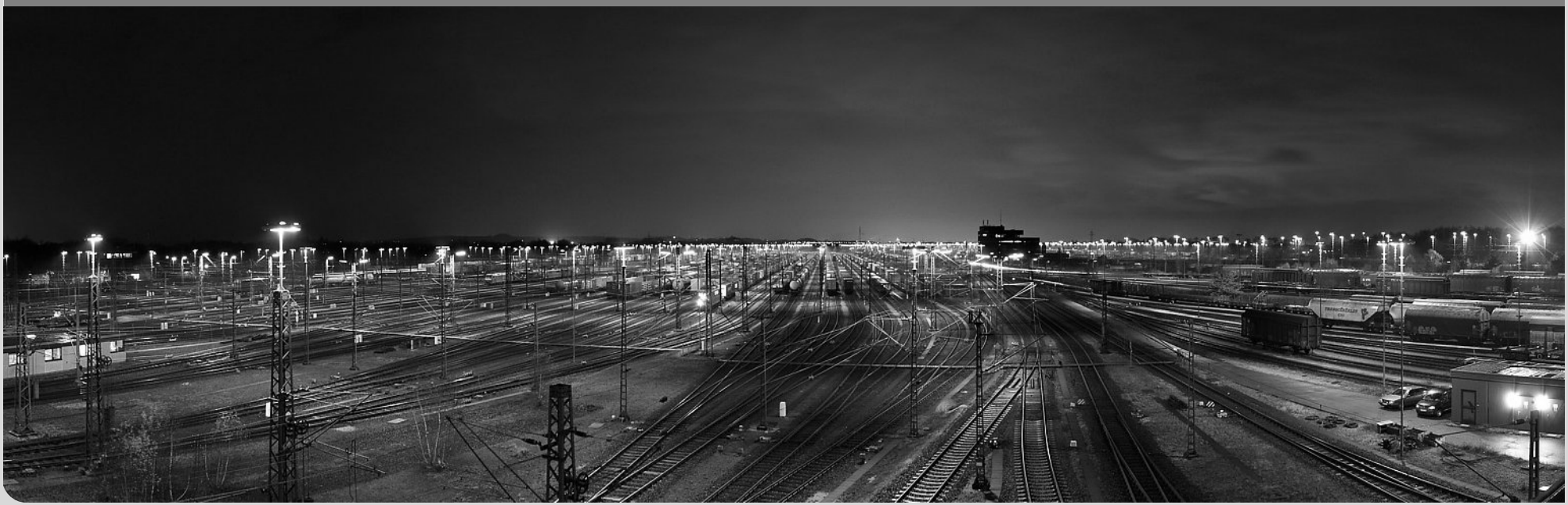
Karlsruher Institut für Technologie

Algorithmen für Routenplanung

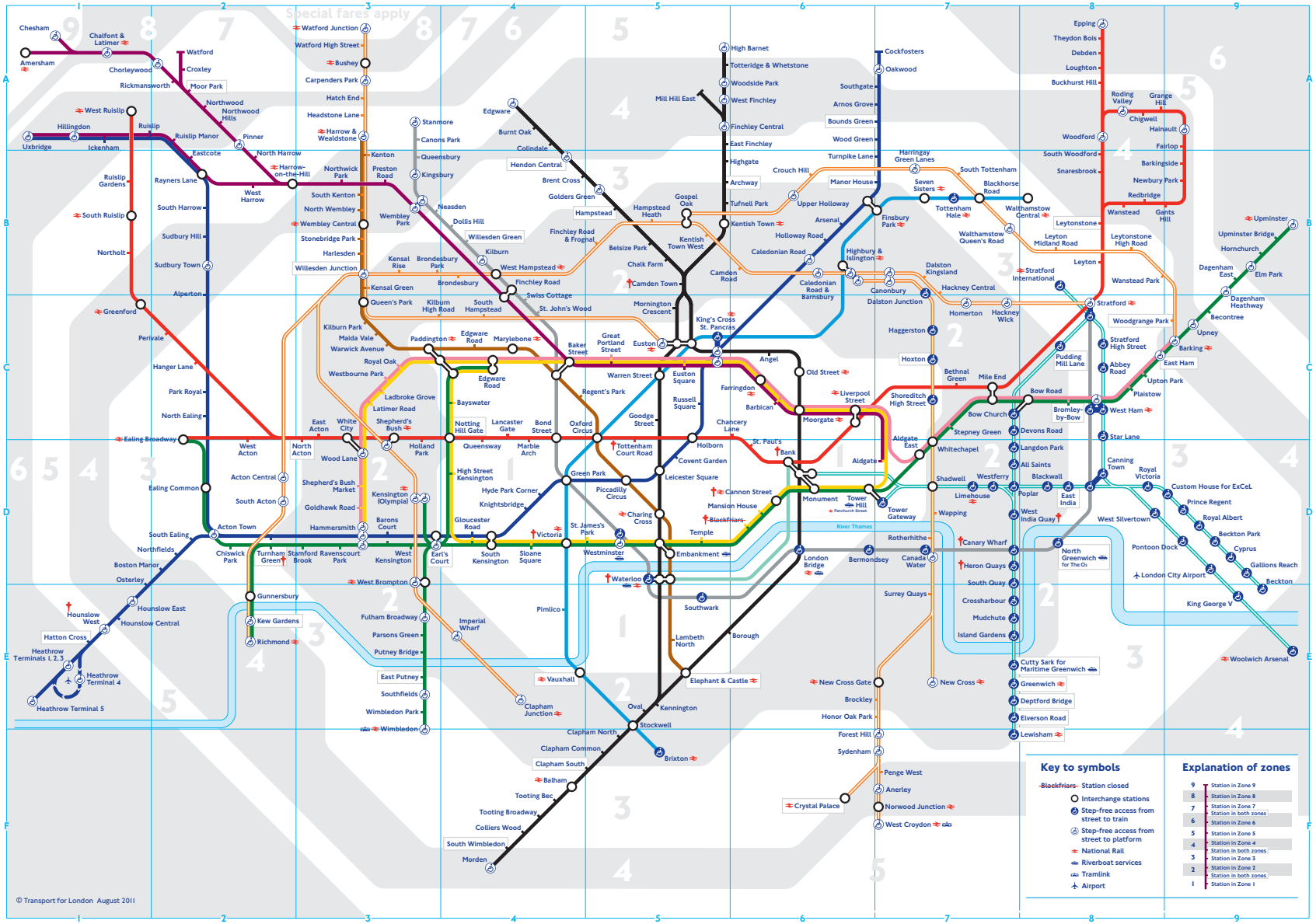
17. Vorlesung, Sommersemester 2017

Tobias Zündorf | 17. Juli 2017

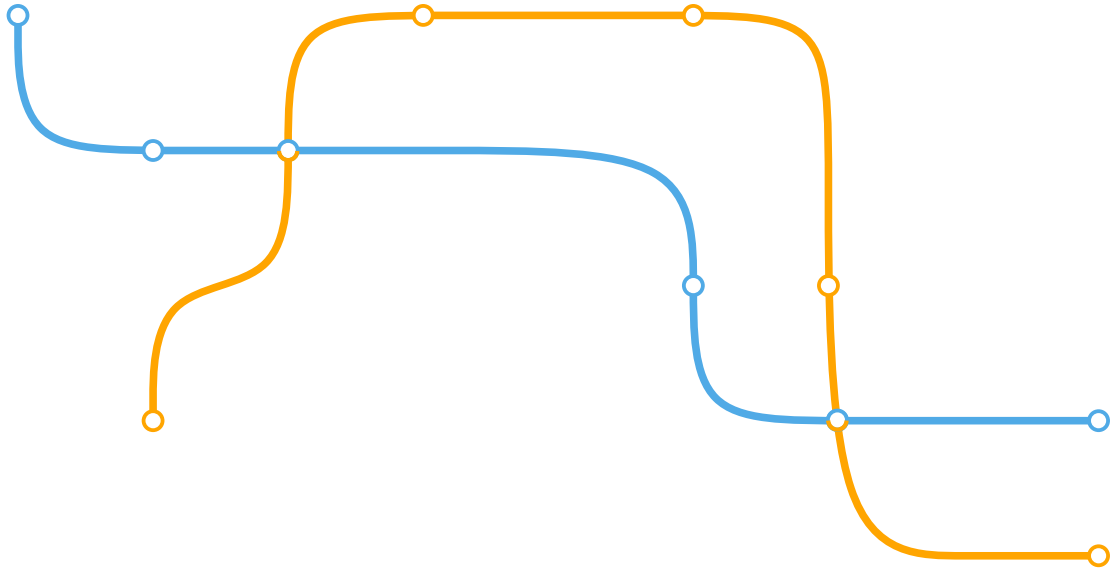
Institut für Theoretische Informatik · Algorithmik · Prof. Dr. Dorothea Wagner



Fahrplanauskunft

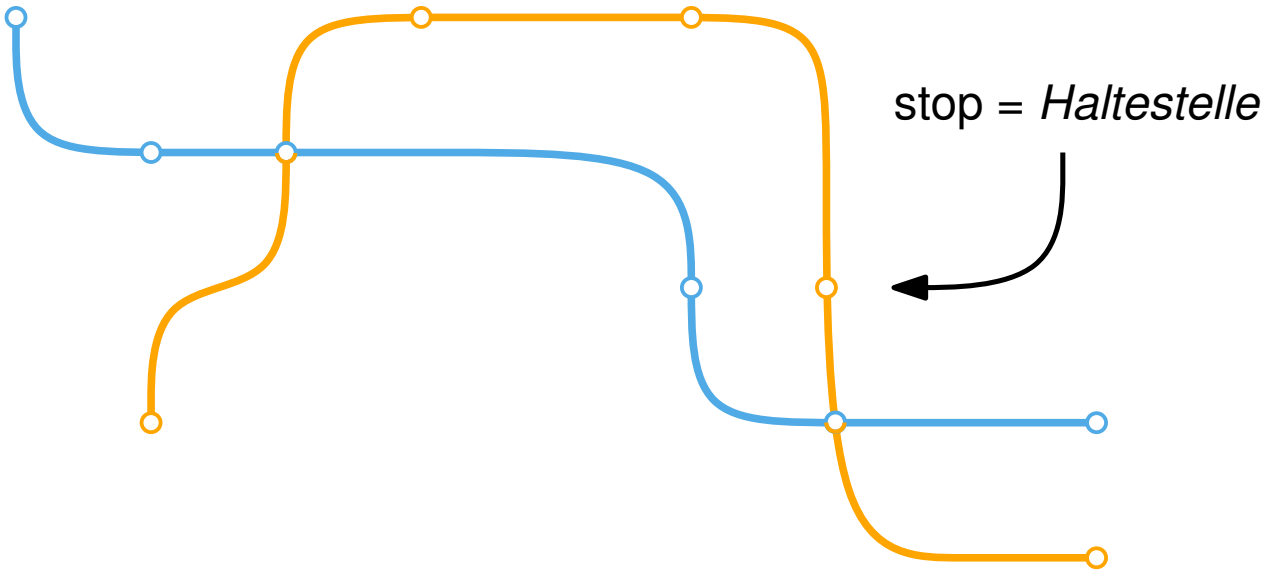


Wdh. Begriffe

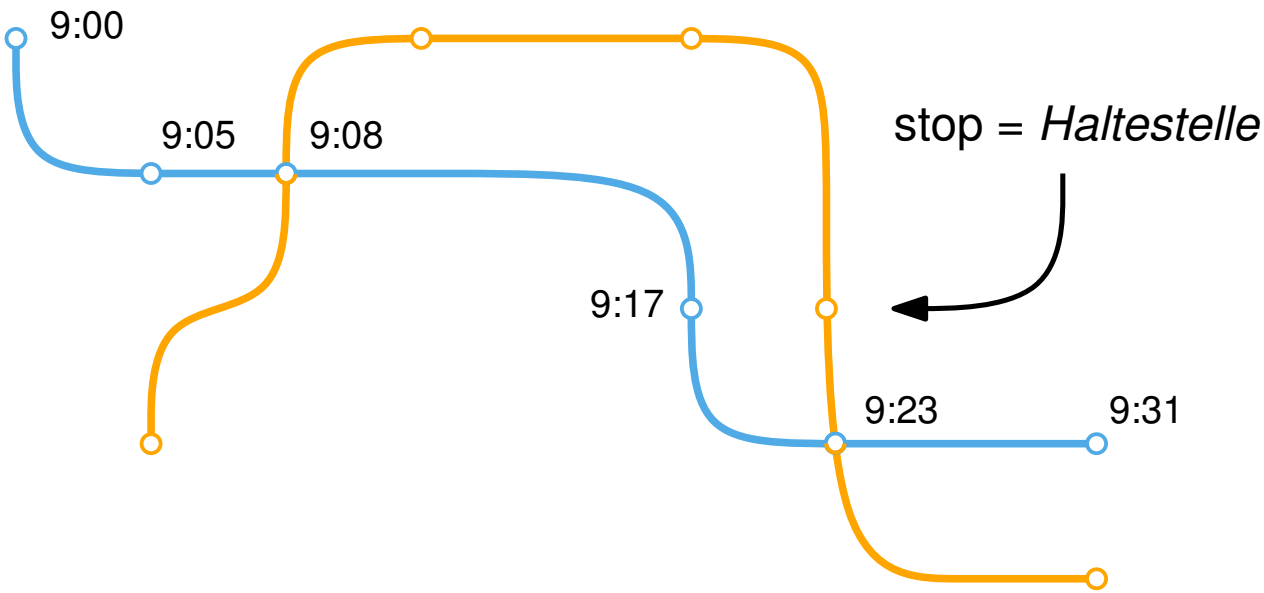


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Wdh. Begriffe



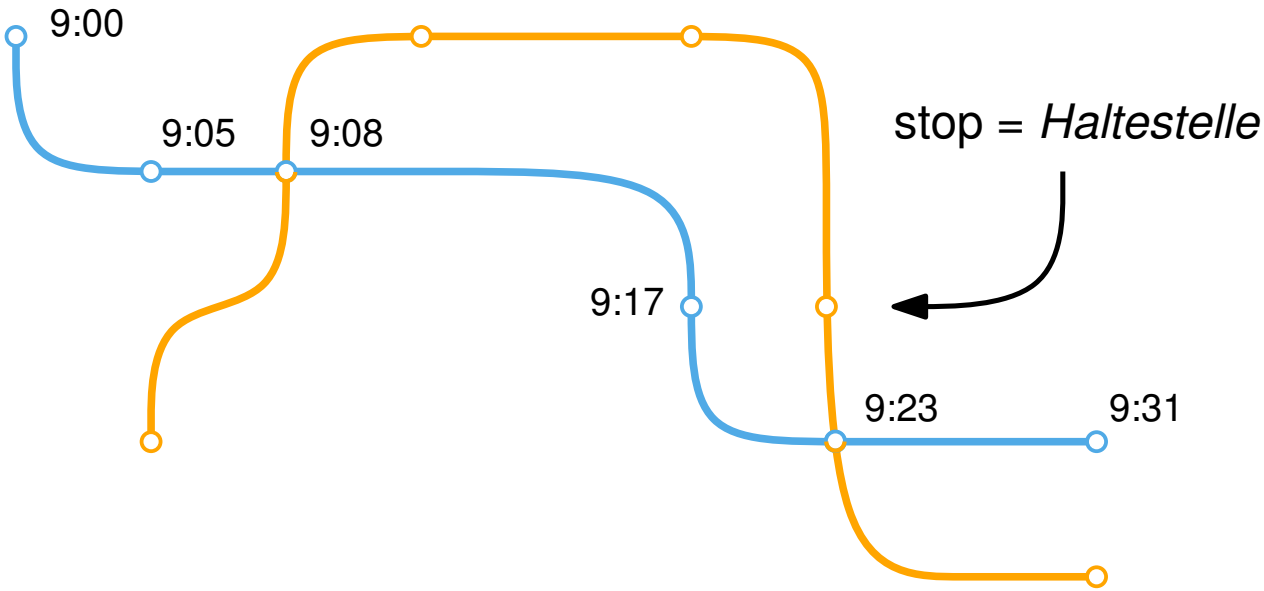
Wdh. Begriffe



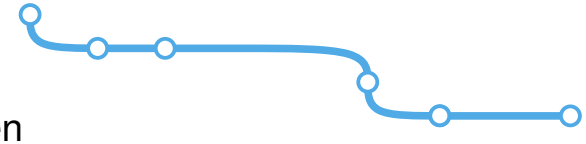
Wdh. Begriffe

Ankunftszeiten

trip t_1	9:00	9:05	9:08	9:17	9:23	9:31

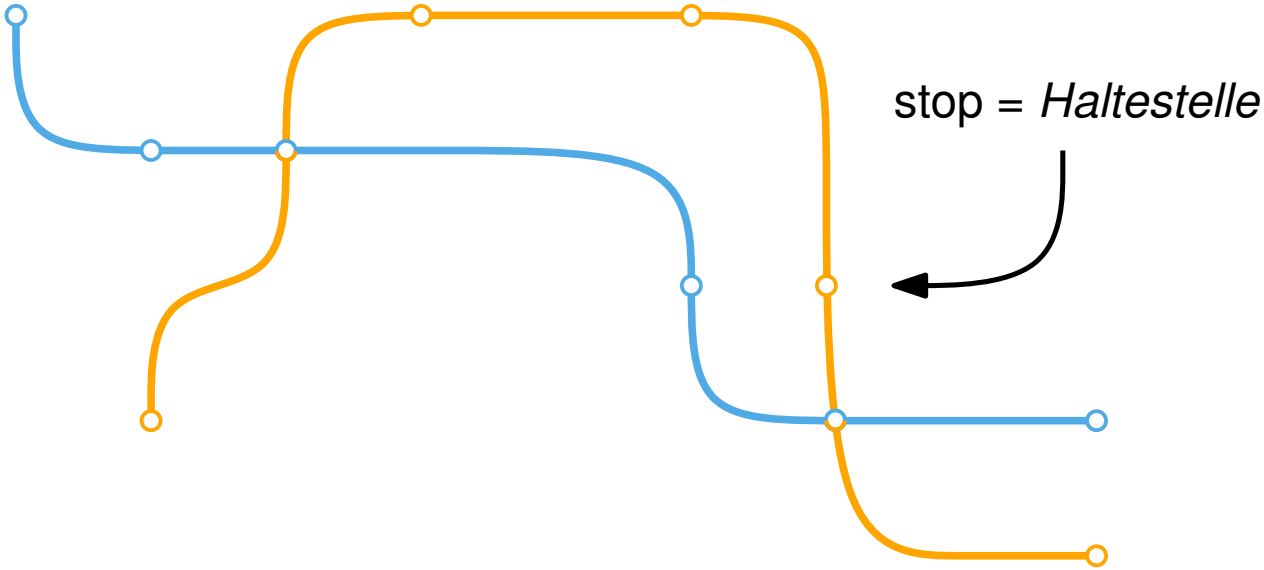


Wdh. Begriffe

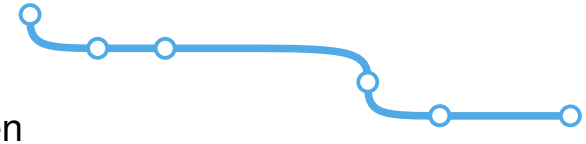


Ankunftszeiten

trip t_1	9:00	9:05	9:08	9:17	9:23	9:31

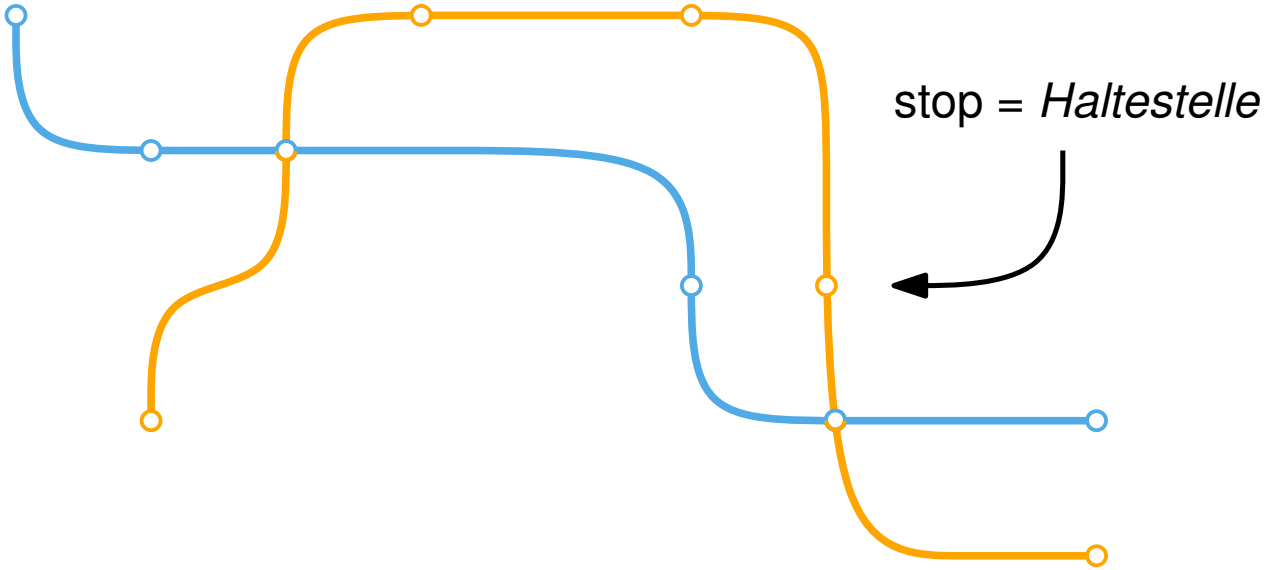


Wdh. Begriffe

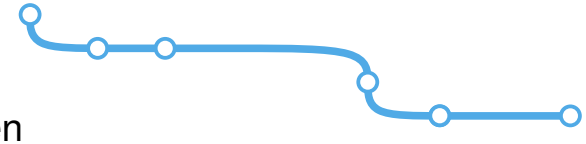


Ankunftszeiten

trip t_1	9:00	9:05	9:08	9:17	9:23	9:31
trip t_2	10:00	10:05	10:08	10:17	10:23	10:31

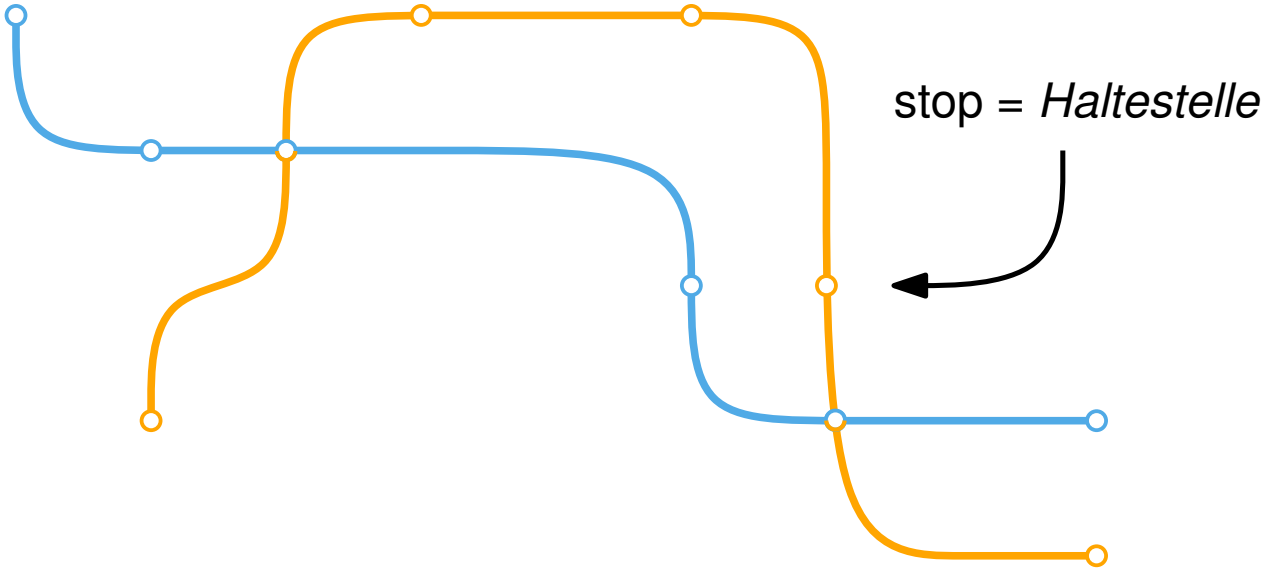


Wdh. Begriffe

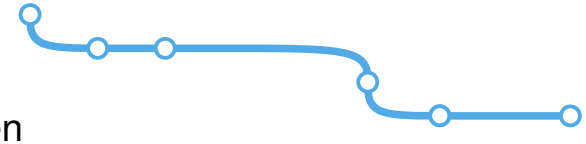


Ankunftszeiten

trip t_1	9:00	9:05	9:08	9:17	9:23	9:31
trip t_2	10:00	10:05	10:08	10:17	10:23	10:31
trip t_3	10:30	10:35	10:38	-	-	10:53



Wdh. Begriffe

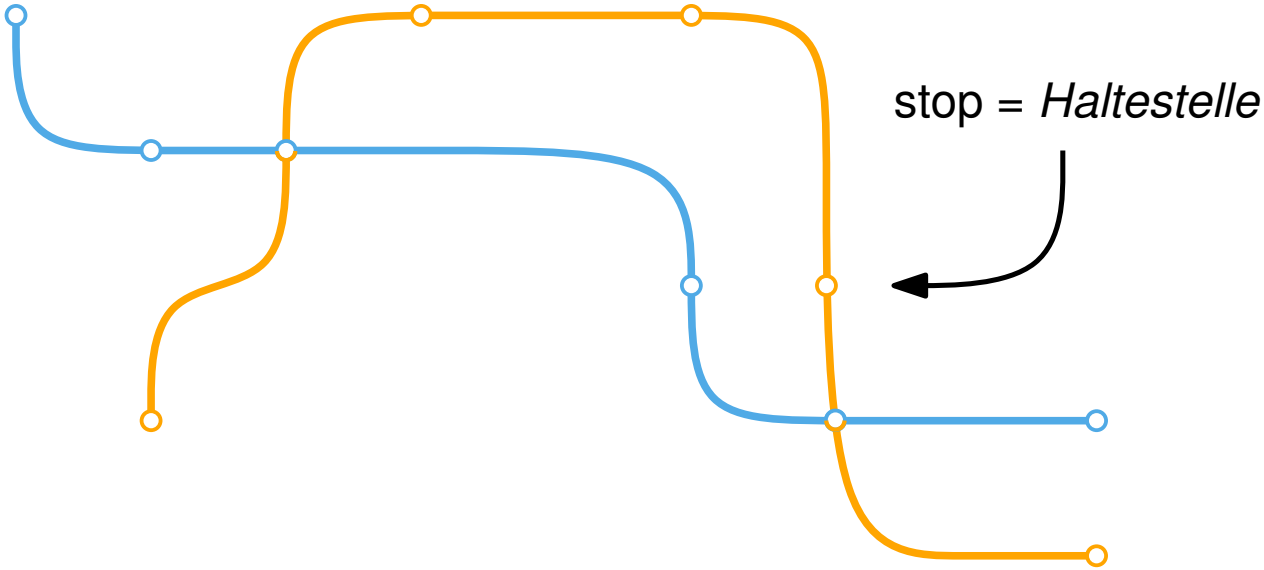


Ankunftszeiten

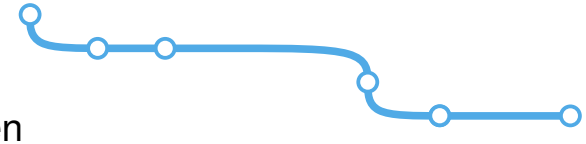
trip t_1	9:00	9:05	9:08	9:17	9:23	9:31
trip t_2	10:00	10:05	10:08	10:17	10:23	10:31
trip t_3	10:30	10:35	10:38	-	-	10:53

Route R_1

Route R_2



Wdh. Begriffe

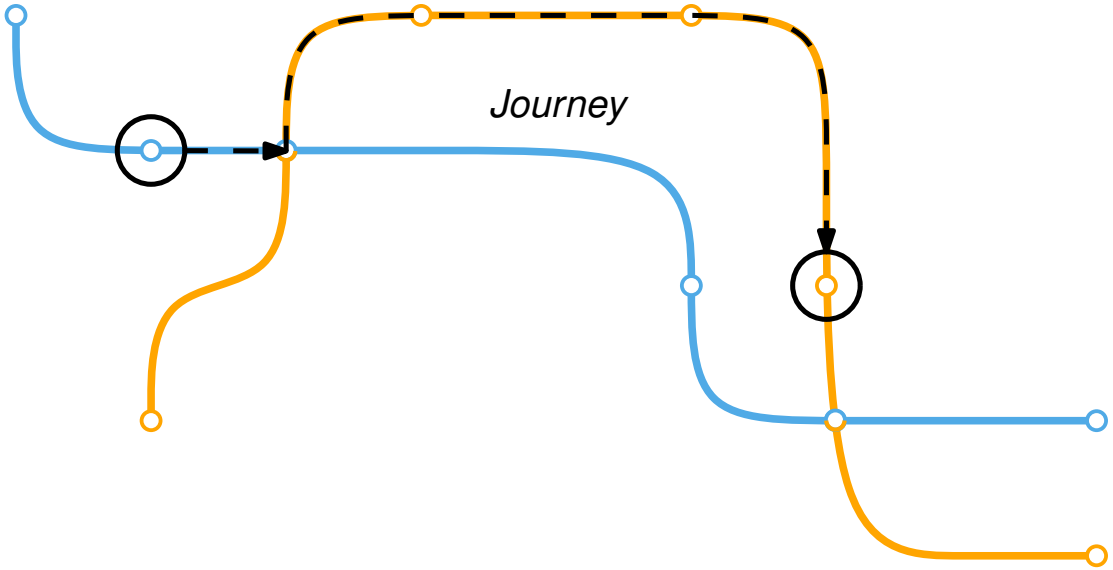


Ankunftszeiten

trip t_1	9:00	9:05	9:08	9:17	9:23	9:31
trip t_2	10:00	10:05	10:08	10:17	10:23	10:31
trip t_3	10:30	10:35	10:38	-	-	10:53

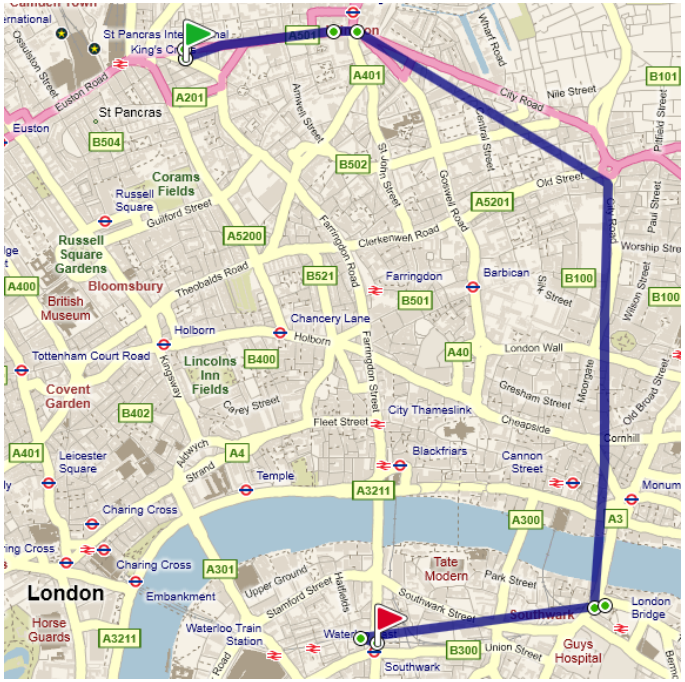
Route R_1

Route R_2

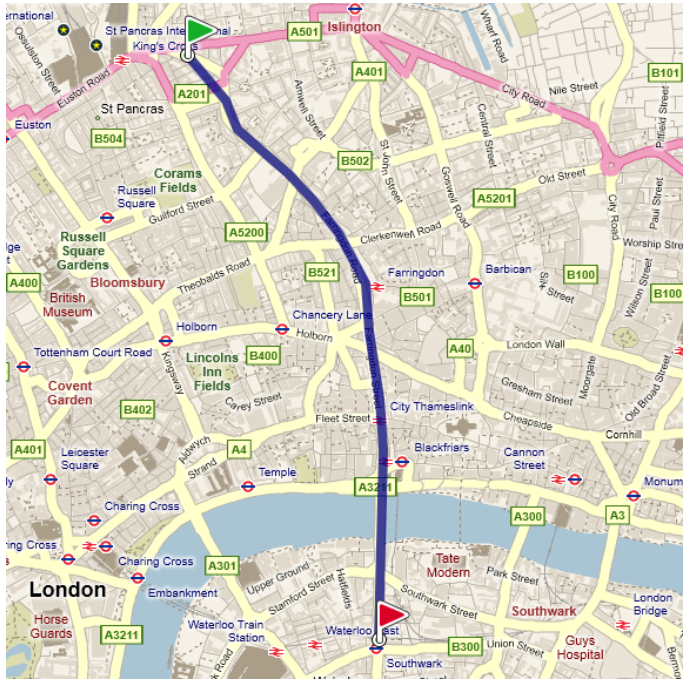


Wdh. Problemstellung

Gesucht: „Gute“ Routen für Ankunftszeit *und* Anzahl Umstiege.



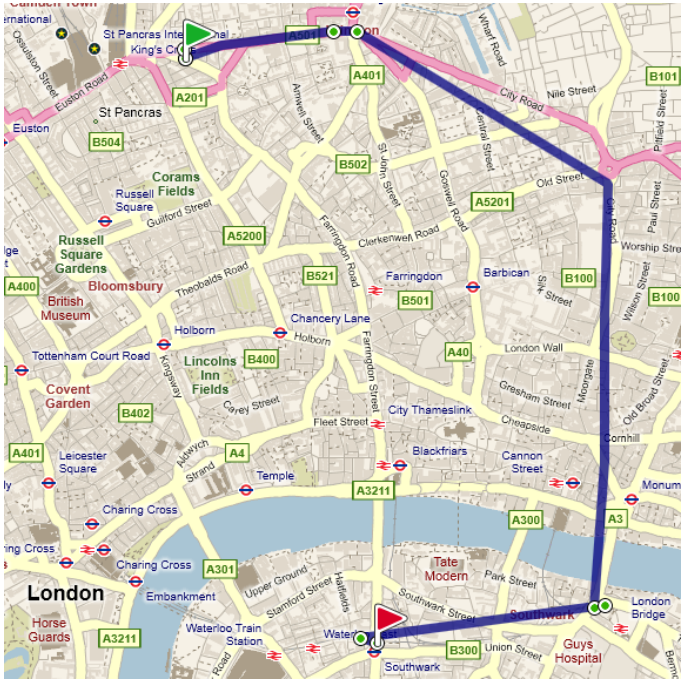
Ankunft 11:08 Uhr, 2 Umstiege



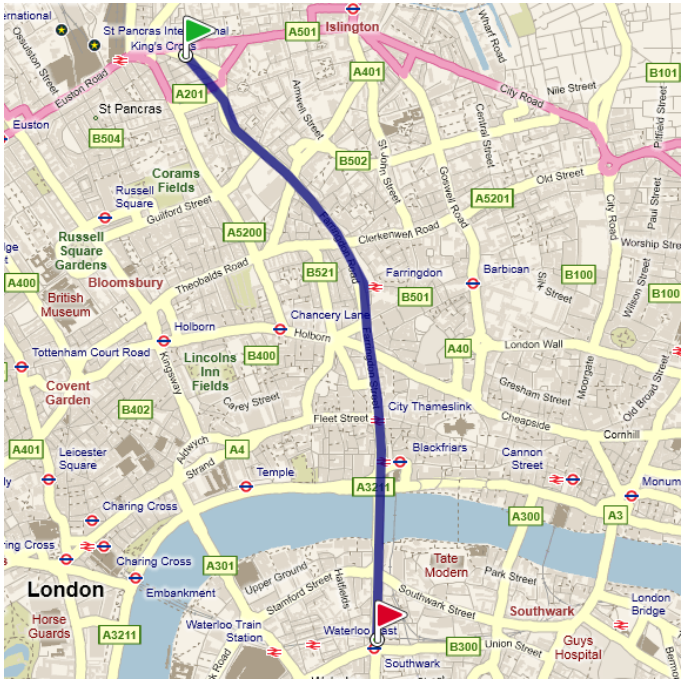
Ankunft 11:09 Uhr, 0 Umstiege

Wdh. Problemstellung

Gesucht: „Gute“ Routen für Ankunftszeit *und* Anzahl Umstiege.



Ankunft 11:08 Uhr, 2 Umstiege



Ankunft 11:09 Uhr, 0 Umstiege

Problem:
Dijkstra basierter Multi-Label-Correcting Ansatz zu langsam

Graph-Modelle?

Bis jetzt:

- Modelliere Fahrplan als gerichteten Graphen
- Zeitexpandiert vs zeitabhängig
- Verschiedene Varianten von Dijkstra's Algorithmus
- Earliest Arrival, Profil-, Multi-Criteria Suchen

Graph-Modelle?

Bis jetzt:

- Modelliere Fahrplan als gerichteten Graphen
- Zeitexpandiert vs zeitabhängig
- Verschiedene Varianten von Dijkstra's Algorithmus
- Earliest Arrival, Profil-, Multi-Criteria Suchen

Probleme

- Viele Knoten und Kanten
- Overhead von Priority Queue
- Wenig explizites Ausnutzen der Fahrplanstruktur
- Dynamische Szenarien erfordern Updates der Graph-Topologie
- Außerdem: Beschleunigungstechniken funktionieren nicht gut

Graph-Modelle?

Bis jetzt:

- Modelliere Fahrplan als gerichteten Graphen
- Zeitexpandiert vs zeitabhängig
- Verschiedene Varianten von Dijkstra's Algorithmus
- Earliest Arrival, Profil-, Multi-Criteria Suchen

Probleme

- Viele Knoten und Kanten
- Overhead von Priority Queue
- Wenig explizites Ausnutzen der Fahrplanstruktur
- Dynamische Szenarien erfordern Updates der Graph-Topologie
- Außerdem: Beschleunigungstechniken funktionieren nicht gut

Sind Graphen die beste Art Fahrpläne zu modellieren?

RAPTOR

Anforderungen:

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

RAPTOR

Anforderungen:

- Berechnen von Pareto-sets,
  mindestens Ankunftszeit und # Umstiege

RAPTOR

Anforderungen:

- Berechnen von Pareto-sets,
  mindestens Ankunftszeit und # Umstiege
- Nutzt die Struktur der Fahrpläne aus,
  benutzt Routen und Trips explizit?

RAPTOR

Anforderungen:

- Berechnen von Pareto-sets,
  mindestens Ankunftszeit und # Umstiege
- Nutzt die Struktur der Fahrpläne aus,
  benutzt Routen und Trips explizit?
- Funktioniert in dynamischen Szenarien,
  Verspätungen, Zugausfälle, Routenänderungen; keine Vorberechnung

RAPTOR

Anforderungen:

- Berechnen von Pareto-sets,
  mindestens Ankunftszeit und # Umstiege
- Nutzt die Struktur der Fahrpläne aus,
  benutzt Routen und Trips explizit?
- Funktioniert in dynamischen Szenarien,
  Verspätungen, Zugausfälle, Routenänderungen; keine Vorberechnung
- Kann auf zusätzliche Kriterien erweitert werden, ...
  z.B. Tarifzonen, Umstiegssicherheit, etc

RAPTOR

Anforderungen:

- Berechnen von Pareto-sets,
  mindestens Ankunftszeit und # Umstiege
- Nutzt die Struktur der Fahrpläne aus,
  benutzt Routen und Trips explizit?
- Funktioniert in dynamischen Szenarien,
  Verspätungen, Zugausfälle, Routenänderungen; keine Vorberechnung
- Kann auf zusätzliche Kriterien erweitert werden, ...
  z.B. Tarifzonen, Umstiegssicherheit, etc
- ... und ist hinreichend schnell
  für interaktive Szenarien

RAPTOR

Anforderungen:

- Berechnen von Pareto-sets,
  mindestens Ankunftszeit und # Umstiege
- Nutzt die Struktur der Fahrpläne aus,
  benutzt Routen und Trips explizit?
- Funktioniert in dynamischen Szenarien,
  Verspätungen, Zugausfälle, Routenänderungen; keine Vorberechnung
- Kann auf zusätzliche Kriterien erweitert werden, ...
  z.B. Tarifzonen, Umstiegssicherheit, etc
- ... und ist hinreichend schnell
  für interaktive Szenarien

RAPTOR: **R**ound-b**A**sed **P**ublic **T**ransit **O**ptimized **R**outer

Runden

Beobachtung: Wechseln zw. Trips führt immer zu einem Umstieg.

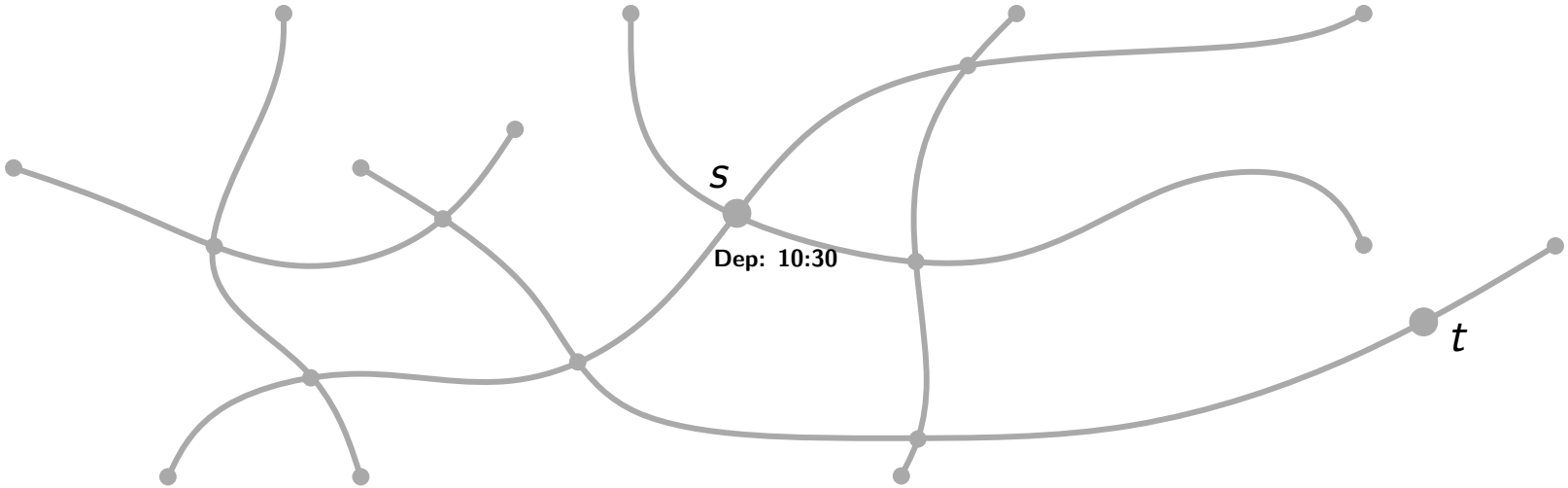
Runden

Beobachtung: Wechseln zw. Trips führt immer zu einem Umstieg.
Idee: Eine *Runde* für jeden genommenen Trip.

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Runden

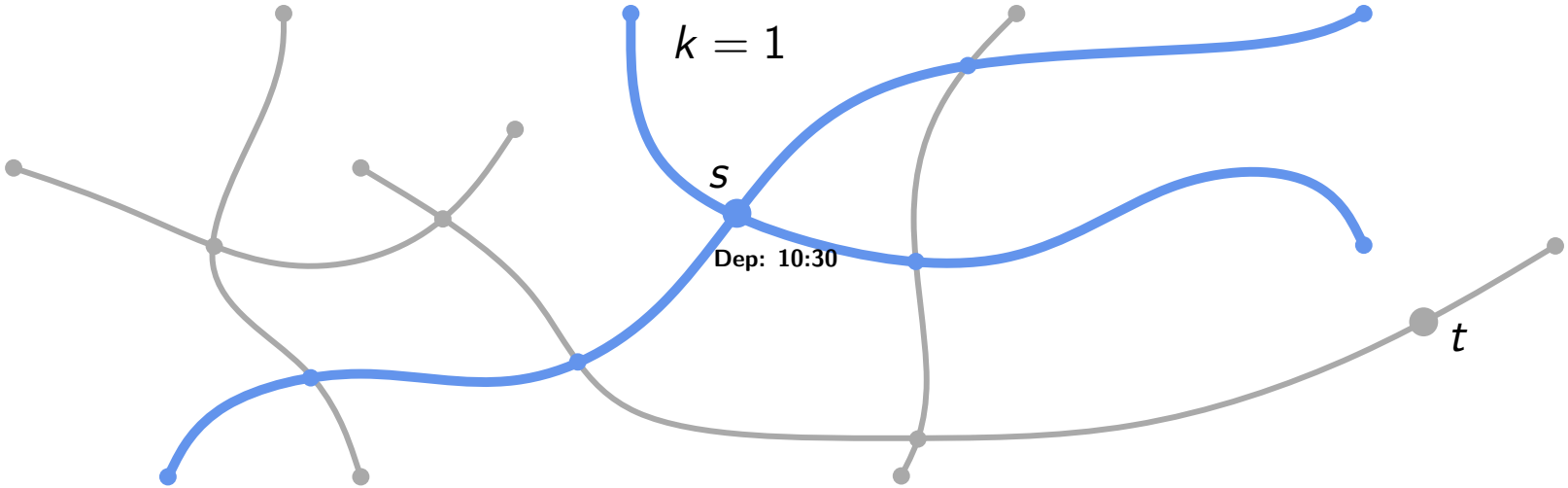
Beobachtung: Wechseln zw. Trips führt immer zu einem Umstieg.
Idee: Eine *Runde* für jeden genommenen Trip.



Ansatz: In Runde k werden Ankunftszeiten für k Trips berechnet.

Runden

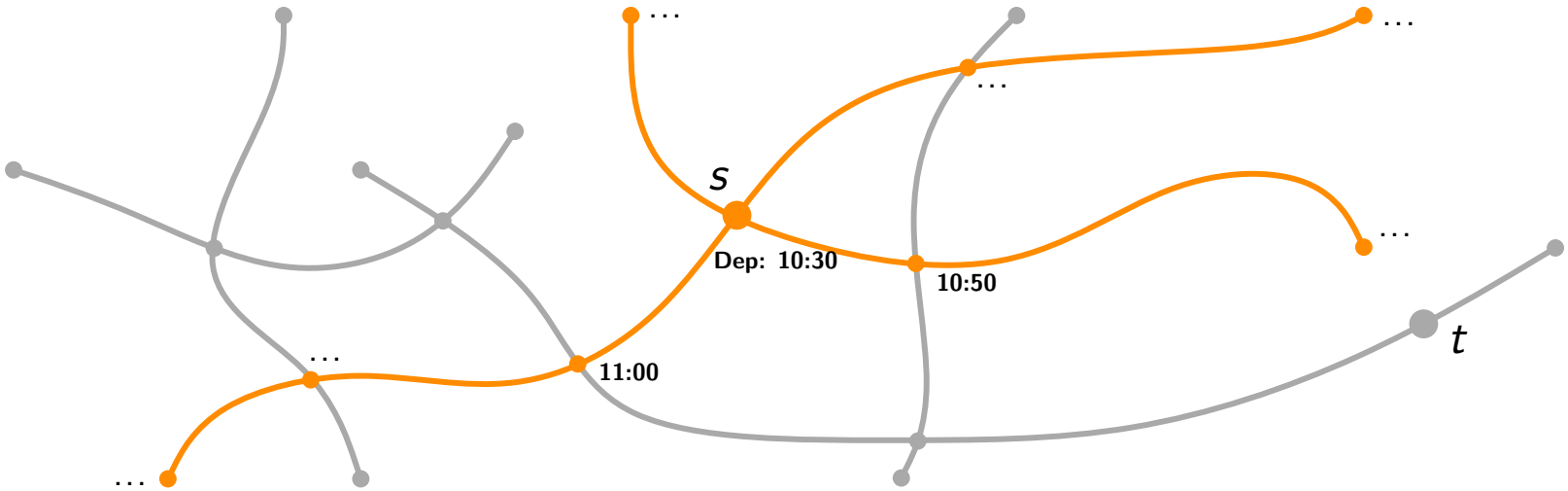
Beobachtung: Wechseln zw. Trips führt immer zu einem Umstieg.
Idee: Eine *Runde* für jeden genommenen Trip.



Ansatz: In Runde k werden Ankunftszeiten für k Trips berechnet.

Runden

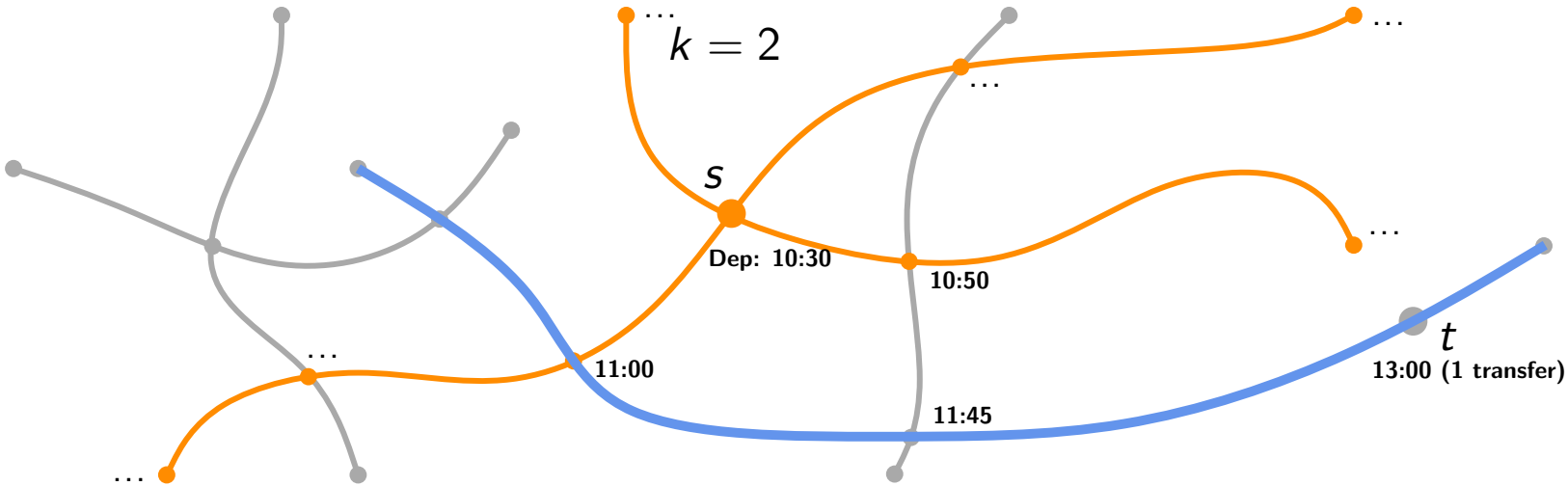
Beobachtung: Wechseln zw. Trips führt immer zu einem Umstieg.
Idee: Eine *Runde* für jeden genommenen Trip.



Ansatz: In Runde k werden Ankunftszeiten für k Trips berechnet.

Runden

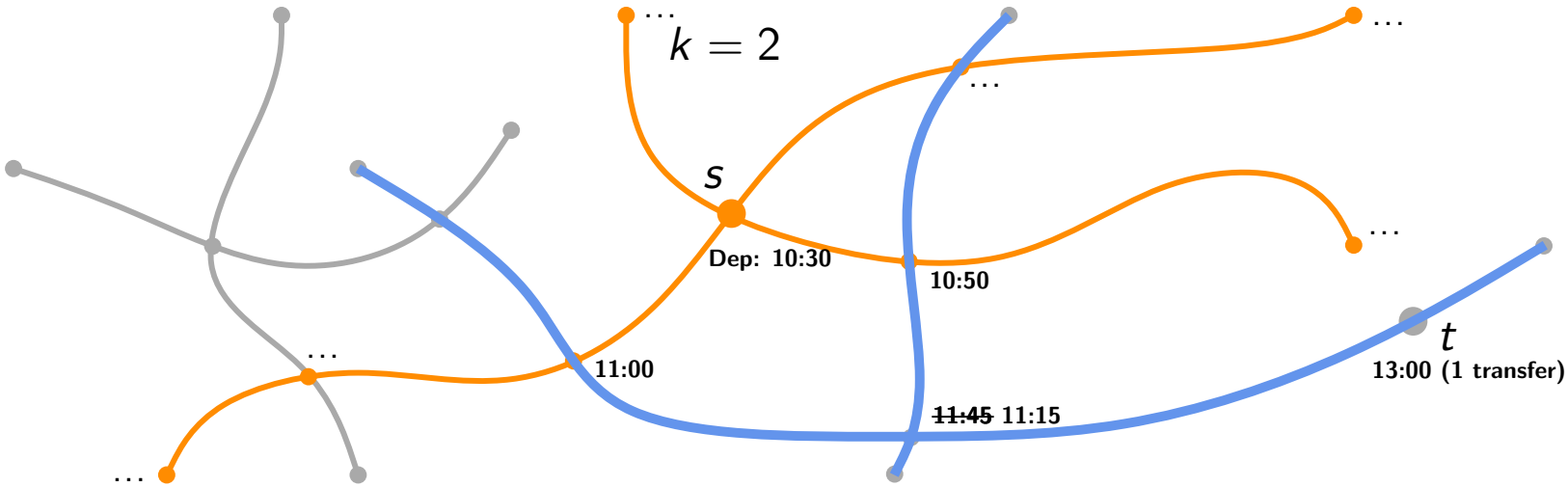
Beobachtung: Wechseln zw. Trips führt immer zu einem Umstieg.
Idee: Eine *Runde* für jeden genommenen Trip.



Ansatz: In Runde k werden Ankunftszeiten für k Trips berechnet.

Runden

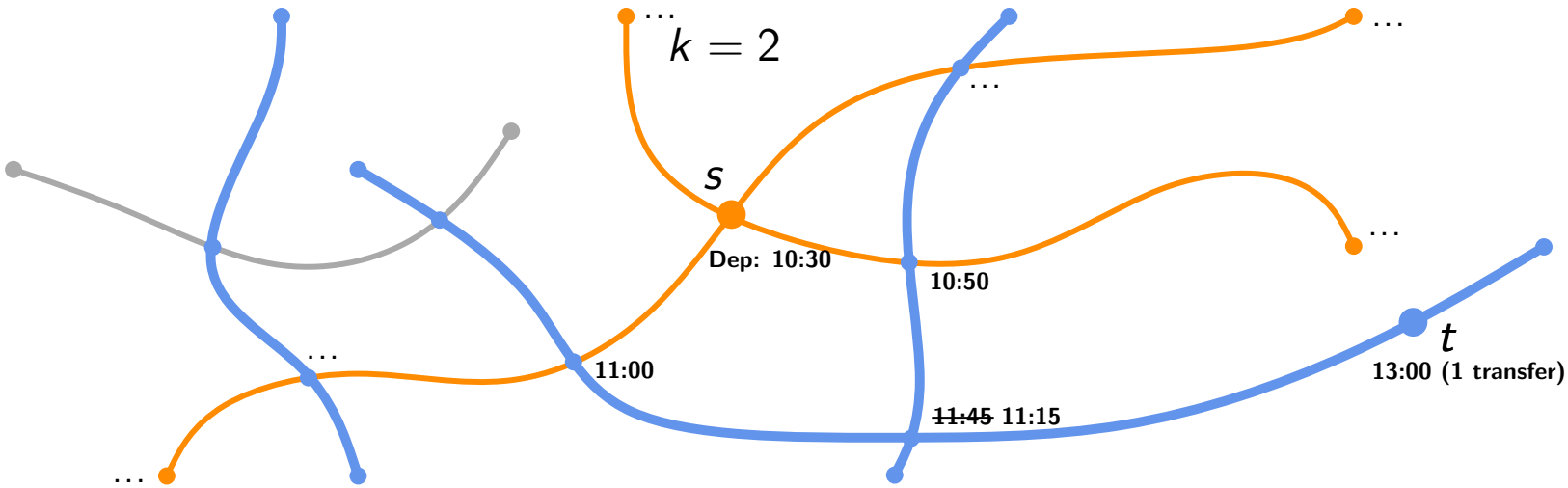
Beobachtung: Wechseln zw. Trips führt immer zu einem Umstieg.
Idee: Eine *Runde* für jeden genommenen Trip.



Ansatz: In Runde k werden Ankunftszeiten für k Trips berechnet.

Runden

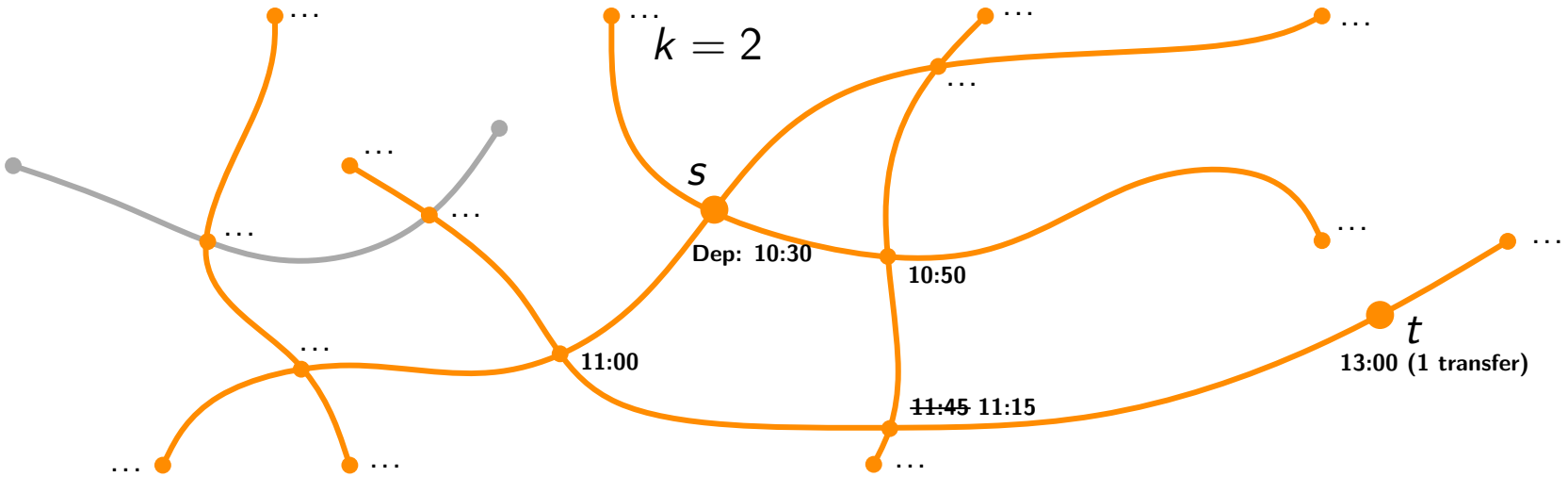
Beobachtung: Wechseln zw. Trips führt immer zu einem Umstieg.
Idee: Eine *Runde* für jeden genommenen Trip.



Ansatz: In Runde k werden Ankunftszeiten für k Trips berechnet.

Runden

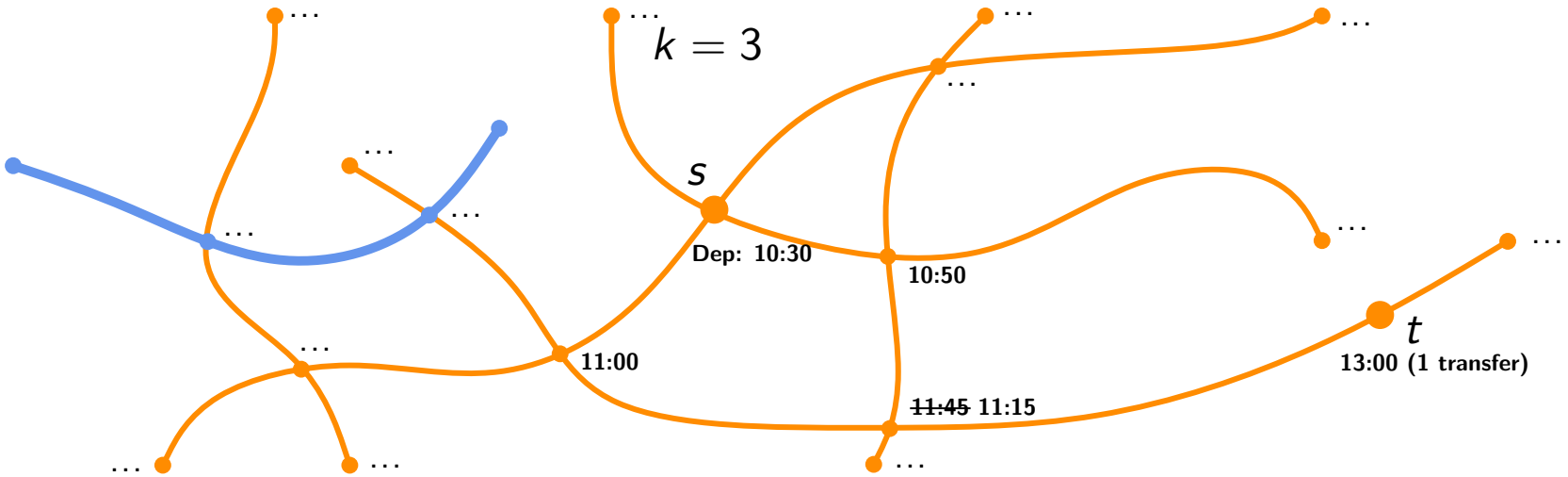
Beobachtung: Wechseln zw. Trips führt immer zu einem Umstieg.
Idee: Eine *Runde* für jeden genommenen Trip.



Ansatz: In Runde k werden Ankunftszeiten für k Trips berechnet.

Runden

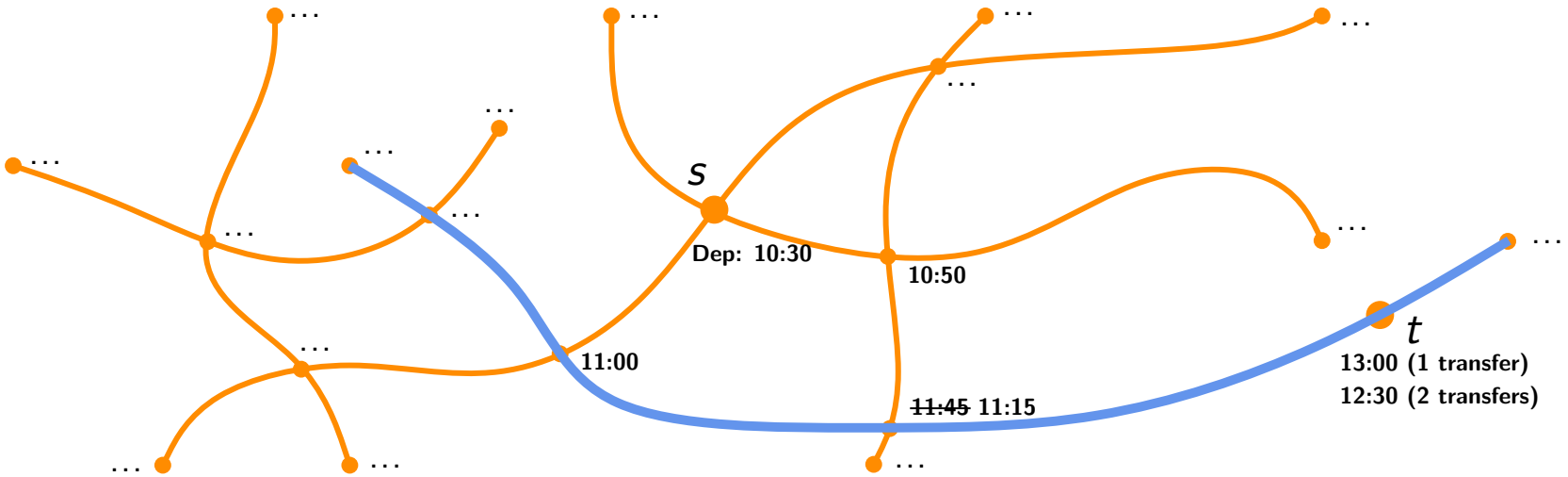
Beobachtung: Wechseln zw. Trips führt immer zu einem Umstieg.
Idee: Eine *Runde* für jeden genommenen Trip.



Ansatz: In Runde k werden Ankunftszeiten für k Trips berechnet.

Runden

Beobachtung: Wechseln zw. Trips führt immer zu einem Umstieg.
Idee: Eine *Runde* für jeden genommenen Trip.



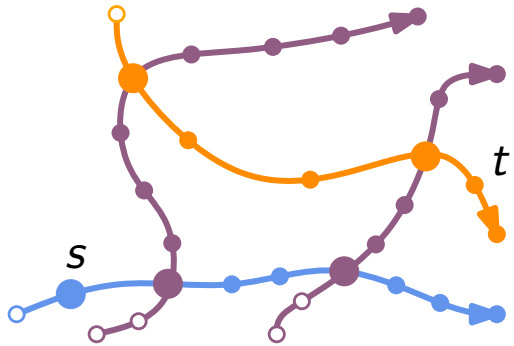
Ansatz: In Runde k werden Ankunftszeiten für k Trips berechnet.

Scanne jede **Route** höchstens einmal pro Runde.

RAPTOR: Übersicht

Für jede Runde $k \leftarrow 1, 2, \ldots$

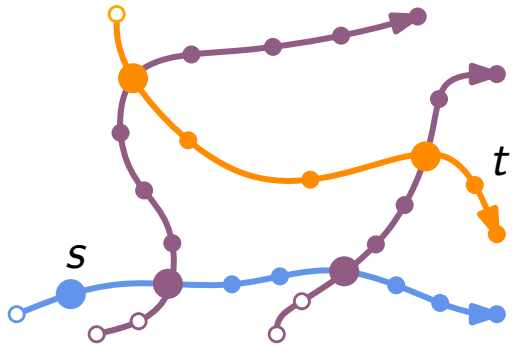
1. Scanne jede Route
2. Relaxiere Fußwege



RAPTOR: Übersicht

Für jede Runde $k \leftarrow 1, 2, \ldots$

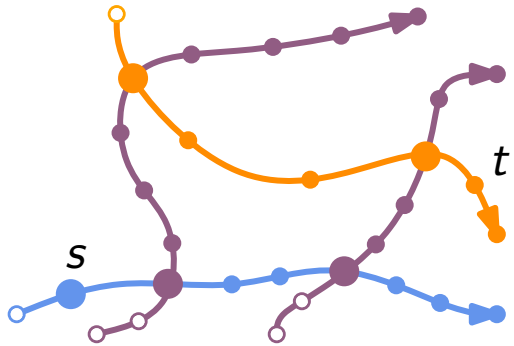
1. Scanne jede Route
2. Relaxiere Fußwege



RAPTOR: Übersicht

Für jede Runde $k \leftarrow 1, 2, \ldots$

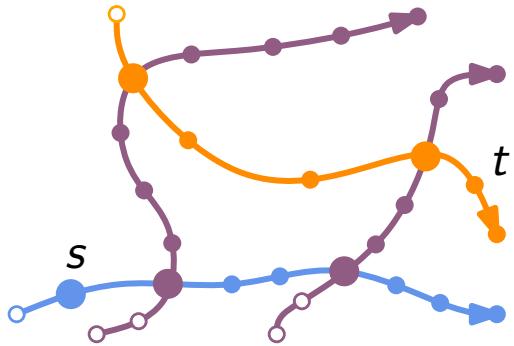
1. Scanne jede Route
2. Relaxiere Fußwege



RAPTOR: Übersicht

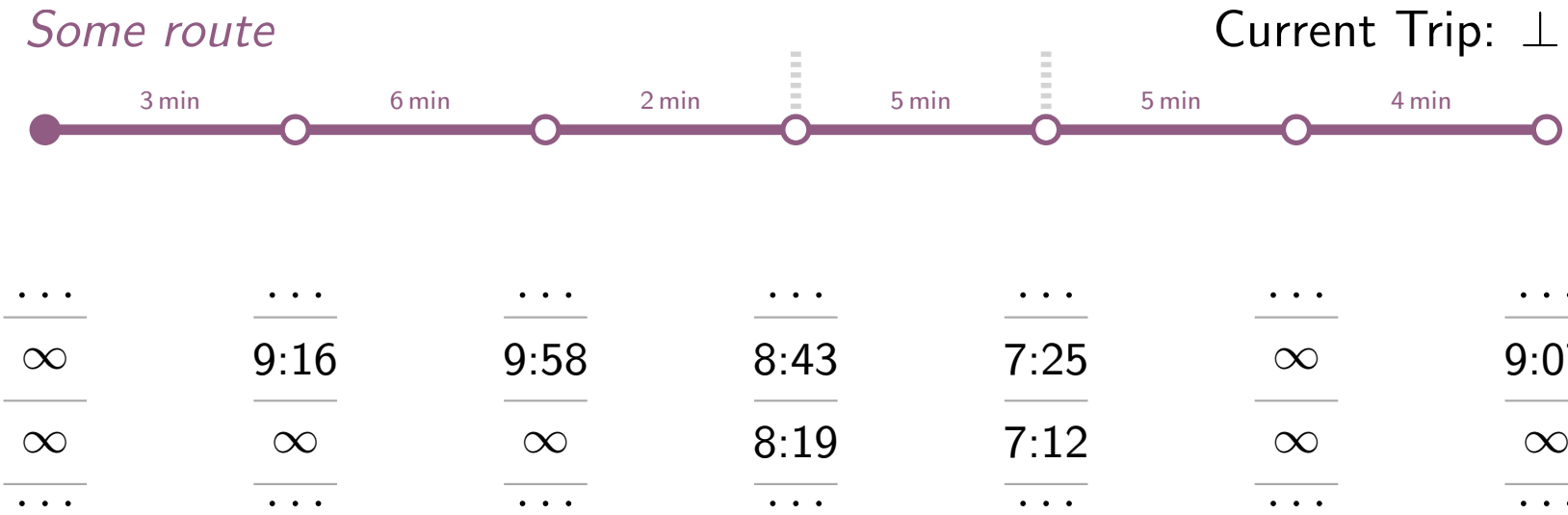
Für jede Runde $k \leftarrow 1, 2, \ldots$

1. Scanne jede Route
2. Relaxiere Fußwege



Terminiere, wenn ... ?

Scannen von Routen

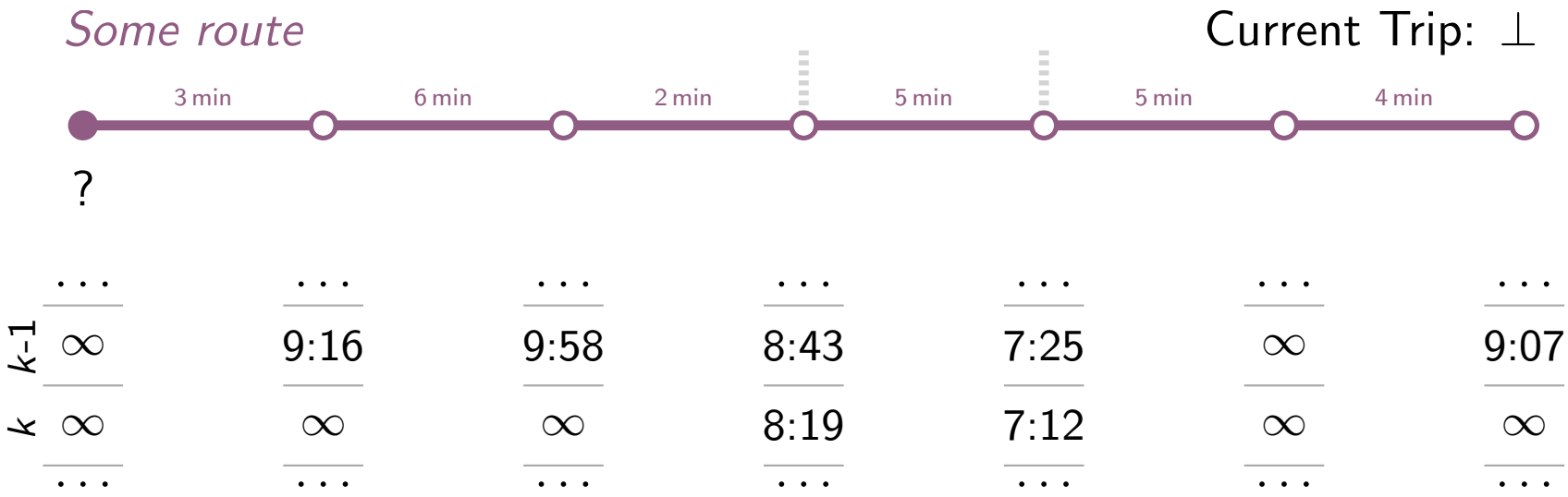


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Aktiver Trip entlang der Route wird stets verbessert.

In Runde k:

- Update Labels von Runde k mit Labels aus Runde $k-1$.

Scannen von Routen

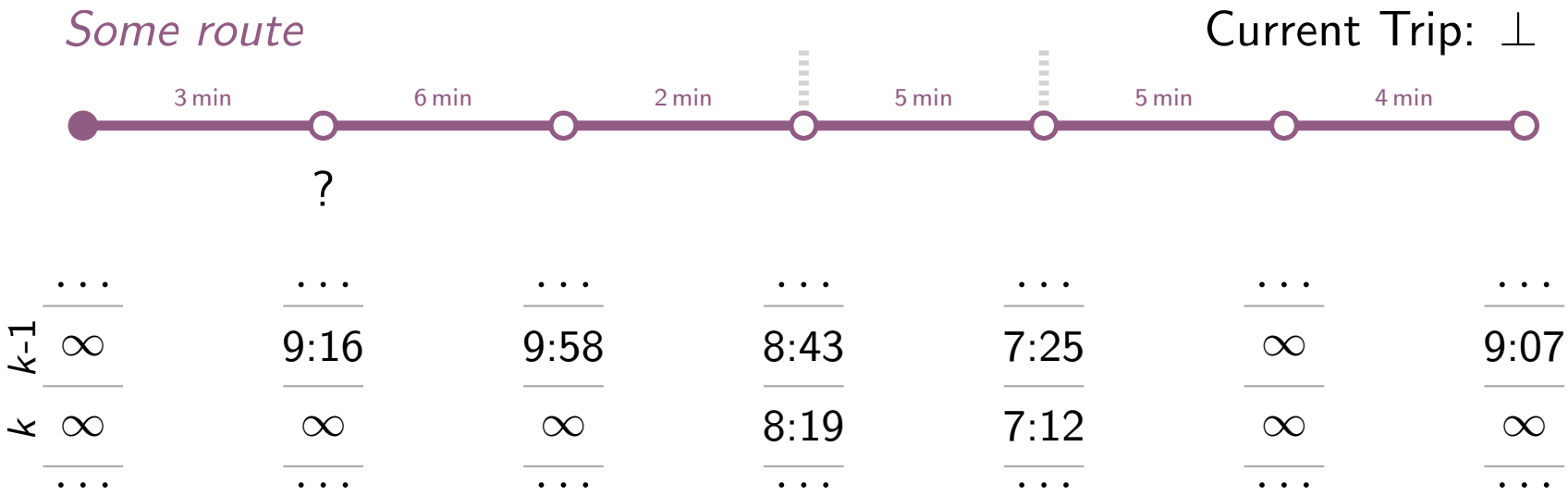


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Aktiver Trip entlang der Route wird stets verbessert.

In Runde k:

- Update Labels von Runde k mit Labels aus Runde $k-1$.

Scannen von Routen

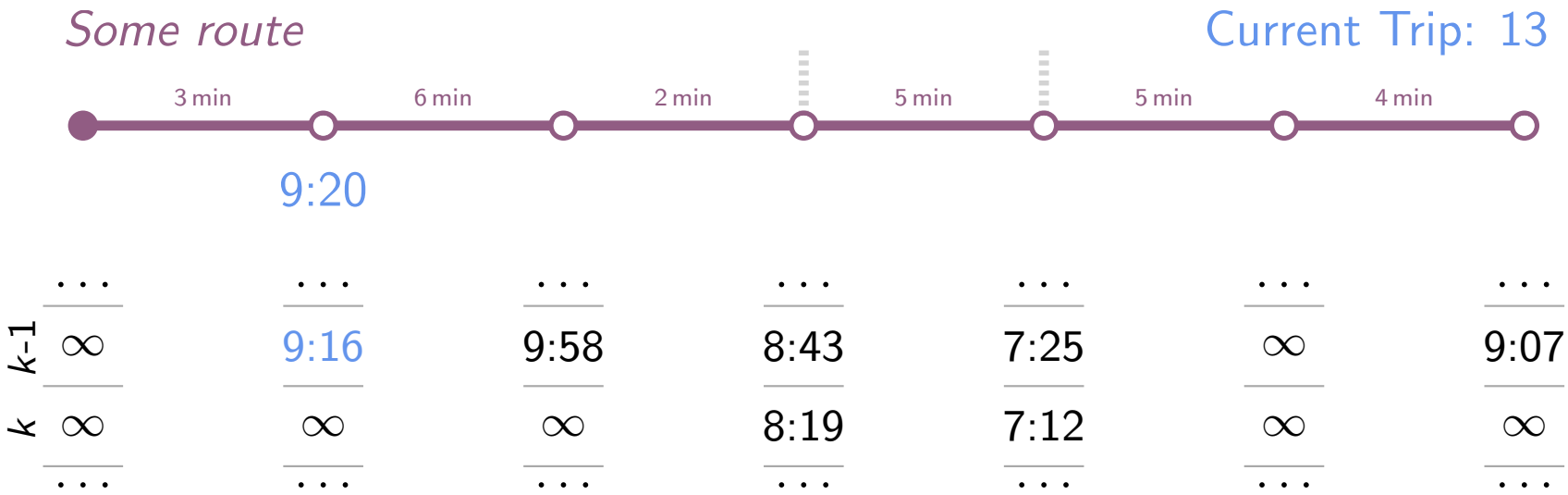


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Aktiver Trip entlang der Route wird stets verbessert.

In Runde k:

- Update Labels von Runde k mit Labels aus Runde $k-1$.

Scannen von Routen

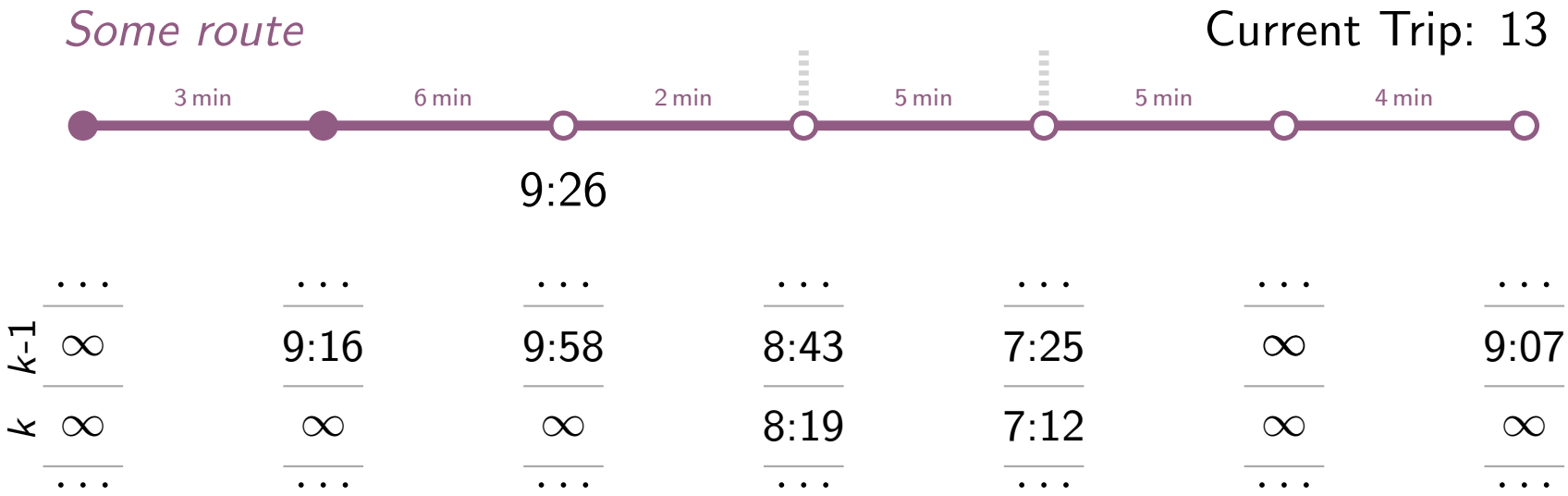


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Aktiver Trip entlang der Route wird stets verbessert.

In Runde k:

- Update Labels von Runde k mit Labels aus Runde $k-1$.

Scannen von Routen

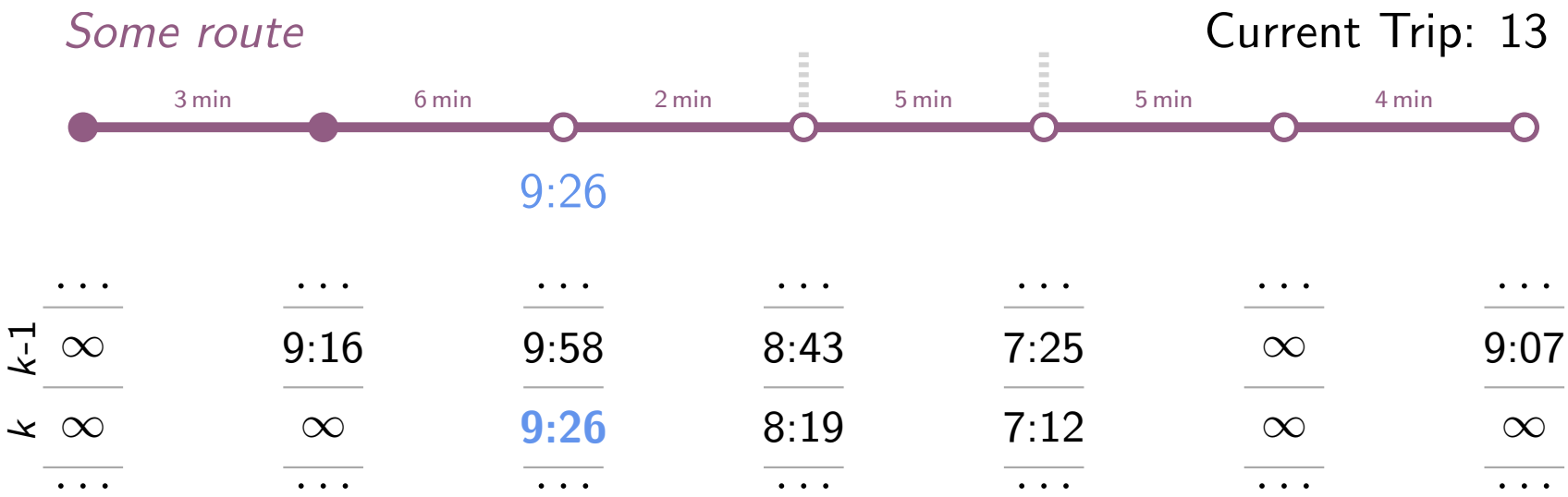


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Aktiver Trip entlang der Route wird stets verbessert.

In Runde k:

- Update Labels von Runde k mit Labels aus Runde $k-1$.

Scannen von Routen

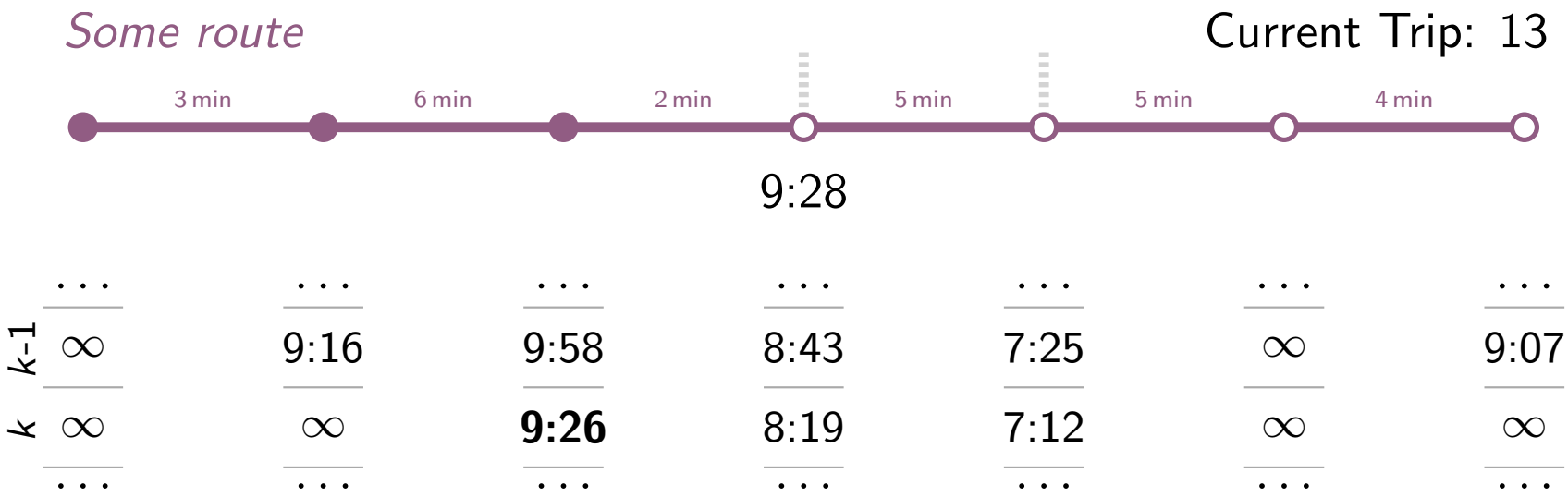


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Aktiver Trip entlang der Route wird stets verbessert.

In Runde k:

- Update Labels von Runde k mit Labels aus Runde $k-1$.

Scannen von Routen

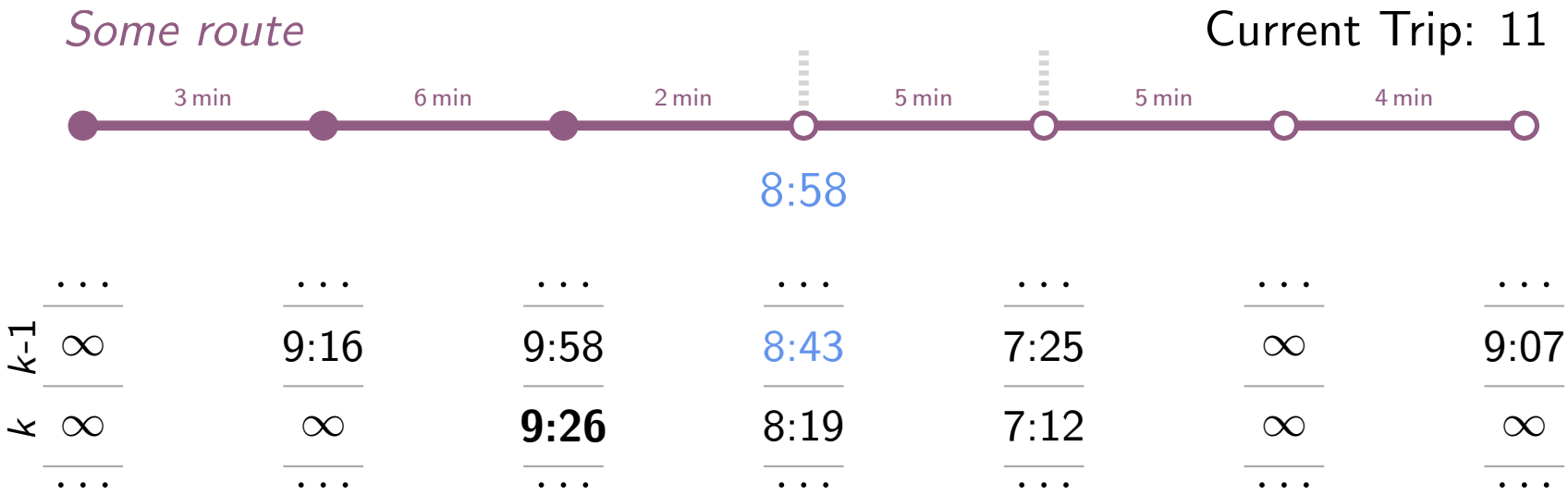


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Aktiver Trip entlang der Route wird stets verbessert.

In Runde k:

- Update Labels von Runde k mit Labels aus Runde $k-1$.

Scannen von Routen

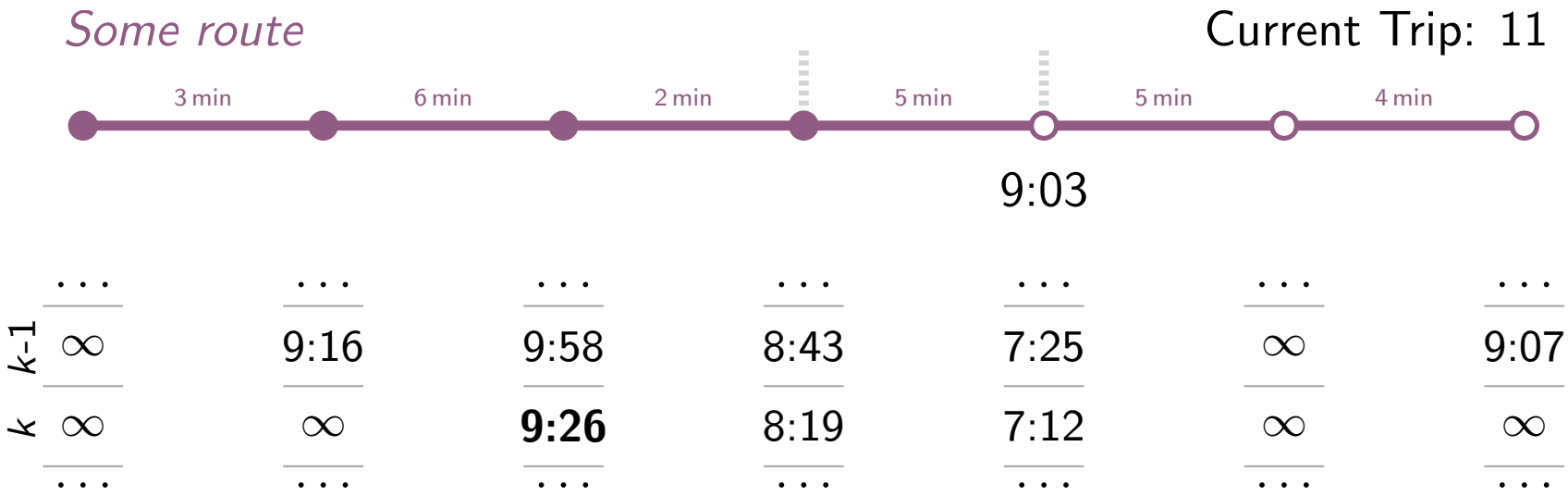


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Aktiver Trip entlang der Route wird stets verbessert.

In Runde k:

- Update Labels von Runde k mit Labels aus Runde $k-1$.

Scannen von Routen

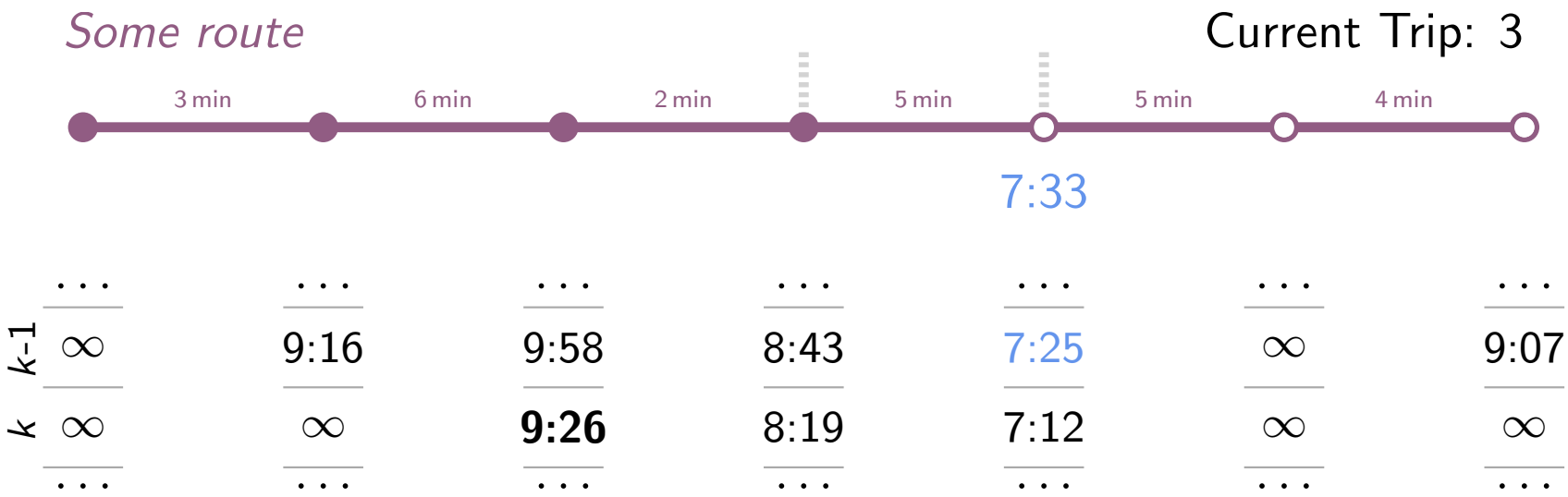


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Aktiver Trip entlang der Route wird stets verbessert.

In Runde k:

- Update Labels von Runde k mit Labels aus Runde $k-1$.

Scannen von Routen

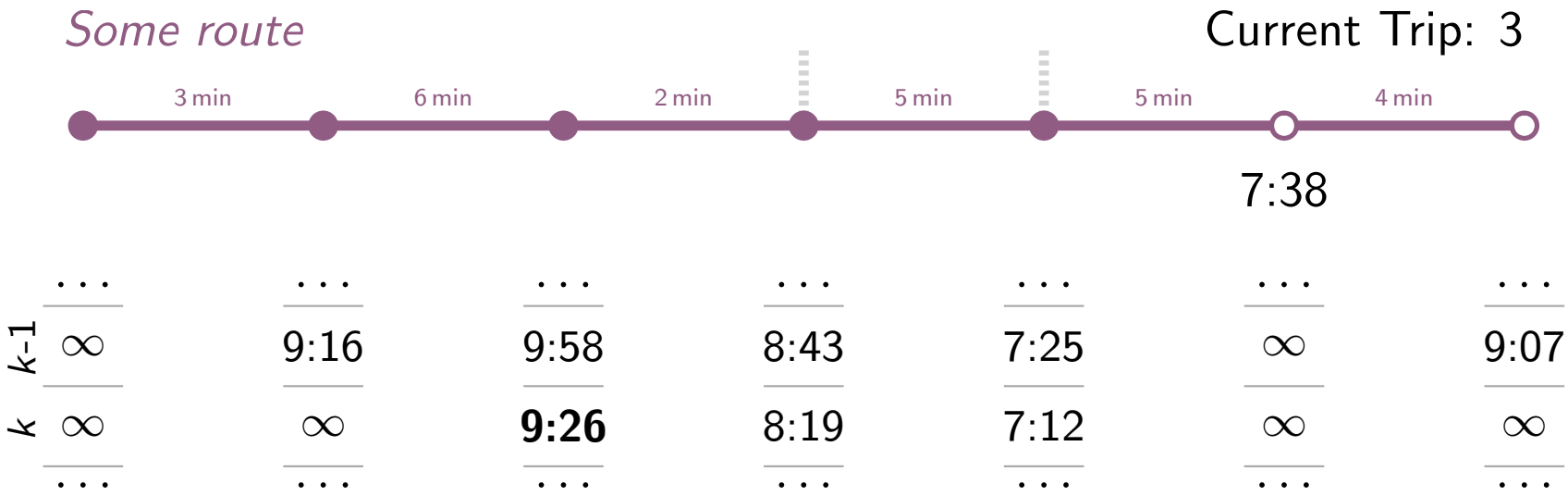


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Aktiver Trip entlang der Route wird stets verbessert.

In Runde k:

- Update Labels von Runde k mit Labels aus Runde $k-1$.

Scannen von Routen

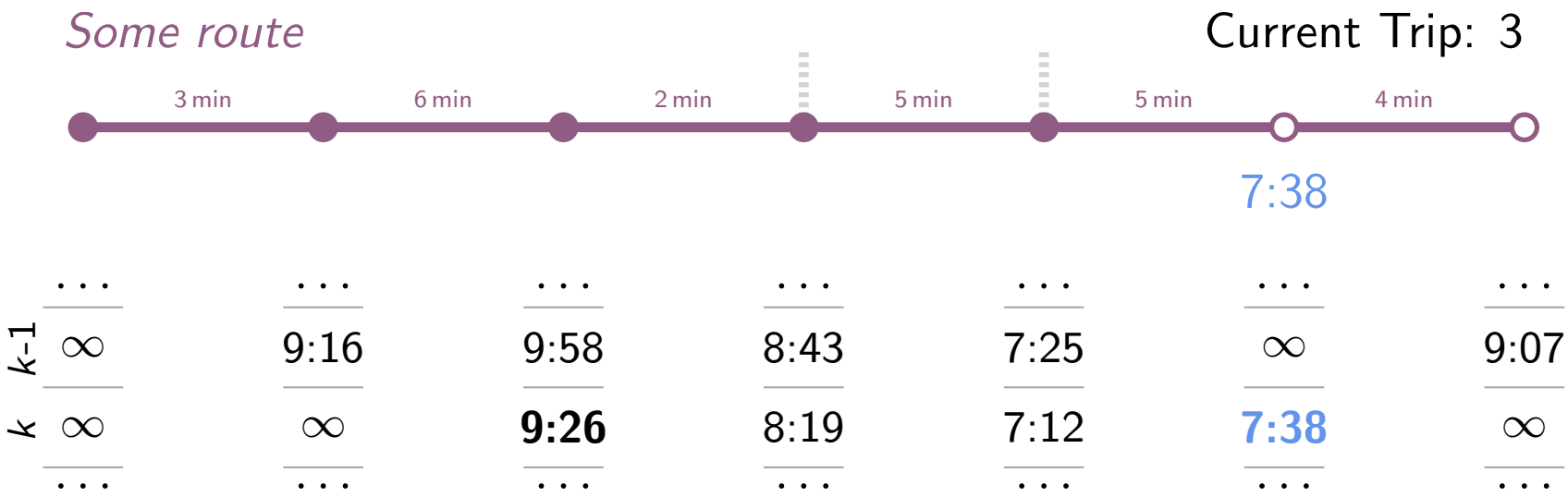


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Aktiver Trip entlang der Route wird stets verbessert.

In Runde k:

- Update Labels von Runde k mit Labels aus Runde $k-1$.

Scannen von Routen

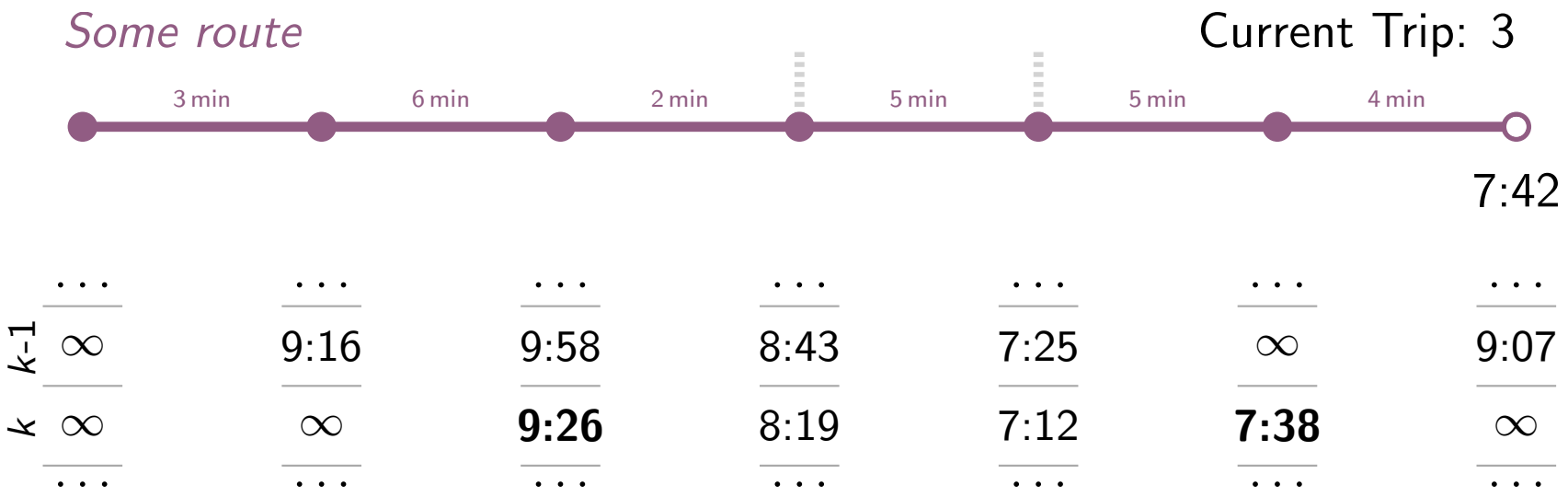


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Aktiver Trip entlang der Route wird stets verbessert.

In Runde k:

- Update Labels von Runde k mit Labels aus Runde $k-1$.

Scannen von Routen

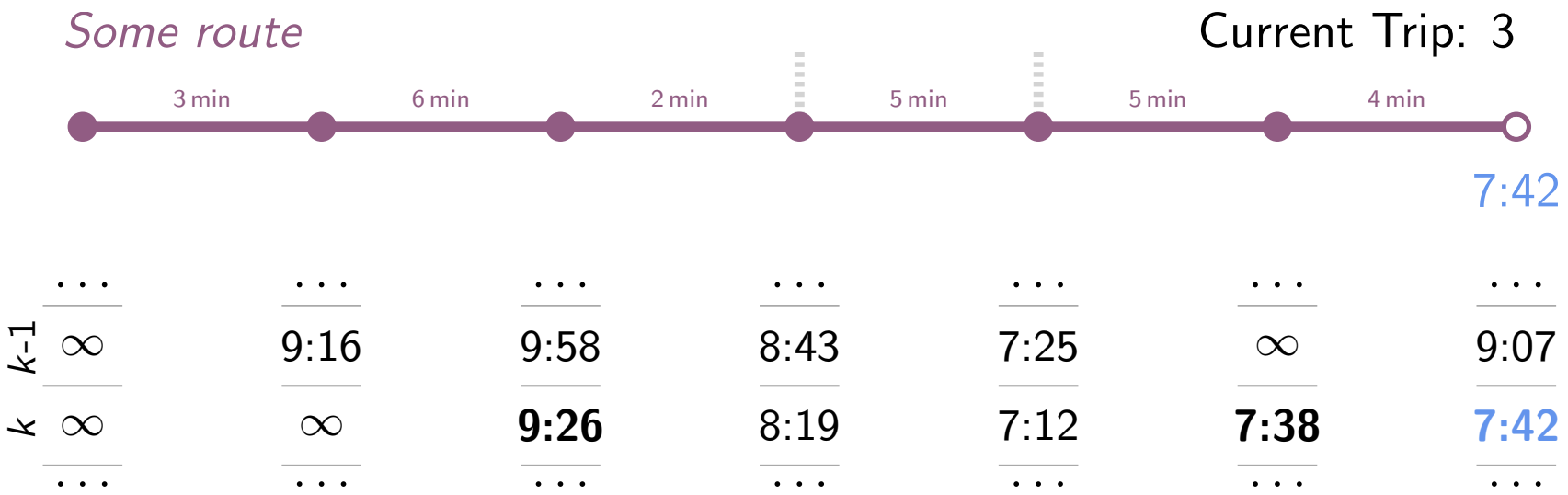


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Aktiver Trip entlang der Route wird stets verbessert.

In Runde k:

- Update Labels von Runde k mit Labels aus Runde $k-1$.

Scannen von Routen

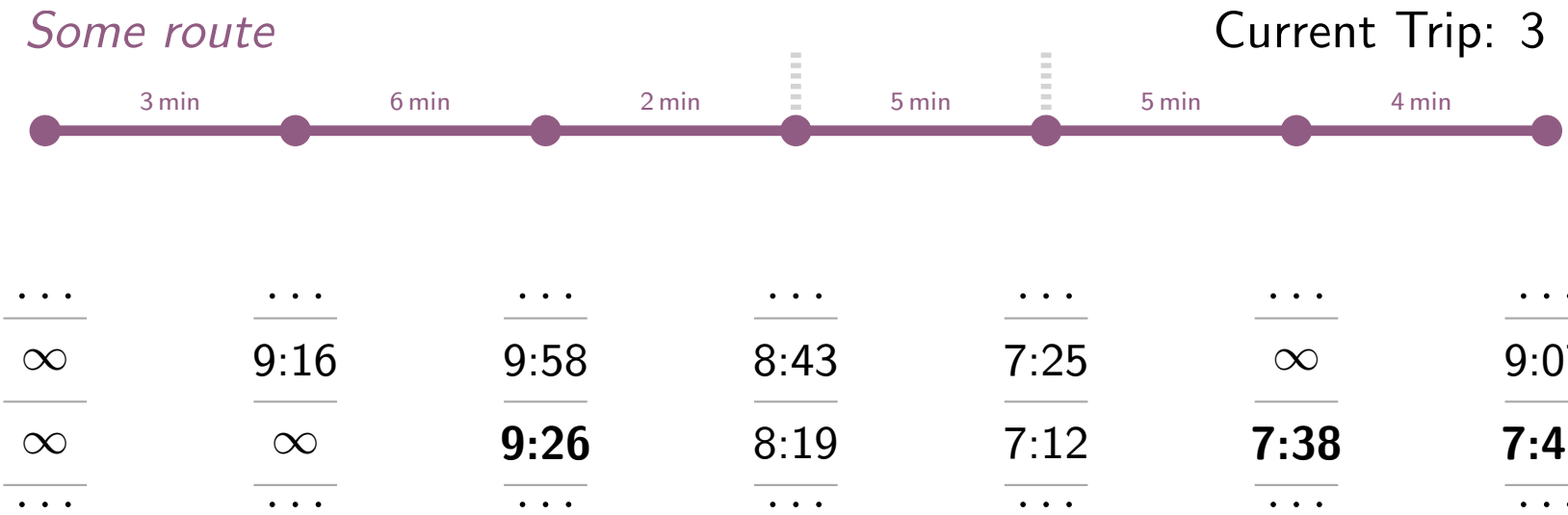


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Aktiver Trip entlang der Route wird stets verbessert.

In Runde k:

- Update Labels von Runde k mit Labels aus Runde $k-1$.

Scannen von Routen

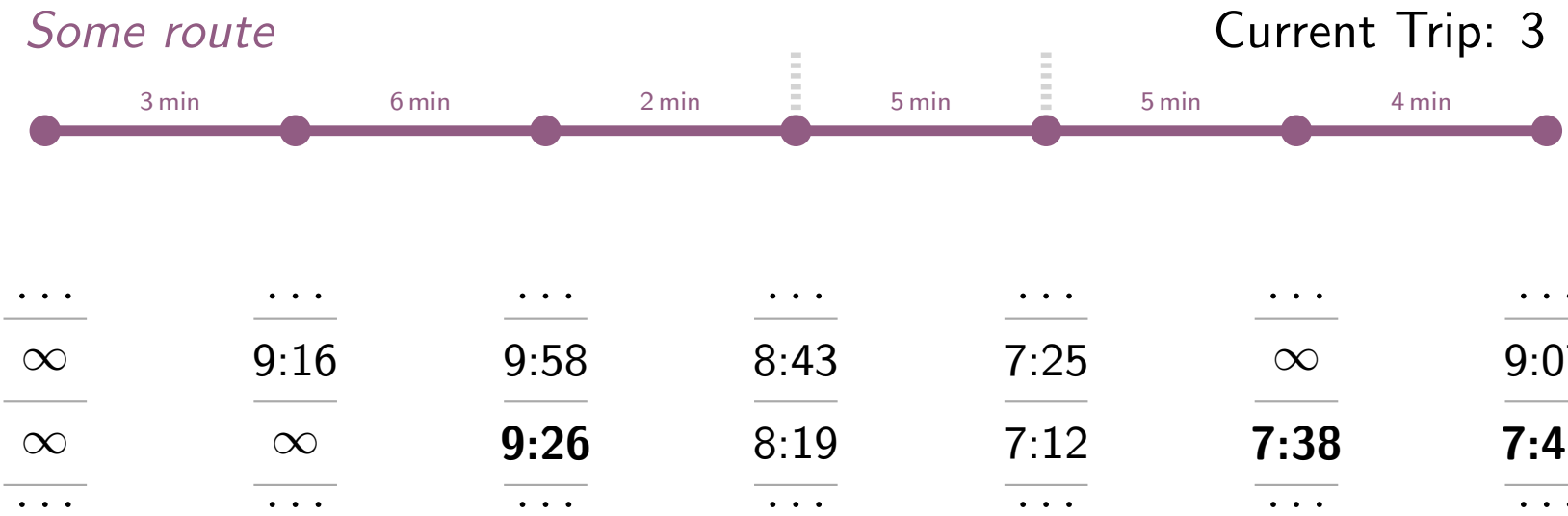


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Aktiver Trip entlang der Route wird stets verbessert.

In Runde k:

- Update Labels von Runde k mit Labels aus Runde $k-1$.

Scannen von Routen



- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Aktiver Trip entlang der Route wird stets verbessert.

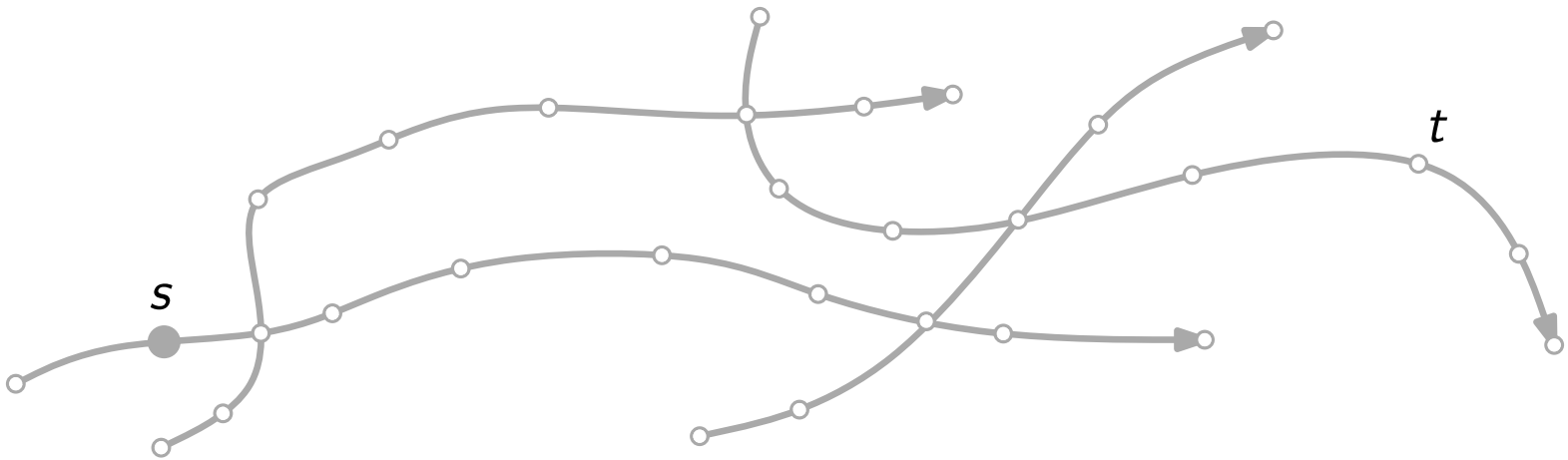
In Runde k:

- Update Labels von Runde k mit Labels aus Runde $k-1$.

Dynamischer Programmierungsansatz.

Markieren und Pruning

Beobachtung: Nicht alle Routen werden in jeder Runde erreicht.

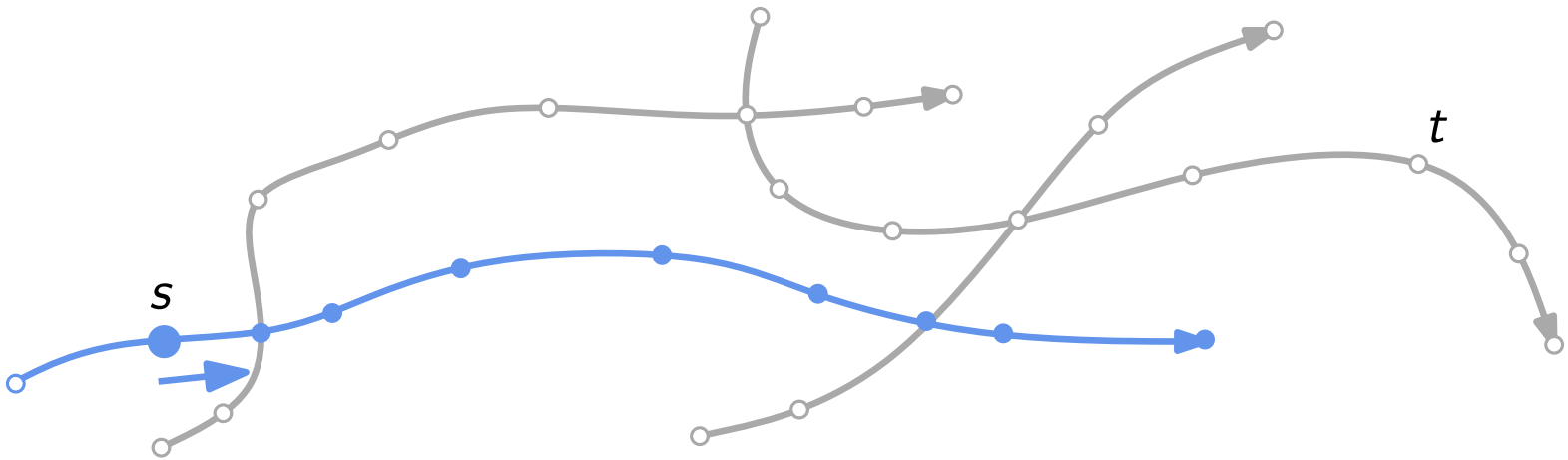


Markieren und Pruning

- Route scannen: Markiere Stop falls Ankunftszeit verbessert.
- Nächste Runde: Nur Routen von markierten Stops scannen.
- Scanne jede Route ab ihrem ersten markierten Stop.

Markieren und Pruning

Beobachtung: Nicht alle Routen werden in jeder Runde erreicht.

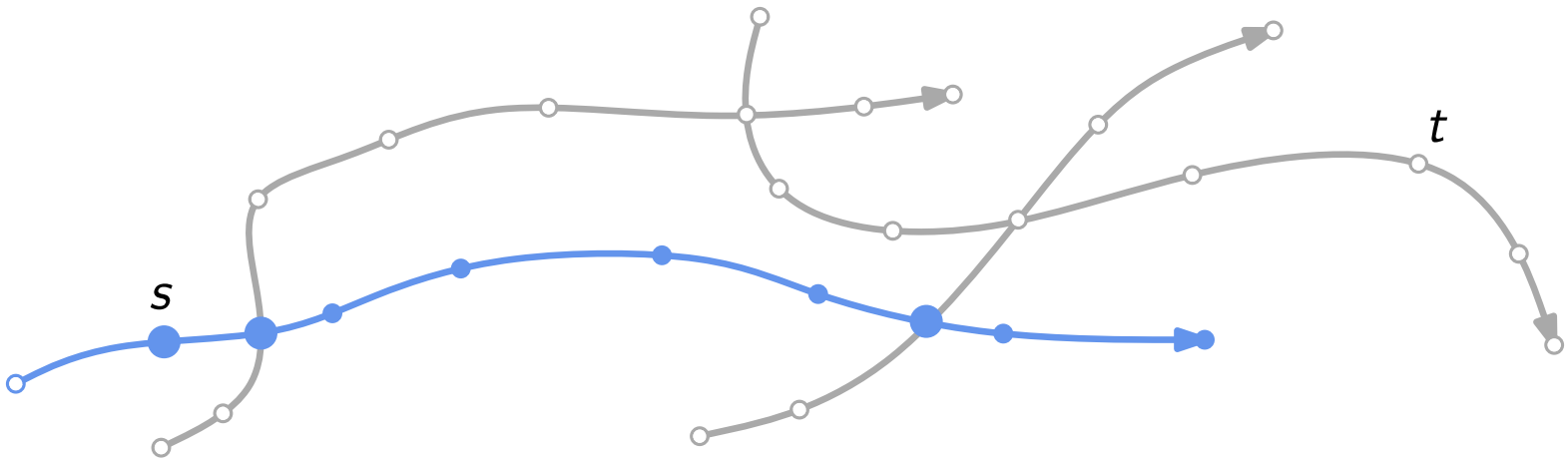


Markieren und Pruning

- Route scannen: Markiere Stop falls Ankunftszeit verbessert.
- Nächste Runde: Nur Routen von markierten Stops scannen.
- Scanne jede Route ab ihrem ersten markierten Stop.

Markieren und Pruning

Beobachtung: Nicht alle Routen werden in jeder Runde erreicht.

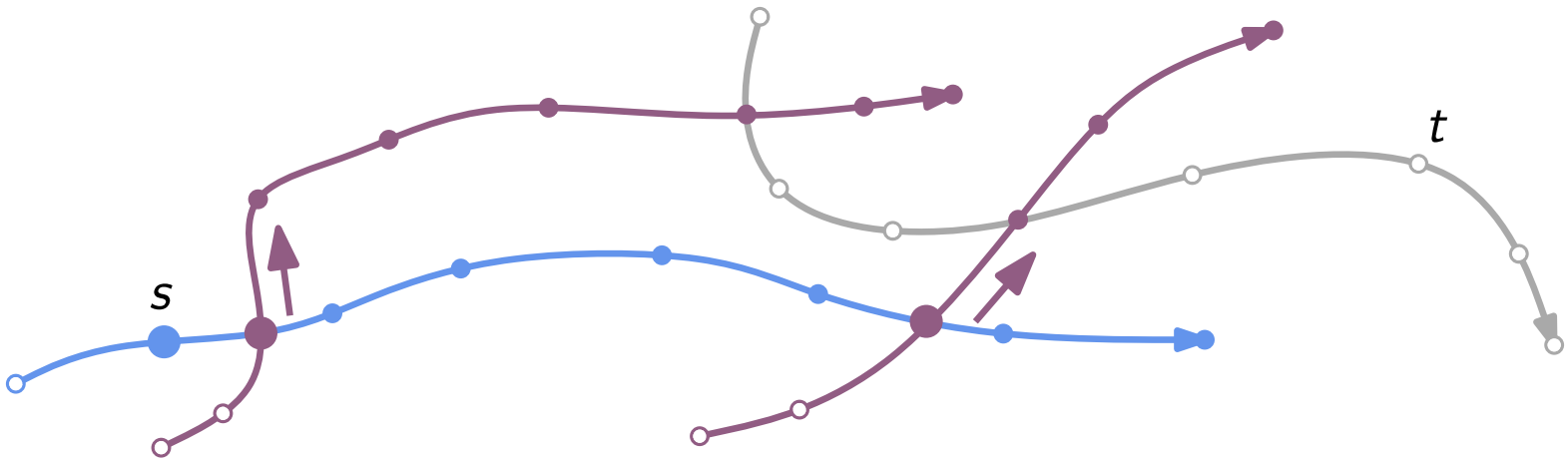


Markieren und Pruning

- Route scannen: Markiere Stop falls Ankunftszeit verbessert.
- Nächste Runde: Nur Routen von markierten Stops scannen.
- Scanne jede Route ab ihrem ersten markierten Stop.

Markieren und Pruning

Beobachtung: Nicht alle Routen werden in jeder Runde erreicht.

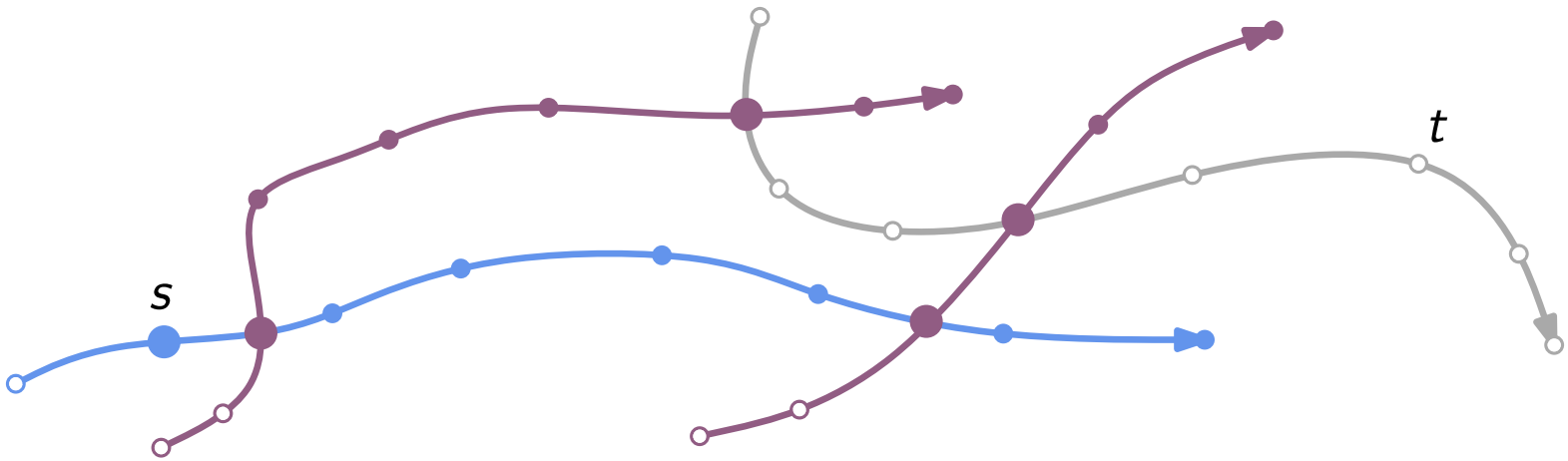


Markieren und Pruning

- Route scannen: Markiere Stop falls Ankunftszeit verbessert.
- Nächste Runde: Nur Routen von markierten Stops scannen.
- Scanne jede Route ab ihrem ersten markierten Stop.

Markieren und Pruning

Beobachtung: Nicht alle Routen werden in jeder Runde erreicht.

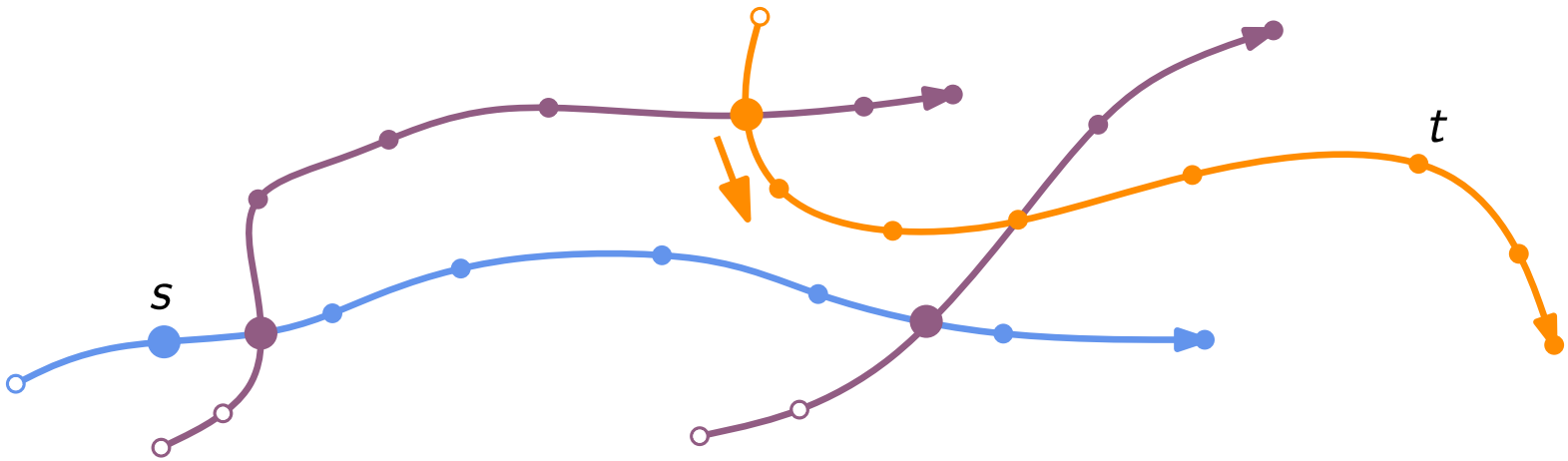


Markieren und Pruning

- Route scannen: Markiere Stop falls Ankunftszeit verbessert.
- Nächste Runde: Nur Routen von markierten Stops scannen.
- Scanne jede Route ab ihrem ersten markierten Stop.

Markieren und Pruning

Beobachtung: Nicht alle Routen werden in jeder Runde erreicht.

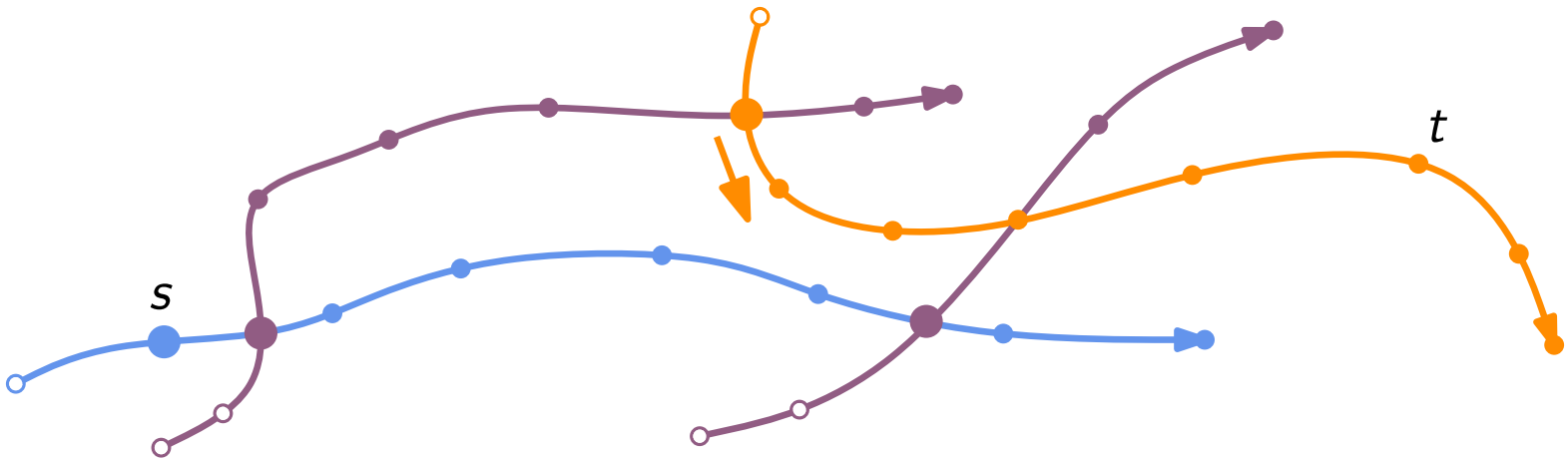


Markieren und Pruning

- Route scannen: Markiere Stop falls Ankunftszeit verbessert.
- Nächste Runde: Nur Routen von markierten Stops scannen.
- Scanne jede Route ab ihrem ersten markierten Stop.

Markieren und Pruning

Beobachtung: Nicht alle Routen werden in jeder Runde erreicht.



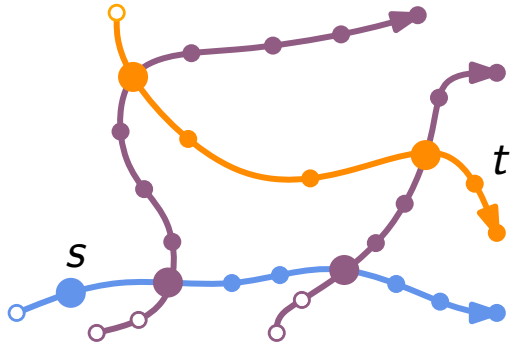
Markieren und Pruning

- Route scannen: Markiere Stop falls Ankunftszeit verbessert.
- Nächste Runde: Nur Routen von markierten Stops scannen.
- Scanne jede Route ab ihrem ersten markierten Stop.
- Markiere nur Stops wenn sie die beste Ankunftszeit am Ziel verbessern.

RAPTOR: Übersicht

Für jede Runde $k \leftarrow 1, 2, \ldots$

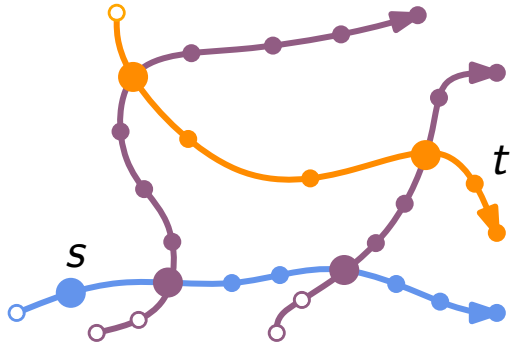
1. Wähle erreichte Routen aus letzter Runde
2. Scanne diese Routen
3. Relaxiere Fußwege



RAPTOR: Übersicht

Für jede Runde $k \leftarrow 1, 2, \ldots$

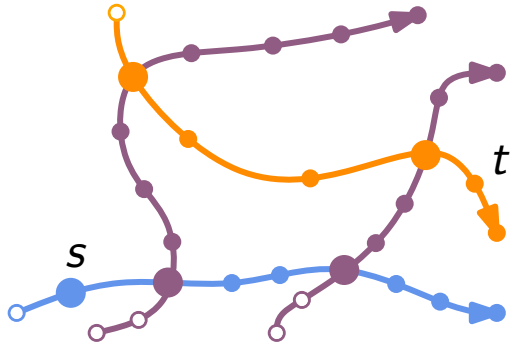
1. Wähle erreichte Routen aus letzter Runde
2. Scanne diese Routen
3. Relaxiere Fußwege



RAPTOR: Übersicht

Für jede Runde $k \leftarrow 1, 2, \ldots$

1. Wähle erreichte Routen aus letzter Runde
2. Scanne diese Routen
3. Relaxiere Fußwege

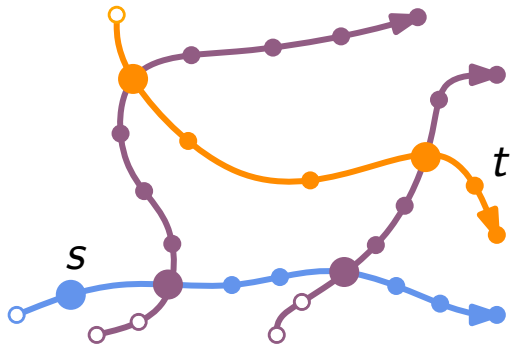


RAPTOR: Übersicht

Relaxiere Fußwege

Für jede Runde $k \leftarrow 1, 2, \ldots$

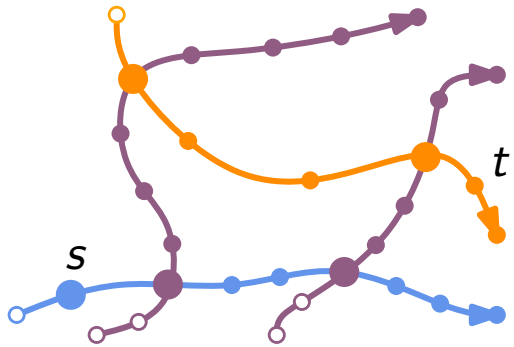
1. Wähle erreichte Routen aus letzter Runde
2. Scanne diese Routen
3. Relaxiere Fußwege



RAPTOR: Übersicht

Relaxiere Fußwege
Für jede Runde $k \leftarrow 1, 2, \ldots$

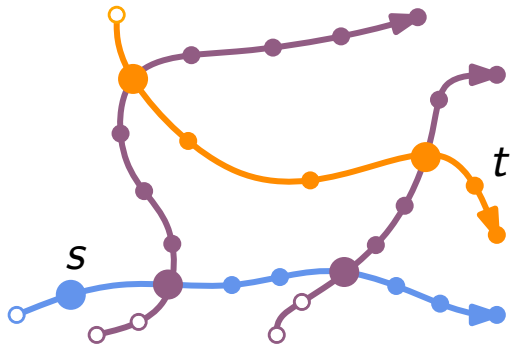
1. Wähle erreichte Routen aus letzter Runde
2. Scanne diese Routen
3. Relaxiere Fußwege



RAPTOR: Übersicht

Relaxiere Fußwege
Für jede Runde $k \leftarrow 1, 2, \ldots$

1. Wähle erreichte Routen aus letzter Runde
2. Scanne diese Routen
3. Relaxiere Fußwege



Terminiere, wenn kein Stop markiert wurde.

Parallelisierung

Beobachtung: Routen werden in bel. Reihenfolge gescannt.

Parallelisierung

Beobachtung: Routen werden in bel. Reihenfolge gescannt.

Verteile Routen auf verschiedene CPU Kerne; Scanne parallel.

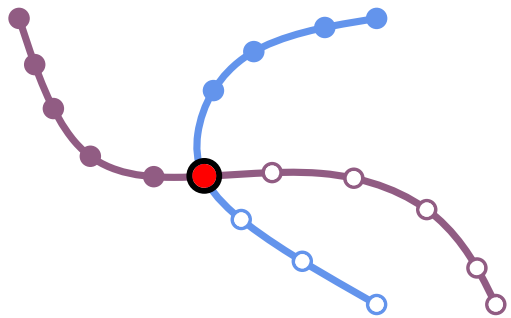
Parallelisierung

Beobachtung: Routen werden in bel. Reihenfolge gescannt.

Verteile Routen auf verschiedene CPU Kerne; Scanne parallel.

Vermeiden von Race-Conditions:

- Lock auf Schreiben von Labels (teuer).
- Synchronisiere Labels nach jeder Runde.
- Sicherstellen dass nur „unabhängige“ Routen gleichzeitig gescannt werden.



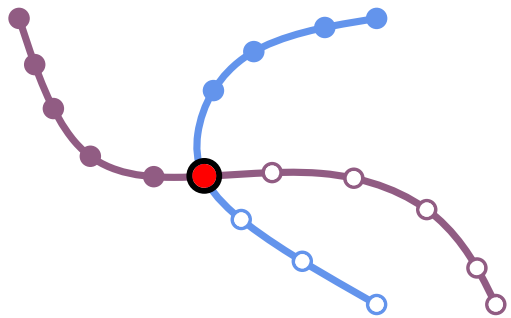
Parallelisierung

Beobachtung: Routen werden in bel. Reihenfolge gescannt.

Verteile Routen auf verschiedene CPU Kerne; Scanne parallel.

Vermeiden von Race-Conditions:

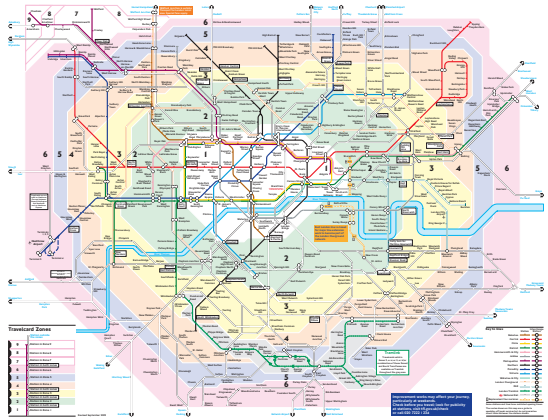
- Lock auf Schreiben von Labels (teuer).
- Synchronisiere Labels nach jeder Runde.
- Sicherstellen dass nur „unabhängige“ Routen gleichzeitig gescannt werden.
  (Reduktion auf Färbeproblem)



Andere Kriterien: Übersicht

Mögliche Erweiterungen

- Profil-Anfragen (Intervallanfragen),
  Flexible Abfahrtszeiten.
- Tarifzonen,
  Längere Routen könnten billiger sein.
- Umstiegssicherheit,
  Routen könnten knappe Umstiege haben.
- ...

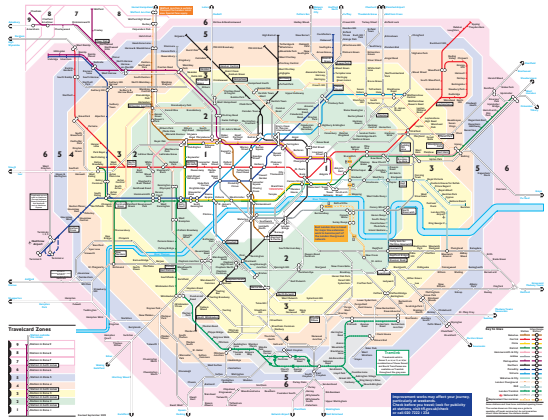


Performance hängt von Anzahl *nichtdominierter* Routen ab.

Andere Kriterien: Übersicht

Mögliche Erweiterungen

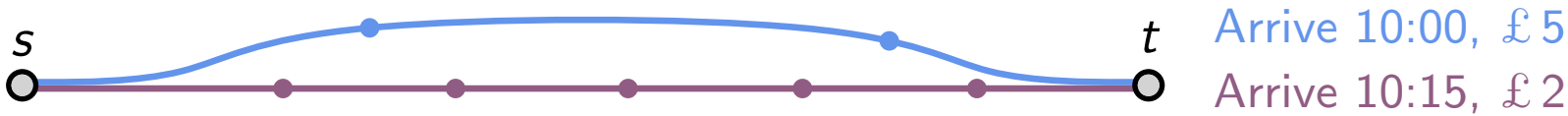
- Profil-Anfragen (Intervallanfragen),
  Flexible Abfahrtszeiten.
- Tarifzonen,
  Längere Routen könnten billiger sein.
- Umstiegssicherheit,
  Routen könnten knappe Umstiege haben.
- ...



Performance hängt von Anzahl *nichtdominierter* Routen ab.

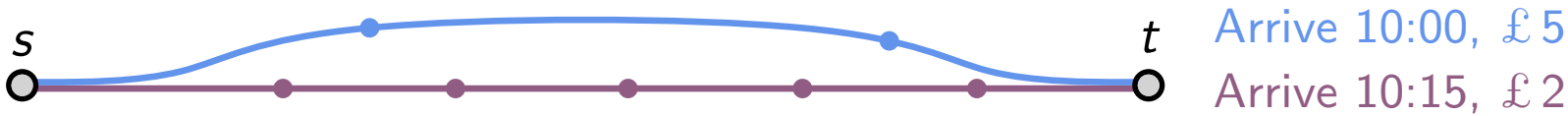
More Criteria: McRAPTOR

Ziel: Erweitern von RAPTOR auf zusätzliche Kriterien.



More Criteria: McRAPTOR

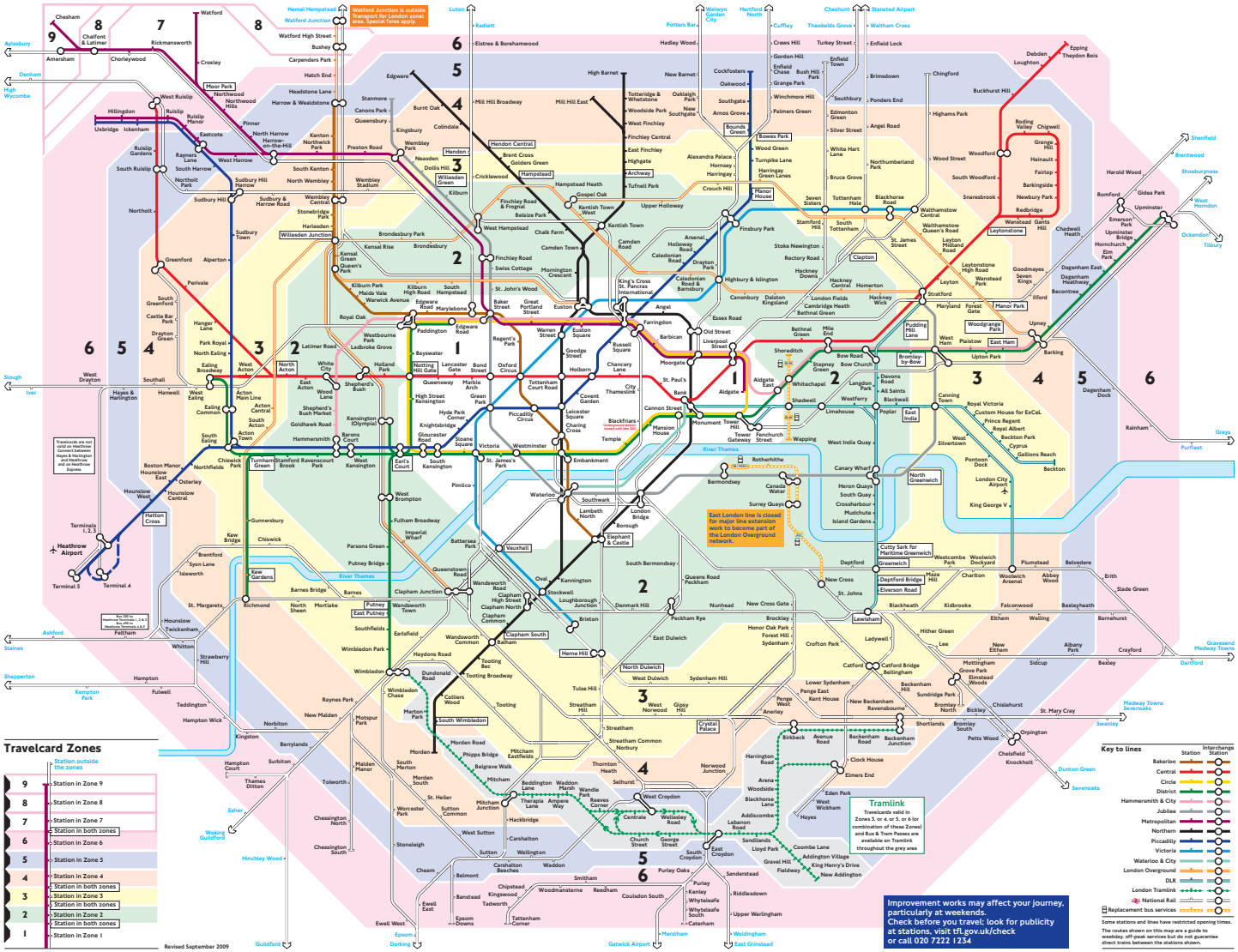
Ziel: Erweitern von RAPTOR auf zusätzliche Kriterien.



Ansatz

- Labels haben Wert für jedes zusätzliche Kriterium.
- Mehrere nichtdominierte Labels pro Stop und Runde.
- Mehrere aktive Trips beim Scannen von Routen.
- Lösche dominierte Labels on-the-fly.

McRAPTOR Beispiel: Tarifzonen



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

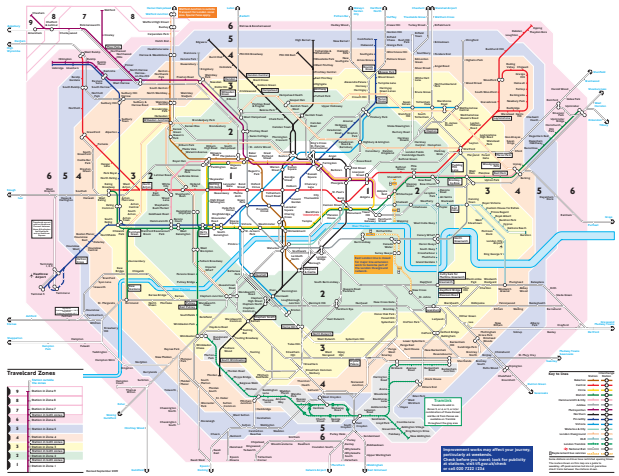
McRAPTOR Beispiel: Tarifzonen

Tarifzonen einbauen

- Direkte Preise (£) nicht handhabbar.

$\Rightarrow$ Berechne alle Kombinationen von Tarifzonen,

- und filtere im Postprocessing.



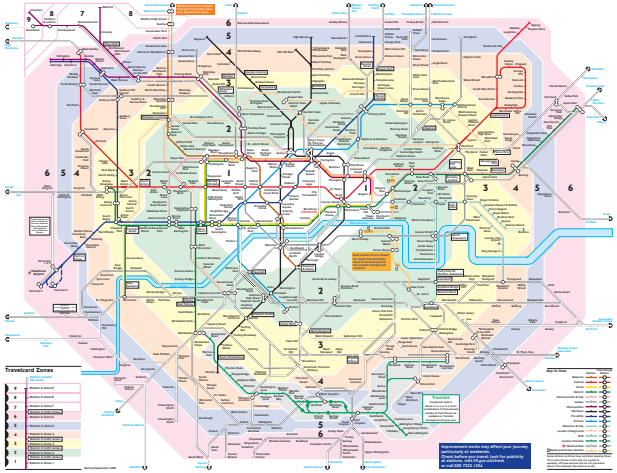
McRAPTOR Beispiel: Tarifzonen

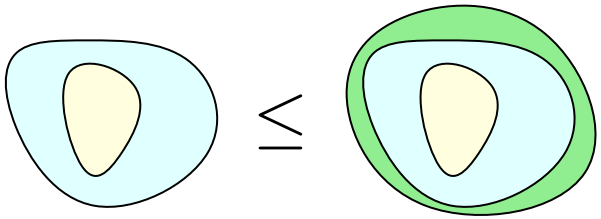
Tarifzonen einbauen

- Direkte Preise (£) nicht handhabbar.

$\Rightarrow$ Berechne alle Kombinationen von Tarifzonen,

- und filtere im Postprocessing.





Implementierung

- *Mengen* von Tarifzonen als Kriterium.
- Dominieren $\widehat{=}$ Teilmengenrelation.
- Benutze Bits von `int64` für Mengen.

Profil-Anfragen: rRAPTOR

Problem: Finde alle besten Verbindungen die in einem Zeitintervall Δ abfahren.

Profil-Anfragen: rRAPTOR

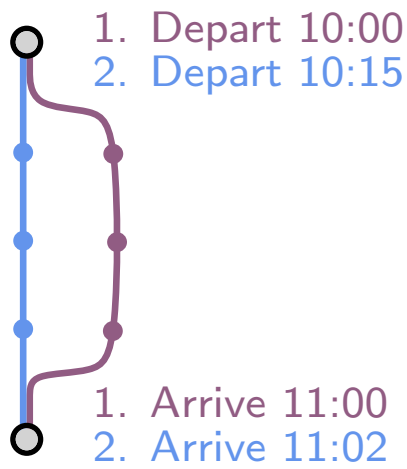
Problem: Finde alle besten Verbindungen die in einem Zeitintervall Δ abfahren.

- Lösbar mit McRAPTOR...
- ...mit Abfahrtszeit als Kriterium.

Profil-Anfragen: rRAPTOR

Problem: Finde alle besten Verbindungen die in einem Zeitintervall Δ abfahren.

- Lösbar mit McRAPTOR...
- ...mit Abfahrtszeit als Kriterium.



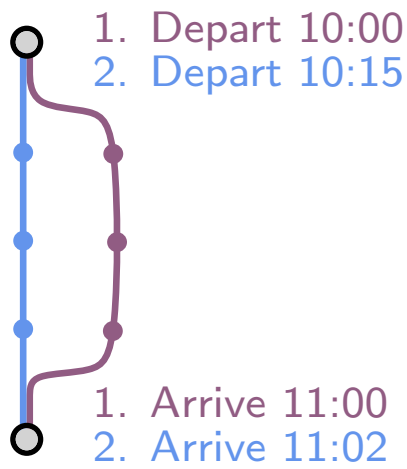
Effizienterer Ansatz: rRAPTOR (Self-Pruning)

- Sammle alle Abfahrten aus Intervall Δ in Menge $\mathcal{D}$.
- Dann: RAPTOR für jedes $\tau \in \mathcal{D}$ geordnet absteigend nach Zeit.
- Reinitialisiere keine Labels zwischen den Aufrufen!

Profil-Anfragen: rRAPTOR

Problem: Finde alle besten Verbindungen die in einem Zeitintervall Δ abfahren.

- Lösbar mit McRAPTOR...
- ...mit Abfahrtszeit als Kriterium.



Effizienterer Ansatz: rRAPTOR (Self-Pruning)

- Sammle alle Abfahrten aus Intervall Δ in Menge $\mathcal{D}$.
- Dann: RAPTOR für jedes $\tau \in \mathcal{D}$ geordnet absteigend nach Zeit.
- Reinitialisiere keine Labels zwischen den Aufrufen!

Prunt implizit Routen die früher abfahren und später ankommen.

Experimente

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Das Londoner Netzwerk

Das vollständige Londoner Netzwerk

- Ein Dienstag.
- Beinhaltet Tube, Bus, DLR, Tram. . .
- 20 843 Stops,
- 2 225 Routen mit 133 011 Trips,
- 5 132 672 einzelne Abfahrten pro Tag.

Das Londoner Netzwerk

Das vollständige Londoner Netzwerk

- Ein Dienstag.
- Beinhaltet Tube, Bus, DLR, Tram. . .
- 20 843 Stops,
- 2 225 Routen mit 133 011 Trips,
- 5 132 672 einzelne Abfahrten pro Tag.

Experimente: 10 000 zufällige s–t-Anfragen.

Vergleich der Algorithmen

(Hardware: Intel Xeon X5680 mit 3.33 GHz und 96 GiB DDR3-1333 RAM)

Algorithm	Ar	R	Tr	Fz	Rounds	Journeys	[ms]
Dijkstra	●	○	○	○	—	0.9	14.2
RAPTOR	●	○	●	○	8.4	1.9	7.3
LD	●	○	●	○	—	1.9	44.5
MLC	●	○	●	○	—	1.9	67.2
McRAPTOR	●	○	●	●	10.8	9.0	107.4
MLC	●	○	●	●	—	9.0	399.5
McRAPTOR	●	●	●	○	9.5	16.3	259.8
rRAPTOR	●	●	●	○	138.5	16.3	87.0
SPCS	●	●	○	○	—	7.8	183.6

(**Ar**: Arrival Time, **R**: Range, **Tr**: Transfers, **Fz**: Fare Zones)

Parallelisierung

Algorithm	Ar	R	Tr	Fz	**1 core** [ms]	**3 cores** [ms]	**6 cores** [ms]	**12 cores** [ms]
RAPTOR	●	○	●	○	7.7	5.0	4.1	3.7
McRAPTOR	●	○	●	●	118.6	49.4	29.9	26.1
rRAPTOR	●	●	●	○	92.3	39.5	26.8	21.6
SPCS	●	●	○	○	183.6	69.1	44.9	38.9

(**Ar**: Arrival Time, **R**: Range, **Tr**: Transfers, **Fz**: Fare Zones)

Parallelisierung

Algorithm	Ar	R	Tr	Fz	1 core [ms]	3 cores [ms]	6 cores [ms]	12 cores [ms]
RAPTOR	●	○	●	○	7.7	5.0	4.1	3.7
McRAPTOR	●	○	●	●	118.6	49.4	29.9	26.1
rRAPTOR	●	●	●	○	92.3	39.5	26.8	21.6
SPCS	●	●	○	○	183.6	69.1	44.9	38.9

(**Ar**: Arrival Time, **R**: Range, **Tr**: Transfers, **Fz**: Fare Zones)

- Exzellente Speedups auf bis zu 6 Kernen.
- RAPTOR immer ≤ 30 ms.

Literatur I

Daniel Delling, Thomas Pajor, and Renato F. Werneck.

Round-Based Public Transit Routing.

In *Proceedings of the 14th Meeting on Algorithm Engineering and Experiments (ALENEX'12)*, pages 130–140. SIAM, 2012.