Karlsruher Institut für Technologie

Algorithmen für Routenplanung

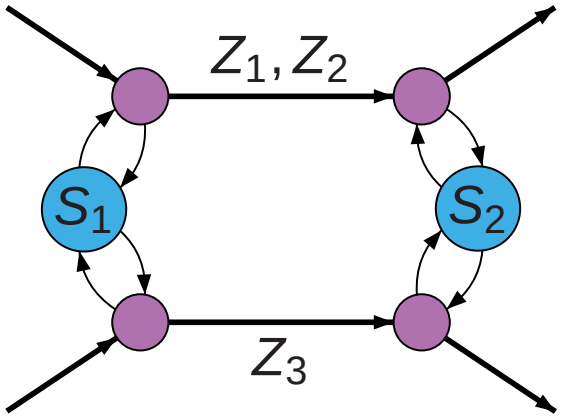
17. Sitzung, Sommersemester 2012

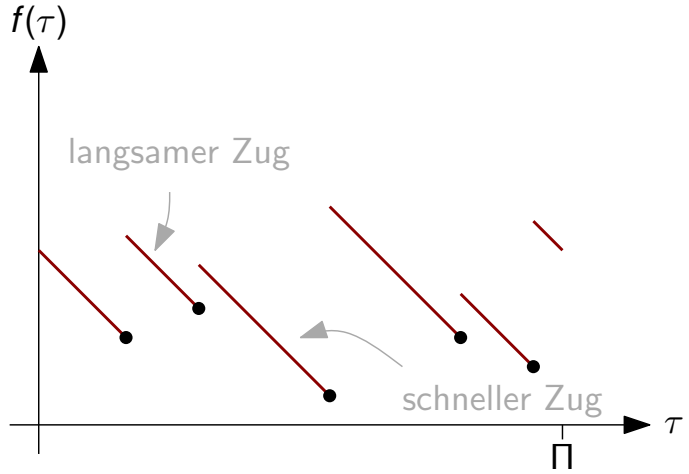
Thomas Pajor | 2. Juli 2012

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

Letztes Mal...

Einführung in die Fahrplanauskunft





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

Wiederholung

Wiederholung: Szenarien

- One-to-All Query (Vorberechnung) vs.
  One-to-One Query (Anfrage)
- Time-Query vs. Profile-Query

Zur Erinnerung:

- Dijkstra's Algorithmus: One-to-All Query
- Stoppkriterium: One-to-One Query
- Grundlage für (fast) alle Beschleunigungstechniken
- Profilsuchen: Label-Correcting statt Label-Setting

Ergebnisse

Technik	Preprocessing time [h:m]	Preprocessing space [B/n]	Preprocessing edge inc.	Time-Queries time [ms]	Time-Queries speed up	Profile-Queries time [ms]	Profile-Queries speed up
TIME-DEPENDENT APPROACH							
Dijkstra	0:00	0	0 %	125.2	1.0	5 327	1.0
uALT	0:02	128	0 %	75.3	1.7	4 384	1.2
eco-SHARC	1:30	113	74 %	17.5	7.2	988	5.4
gen-SHARC	12:15	120	74 %	4.7	26.6	273	19.5
CH*	0:08	87	86 %	0.6	209	9	591
TIME-EXPANDED APPROACH							
Dijkstra	0:00	0	0 %	534.7	1.0	—	—
uALT**	0:01	4	0 %	52.8	10.1	—	—
uALT+AF**	83:58	128	0 %	9.5	56.3	—	—

* Auf Stationsgraph (Transfers durch Algo sichergestellt)
Kontraktion auf real. Graph nicht möglich (Kantenexplosion)

** Optimiertes Routen-Modell

Beschleunigungstechniken

Beobachtung:
Techniken für Straßennetzwerke funktionieren nicht gut

Gründe (Intuition):

- Weniger Hierarchie (insbes. Busnetze)
- Kontraktion lässt Knotengrad explodieren
- Lokale Anfragen sehr “teuer”

$\rightsquigarrow$ “15-Stunden zum nächsten Dorf Problem”

- Lower- und Upper-Bounds weit auseinander

Beschleunigungstechniken

Beobachtung:
Techniken für Straßennetzwerke funktionieren nicht gut

Gründe (Intuition):

- Weniger Hierarchie (insbes. Busnetze)
- Kontraktion lässt Knotengrad explodieren
- Lokale Anfragen sehr “teuer”

⇝ “15-Stunden zum nächsten Dorf Problem”

- Lower- und Upper-Bounds weit auseinander

Jetzt: Erster Ansatz zur Beschleunigung von Profil-Anfragen

- Ausnutzen der Struktur von Schienenfunktionen
- Parallelisierung
- Stoppkriterium
- Distanztabellen

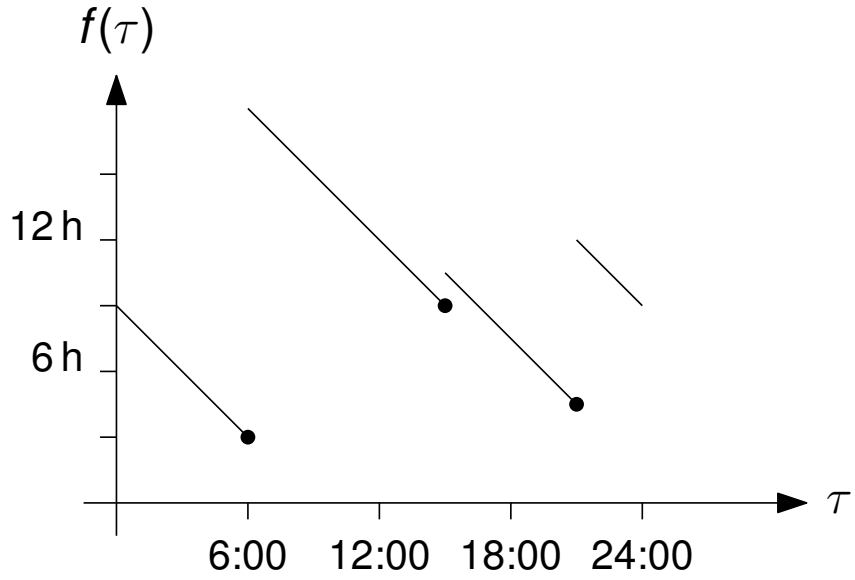
Analyse: Schienenfunktionen

Verbindungen modelliert durch stückweise lineare Funktionen

Verbindungen zw. S_i und S_j:

id	dep.-time	travel-time
1	06:00	3 h 00 min
2	15:00	9 h 00 min
3	21:00	4 h 30 min
⋮	⋮	⋮

Entsprechende Funktion:



- Für jede Verbindung: Connection Point (τ, w)
  $\tau \mathrel{\hat{=}}$ Abfahrtszeit, $w \mathrel{\hat{=}}$ Reisezeit
- Zwischen Verbindungen: Lineares Warten

Profil-Anfragen (One-to-All)

Gegeben:
Zeitabhängiges Netzwerk $G = (V, E)$ und Startbahnhof S.

Profil-Anfragen (One-to-All)

Gegeben:
Zeitabhängiges Netzwerk $G = (V, E)$ und Startbahnhof S.

Problem (Profil-Anfrage):
Berechne die *Reisezeitfunktion* $\text{dist}_S(v, \tau)$, so dass $\text{dist}_S(v, \tau)$ die Länge des kürzesten Weges von S nach v in G zur Abfahrtszeit τ an S für alle $\tau \in \Pi$ und $v \in V$ ist.

Profil-Anfragen (One-to-All)

Gegeben:
Zeitabhängiges Netzwerk $G = (V, E)$ und Startbahnhof S.

Problem (Profil-Anfrage):
Berechne die *Reisezeitfunktion* $\text{dist}_S(v, \tau)$, so dass $\text{dist}_S(v, \tau)$ die Länge des kürzesten Weges von S nach v in G zur Abfahrtszeit τ an S für alle $\tau \in \Pi$ und $v \in V$ ist.

Bisheriger Ansatz:
Erweitere Dijkstra's Algorithmus zu Label-Correcting Algorithmus

- Benutze Funktionen statt Konstanten
- Verliert Label-Setting Eigenschaft von Dijkstra
- Deutlich langsamer als Dijkstra ($\approx$ Faktor 50)

Hauptidee

Beobachtung:
Jeder Reiseplan ab S (irgendwohin) beginnt mit einer Verb. an S.

Hauptidee

Beobachtung:
Jeder Reiseplan ab S (irgendwohin) beginnt mit einer Verb. an S.

Naiver Ansatz

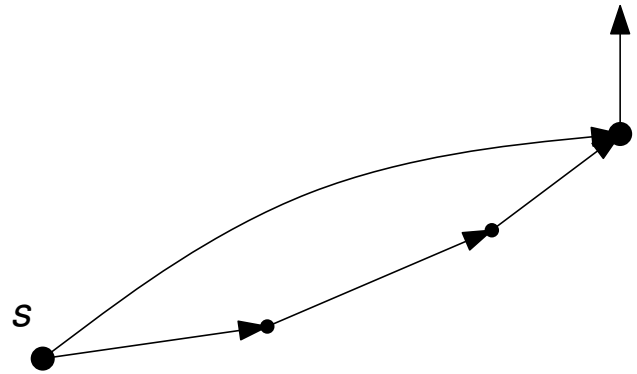
Für jede ausgehende Verbindung c_i an S:
Separate Zeitanfrage mit Abfahrtszeit $\tau_{\text{dep}}(c_i)$.

Nachteile

- Zu viele redundante Berechnungen
  Periodische Natur der Fahrpläne
- Nicht jede Verbindung ab S trägt zu $\text{dist}_S(v, \cdot)$ bei
  Langsame Züge für weite Reisen machen wenig Sinn

Self-Pruning

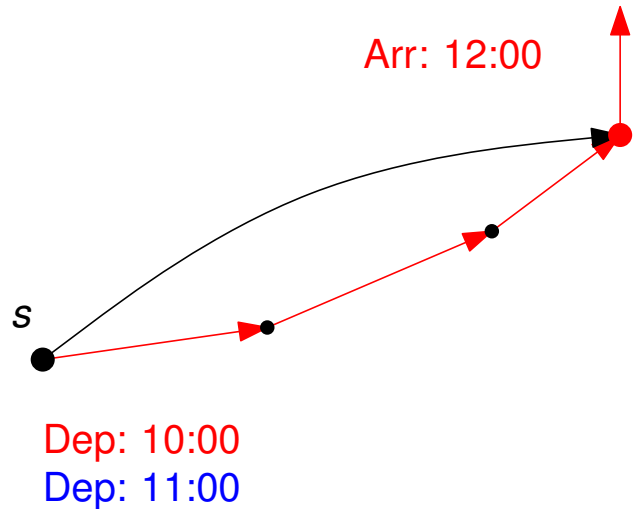
Beobachtung:
Verbindungen können sich dominieren.



Dep: 10:00
Dep: 11:00

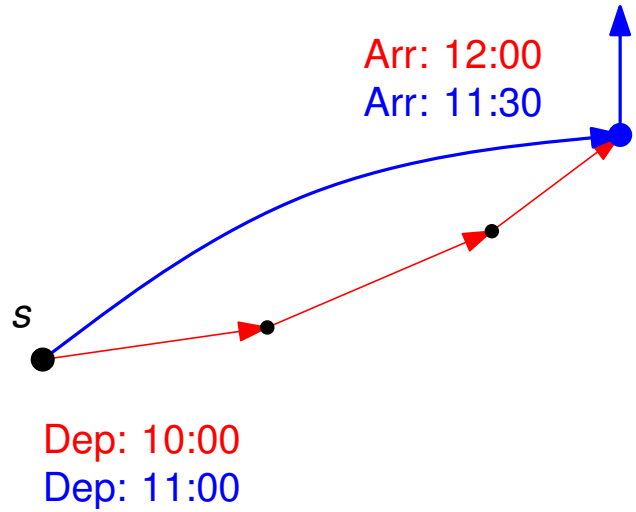
Self-Pruning

Beobachtung:
Verbindungen können sich dominieren.



Self-Pruning

Beobachtung:
Verbindungen können sich dominieren.

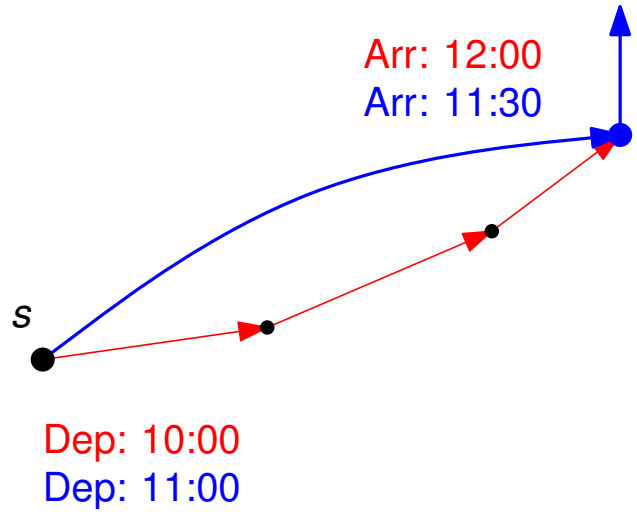


Self-Pruning

Beobachtung:
Verbindungen können sich dominieren.

Einführung: **Self-Pruning** (SP):

1. Benutze eine gemeinsame Queue
2. Keys sind Ankunftszeiten
3. Sortiere Verb. c_i an S nach Abfahrtszeit



Beim Settlen von Knoten v und Verb.-Index i:
Prüfe ob v bereits gesettled mit Verbindung $j > i$; Dann **Prune** i an v

Self-Pruning II

Integration von Self-Pruning (SP):

- Verwalte Label maxconn(v) an jedem Knoten v
  Gibt maximale Verbindung an mit der v gesettled wurde
- Update maxconn(v) beim Settlen von v

Self-Pruning II

Integration von Self-Pruning (SP):

- Verwalte Label maxconn(v) an jedem Knoten v
  Gibt maximale Verbindung an mit der v gesettled wurde
- Update maxconn(v) beim Settlen von v

Beim Settlen von Knoten v und Verb.-Index i:
Prüfe ob v bereits gesettled mit Verbindung $j > i$; Dann **Prune** i an v

Self-Pruning II

Integration von Self-Pruning (SP):

- Verwalte Label maxconn(v) an jedem Knoten v
  Gibt maximale Verbindung an mit der v gesettled wurde
- Update maxconn(v) beim Settlen von v

Beim Settlen von Knoten v und Verb.-Index i:
Prüfe ob maxconn(v) $> i$; Dann **Prune** i an v

Self-Pruning II

Integration von Self-Pruning (SP):

- Verwalte Label maxconn(v) an jedem Knoten v
  Gibt maximale Verbindung an mit der v gesettled wurde
- Update maxconn(v) beim Settlen von v

Beim Settlen von Knoten v und Verb.-Index i:
Prüfe ob maxconn(v) $>$ i; Dann **Prune** i an v

Wiederherstellung von Dijkstra's Label-Setting Eigenschaft pro Verbindung

Self-Pruning II

Integration von Self-Pruning (SP):

- Verwalte Label maxconn(v) an jedem Knoten v
  Gibt maximale Verbindung an mit der v gesettled wurde
- Update maxconn(v) beim Settlen von v

Beim Settlen von Knoten v und Verb.-Index i:
Prüfe ob maxconn(v) $>$ i; Dann **Prune** i an v

Wiederherstellung von Dijkstra's Label-Setting Eigenschaft pro Verbindung

$\Rightarrow$ Self-Pruning Connection-Setting Algorithmus (SPCS)

Connection Reduction

Problem: Falls nicht jede Verbindung optimal für $\text{dist}_S(v, \cdot)$.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
6:30 8:30	7:04 8:30	9:26 14:28	10:34 14:28	11:08 14:28	12:42 14:28	13:01 16:46	13:58 16:46	16:46 23:30	18:24 23:30	19:20 23:30	21:08 23:30

Connection Reduction

Problem: Falls nicht jede Verbindung optimal für $\text{dist}_S(v, \cdot)$.

0	1	2	3	4	5	6	7	8	9	10	11
6:30 8:30	7:04 8:30	9:26 14:28	10:34 14:28	11:08 14:28	12:42 14:28	13:01 16:46	13:58 16:46	16:46 23:30	18:24 23:30	19:20 23:30	21:08 23:30

Connection Reduction

Problem: Falls nicht jede Verbindung optimal für $\text{dist}_S(v, \cdot)$.

0	1	2	3	4	5	6	7	8	9	10	11
6:30 8:30	7:04 8:30	9:26 14:28	10:34 14:28	11:08 14:28	12:42 14:28	13:01 16:46	13:58 16:46	16:46 23:30	18:24 23:30	19:20 23:30	21:08 23:30

Connection Reduction

Problem: Falls nicht jede Verbindung optimal für $\text{dist}_S(v, \cdot)$.

0	1	2	3	4	5	6	7	8	9	10	11
6:30 8:30	7:04 8:30	9:26 14:28	10:34 14:28	11:08 14:28	12:42 14:28	13:01 16:46	13:58 16:46	16:46 23:30	18:24 23:30	19:20 23:30	21:08 23:30

Lösung:
Connection-Reduction auf den Connection Points von $\text{dist}_S(v, \cdot)$

- Wird nach dem Algoithmus durchgeführt
- Linearer Sweep von rechts nach links in $\text{dist}_S(v, \cdot)$

Parallelisierung: Idee

Gegeben:

Shared Memory Processing mit p Cores

Parallelisierung: Idee

Gegeben:
Shared Memory Processing mit p Cores

Idee:
Verteile Verbindungen c_i von S auf verschiedene Threads

0 6:30	**1** 7:04	**2** 9:26	**3** 10:34	**4** 11:08	**5** 12:42	**6** 13:01	**7** 13:58	**8** 16:46	**9** 18:24	**10** 19:20	**11** 21:08
Thread 0			Thread 1			Thread 2			Thread 3		

- Jeder Thread führt SPCS auf seiner Teilmenge der Verbindungen aus
- Ergebnisse werden im Anschluss zu $\text{dist}_S(v, \cdot)$ zusammengeführt
- Führe Connection Reduction auf gemergtem Ergebnis durch

Wahl der Partition

Frage:
Gute Strategie zur Partitionierung der Verbindungen an S

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Wahl der Partition

Frage:
Gute Strategie zur Partitionierung der Verbindungen an S

- EQUICONN:
  Gleiche Anzahl Verbindungen für jeden Thread

Wahl der Partition

Frage:
Gute Strategie zur Partitionierung der Verbindungen an S

- EQUICONN:
  Gleiche Anzahl Verbindungen für jeden Thread
- EQUITIME:
  Gleiches Zeitintervall für jeden Thread

Wahl der Partition

Frage:
Gute Strategie zur Partitionierung der Verbindungen an S

- EQUICONN:
  Gleiche Anzahl Verbindungen für jeden Thread
- EQUITIME:
  Gleiches Zeitintervall für jeden Thread
- K-MEANS
  Techniken aus dem Gebiet der Clusterung

Wahl der Partition

Frage:
Gute Strategie zur Partitionierung der Verbindungen an S

- EQUICONN:
  Gleiche Anzahl Verbindungen für jeden Thread
- EQUITIME:
  Gleiches Zeitintervall für jeden Thread
- K-MEANS
  Techniken aus dem Gebiet der Clusterung

Trade-off

Speed-up durch bessere Qualität der Partition
vs.
Berechnungsoverhead durch Partitionierung

Inter-Thread-Pruning

Problem:
Mit zunehmender Anzahl Threads, nimmt der Vorteil von Self-Pruning ab.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
6:30	7:04	9:26	10:34	11:08	12:42	13:01	13:58	16:46	18:24	19:20	21:08

Inter-Thread-Pruning

Problem:
Mit zunehmender Anzahl Threads, nimmt der Vorteil von Self-Pruning ab.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
6:30	7:04	9:26	10:34	11:08	12:42	13:01	13:58	16:46	18:24	19:20	21:08

Inter-Thread-Pruning

Problem:
Mit zunehmender Anzahl Threads, nimmt der Vorteil von Self-Pruning ab.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
6:30	7:04	9:26	10:34	11:08	12:42	13:01	13:58	16:46	18:24	19:20	21:08

Voraussetzung:
Jeder Thread berechnet nur aufeinanderfolgende Verbindungen

Inter-Thread-Pruning

Problem:
Mit zunehmender Anzahl Threads, nimmt der Vorteil von Self-Pruning ab.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
6:30	7:04	9:26	10:34	11:08	12:42	13:01	13:58	16:46	18:24	19:20	21:08

Voraussetzung:
Jeder Thread berechnet nur aufeinanderfolgende Verbindungen

Inter-Thread-Pruning Regel:

Prune Verbindung i an Knoten v und Thread k wenn:
$\exists$ Thread $l > k$ mit $\min_{j \in \text{Thread } l}\{\text{arr}(v, j)\} \leq \text{arr}(v, i)$.

Inter-Thread-Pruning II

Um Inter-Thread-Pruning zu ermöglichen:

- Pro Knoten v und Thread k: Verwalte Label $\text{minarr}_k(v)$
  Beschreibt minimale Ankunftszeit an v (von Thread k)
- Update $\text{minarr}_k(v)$ beim Settlen von v in Thread k

Inter-Thread-Pruning II

Um Inter-Thread-Pruning zu ermöglichen:

- Pro Knoten v und Thread k: Verwalte Label $\text{minarr}_k(v)$
  Beschreibt minimale Ankunftszeit an v (von Thread k)
- Update $\text{minarr}_k(v)$ beim Settlen von v in Thread k

Inter-Thread-Pruning Rule:

Prune Verbindung i an Knoten v und Thread k wenn:
$\exists$ Thread $l > k$ mit $\min_{j \in \text{Thread } l}\{\text{arr}(v, j)\} \leq \text{arr}(v, i)$.

Inter-Thread-Pruning II

Um Inter-Thread-Pruning zu ermöglichen:

- Pro Knoten v und Thread k: Verwalte Label $\text{minarr}_k(v)$
  Beschreibt minimale Ankunftszeit an v (von Thread k)
- Update $\text{minarr}_k(v)$ beim Settlen von v in Thread k

Inter-Thread-Pruning Rule:

Prune Verbindung i an Knoten v und Thread k wenn:
$\exists$ Thread $l > k$ mit $\text{minarr}_l(v) \leq \text{arr}(v, i)$.

Inter-Thread-Pruning II

Um Inter-Thread-Pruning zu ermöglichen:

- Pro Knoten v und Thread k: Verwalte Label $\text{minarr}_k(v)$
  Beschreibt minimale Ankunftszeit an v (von Thread k)
- Update $\text{minarr}_k(v)$ beim Settlen von v in Thread k

Inter-Thread-Pruning Rule:

Prune Verbindung i an Knoten v und Thread k wenn:
$\exists$ Thread $l > k$ mit $\text{minarr}_l(v) \leq \text{arr}(v, i)$.

- Verallgemeinerung von Self-Pruning auf mehrere Threads

Inter-Thread-Pruning II

Um Inter-Thread-Pruning zu ermöglichen:

- Pro Knoten v und Thread k: Verwalte Label $\text{minarr}_k(v)$
  Beschreibt minimale Ankunftszeit an v (von Thread k)
- Update $\text{minarr}_k(v)$ beim Settlen von v in Thread k

Inter-Thread-Pruning Rule:

Prune Verbindung i an Knoten v und Thread k wenn:
$\exists$ Thread $l > k$ mit $\text{minarr}_l(v) \leq \text{arr}(v, i)$.

- Verallgemeinerung von Self-Pruning auf mehrere Threads
- Prüfen einer konstanten Anzahl von $l > k$ ausreichend

Station-to-Station Anfragen

Eingabe:
Zeitabh. Netzwerk $G = (V, E)$, Start- und Zielbahnhöfe S und T.

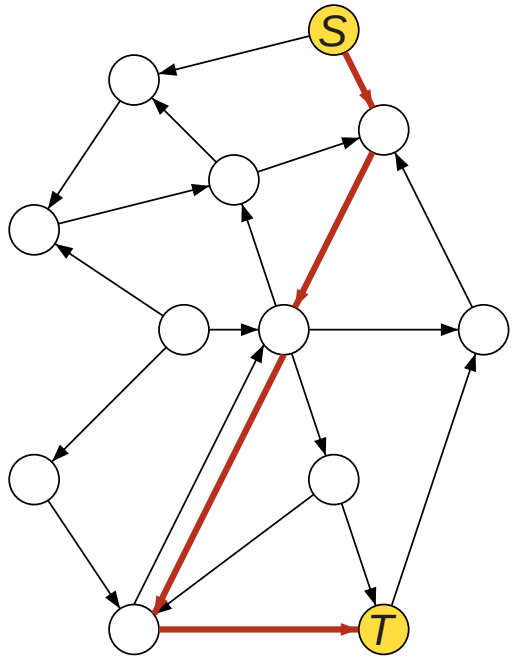
Station-to-Station Anfragen

Eingabe:
Zeitabh. Netzwerk $G = (V, E)$, Start- und Zielbahnhöfe S und T.

Ziel:
Berechne $\text{dist}_S(T, \cdot)$ nur für T

Intuition:
Weniger Berechnungsaufwand als für $\text{dist}_S(\cdot, \cdot)$

$\rightsquigarrow$ **Beschleunigungstechniken**



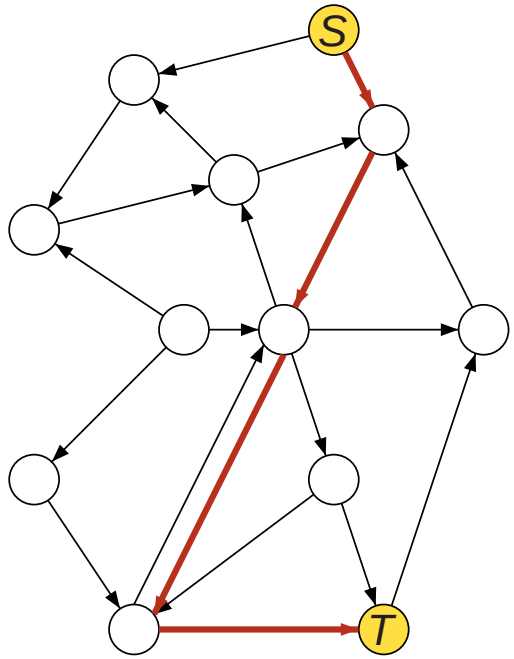
Station-to-Station Anfragen

Eingabe:
Zeitabh. Netzwerk $G = (V, E)$, Start- und Zielbahnhöfe S und T.

Ziel:
Berechne $\text{dist}_S(T, \cdot)$ nur für T

Intuition:
Weniger Berechnungsaufwand als für $\text{dist}_S(\cdot, \cdot)$

$\rightsquigarrow$ **Beschleunigungstechniken**



Nicht-trivial für Dijkstra's Algorithmus (Diese Vorlesung) :-)

Stoppkriterium

Dijkstra's Algorithmus:
Breche die Suche ab, sobald T abgearbeitet wurde.

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Stoppkriterium

Dijkstra's Algorithmus:
Breche die Suche ab, sobald T abgearbeitet wurde.

kann adaptiert werden durch

Parallel Self-Pruning Connection-Setting:

- Verwalte globales Label $T_m := -\infty$

Stoppkriterium

Dijkstra's Algorithmus:
Breche die Suche ab, sobald T abgearbeitet wurde.

kann adaptiert werden durch

Parallel Self-Pruning Connection-Setting:

- Verwalte globales Label $T_m := -\infty$
- Wenn Verbindung i an T abgearbeitet wird, setze $T_m := \max\{T_m, i\}$

Stoppkriterium

Dijkstra's Algorithmus:
Breche die Suche ab, sobald T abgearbeitet wurde.

kann adaptiert werden durch

Parallel Self-Pruning Connection-Setting:

- Verwalte globales Label $T_m := -\infty$
- Wenn Verbindung i an T abgearbeitet wird, setze $T_m := \max\{T_m, i\}$
- Prune alle Verbindungen $j < T_m$ (an jedem Knoten)

Stoppkriterium

Dijkstra's Algorithmus:
Breche die Suche ab, sobald T abgearbeitet wurde.

kann adaptiert werden durch

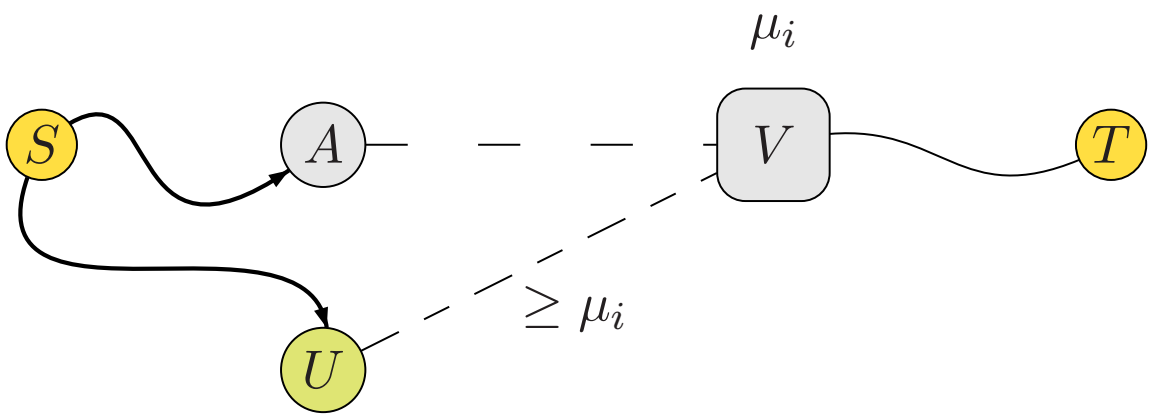
Parallel Self-Pruning Connection-Setting:

- Verwalte globales Label $T_m := -\infty$
- Wenn Verbindung i an T abgearbeitet wird, setze $T_m := \max\{T_m, i\}$
- Prune alle Verbindungen $j < T_m$ (an jedem Knoten)
- Halte an, wenn Priority-Queue leer läuft

Pruning durch Distanztabellen (Idee)

Vorberechnung:

- Wähle Teilmenge $\mathcal{S}_{\text{trans}}$ von Transfer-Stationen
- Berechne komplette Distanztabelle zwischen allen $S \in \mathcal{S}_{\text{trans}}$



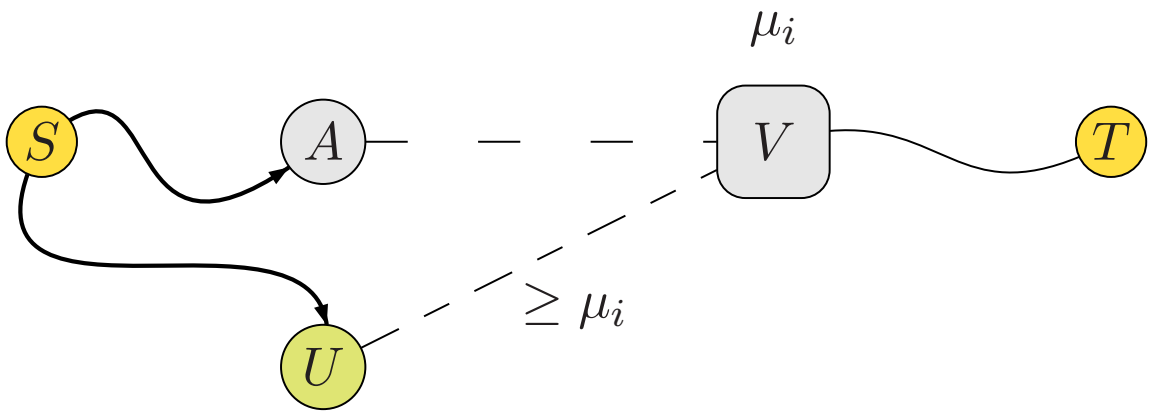
Idee:

- Verwalte vorl. Distanz μ_i zu Transferstationen V "nahe" Ziel T
  μ_i ist **obere Schranke** bzgl. echtem Abstand zu V

Pruning durch Distanztabellen (Idee)

Vorberechnung:

- Wähle Teilmenge $\mathcal{S}_{\text{trans}}$ von Transfer-Stationen
- Berechne komplette Distanztabelle zwischen allen $S \in \mathcal{S}_{\text{trans}}$



Idee:

- Verwalte vorl. Distanz μ_i zu Transferstationen V "nahe" Ziel T
  μ_i ist **obere Schranke** bzgl. echtem Abstand zu V
- Prune Verbindung i an beliebiger Transferstation U wenn

$$\text{dist}_S(U) + \mathcal{D}[U, V] \geq \mu_i$$

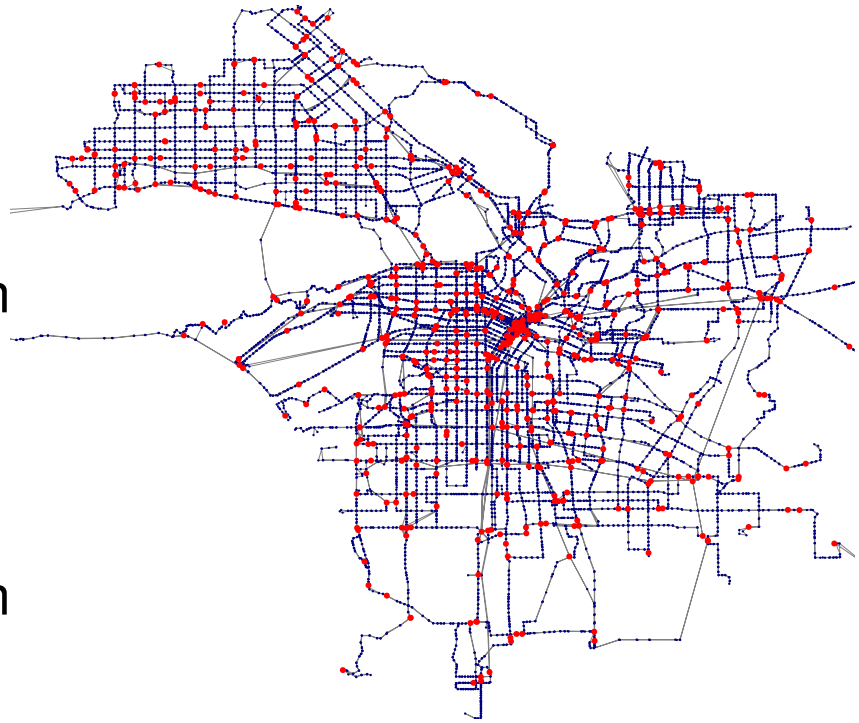
Eingabe

Netzwerk von **Los Angeles**:

- 15 581 Stationen,
- 1 046 580 elem. Verbindungen

Zugnetz von **Europa**:

- 30 517 Stationen,
- 1 775 533 elem. Verbindungen



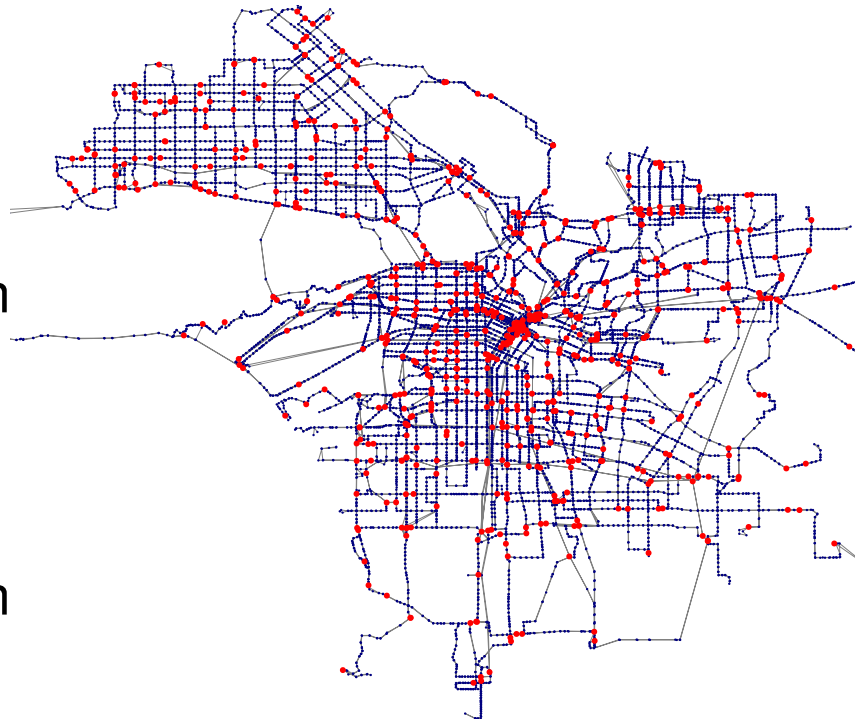
Eingabe

Netzwerk von **Los Angeles**:

- 15 581 Stationen,
- 1 046 580 elem. Verbindungen

Zugnetz von **Europa**:

- 30 517 Stationen,
- 1 775 533 elem. Verbindungen



Auswertung durch 1 000 Anfragen wobei Start- und Zielbahnhöfe gleichverteilt zufällig gewählt.

One-to-All Anfragen

	p	Los Angeles Settled Conns	Los Angeles Time [ms]	Los Angeles Spd Up	Los Angeles Std-Dev	Europe Settled Conns	Europe Time [ms]	Europe Spd Up	Europe Std-Dev
PSPCS:	1	2.5 M	1209.0	1.0	—	3.3 M	2152.0	1.0	—
EQUICONN	2	2.5 M	690.0	1.8	14.7 %	3.1 M	1054.2	2.0	16.1 %
	4	2.5 M	417.4	2.9	18.2 %	3.4 M	673.8	3.2	24.4 %
	8	2.5 M	267.7	4.5	20.0 %	4.2 M	510.9	4.2	23.8 %
LC:	1	18.9 M	1482.1	—	—	17.4 M	2497.1	—	—

One-to-All Anfragen

	p	Los Angeles Settled Conns	Los Angeles Time [ms]	Los Angeles Spd Up	Los Angeles Std-Dev	Europe Settled Conns	Europe Time [ms]	Europe Spd Up	Europe Std-Dev
PSPCS:	1	2.5 M	1209.0	1.0	—	3.3 M	2152.0	1.0	—
EQUICONN	2	2.5 M	690.0	1.8	14.7 %	3.1 M	1054.2	2.0	16.1 %
	4	2.5 M	417.4	2.9	18.2 %	3.4 M	673.8	3.2	24.4 %
	8	2.5 M	267.7	4.5	20.0 %	4.2 M	510.9	4.2	23.8 %
LC:	1	18.9 M	1482.1	—	—	17.4 M	2497.1	—	—

- PSPCS deutlich weniger Verbindungen als LC

One-to-All Anfragen

	p	Los Angeles Settled Conns	Los Angeles Time [ms]	Los Angeles Spd Up	Los Angeles Std-Dev	Europe Settled Conns	Europe Time [ms]	Europe Spd Up	Europe Std-Dev
PSPCS:	1	2.5 M	1209.0	1.0	—	3.3 M	2152.0	1.0	—
EQUICONN	2	2.5 M	690.0	1.8	14.7 %	3.1 M	1054.2	2.0	16.1 %
	4	2.5 M	417.4	2.9	18.2 %	3.4 M	673.8	3.2	24.4 %
	8	2.5 M	267.7	4.5	20.0 %	4.2 M	510.9	4.2	23.8 %
LC:	1	18.9 M	1482.1	—	—	17.4 M	2497.1	—	—

- PSPCS deutlich weniger Verbindungen als LC
- PSPCS skaliert sehr gut mit zunehmender Anzahl Cores

One-to-All Anfragen

		Los Angeles				Europe			
	p	Settled Conns	Time [ms]	Spd Up	Std-Dev	Settled Conns	Time [ms]	Spd Up	Std-Dev
PSPCS:	1	2.5 M	1209.0	1.0	—	3.3 M	2152.0	1.0	—
EQUICONN	2	2.5 M	690.0	1.8	14.7 %	3.1 M	1054.2	2.0	16.1 %
	4	2.5 M	417.4	2.9	18.2 %	3.4 M	673.8	3.2	24.4 %
	8	2.5 M	267.7	4.5	20.0 %	4.2 M	510.9	4.2	23.8 %
LC:	1	18.9 M	1482.1	—	—	17.4 M	2497.1	—	—

- PSPCS deutlich weniger Verbindungen als LC
- PSPCS skaliert sehr gut mit zunehmender Anzahl Cores
- Einfache Partitionierung liefert bereits gute Ergebnisse

Station-to-Station Anfragen

	Los Angeles				Europe			
	PREPRO		QUERY		PREPRO		QUERY	
	Time [m:s]	Space [MiB]	Time [ms]	Spd Up	Time [m:s]	Space [MiB]	Time [ms]	Spd Up
0.0%	—	—	188.2	1.0	—	—	412.4	1.0
5.0%	4:19	240.7	59.1	3.2	20:13	214.3	186.5	2.2
10.0%	8:07	832.2	59.0	3.2	39:05	794.4	151.2	2.7
20.0%	16:21	3006.0	57.7	3.3	75:35	2986.7	132.1	2.9
deg $>$ 2	18:01	3263	51.2	3.7	—	—	—	—

Station-to-Station Anfragen

	Los Angeles				Europe			
	PREPRO		QUERY		PREPRO		QUERY	
	Time [m:s]	Space [MiB]	Time [ms]	Spd Up	Time [m:s]	Space [MiB]	Time [ms]	Spd Up
0.0%	—	—	188.2	1.0	—	—	412.4	1.0
5.0%	4:19	240.7	59.1	3.2	20:13	214.3	186.5	2.2
10.0%	8:07	832.2	59.0	3.2	39:05	794.4	151.2	2.7
20.0%	16:21	3006.0	57.7	3.3	75:35	2986.7	132.1	2.9
deg $>$ 2	18:01	3263	51.2	3.7	—	—	—	—

- Speed-Up durch Distanztabelle bis 3.7

Station-to-Station Anfragen

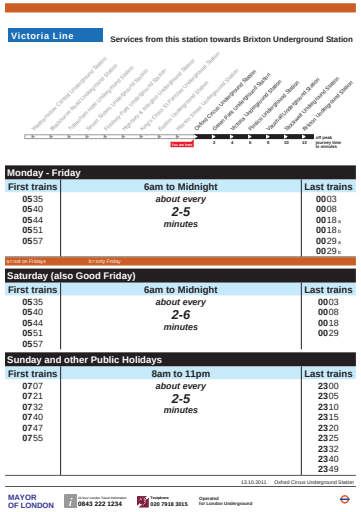
	Los Angeles				Europe			
	PREPRO		QUERY		PREPRO		QUERY	
	Time [m:s]	Space [MiB]	Time [ms]	Spd Up	Time [m:s]	Space [MiB]	Time [ms]	Spd Up
0.0%	—	—	188.2	1.0	—	—	412.4	1.0
5.0%	4:19	240.7	59.1	3.2	20:13	214.3	186.5	2.2
10.0%	8:07	832.2	59.0	3.2	39:05	794.4	151.2	2.7
20.0%	16:21	3006.0	57.7	3.3	75:35	2986.7	132.1	2.9
deg $>$ 2	18:01	3263	51.2	3.7	—	—	—	—

- Speed-Up durch Distanztabelle bis 3.7
- 10 % Transferstationen sind guter Kompromiss

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

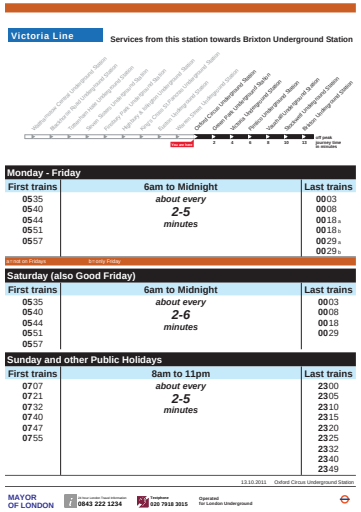
13.10.2011 Oxford Circus Underground Station

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

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

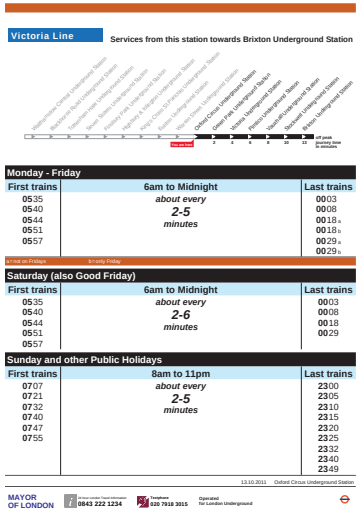
Earliest Arrival Problem:
Gegeben Stops s, t und Abfahrtszeit τ, berechne

- 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

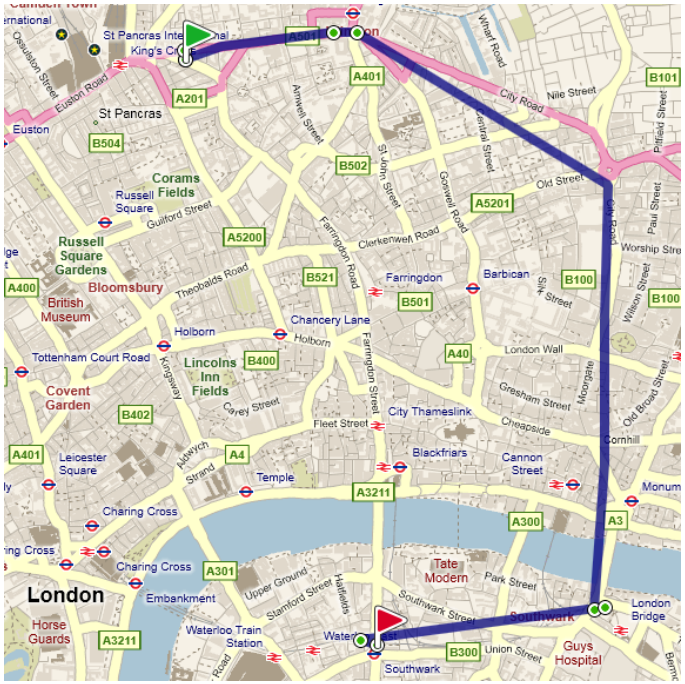
Earliest Arrival Problem:
Gegeben Stops s, t und Abfahrtszeit τ, berechne

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

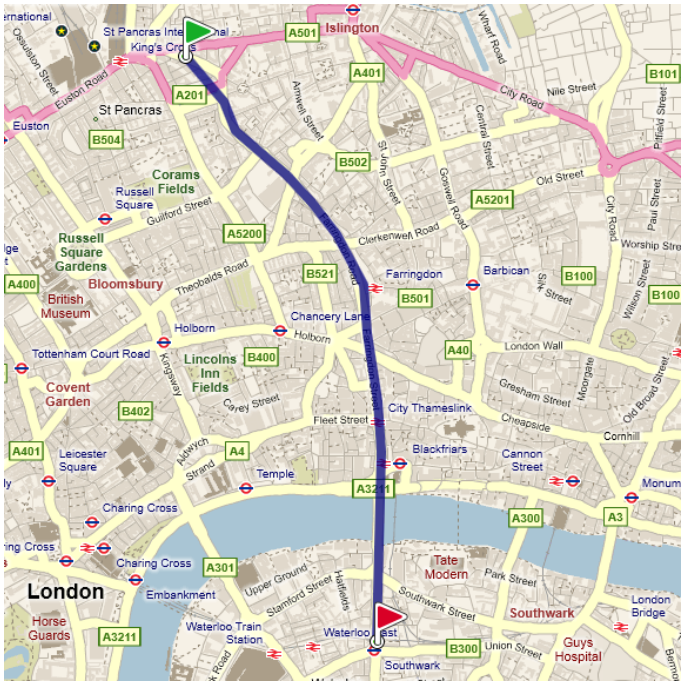
Reicht uns das?

Einbeziehen von Umstiegen

Umstiege zu betrachten ist wichtig!



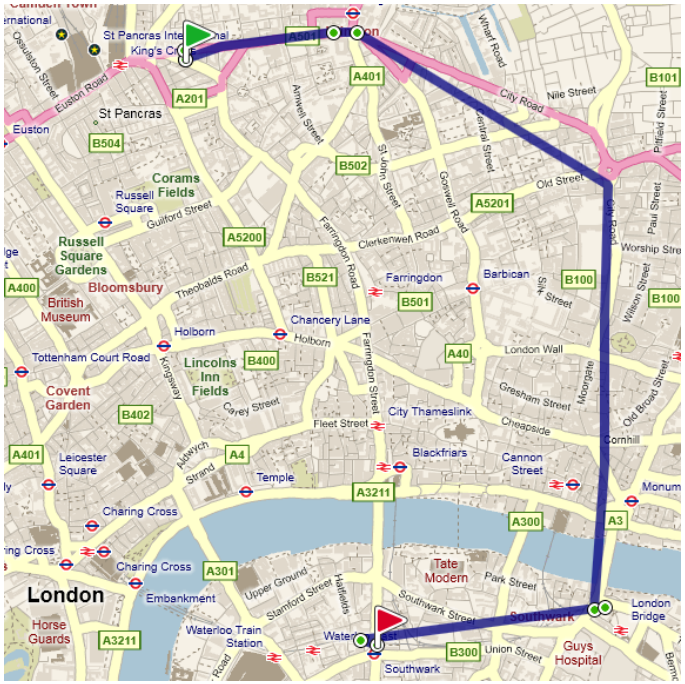
Ankunft 11:08 Uhr, 2 Umstiege



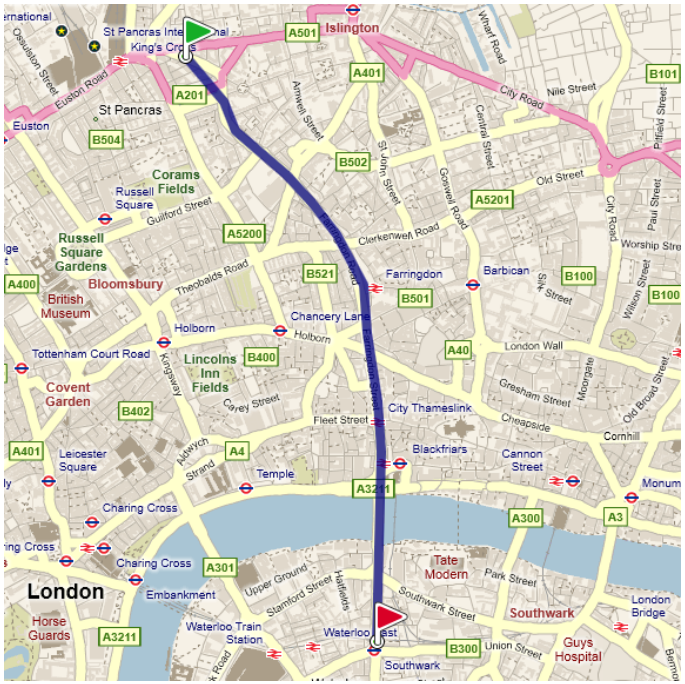
Ankunft 11:09 Uhr, 0 Umstiege

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.

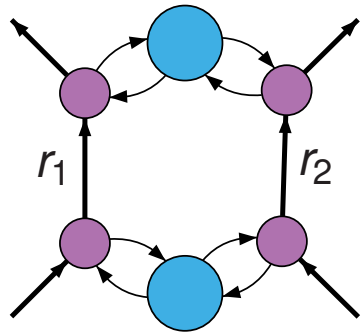
$M = \{(15{:}13 \text{ Uhr}, 3),\ (13{:}45 \text{ Uhr}, 4),\ (15{:}15 \text{ Uhr}, 0)\}$.

Wie effizient berechnen?

Multi-Label-Correcting Ansatz

Idee

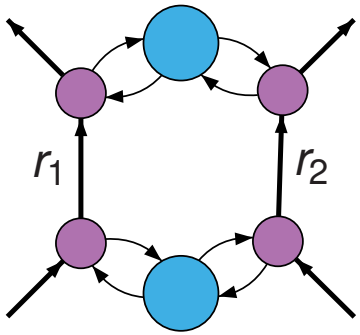
- Benutze (zeitabhängiges) Graphmodell
- Grundlage: Dijkstra's Algorithmus



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

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?

Literatur I


Reinhard Bauer, Daniel Delling, and Dorothea Wagner.
Experimental Study on Speed-Up Techniques for Timetable Information Systems.
Networks, 57(1):38–52, January 2011.


Daniel Delling.
Time-Dependent SHARC-Routing.
Algorithmica, 60(1):60–94, May 2011.
Special Issue: European Symposium on Algorithms 2008.


Daniel Delling, Bastian Katz, and Thomas Pajor.
Parallel Computation of Best Connections in Public Transportation Networks.
ACM Journal of Experimental Algorithmics, 2012.
To appear.


Yann Disser, Matthias Müller–Hannemann, and Mathias Schnee.
Multi-Criteria Shortest Paths in Time-Dependent Train Networks.
In Catherine C. McGeoch, editor, *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08)*, volume 5038 of *Lecture Notes in Computer Science*, pages 347–361. Springer, June 2008.

Literatur II

Daniel Delling, Thomas Pajor, and Dorothea Wagner.
Engineering Time-Expanded Graphs for Faster Timetable Information.
In Ravindra K. Ahuja, Rolf H. Möhring, and Christos Zaroliagis, editors, *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 182–206. Springer, 2009.

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.

Robert Geisberger.
Contraction of Timetable Networks with Realistic Transfers.
In Paola Festa, editor, *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA'10)*, volume 6049 of *Lecture Notes in Computer Science*, pages 71–82. Springer, May 2010.

Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis.
Efficient Models for Timetable Information in Public Transportation Systems.
ACM Journal of Experimental Algorithmics, 12(2.4):1–39, 2008.

Nächste Termine

Karlsruher Institut für Technologie

Montag, 9.7.2012

Montag, 16.7.2012
Ende!