Highway Hierarchies (Dominik Schultes)
Presented by: Andre Rodriguez
Central Idea

- To go from Tallahassee to Gainesville*:
  - Get to the I-10 (8.8 mi)
  - Drive on the I-10 (153 mi)
  - Get to Gainesville (1.8 mi)

- ~94% of the driving is done on the I-10

*According to Google Maps
Central Idea

- This suggests a reasonable approach:
  - To go from A to B:
    - From A, get to the next reasonable highway
    - Drive until we are close enough to B
    - Search for B starting from the highway’s exit
Central Idea

- This approach gives approximate answers

- A variant of this is method is used by most commercial planning systems

- It suggests a way of computing shortest paths faster
Detour – Bidirectional Search

- From S to T
- Search from S
- Search from T (reversed graph)
- Halt when searches meet

Total area decreases by a factor of ~2.67
Central Idea – Suggested Approach

- To go from A to B:
  - Perform a search in a local area around A and around B
  - Search in a (thinner) highway graph*
  - Iterate

* A shortest path preserving graph
Local Area - Concept

- The local area associated with a vertex $v$ is a set of vertices.
- All vertices in such local area are relatively close to $v$.
- For some parameter $H$, the local area must be big enough as to cover the closest $H$ vertices.
- We refer to such local area as neighborhood (of $v$ using $H$) or $N_H(v)$. 
Neighborhood (Local Area) - Definition

Given a graph $G = (V, E)$

Given a vertex $A$

$L \leftarrow$ Sort $V \setminus A$ by their distance from $A$

Let $r_A$ be the distance from $A$ to the $H$-th vertex in $L$

$S \leftarrow [x \in V \text{ if distance from } A \text{ to } x \leq r_A]$

$N_H(A) \leftarrow S$
Neighborhood (Local Area)

In this case $H = 5$
Neighborhood (Local Area) - Implementation

- In practice, to determine the neighborhood of $v$ we do not compute its distance to all other vertices.

- Instead, a Dijkstra is ran from $v$.

- The $H$-th vertex to be popped from the queue determines the radius of $N_H(v)$. 
Highway Network - Definition

- A highway network of a graph $G = (V, E)$ is a graph $G^* = (V^*, E^*)$
  - $V^*$ is a subset of $V$
  - $E^*$ is a subset of $E$
    - $E^*$ consists of all the *highway edges* in $E$
  - $V^*$ consists of all the vertices in $E^*$
Highway Edge – Definition

- $e = (u, v)$ is an edge in the original graph
- $e$ belongs to the shortest path from $s$ to $t$, for some $s$ and $t$
- $e$ is not inside the neighborhood of $s$
- $e$ is not inside the neighborhood of $t$

If all of the above hold, then $e$ is a highway edge
Highway Network

All blue edges and vertices are in the highway network

Search from s and t

When the frontier of the neighborhood is reached continue searching on the highway only
Highway Network - Contraction

- We want to reduce the number of nodes
- If we are on the I-10, we shouldn’t care much about exits nor road segments
- These are low degree vertices that can be bypassed
- (Almost) only the I-10 should belong to the HN
- The structure is preserved by adding shortcuts
To compute the core:

- Remove all bypassed nodes
- Add all shortcut edges
Some terms

- Creating the highway network is also referred to as *edge reduction*

- Computing the core is also referred to as *node reduction*
Highway Hierarchy

- Given a graph $G = (V, E)$
- Given a parameter $H$
- We can iteratively reduce edges and nodes to create a hierarchy
- By introducing shortcut edges the average degree increases
- It increases slowly enough
Highway Hierarchy - Process

- Compute highway edges
- Bypass nodes and introduce shortcuts
- Compute highway edges
- Bypass nodes and introduce shortcuts
- ...

Let $G_0$ be the original graph and $L$ be a parameter.

A highway hierarchy of $L + 1$ levels is given by $L + 1$ graphs: $G_0, \ldots, G_L$.

How is each $G_k$ defined?
- An inductive definition is given
Highway Hierarchy – Definition (base)

- Suppose $G_0 = (V_0, E_0)$ is the original graph
- Define $G'_0 \leftarrow G_0$
Highway Hierarchy – Induction

- For $0 \leq k \leq L$:
  - Let $G_{k+1}$ be the highway network of $G'_k$
  - Let $G'_{k+1}$ be the core of $G_{k+1}$

- So, at each level, we compute the highway network of the previous level’s graph and then we compute its core

- We then pass this to the next level

- Terminate after computing $G'_L$
Highway Network - Computation

- Given $G'_k = (E'_k, V'_k)$
- We want to find $G_{k+1} = (E_{k+1}, V_{k+1})$
- Let $E_{k+1}$ be an empty set of edges
- For each node $s_0$ in $V'_k$:
  - Construct a partial SPDAG* from $s_0$
  - Perform a backward evaluation on all nodes from the SPDAG and decide whether or not to add each edge to $E_{k+1}$

* Shortest Path Directed Acyclic Graph
Highway Network – Computation (SPDAG)

Given $G'_k = (E'_k, V'_k)$

For each $s_0$ in $V'_k$:

1. Mark $s_0$ as *active*
2. Perform a SSSP search from $s_0$
3. When a node is pushed into the queue, it inherits the state of its parent
4. If a node satisfies the *abort condition*, mark it as *passive*
5. Abort the search when all queued nodes are *passive*
SPDAG Abort Condition

• When a node $p$ is popped from the queue consider all SPs from $s_0$ to it

• When $s_1$ (the second node on a SP) and $p$ are very close their neighborhoods will have many nodes in common

• As the search progresses, they will have less and less nodes in common

• When they have less than two nodes in common, abort ($p$ still belongs to the SPDAG)
SPDAG Abort Condition

After a while, all queued nodes will be passive since they will be far enough from the source
Remainder: we were given \( G'_k = (E'_k, V'_k) \)

For each vertex \( p \) a partial SPDAG \( SP(p) \) was computed

Let \( E_{k+1} \) be empty
Highway Network - Evaluation

For each node $s_0$:
- For each edge $e = (u, v)$ on $SP(s_0)$:
  - If the following conditions hold:
    - $e$ belongs to some shortest path between $s_0$ and $p$
    - $u$ is not in the neighborhood of $p$
    - $v$ is not in the neighborhood of $s_0$
  - Then $e$ is added to $E_{k+1}$

Let $V_{k+1}$ be the set of all vertices in $E_{k+1}$

So, from $G'_k$ we have computed $G_{k+1}$

We now need to compute $G'_{k+1}$
Core

- We get $G_{k+1}' = (V_{k+1}', E_{k+1}')$ by computing the core of $G_{k+1}$

- Remainder: we get the core of a graph by removing its bypassed nodes and adding shortcut edges

- How is the core computed?
Core - computation

- We are given $G_{k+1} = (V_{k+1}, E_{k+1})$
- Let $B_{k+1}$ be a stack of all nodes that could be bypassed
- Initially $B_{k+1}$ contains all vertices in $V_{k+1}$
- Until the $B_{k+1}$ is empty:
  - Pop the top node, $u$
  - If $u$ satisfies the bypassability criteria:
    - Add shortcuts to $E_{k+1}$ and erase $u$ from $V_{k+1}$
Core – computation (cont)

- Bypassability Criteria (Heuristic):
  - \#shortcuts ≤ c (deg_{in}(u) + deg_{out}(u))

- Given a node \( u \) and a parameter \( c \), we compare the number of shortcuts introduced by erasing \( u \) and the number of edges we save.

- If the net gain is positive → bypass it (add shortcuts).

- Theorem: if \( c < 2 \), \(|E'_k| = O(|V_k + E_k|)\)
Core – computation (cont)

- After a node $u$ is bypassed, the degrees of adjacent nodes change.
- Therefore, nodes adjacent to $u$ may now be bypassable.
- Reevaluate the criteria for all nodes adjacent to $u$ (that have been popped but not bypassed).
- If they are now bypassable, add them to the stack.
Highway Hierarchy - Contraction

- We now have $(0 \leq k \leq L)$:
  - $G_k = (E_k, V_k)$
  - $G'_k = (E'_k, V'_k)$

- This defines the highway hierarchy
Highway Hierarchy – Some Results

<table>
<thead>
<tr>
<th>reduction type</th>
<th>#nodes</th>
<th>shrink factor</th>
<th>#edges</th>
<th>shrink factor</th>
<th>average degree</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>18029721</td>
<td>44448388</td>
<td></td>
<td></td>
<td>2.5</td>
</tr>
<tr>
<td>node</td>
<td>2739750</td>
<td>6.6</td>
<td>21311324</td>
<td>2.1</td>
<td>7.8</td>
</tr>
<tr>
<td>edge</td>
<td>1672200</td>
<td>1.6</td>
<td>5376800</td>
<td>4.0</td>
<td>3.2</td>
</tr>
<tr>
<td>node</td>
<td>327493</td>
<td>5.1</td>
<td>3766415</td>
<td>1.4</td>
<td>11.5</td>
</tr>
<tr>
<td>edge</td>
<td>270606</td>
<td>1.2</td>
<td>1109315</td>
<td>3.4</td>
<td>4.1</td>
</tr>
<tr>
<td>node</td>
<td>72787</td>
<td>3.7</td>
<td>981297</td>
<td>1.1</td>
<td>13.5</td>
</tr>
<tr>
<td>edge</td>
<td>58008</td>
<td>1.3</td>
<td>248142</td>
<td>4.0</td>
<td>4.3</td>
</tr>
<tr>
<td>node</td>
<td>14791</td>
<td>3.9</td>
<td>212427</td>
<td>1.2</td>
<td>14.4</td>
</tr>
<tr>
<td>edge</td>
<td>11629</td>
<td>1.3</td>
<td>53744</td>
<td>4.0</td>
<td>4.6</td>
</tr>
<tr>
<td>node</td>
<td>2941</td>
<td>4.0</td>
<td>46632</td>
<td>1.2</td>
<td>15.9</td>
</tr>
<tr>
<td>edge</td>
<td>2452</td>
<td>1.2</td>
<td>12340</td>
<td>3.8</td>
<td>5.0</td>
</tr>
<tr>
<td>node</td>
<td>647</td>
<td>3.8</td>
<td>10844</td>
<td>1.1</td>
<td>16.8</td>
</tr>
<tr>
<td>edge</td>
<td>569</td>
<td>1.1</td>
<td>3076</td>
<td>3.5</td>
<td>5.4</td>
</tr>
<tr>
<td>node</td>
<td>163</td>
<td>3.5</td>
<td>2808</td>
<td>1.1</td>
<td>17.2</td>
</tr>
<tr>
<td>edge</td>
<td>160</td>
<td>1.0</td>
<td>798</td>
<td>3.5</td>
<td>5.0</td>
</tr>
<tr>
<td>node</td>
<td>31</td>
<td>5.2</td>
<td>574</td>
<td>1.4</td>
<td>18.5</td>
</tr>
</tbody>
</table>

Queries on each level will use a reduced search space
Now we have a hierarchy of graphs

How do we retrieve a shortest path?

- A variation of bidirectional searching is used (I will talk about the forward search only since backward is similar)

Definition: the level of an edge is the highest level in the hierarchy in which the edge appears
Query – From s to t

- For each vertex u keep three values
  - $d(u) \leftarrow$ distance from the source
  - $l(u) \leftarrow$ level of the u in the search
  - $g(u) \leftarrow$ gap to the next applicable neighborhood border
    - shortest distance from this node to the closest applicable border
Query – From s to t

- Initialization:
  - $d(s) \leftarrow 0$
  - $l(s) \leftarrow 0$
  - $g(s) \leftarrow r_s$
    - $r_s$ is the radius of the neighborhood of s

- A local search in the neighborhood of s is performed
Query – From s to t

- A local search from s is performed
- When a node v with parent u is popped, set its gap value to $g(v) = g(u) - w((u, v))$
- As long as we stay on the same level there is nothing new. Otherwise …
Suppose a node $v$ with parent $u$ is popped and $(u, v)$ crosses the neighborhood.

- In other words, $w((u, v)) \geq g(u)$

If the level of the edge is less than the current level, the edge is not relaxed (speedup, first restriction)

Otherwise, the edge is relaxed:

- $l(v) \leftarrow$ new search level $k$
- $g(v) \leftarrow$ radius of $N(v)$ on level $k$
  - Since we are at the border of the neighborhood
If the entrance point of level $k$ does not belong to level-$k$’s core:

- Continue by using bypassed nodes ($V_k$) until the core is reached
  - That is, when we reach a node in $V'_k$

- Therefore, once the core is reached we forget about bypassed nodes (speedup, second restriction)
Query – From \( s \) to \( t \)

\textit{input}: source node \( s \) and target node \( t \)
\textit{output}: distance \( d(s, t) \)

\begin{verbatim}
\begin{algorithm}
\begin{algorithmic}
\State \( d' := \infty \);
\State insert(\( Q, s, (0, 0, r_0^-(s)) \)); insert(\( Q, t, (0, 0, r_0^-(t)) \));
\While {\( (Q \cup \overrightarrow{Q}) \neq \emptyset \)} do \{ \\
\hspace{1em} \text{select direction} \in \{ \rightarrow, \leftarrow \} \text{ such that } \overrightarrow{Q} \neq \emptyset ;
\hspace{1em} u := \text{deleteMin}(\overrightarrow{Q});
\hspace{1em} \text{if } u \text{ has been settled from both directions then } \\
\hspace{2em} d' := \min(d', \overrightarrow{\delta} (u) + \overrightarrow{\delta} (u));
\hspace{1em} \text{if } \text{gap}(u) \neq \infty \text{ then } \text{gap}' := \text{gap}(u) \text{ else } \text{gap}' := r_{\ell(u)}^{\leftarrow}(u); \\
\hspace{1em} \text{foreach } e = (u, v) \in E \text{ do } \{ \\
\hspace{2em} \text{for } (\ell := \ell(u), \text{gap} := \text{gap}'); \text{ w}(e) > \text{gap}; \\
\hspace{3em} \ell++, \text{gap} := r_{\ell(u)}^{\leftarrow}(u)); \quad \text{// go “upwards”}
\hspace{2em} \text{if } \ell(e) < \ell \text{ then continue; } \quad \text{// Restriction 1}
\hspace{2em} \text{if } u \in V_f' \land v \in B_{\ell} \text{ then continue; } \text{// Restriction 2}
\hspace{2em} k := (\delta(u) + w(e), \ell, \text{gap} - w(e));
\hspace{2em} \text{if } v \text{ has been reached then decreaseKey}(Q, v, k); \\
\hspace{2em} \text{else insert}(Q, v, k);
\hspace{1em} \}\}
\end{algorithmic}
\end{algorithm}
\end{verbatim}

\end{verbatim}

\textbf{Differences:}

- **4**: correctness does not depend on direction chosen but running time does

- **7**: entrance point does not belong to the core at the current level (we are on bypassed nodes)

- **9**: it might be necessary to go upwards more than one level in a single step
Query – From s to t

- **Red** nodes: Level 0
- **Blue** nodes: Level 1
- **Green** nodes: Level 2

- **Dark** shades: core nodes
- Light shades: Bypassed nodes
Query – From s to t – *path*

- The distance from s to t has been computed
- What about the actual path?
- In the search, each node stores a pointer to its parent
- Problems:
  - Introduced shortcuts need to be expanded so that the path is from the original graph
Query – From s to t – path

- How is a shortcut transformed back to its original form?

  - Let \((u, v)\) be one of these shortcuts on \(G'_k\)
    - \(G'_k\) is the graph with shortcuts (the core)

  - Perform a search from \(u\) to \(v\) on \(G_k\) and find a path from \(u\) to \(v\) of the same length
    - \(G_k\) is the graph that is compressed to find the core (so here we must find such a path)

  - Repeat this recursively since the shortcut could have been introduced at a much earlier level
An edge \((u, v)\) in \(E_k'\) (the core of the previous level) is added to \(E_{k+1}\) if \((u, v)\) belongs to some shortest path \(P = [s, \ldots, u, v, \ldots t]\) and:

- \(v\) does not belong to the neighborhood of \(s\)
- \(u\) does not belong to the neighborhood of \(t\)

True by construction
Theorems – (II)

• The query gives a correct shortest path

• Difficult proof:
  ◦ Potentially, there are many correct shortest paths
  ◦ Other algorithms assume uniqueness. This cannot be done here since road networks are inherently ambiguous and shortcuts introduce even more ambiguity
  ◦ We give an outline of the proof
Theorems – Query – Outline

1. Show that the algorithm terminates
2. Deal with the special case that no path from the source to the target exists
3. Define
   i. Contracted path: sub-paths in the original graph are replaced by shortcuts
   ii. Expanded path: shortcuts in the given graph are replaced by the original edges
4. Define:
   i. Last neighbor: last node before leaving a neighborhood
   ii. First core node: first node when entering a neighborhood
5. The definition of *last neighbor* and *first core node* lead to a *unidirectional labeling* of a given path.

6. Apply a forward labeling and a backward labeling to define:
   i. Meeting level: the level at which both searches meet
   ii. Meeting point: the node at which both searches meet
Theorems – Query – Outline

7. Distinguish between two cases:
   i. Searches meet inside some core
   ii. Searches meet in a component of bypassed nodes

8. Define *highways path* to be a path that complies with all restrictions of the query algorithm
   • In other words, highway paths are defined to be all the paths expanded by the query
Theorems – Query – Outline

9. Use these definitions and some lemmas to show that the algorithm is correct
   ◦ Show that at any point the query is in some valid state consisting of a shortest s-t-path that is broken in three pieces by some vertices. These parts of the path consist of:
     • Edges in the forward search
     • Edges in the middle, contracted
     • Edges in the backward search
   ◦ Show the first and third parts are settled with the correct distance values
## Results – Speedups

<table>
<thead>
<tr>
<th>W. Europe (PTV)</th>
<th>USA/CAN (PTV)</th>
</tr>
</thead>
<tbody>
<tr>
<td>18 029 721</td>
<td>18 741 705</td>
</tr>
<tr>
<td>42 199 587</td>
<td>47 244 849</td>
</tr>
<tr>
<td>15 [161]$^3$</td>
<td>#nodes</td>
</tr>
<tr>
<td>0.76 [7.38]$^3$</td>
<td>#directed edges</td>
</tr>
<tr>
<td>8 320</td>
<td>construction [min]</td>
</tr>
<tr>
<td></td>
<td>search time [ms]</td>
</tr>
<tr>
<td></td>
<td>speedup (↔ DIJKSTRA)</td>
</tr>
<tr>
<td></td>
<td>7 232</td>
</tr>
</tbody>
</table>
References
