Efficient Route Planning
SS 2012

Lecture 4, Wednesday May 16th, 2012
(Arc flags, Visualization)

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

UNI FREIBURG

Overview of this lecture

- Organizational
  - Feedback and results from Exercise Sheet 3 (A-Star)
- Arc Flags algorithm
  - A method for very strong **goal direction**
  - How to compute the actual shortest path
    (that is, the arcs along the path and not just the total cost)
- Exercise Sheet 4
  - Implement a part of Arc Flags (single region) and
    - … run some queries as usual
    - … **visualize** the search space for one query

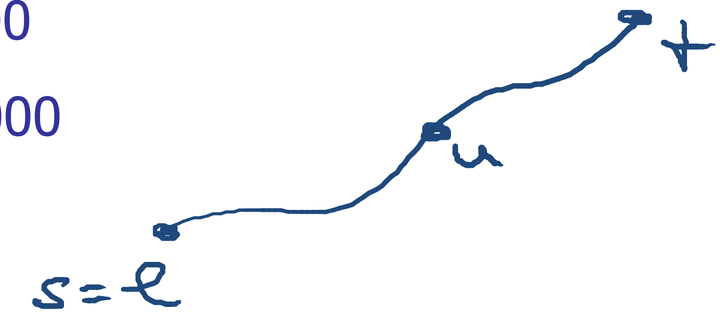
Your Feedback on Ex. Sheet 3 (A-Star)

- Summary / excerpts — last checked May 16, 16:14
  - 6 – 9 hours was typical, a few needed less, some more
  - Some used quite some extra time for refactoring old code
  - Implementation advice in the lecture was useful again
  - Feedback from the tutors was much appreciated again
  - Rounding in A-Star can impact admissability / monotonicity
    - rounding arc costs **up** always works

      $h(u) \leq c(u,v) + h(v)$
  - Question about landmarks: one standard Dijkstra per landmark, or one set Dijkstra for all landmarks together?
    - It's one Dijkstra per landmark, no set Dijkstra here
    - The set Dijkstra was just for selecting a set of landmarks that are as "far apart" from each other as possible

Experimental results from ES 3 (A-Star)

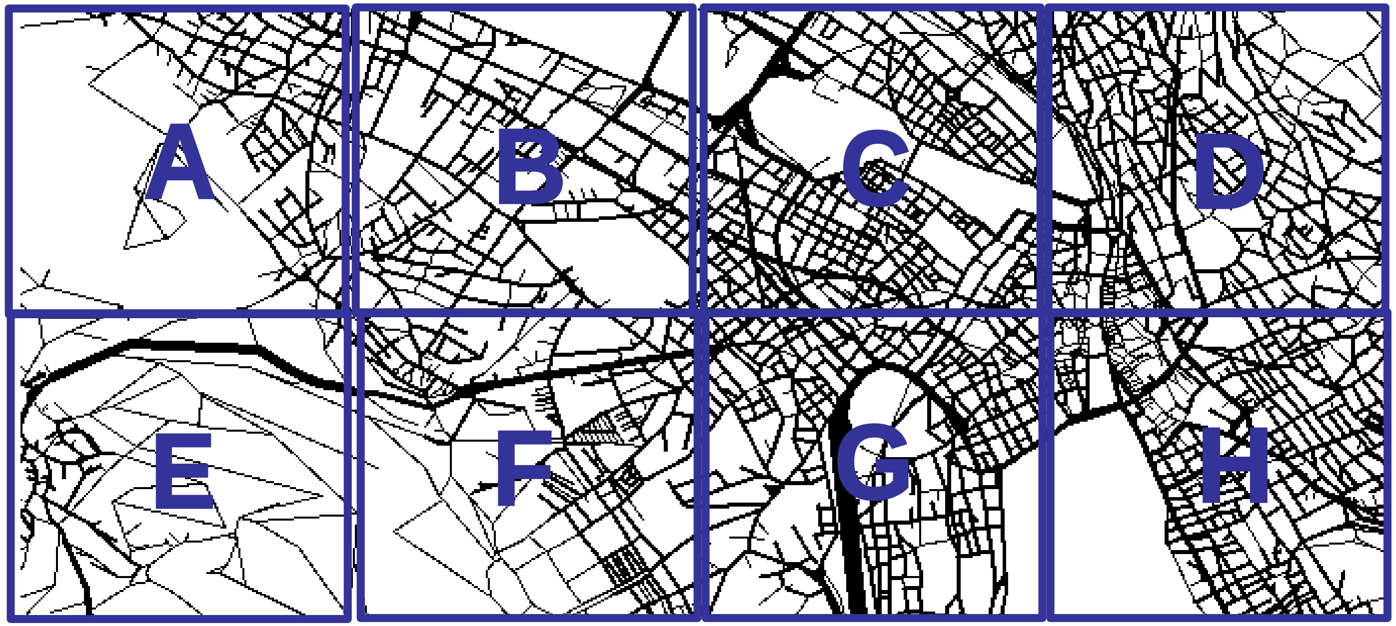
- See the table on the Wiki
  - #settled nodes (= size of search space) much decreased:
    - Dijkstra: 100,000 / 1,200,00 (Saarland / BaWü)
    - A*-Straight: 50,000 / 500,000
    - A*-Landmarks: 5,000 / 50,000
  - query times accordingly
    - Dijkstra: 20ms / 500ms
    - A*-Straight: 20ms / 250ms
    - A*-Landmarks: 1ms / 15ms
  - Bottom line: A*-Straight helps a little, A*-LMs helps **a lot**



$h_\ell(u) = |\,dist(s,t) - dist(s,u)\,|$
$= dist(u,t)$ PERFECT

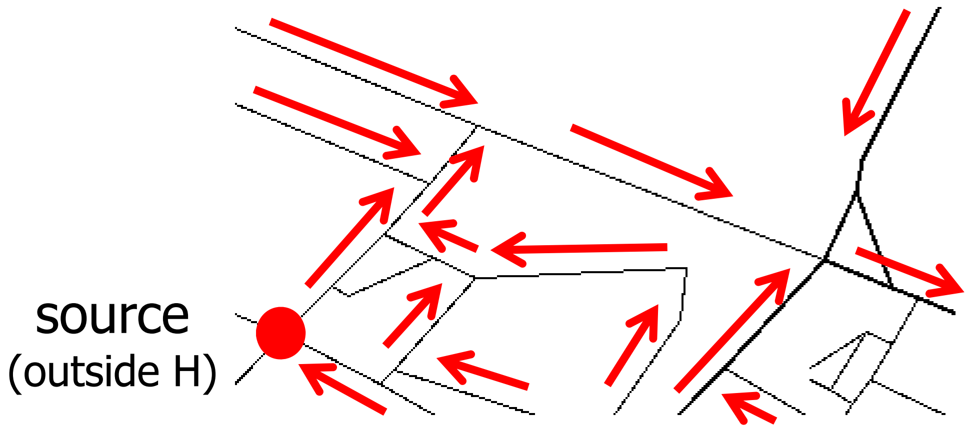
Arc flags — Basic idea 1/2

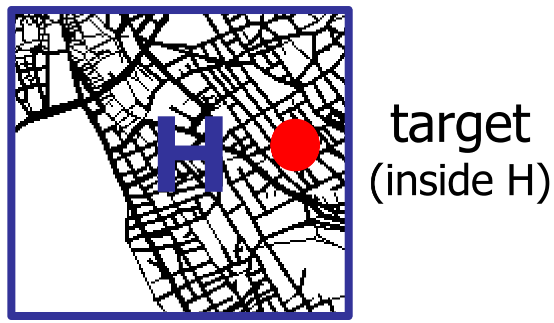
- Precomputation
  - Divide the map into "compact" regions of about equal size
  - For each arc, compute "direction signs" for each region
  - We call these direction signs **arc flags**



Arc flags — Basic idea 2/2

- At query time
  - Determine the region containing the target node
  - In Dijkstra's algorithm, **outside** of that region, consider only arcs with direction signs towards that region

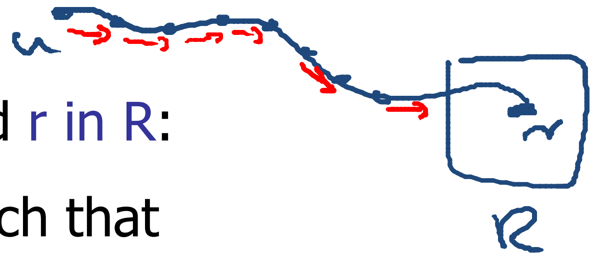




Arc Flags – Formal Definition 1/2

- Properties of the arc flags
  - Each arc has **one arc flag per region**, which is 0 or 1
  - The arc flags for a fixed region R (one per arc) must have the following properties:
    - for nodes u and r, with u arbitrary and r in R:

      there is a shortest path from u to r such that the flags of **all** arcs on that path are set to 1
    - Note: there may be several shortest path from u to r; it is enough that **one of them** has this property
  - Several ways to compute such arc flags ... later slides



Arc Flags – Formal Definition 2/2

- Query algorithm
  - Given a query from a node s to a node t, both arbitrary
  - Determine the region R containing t
    - this requires that each node lies in **some** region
  - Execute an ordinary Dijkstra on the subgraph formed by those arcs with flags for R set to 1
    - this could be implemented by making a **copy** of the graph, where we only consider those arcs
    - but it is equivalent, and more efficient, to simply **ignore** the arcs with flags for R set to 0
    - see implementation advice on later slide

Arc Flags — Correctness

- Consider a query from s to t, both arbitrary
  - Let R be the region containing t
  - Given the properties of the arc flags:
    - there exists a shortest path from s to t such that the flags for R on all arcs on that path are 1
  - Hence that path also exists in the subgraph consisting only of those arcs with flags for R set to 1
  - Since we only remove arcs and don't add any, there can't be a better path in the subgraph
  - Hence Dijkstra will find that path, or one with equal cost

Arc flags — Precomputation 1/7

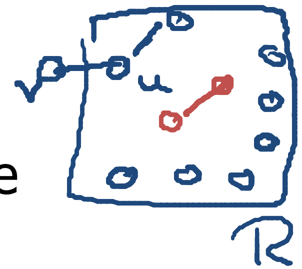
- **Naive way:** For each region R do the following
  - Do the following for **each** node r in R:
    - Run Dijkstra starting from r in the reverse graph, until all nodes (reachable from r) are settled
    - This gives us the shortest path from each node u in the graph to r ... how to obtain **paths** → later slide
    - Set flag for each arc that is on one of these paths
  - This obviously fulfills the arc flags property, recall:
    - for each u and r, with u arbitrary and r in R
    - there must be a SP from u to r with all flags set
  - The above algorithm computes one such SP for each u and r

Arc flags — Precomputation 2/7

- Cost of this naive precomputation
  - One Dijkstra for each node in each region
  - This is one Dijkstra for each node in the graph
  - The cost of each Dijkstra is $\sim m \cdot \log n$
    - where $m = \#arcs$ and $n = \#nodes$
  - This is cost $\sim n \cdot m \cdot \log n$ overall
  - Even when $m = \Theta(n)$ that is **quadratic** in n
  - That would be **infeasible** already for BaWü

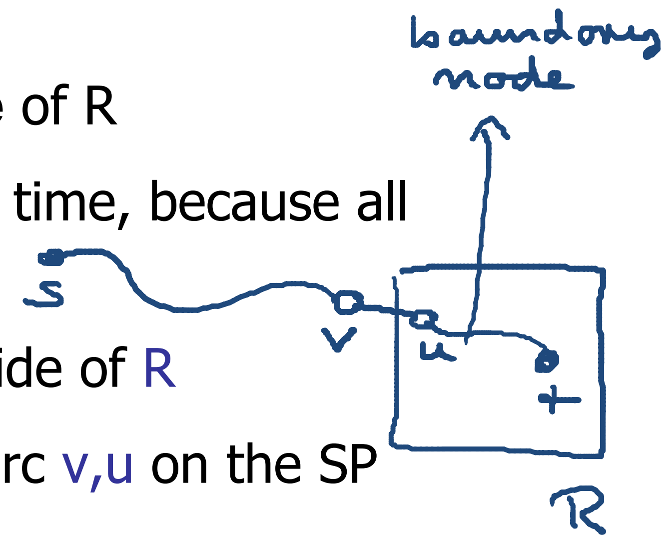
Arc flags — Precomputation 3/7

- **Better way:** For each region R do the following
  - Compute the set of boundary nodes of R:
    - a boundary node is a node u in R with at least one arc u,v such that v not in R
  - As before, but now only for each boundary node r:
    - Run Dijkstra starting from r in the reverse graph, until all nodes (reachable from r) are settled
    - Set all flags on all shortest paths thus computed
  - Additionally, set flags of all arcs u,v **inside** of R
    - an arc u,v is inside of R if **both** u and v are in R



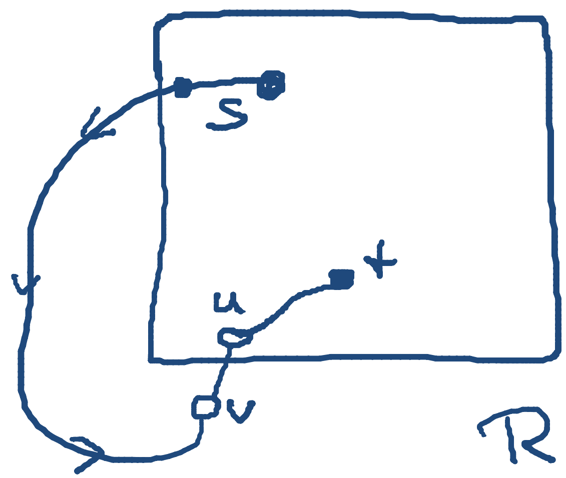
Arc flags — Precomputation 4/7

- Correctness of this "better way":
  - Consider a query from s to t, with s arbitrary and t in R
  - Consider any SP from s to t
  - **Case 1: all** arcs on that SP are **inside** of R
    - Then this SP will be found at query time, because all arcs inside of R are set
  - **Case 2:** not all arcs on that SP are inside of R
    - Since t is in R, there must be one arc v,u on the SP with v not in R and u in R
    - The subpath from s to u is a SP from s to u
    - The precomputation for u will find this path or a path of equal cost and set the flags of all arcs on it

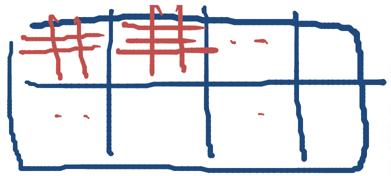


Arc flags — Precomputation 5/7

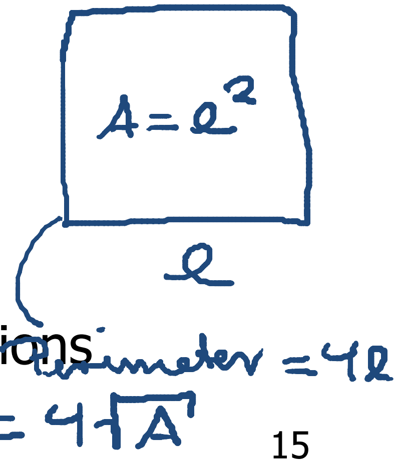
- Finer points of this argument
  - Note that even if s and t both lie in R, both cases can happen:



Arc flags — Precomputation 6/7



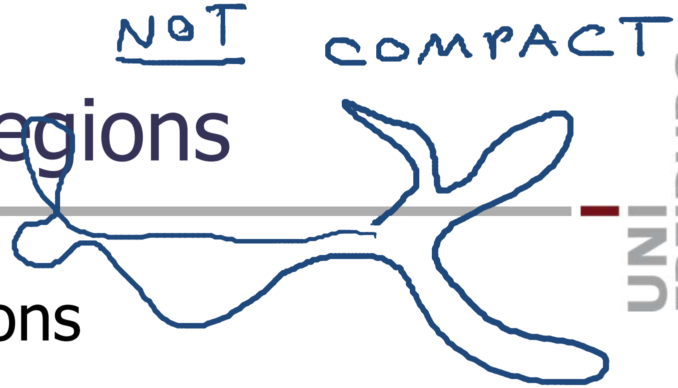
- Precomputation costs of the "better way":
  - We now have one Dijkstra **per boundary node**
  - So the total cost is $\sim b \cdot n \cdot \log m$
    where b = #boundary nodes, n = #nodes, m = #arcs
  - The size of b depends on the division into regions
  - Here is an estimate, if we divide into k **square** regions
    and assuming that the nodes are equally distributed
    - each region contains $\sim n/k$ nodes
    - of those, $\sim 4 \cdot (n/k)^{1/2}$ lie on the boundary
    - hence $b \sim 4 \cdot (n \cdot k)^{1/2}$
  - Hence total cost $\Omega(n^{3/2} \cdot \log m)$ even for $\Theta(1)$ regions



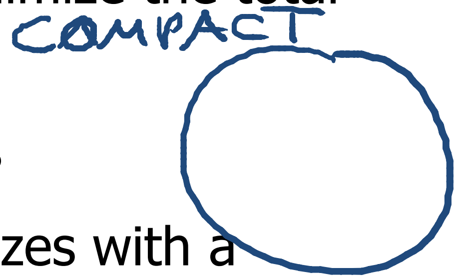
Arc flags — Precomputation 7/7

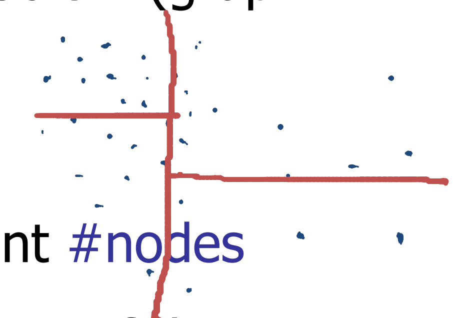
- Space consumption for storing the arc flags
  - Assume we have k regions, then we need k bits per arc
    - That is $k/8 \cdot m$ Bytes, where $m = \#arcs$
  - Let's compare that to the storage needed for the graph
    - 12 bytes per node (OSM id + latitude + longitude)
    - 8 bytes per arc (head node id + cost)
    - That is $12n + 8m$ bytes, where $n = \#nodes$, $m = \#arcs$
    - For road networks we have $m \approx 2.25n$
    - That is, we need about 13 bytes / arc for the graph
    - So for $k > 100$ the arc flags start to become expensive also storage-wise

Arc flags — Division into regions



- What is a good division into regions
  - For a fixed number of regions we want to minimize the total number of boundary nodes
    - Intuitively, this calls for "compact" regions
  - Dividing a graph into k subgraphs of similar sizes with a minimal number of boundary nodes is a hard problem (graph partitioning)
  - Rectangular regions are ok, but not optimal
    - for road networks, can contain widely different #nodes
  - Something like a KD-tree gives an even distribution of the #nodes / region, but not necessarily a small #boundary nodes

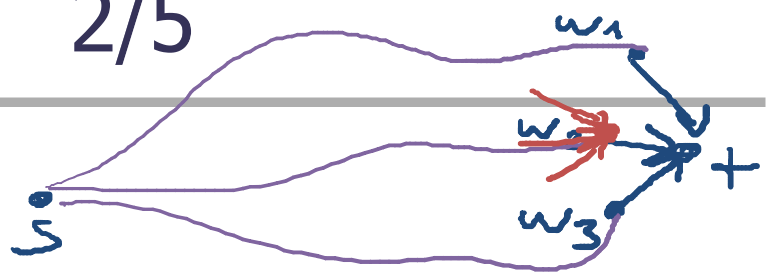




Paths, not only costs 1/5

- So far we only computed SP **costs**, not the **paths**
  - For the arc flags precomputation we need the paths
  - Any route planning system will want to output paths
  - So how do we get the actual paths?

Paths, not only costs 2/5



- There is a generic way
  - Assume we have stored the dist(s, u) for **all** nodes u that we have settled in a Dijkstra / A* computation from s to t
  - Then we can compute an SP from s to t as follows:
    - Consider the set W of all nodes w such that w was settled in the computation above and an arc w,t exists
    - Note that there will be at least one such w, namely the node from which t got its label by relaxing
    - Compute $v = \text{argmin}_{w \in W}\ \text{dist}(s, w) + \text{cost}(w,t)$
    - Then v is a predecessor on an SP from s to t
    - Now repeat with v in place of t ... until s is reached

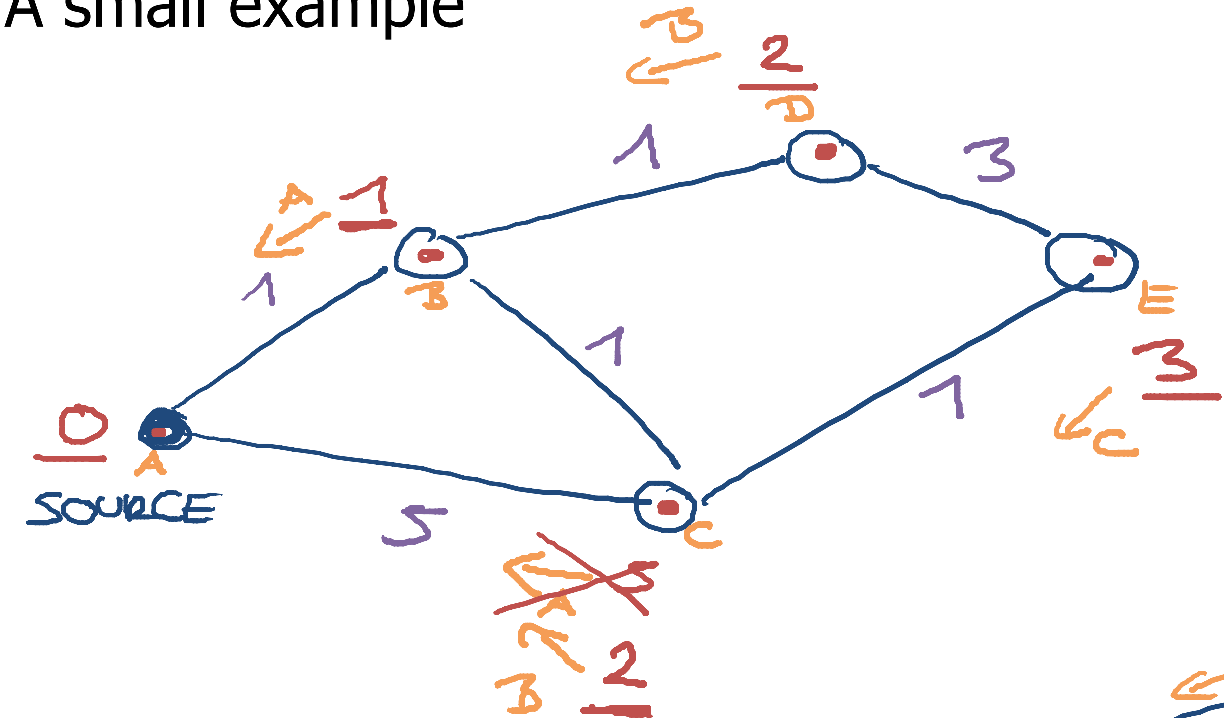
Paths, not only costs 3/5

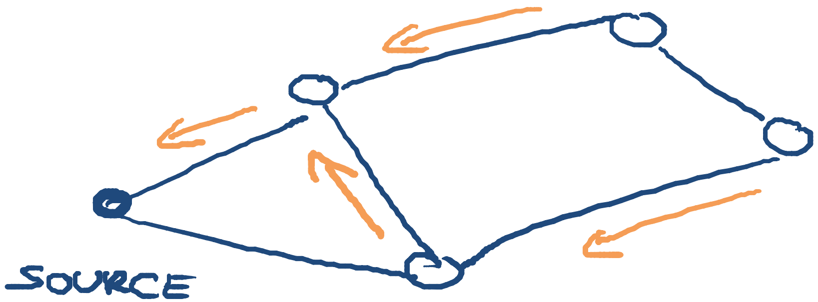
- But easier to compute this **during** Dijkstra
  - Along with the **dist** value for each node
  - Also maintain a **parent pointer** for each node
  - This is simply the id of the node from which the current dist value comes via relaxation

    (Initialize to some non node id, for example -1)
  - By the correctness proof of Dijkstra / A*, the parent pointer of each **settled** node u than points to the predecessor on a shortest path from s to u
  - That is, this pointer exactly points us to the v computed with the argmin on the previous slide

Paths, not only costs 4/5

- A small example

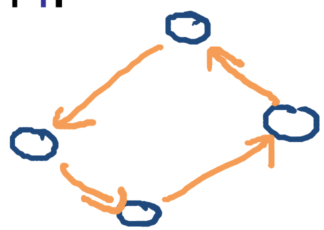




Paths, not only costs 5/5

- The parent pointers form a **tree** rooted at s
  - This can be proven by by a simple extension of our correctness proof for Dijkstra / A*

    (assuming the same order of nodes u_1, u_2, u_3, ...)
  - Namely, we can prove (by induction) that in iteration i:
    - u_i is settled
    - $dist[u_i] = dist(s, u_i)$
    - $parent[u_i]$ = the predecessor of u_i on an SP from s to u
  - This implies that there can be no cycles not containing s
  - And no cycles containing s either, because s has no parent

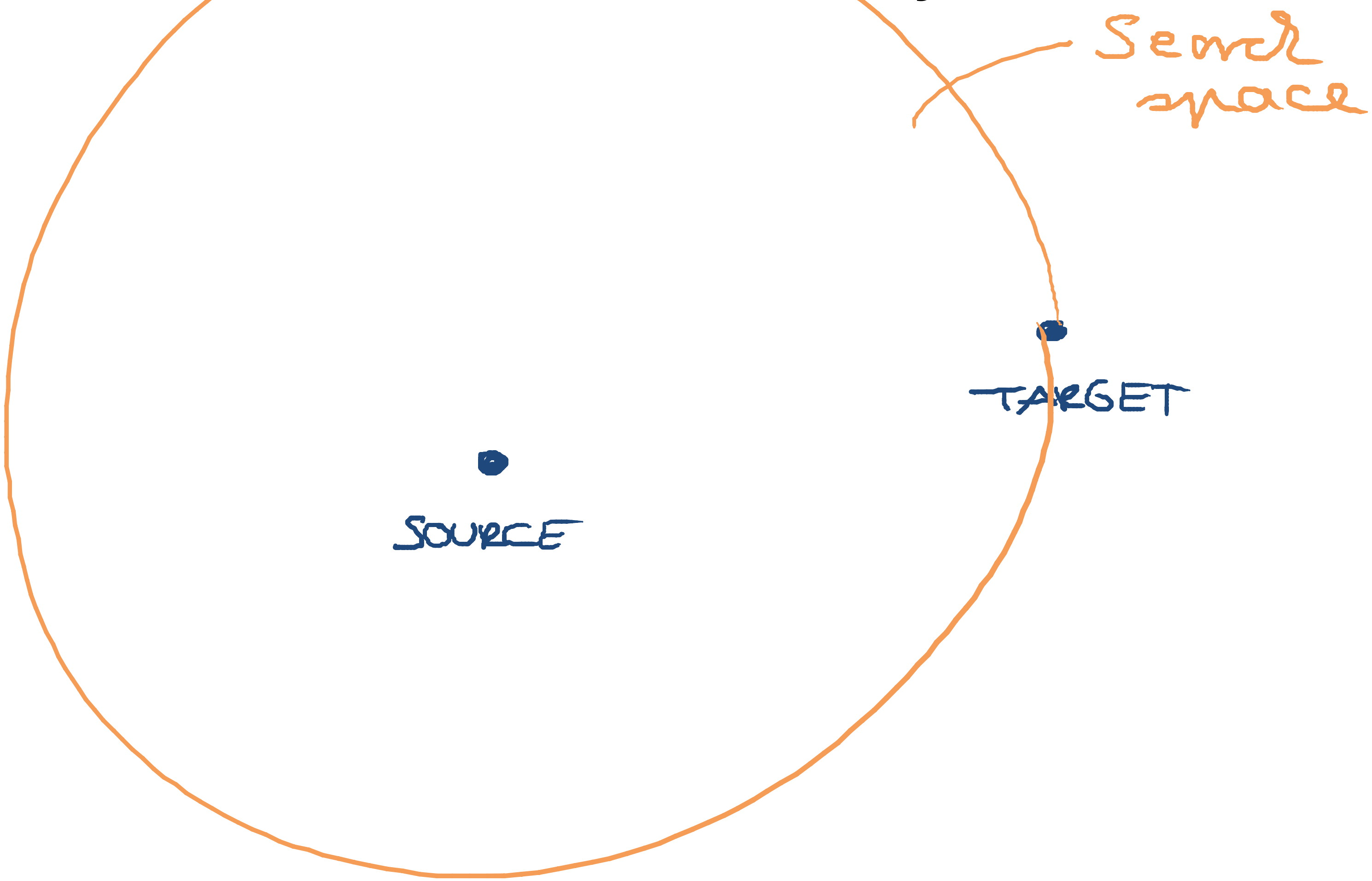


Arc Flags — Implementation Advice

- For the Ex. Sheet: a **single rectangular** region R
  - Write a new class ArcFlagsAlgorithm
  - Ok to compute the boundary nodes in the trivial way
    - iterate over all **arcs** u,v and mark u as boundary node if u in R and v not in R
  - Execute **one** Dijkstra **per** boundary node
  - Add a member variable arcFlag to your Arc class
  - Extend your class DijkstrasAlgorithm by a mode that relaxes an Arc only if the arcFlag is set ... that's trivial
  - See the code design suggestion on the Wiki
  - New stuff is commented with // NEW(lecture-4): ...

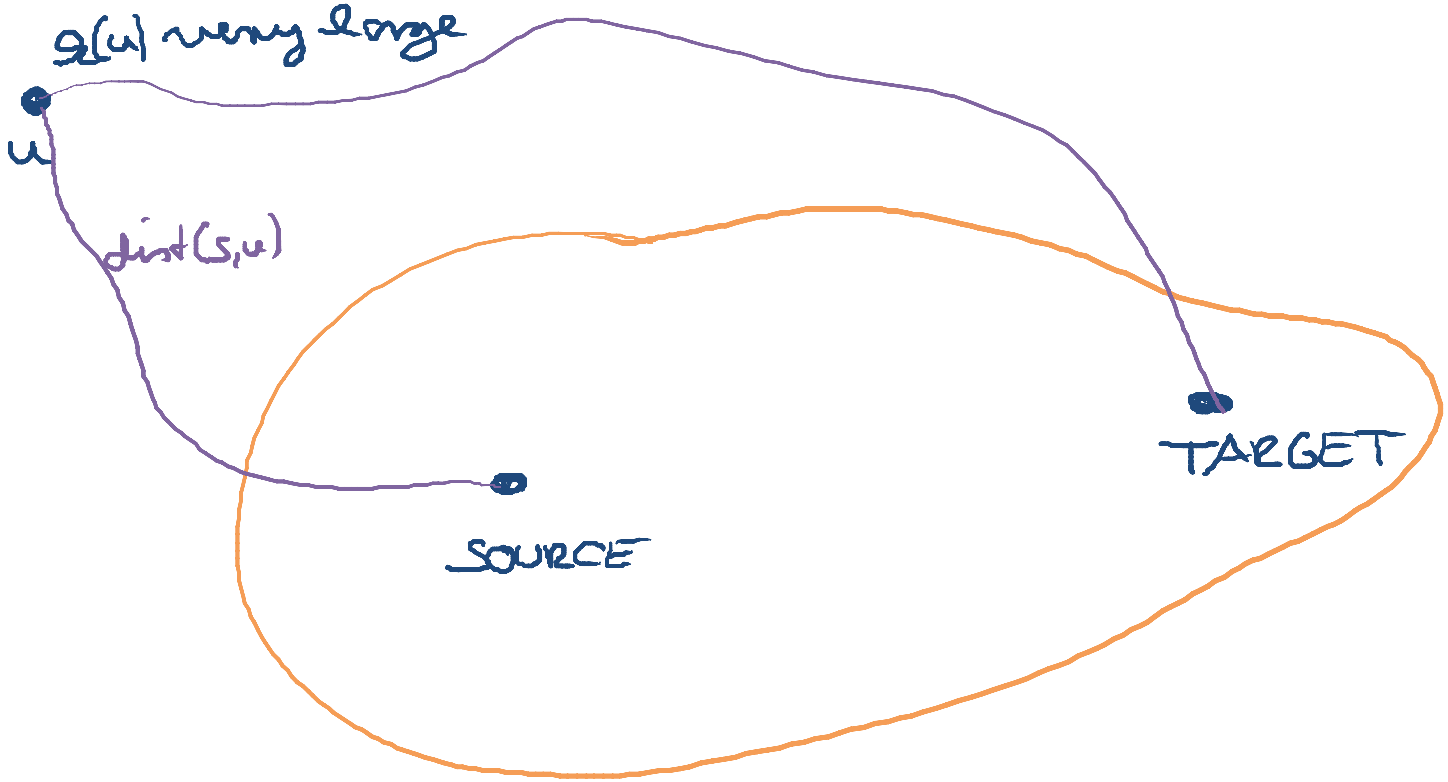
Search space comparison 1/3

- Set of settled / visited nodes for Dijkstra



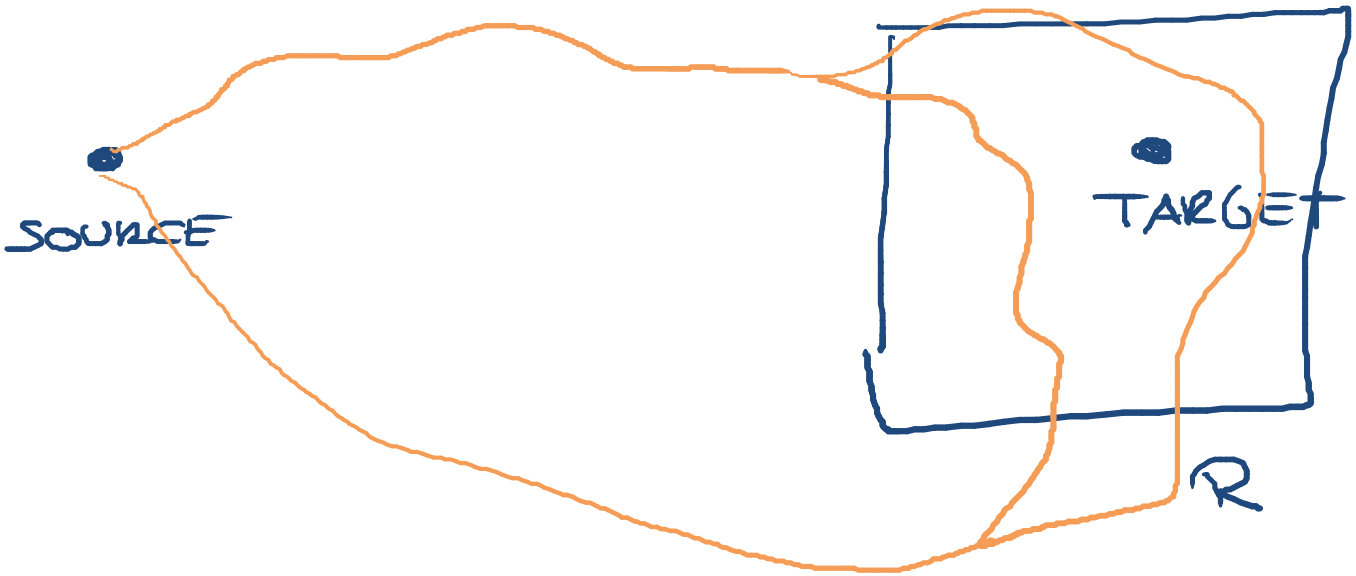
Search space comparison 2/3

- Set of settled / visited nodes for A-Star



Search space comparison 3/3

- Set of settled / visited nodes for Arc Flags



Google Fusion Tables

- Nice tool to visualize geo data on Google Maps
  - You can upload a CSV file with coordinates, e.g.

    47.95 7.75
    47.95 7.90
    48.05 7.75
    48.05 7.90

  - And then draw the points on Google Maps with one click
  - In the visualization, there is a button for a permanent link to your visualization
  - For Ex. Sheet 4: visualize the set of visited nodes for one of your queries and link to it in the result table on the Wiki
  - http://www.google.com/fusiontables

References

- First arc flag paper

  An extremely fast, exact algorithm for finding shortest paths in static networks with geographical background

  Ulrich Lauther, Münsteraner GI-Tage 2004

  https://gor.uni-paderborn.de/Members/AG06/LAUTHER.PDF

- Arc flags with various tricks + a hierarchy of regions

  Acceleration of Shortest Path and Constrained Shortest Path Computation

  E. Köhler and R. Möhring and H. Schilling, WEA 2005

  ftp://ftp.math.tu-berlin.de/pub/Preprints/combi/Report-042-2004.pdf

  http://www.springerlink.com/content/wc06qawxy5bc5bj0/