Efficient Route Planning

SS 2011

Lecture 9, Friday July 15th, 2011

(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 Ex. Sheet #6 (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 #7: Parse a transit network from GTFS and run Dijkstra on it, and if possible, your other alg's too

Feedback on ES#6 (transit node routing)

- Summary / excerpts Stand 15.7 12:59
  - Noch beim Debuggen von contraction hierarchies
  - Gerade Endsemesterstress bei vielen

Coding standards

- Sind jetzt auch für Java ausgearbeitet
  - Siehe Link auf Ihrer Daphne-Seite

    https://daphne.informatik.uni-freiburg.de/svn/CodingStandards/
  - Sie finden dort
    - Eine README.deutsch.txt
    - Ein vollständiges Code-Beispiel für C++
    - Ein vollständiges Code-Beispiel für Java
    - Für eigene Projekte, einfach den entsprechenden Ordner kopieren und Code schreiben, sollte dann gehen

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
  - Since we want to compute shortest path queries also for transit networks (best way to get from A to B), we want to model them as (directed) graphs, too, just like road networks ... but how?

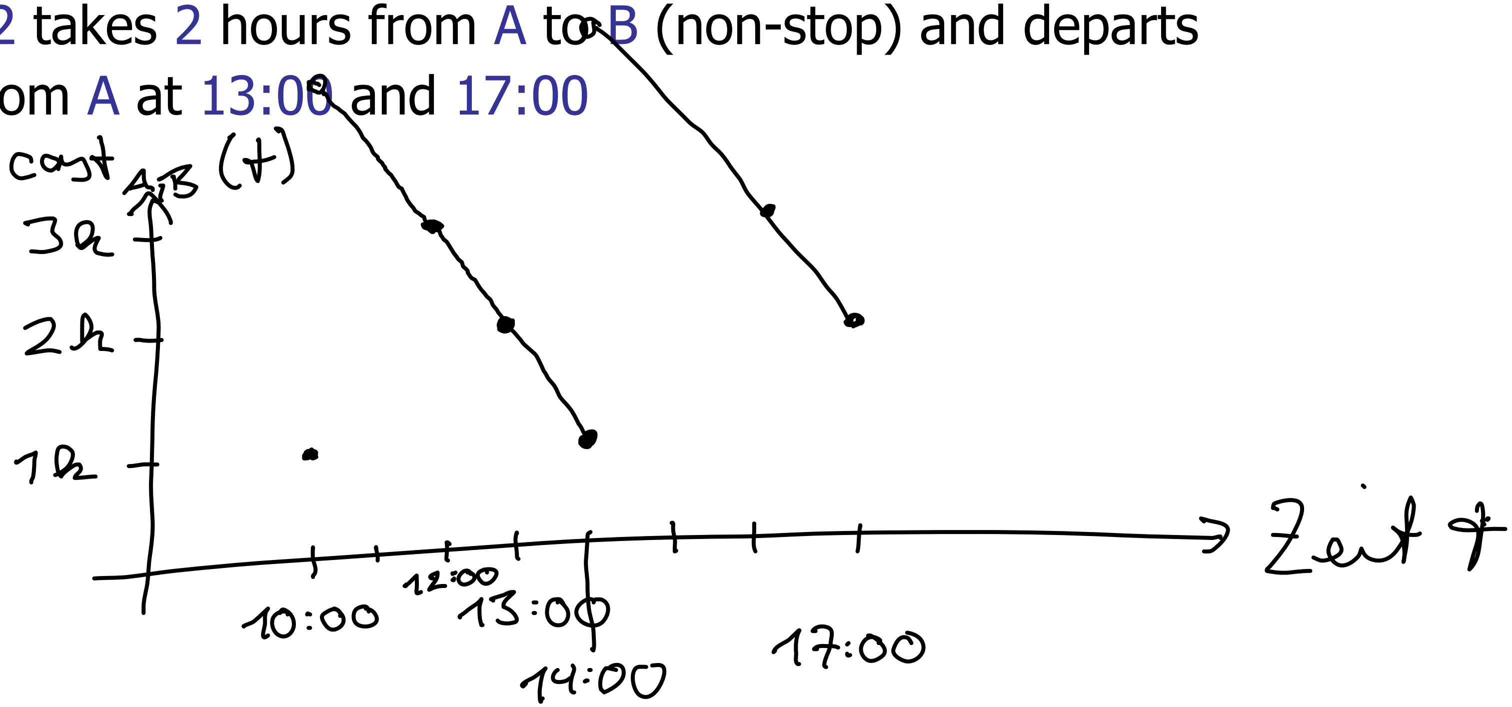
Time-dependent model 1/2

- 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)

    $cost_{u,v}(t)$ = the time to get from u at time t ... to v

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 2 hours from A to B (non-stop) and departs from A at 13:00 and 17:00

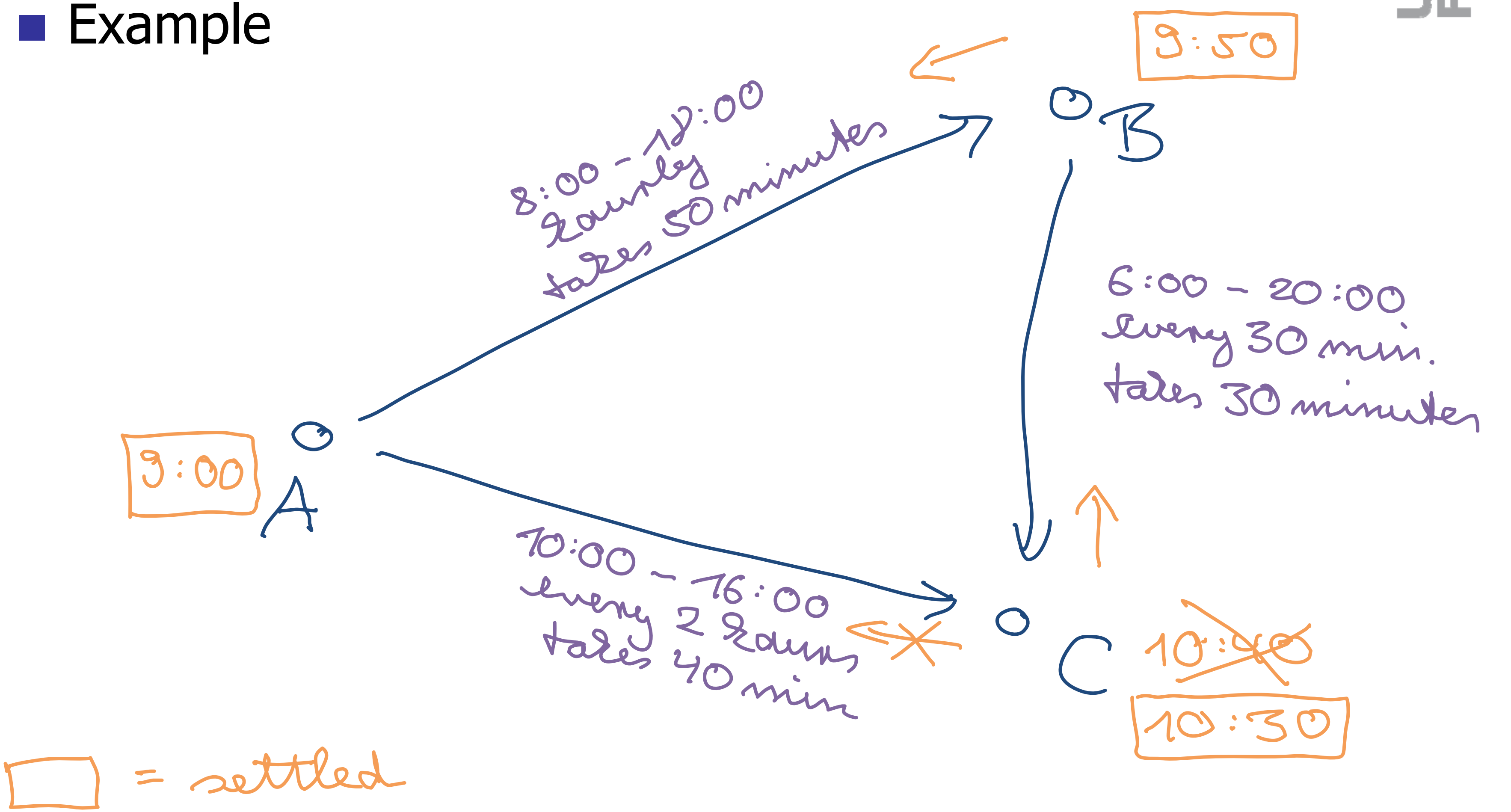


Time-dependent Dijkstra 1/2

- 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

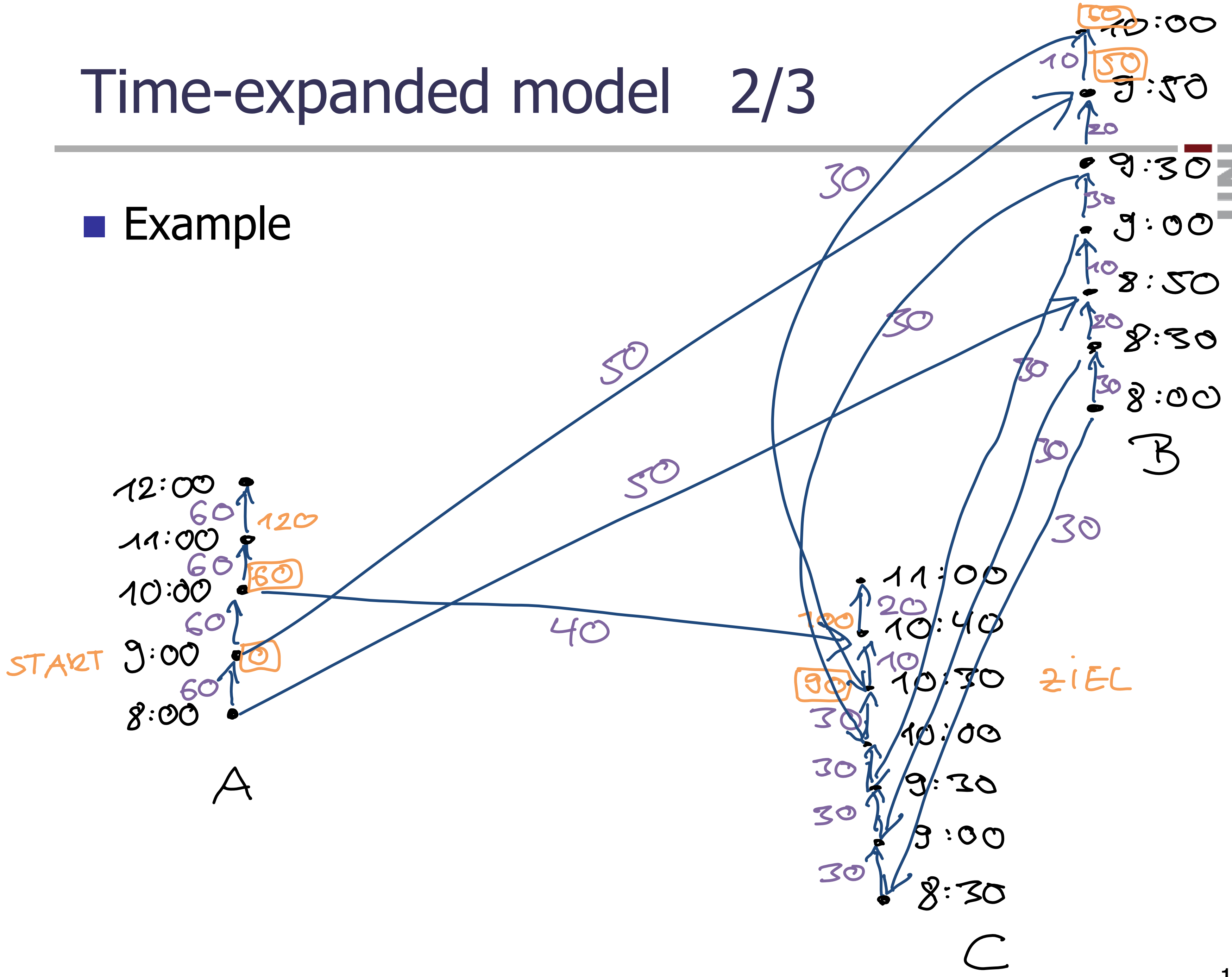


Time-expanded model 1/3

- 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 @ 13:57
  - 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 > t1 ... we call these **waiting** arcs

Time-expanded model 2/3

- Example



Time-expanded model 3/3

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

Time-expanded vs. time-dependent

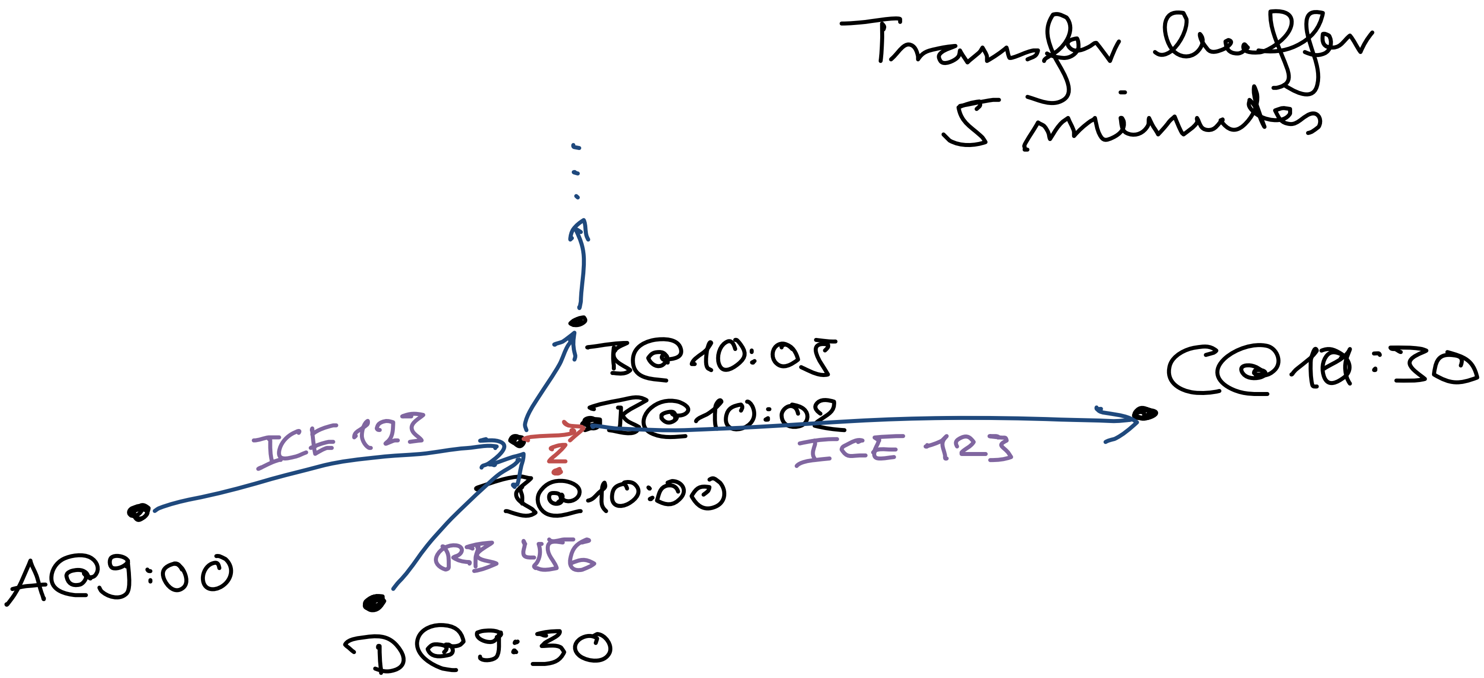
- So far, not much difference
  - Given a query A@t → B, consider the sequence of arcs relaxed by a (time-dependent) Dijkstra on the time-dependent graph
  - The Dijkstra on the time-expanded graph relaxes the same arcs in the same order, **plus** some additional waiting arcs to some additional nodes 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

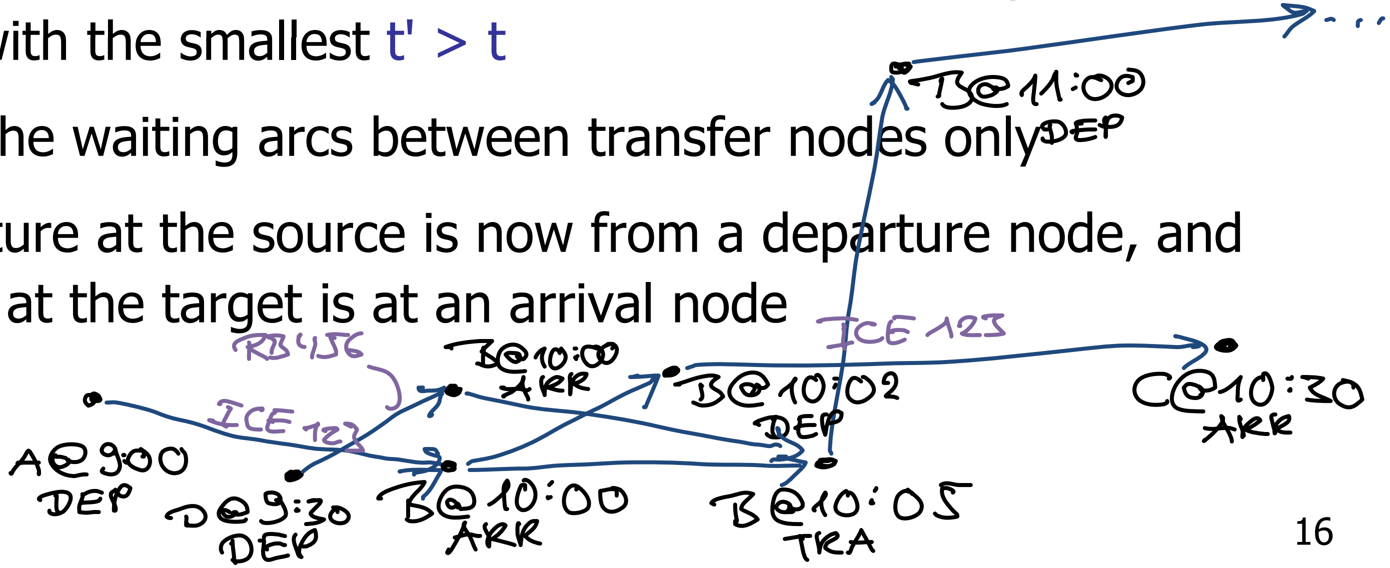
Transfer buffers 1/5

- Time-expanded model
  - This is non-trivial, because we need to distinguish between staying on a vehicle at a station (which must not require any transfer time) and changing the vehicle, for example:



Transfer buffers 2/5

- 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 **transfer node** A@t' and an arc from A@t to A@t', where t' – t is the transfer buffer
  - For each transfer node A@t, add an arc to the departure node A@t' with the smallest t' > 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 3/5

- 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

    $\mathrm{cost}_{u,v}(t, \ell)$ = time to reach v, if we are at u at time t sitting in vehicle ℓ
  - Unfortunately, Dijkstra's algorithm will not always correctly compute the shortest path anymore then

Transfer buffers 4/5

- Time-dependent model, problem

Transfer buffers 5/5

- Time-dependent model, solution 2
  - Have separate arrival and departure nodes, too
  - One arrival and one departure node per line suffices
  - But still, we no longer only have one node per station
- Time-dependent model, solution 3
  - 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 (Dijkstra maintaining several shortest paths to the same node), see next lecture

GTFS

- General Transit Feed Specification
  - Standard format established by Google in 2005
  - Here is a nice story about it: 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 Bertoldsbrunnen
  - trip = journey of a particular vehicle at a particular time
    - e.g. the journey of Bus 11 from Munzinger Straße at 9:28 to Paduaalle at 10:11
  - route = trips that have a common description
    - e.g. all journeys of Bus 11 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 we need for exercise sheet #7
  - 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 schedule only once in stop_times.txt, and the periodicity here
  - 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

- Write a class CsvParser
  - With a method readNextLine() that reads the next line from the file and makes the columns available via another method getItem(int i)
  - You find my CsvParser.h and CsvParserTest.cpp in the course SVN under folder lectures
  - Note: for efficiency reasons, I let my getItem method return a const char* which points to part of an internal string object containing the last line read

    It's slightly easier if you just return an std::string, but then you get one or two additional copies / allocations

Implementation Advice 2/7

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

    ```
    vector<Node> _nodes;
    vector<vector<Arc> > _adjacencyLists;
    ```

    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

- Parse the GTFS files in this order and way
  - 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 (= remembered) service id
  - Then parse stops.txt and store (in a hash map) the names and coordinates by stop id
  - Then parse frequencies.txt and store by trip id
  - 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

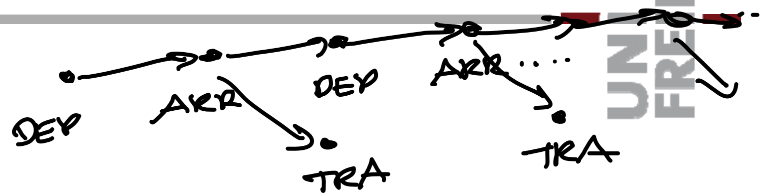
Implementation Advice 4/7

- Blocks with 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
  - 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, repeat accordingly, for example:

Implementation Advice 5/7

- Arrival, departure, transfer nodes



  – Create all nodes already while processing stop_times.txt

  – And also the arcs from arrival to departure nodes and vice versa, and from arrival nodes to transfer nodes

  – While processing stop_times.txt, maintain (in a hash map) for each station the list of nodes of that station, their time, and their type (arrival, departure, transfer)

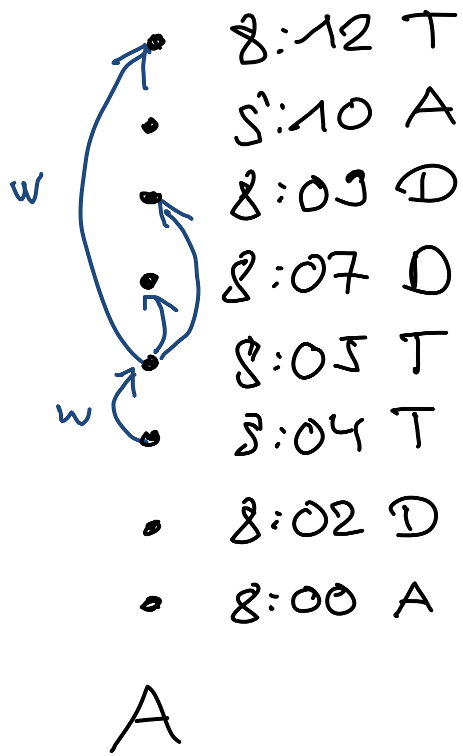
  – After processing stop_times.txt, for each station do:

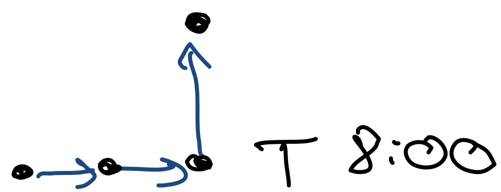
    Sort the nodes by time; then it is easy to add the missing arcs **from** the transfer nodes (waiting arcs to the next transfer node, and boarding arcs to the next departure node)

    - Beware: several nodes with exactly the same time

Implementation Advice 6/7

- Arcs **from** transfer nodes, example





Implementation Advice 7/7

- Tricks to save some arcs
  - We can trivially **contract** all departure nodes
  - This replaces pairs of a boarding and a traveling arc by a single arc ... and actually **descrease** the total number of arcs in the graph, for example:

Road vs. Transit Networks

- Assume the time-expanded model
  - Then we can all our algorithms so far also for transit networks
  - But will the speed-up over ordinary Dijsktra be the same?

References

- Transit network models

  Timetable information: Models and Algorithms

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

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

- Road Networks vs. Transit Networks

  Car or Public Transport — Two Worlds

  Hannah Bast, Efficient Algorithms 2009, LNCS 5760

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

- GTFS
  - http://code.google.com/transit/spec/transit_feed_specification.html