Karlsruher Institut für Technologie

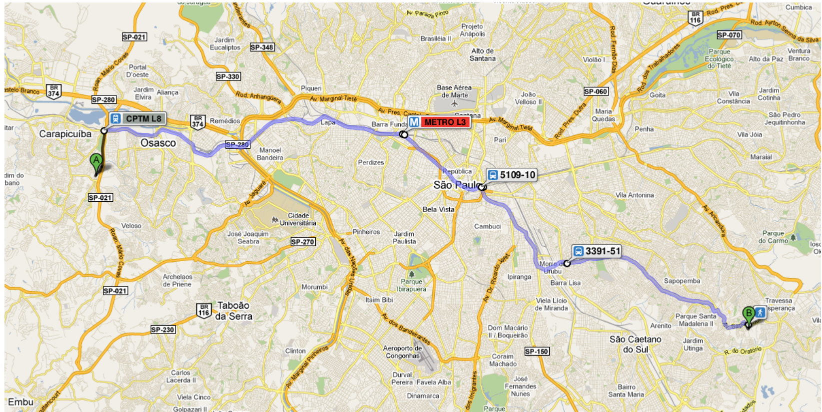
Algorithmen für Routenplanung

15. Sitzung, Sommersemester 2011

Thomas Pajor | 20. Juni 2011

INSTITUT FÜR THEORETISCHE INFORMATIK · ALGORITHMIK I · PROF. DR. DOROTHEA WAGNER

Heute: Routenplanung in Bahnnetzen (Fahrplanauskunft)



Eingabe

Karlsruher Institut für Technologie

Gegeben (Fahrplan):

- Menge $\mathcal{B}$ von Bahnhöfen,
- Menge $\mathcal{Z}$ von Zügen
- Menge $\mathcal{C}$ von elementaren Verbindungen
- Mindestumstiegszeiten transfer : $\mathcal{B} \to \mathbb{N}$.

Elementare Verbindung: Tupel bestehend aus

- Zug $Z \in \mathcal{Z}$
- Abfahrtsbahnhof $S_{\text{dep}} \in \mathcal{B}$
- Zielbahnhof $S_{\text{arr}} \in \mathcal{B}$
- Abfahrtszeit $\tau_{\text{dep}} \in \Pi$
- Ankunftszeit $\tau_{\text{arr}} \in \Pi$

Interpretation: Zug Z fährt von S_{dep} nach S_{arr} ohne Zwischenhalt von τ_{dep}–τ_{arr} Uhr.

Eingabe

KIT Karlsruher Institut für Technologie

Beispiel für einen Fahrplan:

```
(IR 2269, Karlsruhe Hbf,   Pforzheim Hbf,   10:05, 10:23)
(IR 2269, Pforzheim Hbf,   Mühlacker,       10:25, 10:33)
(IR 2269, Mühlacker,       Vaihingen(Enz), 10:34, 10:40)
(IR 2269, Vaihingen(Enz), Stuttgart Hbf,   10:41, 10:57)
...
(ICE 791, Stuttgart Hbf,   Ulm Hbf,         11:12, 12:06)
(ICE 791, Ulm Hbf,         Augsburg Hbf,    12:08, 12:47)
(ICE 791, Augsburg Hbf,    München Hbf,     12:49, 13:21)
```

Ziel:

- Modellierung des Fahrplans als Graphen
- Fahrplanauskunft durch kürzeste-Wege Suche

Stationsgraph

Karlsruher Institut für Technologie

Modellierung:

- Für jeden Bahnhof $S \in \mathcal{B}$: Ein Knoten
- Kante (S_i, S_j) gdw. ex. elementare Verbindung von S_i nach S_j
- Kantengewichte: Lower-Bound der Reisezeit zw. Bahnhöfen

Nachteile:

- Unrealistische Anfragen (keine Zeitabhängigkeit)
- Mit Zeitabhängigkeit: Unrealistische Umstiege

Modellierung: Zwei Ansätze

Karlsruher Institut für Technologie

1. Zeitexpandiert

- Zeitabhängigkeiten ausrollen
- Knoten entsprechen Ereignissen im Fahrplan
- Kanten verbinden Ereignisse miteinander
  - Zugfahrt eines Zuges,
  - Umstieg zwischen Zügen,
  - Warten

\- Großer Graph (viele Knoten und Kanten)

\+ Einfacher Anfragealgorithmus (Dijkstra)

Modellierung: Zwei Ansätze

Karlsruher Institut für Technologie

1. Zeitexpandiert

- Zeitabhängigkeiten ausrollen
- Knoten entsprechen Ereignissen im Fahrplan
- Kanten verbinden Ereignisse miteinander
  - Zugfahrt eines Zuges,
  - Umstieg zwischen Zügen,
  - Warten

\- Großer Graph (viele Knoten und Kanten)

\+ Einfacher Anfragealgorithmus (Dijkstra)

2. Zeitabhängig

- Zeitabhängigkeit an den Kanten
- Knoten entsprechen Bahnhöfen
- Kante $\Leftrightarrow$ Zug verbindet Bahnhöfe
  - Transferzeiten?

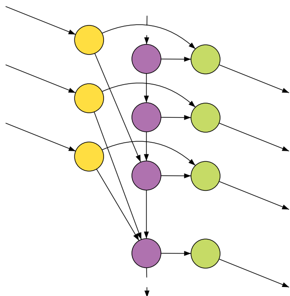
\+ Kleiner Graph

\- Zeitabhängige Routenplanung

Modellierung: Zwei Ansätze

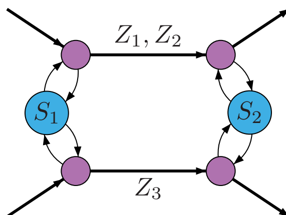
Karlsruher Institut für Technologie

1. Zeitexpandiert



- Arrival-, Transfer- und Departure-Ereignisse
- Für jeden Zug
- Kantengewicht = Zeitdifferenz

2. Zeitabhängig



- Partitioniere Züge in Routen
- Pro Bahnhof: Stationsknoten
- Pro Route: Route-Knoten
- Routenkanten: Mehrere Züge
- Stationskanten: Transferzeit

Zeit-Anfragen

Karlsruher Institut für Technologie

Gegeben: Startbahnhof S, Zielbahnhof T und Abfahrtszeit τ_S

Zeit-Anfragen

Karlsruher Institut für Technologie

Gegeben: Startbahnhof S, Zielbahnhof T und Abfahrtszeit τ_S

1. Zeitexpandiert

Startknoten:

- *Erstes* Transferevent in S mit Zeit $\tau \geq \tau_S$.

Zielknoten:

- Im Vorraus unbekannt!
- Stoppkriterium: Erster gesettleter Knoten an T induziert schnellste Verbindung zu T

2. Zeitabhängig

Startknoten:

- Bahnhofsknoten S

Zielknoten:

- Bahnhofsknoten T

Anfrage:

- Time-Dependent Dijkstra mit Zeit τ_S
- Hier: Ankunfts*zeit* im Vorraus unbekannt

Profil-Anfragen

Karlsruher Institut für Technologie

Gegeben: Startbahnhof S, Zielbahnhof T

Profil-Anfragen

Gegeben: Startbahnhof S, Zielbahnhof T

1. Zeitexpandiert

?

2. Zeitabhängig

Startknoten:

- Bahnhofsknoten S

Zielknoten:

- Bahnhofsknoten T

Anfrage:

- Label-Correcting Algorithmus von S

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

Jetzt: Erster Ansatz zur Beschleunigung von Profil-Anfragen

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

Parallele Profilsuchen

Karlsruher Institut für Technologie

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

Analyse: Schienenfunktionen

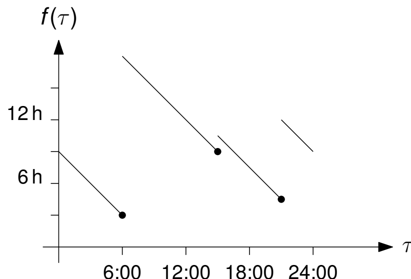
KIT
Karlsruher Institut für Technologie

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)
  $\tau \hat{=}$ Abfahrtszeit, $w \hat{=}$ Reisezeit
- Zwischen Verbindungen: Lineares Warten

Profil-Anfragen (One-to-All)

Karlsruher Institut für Technologie

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

Profil-Anfragen (One-to-All)

Karlsruher Institut für Technologie

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

Problem (Profil-Anfrage):
Berechne die *Reisezeitfunktion* $\mathrm{dist}_S(v, \tau)$, so dass $\mathrm{dist}_S(v, \tau)$ 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)

Karlsruher Institut für Technologie

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

Problem (Profil-Anfrage):
Berechne die *Reisezeitfunktion* $\mathrm{dist}_S(v, \tau)$, so dass $\mathrm{dist}_S(v, \tau)$ 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 ($\approx$ Factor 50)

Hauptidee

Karlsruher Institut für Technologie

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

Hauptidee

Karlsruher Institut für Technologie

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

Naiver Ansatz

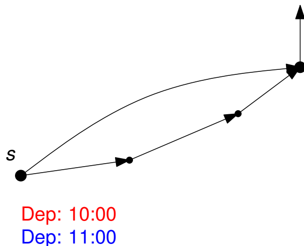
Für jede ausgehende Verbindung c_i an S:
Separate Zeitanfrage mit Abfahrtszeit $\tau_{\text{dep}}(c_i)$.

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

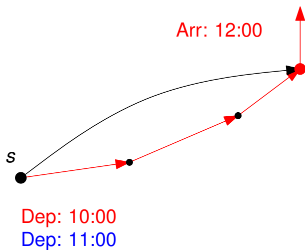
Self-Pruning

Beobachtung:
Verbindungen können sich dominieren.



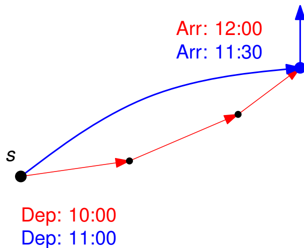
Self-Pruning

Beobachtung:
Verbindungen können sich dominieren.



Self-Pruning

Beobachtung:
Verbindungen können sich dominieren.

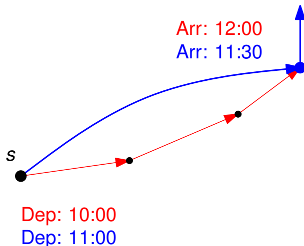


Self-Pruning

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

Karlsruher Institut für Technologie

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

Karlsruher Institut für Technologie

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 $\text{maxconn}(v) > i$; Dann **Prune** i an v

Self-Pruning II

Karlsruher Institut für Technologie

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 $\text{maxconn}(v) > i$; Dann **Prune** i an v

Wiederherstellung von Dijkstra's Label-Setting Eigenschaft pro Verbindung

Self-Pruning II

Karlsruher Institut für Technologie

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

Karlsruher Institut für Technologie

Problem:
Dennoch: Nicht jede Verbindung ist optimal für $\mathrm{dist}_S(v, \cdot)$.

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
8:30	8:30	14:28	14:28	14:28	14:28	16:46	16:46	23:30	23:30	23:30	23:30

Connection Reduction

Karlsruher Institut für Technologie

Problem:
Dennoch: Nicht jede Verbindung ist optimal für $\mathrm{dist}_S(v, \cdot)$.

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

Karlsruher Institut für Technologie

Problem:
Dennoch: Nicht jede Verbindung ist optimal für $\mathrm{dist}_S(v, \cdot)$.

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
8:30	8:30	14:28	14:28	14:28	14:28	16:46	16:46	23:30	23:30	23:30	23:30

Connection Reduction

Karlsruher Institut für Technologie

Problem:
Dennoch: Nicht jede Verbindung ist optimal für $\mathrm{dist}_S(v,\cdot)$.

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 $\mathrm{dist}_S(v,\cdot)$

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

Parallelisierung: Idee

Karlsruher Institut für Technologie

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	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
Thread 0			Thread 1			Thread 2			Thread 3		

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

Choice of Partitioning

KIT Karlsruher Institut für Technologie

Frage:
Gute Strategie zur Partitionierung der Verbindungen an S

Choice of Partitioning

Karlsruher Institut für Technologie

Frage:
Gute Strategie zur Partitionierung der Verbindungen an S

- EQUICONN:
  Gleiche Anzahl Verbindungen für jeden Thread

Choice of Partitioning

Karlsruher Institut für Technologie

Frage:
Gute Strategie zur Partitionierung der Verbindungen an S

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

Choice of Partitioning

Karlsruher Institut für Technologie

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

Choice of Partitioning

Karlsruher Institut für Technologie

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

Karlsruher Institut für Technologie

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

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

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

KIT Karlsruher Institut für Technologie

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 $\min_{j \in \text{Thread } l}\{\text{arr}(v, j)\} < \text{arr}(v, i)$.

Inter-Thread-Pruning II

Um Inter-Thread-Pruning zu ermöglichen:

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

Inter-Thread-Pruning II

KIT – Karlsruher Institut für Technologie

Um Inter-Thread-Pruning zu ermöglichen:

- Pro Knoten v und Thread k: Verwalte Label $\text{minarr}_k(v)$
  Beschreibt Verbindungsindex mit minimaler Ankunftszeit an v
- 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 $\min_{j \in \text{Thread } l}\{\text{arr}(v, j)\} < \text{arr}(v, i)$.

Inter-Thread-Pruning II

Karlsruher Institut für Technologie

Um Inter-Thread-Pruning zu ermöglichen:

- Pro Knoten v und Thread k: Verwalte Label $\text{minarr}_k(v)$
  Beschreibt Verbindungsindex mit minimaler Ankunftszeit an v
- 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 $\text{minarr}_l(v) < \text{arr}(v, i)$.

Inter-Thread-Pruning II

Karlsruher Institut für Technologie

Um Inter-Thread-Pruning zu ermöglichen:

- Pro Knoten v und Thread k: Verwalte Label $\mathrm{minarr}_k(v)$
  Beschreibt Verbindungsindex mit minimaler Ankunftszeit an v
- Update $\mathrm{minarr}_k(v)$ 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 $\mathrm{minarr}_l(v) < \mathrm{arr}(v, i)$.

- Verallgemeinerung von Self-Pruning auf mehrere Threads

Inter-Thread-Pruning II

Karlsruher Institut für Technologie

Um Inter-Thread-Pruning zu ermöglichen:

- Pro Knoten v und Thread k: Verwalte Label $\text{minarr}_k(v)$
  Beschreibt Verbindungsindex mit minimaler Ankunftszeit an v
- 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 $\text{minarr}_l(v) < \text{arr}(v, i)$.

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

Station-to-Station Anfragen

Karlsruher Institut für Technologie

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

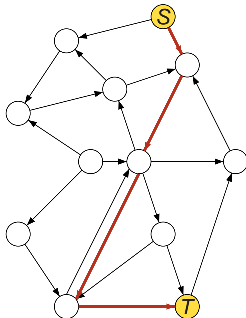
Station-to-Station Anfragen

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

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

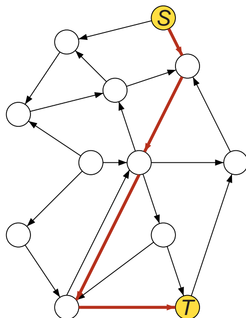
Intuition:
Weniger Berechnungsaufwand als für $\mathrm{dist}_S(\cdot, \cdot)$

$\rightsquigarrow$ **Beschleunigungstechniken**



Station-to-Station Anfragen

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



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

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

$\rightsquigarrow$ **Beschleunigungstechniken**

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

Stoppkriterium

Karlsruher Institut für Technologie

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

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

Karlsruher Institut für Technologie

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

Karlsruher Institut für Technologie

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

kann adaptiert werden durch

Parallel Self-Pruning Connection-Setting:

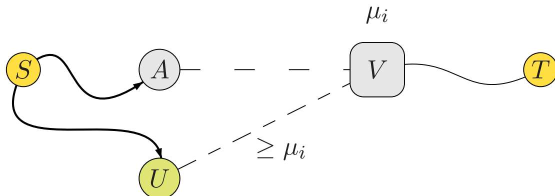
- Verwalte globales Label $T_m := -\infty$
- Wenn Verbindung i an T abgearbeitet wird, setze $T_m := \max\{T_m, i\}$
- Prune alle Verbindungen $j < T_m$ (an jedem Knoten)
- Halte an, wenn Priority-Queue leer läuft

Pruning durch Distanztabellen (Idee)

KIT – Karlsruher Institut für Technologie

Vorberechnung:

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



Idee:

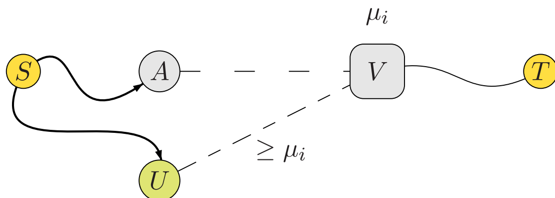
- Verwalte vorl. Distanz μ_i zu Transferstationen V “nahe“ Ziel T
  μ_i ist **obere Schranke** bzgl. echtem Abstand zu V

Pruning durch Distanztabellen (Idee)

Karlsruher Institut für Technologie

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

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

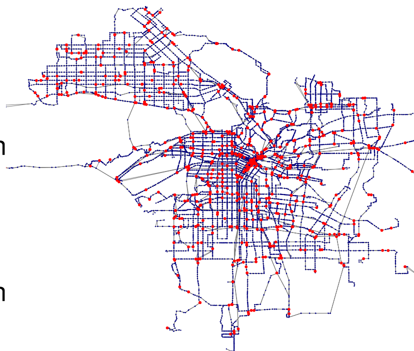
Eingabe

Netzwerk von **Los Angeles**:

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

Zugnetz von **Europa**:

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



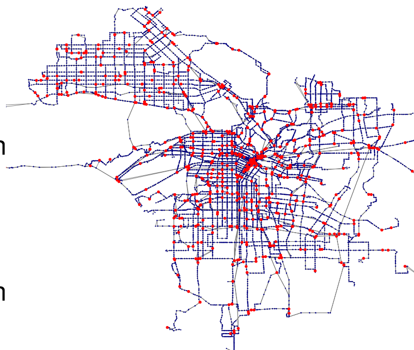
Eingabe

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

Karlsruher Institut für Technologie

	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	Time [ms]	Spd Up	Std-Dev	Europe 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

One-to-All Anfragen

KIT
Karlsruher Institut für Technologie

	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

Karlsruher Institut für Technologie

	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
- 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

Karlsruher Institut für Technologie

	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

Karlsruher Institut für Technologie

	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

Zusammenfassung

Karlsruher Institut für Technologie

Fahrplanauskunft

- Zeitexpandierte vs. zeitabhängige Modellierung
- Beschleunigungstechniken auf Straßennetzwerken schlecht adaptierbar
- Profilsuchen sinnvoller als Zeitanfragen
- PSPCS: Ersetzt Label-Correcting Algorithmus für Profil-Suchen
  - Connection-Setting Eigenschaft
  - Self-Pruning von dominierten Verbindungen
  - Gut Parallelisierbar

Offenes Forschungsgebiet!

Ende

Karlsruher Institut für Technologie

Literatur (Fahrplanauskunft):

- **Evangelia Pyrga, Frank Schulz, Dorothea Wagner, Christos Zaroliagis**
  Efficient Models for Timetable Information in Public Transportation Systems In: *ACM Journal of Experimental Algorithmics*, 2007
- **Daniel Delling, Bastian Katz, Thomas Pajor**
  Parallel Computation of Best Connections in Public Transportation Networks In: *24th International Parallel and Distributed Processing Symposium (IPDPS'10). IEEE Computer Society*, 2010

Nächte Vorlesung: Montag, 4. Juli, 2011