Efficient Route Planning
SS 2011

Lecture 8, Friday July 8^{th}, 2011
(Transit Node Routing)

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

UNI FREIBURG

Overview of this lecture

- Organizational
  - Your feedback from Ex. Sheet #5 (contraction hierarchies)
- Transit Node Routing (TNR)
  - Our last algorithm in the lecture for routing on road networks
  - The (algorithmically) fastest one to date
  - Based on a very simple and intuitive idea
  - Very simple query algorithm
  - Various possibilities for the pre-computation … we will look at one based on contraction hierarchies (CH)
  - Historically TNR came (two years) before CH

  - Exercise Sheet #6: Implement a part of TNR

Feedback on ES#5 (contract hierarchies)

- Summary / excerpts
  - The extra week was helpful
  - Pity that no web app with a Java backend was shown
  - Implementation advice (in general and for contraction hierarchies) was useful, but came too late
  - Pre-processing takes too long (1 hour)

Transit Node Routing 1/5

- Intuition
  - When you go from your home to somewhere far away ...

    then the initial portion of your route will be one of a few standard routes
  - Let's look at a few examples on Google Maps
  - How can we use this to speed up shortest path queries?

Transit Node Routing 2/5

- We want to have the following
  - For each pair of nodes u and v a **locality criterion** $L(u, v)$ that yields true or false

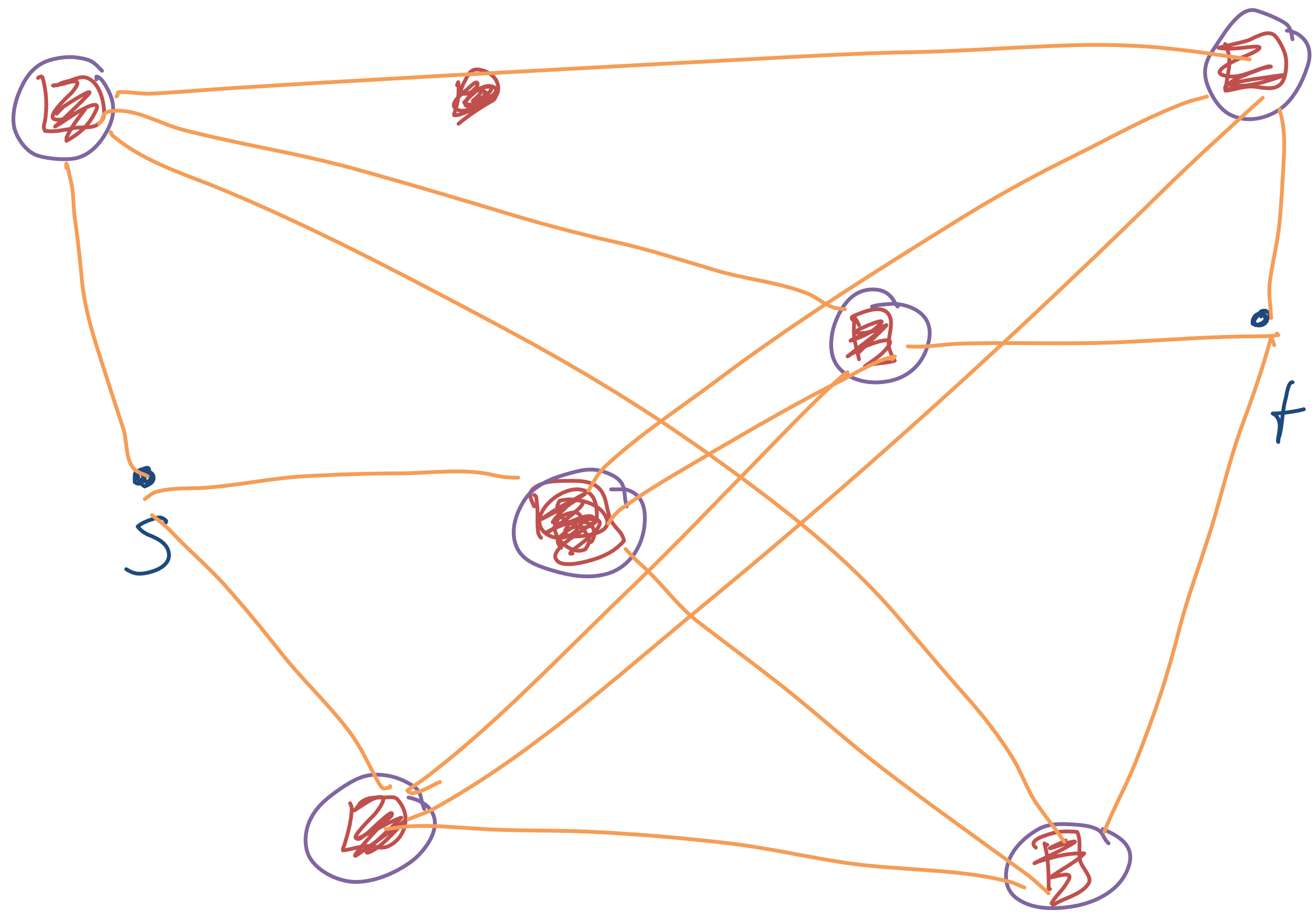
    Intuitively: if $L(u, v) = \text{false}$, then u and v are "far away"
  - For each node u sets $X(u)$ and $Y(u)$ of **access nodes** such that for each v with $L(u, v) = \text{false}$, there exists $x \in X(u)$ on $SP(u, v)$, and for each w with $L(w, u) = \text{false}$, there exists $y \in Y(u)$ on $SP(w, u)$

    Intuitively: $X(u)$ are the nodes via which you leave the neighbourhood of u when you **go to** somewhere far away, and $Y(u)$ are the nodes via which you enter the neighbourhood of u when you **come from** far away
  - Note: for symmetric graphs, $X(u) = Y(u)$

Transit Node Routing 3a/5

- Precomputation (abstract; concrete comes later)
  - Compute something such that $L(u, v)$ can be evaluated quickly for given u and v
  - Compute and store the $X(u)$ and $Y(u)$ for each node u, as well as $dist(u, x)$ for each $x \in X(u)$ and $dist(y, u)$ for each $y \in Y(u)$
    - These are $\Sigma_u (X(u) + Y(u))$ nodes and distances
  - Compute and store the unions $X = \bigcup_u X(u)$ and $Y = \bigcup_u Y(u)$ and the $dist(x, y)$ for all pairs x and y with $x \in X$ and $y \in Y$
    - These are $|X| \cdot |Y|$ distances
    - Our goal will be that both $|X|$ and $|Y|$ are on the order of $\sqrt{n}$ and not n, so that $|X| \cdot |Y| = O(n)$

Transit Node Routing 3b/5



▨ = X = Y (assume symmetric)

Transit Node Routing 4/5

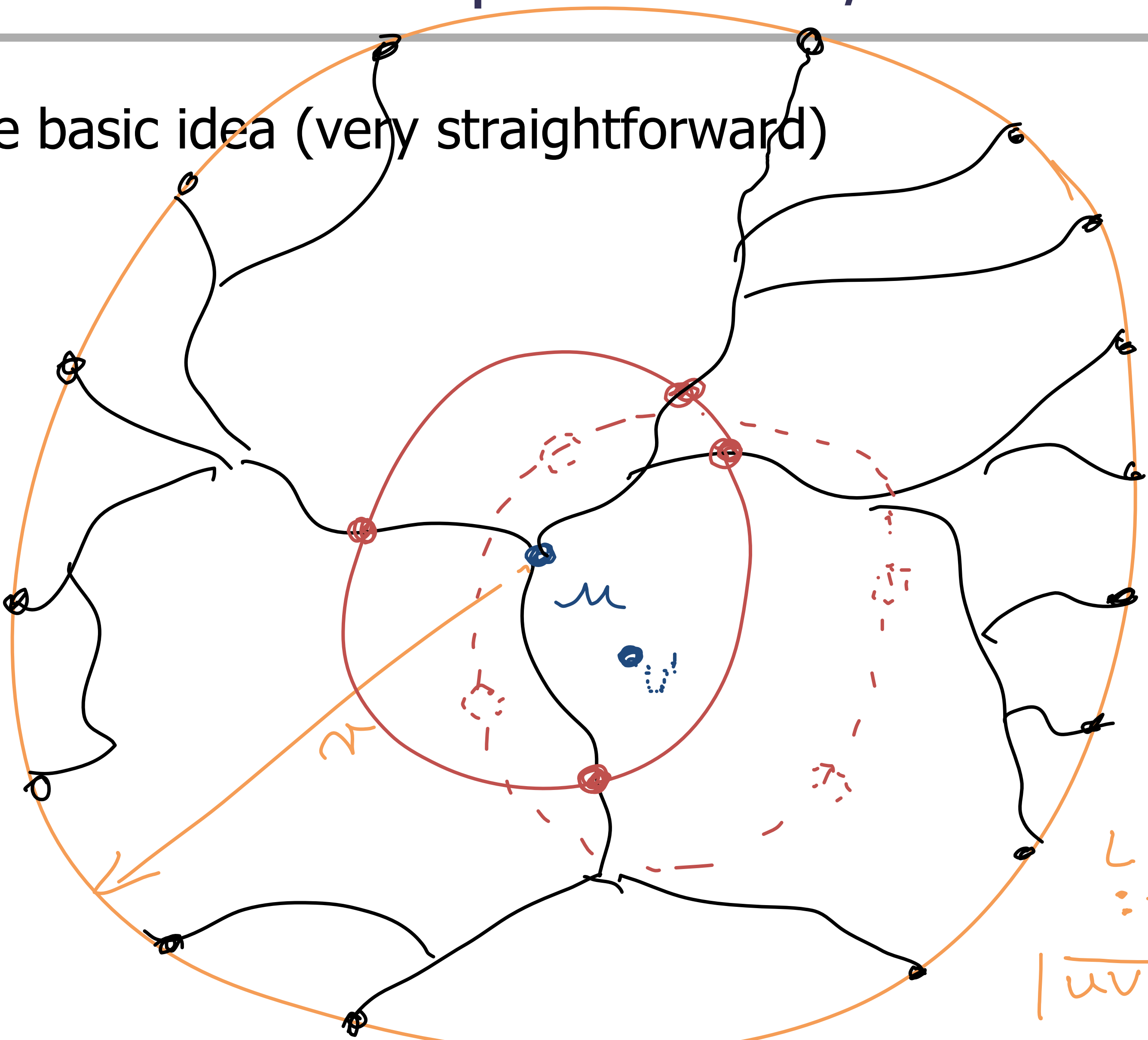
- Processing a query from s to t
  - If $L(s, t) = true$, compute $dist(s, t)$ with another algorithm, for example ordinary Dijkstra; otherwise:
  - Fetch the set $X(s)$ and the $d(s, x)$ for all $x \in X(s)$
  - Fetch the set $Y(t)$ and the $d(y, t)$ for all $y \in Y(t)$
  - Fetch the $d(x, y)$ for all x, y with $x \in X(s)$ and $y \in Y(t)$
  - Compute the minimum $dist(s, x) + dist(x, y) + dist(y, t)$ over all x, y with $x \in X(s)$ and $y \in Y(t)$
    - this is the minimum over $|X(s)| \cdot |Y(t)|$ terms
    - in practice $|X(s)|$ and $|Y(t)|$ can be made as small as **5** on average, hence the extremely fast query times

Transit Node Routing 5/5

- Efficiency
  - **Goal 1:** $L(u, v)$ is easy to evaluate and $L(u, v) = \text{true}$ if and only if $SP(u, v)$ is cheap to compute
    - then we can easily determine whether we have to resort to the fallback algorithm, and if so, it will be cheap
  - **Goal 2:** $X(u)$ and $Y(u)$ are $\leq$ a small C for (almost) all u
    - then the $X(u)$ and $Y(u)$ and the distances to / from them can be stored in $\sim C \cdot n$ space, and queries can be processed in time C^2
  - **Goal 3:** $|X = \cup_u X(u)|$ and $|Y = \cup_v Y(v)|$ are $O(\sqrt{n})$
    - then the pairwise distaces $dist(x, y)$ with $x \in X$ and $Y \in Y$ can be stored in $O(n)$ space

Geometric Precomputation 1/3

- The basic idea (very straightforward)



$X(u)$

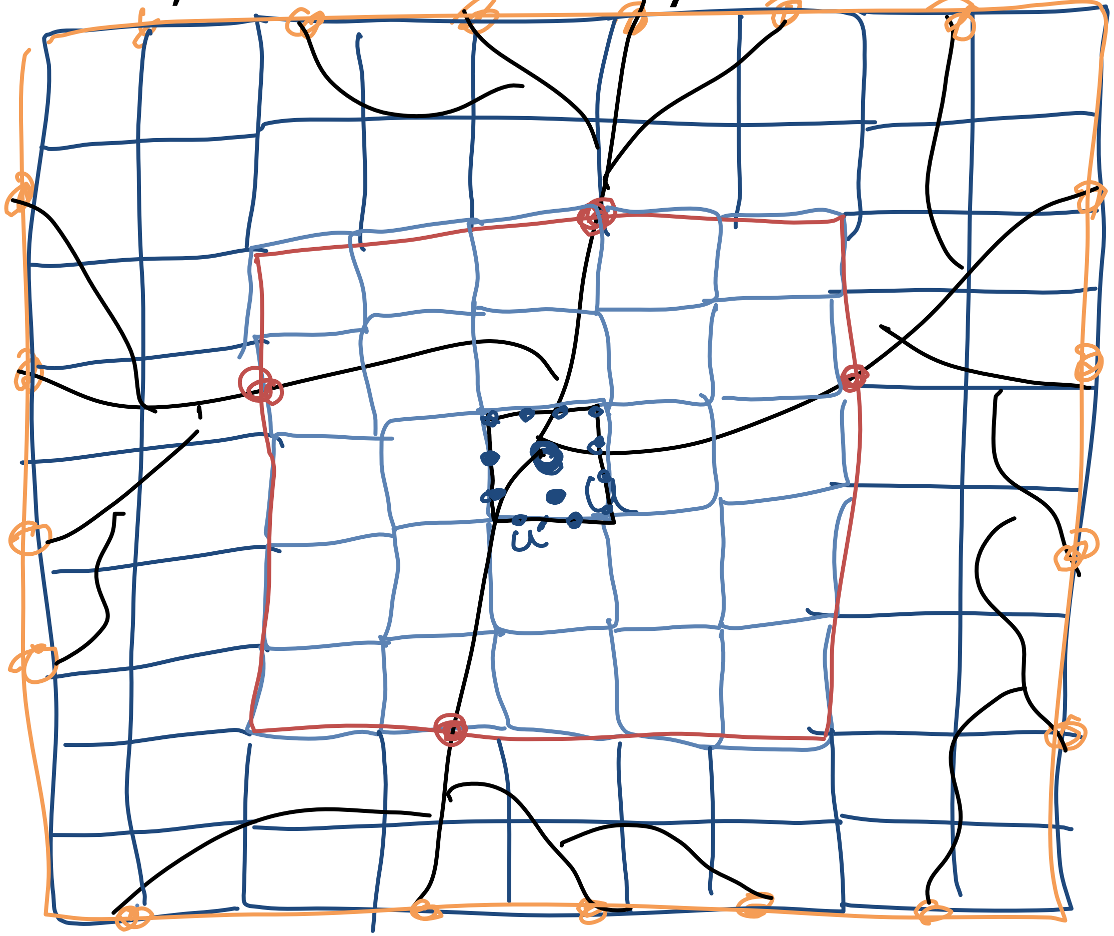
$X(u')$

$L(u,v) := \text{true}$

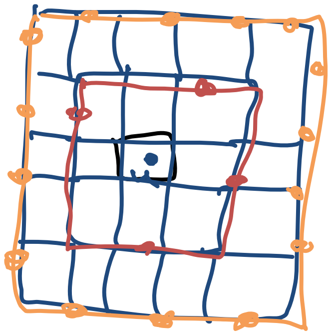
$|\overline{uv}| \leq r$

Geometric Precomputation 2/3

- Refinement, so that it actually works



Warum nicht einfacher so?



Geometric Precomputation 3/3

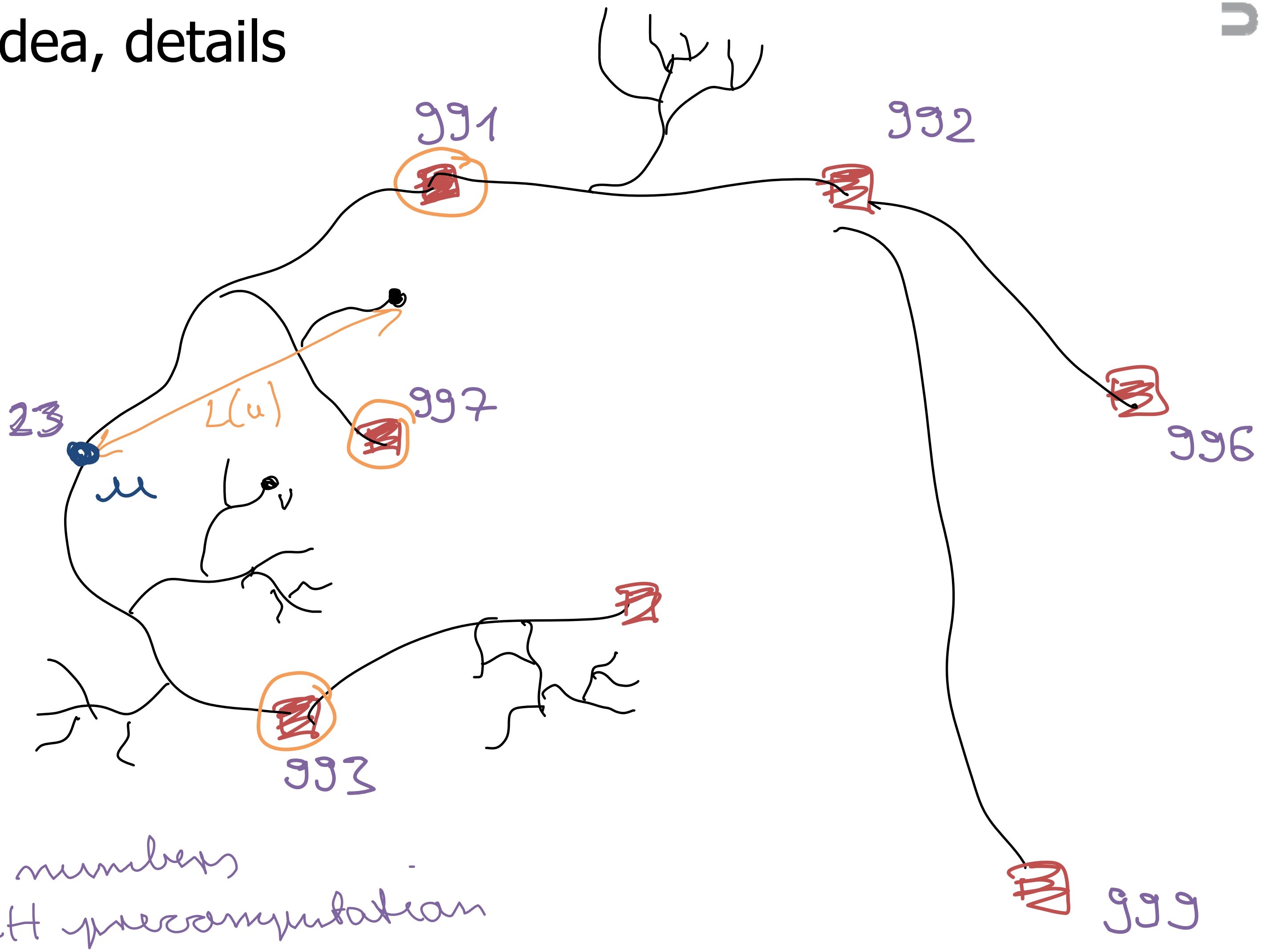
- Resource requirements
  - Small sets of access and transit nodes
  - But precomputation time comparable to that for arc flags (we need a Dijkstra for each boundary node of each cell)
  - There are various tricks to make this faster
  - And we can also make it hierarchical; see later slide
  - See the references for details
  - But let's now look at a precomputation based on CHs

Precomputation based on CHs 1/3

- Basic idea
  - Do the CH precomputation on the given graph
  - Let $X = Y$ be the set of nodes with ordering number above a certain threshold T (we want $|X| = |Y| \sim \sqrt{n}$)
  - For each node u in the graph do a forward search in the upward graph, and for each settled node v compute the first node $x \in X$ on $SP(u, v)$ if any; let $X(u)$ be the union of all these x
  - Similarly, compute $Y(v)$ for each node v in the graph via a backward search in the downward graph

Precomputation based on CHs 2/3

- Basic idea, details



ordering numbers
of the CH precomputation

Precomputation based on CHs 3/3

- Locality criterion
  - Along with the computation of X(u) from the forward search from u also compute the maximal geometric distance L(u) of a node v where SP(u, v) does **not** contain a node from X(u)
  - Then we can set define L(u, v) = true if and only if the geometric distance from u to v is $\leq$ L(u)
  - We can also do the same for Y(v) and thus possibly further improve this locality criterion
  - For more refined locality criteria, see references

Hierarchical TNR (sketch only) 1/2

- TNR can be made hierarchical, too
  - Here is an explanation for two levels of transit nodes
  - For each node, precompute and store the distances to the "closest" level-1 transit nodes (that is, the first level-1 transit nodes on paths to anywhere else)
  - For each level-1 transit node, precompute and store the distances to the "closest" level-2 transit nodes
  - Precompute and store the distances between all pairs of level-2 transit nodes
  - For a query from s to t, now try all combination of (s, x1, x2, y2, y1, t), where x1 and y1 are the level-1 access nodes of s and t, respectively, and x2 and y2 are the level-2 access nodes of the respective x1 and y1

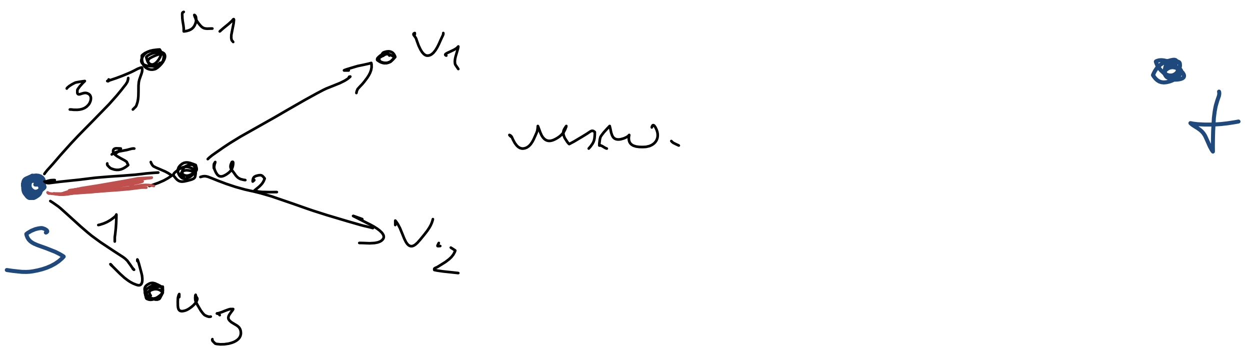
Hierarchical TNR (sketch only) 2/2

- Why does this make sense?
  - We need the pairwise distances only for the level-2 transit nodes
  - Therefore we can have more level-1 transit nodes and hence a better locality criterion = local searches needed only when s and t are very close together
  - But we have to try out more combinations at query time
  - Can be generalized to an arbitrary number of levels
  - Experiments suggest 5 levels for the road network of a whole continent (Western Europe or the US)
  - See the references for details

Computing the arcs on the SP

- Here is a generic method

  ... that works for any algorithm that can compute the cost of a shortest path between any two nodes u and v



$d(s,t) = ?$ 57 min

$d(u_1,t) = ?$ 55 min $55 + 3 > 57$ ✗

$d(u_2,t) = ?$ 52 min $52 + 5 = 57$ ✓

$d(u_3,t) = ?$ 58 min $58 + 1 > 57$ ✗

References

- Transit Node Routing, original paper

  Ultrafast Shortest-Path Queries Via Transit Nodes

  Bast, Funke, Matijevic, DIMACS Shortest Path Challenge

  http://www.mpi-inf.mpg.de/~bast/papers/transit-dimacs.pdf

- Transit Node Routing, based on HH and CH

  PhD thesis from Dominik Schultes (HH), Chapter 6

  http://algo2.iti.kit.edu/schultes/hwy/schultes_diss.pdf

  Master thesis from Robert Geisberger (CH), Section 4.2

  http://algo2.iti.kit.edu/documents/routeplanning/geisberger_dipl.pdf

- Transit Node Routing, article in Science Magazine

  Fast Routing in Road Networks with Transit Nodes

  http://www.sciencemag.org/content/316/5824/566.short