Karlsruhe Institute of Technology

Time-Dependent Contraction Hierarchies and Approximation

Gernot Veit Batz, Robert Geisberger, Sabine Neubauer, and Peter Sanders
– *{batz, geisberger, sanders}@kit.edu*

Institute for Theoretical Computer Science, Algorithmics II

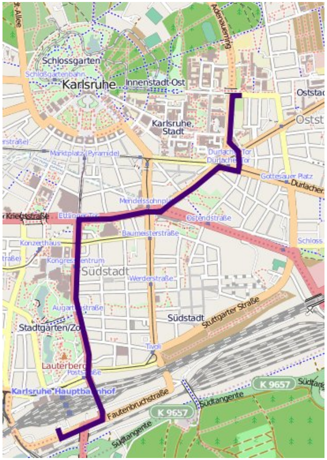
Time-Dependent Route Planning

Motivation

From Karlsruhe Main Station
to Karlsruhe Computer Science Building

At 3:00 at night:

- Empty streets
- Through the city center.



Map (c) www.openstreetmap.org and contributors, licence CC-BY-SA (www.creativecommons.org)

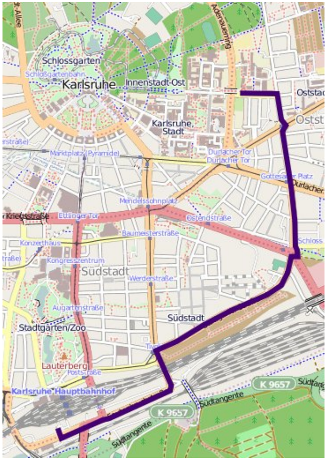
Time-Dependent Route Planning

Motivation

From Karlsruhe Main Station
to Karlsruhe Computer Science Building

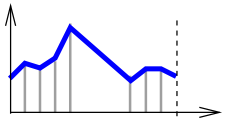
At 8:00 in the morning:

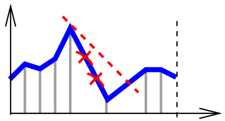
- Rush hour
- Avoid crowded junctions.



Map (c) www.openstreetmap.org and contributors,
licence CC-BY-SA (www.creativecommons.org)

Time-Dependent Route Planning





- Edge weights are travel time functions
  - {point in time $\mapsto$ travel time period}
  - piecewise linear
  - FIFO-property – waiting does not help
- Kinds of user queries
  - Earliest arrival query:
    shortest route depending on a given departure time
  - Profile query:
    travel time profile:
    {departure time $\mapsto$ travel time period}

Problem Statement

Problem 1: Earliest arrival queries

- solved by TCHs [ALENEX 09]
    - "time-dependent Contraction Hierarchies"
    - very fast
    - but need lots of space

Problem 2: Travel time profile queries

- not solved efficiently before

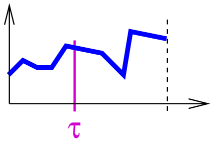
Both solved by carefully using approximation...
...without sacrificing exactness

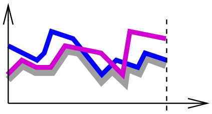
Operations on Travel Time Functions

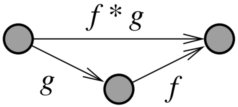
We need three operations

- Evaluation: $f(\tau)$ — "$O(1)$" Time
- Merging: $\min(f, g)$ — $O(|f| + |g|)$ Time
- Chaining: $f * g$ (f "after" g) — $O(|f| + |g|)$ Time

Note: $\min(f, g)$ and $f * g$ have $O(|f| + |g|)$ points each.
$\Rightarrow$ Increase of complexity





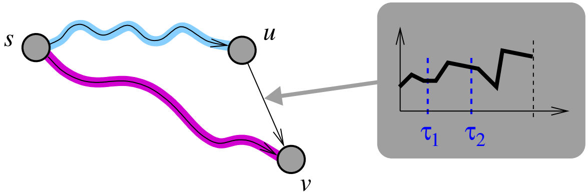


Time-Dependent Dijkstra

[Dreyfus 69]

Only one difference to standard Dijkstra:

- Cost of relaxed edge (u, v) depends...
- ...on current tentative shortest path to u.



Edge relaxation: $d(v) := \min\Big(d(v),\ d(u) + f_{uv}(d(u))\Big)$

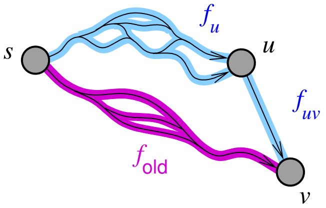
Profile Search

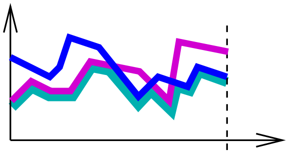
[Orda Rom 90]

Modified Dijkstra:

- Computes travel time profiles
- Node labels are travel time functions

Edge relaxation: $f_{\text{new}} := \min(f_{\text{old}}, f_{uv} * f_u)$





$\Rightarrow$ A **label correcting** **very expensive** algorithm

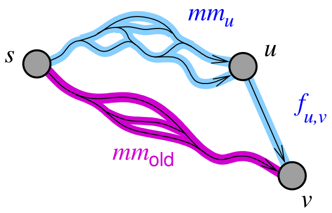
Interval Search

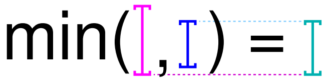
[ALENEX 09]

Approximate version of profile search:

- Computes upper and lower bounds
- Node labels are intervals $mm_u := [\min f_u, \max f_u]$

Edge relaxation: $mm_{\text{new}} := \min(mm_{\text{old}}, mm_u + [\min f_{uv}, \max f_{uv}])$



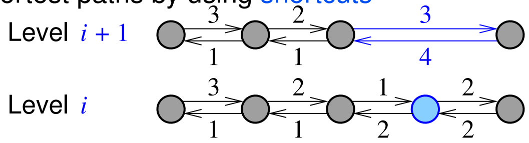


$\Rightarrow$ A **label correcting** **much cheaper** algorithm

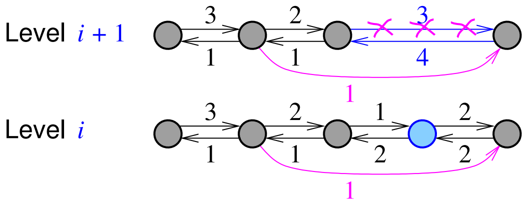
TCH Structure

Time-Dependent Contraction Hierarchies [ALENEX 09]

- Order nodes by importance
- Obtain the next higher level by contracting the next node
- Preserve shortest paths by using shortcuts



- **But** shortcuts are not always needed:



- TCH needs very much memory in time-dependent case

Earliest Arrival Query with TCHs

[ALENEX 09]

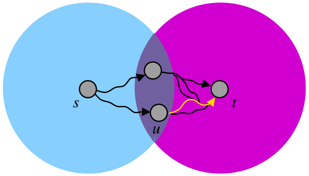
Phase 1: Bidirectional **upward** search:

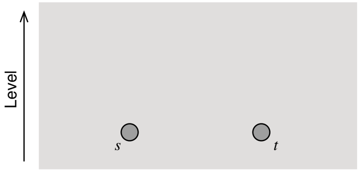
- Forward: time-dependent Dijkstra
- Backward: interval search

⇝ meeting nodes

Phase 2: Downward search

- Forward: time-dependent Dijkstra
- Uses only edges touched by backward/upward search





Earliest Arrival Query with TCHs

[ALENEX 09]

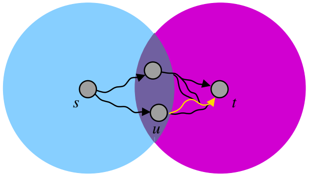
Phase 1: Bidirectional **upward** search:

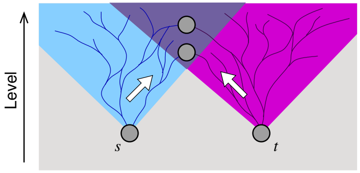
- Forward: time-dependent Dijkstra
- Backward: interval search

$\rightsquigarrow$ meeting nodes

Phase 2: Downward search

- Forward: time-dependent Dijkstra
- Uses only edges touched by backward/upward search





Earliest Arrival Query with TCHs

[ALENEX 09]

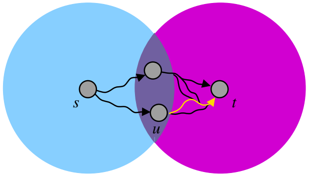
Phase 1: Bidirectional **upward** search:

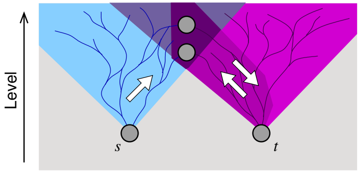
- Forward: time-dependent Dijkstra
- Backward: interval search

$\rightsquigarrow$ meeting nodes

Phase 2: Downward search

- Forward: time-dependent Dijkstra
- Uses only edges touched by backward/upward search





Earliest Arrival Query with TCHs

[ALENEX 09]

Performance (current implementation):

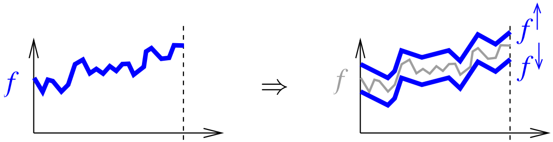
- Running time:
  - Germany: 0.72 ms, 1 440 speedup
  - Europe: 1.89 ms, 1 807 speedup
- Memory usage:
  - Germany: total 994 byte/node, overhead 899 byte/node
  - Europe: total 589 byte/node, overhead 513 byte/node

$\leadsto$ Too much space!

ATCH = Approximated TCH

A Space Efficient Data Structure

- **For each edge of the TCH do**
  - Replace weights of shortcuts by two approximated functions...
  - ...an upper bound
  - ...a lower bound
  - ...both with much less points
  - ...lower bound given implicitly by upper bound



$\Rightarrow$ **Needs much less space (10 vs. 23 points).**

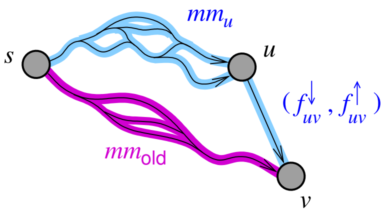
Note: Weights of non-shortcuts still exact

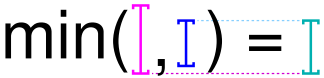
Arrival Interval Search

Approximate version of earliest arrival query:

- Computes upper and lower bounds
- **Node labels** are intervals $mm_u = [a, b] \ni f_u(\tau_0)$

Edge relaxation: $mm_{\text{new}} := \min(mm_{\text{old}}, mm_u + [f^{\downarrow}_{uv}(a), f^{\uparrow}_{uv}(b)])$





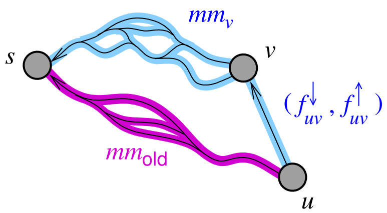
$\Rightarrow$ A **label correcting** algorithm

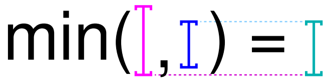
Backward Travel Time Interval Search

Dual to arrival interval search:

- Computes upper and lower bounds of travel time
- **Node labels** are intervals $mm_u = [a, b] \ni$ travel time from u to t

Edge relaxation: $mm_{\text{new}} := \min(mm_{\text{old}}, mm_v + [\min f^{\downarrow}_{uv}|_D, \max f^{\uparrow}_{uv}|_D])$
where $D \ni$ departure time at u for mm_v

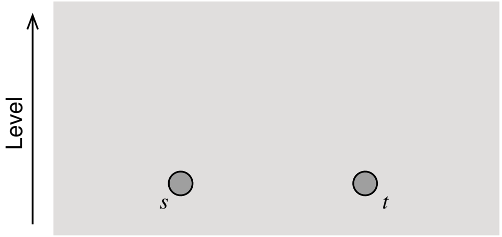




⇒ A **label correcting** algorithm

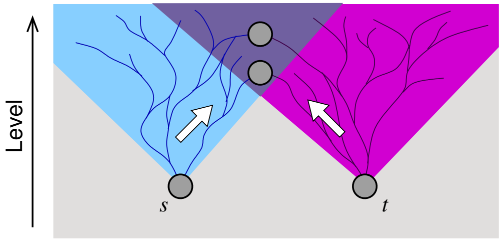
Earliest Arrival Query with ATCHs

- **Phase 1:** Bidirectional **upward** search
  - Forward: arrival interval search
  - Backward: interval search
- **Phase 2:** Forward/downward arrival interval search
- **Phase 3:** Backward/upward travel time interval search
- **Phase 4:** Unpacking and time-dependent Dijkstra



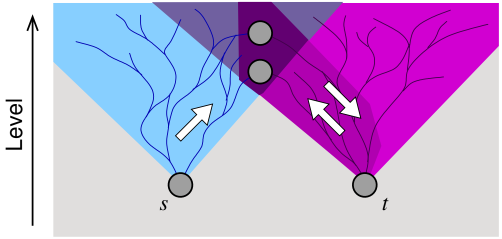
Earliest Arrival Query with ATCHs

- **Phase 1:** Bidirectional **upward** search
  - Forward: arrival interval search
  - Backward: interval search
- **Phase 2:** Forward/downward arrival interval search
- **Phase 3:** Backward/upward travel time interval search
- **Phase 4:** Unpacking and time-dependent Dijkstra



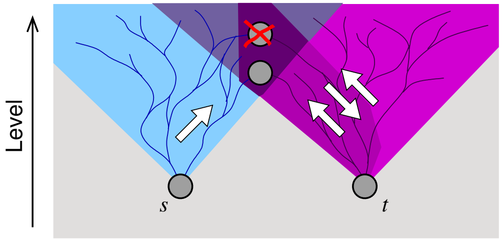
Earliest Arrival Query with ATCHs

- **Phase 1:** Bidirectional **upward** search
  - Forward: arrival interval search
  - Backward: interval search
- **Phase 2:** Forward/downward arrival interval search
- **Phase 3:** Backward/upward travel time interval search
- **Phase 4:** Unpacking and time-dependent Dijkstra



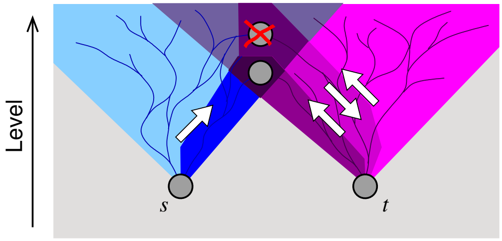
Earliest Arrival Query with ATCHs

- **Phase 1:** Bidirectional **upward** search
  - Forward: arrival interval search
  - Backward: interval search
- **Phase 2:** Forward/downward arrival interval search
- **Phase 3:** Backward/upward travel time interval search
- **Phase 4:** Unpacking and time-dependent Dijkstra



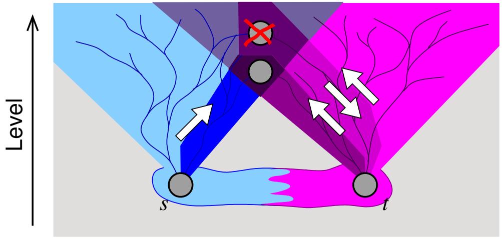
Earliest Arrival Query with ATCHs

- **Phase 1:** Bidirectional **upward** search
  - Forward: arrival interval search
  - Backward: interval search
- **Phase 2:** Forward/downward arrival interval search
- **Phase 3:** Backward/upward travel time interval search
- **Phase 4:** Unpacking and time-dependent Dijkstra



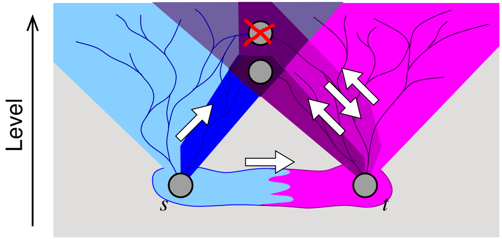
Earliest Arrival Query with ATCHs

- **Phase 1:** Bidirectional **upward** search
  - Forward: arrival interval search
  - Backward: interval search
- **Phase 2:** Forward/downward arrival interval search
- **Phase 3:** Backward/upward travel time interval search
- **Phase 4:** Unpacking and time-dependent Dijkstra



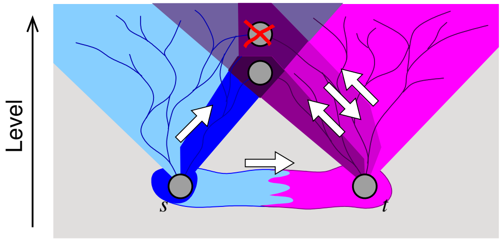
Earliest Arrival Query with ATCHs

- **Phase 1:** Bidirectional **upward** search
  - Forward: arrival interval search
  - Backward: interval search
- **Phase 2:** Forward/downward arrival interval search
- **Phase 3:** Backward/upward travel time interval search
- **Phase 4:** Unpacking and time-dependent Dijkstra



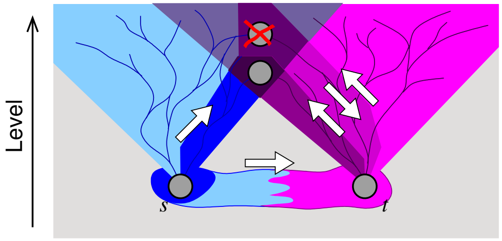
Earliest Arrival Query with ATCHs

- **Phase 1:** Bidirectional **upward** search
  - Forward: arrival interval search
  - Backward: interval search
- **Phase 2:** Forward/downward arrival interval search
- **Phase 3:** Backward/upward travel time interval search
- **Phase 4:** Unpacking and time-dependent Dijkstra



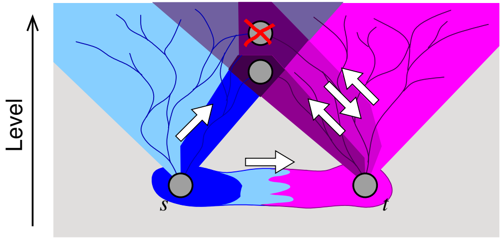
Earliest Arrival Query with ATCHs

- **Phase 1:** Bidirectional **upward** search
  - Forward: arrival interval search
  - Backward: interval search
- **Phase 2:** Forward/downward arrival interval search
- **Phase 3:** Backward/upward travel time interval search
- **Phase 4:** Unpacking and time-dependent Dijkstra



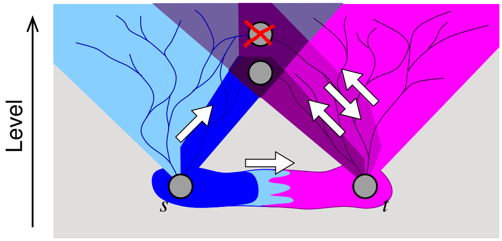
Earliest Arrival Query with ATCHs

- **Phase 1:** Bidirectional **upward** search
  - Forward: arrival interval search
  - Backward: interval search
- **Phase 2:** Forward/downward arrival interval search
- **Phase 3:** Backward/upward travel time interval search
- **Phase 4:** Unpacking and time-dependent Dijkstra



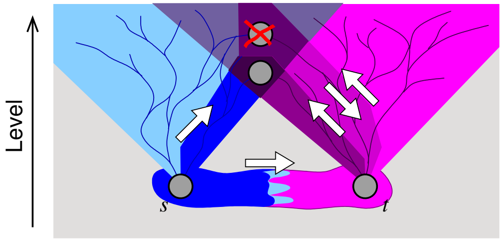
Earliest Arrival Query with ATCHs

- **Phase 1:** Bidirectional **upward** search
  - Forward: arrival interval search
  - Backward: interval search
- **Phase 2:** Forward/downward arrival interval search
- **Phase 3:** Backward/upward travel time interval search
- **Phase 4:** Unpacking and time-dependent Dijkstra



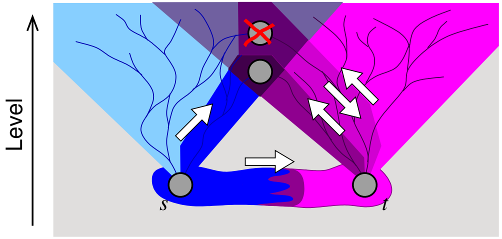
Earliest Arrival Query with ATCHs

- **Phase 1:** Bidirectional **upward** search
  - Forward: arrival interval search
  - Backward: interval search
- **Phase 2:** Forward/downward arrival interval search
- **Phase 3:** Backward/upward travel time interval search
- **Phase 4:** Unpacking and time-dependent Dijkstra



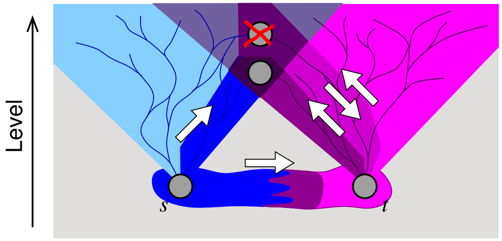
Earliest Arrival Query with ATCHs

- **Phase 1:** Bidirectional **upward** search
  - Forward: arrival interval search
  - Backward: interval search
- **Phase 2:** Forward/downward arrival interval search
- **Phase 3:** Backward/upward travel time interval search
- **Phase 4:** Unpacking and time-dependent Dijkstra



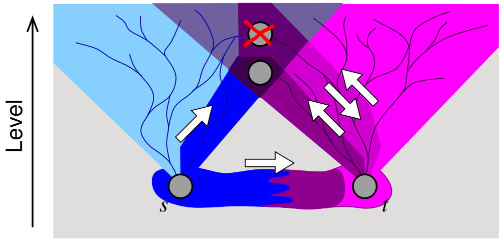
Earliest Arrival Query with ATCHs

- **Phase 1:** Bidirectional **upward** search
  - Forward: arrival interval search
  - Backward: interval search
- **Phase 2:** Forward/downward arrival interval search
- **Phase 3:** Backward/upward travel time interval search
- **Phase 4:** Unpacking and time-dependent Dijkstra



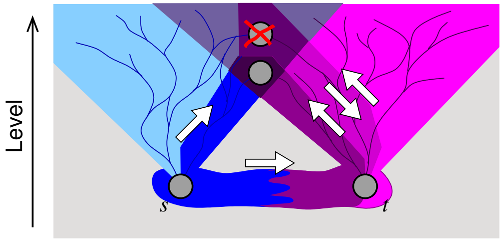
Earliest Arrival Query with ATCHs

- **Phase 1:** Bidirectional **upward** search
  - Forward: arrival interval search
  - Backward: interval search
- **Phase 2:** Forward/downward arrival interval search
- **Phase 3:** Backward/upward travel time interval search
- **Phase 4:** Unpacking and time-dependent Dijkstra



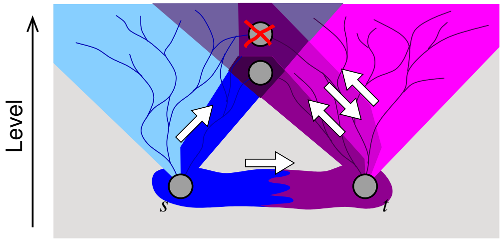
Earliest Arrival Query with ATCHs

- **Phase 1:** Bidirectional **upward** search
  - Forward: arrival interval search
  - Backward: interval search
- **Phase 2:** Forward/downward arrival interval search
- **Phase 3:** Backward/upward travel time interval search
- **Phase 4:** Unpacking and time-dependent Dijkstra



Earliest Arrival Query with ATCHs

- **Phase 1:** Bidirectional **upward** search
  - Forward: arrival interval search
  - Backward: interval search
- **Phase 2:** Forward/downward arrival interval search
- **Phase 3:** Backward/upward travel time interval search
- **Phase 4:** Unpacking and time-dependent Dijkstra



Earliest Arrival Query with ATCHs

Performance:

graph	method	ε [%]	space [B/n] ABS	space [B/n] OVH	query [ms]	query SPD	error [%] MAX	error [%] AVG
Germany	TCH	–	994	899	0.72	1 440	0.00	0.00
	ATCH	1	239	144	1.27	816	0.00	0.00
	ATCH	∞	118	23	1.45	714	0.00	0.00
Europe	TCH	–	589	513	1.89	1 807	0.00	0.00
	ATCH	1	207	131	2.47	1 396	0.00	0.00
	ATCH	∞	99	23	15.43	221	0.00	0.00

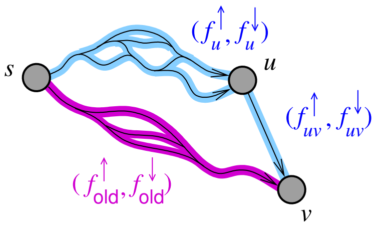
- much less space than TCHs
- only moderate slow down
- still exact

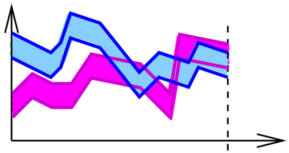
Approximated Profile Search

Approximated version of profile search:

- Computes upper and lower bounds of travel time profiles
- Node labels are pairs of travel time functions

Edge relaxation: $f_{\text{new}} := \min((f^{\uparrow}_{\text{old}}, f^{\downarrow}_{\text{old}}), (f^{\uparrow}_{uv} * f^{\uparrow}_{u}, f^{\downarrow}_{uv} * f^{\downarrow}_{u}))$


$(f^{\uparrow}_{u}, f^{\downarrow}_{u})$
s
u
$(f^{\uparrow}_{uv}, f^{\downarrow}_{uv})$
$(f^{\uparrow}_{\text{old}}, f^{\downarrow}_{\text{old}})$
v



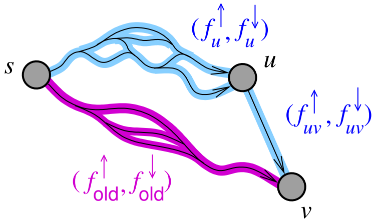
$\Rightarrow$ A **label correcting** algorithm

Approximated Profile Search

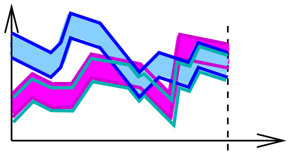
Approximated version of profile search:

- Computes upper and lower bounds of travel time profiles
- Node labels are pairs of travel time functions

Edge relaxation: $f_{\text{new}} := \min((f^{\uparrow}_{\text{old}}, f^{\downarrow}_{\text{old}}), (f^{\uparrow}_{uv} * f^{\uparrow}_{u}, f^{\downarrow}_{uv} * f^{\downarrow}_{u}))$



s
u
v
$(f^{\uparrow}_{u}, f^{\downarrow}_{u})$
$(f^{\uparrow}_{uv}, f^{\downarrow}_{uv})$
$(f^{\uparrow}_{\text{old}}, f^{\downarrow}_{\text{old}})$

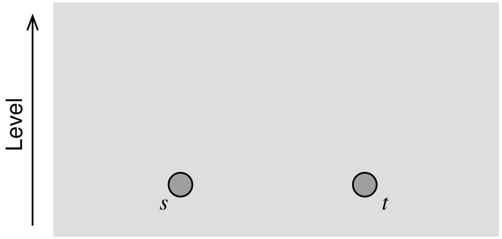


$\Rightarrow$ A **label correcting** algorithm

Travel Time Profile Query with ATCHs

Using Preceeding Interval Search: Space Efficient

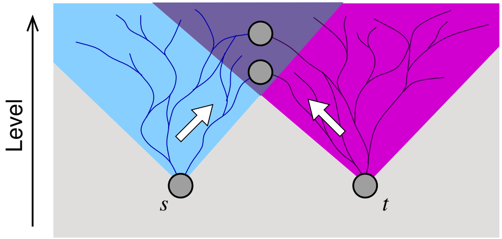
- **Phase 1:** Bidirectional **upward** search
  - Forward: interval search
  - Backward: interval search
- **Phase 2:** Bidirectional **upward** search
  - Forward: approximate profile search
  - Backward: approximate profile search
- **Phase 3:** Unpacking and profile search



Travel Time Profile Query with ATCHs

Using Preceeding Interval Search: Space Efficient

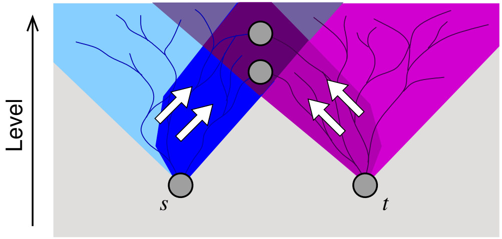
- **Phase 1:** Bidirectional **upward** search
  - Forward: interval search
  - Backward: interval search
- **Phase 2:** Bidirectional **upward** search
  - Forward: approximate profile search
  - Backward: approximate profile search
- **Phase 3:** Unpacking and profile search



Travel Time Profile Query with ATCHs

Using Preceeding Interval Search: Space Efficient

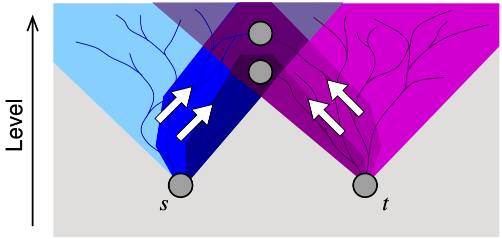
- **Phase 1:** Bidirectional **upward** search
  - Forward: interval search
  - Backward: interval search
- **Phase 2:** Bidirectional **upward** search
  - Forward: approximate profile search
  - Backward: approximate profile search
- **Phase 3:** Unpacking and profile search



Travel Time Profile Query with ATCHs

Using Preceeding Interval Search: Space Efficient

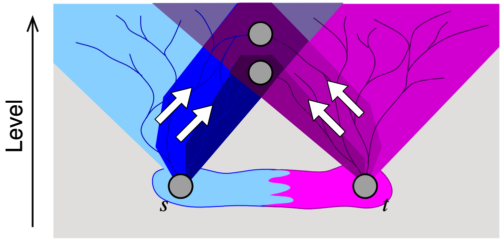
- **Phase 1:** Bidirectional **upward** search
  - Forward: interval search
  - Backward: interval search
- **Phase 2:** Bidirectional **upward** search
  - Forward: approximate profile search
  - Backward: approximate profile search
- **Phase 3:** Unpacking and profile search



Travel Time Profile Query with ATCHs

Using Preceeding Interval Search: Space Efficient

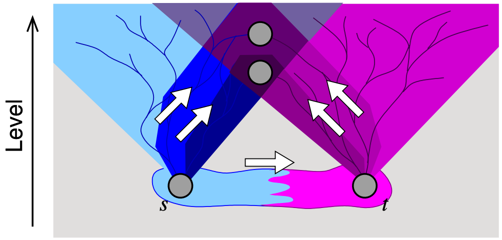
- **Phase 1:** Bidirectional **upward** search
  - Forward: interval search
  - Backward: interval search
- **Phase 2:** Bidirectional **upward** search
  - Forward: approximate profile search
  - Backward: approximate profile search
- **Phase 3:** Unpacking and profile search



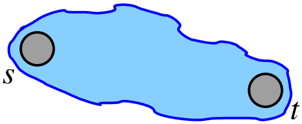
Travel Time Profile Query with ATCHs

Using Preceeding Interval Search: Space Efficient

- **Phase 1:** Bidirectional **upward** search
  - Forward: interval search
  - Backward: interval search
- **Phase 2:** Bidirectional **upward** search
  - Forward: approximate profile search
  - Backward: approximate profile search
- **Phase 3:** Unpacking and profile search



Corridor Contraction

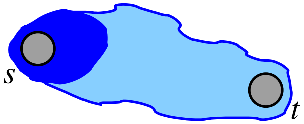


Profile search is very slow

- Edge relaxation processes...
- ...more and more and more points

So: Profile search is very expensive even in a corridor

Corridor Contraction

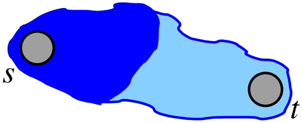


Profile search is very slow

- Edge relaxation processes...
- ...more and more and more points

So: Profile search is very expensive even in a corridor

Corridor Contraction

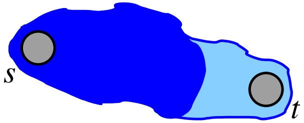


Profile search is very slow

- Edge relaxation processes...
- ...more and more and more points

So: Profile search is very expensive even in a corridor

Corridor Contraction

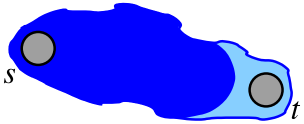


Profile search is very slow

- Edge relaxation processes...
- ...more and more and more points

So: Profile search is very expensive even in a corridor

Corridor Contraction

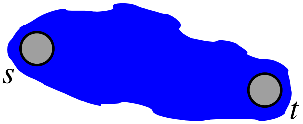


Profile search is very slow

- Edge relaxation processes...
- ...more and more and more points

So: Profile search is very expensive even in a corridor

Corridor Contraction

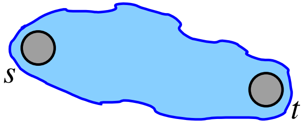


Profile search is very slow

- Edge relaxation processes...
- ...more and more and more points

So: Profile search is very expensive even in a corridor

Corridor Contraction

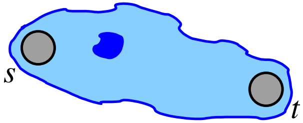


- **Contract a corridor**

```
While uncontracted nodes in corridor do
    Contract simplest node
    Remove that node from corridor
```

$\Rightarrow$ **From "‘quadratic"’ to "‘linear-logarithmic"’ running time**

Corridor Contraction

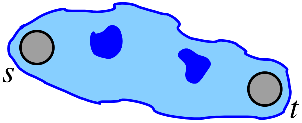


- **Contract a corridor**

```
While uncontracted nodes in corridor do
    Contract simplest node
    Remove that node from corridor
```

$\Rightarrow$ **From "‘quadratic"’ to "‘linear-logarithmic"’ running time**

Corridor Contraction

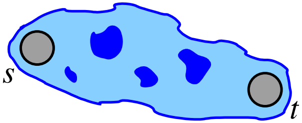


- **Contract** **a** **corridor**

```
While uncontracted nodes in corridor do
    Contract simplest node
    Remove that node from corridor
```

$\Rightarrow$ **From "‘quadratic"’ to "‘linear-logarithmic"’ running time**

Corridor Contraction

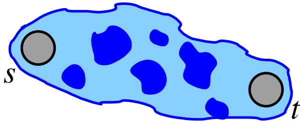


- **Contract a corridor**

```
While uncontracted nodes in corridor do
    Contract simplest node
    Remove that node from corridor
```

$\Rightarrow$ **From "‘quadratic"’ to "‘linear-logarithmic"’ running time**

Corridor Contraction

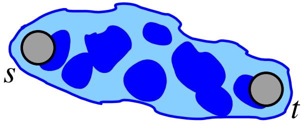


- **Contract** **a** **corridor**

```
While uncontracted nodes in corridor do
    Contract simplest node
    Remove that node from corridor
```

$\Rightarrow$ **From "‘quadratic"’ to "‘linear-logarithmic"’ running time**

Corridor Contraction

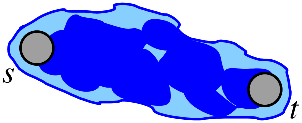


- **Contract a corridor**

```
While uncontracted nodes in corridor do
    Contract simplest node
    Remove that node from corridor
```

$\Rightarrow$ **From "‘quadratic"’ to "‘linear-logarithmic"’ running time**

Corridor Contraction

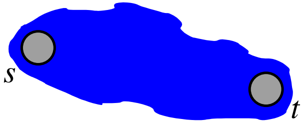


- **Contract a corridor**

```
While uncontracted nodes in corridor do
    Contract simplest node
    Remove that node from corridor
```

$\Rightarrow$ **From "‘quadratic"’ to "‘linear-logarithmic"’ running time**

Corridor Contraction



- **Contract a corridor**

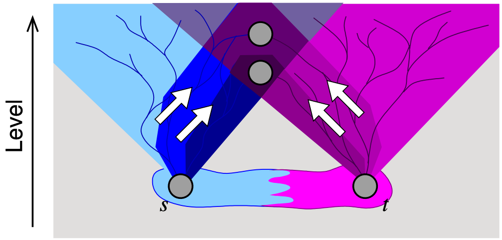
```
While uncontracted nodes in corridor do
    Contract simplest node
    Remove that node from corridor
```

$\Rightarrow$ **From "‘quadratic"’ to "‘linear-logarithmic"’ running time**

Travel Time Profile Query with ATCHs

With Corridor Constraction: Space Efficient and Very Fast

- **Phase 1:** Bidirectional **upward** search
  - Forward: interval search
  - Backward: interval search
- **Phase 2:** Bidirectional **upward** search
  - Forward: approximate profile search
  - Backward: approximate profile search
- **Phase 3:** Unpacking and corridor contraction search



Travel Time Profile Query with ATCHs and Corridor Contraction

Performance

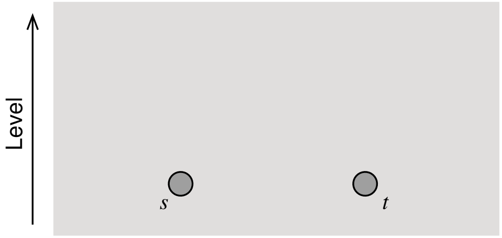
graph	method	ε [%]	space [B/n] ABS	space [B/n] OVH	query [ms]	error [%] MAX	error [%] AVG
Earliest Arrival Query							
Germany	TCH	–	994	899	1 112.04	0.00	0.00
	ATCH	1	239	144	32.75	0.00	0.00
	ATCH	∞	118	23	76.58	0.00	0.00
Europe	TCH	–	589	513	4 308.35	0.00	0.00
	ATCH	1	207	131	382.12	0.00	0.00
	ATCH	∞	99	23	–	–	–

- much less space than TCHs
- very fast
- still exact

Inexact Profile Queries on Inexact TCHs

Extremely Fast and Small Errors

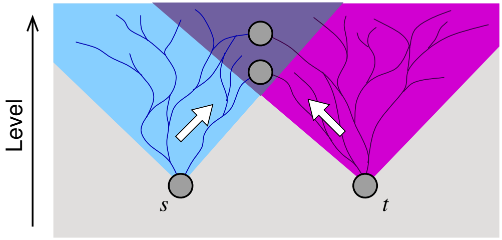
- **Phase 1:** Bidirectional **upward** search
  - Forward: interval search
  - Backward: interval search
- **Phase 2:** Bidirectional **upward** search
  - Forward: profile search
  - Backward: profile search



Inexact Profile Queries on Inexact TCHs

Extremely Fast and Small Errors

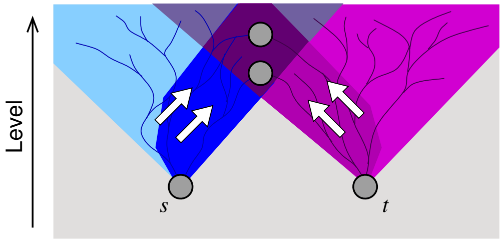
- **Phase 1:** Bidirectional **upward** search
  - Forward: interval search
  - Backward: interval search
- **Phase 2:** Bidirectional **upward** search
  - Forward: profile search
  - Backward: profile search



Inexact Profile Queries on Inexact TCHs

Extremely Fast and Small Errors

- **Phase 1:** Bidirectional **upward** search
  - Forward: interval search
  - Backward: interval search
- **Phase 2:** Bidirectional **upward** search
  - Forward: profile search
  - Backward: profile search



Inexact Profile Queries on Inexact TCHs

Performance

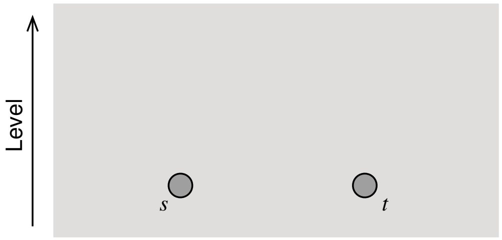
graph	method	ε [%]	space [B/n] ABS	space [B/n] OVH	query [ms]	error [%] MAX	error [%] AVG
Germany	TCH	–	994	899	1 112.04	0.00	0.00
	inex. TCH	1	214	119	2.94	1.03	0.27
	inex. TCH	10	118	18	2.49	9.69	3.84
Europe	TCH	–	589	513	4 308.35	0.00	0.00
	inex. TCH	1	193	117	105.73	1.27	0.20
	inex. TCH	10	143	67	36.94	14.65	2.85

- much less space than TCHs
- extremly fast
- only small error

Earliest Arrival Query with inexact TCHs

Very Fast, Simple, Small Error

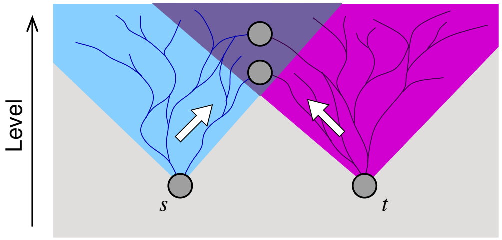
- **Phase 1:** Bidirectional **upward** search
  - Forward: interval search
  - Backward: interval search
- **Phase 2:** Forward/upward time-depdendent Dijkstra
- **Phase 3:** Forward/downward time-depdendent Dijkstra



Earliest Arrival Query with inexact TCHs

Very Fast, Simple, Small Error

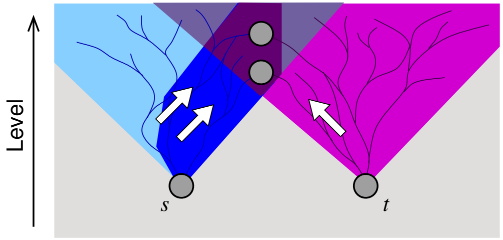
- **Phase 1:** Bidirectional **upward** search
  - Forward: interval search
  - Backward: interval search
- **Phase 2:** Forward/upward time-depdendent Dijkstra
- **Phase 3:** Forward/downward time-depdendent Dijkstra



Earliest Arrival Query with inexact TCHs

Very Fast, Simple, Small Error

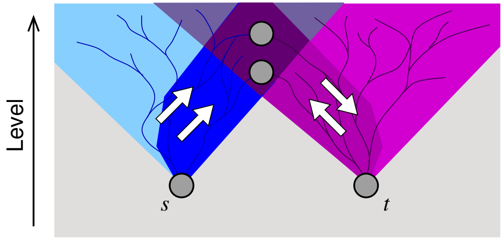
- **Phase 1:** Bidirectional **upward** search
  - Forward: interval search
  - Backward: interval search
- **Phase 2:** Forward/upward time-depdendent Dijkstra
- **Phase 3:** Forward/downward time-depdendent Dijkstra



Earliest Arrival Query with inexact TCHs

Very Fast, Simple, Small Error

- **Phase 1:** Bidirectional **upward** search
  - Forward: interval search
  - Backward: interval search
- **Phase 2:** Forward/upward time-depdendent Dijkstra
- **Phase 3:** Forward/downward time-depdendent Dijkstra



Earliest Arrival Query with Inexact TCHs

Performance

graph	method	ε [%]	space [B/n] ABS	space [B/n] OVH	query [ms]	query SPD	error [%] MAX	error [%] AVG
Germany	TCH	–	994	899	0.72	1 440	0.00	0.00
	inex. TCH	1	239	144	0.72	1 440	1.01	0.27
	inex. TCH	10	118	23	1.03	1 006	9.75	3.84
Europe	TCH	–	589	513	1.89	1 807	0.00	0.00
	inex. TCH	1	207	131	2.85	1 199	1.46	0.20
	inex. TCH	10	99	23	2.68	1 275	15.34	2.85

Summary
Problems Solved

- Earliest arrival queries:
  - Exact: still fast, space efficient $\rightsquigarrow$ ATCHs
  - Inexact: simple, very fast, small error, space efficient $\rightsquigarrow$ inexact TCHs
- Travel time profile queries:
  - Exact: very fast, space efficient $\rightsquigarrow$ ATCHs
  - Inexact: extremely fast, simple, small error, space efficient $\rightsquigarrow$ inexact TCHs

All achieved by carefully using approximation.

Future Work

- generalized time-depdendent objective functions
- better comression of travel time functions
- mobile time-dependent route planning
- compression of routes

Karlsruhe Institute of Technology

Thanks for your attention.

Any question?