Efficient Route Planning

SS 2012

Lecture 9, Wednesday July 4th, 2012

(Transit Networks, GTFS)

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

Overview of this lecture

- Organizational
  - Your feedback from Exercise Sheet #8 (Transit Node Routing)
- Transit Networks
  - In the US, "transit" means "public transportation"
  - Transit node routing has nothing to do with this "transit"
  - We will see how to model a transit network
  - GTFS = General Transit Feed Specification
  - Do our algorithms so far work on transit networks?
  - Exercise Sheet #9: Parse a transit network from GTFS and run 1000 queries on it, using basic Dijkstra

Feedback on ES#8 (Transit Node Routing)

- Summary / excerpts last checked July 4, 15:05
  - Not hard for those with a working CH implementation
  - Otherwise most time used for fixing bugs in old code
  - Not a good idea to make the exercise sheets depending upon each other ... sorry, but it's hard to avoid for this stuff
    - But next sheets will be something completely new!
  - Why store transit nodes in a hash set?

Results for ES#8 (Transit Node Routing)

- Summary
  - Average time to compute access nodes:
    - ~ 1ms for both datasets
    - That would be ~ 1 hour for the whole of BaWü
  - Average number of access nodes
    - 6 for Saarland, 36 for BaWü
  - Average query time
    - ~ 10 µs with C++, 2-3 times slower with Java
    - **Note:** the query times you measured benefit from the fact that all the relevant values are already in the cache

Transit Networks

- What kind of data have we got?
  - **Stations** (train stations, bus stops, etc.)
  - **Lines** (trains, buses, trams, etc.)
  - The **schedule** of these lines, that is, on which days do they serve which stations at which times
  - How to model these as a directed graph?
    - So that "from A to B" queries become shorted path queries on such a graph, just like for road networks

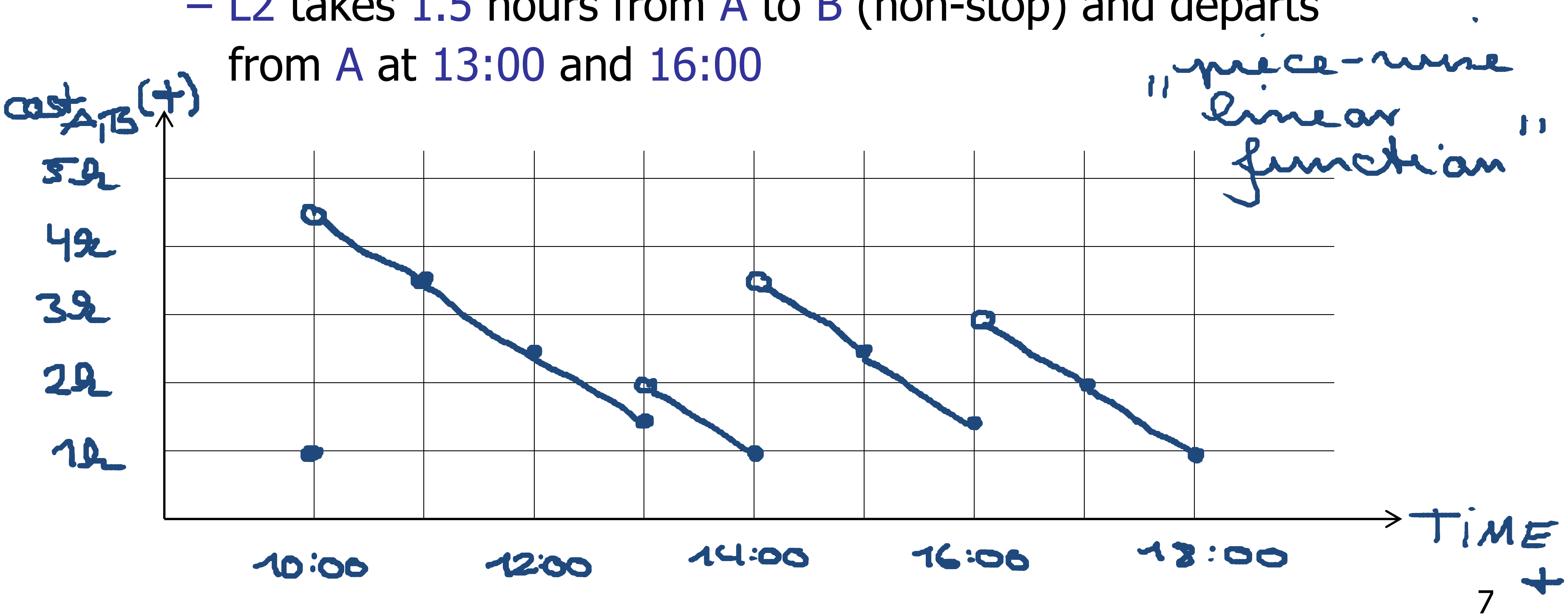
Time-dependent model 1/2

- The first thing that comes to mind
  - Each station is a **node**
  - There is an **arc** between two nodes u and v, if there is a vehicle (train, bus, tram, ...) going **non-stop** from u to v
  - However, that arc can only be used at certain times, and the time it takes to travel across the arc depends on the vehicle commuting at that time
  - We can model this via a **cost function** for each arc (u, v)

    $\text{cost}_{u,v}(t)$ = the time to get from u at time t ... to v
  - Note: for road networks that function was a constant

Time-dependent model 2/2

- Example
  - Stations A and B with two lines L1 and L2
  - L1 takes 1 hour from A to B (non-stop) and departs from A at 10:00, 14:00 and 18:00
  - L2 takes 1.5 hours from A to B (non-stop) and departs from A at 13:00 and 16:00

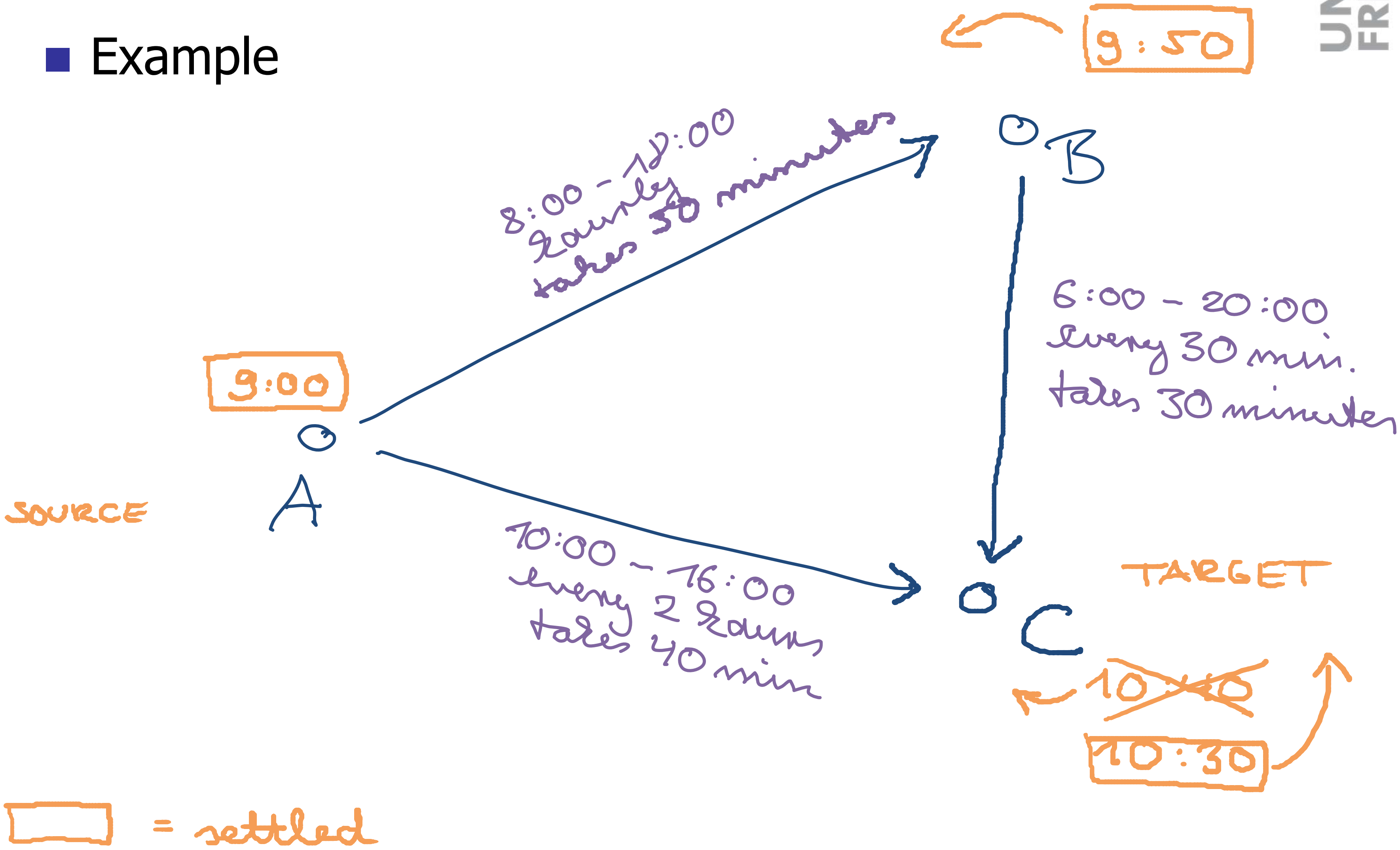


Time-dependent Dijkstra 1/2

- How to compute shortest paths on such a graph?
  - A simple variant of Dijkstra's algorithm does it
  - Tentative distances at the nodes are now **times of day**
    - We will store **absolute** times (like 10:20) and call them t[u] for node u, but we could also store times relative to the start time (like 40 minutes)
  - Start with t[s] = start time and all other t[u] = ∞
  - When relaxing an arc (u, v) we compute $c = c_{u,v}(t[u])$ and take t[v] = t[u] + c if that improves on the previous t[v]
  - As for ordinary Dijkstra process the node u with the smallest t[u] next, and stop when this is the target node

Time-dependent Dijkstra 2/2

- Example

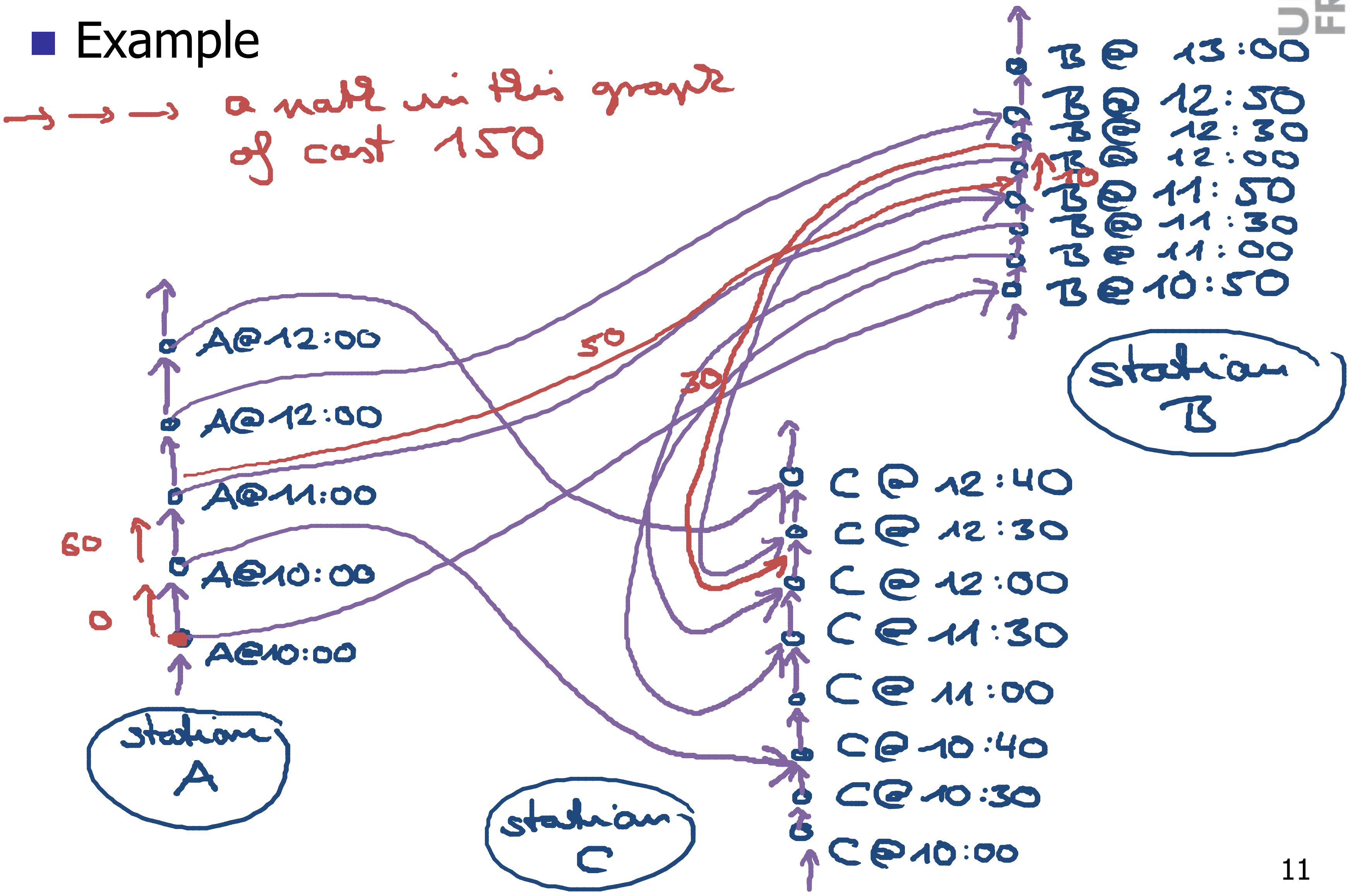


Time-expanded model 1/3

- A node = a particular time at a particular station
  - Only at times, where something (= an arrival or a departure) is happening
  - For example, Freiburg Hbf @ 10:57
  - There is an arc between two nodes A@t1 and B@t2 if there is a vehicle departing from A at time t1 and arriving at B at time t2, without stops inbetween
  - The cost of the arc is simply the travel time t2 – t1
  - There is also an arc from A@t1 to that node A@t2 with the smallest t2 after t1 ... we call these **waiting** arcs

Time-expanded model 2/3

- Example



→ → → a path in this graph of cost 150

Time-expanded model 3/3

- How do we compute shortest paths in this model?
  - It's an ordinary directed graph with (static) non-negative arc costs, so we can use ordinary Dijkstra
  - Problem: We do not have a target node, we only have a target station
  - Solution: Run Dijsktra until **any** node from the target station is settled (which will be the first one reached)

Time-expanded vs. time-dependent

- So far, not much difference
  - Given a query A@t → B, consider the sequence of arcs relaxed by a (time-dependent) Dijkstra on the time-dependent graph
  - The Dijkstra on the time-expanded graph relaxes the same arcs in the same order
    - **plus** some additional waiting arcs to some additional nodes and the arcs leaving from these nodes
  - Intuitively, the time-dependent Dijkstra considers waiting and normal arcs in one (time-dependent) arc
  - The big advantage of the time-expanded model is that we have an ordinary directed graph and can thus use **all** our previous algorithms on it

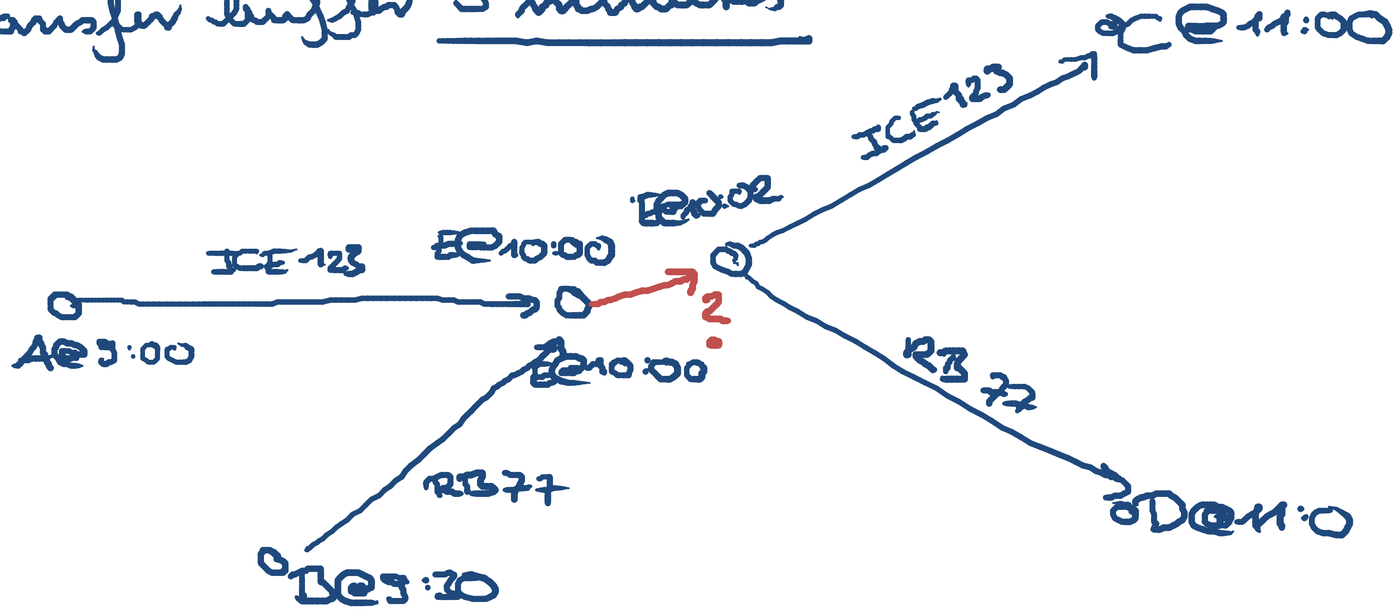
Advanced modelling issues

- For example, what about ...
  - Transfer buffers
    - We need a minimal amount of time to transfer between two vehicles → next slide
  - Service days
    - Different schedules on different weekdays, holidays, etc. → later slide
  - Multi-criteria cost functions
    - Maybe we can get from A@t to B in 3 hours with 0 transfers, or in 2 hours with 2 transfers
    - Which one is better depends on user preference, so we should compute both → next lecture

Transfer buffers 1/6

- Time-expanded model
  - This is non-trivial, because we need to distinguish between
    - staying on a vehicle at a station (no transfer buffer)
    - changing the vehicle (non-zero transfer buffer)

let's say
transfer buffer 5 minutes

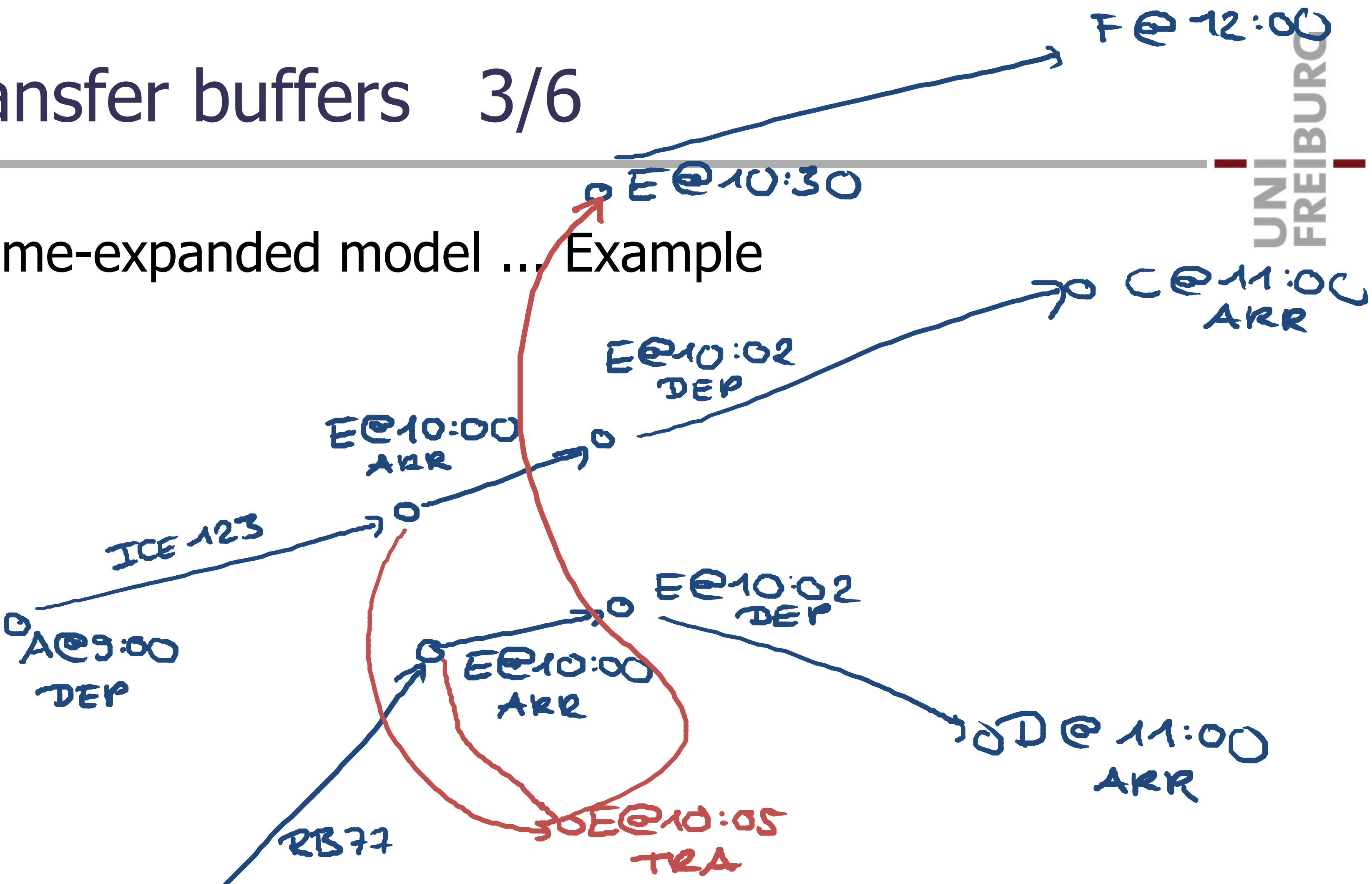


Transfer buffers 2/6

- Time-expanded model ... Solution
  - Split up each node from before into an **arrival node** and a **departure node**, and add an arc between the two
    - we can also model layover time that way now!
  - For each arrival node A@t, add a new **transfer node** A@t' where $t' = t + \Delta$ ... where Δ is the transfer buffer
  - For each departure node A@t have an arc from the transfer node A@t' with the largest t' that is $\leq t$
  - Have the waiting arcs between transfer nodes only
  - Departure at the source is now from a departure node, and arrival at the target is at an arrival node

Transfer buffers 3/6

- Time-expanded model ... Example



DEP → ARR : „traveling"
ARR → DEP : „layover"
ARR → TRA : „alighting"
TRA → DEP : „boarding"

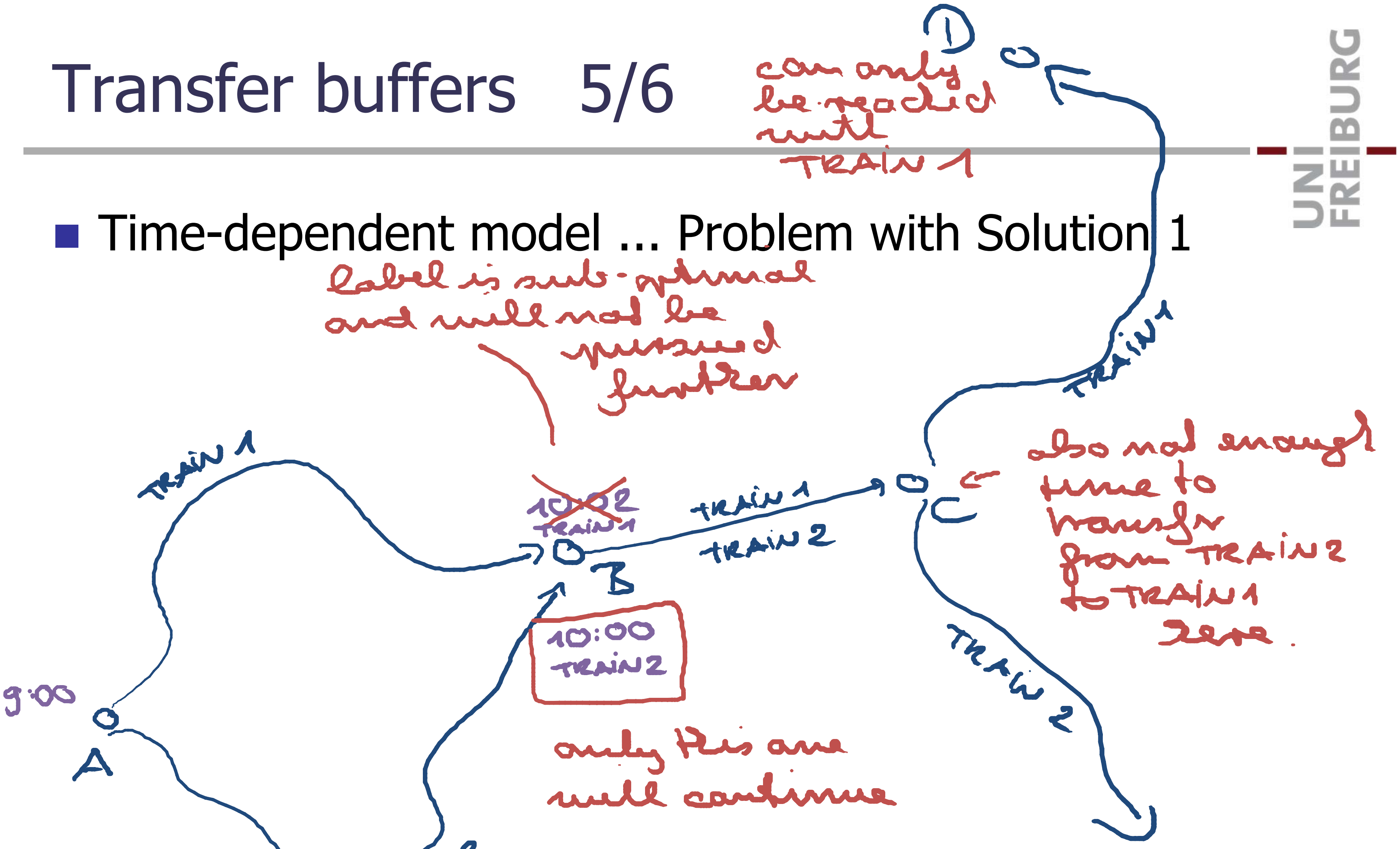
Transfer buffers 4/6

- Time-dependent model ... Solution 1
  - We also have to distinguish here between staying on a vehicle and changing the vehicle at a station
  - It looks like we can do this by simply remembering for each node, along with the tentative arrival time $t[u]$, the id ℓ of the vehicle with which we arrive at u
  - Then we can build the transfer buffer into the cost function

    $\text{cost}_{u,v}(t, \ell)$ = time to reach v, if we are at u at time t sitting in vehicle ℓ
  - Unfortunately, it can happen then that Dijkstra's algorithm **misses** some shortest paths

Transfer buffers 5/6

- Time-dependent model ... Problem with Solution 1



can only be reached with TRAIN 1

label is sub-optimal and will not be pursued further

also not enough time to transfer from TRAIN 2 to TRAIN 1 here.

only this one will continue

The problem is: PREFIXES of shortest paths no longer are (necessarily) SPs themselves !!!

Transfer buffers 6/6

- Time-dependent model ... Solution 2
  - When we can arrive at a station at two different times t1 and t2 with different vehicles, and |t2 – t1| is ≤ the transfer buffer, pursue **both** possibilities
  - Then we need to do a multi-label Dijkstra (maintains **sets** of labels per node) ... see next lecture
- Time-dependent model ... Solution 3
  - Have separate arrival and departure nodes, too
  - One arrival and one departure node per "line" suffices
    - less nice, since no longer only one node per station
    - but often a good compromise in practice

GTFS

- General Transit Feed Specification
  - Standard format established by Google in 2005
  - The story how it started: http://tinyurl.com/6yczek2
  - See the references to the GTFS specification
  - Relatively complex, because there are so many pecularities, special cases, etc. for transit networks
  - For a simple graph model, it is easy though

GTFS

- Basic concepts
  - **stop** = what we call a station
    - e.g. Freiburg Hbf or Siegesdenkmal
  - **trip** = journey of a particular vehicle at a particular time
    - e.g. the journey of Bus 10 from Bärenweg at 17:56 to Siegesdenkmal at 18:07
  - **route** = trips that have a common description (our "line")
    - e.g. all journeys of Bus 10 over the day
  - **service days** = days of the week when a trip is available
    - e.g. on weekdays (Mo-Fr) or on the weekend (Sa-Su)

GTFS

- The files you need for Exercise Sheet #9
  - stop_times.txt : the actual schedule information, what eventually becomes the arcs in the transit graph
  - frequencies.txt : some lines repeat in exactly the same way over the same day, then you have the first trip in stop_times.txt, and how it repeats in frequencies.txt
  - calendar.txt : service day patterns, and which days of the week belong to it
  - trips.txt : tells us which trips commute on which service days (via the patterns from calendar.txt)
  - All files are in CSV format = a table with one record per line, columns separated by a comma, headings in first line

Implementation Advice 1/7

- You need a simple CSV Parser
  - We have written one for you in both C++ and Java
    - because we are so incredibly nice
  - You find it in the SVN folder for this lecture
  - Very simple and easy-to-use interface
    - `openFile(csvFileName)` ... open the CSV file
    - `readNextLine()` ... read next line from file
    - `getItem(i)` ... get column `i` of line just read

Implementation Advice 2/7

- Graph class
  - If you have a graph class with members

    ```
    Array<Array<Arc>> adjacenctArcs;
    Array<Node> nodes;
    ```

    you can use that for the (time-expanded) transit network as well
  - That way, you can run your algorithms with little or no modifications on the transit network as well
  - You might want to add some additional info to the `Node` class (like the station to which a node belongs) and to the `Arc` class (like the name of the GTFS route)

Implementation Advice 3/7

- For the graph on a given weekday, do as follows:
  - First parse calendar.txt and remember (in a hash set) those service ids which contain the given weekday
  - Then parse trips.txt and remember (in a hash set) those trip ids with a valid service id from the hash set above
  - Then parse stops.txt and create a mapping from the GTFS stop ids strings to consecutive numerical stop ids
  - Then parse frequencies.txt and store (in a hash map) the repetitions for each trip id **... not needed for Exercise!**
  - Then parse stop_times.txt and for each block of lines in the file with the same trip id, add the corresponding nodes and arcs to your graph ... see following slides

Implementation Advice 4/7

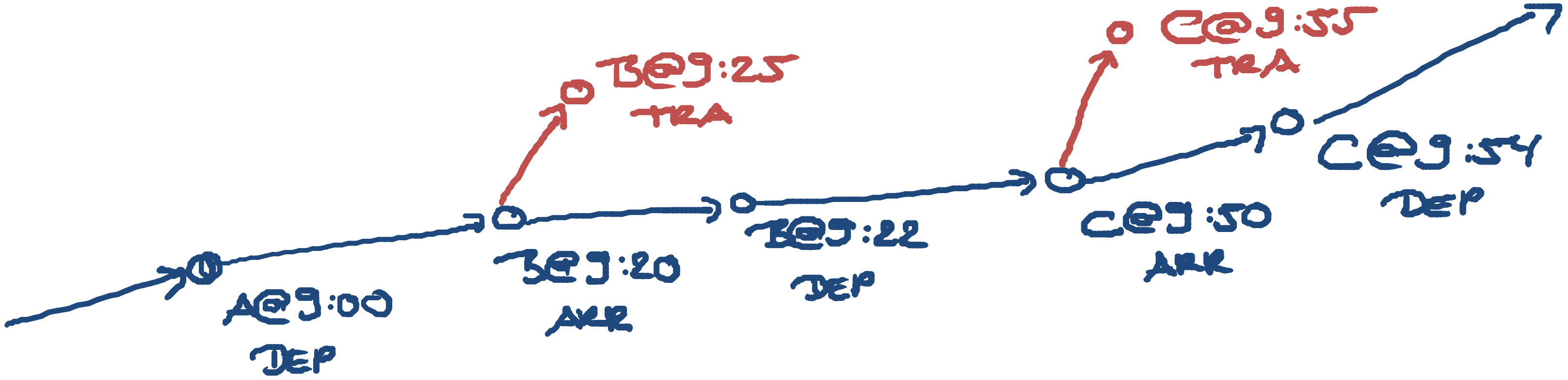
- Blocks with the same trip id in stop_times.txt
  - For the last step, it is very convenient to have all lines with the same trip id together in one block, and within this block have them sorted by stop_sequence
  - The GTFS standard does not demand this ... but you can easily achieve this with a command-line sort

    ```
    sort –t, -k1,1r –k5,5n stop_times.txt > new_file.txt
    ```

  - If you have frequency information for the trip id of a line from stop_times.txt, don't forget to repeat accordingly
    - **Note:** some GTFS feeds write all times explicitly in stop_times.txt and do not have frequencies.txt at all
    - In particular, this is so for the GTFS feed from ExSh#9

Implementation Advice 5/7

- Arrival, departure, transfer nodes ... **Step 1a**
  - While parsing stop_times.txt create the following arcs
    - between arrival and departure nodes ("traveling arcs")
    - from arrival nodes to transfer nodes ("alighting arcs")
  - Create the corresponding nodes at the same time
  - **Note:** you can use entirely new nodes for each trip ... there is no need to share nodes between different trips



Implementation Advice 6/7

- Arrival, departure, transfer nodes ... **Step 1b**
  - While parsing stop_times.txt, also maintain for each station the list of arrival and transfer nodes of that station, with their time and type (arrival or transfer)

    Array<Array<Node>> nodesPerStation;

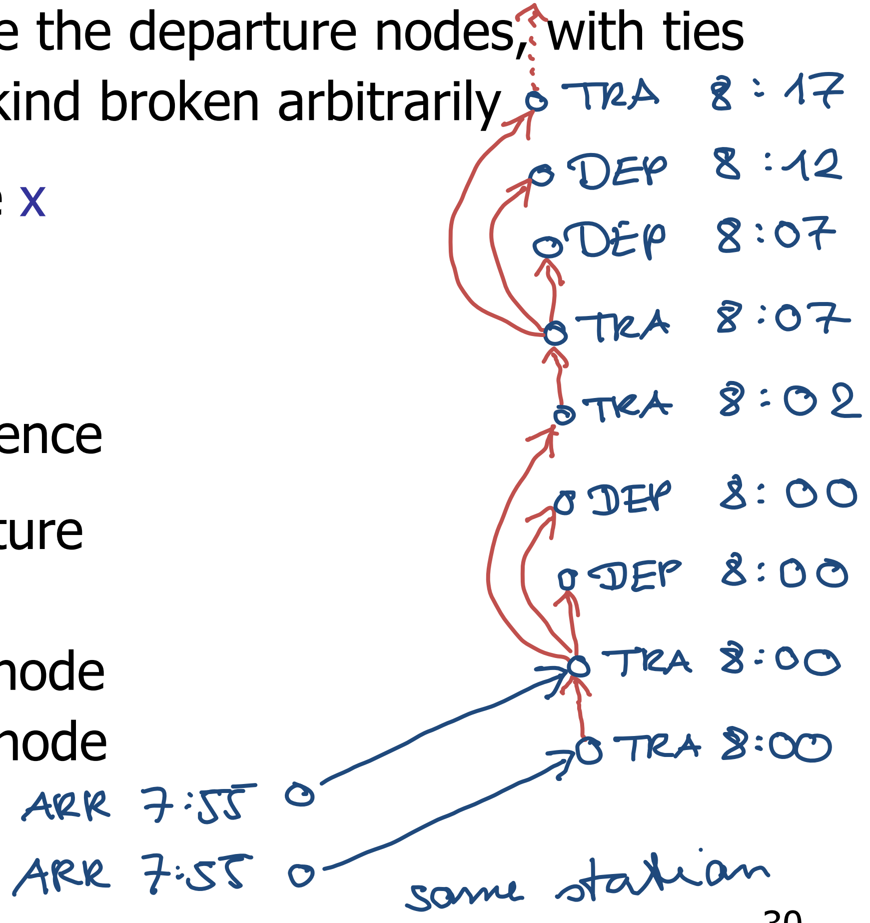
  - In GTFS the station ids are strings, but better convert them to consecutive station ids during the parsing of stops.txt ... remember the correspondence like this:

    HashMap<string, int> stationIdsByGtfsName;

  - It remains to add the following arcs:
    - from transfer nodes to departure nodes ("boarding arcs")
    - from one transfer node to the next ("waiting arcs")

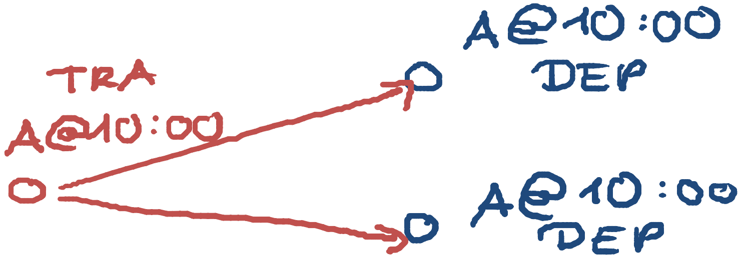
Implementation Advice 7/7

- Arrival, departure, transfer nodes ... **Step 2**
  - For each station: sort the nodes by time, and for equal times, sort the transfer nodes before the departure nodes, with ties between nodes of the same kind broken arbitrarily
  - Then for each **transfer** node x in the sorted sequence
    - add an arc to the next transfer node in the sequence
    - add an arc to each departure node that comes after x without another transfer node inbetween (none, if next node after x is a transfer node)

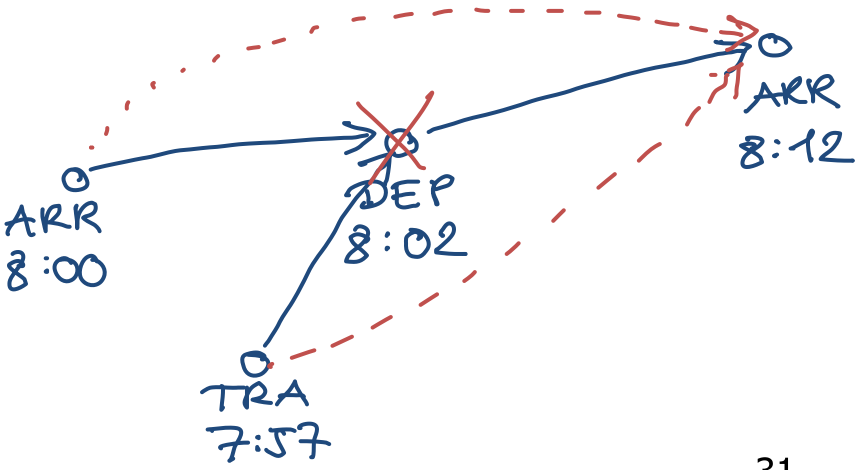


Some simple optimizations (not needed for Exercise)

- Less transfer nodes
  - If a station has several departure nodes at the same time, it suffices to add a single transfer node for all of them



- Contract departure nodes
  - This decreases the number of arcs that were incident to the departure nodes by a factor of 3/2



Road vs. Transit Networks

- Assume the time-expanded model
  - Then we can run all our algorithms so far also for transit networks ... and they will correctly compute shortest paths
  - But will the speed-up over ordinary Dijsktra be the same?
  - We will look at that in the next lecture ...

References

- Transit network models

  Timetable information: Models and Algorithms

  Müller-Hannemann, Schulz, Wagner, Zaroliagis, ATMOS 2007

  http://www.springerlink.com/content/x54715k627860283/

- GTFS
  - https://developers.google.com/transit/gtfs/
  - http://www.gtfs-data-exchange.com/