Karlsruher Institut für Technologie

Algorithmen für Routenplanung

19. Vorlesung, Sommersemester 2017

Tobias Zündorf | 24. Juli 2017

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

Inhalt heute

Profil CSA:

- Pareto-optimierung (Ankunftszeit & Anzahl Umstiege)

Unbeschränktes Laufen:

- Limitierungen von transitiv-abgeschlossenen Graphen
- Auswirkung von vollständigen Fußweggraphen

Verkehrsumlegung:

- Fahrzeugauslastungen bei statistischem Reiseaufkommen

Wdh.: Profil CSA

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Vektoroperationen

Broadcast

- Eingabe: x
- Ausgabe: (x, x, x, x, x, x, x, x)

Minimum

- Eingabe: $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$ und $(y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8)$
- Ausgabe: $(z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8)$ mit $z_i = \min\{x_i, y_i\}$

Shift

- Eingabe: $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$
- Ausgabe: $(\infty, x_1, x_2, x_3, x_4, x_5, x_6, x_7)$

- Geht alles mit SIMD/SSE/AVX

Vektoroperationen

Broadcast

- Eingabe: x
- Ausgabe: (x, x, x, x, x, x, x, x)

Minimum

- Eingabe: $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$ und $(y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8)$
- Ausgabe: $(z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8)$ mit $z_i = \min\{x_i, y_i\}$

Shift

- Eingabe: $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$
- Ausgabe: $(\infty, x_1, x_2, x_3, x_4, x_5, x_6, x_7)$

- Geht alles mit SIMD/SSE/AVX

Vektoroperationen

Broadcast

- Eingabe: x
- Ausgabe: (x, x, x, x, x, x, x, x)

Minimum

- Eingabe: $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$ und $(y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8)$
- Ausgabe: $(z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8)$ mit $z_i = \min\{x_i, y_i\}$

Shift

- Eingabe: $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$
- Ausgabe: $(\infty, x_1, x_2, x_3, x_4, x_5, x_6, x_7)$

- Geht alles mit SIMD/SSE/AVX

Vektoroperationen

Broadcast

- Eingabe: x
- Ausgabe: (x, x, x, x, x, x, x, x)

Minimum

- Eingabe: $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$ und $(y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8)$
- Ausgabe: $(z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8)$ mit $z_i = \min\{x_i, y_i\}$

Shift

- Eingabe: $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$
- Ausgabe: $(\infty, x_1, x_2, x_3, x_4, x_5, x_6, x_7)$

- Geht alles mit SIMD/SSE/AVX

Algorithmus Übersicht

Initialisiere Profil an Stops;
Initialisiere optimale Ankunftszeit an Trips;

for *alle Connections c absteigend nach* $\tau_{\text{dep}}(c)$ **do**

`// 1. Bestimme Ankunftszeit von man in` c `startet`

$\tau_1 \leftarrow$ Ankunftszeit wenn man zum Ziel läuft;
$\tau_2 \leftarrow$ Ankunftszeit wenn Sitzenbleiben (Ankunftszeit von Trip trip(c));
$\tau_3 \leftarrow$ Ankunftszeit wenn Umsteigen (Profil vom Stop $s_{\text{arr}}(c)$);

`//` τ_c `Ankunftszeit wenn man in` c `beginnt`

$\tau_c \leftarrow \min\{\tau_1, \tau_2, \tau_3\}$;

`// 2. Passe die Profile und Ankunftszeiten an`

Füge τ_c zum Profil von Stop $s_{\text{dep}}(c)$ hinzu;
Aktualisiere Ankunftszeit von trip(c) mit τ_c;

- Ankunftszeiten τ_1, τ_2, τ_3, und τ_c sind nun Vektoren
- Abfahrtszeiten bleiben Skalare

Zum Ziel Laufen

Code:

- Aus:

$\tau_1 \leftarrow$ Ankunftszeit wenn man zum Ziel läuft;

- Wird:

if $s_{\text{arr}}(c) = \text{target}$ **then**
  $\tau_1 \leftarrow \text{broadcast}(\tau_{\text{arr}}(c))$;
else
  $\tau_1 \leftarrow \text{broadcast}(\infty)$;

- Finale Fußwege gehen genauso wie bisher

Sitzenbleiben

Trip-Datenstruktur:

- Bisher: T[x] ist eine Ankunftszeit
- Jetzt: T[x] wird Vektor

Code:

- Aus:

```
Initialisiere optimale Ankunftszeit an Trips;
```

- Wird:

```
for alle Trips x do
    T[x] ← broadcast(∞);
```

Sitzenbleiben

Trip-Datenstruktur:

- Bisher: T[x] ist eine Ankunftszeit
- Jetzt: T[x] wird Vektor

Code:

- Aus:

  $\tau_2 \leftarrow$ Ankunftszeit wenn Sitzenbleiben;

- Wird:

  $\tau_2 \leftarrow$ T[trip(c)];

  (Diesmal sind die Variablen aber Vektoren)

Sitzenbleiben

Trip-Datenstruktur:

- Bisher: T[x] ist eine Ankunftszeit
- Jetzt: T[x] wird Vektor

Code:

- Aus:

  Aktualisiere Ankunftszeit von trip(c) mit τ_c;

- Wird:

  T[trip(c)] $\leftarrow \tau_c$;

  (Diesmal sind die Variablen aber Vektoren)

Umsteigen

Stop-Datenstruktur:

- Profile bilden Abfahrtszeit auf Ankunftszeit-Vektoren ab

Code:

- Aus:

```
Initialisiere Profil an Stops;
```

- Wird:

```
for alle Stops x do
    P[x] ← {∀τ : τ ↦ broadcast(∞)};
```

Pareto-Profile

Aufbau:

- **Bisher**: P ist Array von $(\tau_{\text{dep}}, \tau_{\text{arr}})$-Paaren
- **Nun**: P ist Array von $(\tau_{\text{dep}}, \text{vector})$-Paaren
- Arrays sind sortiert nach Abfahrtszeit

Interpretation:

- P[x] ist Profil von Stop x nach target
- P[x] zum Zeitpunkt τ auswerten ergibt Vektor v
- Vector $v[i]$ beschreibt wann man an target ankomme wenn man
  - zur Zeit τ
  - an Stop x losfahre
  - und höchstens i mal aussteigen darf

Pareto-Profile

Aufbau:

- **Bisher**: P ist Array von $(\tau_{\text{dep}}, \tau_{\text{arr}})$-Paaren
- **Nun**: P ist Array von $(\tau_{\text{dep}}, \text{vector})$-Paaren
- Arrays sind sortiert nach Abfahrtszeit

Interpretation:

- P[x] ist Profil von Stop x nach target
- P[x] zum Zeitpunkt τ auswerten ergibt Vektor v
- Vector $v[i]$ beschreibt wann man an target ankomme wenn man
  - zur Zeit τ
  - an Stop x losfahre
  - und höchstens i mal aussteigen darf

Umstiege Pareto-optimieren

Code:

- Jedes Array hat ein $(\infty, \text{broadcast}(\infty))$-Paar
- Aus:

for *alle Stops x* **do**
  P[*x*] $\leftarrow \{\forall \tau : \tau \mapsto \text{broadcast}(\infty)\}$;

- Wird:

for *alle Stops x* **do**
  P[*x*] $\leftarrow \{(\infty, \text{broadcast}(\infty))\}$;

Umstiege Pareto-optimieren

Code:

- Füge Paar ein, wenn es in mindestens einer Komponente besser ist
- Aus:

  Füge $(\tau_{\text{dep}}(c) - \tau_{\text{ch}}(s_{\text{dep}}(c)), \tau_c)$ in $P[s_{\text{dep}}(c)]$ ein;

- Wird:

$d \leftarrow \tau_{\text{dep}}(c) - \tau_{\text{ch}}(s_{\text{dep}}(c))$;
$x \leftarrow \min(\tau_c, \text{vector}(\text{P}[s_{\text{dep}}(c)][0]))$;
if $x \neq \text{vector}(\text{P}[s_{\text{dep}}(c)][0])$ **then**
  if $d = \tau_{\text{dep}}(\text{P}[s_{\text{dep}}(c)][0])$ **then**
    $\text{vector}(\text{P}[s_{\text{dep}}(c)][0]) \leftarrow x$;
  else
    Füge (d, x) am Anfang von $\text{P}[s_{\text{dep}}(c)]$ ein;

Umstiege Pareto-optimieren

- Vektor v der Länge n hat die Komponenten:

$$(v_1, v_2 \ldots v_n)$$

- Journeys werden gefunden bis maximal n Mal einsteigen

$\Rightarrow$ Bis zu $n - 1$ Umstiege

- Vektorlänge in der Regel 8
- 7 Umstiege ist für die meisten Anwendungen gut genug

Umstiege Pareto-optimieren

- Vektor v der Länge n hat die Komponenten:

$$(v_1, v_2 \dots v_n)$$

- Journeys werden gefunden bis maximal n Mal einsteigen

$\Rightarrow$ Bis zu $n - 1$ Umstiege

- Vektorlänge in der Regel 8
- 7 Umstiege ist für die meisten Anwendungen gut genug

Umstiege Pareto-optimieren

Beobachtung:

- Man kann die Shift-Operation so modifizieren, dass v_n die früheste Ankunftszeit ohne beschränkte Umstiege ist.
- Die Bedeutung von v_i für $i < n$ bleibt erhalten: höchstens i Ausstiege
- Ist in manchen Anwendungen nützlich

Modifiziertes Shift

- Eingabe: $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$
- Ausgabe: $(\infty, x_1, x_2, x_3, x_5, x_6, \min\{x_7, x_8\})$

Zeit Kompression

Problem: Algorithmus ist memory-bound

- Der Speicher ist fast vollständig mit Ankunftszeiten gefüllt

$\Rightarrow$ Komprimiere Ankunftszeiten

Beobachtung: Nicht zu jedem Zeitpunkt fährt ein Zug

Idee: Berechne für jeden Stop ein geordnetes Array von Zeitpunkten $t_0, t_1, \ldots, t_n$ an denen ein Zug abfährt oder ankommt

- Indizes respektieren die zeitliche Ordnung, d.h., $t_i < t_j \iff i < j$
- Indizes passen oft in 16 Bit
- Kopiere Indizes anstatt von Zeitpunkten

Fußwege

Nicht triviale Interaktion mit Fußwegen (nicht in der Vorlesung)

Zeit Kompression

Problem: Algorithmus ist memory-bound

- Der Speicher ist fast vollständig mit Ankunftszeiten gefüllt

$\Rightarrow$ Komprimiere Ankunftszeiten

Beobachtung: Nicht zu jedem Zeitpunkt fährt ein Zug

Idee: Berechne für jeden Stop ein geordnetes Array von Zeitpunkten $t_0, t_1, \ldots, t_n$ an denen ein Zug abfährt oder ankommt

- Indizes respektieren die zeitliche Ordnung, d.h., $t_i < t_j \iff i < j$
- Indizes passen oft in 16 Bit
- Kopiere Indizes anstatt von Zeitpunkten

Fußwege

Nicht triviale Interaktion mit Fußwegen (nicht in der Vorlesung)

Pfadendpackung

Option 1:

- Speichere an jeder Ankunftszeit den ersten Trip der Journey
- Entpacke Journeys rekursiv

Option 2:

- Nutze zusätzlichen vorwärts CSA (gestartet bei source)
- Benutze Profile um frühzeitig zu prunen
- Kann genutzt werden um eine Journey zu finden
- Kann auch genutzt werden um alle optimalen Journeys zu finden

Pfadendpackung

Option 1:

- Speichere an jeder Ankunftszeit den ersten Trip der Journey
- Entpacke Journeys rekursiv

Option 2:

- Nutze zusätzlichen vorwärts CSA (gestartet bei source)
- Benutze Profile um frühzeitig zu prunen
- Kann genutzt werden um eine Journey zu finden
- Kann auch genutzt werden um alle optimalen Journeys zu finden

Laufzeiten

Instanz: London mit 4 850 431 Connections

Laufzeiten (Earliest Arrival Queries):

Früheste Ankunft One-to-One:

- Time-Expanded Graph: 64.4 ms
- Time-Dependent Graph: 10.9 ms
- Connection Scan: 2.0 ms

Früheste Ankunft One-to-All:

- Time-Expanded Graph: 876.2 ms
- Time-Dependent Graph: 18.9 ms
- Connection Scan: 9.7 ms

Laufzeiten

Laufzeiten (Profile Queries):

Non-Pareto Profil All-to-One:

- Self-Pruning-Connection-Setting : 1 262 ms
- Connection Scan: 177 ms
- + constant eval: 134 ms
- + time compress: 104 ms

Pareto Profil All-to-One (max. 8 Züge pro Journey):

- RAPTOR : 1 179 ms
- Connection Scan: 255 ms
- + SSE: 221 ms

Unbeschränktes Laufen

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Fußwegegraphen

Bisher:

- Effiziente Algorithmen
- Eingeschränkte Modelle für Laufen

Einschränkungen:

RAPTOR	Transitiv abgeschlossen
CSA	Transitiv abgeschlossen
Trip-Based Routing	Transitiv abgeschlossen
Transfer Patterns	Max. 400 m
PTL	Specified by PT-network
MEAT	Kein laufen möglich

Fußwegegraphen

Argumente für Einschränkung:

- Laufen lohnt sich nie
- Laufen vom Nutzer unerwünscht
- Laufen in der Praxis nicht relevant

Aber:

- Relevanz nie bewiesen/widerlegt
- Algorithmus sollte über Relevanz entscheiden

Transitiver Abschluss

Frage: Wie lang kann man maximal laufen?

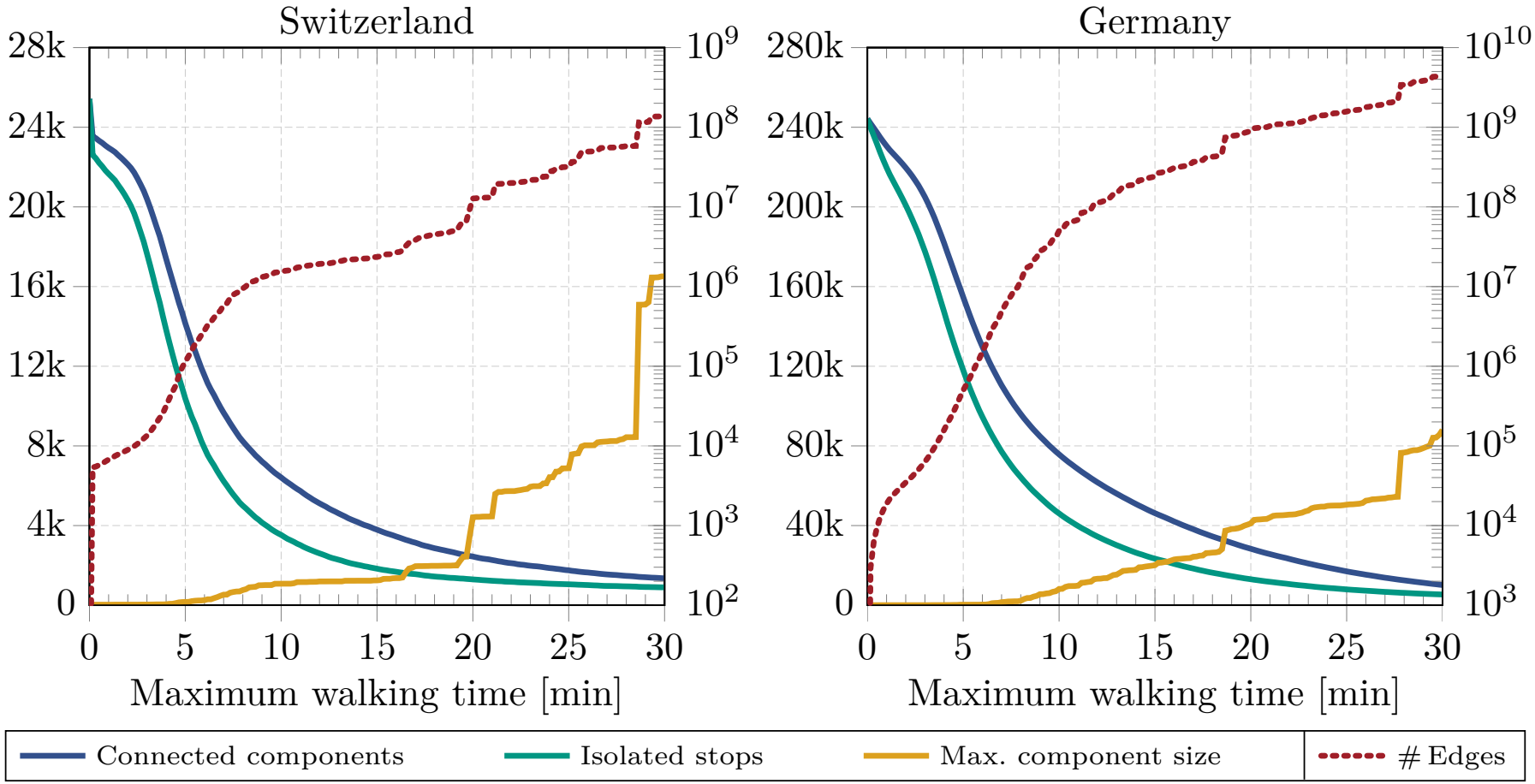
Ansatz:

- Starte mit vollständigem Fußweggraphen
- Wähle maximale Laufzeit τ
- Verbinde Stops $\Leftrightarrow$ Laufdistanz $\leq \tau$
- Bilde transitiven Abschluss des resultierenden Graphen

- Wie groß wird dieser Graph?

Transitiver Abschluss

Größe des transitiven Abschlusses:



Auswirkung des Fußwegegraphen

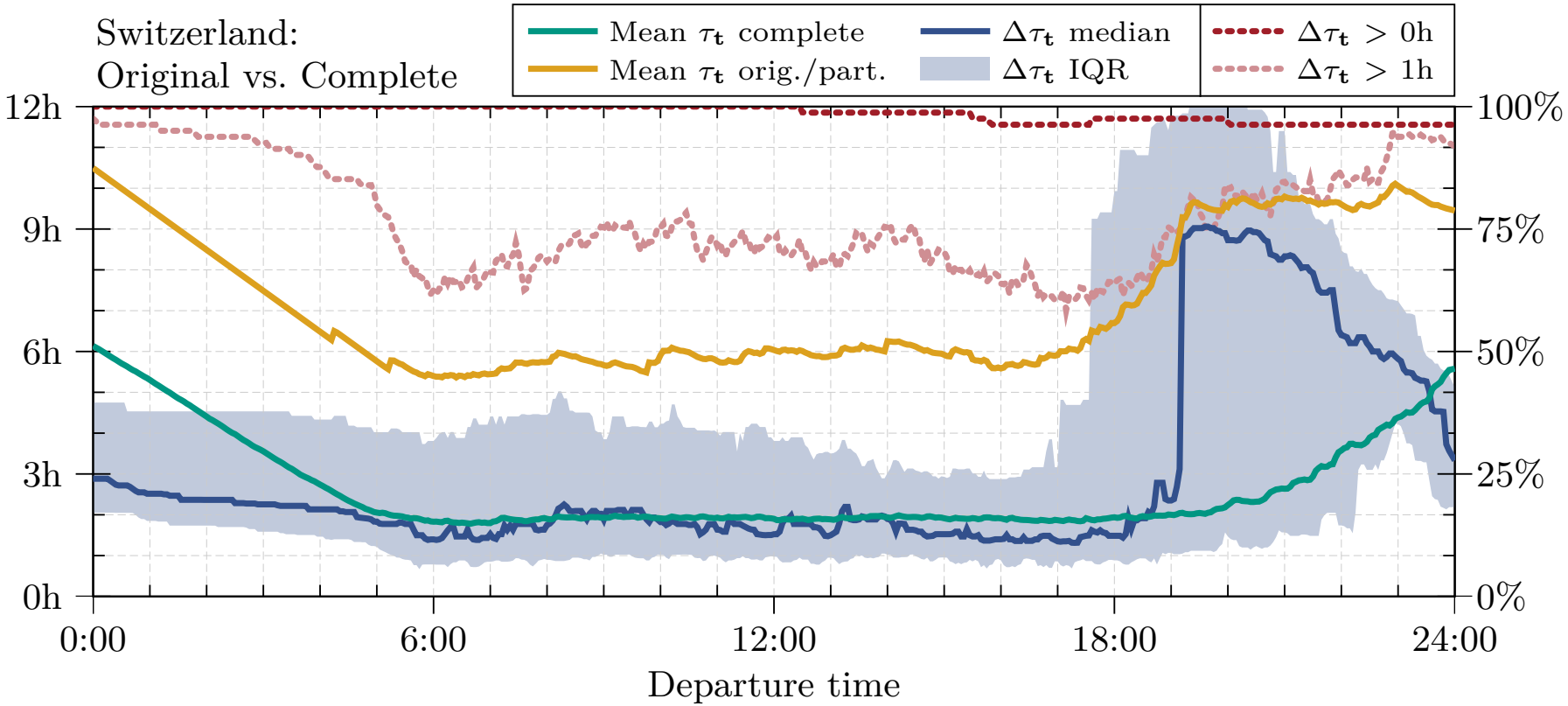
Frage: Wie wirkt sich ein größerer Fußwegegraph aus?

Ansatz:

- Vergleiche Reisezeiten in typischen Netzwerken
  - Nur Fußwege die im Public Transit Network spezifiziert sind
  - Transitiv abgeschlossener Fußwegegraph
  - Vollständiger Fußwegegraph

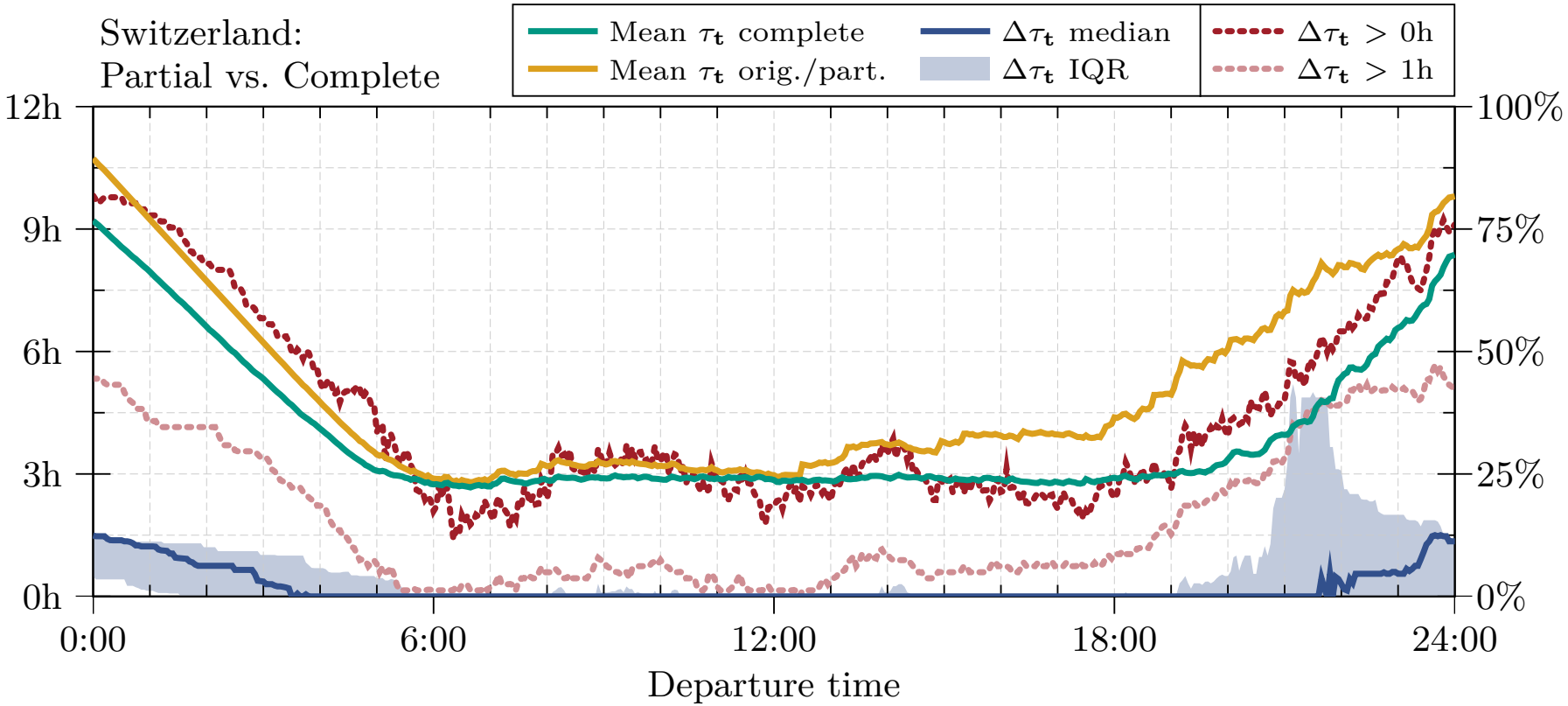
PT network	Footpaths	Stops	Vertices	Edges	Connections	Max. deg.
Switzerland	original	25 427	25 427	5 604	4 373 268	25
	partial	25 427	25 427	3 104 974	4 373 268	1 246
	complete	25 427	604 230	1 844 286	4 373 268	25
Germany	original	244 245	244 245	95 036	46 119 896	18
	partial	244 245	244 245	26 193 136	46 119 896	2 622
	complete	244 245	6 876 758	21 382 408	46 119 896	21

Vergleich der Reisezeiten



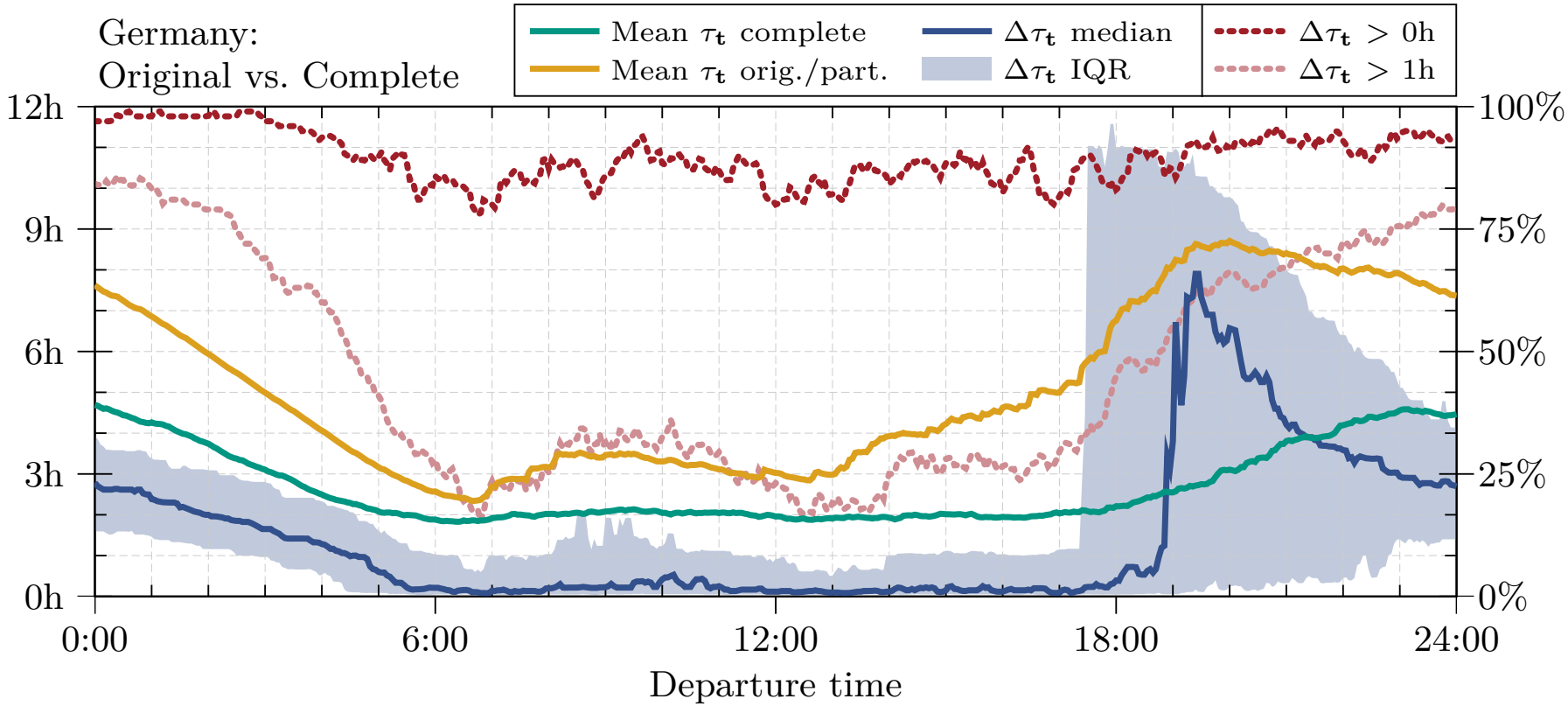
- 24h Profile zwischen 100 Zufälligen Stops (Distance Rank 16)
- Vergleiche Reisezeiten & Anteil suboptimaler Journeys

Vergleich der Reisezeiten



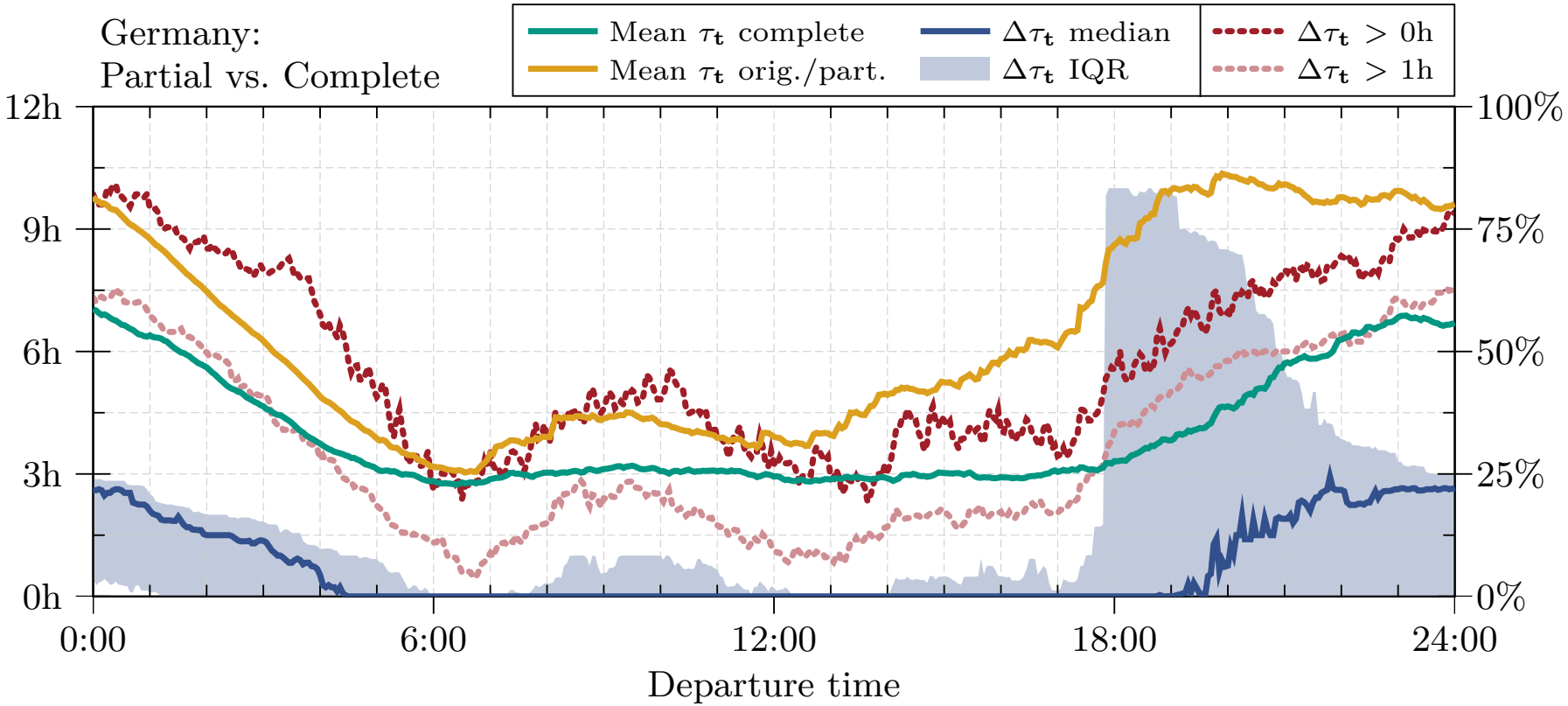
- 24h Profile zwischen 100 Zufälligen Stops (Distance Rank 16)
- Vergleiche Reisezeiten & Anteil suboptimaler Journeys

Vergleich der Reisezeiten



- 24h Profile zwischen 100 Zufälligen Stops (Distance Rank 16)
- Vergleiche Reisezeiten & Anteil suboptimaler Journeys

Vergleich der Reisezeiten



- 24h Profile zwischen 100 Zufälligen Stops (Distance Rank 16)
- Vergleiche Reisezeiten & Anteil suboptimaler Journeys

Verkehrsumlegung

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Verkehrsumlegung

Gegeben:

- Öffentliches Verkehrsnetz
- Liste mit erwarteter Nachfrage
  (Tupel aus: Startknoten, Zielknoten, Abfahrtszeit)

Gesucht:

- Auslastung der Fahrzeuge

Anwendung:

- Planung von neuen Linien
- Optimierung von Umleitungen

Verkehrsumlegung

Gegeben:

- Öffentliches Verkehrsnetz
- Liste mit erwarteter Nachfrage
  (Tupel aus: Startknoten, Zielknoten, Abfahrtszeit)

Gesucht:

- Auslastung der Fahrzeuge

Anwendung:

- Planung von neuen Linien
- Optimierung von Umleitungen

Verkehrsumlegung

Ansatz:

- Weise jedem Passagier (aus Nachfrage) eine Journey zu
- Algorithmus basiert auf CSA

Aber:

- Verhalten der Passagiere nicht immer eindeutig
- Erlaube suboptimale Journeys
- Erlaube mehrere (Anteilig) Journeys pro Passagier

Verkehrsumlegung

Ansatz:

- Weise jedem Passagier (aus Nachfrage) eine Journey zu
- Algorithmus basiert auf CSA

Aber:

- Verhalten der Passagiere nicht immer eindeutig
- Erlaube suboptimale Journeys
- Erlaube mehrere (Anteilig) Journeys pro Passagier

Gefühlte Ankunftszeit (PAT)

Idee:

- PAT für eine Connection c und Zielstop d
- Misst wie nützlich c ist um d zu erreichen
- Berücksichtigt vier Parameter:
  - Umstiegskosten λ_{trans}
  - Wartekosten λ_{wait}
  - Laufkosten λ_{walk}
  - Maximale erwartete Verspätung $\Delta_{\tau}^{\max}$

Annahme:

- Passagiere versuchen die PAT zu minimieren

Gefühlte Ankunftszeit (PAT)

Formale Definition:

$$\tau^{\text{p}}_{\text{arr}}(c, c', d) := \tau^{\text{p}}_{\text{trans}}(c, c') + \tau^{\text{p}}_{\text{wait}}(c, c') + \tau^{\text{p}}_{\text{arr}}(c', d)$$

$$\tau^{\text{p}}_{\text{arr}}(c, d \mid \text{walk}) := \begin{cases} \tau_{\text{arr}}(c) & \text{if } v_{\text{arr}}(c) = d \\ \tau_{\text{arr}}(c) + \lambda_{\text{walk}} \cdot \tau_{\text{trans}}(v_{\text{arr}}(c), d) & \text{otherwise} \end{cases}$$

$$\mathcal{T}(c) := \{c' \in \mathcal{C} \mid \text{trip}(c') = \text{trip}(c) \wedge \tau_{\text{dep}}(c') \geq \tau_{\text{arr}}(c)\}$$

$$\tau^{\text{p}}_{\text{arr}}(c, d \mid \text{trip}) := \begin{cases} \min\{\tau^{\text{p}}_{\text{arr}}(c', d) \mid c' \in \mathcal{T}(c)\} & \text{if } \mathcal{T}(c) \neq \emptyset \\ \infty & \text{otherwise} \end{cases}$$

$$\tau^{\text{p}}_{\text{arr}}(c, c', d) := \tau^{\text{p}}_{\text{trans}}(c, c') + \tau^{\text{p}}_{\text{wait}}(c, c') + \tau^{\text{p}}_{\text{arr}}(c', d)$$

$$\mathcal{R}(c) := \{c' \in \mathcal{C} \mid \tau_{\text{wait}}(c, c') \geq 0\}$$

$$\mathcal{R}_{\text{opt}}(c) := \{c' \in \mathcal{R}(c) \mid \forall \bar{c} \in \mathcal{R}(c) : \tau_{\text{wait}}(c, \bar{c}) \geq \tau_{\text{wait}}(c, c') \Rightarrow \tau^{\text{p}}_{\text{arr}}(c, \bar{c}, d) \geq \tau^{\text{p}}_{\text{arr}}(c, c', d)\}$$

$$\langle c_1, \ldots, c_k \rangle \text{ with } \forall i \in [1, k] : c_i \in \mathcal{R}_{\text{opt}}(c) \wedge \forall i \in [2, k] : \tau_{\text{wait}}(c, c_i) \geq \tau_{\text{wait}}(c, c_{i-1})$$

$$\tau^{c}_{\text{wait}}(i) := \begin{cases} \tau_{\text{wait}}(c, c_i) & \text{if } i \in [1, k] \\ -\infty & \text{otherwise} \end{cases}$$

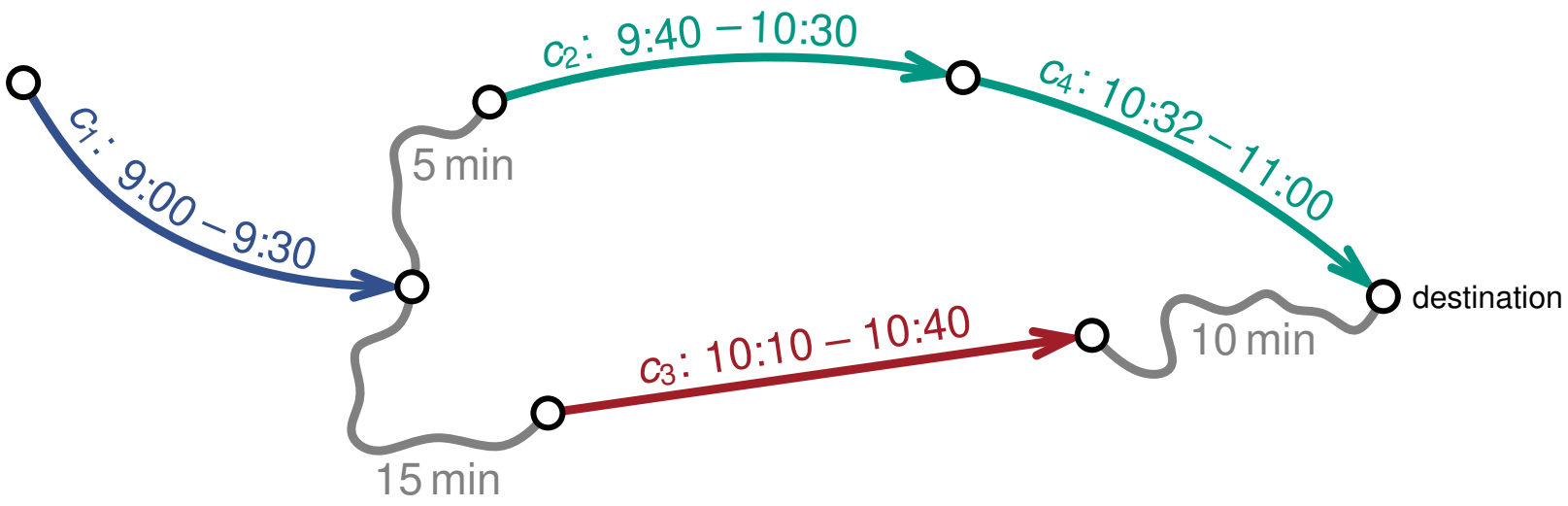
$$\tau^{\text{p}}_{\text{arr}}(c, d \mid \text{trans}) := \begin{cases} \sum_{i=1}^{k} \left(\frac{P[\tau^{c}_{\text{wait}}(i-1) < \Delta^{c}_{\tau} \leq \tau^{c}_{\text{wait}}(i)]}{P[\Delta^{c}_{\tau} \leq \tau^{c}_{\text{wait}}(k)]} \cdot \tau^{\text{p}}_{\text{arr}}(c, c_i, d) \right) & \text{if } k > 0 \\ \infty & \text{otherwise} \end{cases}$$

Beispiel PAT Berechnung

Beispiel:

- $\lambda_{walk} = 3, \quad \lambda_{wait} = 2, \quad \lambda_{trans} = 5\,\text{min}$

Connection	PAT
c_4	
c_3	
c_2	
c_1	



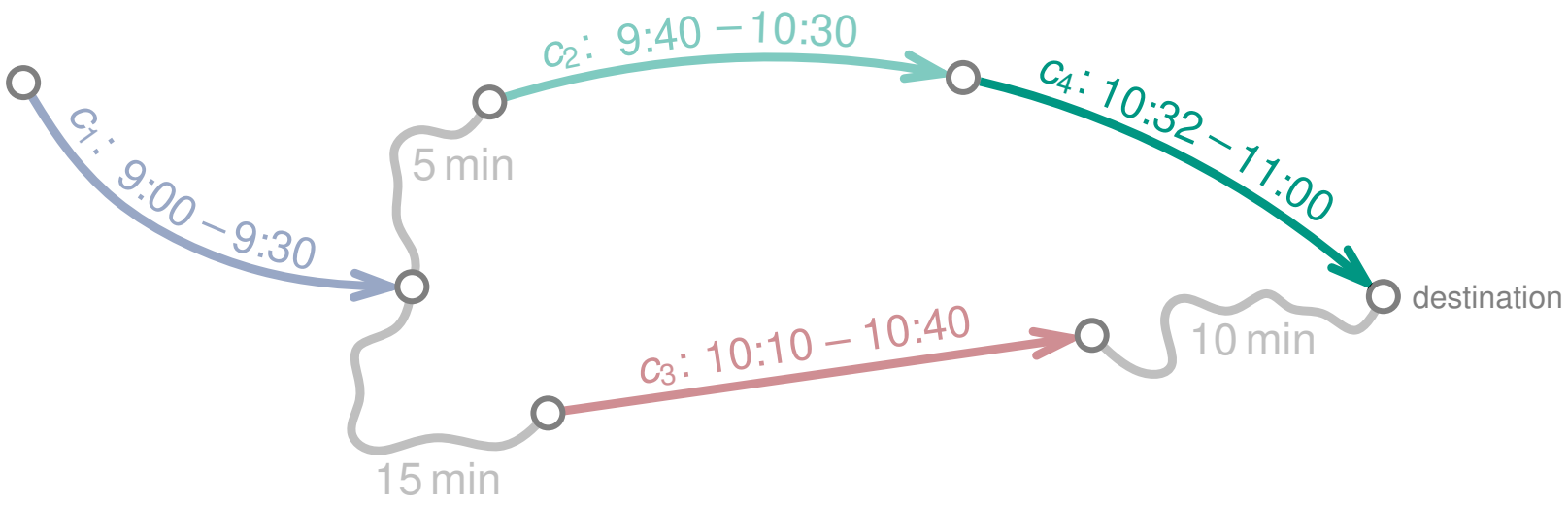
Beispiel PAT Berechnung

Beispiel:

- $\lambda_{walk} = 3, \quad \lambda_{wait} = 2, \quad \lambda_{trans} = 5\,\text{min}$
- **Fall 1:** Connection c erreicht Ziel

  $\Rightarrow$ PAT = arrival time $\tau_{arr}(c)$

Connection	PAT
c_4	11:00
c_3	
c_2	
c_1	



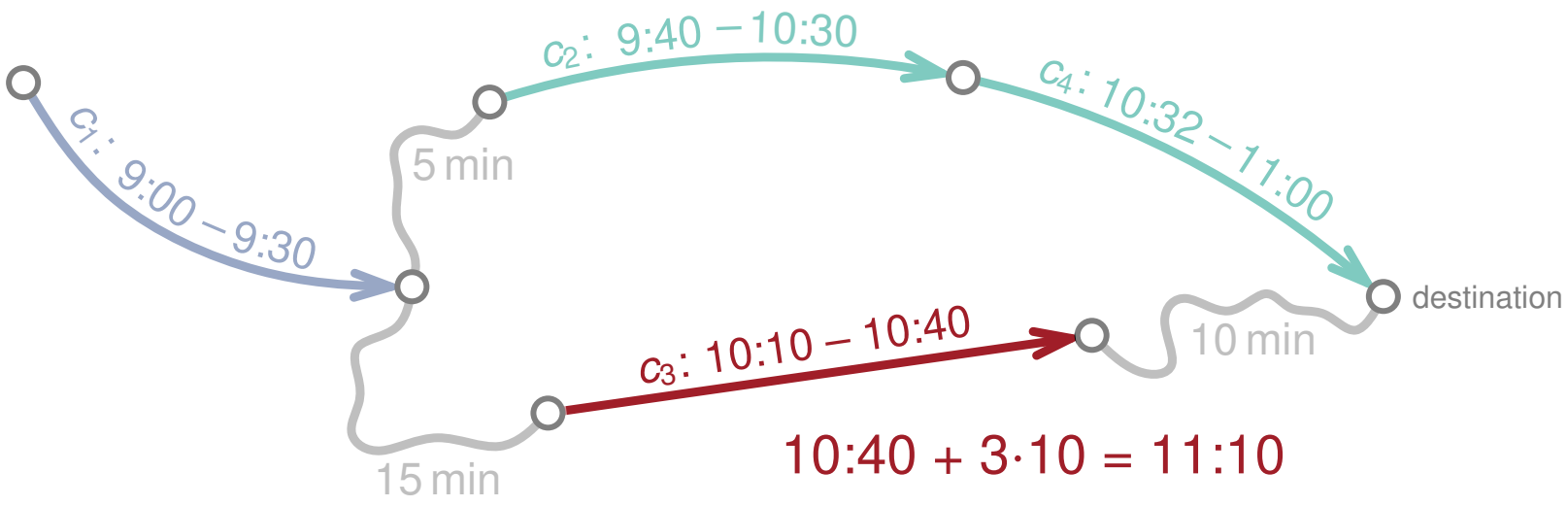
Beispiel PAT Berechnung

Beispiel:

- $\lambda_{walk} = 3, \quad \lambda_{wait} = 2, \quad \lambda_{trans} = 5\,\text{min}$
- **Fall 2:** Lauf von Connection c zum Ziel

  $\Rightarrow \text{PAT} = \tau_{arr}(c) + (\lambda_{walk} \cdot \tau_{walking})$

Connection	PAT
c_4	11:00
c_3	11:10
c_2	
c_1	



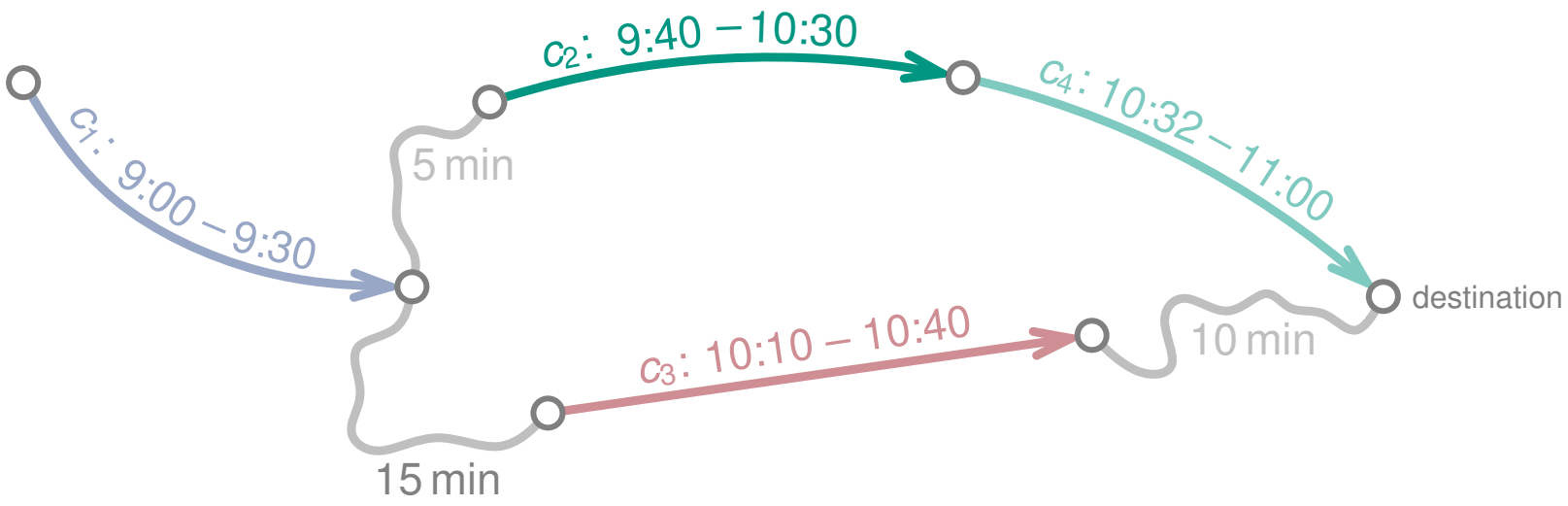
Beispiel PAT Berechnung

Beispiel:

- $\lambda_{\text{walk}} = 3, \quad \lambda_{\text{wait}} = 2, \quad \lambda_{\text{trans}} = 5\,\text{min}$
- **Fall 3:** Weiterfahren mit Con. c' (gleicher Trip)

  $\Rightarrow$ PAT = PAT c'

Connection	PAT
c_4	11:00
c_3	11:10
c_2	11:00
c_1	

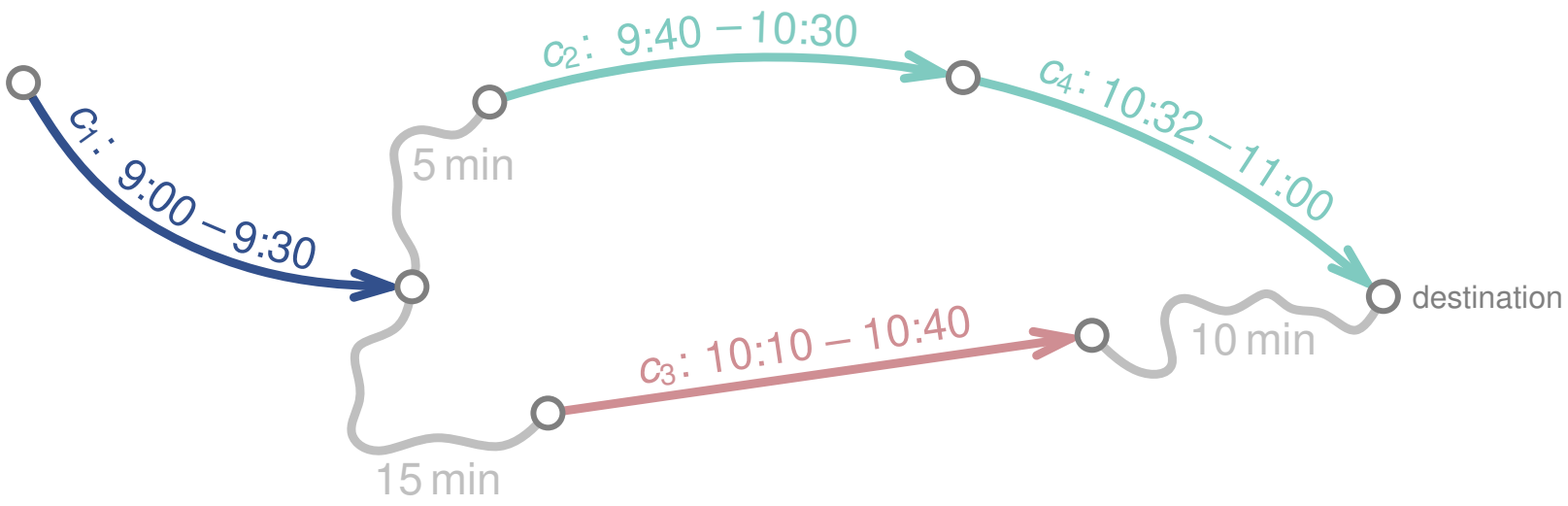


Beispiel PAT Berechnung

Beispiel:

- $\lambda_{\text{walk}} = 3, \quad \lambda_{\text{wait}} = 2, \quad \lambda_{\text{trans}} = 5\,\text{min}$
- **Fall 4:** Weiterfahren mit Con. c' (anderer Trip)

Connection	PAT
c_4	11:00
c_3	11:10
c_2	11:00
c_1	

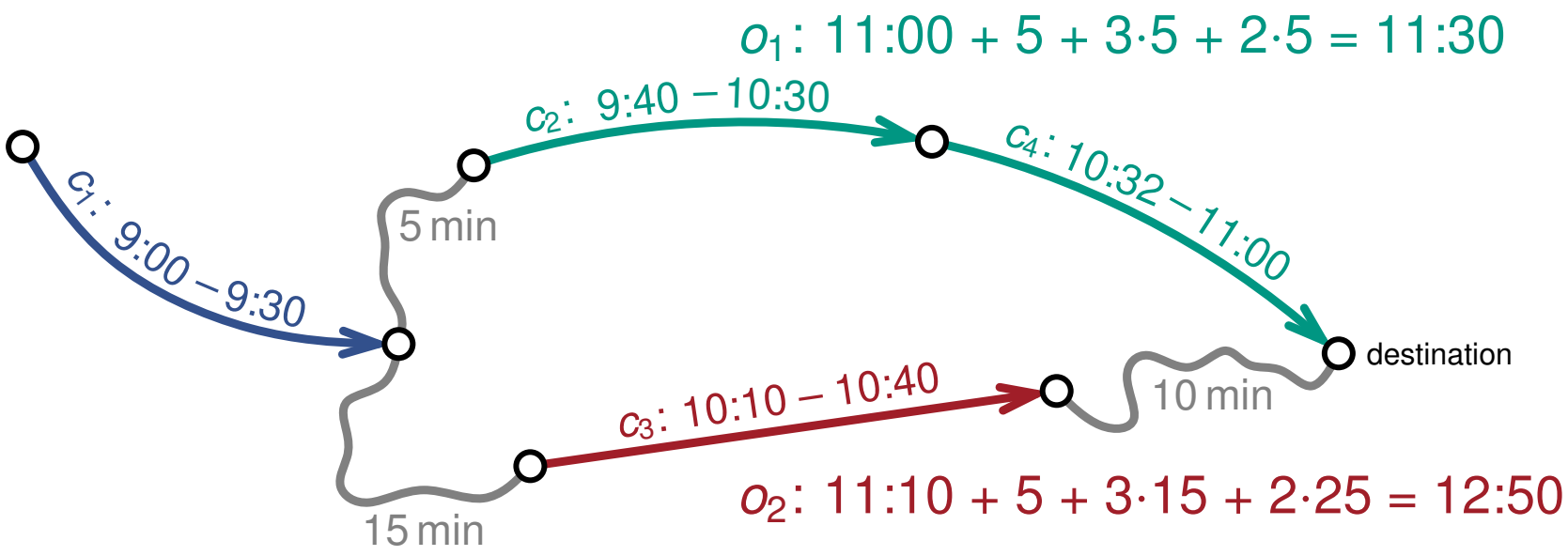


Beispiel PAT Berechnung

Beispiel:

- $\lambda_{walk} = 3, \quad \lambda_{wait} = 2, \quad \lambda_{trans} = 5\,\text{min}$
- **Fall 4:** Weiterfahren mit Con. c' (anderer Trip)

Connection	PAT
c_4	11:00
c_3	11:10
c_2	11:00
c_1	



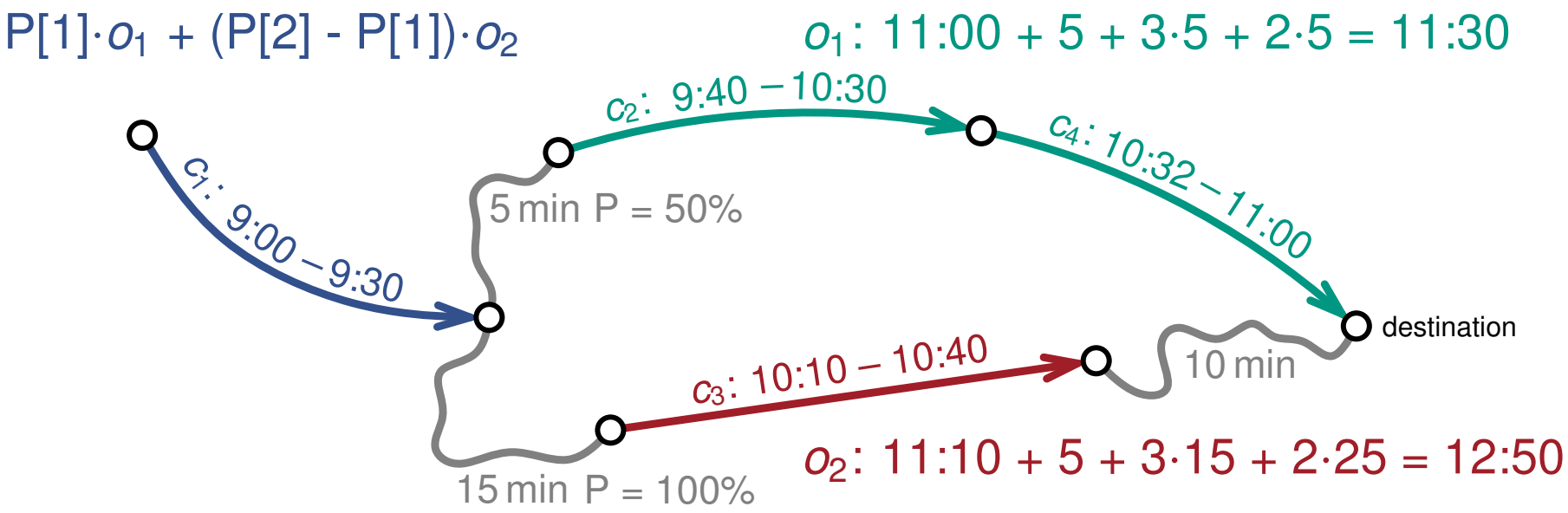
Beispiel PAT Berechnung

Beispiel:

- $\lambda_{walk} = 3, \quad \lambda_{wait} = 2, \quad \lambda_{trans} = 5\,\text{min}$
- **Fall 4:** Weiterfahren mit einer Option o_i

  $\Rightarrow \text{PAT} = \sum_i \left(\text{transfer probability}(o_i) \cdot o_i\right)$

Connection	PAT
c_4	11:00
c_3	11:10
c_2	11:00
c_1	



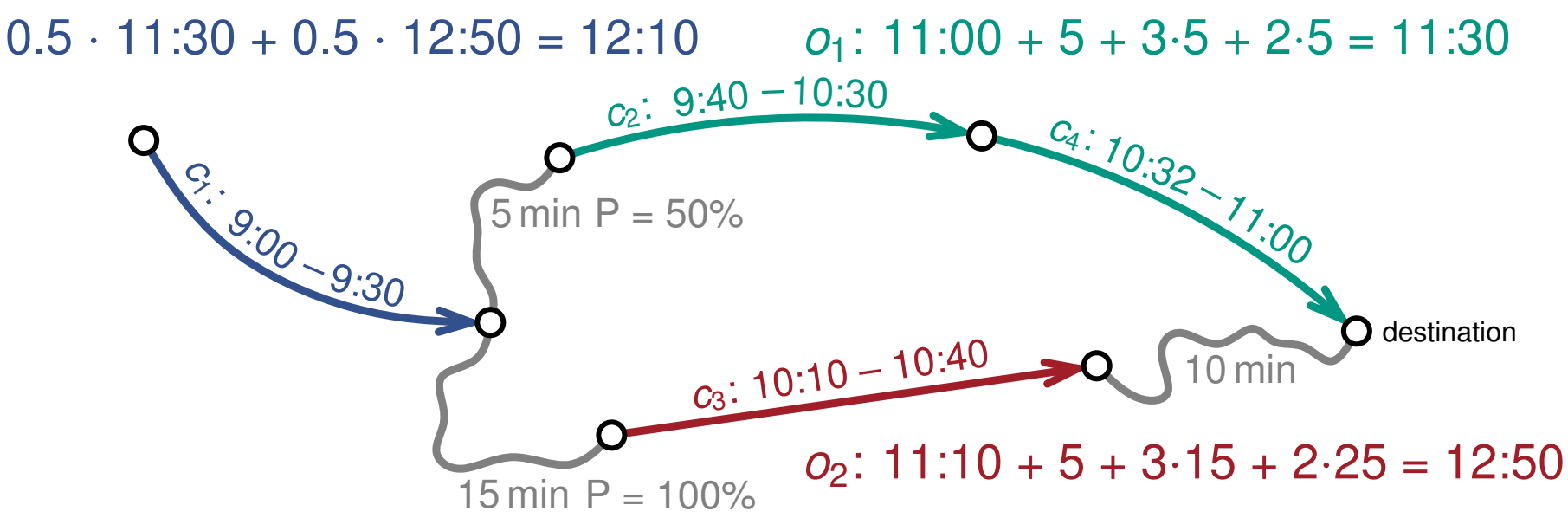
Beispiel PAT Berechnung

Beispiel:

- $\lambda_{walk} = 3, \quad \lambda_{wait} = 2, \quad \lambda_{trans} = 5\,\text{min}$
- **Fall 4:** Weiterfahren mit einer Option o_i

$$\Rightarrow \text{PAT} = \sum_i \left(\text{transfer probability}(o_i) \cdot o_i\right)$$

Connection	PAT
c_4	11:00
c_3	11:10
c_2	11:00
c_1	12:10



Umlegungsalgorithmus

Ansatz:

- Simuliere Bewegung der Passagiere im Netzwerk
- Entscheide pro Connection c, wer c benutzt
- Passagiere mit selbem Ziel werden sich treffen
  $\Rightarrow$ Müssen die selben Entscheidungen treffen
  $\Rightarrow$ Algorithmus profitiert von Synergieeffekten

Umlegungsalgorithmus

Ansatz:

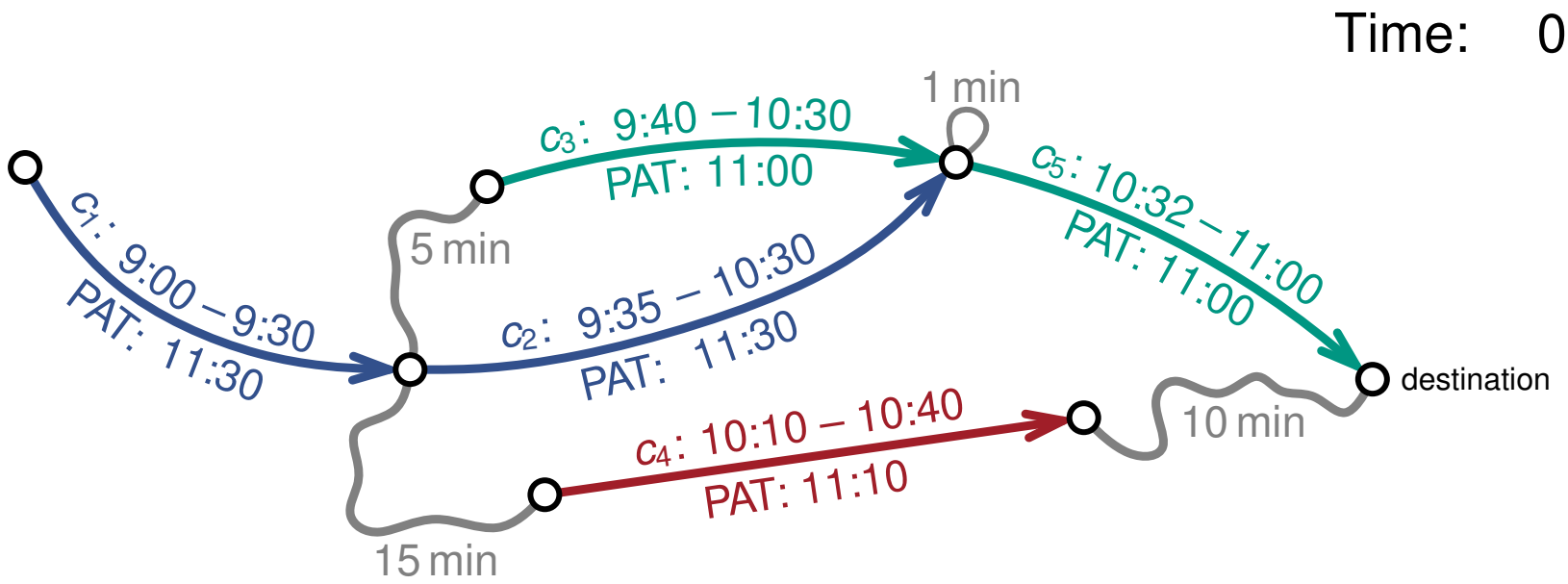
- Simuliere Bewegung der Passagiere im Netzwerk
- Entscheide pro Connection c, wer c benutzt
- Passagiere mit selbem Ziel werden sich treffen
  $\Rightarrow$ Müssen die selben Entscheidungen treffen
  $\Rightarrow$ Algorithmus profitiert von Synergieeffekten

Überblick:

- Sortiere Passagiere nach Zielstop
- Berechne Umlegung pro Zielstop in 3 Schritten:
  - Berechne PATs für jede Connection
  - Simuliere Bewegung der Passagiere basierend auf PATs
  - Entferne überflüssige Kreise aus Journeys (optional)

Beispiel Umlegungsberechnung

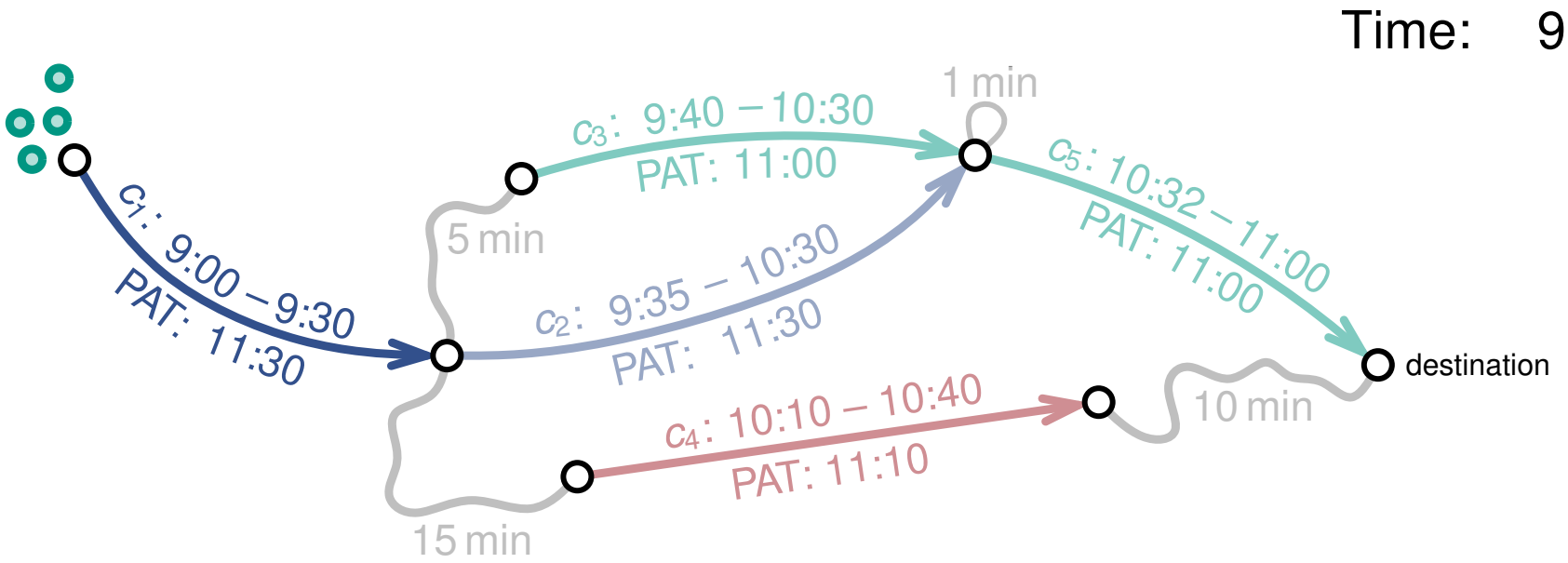
- Bearbeite Connections chronologisch (nach Abfahrtszeit)
- Entscheide welche Passagiere die Connection benutzen



Beispiel Umlegungsberechnung

- Bearbeite Connections chronologisch (nach Abfahrtszeit)
- Entscheide welche Passagiere die Connection benutzen

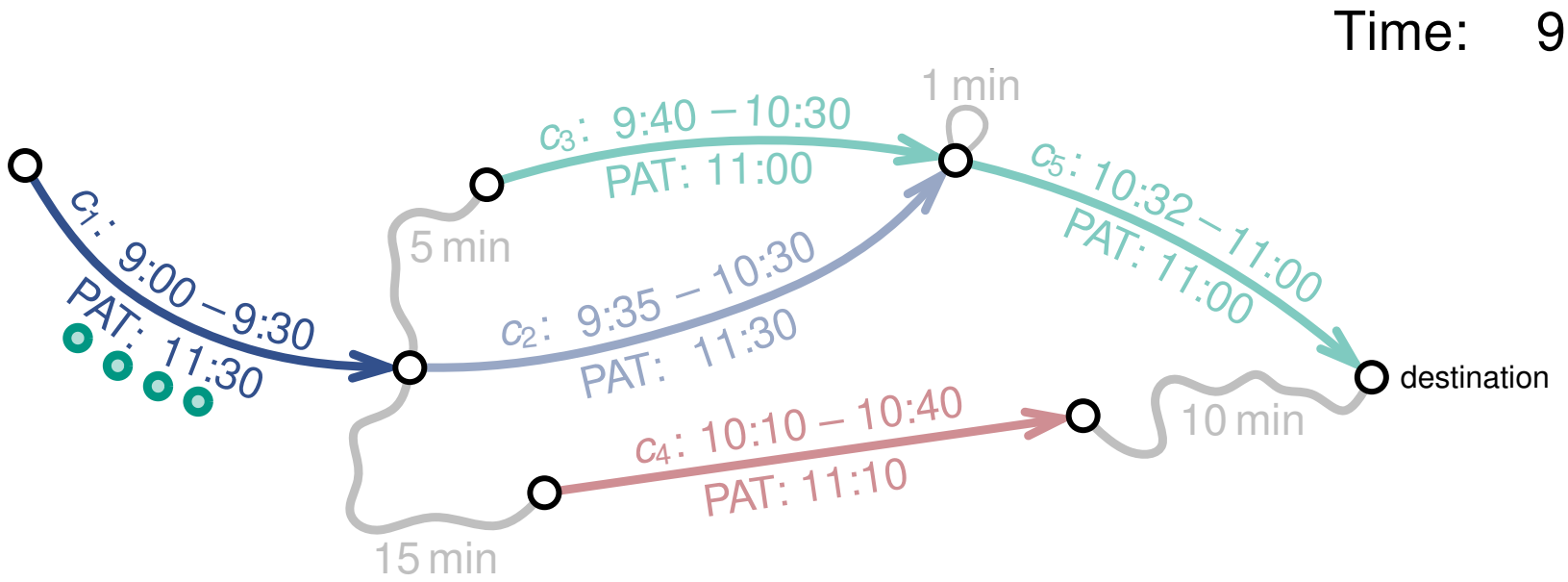
1. Erzeuge Passagiere entsprechend der Nachfrage



Beispiel Umlegungsberechnung

- Bearbeite Connections chronologisch (nach Abfahrtszeit)
- Entscheide welche Passagiere die Connection benutzen

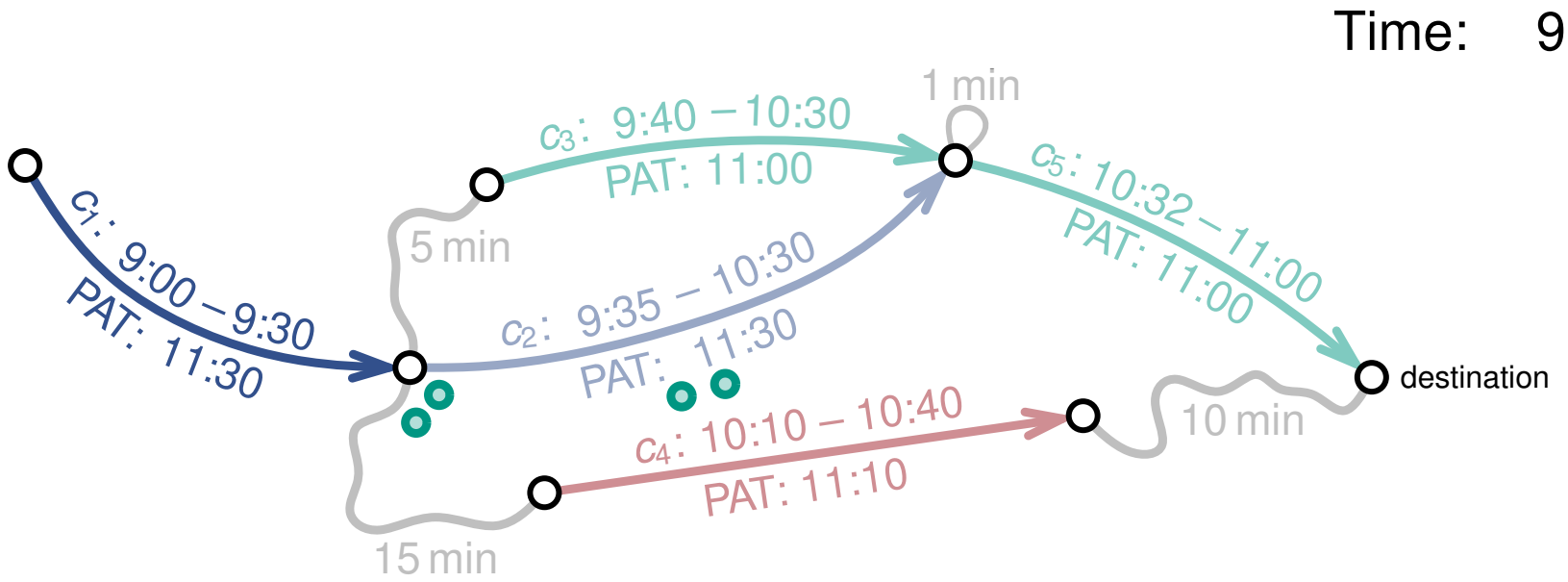
2. Entscheide welche Passagiere einsteigen



Beispiel Umlegungsberechnung

- Bearbeite Connections chronologisch (nach Abfahrtszeit)
- Entscheide welche Passagiere die Connection benutzen

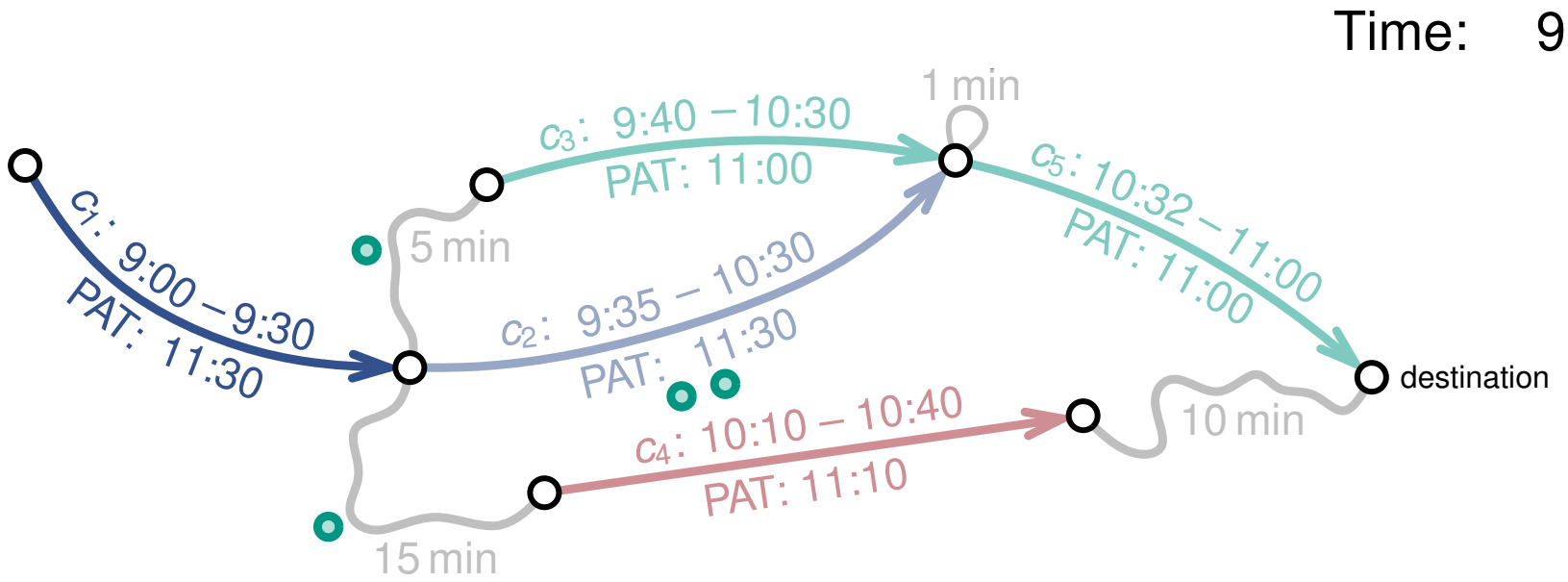
3. Entscheide welche Passagiere aussteigen



Beispiel Umlegungsberechnung

- Bearbeite Connections chronologisch (nach Abfahrtszeit)
- Entscheide welche Passagiere die Connection benutzen

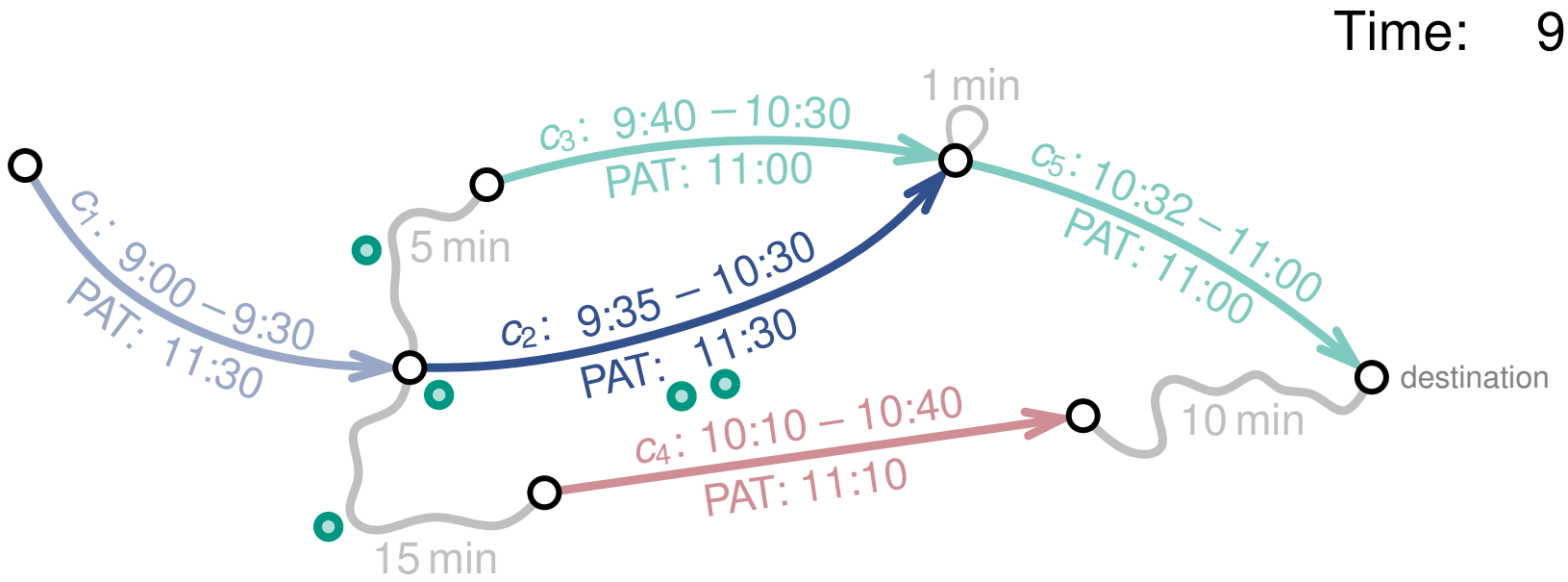
4. Verschiebe ausgestiegene Passagiere zum nächsten Stop



Beispiel Umlegungsberechnung

- Bearbeite Connections chronologisch (nach Abfahrtszeit)
- Entscheide welche Passagiere die Connection benutzen

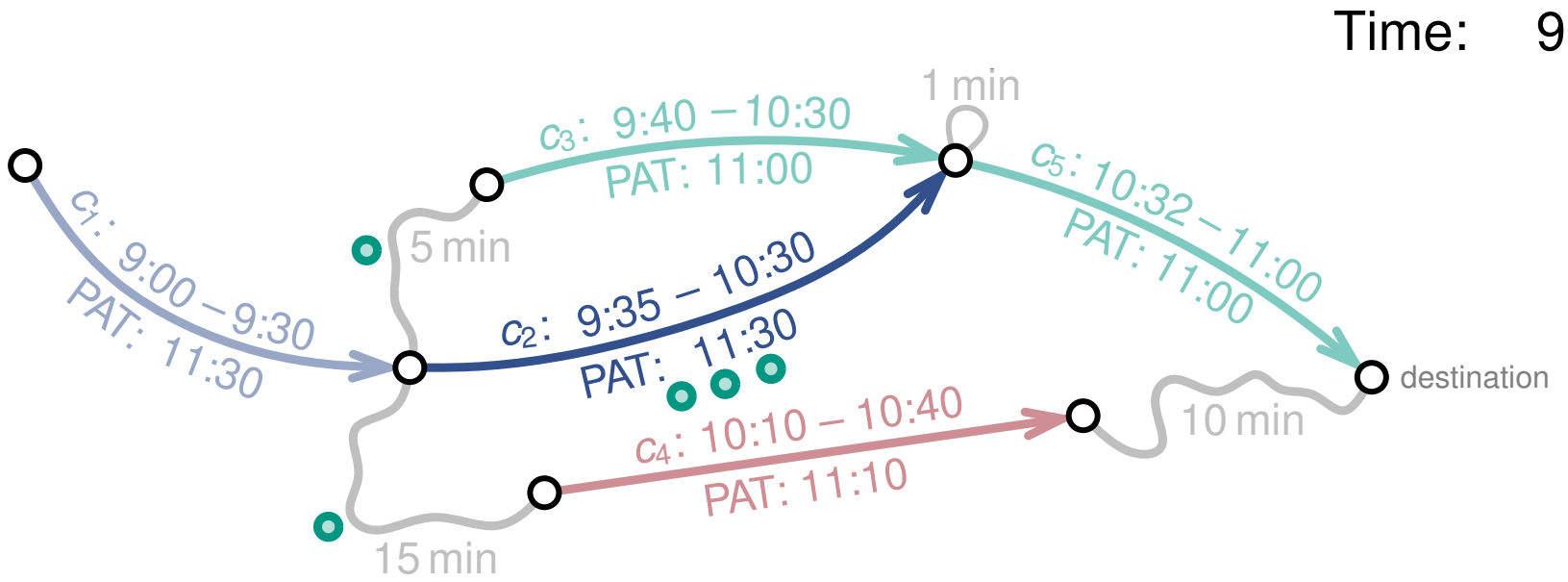
1. Erzeuge Passagiere entsprechend der Nachfrage



Beispiel Umlegungsberechnung

- Bearbeite Connections chronologisch (nach Abfahrtszeit)
- Entscheide welche Passagiere die Connection benutzen

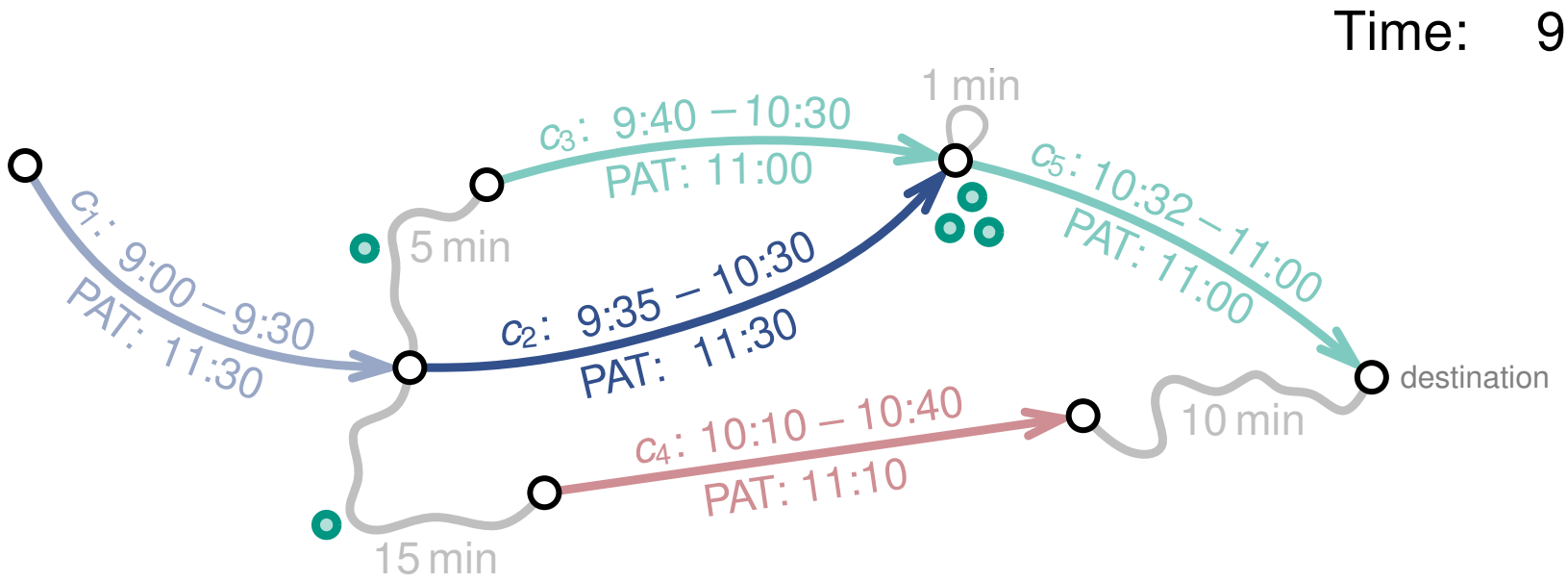
2. Entscheide welche Passagiere einsteigen



Beispiel Umlegungsberechnung

- Bearbeite Connections chronologisch (nach Abfahrtszeit)
- Entscheide welche Passagiere die Connection benutzen

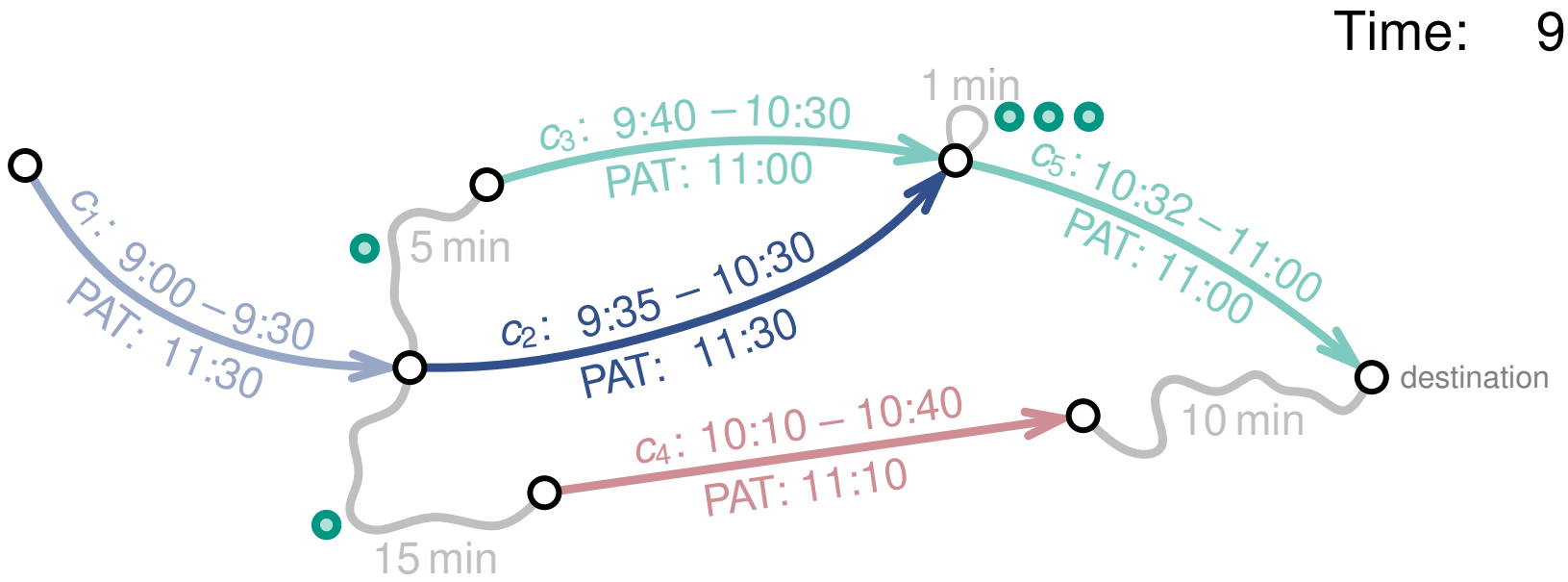
3. Entscheide welche Passagiere aussteigen



Beispiel Umlegungsberechnung

- Bearbeite Connections chronologisch (nach Abfahrtszeit)
- Entscheide welche Passagiere die Connection benutzen

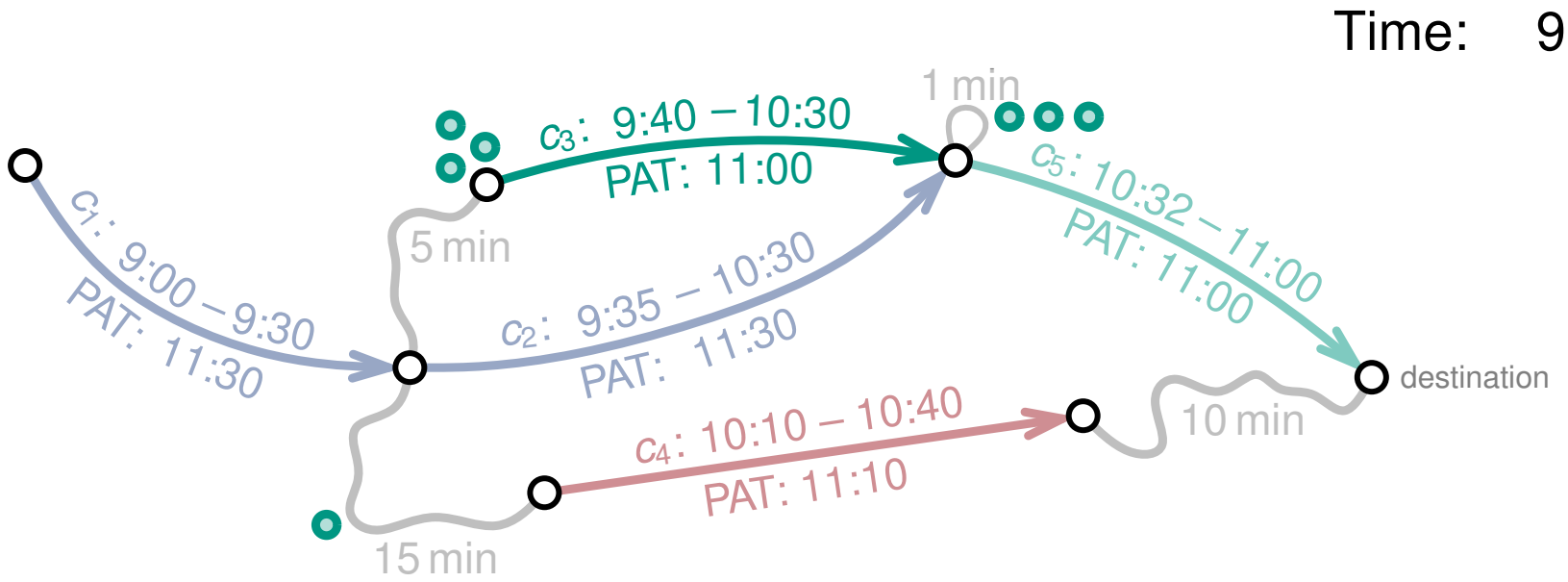
4. Verschiebe ausgestiegene Passagiere zum nächsten Stop



Beispiel Umlegungsberechnung

- Bearbeite Connections chronologisch (nach Abfahrtszeit)
- Entscheide welche Passagiere die Connection benutzen

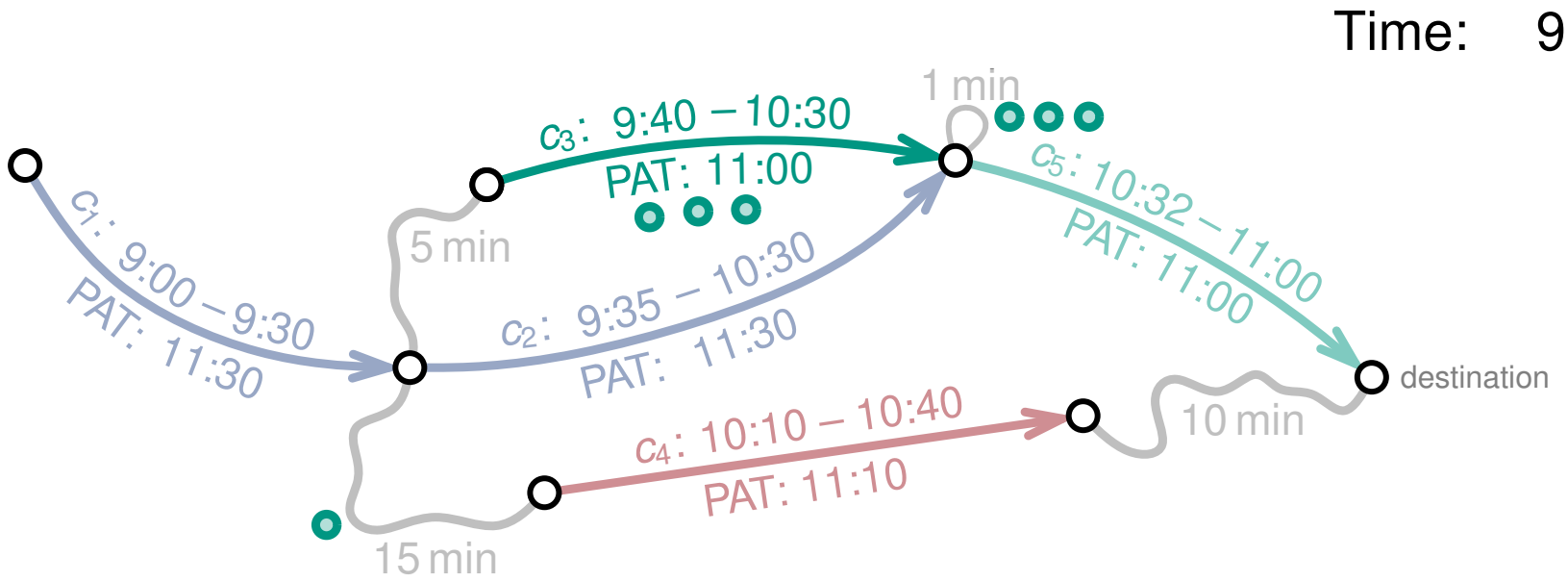
1. Erzeuge Passagiere entsprechend der Nachfrage



Beispiel Umlegungsberechnung

- Bearbeite Connections chronologisch (nach Abfahrtszeit)
- Entscheide welche Passagiere die Connection benutzen

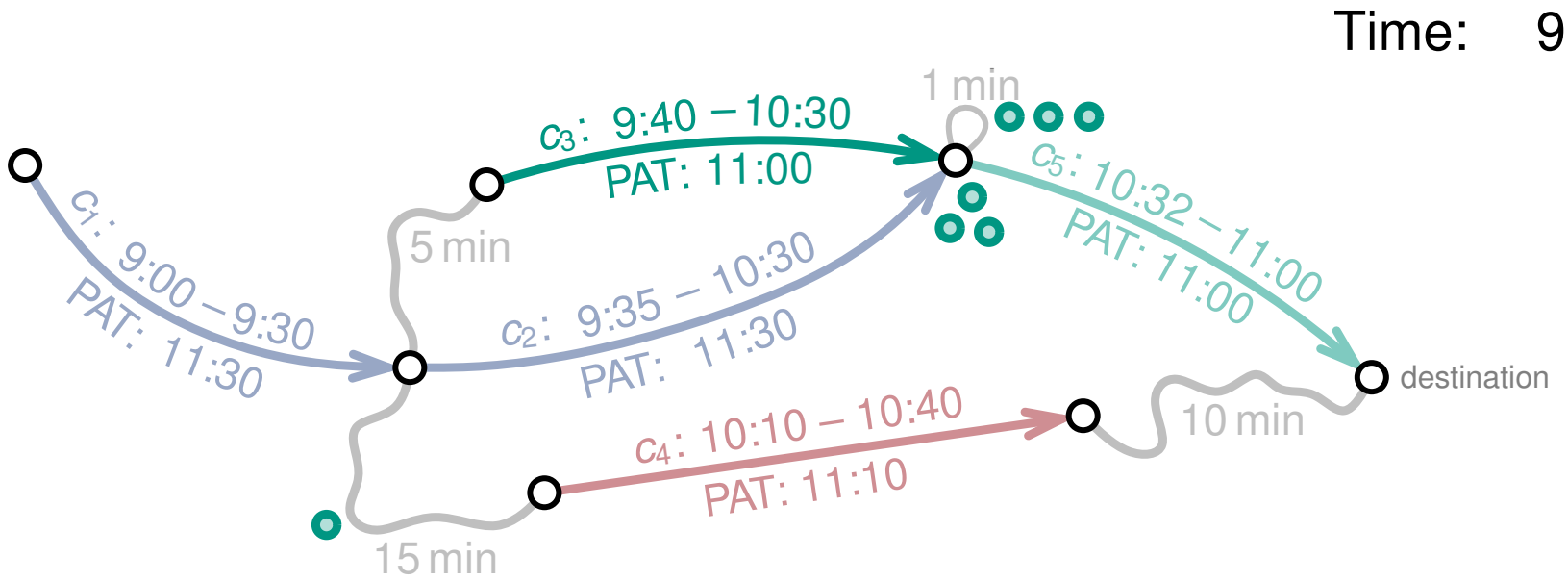
2. Entscheide welche Passagiere einsteigen



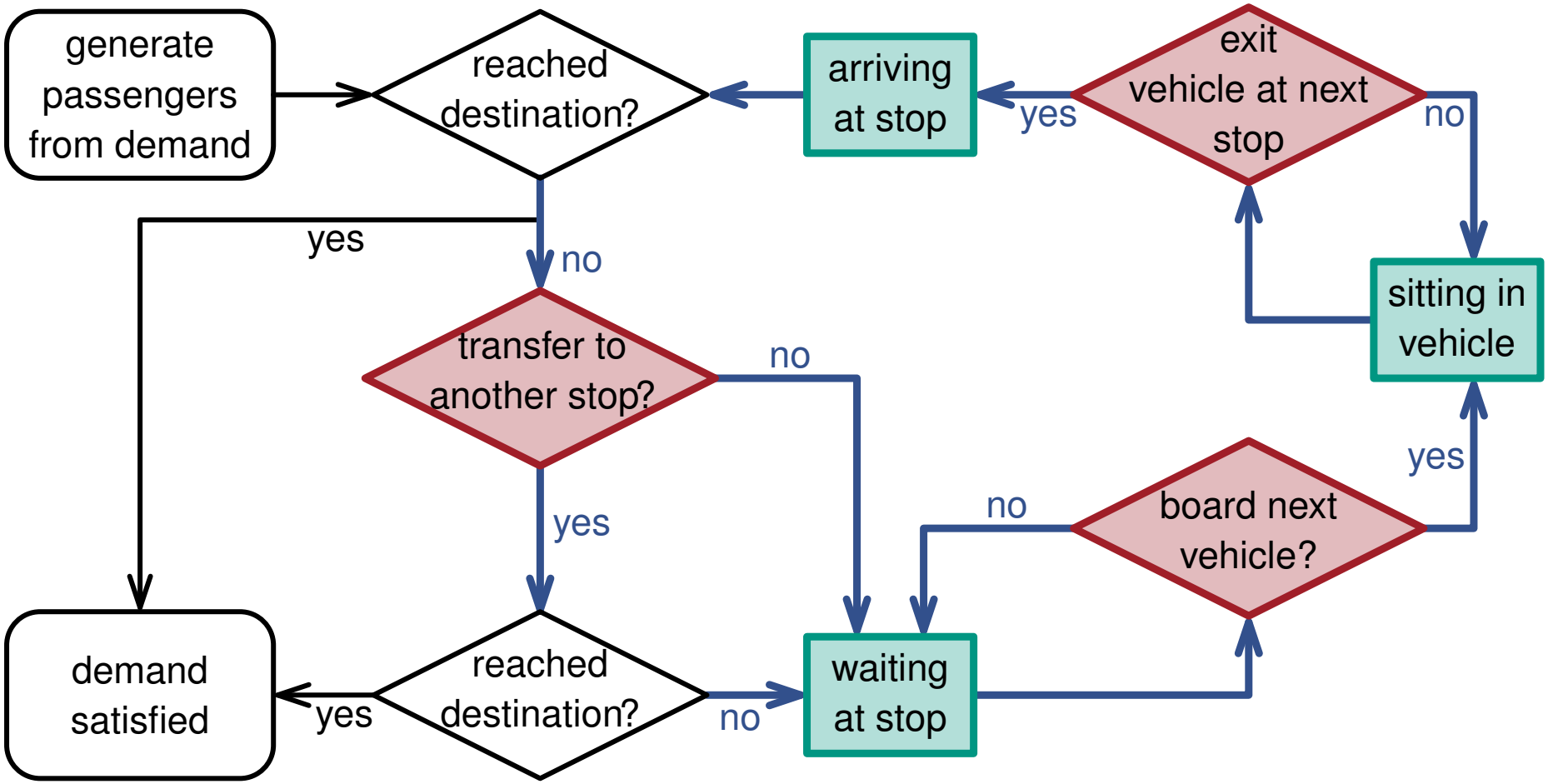
Beispiel Umlegungsberechnung

- Bearbeite Connections chronologisch (nach Abfahrtszeit)
- Entscheide welche Passagiere die Connection benutzen

3. ...



Umlegungsberechnung Übersicht



Entscheidungsmodell

Ziel:

- Entscheidet welche Connection ein Passagier nimmt
- Hängt von der Verspätungstoleranz $\lambda_{\Delta max}$ des Passagieres ab

Entscheidungsmodell

Ziel:

- Entscheidet welche Connection ein Passagier nimmt
- Hängt von der Verspätungstoleranz $\lambda_{\Delta\max}$ des Passagieres ab

Definition:

- Gegeben sind die Optionen $o_1, \ldots, o_k$ und ihre PATs
- Bestimme den Nutzen $g(i)$ jeder Option:

$$g(i) := \max\left(0, \min_{j \neq i}(PAT(o_j)) - PAT(o_i) + \lambda_{\Delta\max}\right)$$

- Die Wahrscheinlichkeit $P[i]$, dass ein Passagier Option i wählt, ist:

$$P[i] := \frac{g(i)}{\sum_{j=1}^{k} g(j)}$$

Kreise

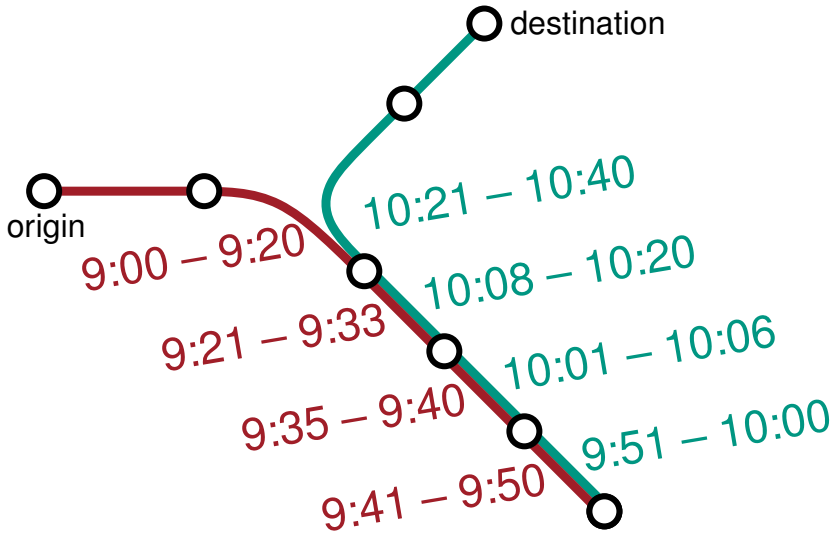
Kreis Definition:

- Den selben Stop mehrfach besuchen
- Umlegungen mit Kreisen können unerwünscht sein
- Eine Journey mit Kreisen kann optimal bezüglich PAT sein
- Hohe Wartekosten können zu Kreisen führen

Kreise

Kreis Definition:

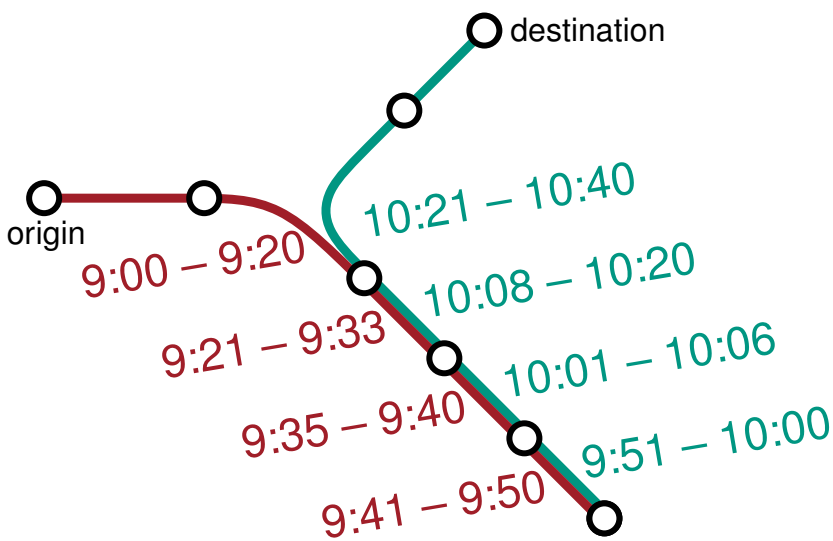
- Den selben Stop mehrfach besuchen
- Umlegungen mit Kreisen können unerwünscht sein
- Eine Journey mit Kreisen kann optimal bezüglich PAT sein
- Hohe Wartekosten können zu Kreisen führen

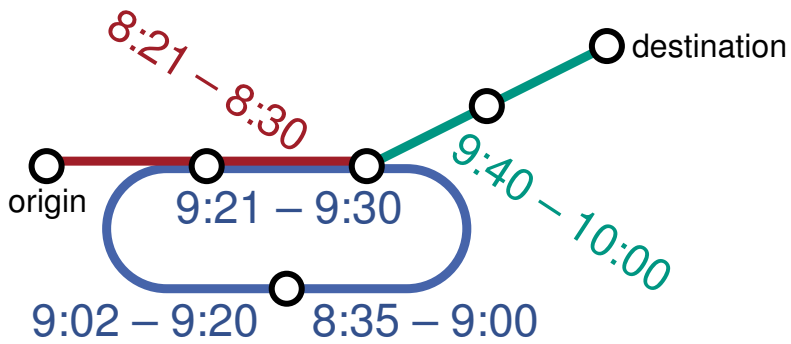


Kreise

Kreis Definition:

- Den selben Stop mehrfach besuchen
- Umlegungen mit Kreisen können unerwünscht sein
- Eine Journey mit Kreisen kann optimal bezüglich PAT sein
- Hohe Wartekosten können zu Kreisen führen



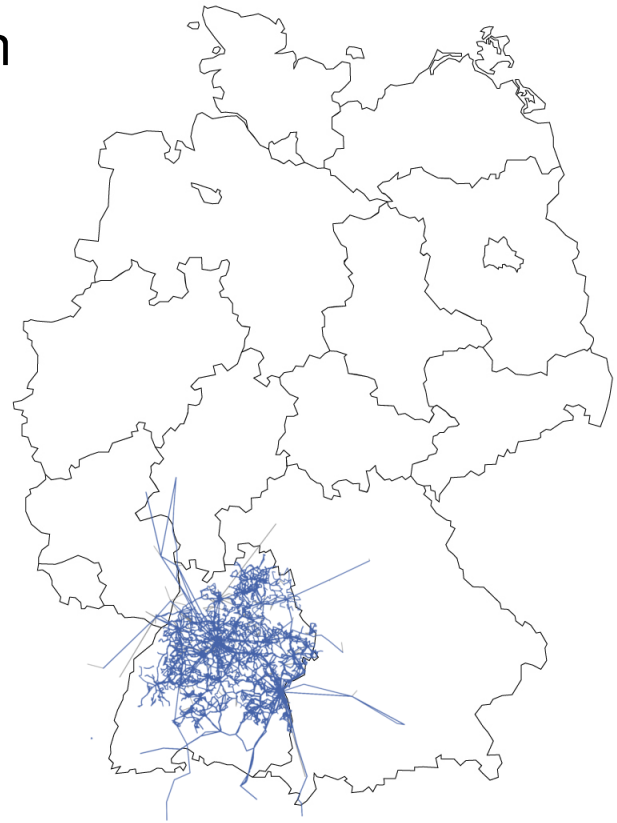


Auswertung

Instanzen:

- Großraum Stuttgart
- Enthält auch Frankfurt, Basel und München
- Beschreibt den Verkehr eines Tages

Anzahl Knoten	15 115
Anzahl Stops	13 941
Anzahl Kanten	33 890
Anzahl Kanten ohne Schlaufen	18 775
Anzahl Connections	780 042
Anzahl Trips	47 844
Anzahl Passagiere	1 249 910



Auswertung – Laufzeiten

Benutzte Parameter:

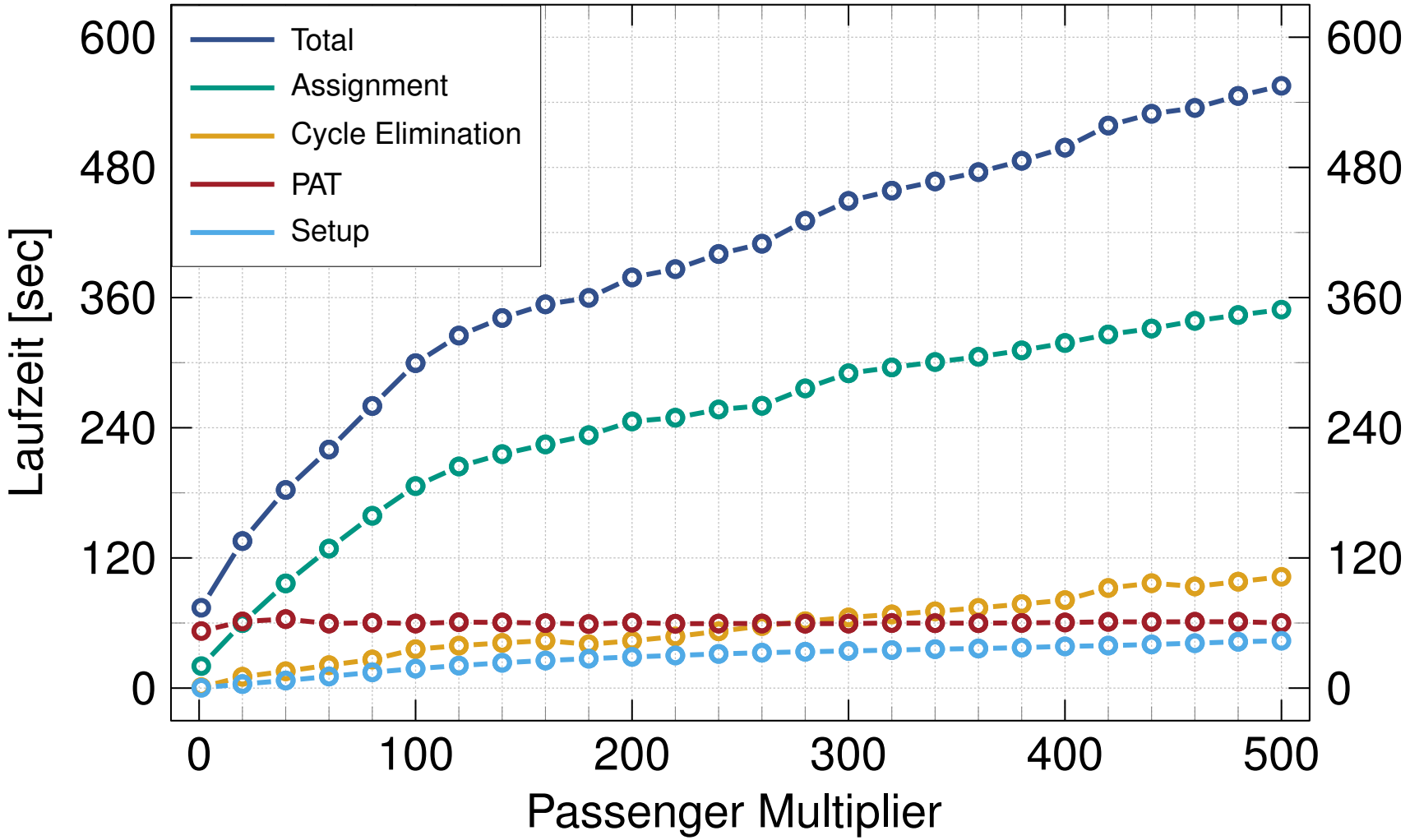
- Laufkosten $\lambda_{\text{walk}} = 2$
- Wartekosten $\lambda_{\text{wait}} = 0.5$
- Umstiegskosten $\lambda_{\text{trans}} = 5\,\text{min}$
- Verspätungstoleranz $\lambda_{\Delta\text{max}} = 5\,\text{min}$
- Maximale erwartete Verspätung $\Delta_{\tau}^{\text{max}} = 1\,\text{min}$

Laufzeitvergleich:

- Laufzeit VISUM $\approx 30\,\text{min}$ (mit 8 Threads)
- PAT basierte Umlegung: (mit passenger multiplier = 10)

Anzahl Threads	1	2	4
Laufzeit [sec]	108.92	65.57	38.41

Auswertung – Laufzeiten



Auswertung – Umlegungsqualität

- Beide Umlegungen sind sehr ähnlich
- VISUM berechnet etwas kürzere Fahrzeiten
- PAT basierter Algorithmus berechnet Journeys mit weniger Umstiegen

	VISUM			PAT basierter Algorithmus		
Eingenschaft	min	mean	max	min	mean	max
Reisezeit [min]	2.98	46.885	429.00	2.98	47.199	429.00
Zeit im Fahrzeug [min]	0.02	21.059	380.00	0.02	21.231	323.97
Laufzeit [min]	2.00	22.394	149.00	2.00	22.476	149.00
Wartezeit [min]	0.00	3.432	217.02	0.00	3.492	217.02
Züge pro Passagier	1.00	1.771	6.00	1.00	1.746	8.00
Connections pro Passagier	1.00	9.396	109.00	1.00	9.474	97.00
Passagiere pro Connection	0.00	12.740	1 290.10	0.00	12.847	1 233.60