Karlsruher Institut für Technologie

Algorithmen für Routenplanung

16. Sitzung, Sommersemester 2011

Thomas Pajor | 4. Juli 2011

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

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

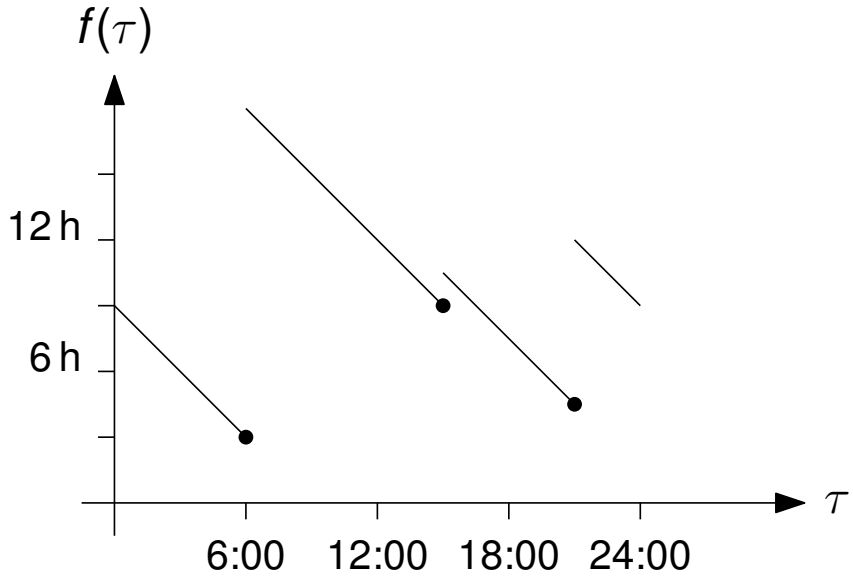
Analyse: Schienenfunktionen

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.

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 ($\approx$ Factor 50)

Profil-Anfragen (One-to-All)

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)

Profil-Anfragen (One-to-All)

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

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

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

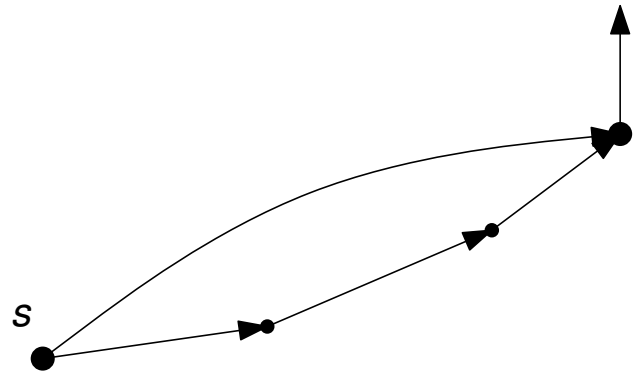
- Zu viele redundante Berechnungen
  Periodische Natur der Fahrpläne
- Nicht jede Verbindung ab S trägt zu $\text{dist}_S(v, \cdot)$ bei
  Langsame Züge für weite Reisen machen wenig Sinn

Self-Pruning

Beobachtung:
Verbindungen können sich dominieren.

Einführung: **Self-Pruning** (SP):

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



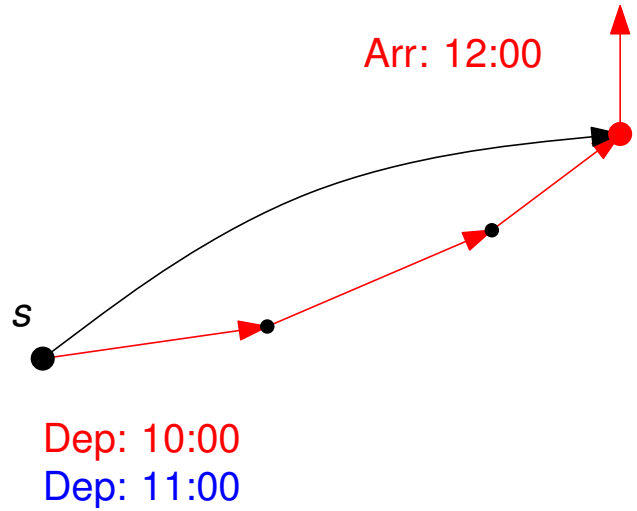
Beim Settlen von Knoten v und Verb.-Index i:
Prüfe ob v bereits gesettled mit Verbindung $j > i$; Dann **Prune** i an v

Self-Pruning

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



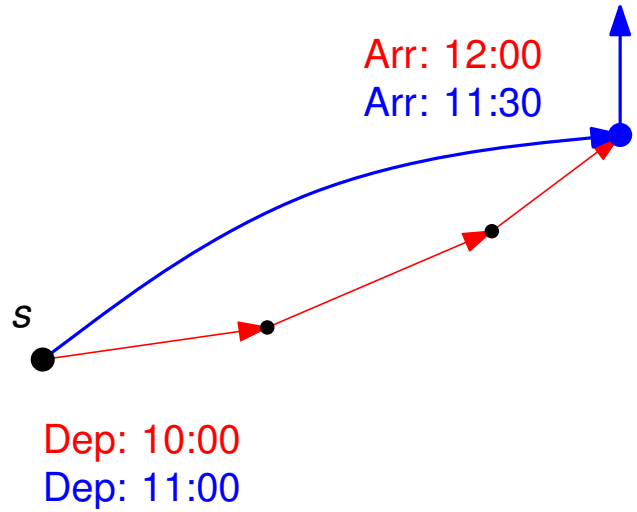
Beim Settlen von Knoten v und Verb.-Index i:
Prüfe ob v bereits gesettled mit Verbindung $j > i$; Dann **Prune** i an v

Self-Pruning

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



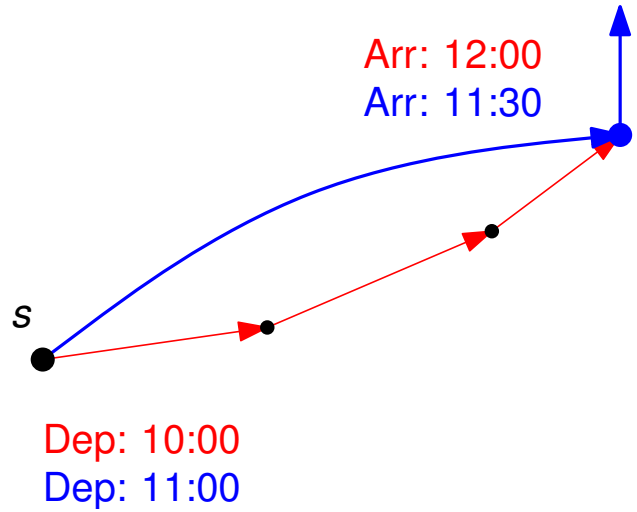
Beim Settlen von Knoten v und Verb.-Index i:
Prüfe ob v bereits gesettled mit Verbindung $j > i$; Dann **Prune** i an v

Self-Pruning

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

Wiederherstellung von Dijkstra's Label-Setting Eigenschaft pro Verbindung

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

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

Wiederherstellung von Dijkstra's Label-Setting Eigenschaft pro Verbindung

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

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)

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)

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

Connection Reduction

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

Connection Reduction

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

Connection Reduction

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

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

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 $\mathrm{dist}_S(v, \cdot)$ zusammengeführt
- Führe Connection Reduction auf gemergtem Ergebnis durch

Choice of Partitioning

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

Choice of Partitioning

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

Choice of Partitioning

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

Choice of Partitioning

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

Choice of Partitioning

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

Inter-Thread-Pruning II

Um Inter-Thread-Pruning zu ermöglichen:

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

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

Inter-Thread-Pruning II

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

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

Inter-Thread-Pruning II

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

Inter-Thread-Pruning II

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

Inter-Thread-Pruning II

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

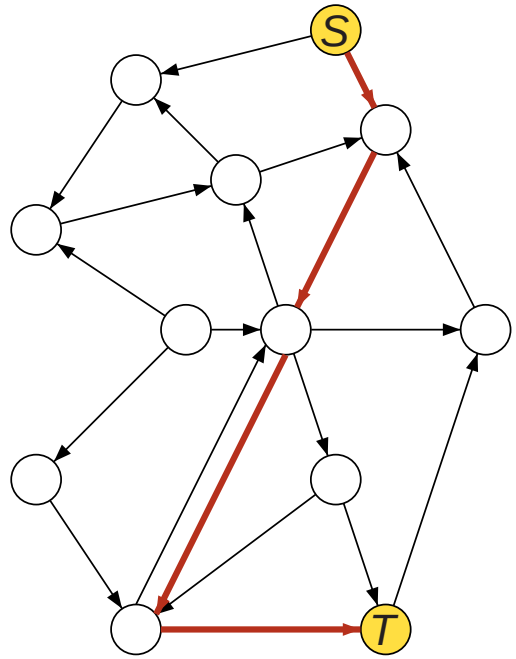
Station-to-Station Anfragen

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) :-)

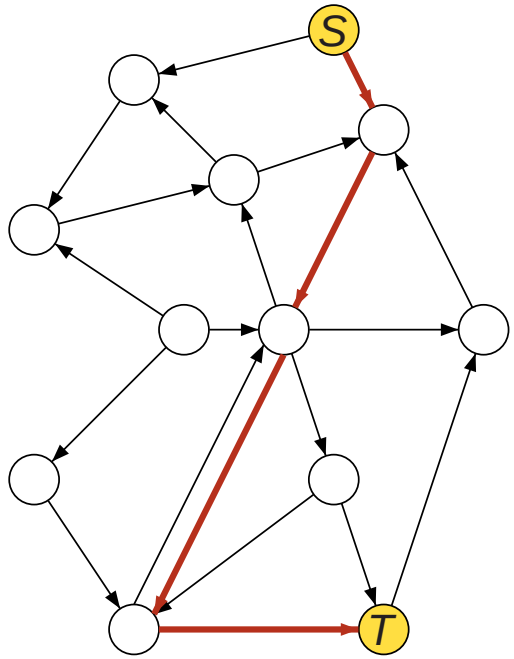
Station-to-Station Anfragen

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) :-)

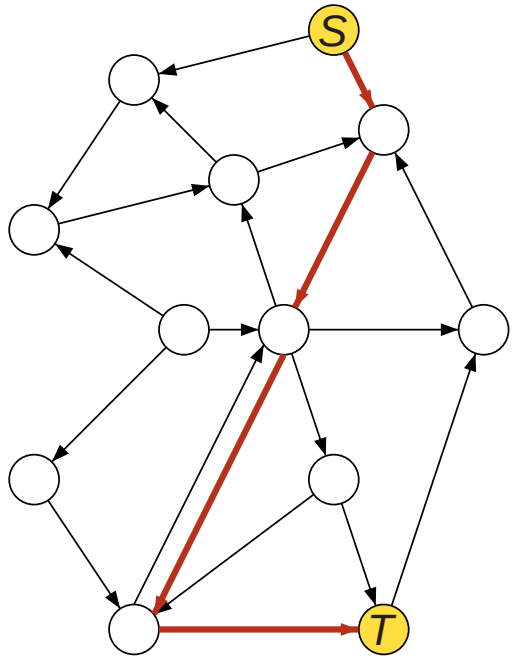
Station-to-Station Anfragen

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

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

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

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

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

Stoppkriterium

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

kann adaptiert werden durch

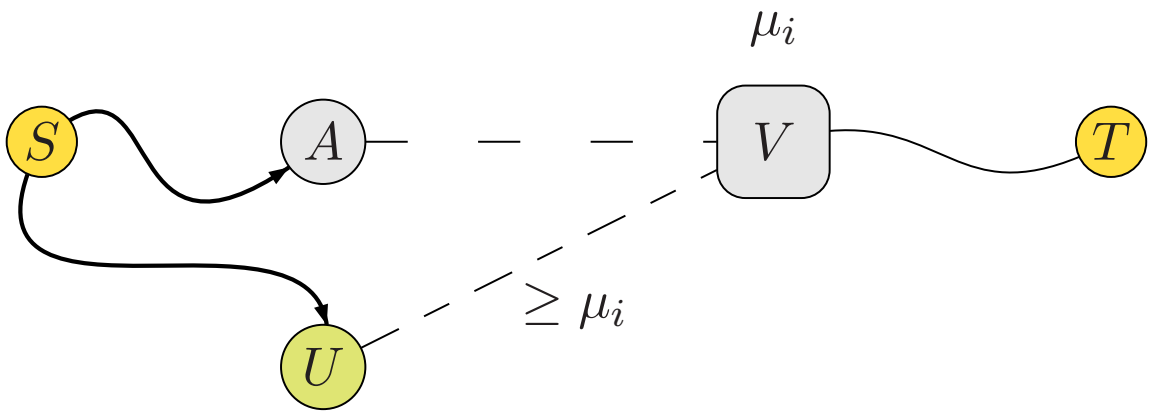
Parallel Self-Pruning Connection-Setting:

- 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:

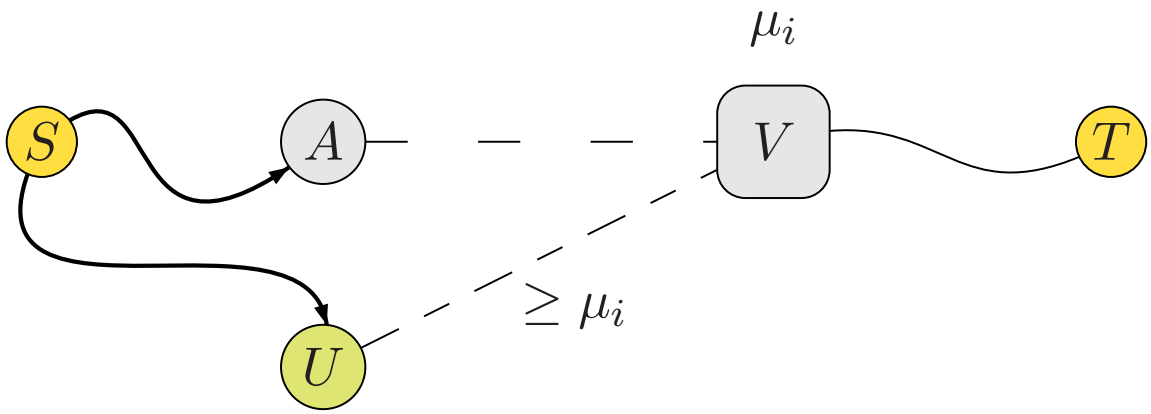
- Verwalte vorl. Distanz μ_i zu Transferstationen V "nahe" Ziel T
  μ_i ist **obere Schranke** bzgl. echtem Abstand zu V
- Prune Verbindung i an beliebiger Transferstation U wenn

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

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

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

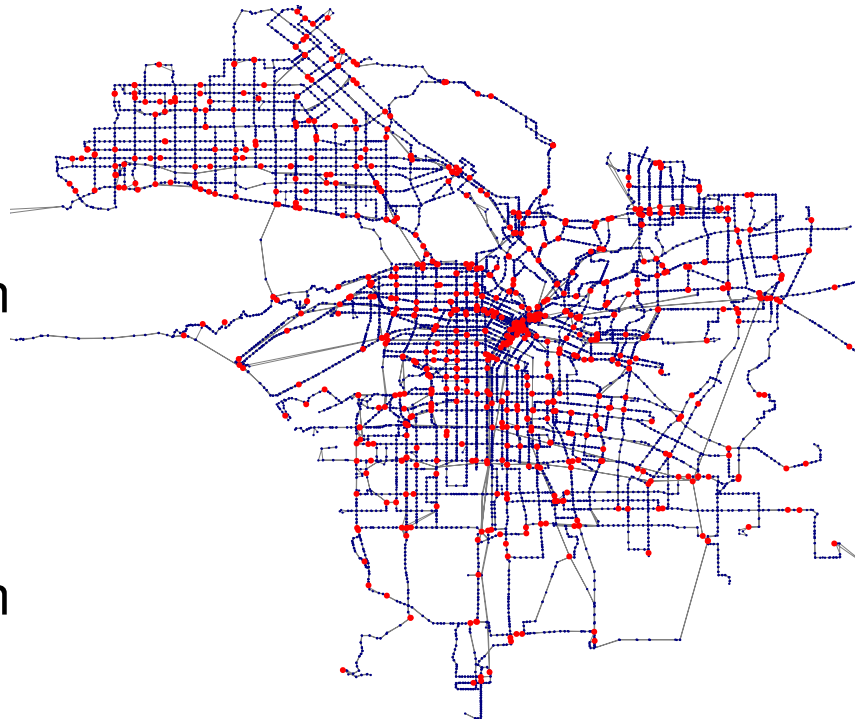
Eingabe

Netzwerk von **Los Angeles**:

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

Zugnetz von **Europa**:

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



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

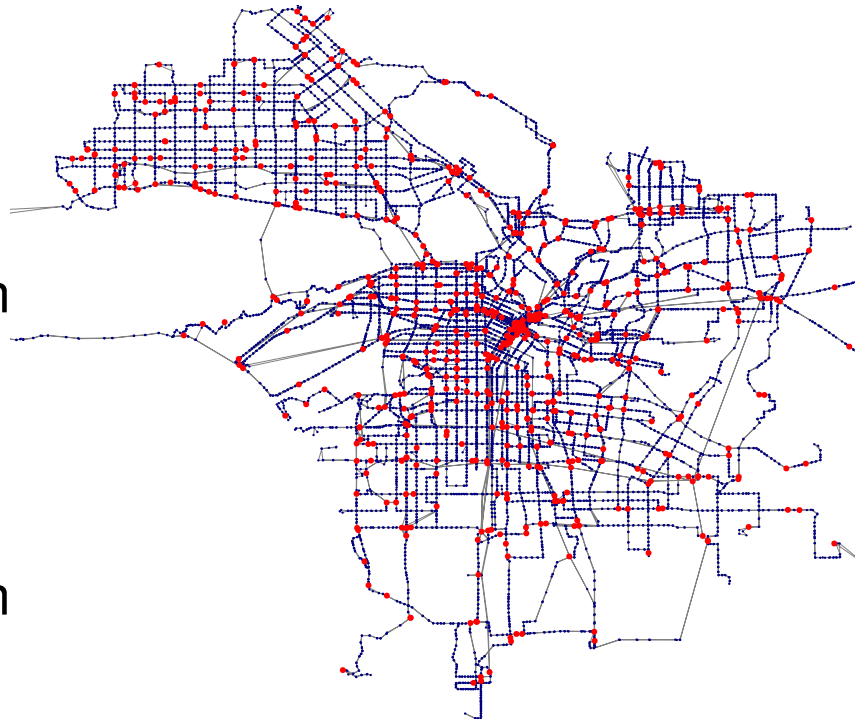
Eingabe

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

		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

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

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

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	—	—	—	—

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

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

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

Nochmal Straßennetzwerke

Karlsruher Institut für Technologie

Modellierung

- Straßennetzwerk als Graph G
- Kantengewichte = Reisezeit
- Beste Route $\hat{=}$ Kürzester Weg in G

Lösung

- Klassisches Problem: Dijkstra's Algorithmus
- Laufzeit in $O(m + n \log n)$
- **Aber:** Mehrere Sekunden auf Europa (Server-Hardware)

$\Rightarrow$ Beschleunigungstechniken!

Nochmal Straßennetzwerke

Karlsruher Institut für Technologie

Modellierung

- Straßennetzwerk als Graph G
- Kantengewichte = Reisezeit
- Beste Route $\hat{=}$ Kürzester Weg in G

Lösung

- Klassisches Problem: Dijkstra's Algorithmus
- Laufzeit in $O(m + n \log n)$
- **Aber:** Mehrere Sekunden auf Europa (Server-Hardware)

$\Rightarrow$ Beschleunigungstechniken!

Nochmal Straßennetzwerke

Karlsruher Institut für Technologie

Modellierung

- Straßennetzwerk als Graph G
- Kantengewichte = Reisezeit
- Beste Route $\hat{=}$ Kürzester Weg in G

Lösung

- Klassisches Problem: Dijkstra's Algorithmus
- Laufzeit in $O(m + n \log n)$
- **Aber:** Mehrere Sekunden auf Europa (Server-Hardware)

$\Rightarrow$ Beschleunigungstechniken!

Ergebnisse auf Straßennetzwerken

Karlsruher Institut für Technologie

Paradigma

- Offline-Phase:
  Vorberechnung zusätzlicher Informationen
- Online-Phase:
  Beschleunigung der Anfragen mit Hilfe der Informationen
  - Meist Dijkstra-basierte Algorithmen

Ergebnisse

- Verschiedene sehr effiziente Verfahren
- Bis zu 3 Millionen mal schneller als Dijkstra
- Kaum zusätzlicher Speicherbedarf

Ergebnisse auf Straßennetzwerken

	Vorberechnung		Anfrage		
	Zeit [h:m]	Platz [byte/n]	Such-raum	Zeit [ms]	Beschl.
Dijkstra	0:00	0	9 114 385	5 591.6	1
Bi-Dijkstra	0:00	0	4 764 110	2 713.2	2
ALT-16	1:25	128	74 669	53.6	104
Arc-Flags-128	17:08	10	2 764	0.8	6 988
RE	1:22	13	4 643	3.5	1 597
REAL-(64,16)	2:20	35	679	1.1	5 037
CH	0:32	-3.0	359	0.15	37 273
TNR	1:15	247	N/A	0.0043	$\approx$ 1 300 000
CALT-(med.,32)	0:10	10.6	1 704	1.6	3 368
gen. SHARC	1:21	14.5	654	0.29	19 281
gen. CHASE	1:39	12	45	0.0173	323 213
TNR+AF	3:49	321	N/A	0.0019	$\approx$ 3 000 000

Input: Europa, $\approx$ 18 Mio Knoten und $\approx$ 42 Mio Kanten

Ergebnisse auf Straßennetzwerken

Problem: Exakt oder Heuristisch?

- Alle Verfahren liefern beweisbar kürzeste Wege
- **ABER:**
  - Basieren auf Intuition und Experimentellen Studien
  - Keine Garantien bezüglich Platzverbrauch und Laufzeit
  - Meist für Straßennetzwerke “maßgeschneidert”

Ausgenutzte Intuition (Auswahl):

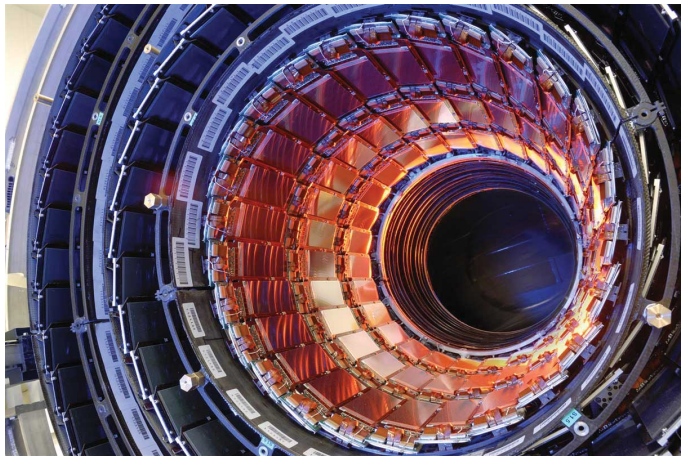
- Kürzeste Wege sind oft eindeutig
- Lange in die falsche Richtung fahren lohnt meist nicht
- Inheränte Hierarchie in Straßennetzwerken
- Es gibt wenige Knoten die für “lange” Wege wichtig sind
- Lokale Suchen sind “billig”

Die Wissenschaftliche Methode

Karlsruher Institut für Technologie

Die Wissenschaftliche Methode

- Beobachtung der Wirklichkeit
- Aufstellen einer Hypothese/Theorie
- Vorhersagen treffen
- Bestätigung durch Experiment



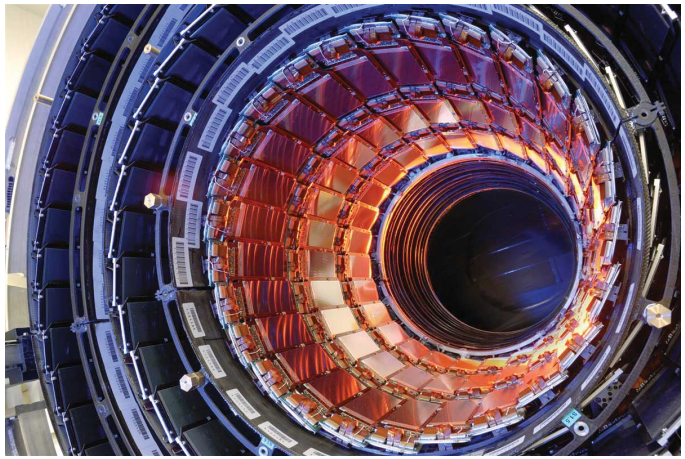
Versch. Vorgehen

- Mathematics of Algorithms $\Rightarrow$ Theoretische Aussagen
- Algorithm Engineering $\Rightarrow$ Praktikable Algorithmen
- **Science of Algorithms**: Mathematik $\Leftrightarrow$ Engineering
  - $\rightsquigarrow$ Benutzung der wissenschaftlichen Methode

Die Wissenschaftliche Methode

Die Wissenschaftliche Methode

- Beobachtung der Wirklichkeit
- Aufstellen einer Hypothese/Theorie
- Vorhersagen treffen
- Bestätigung durch Experiment



Versch. Vorgehen

- Mathematics of Algorithms $\Rightarrow$ Theoretische Aussagen
- Algorithm Engineering $\Rightarrow$ Praktikable Algorithmen
- **Science of Algorithms**: Mathematik $\Leftrightarrow$ Engineering
  - $\rightsquigarrow$ Benutzung der wissenschaftlichen Methode

Lücke: Praxis vs. Theorie

Lässt sich die Lücke zw. Praxis und Theorie schließen?

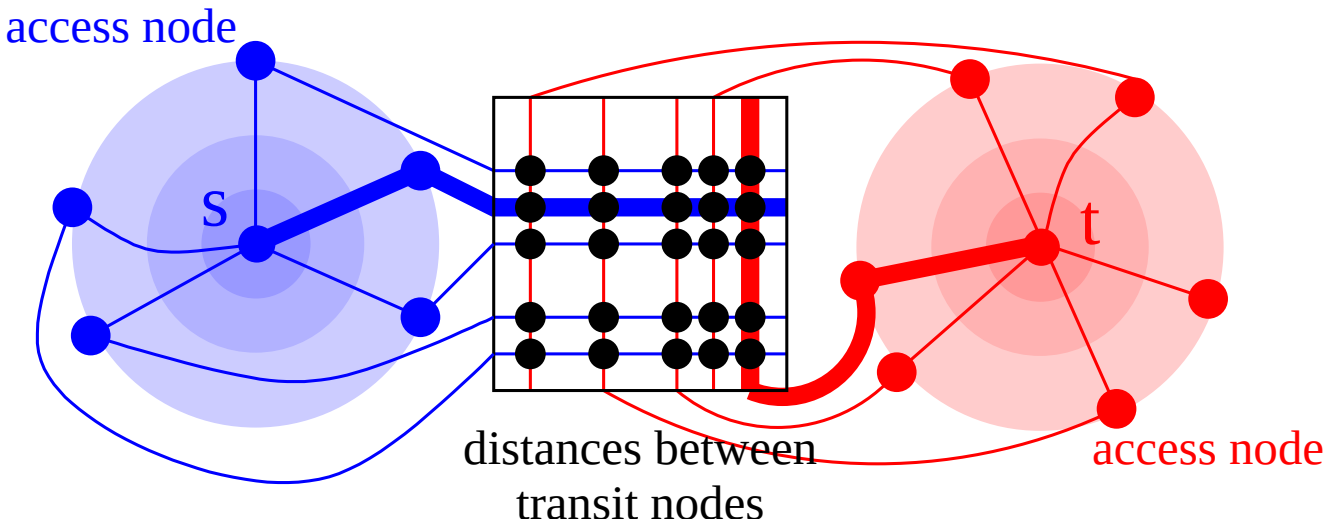
- **Warum** funktionieren die Beschleunigungstechniken so gut?
- **Was** zeichnet die Straßennetzwerke dazu aus?

Dieses Mal:
Ein erster Versuch einige Beschleunigungstechniken zu erklären.

Beob.: Transit-Node Routing

Idee:
Es gibt eine **kleine** Menge von **(Transit)-Knoten**, so dass

- jeder “lange” kürzeste Weg mindestens einen Transit-Knoten enthält
- für jeden Knoten *seine* wichtigen Transit-Knoten (Access-Nodes) nur sehr wenige sind



$\approx$ 10 000 Transit-Knoten u. $\approx$ 10 Access-Nodes pro Knoten (Europa)!

Formalisierung

Wie lässt sich diese Beobachtung formalisieren?

Werkzeuge

Voraussetzungen

- Graph $G = (V, E, \text{len})$
- ungerichtet (lässt sich auf gerichtet verallgemeinern)
- kürzeste Wege sind eindeutig
- Jede Kante $e = \{u, v\}$ ist kürzester Weg zw. u und v (sonst lösche e)

Kugel

Zu $r \in \mathbb{R}^+$ und $u \in V$ ist $\mathcal{B}_{u,r} := \{v \in V \ : \ |P(u, v)| \leq r\}$ die Kugel mit Radius r um u.

Durchmesser

Der Durchmesser D eines Graphen ist $D := \max_{u,v \in V} |P(u, v)|$

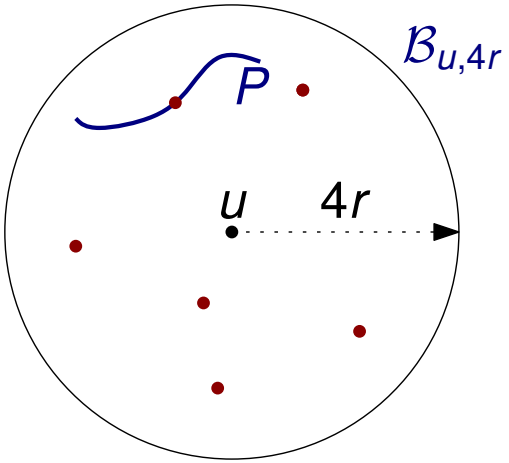
Maximaler Grad

Δ bezeichne den maximalen Grad eines Knotens in G.

Highway Dimension

Idee:
Lokal überdeckt eine kleine Menge Knoten alle hinreichend langen kürzesten Wege.

Gegeben:
Ungerichteter Graph $G = (V, E, \text{len})$.



Definition

Die *Highway-Dimension* von G ist die kleinste Zahl $h \in \mathbb{N}$, so dass

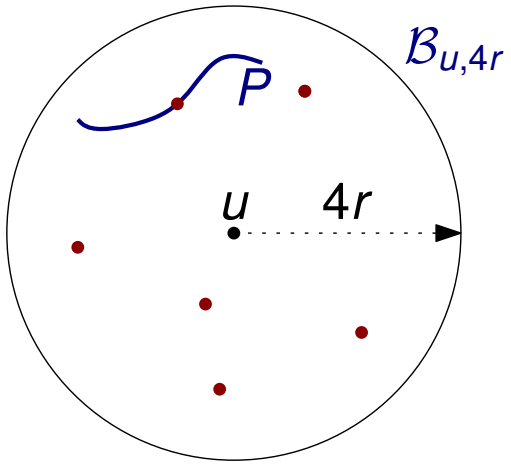
- Für alle $r \in \mathbb{R}^+$ und
- für alle Knoten $u \in V$
- existiert eine Menge $S \subseteq \mathcal{B}_{u,4r}$ mit $|S| \leq h$ so, dass

$$|P(v,w)| > r \text{ und } P(v,w) \subseteq \mathcal{B}_{u,4r} \quad \Rightarrow \quad P(v,w) \cap S \neq \emptyset$$

Highway Dimension

Idee:
Lokal überdeckt eine kleine Menge Knoten alle hinreichend langen kürzesten Wege.

Gegeben:
Ungerichteter Graph $G = (V, E, \text{len})$.



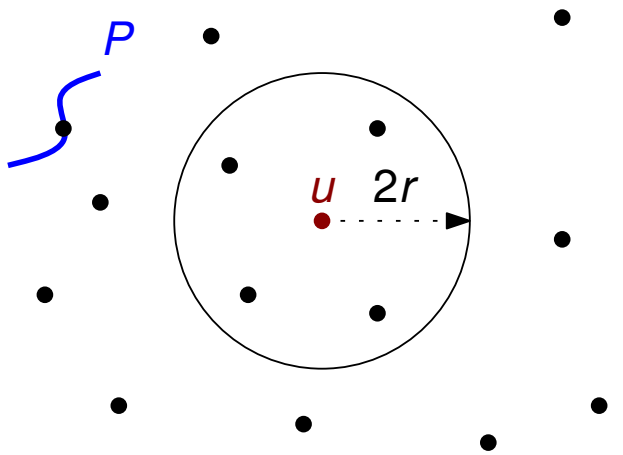
Definition

Die *Highway-Dimension* von G ist die kleinste Zahl $h \in \mathbb{N}$, so dass

- für jede Kugel $\mathcal{B}$ in G (mit Radius $4r$)
- eine Knoten-Menge S mit $|S| \leq h$ existiert, so dass
- jeder kürzeste Weg in $\mathcal{B}$ mit Mindestlänge r einen Knoten aus S enthält.

Shortest-Path Cover (SPC)

Idee:
Alle kürzesten Wege einer gewissen Länge können mit einer kleinen Menge von Knoten überdeckt werden.



Gegeben:
Ungerichteter Graph $G = (V, E, \text{len})$.

Definition

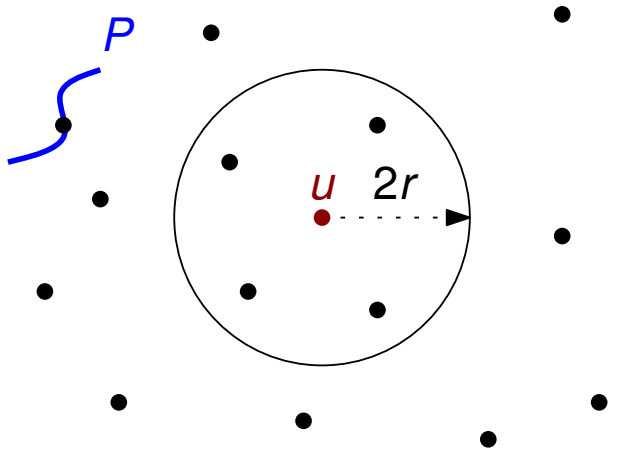
Eine Menge C heißt (r, k)-SPC von G genau dann wenn

- Für alle kürzesten Wege P mit $r < |P| \leq 2r$ gilt dass
- $P \cap C \neq \emptyset$ und
- für alle $u \in V$: $|C \cap \mathcal{B}_{u,2r}| \leq k$

Shortest-Path Cover (SPC)

Idee:
Alle kürzesten Wege einer gewissen Länge können mit einer kleinen Menge von Knoten überdeckt werden.

Gegeben:
Ungerichteter Graph $G = (V, E, \text{len})$.



Definition

Eine Menge C heißt (r, k)-SPC von G genau dann wenn

- jede Kugel $\mathcal{B}$ mit Radius $2r$ höchstens k Knoten aus C enthält und
- jeder kürzeste Weg P der Länge $r \leq |P| \leq 2r$ einen Knoten aus C enthält.

Highway Dimension und SPCs

Theorem

Wenn G Highway-Dimension h hat, dann $\forall r \; \exists$ ein (r, h)-SPC.

Beweis

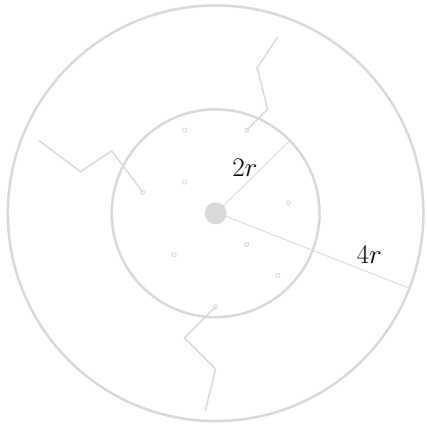
- Sei C^* eine *minimale* Menge die alle kürzesten Wege P mit $r < |P| \leq 2r$ überdeckt.
- zu Zeigen: C^* ist ein (r, h)-SPC

Ann.: $\exists u \in V$ so dass $U := C^* \cap \mathcal{B}_{u,2r}$ und $|U| > h$.

$\Rightarrow$ Nach Def. von h gibt es $H \subseteq V$ mit $|H| \leq h$ die alle kürzesten Wege überdeckt die in $\mathcal{B}_{u,4r}$ enthalten sind und länger als r sind.

$\Rightarrow$ H überdeckt alle KW P mit $r < |P| \leq 2r$

$\Rightarrow$ $|(C^* \setminus U) \cup H| < |C^*|$ und ist (r, h)-SPC. $\lightning$



Highway Dimension und SPCs

Theorem

Wenn G Highway-Dimension h hat, dann $\forall r \; \exists$ ein (r, h)-SPC.

Beweis

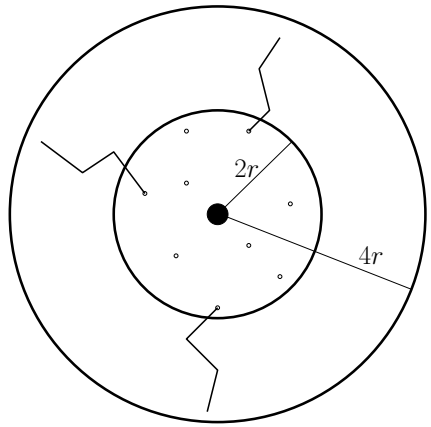
- Sei C^* eine *minimale* Menge die alle kürzesten Wege P mit $r < |P| \leq 2r$ überdeckt.
- zu Zeigen: C^* ist ein (r, h)-SPC

Ann.: $\exists u \in V$ so dass $U := C^* \cap \mathcal{B}_{u,2r}$ und $|U| > h$.

$\Rightarrow$ Nach Def. von h gibt es $H \subseteq V$ mit $|H| \leq h$ die alle kürzesten Wege überdeckt die in $\mathcal{B}_{u,4r}$ enthalten sind und länger als r sind.

$\Rightarrow$ H überdeckt alle KW P mit $r < |P| \leq 2r$

$\Rightarrow$ $|(C^* \setminus U) \cup H| < |C^*|$ und ist (r, h)-SPC. $\lightning$



Highway Dimension und SPCs

Theorem

Wenn G Highway-Dimension h hat, dann $\forall r \; \exists$ ein (r, h)-SPC.

Beweis

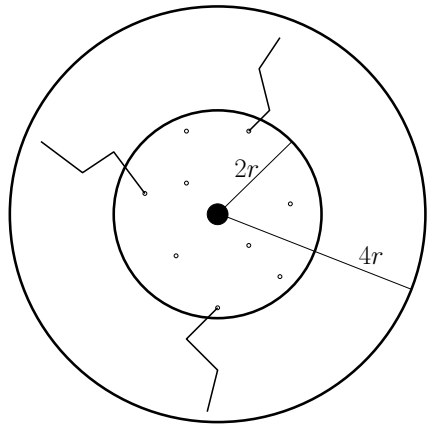
- Sei C^* eine *minimale* Menge die alle kürzesten Wege P mit $r < |P| \leq 2r$ überdeckt.
- zu Zeigen: C^* ist ein (r, h)-SPC

Ann.: $\exists u \in V$ so dass $U := C^* \cap \mathcal{B}_{u,2r}$ und $|U| > h$.

$\Rightarrow$ Nach Def. von h gibt es $H \subseteq V$ mit $|H| \leq h$ die alle kürzesten Wege überdeckt die in $\mathcal{B}_{u,4r}$ enthalten sind und länger als r sind.

$\Rightarrow$ H überdeckt alle KW P mit $r < |P| \leq 2r$

$\Rightarrow$ $|(C^* \setminus U) \cup H| < |C^*|$ und ist (r, h)-SPC. $\lightning$



Highway Dimension und SPCs

Theorem

Wenn G Highway-Dimension h hat, dann $\forall r \; \exists$ ein (r, h)-SPC.

Beweis

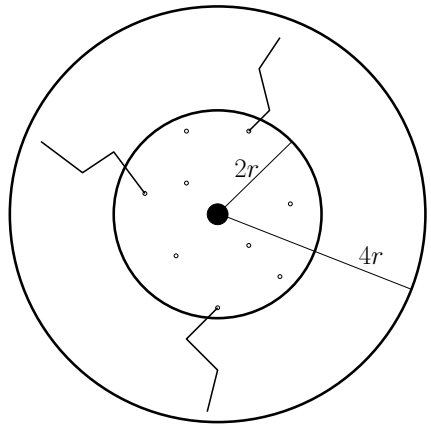
- Sei C^* eine *minimale* Menge die alle kürzesten Wege P mit $r < |P| \leq 2r$ überdeckt.
- zu Zeigen: C^* ist ein (r, h)-SPC

Ann.: $\exists u \in V$ so dass $U := C^* \cap \mathcal{B}_{u,2r}$ und $|U| > h$.

$\Rightarrow$ Nach Def. von h gibt es $H \subseteq V$ mit $|H| \leq h$ die alle kürzesten Wege überdeckt die in $\mathcal{B}_{u,4r}$ enthalten sind und länger als r sind.

$\Rightarrow$ H überdeckt alle KW P mit $r < |P| \leq 2r$

$\Rightarrow$ $|(C^* \setminus U) \cup H| < |C^*|$ und ist (r, h)-SPC. $\lightning$



Highway Dimension und SPCs

Theorem

Wenn G Highway-Dimension h hat, dann $\forall r \; \exists$ ein (r, h)-SPC.

Beweis

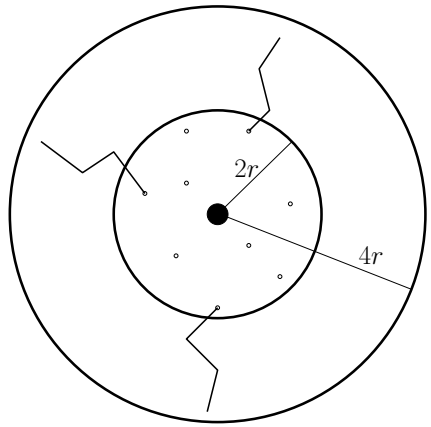
- Sei C^* eine *minimale* Menge die alle kürzesten Wege P mit $r < |P| \leq 2r$ überdeckt.
- zu Zeigen: C^* ist ein (r, h)-SPC

Ann.: $\exists u \in V$ so dass $U := C^* \cap \mathcal{B}_{u,2r}$ und $|U| > h$.

$\Rightarrow$ Nach Def. von h gibt es $H \subseteq V$ mit $|H| \leq h$ die alle kürzesten Wege überdeckt die in $\mathcal{B}_{u,4r}$ enthalten sind und länger als r sind.

$\Rightarrow$ H überdeckt alle KW P mit $r < |P| \leq 2r$

$\Rightarrow$ $|(C^* \setminus U) \cup H| < |C^*|$ und ist (r, h)-SPC. $\lightning$



Highway Dimension und SPCs

Problem:
Wie lässt sich C^* effizient konstruieren?

Ein minimales (r, h)-SPC zu finden ist $\mathcal{NP}$-schwer.

Also: Approximation.

Highway Dimension und SPCs

Karlsruher Institut für Technologie

Theorem

Wenn G Highway-Dimension h hat, dann lässt sich für alle r in polynomieller Zeit ein $(r, O(h \log n))$-SPC konstruieren.

Idee:

Greedy Set-Cover Algorithmus liefert $O(\log n)$-Approximation von C^*.

Highway Dimension und SPCs

Vorgehen zur Konstruktion eines $(r, O(h \log n))$-SPC:

- Beginne mit $C = \emptyset$
- Wähle iterativ den Knoten zu C der die meisten nicht-überdeckten KW überdeckt

Beweis

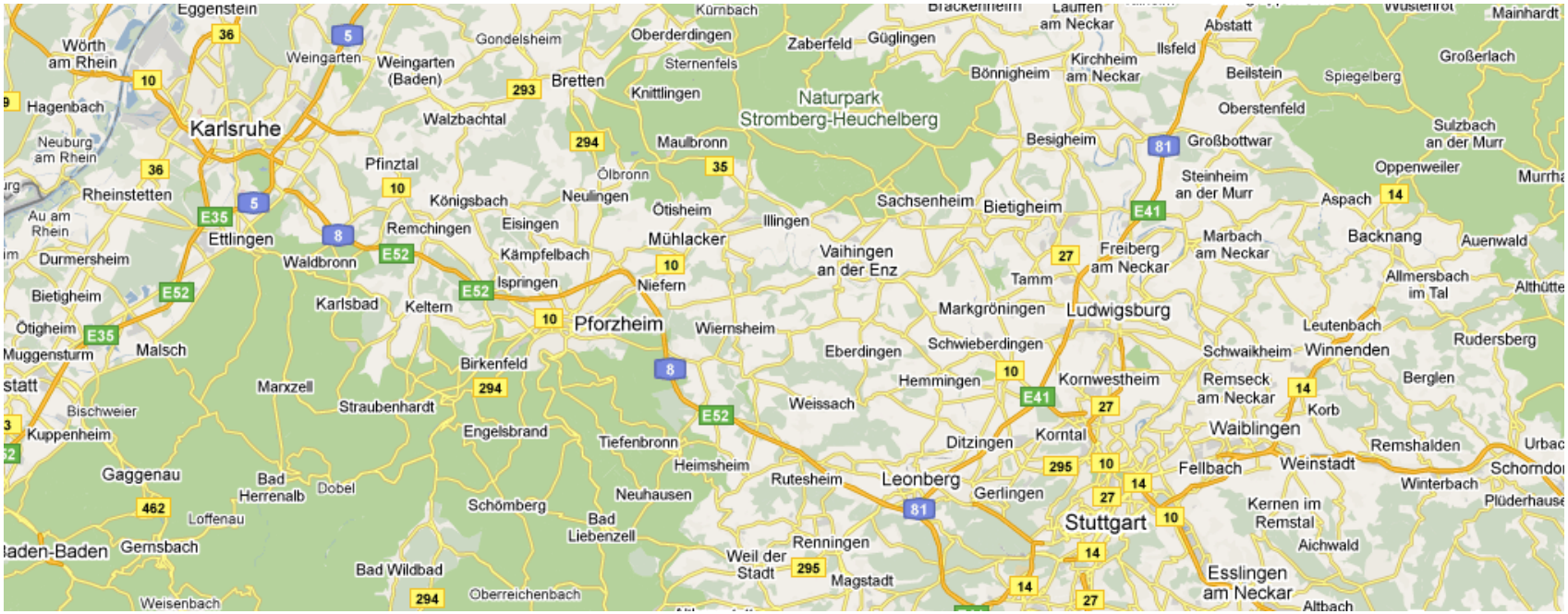
- In jedem Schritt:
  Mind. $1/h$ der unüberdeckten KW P mit $r < |P| \leq 2r$ wird überdeckt
- Es gibt $O(n^2)$ KW in G

$\Rightarrow$ Höchstens $O(h \log n)$ Knoten werden ausgewählt

HD von Straßennetzwerken

Vermutung:

Straßennetzwerke haben kleine (konstante) Highway-Dimension.



Generischer Prepro.-Algorithmus

Ziel:
Konstruktion eines Preprocessing-Verfahrens, so dass

- Platzverbrauch “klein” (möglichst linear in n)
- über die Query später Gütegarantien ausgesagt werden können
  - RE
  - CH
  - TNR
  - SHARC
  - . . . ?
- in Abhängigkeit von h statt n
- Zum Vergleich: Dijkstra $\in O(m + n \log n)$

Idee:
Benutze Preprocessing von Contraction Hierarchies (modifiziert)

Wdh.: Contraction Hierarchies

Preprocessing:

- ordne Knoten nach “Wichtigkeit”
- kontrahier Knoten in dieser Ordnung
- Knoten v wird kontrahiert durch

KONTRAHIERE(v)

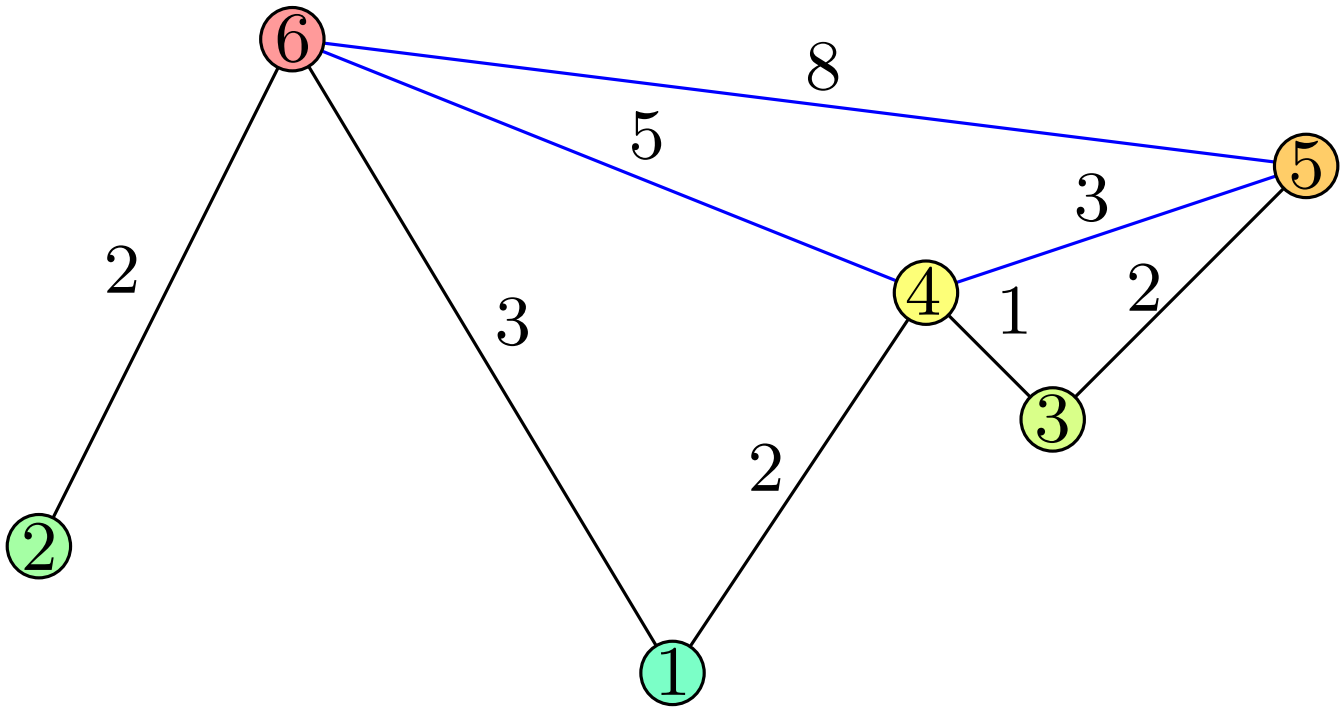
```
1 für alle Paare (u, v) und (v, w) von Kanten tue
2     wenn (u, v, w) eindeutiger kürzester Weg dann
3         Füge Shortcut (u, w) mit Gewicht len(u, v) + len(v, w) ein
```

Query:

- Bidirektional
- Vorwärtssuche: Relaxiert nur Kanten zu wichtigeren Knoten
- Rückwärtssuche: Relaxiert nur Kanten von wichtigeren Knoten

Wdh.: Contraction Hierarchies

Freiheitsgrad: In welcher Reihenfolge Knoten kontrahieren?



Generischer Prepro.-Algorithmus

Idee: Partitioniere V zu einer Hierarchie $L_0, L_1, \ldots, L_{\log D}$

Vorgehen

- Sei $C_0 := V$
- Sei C_i ein $(2^i, k)$-SPC für $i = 1 \ldots \log D$ wobei
  - $k = h$ für unbeschränktes Preprocessing
  - $k = h \log n$ für polynomialzeit Preprocessing (Approx.)
- Dann ist

$$L_i := C_i \setminus \bigcup_{j=i+1}^{\log D} C_j$$

- Kontrahiere Knoten in der Reihenfolge induziert durch L_i.

Ausgabe:

- Menge E^+ von eingefügten Shortcuts
- Hierarchie $L_0, L_1, \ldots, L_{\log D}$

Generischer Prepro.-Algorithmus

Idee: Partitioniere V zu einer Hierarchie $L_0, L_1, \ldots, L_{\log D}$

Vorgehen

- Sei $C_0 := V$
- Sei C_i ein $(2^i, k)$-SPC für $i = 1 \ldots \log D$ wobei
  - $k = h$ für unbeschränktes Preprocessing
  - $k = h \log n$ für polynomialzeit Preprocessing (Approx.)
- Dann ist

$$L_i := C_i \setminus \bigcup_{j=i+1}^{\log D} C_j$$

- Kontrahiere Knoten in der Reihenfolge induziert durch L_i.

Ausgabe:

- Menge E^+ von eingefügten Shortcuts
- Hierarchie $L_0, L_1, \ldots, L_{\log D}$

Generischer Prepro.-Algorithmus

Idee: Partitioniere V zu einer Hierarchie $L_0, L_1, \ldots, L_{\log D}$

Vorgehen

- Sei $C_0 := V$
- Sei C_i ein $(2^i, k)$-SPC für $i = 1 \ldots \log D$ wobei
  - $k = h$ für unbeschränktes Preprocessing
  - $k = h \log n$ für polynomialzeit Preprocessing (Approx.)
- Dann ist

$$L_i := C_i \setminus \bigcup_{j=i+1}^{\log D} C_j$$

- Kontrahiere Knoten in der Reihenfolge induziert durch L_i.

Ausgabe:

- Menge E^+ von eingefügten Shortcuts
- Hierarchie $L_0, L_1, \ldots, L_{\log D}$

Preprocessing: Platz

Lemma

Für $v \in L_i$ und festes $j > i$ ist die Anzahl Kanten $(v, w) \in E^+$ mit $w \in L_j$ höchstens k.

Beweis:

- Kante (v, w) repräsentiert Pfad P dessen innere Knoten *vor* v und w kontrahiert wurden.

$\Rightarrow$ Innere Knoten gehören zu L_x für $x \leq i \leq j$.

$\Rightarrow$ $w \in \mathcal{B}_{v,2 \cdot 2^j}$ (da sonst P durch u mit $u \in L_{y>j}$ abgedeckt sein müsste).

- Def. $(2^j, k)$-SPC besagt, dass $B_{v,2 \cdot 2^j}$ höchstens k Knoten aus L_j enthält.

Preprocessing: Platz

Lemma

Für $v \in L_i$ und festes $j > i$ ist die Anzahl Kanten $(v, w) \in E^+$ mit $w \in L_j$ höchstens k.

Beweis:

- Kante (v, w) repräsentiert Pfad P dessen innere Knoten *vor* v und w kontrahiert wurden.

$\Rightarrow$ Innere Knoten gehören zu L_x für $x \leq i \leq j$.

$\Rightarrow$ $w \in \mathcal{B}_{v,2 \cdot 2^j}$ (da sonst P durch u mit $u \in L_{y>j}$ abgedeckt sein müsste).

- Def. $(2^j, k)$-SPC besagt, dass $B_{v,2 \cdot 2^j}$ höchstens k Knoten aus L_j enthält.

Preprocessing: Platz

Theorem

Für jeden Graphen G mit Highway Dimension h gibt es eine Knotenordnung, durch die das CH-Preprocessing eine Menge von Shortcuts E^+ so erzeugt, dass gilt

- *Der Grad jedes Knotens in $G^+ = (V, E \cup E^+)$ ist höchstens $\Delta + h \log D$*
- $|E^+| \in O(nh \log D)$

Für Polynomialzeit-Preprocessing:

- Maximalgrad in G^+ ist in $O(\Delta + h \log n \log D)$
- $|E^+| \in O(nh \log n \log D)$

Ende

Karlsruher Institut für Technologie

Literatur (Profilsuchen):

- **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

Literatur (Highway Dimension):

- Ittai Abraham, Amos Fiat, Andrew Goldberg und Renato Werneck:
  Highway Dimension, Shortest Paths, and Provably Efficient Algorithms
  In: *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA10)*, 2010.