Efficient Route Planning

SS 2011

Lecture 1, Friday May 6th, 2011

(Introduction, Organizational, Dijkstra, A*)

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

UNI FREIBURG

Overview of this lecture

- Introduction
  - Demos + what you will learn in this course
- Organizational
  - Style of the course
  - Course Systems: Wiki, Forum, Daphne, SVN, Jenkins, ...
  - Exercises + Exam
- And then let's start
  - Modelling road networks as graphs
  - OpenStreetMap data
  - Dijkstra and A*
  - Exercise sheet for this week

Demos + what you will learn

- Google Maps
  - Demo for road networks
  - Demo for transit networks
  - at the end of the course you will be able to build something like this ... and maybe even better
- What you will learn in this course
  - How to model road and transit networks
  - Where to get good data
  - Efficient algorithms for route planning on these networks
  - How to build a web application around this

Style of this course

- What I will do
  - Provide the framework for this course
  - Explain models, data, and the various algorithms
- What you will do
  - Implement the algorithms
  - Do experiments
  - Explore variations / new ideas
  - Read some papers from time to time
  - Some theoretical tasks ... but not too many

Course systems

- Various systems supporting this course
  - The course Wiki is the hub page with links to each of the following
  - Daphne is our course management system
  - There is an SVN repository for your submissions, in particular for your code
  - There is a Forum for asking questions
  - All the course materials will be put online (links on the Wiki): the lecture slides, the exercise sheets, the lecture recordings, any code we write in the lectures
  - We will also provide a continuous build system (Jenkins) that automatically checks the code you commit to our SVN

Exercises + Exam

- There will be one exercise sheet per week
  - Usually a practical one
  - You can work on the sheets in groups of 2-3 people
  - Submit the code to our SVN **show how to register**
  - Follow some basic guidelines for coding → next slide
  - There is no right or wrong for the exercise sheets but you will get points for your effort
  - Each group must provide master solutions at least once
- Exam in the end
  - Will be written or oral, depending on the #participants
  - You need 50% of the points to be admitted

Code

- Please follow these guidelines when writing code
  - Write your programs in C++ or in Java
  - Follow a stylesheet
    - for C++ you find a style checker cpplint.py when you check out your subdirectory from our SVN
  - Write unit tests for all major functions / methods
    - otherwise all the results you produce are wrong with high probability
  - Provide a standard Makefile / Antfile for compilation
  - Document each class and each method

Road networks

- Model as graph
  - each crossing of two or road segments is a node in the graph
  - each road segment is a directed arc in the graph
  - in the simplest model, the cost of an arc is the time to travel along the corresponding road segment

Shortest Path Queries

- Point to point queries
  - For the first lectures, we are interested in finding the shortest path (path of minimal cost) between two given nodes A and B, called source and target node
  - The cost of a path is simply the sum of the costs of the arcs along the graph

OpenStreetMap

- OpenStreetMap (OSM)
  - Is an open-source initiative for gathering geo data
    - not only road network data; e.g. also all kinds of other map data
  - Started in 2004, quite good coverage by now
    - 1 billion nodes, many 100 billions of arcs (May 2011)
  - Data can be downloaded for free **show it**
  - For now we (in particular for Exercise Sheet 1) we need
    - nodes (each with a latitude and a longitude)
    - ways (several arcs together) with <tag k="highway" ...>
    - See Wiki for translations of highway types to speeds

Travel time along an arc

- The OSM data provides node coordinates ...
  - and road types, from which we can infer speeds
  - This gives us travel time via the formula

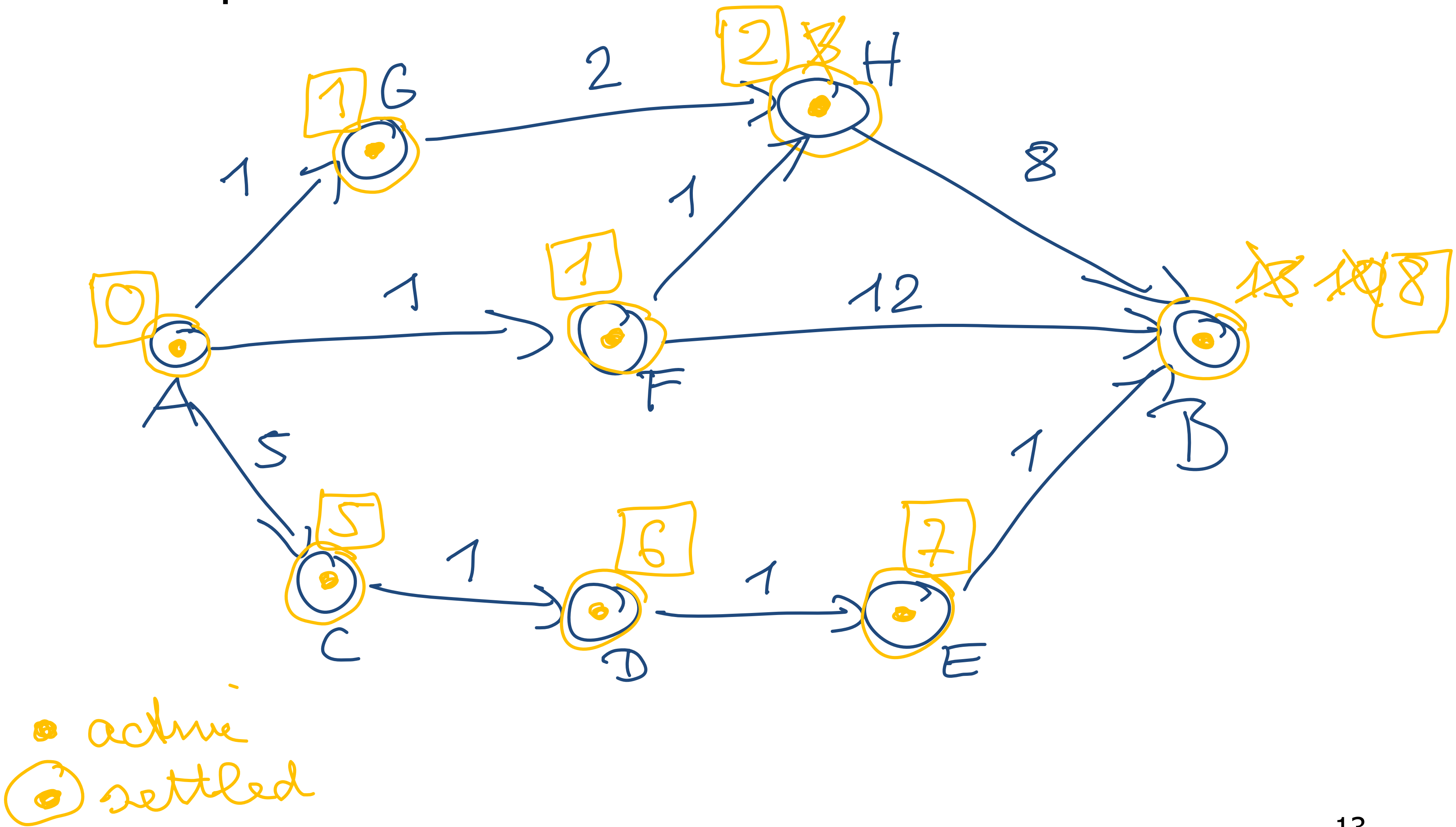
    speed = distance traveled / travel time   (v = s / t)
- How to get the distance between two nodes?
  - The obvious formula is the euclidean distance between the two points
  - However, note that the path between two points on the earths surface is not a straight line, but follows a so-called great circle (Großkreis)
    - http://en.wikipedia.org/wiki/Great_circle
    - but ok to use Euclidean distance for Exercise Sheet 1

Dijkstra's algorithm

- Quick recap
  - Maintains a priority queue of active nodes with tentative distances
  - Initially only the start node is active, with tentative distance 0, all other tentative distances are ∞
  - In each iteration, pick the active node with the smallest tentative distance and change its status from active to settled
    - if all arc costs are non-negative, the tentative distance of each settled node is guaranteed to be the correct distance
  - Relax the outgoing arcs = see if the tentative distances of the adjacent nodes can be improved, if yes do so
  - Stop when the target node is settled

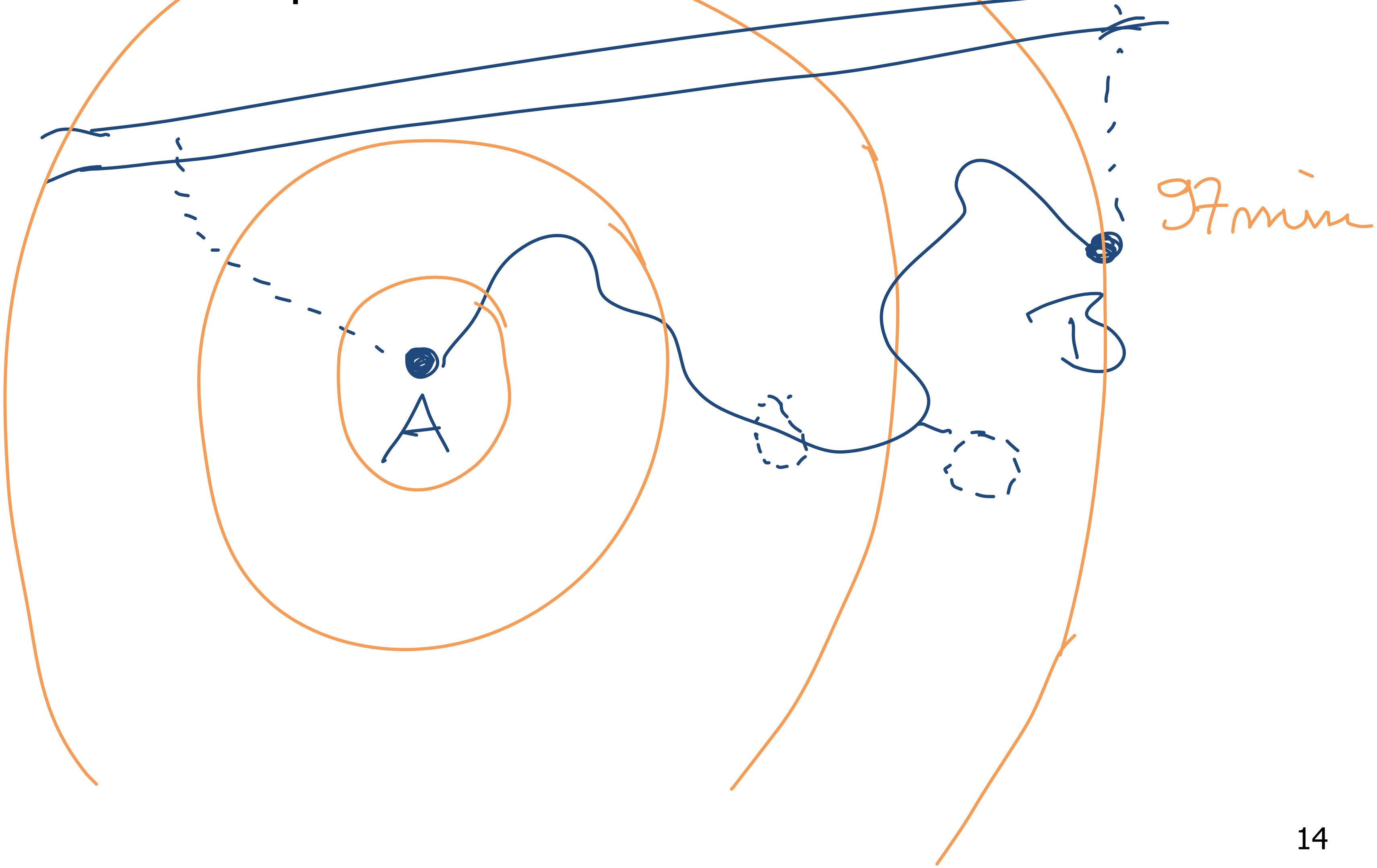
Dijkstra's algorithm

- Example execution



Dijkstra on road networks

- Schematic picture

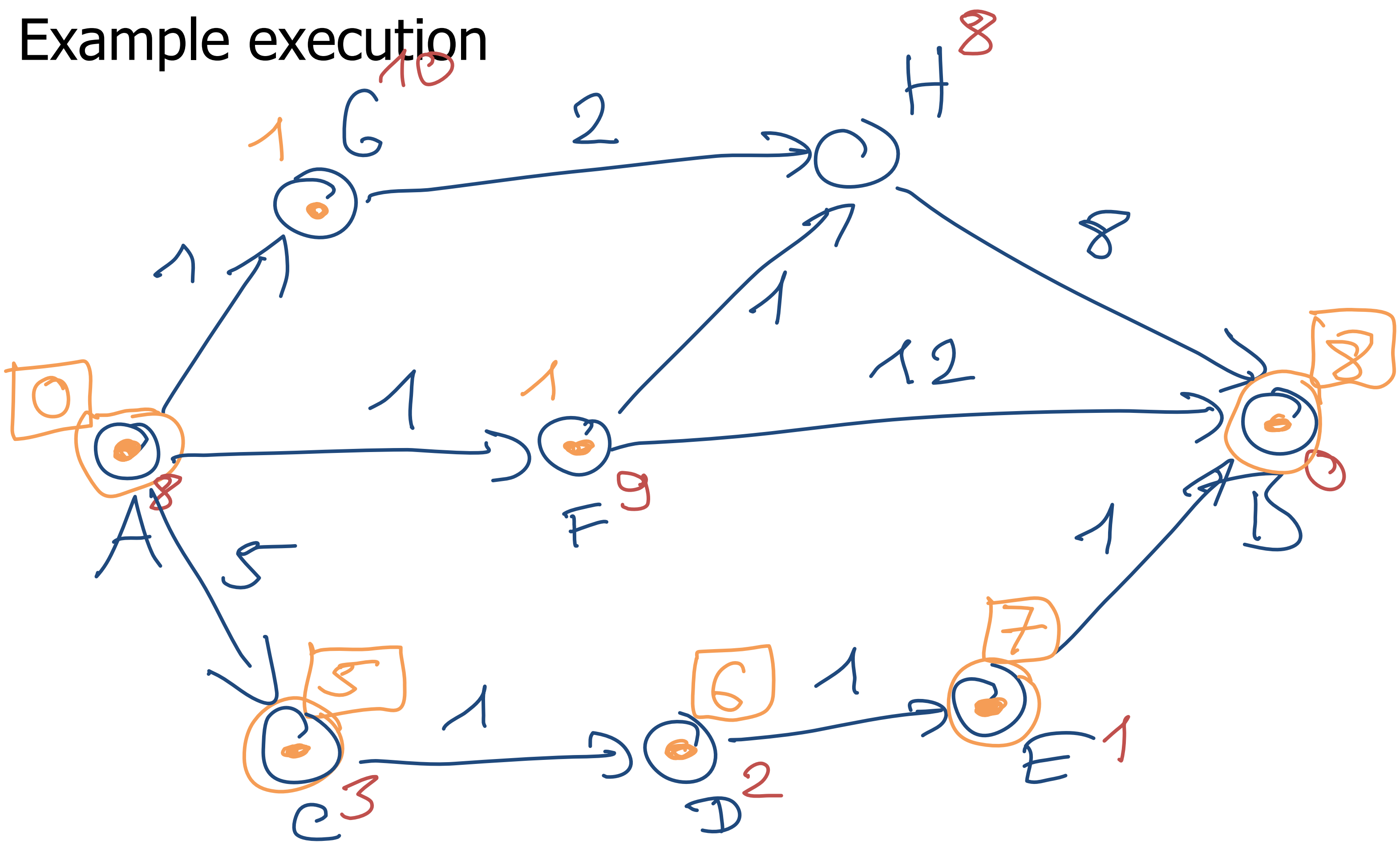


A* algorithm

- A* is a simple extension of Dijkstra
  - In addition to the arc costs, we have a heuristic value h for each node that estimates the cost from there to the target
  - A* then proceeds like Dijkstra, except that the keys with which an active nodes is put into the priority queue is not its tentative distance, but its tentative distance plus its h value
  - If for each node, the value h is ≤ the true cost to the target, the algorithm is correct = it will find the shortest path from the source to the target
  - if for each node, the values h is 0, we have plain Dikjstra
  - if for each node, the value h is the exact distance to the target, A* performs the least number of operations

A* algorithm

- Example execution



A* heuristic for road networks

- Straight-line distance
  - Also called "as the crow flies" distance ("Luftlinie")
  - The straight-line distance from a node to the target, divided by the maximum speed, certainly gives a lower bound on the travel time along an optimal path from that node to the target
  - **Let's see (in the exercise) how much that helps!**

Day and time for the next lectures

- Next Friday (May 13)
  - The lecture starts at 2.45 pm (and goes for one hour only)

    ("Begehung Akkreditierung", meeting of the Akkreditierungs-Board with all the professors)
  - Or is there consensus on a better day and time?

References

- OpenStreetMap
  - http://www.openstreetmap.org/
  - http://en.wikipedia.org/wiki/OpenStreetMap
  - http://wiki.openstreetmap.org/wiki/XML
  - http://wiki.openstreetmap.org/wiki/Data_Primitives
  - http://wiki.openstreetmap.org/wiki/Map_Features
- Dijkstra's algorithm and A*
  - http://en.wikipedia.org/wiki/Dijkstras_algorithm
  - http://en.wikipedia.org/wiki/A*_search_algorithm