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

Takuya Akiba, Yoichi Iwata, Ken-ichi Kawarabayashi, Yuki Kawata

Giulio Bazzanti **Niccolò Biondi**

Advanced Algorithms and Graph Mining, A.A. 2018/19

Outline

Scope of the article

- Fast . . .
  - fast systems for answer to users' queries of distance

Scope of the article

- Fast ...
  - fast systems for answer to users' queries of distance
- ...Shortest-path Distance Queries ...
  - a generic query (u-v) asks for the shortest-path distance between u and v in the graph

Scope of the article

- Fast . . .
  - fast systems for answer to users' queries of distance
- . . . Shortest-path Distance Queries . . .
  - a generic query (u-v) asks for the shortest-path distance between u and v in the graph
- . . . on Road Networks . . .

Scope of the article

- Fast . . .
  - fast systems for answer to users' queries of distance
- . . . Shortest-path Distance Queries . . .
  - a generic query (u-v) asks for the shortest-path distance between u and v in the graph
- . . . on Road Networks . . .
- . . . by Pruned Highway Labeling
  - proposed method

Some applications

- Distance queries are used in systems like Google maps,Bing Maps, Yahoo! Maps
- Dataset: road network of USA and of Western Europe

Proposed methods

The authors propose:

- Highway-based labelings
  - new labeling framework
  - data structure and query algorithm

Proposed methods

The authors propose:

- Highway-based labelings
    - new labeling framework
    - data structure and query algorithm
- Pruned highway labeling
    - preprocessing algorithm
    - small labels for the framework
    - based on pruned Dijkstra search

Timeline

- labeling methods
  - pre-computation on input graph
  - labels are used to answer queries without looking the graph
  - fast query time
  - pruned labeling
- shortest path decomposition of the input graph
- highway
  - many shortest paths must pass through highways

Previous work: Fast Exact Shortest-Path Distance Queries on Large Networks by Pruned Landmark Labeling

Here the authors proposed:

- pruned labeling method
  - data structure
  - query algorithm

Previous work: Fast Exact Shortest-Path Distance Queries on Large Networks by Pruned Landmark Labeling

Here the authors proposed:

- pruned labeling method
  - data structure
  - query algorithm
- Pruned Landmark labeling
  - based on pruned BFS search

Labeling Method and Query Definition

- For each vertex v $\in$ V, pre compute a label L(v) which is a set of pairs (u, δ_{uv})
  - u is a vertex reached from v
  - δ_{uv} = d(u,v)

Labeling Method and Query Definition

- For each vertex v $\in$ V, pre compute a label L(v) which is a set of pairs (u, δ_{uv})
  - u is a vertex reached from v
  - δ_{uv} = d(u,v)

Definition (QUERY(s, t, L))

Given the labels L, for an *s*-*t* query, we define:

$$QUERY(s,t,L) = \min\{\delta_{vs} + \delta_{vt} | (v, \delta_{vs}) \in L(s), (v, \delta_{vt}) \in L(t)\}$$

$QUERY(s,t,L) = \infty$ if L(s) and L(t) not share any vertex

Pruned Landmark Labeling

- Conduct *pruned* BFSs from vertices in the order $v_1 \ldots v_n$
- Start with empty index L'_0 and create index L'_k from L'_{k-1}
- Pruning:
  - If $QUERY(v_k, u, L'_{k-1}) \leq \delta$ then we prune
  - Do not add (v_k, δ) to $L'_k(u)$ and do not traverse any edge from vertex u

Pruned Landmark Labeling

- Conduct *pruned* BFSs from vertices in the order $v_1 \dots v_n$
- Start with empty index L'_0 and create index L'_k from L'_{k-1}
- Pruning:
  - If $QUERY(v_k, u, L'_{k-1}) \leq \delta$ then we prune
  - Do not add (v_k, δ) to $L'_k(u)$ and do not traverse any edge from vertex u
- otherwise set $L'_k(u) = L'_{k-1}(u) \bigcup \{(v_k,$ d(v_k,u))$\}$ and traverse all the edge from vertex u
- set $L'_k(u) = L'_{k-1}(u)$ for all vertex u $\in$ V that were not visited in the k-th *pruned* BFS

Contributions

- highway-based labeling framework:
  1. decompose the input graph in shortest path
  2. create a label for each vertex
  3. store the distances from each vertex to the shortest paths
- pruned highway labeling:
  - small labels based on pruned Dijkstra search
- heuristics for good decomposition and techniques for efficient implementation

Proposed Methods

Proposed Methods

Highway-based Labelings

Highway-based Labelings

The framework has the following structure:

- take in input the graph decomposition
- define the label's structure
- describe how to answer for a query
- describe how to store the labels

Graph Decomposition

Definition (Highway decomposition)

A highway decomposition of a given graph G is a family of ordered sets of vertices $\mathcal{P} = \{P_1, P_2, \ldots, P_N\}$ 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 with $i \neq j$
3. $P_1 \cup P_2 \cup \ldots \cup P_N = V$

Labeling

For each vertex $v \in V$ the label, $L(v)$, is a set of triples $(i, d(p_{i,1}, p_{i,j}), d(v, p_{i,j}))$

- i is a index of a path P_i
- $d(p_{i,1}, p_{i,j})$ is the distance from the starting point $p_{i,1}$ of the path to a vertex $p_{i,j}$ on the path
- $d(v, p_{i,j})$ the distance from the vertex v to the vertex $p_{i,j}$

Storing all the triples for all the index i or all the vertices on a path P_i will be expensive. There is a need to reduce the total size of labels (Pruned Highway Labeling).

Query definition

Definition (QUERY(s, t, L))

Given the labels L, for an *s*-*t* query, we define:

$$QUERY(s,t,L) = \min\{d(s,p_{i,j}) + d(p_{i,j},p_{i,k}) + d(p_{i,k},t) \mid (i, d(p_{i,1},p_{i,j}), d(s,p_{i,j})) \in L(s), (i, d(p_{i,1},p_{i,k}), d(t,p_{i,k})) \in L(t)\} \quad (1)$$

$d(p_{i,j}, p_{i,k})$ is not contained in labels *L(s)* and *L(t)* but can be computed in the following way:

$$d(p_{i,j},p_{i,k}) = |d(p_{i,1},p_{i,j}) - d(p_{i,1},p_{i,k})|$$

Computation time

Computation time?

- $\Theta(|L(s)||L(t)|)$

Computation time

Computation time?

- $\Theta(|L(s)||L(t)|)$
- linear time, $O(|L(s)| + |L(t)|)$, by sorting triples with indexes and the distances from the starting point of the path in ascending order. There is the following lemma:

Lemma

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 $(i, d(p_{i,1}, p_{i,l}), d(t, p_{i,l})) \notin L(t)$.

Proposed Methods

Pruned Highway Labeling

Naive Highway Labeling

- $L_0(v) = \emptyset$ for each vertex $v \in V$

Naive Highway Labeling

- $L_0(v) = \emptyset$ for each vertex $v \in V$
- $L_i(v) = L_{i-1}(v) \cup (i, d(p_{i,1}, p_{i,j}), d(v, p_{i,j}))$, where $d(v, p_{i,j})$ is computed with a Dijkstra search from each vertex $p_{i,j}$ of the path P_i

Naive Highway Labeling

- $L_0(v) = \emptyset$ for each vertex $v \in V$
- $L_i(v) = L_{i-1}(v) \cup (i, d(p_{i,1}, p_{i,j}), d(v, p_{i,j}))$, where $d(v, p_{i,j})$ is computed with a Dijkstra search from each vertex $p_{i,j}$ of the path P_i
- obviously we have $L_N = L$ and the following lemma:

Lemma

For any pair of vertices s and t in V, $QUERY(s, t, L) = d(s, t)$.

Naive Highway Labeling

- $L_0(v) = \emptyset$ for each vertex $v \in V$
- $L_i(v) = L_{i-1}(v) \cup (i, d(p_{i,1}, p_{i,j}), d(v, p_{i,j}))$, where $d(v, p_{i,j})$ is computed with a Dijkstra search from each vertex $p_{i,j}$ of the path P_i
- obviously we have $L_N = L$ and the following lemma:

Lemma

For any pair of vertices s and t in V, $QUERY(s, t, L) = d(s, t)$.

- not so efficient

Pruned Highway Labeling

- efficient algorithm for preprocessing

Pruned Highway Labeling

- efficient algorithm for preprocessing
- $L'_0(v) = \emptyset$ for each vertex $v \in V$
- $L'_{i+1}(v) = L'_i(v)$ for each vertex $v \in V$, and for each path P_i

Pruned Highway Labeling

- efficient algorithm for preprocessing
- $L'_0(v) = \emptyset$ for each vertex $v \in V$
- $L'_{i+1}(v) = L'_i(v)$ for each vertex $v \in V$, and for each path P_i
- if $QUERY(v, p_{i,j}, L'_i) \leq \delta$ then the Dijkstra search is pruned
- otherwise, the triple $(i, d(p_{i,1}, p_{i,j}), \delta)$ is added to $L'_i(v)$ and the edges from v is checked

Algorithm 1 Pruned Dijkstra search

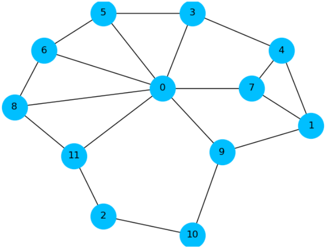
Require: G, P_i, L'_{i-1}
  $Q \leftarrow$ an empty priority queue
  Push $(0, p_{i,j}, p_{i,j})$ onto Q for all $p_{i,j} \in P_i$
  $L'_i(v) \leftarrow L'_{i-1}(v)$ for all $v \in V$
  while Q is not empty **do**
    Pop $(\delta, v, p_{i,j})$ from Q
    if $QUERY(v, p_{i,j}, L'_i) \leq \delta$ **then**
      continue *(prune the search)*
    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 $L'_i(v)$

Algorithm 2 Preprocessing by the pruned highway labeling

Require: G

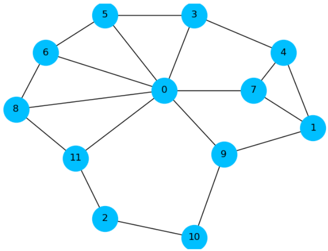
$L'_0(v) \leftarrow \emptyset$ for all $v \in V$
$P \leftarrow$ a highway decomposition of G
$N \leftarrow$ the size of P
for $i = 1$ to N **do**
  $L'_i \leftarrow prunedDijkstraSearch(G, P_i, L'_{i-1})$
end for
return $L'_N = L'$

Example

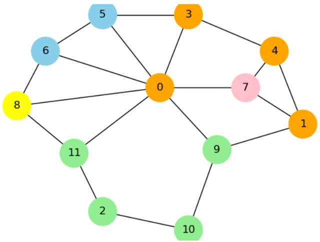


- original

Example

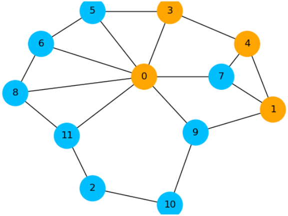


- original



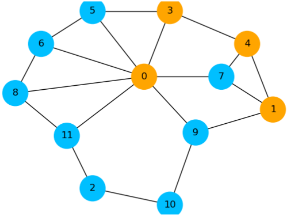
- highway decomposition

Pruned Highway Labeling



- $P_1 = \{0, 3, 4, 1\}$

Pruned Highway Labeling



- $P_1 = \{0, 3, 4, 1\}$

- $L'_1(0) = \{(1, 0, 0)\}$
- $L'_1(1) = \{(1, 2, 0)\}$
- $L'_1(2) = \{(1, 0, 2), (1, 2, 3)\}$
- $L'_1(3) = \{(1, 1, 0)\}$
- $L'_1(4) = \{(1, 2, 0)\}$
- $L'_1(5) = \{(1, 0, 1), (1, 1, 1)\}$
- $L'_1(6) = \{(1, 0, 1)\}$
- $L'_1(7) = \{(1, 0, 1), (1, 2, 1)\}$
- $L'_1(8) = \{(1, 0, 1)\}$
- $L'_1(9) = \{(1, 0, 1), (1, 2, 1)\}$
- $L'_1(10) = \{(1, 0, 2), (1, 2, 2)\}$
- $L'_1(11) = \{(1, 0, 1)\}$

Correctness

- The distance computed by using L' is equal one computed using L

Theorem

For any pair of vertices s and t,

$$QUERY(s, t, L') = QUERY(s, t, L)$$

Correctness

- The distance computed by using L' is equal one computed using L

Theorem

For any pair of vertices s and t,

$$QUERY(s,t,L') = QUERY(s,t,L)$$

- Moreover, a query can be computed correctly using labels from the pruned algorithm

Corollary

For any pair of vertices s and t, $QUERY(s,t,L') = d(s,t)$.

Proof.

Let i the index such that

$$QUERY(s, t, L_{i'}) \neq d(s, t) \quad \forall i' < i$$

and

$$QUERY(s, t, L_i) = d(s, t).$$

$\exists\ p_{i,j}, p_{i,k} \in P_i$ such that

$$d(s, t) = d(s, p_{i,j}) + d(p_{i,j}, p_{i,k}) + d(t, p_{i,k})$$

We choose (j, k) such that no vertices on shortest path between s and $p_{i,j}$ or t and $p_{i,k}$ are in P_i. Suppose that for some $i' < i$, $\exists\ p_{i',j'} \in P_{i'}$ is in a shortest path between s and $p_{i,j} \Rightarrow (i', d(p_{i',1}, p_{i',j'}), d(s, p_{i',j'})) \in L(s)$ and $(i', d(p_{i',1}, p_{i',j'}), d(t, p_{i',j'})) \in L(t)$.

Proof.

Thus, $(i', d(p_{i',1}, p_{i',j'}), d(s, p_{i',j'})) \in L(s)$ and $(i', d(p_{i',1}, p_{i',j'}), d(t, p_{i',j'})) \in L(t) \Rightarrow$

$$QUERY(s, t, L_{i'}) = d(s, t)$$

that is a contradiction to the choice of i.
Moreover, the search from $p_{i,j}$ to s is not pruned, because any path $P_{i'}$ with $i' < i$ contains no vertices on the shortest paths between s and $p_{i,j}$. Thus

$$(i, d(p_{i,1}, p_{i,j}), d(s, p_{i,j})) \in L'(s)$$

$$(i, d(p_{i,1}, p_{i,k}), d(t, p_{i,k})) \in L'(t)$$

As a results, holds that:

$$QUERY(s, t, L') = QUERY(s, t, L)$$

□

Heuristics for Small Labels

Heuristics for Small Labels

Highway Decomposition

Highway Decomposition

- we want to choose a path that *hits* many shortest paths(highway), because this would allow us to prune future searches
- each edge has a speed $(l(u, v)/t(u, v))$

Highway Decomposition

- we want to choose a path that *hits* many shortest paths(highway), because this would allow us to prune future searches
- each edge has a speed $(l(u, v)/t(u, v))$
- vertices are grouped in levels according to the speed of their connected edges
  - faster edges to higher level
  - shortest path is the highest levels
  - few vertices in a level (threshold) $\Rightarrow$ mix the two highest levels

Shortest Path

How to choose the correct path from the highest level?

- a path must be a shortest path between two vertices

Shortest Path

How to choose the correct path from the highest level?

- a path must be a shortest path between two vertices
- compute the shortest path tree from a random root vertex
- pick a path between root and a vertex in the tree
  - many descendants many shortest paths hit the vertex
  - shortest path is chosen iteratively from the root to the child with more descendants

Shortest Path

How to choose the correct path from the highest level?

- a path must be a shortest path between two vertices
- compute the shortest path tree from a random root vertex
- pick a path between root and a vertex in the tree
  - many descendants many shortest paths hit the vertex
  - shortest path is chosen iteratively from the root to the child with more descendants
- not all the nodes are important
- if the difference between the number of descendants of v and one of his child w is small, the vertex v is skipped (with a shortcut edge)

Algorithm 3 Highway Decomposition

Require: G, num_level
for each edge $(u, v) \in E$ compute the edge's speed
add the vertex u, v to a level according to (u, v) speed
for each level **do**
  while level is not empty **do**
    let $v \in level$ the max degree node
    $parent, childhood \leftarrow prim\_MST(v)$
    compute number of descendant of vertex in Prim tree
    create path from v adding $y = \text{argmax}\{descendant(v)\}$
    $highway\_dec \leftarrow highway\_dec \cup shortest\_path$
  end while
end for
return highway_dec

Heuristics for Small Labels

Contraction Technique

Contraction Technique

- for each vertex v such that $deg(v) = 1$ (let w his only child)
  - any shortest path from v must pass from w
  - v isn't contained in shortest path between other vertices
  - $QUERY(v, u, L) = QUERY(w, u, L) + l(v, w)$

Contraction Technique

- for each vertex v such that $deg(v) = 1$ (let w his only child)
  - any shortest path from v must pass from w
  - v isn't contained in shortest path between other vertices
  - $QUERY(v, u, L) = QUERY(w, u, L) + l(v, w)$
- for each vertex v such that $deg(v) > 1$
  - v can be contained in some shortest path between other vertices
  - $QUERY(v, u, L) = \min_{(v,w) \in E} \{QUERY(w, u, L) + l(v, w)\}$
  - larger $deg(v) \Rightarrow$ slower the query time

Experiments

Contributions

- Preprocessing Time
  - Pruned Highway Labeling
- Space Usage
  - with the contraction techniques (less preprocessing time, more query time)
- Query Time
  - faster thanks to the storing of labels
  - in $L(v)$ is saved i and $d(p_{i,1}, p_{i,j})$ separately
  - storing pairs of an index and the number of triples for the index
  - pointer arithmetic and align arrays storing labels to cache line

Results

- Comparison of the performance between Pruned Highway Labeling (PHL) and previous methods

	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

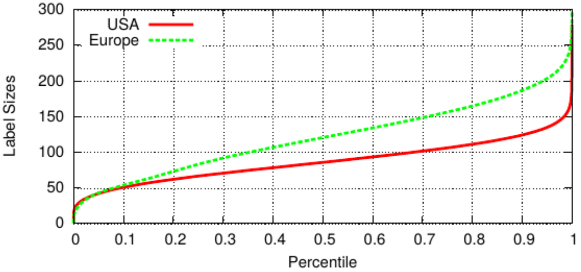
Contraction technique Effects

- Comparison of the performance with the contraction technique

	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

Label Size Distrbution

- different vertices same size of label
- few vertices have larger labels (highway)



Thanks for your attention

Query Time :)