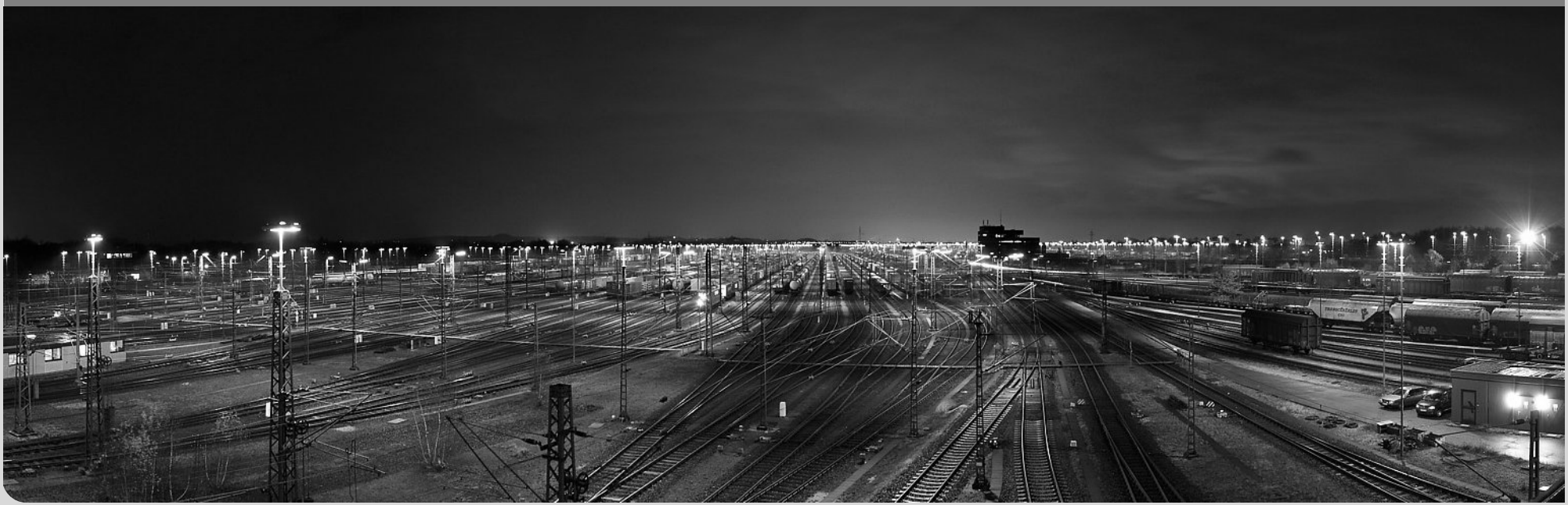
Karlsruher Institut für Technologie

Algorithmen für Routenplanung

18. Vorlesung, Sommersemester 2016

Ben Strasser | 4. Juli 2016

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



Züge vs Straße

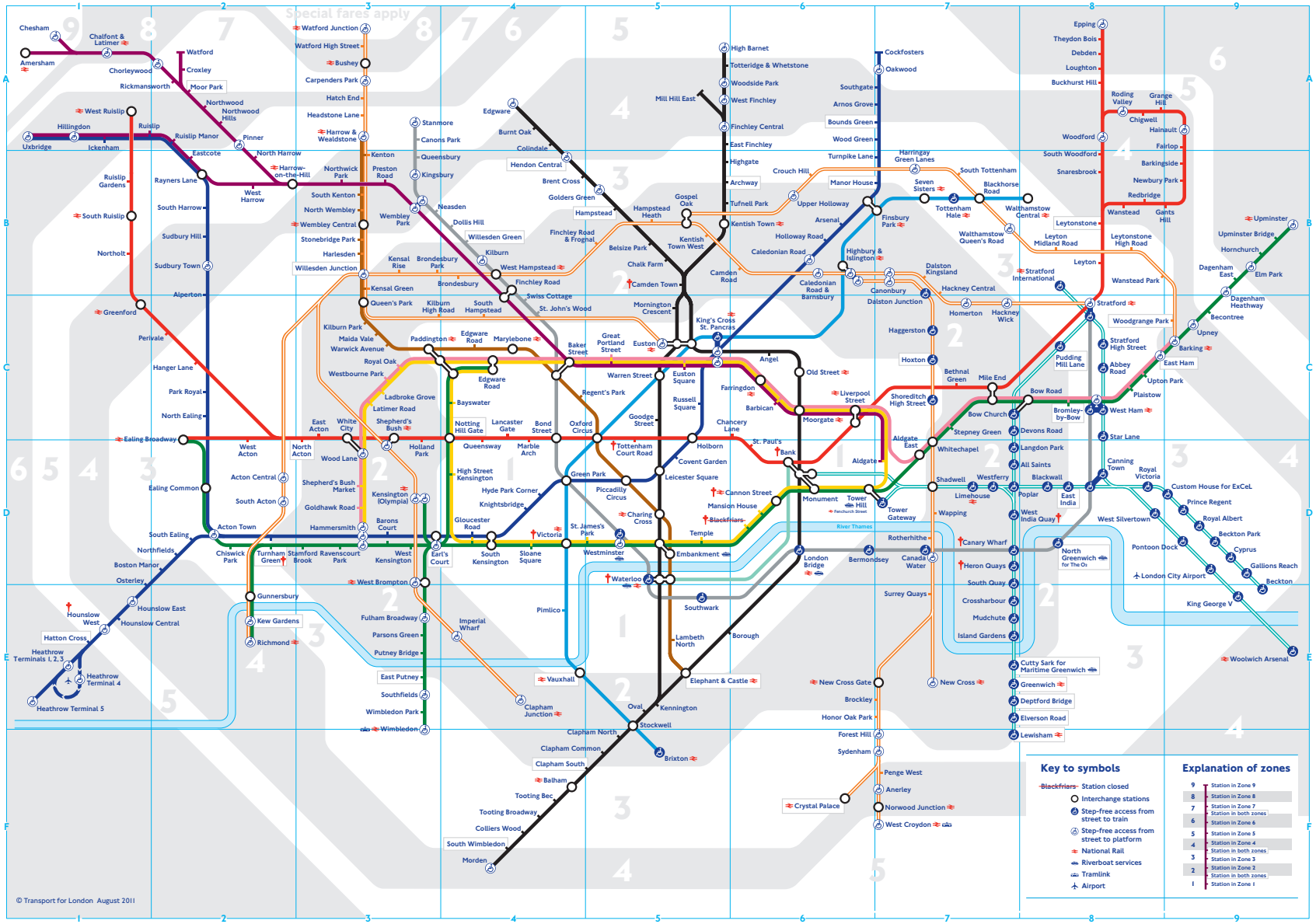
- **Bisher**: Auto auf Straße
- **Jetzt**: Züge auf Schiene
  (oder Flugzeuge, Schiffe, . . .alles mit festem Fahrplan)

Züge vs Straße

- **Bisher**: Auto auf Straße
- **Jetzt**: Züge auf Schiene
  (oder Flugzeuge, Schiffe, . . .alles mit festem Fahrplan)

- Fahrplanauskunft genannt
- Anwendungen:
  - `http://www.bahn.de`
  - `http://www.kvv.de`
  - `...`

Fahrplanauskunft



Karlsruher Institut für Technologie
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Wirrwarr

Auf der Straße:

- Etablierte Formalisierung: Kürzester Weg durch Graph
- Etablierte Terminologie: Jeder redet von Knoten und Kanten
- Etablierte Testinstanz: Die meisten testen auf DIMACS-Europa

Wirrwarr

Auf der Straße:

- Etablierte Formalisierung: Kürzester Weg durch Graph
- Etablierte Terminologie: Jeder redet von Knoten und Kanten
- Etablierte Testinstanz: Die meisten testen auf DIMACS-Europa

Auf der Schiene:

- Kaum zwei Paper wirklich vergleichbar
- Jeder benennt die Sachen verschieden
- Details der Problemstellung sind oft verschieden
- Testinstanz unterschieden sich stark
- Dennoch: Interessante algorithmische Problemstellungen
- Versuch es in der Vorlesung es halbwegs konsistent zu halten (nicht immer möglich)

Die Eingabe: Der Fahrplan

Ein Fahrplan besteht aus:

- Stops
- Connections
- Trips
- Fußwege

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Die Eingabe: Stops

- Stops sind Positionen an denen ein Fahrgast stehen kann
- Beispiel:
  - Karl-Wilhelm-Platz
  - Karlstor
  - ...
- Jeder Stop s hat eine minimale Umstiegszeit $\text{change\_time}(s)$
  - Zwischen ankommenden und abfahrend Züge muss mindestens $\text{change\_time}(s)$ Zeit sein um umsteigen zu können

Aber:

- Sind Doppelstops wie Kronenplatz (Kaiserstraße) und Kronenplatz (Fritz-Erler-Str.) ein oder zwei Stops?
- Ist der Hauptbahnhof ein Stop oder ist jedes Gleis ein Stop?
- Mehr dazu später

Die Eingabe: Connections

- Eine Connection c ist ein Zug der zwischen zwei Stops fährt und **nicht** hält
- Besteht aus:
  - Abfahrtsstop-ID c_{depstop}
  - Ankunftsstop-ID c_{arrpstop}
  - Abfahrtszeit c_{deptime}
  - Ankunftszeit c_{arrtime}
  - Trip-ID c_{tripid} (siehe nächste Folie)
- Beispiel:
  - S-Bahn vom Europaplatz zur Herrenstraße
  - S-Bahn von der Herrenstraße zum Marktplatz
  - **Aber nicht**: S-Bahn vom Europaplatz zum Marktplatz (da die S-Bahn in der Herrenstraße hält)

Die Eingabe: Trip

- Ein Trip ist eine Folge von Connections die vom selben Zug bedient werden
- Beispiel:
  - Die S1 die um 15:55 am Marktplatz ankommt
  - Connection *A* vom Europaplatz zur Herrenstraße und Connection *B* von der Herrenstraße zum Marktplatz können zum selben Trip gehören
  - **Aber nicht**: Die S1
    (nicht alle S1-Fahrten werden vom selben Zug bedient)

Die Eingabe: Fußwege

- Ein Fußweg ist eine gerichtete Kante zwischen zwei Stops
- Besteht aus:
  - Abfahrtsstop
  - Ankunftsstop
  - Laufzeit
- Beispiele:
  - Kante zwischen Hauptbahnhof-Gleise und Hauptbahnhof-Vorplatz
  - Kante zwischen Kronenplatz (Kaiserstraße) und Kronenplatz (Fritz-Erler-Str.)
  - Kante zwischen Europaplatz und Herrenstraße

Wann darf ich Umsteigen?

- Kaum zwei Paper betrachten das selbe Umstiegsmodell
- Viele Realwelt-Instanzen haben unterschiedliche Modelle
- Viele Paper ignorieren Fußwege
- Bei anderen sind Umstiegszeiten alle 0
- Großes Wirrwarr
- und leider machen die Fußweg-Details algorithmisch große Unterschiede

Umstiege bei uns

- Wenn
  - ich von Stop A nach Stop B und
  - von Stop B nach Stop C laufen darf
  - dann darf ich auch von A nach C laufen
- Von Stop A nach Stop B ist es nie länger als ein Umweg über einen weiteren Stop C
- Die Umstiegszeit an einem Stop A ist kürzer als jeder Pfad der bei A beginnt und bei A endet

Umstiege bei uns

- Wenn
  - ich von Stop A nach Stop B und
  - von Stop B nach Stop C laufen darf
  - dann darf ich auch von A nach C laufen
- Von Stop A nach Stop B ist es nie länger als ein Umweg über einen weiteren Stop C
- Die Umstiegszeit an einem Stop A ist kürzer als jeder Pfad der bei A beginnt und bei A endet

Äquivalente Sichtweise

- Stops sind partitioniert
- Für jede Partition gibt es eine vollständige kürzeste Wege-Matrix
- Jede Matrix erfüllt Dreieckungleichung
- Matrix-Diagonale sind Umstiegszeiten

Umstiege bei uns

- Wir wollen quadratische Matrizen klein halten
- Wir wollen deswegen wenige Fußwege haben

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Umstiege bei uns

- Wir wollen quadratische Matrizen klein halten
- Wir wollen deswegen wenige Fußwege haben
- Macht aus Kronenplatz (Kaiserstraße) und Kronenplatz (Fritz-Erler-Str.) am besten ein Stop Kronenplatz mit Umstiegszeit

Umstiege bei uns

- Wir wollen quadratische Matrizen klein halten
- Wir wollen deswegen wenige Fußwege haben
- Macht aus Kronenplatz (Kaiserstraße) und Kronenplatz (Fritz-Erler-Str.) am besten ein Stop Kronenplatz mit Umstiegszeit
- Gleise am Bahnhof sind ein Stop mit entsprechender Umstiegszeit

Umstiege bei uns

- Wir wollen quadratische Matrizen klein halten
- Wir wollen deswegen wenige Fußwege haben
- Macht aus Kronenplatz (Kaiserstraße) und Kronenplatz (Fritz-Erler-Str.) am besten ein Stop Kronenplatz mit Umstiegszeit
- Gleise am Bahnhof sind ein Stop mit entsprechender Umstiegszeit
- Bahnhof-Vorplatz und Bahnhof-Halle sind zu weit voneinander weg $\rightarrow$ Hier ist ein Fußweg sinnvoll

Umstiege bei uns

- Wir wollen quadratische Matrizen klein halten
- Wir wollen deswegen wenige Fußwege haben
- Macht aus Kronenplatz (Kaiserstraße) und Kronenplatz (Fritz-Erler-Str.) am besten ein Stop Kronenplatz mit Umstiegszeit
- Gleise am Bahnhof sind ein Stop mit entsprechender Umstiegszeit
- Bahnhof-Vorplatz und Bahnhof-Halle sind zu weit voneinander weg $\rightarrow$ Hier ist ein Fußweg sinnvoll
- Keine Fußwege zwischen benachbarten Stops
  - Wenn ich vom Europaplatz zur Herrenstraße laufen darf,
  - dann darf ich auch zum Marktplatz weiterlaufen
  - und von da zum Kronenplatz
  - und von da zum Mendelsonplatz
  - ...
  - und auf einmal sind wir am Hauptbahnhof und haben eine gigantische Fußweg-Matrix

Die Ausgabe: Journeys

- Ein Leg ist ein Paar (c_1, c_2) von Einstiegs-Connection c_1 und Ausstiegs-Connection c_2, so dass c_1 und c_2 im selben Trip sitzen und c_2 nach c_1 abfährt
- Eine Journey ist eine alternierende Folge von Legs und Fußwegen oder Umstiegen
- Zwei aufeinanderfolgende Legs in einer Journey müssen
  - Am selben Stop A ankommen und abfahren und die Umstiegszeit an A respektieren
  - Bei A ankommen und bei B abfahren und die Laufzeit eines Fußwegs von A nach B respektieren
- Es darf auf ein Fußweg am Anfang und Ende einer Journey geben
- Eine Journey darf auch nur aus einem Fußweg bestehen
- Zwei direkt aufeinanderfolgende Fußwege sind wegen Dreieckungleichung nie sinnvoll

Problem der frühesten Ankunft

Eingabe:

- Fahrplan
- Startstop s
- Zielstop t
- Startzeit τ

Ausgabe:

- Journey von s nach t die nach τ abfährt

Optimierungsfunktion:

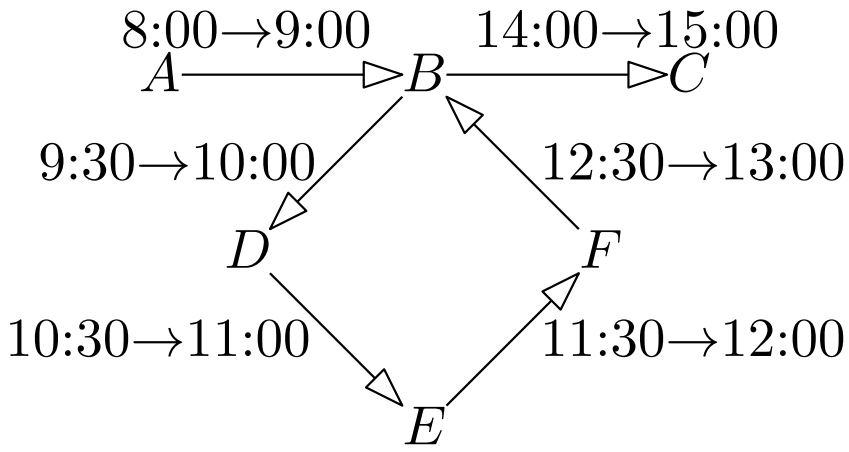
- Minimiere Ankunftszeit an t

Problem der frühesten Ankunft

- Problem der frühesten Ankunft entspricht am ehesten kürzesten Wege Problem in Graphen
- Kürzestes Wege Problem in Graphen ist nützlich
- Ist das Problem der frühesten Ankunft nützlich?

Problem der frühesten Ankunft

- Betrachte folgenden Fahrplan

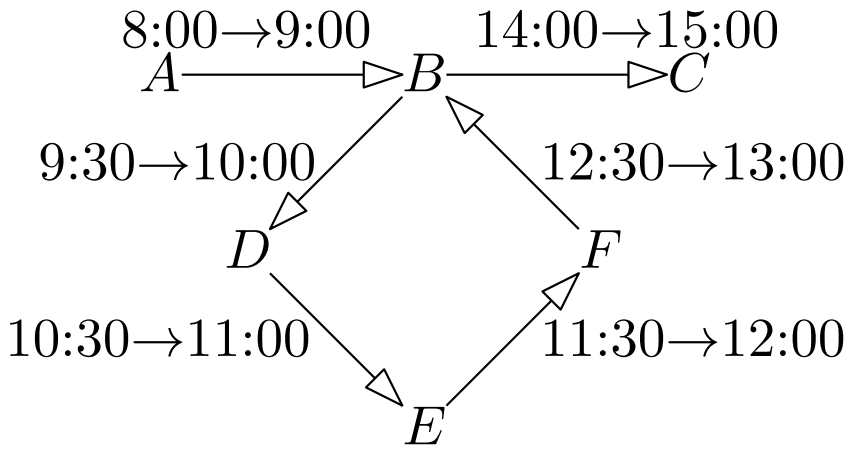


und

- $s = A$
- $t = B$
- $\tau = 8{:}00$

Problem der frühesten Ankunft

- Betrachte folgenden Fahrplan



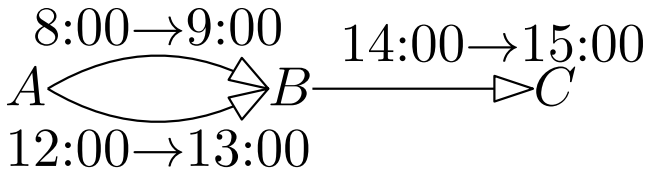
und

- $s = A$
- $t = B$
- $\tau = 8{:}00$

- Optimale Journeys sind:
  - $A \to B \to C$
  - $A \to B \to D \to E \to F \to B \to C$
- Den zweiten will man sicher nicht haben

Problem der frühesten Ankunft

- Betrachte folgenden Fahrplan

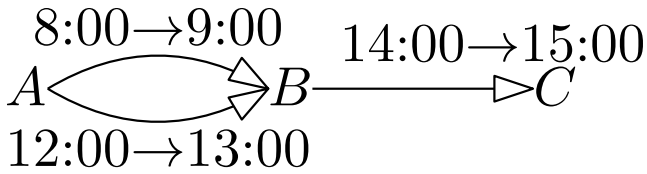


und

- $s = A$
- $t = B$
- $\tau = 8{:}00$

Problem der frühesten Ankunft

- Betrachte folgenden Fahrplan



  und
  - $s = A$
  - $t = B$
  - $\tau =$ 8:00

- Optimale Journeys sind:
  - A@8 $\rightarrow B \rightarrow C$@15
  - A@12 $\rightarrow B \rightarrow C$@15
- Den ersten will man auch nicht haben

Mehrere Kriterien und Eindeutigkeit

Auf der Straße:

- Einfache klare Zielfunktion: Minimiere Reisezeit
- Lösung in der Regel eindeutig

Auf der Schiene:

- Zielfunktion unklar, meistens minimiert man in irgendeiner Kombination:
  - Ankunftszeit
  - Anzahl an Umstiegen
- Oft gibt es mehrere optimale Journeys
  - Nur letzte Connection legt Ankunftszeit fest
  - Viele “äquivalente” Umstiege

Weitere Kriterien

Weitere Kriterien

- Oft möchte man weitere Kriterien optimieren
- Beispiel:
  - Preis minimieren
  - Laufwege minimieren
  - Unsicherheit minimieren
- Aber: Jedes weitere Kriterium macht das Gesamtproblem schwieriger

Probleme beim Kriterium Preis

- Oft gewünscht, kann aber niemand genau lösen

Probleme beim Kriterium Preis

- Oft gewünscht, kann aber niemand genau lösen
- Wie berechne ich den Preis einer gegeben Journey?
  - Genaue Preispolitik oft intransparent
  - Genaue Berechnung scheitert oft aus politischen Gründen
  - Optimierung also nicht möglich

Probleme beim Kriterium Preis

- Oft gewünscht, kann aber niemand genau lösen
- Wie berechne ich den Preis einer gegeben Journey?
  - Genaue Preispolitik oft intransparent
  - Genaue Berechnung scheitert oft aus politischen Gründen
  - Optimierung also nicht möglich
- Preise manchmal absurd
  - Verkehrsverbund-Tickets oft billiger als DB-Tickets
  - Verkehrsverbund-Tickets können nicht immer in Bahnen gelöst werden die durch einen Verkehrsverbund fahren
  - “optimale” Journey steigt also an Verbundsgrenzen aus um neues Ticket zu lösen

Probleme beim Kriterium Preis

- Oft gewünscht, kann aber niemand genau lösen
- Wie berechne ich den Preis einer gegeben Journey?
  - Genaue Preispolitik oft intransparent
  - Genaue Berechnung scheitert oft aus politischen Gründen
  - Optimierung also nicht möglich
- Preise manchmal absurd
  - Verkehrsverbund-Tickets oft billiger als DB-Tickets
  - Verkehrsverbund-Tickets können nicht immer in Bahnen gelöst werden die durch einen Verkehrsverbund fahren
  - “optimale” Journey steigt also an Verbundsgrenzen aus um neues Ticket zu lösen
- Oft approximiert man den Preis durch
  - die Distanz
  - die Zugklassen (ICE vs IC vs Regio)
  - durch die Anzahl an traversierten Tarifwaben

Kriterien Kombinieren

- Man kann mehrere Kriterien im Pareto-Sinn optimieren
  - Hinter einem Pareto-Trade-Off Punkt können sich mehrere Journeys verbergen
  - Möchte man alle oder nur eine Journey pro Pareto-Trade-Off Punkt?
  - **Hinweis**: Bei zu vielen Kriterien können die Pareto-Menge sehr groß werden, selbst bei nur einer Journey pro Punkt

Kriterien Kombinieren

- Man kann mehrere Kriterien im Pareto-Sinn optimieren
  - Hinter einem Pareto-Trade-Off Punkt können sich mehrere Journeys verbergen
  - Möchte man alle oder nur eine Journey pro Pareto-Trade-Off Punkt?
  - **Hinweis**: Bei zu vielen Kriterien können die Pareto-Menge sehr groß werden, selbst bei nur einer Journey pro Punkt
- Man kann die Kriterien ordnen
  - **Beispiel**: Unter allen Journeys mit frühester Ankunft wähle eine mit minimaler Anzahl an Transfers
  - Kombination mit Runden möglich: z.B. Ankunft um 10:00 und 10:01 zum selben Zeitpunkt runden

Kriterien Kombinieren

- Man kann mehrere Kriterien im Pareto-Sinn optimieren
  - Hinter einem Pareto-Trade-Off Punkt können sich mehrere Journeys verbergen
  - Möchte man alle oder nur eine Journey pro Pareto-Trade-Off Punkt?
  - **Hinweis**: Bei zu vielen Kriterien können die Pareto-Menge sehr groß werden, selbst bei nur einer Journey pro Punkt
- Man kann die Kriterien ordnen
  - **Beispiel**: Unter allen Journeys mit frühester Ankunft wähle eine mit minimaler Anzahl an Transfers
  - Kombination mit Runden möglich: z.B. Ankunft um 10:00 und 10:01 zum selben Zeitpunkt runden
- Man kann lineare Kombinationen betrachten
  - **Beispiel**:Ein Transfer ist äquivalent zu einer 10 Minuten späteren Ankunft

Profile

- Oft kennt der Fahrgast seine genaue Abfahrtszeit nicht
- $\rightarrow$ Profil-Anfragen lösen

Profile

- Oft kennt der Fahrgast seine genaue Abfahrtszeit nicht
- $\rightarrow$ Profil-Anfragen lösen

Eingabe:

- Minimale Abfahrtszeit τ_{min}
- Maximale Ankunftszeit τ_{max}

Ausgabe:

- Alle Journeys
  - mit einer Abfahrtszeit nach τ_{min} und
  - einer Ankunftszeit vor τ_{max}
  - die für irgendeinen Startzeitpunkt optimal sind
  - Bei mehreren Journeys die bezüglich aller Kriterien gleich sind reicht eine

Profile

- Profilsuchen kann man als Pareto-Optimierung auffassen
- Wir wollen alle Journeys finden, so dass:
  - Abfahrtszeit am Start maximiert wird
  - und Ankunftszeit am Ziel minimiert wird
- mit Nebenbedingung:
  - Nicht vor τ_{min} abfahren
  - Nicht nach τ_{max} ankommen

Instanzen

Auf der Straße

- Große integrierte Instanzen seit Jahren verfügbar
- DIMACS Graphen
- OpenStreetMap Graphen

Instanzen

Auf der Straße

- Große integrierte Instanzen seit Jahren verfügbar
- DIMACS Graphen
- OpenStreetMap Graphen

Auf der Schiene

- Viele winzige GTFS-Fahrpläne offen
  - General/Google Transit Feed Specification
- In Europa wird eher HAFAS verwendet
  - Format von HaCon
  - Oft sind Daten nicht offen
- Keine große offene integrierte Instanz verfügbar
- Wir haben Zugriff auf alte bahn.de Daten, allerdings mit NDA und nur für Forschung

Instanzen

- Realwelt Daten oft fehlerhaft
- Erster Schritt: Datenfehler beseitigen
- Beispiele für Datenfehler:
  - Züge in die Vergangenheit $c_{\text{dep\_time}} > c_{\text{arr\_time}}$
  - Instentane Züge, d.h., $c_{\text{dep\_time}} = c_{\text{arr\_time}}$
  - Schleifen, d.h., $c_{\text{dep\_stop}} = c_{\text{arr\_stop}}$
  - Instentane Trips mit Connections a, b, c wo, $a_{\text{dep\_time}} = b_{\text{dep\_time}} = c_{\text{dep\_time}} = \ldots$
  - Stops die nie erreicht werden
  - Stops wo man hinkommt aber nicht mehr weg
  - ...

Zeit-Expandierter Graph

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Zeit-Expandierter Graph

Idee

1. Baue aus dem Fahrplan einen Graph
2. Wende Dijkstras Algorithmus auf diesen Graphen an

Zeit-Expandierter Graph

- Knoten stellen Paare von räumlichen und zeitliche Positionen eines Fahrgasts dar
- Beispiel:
  - Am Marktplatz um 8:00
  - In der dritten Fahrt der S1 um 9:00
  - **Aber nicht**: Am Marktplatz
    (zeitliche Komponente fehlt)
  - **Aber nicht**: In der S1
    (zeitliche Komponente fehlt)
  - **Und nicht**: Um 10:00
    (örtliche Komponente fehlt)

Zeit-Expandierter Graph

- Knoten stellen Paare von räumlichen und zeitliche Positionen eines Fahrgasts dar
- Beispiel:
  - Am Marktplatz um 8:00
  - In der dritten Fahrt der S1 um 9:00
  - **Aber nicht**: Am Marktplatz
    (zeitliche Komponente fehlt)
  - **Aber nicht**: In der S1
    (zeitliche Komponente fehlt)
  - **Und nicht**: Um 10:00
    (örtliche Komponente fehlt)
- **Problem**: Unendliche viele Knoten da Zeit kontinuierlich ist
- **Idee**: Nur Knoten an Zeitpunkten an denen etwas passiert

Zeit-Expandierter Graph

- Für jede Connection erstellen wir drei Knoten:
  - Die Stehe-Am-Stop-Knoten $(c_{\text{depstop}}, c_{\text{deptime}})$ und $(c_{\text{arrstop}}, c_{\text{arrtime}} + \text{changetime}(c_{\text{arrstop}}))$
  - Den Sitze-Im-Zug-Knoten $(c_{\text{tripid}}, c_{\text{deptime}})$
- Zeitgleiche Stehe-Am-Stop-Knoten werden zusammengefasst

Zeit-Expandierter Graph

- Für jede Connection erstellen wir drei Knoten:
  - Die Stehe-Am-Stop-Knoten $(c_{\text{depstop}}, c_{\text{deptime}})$ und $(c_{\text{arrstop}}, c_{\text{arrtime}} + \text{changetime}(c_{\text{arrstop}}))$
  - Den Sitze-Im-Zug-Knoten $(c_{\text{tripid}}, c_{\text{deptime}})$
- Zeitgleiche Stehe-Am-Stop-Knoten werden zusammengefasst
- Jeder (nicht zusammengefasste) Knoten hat bis zu zwei ausgehende Kanten
- Für Stehe-Am-Stop-Knoten:
  - **Einstiegs-Kante**: von $(c_{\text{depstop}}, c_{\text{deptime}})$ nach $(c_{\text{tripid}}, c_{\text{deptime}})$
  - **Warte-Kante**: von (s, τ_1) nach (s, τ_2) wobei s ein Stop ist und τ_1 und τ_2 zwei aufeinander folgende Zeitpunkte
- Für Sitze-Im-Zug-Knoten:
  - **Ausstiegs-Kante**: von $(c_{\text{tripid}}, c_{\text{deptime}})$ nach $(c_{\text{arrstop}}, c_{\text{arrtime}} + \text{changetime}(c_{\text{arrstop}}))$ ist
  - **Sitzen-Bleiben-Kante**: von (t, τ_1) nach (t, τ_2) wobei t ein Trip ist und τ_1 und τ_2 zwei aufeinander folgende Zeitpunkte

Zeit-Expandierter Graph

c_{depstop}	c_{arrstop}	c_{deptime}	c_{arrtime}	c_{tripid}
A	B	8	9	α
B	C	9	10	α
B	D	9	10	β
B	D	10	11	γ

stop s	changetime(s)
A	0.5
B	0.3
C	0.2
D	0.1

Zeit-Expandierter Graph

c_{depstop}	c_{arrstop}	c_{deptime}	c_{arrtime}	c_{tripid}
A	B	8	9	α
B	C	9	10	α
B	D	9	10	β
B	D	10	11	γ

stop s	changetime(s)
A	0.5
B	0.3
C	0.2
D	0.1

$(C, 10.2)$ $(\alpha, 9)$ $(B, 9)$ $(\beta, 9)$

$(\alpha, 8)$ $(B, 9.3)$ $(D, 10.1)$ $(D, 11.1)$

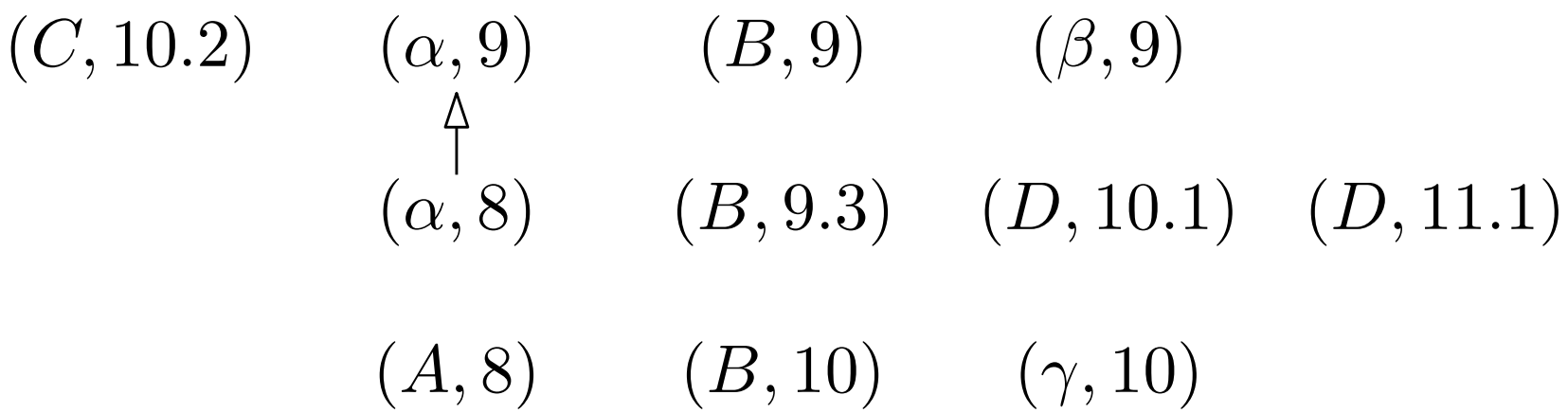
$(A, 8)$ $(B, 10)$ $(\gamma, 10)$

Alle Knoten; $(B, 9)$ wurde zusammengefasst

Zeit-Expandierter Graph

c_{depstop}	c_{arrstop}	c_{deptime}	c_{arrtime}	c_{tripid}
A	B	8	9	α
B	C	9	10	α
B	D	9	10	β
B	D	10	11	γ

stop s	changetime(s)
A	0.5
B	0.3
C	0.2
D	0.1

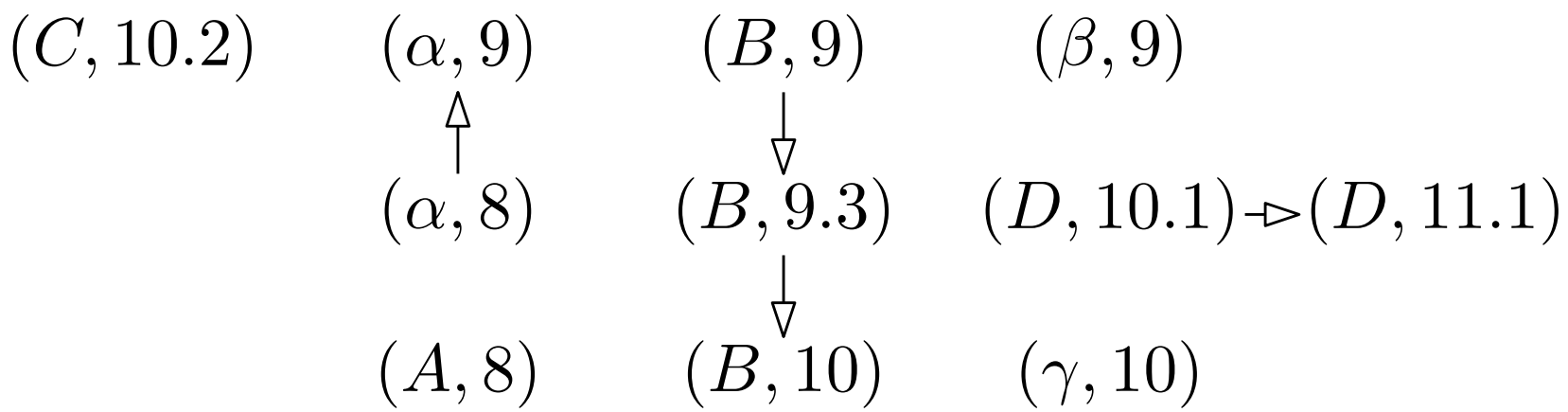


Sitzen-Bleiben-Kanten

Zeit-Expandierter Graph

c_{depstop}	c_{arrstop}	c_{deptime}	c_{arrtime}	c_{tripid}
A	B	8	9	α
B	C	9	10	α
B	D	9	10	β
B	D	10	11	γ

stop s	changetime(s)
A	0.5
B	0.3
C	0.2
D	0.1



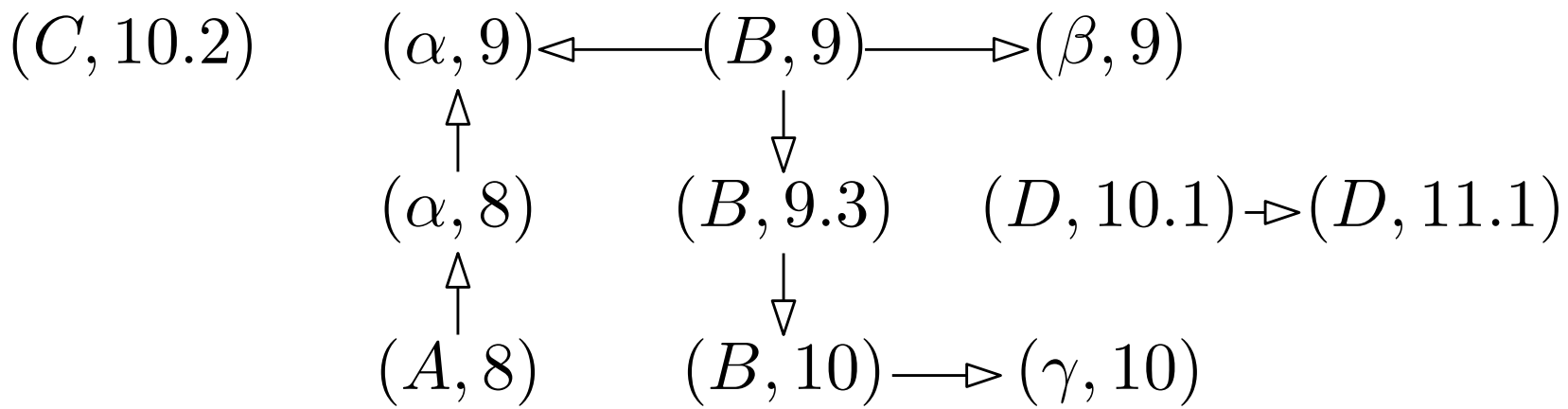
Warte-Kante

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Zeit-Expandierter Graph

c_{depstop}	c_{arrstop}	c_{deptime}	c_{arrtime}	c_{tripid}
A	B	8	9	α
B	C	9	10	α
B	D	9	10	β
B	D	10	11	γ

stop s	changetime(s)
A	0.5
B	0.3
C	0.2
D	0.1

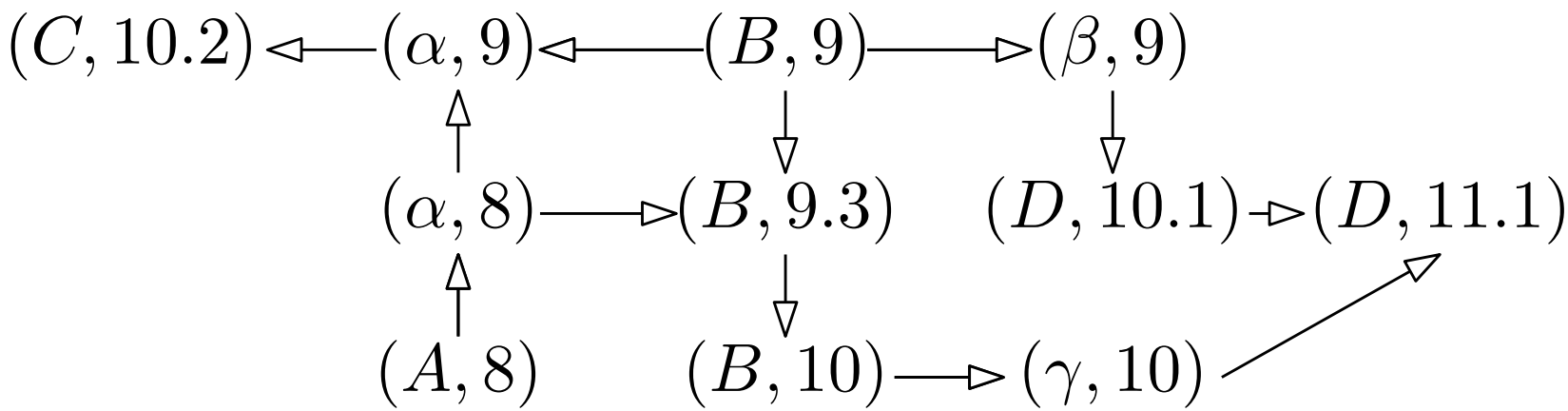


Einstiegs-Kanten

Zeit-Expandierter Graph

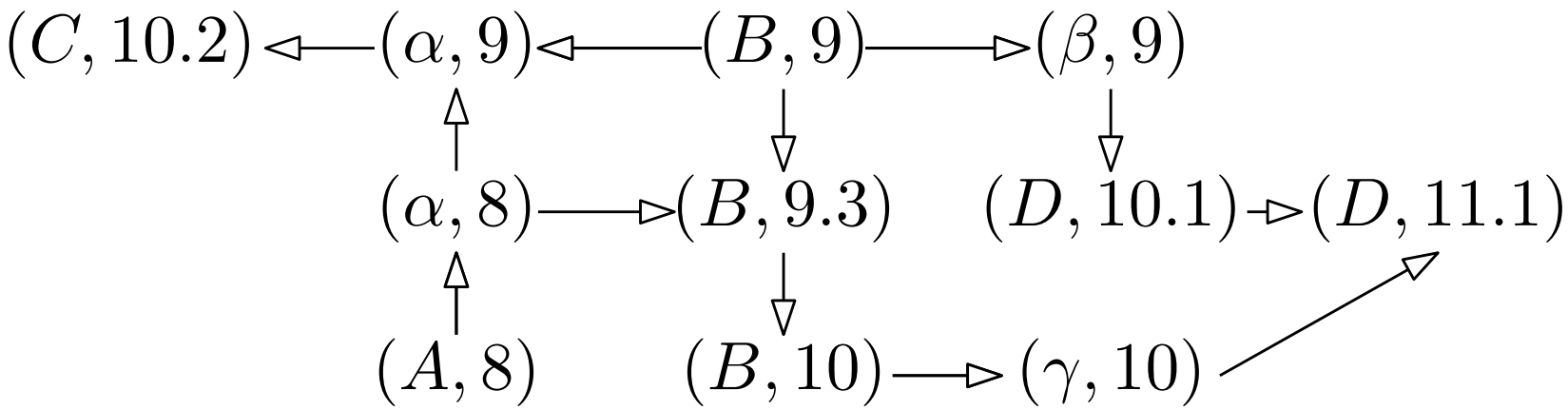
c_{depstop}	c_{arrstop}	c_{deptime}	c_{arrtime}	c_{tripid}
A	B	8	9	α
B	C	9	10	α
B	D	9	10	β
B	D	10	11	γ

stop s	changetime(s)
A	0.5
B	0.3
C	0.2
D	0.1



Ausstiegs-Kanten

Zeit-Expandierter Graph



- Azyklischer Graph, da in der Zeit vorwärts gerichtet
- Kanten werden mit Zeitdifferenz gewichtet
- Grad-1-Knoten können wegkontrahiert werden
  - Es gibt eine Konstruktion mit 2 Knoten pro Connection
  - **Aber**: Viele kleine Sonderfälle in Algorithmen
  - Machen wir nicht in der Vorlesung

Zeit-Expandierter Graph

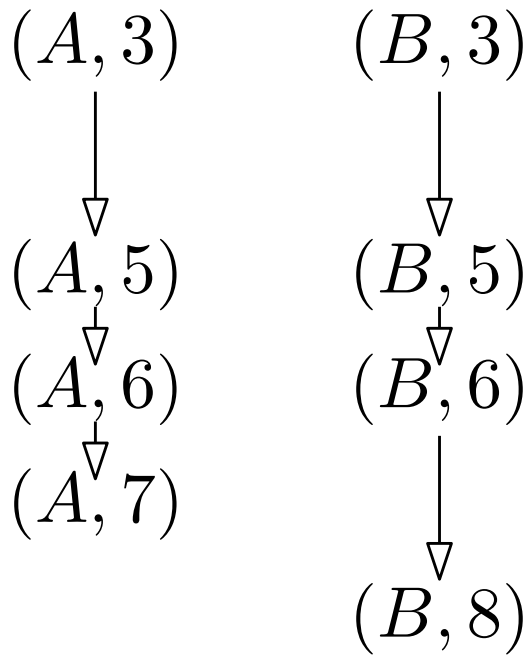
Expandierte Fußwege

- Für einen Fußweg von A nach B der Länge ℓ müssen mehrere Kanten eingefügt werden
- Kante zwischen (A, x) und (B, y) wenn $y - x \geq \ell$ und
  - Es keinen Knoten (A, p) gibt mit $y - p \geq \ell$ und $p > x$ gibt
  - Es keinen Knoten (B, q) gibt mit $q - x \geq \ell$ und $q < y$ gibt
- Beispiel mit $\ell = 1$

Zeit-Expandierter Graph

Expandierte Fußwege

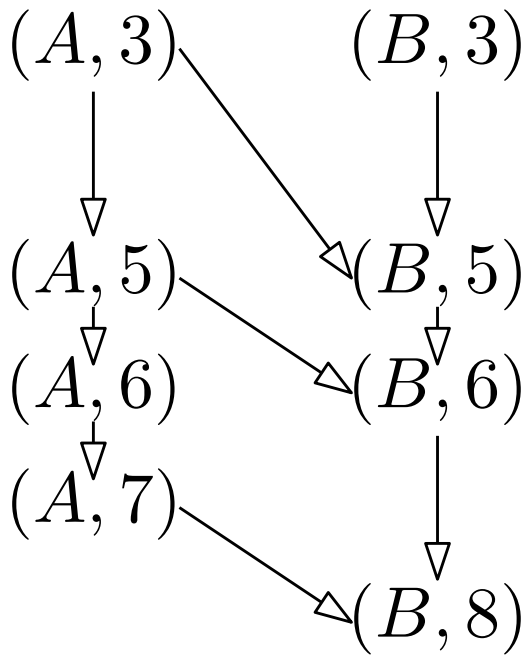
- Für einen Fußweg von A nach B der Länge ℓ müssen mehrere Kanten eingefügt werden
- Kante zwischen (A, x) und (B, y) wenn $y - x \geq \ell$ und
  - Es keinen Knoten (A, p) gibt mit $y - p \geq \ell$ und $p > x$ gibt
  - Es keinen Knoten (B, q) gibt mit $q - x \geq \ell$ und $q < y$ gibt
- Beispiel mit $\ell = 1$



Zeit-Expandierter Graph

Expandierte Fußwege

- Für einen Fußweg von A nach B der Länge ℓ müssen mehrere Kanten eingefügt werden
- Kante zwischen (A, x) und (B, y) wenn $y - x \geq \ell$ und
  - Es keinen Knoten (A, p) gibt mit $y - p \geq \ell$ und $p > x$ gibt
  - Es keinen Knoten (B, q) gibt mit $q - x \geq \ell$ und $q < y$ gibt
- Beispiel mit $\ell = 1$



Zeit-Expandierter Graph

Fußweg-Explosion

- Bei m Fußwege und c Connections können wir $\Theta(cm)$ Kanten kriegen
- $\Rightarrow$ Fußwege in der Modellierung sparsam verwenden

Initiale & Finale Fußwege

- Expandierte Fußwege decken nur Fußwege zwischen zwei Legs ab
- Initiale und finale Fußwege sind Sonderfälle in Algorithmen

Dijkstras Algorithmus im Zeit-Expandierten Graph

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Problem der frühesten Ankunft

Eingabe:

- Startstop s
- Startzeit τ_s
- Zielstop t

Ausgabe:

- Ankunftszeit τ_t

Problem der frühesten Ankunft

Idee:

- Dijkstras Algorithmus auf zeit-expandiertem Graph
- Kantengewichte ist die Zeitdifferenz
- Erster Knoten von s nach τ_s ist erster Knoten in Queue
- Erster Stehe-Am-Stop-Knoten von t der aus der Queue genommen wird hat Ankunftszeit

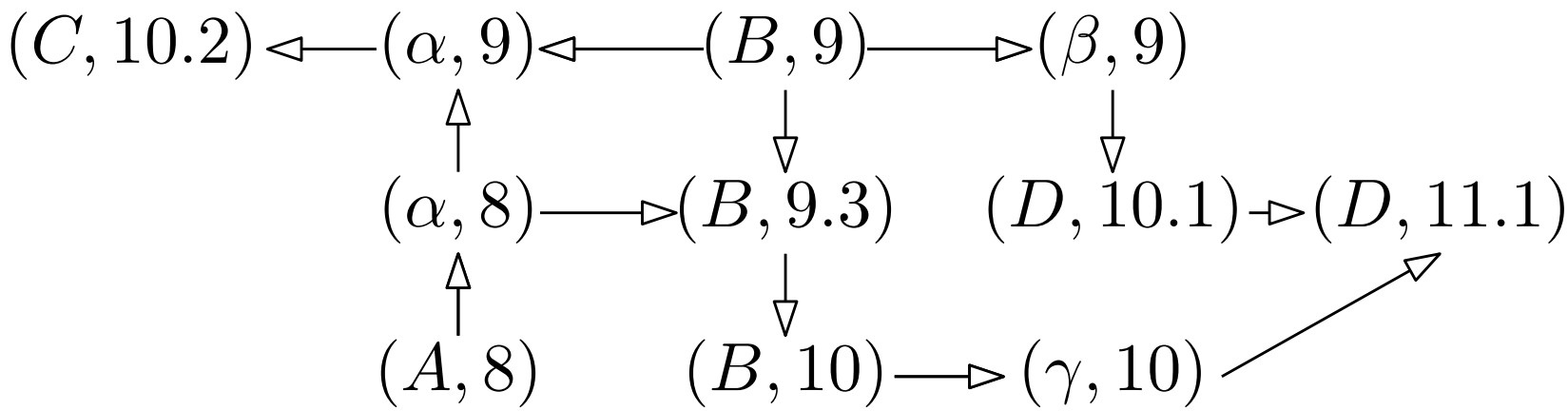
Problem der frühesten Ankunft

Idee:

- Dijkstras Algorithmus auf zeit-expandiertem Graph
- Kantengewichte ist die Zeitdifferenz
- Erster Knoten von s nach τ_s ist erster Knoten in Queue
- Erster Stehe-Am-Stop-Knoten von t der aus der Queue genommen wird hat Ankunftszeit

Beispiel:

- mit $s = A$, $t = C$, $\tau_s = 8$



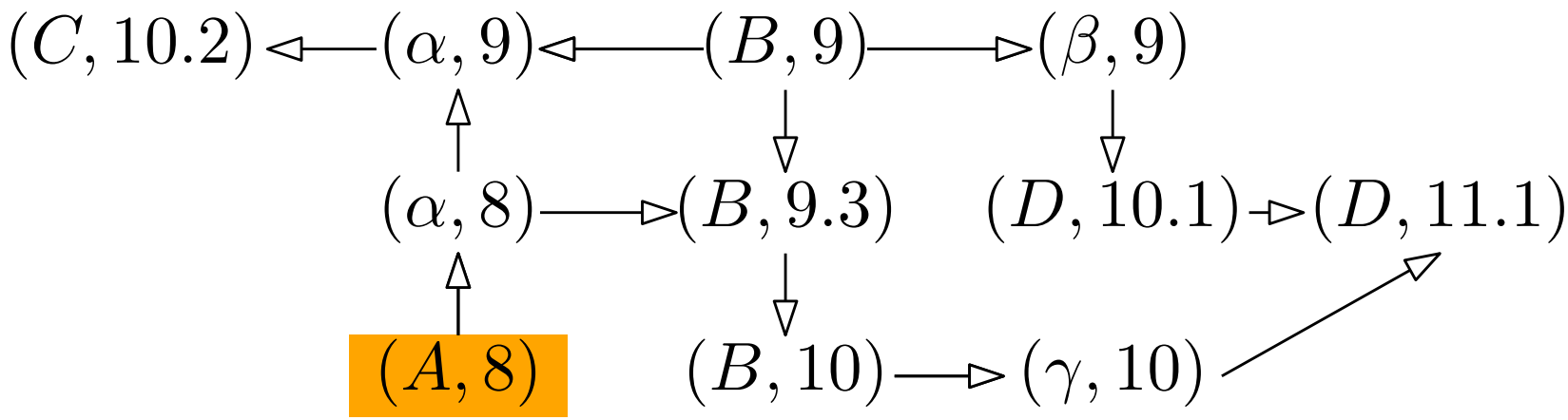
Problem der frühesten Ankunft

Idee:

- Dijkstras Algorithmus auf zeit-expandiertem Graph
- Kantengewichte ist die Zeitdifferenz
- Erster Knoten von s nach τ_s ist erster Knoten in Queue
- Erster Stehe-Am-Stop-Knoten von t der aus der Queue genommen wird hat Ankunftszeit

Beispiel:

- mit $s = A$, $t = C$, $\tau_s = 8$



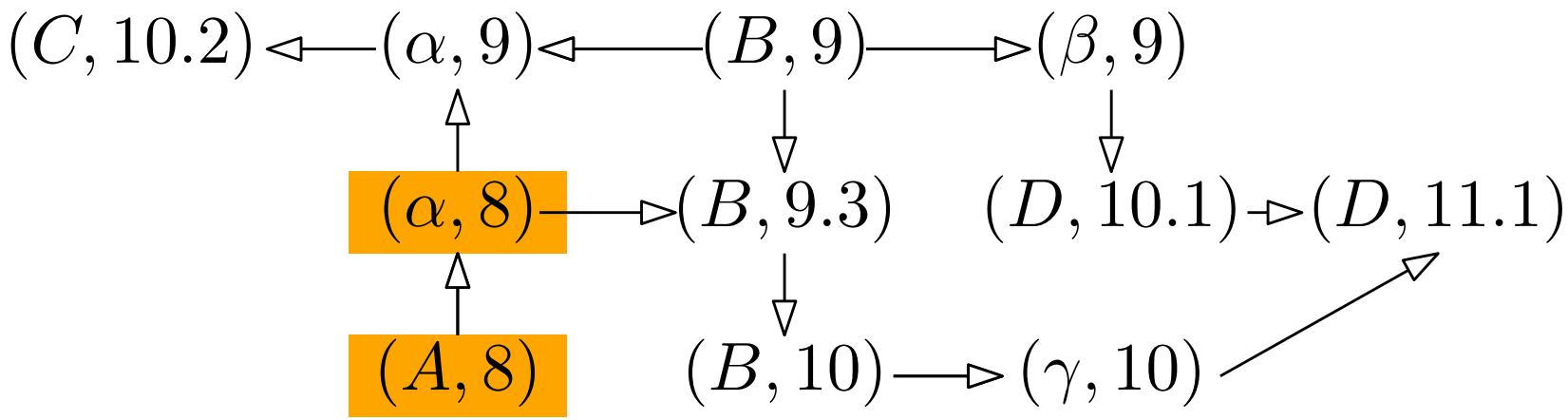
Problem der frühesten Ankunft

Idee:

- Dijkstras Algorithmus auf zeit-expandiertem Graph
- Kantengewichte ist die Zeitdifferenz
- Erster Knoten von s nach τ_s ist erster Knoten in Queue
- Erster Stehe-Am-Stop-Knoten von t der aus der Queue genommen wird hat Ankunftszeit

Beispiel:

- mit $s = A$, $t = C$, $\tau_s = 8$



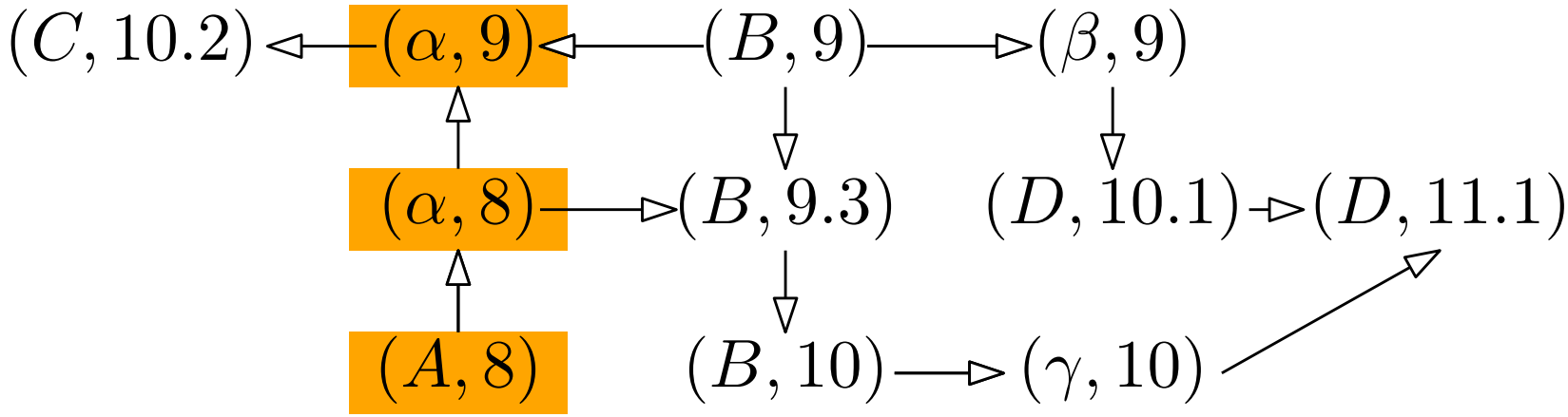
Problem der frühesten Ankunft

Idee:

- Dijkstras Algorithmus auf zeit-expandiertem Graph
- Kantengewichte ist die Zeitdifferenz
- Erster Knoten von s nach τ_s ist erster Knoten in Queue
- Erster Stehe-Am-Stop-Knoten von t der aus der Queue genommen wird hat Ankunftszeit

Beispiel:

- mit $s = A$, $t = C$, $\tau_s = 8$



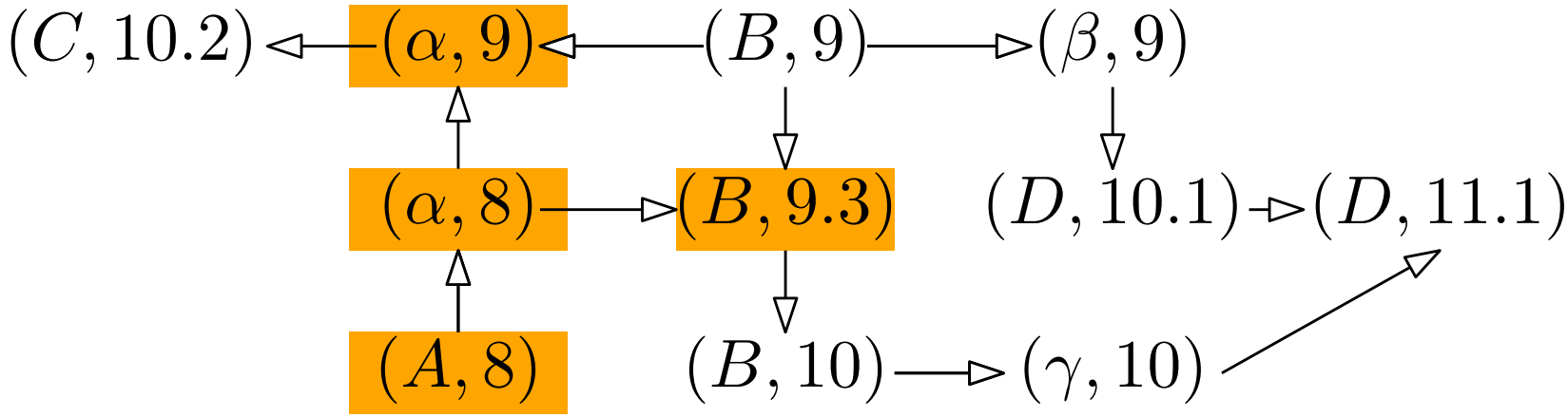
Problem der frühesten Ankunft

Idee:

- Dijkstras Algorithmus auf zeit-expandiertem Graph
- Kantengewichte ist die Zeitdifferenz
- Erster Knoten von s nach τ_s ist erster Knoten in Queue
- Erster Stehe-Am-Stop-Knoten von t der aus der Queue genommen wird hat Ankunftszeit

Beispiel:

- mit $s = A$, $t = C$, $\tau_s = 8$



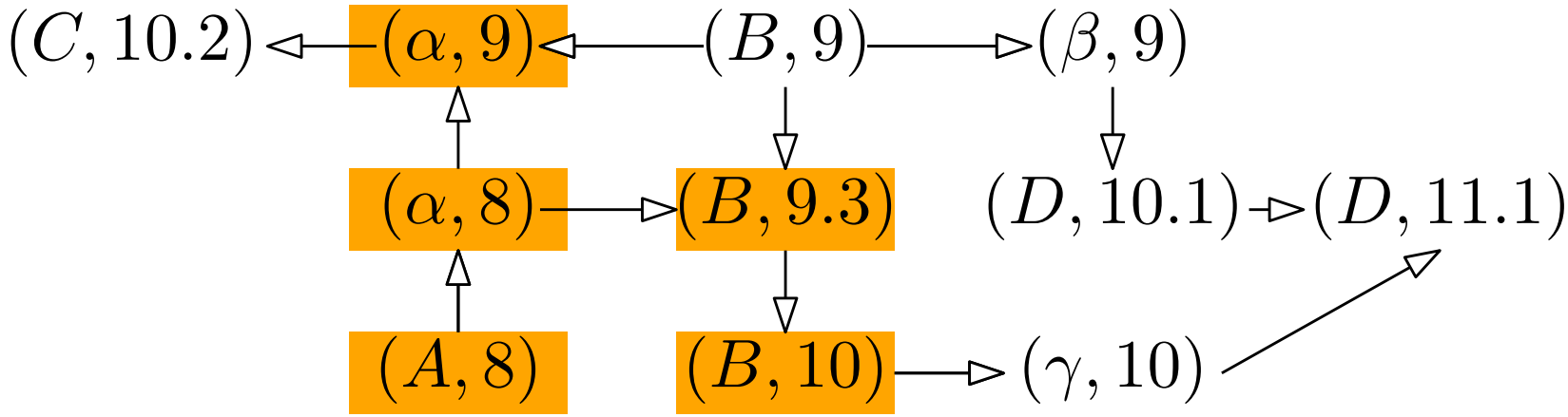
Problem der frühesten Ankunft

Idee:

- Dijkstras Algorithmus auf zeit-expandiertem Graph
- Kantengewichte ist die Zeitdifferenz
- Erster Knoten von s nach τ_s ist erster Knoten in Queue
- Erster Stehe-Am-Stop-Knoten von t der aus der Queue genommen wird hat Ankunftszeit

Beispiel:

- mit $s = A$, $t = C$, $\tau_s = 8$



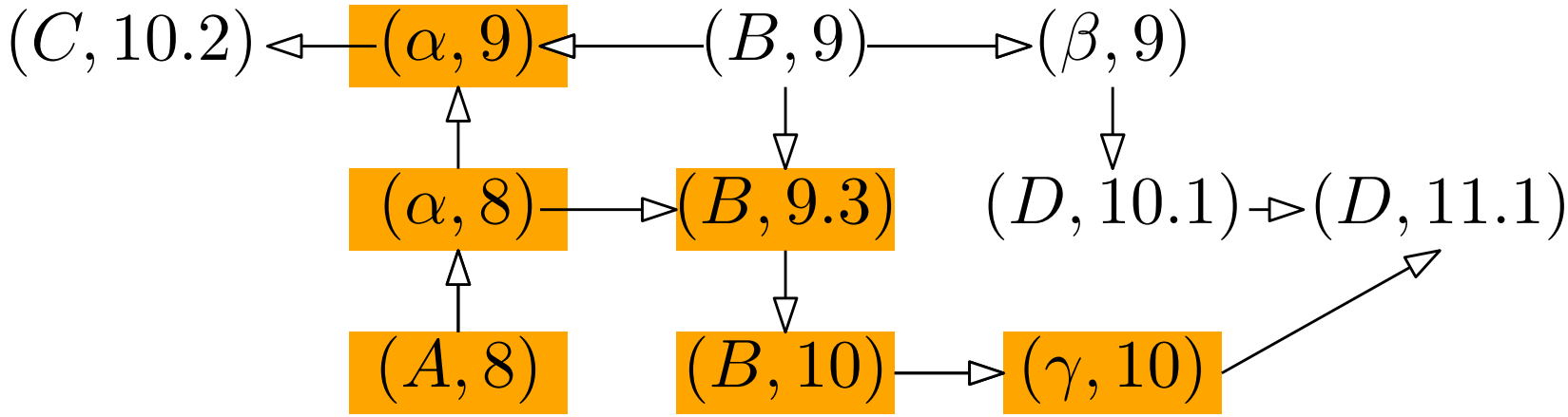
Problem der frühesten Ankunft

Idee:

- Dijkstras Algorithmus auf zeit-expandiertem Graph
- Kantengewichte ist die Zeitdifferenz
- Erster Knoten von s nach τ_s ist erster Knoten in Queue
- Erster Stehe-Am-Stop-Knoten von t der aus der Queue genommen wird hat Ankunftszeit

Beispiel:

- mit $s = A$, $t = C$, $\tau_s = 8$



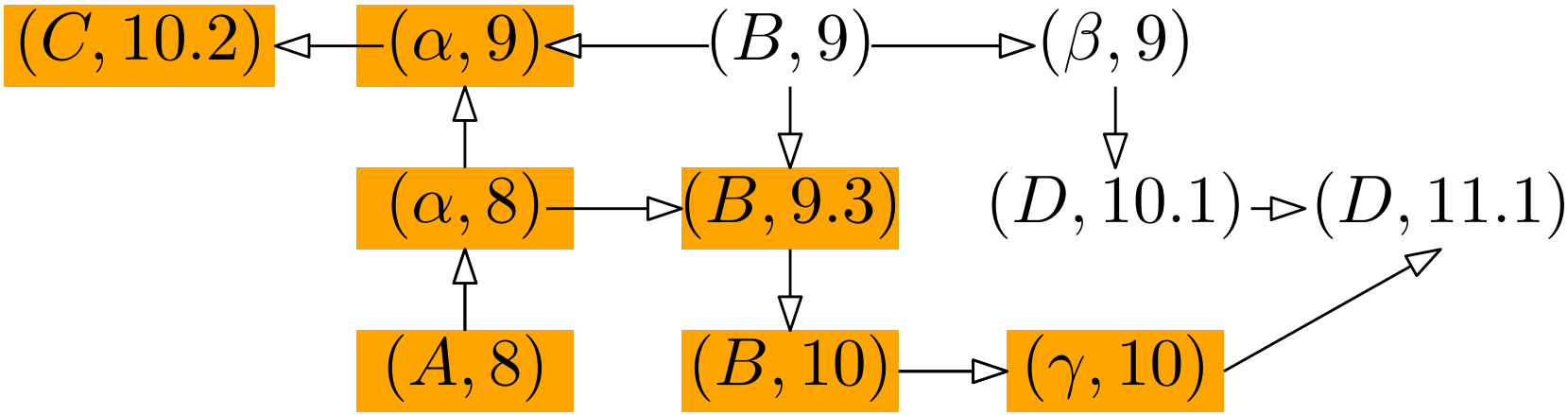
Problem der frühesten Ankunft

Idee:

- Dijkstras Algorithmus auf zeit-expandiertem Graph
- Kantengewichte ist die Zeitdifferenz
- Erster Knoten von s nach τ_s ist erster Knoten in Queue
- Erster Stehe-Am-Stop-Knoten von t der aus der Queue genommen wird hat Ankunftszeit

Beispiel:

- mit $s = A$, $t = C$, $\tau_s = 8$



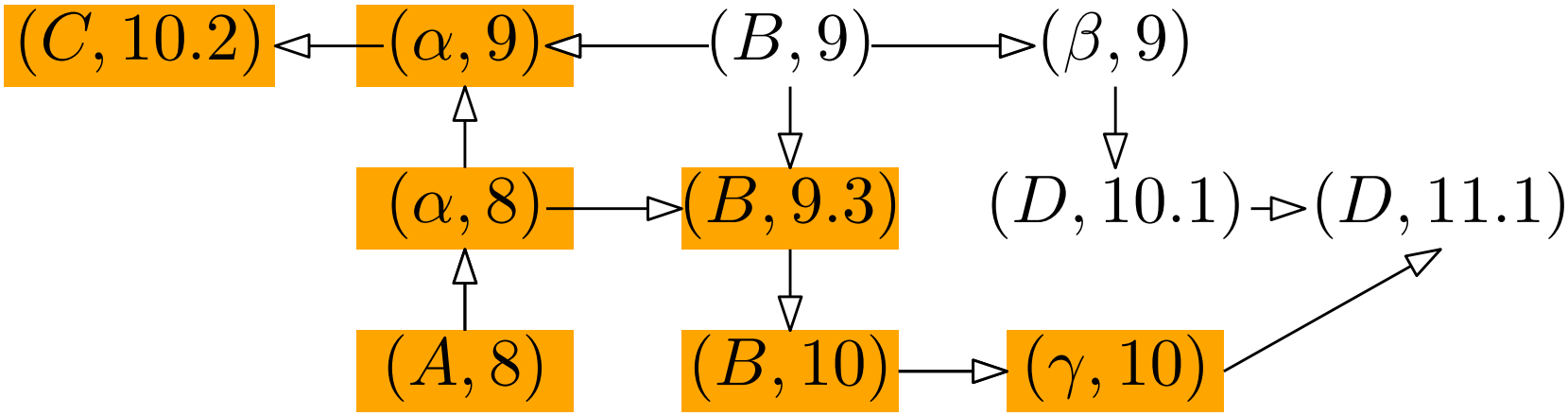
Problem der frühesten Ankunft

Idee:

- Dijkstras Algorithmus auf zeit-expandiertem Graph
- Kantengewichte ist die Zeitdifferenz
- Erster Knoten von s nach τ_s ist erster Knoten in Queue
- Erster Stehe-Am-Stop-Knoten von t der aus der Queue genommen wird hat Ankunftszeit

Beispiel:

- mit $s = A$, $t = C$, $\tau_s = 8$



- $\tau_t = 10.2$

Initiale Fußwege

Problem:

- Initiale Fußwege werden nicht immer beachtet

Initiale Fußwege

Problem:

- Initiale Fußwege werden nicht immer beachtet

Angepasster Algorithmus:

- Finde ersten Knoten von s nach τ_s und füge ihn mit Gewicht 0 in die Queue
- Für jeden ausgehenden Fußweg (s, x) mit Länge ℓ:
  - Finde ersten Knoten von x nach $\tau_s + \ell$ füge ihn mit Gewicht ℓ in die Queue

Finale Fußwege

Problem:

- Finale Fußwege werden nicht immer beachtet

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Finale Fußwege

Problem:

- Finale Fußwege werden nicht immer beachtet

Angepasster Algorithmus:

- **Idee:** global Array D, $D[x]$ ist die Fußwegdistanz von x zu t
- Fülle D bei Programmstart mit ∞

Finale Fußwege

Problem:

- Finale Fußwege werden nicht immer beachtet

Angepasster Algorithmus:

- **Idee:** global Array D, $D[x]$ ist die Fußwegdistanz von x zu t
- Fülle D bei Programmstart mit ∞
- Vor Dijkstras Algorithmus:
  - Iteriere über alle eingehenden Kanten von t und setze D
  - und setze $D[t] = 0$

Finale Fußwege

Problem:

- Finale Fußwege werden nicht immer beachtet

Angepasster Algorithmus:

- **Idee:** global Array D, $D[x]$ ist die Fußwegdistanz von x zu t
- Fülle D bei Programmstart mit ∞
- Vor Dijkstras Algorithmus:
  - Iteriere über alle eingehenden Kanten von t und setze D
  - und setze $D[t] = 0$
- Nach Dijkstras Algorithmus:
  - Iteriere über alle eingehenden Kanten von t und setze D auf ∞ zurück
  - und setze $D[t] = \infty$

Finale Fußwege

Problem:

- Finale Fußwege werden nicht immer beachtet

Angepasster Algorithmus:

- **Idee:** global Array D, $D[x]$ ist die Fußwegdistanz von x zu t
- Fülle D bei Programmstart mit ∞
- Vor Dijkstras Algorithmus:
  - Iteriere über alle eingehenden Kanten von t und setze D
  - und setze $D[t] = 0$
- Nach Dijkstras Algorithmus:
  - Iteriere über alle eingehenden Kanten von t und setze D auf ∞ zurück
  - und setze $D[t] = \infty$
- Während Dijkstras Algorithmus:
  - Wenn Stehe-Am-Stop-Knoten (x, τ) aus der Queue genommen wird:
  - Wenn $\tau + D[x] \neq \infty$ dann wurde ein Weg gefunden

Mehrere Kriterien

- Um mehrere Kriterien im Pareto-Sinn zu optimieren kann man eine Erweiterung von Dijkstras Algorithmus einsetzen

Grobe Wiederholung

- Jeder Knoten hat eine Menge genannt Bag von Label
- Bags enthalten nur nicht dominierte Label
- Die Queue verwaltet Label

Späteste Abfahrtszeit

Ziel:

- Wir wollen unter allen Journeys die so früh wie möglich ankommen eine wählen die so spät wie möglich abfährt

Späteste Abfahrtszeit

Ziel:

- Wir wollen unter allen Journeys die so früh wie möglich ankommen eine wählen die so spät wie möglich abfährt

Option 1:

- Zwei Anfragen:
  - Erste bestimmt die früheste Ankunftszeit
  - Zweite arbeitet auf Rückwärtsgraph und bestimmt späteste Abfahrtszeit

Späteste Abfahrtszeit

Ziel:

- Wir wollen unter allen Journeys die so früh wie möglich ankommen eine wählen die so spät wie möglich abfährt

Option 1:

- Zwei Anfragen:
  - Erste bestimmt die früheste Ankunftszeit
  - Zweite arbeitet auf Rückwärtsgraph und bestimmt späteste Abfahrtszeit

Option 2:

- Mach eine Profilanfrage

Pfadextraktion

Ziel:

- Finde Journey und nicht nur Ankunftszeit

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Pfadextraktion

Ziel:

- Finde Journey und nicht nur Ankunftszeit

Option 1:

- Speichere Vorgängerzeiger an jedem Knoten
- Ergibt nur ein Pfad

Pfadextraktion

Ziel:

- Finde Journey und nicht nur Ankunftszeit

Option 1:

- Speichere Vorgängerzeiger an jedem Knoten
- Ergibt nur ein Pfad

Option 2:

- Bestimme Vorgängerknoten über Distanzen
- Sei $d(x)$ die Distanz an Knoten x
- Jeder Knoten y mit Kante (y, x) und $d(y) + \ell(y, x) = d(x)$ ist gültiger Knoten
- Iteriere über eingehende Kanten um alle oder nur ein y zu bestimmen
- Kann genutzt werde um ein Pfad zu finden
- Kann auch alle Pfade finden
- **Achtung**: Es kann sehr viele optimale Pfade geben

Profile

Eingabe:

- Startstop s
- minimale Abfahrtszeit τ_{min}
- Zielstop t
- maximale Ankunftszeit τ_{max}

Ausgabe:

- Menge an optimalen Journeys die
  - von s nach t fahren und
  - nach τ_{min} abfahren und
  - vor τ_{max} ankommen

Profile

Einfache algorithmische Anpassung:

- Füge alle Knoten (s, τ) für jedes $\tau_{\text{min}} \leq \tau \leq \tau_{\text{max}}$ mit Gewicht 0 in die Queue
- Prune die Suche an allen Knoten die später als τ_{max} sind
- Extrahiere den Pfad für jeden Knoten (t, τ) für jedes $\tau_{\text{min}} \leq \tau \leq \tau_{\text{max}}$

Profile

Einfache algorithmische Anpassung:

- Füge alle Knoten (s, τ) für jedes $\tau_{\text{min}} \leq \tau \leq \tau_{\text{max}}$ mit Gewicht 0 in die Queue
- Prune die Suche an allen Knoten die später als τ_{max} sind
- Extrahiere den Pfad für jeden Knoten (t, τ) für jedes $\tau_{\text{min}} \leq \tau \leq \tau_{\text{max}}$

Kombinationen:

- Bei mehreren Kriterien: Führe Pfadextraktion für jedes Label durch
- Bei initialen und finalen Fußwegen: Anpassungen analog

Literatur I

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.