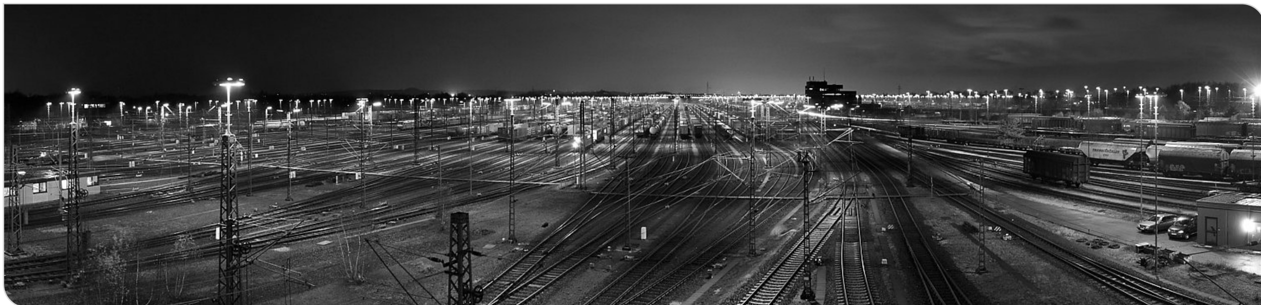
Karlsruher Institut für Technologie

Algorithmen für Routenplanung

18. Vorlesung, Sommersemester 2023

Adrian Feilhauer | 28. Juni 2023

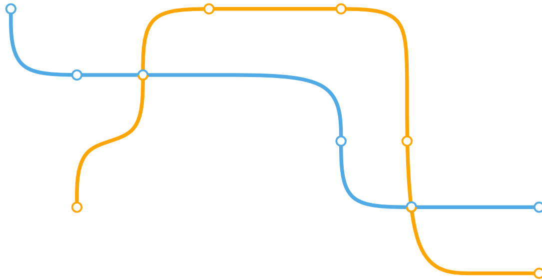


Fahrplanauskunft

Karlsruher Institut für Technologie

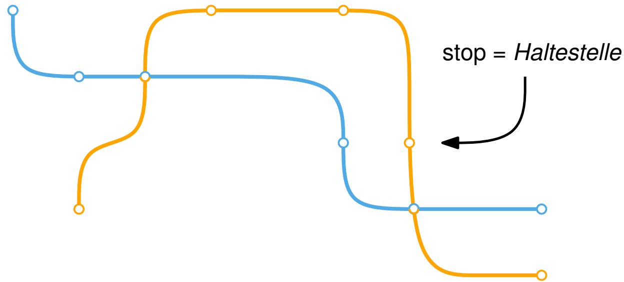
© Transport for London August 2011

Wdh. Begriffe

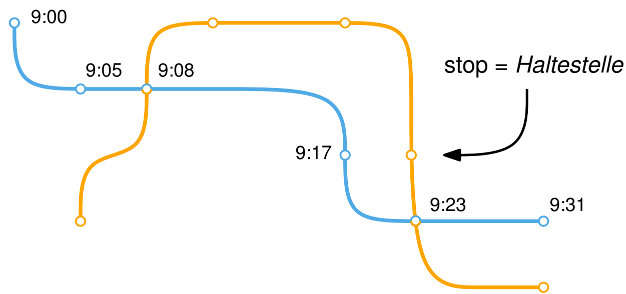


Wdh. Begriffe

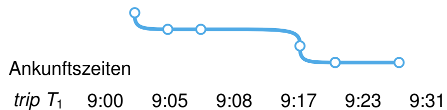
Karlsruher Institut für Technologie

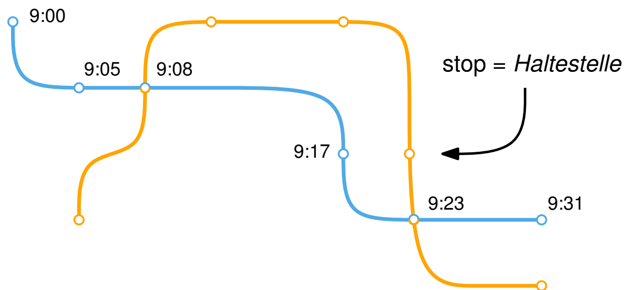


Wdh. Begriffe

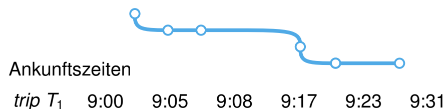


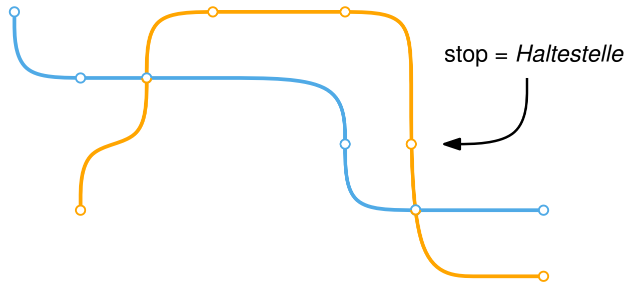
Wdh. Begriffe



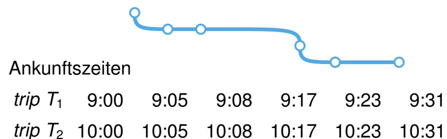


Wdh. Begriffe



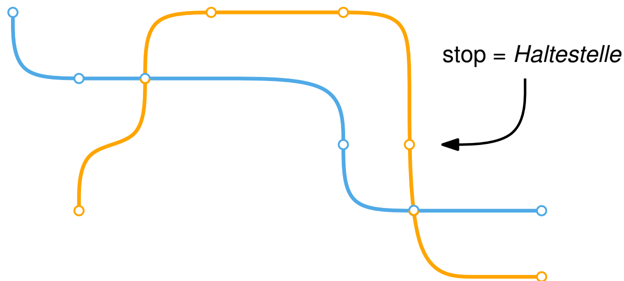


Wdh. Begriffe

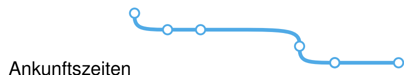


Ankunftszeiten

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

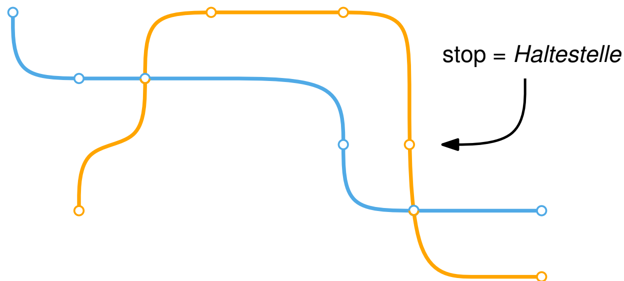


Wdh. Begriffe

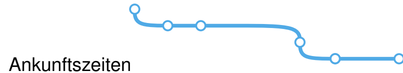


Ankunftszeiten

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

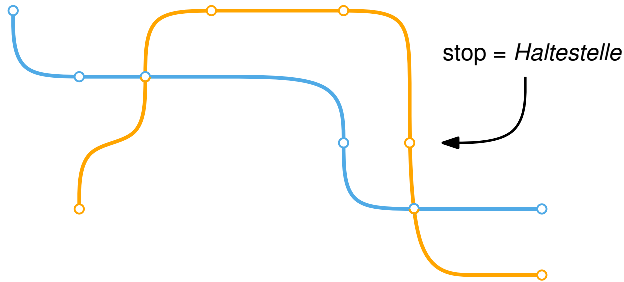


Wdh. Begriffe



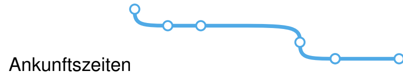
Ankunftszeiten

trip T_1	9:00	9:05	9:08	9:17	9:23	9:31	Route R_1
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_2



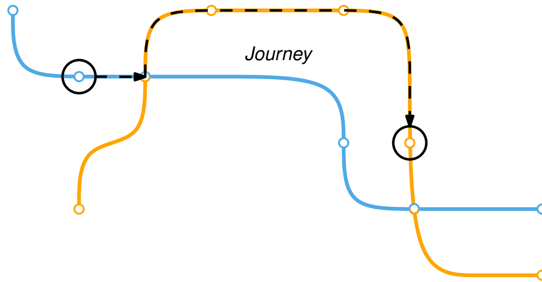
stop = *Haltestelle*

Wdh. Begriffe



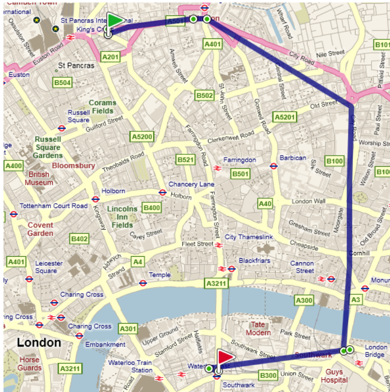
Ankunftszeiten

trip T_1	9:00	9:05	9:08	9:17	9:23	9:31	Route R_1
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_2

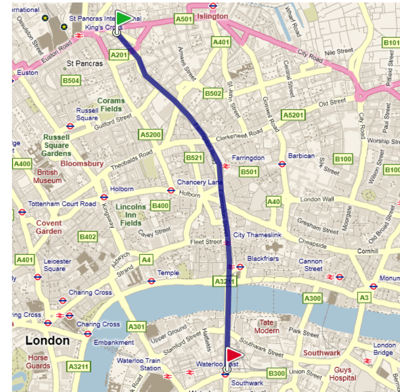


Wdh. Problemstellung

Gesucht: „Gute“ Journeys für Ankunftszeit und Anzahl Umstiege



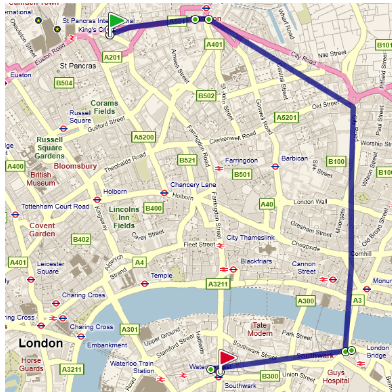
Ankunft 11:08 Uhr, 2 Umstiege



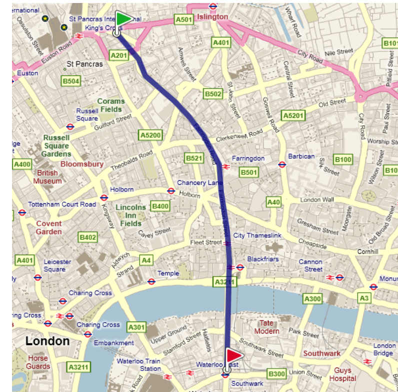
Ankunft 11:09 Uhr, 0 Umstiege

Wdh. Problemstellung

Gesucht: „Gute“ Journeys 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?

Karlsruher Institut für Technologie

Bis jetzt:

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

Graph-Modelle?

Karlsruher Institut für Technologie

Bis jetzt:

- Modelliere Fahrplan als gerichteten Graphen
- Zeitexpandiert vs. zeitabhängig
- Verschiedene Varianten von Dijkstras 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?

Karlsruher Institut für Technologie

Bis jetzt:

- Modelliere Fahrplan als gerichteten Graphen
- Zeitexpandiert vs. zeitabhängig
- Verschiedene Varianten von Dijkstras 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

Karlsruher Institut für Technologie

Anforderungen:

RAPTOR

Karlsruher Institut für Technologie

Anforderungen:

- Berechnet Pareto-Mengen

  mindestens Ankunftszeit und # Umstiege

RAPTOR

Karlsruher Institut für Technologie

Anforderungen:

- Berechnet Pareto-Mengen
  mindestens Ankunftszeit und # Umstiege
- Nutzt die Struktur der Fahrpläne aus
  benutzt Routen und Trips explizit?

RAPTOR

Karlsruher Institut für Technologie

Anforderungen:

- Berechnet Pareto-Mengen
  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

Karlsruher Institut für Technologie

Anforderungen:

- Berechnet Pareto-Mengen
  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

Karlsruher Institut für Technologie

Anforderungen:

- Berechnet Pareto-Mengen
  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

Karlsruher Institut für Technologie

Anforderungen:

- Berechnet Pareto-Mengen
  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

Karlsruher Institut für Technologie

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

Runden

Karlsruher Institut für Technologie

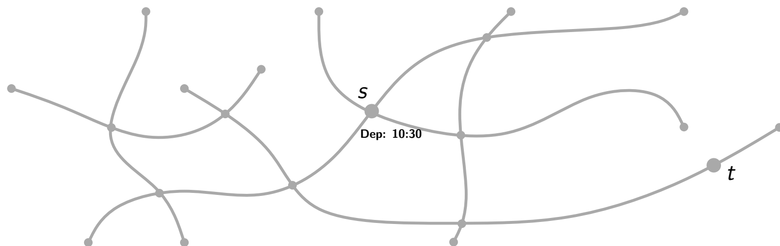
Beobachtung: Wechseln zw. Trips führt immer zu einem Umstieg.

Idee: Eine Runde für jeden genommenen Trip.

Runden

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

Idee: Eine Runde für jeden genommenen Trip.

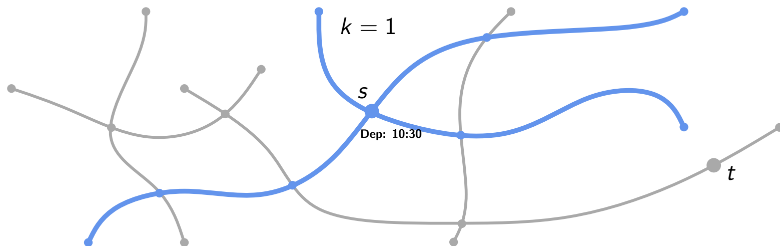


Ansatz: Runde k berechnet Ankunftszeiten für Journeys mit k Trips.

Runden

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

Idee: Eine Runde für jeden genommenen Trip.



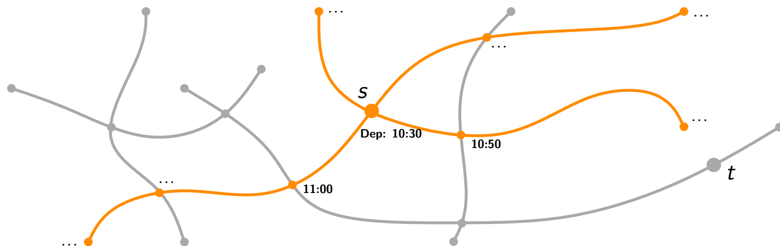
Ansatz: Runde k berechnet Ankunftszeiten für Journeys mit k Trips.

Runden

Karlsruher Institut für Technologie

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

Idee: Eine Runde für jeden genommenen Trip.

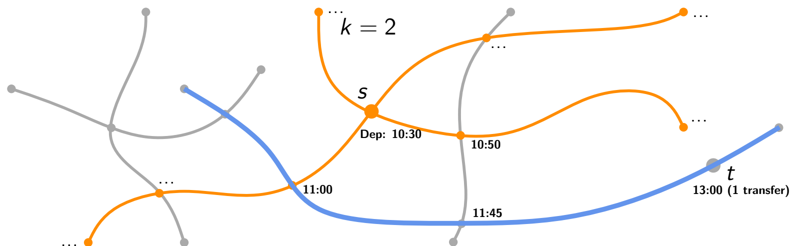


Ansatz: Runde k berechnet Ankunftszeiten für Journeys mit k Trips.

Runden

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

Idee: Eine Runde für jeden genommenen Trip.



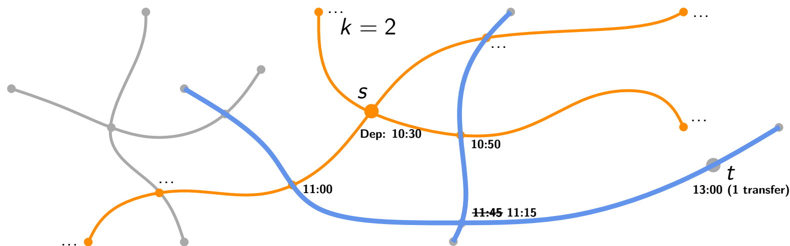
Ansatz: Runde k berechnet Ankunftszeiten für Journeys mit k Trips.

Runden

Karlsruher Institut für Technologie

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

Idee: Eine Runde für jeden genommenen Trip.



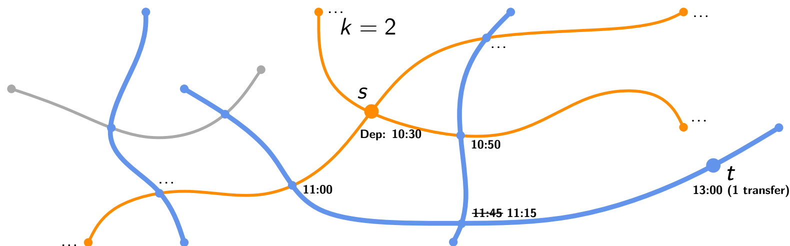
Ansatz: Runde k berechnet Ankunftszeiten für Journeys mit k Trips.

Runden

Karlsruher Institut für Technologie

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

Idee: Eine Runde für jeden genommenen Trip.

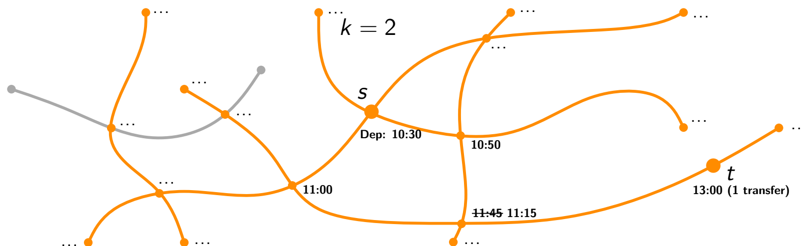


Ansatz: Runde k berechnet Ankunftszeiten für Journeys mit k Trips.

Runden

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

Idee: Eine Runde für jeden genommenen Trip.

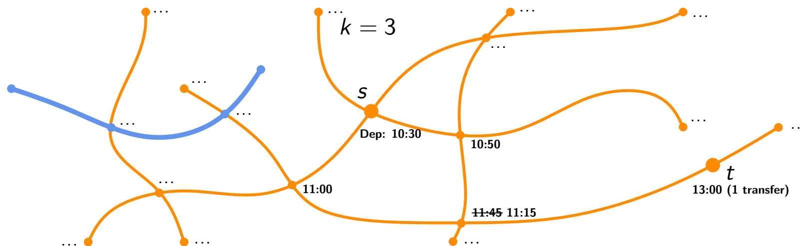


Ansatz: Runde k berechnet Ankunftszeiten für Journeys mit k Trips.

Runden

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

Idee: Eine Runde für jeden genommenen Trip.

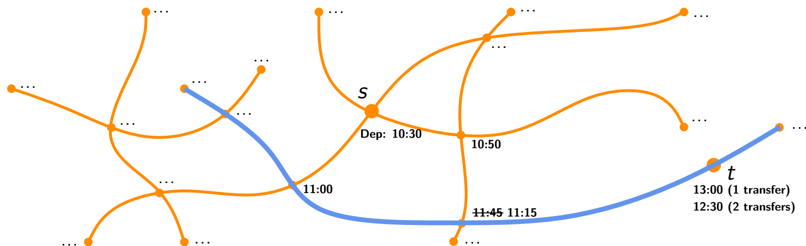


Ansatz: Runde k berechnet Ankunftszeiten für Journeys mit k Trips.

Runden

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

Idee: Eine Runde für jeden genommenen Trip.



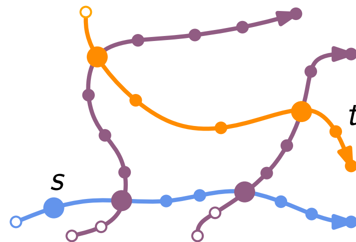
Ansatz: Runde k berechnet Ankunftszeiten für Journeys mit k Trips.

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

RAPTOR: Übersicht

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

1. Scanne jede Route
2. Relaxiere Fußwege

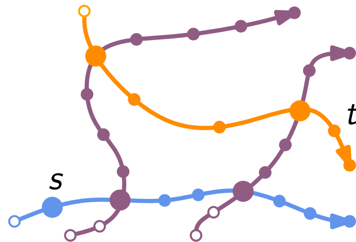


RAPTOR: Übersicht

Karlsruher Institut für Technologie

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*
- Routenscan in Runde k: Iteriere über Stops der Route
- Aktiver Trip: Frühester Trip, in den wir bisher einsteigen konnten

Prüfe für jeden Stop:

- Aussteigen: Verbessert aktiver Trip das Label aus Runde k?
- Einsteigen: Verbessert Label aus Runde $k-1$ den aktiven Trip?

Scannen von Routen

- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Routenscan in Runde k: Iteriere über Stops der Route
- Aktiver Trip: Frühester Trip, in den wir bisher einsteigen konnten

Prüfe für jeden Stop:

- Aussteigen: Verbessert aktiver Trip das Label aus Runde k?
- Einsteigen: Verbessert Label aus Runde $k - 1$ den aktiven Trip?

Scannen von Routen

- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Routenscan in Runde k: Iteriere über Stops der Route
- Aktiver Trip: Frühester Trip, in den wir bisher einsteigen konnten

Prüfe für jeden Stop:

- Aussteigen: Verbessert aktiver Trip das Label aus Runde k?
- Einsteigen: Verbessert Label aus Runde $k-1$ den aktiven Trip?

Scannen von Routen

- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Routenscan in Runde k: Iteriere über Stops der Route
- Aktiver Trip: Frühester Trip, in den wir bisher einsteigen konnten

Prüfe für jeden Stop:

- Aussteigen: Verbessert aktiver Trip das Label aus Runde k?
- Einsteigen: Verbessert Label aus Runde $k-1$ den aktiven Trip?

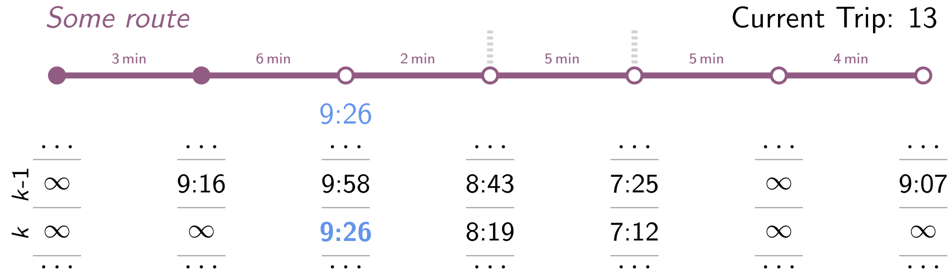
Scannen von Routen

- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Routenscan in Runde k: Iteriere über Stops der Route
- Aktiver Trip: Frühester Trip, in den wir bisher einsteigen konnten

Prüfe für jeden Stop:

- Aussteigen: Verbessert aktiver Trip das Label aus Runde k?
- Einsteigen: Verbessert Label aus Runde $k - 1$ den aktiven Trip?

Scannen von Routen



- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Routenscan in Runde k: Iteriere über Stops der Route
- Aktiver Trip: Frühester Trip, in den wir bisher einsteigen konnten

Prüfe für jeden Stop:

- Aussteigen: Verbessert aktiver Trip das Label aus Runde k?
- Einsteigen: Verbessert Label aus Runde $k-1$ den aktiven Trip?

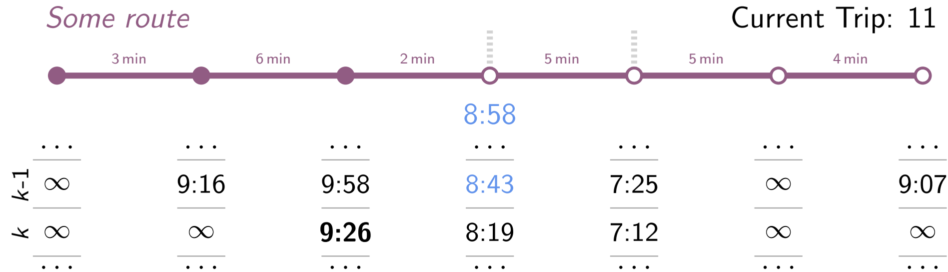
Scannen von Routen

- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Routenscan in Runde k: Iteriere über Stops der Route
- Aktiver Trip: Frühester Trip, in den wir bisher einsteigen konnten

Prüfe für jeden Stop:

- Aussteigen: Verbessert aktiver Trip das Label aus Runde k?
- Einsteigen: Verbessert Label aus Runde $k-1$ den aktiven Trip?

Scannen von Routen

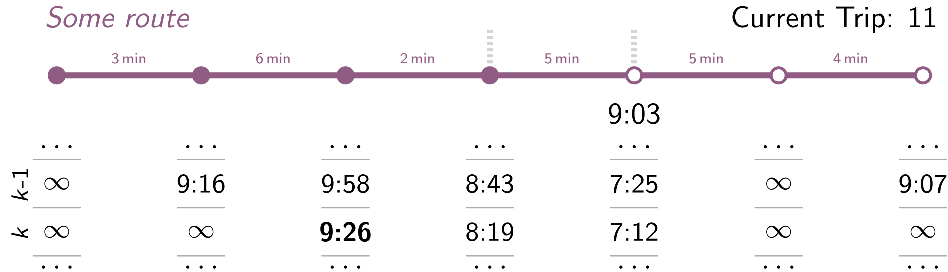


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Routenscan in Runde k: Iteriere über Stops der Route
- Aktiver Trip: Frühester Trip, in den wir bisher einsteigen konnten

Prüfe für jeden Stop:

- Aussteigen: Verbessert aktiver Trip das Label aus Runde k?
- Einsteigen: Verbessert Label aus Runde $k - 1$ den aktiven Trip?

Scannen von Routen

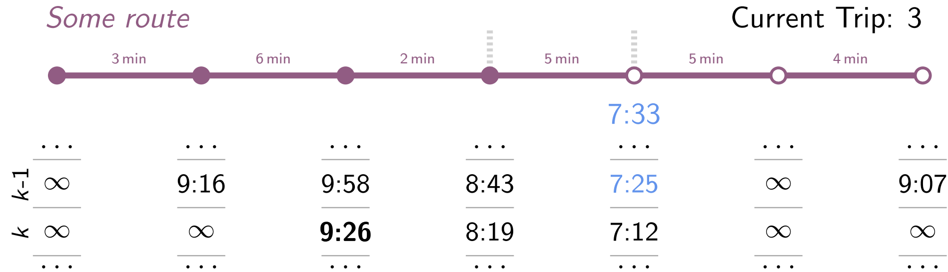


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Routenscan in Runde k: Iteriere über Stops der Route
- Aktiver Trip: Frühester Trip, in den wir bisher einsteigen konnten

Prüfe für jeden Stop:

- Aussteigen: Verbessert aktiver Trip das Label aus Runde k?
- Einsteigen: Verbessert Label aus Runde $k - 1$ den aktiven Trip?

Scannen von Routen

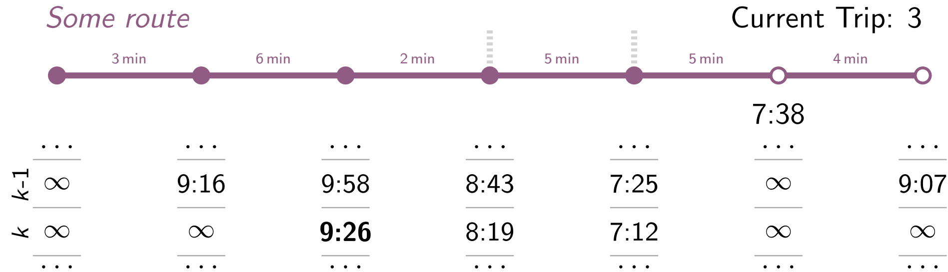


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Routenscan in Runde k: Iteriere über Stops der Route
- Aktiver Trip: Frühester Trip, in den wir bisher einsteigen konnten

Prüfe für jeden Stop:

- Aussteigen: Verbessert aktiver Trip das Label aus Runde k?
- Einsteigen: Verbessert Label aus Runde $k-1$ den aktiven Trip?

Scannen von Routen



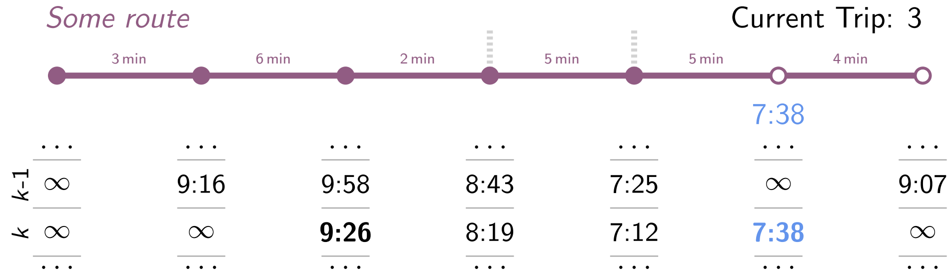
- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Routenscan in Runde k: Iteriere über Stops der Route
- Aktiver Trip: Frühester Trip, in den wir bisher einsteigen konnten

Prüfe für jeden Stop:

- Aussteigen: Verbessert aktiver Trip das Label aus Runde k?
- Einsteigen: Verbessert Label aus Runde $k-1$ den aktiven Trip?

Scannen von Routen

KIT Karlsruher Institut für Technologie

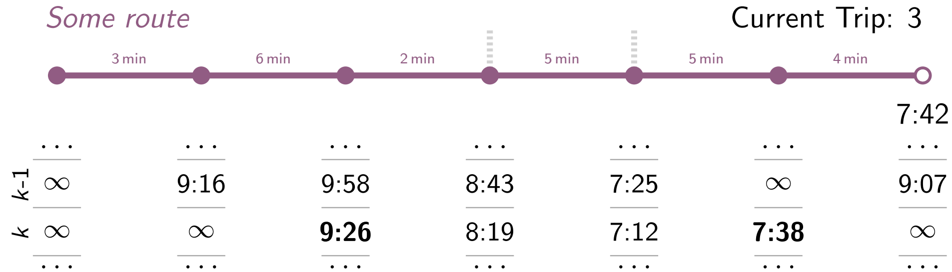


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Routenscan in Runde k: Iteriere über Stops der Route
- Aktiver Trip: Frühester Trip, in den wir bisher einsteigen konnten

Prüfe für jeden Stop:

- Aussteigen: Verbessert aktiver Trip das Label aus Runde k?
- Einsteigen: Verbessert Label aus Runde $k-1$ den aktiven Trip?

Scannen von Routen

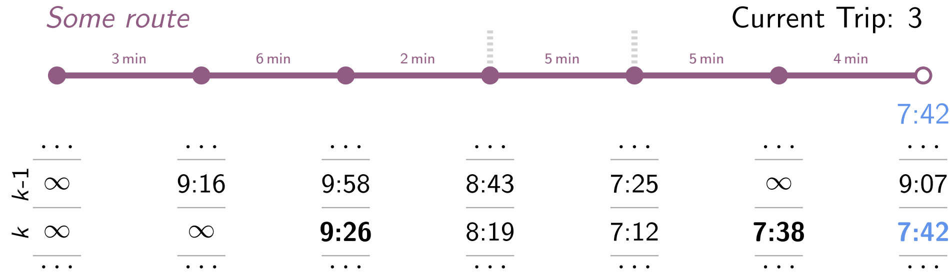


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Routenscan in Runde k: Iteriere über Stops der Route
- Aktiver Trip: Frühester Trip, in den wir bisher einsteigen konnten

Prüfe für jeden Stop:

- Aussteigen: Verbessert aktiver Trip das Label aus Runde k?
- Einsteigen: Verbessert Label aus Runde $k-1$ den aktiven Trip?

Scannen von Routen

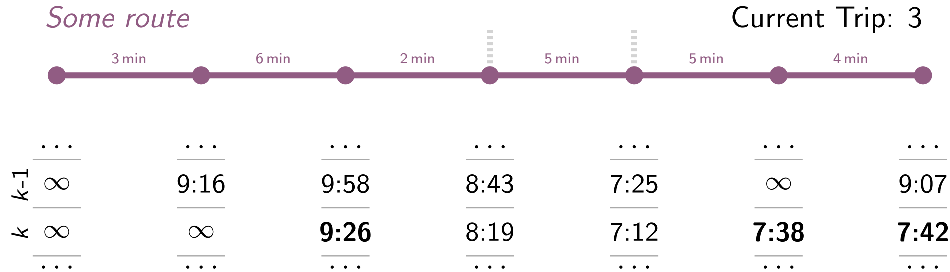


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Routenscan in Runde k: Iteriere über Stops der Route
- Aktiver Trip: Frühester Trip, in den wir bisher einsteigen konnten

Prüfe für jeden Stop:

- Aussteigen: Verbessert aktiver Trip das Label aus Runde k?
- Einsteigen: Verbessert Label aus Runde $k-1$ den aktiven Trip?

Scannen von Routen

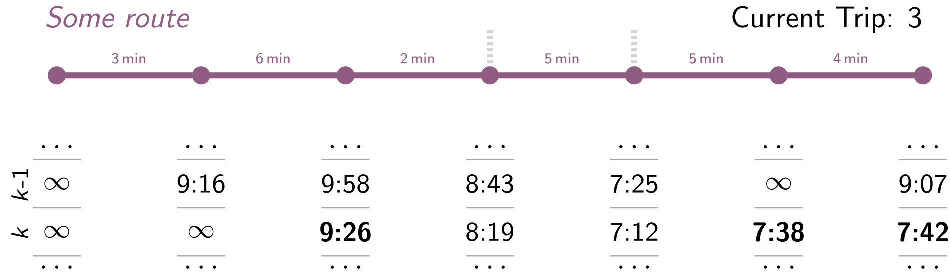


- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Routenscan in Runde k: Iteriere über Stops der Route
- Aktiver Trip: Frühester Trip, in den wir bisher einsteigen konnten

Prüfe für jeden Stop:

- Aussteigen: Verbessert aktiver Trip das Label aus Runde k?
- Einsteigen: Verbessert Label aus Runde $k-1$ den aktiven Trip?

Scannen von Routen



- Jeder Stop hat ein Label (Ankunftszeit) *pro Runde*
- Routenscan in Runde k: Iteriere über Stops der Route
- Aktiver Trip: Frühester Trip, in den wir bisher einsteigen konnten

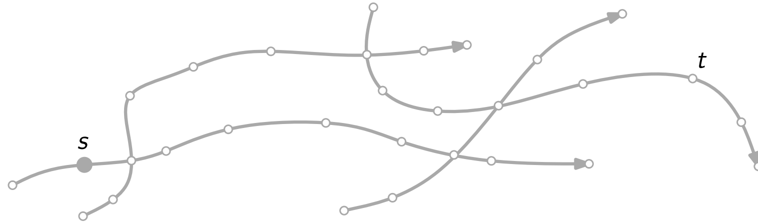
Prüfe für jeden Stop:

- Aussteigen: Verbessert aktiver Trip das Label aus Runde k?
- Einsteigen: Verbessert Label aus Runde $k-1$ den aktiven Trip?

Dynamischer Programmierungsansatz

Markieren und Pruning

Beobachtung: Nicht alle Routen werden in jeder Runde erreicht.



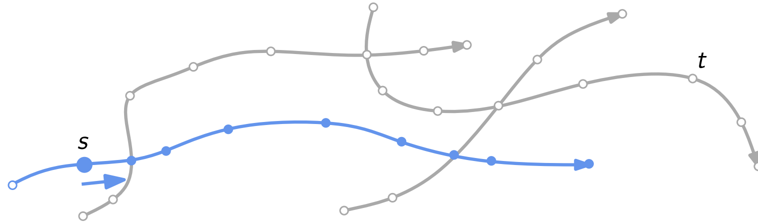
Markieren und Pruning:

- 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

Karlsruher Institut für Technologie

Beobachtung: Nicht alle Routen werden in jeder Runde erreicht.

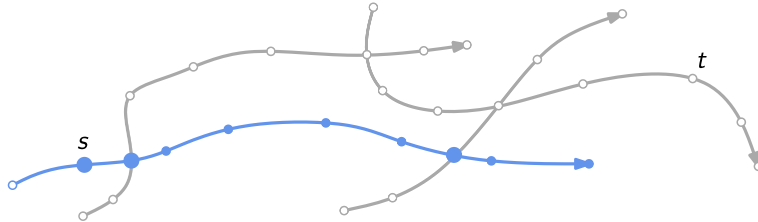


Markieren und Pruning:

- 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.

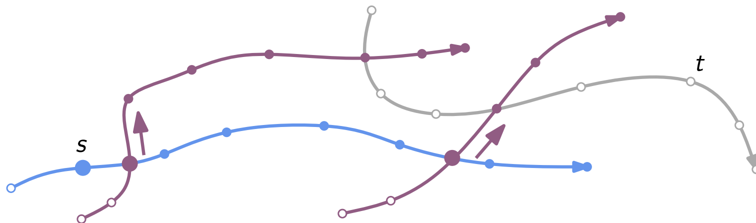


Markieren und Pruning:

- 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.

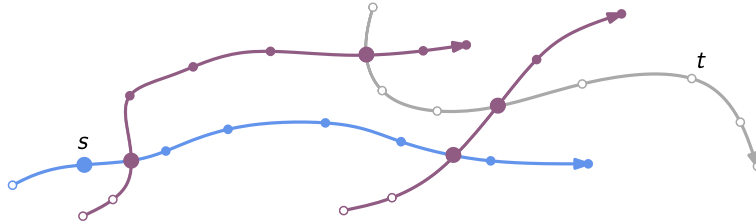


Markieren und Pruning:

- 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.

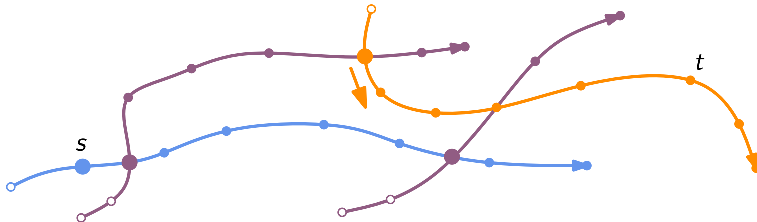


Markieren und Pruning:

- 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.

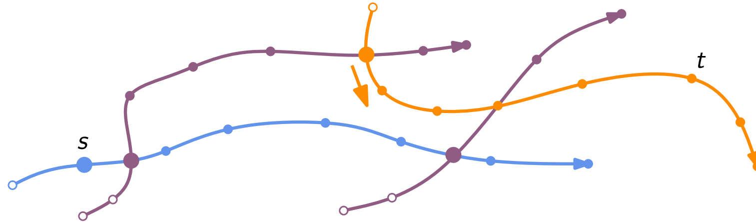


Markieren und Pruning:

- 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.



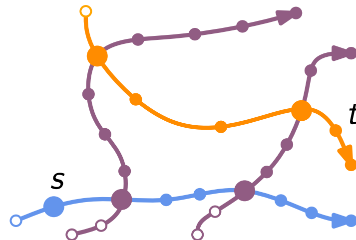
Markieren und Pruning:

- Route scannen: Markiere Stop, falls Ankunftszeit verbessert
- Nächste Runde: Nur Routen von markierten Stops scannen
- Scanne jede Route ab ihrem ersten markierten Stop
- Markiere Stops nur wenn bessere Ankunftszeit als am Ziel

RAPTOR: Übersicht

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

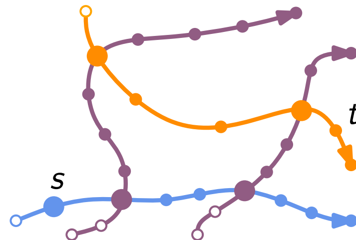
1. Wähle erreichte Routen aus letzter Runde
2. Scanne diese Routen
3. Relaxiere Fußwege



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.

Pseudocode – Routen einsammeln

Karlsruher Institut für Technologie

Datenstrukturen:

- Q: Menge von Tupeln (R, i)
  - Erreichte Route R mit Stopsequenz $(v_0, v_1, \ldots, v_{|R|-1})$
  - Frühester erreichter Stop auf R ist v_i

```
Leere Q
for alle markierten Stops v do
    for alle Routen R, die v besuchen do
        i ←Position von v in R
        if ∃j mit (R, j) ∈ Q then
            Ersetze (R, j) durch (R, min{i, j}) in Q
        else
            Füge (R, i) in Q ein
Lösche alle Markierungen
```

Pseudocode – Routenscan

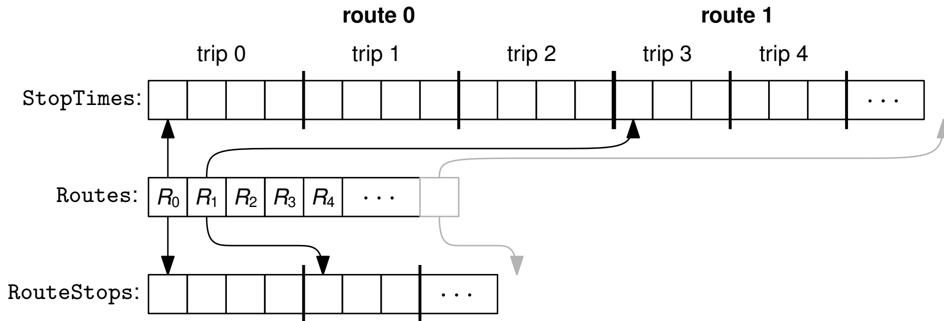
Karlsruher Institut für Technologie

Datenstrukturen:

- $\tau_k(v)$: Ankunftszeit an v in Runde k
- $\tau^*(v)$: Beste Ankunftszeit an v bisher
- $\text{pred}(T)$: Nächstfrüherer Trip vor T in der Route

```
for alle (R, i) ∈ Q do
    T ← arg min_{T∈R} {τ_dep(T, v_i) ≥ τ_{k−1}(v_i)}
    for alle v_j ∈ R mit j > i do
        if T ≠ ⊥ und τ_arr(T, v_j) < min{τ*(v_j), τ*(t)} then
            τ_k(v_j) ← τ_arr(T, v_j)
            τ*(v_j) ← τ_arr(T, v_j)
            Markiere v_j
        while pred(T) ≠ ⊥ und τ_{k−1}(v_j) < τ_dep(pred(T), v_j) do
            T ← pred(T)
```

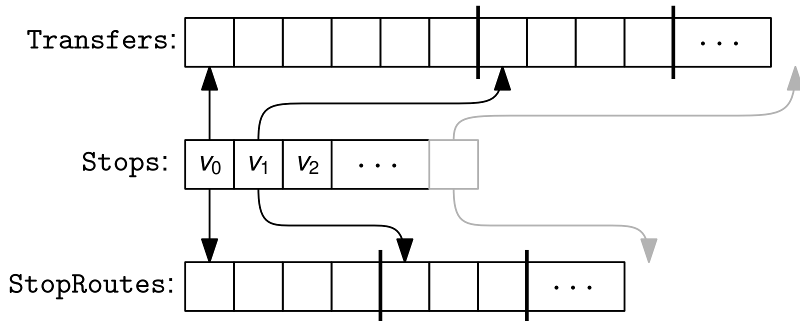
Datenstrukturen – Routen



Route R_i scannen:

- Finde ersten Stop in `RouteStops`
- Finde aktiven Trip in `StopTimes`
- Iteriere simultan über `RouteStops` und aktiven Trip
- Nächstfrüherer Trip: Springe in `StopTimes` um $|R_i|$ nach links

Datenstrukturen – Stops



Transfers relaxieren:

- Iteriere über `Transfers` von allen markierten Stops

Erreichte Routen einsammeln:

- Iteriere über `StopRoutes` von allen markierten Stops

Parallelisierung

Karlsruher Institut für Technologie

Beobachtung: Routen werden in beliebiger Reihenfolge gescannt.

Parallelisierung

Karlsruher Institut für Technologie

Beobachtung: Routen werden in beliebiger Reihenfolge gescannt.

Verteile Routen auf verschiedene CPU-Kerne; scanne parallel.

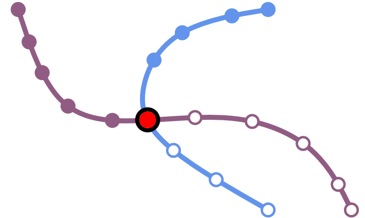
Parallelisierung

Beobachtung: Routen werden in beliebiger Reihenfolge gescannt.

Verteile Routen auf verschiedene CPU-Kerne; scanne parallel.

Vermeiden von Race-Conditions:

- Lock auf Schreiben von Labels (teuer)



Parallelisierung

Beobachtung: Routen werden in beliebiger Reihenfolge gescannt.

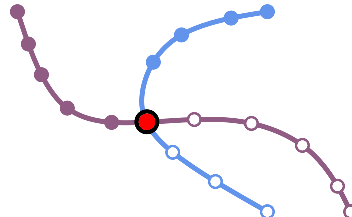
Verteile Routen auf verschiedene CPU-Kerne; scanne parallel.

Vermeiden von Race-Conditions:

- Lock auf Schreiben von Labels (teuer)

Oder ohne Locks:

- Synchronisiere Labels nach jeder Runde
- Sicherstellen, dass nur „unabhängige“ Routen gleichzeitig gescannt werden
  (Reduktion auf Färbeproblem)

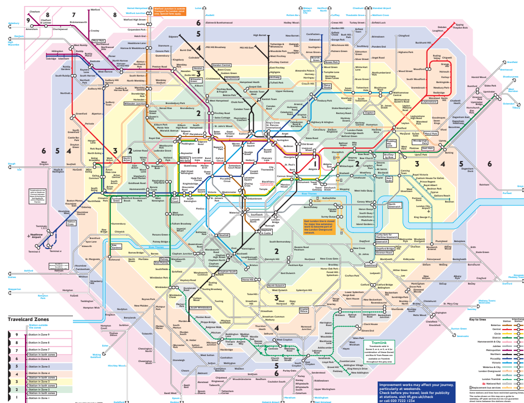


Andere Kriterien: Übersicht

KIT Karlsruher Institut für Technologie

Mögliche Erweiterungen:

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

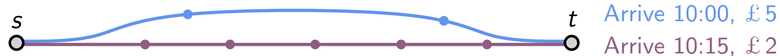


Performance hängt von Anzahl *nichtdominierter* Journeys ab.

More Criteria: McRAPTOR

Karlsruher Institut für Technologie

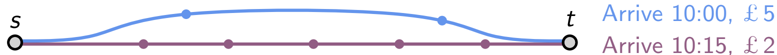
Ziel: Erweitern von RAPTOR auf zusätzliche Kriterien



More Criteria: McRAPTOR

Karlsruher Institut für Technologie

Ziel: Erweitern von RAPTOR auf zusätzliche Kriterien

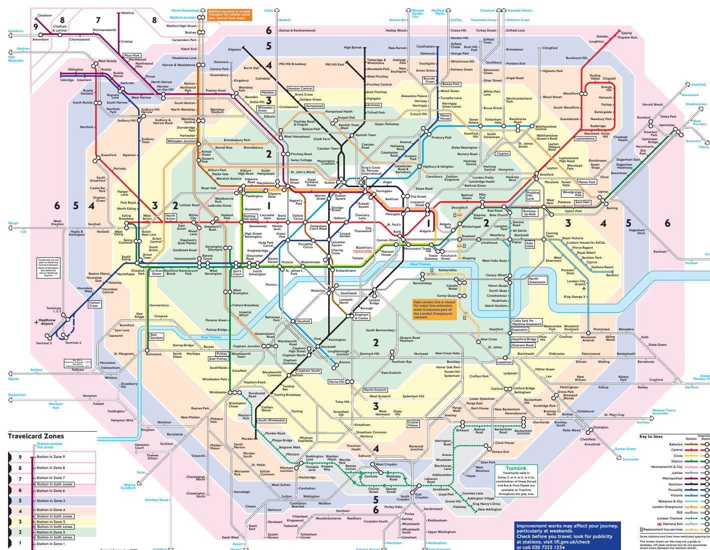


Ansatz:

- Labels haben Wert für jedes zusätzliche Kriterium
- Mehrere nichtdominierte Labels pro Stop und Runde
- Beim Routenscan: Ein aktiver Trip pro nichtdominiertem Label
- Lösche dominierte Labels on-the-fly

McRAPTOR Beispiel: Tarifzonen

Karlsruher Institut für Technologie



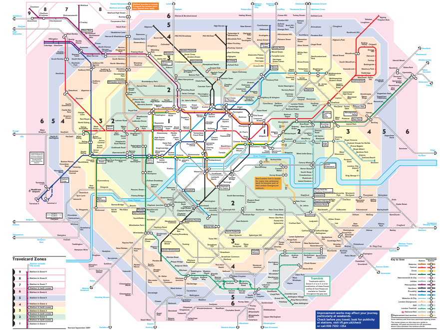
McRAPTOR Beispiel: Tarifzonen

Tarifzonen einbauen:

- Direkte Preise (■) nicht handhabbar

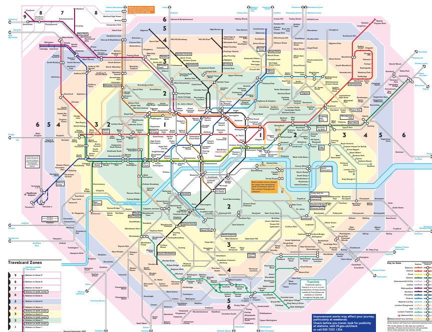
⇒ Berechne alle Kombinationen von Tarifzonen

- . . . und filtere im Postprocessing



McRAPTOR Beispiel: Tarifzonen

KIT Karlsruher Institut für Technologie

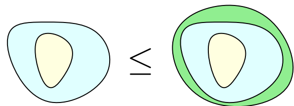


Tarifzonen einbauen:

- Direkte Preise (■) nicht handhabbar

⇒ Berechne alle Kombinationen von Tarifzonen

- ...und filtere im Postprocessing

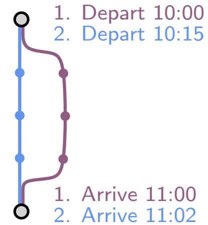


Implementierung:

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

Profil-Anfragen: rRAPTOR

Problem: Finde alle optimalen Journeys, die in einem Zeitintervall Δ abfahren.

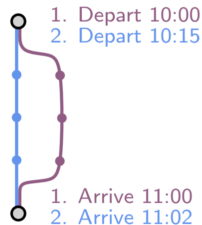


Profil-Anfragen: rRAPTOR

Karlsruher Institut für Technologie

Problem: Finde alle optimalen Journeys, die in einem Zeitintervall Δ abfahren.

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

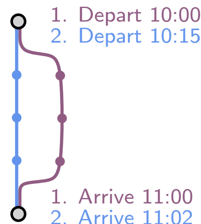


Profil-Anfragen: rRAPTOR

Karlsruher Institut für Technologie

Problem: Finde alle optimalen Journeys, die in einem Zeitintervall Δ abfahren.

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



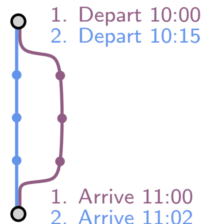
Effizienterer Ansatz: rRAPTOR (Self-Pruning)

- 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 optimalen Journeys, 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 Journeys, die früher abfahren und später ankommen.

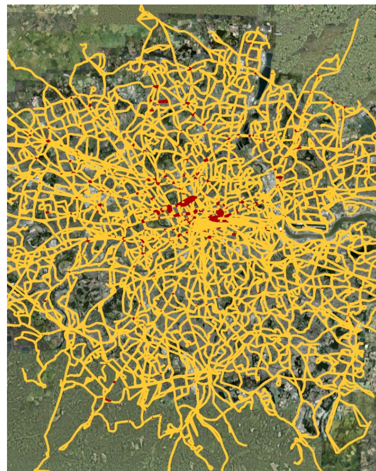
Experimente

Das Londoner Netzwerk

Karlsruher Institut für Technologie

Das vollständige Londoner Netzwerk

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

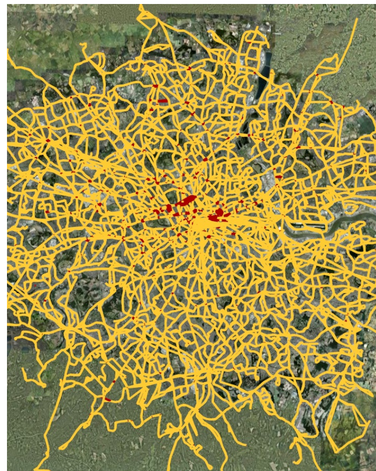


Das Londoner Netzwerk

Das vollständige Londoner Netzwerk

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

Experimente: 10 000 zufällige s–t-Anfragen



Vergleich der Algorithmen

Karlsruher Institut für Technologie

(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

Karlsruher Institut für Technologie

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

Karlsruher Institut für Technologie

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)

- Gute Speedups auf bis zu 6 Kernen
- RAPTOR immer $\leq 30\,\text{ms}$

Literatur I

Karlsruher Institut für Technologie

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.