Karlsruher Institut für Technologie

Algorithmen für Routenplanung

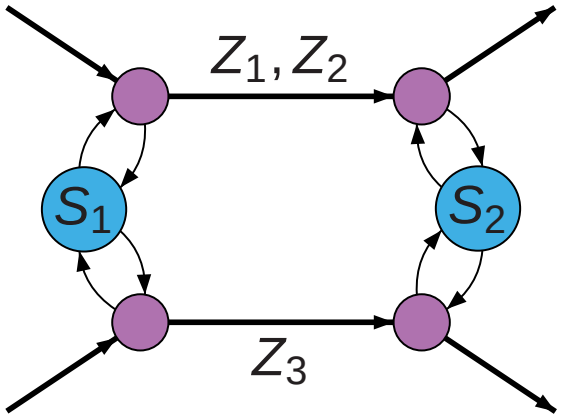
16. Sitzung, Sommersemester 2013

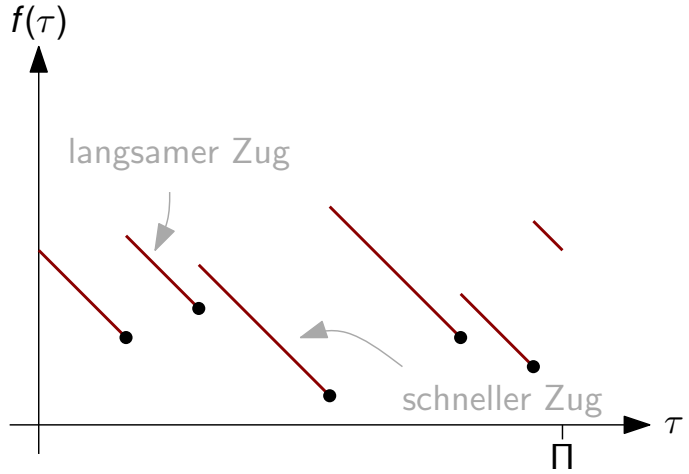
Thomas Pajor | 1. Juli 2013

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

Letztes Mal...

Einführung in die Fahrplanauskunft



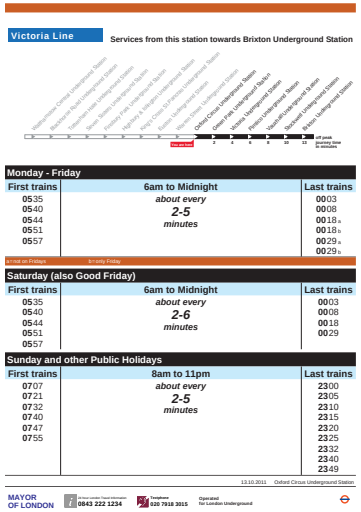


Funktion f durch Punkte: $I^f := \{(t_1^f, w_1^f), \ldots, (t_k^f, w_k^f)\}$

Bis jetzt. . .

Erinnerung: Ein Fahrplan besteht aus

- Stops (Bahnhöfe, Bahnsteige, . . .),
- Routen (Bus-, U-Bahn Linien, . . .),
- Trips mit Abfahrt-/Ankunftszeiten,
- und Fußwegen zum Umsteigen.



Victoria Line — Services from this station towards Brixton Underground Station

Monday - Friday

First trains	6am to Midnight	Last trains
0535	about every	0003
0540	2-5	0008
0544	minutes	0018 a
0551		0018 b
0557		0029 a
		0029 b

a Not on Fridays b Only Friday

Saturday (also Good Friday)

First trains	6am to Midnight	Last trains
0535	about every	0003
0540	2-6	0008
0544	minutes	0018
0551		0029
0557		

Sunday and other Public Holidays

First trains	8am to 11pm	Last trains
0707	about every	2300
0721	2-5	2305
0732	minutes	2310
0740		2315
0747		2320
0755		2325
		2332
		2340
		2349

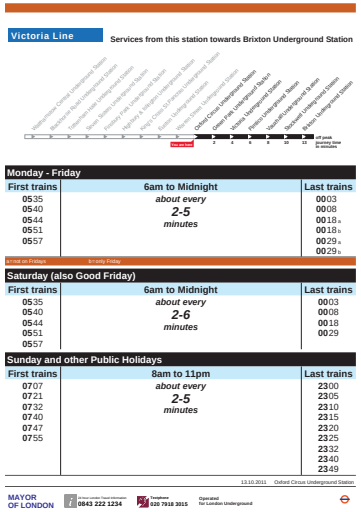
13.10.2011 Oxford Circus Underground Station

MAYOR OF LONDON — 0843 222 1234 — 020 7918 3015 — Operated for London Underground

Bis jetzt. . .

Erinnerung: Ein Fahrplan besteht aus

- Stops (Bahnhöfe, Bahnsteige, . . .),
- Routen (Bus-, U-Bahn Linien, . . .),
- Trips mit Abfahrt-/Ankunftszeiten,
- und Fußwegen zum Umsteigen.



Victoria Line — Services from this station towards Brixton Underground Station

Monday - Friday

First trains	6am to Midnight	Last trains
0535	about every	0003
0540	2-5	0008
0544	minutes	0018 a
0551		0018 b
0557		0029 a
		0029 b

Saturday (also Good Friday)

First trains	6am to Midnight	Last trains
0535	about every	0003
0540	2-6	0008
0544	minutes	0018
0551		0029
0557		

Sunday and other Public Holidays

First trains	8am to 11pm	Last trains
0707	about every	2300
0721	2-5	2305
0732	minutes	2310
0740		2315
0747		2320
0755		2325
		2332
		2340
		2349

13.10.2011 Oxford Circus Underground Station

MAYOR OF LONDON — 0843 222 1234 — 020 7918 3015 — Operated for London Underground

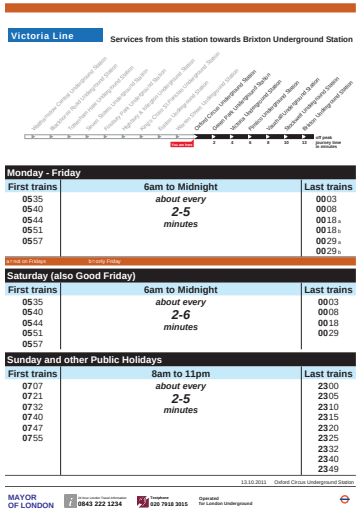
Earliest Arrival Problem:
Gegeben Stops s, t und Abfahrtszeit τ, berechne

- Route zu t die an s nicht früher als τ abfährt,
- und an t frühstmöglich ankommt.

Bis jetzt...

Erinnerung: Ein Fahrplan besteht aus

- Stops (Bahnhöfe, Bahnsteige, ...),
- Routen (Bus-, U-Bahn Linien, ...),
- Trips mit Abfahrt-/Ankunftszeiten,
- und Fußwegen zum Umsteigen.



Victoria Line — Services from this station towards Brixton Underground Station

Monday - Friday

First trains	6am to Midnight	Last trains
0535	about every	0003
0540	2-5	0008
0544	minutes	0018 a
0551		0018 b
0557		0029 a
		0029 b

Saturday (also Good Friday)

First trains	6am to Midnight	Last trains
0535	about every	0003
0540	2-6	0008
0544	minutes	0018
0551		0029
0557		

Sunday and other Public Holidays

First trains	8am to 11pm	Last trains
0707	about every	2300
0721	2-5	2305
0732	minutes	2310
0740		2315
0747		2320
0755		2325
		2332
		2340
		2349

13.10.2011 Oxford Circus Underground Station

MAYOR OF LONDON 0843 222 1234 020 7918 3015 Operated for London Underground

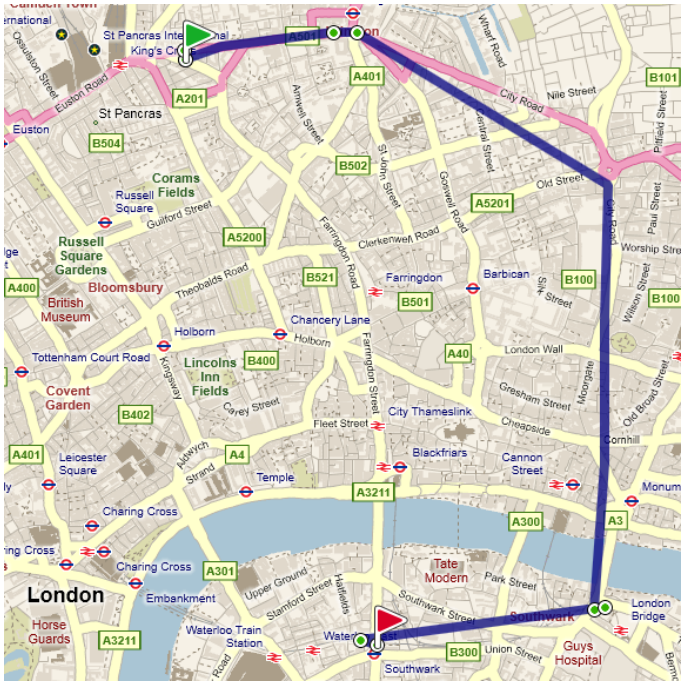
Earliest Arrival Problem:
Gegeben Stops s, t und Abfahrtszeit τ, berechne

- Route zu t die an s nicht früher als τ abfährt,
- und an t frühstmöglich ankommt.

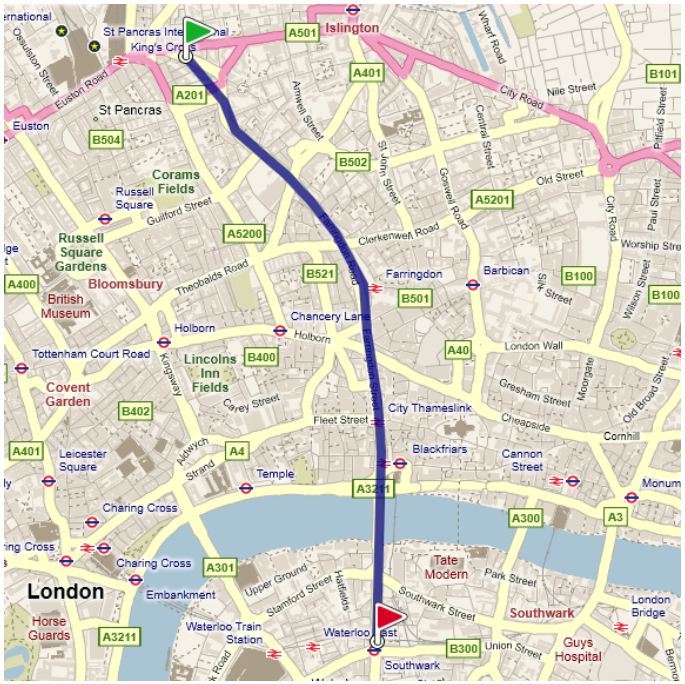
Reicht uns das?

Einbeziehen von Umstiegen

Umstiege zu betrachten ist wichtig!



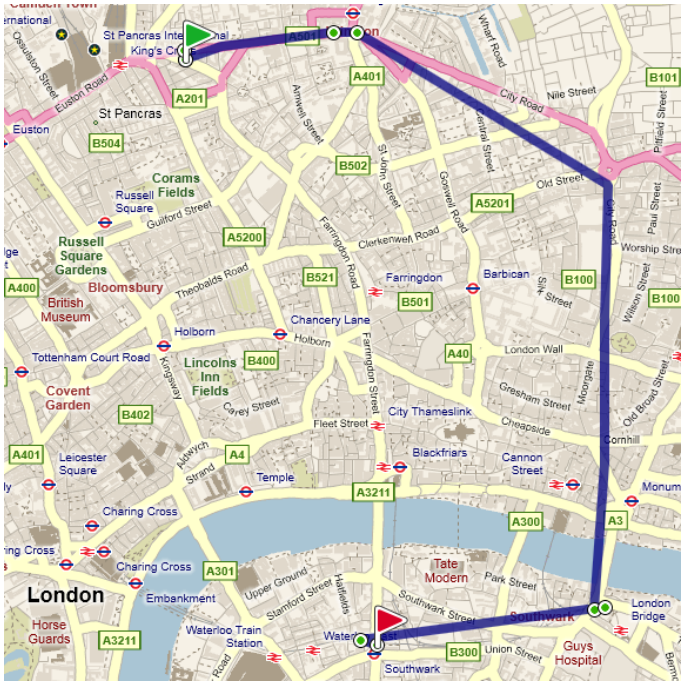
Ankunft 11:08 Uhr, 2 Umstiege



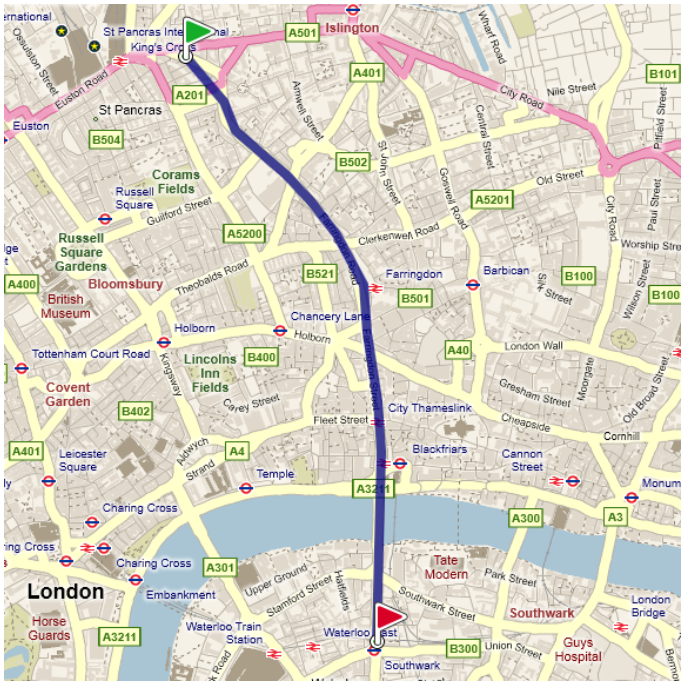
Ankunft 11:09 Uhr, 0 Umstiege

Einbeziehen von Umstiegen

Umstiege zu betrachten ist wichtig!



Ankunft 11:08 Uhr, 2 Umstiege



Ankunft 11:09 Uhr, 0 Umstiege

Idee: Berechne „gute“ Routen für Ankunftszeit *und* Anzahl Umstiege.

Pareto-Menge

Definition (Pareto-Optimum)

Zu einer Menge M von n-Tupeln heißt ein Tupel $m_i = (x_1, \ldots, x_n) \in M$ *Pareto-Optimum*, wenn es kein anderes $m_j \in M$ gibt, so dass m_j in allen Werten besser als m_i ist (m_j *dominiert* m_i).

Pareto-Menge

Definition (Pareto-Optimum)

Zu einer Menge M von n-Tupeln heißt ein Tupel $m_i = (x_1, \ldots, x_n) \in M$ *Pareto-Optimum*, wenn es kein anderes $m_j \in M$ gibt, so dass m_j in allen Werten besser als m_i ist (m_j *dominiert* m_i).

Die Menge M heißt *Pareto-Menge*, wenn alle $m \in M$ Pareto-optimal.

Pareto-Menge

Definition (Pareto-Optimum)

Zu einer Menge M von n-Tupeln heißt ein Tupel $m_i = (x_1, \ldots, x_n) \in M$ *Pareto-Optimum*, wenn es kein anderes $m_j \in M$ gibt, so dass m_j in allen Werten besser als m_i ist (m_j *dominiert* m_i).

Die Menge M heißt *Pareto-Menge*, wenn alle $m \in M$ Pareto-optimal.

Beispiel: Betrachte Tupel aus Ankunftszeit und # Umstiege.

$M = \{(14{:}00 \text{ Uhr}, 5),\ (15{:}13 \text{ Uhr}, 3),\ (13{:}45 \text{ Uhr}, 4),\ (15{:}15 \text{ Uhr}, 0)\}$.

Pareto-Menge

Definition (Pareto-Optimum)

Zu einer Menge M von n-Tupeln heißt ein Tupel $m_i = (x_1, \ldots, x_n) \in M$ *Pareto-Optimum*, wenn es kein anderes $m_j \in M$ gibt, so dass m_j in allen Werten besser als m_i ist (m_j *dominiert* m_i).

Die Menge M heißt *Pareto-Menge*, wenn alle $m \in M$ Pareto-optimal.

Beispiel: Betrachte Tupel aus Ankunftszeit und # Umstiege.

$M = \{(15{:}13 \text{ Uhr}, 3),\ (13{:}45 \text{ Uhr}, 4),\ (15{:}15 \text{ Uhr}, 0)\}$.

Pareto-Menge

Definition (Pareto-Optimum)

Zu einer Menge M von n-Tupeln heißt ein Tupel $m_i = (x_1, \ldots, x_n) \in M$ *Pareto-Optimum*, wenn es kein anderes $m_j \in M$ gibt, so dass m_j in allen Werten besser als m_i ist (m_j *dominiert* m_i).

Die Menge M heißt *Pareto-Menge*, wenn alle $m \in M$ Pareto-optimal.

Beispiel: Betrachte Tupel aus Ankunftszeit und # Umstiege.

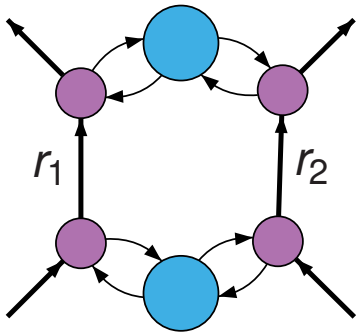
$M = \{(15{:}13 \text{ Uhr}, 3),\ (13{:}45 \text{ Uhr}, 4),\ (15{:}15 \text{ Uhr}, 0)\}$.

Wie effizient berechnen?

Multi-Label-Correcting Ansatz

Idee

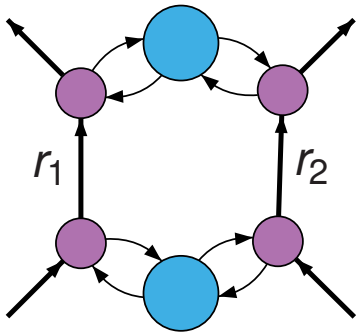
- Benutze (zeitabhängiges) Graphmodell
- Grundlage: Dijkstra's Algorithmus



Multi-Label-Correcting Ansatz

Idee

- Benutze (zeitabhängiges) Graphmodell
- Grundlage: Dijkstra's Algorithmus



. . . aber. . .

- Label ℓ sind 2-Tupel aus Ankunftszeit und # Umstiege
- An jedem Knoten $u \in V$: Pareto-Menge B_u von Labeln
- Priority Queue verwaltet Label statt Knoten
- Priorität ist Ankunftszeit
- Dominanz von Labeln in B_u on-the-fly

Multi-Label-Correcting (MLC)

MLC($G = (V, E)$, s, τ)

```
1  B_u ← {} for each u ∈ V; B_s ← {(τ, 0)}
2  Q.clear(), Q.insert(s, (τ, 0))
3  while !Q.empty() do
4      u and ℓ = (τ, tr) ← Q.deleteMin()
5      for all edges e = (u, v) ∈ E do
6          if e is a transfer edge then tr' ← tr + 1
7          else tr' ← tr
8          ℓ' ← (τ + len(e, τ), tr')
9          if ℓ' is not dominated by any ℓ'' ∈ B_v then
10             B_v.insert(ℓ')
11             Remove non-Pareto-optimal labels from B_v
12             Q.insert(v, ℓ')
```

Diskussion: MLC

Diskussion:

- Pareto-Mengen B_u sind *dynamische* Datenstrukturen $\rightsquigarrow$ teuer!
- Sehr viele Queue-Operationen
- Testen der Dominanz in $\mathcal{O}(|B_u|)$ möglich
- Stoppkriterium?

Diskussion: MLC

Diskussion:

- Pareto-Mengen B_u sind *dynamische* Datenstrukturen $\rightsquigarrow$ teuer!
- Sehr viele Queue-Operationen
- Testen der Dominanz in $\mathcal{O}(|B_u|)$ möglich
- Stoppkriterium?

Verbesserungen für MLC:

- Jedes B_u verwaltet bestes ungesettletes Label selbst
  $\Rightarrow$ Priority Queue auf Knoten statt Labeln
- Label-Forwarding:
  Wenn Kante keine Kosten hat, überspringe Queue
- Target-Pruning:
  An Knoten u, verwerfe Label ℓ', wenn B_t dominiert ℓ'

Layered Dijkstra

Bottleneck: Dynamische Datenstruktur für Pareto-Mengen

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Layered Dijkstra

Bottleneck: Dynamische Datenstruktur für Pareto-Mengen

Beobachtung:
Kriterium Umstiege ist *diskret* und nimmt nur *kleine Werte* $0 \ldots K$ an.

Layered Dijkstra

Bottleneck: Dynamische Datenstruktur für Pareto-Mengen

Beobachtung:
Kriterium Umstiege ist *diskret* und nimmt nur *kleine Werte* $0 \dots K$ an.

Idee (Layered Dijkstra):

- Kopiere Graph K-Mal in Layer $G_0, G_1, \dots, G_K$
- Biege in jedem Layer i die Transfer Kanten $e = (u, v)$ so um, dass sie in den nächsten Layer zeigen ($u \in V_i$, $v \in V_{i+1}$)
- Benutze zeitabh. Dijkstra mit Startknoten $s_0 \in V_0$ und Zeit τ
- Adaptiere Dominanz zwischen Layern und Target-Pruning

Layered Dijkstra

Bottleneck: Dynamische Datenstruktur für Pareto-Mengen

Beobachtung:
Kriterium Umstiege ist *diskret* und nimmt nur *kleine Werte* $0 \dots K$ an.

Idee (Layered Dijkstra):

- Kopiere Graph K-Mal in Layer $G_0, G_1, \dots, G_K$
- Biege in jedem Layer i die Transfer Kanten $e = (u, v)$ so um, dass sie in den nächsten Layer zeigen ($u \in V_i$, $v \in V_{i+1}$)
- Benutze zeitabh. Dijkstra mit Startknoten $s_0 \in V_0$ und Zeit τ
- Adaptiere Dominanz zwischen Layern und Target-Pruning

Dann: $d_\tau[u_i \in V_i]$ ist beste Ankunftszeit an u mit genau i Umstiegen.

Layered Dijkstra (LD)

LD(Layered Graph G, s, τ)

```
1 d_τ[u] ← ∞ for each u ∈ V; d_τ[s_0] ← 0
2 Q.clear(), Q.insert(s_0, 0)
3 while !Q.empty() do
4     u_i ← Q.deleteMin()
5     for all edges e = (u_i, v_j) ∈ E do
6         if d_τ[u_i] + len(e, d_τ[u_i]) < min_{k≤j}{d_τ[v_k]} then
7             d_τ[v_j] ← d_τ[u_i] + len(e, d_τ[u_i])
8             p_τ[v_j] ← u_i
9             Q.update(v_j, d_τ[v_j])
```

Layered Dijkstra (LD)

LD(Layered Graph G, s, τ)

```
1 d_τ[u] ← ∞ for each u ∈ V; d_τ[s_0] ← 0
2 Q.clear(), Q.insert(s_0, 0)
3 while !Q.empty() do
4     u_i ← Q.deleteMin()
5     for all edges e = (u_i, v_j) ∈ E do
6         if d_τ[u_i] + len(e, d_τ[u_i]) < min_{k≤j}{d_τ[v_k], d_τ[t_k]} then
7             d_τ[v_j] ← d_τ[u_i] + len(e, d_τ[u_i])
8             p_τ[v_j] ← u_i
9             Q.update(v_j, d_τ[v_j])
```

Layered Dijkstra: Diskussion

Karlsruher Institut für Technologie

Diskussion

- Schneller und einfacher als MLC
- Speichere Layer *implizit*: Jeder Knoten hat (festes!) Array der Länge $K + 1$ von Labeln $d_\tau[u_0] \dots d_\tau[u_K]$.
- Mehr Umstiege: Verlängere Label-Arrays on-demand (kommt fast nie vor wenn K hinreichend groß)

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 Akunftszeit und # Umstiege

RAPTOR

Anforderungen:

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

RAPTOR

Anforderungen:

- Berechnen von Pareto-sets,
  mindestens Akunftszeit 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 Akunftszeit 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 Akunftszeit 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 Akunftszeit 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: Round-bAsed Public Transit Optimized Router

Runden

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

Runden

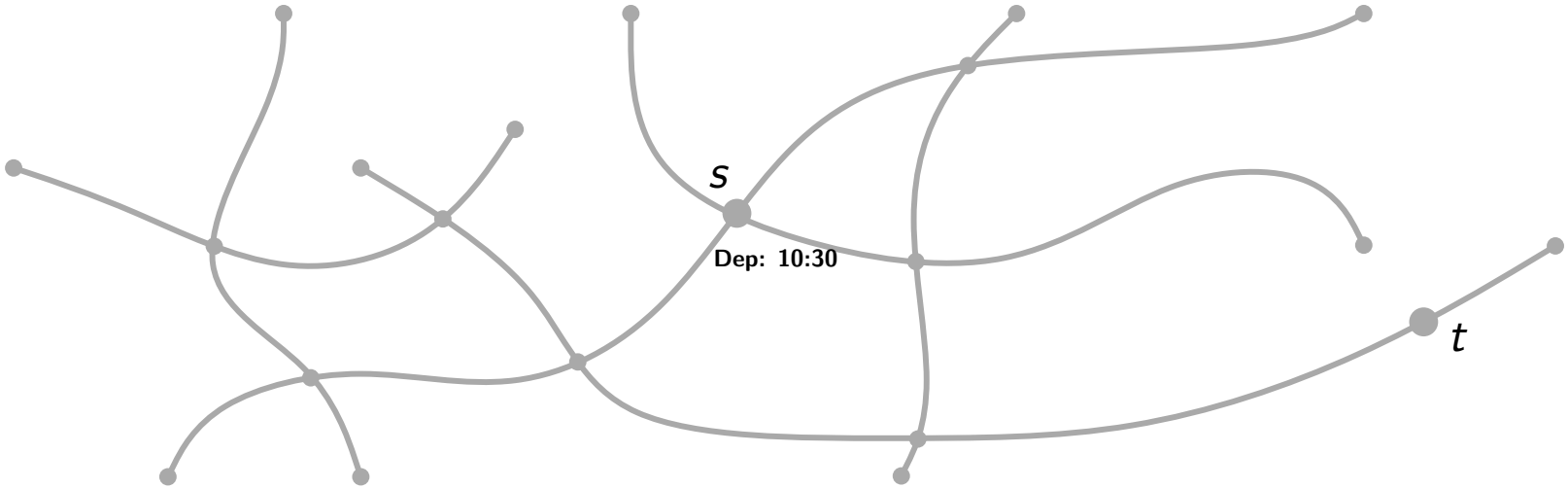
Beobachtung: Wechseln zw. Trips führt immer zu einem Umstieg.

Idee: Eine *Runde* für jeden genommenen Trip.

Runden

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

Idee: Eine *Runde* für jeden genommenen Trip.

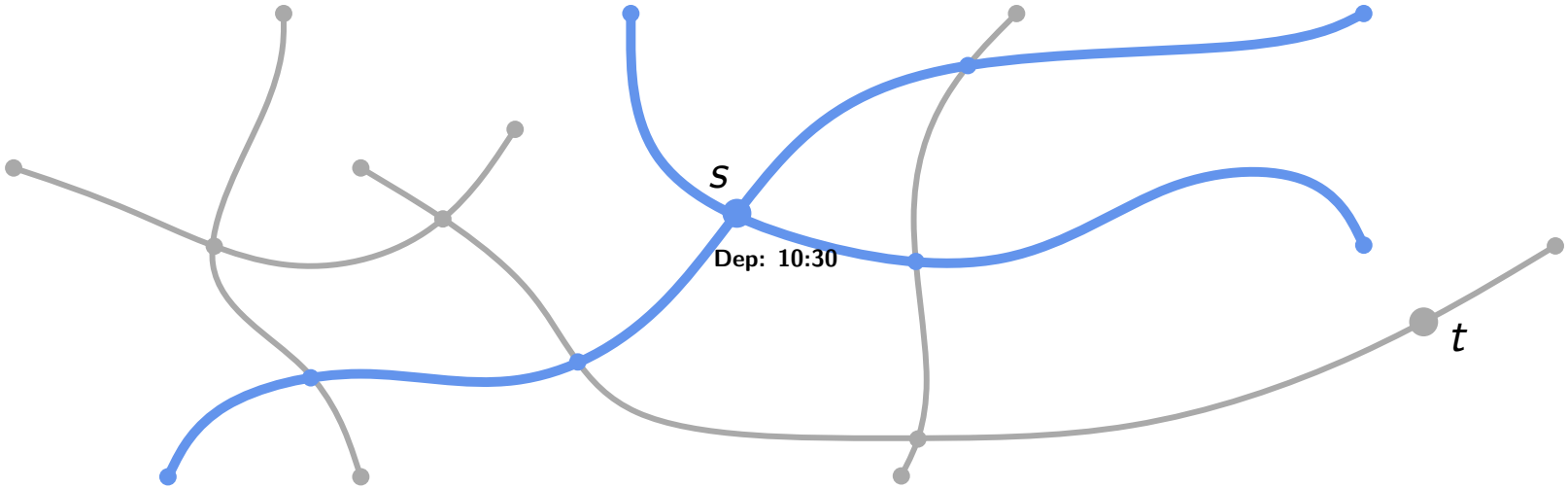


Ansatz: In Runde k werden Ankunftszeiten für k Trips berechnet.

Runden

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

Idee: Eine *Runde* für jeden genommenen Trip.

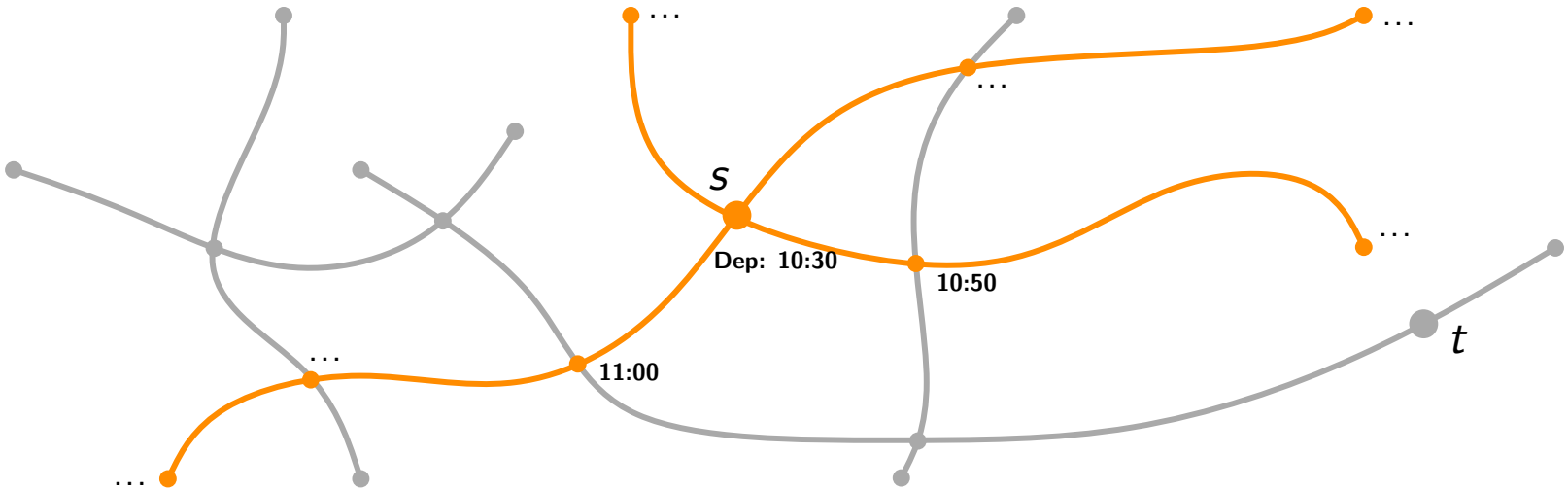


Ansatz: In Runde k werden Ankunftszeiten für k Trips berechnet.

Runden

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

Idee: Eine *Runde* für jeden genommenen Trip.

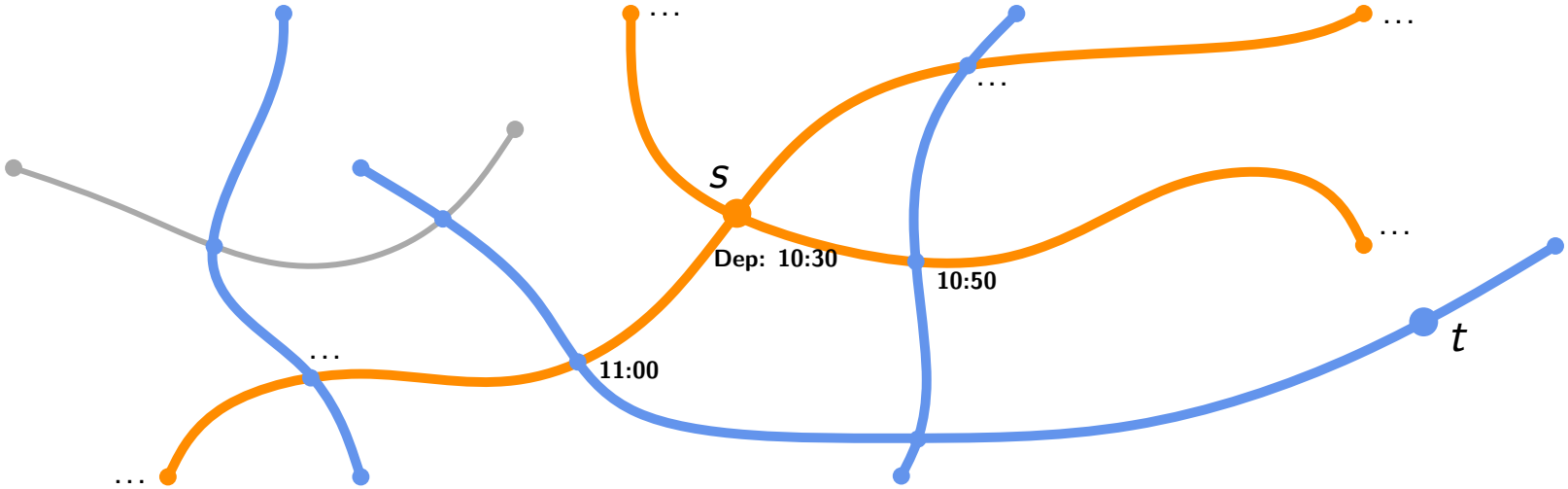


Ansatz: In Runde k werden Ankunftszeiten für k Trips berechnet.

Runden

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

Idee: Eine *Runde* für jeden genommenen Trip.

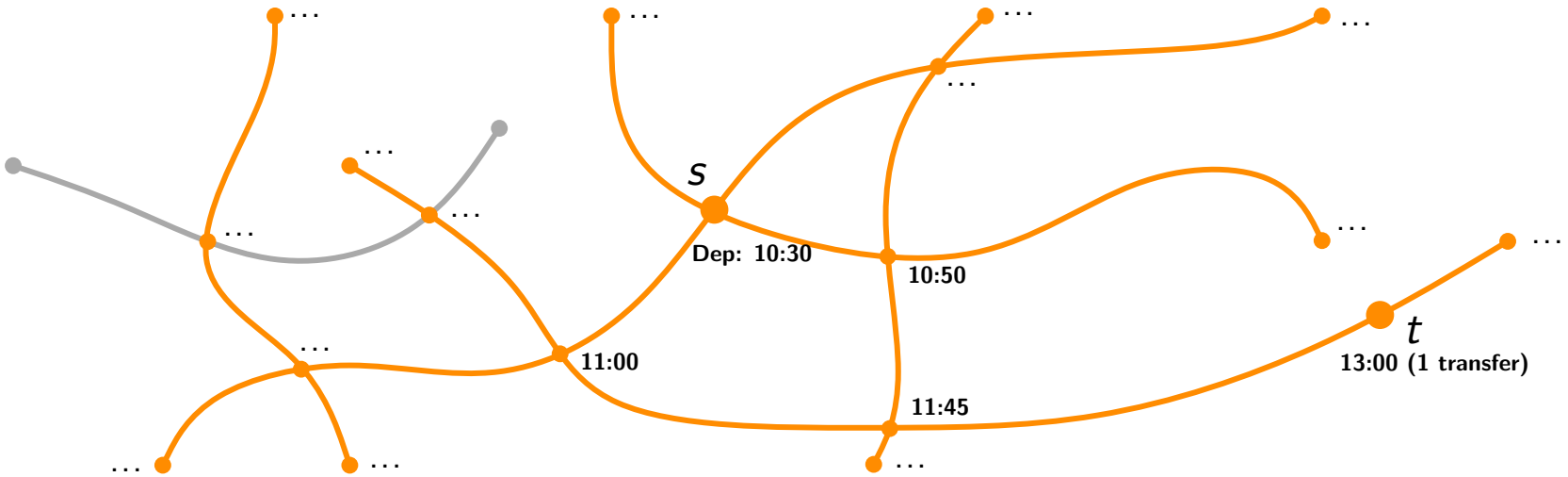


Ansatz: In Runde k werden Ankunftszeiten für k Trips berechnet.

Runden

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

Idee: Eine *Runde* für jeden genommenen Trip.

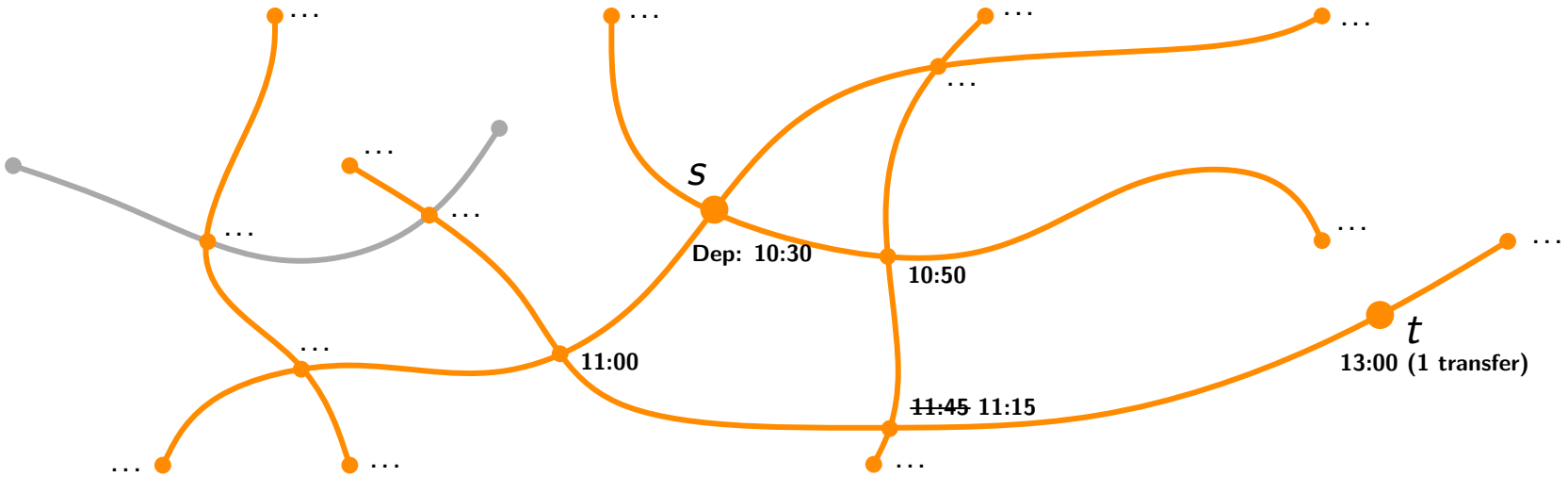


Ansatz: In Runde k werden Ankunftszeiten für k Trips berechnet.

Runden

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

Idee: Eine *Runde* für jeden genommenen Trip.

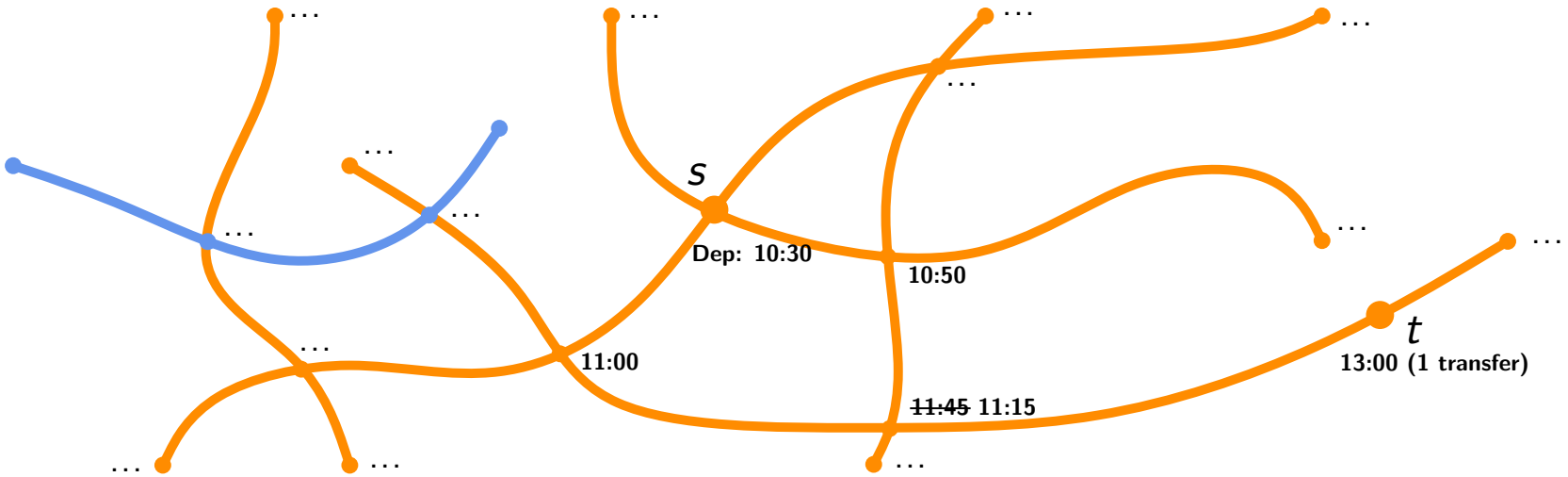


Ansatz: In Runde k werden Ankunftszeiten für k Trips berechnet.

Runden

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

Idee: Eine *Runde* für jeden genommenen Trip.

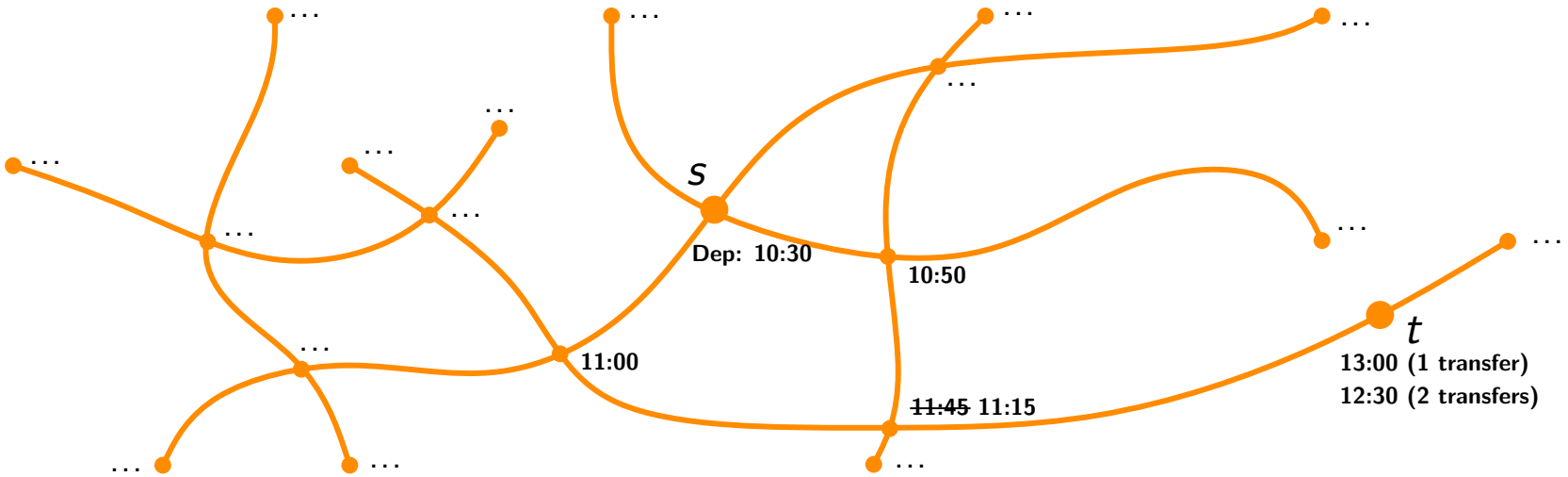


Ansatz: In Runde k werden Ankunftszeiten für k Trips berechnet.

Runden

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

Idee: Eine *Runde* für jeden genommenen Trip.

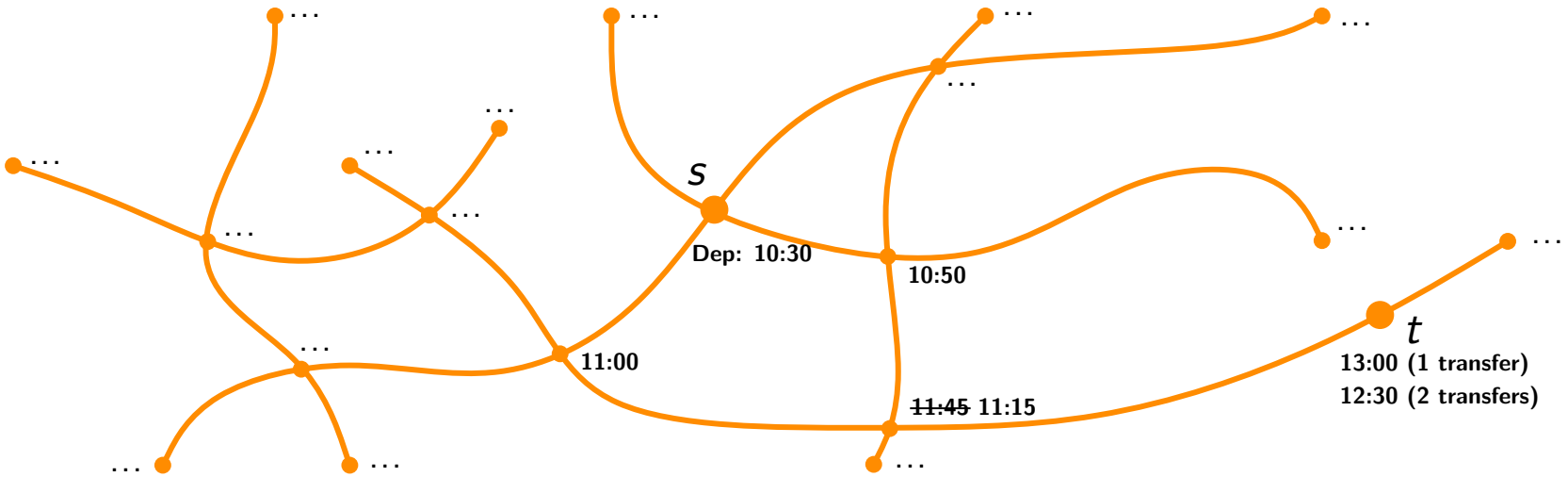


Ansatz: In Runde k werden Ankunftszeiten für k Trips berechnet.

Runden

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

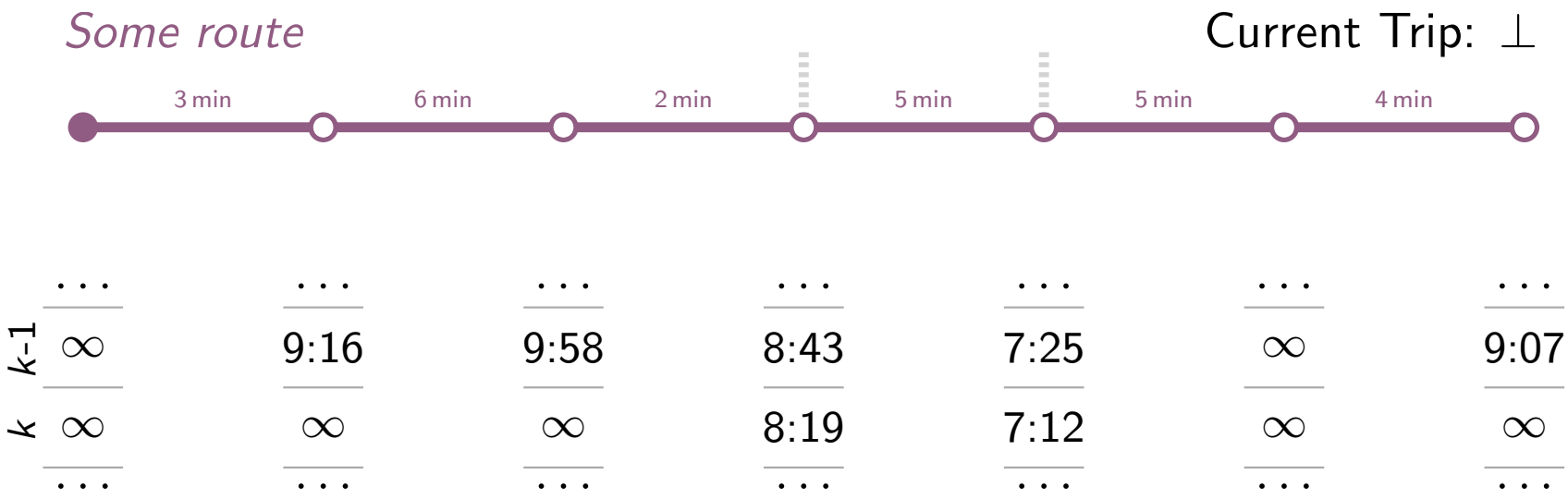
Idee: Eine *Runde* für jeden genommenen Trip.



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

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

Scannen von Routen

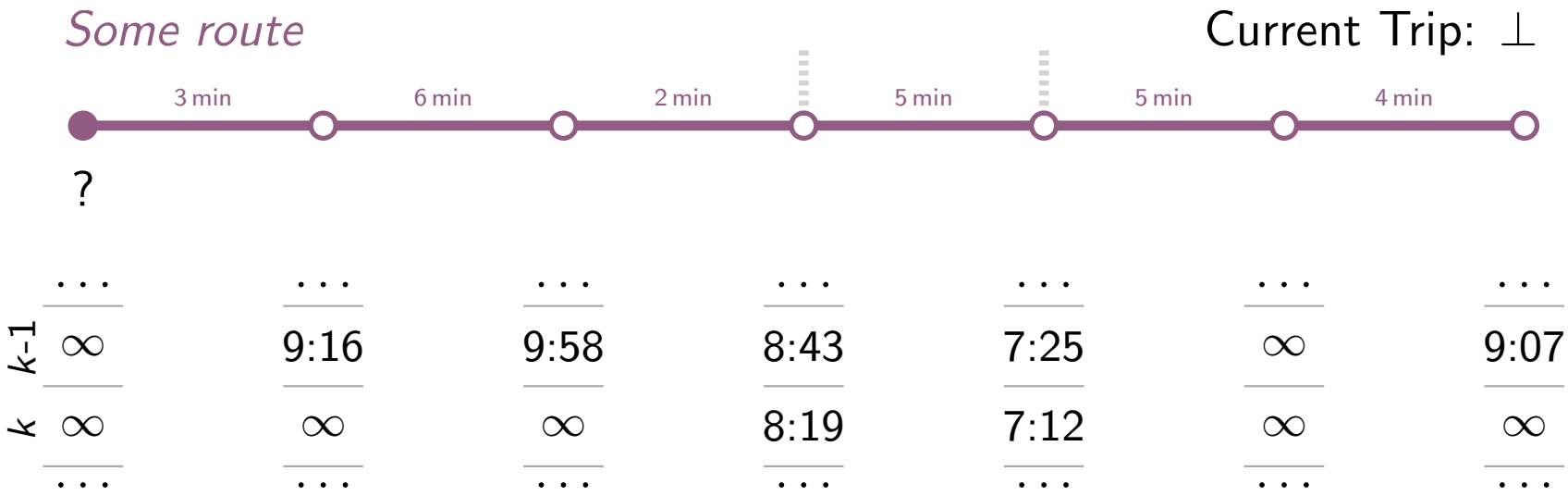


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Aktiver Trip entlang der Route wird stets verbessert.

In Runde k:

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

Scannen von Routen

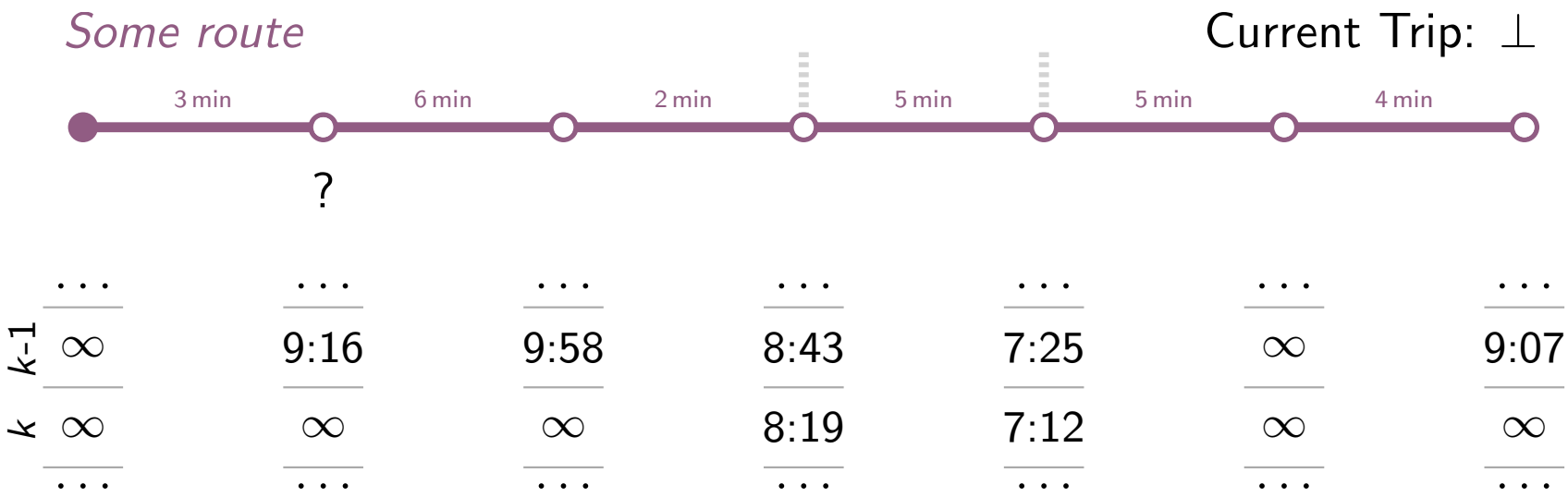


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Aktiver Trip entlang der Route wird stets verbessert.

In Runde k:

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

Scannen von Routen

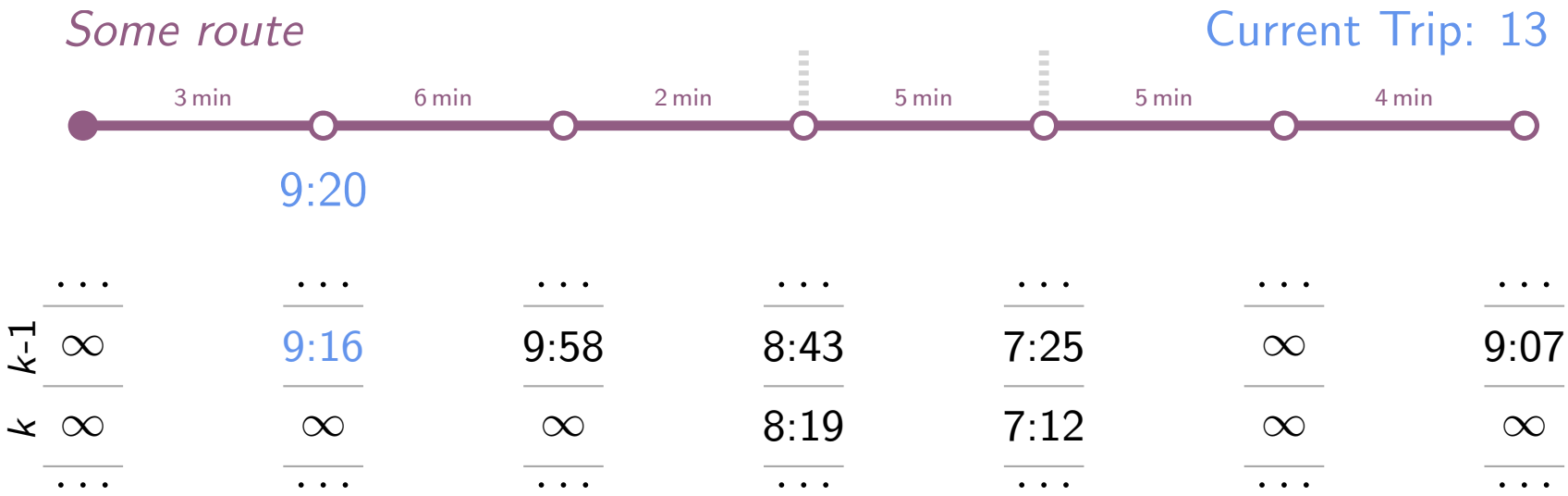


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Aktiver Trip entlang der Route wird stets verbessert.

In Runde k:

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

Scannen von Routen

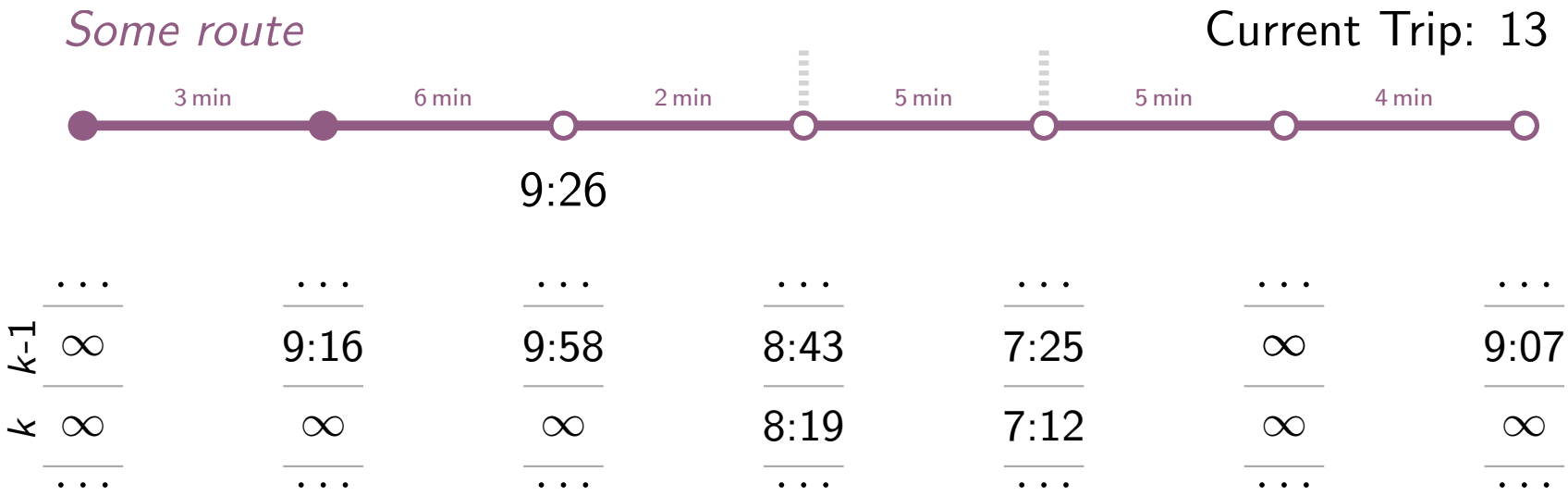


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Aktiver Trip entlang der Route wird stets verbessert.

In Runde k:

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

Scannen von Routen

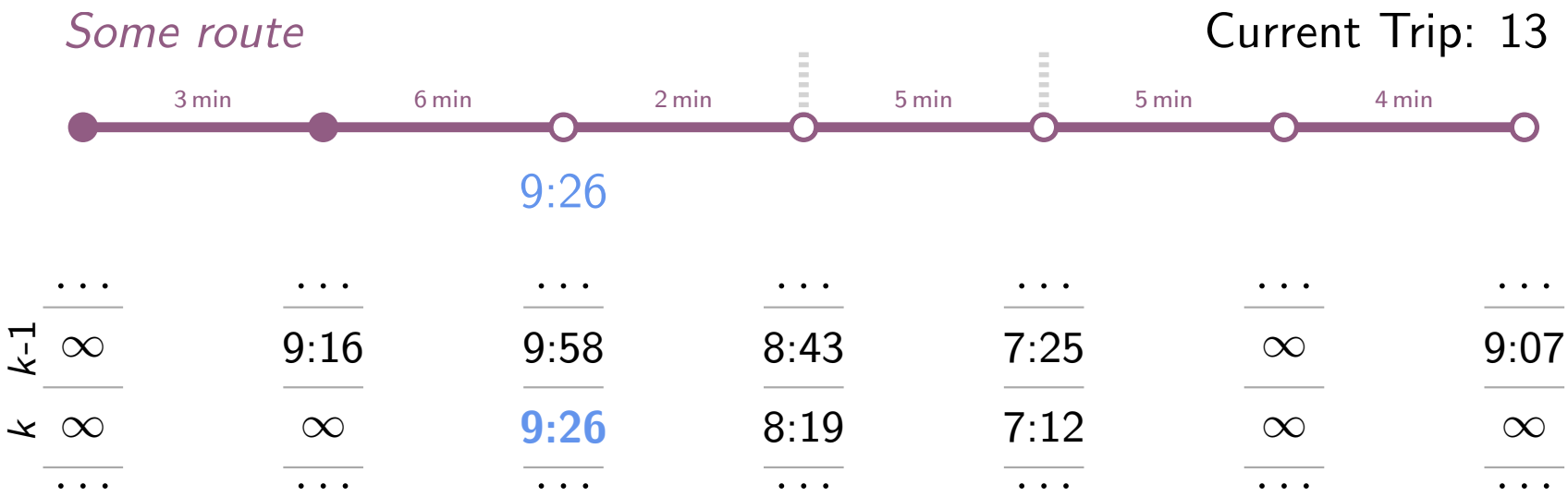


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Aktiver Trip entlang der Route wird stets verbessert.

In Runde k:

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

Scannen von Routen

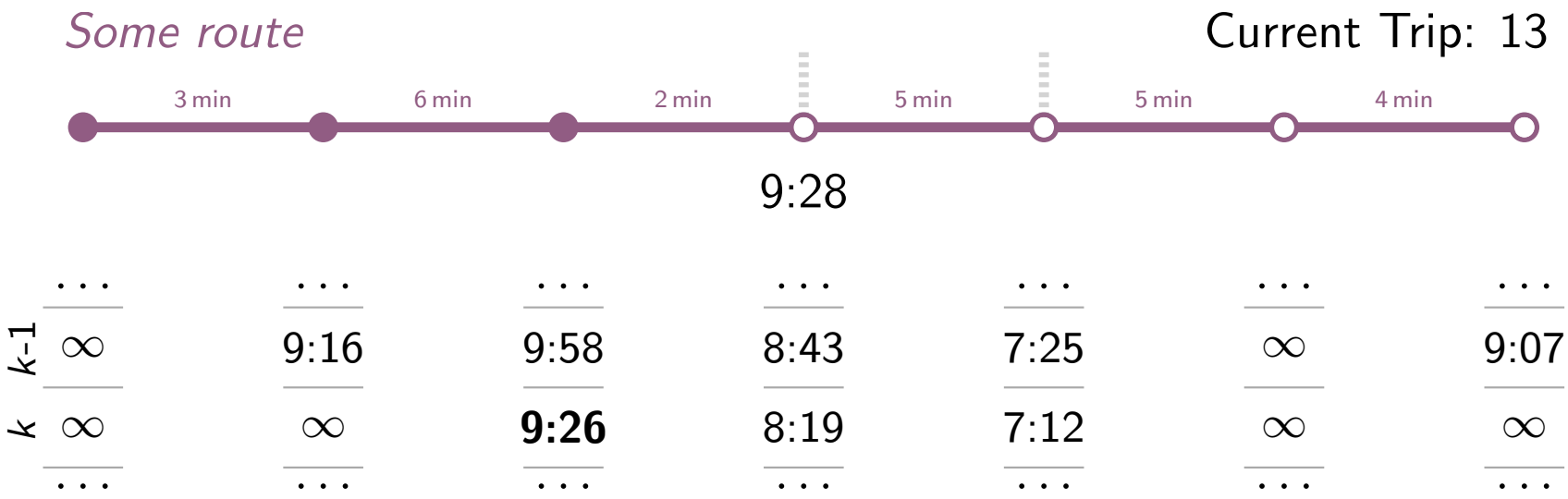


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Aktiver Trip entlang der Route wird stets verbessert.

In Runde k:

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

Scannen von Routen

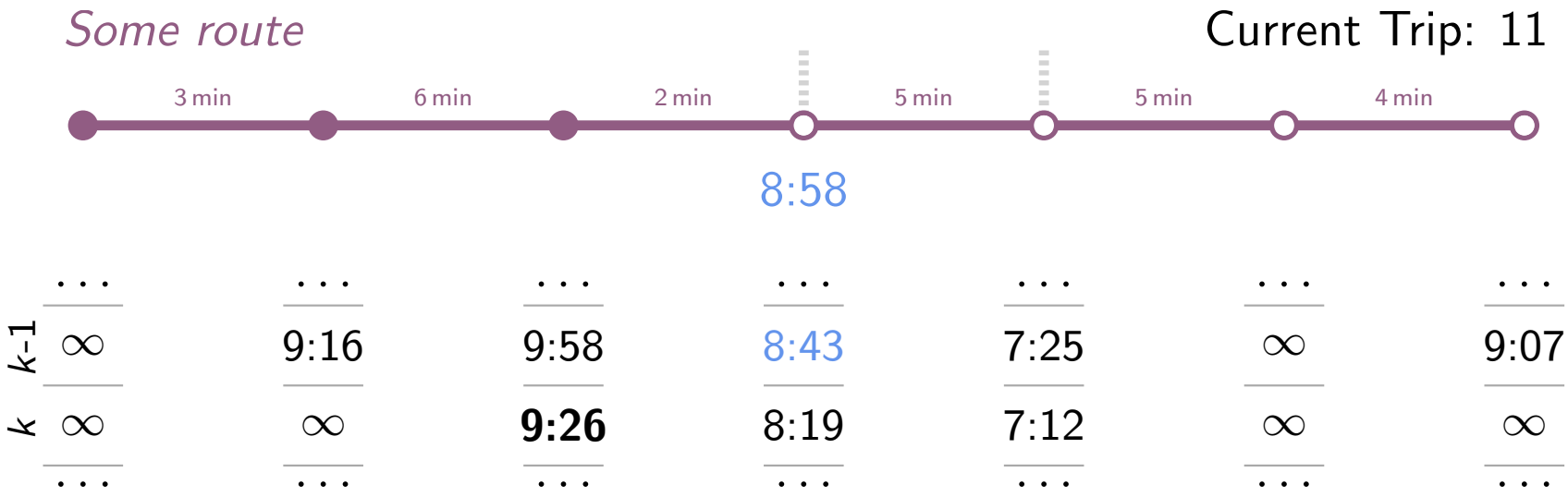


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Aktiver Trip entlang der Route wird stets verbessert.

In Runde k:

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

Scannen von Routen

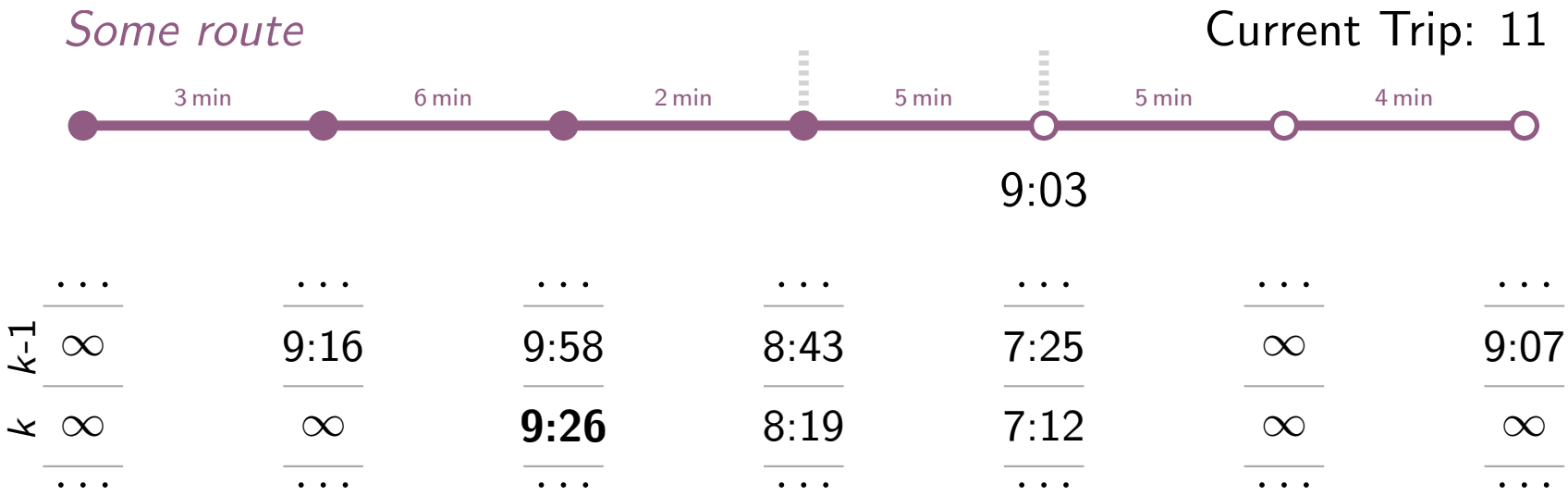


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Aktiver Trip entlang der Route wird stets verbessert.

In Runde k:

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

Scannen von Routen

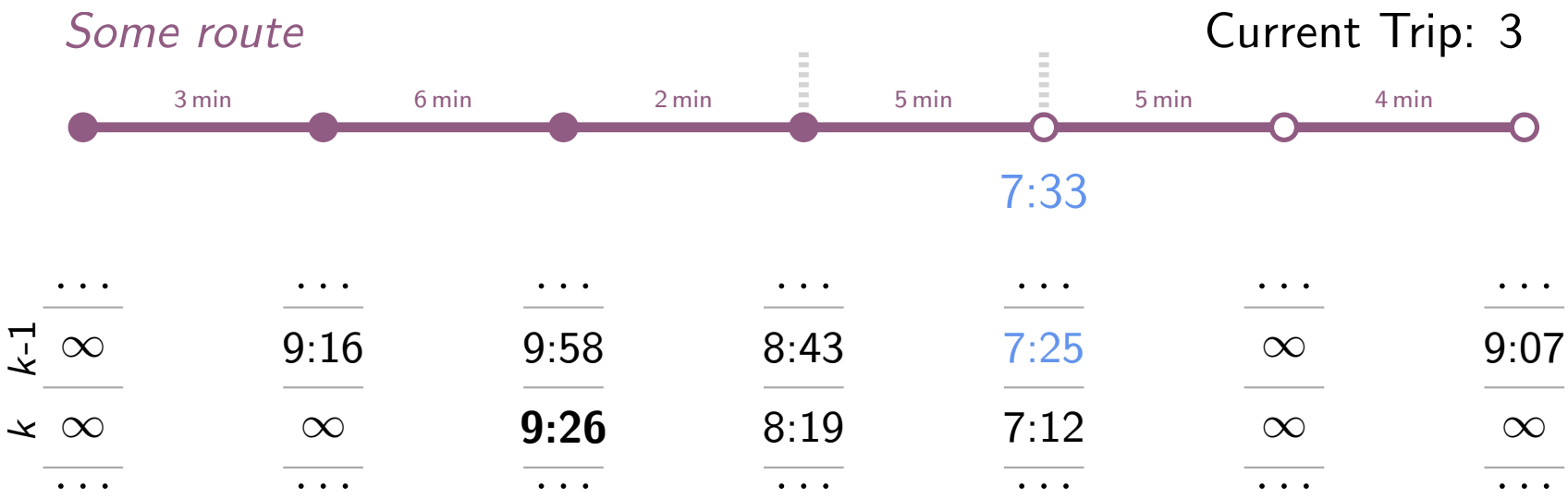


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Aktiver Trip entlang der Route wird stets verbessert.

In Runde k:

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

Scannen von Routen

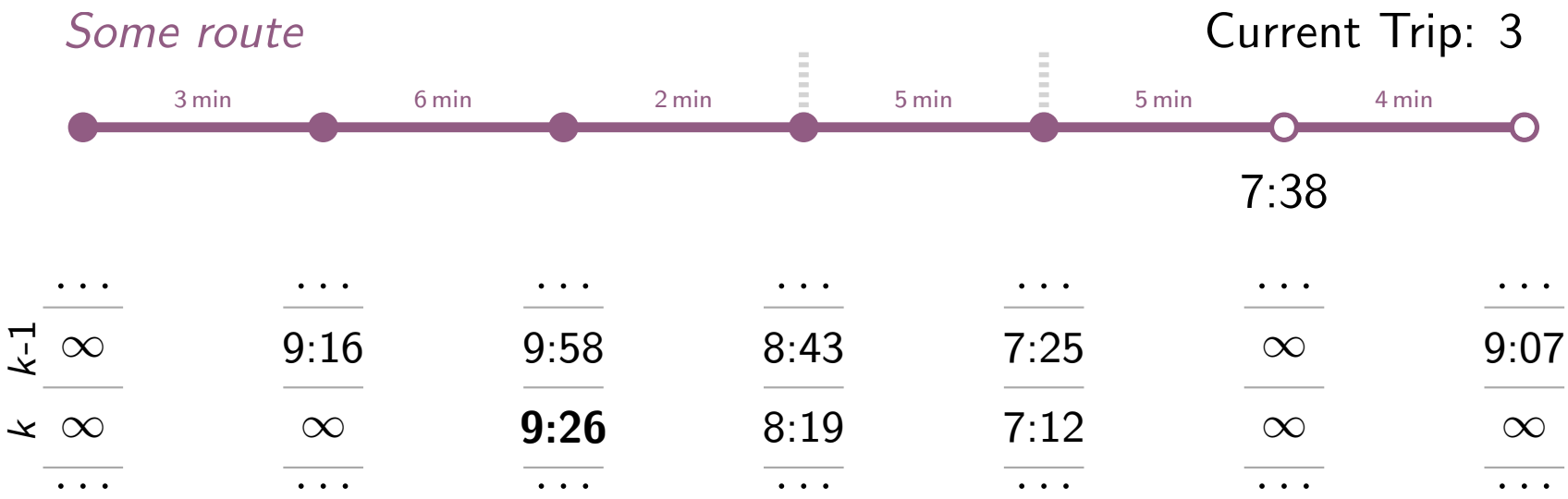


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Aktiver Trip entlang der Route wird stets verbessert.

In Runde k:

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

Scannen von Routen

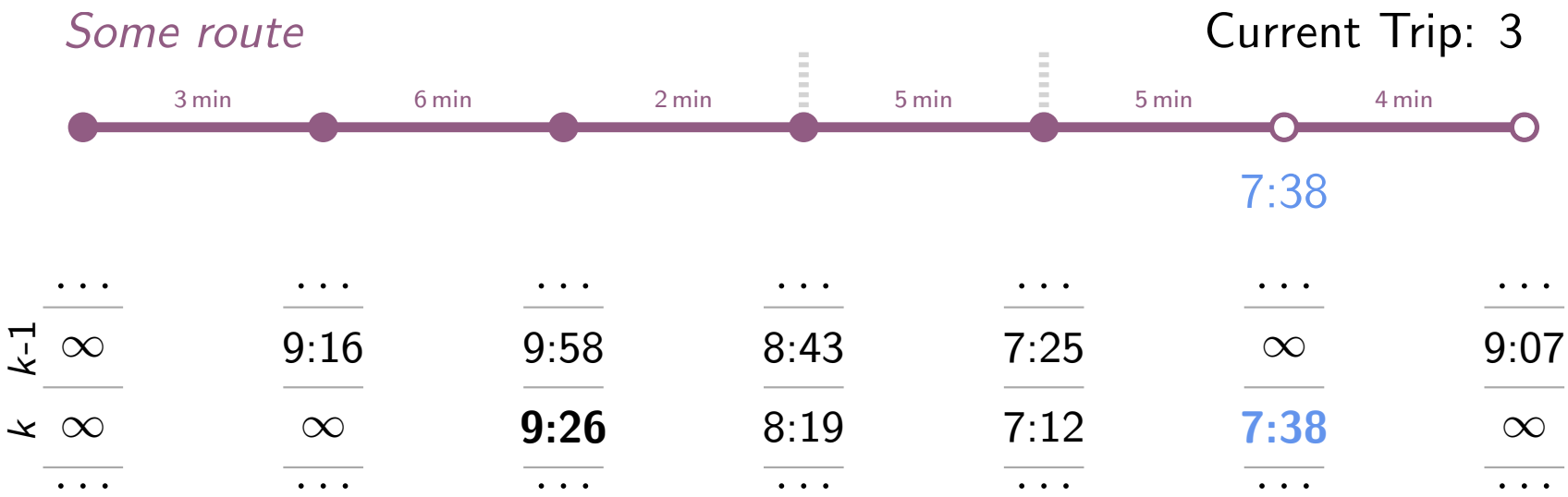


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Aktiver Trip entlang der Route wird stets verbessert.

In Runde k:

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

Scannen von Routen

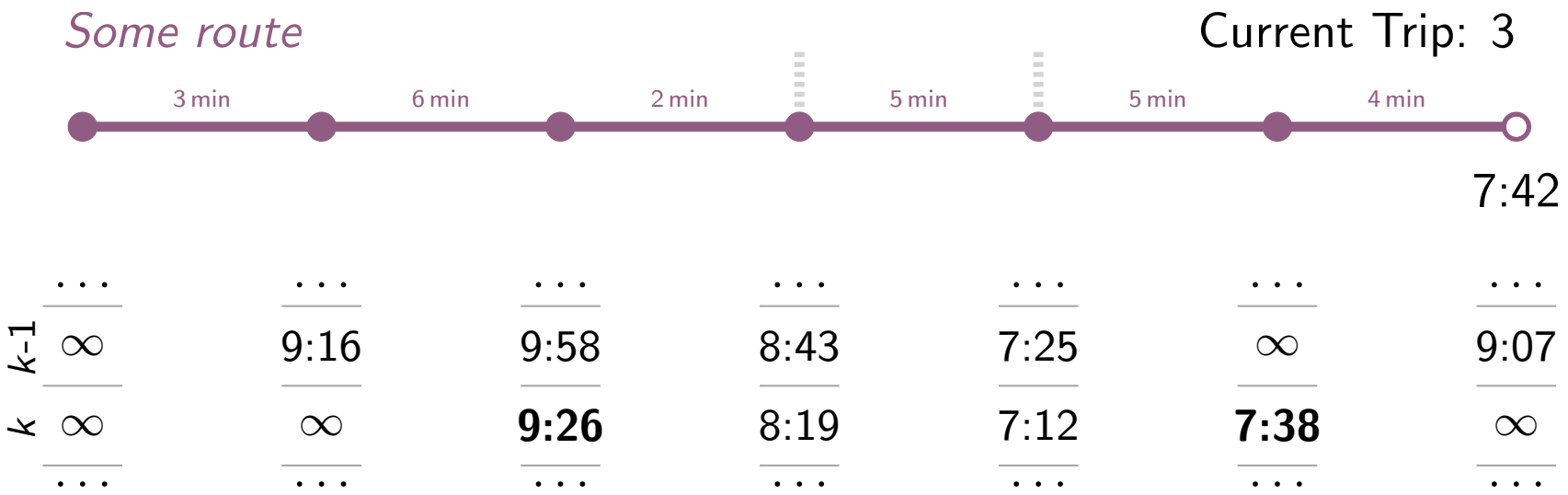


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Aktiver Trip entlang der Route wird stets verbessert.

In Runde k:

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

Scannen von Routen

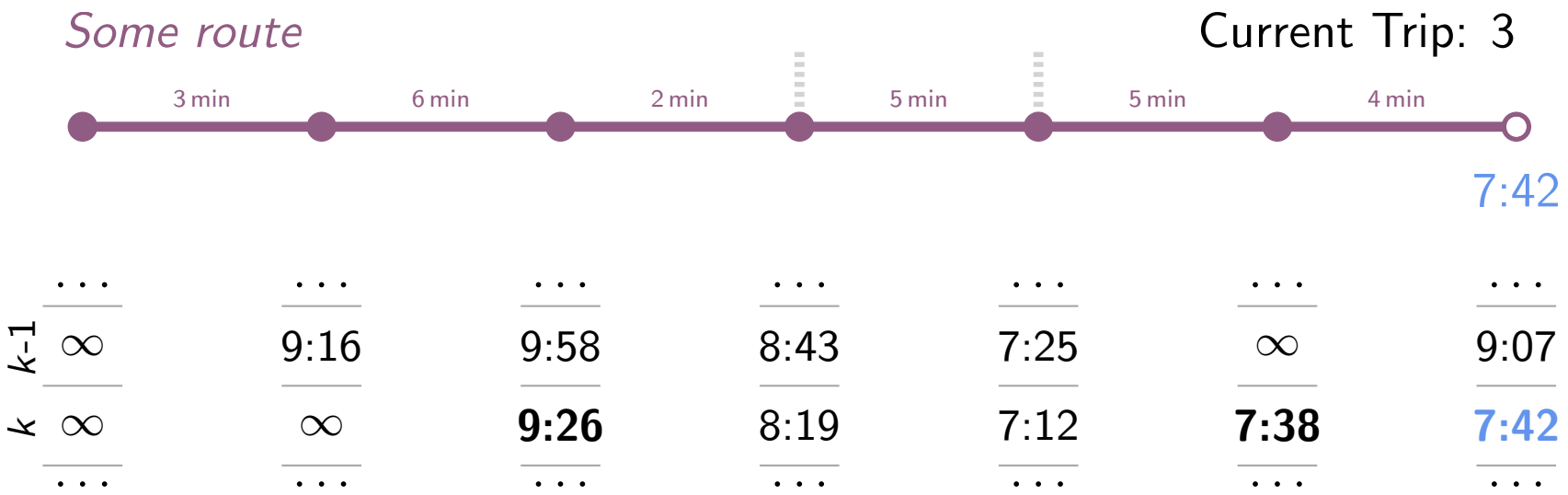


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Aktiver Trip entlang der Route wird stets verbessert.

In Runde k:

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

Scannen von Routen

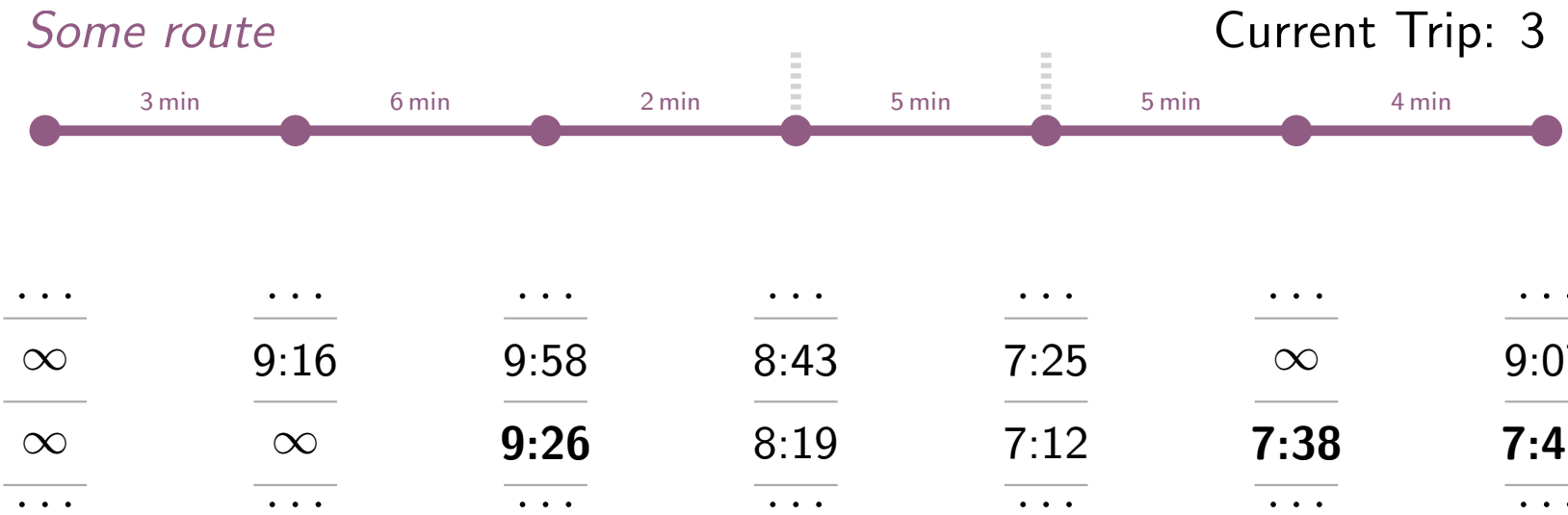


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Aktiver Trip entlang der Route wird stets verbessert.

In Runde k:

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

Scannen von Routen

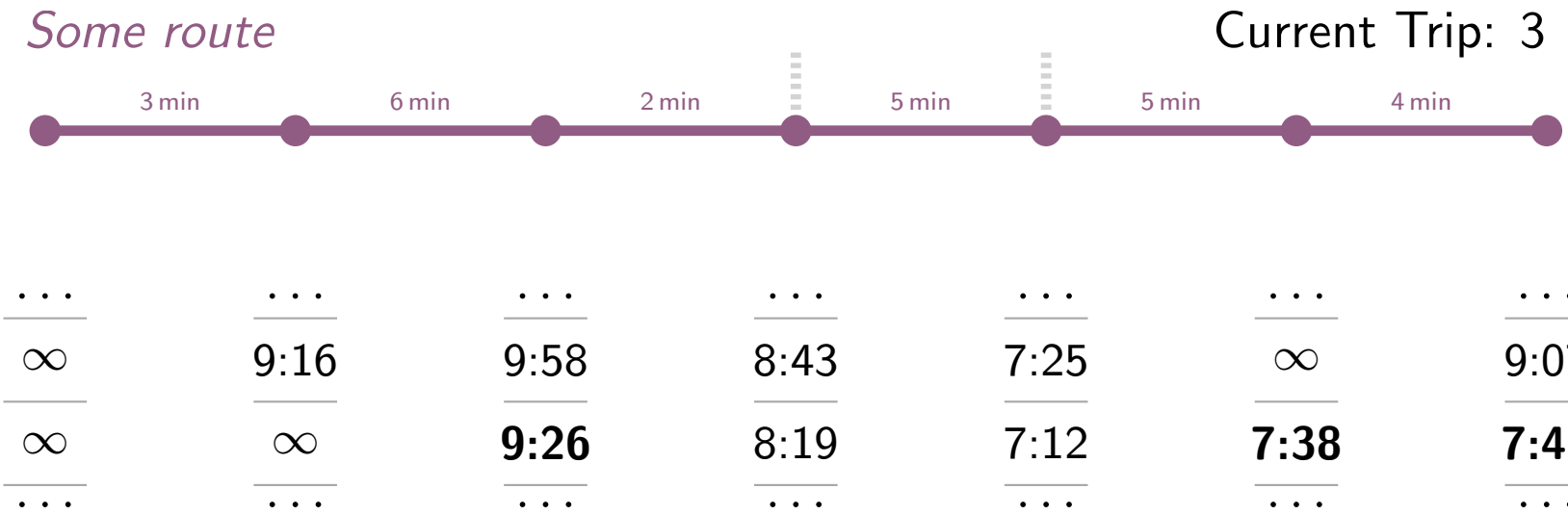


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Aktiver Trip entlang der Route wird stets verbessert.

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.

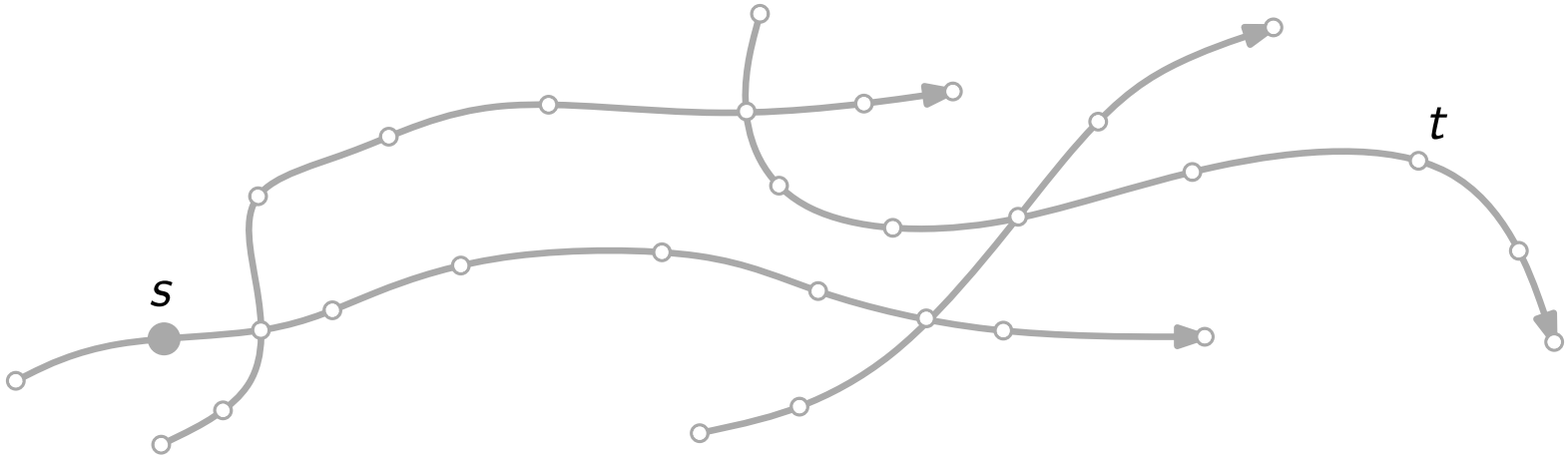
In Runde k:

- 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.

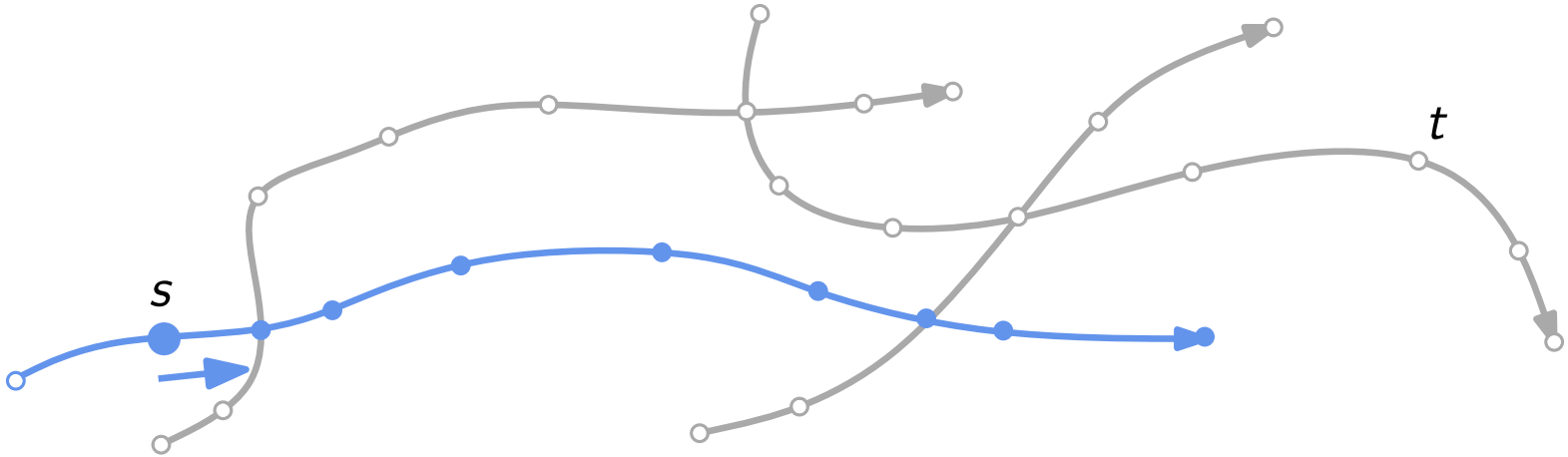


Markieren und Pruning

- 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.

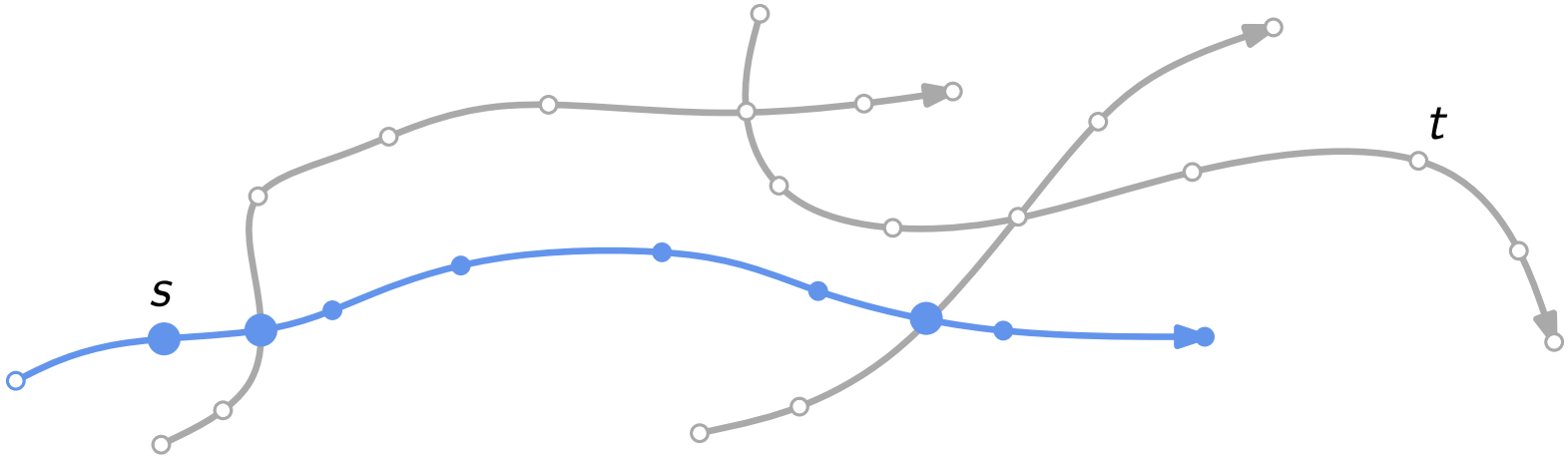


Markieren und Pruning

- 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.

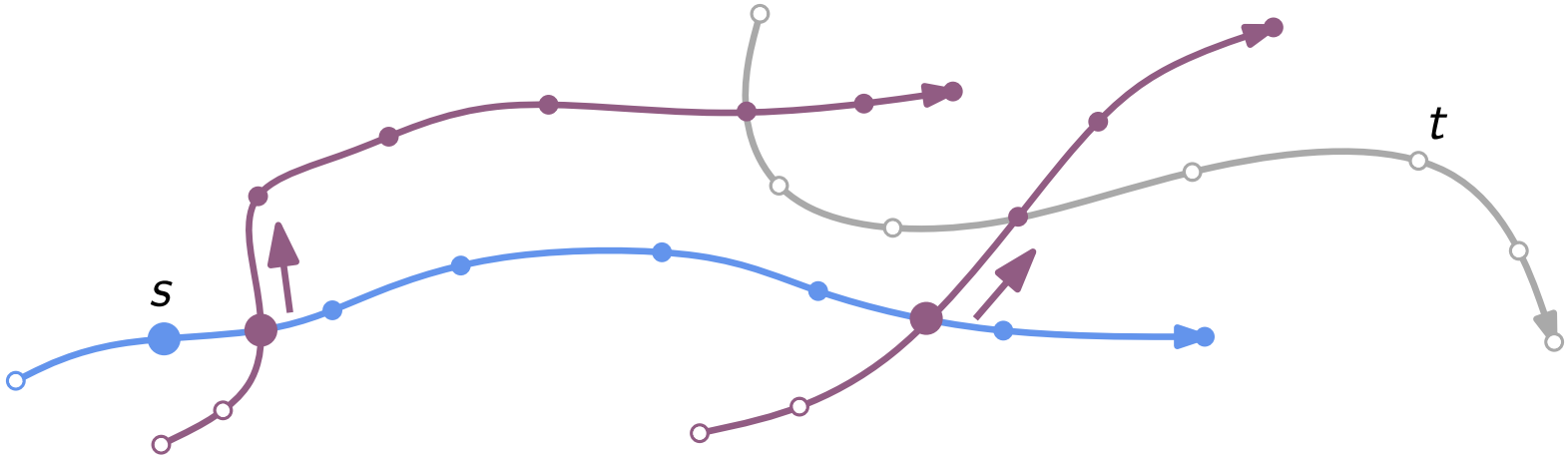


Markieren und Pruning

- 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.

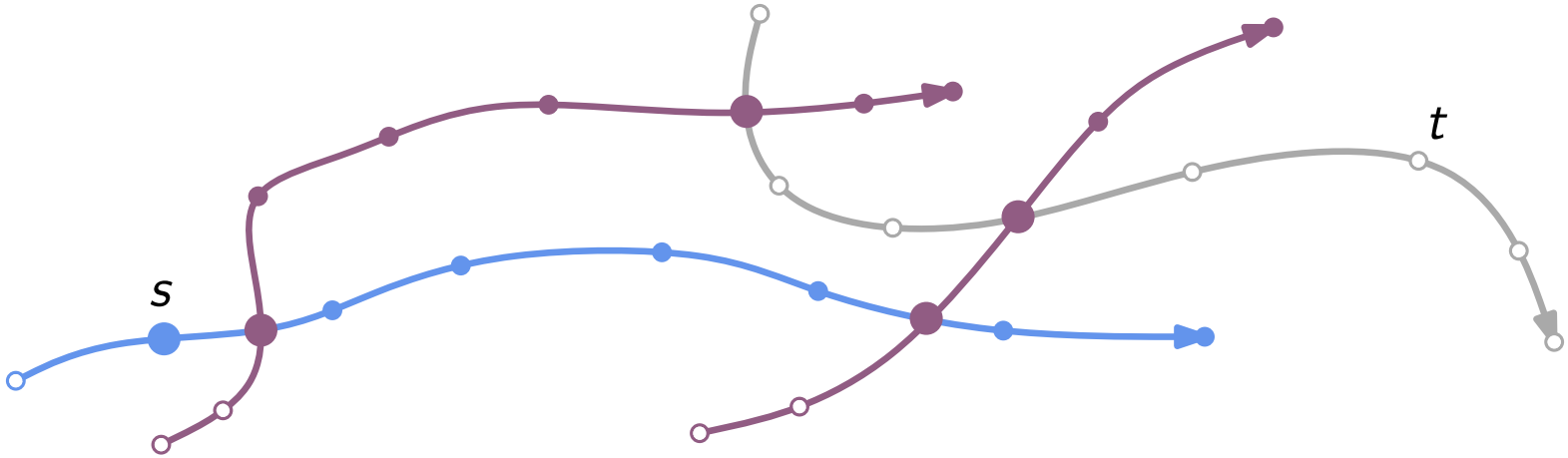


Markieren und Pruning

- 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.

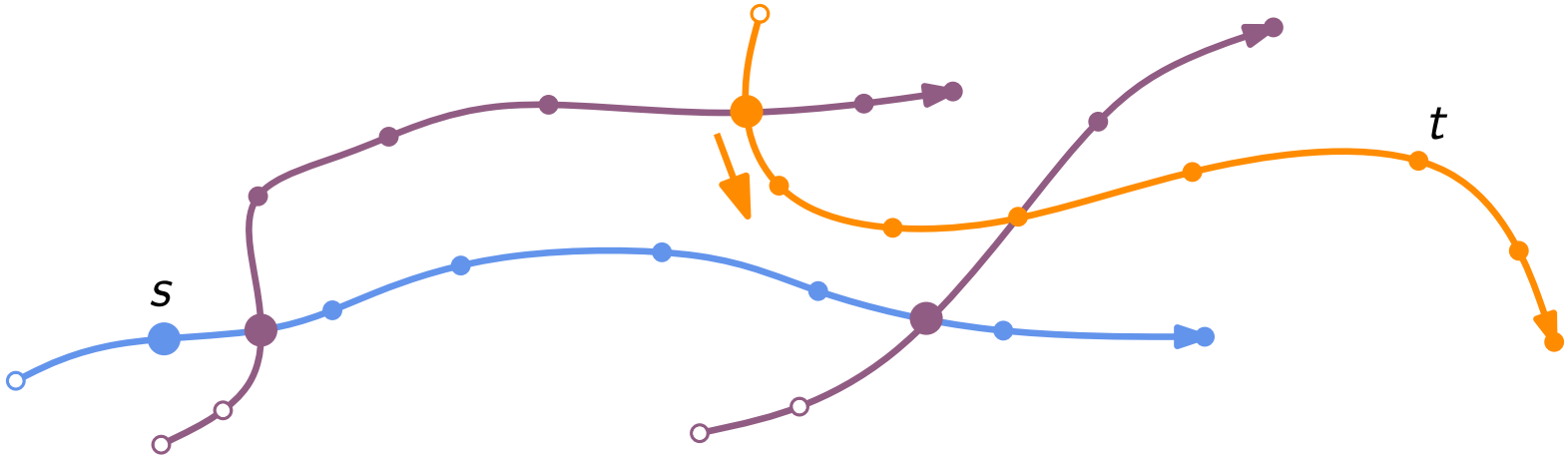


Markieren und Pruning

- 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.

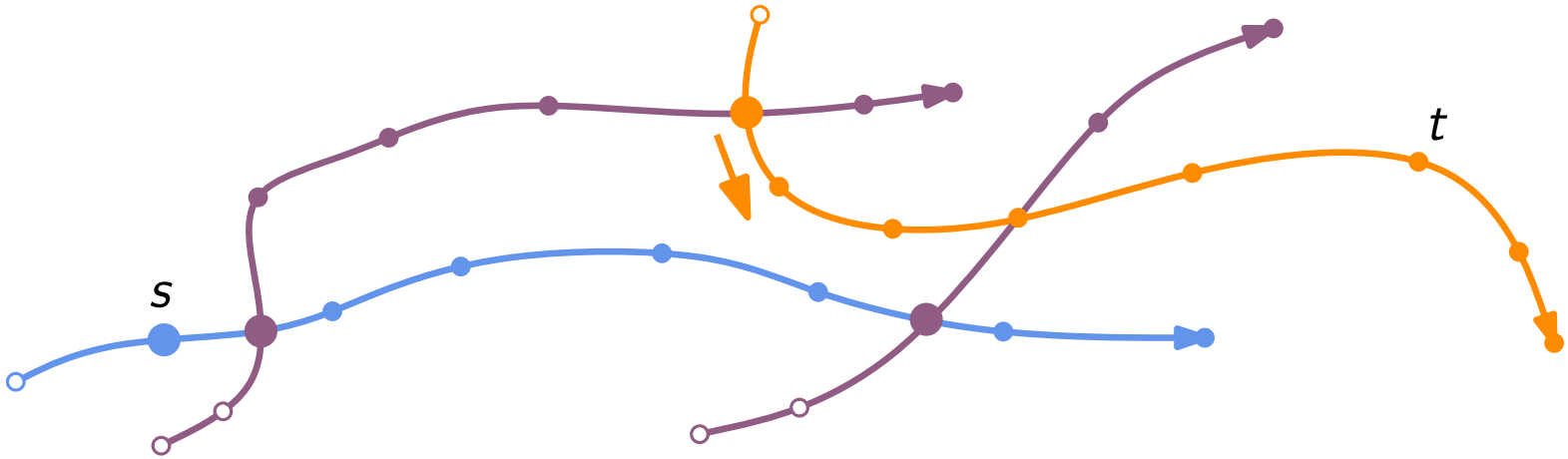


Markieren und Pruning

- 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.



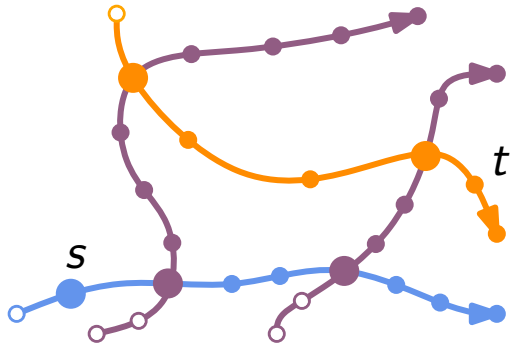
Markieren und Pruning

- 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 die beste Ankunftszeit am Ziel verbessern.

RAPTOR: Übersicht

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

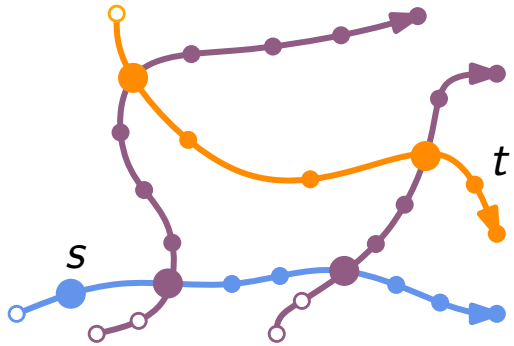
1. Wähle erreichte Routen aus letzter Runde
2. Scanne diese Routen
3. Relaxiere Fußwege



RAPTOR: Übersicht

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

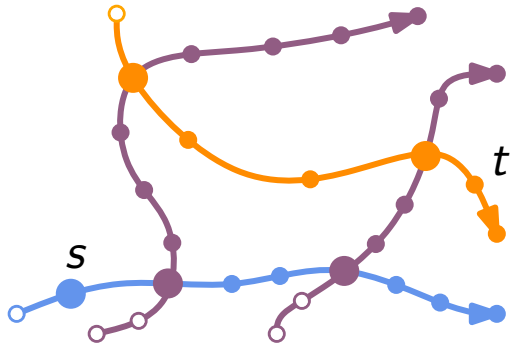
1. Wähle erreichte Routen aus letzter Runde
2. Scanne diese Routen
3. Relaxiere Fußwege



RAPTOR: Übersicht

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

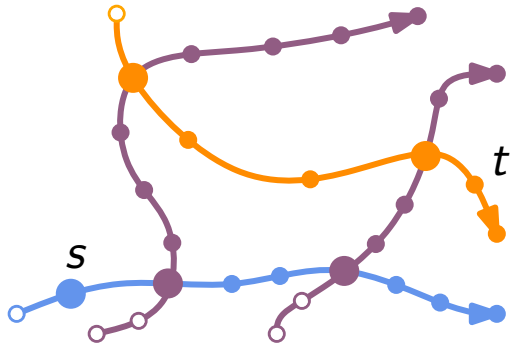
1. Wähle erreichte Routen aus letzter Runde
2. Scanne diese Routen
3. Relaxiere Fußwege



RAPTOR: Übersicht

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

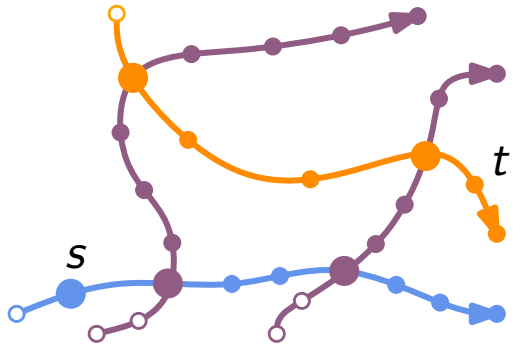
1. Wähle erreichte Routen aus letzter Runde
2. Scanne diese Routen
3. Relaxiere Fußwege



RAPTOR: Übersicht

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

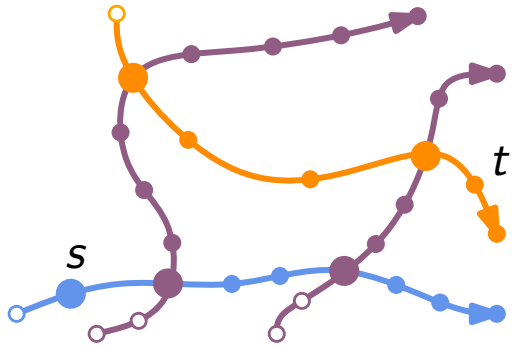
1. Wähle erreichte Routen aus letzter Runde
2. Scanne diese Routen
3. Relaxiere Fußwege



RAPTOR: Übersicht

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

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



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.

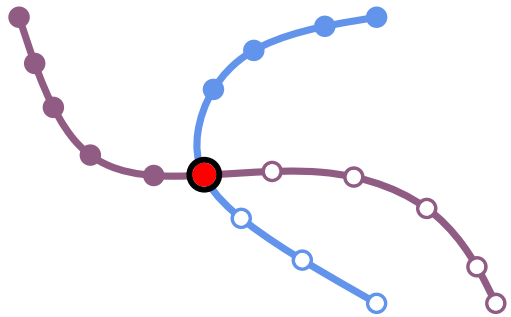
Parallelisierung

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.



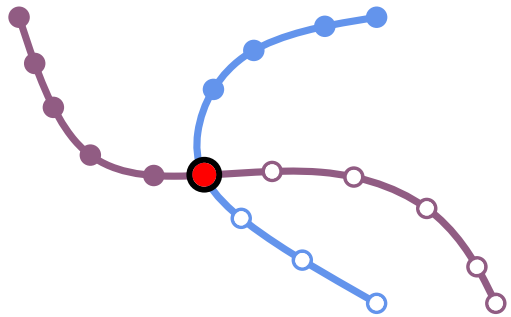
Parallelisierung

Beobachtung: Routen werden in bel. Reihenfolge gescannt.

Verteile Routen auf verschiedene CPU Kerne; Scanne parallel.

Vermeiden von Race-Conditions

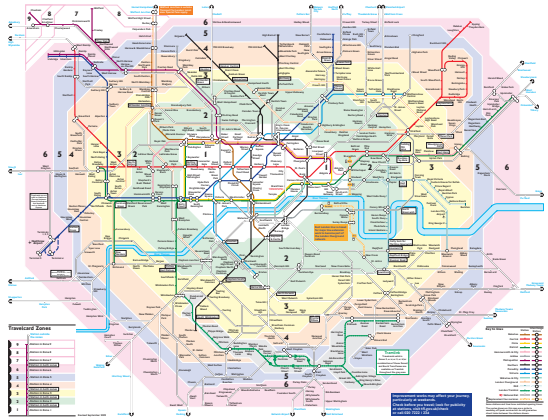
- Lock auf Schreiben von Labels (teuer).
- Synchronisiere Labels nach jeder Runde.
- Sicherstellen dass nur „unabhängige“ Routen gleichzeitig gescannt werden.



Andere Kriterien: Übersicht

Mögliche Erweiterungen

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

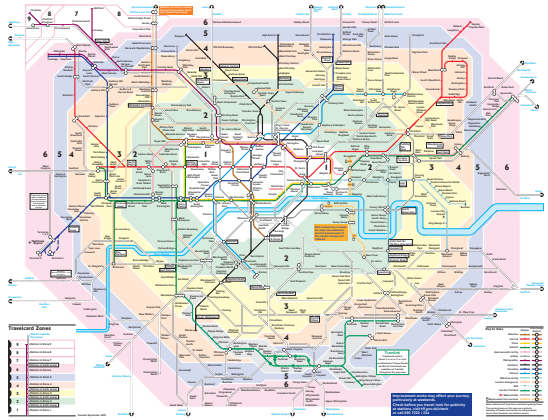


Performance hängt von Anzahl *nichtdominierter* Routen ab.

Andere Kriterien: Übersicht

Mögliche Erweiterungen

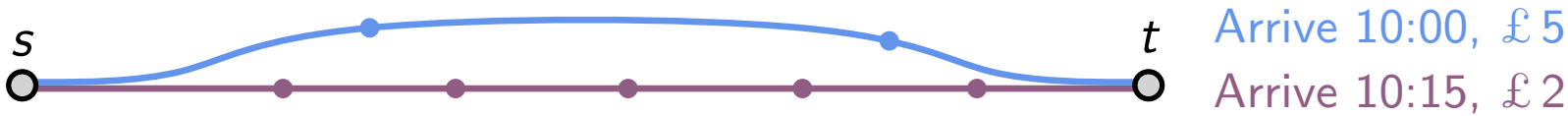
- Intervallanfragen (Profil-Anfragen),
  Flexible Abfahrtszeiten.
- Tarifzonen,
  Längere Routen könnten billiger sein.
- Umstiegssicherheit,
  Routen könnten knappe Umstiege haben.
- ...



Performance hängt von Anzahl *nichtdominierter* Routen ab.

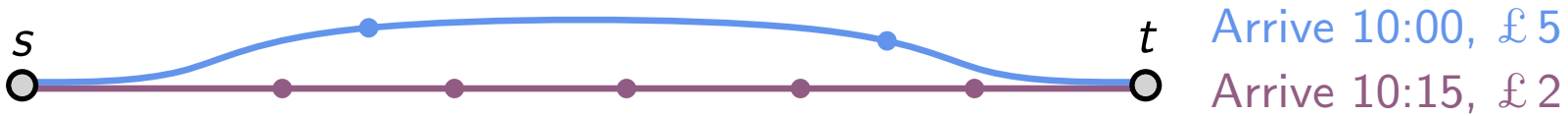
More Criteria: McRAPTOR

Ziel: Erweitern von RAPTOR auf zusätzliche Kriterien.



More Criteria: McRAPTOR

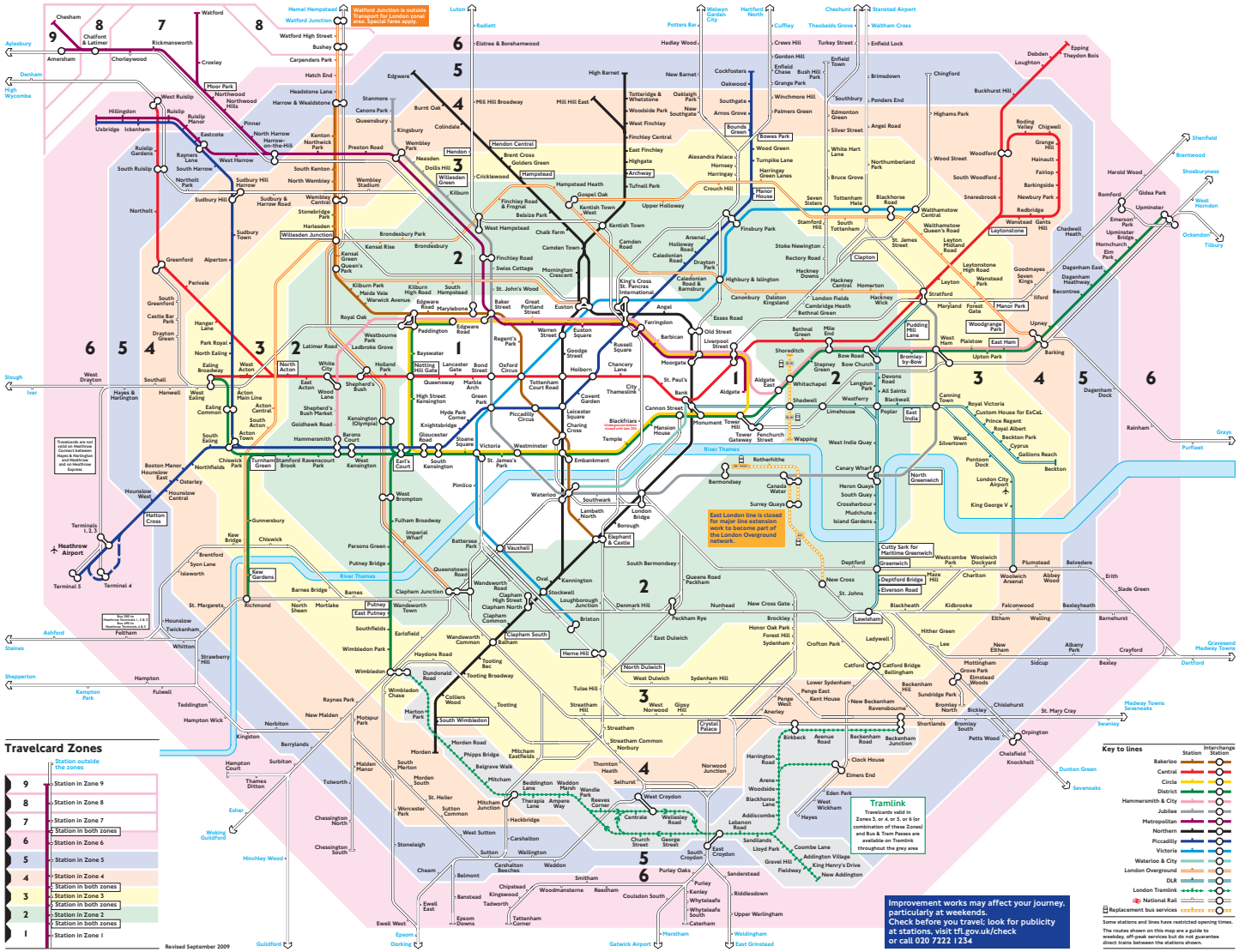
Ziel: Erweitern von RAPTOR auf zusätzliche Kriterien.



Ansatz

- Labels haven 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

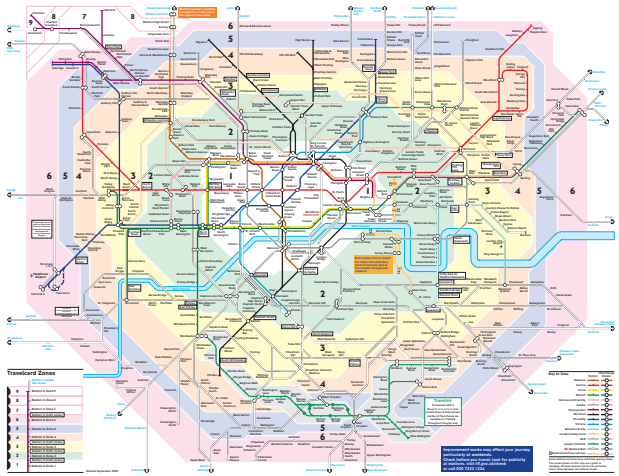
McRAPTOR Beispiel: Tarifzonen

Tarifzonen einbauen

- Direkte Preise (£) können nicht handhabbar.

⇒ Berechne alle Kombinationen von Tarifzonen,

- und filtere im Postprocessing.



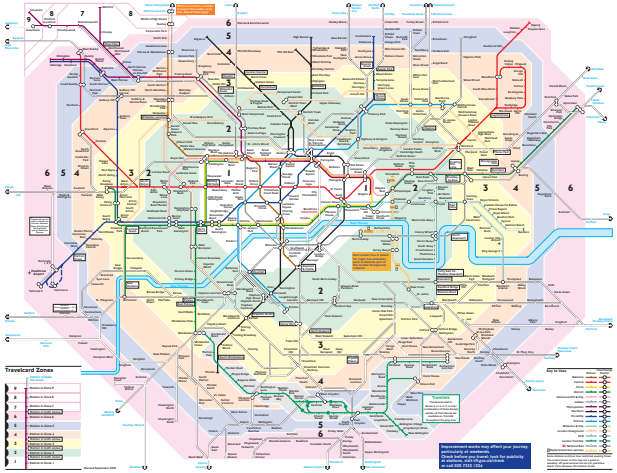
McRAPTOR Beispiel: Tarifzonen

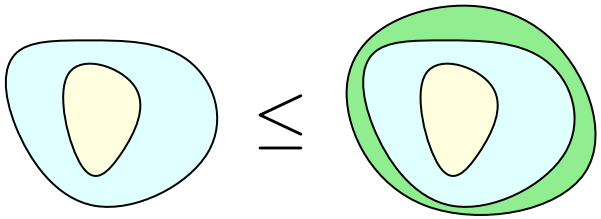
Tarifzonen einbauen

- Direkte Preise (£) können 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.

Intervallanfragen: rRAPTOR

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

Intervallanfragen: rRAPTOR

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

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

Intervallanfragen: rRAPTOR

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

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

Effizienterer Ansatz: rRAPTOR

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

Intervallanfragen: rRAPTOR

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

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

Effizienterer Ansatz: rRAPTOR

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

Prunt implizit Routen die früher abfahren und später ankommen.

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)

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)

- RAPTOR schneller als Dijkstra-basierte Ansätze.

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)

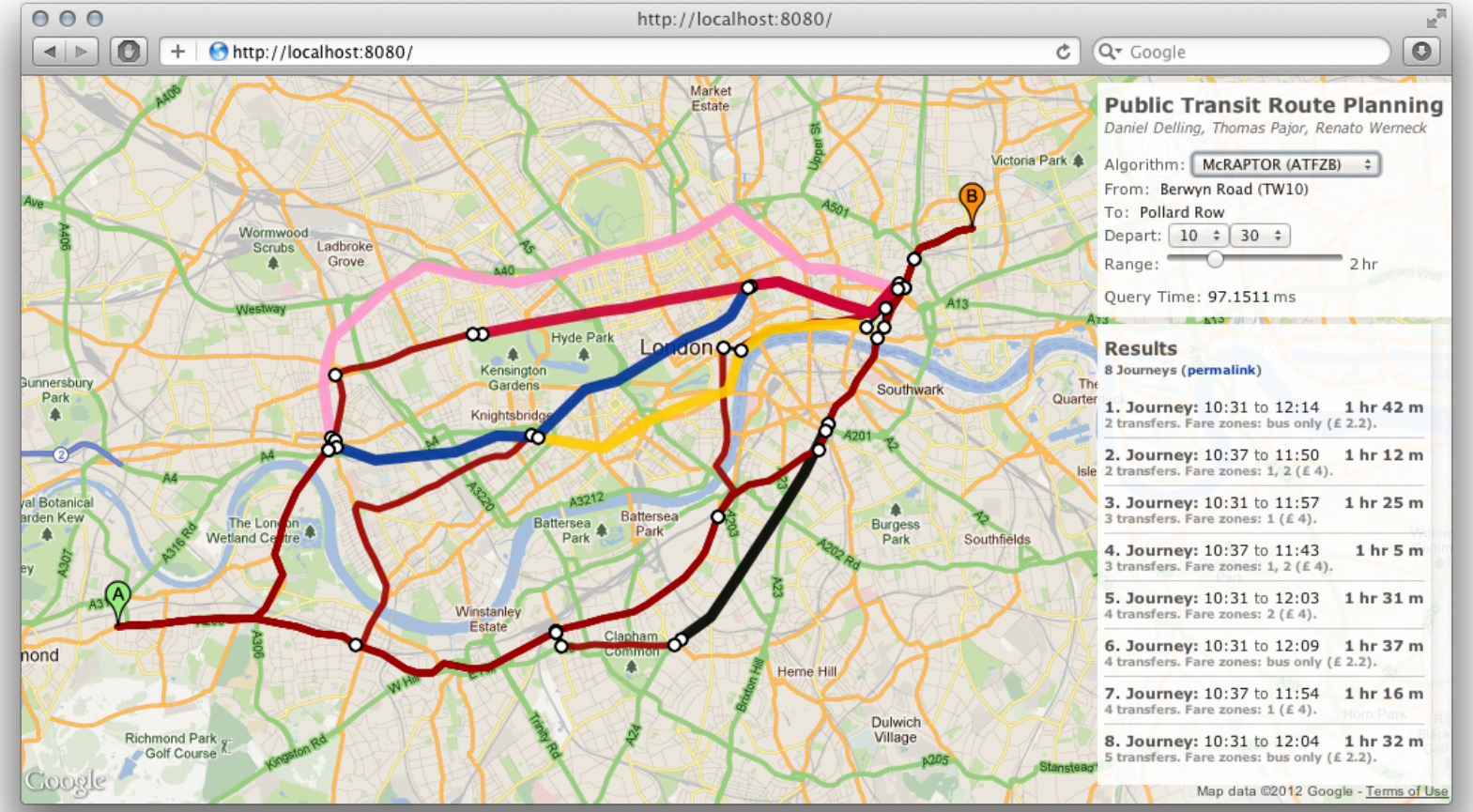
Parallelisierung

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.

Demo


http://localhost:8080/
http://localhost:8080/
Google
Public Transit Route Planning
Daniel Delling, Thomas Pajor, Renato Werneck
Algorithm: McRAPTOR (ATFZB)
From: Berwyn Road (TW10)
To: Pollard Row
Depart: 10 30
Range: 2 hr
Query Time: 97.1511 ms
Results
8 Journeys (permalink)
1. Journey: 10:31 to 12:14 1 hr 42 m
2 transfers. Fare zones: bus only (£ 2.2).
2. Journey: 10:37 to 11:50 1 hr 12 m
2 transfers. Fare zones: 1, 2 (£ 4).
3. Journey: 10:31 to 11:57 1 hr 25 m
3 transfers. Fare zones: 1 (£ 4).
4. Journey: 10:37 to 11:43 1 hr 5 m
3 transfers. Fare zones: 1, 2 (£ 4).
5. Journey: 10:31 to 12:03 1 hr 31 m
4 transfers. Fare zones: 2 (£ 4).
6. Journey: 10:31 to 12:09 1 hr 37 m
4 transfers. Fare zones: bus only (£ 2.2).
7. Journey: 10:37 to 11:54 1 hr 16 m
4 transfers. Fare zones: 1 (£ 4).
8. Journey: 10:31 to 12:04 1 hr 32 m
5 transfers. Fare zones: bus only (£ 2.2).
Map data ©2012 Google - Terms of Use
Google
London
Hyde Park
Kensington Gardens
Knightsbridge
Southwark
Battersea Park
Clapham Common
Herne Hill
Dulwich Village
Winstanley Estate
Wormwood Scrubs
Ladbroke Grove
Westway
Victoria Park
Market Estate
Richmond Park
Golf Course
Kingston Rd
Trinity Rd
Brixton Hill
Burgess Park
Southfields
A4
A40
A13
A201
A3212
A3220
A202 Rd
A316 Rd
A306
A24
A205

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.

Nächste Termine

Karlsruher Institut für Technologie

Montag, 8.7.2013

Mittwoch, 10.7.2013