Contraction Hierarchies

Ferienakademie im Sarntal — Course 2
Distance Problems: Theory and Praxis

Mykola Protsenko

Institut für Informatik
FAU Erlangen-Nürnberg

28. September 2010

Friedrich-Alexander-Universität Erlangen-Nürnberg

Technische Universität München

Universität Stuttgart

Outline

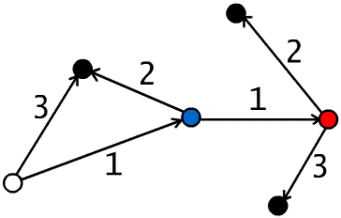
Contraction hierarchies

Another *hierarchical* approach.

- Preprocessing:
    - Nodes are numbered according to their *'importance'*
    - Hierarchy: *contract* the nodes in this order
    - To preserve *shortest paths* shortcuts are added
- Queries: try to avoid less important nodes (use shortcuts)
    - Modified bidirectional Dijkstra
    - Forward search: only edges in ASC importance
    - Backward search: only edges in DESC importance

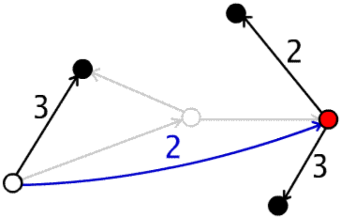
Importance

Which node is more important?



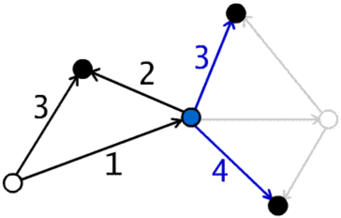
Importance

Contracting blue node



Importance

Contracting red node



Node ordering

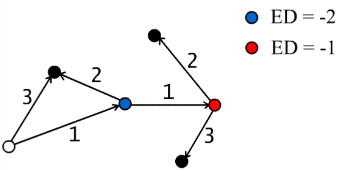
- *priority queue*, minimum element is to be contracted next
- *priority* = the "importance" of a node = **linear combination of several terms**
- difficulty: contraction of a node may affect the priorities of others

Techniques to keep priority up-to-date

- *lazy update:*
  - before contracting `v` update its priority
  - if new priority of `v` is greater than priority of the second largest element `v'`: reinsert `v`
  - repeat until consistent minimum found
- recompute priority of the neighbors of the contracted node
- periodically recompute all priorities

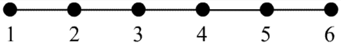
Parameters of the priority function

- **Edge difference**:
  - number of shortcuts needed - number of incident edges
  - the most important term
  - exact computation of the number of shortcuts may be expensive
  - ⇒ search with limited number of hops



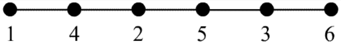
Parameters of the priority function

Example: bad node ordering



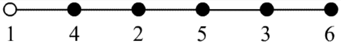
Parameters of the priority function

Example: good node ordering



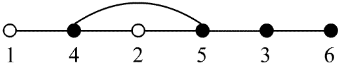
Parameters of the priority function

Example: good node ordering



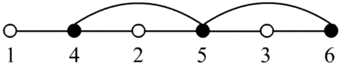
Parameters of the priority function

Example: good node ordering



Parameters of the priority function

Example: good node ordering

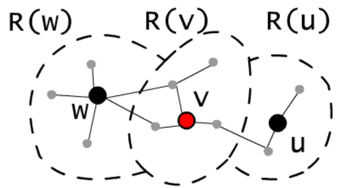


Parameters of the priority function

- **Uniformity**: Contract nodes everywhere in the graph in a uniform way
  - *Deleted Neighbors:* count already contracted neighbors
  - *Voronoi Regions:* $\sqrt{|R|}$
    - $R(v) := \{u | d(v,u) < d(w,u) \; \forall w \in E\}$
    - neighbors of contracted node 'eat up' its *Voronoi region*

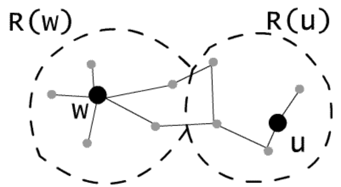
Parameters of the priority function

Voronoi Regions:



Parameters of the priority function

Voronoi Regions:



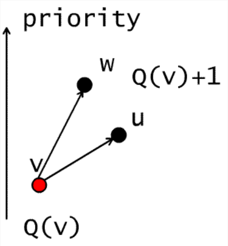
Parameters of the priority function

- **Cost of contraction**: the cost of making a decision, if a shortcut is needed

Parameters of the priority function

- **Cost of queries**: how contracting affects the size of query search space
    - estimate `Q(v)`, the upper bound of number of hops of a path $\langle s, ..., v \rangle$
    - initially: $Q(v) = 0$
    - when `v` is contracted, for each neighbor `u`:

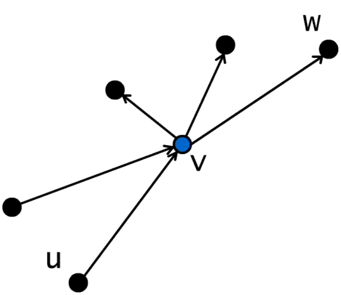
$$Q(u) := max(Q(u), Q(v) + 1)$$



Contraction

- Given: overlay graph $G' = (V', E')$
- v is the next node to contract

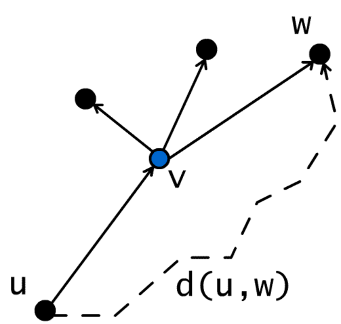
Important: add shortcuts to replace unique shortest paths, going through v



Shortcuts

For $\forall u \in V'$ with $(u, v) \in E'$ and $\forall w \in V'$ with $(v, w) \in E'$:

- search for a shortest distance $d(u, w)$ ignoring v
- if $d(u, w) > c(u, v) + c(v, w)$ - shortcut is needed



Shortcuts

To find out, if shortcut is really needed:

- start forward shortest-distance search from every source u (Dijkstra)
- exact shortest distance search can be expensive ⇒ restrict the maximum number of hops:
  - small hop limit ⇒ fast contraction, but possibly unneeded shortcuts...
  - large hop limit ⇒ slower contraction, but more sparse graph, better query time...

Queries

Split the contraction hierarchy `CH(V,E)` (original nodes, original edges + shortcuts):

- *upward graph* $G_\uparrow := (V, E_\uparrow)$ with $E_\uparrow := \{(u, v) \in E : u < v\}$
- *downward graph* $G_\downarrow := (V, E_\downarrow)$ with $E_\downarrow := \{(u, v) \in E : u > v\}$

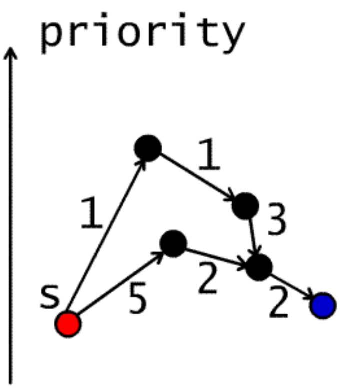
Queries

Modified bidirectional Dijkstra:

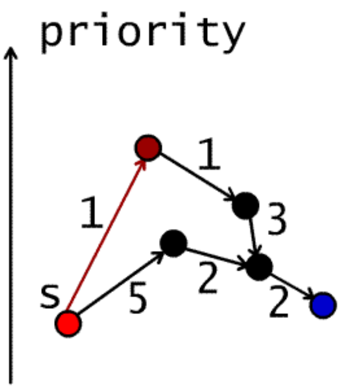
- forward search in $G_{\uparrow}$
- backward search in $G_{\downarrow}$
- alternate both searches

Search can not be stopped, if **one** node is settled in both directions!

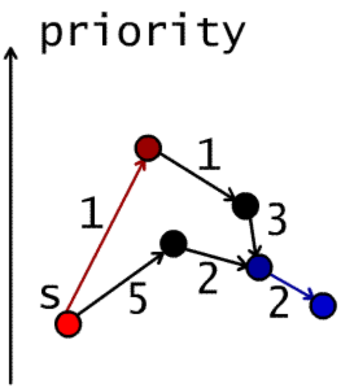
Bidirectional Search



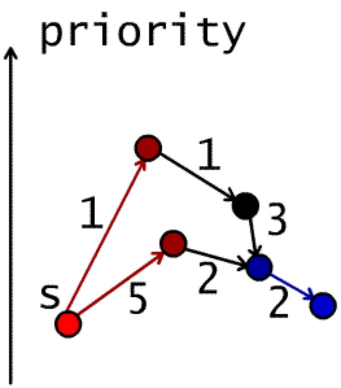
Bidirectional Search



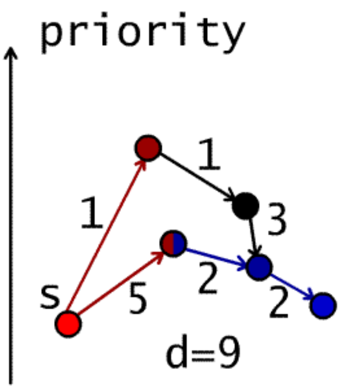
Bidirectional Search



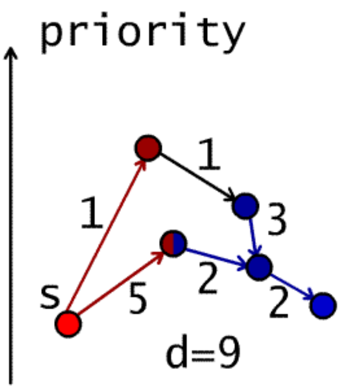
Bidirectional Search



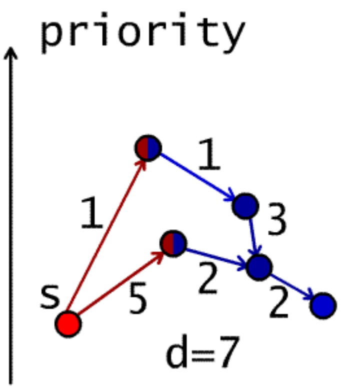
Bidirectional Search



Bidirectional Search



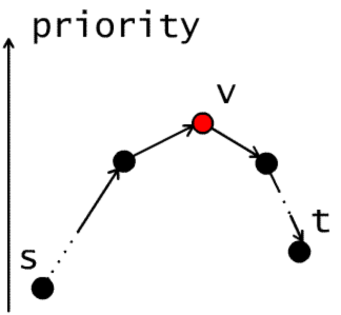
Bidirectional Search



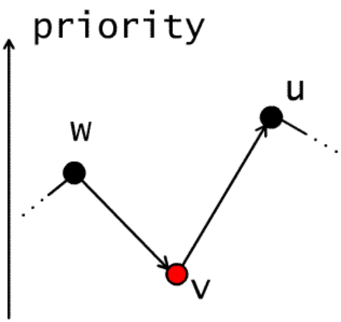
Lemma. $d(s,t) = min\{d(s,v) + d(v,t) : \text{v is settled in both searches}\}$

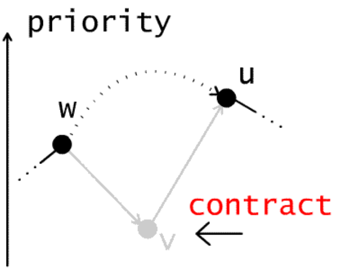
$\Leftrightarrow \exists P = \langle s, ..v, ..t \rangle$ - shortest path with:

- v - the node with highest priority in P
- $\langle s, ..v \rangle$ - ASC priority
- $\langle v, ..t \rangle$ - DESC priority



Proof (contradiction). Suppose:





$\Rightarrow$ contradiction.

Shortest distance vs. shortest paths

If only **shortest distance** needed:

- store edge (u, v) only in $min\{u, v\}$
- $\Rightarrow$ reduces space consumption

To find a **shortest path**:

- each shortcut `(u,w)` bypasses exactly one node `v`
- $\Rightarrow$ store `v` together with the shortcut
- unpack paths recursively

Experiments

- road network of Western Europe: 18 M nodes, 42 M edges
- different variants of CH:
  - E = edge difference
  - D = deleted neighbors
  - S = search space size
  - V = $\sqrt{\text{Voronoi region size}}$
  - Q = upper bound on edges in search paths

  - L = limit search space on weight calculation
  - W = relative betweenness

  - digits: hop limit

method	node ordering [s]	hierarchy construction [s]	query [μs]
E	13010	1739	670
ED	7746	1062	183
ES	5355	123	245
ED5	634	98	224
EDS5	652	99	213
EDS1235	**545**	**57**	223
EDSQ1235	591	64	211
EDSQL	1648	199	173
EVSQL	1627	170	159
EDSQWL	1629	199	163
EVSQWL	1734	180	**154**
HNR	594	203	802

Conclusion

CHs are simple and efficient

- can be used for dynamic weights
- as base for other routing methods: preprocessing in Transit-Node Routing

Thank you!