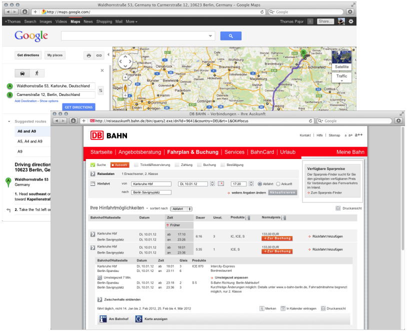
Karlsruhe Institute of Technology

Traffic Assignment in Transportation Networks

Dorothea Wagner - September 12, 2019

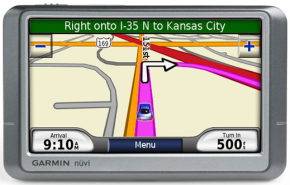
Institute of Theoretical Informatics - Research Group Algorithmics

Shortest-Path Applications



Important applications, e.g.,

- Navigation systems for cars
- Apple Maps, Google Maps, Bing Maps, OpenStreetMap, . . .
- Timetable information
- Transportation and urban planning



Core Problem

Request:

- Find the best connection in a transportation network w.r.t. some metric

Idea:

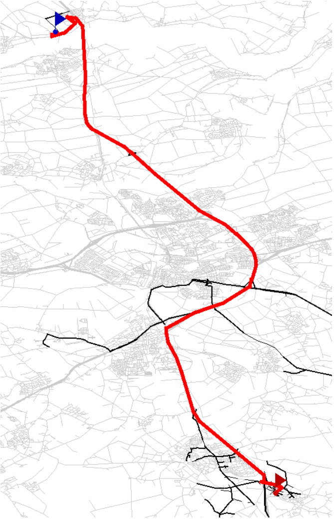
- Network as graph $G = (V, E)$
- Edge weights are according to metric
- Shortest paths in G equal best connections
- Classic problem (Dijkstra 1959)

Problems:

- Transport networks are huge
- Dijkstra too slow ($>$ 1 second)

Institute of Theoretical Informatics
Research Group Algorithmics

Speed-Up Techniques



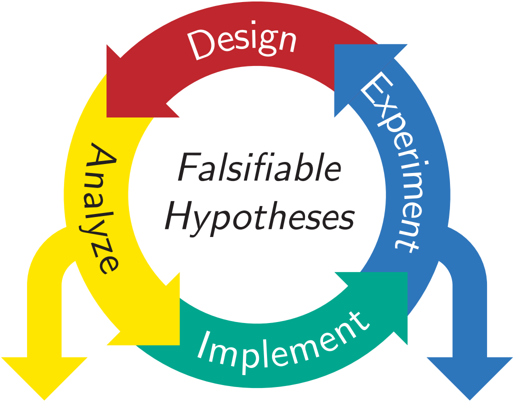
Observations:

- Dijkstra visits all nodes closer than the target
- Unnecessary computations
- Many requests in a hardly changing network

Idea:

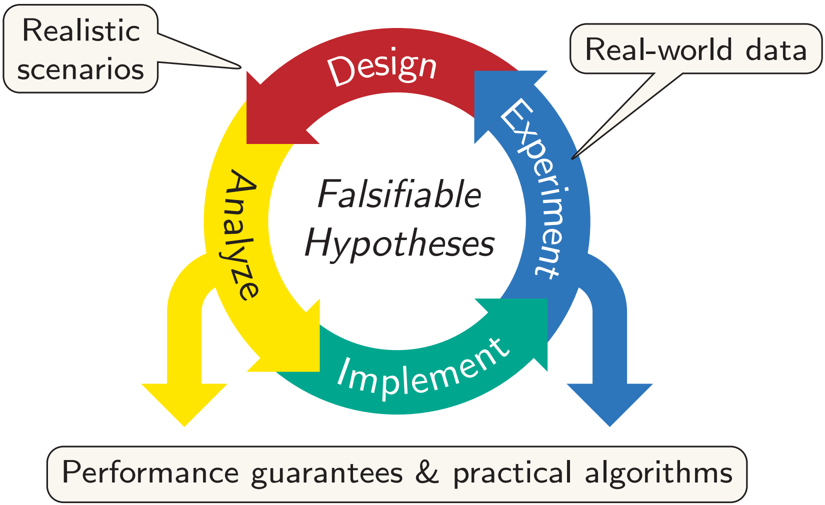
- Two-phase algorithm:
  - Offline: compute additional data during preprocessing
  - Online: speed-up query with this data
- 3 criteria: preprocessing time and space, speed-up over Dijkstra

Showpiece of Algorithm Engineering



Showpiece of Algorithm Engineering

Karlsruhe Institute of Technology



State-of-the-Art

Karlsruhe Institute of Technology

Many techniques tuned for continent-sized road networks:

- Arc-Flags (2004, 2006, 2009, 2013)
- Multi-Level Dijkstra (2000, 2008, 2009, 2011, 2016)
- ALT: A*, Landmarks, Triangle Inequality (1968, 2005, 2012)
- Reach (2004, 2007)
- Contraction Hierarchies (CH, CCH) (2008, 2013, 2014, 2016)
- Transit Node Routing (TNR) (2007, 2013)
- Hub Labeling (HL) (2003, 2011, 2013, 2014)

Timetable information:

- Transfer Patterns (2010, 2016)
- RAPTOR (2013)
- Connection Scan (2013, 2014, 2017)
- Trip-Based Public Transit Routing (2015, 2016)

Survey on “Route Planning in Transportation Networks” (Bast et al. 2016)

Next Steps

Karlsruhe Institute of Technology

State of the art:

- Portfolio of fast shortest-path algorithms
- Different trade-offs between:
  - Preprocessing time and space
  - Query time
  - Implementation complexity
  - Versatility

⇒ Leverage these in transportation applications

Case study in this talk: traffic assignment

- Major problem in transport and urban planning
- Goal: analyze utilization of roads, trains, buses
- Requires many shortest-path computations

Joint Work with

Valentin Buchhold

Tobias Zündorf

Moritz Baum   Peter Sanders   Jonas Sauer   Ben Strasser

Institute of Theoretical Informatics
Research Group Algorithmics

In Road Networks

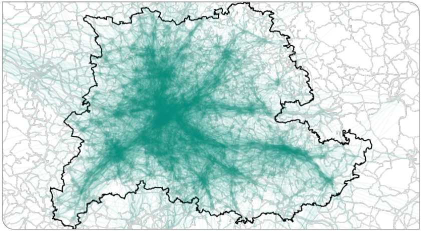
Institute of Theoretical Informatics
Research Group Algorithmics

Traffic Assignment in Road Networks

Karlsruhe Institute of Technology

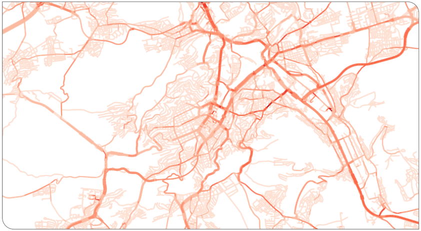
Input:

- Urban road network
- Set of origin–destination pairs



Output:

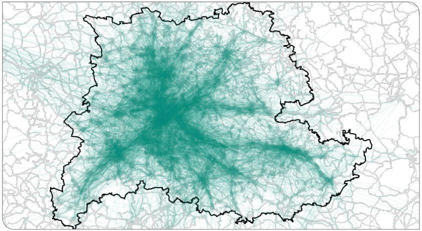
- Equilibrium flow pattern
- i.e. flow on each segment



Traffic Assignment in Road Networks

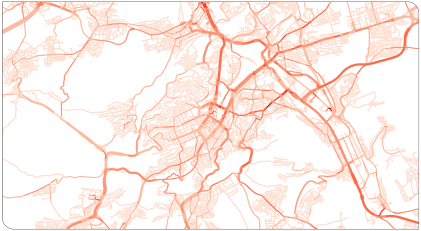
Input:

- Urban road network
- Set of origin–destination pairs



Output:

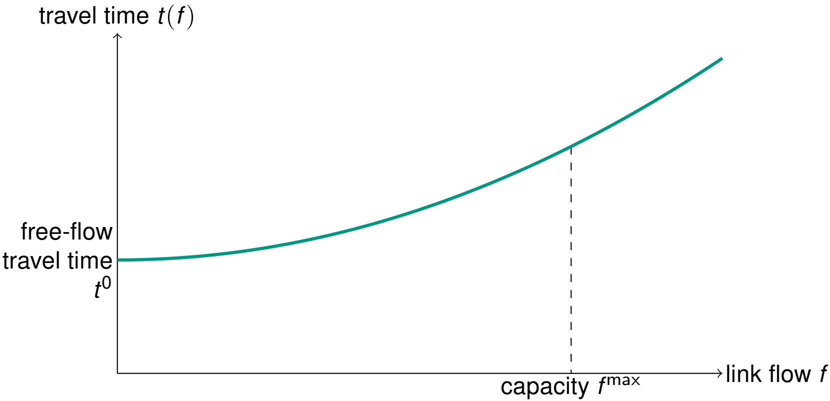
- Equilibrium flow pattern
- i.e. flow on each segment



Assumption:

- Motorists choose path with minimum travel time. . .
- . . . but travel time changes with flow (congestion)

Relation between Flow and Travel Time



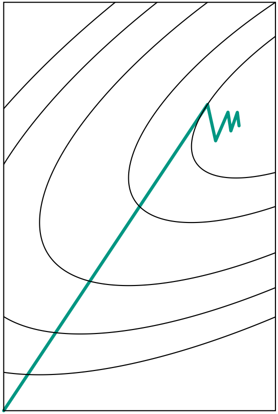
Link cost function: $t(f) = t^0 \left(1 + \alpha \left(\frac{f}{f^{\max}}\right)^{\beta}\right)$ (Bureau of Public Roads 1964)

Solution Algorithms

Karlsruhe Institute of Technology

Link-based methods:

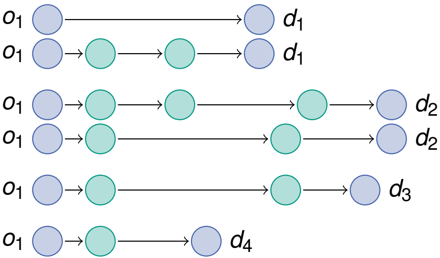
- Represent solution by link flows f_e (flow on link e)
- Feasible-direction methods
  - Start from initial solution
  - Generate feasible direction of descent
  - Shift current solution along descent direction
- Examples: Frank-Wolfe (1956), conjugate FW (2013), biconjugate FW (2013)



Solution Algorithms

Path-based methods:

- Represent solution by path flows F_k (flow on path k)
- Maintain set K_p^+ of promising paths between each O-D pair p
- In each iteration, process O-D pairs p one by one
  1. Update K_p^+ (remove unpromising paths, insert new promising paths)
  2. Equilibrate K_p^+ (shift flow between paths in K_p^+)
- Examples: PE (1968), GP (1994), PG (2009), ISP (2011)

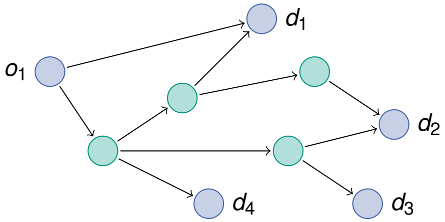


Institute of Theoretical Informatics
Research Group Algorithmics

Solution Algorithms

Bush-based methods:

- Represent solution by origin flows f_{eo} (flow on link e that originates at origin o)
- Maintain bush B_o for each origin o
- B_o is DAG that comprises promising paths from o to all destinations
- In each iteration, process origins o one by one
  1. Update B_o (remove zero-flow links, insert new links giving rise to cheaper paths)
  2. Equilibrate B_o (shift flow on B_o)
- Examples: Algorithm B (2006), LUCE (2014), TAPAS (2010)



Frank-Wolfe Algorithm

- Represents solution (before iteration i) by link flows $f^i = (f^i_1, \ldots, f^i_{|E|})$
- Main subroutine is all-or-nothing (AON) assignment
  - Process O-D pairs one by one
  - Assign one flow unit to each link on shortest path

FrankWolfe

Generate initial solution by performing free-flow AON assignment
while *convergence criterion is not satisfied* **do**
  Update link costs based on current link flows
  Perform AON assignment based on current link costs, yielding y^i
  Let descent direction d^i be $y^i - f^i$
  Determine how far current solution must be moved along descent direction
  Move current solution along descent direction, i.e., set $f^{i+1} = f^i + \lambda^i d^i$

Frank-Wolfe Algorithm

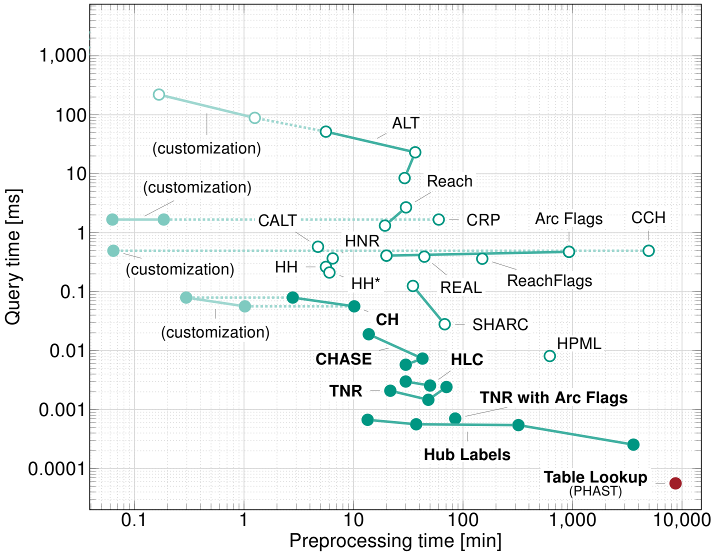
Karlsruhe Institute of Technology

- Represents solution (before iteration i) by link flows $f^i = (f^i_1, \ldots, f^i_{|E|})$
- Main subroutine is all-or-nothing (AON) assignment
  - Process O-D pairs one by one
  - Assign one flow unit to each link on shortest path

```
FrankWolfe
Generate initial solution by performing free-flow AON assignment
while convergence criterion is not satisfied do
    Update link costs based on current link flows
    Perform AON assignment based on current link costs, yielding y^i
    Let descent direction d^i be y^i − f^i
    Determine how far current solution must be moved along descent direction
    Move current solution along descent direction, i.e., set f^{i+1} = f^i + λ^i d^i
```

$\Rightarrow$ **Benefits particularly from recent advances in route planning**

State of the Art in Routing (Bast et al. 2016)

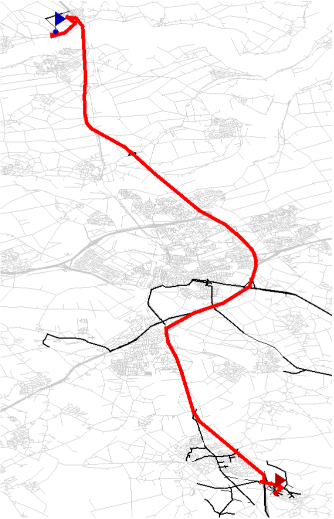


Institute of Theoretical Informatics
Research Group Algorithmics

Speedup Techniques

Two-phase:

- Preprocessing (slow): compute additional data
- Query (fast): answer *s*–*t* queries using data from preprocessing



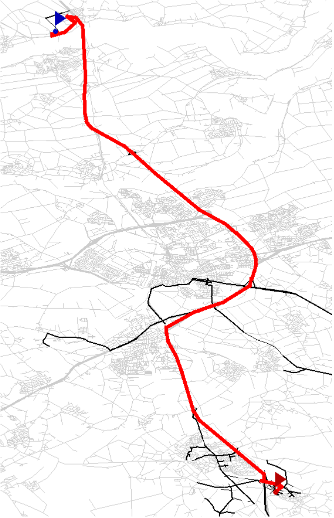
Speedup Techniques

Two-phase:

- Preprocessing (slow): compute additional data
- Query (fast): answer *s*–*t* queries using data from preprocessing

Three-phase:

- Preprocessing (slow): compute additional weight-independent data
- Customization (reasonably fast): introduce weights
- Query (fast): answer *s*–*t* queries using data from preprocessing and customization



Shortest-Path Algorithm for Frank-Wolfe?

Requirements:

- Fast point-to-point shortest-path computations
- Easy retrieval of actual shortest paths (not only distances)
- Edge weights change in each iteration $\rightarrow$ dynamic scenario

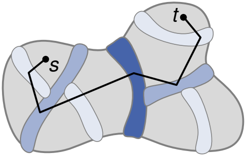
Shortest-Path Algorithm for Frank-Wolfe?

Requirements:

- Fast point-to-point shortest-path computations
- Easy retrieval of actual shortest paths (not only distances)
- Edge weights change in each iteration $\rightarrow$ dynamic scenario

Best fit: customizable contraction hierarchies

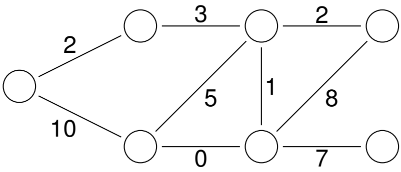
- Uses metric-independent nested dissection order
- Customization: compute shortcut weights
- Elimination tree query (requires no queue)



Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Preprocessing:

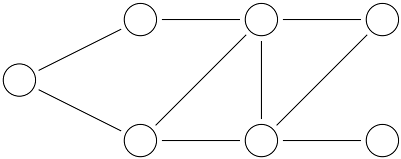


Institute of Theoretical Informatics
Research Group Algorithmics

Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Preprocessing:



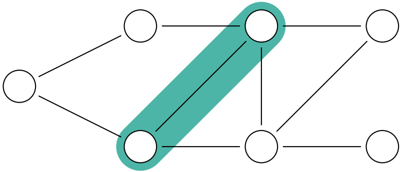
Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Karlsruhe Institute of Technology

Preprocessing:

- Partitioning: compute nested dissection order
  - Recursively split graph into two parts
  - Place separator vertices at end of order



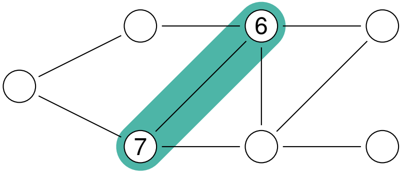
Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Karlsruhe Institute of Technology

Preprocessing:

- Partitioning: compute nested dissection order
  - Recursively split graph into two parts
  - Place separator vertices at end of order

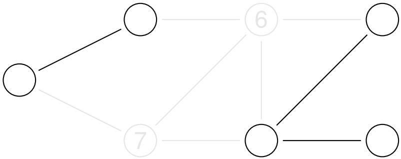


Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Preprocessing:

- Partitioning: compute nested dissection order
  - Recursively split graph into two parts
  - Place separator vertices at end of order

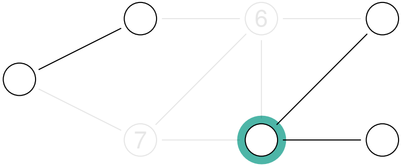


Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Preprocessing:

- Partitioning: compute nested dissection order
  - Recursively split graph into two parts
  - Place separator vertices at end of order

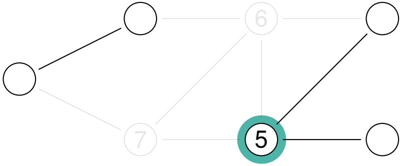


Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Preprocessing:

- Partitioning: compute nested dissection order
  - Recursively split graph into two parts
  - Place separator vertices at end of order

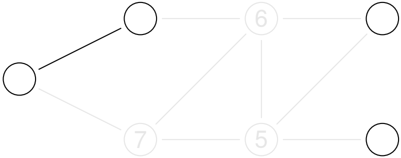


Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Preprocessing:

- Partitioning: compute nested dissection order
    - Recursively split graph into two parts
    - Place separator vertices at end of order

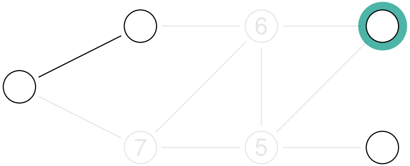


Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Preprocessing:

- Partitioning: compute nested dissection order
  - Recursively split graph into two parts
  - Place separator vertices at end of order

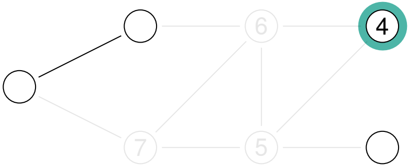


Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Preprocessing:

- Partitioning: compute nested dissection order
  - Recursively split graph into two parts
  - Place separator vertices at end of order

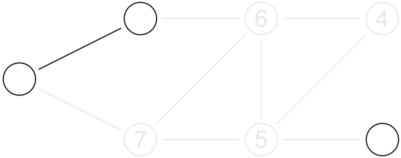


Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Preprocessing:

- Partitioning: compute nested dissection order
  - Recursively split graph into two parts
  - Place separator vertices at end of order

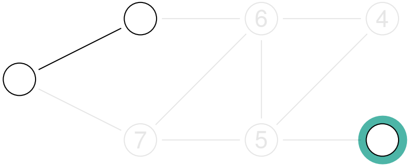


Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Preprocessing:

- Partitioning: compute nested dissection order
  - Recursively split graph into two parts
  - Place separator vertices at end of order

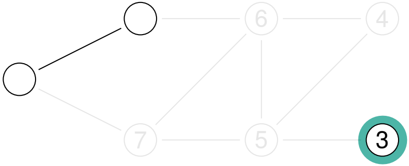


Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Preprocessing:

- Partitioning: compute nested dissection order
  - Recursively split graph into two parts
  - Place separator vertices at end of order

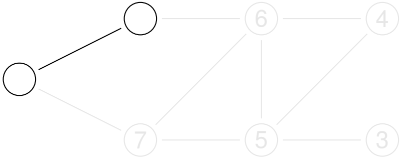


Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Preprocessing:

- Partitioning: compute nested dissection order
  - Recursively split graph into two parts
  - Place separator vertices at end of order

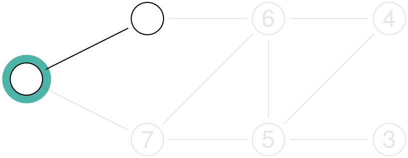


Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Preprocessing:

- Partitioning: compute nested dissection order
  - Recursively split graph into two parts
  - Place separator vertices at end of order

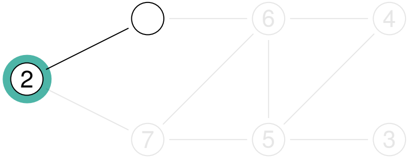


Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Preprocessing:

- Partitioning: compute nested dissection order
  - Recursively split graph into two parts
  - Place separator vertices at end of order

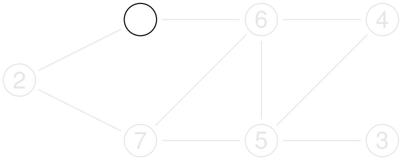


Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Preprocessing:

- Partitioning: compute nested dissection order
  - Recursively split graph into two parts
  - Place separator vertices at end of order

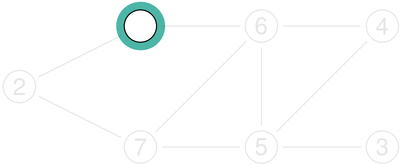


Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Preprocessing:

- Partitioning: compute nested dissection order
  - Recursively split graph into two parts
  - Place separator vertices at end of order



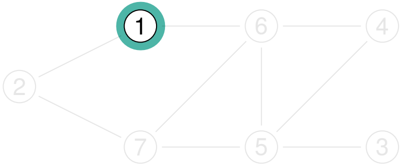
Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Karlsruhe Institute of Technology

Preprocessing:

- Partitioning: compute nested dissection order
  - Recursively split graph into two parts
  - Place separator vertices at end of order

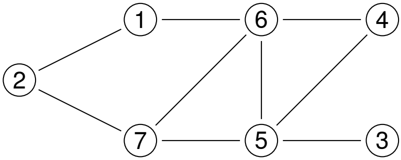


Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Preprocessing:

- Partitioning: compute nested dissection order
  - Recursively split graph into two parts
  - Place separator vertices at end of order



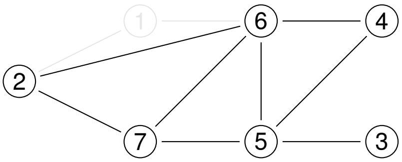
Institute of Theoretical Informatics
Research Group Algorithmics

Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Preprocessing:

- Partitioning: compute nested dissection order
  - Recursively split graph into two parts
  - Place separator vertices at end of order
- Contraction: shortcut vertices in this order
  - Temporarily remove vertex from graph
  - Add shortcut edges between its neighbors

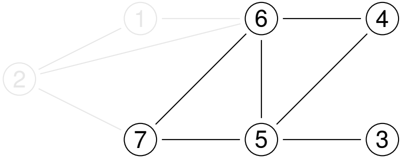


Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Preprocessing:

- Partitioning: compute nested dissection order
  - Recursively split graph into two parts
  - Place separator vertices at end of order
- Contraction: shortcut vertices in this order
  - Temporarily remove vertex from graph
  - Add shortcut edges between its neighbors

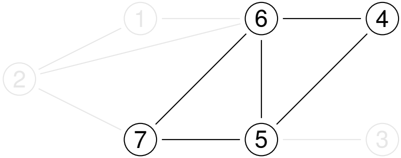


Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Preprocessing:

- Partitioning: compute nested dissection order
  - Recursively split graph into two parts
  - Place separator vertices at end of order
- Contraction: shortcut vertices in this order
  - Temporarily remove vertex from graph
  - Add shortcut edges between its neighbors

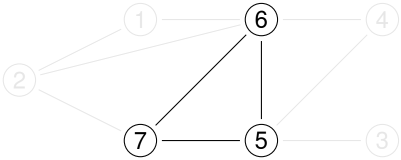


Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Preprocessing:

- Partitioning: compute nested dissection order
  - Recursively split graph into two parts
  - Place separator vertices at end of order
- Contraction: shortcut vertices in this order
  - Temporarily remove vertex from graph
  - Add shortcut edges between its neighbors

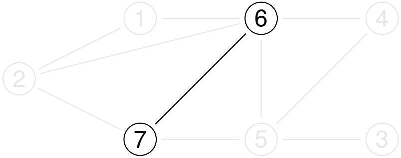


Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Preprocessing:

- Partitioning: compute nested dissection order
  - Recursively split graph into two parts
  - Place separator vertices at end of order
- Contraction: shortcut vertices in this order
  - Temporarily remove vertex from graph
  - Add shortcut edges between its neighbors

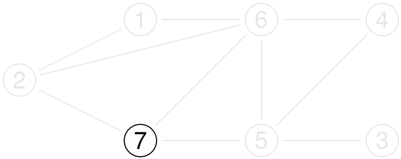


Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Preprocessing:

- Partitioning: compute nested dissection order
  - Recursively split graph into two parts
  - Place separator vertices at end of order
- Contraction: shortcut vertices in this order
  - Temporarily remove vertex from graph
  - Add shortcut edges between its neighbors



Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Preprocessing:

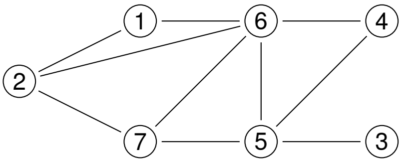
- Partitioning: compute nested dissection order
  - Recursively split graph into two parts
  - Place separator vertices at end of order
- Contraction: shortcut vertices in this order
  - Temporarily remove vertex from graph
  - Add shortcut edges between its neighbors

Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Preprocessing:

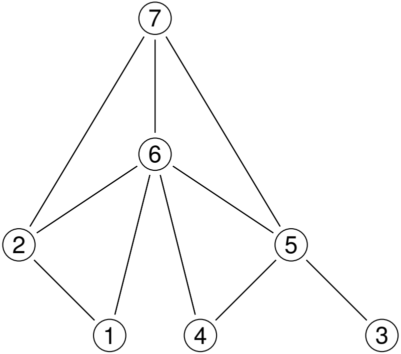
- Partitioning: compute nested dissection order
  - Recursively split graph into two parts
  - Place separator vertices at end of order
- Contraction: shortcut vertices in this order
  - Temporarily remove vertex from graph
  - Add shortcut edges between its neighbors



Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Customization:

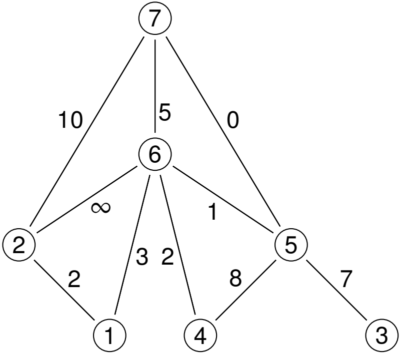


Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Customization:

- Assign orig edges their input weight

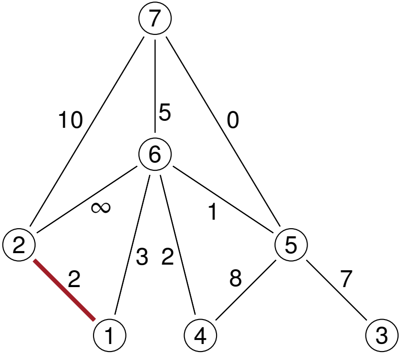


Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Customization:

- Assign orig edges their input weight
- Process edges in bottom-up fashion
  - Enumerate all lower triangles
  - Check if it improves edge weight

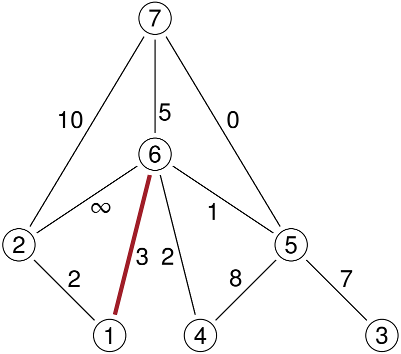


Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Customization:

- Assign orig edges their input weight
- Process edges in bottom-up fashion
  - Enumerate all lower triangles
  - Check if it improves edge weight

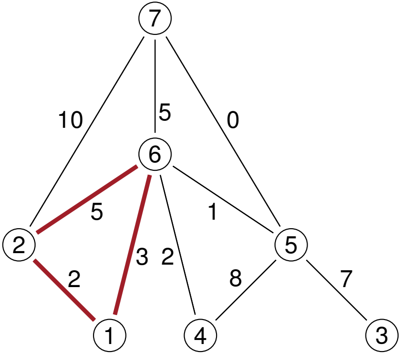


Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Customization:

- Assign orig edges their input weight
- Process edges in bottom-up fashion
  - Enumerate all lower triangles
  - Check if it improves edge weight



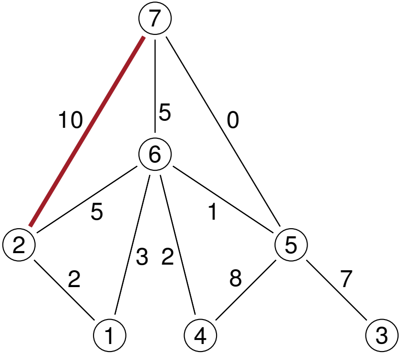
Institute of Theoretical Informatics
Research Group Algorithmics

Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Customization:

- Assign orig edges their input weight
- Process edges in bottom-up fashion
    - Enumerate all lower triangles
    - Check if it improves edge weight



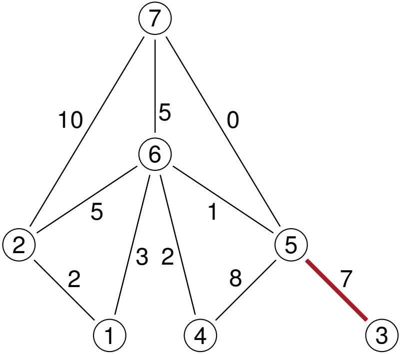
Institute of Theoretical Informatics
Research Group Algorithmics

Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Customization:

- Assign orig edges their input weight
- Process edges in bottom-up fashion
  - Enumerate all lower triangles
  - Check if it improves edge weight



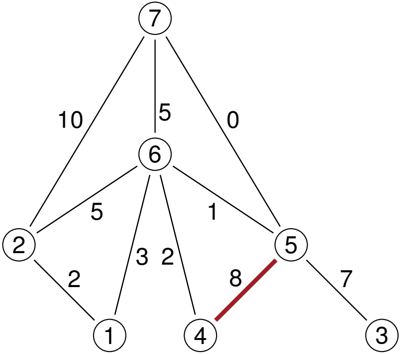
Institute of Theoretical Informatics
Research Group Algorithmics

Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Customization:

- Assign orig edges their input weight
- Process edges in bottom-up fashion
  - Enumerate all lower triangles
  - Check if it improves edge weight

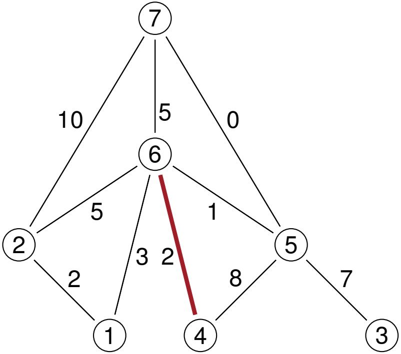


Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Customization:

- Assign orig edges their input weight
- Process edges in bottom-up fashion
  - Enumerate all lower triangles
  - Check if it improves edge weight



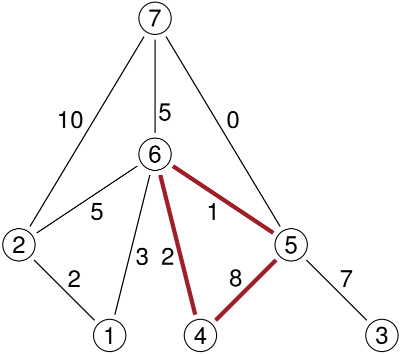
Institute of Theoretical Informatics
Research Group Algorithmics

Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Customization:

- Assign orig edges their input weight
- Process edges in bottom-up fashion
  - Enumerate all lower triangles
  - Check if it improves edge weight

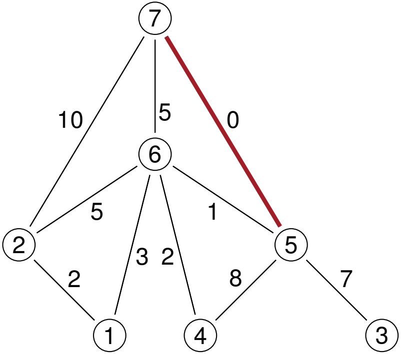


Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Customization:

- Assign orig edges their input weight
- Process edges in bottom-up fashion
  - Enumerate all lower triangles
  - Check if it improves edge weight



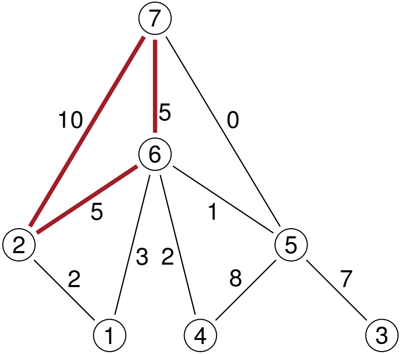
Institute of Theoretical Informatics
Research Group Algorithmics

Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Customization:

- Assign orig edges their input weight
- Process edges in bottom-up fashion
  - Enumerate all lower triangles
  - Check if it improves edge weight

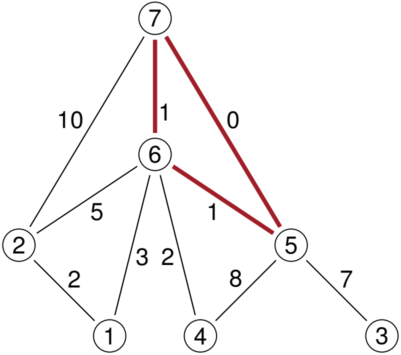


Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Customization:

- Assign orig edges their input weight
- Process edges in bottom-up fashion
  - Enumerate all lower triangles
  - Check if it improves edge weight

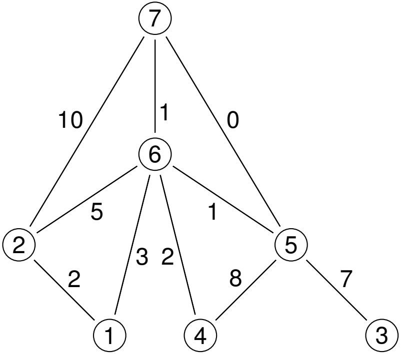


Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Customization:

- Assign orig edges their input weight
- Process edges in bottom-up fashion
  - Enumerate all lower triangles
  - Check if it improves edge weight



Institute of Theoretical Informatics
Research Group Algorithmics

Customizable Contraction Hierarchies

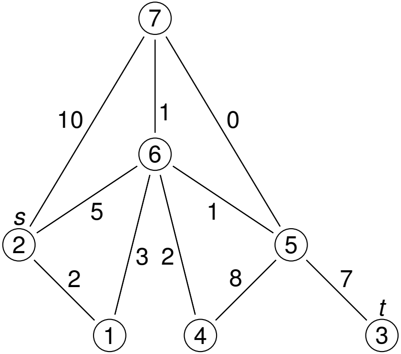
(Dibbelt et al. 2016)

Customization:

- Assign orig edges their input weight
- Process edges in bottom-up fashion
  - Enumerate all lower triangles
  - Check if it improves edge weight

Query algorithm:

- Bidirectional Dijkstra
- Only relax edges to higher ranks



Customizable Contraction Hierarchies

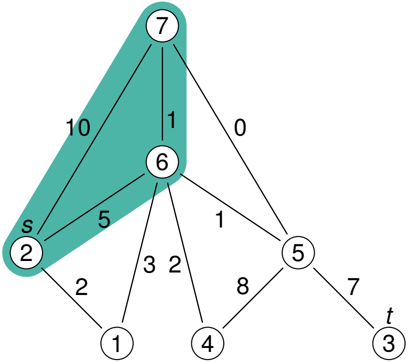
(Dibbelt et al. 2016)

Customization:

- Assign orig edges their input weight
- Process edges in bottom-up fashion
  - Enumerate all lower triangles
  - Check if it improves edge weight

Query algorithm:

- Bidirectional Dijkstra
- Only relax edges to higher ranks



Institute of Theoretical Informatics
Research Group Algorithmics

Customizable Contraction Hierarchies

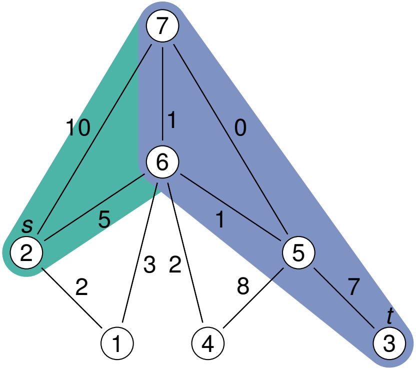
(Dibbelt et al. 2016)

Customization:

- Assign orig edges their input weight
- Process edges in bottom-up fashion
  - Enumerate all lower triangles
  - Check if it improves edge weight

Query algorithm:

- Bidirectional Dijkstra
- Only relax edges to higher ranks



Customizable Contraction Hierarchies

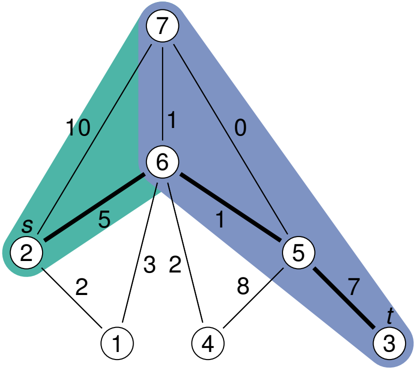
(Dibbelt et al. 2016)

Customization:

- Assign orig edges their input weight
- Process edges in bottom-up fashion
  - Enumerate all lower triangles
  - Check if it improves edge weight

Query algorithm:

- Bidirectional Dijkstra
- Only relax edges to higher ranks



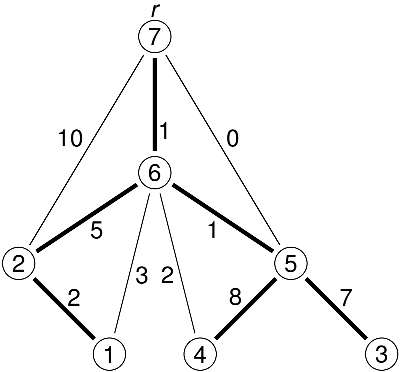
Institute of Theoretical Informatics
Research Group Algorithmics

Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Alternative query algorithm:

- Based on elimination tree

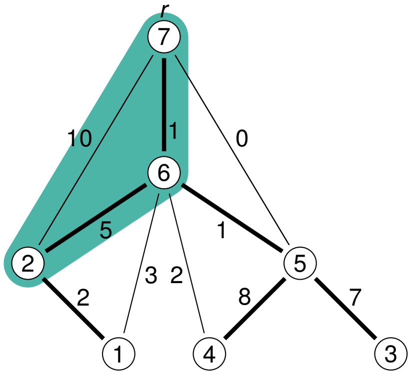


Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Alternative query algorithm:

- Based on elimination tree
- Elimination tree efficiently encodes CCH search space of each vertex



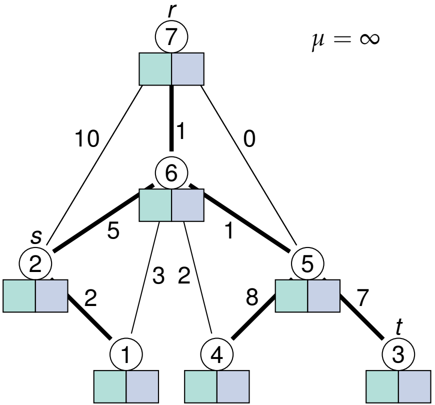
Customizable Contraction Hierarchies

(Dibbelt et al. 2016)

Alternative query algorithm:

- Based on elimination tree
- Elimination tree efficiently encodes CCH search space of each vertex

Elimination tree search:



Institute of Theoretical Informatics
Research Group Algorithmics

Customizable Contraction Hierarchies

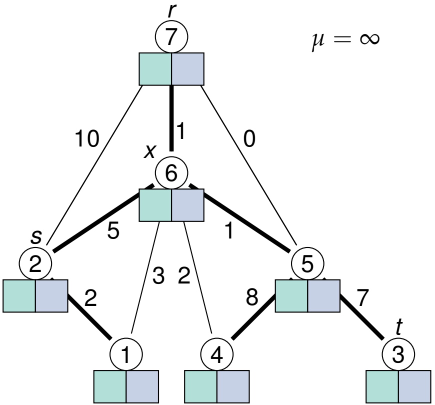
(Dibbelt et al. 2016)

Alternative query algorithm:

- Based on elimination tree
- Elimination tree efficiently encodes CCH search space of each vertex

Elimination tree search:

1. Compute LCA x of s and t



Customizable Contraction Hierarchies

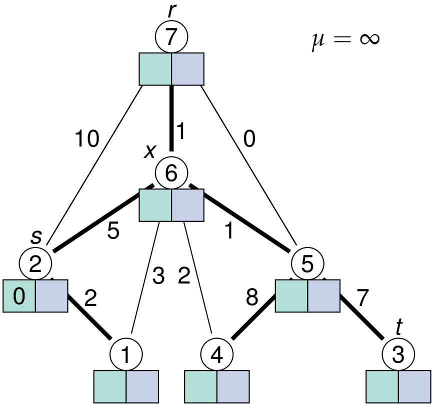
(Dibbelt et al. 2016)

Alternative query algorithm:

- Based on elimination tree
- Elimination tree efficiently encodes CCH search space of each vertex

Elimination tree search:

1. Compute LCA x of s and t
2. Scan all vertices on s–x path



Customizable Contraction Hierarchies

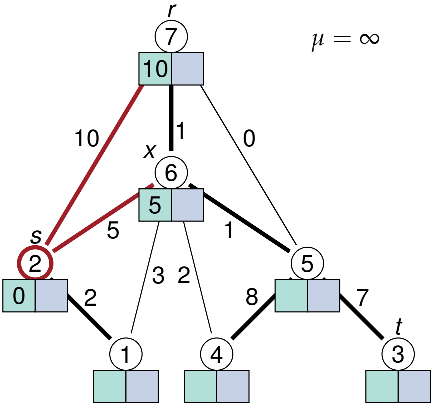
(Dibbelt et al. 2016)

Alternative query algorithm:

- Based on elimination tree
- Elimination tree efficiently encodes CCH search space of each vertex

Elimination tree search:

1. Compute LCA x of s and t
2. Scan all vertices on s–x path



Institute of Theoretical Informatics
Research Group Algorithmics

Customizable Contraction Hierarchies

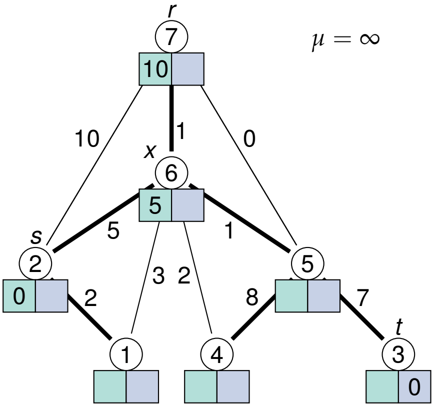
(Dibbelt et al. 2016)

Alternative query algorithm:

- Based on elimination tree
- Elimination tree efficiently encodes CCH search space of each vertex

Elimination tree search:

1. Compute LCA x of s and t
2. Scan all vertices on s–x path
3. Scan all vertices on t–x path



Customizable Contraction Hierarchies

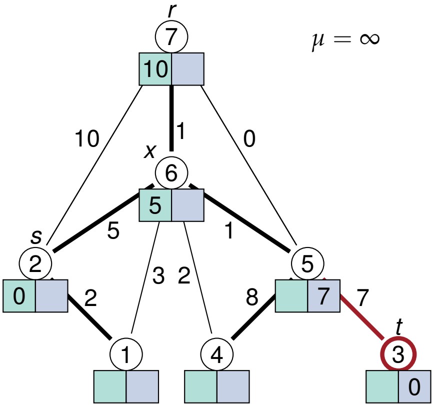
(Dibbelt et al. 2016)

Alternative query algorithm:

- Based on elimination tree
- Elimination tree efficiently encodes CCH search space of each vertex

Elimination tree search:

1. Compute LCA *x* of *s* and *t*
2. Scan all vertices on *s*–*x* path
3. Scan all vertices on *t*–*x* path



Customizable Contraction Hierarchies

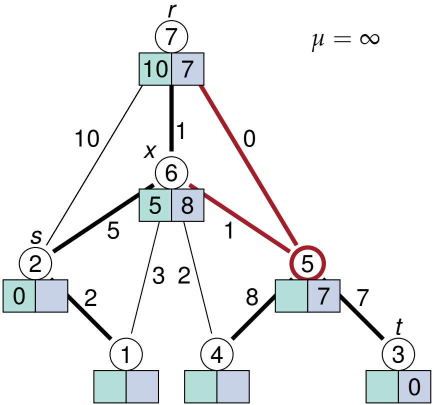
(Dibbelt et al. 2016)

Alternative query algorithm:

- Based on elimination tree
- Elimination tree efficiently encodes CCH search space of each vertex

Elimination tree search:

1. Compute LCA x of s and t
2. Scan all vertices on s–x path
3. Scan all vertices on t–x path



Customizable Contraction Hierarchies

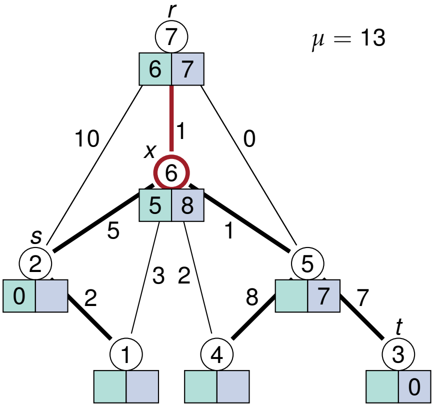
(Dibbelt et al. 2016)

Alternative query algorithm:

- Based on elimination tree
- Elimination tree efficiently encodes CCH search space of each vertex

Elimination tree search:

1. Compute LCA x of s and t
2. Scan all vertices on s–x path
3. Scan all vertices on t–x path
4. Scan all vertices on x–r path



Institute of Theoretical Informatics
Research Group Algorithmics

Customizable Contraction Hierarchies

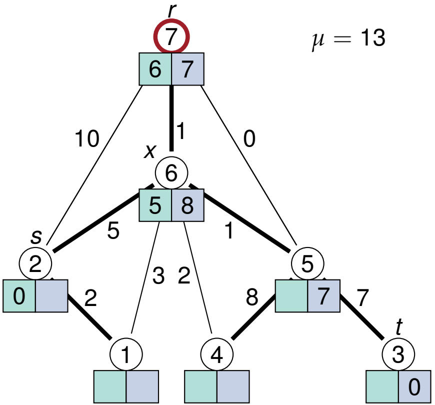
(Dibbelt et al. 2016)

Alternative query algorithm:

- Based on elimination tree
- Elimination tree efficiently encodes CCH search space of each vertex

Elimination tree search:

1. Compute LCA x of s and t
2. Scan all vertices on s–x path
3. Scan all vertices on t–x path
4. Scan all vertices on x–r path



Customizable Contraction Hierarchies

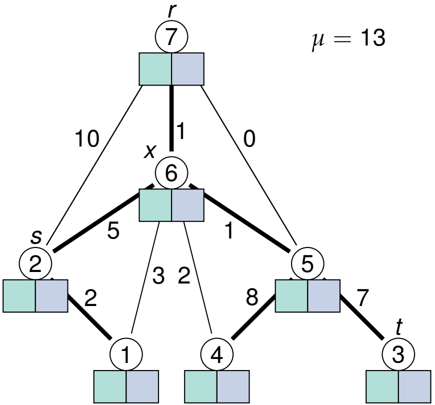
(Dibbelt et al. 2016)

Alternative query algorithm:

- Based on elimination tree
- Elimination tree efficiently encodes CCH search space of each vertex

Elimination tree search:

1. Compute LCA x of s and t
2. Scan all vertices on s–x path
3. Scan all vertices on t–x path
4. Scan all vertices on x–r path
5. Reset labels on s–r and t–r path

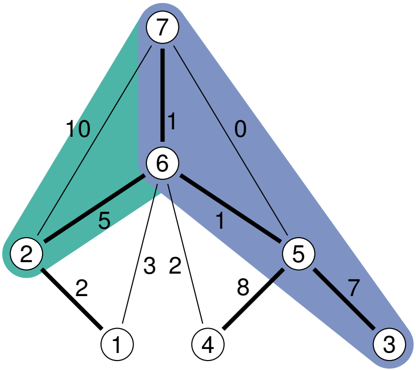


Faster Batched One-to-One Shortest Paths

(Buchhold et al. 2018)

Observation:

- Processing similar OD-pairs in succession improves locality
- Size of sym. diff between search spaces of *u* and *v* is equal to *u*–*v* distance in elimination tree



Institute of Theoretical Informatics
Research Group Algorithmics

Faster Batched One-to-One Shortest Paths

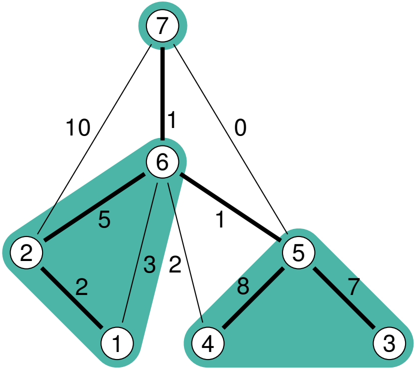
(Buchhold et al. 2018)

Observation:

- Processing similar OD-pairs in succession improves locality
- Size of sym. diff between search spaces of *u* and *v* is equal to *u*–*v* distance in elimination tree

Idea:

- Partition elimination tree into few cells with bounded diameter
- Assign IDs according to DFS order
- Reorder OD-pairs by src and dst cell

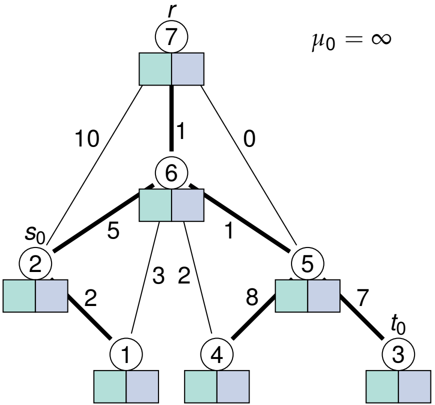


Centralized Elimination Tree Searches

(Buchhold et al. 2018)

Bundling together multiple runs:

- *k* distance labels for each vertex
- *i*-th label is distance from *i*-th src
- Relaxation updates all labels at once



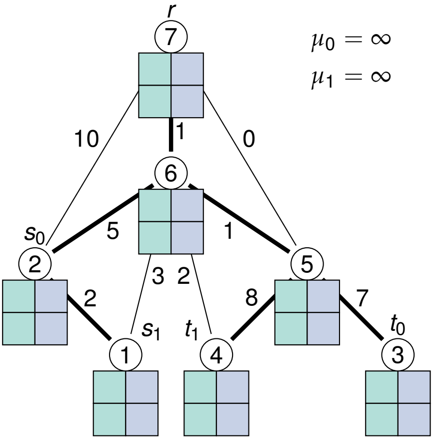
$\mu_0 = \infty$

Centralized Elimination Tree Searches

(Buchhold et al. 2018)

Bundling together multiple runs:

- *k* distance labels for each vertex
- *i*-th label is distance from *i*-th src
- Relaxation updates all labels at once

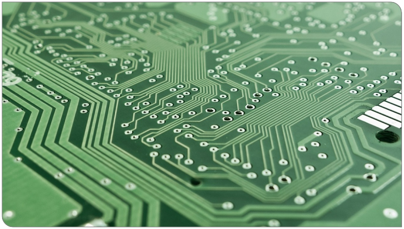


Exploiting Parallelism

(Buchhold et al. 2018)

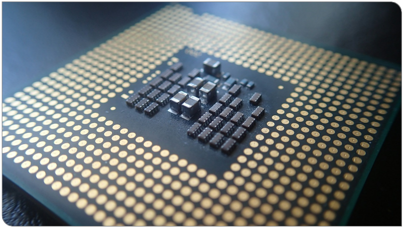
Instruction-level parallelism:

- 128-/256-bit registers
- Basic operations on multiple data items simultaneously
- We use SSE and AVX instructions



Core-level parallelism:

- SP computations are independent
- Assign OD-pairs to distinct cores
- Cumulate flow units locally, aggregate after computing all paths



Single-Threaded Traffic Assignment

algo	sorted	*k*	SIMD	S-morn	S-even	S-day	L-peak
Dij	○	1	–	5753.22	8239.57	106 687.46	1648.98
Bi-Dij	○	1	–	2459.27	3265.95	44 078.13	907.85
CH	○	1	–	90.89	120.83	1048.10	86.58
CCH	○	1	–	41.50	55.02	698.16	49.01
CCH	●	1	–	26.98	35.45	372.34	32.23
CCH	●	4	–	31.73	42.10	452.73	40.03
CCH	●	4	SSE	18.29	23.95	230.18	20.47
CCH	●	8	–	34.39	45.32	472.77	42.69
CCH	●	8	SSE	17.45	22.74	211.26	18.65
CCH	●	8	AVX	15.30	19.94	175.72	15.89
CCH	●	16	AVX	14.46	18.68	153.06	13.52
CCH	●	32	AVX	14.12	18.20	132.54	11.44
CCH	●	64	AVX	18.83	24.27	160.51	13.07

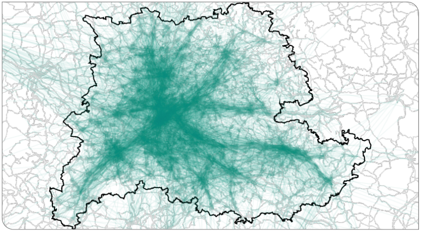
Multi-Threaded Traffic Assignment

		S-morn			S-day		
algo	cores	cust	query	total	cust	query	total
CH	1	36.12	54.06	90.89	49.52	997.60	1048.10
	16	36.46	3.95	40.48	50.24	67.66	118.01
CCH	1	1.77	11.77	14.12	2.40	129.34	132.54
	2	1.13	6.58	8.02	1.54	68.96	70.93
	4	0.61	3.85	4.62	0.83	36.42	37.48
	8	0.32	2.53	2.94	0.43	19.28	19.85
	12	0.28	2.09	2.44	0.38	13.42	13.91
	16	0.38	1.99	2.43	0.42	10.60	11.10

Traffic Assignment in Road Networks

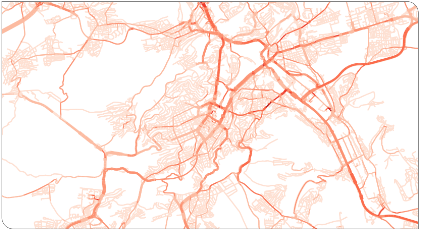
Summary:

- Traffic assignment in only 2.4 sec.
- Makes interactive apps practical
  - Road traffic centers
  - Monitoring and controlling road traffic in real time



Ongoing and future research:

- Sample demand in early iterations
- Realistic demand data generation
- Time-dependent travel-time profiles



In Timetable Networks

Institute of Theoretical Informatics
Research Group Algorithmics

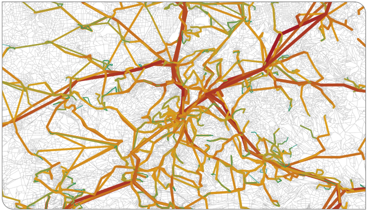
Assignments for Timetable Networks

Objective:

- Determine the utilization of vehicles in the network
- For optimizing existing networks
- For planning new lines

Data Basis:

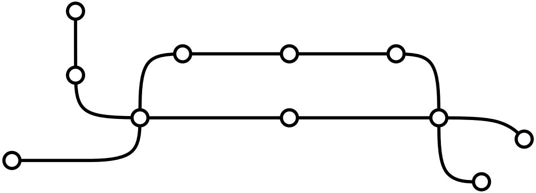
- Set of O-D pairs (as before)
- Timetable network
  - Consisting of lines and stops
  - Not represented as graph



Timetable networks

Network components:

- Set of stops (representing stops, stations, platforms, ...)
- Set of elementary connections
- Partition of the set of connections into trips

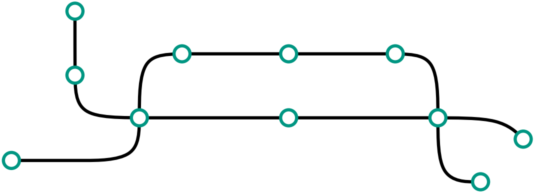


Institute of Theoretical Informatics
Research Group Algorithmics

Timetable networks

Network components:

- Set of stops (representing stops, stations, platforms, ...)
- Set of elementary connections
- Partition of the set of connections into trips

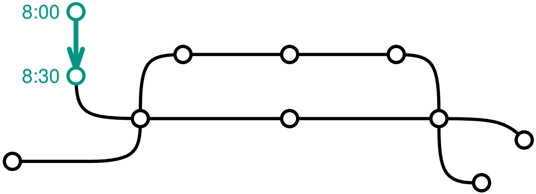


Timetable networks

Karlsruhe Institute of Technology

Network components:

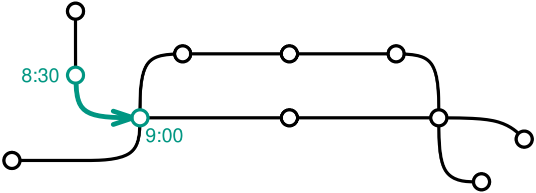
- Set of stops (representing stops, stations, platforms, ...)
- Set of elementary connections
- Partition of the set of connections into trips



Timetable networks

Network components:

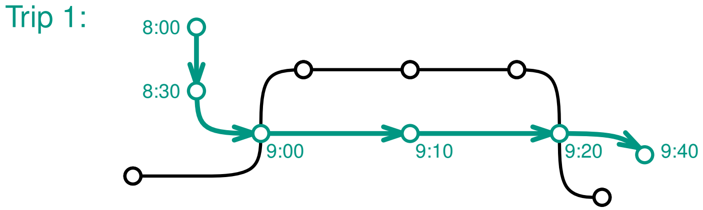
- Set of stops (representing stops, stations, platforms, ...)
- Set of elementary connections
- Partition of the set of connections into trips



Timetable networks

Network components:

- Set of stops (representing stops, stations, platforms, ...)
- Set of elementary connections
- Partition of the set of connections into trips



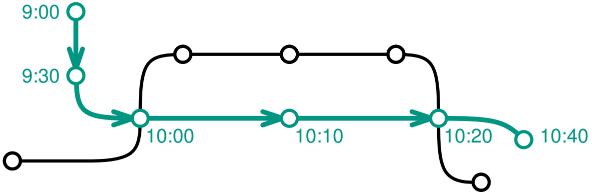
Institute of Theoretical Informatics
Research Group Algorithmics

Timetable networks

Network components:

- Set of stops (representing stops, stations, platforms, ...)
- Set of elementary connections
- Partition of the set of connections into trips

Trip 2:

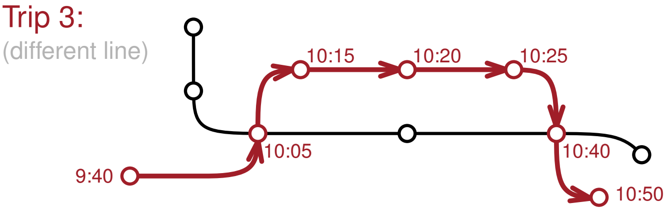


Institute of Theoretical Informatics
Research Group Algorithmics

Timetable networks

Network components:

- Set of stops (representing stops, stations, platforms, ...)
- Set of elementary connections
- Partition of the set of connections into trips



Route Planning on Timetable Networks

Types of Algorithms:

- Graph based
  - Transform timetable into time-dependent or time-expanded graph
  - Graph algorithms are applicable
  - But: Graphs get huge, special structure of timetable is lost
- Timetable based
  - Operate directly on timetable
  - Exploit knowledge of the network (chronological order, repetition of trips, ...)

Route Planning on Timetable Networks

Types of Algorithms:

- Graph based
  - Transform timetable into time-dependent or time-expanded graph
  - Graph algorithms are applicable
  - But: Graphs get huge, special structure of timetable is lost
- Timetable based
  - Operate directly on timetable
  - Exploit knowledge of the network (chronological order, repetition of trips, ...)

Special Algorithms for timetables:

- RAPTOR
- CSA
- Transfer Patterns
- Trip-Based

Route Planning on Timetable Networks

Types of Algorithms:

- Graph based
  - Transform timetable into time-dependent or time-expanded graph
  - Graph algorithms are applicable
  - But: Graphs get huge, special structure of timetable is lost
- Timetable based
  - Operate directly on timetable
  - Exploit knowledge of the network (chronological order, repetition of trips, ...)

Special Algorithms for timetables:

- RAPTOR
- CSA
- Transfer Patterns
- Trip-Based

Methods for Public Transit Traffic Assignments?

Karlsruhe Institute of Technology

Requirements:

- Fast shortest-path computations
- Easy retrieval of actual shortest paths
- Realistic assessment of a journeys quality: Perceived Travel Time
  - Time in vehicle
  - Time spent waiting
  - Number of transfers
  - Delay robustness
  - ...

Methods for Public Transit Traffic Assignments?

Requirements:

- Fast shortest-path computations
- Easy retrieval of actual shortest paths
- Realistic assessment of a journeys quality: Perceived Travel Time
  - Time in vehicle
  - Time spent waiting
  - Number of transfers
  - Delay robustness
  - ...

Best fit: CSA respectively MEAT

- Fast one-to-many queries
- Natural integration of delay robustness

Connection Scan (CSA) (Dibbelt et al. 2013, 2018)

Basic idea:

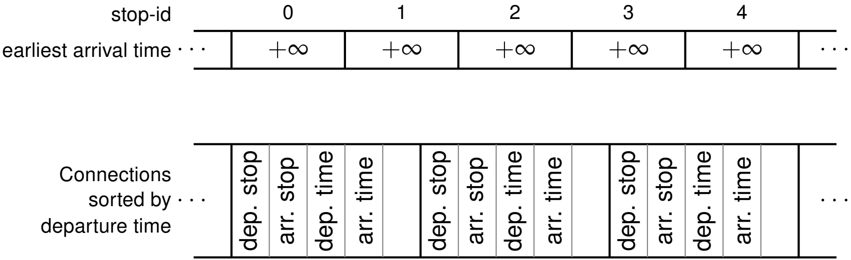
- Maintain earliest arrival times per stop
- Sort connections by their departure time
- Scan through the connections once

Special properties:

- Does not require a queue
- Uses chronological order of connections instead

Connection Scan (CSA) (Dibbelt et al. 2013, 2018)

Given: Timetable as array of connections, departure stop, departure time
Objective: Earliest arrival time at the destination



Institute of Theoretical Informatics
Research Group Algorithmics

Connection Scan (CSA) (Dibbelt et al. 2013, 2018)

Given: Timetable as array of connections, departure stop, departure time
Objective: Earliest arrival time at the destination

stop-id		0	1	2	3	4	
earliest arrival time	…	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	…

Connections sorted by departure time	…	from 1	to 3	9:00	9:25		from 3	to 4	9:15	9:45		from 3	to 4	9:40	9:55		…

Connection Scan (CSA) (Dibbelt et al. 2013, 2018)

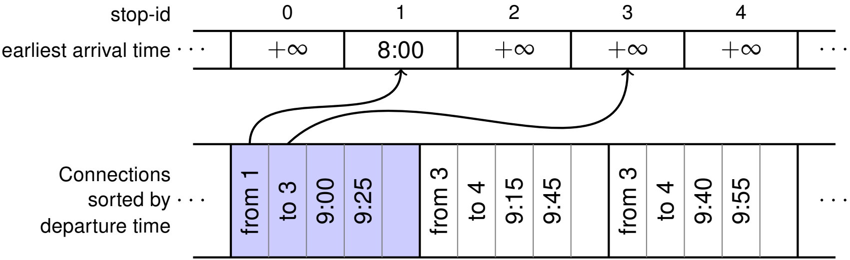
Given: Timetable as array of connections, departure stop, departure time
Objective: Earliest arrival time at the destination

stop-id		0	1	2	3	4	
earliest arrival time	…	$+\infty$	8:00	$+\infty$	$+\infty$	$+\infty$	…

Connections sorted by departure time						
…	from 1	to 3	9:00	9:25		
	from 3	to 4	9:15	9:45		
	from 3	to 4	9:40	9:55		…

Connection Scan (CSA) (Dibbelt et al. 2013, 2018)

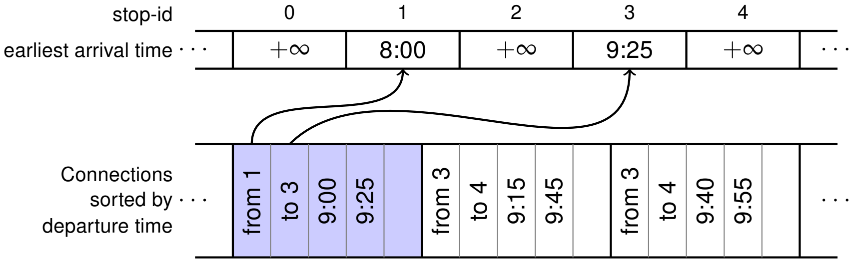
Given: Timetable as array of connections, departure stop, departure time
Objective: Earliest arrival time at the destination



Institute of Theoretical Informatics
Research Group Algorithmics

Connection Scan (CSA) (Dibbelt et al. 2013, 2018)

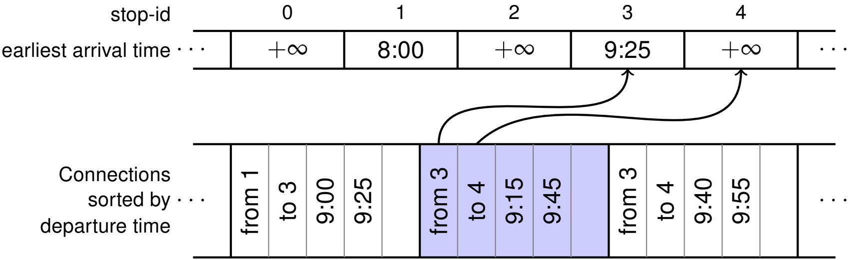
Given: Timetable as array of connections, departure stop, departure time
Objective: Earliest arrival time at the destination



Institute of Theoretical Informatics
Research Group Algorithmics

Connection Scan (CSA) (Dibbelt et al. 2013, 2018)

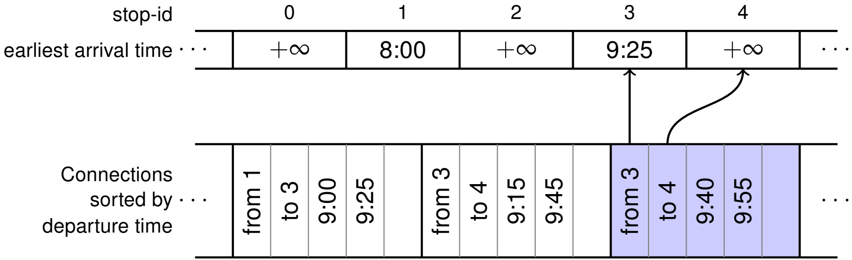
Given: Timetable as array of connections, departure stop, departure time
Objective: Earliest arrival time at the destination



Institute of Theoretical Informatics
Research Group Algorithmics

Connection Scan (CSA) (Dibbelt et al. 2013, 2018)

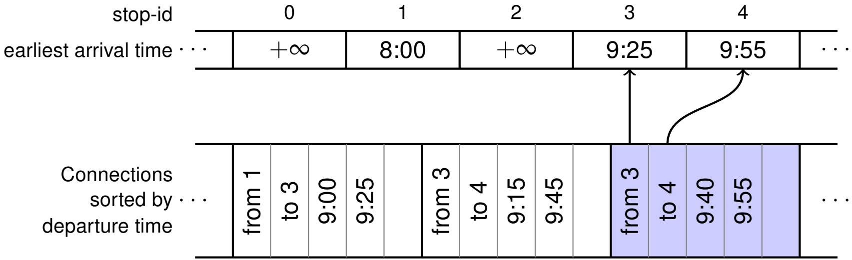
Given: Timetable as array of connections, departure stop, departure time
Objective: Earliest arrival time at the destination



Institute of Theoretical Informatics
Research Group Algorithmics

Connection Scan (CSA) (Dibbelt et al. 2013, 2018)

Given: Timetable as array of connections, departure stop, departure time
Objective: Earliest arrival time at the destination



Connection Scan (CSA) (Dibbelt et al. 2013, 2018)

Karlsruhe Institute of Technology

Given: Timetable as array of connections, departure stop, departure time
Objective: Earliest arrival time at the destination

stop-id		0	1	2	3	4	
earliest arrival time	···	$+\infty$	8:00	$+\infty$	9:25	9:55	···

Connections sorted by departure time	···	from 1	to 3	9:00	9:25		from 3	to 4	9:15	9:45		from 3	to 4	9:40	9:55		···

High efficiency since modern processors are optimized for linear memory scans

Institute of Theoretical Informatics
Research Group Algorithmics

Minimum Expected Arrival Time (MEAT)

(Dibbelt et al. 2013, 2018)

Karlsruhe Institute of Technology

Extension of CSA:

- Can handle probabilistic delays of public transit vehicles
- Enables delay robust journey planning
- Computes expected arrival times instead of absolute arrival times

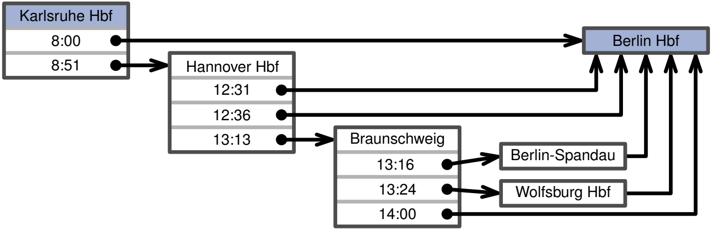
Minimum Expected Arrival Time (MEAT)
(Dibbelt et al. 2013, 2018)

Extension of CSA:

- Can handle probabilistic delays of public transit vehicles
- Enables delay robust journey planning
- Computes expected arrival times instead of absolute arrival times

Interpretation of the result:

- Consider all journeys that contribute to the expected value
- These journeys represent fall back plans:



Minimum Expected Arrival Time (MEAT)

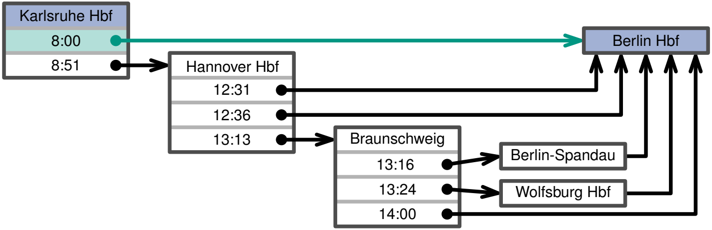
(Dibbelt et al. 2013, 2018)

Extension of CSA:

- Can handle probabilistic delays of public transit vehicles
- Enables delay robust journey planning
- Computes expected arrival times instead of absolute arrival times

Interpretation of the result:

- Consider all journeys that contribute to the expected value
- These journeys represent fall back plans:



Institute of Theoretical Informatics
Research Group Algorithmics

Minimum Expected Arrival Time (MEAT)

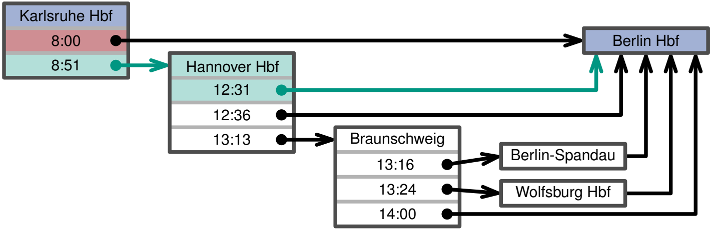
(Dibbelt et al. 2013, 2018)

Extension of CSA:

- Can handle probabilistic delays of public transit vehicles
- Enables delay robust journey planning
- Computes expected arrival times instead of absolute arrival times

Interpretation of the result:

- Consider all journeys that contribute to the expected value
- These journeys represent fall back plans:



Minimum Expected Arrival Time (MEAT)

(Dibbelt et al. 2013, 2018)

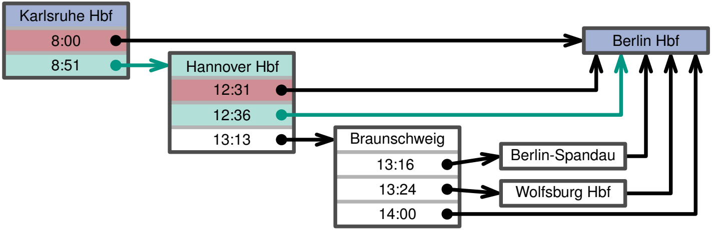
Karlsruhe Institute of Technology

Extension of CSA:

- Can handle probabilistic delays of public transit vehicles
- Enables delay robust journey planning
- Computes expected arrival times instead of absolute arrival times

Interpretation of the result:

- Consider all journeys that contribute to the expected value
- These journeys represent fall back plans:



Perceived Arrival Time (PAT)

Karlsruhe Institute of Technology

Further extending CSA:

- Represents the perceived cost of a journey
- Builds upon MEAT
- Also includes weighted costs for
  - Walking
  - Changing vehicles
  - Waiting at a stop

Institute of Theoretical Informatics
Research Group Algorithmics

Perceived Arrival Time (PAT)

Further extending CSA:

- Represents the perceived cost of a journey
- Builds upon MEAT
- Also includes weighted costs for
  - Walking
  - Changing vehicles
  - Waiting at a stop

Properties:

- As efficient as plain CSA
- Requires only a single scan of the connection array
- Builds the foundation of an efficient CSA based assignment algorithm

CSA Based Assignment (Briem et al. 2017)

Algorithm overview:

- Partition O-D pairs by destination
- Handle destinations independently of each other
- For each destination:
  1. Compute PATs from everywhere to the destination
  2. Simulate Passenger movements through the network
  3. Refine the resulting journeys

Institute of Theoretical Informatics
Research Group Algorithmics

CSA Based Assignment (Briem et al. 2017)

Algorithm overview:

- Partition O-D pairs by destination
- Handle destinations independently of each other
- For each destination:
  1. Compute PATs from everywhere to the destination
     - Using a single scan of all connections
     - In reverse (descending order of arrival time, starting from the destination)
  2. Simulate Passenger movements through the network
  3. Refine the resulting journeys

CSA Based Assignment (Briem et al. 2017)

Algorithm overview:

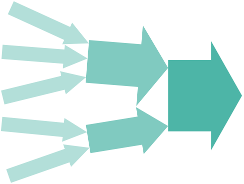
- Partition O-D pairs by destination
- Handle destinations independently of each other
- For each destination:
  1. Compute PATs from everywhere to the destination
     - Using a single scan of all connections
     - In reverse (descending order of arrival time, starting from the destination)
  2. Simulate Passenger movements through the network
     - Also using a single scan of all connections
     - In normal order (ascending order of arrival time)
     - Use PATs to decide if passengers use a connection or not
  3. Refine the resulting journeys

Institute of Theoretical Informatics
Research Group Algorithmics

CSA Based Assignment (Briem et al. 2017)

Passenger Movement Simulation:

- PAT of each connection is known
- Passengers are generated at their origin
- Passengers move towards their destination
  (One connection at a time)
- Whether a connection is used, depends on the connections PAT
- While getting closer to the destination:
  - Paths of individual passengers converge
  - More and more passengers collect at the same stops
  - All passengers at stop can use the same connections
  - Computation for this connection is only performed once

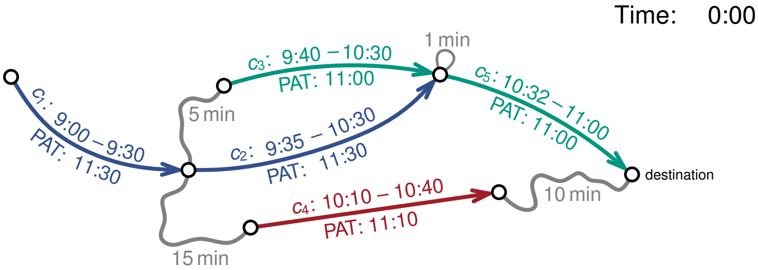


⇒ Synergy effects as more passengers gather at the same stops

CSA Based Assignment (Briem et al. 2017)

Passenger Movement Simulation Example:

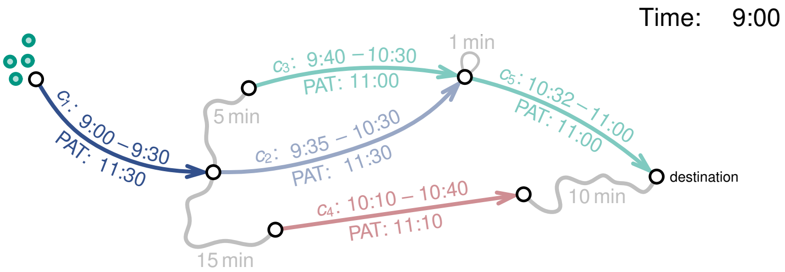
- Process connections in ascending order by departure time
- For each connection *c*:



CSA Based Assignment (Briem et al. 2017)

Passenger Movement Simulation Example:

- Process connections in ascending order by departure time
- For each connection c:
  1. Generate passengers with origin at the departure stop of c

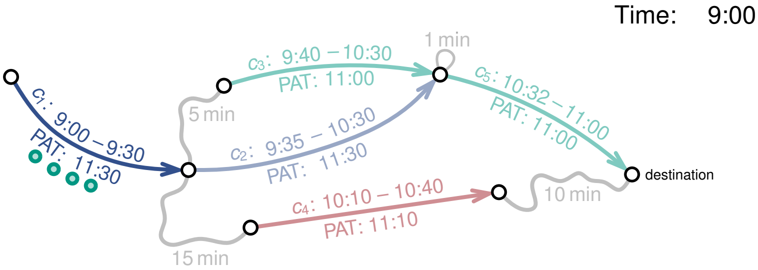


Institute of Theoretical Informatics
Research Group Algorithmics

CSA Based Assignment (Briem et al. 2017)

Passenger Movement Simulation Example:

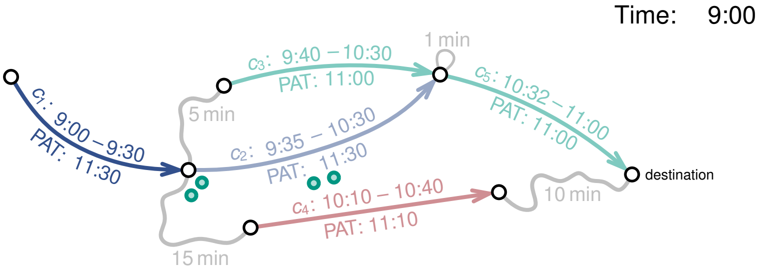
- Process connections in ascending order by departure time
- For each connection c:
  - ❷ Decide which passengers enter the connection



CSA Based Assignment (Briem et al. 2017)

Passenger Movement Simulation Example:

- Process connections in ascending order by departure time
- For each connection c:
  - ③ Decide which passengers leave the trip

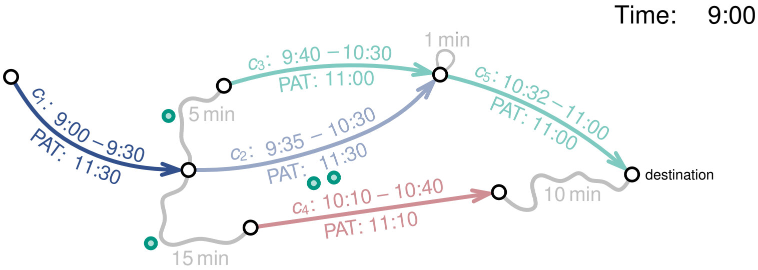


Institute of Theoretical Informatics
Research Group Algorithmics

CSA Based Assignment (Briem et al. 2017)

Passenger Movement Simulation Example:

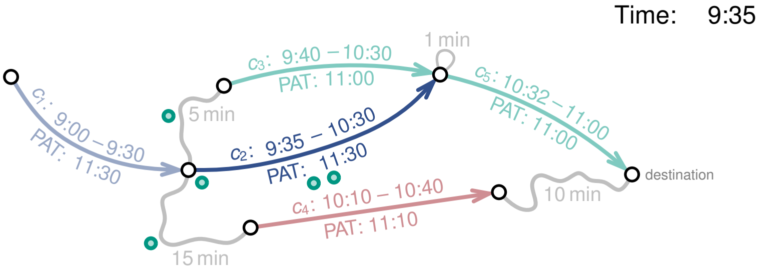
- Process connections in ascending order by departure time
- For each connection c:
  - ④ Move disembarking passengers to their next stop



CSA Based Assignment (Briem et al. 2017)

Passenger Movement Simulation Example:

- Process connections in ascending order by departure time
- For each connection *c*:
  1. Generate passengers with origin at the departure stop of *c*

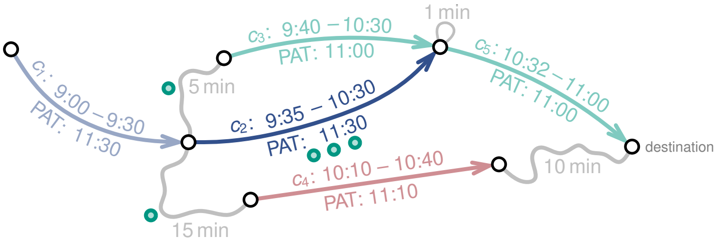


CSA Based Assignment (Briem et al. 2017)

Passenger Movement Simulation Example:

- Process connections in ascending order by departure time
- For each connection c:
  - ❷ Decide which passengers enter the connection

Time: 9:35

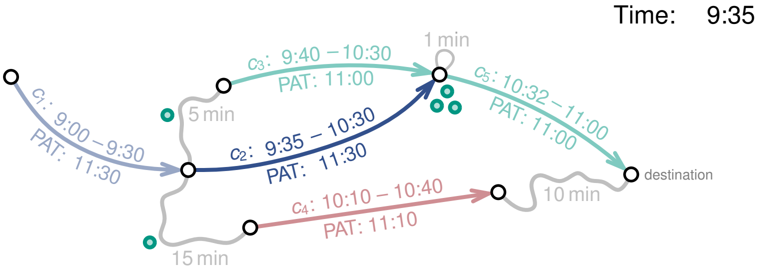


Institute of Theoretical Informatics
Research Group Algorithmics

CSA Based Assignment (Briem et al. 2017)

Passenger Movement Simulation Example:

- Process connections in ascending order by departure time
- For each connection c:
  - ③ Decide which passengers leave the trip

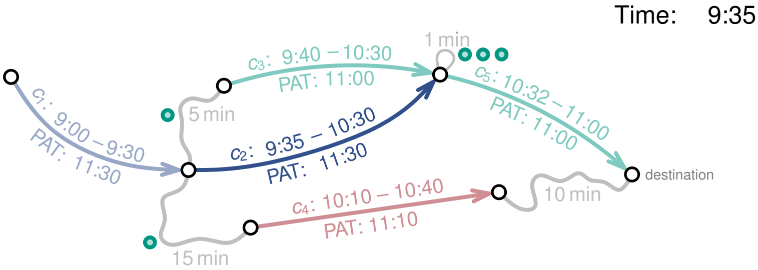


Institute of Theoretical Informatics
Research Group Algorithmics

CSA Based Assignment (Briem et al. 2017)

Passenger Movement Simulation Example:

- Process connections in ascending order by departure time
- For each connection c:
  - ④ Move disembarking passengers to their next stop



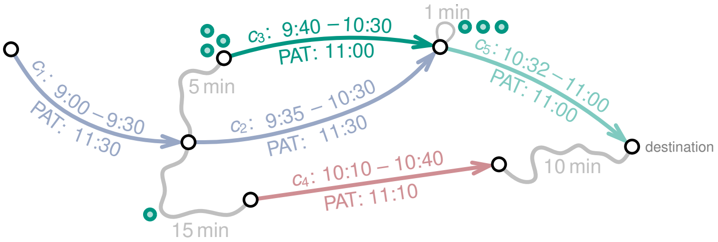
Institute of Theoretical Informatics
Research Group Algorithmics

CSA Based Assignment (Briem et al. 2017)

Passenger Movement Simulation Example:

- Process connections in ascending order by departure time
- For each connection c:
  1. Generate passengers with origin at the departure stop of c

Time: 9:40



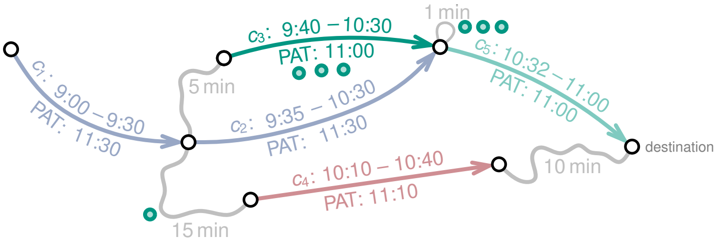
Institute of Theoretical Informatics
Research Group Algorithmics

CSA Based Assignment (Briem et al. 2017)

Passenger Movement Simulation Example:

- Process connections in ascending order by departure time
- For each connection c:
  - ❷ Decide which passengers enter the connection

Time: 9:40

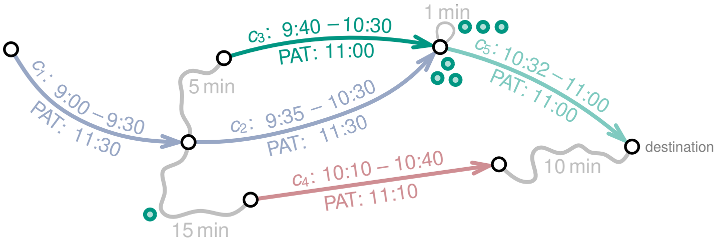


CSA Based Assignment (Briem et al. 2017)

Passenger Movement Simulation Example:

- Process connections in ascending order by departure time
- For each connection c:
  - ❸ ...

Time: 9:40



Institute of Theoretical Informatics
Research Group Algorithmics

CSA Based Assignment (Briem et al. 2017)

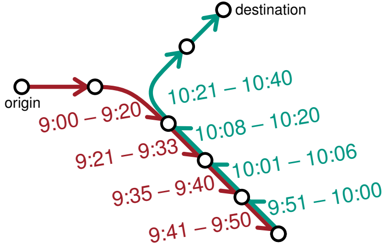
Journey Refinement: **(Remove unwanted cycles)**

- Cycle definition: Visiting a stop more than once
- Assigning cycles might be undesirable
- Journey with cycle can have minimum PAT
- High waiting cost leads to cycles

CSA Based Assignment (Briem et al. 2017)

Journey Refinement: **(Remove unwanted cycles)**

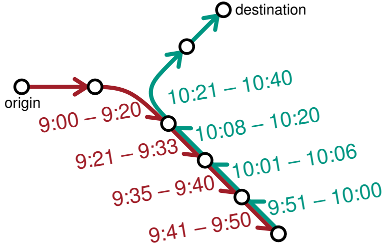
- Cycle definition: Visiting a stop more than once
- Assigning cycles might be undesirable
- Journey with cycle can have minimum PAT
- High waiting cost leads to cycles

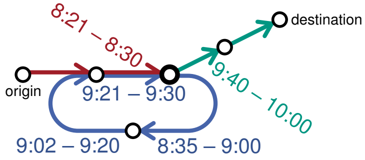


CSA Based Assignment (Briem et al. 2017)

Journey Refinement: **(Remove unwanted cycles)**

- Cycle definition: Visiting a stop more than once
- Assigning cycles might be undesirable
- Journey with cycle can have minimum PAT
- High waiting cost leads to cycles





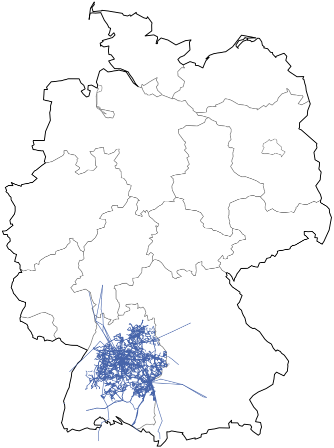
Institute of Theoretical Informatics
Research Group Algorithmics

CSA Based Assignment (Briem et al. 2017)

Benchmark Instance:

- Greater region of Stuttgart
- Reaching as far as Frankfurt, Basel or Munich
- Comprises the traffic of one day

Number of vertices	15 115
Number of stops	13 941
Number of edges	33 890
Number of edges without loops	18 775
Number of connections	780 042
Number of trips	47 844
Number of passenger	1 249 910



CSA Based Assignment (Briem et al. 2017)

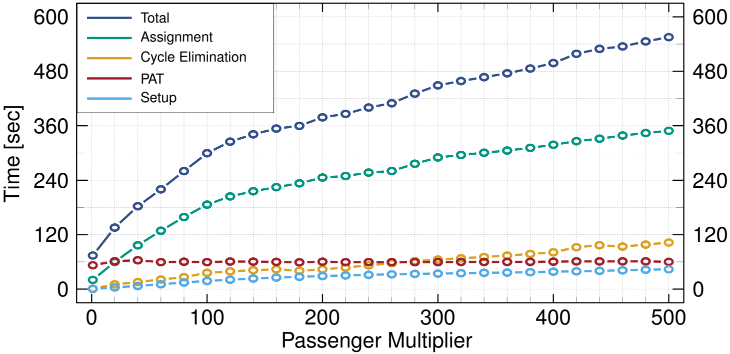
Running Time and Passenger Multiplier:

- Algorithm assigns only one journey per O-D pair
- However probabilistic distribution of journeys is desired
- Solution: simulate multiple passengers per O-D pair

CSA Based Assignment (Briem et al. 2017)

Running Time and Passenger Multiplier:

- Algorithm assigns only one journey per O-D pair
- However probabilistic distribution of journeys is desired
- Solution: simulate multiple passengers per O-D pair



CSA Based Assignment (Briem et al. 2017)

Comparison with Visum:

- Commercial tool for traffic planning

CSA Based Assignment (Briem et al. 2017)

Comparison with Visum:

- Commercial tool for traffic planning
- The computation in Visum takes ~30 minutes (8 threads)
- The CSA based assignment takes 39 seconds (4 threads)
- Both assignments look similar

	VISUM			CSA Based Assignment		
Quantity	min	mean	max	min	mean	max
Total travel time [min]	2.98	46.885	429.00	2.98	47.199	429.00
Time spent in vehicle [min]	0.02	21.059	380.00	0.02	21.231	323.97
Time spent walking [min]	2.00	22.394	149.00	2.00	22.476	149.00
Time spent waiting [min]	0.00	3.432	217.02	0.00	3.492	217.02
Trips per passenger	1.00	1.771	6.00	1.00	1.746	8.00
Connections per passenger	1.00	9.396	109.00	1.00	9.474	97.00
Passengers per connection	0.00	12.740	1 290.10	0.00	12.847	1 233.60

Ongoing Research

Karlsruhe Institute of Technology

Multimodal Assignments:

- Goal: Consider multiple modes of transportation at once
- Problem: Combining timetable and non-timetable networks is hard

Ongoing Research

Multimodal Assignments:

- Goal: Consider multiple modes of transportation at once
- Problem: Combining timetable and non-timetable networks is hard
- ULTRA: (Baum et al. 2019)
  - Enables UnLimited TRAnsfers for many Public Transit algorithms
  - Is also combinable with the CSA based assignment (Sauer et al. 2019)
  - First efficient assignment for public transit with secondary transfer mode

Ongoing Research

Multimodal Assignments:

- Goal: Consider multiple modes of transportation at once
- Problem: Combining timetable and non-timetable networks is hard
- ULTRA: (Baum et al. 2019)
  - Enables UnLimited TRAnsfers for many Public Transit algorithms
  - Is also combinable with the CSA based assignment (Sauer et al. 2019)
  - First efficient assignment for public transit with secondary transfer mode

Consider Vehicle capacities:

- Similar to assignments on road networks
- PAT depends on utilization
- Iterative approach

In Combined Networks

Next steps: True Multimodal Assignments

State of the art:

- Applications already handle different modes of transportation
- However: mode choice and assignment sequentially
  1. Choose travel mode
  2. Select route for chosen mode of transportation

Integrate mode choice with route assignment:

- Integrate both assignment types
- Combine O-D for road networks and for public transit
- Algorithms assigns both: journey and travel mode

Institute of Theoretical Informatics
Research Group Algorithmics

Thank you for your attention!

References I

Bar-Gera, H. (2010). Traffic assignment by paired alternative segments. *Transportation Research Part B: Methodological*, 44(8–9):1022–1046.

Bast, H., Delling, D., Goldberg, A. V., Müller-Hannemann, M., Pajor, T., Sanders, P., Wagner, D., and Werneck, R. F. (2016). Route planning in transportation networks. In Kliemann, L. and Sanders, P., editors, *Algorithm Engineering: Selected Results and Surveys*, volume 9220 of *Lecture Notes in Computer Science*, pages 19–80. Springer.

Baum, M., Buchhold, V., Sauer, J., Wagner, D., and Zündorf, T. (2019). UnLimited TRAnsfers for Multi-Modal Route Planning: An Efficient Solution. In *27th Annual European Symposium on Algorithms (ESA 2019)*, Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

Briem, L., Buck, S., Ebhart, H., Mallig, N., Strasser, B., Vortisch, P., Wagner, D., and Zündorf, T. (2017). Efficient Traffic Assignment for Public Transit Networks. In *16th International Symposium on Experimental Algorithms (SEA 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

References II

Buchhold, V., Sanders, P., and Wagner, D. (2018). Real-time traffic assignment using fast queries in customizable contraction hierarchies. In D'Angelo, G., editor, *Proceedings of the 17th International Symposium on Experimental Algorithms (SEA'18)*, volume 103 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:15. Schloss Dagstuhl.

Bureau of Public Roads (1964). *Traffic Assignment Manual*. U.S. Department of Commerce.

Dafermos, S. (1968). *Traffic Assignment and Resource Allocation in Transportation Networks*. PhD thesis, Johns Hopkins University.

Dial, R. B. (2006). A path-based user-equilibrium traffic assignment algorithm that obviates path storage and enumeration. *Transportation Research Part B: Methodological*, 40(10):917–936.

Dibbelt, J., Pajor, T., Strasser, B., and Wagner, D. (2013). Intriguingly Simple and Fast Transit Routing. In *International Symposium on Experimental Algorithms*, pages 43–54. Springer.

References III

Dibbelt, J., Pajor, T., Strasser, B., and Wagner, D. (2018). Connection Scan Algorithm. *Journal of Experimental Algorithmics (JEA)*, 23(1):1–7.

Dibbelt, J., Strasser, B., and Wagner, D. (2016). Customizable contraction hierarchies. *ACM Journal of Experimental Algorithmics*, 21(1):1.5:1–1.5:49.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271.

Florian, M., Constantin, I., and Florian, D. (2009). A new look at projected gradient method for equilibrium assignment. *Transportation Research Record*, 2090(1):10–16.

Frank, M. and Wolfe, P. (1956). An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110.

Gentile, G. (2014). Local user cost equilibrium: A bush-based algorithm for traffic assignment. *Transportmetrica A: Transport Science*, 10(1):15–54.

Jayakrishnan, R., Tsai, W. K., Prashker, J., and Rajadhyaksha, S. (1994). Faster path-based algorithm for traffic assignment. *Transportation Research Record*, 1443:75–83.

References IV

Kumar, A. and Peeta, S. (2011). An improved social pressure algorithm for static deterministic user equilibrium traffic assignment problem. In *Proceedings of the 90th Transportation Research Board Annual Meeting (TRB'11)*.

Mitradjieva, M. and Lindberg, P. O. (2013). The stiff is moving – conjugate direction frank-wolfe methods with applications to traffic assignment. *Transportation Science*, 47(2):280–293.

Sauer, J., Wagner, D., and Zündorf, T. (2019). Efficient Computation of Multi-Modal Public Transit Traffic Assignments using ULTRA. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM.