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

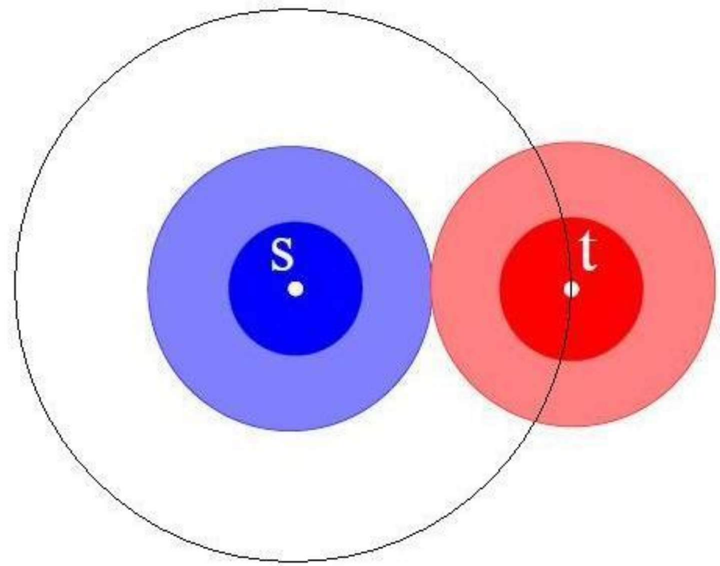
- This suggests a reasonable approach:
  - To go from *A* to *B*:
    - From *A*, get to the next reasonable highway
    - Drive until we are close enough to *B*
    - Search for *B* starting from the highway's exit

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

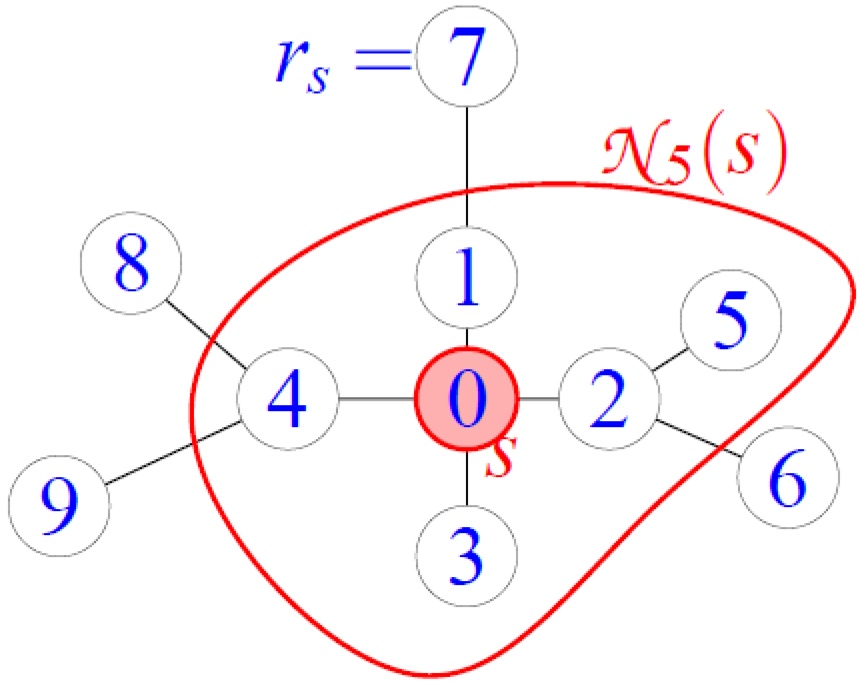
$L \leftarrow$ Sort $V \setminus A$ by their distance from A

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

$S \leftarrow$ [x in V if distance from A 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

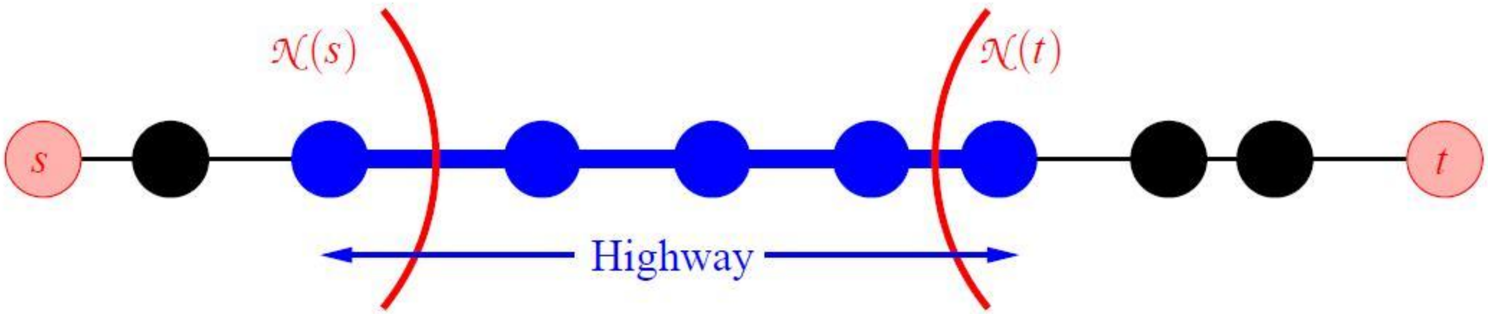
- A highway network of a graph $G = (V, E)$ is a graph $G^* = (V^*, E^*)$
- V^* is a subset of V
- E^* is a subset of E
- E^* consists of all the *highway edges* in E
- V^* consists of all the vertices in E^*

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

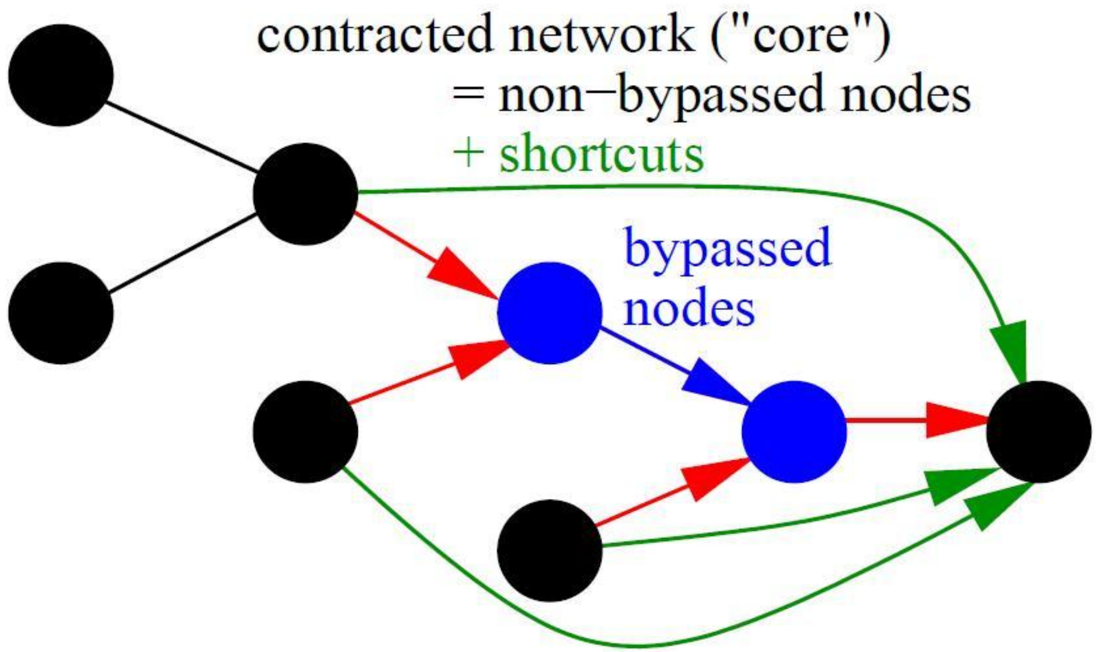
Search from *s* and *t*

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*

Highway Network - Contraction



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
- …

Highway Hierarchy - Definition

- 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 $0 <= k <= L$:
  - 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 $\mathrm{V'}_{\mathrm{k}}$:

  - 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:

Mark s_0 as *active*

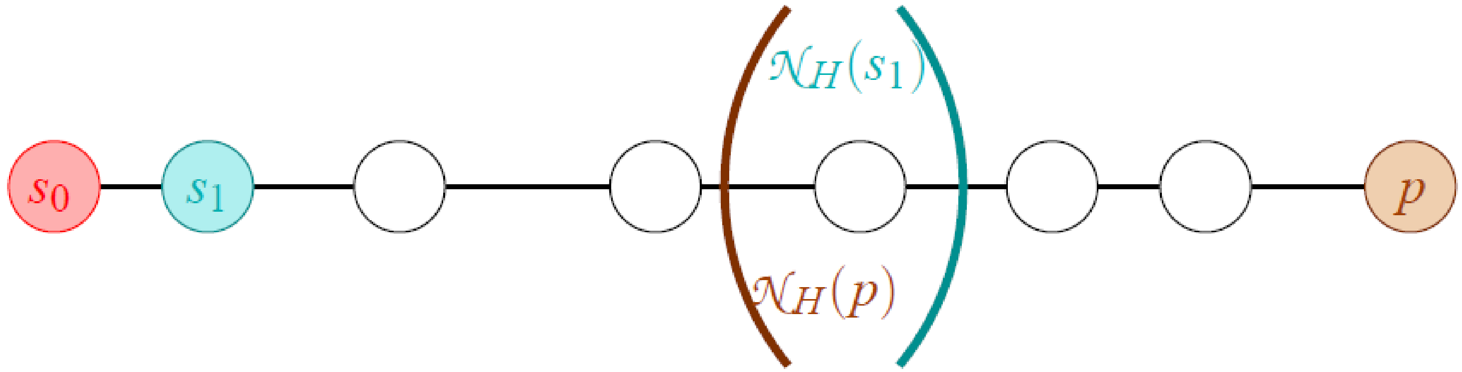
Perform a SSSP search from s_0

When a node is pushed into the queue, it inherits the state of its parent

If a node satisfies the *abort condition*, mark it as *passive*

Abort the search when all queued nodes are *passive*

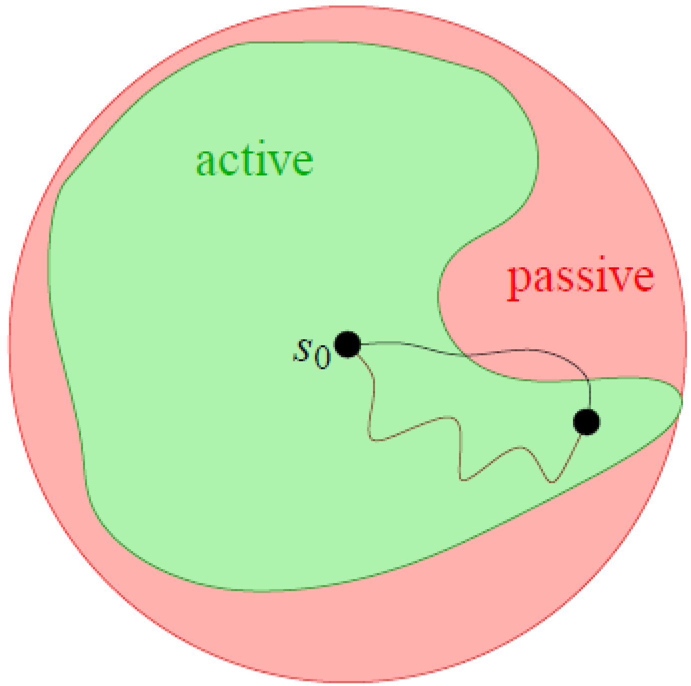
SPDAG Abort Condition



$$|\mathcal{N}_H(s_1) \cap \mathcal{N}_H(p)| \leq 1$$

- 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

Highway Network - Evaluation

- 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

reduction type	#nodes	shrink factor	#edges	shrink factor	average degree
	18 029 721		44 448 388		2.5
node	2 739 750	6.6	21 311 324	2.1	7.8
edge	1 672 200	1.6	5 376 800	4.0	3.2
node	327 493	5.1	3 766 415	1.4	11.5
edge	270 606	1.2	1 109 315	3.4	4.1
node	72 787	3.7	981 297	1.1	13.5
edge	58 008	1.3	248 142	4.0	4.3
node	14 791	3.9	212 427	1.2	14.4
edge	11 629	1.3	53 744	4.0	4.6
node	2 941	4.0	46 632	1.2	15.9
edge	2 452	1.2	12 340	3.8	5.0
node	647	3.8	10 844	1.1	16.8
edge	569	1.1	3 076	3.5	5.4
node	163	3.5	2 808	1.1	17.2
edge	160	1.0	798	3.5	5.0
node	31	5.2	574	1.4	18.5

Queries on each level will use a reduced search space

Highway Hierarchy – Query

- 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

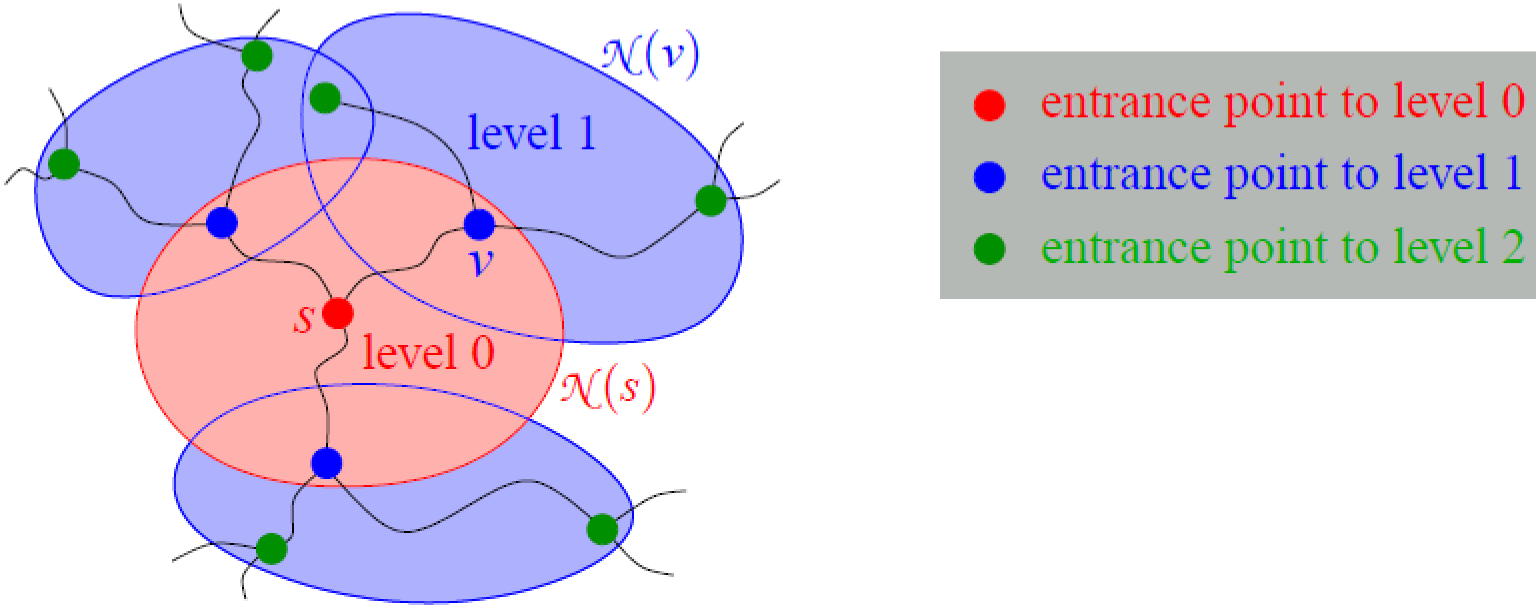
Query – From s to t

- 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 …

Query – From s to t

- 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

Query – From s to t



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

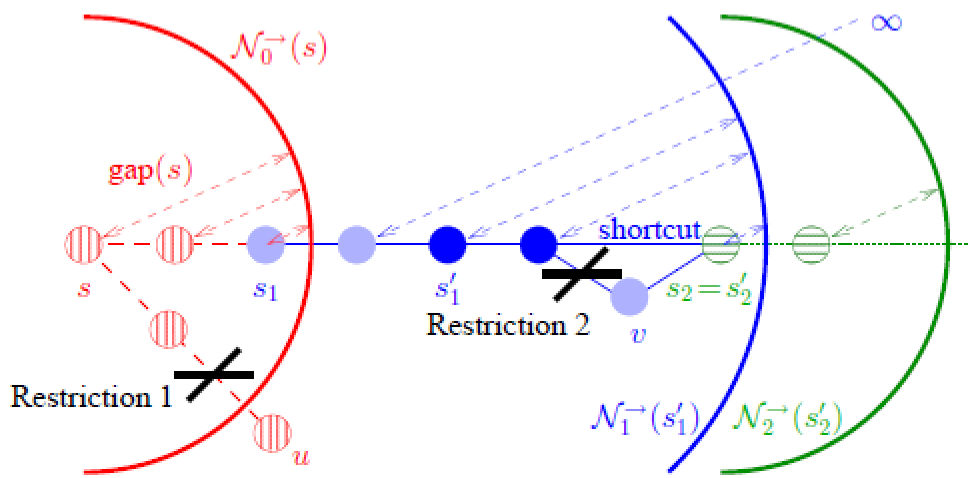
input: source node s and target node t
output: distance $d(s,t)$

1 $d' := \infty$;
2 insert($\overrightarrow{Q}$, s, $(0, \underline{0, r_0^{\rightarrow}(s)})$); insert($\overleftarrow{Q}$, t, $(0, \underline{0, r_0^{\leftarrow}(t)})$);
3 **while** $(\overrightarrow{Q} \cup \overleftarrow{Q} \neq \emptyset)$ **do** {
4 select direction $\leftrightarrows \in \{\rightarrow, \leftarrow\}$ such that $\overset{\leftrightarrows}{Q} \neq \emptyset$;
5 $u :=$ deleteMin($\overset{\leftrightarrows}{Q}$);
6 **if** u has been settled from both directions **then** $d' := \min(d', \overrightarrow{\delta}(u) + \overleftarrow{\delta}(u))$;
7 **if** gap$(u) \neq \infty$ **then** gap$'$:= gap(u) **else** gap$'$:= $r_{\ell(u)}^{\leftrightarrows}(u)$;
8 **foreach** $e = (u, v) \in \overset{\leftrightarrows}{E}$ **do** {
9 **for** ($\ell := \ell(u)$, gap := gap$'$; $w(e) >$ gap; ℓ++, gap := $r_{\ell}^{\leftrightarrows}(u)$); // *go "upwards"*
10 **if** $\ell(e) < \ell$ **then continue**; // *Restriction 1*
11 **if** $u \in V'_{\ell} \wedge v \in B_{\ell}$ **then continue**; // *Restriction 2*
12 $k := (\delta(u) + w(e), \underline{\ell, \text{gap} - w(e)})$;
13 **if** v has been reached **then** decreaseKey($\overset{\leftrightarrows}{Q}$, v, k); **else** insert($\overset{\leftrightarrows}{Q}$, v, k);
14 }
15 }
16 **return** d';

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

Theorems – (1)

- 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
   1. Contracted path: sub-paths in the original graph are replaced by shortcuts
   2. Expanded path: shortcuts in the given graph are replaced by the original edges
4. Define:
   1. Last neighbor: last node before leaving a neighborhood
   2. First core node: first node when entering a neighborhood

Theorems – Query – Outline

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

W. Europe (PTV)			USA/CAN (PTV)
18 029 721		#nodes	18 741 705
42 199 587		#directed edges	47 244 849
15	[161][3]	construction [min]	20
0.76	[7.38][3]	search time [ms]	0.90
8 320		speedup ($\leftrightarrow$ DIJKSTRA)	7 232

References

- [1] Schultes, D., *Route Planning in Road Networks*. Doctoral Dissertation. 2008
- [2] Sanders, P., Schultes, D., *Engineering Highway Hierarchies*. Master's Thesis Presentation. 2006 (most images are from here)