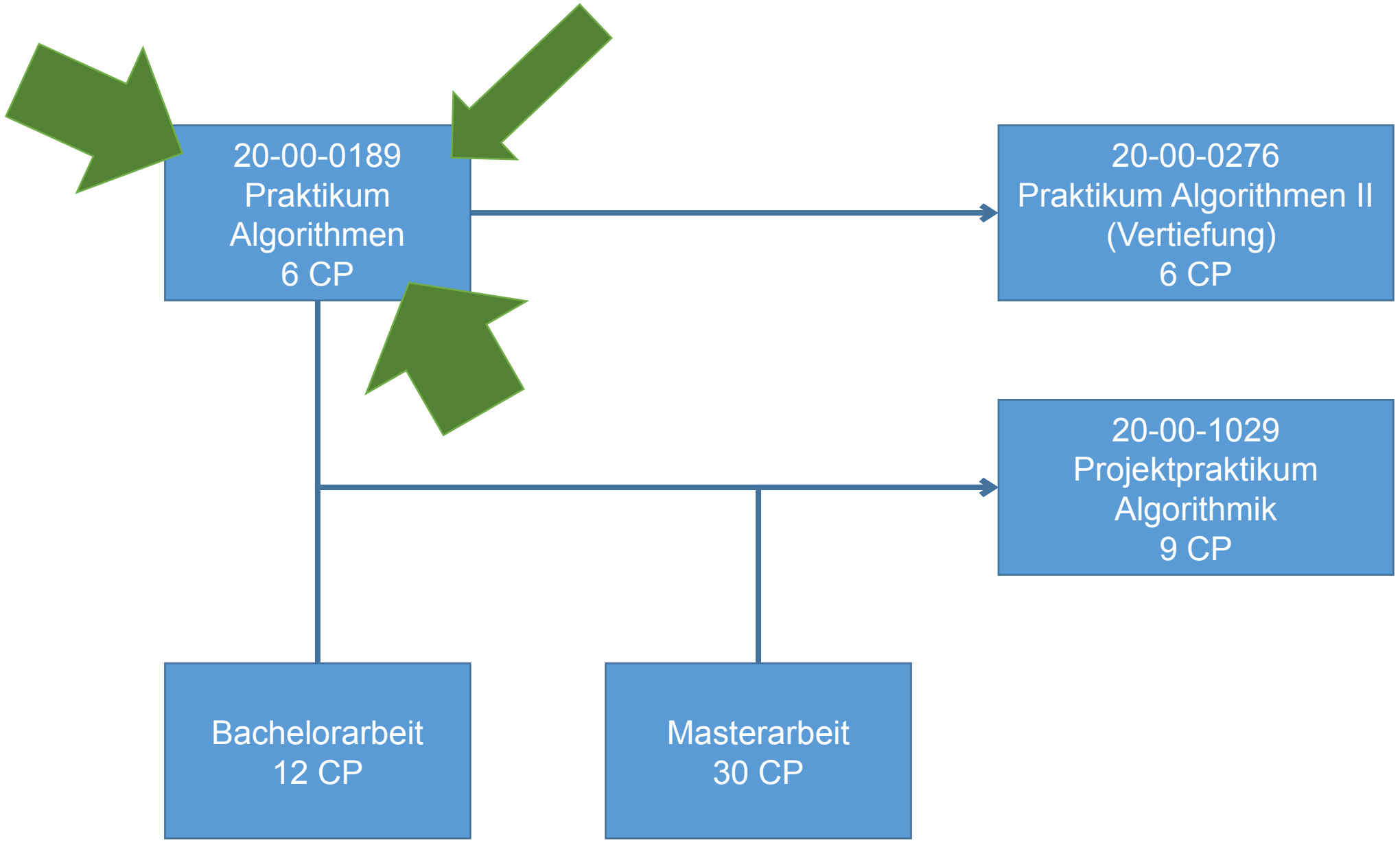
Algorithms Lab

summer semester 2026


20-00-0189
Praktikum
Algorithmen
6 CP
20-00-0276
Praktikum Algorithmen II
(Vertiefung)
6 CP
20-00-1029
Projektpraktikum
Algorithmik
9 CP
Bachelorarbeit
12 CP
Masterarbeit
30 CP

Email address

- for all communication and submissions

`lab@algo.informatik.tu-darmstadt.de`

- we will answer with our individual mail accounts
- please make sure the lab address stays in CC
- keeps everyone supervising the lab informed

Timeline

Datum	Event
15.04.2026 15:00 Uhr	Kick-Off in S2|02 E202
27.04.2026	Fortschrittsbericht / Anmeldung
11.05.2026	Fortschrittsbericht
25.05.2026	Fortschrittsbericht
08.06.2026	Fortschrittsbericht
22.06.2026	Fortschrittsbericht
06.07.2026	Fortschrittsbericht
20.07.2026	Fortschrittsbericht
31.07.2026	1. Milestone Git Patch
10.08.2026	Fortschrittsbericht
24.08.2026	Fortschrittsbericht
bis 11.09.2026	feature-vollständiger Git Patch
ab 12.09.2026	Pull Request stellen
bis 30.09.2026	Finale Abgabe

How to participate?

- register for the course in TUCaN
  - 20-00-0189 Praktikum Algorithmen
- send a first progress report until 27.04.2026

Progress reports

- encourage steady work on the lab
- 1-2 sentences each:
  - progress since last report
  - next steps planned
  - questions?
  - Do you want to attend the consultation hour?
- submission via email
- if you stop sending progress reports, we presume you do not wish to participate in the lab anymore → no need to deregister with us

Consultation hour

- Wednesdays, 15:00 - 16:00 in S2|02 E123
- please announce your participation by Monday via email if you want to attend

Submissions

- 1st milestone (deadline: 31.07.2026)
  - prototype implementation
  - does not need to be 100% correct
  - there is still time to improve it later
  - send git patch via email
- feature-complete (deadline: 11.09.2026)
  - send git patch via email
- open pull request (start date: 12.09.2026)
  - can still commit fixes to get a green CI
- final submission (deadline: 30.09.2026)
  - last commit in PR before the deadline (or specify commit explicitly)
  - send lab report via email

Lab report

- scientific work requires proper citations
- target audience: computer scientist, who has not read the lab assignment
- ~3 pages
  - no strict limit
  - depends on how many diagrams/tables there are
  - focus on the essentials
  - no table of contents
- Outline
  - Introduction
  - Problems encountered & solutions applied
  - Correctness (usually through validation)
  - Performance measurements
  - Conclusion

Lab report

- Introduction
  - describe the task in your own words
- Problems encountered & solutions applied
  - data structures that are not straightforward
  - tricky code sections: insert a code listing, explain your code
- Correctness
  - How do you ensure correctness of your implementation?
    - e.g. unit tests or comparison with reference implementation
- Performance measurements
  - see later slide
- Conclusion
  - summarize your findings
  - concise evaluation of the approach

Problems encountered & solutions applied

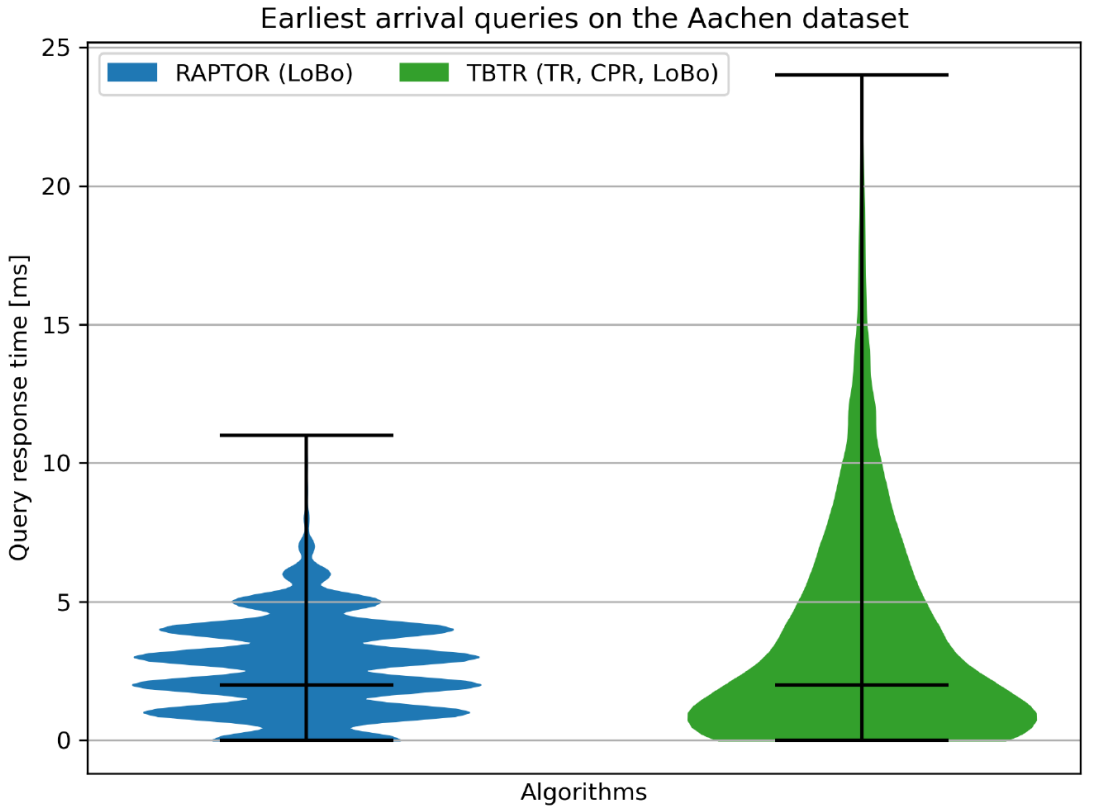
- the report is a scientific work, not a diary!
  - do not write:
    - *„The algorithm itself was relatively easy for me to grasp, since I had already encountered it before.“*
    - *“Getting up to speed with the existing codebase, in particular, posed a significant challenge at the outset.”*
    - *„However, these detours also proved to be valuable, as they contributed to a deeper understanding of the architecture.“*
- describe a specific problem
  - *„To prevent redundant allocations for <purpose>...“*
- and its solution
  - *„...we encapsulate its data structures inside a state struct (cf. Listing 1) for reuse.“*

Performance measurements

- ensure replicability through documentation
  - hardware
  - dataset
  - queries (number, uniformly at random, geo-ranked...)
  - responses
- use sensible units
  - 1722087475 ns → 1722 ms or even 1.7 s
- calculate the speedup when comparing different approaches
- what are the trade-offs?
  - memory vs. time
  - preprocessing vs. response time

Performance measurements

- Use descriptive statistics
  - e.g. quantiles, mean
- Use plots
  - e.g. scatter, violin
- Describe characteristics you observe
  - e.g. *„low average but high tail latency“*



Performance or the lack thereof

- heap allocations
  - every time a container like **`vector`** is created
  - in some function
  - that is called in a loop
  - that is nested in another loop...
- unnecessary copies
  - a function returns/gets a parameter by value, copy leads to another allocation
  - use references (**`&`**, **`const&`**)
- What are the hot data structures of the algorithm?
  - allocate them once
  - then reuse

Road networks as graphs

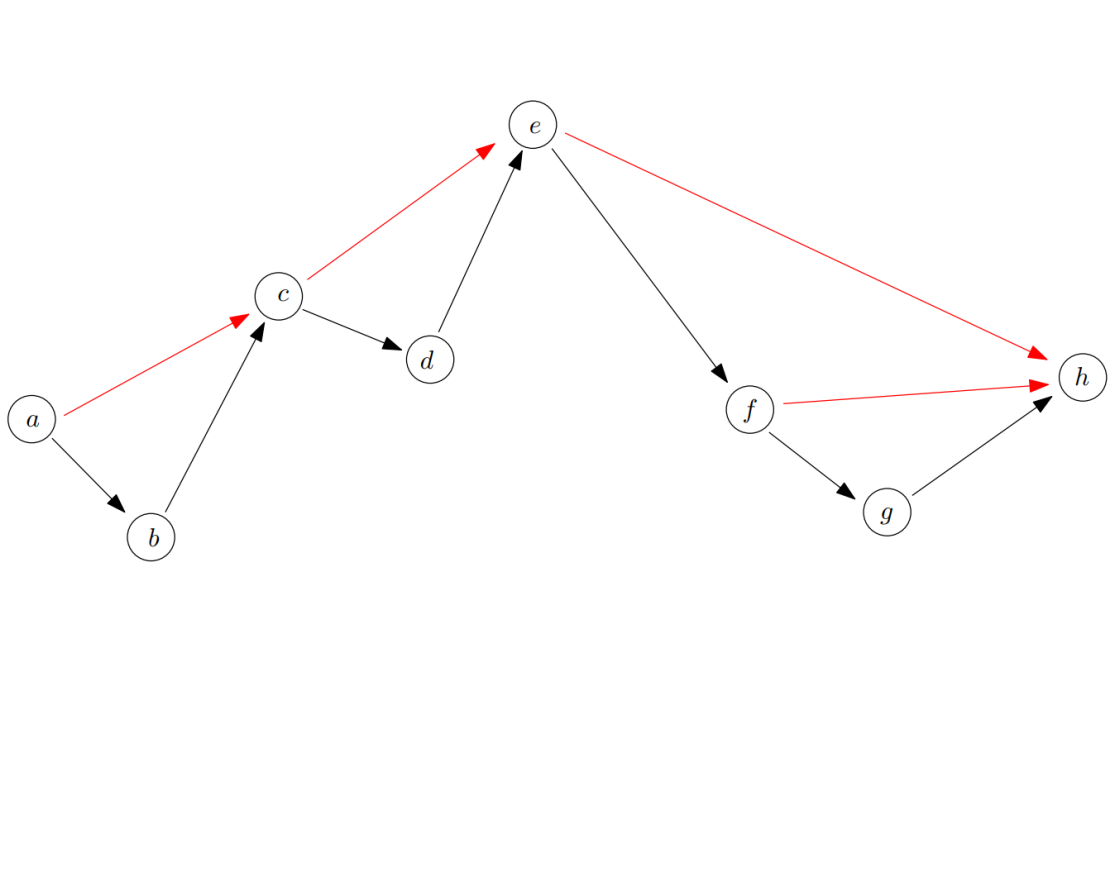
- undirected graph
  - vertices: intersections
  - edges: ways
- directed weight function (speed)
  - one-way streets
  - speed limit dependent on direction of travel
  - no entry for vehicle type

Assignment in OSR: Customizable Contraction Hierarchies

- Car Journey Hamburg-Munich
- Majority of the itinerary will be on “more important” roads
- Naive Dijkstra would visit all vertices even of the smallest country road somewhere around Kassel
- Bidirectional Dijkstra
- Every node has a certain “importance”
- In a preprocessing step, extend graph so that for a query, we only have to go from less important nodes to more important nodes (forwards and backwards)
- How can we continue to guarantee optimality?

Contraction Hierarchies

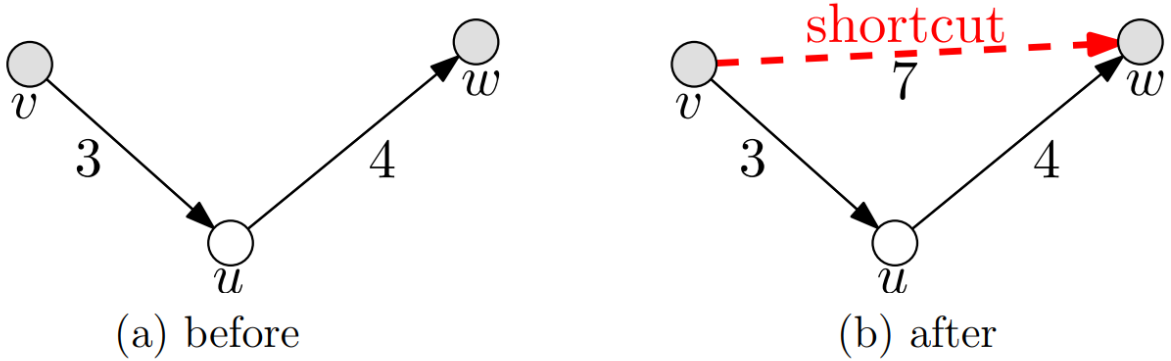
- e.g.: shortest path a-h
- y-axis: Importance/“Level”
- Preprocessing: insert red edges (“shortcuts”)
- So that forward search only has to search upwards (c, e)
- Backwards search from h analogously (only e)



Taken from: Lecture Algorithm Engineering, Prof. Stefan Funke, Uni Stuttgart

Preprocessing

- Select a vertex u for “contraction”
- Contraction order (“level”/“node order”) is a proxy for “importance” – vertices contracted first are least important
- Optimality is guaranteed regardless of order
- We want to contract/”remove” vertex u without introducing wrong shortest distances between other nodes
- To ensure that, we introduce a “shortcut”
- A shortcut represents existing shortest paths and does not make them shorter!
- Do this with all vertices



Taken from : Geisberger, Robert, et al. ”Contraction hierarchies: Faster and simpler hierarchical routing in road networks.” Experimental Algorithms: 7th International Workshop, WEA 2008 Provincetown, MA, USA, May 30-June 1, 2008 Proceedings 7. Springer Berlin Heidelberg, 2008.

Customizable Contraction Hierarchies (CCH)

- Splitting Preprocessing into two stages
  - Metric-independent preprocessing: Defining node order and introducing shortcuts just based on the topology
  - Customization: Apply metric, remove superfluous shortcuts (which are not on shortest paths)
- result: additional shortcut edges E' and mapping level : $V \rightarrow N$
- Query works on $G * = (V, E \cup E')$
- Note: Contracted nodes with their edges stay in the graph, but while querying, they will only be visited when coming from nodes with lower level

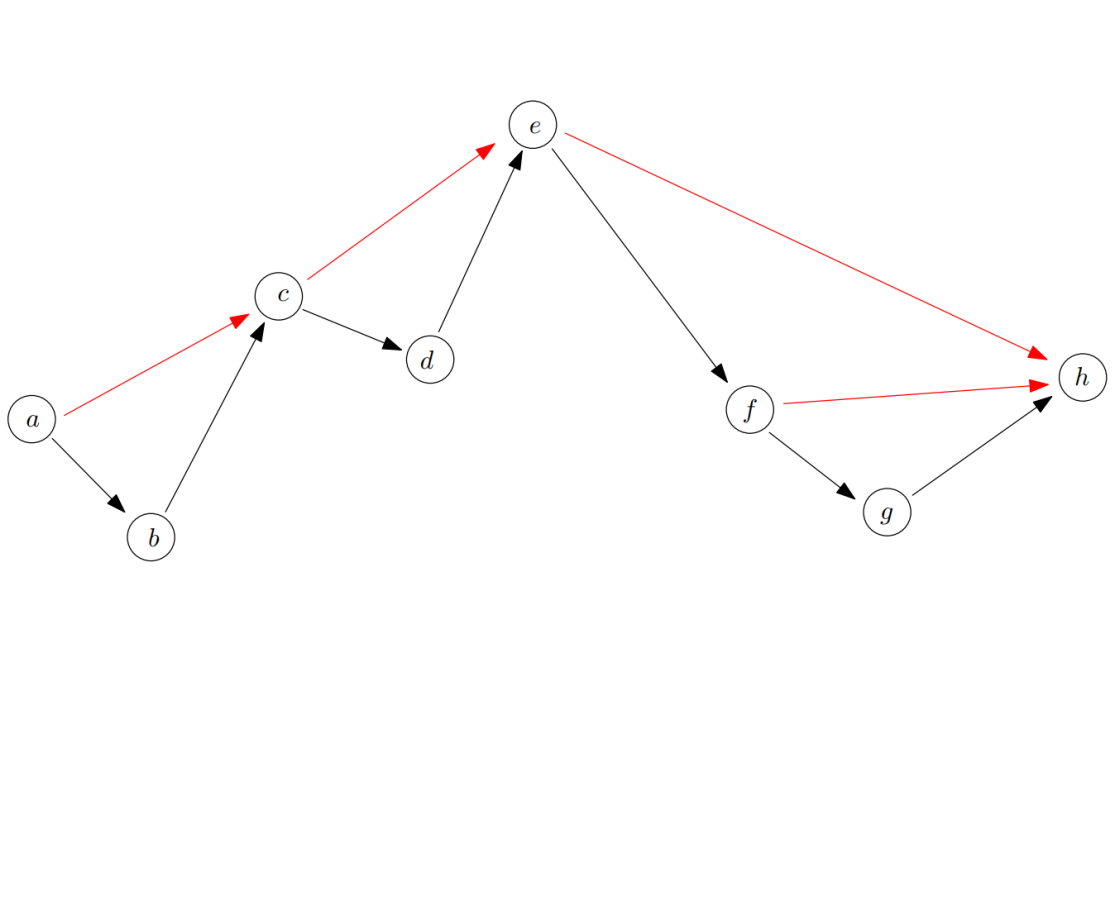
Query

- For a query s, t, run a modified bidirectional Dijkstra
- Forward-Dijkstra on $G*$ calculates distances from s
- Backward-Dijkstra calculates distances from t (follows edges in reverse direction)
- Modification: when vertex v is popped from the priority queue, only consider edges (v, w) with level(v) < level(w) – backwards search analogously (w, v)
- For Customizable Contraction Hierarchies, we might not even need a Dijkstra (instead: single order of edges with increasing level, but: in OSR we usually want to start from multiple sources, more thought is needed, or fallback to Dijkstra)

Query

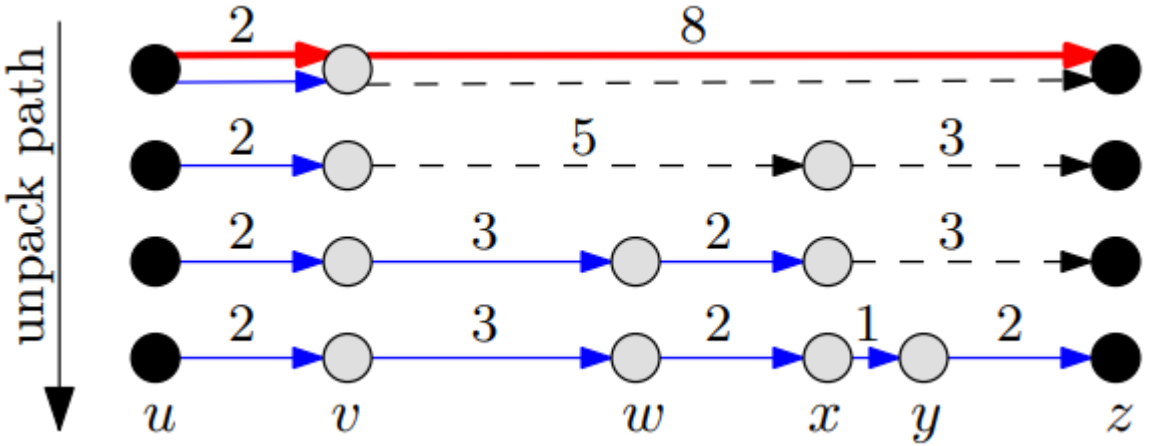
- here: forward search starts at a, visits c, e
- backwards h to e
- minimum sum of forward and backward distance is shortest distance s – t
- Microseconds instead of seconds!

Taken from: Lecture Algorithm Engineering, Prof. Stefan Funke, Uni Stuttgart



Some notes

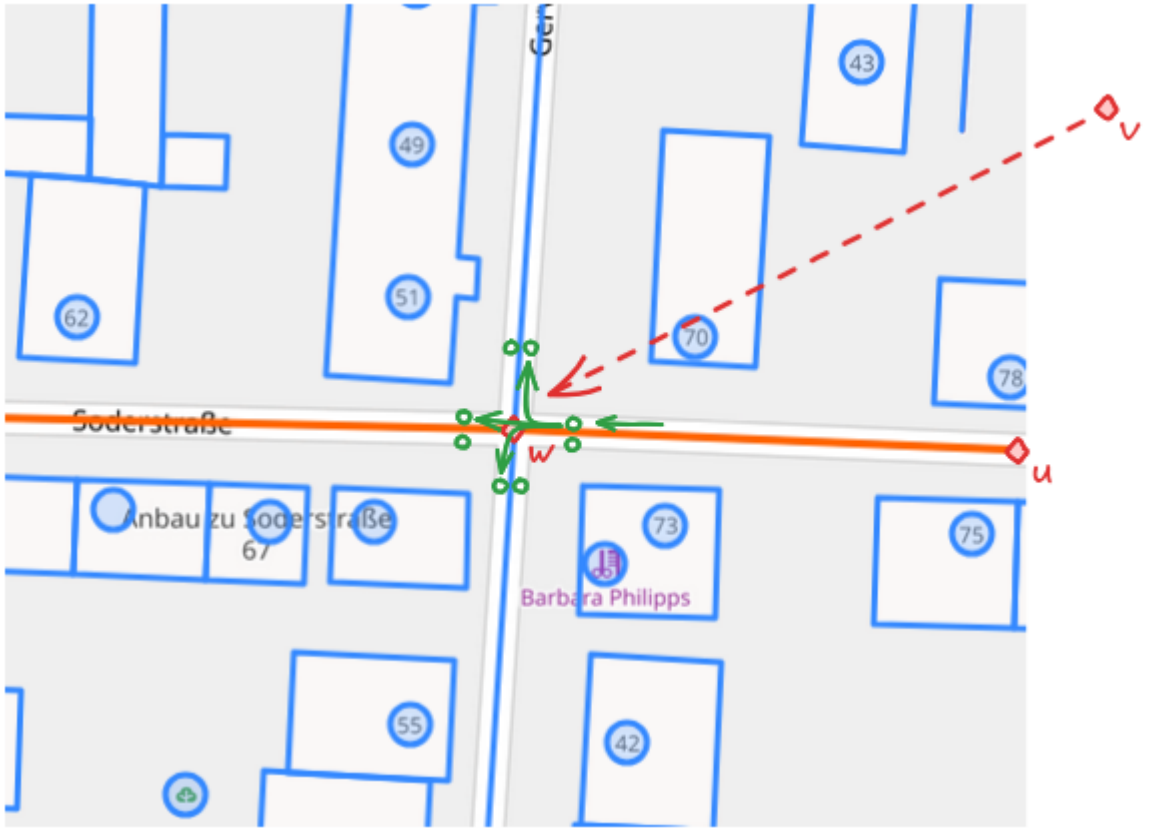
- Contraction order very relevant for performance – you can start with random, we will give you a fixed one
- Up-Graph does not only contain shortest path from s! (Why?)
- Thus: Stopping as soon as a common vertex is settled does not work as with normal bidirectional Dijkstra!
- Shortcuts need to be unpacked for reconstruction of the actual path – remember actual edges with the shortcuts



Taken from : Geisberger, Robert, et al. ”Contraction hierarchies: Faster and simpler hierarchical routing in road networks.” Experimental Algorithms: 7th International Workshop, WEA 2008 Provincetown, MA, USA, May 30-June 1, 2008 Proceedings 7. Springer Berlin Heidelberg, 2008.

OSR

- Data in OSR is not directly stored in graph form
- try not to have too much overhead for CCH construction/run
- turn restrictions! – virtual nodes
- multiple profiles (car, foot, bike etc) have different restrictions
- metric-independent preprocessing should be as profile-independent as possible
- Customization/querying should at least work with car profile
- ideally also with bus profile (car with IsBus=true)



OSR

- you may get inspiration from previous CH-related PRs and the existing bidirectional Dijkstra implementation (of course without the A* heuristic and possibly even without the Dijkstra – cf. CCH!)
- Feature Flag (should still be usable without CCH also at runtime)
- Integrate yourself into algorithms.h and route.cc
- Adapt and use https://github.com/motis-project/osr/blob/master/test/dijkstra_astarbidir_test.cc to test your implementation
- Download OSM data as input for OSR – start with small extracts: https://download.geofabrik.de/
- Or in the OSR repo and here https://github.com/motis-project/test-data/releases/tag/osr-test-data-1 for the extracts used in the tests
- Use the OSR debug UI `./osr-backend -d ./test/monaco -s ./web/`

Useful Links

- https://www.algo.informatik.tu-darmstadt.de/dl/algoprak_ss26.pdf
- Customizable Contraction Hierarchies Survey Paper: https://arxiv.org/abs/2502.10519
- Some documentation about OSR: https://github.com/motis-project/osr