Efficient Route Planning

SS 2011

Lecture 6, Friday June 24th, 2011

(Highway & Contraction Hierarchies)

Prof. Dr. Hannah Bast
Chair of Algorithms and Data Structures
Department of Computer Science
University of Freiburg

Overview of this lecture

- Exercise Sheet 4 (Maps API etc.)
  - Your web applications
  - Your feedback
- Today: two new algorithms
  - Both based on the inherent hierarchy in road networks
  - Highway Hierarchies (from 2005): intuitive and simple, but quite complex to implement in practice (even the basics)
  - Contraction Hierarchies (from 2008): similar idea, but simpler (in its basic form) and faster
  - I will given you the idea and an outline for HHs and the details, including correctness proofs, for CHs
  - Exercise Sheet for this week: implement Contraction Hierarchies and test its performance …

Feedback for Exercise Sheet **4** (Maps API etc.)

- Summary / excerpts

  last update 24.6 01:15

  - Interesting exercise sheet
  - This time, the amount of work was ok
  - Technologies involved were well explained
  - Web applications are fun
  - Please leave time to improve code from previous exercises
  - Exercise was boring and unnecessary, the exercises on the algorithms were much more interesting

A* with Landmarks — Reprise

- Note the following optimization (thanks, Robin)
  - We used the following lower bound on dist(u, v)

    dist(u, ℓ) ≤ dist(u, v) + dist(v, ℓ)

    hence dist(u, ℓ) – dist(v, ℓ) ≤ dist(u, v)
  - Similarly, using distances **from** ℓ, we get

    dist(ℓ, v) ≤ dist(ℓ, u) + dist(u, v)

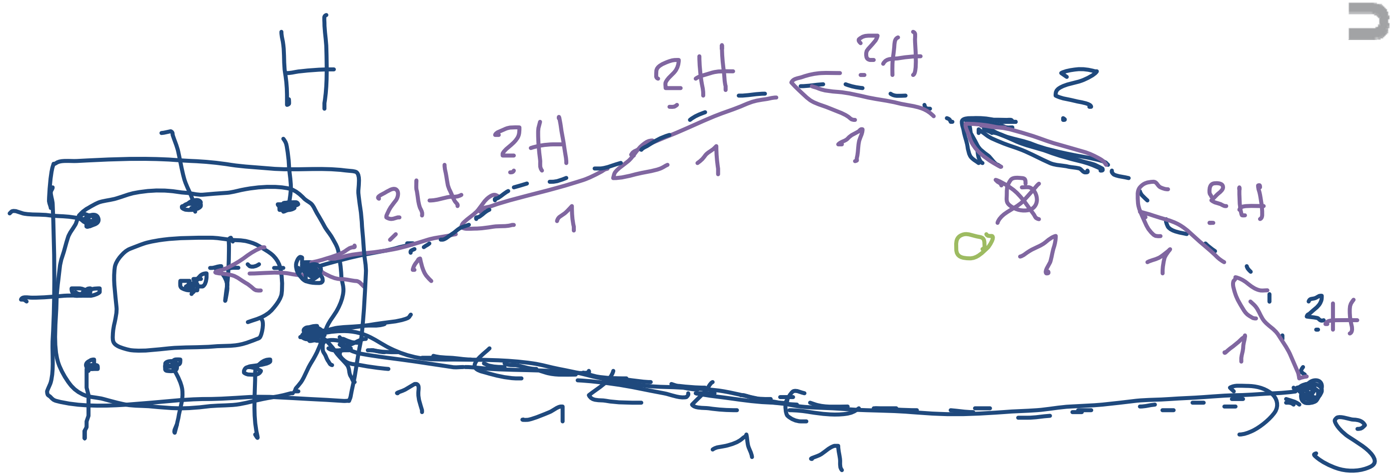
    hence dist(ℓ, v) – dist(ℓ, u) ≤ dist(u, v)
  - For us, dist(u, ℓ) = dist(ℓ, u) and dist(v, ℓ) = dist(ℓ, v)

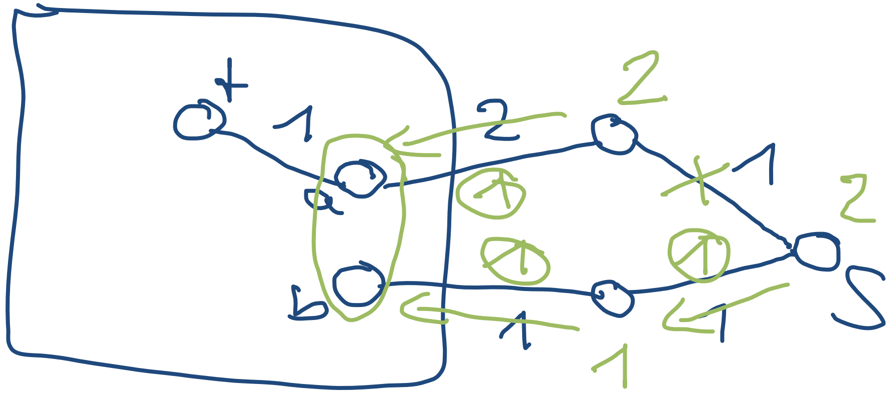
    hence |dist(u, ℓ) – dist(v, ℓ)| ≤ dist(u, v)
  - According to Robin, factor 3-4 smaller search space

Arc flags — Reprise

- The following had to be done for Exercise Sheet 3
  - Compute rectangular regions
    - KD-tree was quite some extra work, but voluntary
  - Compute boundary nodes for each region
  - Full Dijkstra computation from each boundary node
  - Maintain parent pointers during Dijkstra computation and backtrack them to get arcs on shortest paths
  - Set arc flags accordingly; use a vector<bool> !
  - At query time, outside target region, check arc flags
  - **Writing tests is more than half of the work**
  - But for good reason, without tests you are **doomed**

Arc flags — Reprise





Highway Hierarchies 1/5

- Basic idea
  - Road networks have an inherent hierarchy of more and more important roads: residential roads, national roads, motorways, etc.
  - Intuitively, far away from source and target it suffices to use only more important roads
  - Let's see some examples on Google Maps ...
  - The question is: how to make "far away" and "important" precise, so that it's not just a heuristic with approximate results, but an exact algorithm

Highway Hierarchies 2/5

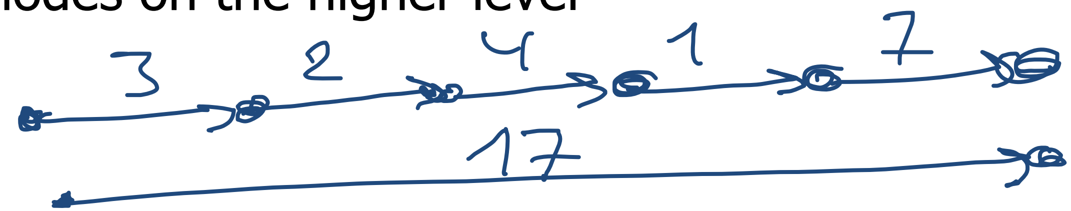
- Precomputation (for undirected graphs)
  - Don't rely on road labels, but compute a level for each arc
  - Initially all arcs have level 0
  - Consider the graph of all arcs with level ℓ
  - Let r_ℓ be a suitably chosen neighbourhood radius for that level
  - Neighbourhood of a node $N(u) = \{v : dist(u, v) \leq r_\ell\}$
  - Raise an arc (u, v) to level $\ell + 1$ if and only if
    - there exists a shortest path $(s, \ldots, u, v, \ldots, t)$ such that v is **not** in $N(s)$ and u is **not** in $N(t)$
    - Intuitively: (u, v) is in the middle of a long shortest path

Highway Hierarchies 3/5

- Query algorithm (high-level description)
  - Bidirectional Dijkstra from source and target
  - For each node, maintain not only the tentative distance, but also the current level and the distance gap to the next applicable neighbourhood border
  - Initially (at source and target), the level is 0 and the gap is the neighbourhood radius r_0
  - When relaxing an arc from a node with level ℓ (= try to improve tentative distance of the head node of the arc)
    - consider only arcs of current level or higher
    - if cost of arc is > gap, increase level for adjacent node

Highway Hierarchies 4/5

- Optimizations
  - Without further ado, there will be long chains of degree-2 nodes on the higher level



  - Those can and should be **contracted**
  - This contraction can be generalized to higher degrees
    - will be explained in detail for contraction hierarchies
  - Important for performance, but makes query algorithm even more complicated and error-prone than it already is

Highway Hierarchies 5/5

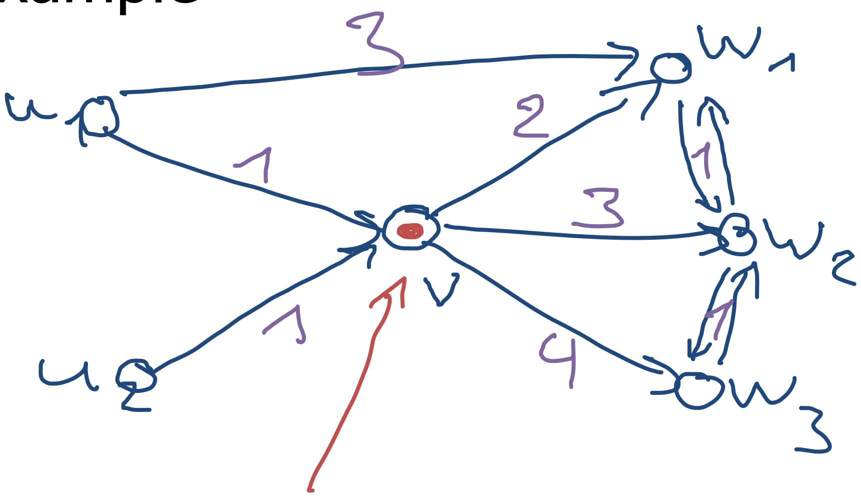
- Performance
  - Can preprocess the road network of Western Europe in less than 1 hour, with small space overhead
  - Achieves average query times around 1 millisecond
- Problems
  - Non-trivial to get the levels right
  - Complicated rule for when to switch to the next level in the Dijkstra computation at query time
    - especially when the various optimizations are aded
  - Easy to make mistakes, and then optimal paths are lost

Contraction Hierarchies

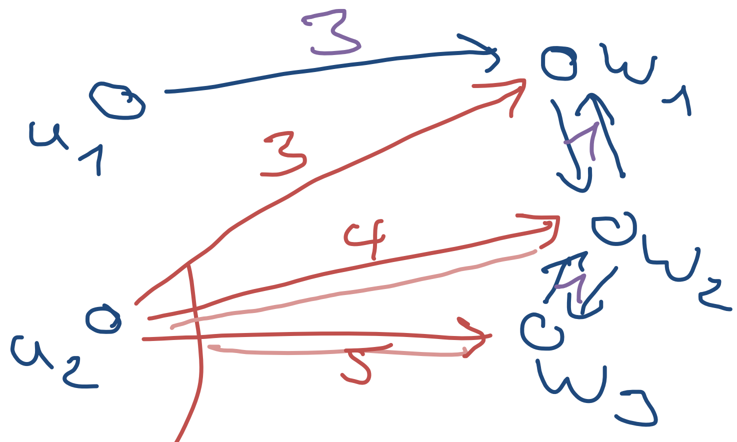
- Precomputation of a given graph $G = (V, E)$
  - Consider an arbitrary ordering of the nodes
    - the algorithm that follows is correct for any order, but it is more efficient for some orders than for others
  - Process the nodes in this order, let v be the next node
  - Then v will be **contracted** as follows
    - Let $\{u_1,\ldots,u_l\}$ be the incoming arcs, i.e. $(u_i, v) \in E$
    - Let $\{w_1,\ldots,w_k\}$ be the outgoing arcs, i.e. $(v, w_j) \in E$
    - For each pair $\{u_i, w_j\}$, if (u_i, v, w_j) is the **only** shortest path from u_i to w_j, add the **shortcut** arc (u_i, w_j)
    - Then **remove** v and its adjacent arcs from the graph

Contraction Hierarchies

- Example



contract v



not needed, but also does not harm if we add them

shortcuts

Contraction Hierarchies

- Query algorithm
  - Given a source s and a target t
  - Do a full Dijkstra computation from s forwards, considering only arcs (u, v) with $u < v$
    - we call $G\uparrow = (V, \{(u, v) : u < v\})$ the **upward graph**
  - Do a full Dijkstra computation from t backwards, considering only arcs (u, v) with $u > v$
    - we call $G\downarrow = (V, \{(u, v): u > v\})$ the **downward graph**
  - Let I be the set of nodes settled in both Dijkstras
  - Take $dist(s, t) = \min \{dist(s, v) + dist(v, t) : v \in I\}$
  - Is this correct and if yes why?

CH — Correctness Proof 1/4

- Contraction preserves shortest paths
  - Lemma: Let $G = (V, E)$ be an arbitrary graph, and let $G' = (V', E')$ be the graph after the contraction of an arbitrary node $v \in V$, that is, $V' = V - \{v\}$.

    Then for all $s, t \in V'$ it holds that $dist_{G'}(s, t) = dist_G(s, t)$

Let P be the SP from s to t in G

Case: v does not occur on P $\Rightarrow$ P is also a path in G' $\Rightarrow dist_{G'}(s,t) \leq dist_G(s,t)$

Case: v does occur on P; $P = s, \ldots, u, v, w, \ldots, t$

$\Rightarrow$ u, v, w is a SP $\Rightarrow$ shortcut $(u,w) \in E'$

Then $P' = s, \ldots, u, w, \ldots, t$ is a path in G' with same cost $\Rightarrow dist_{G'}(s,t) \leq dist_G(s,t)$

CH — Correctness Proof 2/4

- Proof of Lemma continued ...

So far we have shown $dist_{G'}(s,t) \leq dist_G(s,t)$

Vice versa:

Let p' be the SP from s to t in G'

Case: p' contains no shortcuts, i.e., all arcs on p' are in E $\Rightarrow$ p' is also a path in G $\Rightarrow$ $dist_G(s,t) \leq dist_{G'}(s,t)$

Case: p' contains a shortcut

$\Longrightarrow \; \ldots\ldots \; \Longrightarrow dist_G(s,t) \leq dist_{G'}(s,t)$

analogously to previous slide, replace each shortcut (u,w) by (u,v,w) ∎

CH — Correctness Proof 3/4

- The query algorithm is correct
  - Lemma: $\text{dist}(s, t) = \min \{\text{dist}(s, v) + \text{dist}(v, t) : v \in I\}$
  - Let P be the shortest path from s to t in G
  - Let v be the largest node (wrt the node ordering) on P in G
  - Consider the **prefix maxima** on the path from s to v, that is, the nodes $u_0 < u_1 < \ldots < u_k$ such that P is

    $s = u_0 \rightarrow^* u_1 \rightarrow^* u_2 \rightarrow^* \ldots \rightarrow^* u_k = v$

    where the subpaths $u_{i-1} \rightarrow^* u_i$ use only nodes $< u_{i-1}$
  - Claim: $P' = (u_0, u_1, \ldots, u_k)$ is the shortest path from s to v in the upward graph (and we can prove a similar thing for the path from v to t in the downward graph)

CH — Correctness Proof 4/4

- Proof of the claim

$u_0 < u_1 < \dots < u_r$

$P = \underset{\overset{\shortparallel}{s}}{u_0} \xrightarrow{}^{*} u_1 \xrightarrow{}^{*} \dots \xrightarrow{}^{*} \underset{\overset{\shortparallel}{t}}{u_r}$

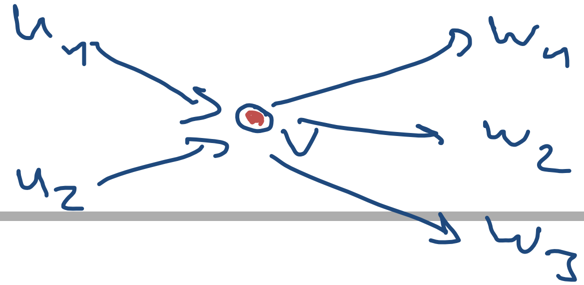
$P' = u_0, u_1, \dots, u_r$ is a path in $G\uparrow$

By the previous lemma, $(u_{i-1}; u_i) \in E\uparrow$ and with the same cost as $u_{i-1} \xrightarrow{}^{*} u_i$

$\Rightarrow \mathrm{dist}_{G\uparrow}(s,t) \leq \mathrm{dist}_G(s,t)$

other direction as before ...

Shortcuts 1/2



- How to determine when a shortcut is needed?
  - Recall: when contracting node v, we need to insert the shortcut arc (u, w), if and only if (u, v) ∈ E and (v, w) ∈ E and (u, v, w) is the only shortest path from u to w
  - As before, $\{u_i\}$ = incoming arcs and $\{w_j\}$ = outgoing arcs
  - Straightforward approach: for each u_i, do a Dijkstra computation until all w_j are settled and see for which w_j v lies on the shortest path from u_i to w_j
  - Improvement 1: We can stop the search when we settle a node with cost $> \text{dist}(u_i, v) + \max_j \text{cost}(v, w_j)$
  - Improvement 2 (sketch): A "1-hop backward search" from each of the w_j gives an even better bound → see paper

Shortcuts 2/2

- How to determine when a shortcut is needed?
  - Improvement 3 (heuristic): For each Dijkstra computation (from each of the u_i), put a **limit** on the number of hops (distance in number of arcs from u_i) and on the size of the search space (number of nodes settled)
    - With this heuristic, we may fail to find a shortest path from u_i to w_j that does not use v, and thus insert the shortcut (u_i, w_j) **unnecessarily**
    - But unnecessary shortcuts do not harm correctness, only performance (if there are too many of them)
    - So there is a trade-off: if the heuristic saves a lot of time in the precomputation at the cost of only a few unnecessary shortcuts, than it is worth it

Implementation advice

- How to add shortcuts / remove contracted nodes?
  - If you implemented the adjacency lists with a vector<vector<Arc> >, adding arcs is straightforward
  - Removing nodes / arcs from the graph is more cumbersome, but luckily there is **no need** to do that
  - Instead, you can just ignore the respective nodes / arcs
    - In the precomputation, ignore all nodes with smaller id than the current one, and their incident arcs
    - At query time, for a Dijkstra search in the upward graph only consider arcs (u, v) with $u < v$, and similarly for the downward graph

Node ordering 1/3

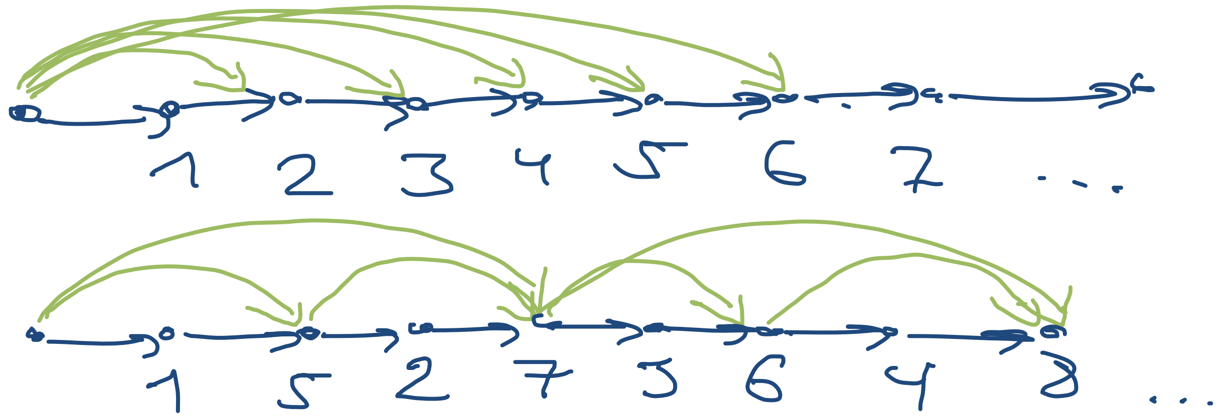
- General approach
  - Maintain the nodes in a priority queue, in the order of how attractive it is to contract the respective node next
  - Intuitively: the less shortcuts we have to add, the better
  - For each node, maintain the edge difference (ED):
    - S = the number of shortcuts that would have to be added if that node were contracted
    - E = the number of arcs incident to that node
    - Then the edge difference is simply ED = S – E
  - Note that when we contract a node, the edge difference of other nodes (not only the neighbours) may get affected

Node ordering 2/3

- How to maintain the ED for each node?
  - Initially compute the ED for each node (linear time)
  - Straightforward approach: recompute for all nodes after each single contraction → quadratic running time
  - Lazy approach: update EDs "on demand" as follows:
    - Before contracting node with currently smallest ED, recompute its ED and see if it is still the smallest
    - If not pick next smallest one, recompute its ED and see if that is the smallest now; if not, continue in same way ...
  - Neighbour heuristic: after each contraction, recompute EDs, but only for the neighbours of the contracted node
  - Periodic update heuristic: Full recomputation every x rounds

Node ordering   3/3

- Other criteria
  - Spatial uniformity is also important, here is an example:



  - Simple heuristic: for each node maintain a count of the number of neighbours that have already been contracted, and add this to the ED

    → the more neighbours have already been contracted, the later this node will be contracted

Getting the arcs on the shortest path

- The query algorithm as described so far ...

  ... gives a shortest path P' in the graph **with shortcuts**

  - find the v which minmizes dist(s, v) + dist(v, t)
  - backtrack path to v in the forward search from s
  - backtrack path to v in the backward search from t

  – How to obtain the shortest path P in the original graph?

  – During the precomputation, for each shortcut arc e remember the contracted node due to which e was inserted

  – Replace each shortcut arc e = (u, w) in P' by the two arcs (u, v), (v, w), where v is the node remembered for e

  – Repeat until no more shortcut arcs left

References

- Highway Hierarchies

  Engineering Highway Hierarchies

  Highway Hierarchies Hasten Exact Shortest Path Queries

  Dominik Schultes and Peter Sanders, ESA 2005 & 2006

  http://algo2.iti.uka.de/schultes/hwy/esa06HwyHierarchies.pdf

  http://algo2.iti.uka.de/schultes/hwy/esaHwyHierarchies.pdf

- Contraction Hierarchies

  Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks

  Geisberger, Sanders, Schultes, Delling, WEA 2008

  http://algo2.iti.uka.de/schultes/hwy/contract.pdf