance labelling and online sparsified searches.
(b) Our fully dynamic labelling algorithm can uniformly handle both incremental and decremental updates efficiently using a novel parallel search strategy and a bounded repairing inference mechanism.
(c) We prove the correctness of our method and show that it can preserve the minimality of distance labelling, as well as provide the complexity analysis.
(d) We have evaluated our method on 10 real-world large networks to verify their efficiency, scalability and robustness. The results show that our algorithms significantly outperform the state-of-the-art methods. It can answer queries in the order of milliseconds and maintain very small labelling sizes, even for large graphs with billions of edges. To the best of our knowledge, this is the first study to develop a parallel fully dynamic labelling method for distance queries.

1.2 Outline

The rest of the article is organized as follows. In Section 2, we review existing works that are related to our work. In Section 3, we present preliminary notations and definitions used in this article along with problem formulation. In Section 4, we discuss our parallel fully dynamic framework. In Section 5, we present our proposed method that can maintain distance labelling in parallel to efficiently reflect graph changes for distance querying. A formal proof for showing that our method is correct and preserve the property of minimality of highway cover labelling, along with complexity analysis is detailed in Section 6. Then, in Section 7, we discuss our experimental results. Finally, we conclude the article and outline future research directions in Section 9.

2 Related work

We review previous works that have attempted to address the shortest-path distance query answering problem on dynamic graphs.

In [13], Akiba et al. proposed the pruned landmark labelling (PLL) method which precomputes a 2-hop cover distance labelling [27] by performing a pruned breadth-first search (BFS) from every vertex. The idea is to prune vertices whose distance information can be obtained from the partially available 2-hop cover distance labelling constructed during previous BFSs. This work helps to lower construction cost and labelling size. Later on, Li et al. [20] developed a parallel method called parallel shortest distance labelling (PSL) for constructing PLL in parallel in order to increase scalability. These labelling-based methods serve as the state-of-the-art for answering exact distance queries. However, they are designed for static graphs whose topological structure remains unchanged over time.

Some early works on extending 2-hop cover distance labelling from static graphs to dynamic graphs have considered either incremental updates (i.e., edge insertions) or decremental updates (i.e., edge deletions). In [24], Akiba et al. studied the problem of maintaining a 2-hop cover distance labelling on graphs undergoing incremental updates. This work claims fast update time at the cost of increased labelling size as they do not remove outdated and redundant distance entries from the labels of affected vertices which may significantly affect query performance over time. They consider that removing outdated and redundant distance entries would be costly and may make the update operation slower. Another

recently proposed method [25] studied incremental updates on graphs which solves the problem of eliminating outdated and redundant entries, thereby guaranteeing minimality of labelling. Note that, in the case of decremental updates, outdated and redundant distance entries must be removed; otherwise distance labelling cannot be correctly updated. To tackle the problem of maintaining 2-hop cover distance labelling for decremental updates, Qin et al. [8] proposed a method using the *well-ordering* property of 2-hop cover distance labelling. However, this method is inefficient to perform updates for even moderately large graphs and can only scale to graphs that have a few millions of vertices and edges. It has been shown in the experiments that the average update time of 1000 deletions on a network of size 19 millions edges is 135 seconds which is too high to be used in real-world scenarios.

Several methods [9–11, 28, 29] have been proposed to consider both incremental and decremental updates on graphs. Hayashi et al. [10] proposed a fully dynamic (FD) method to perform distance queries on dynamic graphs. The key idea is to select a small set of landmarks R and precompute shortest-path trees (SPTs) w.r.t. each $r \in R$. Then, a distance query traverses a sparsified graph under an upper distance bound being computed via the SPTs. To reflect graph changes, they maintain SPTs to ensure correct distances. However, this method cannot perform decremental updates efficiently, e.g., as shown in [10], it takes about 6 seconds on average to reflect one graph change (i.e., an edge deletion) on Twitter network. D'angelo et al. [9] maintains a 2-hop cover distance labelling for dynamic graphs. They first developed a method for reflecting decremental updates on a graph, which works in three phases, 1) identify the affected vertices, 2) remove the outdated entries, and 3) restore the 2-hop cover property. Then, they combined the work proposed in [24] for incremental updates with their method for decremental updates to form a fully dynamic algorithm. However, this fully dynamic algorithm has a very high complexity of performing decremental updates, e.g., on a network with 16 millions of edges, it takes 19 seconds, which would be impractical for many real-world applications. Another method by D'Emidio et al. [29] claims an improvement over the method proposed in [9] for decremental updates. However, this method is limited to graphs with few millions of nodes when updating labels is required to be in the order of seconds. A recent method [28] has also attempted to apply a parallel method (PSL) for constructing PLL [20] on dynamic graphs for fast distance computation which unfortunately can only accommodate million-scale graphs. Very recently, Farhan et al. [11] proposed a fully dynamic method that leverages the advantages of highway cover distance labelling proposed in [18] for fast processing of dynamic changes on graphs. Nonetheless, their method, similar to previous methods, processes graph updates in the single-update setting which may cause repeated computation w.r.t. multiple updates and thus is slow as shown in our experiments. In this article, we have adopted batch-update setting to reflect graph changes much more efficiently by exploiting different interaction between updates in a batch.

3 Problem formulation

Let $G = (V, E)$ be a graph where V is a set of vertices and E is a set of edges. The *distance* between two vertices s and t in G, denoted as $d_G(s, t)$, is the length of a shortest-path between s and t. If there does not exist any path between two vertices s and t, then we consider $d_G(s, t) = \infty$. We use $P_G(s, t)$ to denote the set of all shortest-paths between s and t in G, and $N_G(v)$ the set of neighbors of a vertex $v \in V$, i.e. $N_G(v) = \{w \in V \mid (v, w) \in E\}$. Without loss of generality, we assume that G is undirected and unweighted

and explain how this work can be extended to directed graphs and non-negative weighted graphs in Section 8.

In this work, we consider two fundamental types of updates on graphs, *edge insertion* and *edge deletion*. Given a graph $G = (V, E)$, an edge insertion is to add an edge (a, b) into G where $\{a, b\} \subseteq V$ and $(a, b) \notin E$. Conversely, an edge deletion is to delete an edge (a, b) from G where $(a, b) \in E$. We consider to perform multiple updates aggregated as a batch in parallel. Without loss of generality, we assume that there is no insertion and deletion on the same edge in a batch since these two operations cancel each other out. It is worth noting that node insertion or node deletion can be treated as a set of updates which contains only edge insertions or only edge deletions, respectively.

Let $R \subseteq V$ be a small set of special vertices in a graph G, called *landmarks*. A *label* $L(v)$ for each vertex $v \in V$ is a set of *distance entries* $\{(r_i, \delta_L(r_i, v))\}_{i=1}^{n}$ where $r_i \in R$, $\delta_L(r_i, v) = d_G(r_i, v)$ and $n \leq |R|$. The set of labels for all vertices in V, i.e., $L = \{L(v)\}_{v \in V}$, is a *distance labelling* over G. The *size* of a distance labelling L is defined as $size(L) = \sum_{v \in V} |L(v)|$. In the literature, a distance labelling is often constructed by following the 2-hop cover property [27]. For any two vertices (u, v) in a graph, the 2-hop cover property requires at least one common vertex w existing in both $L(u)$ and $L(v)$ and w must also be on one shortest-path between u and v.

Definition 1 (2-hop cover distance labelling) A distance labelling L over a graph $G = (V, E)$ is a *2-hop cover labeling* if the following holds for any pair of vertices $u, v \in V$:

$$d_G(u, v) = \min\{\delta_L(w, u) + \delta_L(w, v) \mid (w, \delta_L(w, u)) \in L(u), (w, \delta_L(w, v)) \in L(v)\}. \quad (1)$$

In our work, we consider a distance labelling property based on the notion of highway, i.e., highway cover labelling [18], which has recently been shown to outperform the state-of-the-art methods in the single-update setting [11]. A *highway* $H = (R, \delta_H)$ consists of a set R of landmarks and a distance decoding function $\delta_H : R \times R \to \mathbb{N}^+$ s.t. $\delta_H(r_1, r_2) = d_G(r_1, r_2)$ for any two landmarks $r_1, r_2 \in R$.

Definition 2 (Highway cover distance labelling) A *highway cover labelling* $\Gamma = (H, L)$ consists of a highway H and a distance labelling L satisfying that, for any $v \in V \backslash R$ and $r \in R$,

$$d_G(r, v) = \min\{\delta_L(r_i, v) + \delta_H(r, r_i) \mid (r_i, \delta_L(r_i, v)) \in L(v)\} \quad (2)$$

Intuitively, a highway cover labelling requires that, for every vertex $v \in V$, its label $L(v)$ must contain a distance entry to each landmark $r \in R$ unless there is another landmark occurring on a shortest-path between r and v. For a detailed explanation, please refer to [18], which has also provided some illustrative examples for highway and highway cover labelling.

In this work, we study the problem of answering distance queries on dynamic graphs that undergo rapid changes in their topological structures. Since both edge insertion and deletion are considered, we formulate the problem as a fully dynamic distance query answering problem.

Definition 3 (Fully dynamic distance query) Given a dynamic graph G' that undergoes edge insertions and edge deletions, a *fully dynamic distance query* for any two vertices s and t in G' is to compute their distance on G'.

4 Parallel fully dynamic framework

In this section, we explore an efficient and scalable parallel fully dynamic framework that combines offline distance labelling and online searching to answer exact distance queries. Our proposed framework has two main components: (i) *fully dynamic distance labelling*, and (ii) *sparsified searching*. The key idea is to dynamically maintain a distance labelling for a graph G that undergoes updates, and then use such a fully dynamic distance labelling to guide online searches on a sparsified subgraph of G in order to answer fully dynamic distance queries efficiently.

4.1 Fully dynamic distance labelling

Let G be a graph that undergoes updates (edge insertions and deletions) and B be a sequence of updates occurring on G. We use $G \circ B$ to indicate the graph that corresponds to applying updates B on G. A *fully dynamic distance labelling* on $G \circ B$ is a highway cover distance labelling that is dynamically maintained to reflect updates B on G. That is, a fully dynamic distance labelling $\Gamma = (H, L)$ on G can be dynamically maintained to a fully dynamic distance labelling $\Gamma' = (H', L')$ on $G \circ B$ for any sequence B of updates containing edge insertions and deletions. Note that the set of landmarks on G and $G \circ B$ remains the same during the maintenance.

As discussed in [9], minimality is a highly desirable property to consider when designing a distance labelling over dynamic graphs. Otherwise, a distance labelling may have increasingly unnecessary entries in its labels and query performance deteriorates over time.

Definition 4 (Minimality) A fully dynamic distance labelling $\Gamma = (H, L)$ on $G \circ B$ is *minimal* if $size(L') \geq size(L)$ always holds for any fully dynamic distance labelling $\Gamma' = (H', L')$ on $G \circ B$.

Since the set of landmarks is unchanged, the highways H and H' in the above definition always have the same size, i.e., only differing in distance values between landmarks. It has been shown in [18] that, given a graph and a set of landmarks on the graph, there exists a unique minimal highway cover labelling on the graph.

4.2 Fully dynamic distance querying

Given a fully dynamic distance labelling $\Gamma = (H, L)$ on a graph G, an upper bound on the distance between any pair of vertices $s, t \in V \backslash R$ in G is computed as follows:

$$d_{st}^{\top} = \min\{\delta_L(r_i, s) + \delta_H(r_i, r_j) + \delta_L(r_j, t) \mid (r_i, \delta_L(r_i, s)) \in L(s), \\ (r_j, \delta_L(r_j, t)) \in L(t)\} \tag{3}$$

A fully dynamic distance query $Q(s, t, \Gamma)$ on G using a fully dynamic distance labelling Γ can be answered by conducting a bi-directional BFS search over a sparsified graph $G[V \backslash R]$ (i.e., removing all landmarks in R from G) under the upper bound $d_{st}^{\top}$ such that:

$$Q(s, t) = \begin{cases} d_{G[V \backslash R]}(s, t) & \text{if } d_{G[V \backslash R]}(s, t) \leq d_{st}^{\top}, \\ d_{st}^{\top} & \text{otherwise.} \end{cases} \tag{4}$$

One major challenge of this framework is "*how to design an algorithm that can efficiently compute a fully dynamic distance labelling on $G \circ B$ for any sequence B of updates on a*

Input: $G, \Gamma = (H, L)$ over G, B, $G' = G \circ B$
Output: $\Gamma' = (H', L')$ over G'
1 **foreach** $r \in R$ *in parallel* **do**
2 $\quad \mathcal{A}(r, B) \leftarrow$ FINDAFFECTED(G', B, r, Γ);
3 $\quad \mathcal{B}(r, B) \leftarrow$ FINDBOUNDARY$(G', \mathcal{A}(r, B))$;
4 $\quad \Gamma' =$ REPAIRAFFECTED$(G', \mathcal{A}(r, B), \mathcal{B}(r, B), r, \Gamma)$;
5 **end**

Algorithm 1 PARDHL.

graph G in order to perform fully dynamic distance queries on graphs undergoing updates, particularly when such graphs are very large?"

5 Parallel maintenance of distance labelling

Below, we introduce a parallel fully dynamic method that can handle all updates including both edge insertions and edge deletions in parallel and reflect the effects of these updates into a distance labeling efficiently. Our proposed method, denoted as PARDHL (an abbreviation for *Parallel Dynamic Highway Labelling*), involves three main steps: *finding affected vertices*, *finding boundary vertices*, and *repairing affected vertices*. Algorithm 1 describes a high-level view of PARDHL. We discuss these three steps in detail.

5.1 Finding affected vertices

We start with defining "*affected vertices*" whose labels may need to be updated as a consequence of edge insertions and edge deletions. Let $G = (V, E)$ be changed to $G' = (V', E')$ by a sequence of updates B, i.e., $G' = G \circ B$, R be a set of landmarks, and Γ be a fully dynamic distance labelling on G.

Definition 5 (Affected vertex) A vertex $v \in V$ is *affected* w.r.t. a landmark $r \in R$ by a sequence of updates B iff $P_G(v, r) \neq P_{G'}(v, r)$, and *unaffected* otherwise.

We use $\mathcal{A}(r, B) = \{v \in V \mid P_G(v, r) \neq P_{G'}(v, r)\}$ to denote the set of all affected vertices by B w.r.t. a landmark r and $\mathcal{A} = \bigcup_{r \in R} \mathcal{A}(r, B)$ refers to the set of all affected vertices.

The following lemma states how affected vertices relate to a single update (edge insertion or edge deletion).

Lemma 1 *A vertex v is affected w.r.t. a landmark r iff there exists a shortest-path between v and r that passes through an inserted edge (a, b) in G' or a deleted edge (a, b) in G.*

By this lemma, we know that any update on an edge (a, b) satisfying $d_G(r, a) = d_G(r, b)$ is *trivial* w.r.t. a landmark r, since such an update does not affect any vertices w.r.t. the landmark r. Without loss of generality, we assume that $d_G(r, b) > d_G(r, a)$ for updates on an edge (a, b) in the rest of this article.

Since a vertex v is affected w.r.t a landmark r iff $P_G(v, r) \neq P_{G'}(v, r)$ (cf. Definition 5), a "naive" way of finding all affected vertices is to conduct a BFS from each landmark $r \in R$ on both graphs G and G', respectively, and then compare whether $P_G(v, r) \neq P_{G'}(v, r)$

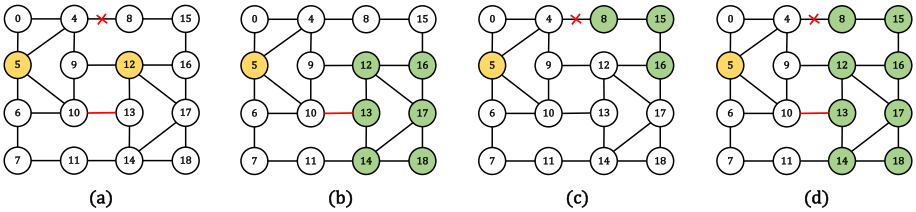

Fig. 3 (a) A graph with two landmarks 5 and 12 and a sequence of updates $B = \langle(4, 8), (10, 13)\rangle$ containing a deleted edge (4, 8) and an inserted edge (10, 13), and (d) affected vertices by $B = \langle(4, 8), (10, 13)\rangle$ w.r.t. landmark 5 are highlighted in green color

holds for each vertex v. However, this has the time complexity $O((|V| + |E|)|R|)$ which is prohibitively high for large graphs. In the following, we propose a parallel algorithm to identify affected vertices. The key ideas are: (1) to search only on affected vertices on the changed graph G' by leveraging the distance labelling on the original graph G and an observation on how affected vertices relating to anchor vertices; (2) to parallelise searches for multiple updates based on their anchor vertices, regardless whether they are edge insertions or deletions. Below, we first define the notions of *anchor vertex* and *pre-anchor vertex* in terms of an update (a, b), which can be an edge insertion or an edge deletion.

Definition 6 (Anchor vertex and pre-anchor vertex) The *anchor vertex* of an update (a, b) is either a or b, whichever is further away from r, and the *pre-anchor vertex* of (a, b) is a vertex in $\{a, b\}$ that is not the anchor.

Note that when $d_G(r, a) = d_G(r, b)$ there is no anchor vertex nor pre-anchor vertex corresponding to the update (a, b). It can be easily proven that, for any update (a, b), if it is not trivial w.r.t. a landmark r, i.e. $d_G(r, a) \neq d_G(r, b)$, its anchor vertex must be affected by (a, b) whereas its pre-anchor vertex is unaffected by (a, b). Based on this observation, we can use the pre-anchor vertex of an update to compute the *anchor distance* for such an update.

Definition 7 (Anchor distance) The *anchor distance* of an update (a, b) w.r.t. a landmark r is $d_G(r, u') + 1$ where u' is its pre-anchor vertex.

For an edge insertion, this anchor distance indicates the new distance of the anchor vertex to the landmark r on the changed graph $G \circ (a, b)$; for an edge deletion, this anchor distance indicates the old distance of the anchor vertex to the landmark r on the original graph G. In our work, regardless of edge insertions or edge deletions, we use anchor distances to devise a unified algorithm that can find vertices affected by updates in parallel. Concretely, given a sequence of updates B, there exists a set of anchor vertices corresponding to the updates in B. We can then perform parallel searches from multiple or all anchor vertices in this set to find affected vertices efficiently, for which we call *anchor parallelism*.

Example 1 Consider the example graph in Figure 3(a) which has two landmarks 5 and 12. After applying a sequence of updates $B = \langle(4, 8), (10, 13)\rangle$ containing a deleted edge (4, 8) and an inserted edge (10, 13), the set of all vertices affected by B w.r.t. the landmark 5 is {8, 12, 13, 14, 15, 16, 17, 18} because their sets of shortest-path(s) to landmark 5 have

```
1  Function FindAffected(G', B, r, Γ)
2      A(r, B) ← ∅;
3      Q ← ∅;
4      for (a, b) ∈ B in parallel do
5          π ← Q(r, a, Γ) + 1;
6          Enqueue (b, π) to Q;
7          while Q is not empty do
8              Dequeue (v, π) from Q;
9              foreach w ∈ N_G'(v) s.t. Q(r, w, Γ) ≥ π + 1 do
10                 Enqueue (w, π + 1) to Q;
11             end
12             A(r, B) = A(r, B) ∪ {v};
13         end
14     end
15     return A(r, B);
16 end
```

Algorithm 2 Finding affected vertices.

changed that is also highlighted in Figure 3(d). We have two anchor vertices 8 and 13 for B w.r.t. 5. The simultaneous searches from these anchors are shown in Figure 3(b)-(c) which results in identifying all affected vertices.

The following lemma states that we can start with an anchor vertex to traverse its local neighbourhood and then identify affected vertices based on their distances to the anchor vertex on G', the anchor distance on G and the distance labelling information on G. We can also easily see that this lemma allows us to parallelise searches from anchor vertices.

Lemma 2 *Let $G' = G \circ B$. A vertex v in G is affected by B w.r.t. a landmark r if the following condition holds for at least one update $(u', u) \in B$:*

$$d_G(r, v) \geq (d_G(r, u') + 1) + d_{G'}(u, v). \tag{5}$$

Proof Given a vertex v in G which satisfies the condition in (5), we know that there must exist at least one new shortest path between r and v that goes through an inserted edge (u', u) in G', or at least one old shortest path between r and u that goes through a deleted edge (u, u'). By Lemma 1, we thus know that v is affected. □

Algorithm 2 shows the pseudo-code of our algorithm that finds affected vertices. Given a sequence of updates B on G and a highway cover distance labelling Γ on G, we conduct a partial BFS search for each update $(a, b) \in B$ w.r.t. a landmark r in parallel. Specifically, for each update $(a, b) \in B$, we start from the anchor vertex b with its new depth $\pi = Q(r, a, \Gamma) + 1$ (Lines 5-6). Then, for every $(v, \pi) \in \mathcal{Q}$, we examine the neighbors of v and enqueue the ones into $\mathcal{Q}$ that are affected based on Lemma 1 with their new depths $\pi + 1$ (Lines 9-10) and add v to $\mathcal{A}(r, B)$ as an affected vertex (Line 9). This process continues until $\mathcal{Q}$ is empty.

Example 2 Consider the graph in Figure 3(b)-(c), the jumped BFS for the deleted edge $(4, 8)$ iteratively finds 3 affected vertices $\{8, 15, 16\}$ starting from the anchor vertex 8, and the jumped BFS for the inserted edge $(10, 13)$ finds 6 affected vertices $\{12, 13, 14, 16, 17, 18\}$ w.r.t. the landmark 5. After all parallel jumped BFSs, Algorithm 2 will return the set $\mathcal{A}(r, B) = \{8, 12, 13, 14, 15, 16, 17, 18\}$.

5.2 Finding boundary vertices

Now we define a special kind of affected vertices, called *boundary vertices*, which allows us to bound the update space so as to efficiently repair the labels of affected vertices. These boundary vertices lie at the boundary of affected and unaffected vertices.

Definition 8 (Boundary vertex) A vertex v is a *boundary vertex* w.r.t. a landmark $r \in R$ and a sequence of updates B if v is an affected vertex and has at least one unaffected neighbor w, i.e. $w \notin \mathcal{A}(r, B) \wedge w \in N_{G'}(v)$.

We use $\mathcal{B}(r, B)$ to refer to the set of boundary vertices w.r.t. a landmark r and a sequence of updates B. Since a boundary vertex v has at least one unaffected neighbor w, $d_{G'}(r, v)$ must be upper-bounded by $d_G(r, w) + 1$. It is worth noting that, if v is disconnected from the landmark r after applying the updates B, then the upper bound of v is undefined. In Algorithm 3, we treat $d_{G'}(r, v) = \infty$ (Line 2) in this case. Then, we define the *distance bound* of v, denoted as $d^*(r, v)$, by its unaffected neighbors such that:

$$d^*(r, v) = min\{d_G(r, w) + 1 \mid w \notin \mathcal{A}(r, B) \wedge w \in N_{G'}(v)\}. \tag{6}$$

The following lemma states that the smallest distance bound of a boundary vertex v equals to the new distance from v to the landmark r on the changed graph G'.

Lemma 3 *Let $G' = G \circ B$. If a vertex $v \in \mathcal{A}(r, B)$ has the smallest distance bound $d^*(r, v)$, then $d_{G'}(r, v) = d^*(r, v)$ holds.*

Proof We prove this by contradiction. Assume that $d_{G'}(r, v) \neq d^*(r, v)$. Since $d^*(r, v)$ is the minimum length of all paths between v and r that go through only unaffected vertices, it means that $d_{G'}(r, v) < d^*(r, v)$ and one shortest-path between v and r must go through at least one affected vertex $v' \in \mathcal{A}(r, B)$. Then $d_{G'}(r, v) > d_{G'}(r, v')$ must hold. This contradicts with the assumption that v has the smallest distance bound and thus $d_{G'}(r, v) = d^*(r, v)$. □

Note that, $d_{G'}(r, v) = d^*(r, v)$ does not generally hold for every boundary vertex v. If the distance bound of a boundary vertex v is not the smallest in comparison with the distance bounds of other boundary vertices in $\mathcal{B}(r, B)$, the distance $d_{G'}(r, v)$ can be computed from its affected neighbors rather than its unaffected neighbors. Nonetheless, in such cases, the affected neighbors of v needs to find their new distances to the landmark r on the changed graph G' before computing $d_{G'}(r, v)$.

Example 3 Consider the graph in Figure 4(a), three boundary vertices with their distance bounds w.r.t. the landmark 5 are highlighted in dark green. Although the distance bound of vertex 14 is 4 through its unaffected neighbor 11, the distance $d_{G'}(5, 14) = 3$ can be obtained through its affected neighbor 13 as shown in Figure 4(b).

```
1  Function FindBoundary(G′, A(r, B))
2      π ← ∞;
3      forall the v ∈ A(r, B) do
4          foreach w ∈ N_G′(v) s.t. w ∉ A(r, B) do
5              π ← min{π, Q(r, w, Γ)+1};
6          end
7          if π is not ∞ then
8              B(r, B) = B(r, B) ∪ {(π, v)};
9          end
10         Remove entry for r from L(v) (if exists);
11     end
12     return B(r, B);
13 end
```

Algorithm 3 Finding boundary vertices.

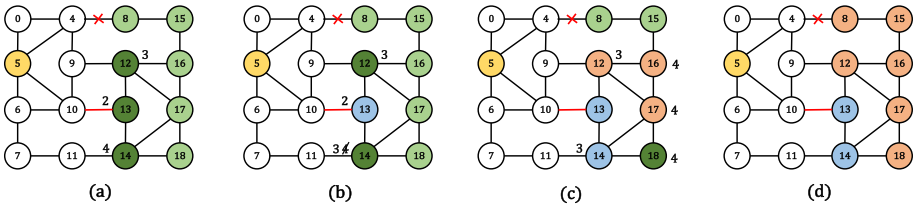

Fig. 4 An illustration of our bounded repair algorithm w.r.t. the landmark 5. Affected vertices are highlighted in green color, boundary vertices are highlighted in dark green color, blue color denotes vertices that are being repaired, and red color denotes vertices that are being pruned. Their distance bounds are also provided next to these vertices

Algorithm 3 describes our algorithm for finding boundary vertices. Given a changed graph G' by a sequence of updates B and a set of affected vertices $\mathcal{A}(r, B)$, for each vertex $v \in \mathcal{A}(r, B)$, we compute the distance bound of v using the distance information of its unaffected neighbors (Lines 4-7). Additionally, to improve efficiency, we remove outdated distance information from the labels of all affected vertices because they will be repaired based on the changed graph G' in the next section (Line 8).

Example 4 Consider the graph in Figure 4(a), we start with eight affected vertices and find only three of them as boundary vertices {12, 13, 14} because of the presence of at least one unaffected vertex in their neighborhood and their bounded distances are being computed using the distance information of their unaffected neighbors.

5.3 Repairing affected vertices

In this section, we propose a repairing strategy to efficiently update the labels of affected vertices in order to reflect graph changes. The key idea is to conduct a BFS only on affected vertices bounded by the "*boundary vertices*".

Concretely, based on Lemma 3, we can conduct a BFS from boundary vertices with the smallest distance bound to infer the distances of their affected neighbors and repair their labels. After each iteration, we treat affected vertices with repaired labels as being unaffected and further process boundary vertices that have the smallest distance bound again. This process terminates only when the labels of all affected vertices are repaired.

To further improve efficiency, we develop a landmark pruning strategy based on the following lemma.

Lemma 4 *A vertex $v \in \mathcal{A}(r, B)$ can be pruned from repairing (i.e., do not repair its label) if there exists a landmark $r' \in R \backslash \{r\}$ lying on one shortest-path in $P_{G'}(v, r)$.*

Proof By Definition 2, if there exists such a landmark r', $d_{G'}(r, v)$ can be computed using (3) based on the highway and label of v w.r.t. the landmark r'. Thus, the entry in $L(v)$ w.r.t. the landmark r is not needed. □

From Lemma 4, we notice that, if a vertex $v \in \mathcal{A}(r, B)$ can be pruned, any vertex $v' \in \mathcal{A}(r, B)$ satisfying $d_{G'}(r, v') = d_{G'}(r, v) + d_{G'}(v, v')$ can also be pruned. Putting all together, we incorporate a landmark pruning strategy into our repairing algorithm.

Algorithm 4 describes our algorithm for repairing affected vertices. Given a graph $G' = G \circ B$, a set of affected vertices $\mathcal{A}(r, B)$ and a set of boundary vertices $\mathcal{B}(r, B)$, we first sort vertices in $\mathcal{B}(r, B)$ w.r.t. their boundary distances and set π as the minimum boundary distance (Lines 2-3). We use two queues $\mathcal{Q}_l$ and $\mathcal{Q}_p$ to process vertices to be labeled and pruned, respectively. Then, we conduct a BFS w.r.t. a landmark r starting from vertices in $\mathcal{B}(r, B)$ with the smallest distance bound. We process vertices in $\mathcal{B}(r, B)$ that have distance π and enqueue to $\mathcal{Q}_p$ or $\mathcal{Q}_l$ based on whether they need to be labeled or have been pruned. For each vertex $v \in \mathcal{Q}_l$ at distance π, we examine affected neighbors w of v. If w is pruned, and if w is a landmark, then we repair the highway (Line 12) and add w to $\mathcal{Q}_{pruned}$ because it is pruned (Line 13). Otherwise, we repair the label of w i.e., add an entry for r with the new distance $\pi + 1$ in $L(w)$ and enqueue w to $\mathcal{Q}_l$ (Lines 14-15). After that, we remove w from $\mathcal{A}(r, B)$ because it has been repaired (line 17). Now we process vertices with distance $\pi + 1$ in $\mathcal{B}(r, B)$ and put them to respective queues before processing pruned vertices, because otherwise vertices in $\mathcal{Q}_p$ may prune out some of them that should not have been pruned (Lines 19-20). Next, we process vertices $\mathcal{Q}_p$ and for each $(v, \pi) \in \mathcal{Q}_p$ at depth π, we enqueue affected neighbors w of v to $\mathcal{Q}_p$ and remove them from $\mathcal{A}(r, B)$ (Lines 22-24). We process these two queues, one after the other, until $\mathcal{Q}_l$ is empty.

Example 5 In the graph of Figure 4(a), we have three boundary vertices with their distance bounds. In the graph of Figure 4(b), we start with the boundary vertex 13 which has the smallest distance bound 2 w.r.t. the landmark 5, repair its label and infer new distance bound 3 for its affected neighbors 12 and 14. Then, in the graph of Figure 4(c), we repair the labels of boundary vertices 12 and 14 which have the smallest distance bound 3 and infer new distance bound 4 for their affected neighbors 16, 17 and 18. Note that vertex 12 is pruned because it is a landmark. Next, in the graph of Figure 4(c), we only repair the label of boundary vertex 18 because the other two boundary vertices 16 and 17 are pruned due to the presence of landmark 12 in their shortest-paths to the root landmark 5. Finally, in the graph of Figure 4(d), boundary vertices 8 and 12 are also pruned based on Lemma 4.

```
1  Function RepairAffected(G', A(r, B), B(r, B), r, Γ)
2      Sort vertices in B(r, B) w.r.t. boundary distances;
3      π ← min. boundary distance;
4      Q_l ← ∅, Q_p ← ∅;
5      Q_p ← {v | (v, π') ∈ B(r, B) and π = π' and v is pruned};
6      Q_l ← {v | (v, π') ∈ B(r, B) and π = π' and v is not pruned};
7      while Q_l is not empty do
8          foreach v ∈ Q_l at depth π do
9              foreach w ∈ N_{G'}(v) s.t. w ∈ A(r, B) do
10                 if w is pruned then
11                     if w is a landmark then
12                         δ_{H'}(r, w) ← π + 1;
13                     Q_p ← w;
14                 else
15                     L'(w) ← {(r, π + 1)};
16                     Q_l ← w;
17                 end
18                 Remove w from A(r, B);
19             end
20         end
21         Q_p ← {v | (v, π') ∈ B(r, B) and π + 1 = π' and v is pruned};
22         Q_l ← {v | (v, π') ∈ B(r, B) and π + 1 = π' and v is not pruned};
23         foreach v ∈ Q_p at depth π do
24             foreach w ∈ N_{G'}(v) s.t. w ∈ A(r, B) do
25                 Q_p ← w;
26                 Remove w from A(r, B);
27             end
28         end
29         π ← π + 1;
30     end
31     return Γ' = (L', H');
32 end
```

Algorithm 4 Repairing affected vertices.

Due to the highway cover property of distance labelling, we can also leverage the parallelism at the landmark level, and we call this *landmark parallelism*. As shown in Algorithm 1, all these three steps can be conducted in parallel with respect to each landmark $r \in R$.

6 Theoretical discussion

In this section, we discuss the correctness of our parallel fully dynamic method and show that it can preserve the minimality property of highway cover labelling. We also analyse the time and space complexity of our proposed method.

6.1 Proof of correctness

When a graph G is changed to a graph G' after undergoing a sequence of updates B, our proposed method ParDHL can dynamically maintain a highway cover labelling Γ over G to a highway cover labelling Γ' over G'. Formally, ParDHL is *correct* iff, whenever $Q(u, v, \Gamma) = d_G(u, v)$ holds for any two vertices u and v in G, $Q(u', v', \Gamma') = d_{G'}(u', v')$ also holds for any two vertices u' and v' in G'. We prove the following theorem to show the correctness of ParDHL.

Theorem 1 *Let $G' = G \circ B$ and $\Gamma' =$ ParDHL(G, Γ, B, G'). Then Γ' is a highway cover labelling on G'.*

Proof First, we prove that `FindAffected` finds the set of all affected vertices w.r.t. a landmark r by a sequence of updates B. By Lemma 1, we know that each affected vertex is enqueued into $\mathcal{Q}$ during a BFS search from b (Lines 4 and 7-8 in Algorithm 2). Thus, $\mathcal{A}(r, B)$ contains all affected vertices w.r.t. r and (a, b).

Then, we show that `FindBoundary` finds the set of all boundary vertices w.r.t. a landmark r and updates B. Algorithm 3 finds boundary vertices $\mathcal{B}(r, B)$ from all affected vertices $v \in \mathcal{A}(r, B)$. It adds an affected vertex $v \in \mathcal{A}(r, B)$ into $\mathcal{B}(r, B)$ iff v has at least one unaffected neighbor (Lines 4-7). Algorithm 3 also removes the distance entries of r from all affected vertices (Line 8).

Now, we prove that `RepairAffected` modifies $\Gamma = (H, L)$ to $\Gamma' = (H', L')$ s.t. (1) $(r, d_{G'}(r, v)) \in L'(v)$ for $v \in \mathcal{A}(r, B)$ iff $P_{G'}(r, v)$ does not contain any other landmark in the shortest-path from v to r; (2) $\delta_{H'}(r, r') = d_{G'}(r, r')$ for any $r' \in R \backslash \{r\}$. By Lemma 3, starting from boundary vertices with the smallest distance bound, the distances of affected vertices on G' are iteratively inferred in $\mathcal{A}(r, B)$ and added into their labels via $\mathcal{Q}_l$ if these affected vertices are not prunable (Lines 6, 15-16 and 19). If an affected vertex v is prunable, it is kept in $\mathcal{Q}_p$; if v is also a landmark, $\delta_{H'}(r, v)$ in H' is updated (Lines 5, 10-13, 18). Thus, every vertex v appearing in $\mathcal{Q}_p$ has no distance entry of r in $L'(v)$, whereas every vertex v appearing in $\mathcal{Q}_l$ must have $(r, d_{G'}(r, v))) \in L'(v)$. By Lemma 4, this proves both (1) and (2). □

The following theorem states that the minimality of labelling can be preserved by ParDHL.

Theorem 2 *If Γ is minimal, then $\Gamma' =$ ParDHL(G, Γ, B, G') is also minimal.*

Proof The proof for Theorem 1 shows that $(r, d_{G'}(r, v)) \in L'(v)$ for $v \in \mathcal{A}(r, B)$ iff $P_{G'}(r_i, v)$ does not contain any other landmark on any shortest-path between r and v. This ensures the minimality for the labels of all affected vertices. For unaffected vertices, by Definition 5, their labels should remain unchanged. Thus, if Γ is minimal, then Γ' obtained by updating the labels of all affected vertices must also be minimal. □

6.2 Complexity analysis

Let m be the total number of affected vertices w.r.t. a landmark r, l be the average label size of a highway cover distance labelling (i.e., $l = size(L)/|V|$), and d be the average degree of affected vertices.

Algorithm 2 takes $O((m \cdot d \cdot l)/t)$ time and $O(|V|)$ space to find all affected vertices, where t is the total number of parallel processes. Algorithm 3 takes $O(m \cdot d)$ to compute distance bounds. Then, Algorithm 4 takes $O(m \cdot d^2)$ to repair the labels of all affected vertices because the pruning step may be checked for each affected vertex in the worst case. We omit l from $O(m \cdot d)$ for Algorithm 3 and from $O(m \cdot d^2)$ for Algorithm 4 because distances for all unaffected neighbors of affected vertices are stored in Algorithm 2. Therefore, ParDHL takes $O(|R| \times (m \cdot d \cdot l) + (m \cdot d) + (m \cdot d^2)) = O(|R| \times m \cdot d(l + d + 1))$ time and space $O(|V|)$ space. In practice, m is usually orders of magnitudes smaller than $|V|$ and l is also significantly smaller than $|R|$.

7 Experiments

We implemented our proposed method ParDHL to answer the following questions:

Q1. How efficiently can ParDHL process graph updates in comparison with the-state-of-the-art dynamic algorithms?
Q2. How does the number of landmarks affect the performance of ParDHL?
Q3. What is the effect of the size of updates on the performance of ParDHL?
Q4. Is there an upper bound on the size of updates, for which ParDHL is better than reconstruction (i.e., recomputing a labelling from scratch) and online search algorithms without using any labelling?

7.1 Experimental setup

We implemented our method in C++11 and compiled it then using gcc 5.5.0 with the -O3 option. We performed all the experiments using 28 threads on a Linux server (Intel Xeon W-2175 with 2.50GHz, 28 cores and 512GB of main memory).

7.1.1 Baseline methods

We compared our method with the following state-of-the-art algorithms:

- FulFD: a fully dynamic algorithm proposed in [10] which combines a distance labelling with a graph traversal algorithm to answer distance queries.
- FulHL: a fully dynamic labelling algorithm proposed in [11] which combines a highway cover labelling with graph traversal algorithm to answer distance queries.
- IncHL$^+$: an online incremental algorithm proposed in [25], which combines a highway cover labelling with a graph traversal algorithm for answering distance queries;
- Opt-BiBFS: an online bidirectional BFS algorithm to answer distance queries, using an optimized strategy to expand searches from a direction with less vertices [10].

Besides these methods, there are several other methods for answering distance queries on dynamic graphs, such as FulPLL [9], DecM [6], and WPSL [28] which can only process one single update at a time. Since the experimental results of the previous works [10, 11, 28] have shown that FulFD and FulHL outperform FulPLL, and WPSL outperforms DecM and can only scale to graphs with millions of nodes and edges, we omit the comparison with these methods. To distinguish the parallelism power of landmark parallelism from anchor parallelism, we also consider another variant of our proposed method, called ParDHL$^-$, which is obtained by removing landmark parallelism from ParDHL. The code of FulFD

Table 1 Summary of datasets

Dataset	Network	$\|V\|$	$\|E\|$	Avg. deg.	Avg. dist.
Skitter	comp (u)	1.7M	11M	13.08	5.0
Hollywood	social (u)	1.1M	114M	98.91	3.9
Orkut	social (u)	3.1M	117M	76.28	4.2
Enwiki	social (d)	4.2M	101M	43.75	3.4
Livejournal	social (d)	4.8M	69M	17.68	5.6
Indochina	web (d)	7.4M	194M	40.73	7.7
IT	web (d)	41M	1.2B	49.77	7.0
Twitter	social (d)	42M	1.5B	57.74	3.6
Friendster	social (u)	66M	1.8B	55.06	5.0
UK	web (d)	106M	3.7B	62.77	6.9

and FULHL was provided by their authors and implemented in C++. We used the same parameter settings as suggested by their authors unless otherwise stated. To have a fair comparison, we set the number of landmarks to 20.

7.1.2 Datasets

We used 10 real-world large complex networks to verify the efficiency of our algorithms. We treated these networks as undirected and unweighted graphs. The statistics of these datasets are summarized in Table 1. They can be easily downloaded though the links of the Stanford Network Analysis Project [30] and the Laboratory for web Algorithmics [31].

7.1.3 Test data generation

We applied the following principles to sample updates and queries in our experiments.

For updates, we considered three update settings: (1) *fully dynamic* - contains 50% edge insertions and 50% edge deletions, (2) *incremental* - contains only edge insertions, and (3) *decremental* - contains only edge deletions. For each update setting, we randomly generated 10 sequences of updates, where each sequence contains 1,000 updates. These settings enable us to explore the impacts of edge insertions and edge deletions respectively, in addition to their combined impact.

In Figure 5, we report the distance distribution for updates in the incremental setting before applying updates and the decremental setting after applying updates. For the incremental setting, the distances between vertices of updates mostly range from 2 to 8 in all datasets, except for only Indochina and UK which have 15%-30% of updates with distances larger than 8. For the decremental setting, the distances between vertices of updates in all datasets are small ranging from 1 to 6, which shows that the updates are selected from densely connected components of these networks allowing us to evaluate the methods more effectively. Further, only a small number of updates are disconnected (i.e., have distance ∞) in most of the datasets.

For queries, we randomly sampled 100,000 pairs of vertices in each dataset to evaluate the average querying time on graphs after being changed as a result of updates. We also report the average size of distance labellings after being changed as a result of updates.

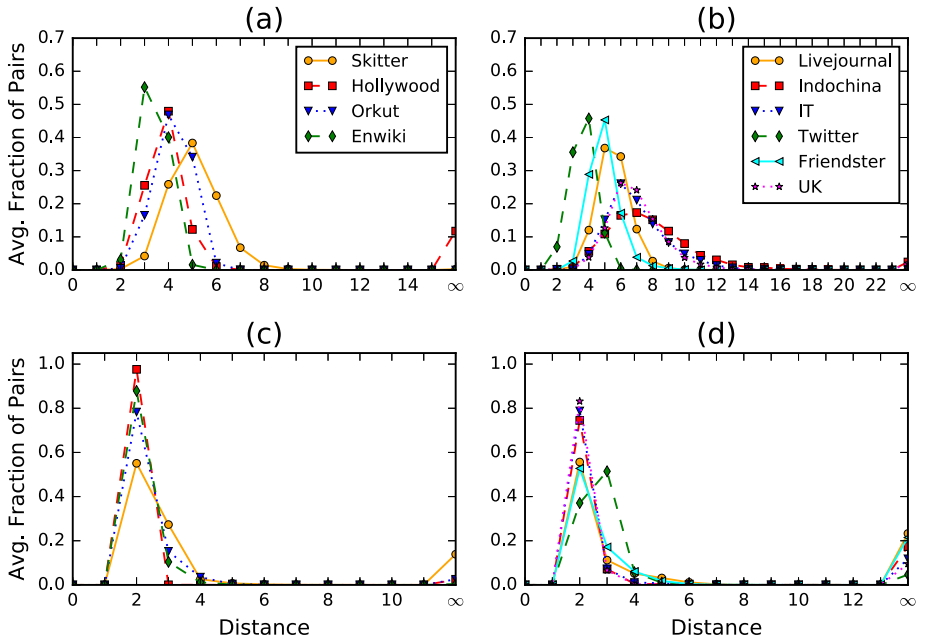

Fig. 5 Distance distributions for updates: (a)-(b) in the incremental setting (only edge insertions), and (c)-(d) in the decremental setting (only edge deletions)

7.2 Performance comparison

In the following, we compare our proposed method with the baseline methods in terms of update time, labelling size and query time.

7.2.1 Update time

Tables 2, 3 and 4 show the average update time of the proposed and baseline methods after performing updates in different settings. We also report the fraction of affected vertices (i.e., $\frac{|\mathcal{A}|}{|V|}$) for 1000 updates which is averaged over 10 sequences for our algorithms. From Table 2, we see that the average update time of our method ParDHL is considerably less than FulHL and significantly much less than FulFD on all the datasets in the fully dynamic setting. We can also see that ParDHL$^-$ is comparable with FulHL and significantly outperforms FulFD. Our proposed methods have promising results for large datasets. In particular, they process updates on networks with over billions of edges and on networks that have large fractions of affected vertices, much more efficiently than the baseline methods.

Table 3 shows that the average update time of ParDHL outperforms IncHL on all the datasets, and outperforms IncHL$^+$ and IncFD except on Twitter and Friendster in the incremental setting. This is because ParDHL leverages advantages from parallelism when a large fraction of vertices is affected and as we can see that the fraction is negligibly small for Twitter and Friendster which make it slower on these datasets as compared to sequential methods IncFD and IncHL$^+$. We can also see that ParDHL$^-$ has comparable results

Table 2 Comparison of the update time between our methods and the baseline methods. The update time is reported for 1,000 updates, averaged over 10 sequences

Dataset	Fully Dynamic Update Time (sec.)				$\frac{\lvert\mathcal{A}\rvert}{\lvert V\rvert}$
	ParDHL$^-$	ParDHL	FulHL	FulFD	
Skitter	1.053	0.249	0.479	20.22	0.7857
Hollywood	0.188	0.046	0.117	3.136	0.0379
Orkut	4.118	0.998	1.833	35.98	0.2780
Enwiki	5.103	2.244	3.586	115.9	0.4960
Livejournal	0.591	0.233	0.502	13.07	0.0492
Indochina	5.182	1.487	2.674	270.7	2.1660
IT	44.04	9.630	55.74	416.9	2.9187
Twitter	62.82	31.24	129.8	5010	0.3939
Friendster	4.597	1.856	44.31	17.16	0.0009
UK	33.14	11.22	74.56	431.0	0.6949

Table 3 Comparison of the update time in the incremental setting between our methods and the baseline methods

Dataset	Incremental Update Time (sec.)					$\frac{\lvert\mathcal{A}\rvert}{\lvert V\rvert}$
	ParDHL$^-$	ParDHL	IncHL	IncHL$^+$	IncFD	
Skitter	0.446	0.090	0.170	0.457	0.576	0.3847
Hollywood	0.167	0.040	0.071	0.081	0.087	0.0405
Orkut	1.949	0.430	1.698	3.026	1.990	0.1700
Enwiki	0.285	0.107	0.284	0.229	0.157	0.0075
Livejournal	0.300	0.096	0.222	0.325	0.251	0.0209
Indochina	6.877	1.214	3.525	167.7	504.5	2.9487
IT	60.89	12.88	62.45	95.92	335.7	4.2755
Twitter	2.891	1.198	14.16	0.037	0.107	0.0004
Friendster	4.388	1.724	21.57	0.169	0.220	0.0006
UK	20.56	5.505	42.27	21.49	469.5	0.5029

with the baseline methods IncHL and IncFD and particularly performs well for datasets which have relatively large fractions of affected vertices. Overall, Our methods can perform incremental updates more efficiently under large fractions of affected vertices.

We can also verify from Table 4 that the average update time of ParDHL is significantly faster than the state-of-the-art methods DecHL and DecFD on all the datasets in the decremental setting. Particularly, ParDHL has much improved results on networks with high average degree such as Twitter and Hollywood. Due to the inherent complexity of decremental operation on graphs (i.e., increasing distances), DecFD takes very long time in identifying and updating labels of affected vertices. We can also observe that ParDHL$^-$

Table 4 Comparison of the update time in the decremental setting between our methods and the baseline methods

Dataset	Decremental Update Time (sec.)				$\frac{\lvert\mathcal{A}\rvert}{\lvert V\rvert}$
	ParDHL$^-$	ParDHL	DecHL	DecFD	
Skitter	1.214	0.317	0.700	24.22	0.7466
Hollywood	0.218	0.053	0.129	6.912	0.0368
Orkut	4.859	1.090	1.537	69.44	0.3197
Enwiki	8.433	3.534	6.539	239.1	0.8536
Livejournal	1.032	0.384	0.665	15.00	0.0937
Indochina	2.111	1.265	1.425	43.12	0.4162
IT	8.351	3.54	31.26	350.5	0.1637
Twitter	90.03	50.02	231.9	10628	0.4971
Friendster	4.784	1.957	44.62	28.54	0.0012
UK	31.76	13.07	70.53	254.1	0.5165

outperforms DecFD and is comparable with DecHL and again better exploits parallelism on datasets with large fraction of affected vertices w.r.t. dataset size.

7.2.2 Labelling size

Table 5 shows that ParDHL has significantly smaller labelling sizes as compared to FulFD on all the datasets. We did not provide the labelling sizes of FulHL because it is also designed based on highway cover distance labelling and due to the minimality property has the same labelling sizes as ParDHL. The labelling size of FulFD remains unchanged at all times because they maintain fixed bit-parallel shortest-path trees. On the other hand, ParDHL stores pruned shortest-path trees; therefore, to preserve the property of minimality, labels need to be added or deleted as a result of graph updates. However, the labelling size of ParDHL remains stable in practice because the average label size is bounded by a constant, i.e. the number of landmarks.

7.2.3 Query time

Table 5 shows the average query time of ParDHL is comparable with FulFD. Again, we did not provide query time of FulHL because it has the same results as our method ParDHL. It has been previously shown [9] that the average query time is mainly dependent on labelling size. Since the dynamic operations do not considerably affect the labelling size for ParDHL and FulFD, their query time also remains stable.

We compare the total time of querying and updating of our methods with the baseline methods in Figure 6. For a fair comparison, we take the sum of the total update time for randomly sampled updates of varying sizes i.e., 1 to 10,000 plus the query time of 1,000 queries after applying the updates as the total time of our methods, denoted as ParDHL+QT and ParDHL$^-$+QT, and the baseline methods, denoted as FulHL+QT and FulFD+QT.

Table 5 Comparison of the labelling size and query time between our proposed methods and the baseline methods

Dataset	Query Time [ms]		Labelling Size	
	ParDHL	FulFD	ParDHL	FulFD
Skitter	0.029	0.020	42 MB	153 MB
Hollywood	0.026	0.036	27 MB	263 MB
Orkut	0.102	0.156	70 MB	711 MB
Enwiki	0.053	0.051	82 MB	608 MB
Livejournal	0.043	0.051	122 MB	663 MB
Indochina	0.788	0.767	85 MB	838 MB
IT	1.167	1.103	866 MB	4.74 GB
Twitter	0.868	0.174	1.14 GB	3.83 GB
Friendster	0.815	0.902	2.43 GB	9.14 GB
UK	1.174	5.233	1.78 GB	11.8 GB

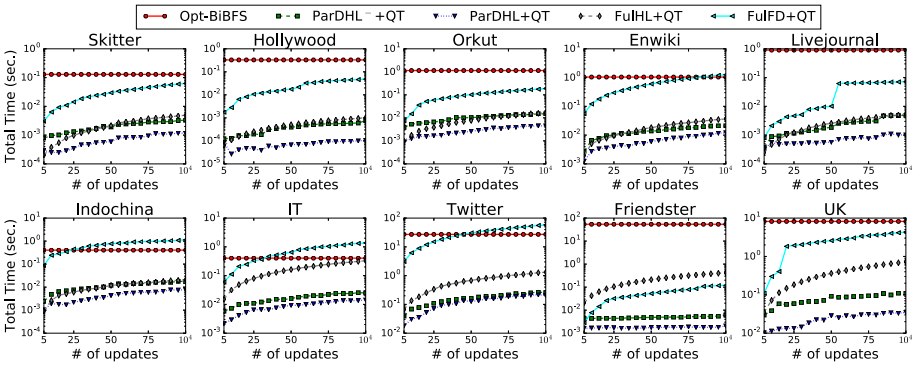

Fig. 6 Comparison between the total time of querying and updating w.r.t varying size of updates up to 10,000 updates of the proposed methods ParDHL$^-$ and ParDHL, and the baseline methods Opt-BiBFS, FulHL, and FulFD

For the baseline method Opt-BiBFS, we take only the query time of 1,000 queries after applying the updates. We see that, even adding the update time of maintaining the labelling under updates of varying sizes, the overall query performance of our methods is significantly better than the baseline methods on all the datasets. In particular, our methods show a promising query performance on large networks Twitter, Friendster and UK.

7.3 Impact of varying landmarks

We analyse the performance of our proposed method ParDHL with the baseline methods FulHL and FulFD under varying landmarks, i.e., $|R| \in [10, 20, 30, 40, 50, 150]$.

Figure 7 shows that the update time of our method ParDHL and the baseline methods FulHL and FulFD, after applying a sequence of 1000 updates in the fully dynamic setting, under varying landmarks. We can see from Figure 7 that our method ParDHL outperforms FulHL and FulFD on all the datasets against each setting of landmarks. This

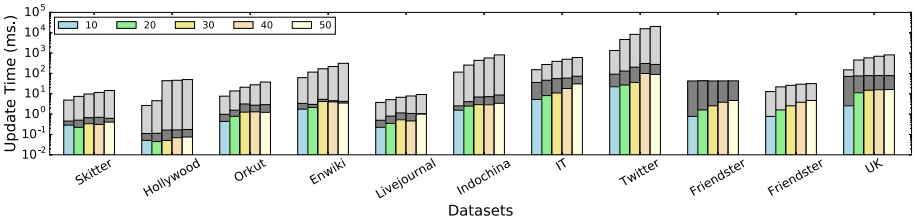

Fig. 7 Update time of ParDHL (in colored bars) and the baseline methods FulHL (in colored plus grey bars) and FulFD (in colored plus grey plus light grey bars) under 10-50 landmarks

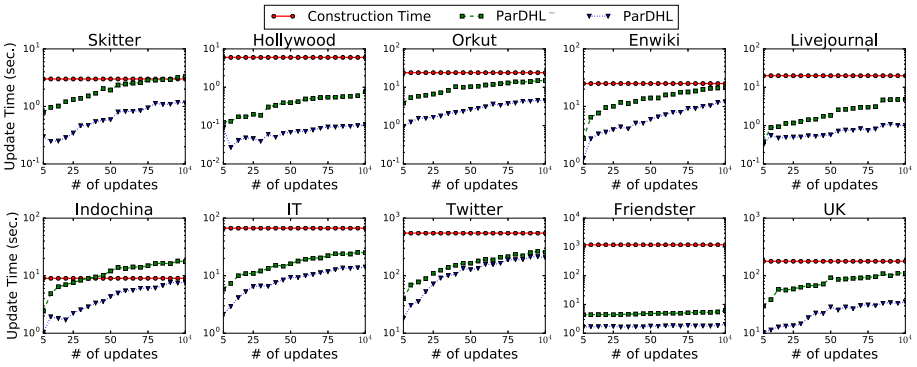

Fig. 8 Update time of ParDHL$^-$ and ParDHL for performing updates of varying size up to 10,000 updates against construction time of highway cover labelling from scratch

is due to parallel searches for finding affected vertices and the novel pruning approach for repairing labels. Particularly, our method ParDHL can perform much better than FulHL and FulFD on large datasets with over billions of edges when the number of landmarks is increased. For clarity of performance illustration, we present results for Friendster separately for FulHL and FulFD.

7.4 Impact of varying size of updates

We evaluate the performance of our methods ParDHL and ParDHL$^-$ against the increasing size of updates that are selected randomly. We start with 500 updates and then iteratively add 500 updates up to 10,000 updates in the fully dynamic setting.

Figure 8 shows the average update time after constructing a distance labelling from scratch, and updating the distance labelling using our fully dynamic algorithms after each increase. We observe from Figure 8 that our methods perform well on all the datasets i.e., the update time remains lower under the construction time for almost all the datasets. We can also observe that increasing the size of updates tends towards slower increase in update time which shows that parallel searches and efficient repairing under large sizes of updates is much more efficient in processing the graph updates. It is worth noticing that ParDHL is more efficient than ParDHL$^-$ which shows that landmarks parallelism in ParDHL can be better leveraged for larger sizes of updates particularly for Indochina, IT and UK. This is

due to the updates in these networks have larger distances, as shown in their distance distributions in Figure 5 which may result in much more affected vertices and thus parallelism is better leveraged.

8 Extensions

We can easily extend our proposed method to directed or weighted graphs.

For directed graphs, more specifically, we can redefine $d_G(s,t)$ as the distance from vertex s to vertex t. We store two label sets for each vertex $v \in V$, namely *forward label* $L_f(v)$ and *backward label* $L_b(v)$, which contain pairs $(r_i, \delta_{r_i v})$ after performing forward and backward pruned BFSs w.r.t. each landmark $r_i \in R$, respectively. We also store two highways, namely *forward highway* $H_f = (R, \delta_{H_f})$ and *backward highway* $H_b = (R, \delta_{H_b})$ such that for any two landmarks $r_i, r_j \in R$, $\delta_{H_f}(r_i, r_j) = d_G(r_i, r_j)$ and $\delta_{H_b}(r_j, r_i) = d_G(r_j, r_i)$.

For maintaining a precomputed highway cover distance labelling as a result of graph changes, we run our method twice, one for fixing the forward labels and highway and the other for fixing the backward labels and highway. To repair the forward labels and highway, we first perform parallel BFSs using Algorithm 2 on the forward adjacency list of a changed graph to find the set of all affected vertices. Then, we find boundary vertices using Algorithm 3 by checking the neighbors of all affected vertices in the reverse adjacency list of a changed graph. Finally, we repair the forward labels and highway starting from the boundary vertices that have the minimum distance using the forward adjacency list of a changed graph and infer the distances of affected vertices on the changed graph via a level-by-level inference. Similarly, the backward labels and highway can be repaired in the same manner. For a given query pair (s,t), we can use $L_f(s)$ and $L_b(t)$ to compute the upper bound distance from s to t in the same way as described in (3).

We can also extend our proposed method to non-negative weighted graphs. In such cases, we use Dijkstra's algorithm in place of BFSs in order to compute and maintain a highway cover distance labelling for dynamic weighted graphs.

9 Conclusion and future work

In this article, we have proposed a novel parallel method for answering distance queries on dynamic graphs. Our proposed method exploits anchor parallelism by parallelising searches for multiple updates that can find affected vertices simultaneously. We have also introduced an efficient repairing mechanism based on the observation of boundary vertices, which can bound a search space to only affected vertices while repairing their labels. Our repairing mechanism uses a novel pruning strategy to further bound the search space of affected vertices for efficient maintenance of a highway cover distance labelling. We have analyzed the correctness and complexity of our method and showed that it preserves the labelling minimality. We have empirically verified the efficiency, scalability and robustness of our method on 10 real-world networks.

For future work, we plan to further investigate the following research directions: 1) it would be interesting to explore the opportunity to extend the proposed algorithms to road networks, and 2) re-positioning/selection of landmarks in a dynamic setting in order to reduce the size of the labelling and hence of the query time. Re-selection of highly central

landmarks could also be required after a certain amount of changes occurring on the topological structure of a dynamic network. This would help optimize the size of a highway cover distance labelling and query performance. Therefore, it is also interesting to explore the problems such as: after how much changes on the topological structure of a dynamic graph, re-positioning of landmarks is required.

Acknowledgements Not applicable.

Author Contributions Not applicable.

Funding Open Access funding enabled and organized by CAUL and its Member Institutions

Data Availability Not applicable.

Declarations

Ethics approval and consent to participate Not applicable.

Human and Animal Ethics Not applicable.

Consent for Publication Not applicable.

Competing interests The authors have no financial or non-financial interests to disclose that are related to the content of this article.

References

1. Potamias, M., Bonchi, F., Castillo, C., Gionis, A.: Fast shortest path distance estimation in large networks. In: Proceedings of the 18th ACM Conference on Information and Knowledge Management, pp. 867–876 (2009)
2. Ukkonen, A., Castillo, C., Donato, D., Gionis, A.: Searching the wikipedia with contextual information. In: Proceedings of the 17th ACM Conference on Information and Knowledge Management. CIKM, pp. 1351–1352. https://doi.org/10.1145/1458082.1458274 (2008)
3. Vieira, M.V., Fonseca, B.M., Damazio, R., Golgher, P.B., Reis, D.dC., Ribeiro-Neto, B.: Efficient search ranking in social networks. In: Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management. CIKM, pp. 563–572. https://doi.org/10.1145/1321440.1321520 (2007)
4. Backstrom, L., Huttenlocher, D., Kleinberg, J., Lan, X.: Group formation in large social networks: Membership, growth, and evolution. In: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD, pp. 44–54. https://doi.org/10.1145/1150402.1150412 (2006)
5. Abraham, I., Delling, D., Goldberg, A.V., Werneck, R.F.: Hierarchical hub labeling for shortest paths. In: Proceedings of the 20th Annual European Conference on Algorithms, pp. 24–35 (2012)
6. Delling, D., Goldberg, A.V., Pajor, T., Werneck, R.F.: Robust distance queries on massive networks. In: European Symposium on Algorithms, Berlin, Heidelberg, pp. 321–333. https://doi.org/10.1007/978-3-662-44777-2_27 (2014)
7. Boccaletti, S., Latora, V., Moreno, Y., Chavez, M., Hwang, D.-U.: Complex networks: structure and dynamics. Phys Rep **424**(4-5), 175–308 (2006). https://doi.org/10.1016/j.physrep.2005.10.009

8. Qin, Y., Sheng, Q.Z., Falkner, N.J., Yao, L., Parkinson, S.: Efficient computation of distance labeling for decremental updates in large dynamic graphs. World Wide Web **20**(5), 915–937 (2017). https://doi.org/10.1007/s11280-016-0421-1
9. D'Angelo, G., D'Emidio, M., Frigioni, D.: Fully dynamic 2-hop cover labeling. J Exp Algo., vol. 24(1). https://doi.org/10.1145/3299901 (2019)
10. Hayashi, T., Akiba, T., Kawarabayashi, K.-I.: Fully dynamic shortest-path distance query acceleration on massive networks. In: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, pp. 1533–1542 (2016)
11. Farhan, M., Wang, Q., Lin, Y., McKay, B.: Fast fully dynamic labelling for distance queries. VLDB J. **31**(3), 483–506 (2022). https://doi.org/10.1007/s00778-021-00707-z
12. Tarjan, R.E.: Data Structures and Network Algorithms. Society for Industrial and Applied Mathematics, 3600 University City Science Center Philadelphia, PA, United States. https://doi.org/10.1137/1.9781611970265 (1983)
13. Akiba, T., Iwata, Y., Yoshida, Y.: 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, pp. 349–360 (2013)
14. Fu, A.W.-C., Wu, H., Cheng, J., Wong, R.C.-W.: Is-label: an independent-set based labeling scheme for point-to-point distance querying. Proc VLDB Endow **6**(6), 457–468 (2013)
15. Jin, R., Ruan, N., Xiang, Y., Lee, V.: A highway-centric labeling approach for answering distance queries on large sparse graphs. In: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, pp. 445–456 (2012)
16. Abraham, I., Delling, D., Goldberg, A.V., Werneck, R.F.: A hub-based labeling algorithm for shortest paths in road networks. In: Proceedings of the 10th International Conference on Experimental Algorithms, pp. 230–241 (2011)
17. Wei, F.: Tedi: efficient shortest path query answering on graphs. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, pp. 99–110 (2010)
18. Farhan, M., Wang, Q., Lin, Y., McKay, B.D.: A highly scalable labelling approach for exact distance queries in complex networks. In: 22Nd International Conference on Extending Database Technology EDBT, pp. 13–24 (2019)
19. Chang, L., Yu, J.X., Qin, L., Cheng, H., Qiao, M.: The exact distance to destination in undirected world. VLDB J. **21**(6), 869–888 (2012). https://doi.org/10.1007/s00778-012-0274-x
20. Li, W., Qiao, M., Qin, L., Zhang, Y., Chang, L., Lin, X.: Scaling distance labeling on small-world networks. In: Proceedings of the 2019 International Conference on Management of Data, pp. 1060–1077 (2019)
21. Wang, Y., Wang, Q., Koehler, H., Lin, Y.: Query-by-sketch: Scaling shortest path graph queries on very large networks. In: Proceedings of the 2021 International Conference on Management of Data, pp. 1946–1958 (2021)
22. Kumar, R., Novak, J., Tomkins, A.: Structure and evolution of online social networks. In: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD, pp. 611–617. https://doi.org/10.1145/1150402.1150476 (2006)
23. Myers, S.A., Leskovec, J.: The bursty dynamics of the twitter information network. In: Proceedings of the 23rd International Conference on World Wide Web, pp. 913–924 (2014)
24. Akiba, T., Iwata, Y., Yoshida, Y.: Dynamic and historical shortest-path distance queries on large evolving networks by pruned landmark labeling. In: Proceedings of the 23rd International Conference on World Wide Web, pp. 237–248 (2014)
25. Farhan, M., Wang, Q.: Efficient maintenance of distance labelling for incremental updates in large dynamic graphs. arXiv preprint arXiv:2102.08529 (2021)
26. D'Andrea, A., D'Emidio, M., Frigioni, D., Leucci, S., Proietti, G.: Dynamically maintaining shortest path trees under batches of updates. In: International Colloquium on Structural Information and Communication Complexity, pp. 286–297 (2013)
27. Cohen, E., Halperin, E., Kaplan, H., Zwick, U.: Reachability and distance queries via 2-hop labels. In: Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 937–946 (2002)
28. Zhang, M., Li, L., Hua, W., Zhou, X.: Efficient 2-hop labeling maintenance in dynamic small-world networks. In: 2021 IEEE 37th International Conference on Data Engineering (ICDE), IEEE, pp. 133–144 (2021)
29. D'Emidio, M.: Faster algorithms for mining shortest-path distances from massive time-evolving graphs. Algorithms **13**(8), 191 (2020)
30. Leskovec, J., Sosič, R.: Snap: a general-purpose network analysis and graph-mining library. ACM Trans Intell Syst Technol, vol. 8(1). https://doi.org/10.1145/2898361 (2016)

31. Boldi, P., Vigna, S.: The webgraph framework i: compression techniques. In: Proceedings of the 13th International Conference on World Wide Web. WWW, pp. 595–602. https://doi.org/10.1145/988672.988752 (2004)