Relative Subboundedness of Contraction Hierarchy and Hierarchical 2-Hop Index in Dynamic Road Networks

Yikai Zhang
Chinese University of Hong Kong
ykzhang@se.cuhk.edu.hk

Jeffrey Xu Yu
Chinese University of Hong Kong
yu@se.cuhk.edu.hk

ABSTRACT

Computing the shortest path for any two given vertices is an important problem in road networks. Since real road networks are dynamically updated due to real-time traffic conditions and it is costly to recompute the oracle O in use from scratch, O needs to be updated to reflect the changes in the network using incremental algorithms. An incremental algorithm is said to be *bounded* if its cost is polynomial in |CHANGED|, where CHANGED comprises both the changes to the graph and the resulting changes to O. An incremental problem is *bounded* if it has a bounded algorithm and is *unbounded* otherwise. We study the boundedness of the incremental counterparts of two state-of-the-art oracles, namely *contraction hierarchy* (CH) and *hierarchical 2-hop index* (H2H). We prove that under specific computational models, both CH and H2H are unbounded to maintain. Despite this fact, we introduce *relative subboundedness* as an alternative to boundedness. We prove that the state-of-the-art incremental algorithm for CH is relatively subbounded, and moreover, we propose a relatively subbounded algorithm for H2H. Our experimental study on real road networks shows that the algorithms studied are faster than recomputing from scratch even when 10% of the index needs to be updated, thereby verifying the effectiveness of relative subboundedness.

CCS CONCEPTS

• **Theory of computation → Dynamic graph algorithms**.

KEYWORDS

Boundedness; Relative Subboundedness; Dynamic Road Networks

ACM Reference Format:
Yikai Zhang and Jeffrey Xu Yu. 2022. Relative Subboundedness of Contraction Hierarchy and Hierarchical 2-Hop Index in Dynamic Road Networks. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22), June 12–17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3514221.3517875

1 INTRODUCTION

As an important class of graphs in graph analytics, road networks have attracted much attention from both academic and industrial communities. Specifically, a road network is a weighted graph $G = (V, E, \phi)$, where a vertex $v \in V$ represents an intersection, an edge $e \in E$ represents a road, and the weight $\phi(e)$ of an edge e may represent the transit time, physical distance or any other metric of the road. Given a road network, the problem of efficiently computing the shortest path or distance between two given vertices is important in that it benefits many applications such as route planning, POI recommendation and GPS navigation. Because of its significance, many path and distance oracles (*oracles* for short) for road networks have been proposed, such as contraction hierarchy (CH) [26], arterial hierarchy (AH) [52], hierarchical 2-hop index (H2H) [37] and projected vertex separator based 2-hop index (P2H) [13], each with a different tradeoff among indexing time, index space and query time. All these oracles are for static networks.

However, in real road networks, while the vertices and edges can be reasonably assumed to be intact (since road construction and destruction are rare in practice), the weights (*e.g.*, transit time) of the edges may continuously change over time due to real-time traffic conditions. Accordingly, the shortest path and distance between two vertices may also change. As a result, it is required that the oracle in use be updated to reflect the changes in the road network. Formally, let $O(G)$ be the oracle currently in use, ΔG be a set of weight updates to G and $G \oplus \Delta G$ be the resulting network after applying ΔG to G. Given ΔG, the problem of oracle maintenance is to update O, thereby obtaining $O(G \oplus \Delta G)$.

Obviously, it is too costly to compute $O(G \oplus \Delta G)$ starting from scratch where G is large and ΔG is small. After all, the state-of-the-art oracles all require a non-trivial amount of computation to construct. In light of this, it is more practical to maintain O incrementally by using an incremental algorithm. Specifically, fed with G, ΔG and $O(G)$, an incremental algorithm $\mathcal{A}_\Delta$ for O computes an update ΔO to $O(G)$ such that $O(G \oplus \Delta G) = O(G) \oplus \Delta O$. Such an algorithm is usually more efficient because (1) when ΔG is small (*e.g.*, a single weight update), ΔO is also likely to be small; and (2) $\mathcal{A}_\Delta$ can effectively capitalize on the previous oracle $O(G)$, thereby avoiding some duplicate computation when dealing with ΔG.

An incremental algorithm $\mathcal{A}_\Delta$ for O is desired to be *bounded* [8, 41]. Specifically, $\mathcal{A}_\Delta$ is bounded if there exists a polynomial function f such that the cost of $\mathcal{A}_\Delta$ is bounded by $O(f(|\text{CHANGED}|))$, where $|\text{CHANGED}| = |\Delta G| + |\Delta O|$ is the amount of changes in the input and output after applying ΔG to G. Intuitively, |CHANGED| represents the minimum amount of work that *any* incremental algorithm for O needs to do in order to handle ΔG and is likely to be small when $|\Delta G|$ is small. Therefore, a bounded $\mathcal{A}_\Delta$ guarantees the efficient updatability of O, compared with recomputing O from scratch. The problem of maintaining oracle O is *bounded* if a bounded $\mathcal{A}_\Delta$ for O exists and is *unbounded* otherwise.

We study the boundedness of contraction hierarchy (CH) [26] and hierarchical 2-hop index (H2H) [37] in this paper. We select

ACM ISBN 978-1-4503-9249-5/22/06…$15.00
https://doi.org/10.1145/3514221.3517875

these two oracles for the following reasons: (1) CH is a successful oracle with outstanding performance in practice, and efficient incremental algorithms [39, 48] have been proposed to update CH; however, the (un)boundedness of incremental CH has not been investigated yet; and (2) H2H is one of the state-of-the-art oracles and has been a building block for other applications such as nearest neighbor search [31, 38]; the incremental counterpart of H2H, however, has not been well studied in the literature. In view of these facts, we would like to close these gaps in this paper.

Contributions. Our main contributions are as follows.

Theoretical Results. This paper is one of few works [8, 21, 22, 41] that study boundedness. Under the class of *locally persistent* (LP) algorithms, which is the standard model of computation to study boundedness [8, 21, 22, 41], we prove both the problem of maintaining CH and that of maintaining H2H are unbounded even for single weight updates (*i.e.*, only one edge has weight changed). We also consider two classes of incremental algorithms, denoted $\mathsf{INC}_{\mathsf{CH}}$ and $\mathsf{INC}_{\mathsf{H2H}}$, that include practically efficient algorithms [39, 48] for CH and H2H, respectively. We prove that the algorithms in $\mathsf{INC}_{\mathsf{CH}}$ and $\mathsf{INC}_{\mathsf{H2H}}$ are also unbounded for single weight updates.

Relative Subboundedness. In spite of the impossibility results, we show that both CH and H2H can still be maintained more efficiently than recomputing from scratch. Specifically, as an alternative to boundedness, we propose *relative subboundedness* for CH and H2H. In detail, let Q be either the problem of constructing CH or the problem of constructing H2H on a graph G. Let $Q(G)$ denote the result of Q on G (*e.g.*, the CH oracle constructed on G). Then the incremental counterpart of Q is to update the result from $Q(G)$ to $Q(G \oplus \Delta G)$ when an update ΔG is applied to G. The introduction of relative subboundedness is to characterize the efficiency of an incremental algorithm $\mathcal{A}_\Delta$ that maintains $Q(\cdot)$.

Let $D(G)$ and $D(G \oplus \Delta G)$ denote the data that every algorithm for Q needs to inspect when computing $Q(G)$ and $Q(G \oplus \Delta G)$, respectively. AFF is defined as the part of $D(G \oplus \Delta G)$ that differs from $D(G)$. Intuitively, it captures the part *affected* by ΔG. Let $\mathcal{A}$ be any algorithm for Q. We define $\|\mathsf{AFF}\|$ as the part of time spent on AFF by $\mathcal{A}$ when computing $Q(G \oplus \Delta G)$ and say an incremental algorithm $\mathcal{A}_\Delta$ that updates the result from $Q(G)$ to $Q(G \oplus \Delta G)$ is *subbounded relative to* $\mathcal{A}$ if its cost is of $O(\|\mathsf{AFF}\| \log \|\mathsf{AFF}\|)$. By definition, we have $\|\mathsf{AFF}\| \leq \mathcal{T}_{\mathcal{A}}$, where $\mathcal{T}_{\mathcal{A}}$ is the cost of $\mathcal{A}$ to compute $Q(G \oplus \Delta G)$, *i.e.*, the cost to deal with ΔG by recomputing from scratch. Hence, an incremental algorithm $\mathcal{A}_\Delta$ subbounded relative to $\mathcal{A}$ is likely to outperform recomputing from scratch using $\mathcal{A}$.

Compared with *relative boundedness* [21], the definition of relative subboundedness employs a different characterization of affected area (*i.e.*, $\|\mathsf{AFF}\|$) and requires that an algorithm be linearithmically dependent on $\|\mathsf{AFF}\|$ instead. Therefore, in general, neither of them can dominate the other. Ideally, an incremental algorithm is desired to be both relatively subbounded and relatively bounded. However, our study on CH and H2H shows that relative subboundedness alone is enough to warrant efficiency.

We prove that the state-of-the-art incremental algorithm DCH [39] for CH (1) is *subbounded relative to* CHIndexing under weight-increase updates, where CHIndexing [39, 48] is the state-of-the-art algorithm to construct CH, and (2) is both *subbounded* and *bounded relative to* CHIndexing under weight-decrease updates. Therefore, CH can still be maintained more efficiently than recomputing from scratch, although no bounded algorithms are known.

New Algorithms. In addition, we propose a new incremental algorithm IncH2H to maintain H2H. We prove that IncH2H (1) is *subbounded relative to* H2HIndexing under weight-increase updates, where H2HIndexing [37] is the state-of-the-art algorithm to construct H2H, and (2) is both *subbounded* and *bounded relative to* H2HIndexing under weight-decrease updates.

Experimental Evaluation. We conducted experiments to verify the effectiveness of relative subboundedness on nine road networks. Our experiments show that DCH and IncH2H respectively are able to outperform CHIndexing and H2HIndexing even if 10% of the index needs to be updated, indicating the superiority of relatively subbounded algorithms over recomputing from scratch. Therefore, relative subboundedness is an effective alternative to boundedness and may be used to characterize the update efficiency of other unbounded incremental problems. We also conducted experiments to compare CH and H2H in dynamic networks. From the experiments, we find that CH and H2H show different tradeoffs between query time and update time. While H2H takes more time to update, it provides striking query performance in practice.

Related Work. We classify the related work as follows.

Static Road Networks. Many oracles have been proposed for static road networks where the weights of the edges are fixed. Dijkstra's algorithm is a classic algorithm for computing shortest path. Many different techniques, such as ALT [28], REACH [30] and *arc-flag* [32], have been proposed to reduce the search space of Dijkstra's algorithm, thereby gaining performance improvement. *Highway Hierarchy* (HH) [43] exploits the hierarchical nature of road networks so that queries can be answered by searching the sparse high levels. *Contraction Hierarchy* (CH) [26] is an oracle with outstanding overall performance. Its efficiency relies heavily on the notion of *shortcut*, which represents a shortest path in the graph. *Arterial Hierarchy* (AH) [52] is similar to CH except that it further exploits spatial information. Spatial-coherence-based methods such as *Spatially Induced Linkage Cognizance* (SILC) [42] and *Path-Coherent Pair Decomposition* (PCPD) [44] exploit the *spatial coherence* among shortest paths. Both indices require the precomputation of all-pairs shortest paths and thus are prohibitively expensive to construct. *Hierarchical Labeling* (HL) [2] and *Pruned Highway Labeling* (PHL) [4] are two indices based on hub labeling [15]. Such methods permit efficient query evaluation since they do not need to do any search when answering a query. *Hierarchical 2-Hop Index* (H2H) [37] combines the merits of hierarchical methods and hub labeling methods to provide striking query performance. *Projected Vertex Separator Based 2-Hop Index* (P2H) [13] improves on H2H by further reducing the hubs to inspect during query processing. *Customizable Route Planning* (CRP) [19] is an oracle supporting any metric in a unified manner.

Dynamic Road Networks. The weights of the edges in real road networks may change over time (e.g., from normal condition to congestion). For this reason, several algorithms have been proposed to update oracles in dynamic road networks. In [19], an algorithm to maintain CRP is presented. Ouyang et al. [39] proposed a shortcut-centric algorithm called DCH to update CH and showed its superiority over both CRP [19] and the vertex-centric algorithm proposed in [27]. Independently in [48], Wei et al. proposed another algorithm UE to maintain CH. As baselines, the authors of [48] also proposed algorithms such as PCPDAdapt, SILCAdapt, H2HAdapt and so on for the corresponding oracles. Their experiments show that CH + UE has the best overall performance. It is worth mentioning that the aforementioned H2HAdapt may fail to correctly update H2H even in some trivial cases (please refer to the full version [46] for a counterexample). In [13], an algorithm, denoted DynH2H, for maintaining H2H is shown. Independently, in [51], Zhang et al. proposed an algorithm, DTDHL, which can be considered as an optimized version of DynH2H and thus is the state of the art to maintain H2H. Still, DTDHL may take seconds even if only one edge has weight changed.

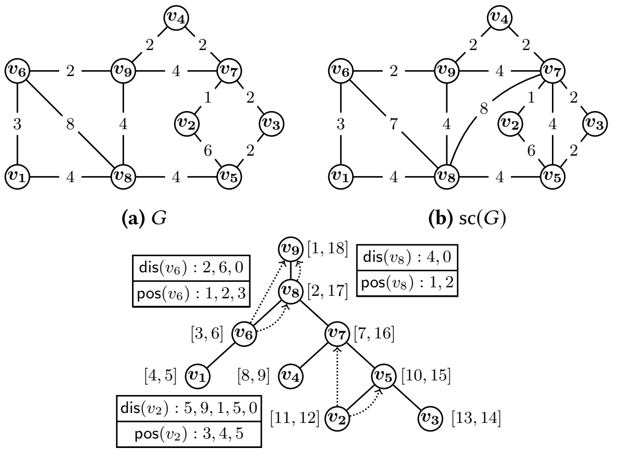
Boundedness. The notion of *boundedness* has been used to evaluate incremental algorithms for problems such as single source reachability [41], graph pattern matching [24] and so on [8, 21]. All unboundedness results in these works are under the model of locally persistent algorithms. Since the notion of boundedness is often too strict, several alternative measures have been proposed. In [24], Fan et al. showed that incremental graph simulation has *semibounded* algorithms. Specifically, let $\mathsf{AFF}^{\forall}$ be the changes in the result and in auxiliary structures that are necessarily maintained for any incremental algorithms for the problem. An incremental algorithm is semibounded if its cost is bounded by a polynomial in $|\Delta G|$, $|\mathsf{AFF}^{\forall}|$ and the size $|P|$ of pattern. In [21], Fan et al. proposed two new standards, namely, *localizable computation* and *relative boundedness*: (1) localizable computation is to restrict incremental computation to a bounded neighborhood around ΔG, and (2) relative boundedness is to characterize the efficiency of an algorithm obtained by incrementalization. We shall discuss their differences with relative subboundedness in detail in Section 4. In [23], it is proved that for a contracting and monotonic fixpoint algorithm, a relatively bounded incremental algorithm can be deduced if one can identify a bounded scope function. However, for both CH and H2H, we are unable to identify such a scope function for the case of weight increase despite our efforts. Therefore, it remains open if relatively bounded incremental algorithms exist.

Organization. The rest of the paper is organized as follows. Section 2 provides the preliminaries. We show our theoretical results in Section 3 and introduce relative subboundedness in Section 4. The incremental algorithms for H2H are in Section 5. Section 6 reports the experimental results and Section 7 discusses algorithms for edge insertion/deletion. Section 8 concludes the paper.

2 PRELIMINARIES

We first introduce some notations, following which we review *contraction hierarchy* [39, 48] and *hierarchical 2-hop index* [37].

Notations. Let $G = (V, E)$ be a connected road network, where V is the vertex set and $E \subseteq V \times V$ is the edge set. For each vertex $v \in V$, let $\mathsf{nbr}(v, G)$ be the neighbors of v in G. For each edge $e \in E$, let $\phi(e, G) \geq 0$ be the weight of e in G. A path p from vertex s to vertex t is defined as a sequence of vertices $(s = v_1, v_2, \ldots, v_\ell = t)$ such that $(v_i, v_{i+1}) \in E$ for $\forall 1 \leq i < \ell$. The weight of p, denoted by $\phi(p, G)$, is defined as $\phi(p, G) = \sum_{i=1}^{\ell-1} \phi((v_i, v_{i+1}), G)$. A path from s to t is shortest if its weight is no larger than any other path from s to t. The shortest distance between s and t in G, denoted by $\mathsf{sd}_G(s, t)$, is defined to be the weight of the shortest paths between s and t. For notational convenience, we omit the G in the notations when the context is clear. Hereafter, we assume that G is undirected for ease of exposition, emphasizing that our results and algorithms can be extended to the directed case. We focus on weight updates and discuss edge insertions/deletions in Section 7.



2
2
2
4
3
8
4
1
2
6
2
4
4
7
8
$\mathsf{dis}(v_6) : 2, 6, 0$
$\mathsf{pos}(v_6) : 1, 2, 3$
$\mathsf{dis}(v_8) : 4, 0$
$\mathsf{pos}(v_8) : 1, 2$
$\mathsf{dis}(v_2) : 5, 9, 1, 5, 0$
$\mathsf{pos}(v_2) : 3, 4, 5$
[1, 18]
[2, 17]
[3, 6]
[7, 16]
[4, 5]
[8, 9]
[10, 15]
[11, 12]
[13, 14]

(a) G **(b)** $\mathsf{sc}(G)$ **(c)** $\mathcal{T}$

Figure 1: Demonstration of CH **and** H2H

Contraction Hierarchy (CH**).** CH is defined as follows.

Indexing. Given a road network G and a total order π over V, CH constructs a shortcut graph $\mathsf{sc}(G)$ of G as the underlying index. Specifically, for a vertex v, let $\pi(v)$ be the position of v in the ordering π. A vertex u is said to have a higher rank than another vertex v if $\pi(u) > \pi(v)$. A path $p = (v_1, v_2, \ldots, v_\ell)$ between v_1 and v_ℓ is said to be a *valley path* [48] if $\pi(v_i) < \min\{\pi(v_1), \pi(v_\ell)\}$ for $\forall 2 \leq i < \ell$; that is, any intermediate vertex of p has a lower rank than the endpoints. As a special case, an edge in G is also a valley path. The shortcut graph $\mathsf{sc}(G)$ is constructed such that for any two distinct vertices u and v, there is an edge (u, v) in $\mathsf{sc}(G)$ if and only if there exists a valley path between u and v in G. Following conventions, we refer to the edges in $\mathsf{sc}(G)$ as *shortcut*s and to the edges in G as *edge*s. Hereafter, we use $\langle u, v \rangle$ to denote a shortcut in $\mathsf{sc}(G)$ and use (u, v) to denote an edge in G.

Each shortcut $e = \langle u, v \rangle$ in $\mathsf{sc}(G)$ is associated with a weight $\phi(e, \mathsf{sc}(G))$ (or $\phi(e)$), which is defined to be the weight of the shortest valley path between u and v in G. For each vertex u, its *upward neighbors*, denoted by $\mathsf{nbr}^+(u)$, in $\mathsf{sc}(G)$ are the neighbors that are ranked higher than u. That is, $\mathsf{nbr}^+(u) = \{v \mid \langle u, v \rangle \in \mathsf{sc}(G) \wedge \pi(v) > \pi(u)\}$. The *downward neighbors* of u, denoted by $\mathsf{nbr}^-(u)$, is defined as $\mathsf{nbr}^-(u) = \{v \mid \langle u, v \rangle \in \mathsf{sc}(G) \wedge \pi(v) < \pi(u)\}$. For a shortcut $\langle u, v \rangle$, a pair of shortcuts $(\langle t, u \rangle, \langle t, v \rangle)$ is said to be a *downward shortcut pair* of $\langle u, v \rangle$ if $\pi(t) < \min\{\pi(u), \pi(v)\}$. We denote by $\mathsf{scp}^-(\langle u, v \rangle)$ all the downward shortcut pairs of $\langle u, v \rangle$.

```
Input : $G$: the road network; $\pi$: an ordering over $V$
Output : the shortcut graph $\mathrm{sc}(G)$ of $G$ under $\pi$
1 $\mathrm{sc}(G) \leftarrow G$;
2 for each $u$ in the order $\pi$ do
3     for distinct $v, w \in \mathrm{nbr}(u, \mathrm{sc}(G))$ with $\pi(v), \pi(w) > \pi(u)$ do
4         if $\langle v, w\rangle \notin \mathrm{sc}(G)$ then
5             insert shortcut $\langle v, w\rangle$ to $\mathrm{sc}(G)$;
6             $\phi(\langle v, w\rangle) \leftarrow \phi(\langle u, v\rangle) + \phi(\langle u, w\rangle)$;
7         else if $\phi(\langle v, w\rangle) > \phi(\langle u, v\rangle) + \phi(\langle u, w\rangle)$ then
8             $\phi(\langle v, w\rangle) \leftarrow \phi(\langle u, v\rangle) + \phi(\langle u, w\rangle)$;
9 return $\mathrm{sc}(G)$;
```

Algorithm 1: CHIndexing

For any shortcut e, as shown in [39], we have

$$\phi(e) = \min\{\phi(e, G), \phi(e'_1) + \phi(e''_1), \ldots, \phi(e'_k) + \phi(e''_k)\} \qquad (\diamond)$$

where (1) (e'_i, e''_i) is the i-th downward shortcut pair of e, and (2) $\phi(e, G)$ is the weight of edge e if e exists in G and is ∞ otherwise. An ordered pair of shortcuts $(\langle u, w\rangle, \langle w, v\rangle)$ is an *upward shortcut pair* of $\langle u, v\rangle$ if $\pi(w) > \pi(u)$ and $\pi(v) > \pi(u)$. Note that $\langle u, v\rangle$ and $\langle u, w\rangle$ make up a downward shortcut pair of $\langle w, v\rangle$. The set of upward shortcut pairs of $\langle u, v\rangle$ is denoted by $\mathrm{scp}^+(\langle u, v\rangle)$.

Example 2.1: For the graph G in Figure 1a, its shortcut graph $\mathrm{sc}(G)$ under the ordering $\pi = (v_1, v_2, \ldots, v_9)$ is shown in Figure 1b. There is a shortcut $\langle v_7, v_8\rangle$ in $\mathrm{sc}(G)$ because valley paths (*e.g.*, $p = (v_7, v_3, v_5, v_8)$) between v_7 and v_8 exist in G. The weight $\phi(\langle v_7, v_8\rangle) = 8$ since the weight of the shortest valley path (*i.e.*, p) is 8. As can be seen, for v_7, $\mathrm{nbr}^+(v_7) = \{v_8, v_9\}$ and $\mathrm{nbr}^-(v_7) = \{v_2, v_3, v_4, v_5\}$. The pair $(\langle v_5, v_7\rangle, \langle v_5, v_8\rangle)$ is a downward shortcut pair of $\langle v_7, v_8\rangle$ because $\pi(v_5) < \min\{\pi(v_7), \pi(v_8)\}$. No other downward shortcut pairs exist for $\langle v_7, v_8\rangle$. Hence, $\mathrm{scp}^-(\langle v_7, v_8\rangle) = \{(\langle v_5, v_7\rangle, \langle v_5, v_8\rangle)\}$. Regarding the upward shortcut pairs, it can be shown that $\mathrm{scp}^+(\langle v_7, v_8\rangle) = \{(\langle v_7, v_9\rangle, \langle v_8, v_9\rangle)\}$. □

Algorithm 1 shows CHIndexing [39, 48], which is the state of the art to construct $\mathrm{sc}(G)$. Following the ordering π, it performs iterative vertex contractions (lines 3-8). Although CHIndexing assumes π is given beforehand, π can be constructed on-the-fly. For example, the *minimum degree heuristic* [12] iteratively chooses the vertex u with the least number of uncontracted neighbors for contraction. Following [39], we use minimum degree heuristic.

The definition of CH above is a variant of the original one [26]. Under the definition of [26], a shortcut $\langle u, v\rangle$ is generated only if the distance between u and v cannot be preserved after vertex contraction. This way, the number of shortcuts generated can be greatly reduced. However, such a definition is hard to maintain under weight changes [27]. Indeed, under this definition, the presence of a shortcut is weight dependent, indicating that when some weights change, new shortcuts or existing shortcuts may need to be added or deleted, respectively. By contrast, the variant we use is amenable to weight changes in that changing weights only affects the weights of the shortcuts. The state-of-the-art algorithms DCH[39] and UE[48] for updating CH both use the variant above.

Query. Given a query (s, t), CH reports the distance $\mathrm{sd}(s, t)$ by performing a variant of bidirectional Dijkstra's algorithm on $\mathrm{sc}(G)$ from s and t. During the search, a shortcut is relaxed only if it leads to a vertex with a higher rank. When one search settles a vertex which has been settled by the other search, the tentative distance between s and t is updated. The algorithm terminates when the keys in the priority queues of the searches are not smaller than the tentative distance. Finally, the tentative distance is reported.

Example 2.2: Consider $\mathrm{sc}(G)$ in Figure 1b and a query (v_6, v_7). Starting from each of v_6 and v_7, a variant of Dijkstra's algorithm is conducted. Only the shortcuts $\langle v_7, v_8\rangle$ and $\langle v_7, v_9\rangle$ are relaxed in the search from v_7 since only them lead to vertices with higher ranks than v_7. Similarly, only $\langle v_6, v_8\rangle$ and $\langle v_6, v_9\rangle$ are relaxed in the other search. v_9 is the only vertex settled by both searches. Therefore, $\mathrm{sd}(v_6, v_7) = \phi(\langle v_6, v_9\rangle) + \phi(\langle v_7, v_9\rangle) = 6$. □

Incremental CH. Since we use minimum degree heuristic to generate π, which is weight independent, CHANGED comprises ΔG and the shortcuts whose weights change, when ΔG is applied to G.

Hierarchical 2-Hop Index (H2H**).** H2H exploits the fact that road networks are of bounded treewidth. It is based on CH.

Indexing. Suppose the shortcut graph $\mathrm{sc}(G)$ has been constructed. For each vertex u whose $\mathrm{nbr}^+(u)$ is not empty, we define $x(u)$ as the vertex in $\mathrm{nbr}^+(u)$ with the lowest rank; that is, $x(u) \in \mathrm{nbr}^+(u)$ and $\pi(x(u)) \leq \pi(v)$ for $\forall v \in \mathrm{nbr}^+(u)$. The vertex with the highest rank is the only vertex that $x(\cdot)$ is undefined, since it has no upward neighbors. By letting $x(u)$ be the parent of u, we obtain a tree decomposition $\mathcal{T}$ of G, whose root is the vertex with the highest rank. For each u, let $\mathrm{depth}(u)$ denote the depth of u in $\mathcal{T}$ such that the root has depth 1. $\mathcal{T}$ has the following nice properties [37]: (1) Consider two vertices s and t. Let a be the lowest common ancestor of s and t in $\mathcal{T}$. Then, all shortest paths between s and t pass through $X(a) = \{a\} \cup \mathrm{nbr}^+(a)$. (2) For each vertex u, the vertices in $\mathrm{nbr}^+(u)$ are also ancestors of u in $\mathcal{T}$.

H2H computes for each vertex u the following information.

- *Ancestor* array $\mathrm{anc}(u)$: For each ancestor a of u, $\mathrm{anc}(u)$ is defined such that $\mathrm{anc}(u)[\mathrm{depth}(a)] = a$. Particularly, for notational convenience, u is included in $\mathrm{anc}(u)$ such that $\mathrm{anc}(u)[\mathrm{depth}(u)] = u$. In other words, $(\mathrm{anc}(u)[1], \mathrm{anc}(u)[2], \ldots, \mathrm{anc}(u)[\mathrm{depth}(u)])$ is the path from the root to u in $\mathcal{T}$. Hereafter, for each ancestor a of u, the pair $\langle\langle u, a\rangle\rangle$ is called a *super-shortcut.*
- *Distance* array $\mathrm{dis}(u)$: $\mathrm{dis}(u)[i]$ stores the distance between u and its i-th ancestor; that is, $\mathrm{dis}(u)[i] = \mathrm{sd}(u, \mathrm{anc}(u)[i])$.
- *Position* array $\mathrm{pos}(u)$: Let $X(u) = \mathrm{nbr}^+(u) \cup \{u\} = \{v_1, \ldots, v_k\}$. $\mathrm{pos}(u)$ is defined such that $\mathrm{pos}(u)[i] = \mathrm{depth}(v_i)$ for $\forall 1 \leq i \leq k$.

Example 2.3: Figure 1c shows a tree decomposition $\mathcal{T}$ of G that is constructed on $\mathrm{sc}(G)$ (Figure 1b). For each of v_2, v_6 and v_8, we show its upward neighbors (dotted arcs), distance array and position array. Take v_2 as an example. Because $\mathrm{nbr}^+(v_2) = \{v_5, v_7\}$ and $\pi(v_5) < \pi(v_7)$, v_5 is the parent of v_2 in $\mathcal{T}$. By definition, $\mathrm{anc}(v_2) = (v_9, v_8, v_7, v_5, v_2)$. Accordingly, $\mathrm{dis}(v_2) = (5, 9, 1, 5, 0)$ because the distances from v_2 to v_9, v_8, v_7, v_5 and v_2 are 5, 9, 1, 5 and 0, respectively. We have $\mathrm{pos}(v_2) = \{3, 4, 5\}$ because $\mathrm{depth}(v_7)$, $\mathrm{depth}(v_5)$ and $\mathrm{depth}(v_2)$ are 3, 4 and 5, respectively. □

Query. Consider a query (s, t) with $s \neq t$. By properties (1) and (2) of $\mathcal{T}$ mentioned above, with a being the least common ancestor of

Notation	Description
$\mathsf{sc}(G)$	the shortcut graph of G
$\phi(\cdot)$	the weight of an edge or a shortcut
$\langle u, v\rangle$	a shortcut between u and v
$\mathsf{nbr}^+(u)$	the upward neighbors of u
$\mathsf{nbr}^-(u)$	the downward neighbors of u
$\mathsf{scp}^+(e)$	the upward shortcut pairs of shortcut e
$\mathsf{scp}^-(e)$	the downward shortcut pairs of shortcut e
$\mathcal{T}$	a tree decomposition of G
$\mathsf{depth}(u)$	the depth of u in $\mathcal{T}$
$\mathsf{anc}(u)$	the ancestor array of u
$\mathsf{dis}(u)$	the distance array of u
$\langle\langle u, a\rangle\rangle$	a super-shortcut where a is an ancestor of u in $\mathcal{T}$

Table 1: Frequently Used Notations

s and t, we have (1) $\mathsf{sd}(s,t) = \min_{v\in X(a)} \mathsf{sd}(s,v) + \mathsf{sd}(t,v)$, and (2) all the vertices $X(a)$ are in $\mathsf{anc}(\cdot)$ of s and t. As a result, we have $\mathsf{sd}(s,t) = \min_{i\in\mathsf{pos}(a)} \mathsf{dis}(s)[i] + \mathsf{dis}(t)[i]$.

Example 2.4: Consider a query (v_2, v_6). In $\mathcal{T}$ (Figure 1c), v_8 is the least common ancestor of v_2 and v_6. Therefore, $\mathsf{sd}(v_2, v_6) = \min_{i\in\mathsf{pos}(v_8)} \mathsf{dis}(v_2)[i] + \mathsf{dis}(v_6)[i] = \min\{5+2, 9+6\} = 7$. □

Incremental H2H. Given weight updates ΔG, the ancestor arrays and position arrays do not need to be updated since they are constructed in a weight-independent manner. Therefore, CHANGED consists of ΔG and the super-shortcuts whose $\mathsf{dis}(\cdot)$ values change.

We summarize the notations frequently used in Table 1.

Remark. CH and H2H are not appropriate for scale-free networks in light of the core-periphery structure inherent in such networks. Specifically, a scale-free network typically contains a core which is a dense subgraph of high treewidth. Hence, for scale-free networks, it is inevitable that the resulting indices have much more shortcuts and super-shortcuts generated, degrading the performance.

3 THEORETICAL RESULTS

In this section, we show the unboundedness of incremental CH and incremental H2H. Like all prior works on boundedness [21, 22, 24, 41], the results presented are *conditional* in that they prespecify a model of computation. In Section 3.1, we focus on the class of locally persistent algorithms, which is the standard model used to prove unboundedness in previous works. In Sections 3.2 and 3.3, we prove the unboundedness under two practical classes of algorithms, $\mathsf{INC}_{\mathsf{CH}}$ and $\mathsf{INC}_{\mathsf{H2H}}$, for CH and H2H, respectively.

3.1 Unboundedness under LP Algorithms

Locally Persistent (LP) Algorithms. In an LP algorithm, each vertex v and edge e of G is allowed to use (1) an arbitrary amount of status information, denoted by $\mathsf{status}(v)$ and $\mathsf{status}(e)$, respectively, and (2) pointers to adjacent vertices and edges in G. However, no global information is maintained between successive calls of the algorithm. Moreover, auxiliary pointers to non-adjacent vertices and edges are not allowed. An LP algorithm handles ΔG by starting from the vertices and edges contained in ΔG. It works in a deterministic manner; that is, both the choice of which pointer to follow next and the way to update $\mathsf{status}(\cdot)$ depend deterministically on the status of the vertices and edges visited so far.

The main theorem of this section is shown in the following.

Theorem 3.1: *Under the class of* LP *algorithms, the problems of maintaining* CH *and* H2H *under weight changes are unbounded even for single weight decrease and single weight increase.*

Proof: We show the key idea here and the complete proof can be found in [46]. Let $\mathcal{A}_\Delta$ be an LP algorithm and $\mathsf{trace}(G, \Delta G)$ be the edges and vertices that $\mathcal{A}_\Delta$ inspects when handling ΔG. We intentionally construct a graph G and two updates Δ_1 and Δ_2 such that, if $\mathcal{A}_\Delta$ is bounded, both $|\mathsf{trace}(G,\Delta_1)|$ and $|\mathsf{trace}(G,\Delta_2)|$ are $O(1)$, but $|\mathsf{trace}(G,\Delta_1)| + |\mathsf{trace}(G,\Delta_2)| = \Omega(\ell)$ for some non-constant integer ℓ, thereby leading to a contradiction. □

Limitation of LP Algorithms. The class of LP algorithms is not practical for incremental CH and H2H, although it has been the standard model of computation in the study of boundedness. Indeed, it disallows graph traversal via shortcuts or super-shortcuts, which are the most important components in CH and H2H. Therefore, the class fails to include many practically efficient algorithms for incremental CH and H2H. For this reason, we next consider computational models that are more realistic.

3.2 Unboundedness under $\mathsf{INC}_{\mathsf{CH}}$ Algorithms

Many efficient incremental algorithms are obtained by incrementalizing the static counterpart. Therefore, in this section, we focus on a more practical class of incremental algorithms, denoted $\mathsf{INC}_{\mathsf{CH}}$, for CH, which models (at least a part of) incremental algorithms obtainable by incrementalizing CHIndexing (Algorithm 1). The state-of-the-art algorithms DCH [39] and UE [48] are two instances of $\mathsf{INC}_{\mathsf{CH}}$. The class $\mathsf{INC}_{\mathsf{CH}}$ is defined as follows.

The Class of $\mathsf{INC}_{\mathsf{CH}}$ Algorithms. An $\mathsf{INC}_{\mathsf{CH}}$ algorithm works on the shortcut graph $\mathsf{sc}(G)$ of G. For each vertex v of $\mathsf{sc}(G)$, its status information $\mathsf{status}(v)$ contains only immutable information such as $\pi(v)$. For each shortcut e, $\mathsf{status}(e) = s(e, \mathsf{scp}^+(e), \mathsf{scp}^-(e))$, where s can be any function which takes the edge e (if e exists in G), the upward shortcut pairs $\mathsf{scp}^+(e)$ and the downward shortcut pairs $\mathsf{scp}^-(e)$ of e as input, and determines the output solely based on the weights of the input. Intuitively, $\mathsf{status}(e)$ keeps track of the information about the edges and shortcuts that e depends on and the information about the shortcuts that depend on e (see Equation (◇)). Each vertex and shortcut is allowed to use pointers to adjacent vertices and shortcuts in $\mathsf{sc}(G)$. As in LP algorithms, no global information is maintained between successive calls of the algorithms. $\mathsf{INC}_{\mathsf{CH}}$ algorithms behave deterministically according to the $\mathsf{status}(\cdot)$ of the vertices and shortcuts visited.

$\mathsf{INC}_{\mathsf{CH}}$ algorithms differ from LP algorithms in two fundamental aspects. (1) Unlike LP algorithms, $\mathsf{INC}_{\mathsf{CH}}$ allows traversal via shortcuts. (2) $\mathsf{INC}_{\mathsf{CH}}$ restricts the status information that an incremental algorithm can use. The restriction, however, does not indicate inefficiency. The two practical algorithms DCH and UE both belong to this class. For example, in DCH, for each shortcut e, $\mathsf{status}(e)$ contains (1) the number of terms in Equation (◇) that lead to $\phi(e)$ and (2) all the downward shortcut pairs $\mathsf{scp}^-(e)$.

The main theorem is as follows. It shows that the notion of boundedness fails to characterize the efficiency of DCH and UE.

Theorem 3.2: *Under the class of* $\mathsf{INC}_{\mathsf{CH}}$ *algorithms, the problem of maintaining* CH *under weight changes is unbounded even for single weight decrease and single weight increase.*

Proof: We show the key idea of our proof. The complete proof can be found in Appendix A of [46]. Let $\mathcal{A}_\Delta$ be an algorithm from $\mathsf{INC}_{\mathsf{CH}}$. We first construct a graph G and an update ΔG such that applying ΔG to G results in CHANGED of constant size. We next show that a correct algorithm $\mathcal{A}_\Delta$ needs to update the status information for $\Omega(\ell)$ shortcuts for some non-constant ℓ when processing ΔG. Hence, $\mathcal{A}_\Delta$ cannot be bounded. □

3.3 Unboundedness under $\mathsf{INC}_{\mathsf{H2H}}$ Algorithms

In this section, we focus on a class of incremental algorithms, denoted $\mathsf{INC}_{\mathsf{H2H}}$, for H2H, which models algorithms obtainable by incrementalizing the construction algorithm H2HIndexing [37] of H2H. The class $\mathsf{INC}_{\mathsf{H2H}}$ is defined as follows.

The Class of $\mathsf{INC}_{\mathsf{H2H}}$ Algorithms. Given G, H2HIndexing constructs H2H by capitalizing on the shortcut graph $\mathrm{sc}(G)$. In light of this, it is natural that an algorithm obtained by incrementalizing H2HIndexing maintains $\mathrm{sc}(G)$ in the course of computation. We thus define $\mathsf{INC}_{\mathsf{H2H}}$ to include all incremental algorithms of H2H that update $\mathrm{sc}(G)$ as a subtask. DTDHL [51] and our algorithm for incremental H2H belong to this class. Regarding $\mathsf{INC}_{\mathsf{H2H}}$, we have:

Theorem 3.3: *Under the class of* $\mathsf{INC}_{\mathsf{H2H}}$ *algorithms, the problem of maintaining* H2H *under weight changes is unbounded even for single weight decrease and single weight increase.*

Proof: We construct a graph G and an update ΔG to G such that applying ΔG to G results in $O(1)$ changes in $\mathrm{dis}(\cdot)$ but $\Omega(\ell)$ changes in $\mathrm{sc}(G)$. Therefore, any $\mathsf{INC}_{\mathsf{H2H}}$ algorithm cannot be bounded. The detailed proof can be found in Appendix A of [46]. □

Remark. The results proved in this section are *conditional* in that each of them assumes a specific model of computation. So far, we are still unable to classify the problems *unconditionally*. Specifically, we are neither able to prove unconditionally that the problems are unbounded nor able to propose bounded algorithms to show that they are bounded, despite our efforts. This is a common limitation of all existing studies on boundedness [8, 21, 22, 24, 41].

4 RELATIVE SUBBOUNDEDNESS

Although bounded incremental algorithms are unavailable at the moment, we introduce *relative subboundedness* and show that the incremental counterparts of CH and H2H are still efficiently solvable under this new characterization of efficiency; that is, they can still be solved more efficiently than recomputing from scratch.

4.1 Relative Subboundedness

Let Q be either the problem of constructing CH or the problem of constructing H2H for a graph G. Let $Q(G)$ denote the result. The incremental counterpart of Q is to update the result from $Q(G)$ to $Q(G \oplus \Delta G)$ when an update ΔG to G is given. Relative subboundedness is to characterize the efficiency of an incremental algorithm $\mathcal{A}_\Delta$ that updates $Q(\cdot)$ from $Q(G)$ to $Q(G \oplus \Delta G)$.

Let $D(G)$ and $D(G \oplus \Delta G)$ denote the data that every algorithm for Q needs to inspect when computing $Q(G)$ and $Q(G \oplus \Delta G)$, respectively. Intuitively, $D(\cdot)$ captures only the essential data for Q and thus is algorithm-invariant. We define AFF to be the difference of $D(G \oplus \Delta G)$ with $D(G)$, *i.e.*, the part of data *affected* by ΔG.

Let $\mathcal{A}$ be any algorithm for Q. For $\forall x \in \mathsf{AFF}$, we denote by $\mathsf{amt}(x)$ the amount of work done by $\mathcal{A}$ on x. That is, $\mathsf{amt}(x)$ represents the part of time spent on x by $\mathcal{A}$ when $\mathcal{A}$ is invoked to compute $Q(G \oplus \Delta G)$. We define $\|\mathsf{AFF}\| = \sum_{x \in \mathsf{AFF}} \mathsf{amt}(x)$ to denote the total time spent on AFF by $\mathcal{A}$. It is important to emphasize that $\|\mathsf{AFF}\|$ is defined with respect to $\mathcal{A}$. We illustrate these concepts using CH as an example in the following.

Example 4.1: For CH, $D(G)$ consists of G and the shortcut graph $\mathrm{sc}(G)$, since every algorithm needs to read G as input and compute $\mathrm{sc}(G)$ as output. Therefore, the corresponding AFF comprises (1) AFF_1: the edges in G with weights changed, *i.e.*, ΔG (changes in input) and (2) AFF_2: the shortcuts in $\mathrm{sc}(G)$ with weights changed after applying ΔG (changes in output). Hence, $\mathsf{AFF} = \mathsf{AFF}_1 \cup \mathsf{AFF}_2$ happens to be exactly CHANGED. We next compute $\|\mathsf{AFF}\| = \sum_{x \in \mathsf{AFF}} \mathsf{amt}(x)$ for $\mathcal{A}$ = CHIndexing. For $\forall e \in \mathsf{AFF}_1$, $\mathsf{amt}(e) = \Theta(1)$ because reading an edge takes constant time. For any shortcut $e \in \mathsf{AFF}_2$, $\mathsf{amt}(e) = \Theta(|\mathrm{scp}^-(e)| + |\mathrm{scp}^+(e)| + 1)$. Indeed, (1) for each downward shortcut pair $(e', e'') \in \mathrm{scp}^-(e)$ of e, $\phi(e)$ is inspected once (Equation ($\diamond$)) in CHIndexing; and (2) for each upward shortcut pair $(e', e'') \in \mathrm{scp}^+(e)$ of e, $\phi(e') + \phi(e)$ is evaluated once to update $\phi(e'')$, since (e, e') is a downward shortcut pair of e''. Hence, $\|\mathsf{AFF}\| = \Theta\left(|\mathsf{CHANGED}| + \sum_{e \in \mathsf{AFF}_2} |\mathrm{scp}^-(e)| + |\mathrm{scp}^+(e)|\right)$. This is the time spent by CHIndexing related to AFF. □

Let $\mathcal{T}_{\mathcal{A}}$ be the time taken by $\mathcal{A}$ to handle ΔG starting from scratch. That is, $\mathcal{T}_{\mathcal{A}}$ is the time cost by $\mathcal{A}$ to compute $Q(G \oplus \Delta G)$. By definition, we have $\|\mathsf{AFF}\| \leq \mathcal{T}_{\mathcal{A}}$. Moreover, it is likely that $\|\mathsf{AFF}\| \ll \mathcal{T}_{\mathcal{A}}$ when ΔG is of small size (*e.g.*, only one edge has weight changed), since smaller ΔG usually results in smaller affected area and thus smaller AFF. In view of these, we define *relative subboundedness* such that an incremental algorithm $\mathcal{A}_\Delta$ that updates the result from $Q(G)$ to $Q(G \oplus \Delta G)$ is said to be *subbounded relative to* $\mathcal{A}$ only if its cost to handle ΔG is of $O(\|\mathsf{AFF}\| \log \|\mathsf{AFF}\|)$. Here the factor $\log \|\mathsf{AFF}\|$ is to allow the use of auxiliary structures such as priority queues. Intuitively, the near linear dependence on $\|\mathsf{AFF}\|$ requires that a relatively subbounded algorithm focus on AFF, which is exactly the part affected by ΔG. Therefore, an unbounded but relatively subbounded incremental algorithm is still likely to be more efficient than recomputing from scratch by $\mathcal{A}$.

Note that the definition of relative subboundedness does not assume specific $\mathcal{A}$. Ideally, $\mathcal{A}_\Delta$ is desired to be subbounded relative to an *optimal* $\mathcal{A}$. However, that is usually technically infeasible, since for many problems, their optimality has not yet been studied and thus it cannot be determined whether a given $\mathcal{A}$ is optimal. Therefore, instead, we desire an $\mathcal{A}_\Delta$ that is subbounded relative to the state-of-the-art $\mathcal{A}$, which is the fastest algorithm in hand to compute from scratch. We stress that our proposed incremental algorithms are subbounded relative to the state-of-the-arts CHIndexing and H2HIndexing for CH and H2H, respectively.

We stress that (1) for $\mathcal{A}_\Delta$ and $\mathcal{A}'_\Delta$ which are respectively subbounded relative to $\mathcal{A}$ and $\mathcal{A}'$, they are not necessarily equally efficient though both relatively subbounded, since they are relative to different algorithms; and (2) though relative subboundedness is introduced in the context of CH and H2H, we believe that it can also be extended to analyze the efficiency of algorithms for other problems; we consider this our future work.

Difference with Relative Boundedness [21]. For a graph G and a graph problem Q, let $G_{(\mathcal{A},Q)}$ be the data inspected by $\mathcal{A}$ when computing $Q(G)$. For updates ΔG to G, let DIFF be the difference between $G_{(\mathcal{A},Q)}$ and $(G \oplus \Delta G)_{(\mathcal{A},Q)}$. An incremental algorithm $\mathcal{A}_\Delta$ for Q is said to be *bounded relative to* $\mathcal{A}$ if its cost can be expressed as a polynomial function in |DIFF|. Intuitively, |DIFF| captures the amount of work that every $\mathcal{A}_\Delta$ obtained by incrementalizing $\mathcal{A}$ needs to do. Therefore, relative boundedness guarantees the effectiveness of incrementalization. Relative subboundedness has two fundamental differences with relative boundedness: (1) for CHIndexing and H2HIndexing, we have $|\mathsf{DIFF}| \leq \|\mathsf{AFF}\|$; although greater, $\|\mathsf{AFF}\|$ is still upper bounded by $\mathcal{T}_\mathcal{A}$, the cost of $\mathcal{A}$; and (2) while relative boundedness requires only *polynomial* dependence on |DIFF|, relative subboundedness requires *linearithmic* dependence on $\|\mathsf{AFF}\|$ (*i.e.*, $\|\mathsf{AFF}\| \log \|\mathsf{AFF}\|$) so that even for large $\|\mathsf{AFF}\|$, $\|\mathsf{AFF}\| \log \|\mathsf{AFF}\| \leq \mathcal{T}_\mathcal{A}$ is still likely to hold, thereby warranting efficiency. Therefore, in general, neither of relative subboundedness and relative boundedness can dominate the other. Ideally, it is desired that an incremental algorithm be both relatively subbounded and relatively bounded. However, as will be shown for DCH (Section 4.2) and IncH2H (Section 5), relative subboundedness alone is enough to warrant efficiency.

Example 4.2: Continuing with Example 4.1, we use CHIndexing to illustrate DIFF and |DIFF|. DIFF consists of two parts: (1) obviously, the edges and shortcuts in CHANGED and (2) the upward shortcut pairs of each shortcut in CHANGED. Indeed, for any shortcut e in CHANGED and any upward shortcut pair (e', e'') of e, $\phi(e) + \phi(e')$ is evaluated to different values in CHIndexing(G) and CHIndexing($G \oplus \Delta G$). Hence, we have $|\mathsf{DIFF}| = \Theta(|\mathsf{CHANGED}| + \sum_{e \in \mathsf{AFF}_2} |\mathsf{scp}^+(e)|)$. We can find that $|\mathsf{DIFF}| \leq \|\mathsf{AFF}\|$. □

Difference with Local Computation [21]. By definition, an incremental problem is localizable only if the part of index that needs to be updated locates in the c-hop neighborhood of the update for some constant c. For CH and H2H, they do not possess such a locality property. Hence, localizability does not apply here.

Difference with Semiboundedness [24]. As shown in the related work (Section 1), *semiboundedness* is tailored for graph pattern matching and thus does not apply to CH and H2H.

4.2 DCH Revisited

DCH [39] comprises two parts: DCH^+ for weight increase and DCH^- for weight decrease. The corresponding $\|\mathsf{AFF}\|$ and |DIFF| have been derived in Example 4.1 and 4.2, respectively. We prove:

Theorem 4.1: DCH^+ *and* DCH^- *are subbounded and bounded relative to* CHIndexing*, respectively; specifically, they run in* $O(\|\mathsf{AFF}\| \log \|\mathsf{AFF}\|)$ *and* $O(|\mathsf{DIFF}| \log |\mathsf{DIFF}|)$ *time, respectively.*

```
1  $Q \leftarrow$ an empty priority queue;
2  for each $(e, w) \in \Delta G$ do // the new weight of $e$ is $w$
3      if $\phi(e, G) = \phi(e, \mathsf{sc}(G))$ then
4          $\mathsf{sup}(e) \leftarrow \mathsf{sup}(e) - 1$;
5          if $\mathsf{sup}(e) = 0$ then insert $e$ into $Q$;
6      $\phi(e, G) \leftarrow w$;
7  while $Q$ is not empty do
8      $e = \langle u, v \rangle \leftarrow$ the one in $Q$ with min $\pi(u)$; remove $e$ from $Q$;
9      for each upward shortcut pair $(e', e'') \in \mathsf{scp}^+(e)$ of $e$ do
10         if $\phi(e'') = \phi(e) + \phi(e')$ then
11             $\mathsf{sup}(e'') \leftarrow \mathsf{sup}(e'') - 1$;
12             if $\mathsf{sup}(e'') = 0$ then insert $e''$ into $Q$;
13     compute $\phi(e)$ and $\mathsf{sup}(e)$;
14 return $\mathsf{sc}(G)$;
```

Algorithm 2: DCH^+

Note that DCH^- is also bounded by $O(\|\mathsf{AFF}\| \log \|\mathsf{AFF}\|)$ because $|\mathsf{DIFF}| \leq \|\mathsf{AFF}\|$. As a result, DCH^- is both subbounded and bounded relative to CHIndexing. In contrast, DCH^+ is only subbounded but not bounded relative to CHIndexing. It remains open whether a relatively bounded algorithm for weight increase exists.

Relative Subboundedness of DCH^+. We start by introducing DCH^+. In DCH^+, each shortcut e is associated with $\mathsf{scp}^-(e)$ and an integer $\mathsf{sup}(e)$, called the support of e, which is equal to the number of terms in Equation (◇) that lead to $\phi(e)$. For example, in Figure 1b, $\mathsf{sup}(\langle v_5, v_7 \rangle) = 1$ since $(v_5, v_7) \notin G$ and $(\langle v_3, v_5 \rangle, \langle v_3, v_7 \rangle)$ is the only pair such that $\phi(\langle v_3, v_5 \rangle) + \phi(\langle v_3, v_7 \rangle) = \phi(\langle v_5, v_7 \rangle)$; similarly, we have $\mathsf{sup}(\langle v_3, v_5 \rangle) = 1$ and $\mathsf{sup}(\langle v_7, v_8 \rangle) = 1$.

DCH^+ is shown in Algorithm 2. It uses a priority queue Q and guarantees that only shortcuts whose weights need to be updated can be pushed to Q. The use of Q ensures that for the shortcut being handled, the weights of all shortcuts it depends on have been correctly updated. At lines 2-6, it deals with ΔG. Specifically, for an edge $e \in \Delta G$, before increasing $\phi(e, G)$ to w (line 6), DCH^+ tests whether the old $\phi(e, G)$ is equal to $\phi(e, \mathsf{sc}(G))$ (line 3). If so, it means that there will be one less term in Equation (◇) to support $\phi(e, \mathsf{sc}(G))$ after updating $\phi(e, G)$ and thus $\mathsf{sup}(e)$ is decreased by one (line 4). If $\mathsf{sup}(e)$ becomes 0, $\phi(e, \mathsf{sc}(G))$ necessarily increases and DCH^+ pushes it to Q for later processing (line 5). Lines 7-13 work in a similar manner. At line 13, the new $\phi(e, \mathsf{sc}(G))$ and $\mathsf{sup}(e)$ of shortcut e are computed from scratch using Equation (◇).

Example 4.3: Consider an update ΔG which increases the weight of (v_3, v_5) from 2 to 3 (Figure 1a). Because $\phi((v_3, v_5), G)$ is the only term supporting $\phi(\langle v_3, v_5 \rangle)$ in Equation (◇), $\mathsf{sup}(\langle v_3, v_5 \rangle)$ becomes 0 after the update. Hence, $\langle v_3, v_5 \rangle$ is inserted into Q (line 5). To illustrate lines 7-13, take as an example the iteration for $\langle v_3, v_5 \rangle$. In line 9, the only upward shortcut pair $(\langle v_3, v_7 \rangle, \langle v_5, v_7 \rangle)$ is inspected. It is found that $\phi(\langle v_5, v_7 \rangle) = \phi(\langle v_3, v_5 \rangle) + \phi(\langle v_3, v_7 \rangle)$. Therefore, $\mathsf{sup}(\langle v_5, v_7 \rangle)$ is decreased from 1 to 0 and $\langle v_5, v_7 \rangle$ is inserted into Q accordingly. At line 13, the new $\phi(\langle v_3, v_5 \rangle) = 3$ and $\mathsf{sup}(\langle v_3, v_5 \rangle) = 1$ are computed. □

Assuming the shortcuts can be inspected in $O(1)$ time using hashing, (1) lines 2-6 take $O(|\mathsf{AFF}_1| \cdot \log |\mathsf{CHANGED}|)$ time, in which the factor $\log |\mathsf{CHANGED}|$ is due to Q; and (2) since each

```
1  $Q \leftarrow$ an empty priority queue;
2  for each $(e, w) \in \Delta G$ do
3      $\phi(e, G) \leftarrow w$;
4      if $\phi(e, G) < \phi(e, \mathrm{sc}(G))$ then
5          $\phi(e, \mathrm{sc}(G)) \leftarrow \phi(e, G)$;
6          if $e \notin Q$ then insert $e$ into $Q$;
7  while $Q$ is not empty do
8      $e = \langle u, v\rangle \leftarrow$ the one in $Q$ with min $\pi(u)$; remove $e$ from $Q$;
9      for each upward shortcut pair $(e', e'') \in \mathrm{scp}^+(e)$ of $e$ do
10         if $\phi(e) + \phi(e') < \phi(e'')$ then
11             $\phi(e'') \leftarrow \phi(e) + \phi(e')$;
12             if $e'' \notin Q$ then insert $e''$ into $Q$;
13 return $\mathrm{sc}(G)$;
```

Algorithm 3: DCH$^-$

shortcut in AFF_2 is inserted into Q exactly once, lines 7-13 in total take $O\left(\left(\sum_{e\in\mathsf{AFF}_2}|\mathrm{scp}^+(e)| + |\mathrm{scp}^-(e)| + 1\right)\cdot\log|\mathsf{CHANGED}|\right)$ time. Hence, DCH$^+$ takes $O(\|\mathsf{AFF}\|\cdot\log|\mathsf{CHANGED}|) \in O(\|\mathsf{AFF}\|\cdot\log\|\mathsf{AFF}\|)$ time and thus is subbounded relative to CHIndexing.

Relative Boundedness of DCH$^-$**.** DCH$^-$ is shown in Algorithm 3. Like in DCH$^+$, only shortcuts in CHANGED can be added to priority queue Q. In lines 2-6, for each edge e involved in ΔG, it updates $\phi(e, G)$ (line 3). If the new $\phi(e, G)$ is smaller than $\phi(e, \mathrm{sc}(G))$, the weight of shortcut e changes. Hence, DCH$^-$ sets $\phi(e, \mathrm{sc}(G))$ to the new $\phi(e, G)$ and pushes e to Q. Lines 7-12 follow a similar idea.

It can be verified lines 2-6 take $O(|\mathsf{AFF}_1|\cdot\log|\mathsf{CHANGED}|)$ time and lines 7-12 take $O\left(\left(\sum_{e\in\mathsf{AFF}_2}|\mathrm{scp}^+(e)| + 1\right)\cdot\log|\mathsf{CHANGED}|\right)$ time. As a result, DCH$^-$ costs $O(|\mathsf{DIFF}|\cdot|\mathsf{CHANGED}|) \in O(|\mathsf{DIFF}|\cdot\log|\mathsf{DIFF}|)$ time and thus is bounded relative to CHIndexing.

4.3 UE Revisited

Independently of DCH, Wei et al. proposed UE [48] to maintain CH. UE can be considered as an unoptimized version of DCH. To see this, consider a shortcut $e \in$ CHANGED. Unlike DCH, for each upward shortcut pair (e', e'') of e, UE computes the weight of e'' from scratch no matter if the weight of e'' needs to be updated or not. Consequently, UE does more work than DCH. In particular, UE is neither bounded nor subbounded relative to CHIndexing.

5 INCREMENTAL H2H

Next, we propose a new incremental algorithm IncH2H with performance guarantee for H2H by incrementalizing the state-of-the-art construction algorithm H2HIndexing [37]. Such an algorithm is also a necessary routine to maintain indices that are built on H2H, *e.g.*, TEN [38] for the task of nearest neighbor search. IncH2H comprises two parts, *i.e.*, IncH2H$^+$ for weight increase and IncH2H$^-$ for weight decrease. The main theorem of this section is as follows.

Theorem 5.1: IncH2H$^+$ *and* IncH2H$^-$ *are subbounded and bounded relative to* H2HIndexing*, respectively; specifically, they run in* $O(\|\mathsf{AFF}\|\log\|\mathsf{AFF}\|)$ *and* $O(|\mathsf{DIFF}|\log|\mathsf{DIFF}|)$ *time, respectively, where* $\|\mathsf{AFF}\|$ *and* $|\mathsf{DIFF}|$ *are defined with respect to* H2HIndexing.

Analogous to DCH$^-$, because $|\mathsf{DIFF}| \leq \|\mathsf{AFF}\|$, IncH2H$^-$ can also be bounded by $O(\|\mathsf{AFF}\|\log\|\mathsf{AFF}\|)$. Therefore, IncH2H$^-$ is both subbounded and bounded relative to H2HIndexing. As for IncH2H$^+$, it is only relatively subbounded.

To start with, we revisit H2HIndexing for H2H, following which the corresponding AFF and DIFF are identified.

Algorithm H2HIndexing **Revisited.** To construct H2H for a graph G, H2HIndexing first calls CHIndexing to construct the shortcut graph $\mathrm{sc}(G)$ of G. After that, it constructs a tree decomposition $\mathcal{T}$ of G with the help of the ordering π and the upward neighbors $\mathrm{nbr}^+(u)$ of each vertex u (Section 2). Based on $\mathcal{T}$, three arrays are constructed for each vertex u, namely ancestor array $\mathrm{anc}(u)$, position array $\mathrm{pos}(u)$ and distance array $\mathrm{dis}(u)$. Given $\mathcal{T}$, both $\mathrm{anc}(u)$ and $\mathrm{pos}(u)$ are easy to construct. As for $\mathrm{dis}(u)$, it can be computed as follows. Let H_u be the subgraph of $\mathrm{sc}(G)$ induced by u and its ancestors. H_u is distance preserving in that the distance from u to any ancestor a in H_u is the same as that in G [37]. Therefore,

$$\mathrm{dis}(u)[\mathrm{depth}(a)] = \mathrm{sd}(u, a) = \min_{v\in\mathrm{nbr}^+(u)} \phi(\langle u, v\rangle) + \mathrm{sd}(v, a) \quad (\star)$$

where $\mathrm{depth}(a)$ is the depth of a in $\mathcal{T}$. Furthermore, because both v and a in Equation ($\star$) are ancestors of u, when $v \neq a$,

$$\mathrm{sd}(v, a) = \begin{cases} \mathrm{dis}(v)[\mathrm{depth}(a)], & \text{if } \mathrm{depth}(v) > \mathrm{depth}(a) \\ \mathrm{dis}(a)[\mathrm{depth}(v)], & \text{if } \mathrm{depth}(v) < \mathrm{depth}(a) \end{cases} \quad (\triangledown)$$

In other words, $\mathrm{dis}(u)$ can be computed by inspecting $\mathrm{dis}(\cdot)$ of vertices with higher ranks. Hence, H2HIndexing iteratively computes $\mathrm{dis}(u)$ for each u in the reverse order of π. Recall that for each ancestor a of u, the pair $\langle\langle u, a\rangle\rangle$ is called a super-shortcut.

Characterization of AFF **and** DIFF**.** For H2H, the corresponding $D(G)$ consists of graph G itself, the shortcut graph $\mathrm{sc}(G)$ and the three arrays (*i.e.*, $\mathrm{anc}(\cdot)$, $\mathrm{pos}(\cdot)$ and $\mathrm{dis}(\cdot)$). Here, $\mathrm{sc}(G)$ is included in $D(G)$ because the three arrays are defined over $\mathrm{sc}(G)$ and thus every algorithm needs to first construct $\mathrm{sc}(G)$ to guide the construction of H2H. Let $\mathsf{AFF}_{\mathsf{CH}}$ and $\mathsf{DIFF}_{\mathsf{CH}}$ be the corresponding AFF and DIFF for CH, respectively. We have:

$\underline{\mathsf{AFF}}$: Because (1) $\mathsf{AFF}_{\mathsf{CH}}$ has included the changes in both the graph and the shortcut graph and (2) $\mathrm{anc}(\cdot)$ and $\mathrm{pos}(\cdot)$ remain unchanged under ΔG, we have $\mathsf{AFF} = \mathsf{AFF}_{\mathsf{CH}} \cup \mathsf{AFF}_3$ where AFF_3 comprises the super-shortcuts whose $\mathrm{dis}(\cdot)$ values change after applying ΔG.

$\underline{\|\mathsf{AFF}\|}$: Let $\mathcal{A}$ = H2HIndexing. Consider $\langle\langle u, a\rangle\rangle \in \mathsf{AFF}_3$. By definition, $\mathrm{dis}(u)[\mathrm{depth}(a)]$ changes under ΔG. With $\mathrm{des}(u)$ denoting the descendants of u in $\mathcal{T}$, we have $\mathrm{amt}(\langle\langle u, a\rangle\rangle) = \Theta(|\mathrm{nbr}^+(u)| + |\mathrm{nbr}^-(u)| + |\mathrm{nbr}^-(a) \cap \mathrm{des}(u)|)$. Indeed, (1) $\mathrm{dis}(u)[\mathrm{depth}(a)]$ is evaluated using Equation ($\star$) by iterating over the upward neighbors $\mathrm{nbr}^+(u)$ of u; (2) for each downward neighbor $v \in \mathrm{nbr}^-(u)$, a is an ancestor of v; to compute $\mathrm{dis}(v)[\mathrm{depth}(a)]$, $\mathrm{dis}(u)[\mathrm{depth}(a)]$ is accessed exactly once since $u \in \mathrm{nbr}^+(v)$; and (3) for each $v \in \mathrm{nbr}^-(a) \cap \mathrm{des}(u)$ (*i.e.*, v is both a descendant of u and a downward neighbor of a), $\mathrm{dis}(u)[\mathrm{depth}(a)]$ is inspected once when evaluating $\mathrm{dis}(v)[\mathrm{depth}(u)]$. Having calculated the amount of work done on AFF_3, we next examine AFF_2 (Example 4.1), which comprises the shortcuts that change under ΔG. Besides the work counted in $\|\mathsf{AFF}_{\mathsf{CH}}\|$, a shortcut $\langle u, v\rangle \in \mathsf{AFF}_2$ is additionally inspected once in H2HIndexing to evaluate $\mathrm{dis}(u)[\mathrm{depth}(a)]$ (Equation ($\star$))

for each ancestor a of u. Therefore, we have

$$\|\mathsf{AFF}\| = \Theta(\|\mathsf{AFF_{CH}}\| + \|\mathsf{AFF_3}\| + \mathcal{K})$$

where $\|\mathsf{AFF_3}\| = \sum_{\langle\langle u,a\rangle\rangle \in \mathsf{AFF_3}} |\mathsf{nbr}^+(u)| + |\mathsf{nbr}^-(u)| + |\mathsf{nbr}^-(a) \cap \mathsf{des}(u)|$ and $\mathcal{K} = \sum_{\langle u,v\rangle \in \mathsf{AFF_2}} |\mathsf{anc}(u)|$. Here, $|\mathsf{anc}(u)|$ is the length of the ancestor array $\mathsf{anc}(u)$ of u.

DIFF: By definition, DIFF contains CHANGED and $\mathsf{DIFF_{CH}}$ obviously. In addition, it contains all such pairs $(\langle u,v\rangle, (v,a))$, where v is an upward neighbor of u and a is an ancestor of u, that the term $\phi(\langle u,v\rangle) + \mathsf{sd}(v,a)$ in Equation ($\star$) is evaluated to different values in $\mathsf{H2HIndexing}(G)$ and $\mathsf{H2HIndexing}(G \oplus \Delta G)$.

|DIFF|: With DIFF, it can be shown that $|\mathsf{DIFF}| = \Theta(|\mathsf{DIFF_{CH}}| + |\mathsf{CHANGED}| + \mathcal{K}' + \mathcal{K}'')$, where $\mathcal{K}' = \sum_{\langle u,v\rangle \in \mathsf{AFF_2}} |\mathsf{anc}(u)|$ and $\mathcal{K}'' = \sum_{\langle\langle u,a\rangle\rangle \in \mathsf{AFF_3}} |\mathsf{nbr}^-(u)| + |\mathsf{nbr}^-(a) \cap \mathsf{des}(u)|$. This is because (1) for each $\langle u,v\rangle \in \mathsf{AFF_2}$, the value $\phi(\langle u,v\rangle) + \mathsf{sd}(v,a)$ changes for each ancestor a of u; and (2) for each $\langle\langle u,a\rangle\rangle \in \mathsf{AFF_3}$, the distance between u and a changes; therefore, the value $\phi(\langle v,u\rangle) + \mathsf{sd}(u,a)$ changes for each $v \in \mathsf{nbr}^-(u)$ and the value $\phi(\langle v,a\rangle) + \mathsf{sd}(u,a)$ changes for each $v \in \mathsf{nbr}^-(a) \cap \mathsf{des}(u)$.

Algorithm Overview. We give an overview of IncH2H before we get into the details. According to Equations ($\star$) and ($\triangledown$), a super-shortcut $\langle\langle u,a\rangle\rangle$ has $\mathsf{dis}(u)[\mathsf{depth}(a)]$ changed only if $\phi(\langle u,v\rangle) + \mathsf{sd}(v,a)$ changes for some $v \in \mathsf{nbr}^+(u)$. Specifically, (1) if $\phi(\langle u,v\rangle)$ changes, $\mathsf{dis}(u)[\mathsf{depth}(a)]$ for each ancestor a of u may change; (2) if $\mathsf{dis}(v)[\mathsf{depth}(a)]$ changes, both $\mathsf{dis}(u)[\mathsf{depth}(a)]$ for $u \in \mathsf{nbr}^-(v)$ and $\mathsf{dis}(u)[\mathsf{depth}(v)]$ for $u \in \mathsf{nbr}^-(a) \cap \mathsf{des}(v)$ may be affected. In view of these, starting from the shortcuts and super-shortcuts whose weights are known to have changed, IncH2H further identifies super-shortcuts that may change. To make this process efficient, IncH2H employs several auxiliary structures.

Auxiliary Structures. IncH2H uses the following auxiliary structures. (1) For each vertex u, let $u.d$ and $u.f$ be the *discovery* time and *finishing* time of u in a depth-first search from the root of $\mathcal{T}$, respectively [16]. Note that given two vertices u and v, u is an ancestor of v if and only if $u.d < v.d$ and $v.f < u.f$. (2) For each vertex u, the downward neighbors $\mathsf{nbr}^-(u)$ are kept in an array and sorted in ascending order of discovery time. (3) For each super-shortcut $\langle\langle u,a\rangle\rangle$, we store $\mathsf{first}(\langle\langle u,a\rangle\rangle)$, which is the smallest index i such that the i-th downward neighbor in $\mathsf{nbr}^-(a)$ has greater discovery time than u. (4) For each super-shortcut $\langle\langle u,a\rangle\rangle$, let the support of $\langle\langle u,a\rangle\rangle$, denoted $\mathsf{sup}(\langle\langle u,a\rangle\rangle)$, be the number of terms in Equation ($\star$) that lead to $\mathsf{dis}(u)[\mathsf{depth}(a)]$; in other words, $\mathsf{sup}(\langle\langle u,a\rangle\rangle)$ is the number of upward neighbors v of u such that $\mathsf{dis}(u)[\mathsf{depth}(a)] = \phi(\langle u,v\rangle) + \mathsf{sd}(v,a)$. All these can be obtained by adapting H2HIndexing without affecting its complexity.

Example 5.1: Consider $\mathsf{sc}(G)$ and the tree decomposition $\mathcal{T}$ in Figures 1b and 1c. In Figure 1c, the discovery time and finishing time $[v_i.d, v_i.f]$ are shown next to each vertex v_i. The downward neighbors $\mathsf{nbr}^-(v_9)$ of v_9, when sorted in ascending order of discovery time, are (v_8, v_6, v_7, v_4). Therefore, for the super-shortcut $\langle\langle v_6, v_9\rangle\rangle$, we have $\mathsf{first}(\langle\langle v_6, v_9\rangle\rangle) = 3$ because the 3rd downward neighbor (*i.e.*, v_7) in $\mathsf{nbr}^-(v_9)$ is the first that has greater discovery time than v_6. We have $\mathsf{sup}(\langle\langle v_6, v_9\rangle\rangle) = 1$ because among the upward neighbors of v_6, only v_9 satisfies $\phi(\langle v_6, v_9\rangle) + \mathsf{sd}(v_9, v_9) = 2 = \mathsf{sd}(v_6, v_9)$, while for v_8, $\phi(\langle v_6, v_8\rangle) + \mathsf{sd}(v_8, v_9) = 7 + 4 = 11$. □

```
1  $Q \leftarrow$ an empty priority queue;
2  call $\mathsf{DCH}^+$ and denote by $C$ the shortcuts that are updated;
3  for each shortcut $\langle u,v\rangle \in C$ with $\pi(u) < \pi(v)$ do
4      for each ancestor $a$ of $u$ do
5          $\mathsf{tmp} \leftarrow w$;      // the original weight of $\langle u,v\rangle$
6          if $\mathsf{depth}(v) > \mathsf{depth}(a)$ then // $a$ is ancestor of $v$
7              $\mathsf{tmp}$ += $\mathsf{dis}(v)[\mathsf{depth}(a)]$;
8          else if $\mathsf{depth}(v) < \mathsf{depth}(a)$ then // $v$ is ancestor
9              $\mathsf{tmp}$ += $\mathsf{dis}(a)[\mathsf{depth}(v)]$;
10         if $\mathsf{tmp} = \mathsf{dis}(u)[\mathsf{depth}(a)]$ then
11             $\mathsf{sup}(\langle\langle u,a\rangle\rangle) \leftarrow \mathsf{sup}(\langle\langle u,a\rangle\rangle) - 1$;
12             if $\mathsf{sup}(\langle\langle u,a\rangle\rangle) = 0$ then insert $\langle\langle u,a\rangle\rangle$ into $Q$;
13 while $Q$ is not empty do
14     $\langle\langle u,a\rangle\rangle \leftarrow$ the one in $Q$ with max $\pi(u)$; remove it from $Q$;
15     for each downward neighbor $v \in \mathsf{nbr}^-(u)$ of $u$ do
16         if $\phi(\langle v,u\rangle) + \mathsf{dis}(u)[\mathsf{depth}(a)] = \mathsf{dis}(v)[\mathsf{depth}(a)]$ then
17             $\mathsf{sup}(\langle\langle v,a\rangle\rangle) \leftarrow \mathsf{sup}(\langle\langle v,a\rangle\rangle) - 1$;
18             if $\mathsf{sup}(\langle\langle v,a\rangle\rangle) = 0$ then insert $\langle\langle v,a\rangle\rangle$ into $Q$;
19     for each $v \in \mathsf{nbr}^-(a) \cap \mathsf{des}(u)$ do
20         if $\phi(\langle v,a\rangle) + \mathsf{dis}(u)[\mathsf{depth}(a)] = \mathsf{dis}(v)[\mathsf{depth}(u)]$ then
21             $\mathsf{sup}(\langle\langle v,u\rangle\rangle) \leftarrow \mathsf{sup}(\langle\langle v,u\rangle\rangle) - 1$;
22             if $\mathsf{sup}(\langle\langle v,u\rangle\rangle) = 0$ then insert $\langle\langle v,u\rangle\rangle$ into $Q$;
23     compute new $\mathsf{dis}(u)[\mathsf{depth}(a)]$ and $\mathsf{sup}(\langle\langle u,a\rangle\rangle)$;
24 return $\mathsf{dis}(\cdot)$;
```

Algorithm 4: $\mathsf{IncH2H}^+$

5.1 $\mathsf{IncH2H}^+$ for Weight Increase

We show $\mathsf{IncH2H}^+$ in Algorithm 4. $\mathsf{IncH2H}^+$ maintains a priority queue Q (line 1) such that only super-shortcuts in CHANGED can be inserted into Q. At line 2, DCH^+ is called to update $\mathsf{sc}(G)$, with all shortcuts changed kept in C. At lines 3-12, in light of Equation ($\star$), for each shortcut $\langle u,v\rangle \in C$, all super-shortcuts $\langle\langle u,a\rangle\rangle$ that may be affected by the change of $\phi(\langle u,v\rangle)$ are inspected. Specifically, for each $\langle\langle u,a\rangle\rangle$, a value $\mathsf{tmp} = w + \mathsf{sd}(v,a)$ is obtained using the original weight w of $\langle u,v\rangle$ and the original distance between v and a. If $\mathsf{tmp} = \mathsf{dis}(u)[\mathsf{depth}(a)]$, it means there is one term less in Equation ($\star$) to support the value $\mathsf{dis}(u)[\mathsf{depth}(a)]$ after the weight of $\langle u,v\rangle$ changes. Therefore, $\mathsf{sup}(\langle\langle u,a\rangle\rangle)$ is decreased by 1 at line 11. If $\mathsf{sup}(\langle\langle u,a\rangle\rangle)$ becomes zero, $\mathsf{dis}(u)[\mathsf{depth}(a)]$ necessarily changes, thus $\langle\langle u,a\rangle\rangle$ is inserted into Q for later processing (line 12). At lines 13-23, super-shortcuts $\langle\langle u,a\rangle\rangle$ that change are processed in non-ascending order of $\pi(u)$. This way, for each $\langle\langle u,a\rangle\rangle$ being processed, all super-shortcuts that $\langle\langle u,a\rangle\rangle$ depends on in Equation ($\star$) have been correctly updated. Therefore, at line 23, we can correctly compute the new $\mathsf{dis}(u)[\mathsf{depth}(a)]$ and $\mathsf{sup}(\langle\langle u,a\rangle\rangle)$ from scratch using Equation ($\star$). Before that, at lines 15-22, all super-shortcuts that depend upon $\langle\langle u,a\rangle\rangle$ are identified and processed accordingly in a manner similar to lines 3-12. It is worth mentioning that, at line 19, $\mathsf{nbr}^-(a) \cap \mathsf{des}(u)$ can be obtained in $O(|\mathsf{nbr}^-(a) \cap \mathsf{des}(u)|)$ time by using $\mathsf{nbr}^-(a)$ and $\mathsf{first}(\langle\langle u,a\rangle\rangle)$. Indeed, by construction, the vertices in $\mathsf{nbr}^-(a) \cap \mathsf{des}(u)$ reside exactly in the range $[\mathsf{first}(\langle\langle u,a\rangle\rangle), \mathsf{last}]$ of

$\mathsf{nbr}^-(a)$, where last is the largest index such that $v = \mathsf{nbr}^-(a)[\mathsf{last}]$ has finish time earlier than u, *i.e.*, $v.f < u.f$.

Example 5.2: Continuing with Example 5.1, consider increasing the weight of the edge (v_6, v_9) from 2 to 3 (Figure 1a). As a result, $\langle v_6, v_9\rangle$ is the only shortcut whose weight changes (Figure 1b) and thus $C = \{\langle v_6, v_9\rangle\}$. In lines 4-12, for ancestors v_8 and v_9 of v_6, it is found that both $\mathsf{sup}(\langle\!\langle v_6, v_8\rangle\!\rangle)$ and $\mathsf{sup}(\langle\!\langle v_6, v_9\rangle\!\rangle)$ decrease to 0 after the change of $\phi(\langle v_6, v_9\rangle)$. Hence, super-shortcuts $\langle\!\langle v_6, v_8\rangle\!\rangle$ and $\langle\!\langle v_6, v_9\rangle\!\rangle$ are inserted into Q for later processing (line 12). We demonstrate lines 15-22 using $\langle\!\langle v_6, v_9\rangle\!\rangle$. v_1 is inspected in line 15, since the weight of the super-shortcut $\langle\!\langle v_1, v_9\rangle\!\rangle$ may be affected by super-shortcut $\langle\!\langle v_6, v_9\rangle\!\rangle$. With the old $\mathsf{sd}(v_6, v_9) = 2$, $\phi(\langle v_1, v_6\rangle) + \mathsf{sd}(v_6, v_9) = \mathsf{dis}(v_1)[\mathsf{depth}(v_9)]$. Thus, $\mathsf{sup}(\langle\!\langle v_1, v_9\rangle\!\rangle)$ is decreased, becoming 0. Hence, $\langle\!\langle v_1, v_9\rangle\!\rangle$ is pushed to Q. No vertices are inspected in line 19 because $\mathsf{nbr}^-(v_9) \cap \mathsf{des}(v_6)$ is empty. In line 23, the new value of $\langle\!\langle v_6, v_9\rangle\!\rangle$ (*i.e.*, $\mathsf{dis}(v_6)[\mathsf{depth}(v_9)]$) is evaluated. □

Correctness. We prove by contradiction. Assume that IncH2H$^+$ fails to handle ΔG. Let W be the super-shortcuts that are not correctly updated and let $\langle\!\langle u, a\rangle\!\rangle \in W$ be the one with the maximum $\pi(u)$, breaking ties arbitrarily. By construction, all shortcuts and super-shortcuts that $\langle\!\langle u, a\rangle\!\rangle$ depends on in Equation (★) are correctly updated. In view of this, we conclude that $\langle\!\langle u, a\rangle\!\rangle$ is not inserted into Q since otherwise, $\mathsf{dis}(u)[\mathsf{depth}(a)]$ has been updated correctly at line 23. Let $v_1, v_2, \ldots, v_k \in \mathsf{nbr}^+(u)$ be all the upward neighbors of u such that $\mathsf{dis}(u)[\mathsf{depth}(a)] = \phi(\langle u, v_i\rangle) + \mathsf{sd}(v_i, a)$ for $\forall i \in [1, k]$ before applying ΔG. For convenience, we assume $\pi(v_i) < \pi(a)$. Since $\mathsf{dis}(u)[\mathsf{depth}(a)]$ increases after applying ΔG, the values of $\phi(\langle u, v_i\rangle) + \mathsf{dis}(v_i)[\mathsf{depth}(a)]$ for $\forall i \in [1, k]$ also increase. Without loss of generality, we assume that $\phi(\langle u, v_k\rangle) + \mathsf{dis}(v_k)[\mathsf{depth}(a)]$ is the last that is updated by IncH2H$^+$ and that it is caused by the update of $\mathsf{dis}(v_k)[\mathsf{depth}(a)]$. Then, when processing $\langle\!\langle v_k, a\rangle\!\rangle$, $\langle\!\langle u, a\rangle\!\rangle$ is inspected with its support decreased to 0, resulting in $\langle\!\langle u, a\rangle\!\rangle$ inserted into Q. A contradiction.

Relative Subboundedness of IncH2H$^+$**.** With $\tilde{O}(\psi)$ as a shorthand for $O(\psi \log \psi)$, (1) line 2 takes $\tilde{O}(\|\mathsf{AFF}_{\mathsf{CH}}\|)$ time; (2) lines 3-12 take $\tilde{O}(\sum_{\langle u,v\rangle \in \mathsf{AFF}_2} |\mathsf{anc}(u)|)$ time since C is exactly AFF_2; and (3) lines 13-23 in total take $\tilde{O}(\|\mathsf{AFF}_3\|)$ time since only super-shortcuts in AFF_3 can be inserted into Q. Therefore, IncH2H$^+$ takes $\tilde{O}(\|\mathsf{AFF}\|)$ time and thus is subbounded relative to H2HIndexing.

5.2 IncH2H$^-$ for Weight Decrease

We show IncH2H$^-$ in Algorithm 5. Like IncH2H$^+$, IncH2H$^-$ maintains a priority queue Q (line 1) such that only super-shortcuts in CHANGED can be added to Q. At line 2, DCH$^-$ is called to update $\mathsf{sc}(G)$ and all shortcuts changed are kept in C. At lines 3-12, in light of Equation (★), for each shortcut $\langle u, v\rangle \in C$, all super-shortcuts $\langle\!\langle u, a\rangle\!\rangle$ that may be affected by the change of $\phi(\langle u, v\rangle)$ are inspected. Specifically, for each ancestor a of u, a new value tmp of $\langle\!\langle u, a\rangle\!\rangle$ is obtained using the latest $\phi(\langle u, v\rangle)$. If tmp $< \mathsf{dis}(u)[\mathsf{depth}(a)]$, $\mathsf{dis}(u)[\mathsf{depth}(a)]$ is updated (line 11). Accordingly, $\langle\!\langle u, a\rangle\!\rangle$ is inserted into Q for later processing (line 12). At lines 13-22, as in IncH2H$^+$, super-shortcuts $\langle\!\langle u, a\rangle\!\rangle$ are processed in non-ascending order of $\pi(u)$. This way, it is guaranteed that, for each $\langle\!\langle u, a\rangle\!\rangle$ being processed, its value $\mathsf{dis}(u)[\mathsf{depth}(a)]$ has been updated to its final value.

```
1  $Q \leftarrow$ an empty priority queue;
2  call DCH$^-$ and denote by $C$ the shortcuts that are updated;
3  **for each** *shortcut* $\langle u, v\rangle \in C$ *with* $\pi(u) < \pi(v)$ **do**
4      **for each** *ancestor* $a$ *of* $u$ **do**
5          tmp $\leftarrow \phi(\langle u, v\rangle)$;
6          **if** depth($v$) > depth($a$) **then** // $a$ is ancestor of $v$
7              tmp += dis($v$)[depth($a$)];
8          **else if** depth($v$) < depth($a$) **then** // $v$ is ancestor
9              tmp += dis($a$)[depth($v$)];
10         **if** tmp < dis($u$)[depth($a$)] **then**
11             dis($u$)[depth($a$)] $\leftarrow$ tmp;
12             **if** $\langle\!\langle u, a\rangle\!\rangle \notin Q$ **then** push $\langle\!\langle u, a\rangle\!\rangle$ to $Q$;
13 **while** $Q$ *is not empty* **do**
14     $\langle\!\langle u, a\rangle\!\rangle \leftarrow$ the one in $Q$ with max $\pi(u)$; remove it from $Q$;
15     **for each** *downward neighbor* $v \in \mathsf{nbr}^-(u)$ *of* $u$ **do**
16         **if** $\phi(\langle v, u\rangle)$ + dis($u$)[depth($a$)] < dis($v$)[depth($a$)] **then**
17             dis($v$)[depth($a$)] $\leftarrow$ dis($u$)[depth($a$)] + $\phi(\langle v, u\rangle)$;
18             **if** $\langle\!\langle v, a\rangle\!\rangle \notin Q$ **then** push $\langle\!\langle v, a\rangle\!\rangle$ to $Q$;
19     **for each** $v \in \mathsf{nbr}^-(a) \cap \mathsf{des}(u)$ **do**
20         **if** $\phi(\langle v, a\rangle)$ + dis($u$)[depth($a$)] < dis($v$)[depth($u$)] **then**
21             dis($v$)[depth($u$)] $\leftarrow$ dis($u$)[depth($a$)] + $\phi(\langle v, a\rangle)$;
22             **if** $\langle\!\langle v, u\rangle\!\rangle \notin Q$ **then** push $\langle\!\langle v, u\rangle\!\rangle$ to $Q$;
23 **return** dis($\cdot$);
```

Algorithm 5: IncH2H$^-$

Due to the change of $\mathsf{dis}(u)[\mathsf{depth}(a)]$, at lines 15-22, all super-shortcuts that may be affected by $\langle\!\langle u, a\rangle\!\rangle$ are identified and updated accordingly. Though not presented in Algorithm 5, IncH2H$^-$ additionally needs to maintain $\mathsf{sup}(\cdot)$ of each super-shortcut so that later calls of IncH2H$^+$ can work correctly. We stress that this can be done on-the-fly without affecting the complexity. The correctness of IncH2H$^-$ can be proved similarly to how IncH2H$^+$ is proved.

Relative Boundedness of IncH2H$^-$**.** Let $\tilde{O}(\psi)$ denote $O(\psi \log \psi)$. We show that IncH2H$^-$ runs in $\tilde{O}(|\mathsf{DIFF}|)$ time. Specifically, (1) line 2 takes $\tilde{O}(|\mathsf{DIFF}_{\mathsf{CH}}|)$ time; (2) since C is exactly AFF_2, lines 3-12 take $\tilde{O}(\sum_{\langle u,v\rangle \in \mathsf{AFF}_2} |\mathsf{anc}(u)|)$ time; and (3) because only AFF_3 can be inserted into Q, lines 13-22 take $\tilde{O}(\sum_{\langle\!\langle u,a\rangle\!\rangle \in \mathsf{AFF}_3} |\mathsf{nbr}^-(u)| + |\mathsf{nbr}^-(a) \cap \mathsf{des}(u)| + 1)$ time. Therefore, we conclude that IncH2H$^-$ is bounded relative to H2HIndexing.

5.3 Parallelizing IncH2H

Recall that in IncH2H, the super-shortcuts that change are processed in non-ascending order of $\pi(u)$ so that when processing a super-shortcut $\langle\!\langle u, a\rangle\!\rangle$, all the super-shortcuts that $\langle\!\langle u, a\rangle\!\rangle$ relies on, *i.e.*, those contributing to its weight in Equation (★), have been correctly updated. For each such super-shortcut $\langle\!\langle u', a'\rangle\!\rangle$, we have $\mathsf{depth}(u') < \mathsf{depth}(u)$. Therefore, without affecting the correctness of IncH2H, the super-shortcuts in CHANGED can be processed in non-descending order of $\mathsf{depth}(u)$. Let $h = \max_u \mathsf{depth}(u)$ be the maximum depth. For $\forall 1 \leq i \leq h$, let $\mathsf{CHANGED}_i$ be the super-shortcuts $\langle\!\langle u, a\rangle\!\rangle \in \mathsf{CHANGED}$ with $\mathsf{depth}(u) = i$. The super-shortcuts in $\mathsf{CHANGED}_i$ are processed in parallel such that for two super-shortcuts $\langle\!\langle u, a\rangle\!\rangle$ and $\langle\!\langle u', a'\rangle\!\rangle$, they are assigned to the

Name	Description	$\lvert V\rvert$	$\lvert E\rvert$	# of SCs	# of SSCs
NY	New York City	0.26 M	0.37 M	1.22 M	99.48 M
COL	Colorado	0.43 M	0.53 M	1.23 M	166.30 M
FLA	Florida	1.07 M	1.35 M	3.13 M	282.93 M
CAL	California and Nevada	1.89 M	2.33 M	5.76 M	1.02 B
EUS	Eastern US	3.60 M	4.39 M	11.00 M	2.63 B
WUS	Western US	6.26 M	7.63 M	18.49 M	4.60 B
CUS	Central US	14.08 M	17.15 M	46.32 M	23.25 B
US	Full US	23.95 M	29.17 M	76.87 M	40.22 B
ENG	England	2.35 M	2.71 M	6.07 M	1.71 B

Table 2: Datasets (M = 10^6 and B = 10^9)

same processor if $u = u'$. This way, it is guaranteed that the super-shortcuts inspected by the processors are independent of each other. More details can be found in Appendix C of [46].

5.4 DTDHL Revisited

For completeness, we discuss DTDHL [51], which also is an incremental algorithm to maintain H2H. DTDHL is neither subbounded nor bounded relative to H2HIndexing. Indeed, (1) DTDHL^+ (for weight increase) needs to inspect all members in $\mathsf{nbr}^-(u) \cup \mathsf{nbr}^-(a)$ for each super-shortcut $\langle\langle u, a\rangle\rangle \in \mathsf{CHANGED}$ to identify the super-shortcuts affected; and (2) DTDHL^- (for weight decrease) may recalculate $\mathsf{dis}(u)[\mathsf{depth}(a)]$ even for some $\langle\langle u, a\rangle\rangle \notin \mathsf{CHANGED}$.

6 EXPERIMENTAL STUDY

We conducted experiments to verify the effectiveness of relative subboundedness. The experiments were carried out on a Linux machine with Intel Xeon E5-2697 CPU and 500 GB main memory.

Datasets. We used nine publicly available road networks, where the ENG network was downloaded from Geofabrik[1] and the rest were downloaded from DIMACS[2]. The statistics of the networks are shown in Table 2, where the columns "# of SCs" and "# of SSCs" show the # of shortcuts and super-shortcuts in CH and H2H, respectively. In each network, the weights represent the estimated transit time of the edges. We treated all networks as undirected.

Algorithms. We implemented and evaluated the following algorithms in our experiments: (1) DCH [39], (2) UE [48], (3) IncH2H, (4) DTDHL [51], (5) ParIncH2H (Section 5.3), (6) CHIndexing [39, 48] and (7) H2HIndexing [37]. The code of DCH and that of DTDHL were kindly provided by the authors of [39] and [51], respectively. We excluded H2HAdapt [48] and DynH2H [13] due to the reasons mentioned in the related work (Section 1). All algorithms were implemented in C++ and compiled by g++ at -O3 optimization level. ParIncH2H used OpenMP for parallelism.

6.1 Effectiveness of Relative Subboundedness

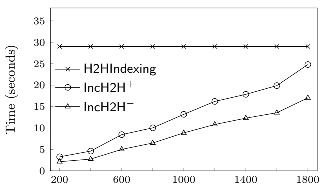
Exp-1: Efficiency of IncH2H. In this experiment, we compared IncH2H and H2HIndexing with varied $|\Delta G|$. To generate ΔG, we randomly sampled 200 to 1,800 edges from each road network. In practice, initially, the weight of an edge is usually set to an estimated value (*e.g.*, the median transit time of historical data) and does not need to be updated unless it changes significantly (*e.g.*, from normal condition to congestion). Therefore, for each edge e sampled, we increased its weight to $2.0 \times \phi(e)$. For each ΔG, we recorded the time taken by $\mathsf{IncH2H}^+$ to apply ΔG to G and the time taken by $\mathsf{IncH2H}^-$ to restore the weights of the edges to their original, *i.e.*, from $2.0 \times \phi(e)$ to $\phi(e)$. For H2HIndexing, we only recorded the time to compute the weights of the shortcuts and super-shortcuts from scratch, since the other parts of H2H remain unchanged given ΔG. The results for ENG, CAL, CUS and US are shown in Figures 2a, 2b, 2c and 2d, respectively. As can be observed, (1) $\mathsf{IncH2H}^-$ is consistently more efficient than $\mathsf{IncH2H}^+$ because $\mathsf{IncH2H}^-$ is both relatively subbounded and relatively bounded while $\mathsf{IncH2H}^+$ is relatively subbounded only; and (2) when $|\Delta G|$ reaches 1,600, $\mathsf{IncH2H}^+$ outperforms H2HIndexing only marginally. We emphasize that this does not imply IncH2H is inefficient. To see this, we show in Figure 2e for each ΔG the ratio of the number of super-shortcuts whose $\mathsf{dis}(\cdot)$ values change to the total number of super-shortcuts in H2H (see the column "# of SSCs" of Table 2). As can be seen, H2H is sensitive to changes. Indeed, a ΔG whose size is of hundreds can result in a non-trivial proportion of the index being affected. For example, even when $|\Delta G|$ is only $1,600$ (CUS) and $1,600$ (US), 10.2% and 9.7% of the super-shortcuts are affected by ΔG and need to be updated, respectively. An update of larger size will make the situation worse. Note that it is an issue caused by the nature of H2H rather than the algorithm. In view of this, *no* incremental algorithm for H2H is likely to outperform recomputing from scratch when the size of the updates becomes large, since the amount of data to be updated becomes too huge. Our IncH2H is efficient in the sense that it is able to outperform recomputing from scratch even if more than 10% of the index needs to be updated, indicating the effectiveness of relative subboundedness.

In addition, we observe that IncH2H performs better on networks of medium size (ENG and CAL) than those of large size (CUS and US) in terms of $|\Delta G|/|G|$. Consider ENG. Figure 2a suggests that IncH2H can outperform recomputing from scratch even if there come 200 to 1,800 weight updates per minute. This is sufficient most of the time in practice. To see this, below we show our analysis of a copy of historical traffic data of England, which was kindly provided by the authors of [48]. The data contain information about 600 major highways over the course of March, 2015. For each road e, we used the 10th percentile of its historical transit times as its reference weight $\omega(e)$. That is, 90% of the historical transit times have values higher than $\omega(e)$. Therefore, $\omega(e)$ is an optimistic estimation of the traffic condition of e. As mentioned in last paragraph, the weight of an edge does not need to be updated usually unless it changes significantly (*e.g.*, from normal condition to congestion). We say e is in normal condition if its current transit time is $\leq c \cdot \omega(e)$ and is in congestion otherwise. We define an update as either a change from normal condition to congestion or the reverse. For example, for $c = 2$, if the reference transit time $\omega(e)$ corresponds to a speed of 60 mph, an update is triggered when the speed drops from 60 mph to below 30 mph. We show the average # of updates per minute per road in Figure 2f. As can be seen, the # of updates is low (≤ 0.0004) most of the time.

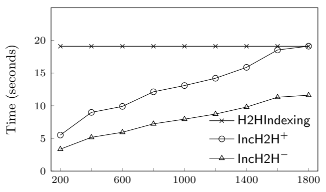
Exp-2: Efficiency of DCH. We conducted a similar experiment on DCH by varying $|\Delta G|$ from $20,000$ to $180,000$. For CHIndexing, the value reported is the time taken to compute the weights of the

[1] http://download.geofabrik.de/

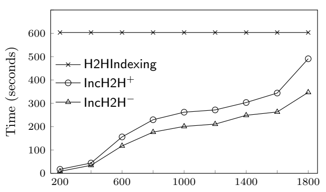
[2] http://www.dis.uniroma1.it/challenge9/download.shtml



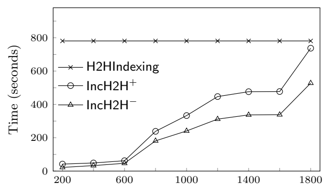
(a) Varying $|\Delta G|$ (ENG)



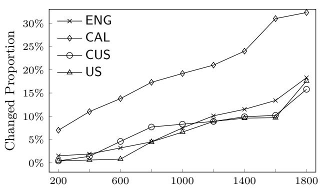
(b) Varying $|\Delta G|$ (CAL)



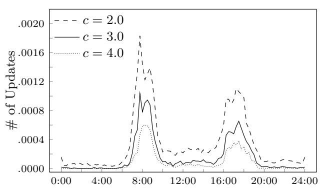
(c) Varying $|\Delta G|$ (CUS)



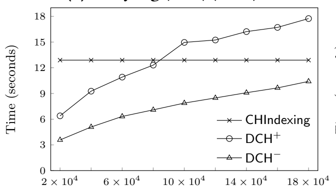
(d) Varying $|\Delta G|$ (US)



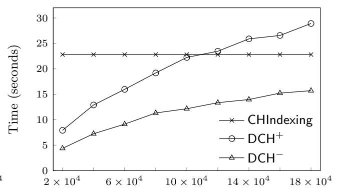
(e) Sensitivity of H2H



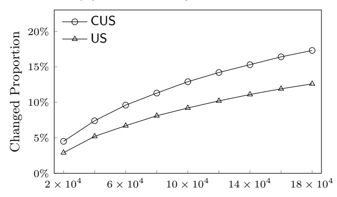
(f) #Updates per Minute per Road



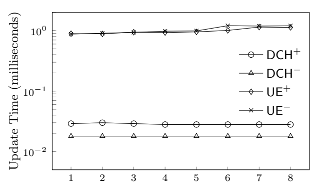
(g) Varying $|\Delta G|$ (CUS)



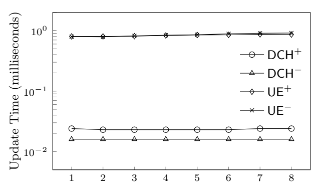
(h) Varying $|\Delta G|$ (US)



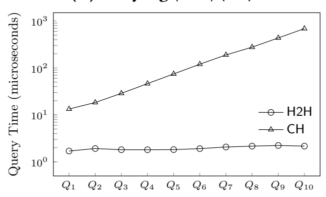
(i) Sensitivity of CH



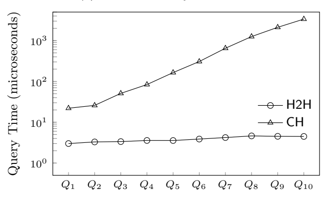
(j) DCH **versus** UE (NY)



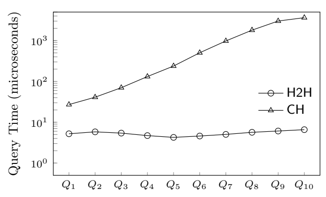
(k) DCH **versus** UE (COL)



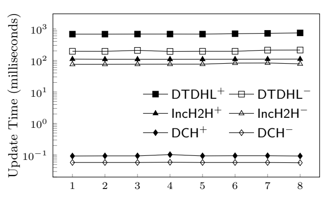
(l) Query Time (WUS)



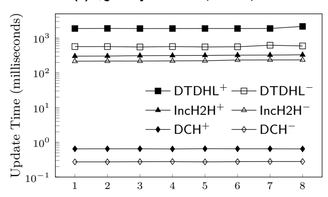
(m) Query Time (CUS)



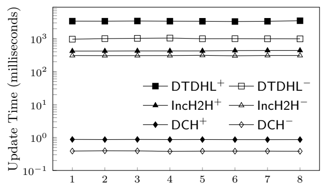
(n) Query Time (US)



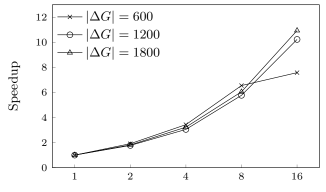
(o) Update Time (WUS)



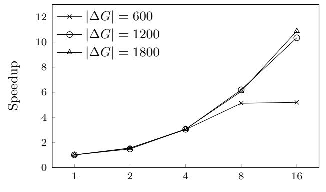
(p) Update Time (CUS)



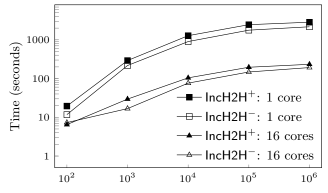
(q) Update Time (US)



(r) Varying # Cores (ParIncH2H$^+$)



(s) Varying # Cores (ParIncH2H$^-$)



(t) Scalability *w.r.t.* $|\Delta G|$ (US)

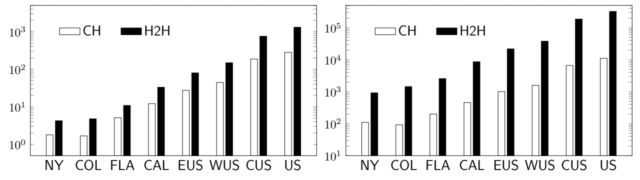
Figure 2: Performance Evaluation

shortcuts from scratch. The results are shown in Figures 2g-2h. We also show the proportion of the index affected by ΔG in Figure 2i. As can be seen, (1) CH is much less sensitive to changes than H2H; and (2) when $|\Delta G|$ is $60,000$ and $80,000$, DCH$^+$ is 1.18 and 1.18 times faster than CHIndexing for CUS and US, respectively; that is, even when 9.6% and 8.1% of the shortcuts need to be updated, DCH$^+$ is still more efficient than reconstructing CH from scratch, again verifying the effectiveness of relative subboundedness.

6.2 Comparison between CH and H2H

We conducted experiments to compare CH and H2H in dynamic networks. As explained in Section 4.3, UE is an unoptimized version of DCH and thus is less efficient (see Figures 2j and 2k whose settings are the same as Exp-4). We omit UE in the following.

Exp-3: Query Time. In this experiment, we tested the query efficiency of CH and H2H. Following [49], for each network tested, we generated 10 groups of queries, denoted $Q_1, Q_2, \ldots, Q_{10}$, as follows: given a network, we first obtained an estimation about the maximum distance d_{max} between two vertices; then, we randomly generated 10,000 pairs of vertices (s_i, t_i) for Q_i such that the distance between s_i and t_i is in the range $[2^{i-11}d_{max}, 2^{i-10}d_{max})$. As a result, for $\forall 1 \leq i < 10$, a pair (s_i, t_i) in Q_i has a smaller distance than a pair (s_{i+1}, t_{i+1}) in Q_{i+1}. For each Q_i, we recorded the average time taken to evaluate the queries in Q_i. The results for WUS, CUS and US are shown in Figures 2l, 2m and 2n, respectively. The results for other networks are similar. We have: (1) As the distance between two query vertices increases, while the query efficiency of H2H does not vary too much, the query time of CH tends to



(a) Time (Seconds) **(b) Space (MB)**

Figure 3: Indexing Time and Index Space

increase in all networks. (2) H2H is about one to three orders of magnitude faster than CH, especially for distant queries, because H2H can avoid searching G or sc(G) when evaluating a query.

Exp-4: Update Time. Following [39], we sampled eight groups of updates, each with 1,000 edges. For the i-th group, we increased the weights of the edges from $\phi(e)$ to $(i+1) \times \phi(e)$ and recorded the average time taken by DCH$^+$/IncH2H$^+$/DTDHL$^+$ to deal with the updates one by one. Then, we restored the weights from $(i+1)\times\phi(e)$ to $\phi(e)$ and recorded the average time by DCH$^-$/IncH2H$^-$/DTDHL$^-$. We show the results for WUS, CUS and US in Figures 2o, 2p and 2q, respectively. As can be observed, (1) IncH2H$^+$ takes about 110, 320 and 420 milliseconds per update on average for WUS, CUS and US, respectively, and IncH2H$^-$ is slightly better; (2) DTDHL$^+$ and DTDHL$^-$ are 6 times and 2 times slower than IncH2H$^+$ and IncH2H$^-$, respectively; (3) DCH$^+$ takes about 0.10, 0.65 and 0.87 milliseconds separately per update on average; and (4) DCH is two to three orders of magnitude faster than IncH2H. This gap in update time, however, does not imply IncH2H is inefficient. After all, DCH and IncH2H maintain different oracles. Unlike CH, H2H caches abundant distance information in its index in order to provide striking query performance, and thus is more sensitive to updates. For example, on US, the average sizes of CHANGED per update for CH and H2H are 181.5 and 26.8×10^5, respectively. It is this difference that leads to the huge gap in update time between DCH and IncH2H. In addition, it is important to note that for indices that are built on H2H, *e.g.*, the state-of-the-art TEN index [38] for the task of nearest neighbor search, IncH2H is a necessary routine to maintain those indices while DCH is not.

Exp-5: Indexing Time and Index Space. We show the indexing time and index space of H2H and CH for all networks in Figures 3a and 3b. We have the following observations. (1) Regarding indexing time, H2H is 2 to 5 times slower than CH to construct. On the other hand, even for the road network of whole United States, H2H can be constructed in half an hour. (2) H2H requires a large amount of memory, compared with CH. This is because (a) in order to do efficient incremental computation, IncH2H additionally needs sup($\cdot$) and first($\cdot$) for each super-shortcut; as a result, incremental H2H needs about two times the memory of static H2H; and (b) due to the large number of super-shortcuts (see Table 2), static H2H itself already consumes a large amount of memory; for example, on the road network US, static H2H requires more than 150GB memory.

$\|\Delta G\|$	100	1,000	10,000	100,000	1,000,000
Proportion	0.17%	6.59%	48.01%	91.63%	98.75%

Table 3: The Proportion Updated *w.r.t.* $|\Delta G|$

6.3 Scalability

Exp-6: Scalability *w.r.t.* # of Cores. Under the same settings as Exp-1 and Exp-2, we tested ParIncH2H on the largest two networks, namely CUS and US, by varying the # of cores from 1 to 16. We report the speedup relative to one core on US in Figures 2r-2s. The results for CUS are similar and thus are omitted. As can be seen, ParIncH2H shows near linear speedup with respect to # of cores, especially when $|\Delta G|$ becomes larger.

Exp-7: Scalability *w.r.t.* $|\Delta G|$. We also tested the scalability of IncH2H on US with respect to the size of the update workload. We varied $|\Delta G|$ from 100 to 1,000,000. The results are shown in Figure 2t. As can be seen, IncH2H scales well even when $|\Delta G|$ becomes large. This is because the performance of IncH2H is largely affected by the # of super-shortcuts to be updated. In Table 3, we show for each $|\Delta G|$ the ratio of the # of super-shortcuts that need to be updated to the total # of super-shortcuts. We can see that when $|\Delta G|$ becomes large, the growth of the proportion to be updated becomes slower, explaining Figure 2t.

7 EXTENSION: EDGE INSERTION/DELETION

Edge updates are infrequent in road networks since road construction and destruction are rare. In [39], an algorithm for edge updates has been proposed for CH. Thus, we focus on H2H below.

For edge deletions, they can be dealt with by increasing the weights of the deleted edges to ∞. As for edge insertions, we first update the underlying shortcut graph sc(G) by invoking the algorithm of [39]. Note that after the update, both the upward neighbors of the vertices and the weights of the shortcuts may change. As a result, the tree decomposition $\mathcal{T}$ is reorganized. Let S_1 be the set of vertices whose parents (in $\mathcal{T}$) or incident shortcuts change, and let $S_2 \subseteq S_1$ be the vertices whose ancestors in $\mathcal{T}$ contain no vertices from S_1. Then, starting from each $v \in S_2$, we update the arrays of its descendants in a top-down manner as in H2HIndexing.

8 CONCLUSIONS

For CH and H2H, we show their unboundedness under specific models of computation. As an alternative to boundedness, we propose relative subboundedness. Despite the unboundedness of CH and H2H, we prove that the state-of-the-art incremental algorithm DCH for CH is relatively subbounded and propose for H2H an incremental algorithm IncH2H that is relatively subbounded. Our experimental results show that both DCH and IncH2H are able to outperform recomputing from scratch even when a non-trivial portion of the index needs to be updated. Our study deepens the understanding of CH and H2H in dynamic road networks.

ACKNOWLEDGMENTS

This work is supported by the Research Grants Council of Hong Kong, China under No. 14203618, No. 14202919 and No. 14205520.

REFERENCES

[1] Ittai Abraham, Daniel Delling, Andrew V Goldberg, and Renato F Werneck. 2011. A hub-based labeling algorithm for shortest paths in road networks. In *SEA*. 230–241.

[2] Ittai Abraham, Daniel Delling, Andrew V Goldberg, and Renato F Werneck. 2012. Hierarchical hub labelings for shortest paths. In *ESA*. 24–35.

[3] Ittai Abraham, Amos Fiat, Andrew V Goldberg, and Renato F Werneck. 2010. Highway dimension, shortest paths, and provably efficient algorithms. In *SODA*. 782–793.

[4] Takuya Akiba, Yoichi Iwata, Ken-ichi Kawarabayashi, and Yuki Kawata. 2014. Fast shortest-path distance queries on road networks by pruned highway labeling. In *ALENEX*. 147–154.

[5] Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. 2013. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *SIGMOD*. 349–360.

[6] Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. 2014. Dynamic and historical shortest-path distance queries on large evolving networks by pruned landmark labeling. In *WWW*. 237–248.

[7] Takuya Akiba, Christian Sommer, and Ken-ichi Kawarabayashi. 2012. Shortest-path queries for complex networks: Exploiting low tree-width outside the core. In *EDBT*. 144–155.

[8] Bowen Alpern, Roger Hoover, Barry K Rosen, Peter F Sweeney, and F Kenneth Zadeck. 1990. Incremental evaluation of computational circuits. In *SODA*. 32–42.

[9] Holger Bast, Stefan Funke, and Domagoj Matijević. 2009. Ultrafast shortest-path queries via transit nodes. In *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*. AMS, 175–192.

[10] Reinhard Bauer and Daniel Delling. 2008. SHARC: Fast and robust unidirectional routing. In *ALENEX*. 13–26.

[11] Arthur M Berman. 1992. *Lower and upper bounds for incremental algorithms*. Ph.D. Dissertation. Rutgers University.

[12] Anne Berry, Pinar Heggernes, and Genevieve Simonet. 2003. The minimum degree heuristic and the minimal triangulation process. In *WG*. 58–70.

[13] Zitong Chen, Ada Wai-Chee Fu, Minhao Jiang, Eric Lo, and Pengfei Zhang. 2021. P2H: Efficient distance querying on road networks by projected vertex separators. In *SIGMOD*. 313–325.

[14] James Cheng, Yiping Ke, Shumo Chu, and Carter Cheng. 2012. Efficient processing of distance queries in large graphs: A vertex cover approach. In *SIGMOD*. 457–468.

[15] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. 2003. Reachability and distance queries via 2-hop labels. *SICOMP* 32, 5 (2003), 1338–1355.

[16] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2009. *Introduction to algorithms*. MIT press.

[17] Gianlorenzo D'angelo, Mattia D'emidio, and Daniele Frigioni. 2019. Fully dynamic 2-hop cover labeling. *JEA* 24, 1 (2019), 1.6:1–1.6:36.

[18] Daniel Delling, Andrew V Goldberg, Thomas Pajor, and Renato F Werneck. 2014. Robust distance queries on massive networks. In *ESA*. 321–333.

[19] Daniel Delling, Andrew V Goldberg, Thomas Pajor, and Renato F Werneck. 2017. Customizable route planning in road networks. *Transportation Science* 51, 2 (2017), 566–591.

[20] Ronald Fagin, Amnon Lotem, and Moni Naor. 2001. Optimal Aggregation Algorithms for Middleware. In *PODS*. 102–113.

[21] Wenfei Fan, Chunming Hu, and Chao Tian. 2017. Incremental graph computations: Doable and undoable. In *SIGMOD*. 155–169.

[22] Wenfei Fan, Muyang Liu, Chao Tian, Ruiqi Xu, and Jingren Zhou. 2020. Incrementalization of graph partitioning algorithms. *PVLDB* 13, 8 (2020), 1261–1274.

[23] Wenfei Fan, Chao Tian, Ruiqi Xu, Qiang Yin, Wenyuan Yu, and Jingren Zhou. 2021. Incrementalizing Graph Algorithms. In *SIGMOD*. 459–471.

[24] Wenfei Fan, Xin Wang, and Yinghui Wu. 2013. Incremental graph pattern matching. *TODS* 38, 3 (2013), 18.

[25] Ada Wai-Chee Fu, Huanhuan Wu, James Cheng, and Raymond Chi-Wing Wong. 2013. IS-Label: An independent-set based labeling scheme for point-to-point distance querying. *PVLDB* 6, 6 (2013), 457–468.

[26] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. 2008. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *WEA*. 319–333.

[27] Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. 2012. Exact routing in large road networks using contraction hierarchies. *Transportation Science* 46, 3 (2012), 388–404.

[28] Andrew V Goldberg and Chris Harrelson. 2005. Computing the shortest path: A* search meets graph theory. In *SODA*. 156–165.

[29] Andrew V Goldberg, Haim Kaplan, and Renato F Werneck. 2009. Reach for A*: Shortest path algorithms with preprocessing. In *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*. AMS, 93–140.

[30] Ron Gutman. 2004. Reach-based routing: A new approach to shortest path algorithms optimized for road networks. In *ALENEX*. 100–111.

[31] Dan He, Sibo Wang, Xiaofang Zhou, and Reynold Cheng. 2019. An efficient framework for correctness-aware knn queries on road networks. In *ICDE*. 1298–1309.

[32] Moritz Hilger, Ekkehard Köhler, Rolf H Möhring, and Heiko Schilling. 2009. Fast point-to-point shortest path computations with arc-flags. In *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*. AMS, 41–72.

[33] Minhao Jiang, Ada Wai-Chee Fu, Raymond Chi-Wing Wong, and Yanyan Xu. 2014. Hop doubling label indexing for point-to-point distance querying on scale-free networks. *PVLDB* 7, 12 (2014), 1203–1214.

[34] Ruoming Jin, Ning Ruan, Yang Xiang, and Victor Lee. 2012. A highway-centric labeling approach for answering distance queries on large sparse graphs. In *SIGMOD*. 445–456.

[35] Sebastian Knopp, Peter Sanders, Dominik Schultes, Frank Schulz, and Dorothea Wagner. 2007. Computing many-to-many shortest paths using highway hierarchies. In *ALENEX*. 36–45.

[36] Ye Li, Man Lung Yiu, Ngai Meng Kou, et al. 2017. An experimental study on hub labeling based shortest path algorithms. *PVLDB* 11, 4 (2017), 445–457.

[37] Dian Ouyang, Lu Qin, Lijun Chang, Xuemin Lin, Ying Zhang, and Qing Zhu. 2018. When hierarchy meets 2-hop-labeling: Efficient shortest distance queries on road networks. In *SIGMOD*. 709–724.

[38] Dian Ouyang, Dong Wen, Lu Qin, Lijun Chang, Ying Zhang, and Xuemin Lin. 2020. Progressive Top-K Nearest Neighbors Search in Large Road Networks. In *SIGMOD*. 1781–1795.

[39] Dian Ouyang, Long Yuan, Lu Qin, Lijun Chang, Ying Zhang, and Xuemin Lin. 2020. Efficient shortest path index maintenance on dynamic road networks with theoretical guarantees. *PVLDB* 13, 5 (2020), 602–615.

[40] Yongrui Qin, Quan Z Sheng, Nickolas JG Falkner, Lina Yao, and Simon Parkinson. 2017. Efficient computation of distance labeling for decremental updates in large dynamic graphs. *WWWJ* 20, 5 (2017), 915–937.

[41] Ganesan Ramalingam and Thomas Reps. 1996. On the computational complexity of dynamic graph problems. *Theoretical Computer Science* 158, 1 (1996), 233–277.

[42] Hanan Samet, Jagan Sankaranarayanan, and Houman Alborzi. 2008. Scalable network distance browsing in spatial databases. In *SIGMOD*. 43–54.

[43] Peter Sanders and Dominik Schultes. 2005. Highway hierarchies hasten exact shortest path queries. In *ESA*. 568–579.

[44] Jagan Sankaranarayanan, Hanan Samet, and Houman Alborzi. 2009. Path oracles for spatial networks. *PVLDB* 2, 1 (2009), 1210–1221.

[45] Dominik Schultes and Peter Sanders. 2007. Dynamic highway-node routing. In *WEA*. 66–79.

[46] Full Version. 2022. `https://www.se.cuhk.edu.hk/~ykzhang/mod22-full.pdf`

[47] Fang Wei. 2010. TEDI: Efficient shortest path query answering on graphs. In *SIGMOD*. 99–110.

[48] Victor Junqiu Wei, Raymond Chi-Wing Wong, and Cheng Long. 2020. Architecture-intact oracle for fastest path and time queries on dynamic spatial networks. In *SIGMOD*. 1841–1856.

[49] Lingkun Wu, Xiaokui Xiao, Dingxiong Deng, Gao Cong, Andy Diwen Zhu, and Shuigeng Zhou. 2012. Shortest path and distance queries on road networks: an experimental evaluation. *PVLDB* 5, 5 (2012), 406–417.

[50] Yosuke Yano, Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. 2013. Fast and scalable reachability queries on graphs by pruned labeling with landmarks and paths. In *CIKM*. 1601–1606.

[51] Mengxuan Zhang, Lei Li, Wen Hua, Rui Mao, Pingfu Chao, and Xiaofang Zhou. 2021. Dynamic hub labeling for road networks. In *ICDE*. 336–347.

[52] Andy Diwen Zhu, Hui Ma, Xiaokui Xiao, Siqiang Luo, Youze Tang, and Shuigeng Zhou. 2013. Shortest path and distance queries on road networks: Towards bridging theory and practice. In *SIGMOD*. 857–868.