Fast Shortest-path Distance Queries on Road Networks by Pruned Highway Labeling

Takuya Akiba* Yoichi Iwata† Ken-ichi Kawarabayashi‡ Yuki Kawata§

Abstract

We propose a new labeling method for shortest-path and distance queries on road networks. We present a new framework (i.e. data structure and query algorithm) referred to as *highway-based labelings* and a preprocessing algorithm for it named *pruned highway labeling*.

Our proposed method has several appealing features from different aspects in the literature. Indeed, we take advantages of theoretical analysis of the seminal result by Thorup for distance oracles, more detailed structures of real road networks, and the pruned labeling algorithm that conducts *pruned* Dijkstra's algorithm.

The experimental results show that the proposed method is comparable to the previous state-of-the-art labeling method in both query time and in data size, while our main improvement is that the preprocessing time is much faster.

1 Introduction

Computing shortest path or distance between two points is one of the most fundamental and important problems on road networks with many applications. In most of cases, this computation problem can be described in the following algorithmic problem: Given a (almost) planar graph, we preprocess it subject to the restriction that the space for the preprocessing results is limited (e.g., a small constant times the space used to store the graph). Then we would like to quickly answer queries on single-pair (or multipair) shortest paths. This is a natural variant of the problem as in many applications, such as that of computing driving directions (e.g., Google Maps, Bing Maps, Yahoo! Maps). Therefore theoretical and practical approaches involving precomputation on input graphs have been particularly a central topic in the literature for more than two decades and continuing [11, 4, 9, 10, 5, 6, 1, 2, 7].

One of the most notable recent developments is the emergence of practical *labeling methods* [1, 2]. Labeling methods precompute *label* $L(v)$ for each vertex v and answer distance between two points s and t using labels $L(s)$ and $L(t)$ without looking at the input graph. These practical methods adopt the most standard labeling framework (i.e., data structure and query algorithm) called *hub-based labelings* [1]. The label $L(v)$ is a set of pairs (u, δ_{uv}), where u is a vertex and δ_{uv} is distance between v and u. Since the labeling method can answer distance by only looking at two consecutive segments of precomputed data, their query time is known to be the fastest among all other proposed methods.

1.1 Contribution at a High Level

In this paper, we shall improve the state-of-the-art algorithm (i.e, the labeling method) in several ways. Specifically, we present a new labeling framework referred to as the *highway-based labeling framework* and a preprocessing algorithm for the framework named *pruned highway labeling*. Our proposed method has several appealing features from different aspects in the literature.

Distance oracles. The idea behind our approach has some similarity with Thorup's distance oracle for planar graphs [12]. Specifically, our approach repeatedly separates input graphs by shortest paths, and then stores distance from vertices to the shortest paths. These are also keys in Thorup's paper. Indeed, we can adapt Thorup's proof to justify the correctness and efficiency of our proposed algorithm theoretically, to some extent.

Highways. Several previous methods exploit *highway* structure in road networks. Highways are useful since they are *central*, in the sense that many shortest paths must pass through highways. However, the hub-based labeling algorithm does not directly exploit highways. In contrast, our approach takes more advantage of these highways.

Pruned labeling. The notion of *pruned labeling* [3] is the key ingredient of the recent state-of-the-art label computation algorithms for distance queries on complex networks [3] and reachability queries on DAGs [13].

*The University of Tokyo and JST, ERATO, Kawarabayashi Project, t.akiba@is.s.u-tokyo.ac.jp

†The University of Tokyo and JST, ERATO, Kawarabayashi Project, y.iwata@is.s.u-tokyo.ac.jp

‡National Institute of Informatics and JST, ERATO, Kawarabayashi Project, k_keniti@nii.ac.jp

§The University of Tokyo and JST, ERATO, Kawarabayashi Project, kawatea@lager.is.s.u-tokyo.ac.jp

While the highway-based labeling framework is a natural generalization of the hub-based labeling framework, it seems hard to extend previous label computation algorithms for hub-based labelings to road networks [1, 2]. In contrast, the notion of pruned labeling turns out to be possible to generalize naturally and efficiently to highway-based labelings.

1.2 Technical Contributions Let us state our technical contributions more precisely:

1. We propose a new labeling framework referred to as the *highway-based labeling framework*, which is a natural generalization of hub-based labeling (Section 3). It decomposes a graph into shortest paths, and its labels store distances from each vertex to the shortest paths.
2. We propose an algorithm for computing small labels for highway-based labelings named *pruned highway labeling* (Section 4). It is a simple algorithm based on pruned labeling [3] that conducts *pruned* Dijkstra's algorithm from each path.
3. We discuss heuristics for obtaining good decompositions, as well as techniques for efficient implementation (Section 5).
4. We experimentally evaluate our proposed method (Section 6). As it is also a labeling method, its query time is sufficiently fast. In addition, the data size is comparable to the previous state-of-the-art labeling method. Our main improvement is that the preprocessing time is much faster.

2 Preliminaries

2.1 Notations In this paper, a road network is expressed in a graph $G = (V, E)$. Here V is the vertex set and E is the edge set. Throughout the paper, we assume that a graph is undirected. For each edge $(v, w) \in E$, we denote its length or travel time by $l(v, w)$. Let $d(v, w)$ denote the distance between vertices v and w. If there is no fear of confusion, we do not distinguish between a subgraph W of G and a vertex set $V(W)$ of $V(G)$.

2.2 Hub-based Labeling Framework In the hub-based labeling framework [1], we precompute a label $L(v)$ for each vertex v. Each label $L(v)$ is a set of pairs $(w, d(v, w))$, where w is a vertex and $d(v, w)$ is the distance between v and w.

For an s-t query, we search for the vertex v that is contained in both $L(s)$ and $L(t)$ and minimizes $d(s, v) + d(v, t)$. That is, $\textsc{Query}(s, t, L) = \min\{d(s, v) + d(v, t) \mid (v, d(s, v)) \in L(s), (v, d(t, v)) \in L(t)\}$.

If for any pair of vertices s and t, labels $L(s)$ and $L(t)$ contain at least one common vertex on the shortest paths between vertices s and t, then we can answer any query correctly by using only these labels. In addition, by sorting vertices in ascending order for every label, we can answer an s-t query in $O(|L(s)| + |L(t)|)$ time using a merge-sort-like algorithm.

2.3 Pruned Landmark Labeling Pruned landmark labeling [3] is an algorithm first proposed for computing hub-based labels efficiently on complex networks such as social networks and web graphs. In this algorithm, we conduct a breath first search from every vertex u one by one, and for each visited vertex v, we add the distance $d(v, u)$ to the label $L(v)$. Although this seems inefficient at first, we can introduce effective "pruning" to this process as follows. Suppose we are conducting a breadth first search from vertex u and visiting vertex v with distance δ. Then we estimate an u-v query by using current incomplete labels, and we prune v if the result is less than or equal to δ. That is, we need neither to add the pair $(u, d(v, u))$ to $L(v)$ nor to check edges from the vertex v. For more details, we refer the reader to [3].

3 Highway-based Labeling Framework

In this section, we propose a new labeling framework (i.e., data structure and query algorithm) referred to as the "highway-based labeling framework". We first introduce the notion of a *highway decomposition* and explain what we store for labels (Section 3.1). This concept is similar to that used in Thorup [12] (but a slightly different). We then present our query algorithm (Section 3.2).

3.1 Highway Decomposition and Labels First, we define a *highway decomposition*, which is a key for our proposed framework. In what follows, we identify a path with an ordered set of vertices.

DEFINITION 3.1. *A* highway decomposition *of a given graph G is a family of ordered sets of vertices $\mathscr{P} = \{P_1, P_2, ..., P_N\}$ such that,*

1. *$P_i = (p_{i,1}, p_{i,2}, \ldots, p_{i,l_i})$ is a shortest path between two vertices $p_{i,1}$ and p_{i,l_i},*
2. *$P_i \cap P_j = \emptyset$ for any i and j $(i \neq j)$, and*
3. *$P_1 \cup P_2 \cup \ldots \cup P_N = V$.*

In Section 5 we will describe how to compute a highway decomposition of a graph. In the following, we assume that we are given such a decomposition. In the highway-based labeling framework, a label $L(v)$ for a vertex v is a set of triples $(i, d(p_{i,1}, p_{i,j}), d(v, p_{i,j}))$, that is, for each vertex v, we store an index i of a path P_i, distance from the starting point $p_{i,1}$ of the path to

a vertex $p_{i,j}$ on the path, and distance from the vertex v to the vertex $p_{i,j}$[1].

In the hub-based labeling framework, a candidate for the distance between s and t are computed from pairs $(v, d(s,v)) \in L(s)$ and $(v, d(t,v)) \in L(t)$ for the same vertex v. On the other hand, in the highway-based labeling framework, we can compute a candidate for the distance between s and t even for the case that the vertices stored in two triples are different. Indeed, suppose that the label $L(s)$ contains a triple $(i, d(p_{i,1}, p_{i,j}), d(s, p_{i,j}))$ and the label $L(t)$ contains a triple $(i, d(p_{i,1}, p_{i,k}), d(t, p_{i,k}))$. Then we can obtain a candidate distance by looking at $d(s, p_{i,j}) + d(p_{i,j}, p_{i,k}) + d(p_{i,k}, t) = d(s, p_{i,j}) + |d(p_{i,1}, p_{i,j}) - d(p_{i,1}, p_{i,k})| + d(p_{i,k}, t)$. This observation is crucial in our experiment. It allows us to make the label for each vertex to be smaller size. This is a clear advantage of the highway-based labeling framework. In particular, a large part of distances between pairs of far vertices can be answered by storing a few distances to central paths in real road networks (e.g. highways).

3.2 Query For an s-t query, we search for the triples that minimize the candidate distance. That is, we define the answer to an s-t query using labels L as

$$\begin{aligned}&\textsc{Query}(s,t,L)\\ &= \min\{d(s,p_{i,j}) + d(p_{i,j},p_{i,k}) + d(p_{i,k},t) \mid\\ &\qquad (i, d(p_{i,1},p_{i,j}), d(s,p_{i,j})) \in L(s),\\ &\qquad (i, d(p_{i,1},p_{i,k}), d(t,p_{i,k})) \in L(t)\}.\end{aligned}$$

The distance $d(p_{i,j}, p_{i,k})$ itself is not contained in labels $L(s)$ and $L(t)$, but can be computed by using $d(p_{i,1}, p_{i,j})$ and $d(p_{i,1}, p_{i,k})$ as mentioned above.

Naively computing the function above takes $\Theta(|L(s)|\,|L(t)|)$ time, but we can obtain a linear-time algorithm based on the following lemma.

LEMMA 3.1. *There exist triples* $(i, d(p_{i,1}, p_{i,j}), d(s, p_{i,j})) \in L(s)$ *and* $(i, d(p_{i,1}, p_{i,k}), d(t, p_{i,k})) \in L(t)$ *that achieve the minimum candidate distance and satisfy the following: for any vertex* $p_{i,l}$ *with* $\min(j,k) < l < \max(j,k)$, $(i, d(p_{i,1}, p_{i,l}), d(s, p_{i,l})) \notin L(s)$ *and* $(i, d(p_{i,1}, p_{i,l}), d(t, p_{i,l})) \notin L(t)$.

Proof. Let $(i, d(p_{i,1}, p_{i,j}), d(s, p_{i,j})) \in L(s)$ and $(i, d(p_{i,1}, p_{i,k}), d(t, p_{i,k})) \in L(t)$ be triples that minimize the candidate distance. Let us choose j, k with $j < k$ such that $k - j$ is as small as possible. Suppose there exists $p_{i,l}$ with $j < l < k$ such that $(i, d(p_{i,1}, p_{i,l}), d(s, p_{i,l})) \in L(s)$. Then the candidate distance computed from triples $(i, d(p_{i,1}, p_{i,l}), d(s, p_{i,l}))$ and $(i, d(p_{i,1}, p_{i,k}), d(t, p_{i,k}))$ must also be the minimum, because

$$\begin{aligned}& d(s,p_{i,l}) + d(p_{i,l},p_{i,k}) + d(p_{i,k},t)\\ =\ & d(s,p_{i,l}) + d(p_{i,1},p_{i,k}) - d(p_{i,1},p_{i,l}) + d(p_{i,k},t)\\ =\ & (d(s,p_{i,j}) + d(p_{i,j},p_{i,l})) + d(p_{i,1},p_{i,k})\\ & -(d(p_{i,1},p_{i,j}) + d(p_{i,j},p_{i,l})) + d(p_{i,k},t)\\ =\ & d(s,p_{i,j}) + d(p_{i,1},p_{i,k}) - d(p_{i,1},p_{i,j}) + d(p_{i,k},t)\\ =\ & d(s,p_{i,j}) + d(p_{i,j},p_{i,k}) + d(p_{i,k},t)\end{aligned}$$

Note that P_i is a shortest path. This is a contradiction.

Therefore, like previous labeling methods, by sorting triples in labels with indexes and the distances from the starting point of the path in ascending order beforehand, we can answer an s-t query in $O(|L(s)| + |L(t)|)$ time using a merge-sort-like algorithm.

4 Pruned Highway Labeling

In this section we propose our label computation algorithm for highway-based labelings. Throughout this section, we assume that a highway decomposition $\mathscr{P}$ is given (in the next section we will show how to obtain such a decomposition).

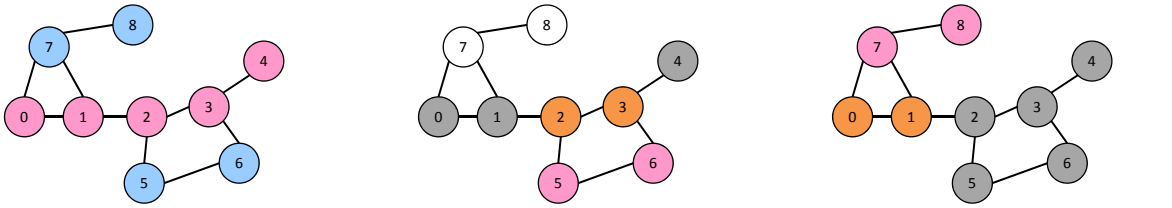
4.1 Naive Highway Labeling Before presenting our efficient preprocessing algorithm, we first give a naive algorithm to compute correct labels for the highway-based labeling framework. We start with empty labels L_0 (i.e., $L_0(v) = \emptyset$ for each vertex v) and then construct new labels L_{i+1} from L_i iteratively. In order to construct the labels L_i, we first copy the labels L_{i-1} to L_i, and then conduct the Dijkstra search from each vertex $p_{i,j}$ on the path P_i and add the distance between vertices v and $p_{i,j}$ to the label $L_i(v)$. That is, $L_i(v) = L_{i-1}(v) \cup (i, d(p_{i,1}, p_{i,j}), d(v, p_{i,j}))$ for all $p_{i,j} \in P_i$. After we conduct all Dijkstra searches, we obtain the labels $L = L_N$. Because $L(v)$ contains the distance from v to all vertices, the labels can answer correct distances between v and any other vertex. Therefore we have the following.

LEMMA 4.1. *For any pair of vertices* s *and* t, $\textsc{Query}(s,t,L) = d(s,t)$.

4.2 Pruned Highway Labeling We now propose an efficient algorithm for computing labels for the highway-based labeling framework named "pruned highway labeling". This algorithm is based on the pruned landmark labeling [3].

Similarly to the naive algorithm, we start with empty labels L'_0 and then construct new labels L'_{i+1}

[1] The label $L(v)$ does not necessarily contain triples from all indexes i nor contains triples for all the vertices on a path P_i, because in our preprocessing, we will reduce the total size of labels $L(v)$.



(a) Pruned Dijkstra search from the path P_1 (b) Pruned Dijkstra search from the path P_2 (c) Pruned Dijkstra search from the path P_3

Figure 1: Examples for the pruned highway labeling. Pink vertices are on the starting path P_i, blue vertices are visited and added to some labels, gray vertices are already used as the starting points of the previous searches, orange vertices are visited but pruned, and white vertices are not visited.

Algorithm 1 Pruned Dijkstra search from P_i to compute the labels L'_i

procedure PRUNEDDIJKSTRASEARCH(G, P_i, L'_{i-1})
  $Q \leftarrow$ an empty priority queue
  Push $(0, p_{i,j}, p_{i,j})$ onto Q for all $p_{i,j} \in P_i$
  $L'_i(v) \leftarrow L'_{i-1}(v)$ for all $v \in V$
  while Q is not empty **do**
    Pop $(\delta, v, p_{i,j})$ from Q
    if QUERY$(v, p_{i,j}, L'_i) \leq \delta$ **then**
      continue
    end if
    $L'_i(v) \leftarrow L'_i(v) \cup (i, d(p_{i,1}, p_{i,j}), \delta)$
    Push $(\delta + l(v, w), w, p_{i,j})$ onto Q for all $(v, w) \in E$
  end while
  return L'_i
end procedure

Algorithm 2 Preprocessing by the pruned highway labeling

procedure PREPROCESS(G)
  $L'_0(v) \leftarrow \emptyset$ for all $v \in V$
  $\mathscr{P} \leftarrow$ a highway decomposition of G
  $N \leftarrow$ the size of $\mathscr{P}$
  for $i = 1$ to N **do**
    $L'_i \leftarrow$ PRUNEDDIJKSTRASEARCH(G, P_i, L'_{i-1})
  end for
  return L'_N
end procedure

from L'_i iteratively. In order to construct the labels L'_i, we conduct the *pruned* Dijkstra search from all the vertices on the path P_i simultaneously as described in Algorithm 1. When we visit a vertex v from the vertex $p_{i,j}$ with distance δ, if QUERY$(v, p_{i,j}, L'_i)$ is less than or equal to δ, we prune the search. Otherwise, we add the triple $(i, d(p_{i,1}, p_{i,j}), \delta)$ to $L'_i(v)$ and check edges from the vertex v. After we conduct the pruned Dijkstra searches from all the paths, we obtain the labels $L' = L'_N$ for the highway-based labeling framework. We prove the correctness of our algorithm in Section 4.4, but in the next subsection, we shall give intuition how our algorithm goes.

4.3 Example For Pruned Highway Labeling

Figure 1 illustrates examples for the pruned highway labeling. For simplicity, we assume that the length of all edges is 1.

First, we decompose the input graph. Suppose that we are given a highway decomposition $\mathscr{P} = \{P_1, P_2, P_3\}$, where $P_1 = \{0, 1, 2, 3, 4\}, P_2 = \{5, 6\}, P_3 = \{7, 8\}$. Then we conduct the pruned Dijkstra search from the path P_1 (Figure 1a). A triple (1, 2, 1) is added to the label $L'_1(5)$. However, a triple (1, 3, 2) is not added to the label $L'_1(5)$ because QUERY$(5, 3, L'_1) = d(5, 2) + d(2, 3) + d(3, 3) = 1 + 1 + 0 = 2$. Therefore, the search from the vertex 3 is pruned. In the same way, only one triple (1, 3, 1) is added to the label $L'_1(6)$. On the other hand, two triples (1, 0, 1) and (1, 1, 1) are added to the label $L'_1(7)$ because we cannot prune the search.

Next, we conduct the pruned Dijkstra search from the path P_2 (Figure 1b). Because the search is pruned, no triples are added to labels $L'_2(2)$ nor $L'_2(3)$ and no other vertices are visited. Similarly, we conduct the pruned Dijkstra search from the path P_3 (Figure 1c).

For a query between vertices 6 and 7, we check the two labels $L'(6)$ and $L'(7)$. The triple (1, 3, 1) is contained in $L'(6)$ and the triple (1, 0, 1) is contained in $L'(7)$, so the distance computed by these triples is $d(6, 3) + d(3, 0) + d(0, 7) = 1 + 3 + 1 = 5$. $L'(7)$ also contains the triple (1, 1, 1). We can indeed get 4 as the distance by using this triple. Although both labels $L'(6)$ and $L'(7)$ contain other triples, these triples

do not coincide to the index of the path. Therefore, we do not need to check these triples. As a result, $\textsc{Query}(6, 7, L') = 4$.

4.4 Proof of Correctness In this subsection, we prove the correctness of the pruning. Specifically, we prove that the distance computed by using labels L' from the pruned highway labeling is equal to the distance computed by using labels L from the naive highway labeling.

THEOREM 4.1. *For any pair of vertices s and t, $\textsc{Query}(s, t, L') = \textsc{Query}(s, t, L)$.*

Proof. For vertices s and t, let i be the index such that $\textsc{Query}(s, t, L_{i'}) \neq d(s, t)$ for any $i' < i$ and $\textsc{Query}(s, t, L_i) = d(s, t)$. Then there exist vertices $p_{i,j}, p_{i,k} \in P_i$ that satisfy $d(s, t) = d(s, p_{i,j}) + d(p_{i,j}, p_{i,k}) + d(p_{i,k}, t)$. Among others, we choose a pair (j, k) such that no vertices on the shortest path between s and $p_{i,j}$ or between $p_{i,k}$ and t are on the path P_i. Suppose that for some $i' < i$, a vertex $p_{i',j'}$ on the path $P_{i'}$ is on a shortest path between s and $p_{i,j}$. Then the label $L(s)$ contains the triple $(i', d(p_{i',1}, p_{i',j'}), d(s, p_{i',j'}))$ and the label $L(t)$ contains the triple $(i', d(p_{i',1}, p_{i',j'}), d(t, p_{i',j'}))$, and therefore $\textsc{Query}(s, t, L_{i'}) = d(s, t)$ holds, which is a contradiction to the choice of i. Therefore any path $P_{i'}$ with $i' < i$ contains no vertices on the shortest paths between s and $p_{i,j}$. Thus the search from $p_{i,j}$ to s is not pruned and the triple $(i, d(p_{i,1}, p_{i,j}), d(s, p_{i,j}))$ is added to the label $L'(s)$. In the same way, the triple $(i, d(p_{i,1}, p_{i,k}), d(t, p_{i,k}))$ is added to the label $L'(t)$. As a result, $\textsc{Query}(s, t, L') = \textsc{Query}(s, t, L)$ holds.

From this theorem, we can answer a query for any pair of vertices correctly by using labels computed by the pruned highway labeling.

COROLLARY 4.1. *Let L' be the labels computed by the pruned highway labeling. Then for any pair of vertices s and t,*

$$\textsc{Query}(s, t, L') = d(s, t).$$

5 Detailed Algorithm Description

5.1 Highway Decomposition Until this point, we assumed that a highway decomposition is already given, and then we conduct the first pruned Dijkstra search. We now show how to construct a highway decomposition and labeling simultaneously. Specifically, we choose a path P_i from remaining vertices just before conducting the i-th pruned Dijkstra search. In the pruned landmark labeling algorithm, the order of vertices to start BFSs is crucial to achieve small labels [3]. Similarly, in our method, the order of paths P_i in a highway decomposition is important, and this allows us to achieve small label sizes.

At a high level, we want to choose a path that *hits* many shortest paths at the early stage of Dijkstra search, because this would allow us to prune future Dijkstra searches. In what follows, we propose strategies for selecting such a good path.

First, we focus on the speed of an edge (i.e., the geometrical length divided by the travel time). In the real life, we tend to use fast highways when we travel long distance. Therefore, we may assume that the vertices connected to fast edges tend to be passed by many shortest paths. For this reason, we group vertices into several levels according to the speed of their connected edges. We assign vertices connected to faster edges to higher levels and then we choose a path consisting of the highest level vertices. When the number of vertices in the highest level is fewer than some threshold, we mix unused vertices in the highest level with vertices in the second highest level to make it possible to select a path of enough length.

We now describe how to choose a path from the highest level vertices. Because the selected path must be a shortest path between two vertices, we first compute the shortest path tree from a randomly selected root vertex, and then pick a path between the root vertex and another vertex in the shortest path tree. The more descendants a vertex has in the shortest path tree, the more shortest paths on the tree hit the vertex. Therefore, in order to select a path that hits many shortest paths, we choose the path by starting from the root vertex and iteratively pick a child with the largest number of descendants.

Finally, we describe our technique to skip unimportant vertices. Suppose that a vertex v on the selected path hits many shortest paths, but most of them also contain another vertex w on the selected path. In this case, even if we skip the vertex v while keeping the vertex w, most of the future searches that will be pruned by v are still pruned by w. Thus we skip such unimportant vertices from the selected path. Suppose that we chose a vertex v on the shortest path tree, and then choose its child w. If the difference between the number of descendants of v and w is small, most of the shortest paths on the tree that contain v also contain w, and therefore we skip v. This does not affect the correctness of the algorithm because this operation corresponds to add shortcut edges across skipped vertices.

In our implementation used in the experiments, we group vertices into four levels and we skip a vertex v when the difference between the number of descendants of vertices v and its child w is smaller than five percents.

Table 1: Comparison of the performance between pruned highway labeling and previous methods. HL is parallelized to use 12 cores in preprocessing and all other methods are not parallelized.

	USA			Europe		
Method	Preprocessing [h:m]	Space [GB]	Query [ns]	Preprocessing [h:m]	Space [GB]	Query [ns]
CH [5]	0:27	0.5	130000	0:25	0.4	180000
TNR [5]	1:30	5.4	3000	1:52	3.7	3400
TNR+AF [5]	2:37	6.3	1700	3:49	5.7	1900
HL local [1]	2:24	22.7	627	2:39	20.1	572
HL global [1]	2:35	25.4	266	2:45	21.3	276
HL-15 local [2]	-	-	-	0:05	18.8	556
HL-∞ global [2]	-	-	-	6:12	17.7	254
HLC-15 [7]	0:53	2.9	2486	0:50	1.8	2554
PHL-1	0:29	16.4	941	0:34	14.9	1039

5.2 Contraction Technique In this subsection, we introduce a new technique called the *contraction technique*. First, we consider a vertex v of degree one. Any shortest path from v to another vertex passes through its adjacent vertex w. Moreover, the vertex v is never contained in shortest paths between other vertices. Therefore, we can remove the vertex v from the graph in the preprocessing. We can correctly answer the query between the vertex v and another vertex u by referring the label $L(w)$ and adding the length $l(v, w)$, that is, $\textsc{Query}(v, u, L) = \textsc{Query}(w, u, L) + l(v, w)$.

Next, we consider a vertex v of degree larger than one. In this case, the vertex v may be contained in a shortest path between other vertices. Therefore, we need to add some shortcut edges before removing the vertex v to preserve the distances. We can correctly answer the distance between v and another vertex u by evaluating the distances from all the neighbors of v to u, that is, $\textsc{Query}(v, u, L) = \min_{(v,w)\in E(G)}\{\textsc{Query}(w, u, L) + l(v, w)\}$. However, the larger the degree limit is, the slower query time becomes due to the number of neighbors to check. In our experiments, we only set the degree limit to be at most three.

5.3 Storing Labels To make highway-based labelings more practical, we describe the way to store labels in this subsection. The label $L(v)$ is a set of triples $(i, d(p_{i,1}, p_{i,j}), d(v, p_{i,j}))$, where i is an index of a path. The distance information in the label is used in the query only when the indexes of the paths are same in two triples. Therefore, storing the index information and the distance information separately makes the query time faster because we can avoid cache misses. For a single path, multiple triples may be stored to a vertex. Therefore, by storing pairs of an index and the number of triples for the index, we can reduce the space usage. Moreover, this also makes the query time faster because we can reduce unnecessary comparisons when the indexes do not match in two triples. For more efficient implementation, we use pointer arithmetic and align arrays storing labels to cache line.

6 Experimental Evaluation

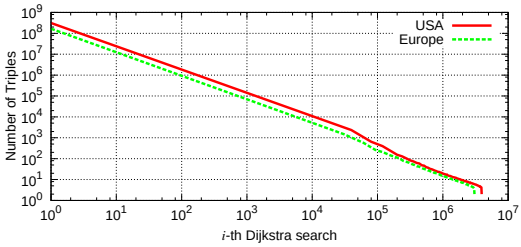
6.1 Setup We conducted experiments on a Linux server with Intel Xeon X5675 processor (3.06 GHz) and 288 GB for main memory. We implemented the proposed method in C++ and compiled it with the GNU C++ compiler 4.4.6 using optimization level 3. We did not parallelize the preprocessing and queries and used one core. We evaluate query time as the average time for 1,000,000 random queries.

We used two graph instances from 9th DIMACS Implementation Challenge [8]. One is a road network of USA that has about 24 million vertices and 58 million edges. The other is a road network of Western Europe by PTV AG with about 18 million vertices and 42 million edges. In both instances, we use travel times as the length of edges and treat a graph as an undirected graph.

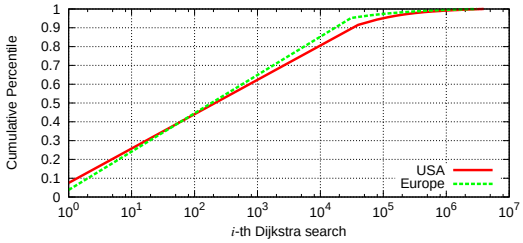
6.2 Performance Comparison First, we compare the proposed method with several previous methods with regard to preprocessing time, space usage and query time (Table 1). The proposed method is set to use the contraction technique for vertices of degree one. We compare the proposed method with the following five previous methods: *contraction hierarchies* (CH) [9], *transit node routing* (TNR) [4], combination of TNR and *arc flags* (TNR+AF) [5], *hub-based labeling* (HL) [1, 2], and *hub label compression* (HLC) [7]. For HL, we used four variants: HL local [1], HL global [1], HL-15 local [2] and HL-∞ global [2]. All these methods

Table 2: Comparison of the performance by the contraction technique. The contraction level indicates the degree of removed vertices.

	USA			Europe		
Contraction level	Preprocessing [h:m]	Space [GB]	Query [ns]	Preprocessing [h:m]	Space [GB]	Query [ns]
0	0:38	19.8	906	0:50	20.2	1080
1	0:29	16.4	941	0:34	14.9	1039
2	0:11	6.4	1793	0:22	8.5	2011
3	0:07	4.1	2970	0:11	4.6	3344



(a) Number of triples added in each pruned Dijkstra search.



(b) Cumulative distribution of the number of triples added in each pruned Dijkstra search.

Figure 2: Effect of pruning.

are also implemented in C++. CH, TNR and TNR+AF were evaluated on a machine with an AMD Opteron 270 processor (2.0 GHz) [5], and HL and HLC were evaluated on a machine with two Intel Xeon X5680 processors (3.33 GHz) [1, 2, 7]. The preprocessing is parallelized for only HL but not parallelized for other methods including our method. Queries are not parallelized for all methods.

Preprocessing time. We would like to emphasize our big improvement in preprocessing time. The preprocessing time for the proposed method is much faster than previous labeling methods. Although it seems that HL-15 local is faster than our method, let us observe that its preprocessing is parallelized to use 12 cores.

Space usage. The space consumption for our method is also smaller than HL. While this may be still not very compact, in the situations when space consumption is severe, we can greatly reduce the space usage with help of the contraction technique, at the cost of a little slower query time, as discussed in Section 6.3.1.

Query time. With regard to query time, although our method is a bit slower than HL, it is still sufficiently fast and around 1 μs on average.

6.3 Analysis

6.3.1 Contraction Technique Table 2 lists the performance of our method with different contraction level. The contraction level denotes the limit of the degree of vertices to remove. Setting the contraction level zero means we do not use the contraction technique at all.

Even without the contraction technique, our preprocessing is much faster than previous labeling methods. However, by applying the contraction technique, preprocessing time becomes even faster. In particular, it takes only about ten minutes with contraction level three. This is because the input graph size is fairly reduced by this contraction. Moreover, the space usage gets smaller than five gigabytes with contraction level three, while the query time is not very slower.

6.3.2 Pruning Figure 2a illustrates the number of triples added to labels in each pruned Dijkstra search and Figure 2b illustrates the cumulative distribution of it. We can confirm big effect of pruning from these figures. From Figure 2a, we observe that the number of triples added to labels in each pruned Dijkstra search decreases dramatically. In consequence, as Figure 2b shows, most triples are added at the beginning.

Figure 3 reports the average number of common paths in labels of two vertices against Dijkstra rank of them. We used labels constructed without the contraction technique and computed the average on

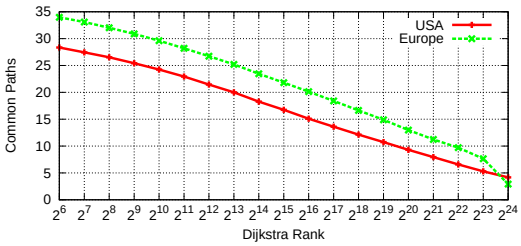


Figure 3: Number of common paths in labels against Dijkstra Rank

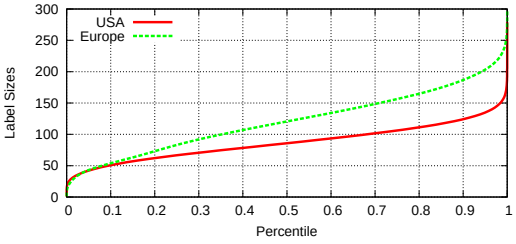


Figure 4: Distribution of the sizes of labels

10,000 pairs of vertices for each rank. We observe that the number of common paths decreases gradually as the two vertices get far. This indicates that our preprocessing algorithm successfully separates input graphs by shortest paths.

6.3.3 Sizes of Labels Figure 4 shows the distribution of the sizes of labels. We constructed labels without the contraction technique. We can confirm that the size of a label for each vertex does not vary much for different vertices, and few vertices have much larger labels than the average. This indicates that the query time for our method is quite stable.

7 Conclusion

In this paper, we proposed a new labeling framework referred to as the "highway-based labeling framework". Highway-based labeling exploits highways and decomposes an input graph into shortest paths. Then, it computes labels that contain the distance to the shortest paths. In addition, we proposed a preprocessing algorithm for the framework named "pruned highway labeling". This algorithm is based on the pruned landmark labeling and can compute labels efficiently. We only conduct a pruned Dijkstra search from each shortest path. Our experimental results show that our proposed method is much faster than previous (and state-of-the-art) labeling methods in the preprocessing time. Moreover, the query time is sufficiently fast as it is also a labeling method.

References

[1] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck. A hub-based labeling algorithm for shortest paths in road networks. In *SEA*, pages 230–241, 2011.

[2] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck. Hierarchical hub labelings for shortest paths. In *ESA*, pages 24–35, 2012.

[3] T. Akiba, Y. Iwata, and Y. Yoshida. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *SIGMOD*, pages 349–360, 2013.

[4] H. Bast, S. Funke, D. Matijevic, P. Sanders, and D. Schultes. In transit to constant time shortest-path queries in road networks. In *ALENEX*, pages 46–59, 2007.

[5] R. Bauer, D. Delling, P. Sanders, D. Schieferdecker, D. Schultes, and D. Wagner. Combining hierarchical and goal-directed speed-up techniques for dijkstra's algorithm. *ACM Journal of Experimental Algorithmics*, 15(2.3):1–31, 2010.

[6] D. Delling, A. V. Goldberg, T. Pajor, and R. F. Werneck. Customizable route planning. In *SEA*, pages 376–387, 2011.

[7] D. Delling, A. V. Goldberg, and R. F. Werneck. Hub label compression. In *SEA*, pages 18–29, 2013.

[8] C. Demetrescu, A. V. Goldberg, and D. S. Johnson. *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, volume 74. AMS, 2009.

[9] R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *WEA*, pages 319–333, 2008.

[10] M. Hilger, E. Köhler, R. H. Möhring, and H. Schilling. Fast point-to-point shortest path computations with arc-flags. *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, 74:41–72, 2009.

[11] T. Ikeda, M. Y. Hsu, H. Imai, S. Nishimura, H. Shimoura, T. Hashimoto, K. Tenmoku, and K. Mitoh. A fast algorithm for finding better routes by ai search techniques. In *VNSI*, pages 291–296, 1994.

[12] M. Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *Journal of the ACM*, 51(6):993–1024, 2004.

[13] Y. Yano, T. Akiba, Y. Iwata, and Y. Yoshida. Fast and scalable reachability queries on graphs by pruned labeling with landmarks and paths. In *CIKM*, pages 1601–1606, 2013.