Efficient Route Planning

SS 2011

Lecture 11, Friday July 29th, 2011
(Transfer Patterns, course evaluation)

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

Overview of this lecture

- Organizational
  - Your results from Ex. Sheet #7 (Transit Networks, GTFS)
  - This is the **second to last** lecture
- Transfer patterns
  - A technique for fast routing on transit networks
  - We will also discuss why our previous methods have problems for transit networks
- Exercise sheet … the last one!
  - Fill out the evaluation form for this course → **10 points !**
  - Compute the number of transfer patterns between each pair of stations

Feedback from ES#7 (GTFS)

- Summary / excerpts
  - Verständnisprobleme mit der (optionalen) frequencies.txt
  - GTFS bothersome to parse
  - Implementation advice zum Netzwerk kam etwas spät
  - Neuer Rekord: 5-fach verschachtelte Schleife
  - Hin- und Rückrichtung sind zwei verschiedene "stations"
  - Wichtige Termine standen an. Unser Volk hungert.
  - Difficult to find the motivation
  - Zeitprobleme jetzt gegen Ende der Vorlesungszeit
  - Kein Unterschied zwischen Dijkstra und A*, warum?

Summary of our algorithms so far

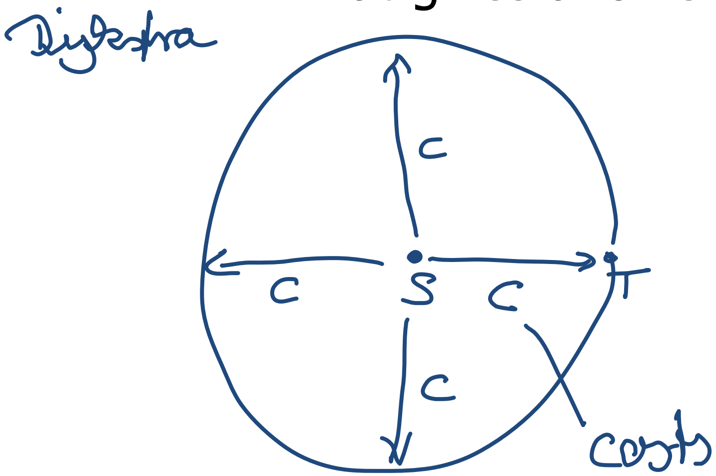
- For computing the shortest path from A to B
  - In the following list, Q = average query time, and P = precomputation time for the OSM network of BaWü
  - Times are only the order of magnitude, not exact, hence ~
  - Dijkstra's algorithm ... Q ~ 0.5 sec, P ~ 0
  - A* with the straightline heuristic ... Q ~ 0.2 sec, P ~ 0
  - A* with the landmark heuristic ... Q ~ 0.1 sec, P ~ 10 sec
  - Arc flags ... Q ~ 1 msec, P ~ 10 hours
  - Contraction Hierarchies ... Q ~ 1 msec, P ~ 1 min
  - Transit Node Routing ... Q ~ 10 – 100 µsec, P ~ 1 min
  - Performance on (time-expanded) transit networks?

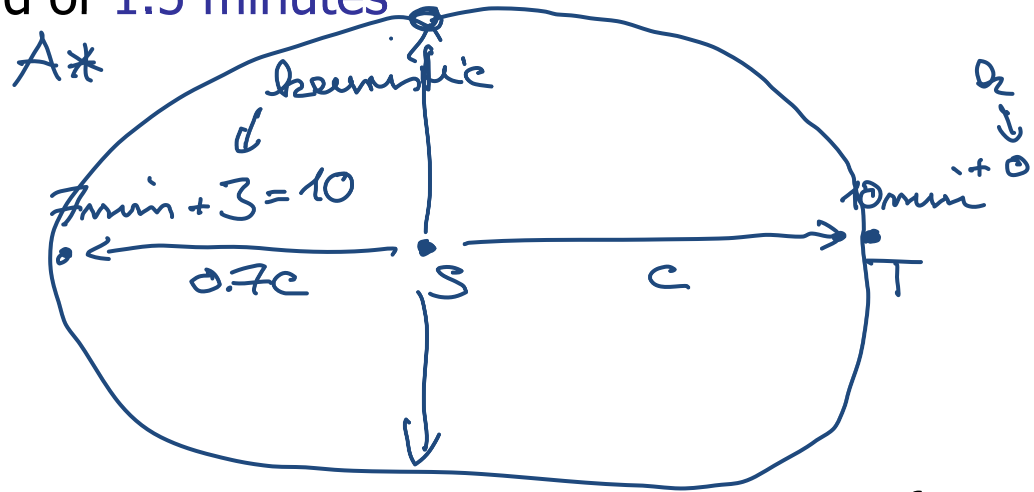
Dijkstra's algorithms

- Performance on time-expanded transit networks
  - One iteration takes ~ 1 µsec / node, as usual
  - But graphs are bigger, for example
    - New York area: ~ 50 000 stations, ~ 2 million nodes
  - When travel time from source to target is T, we settle evything that can be reached within time T, like with the Dijkstra for road networks
  - When T is large or ∞ we search the whole graph ... this happens more often than in road networks, reasons e.g.:
    - bad connectivity by bus / train
    - overnight connections

A* algorithm with straightline heuristic

- Performance on time-expanded transit networks
  - Experiment on the Wiki show hardly any improvement over Dijkstra's algorithm, why?
  - Consider Bus 10 from Bärenweg to Siegesdenkmal
  - Takes 10 minutes, straightline distance is ≈ 2.5 km
  - Let's say the maximum speed is 100 km/h (trains!)
  - That gives a lower bound of 1.5 minutes





A* algorithm with landmark heuristic

- Performance on time-expanded transit networks
  - Also not much improvement over Dijkstra, why?

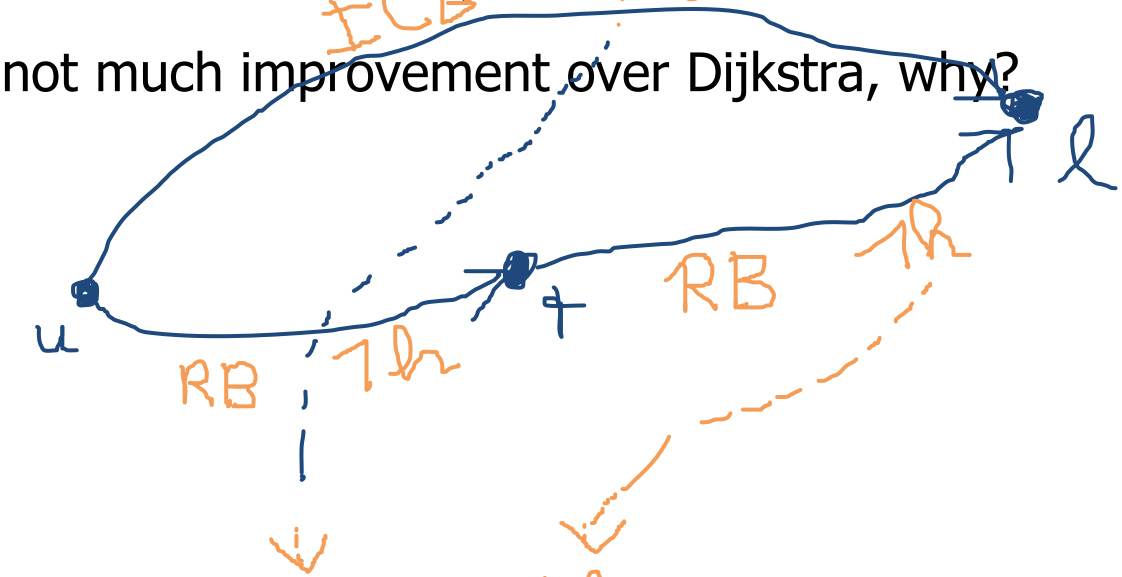
WRONG!

$$h(u) = d(s,l) - d(u,l)$$

Schwacher Zusammenhang zwischen geometrischer Entfernung und Länge (Zeit) eines kürzesten Weges.

A* algorithm with landmark heuristic

- Performance on time-expanded transit networks
  - Also not much improvement over Dijkstra, why?



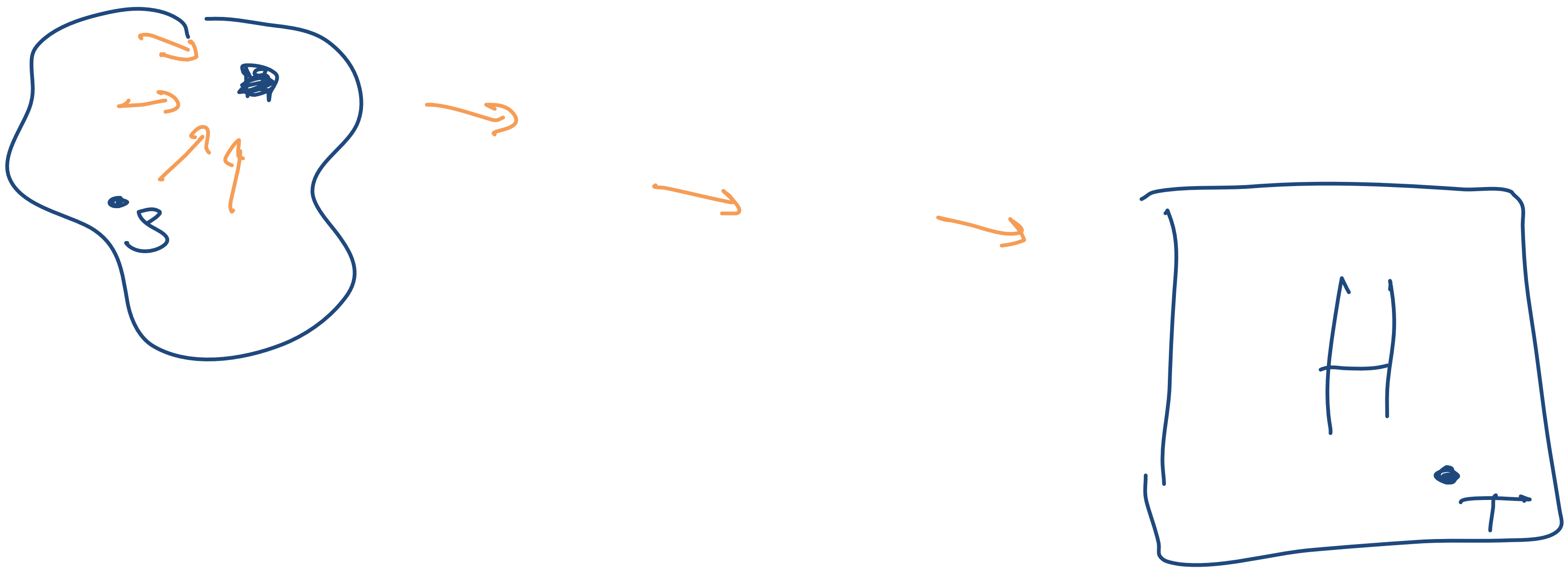
$0h = 1h - 1h$

$h(u) = d(u, l) - d(t, l)$

Schwacher Zusammenhang zwischen geometrischer Entfernung und Länge (Zeit) eines kürzesten Weges.

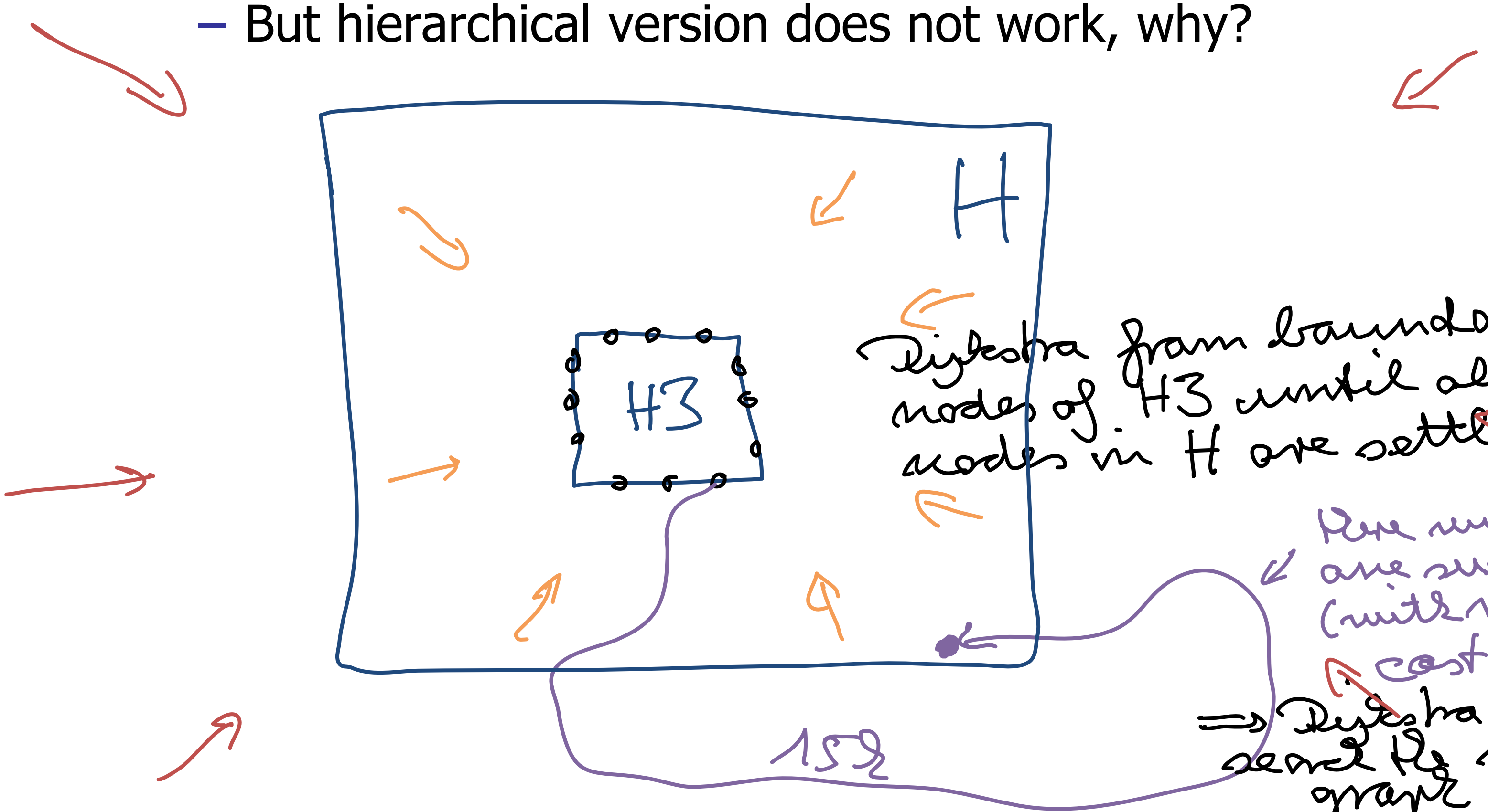
Arc flags 1/2

- Performance on time-expanded transit networks
  - Goal direction works well, good query times, why?



Arc flags 2/2

- Performance on time-expanded transit networks
  - But precomputation cost (for the much larger transit graph is enormous) → calls for hierarchical version
  - But hierarchical version does not work, why?



Dijkstra from boundary nodes of H3 until all nodes in H are settled

There will be one such path (with very high cost).

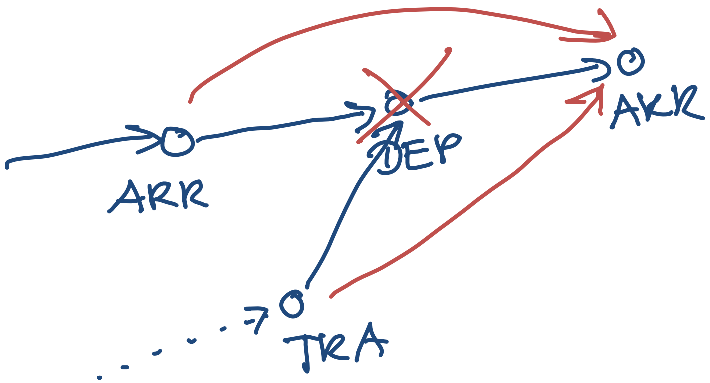
⇒ Dijkstra will search the whole graph.

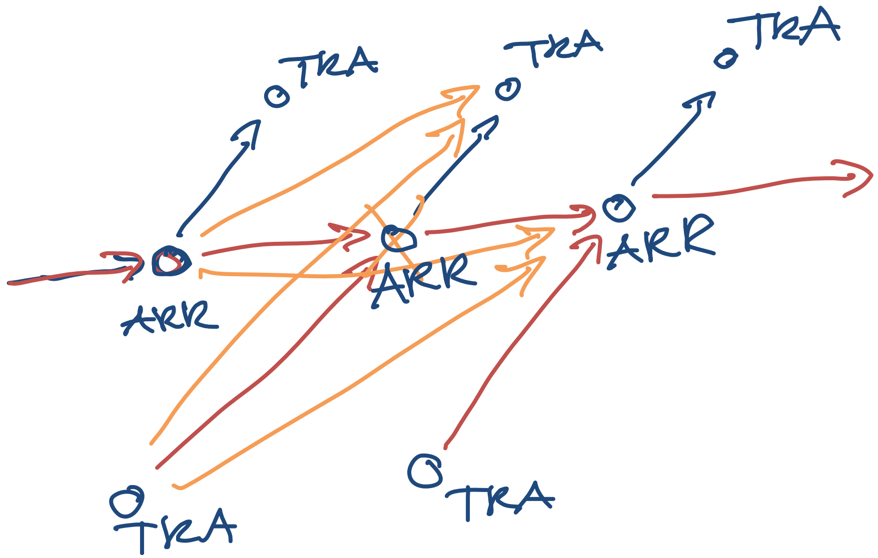
Transit-node routing

- Performance on time-expanded transit networks
  - We can also find small sets of transit / access nodes
  - But how do we compute them efficiently?
  - Also, local queries are not necessarily cheap in transit networks → the "15 hours to the next village problem"

Contraction Hierarchies 1/5

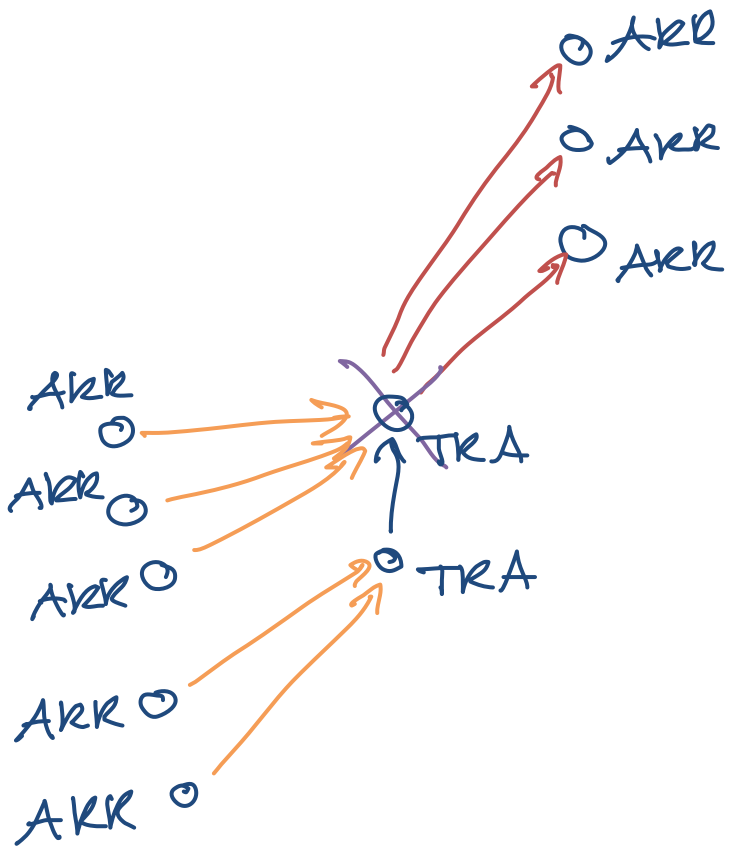
- Performance on time-expanded transit networks
  - Certain nodes can be contracted very well
  - We have already seen that the departure nodes can be contracted trivially, and this even saves us arcs
  - It seems like arrival nodes can also be contracted without loss





Contraction Hierarchies 2/5

- Performance on time-expanded transit networks
  - When we start to contract transfer nodes, the degree explodes:

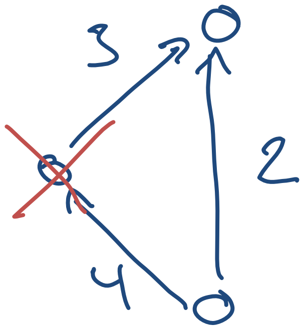


$3 \times 3 + 2 \times 3 = 15$

Contraction Hierarchies 3/5

- Performance on time-expanded transit networks
  - Here is one surprising explanation why we need to add so many arcs in transit networks but not in road networks
  - Note that in transit networks (with cost = travel time), whenever we contract a node (with in and out degree > 0), we always need to add **all** the potentially necessary shortcuts
  - In road networks this is not the case, why this difference?

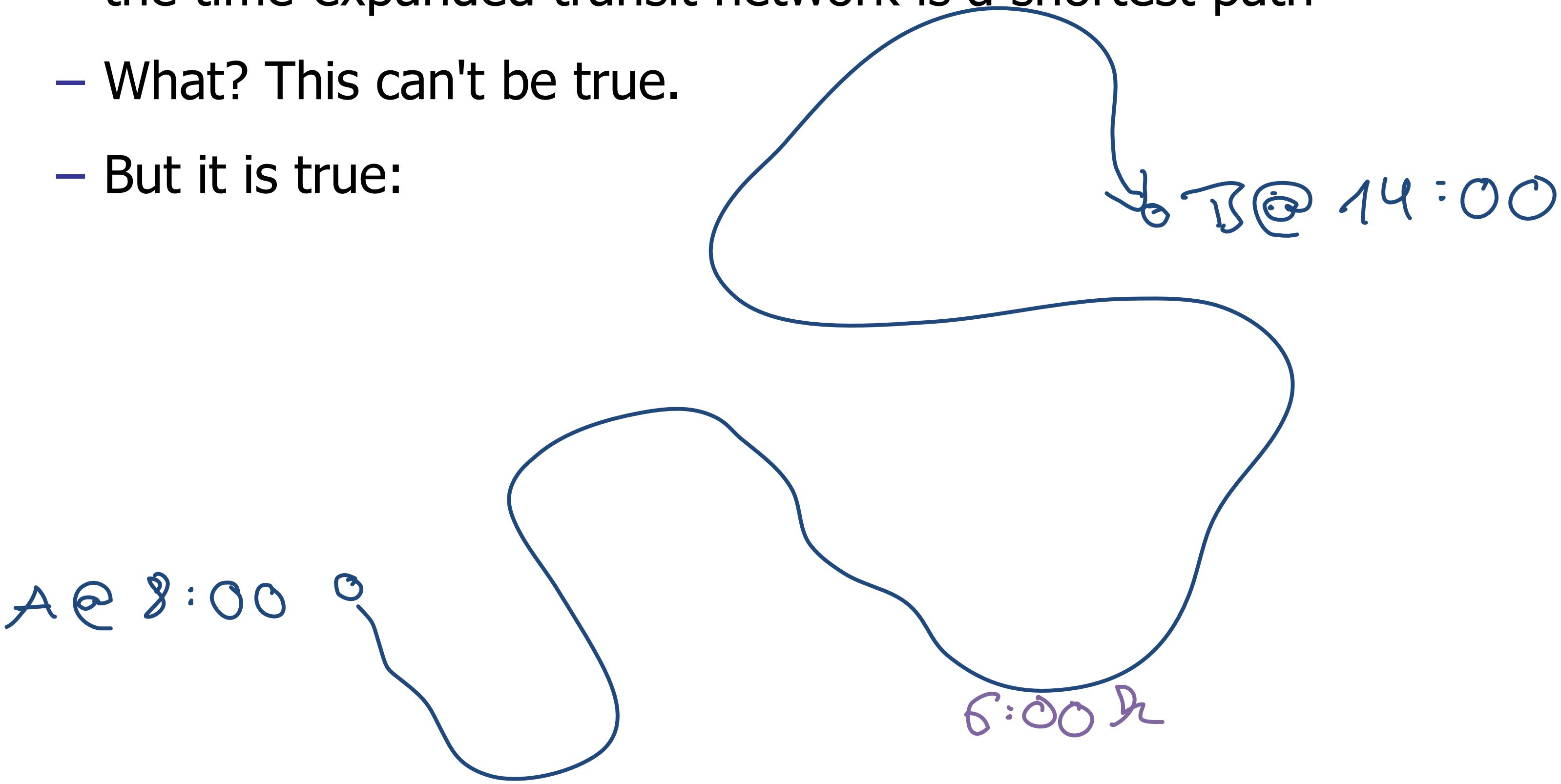
Road network



no need to add shortcut

Contraction Hierarchies 4/5

- Performance on time-expanded transit networks
  - The reason is that (when cost = travel time), **every** path in the time-expanded transit network is a shortest path
  - What? This can't be true.
  - But it is true:



Contraction Hierarchies 5/5

S

T

- Performance on time-expanded transit networks
  - For multi-criteria costs the situation becomes even worse
  - We have seen that in that case, a part P' of a shortest path from A to B might be a non-optimal path itself, and this P' can be arbitrarily far away from A and B
  - How are we supposed to figure out that we need P' (when contracting a node on P') with local Dijkstra computations
    - this is at the heart of contraction hierarchies

Faster

TS EXPRESS

Summary until here

- None of our algorithms so far
  - ... that is: Dijkstra, A* with the straightline heuristic, A* with the landmark heuristic, arc flags, contraction hierarchies, and transit node routing
  - ... is practical for large transit networks
  - And matters seem to become hopeless when realistic features like transfer buffers and multi-criteria cost function come into play
  - And fully-realistic models pose even more challenges:

    service days, vehicle restrictions, finding a suitable source and target station, walking between stations, ...

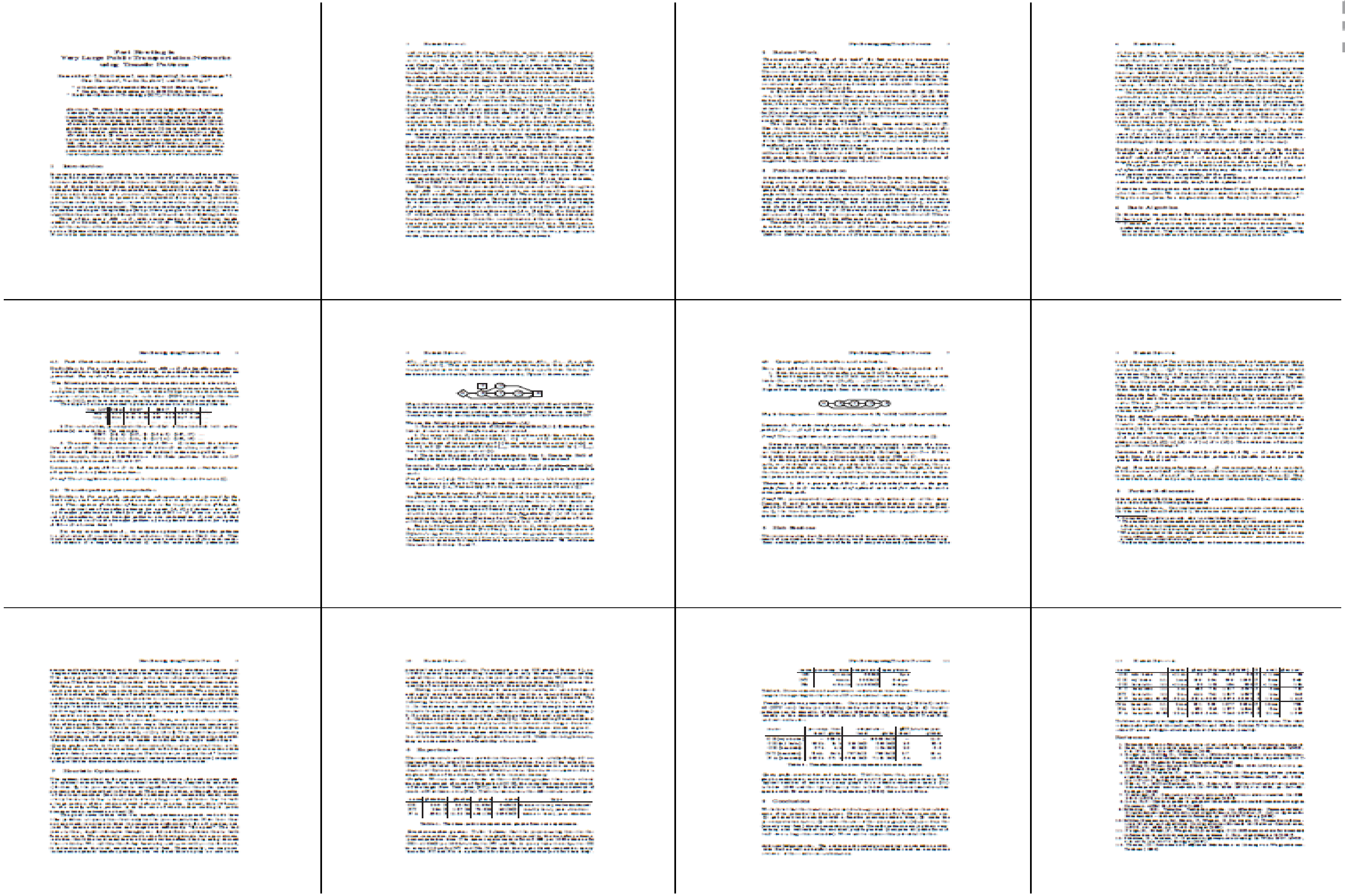
Transfer Patterns 1/3

- An algorithm designed for transit networks
  - Trying to exploit what is special about transit networks
  - But what could this be? So far we have only seen things which are harder on transit networks than on road networks
  - Here is one thing special about transit networks:

    transfers

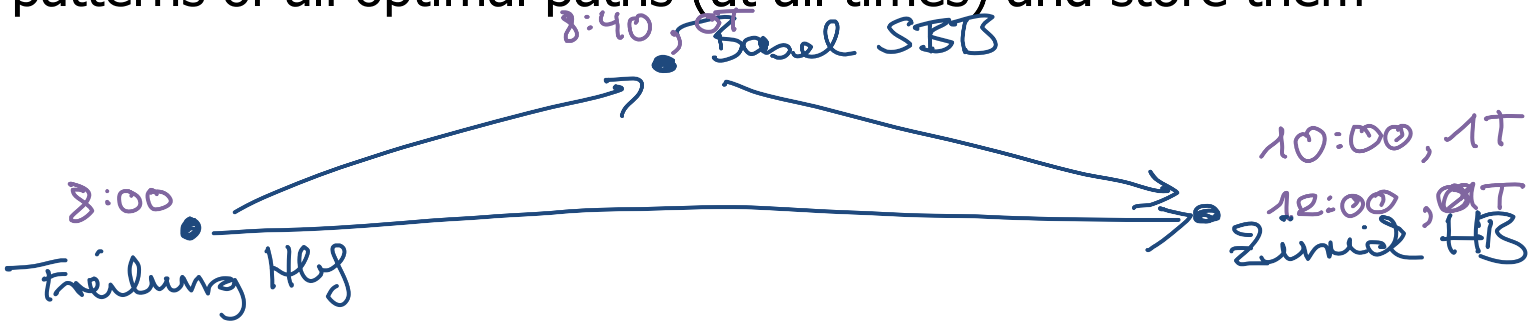
  - Even when you take a very long trip, the number of transfers is almost always a very small number
  - And more than that, for a given source and destination, there is only a very limited number of "**patterns**" where it makes sense to transfer

Transfer Patterns 2/3



Transfer Patterns 3/3

- The basic idea on one slide
  - The **transfer pattern** of a path = the sequence of stations on the path where one boards, transfers, or alights
  - Idea: for each pair of stations, precompute all transfer patterns of all optimal paths (at all times) and store them



  - Then, at query time, do a time-dependent Dijkstra computation on this so-called **query graph**, where each arc evaluation is again a shortest path query, but restricted to **no transfers**
  - Such **direct-connection** queries are easy to compute fast

Direct-Connection Queries

- One table per "line", let's call this one L17

Stations:	S154	S97	S987	S111	...	
Time from start:	0min	7min	12min	21min	...	
Start times:	8:15	9:15	10:15	11:20	12:20	...

- Lines per station (with positions in the respective line table)

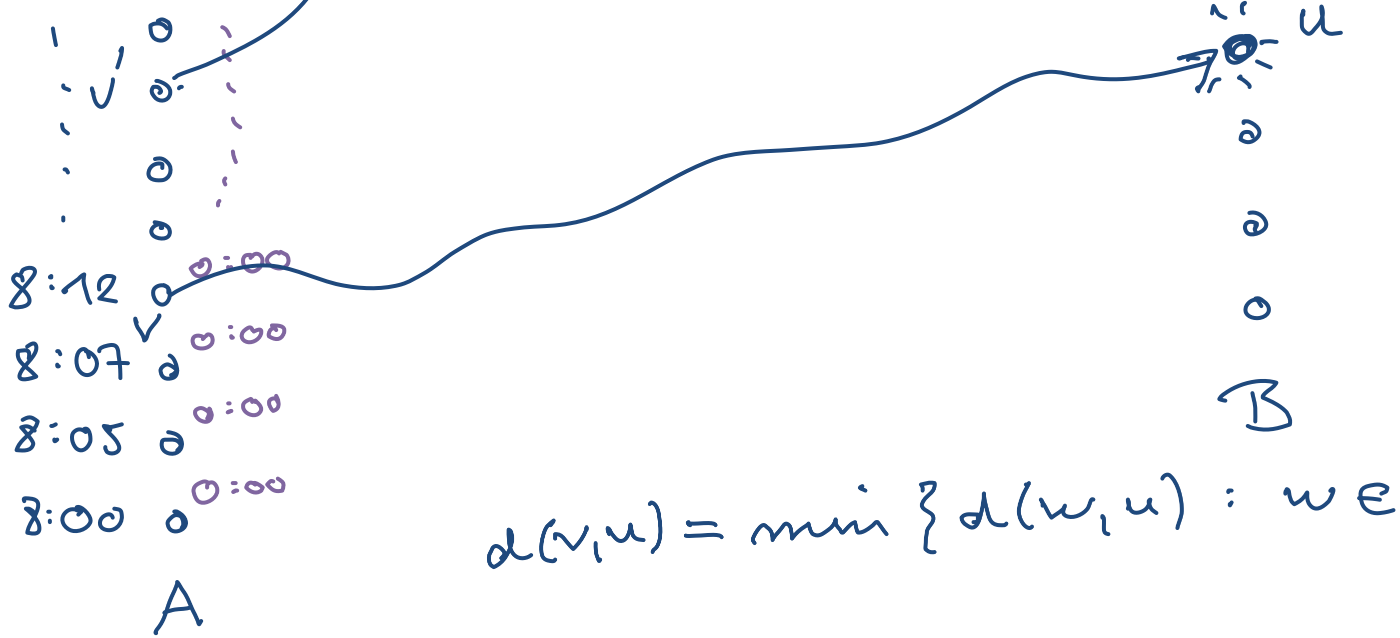
  Station S97: (L8, 4) (L17, 2) (L34, 5) (L87, 17) ...

  Station S111: (L9, 1) (L13, 5) (L17, 4) (L55, 16) ...

- Example query from S97 @ 10:20 to S111
  - Intersect the lists of the two stations : (L17, 2 → 4) ...
  - Find time from start to S97 and to S111 : 7min and 21min
  - Find first start time after 10:20 – 7min : 10:15 → **depart 10:22**
  - Compute arrival time at S111 : 10:15 + 21min → **arrive 10:36**

Transfer patterns precomputation 1/4

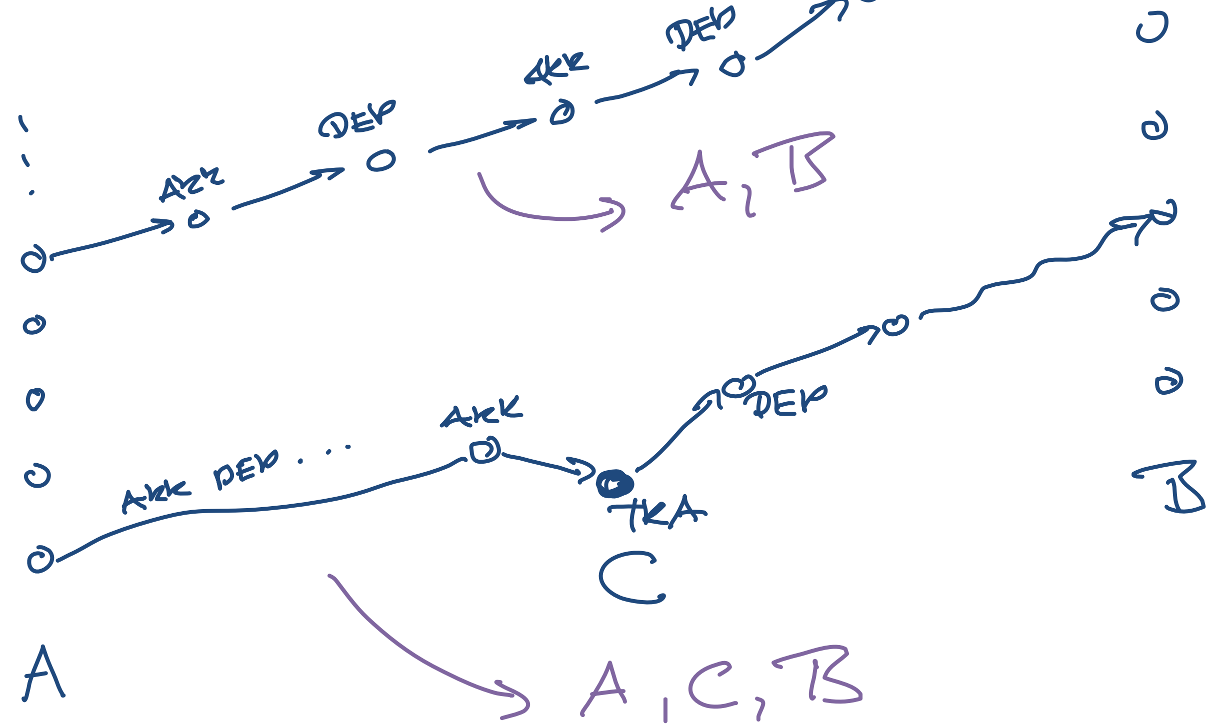
- Can be done via a Set-Dijkstra search
  - For each station A, do a Dijkstra starting from **all nodes** at that station (all with cost = travel time zero)
  - For each other node u in the graph, this will give us the path from the latest node at A so that u can still be reached



$$d(v,u) = \min\{d(w,u) : w \in A\}$$

Transfer patterns precomputation 2/4

- Now we have all optimal paths at all times
  - To obtain the transfer patterns for a station pair (A, B), simply trace back, in the Dijkstra search from A, the paths from all nodes in B and keep track of the transfers

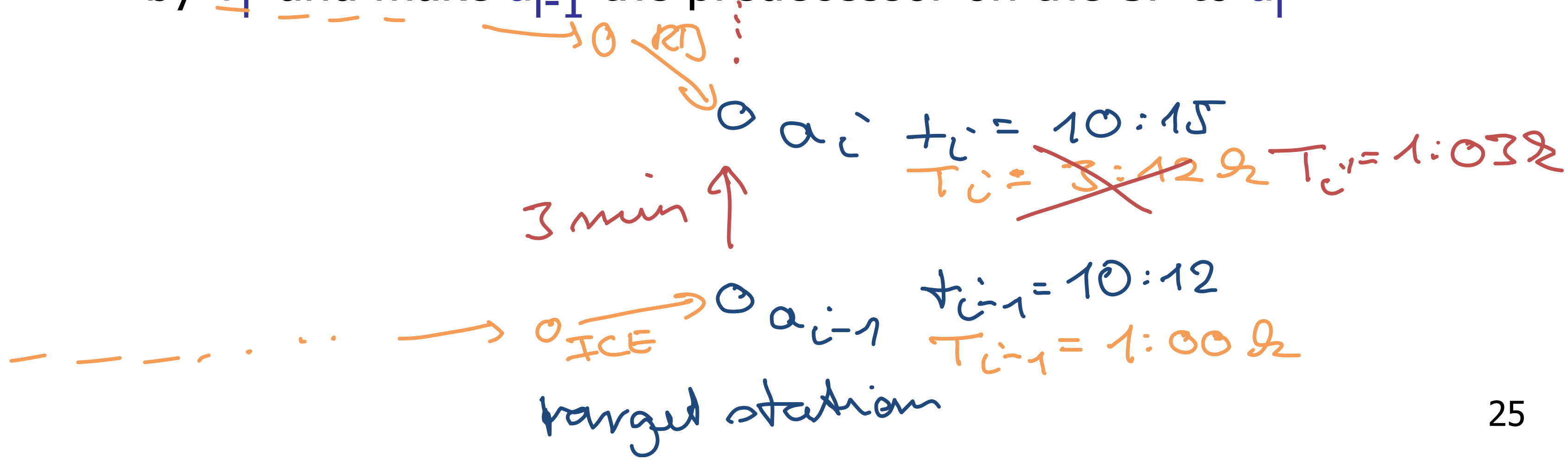


Transfer patterns precomputation 3/4

- Beware of non-optimal paths to arrival nodes
  - Note that there are no arcs between the arrival nodes at the target station
  - They would harm the Dijkstra search (because they would allow us to switch between lines when we shouldn't)
  - But after the Dijkstra search is done, we need them to discard non-optimal paths

Transfer patterns precomputation 4/4

- Arrival-loop algorithm for a target station B
  - Order the arrival nodes by time $t_1 \leq t_2 \leq t_3 \leq \ldots$ and call the corresponding arrival nodes $a_1, a_2, a_3, \ldots$
  - Do the following in the order of increasing time
  - Let T_{i-1} and T_i be the travel time of the shortest path to ai-1 and ai, respectively
  - If $T_i' := T_{i-1} + (t_i - t_{i-1}) \leq T_i$, replace the travel time at a_i by T_i' and make a_{i-1} the predecessor on the SP to a_i



References

- Road Networks vs. Transit Networks

  Car or Public Transport — Two Worlds

  Hannah Bast, Efficient Algorithms 2009, LNCS 5760

  http://www.springerlink.com/content/y46257m66372x730/

- Transfer Patterns

  Fast Routing in Very Large Transportation Networks using Transfer Patterns

  Bast, Carlsson, Eigenwillig, Geisberger, Harrelson, Rachyev, Viger ESA 2010

  http://www.springerlink.com/content/c873271685124v42/