max planck institut
informatik

Kürzeste und Schnellste Wege

Wie funktionieren Navis?

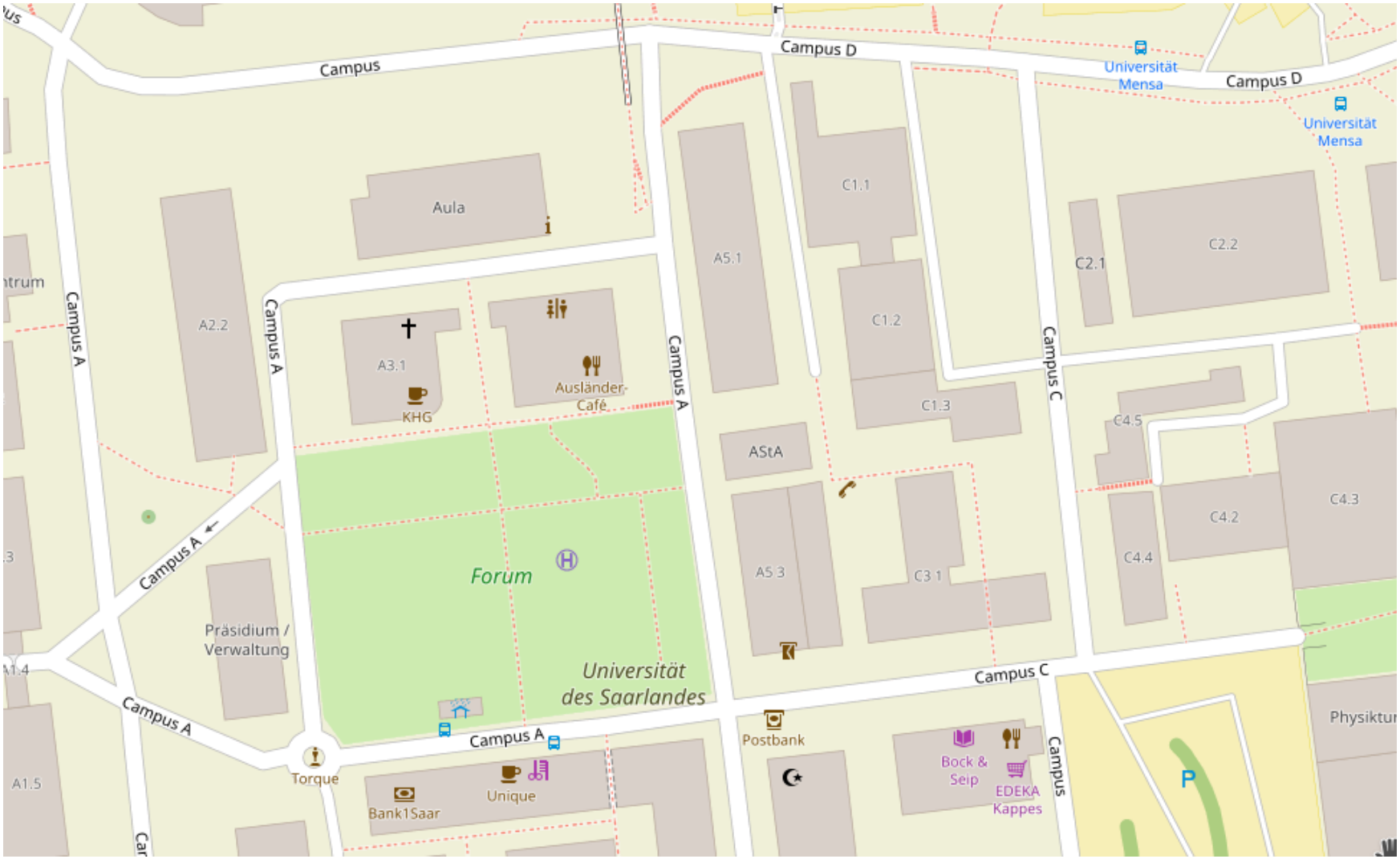
André Nusser
(Folien inspiriert von Kurt Mehlhorn)

20.11.2017

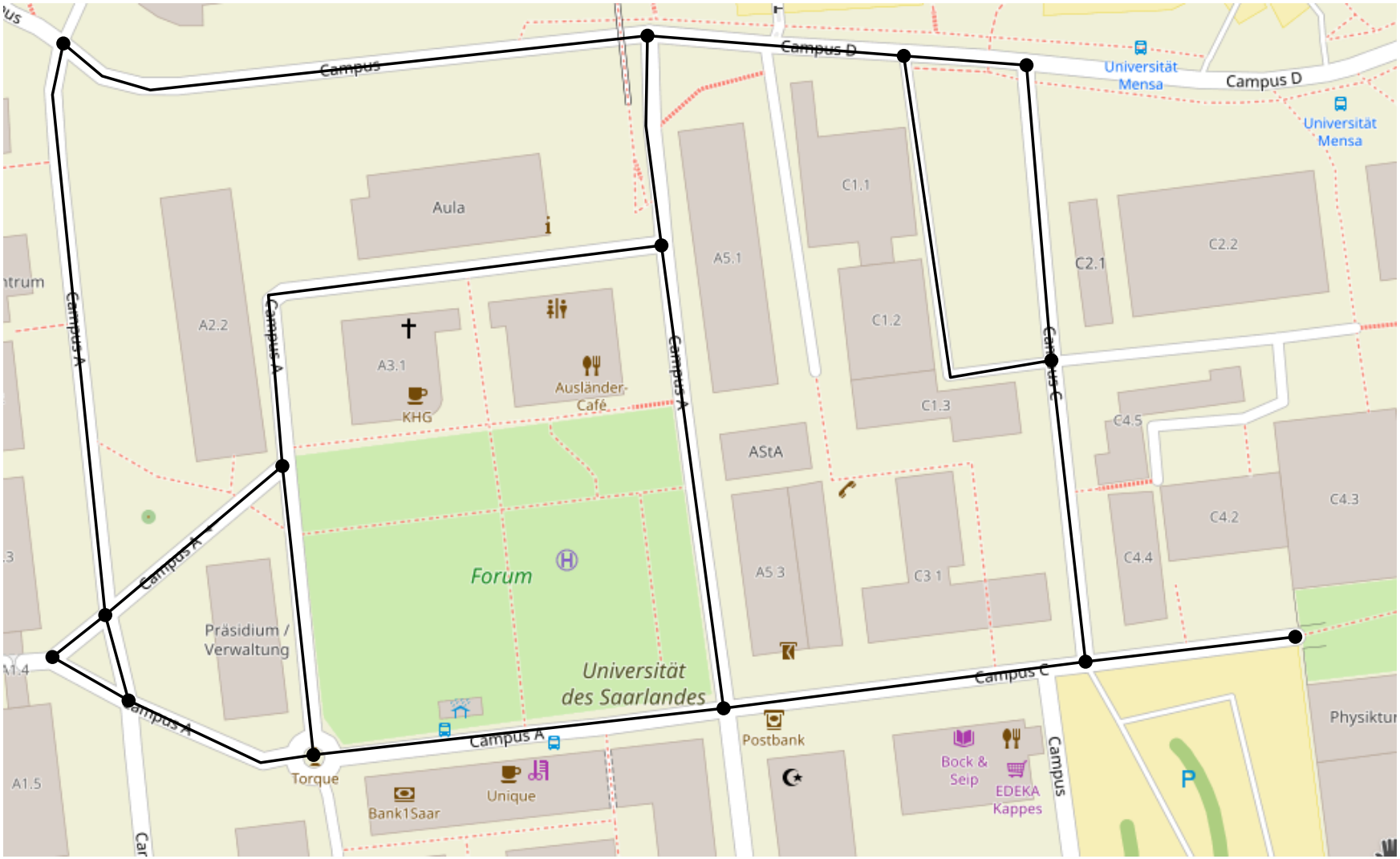
Struktur

Straßennetzwerke

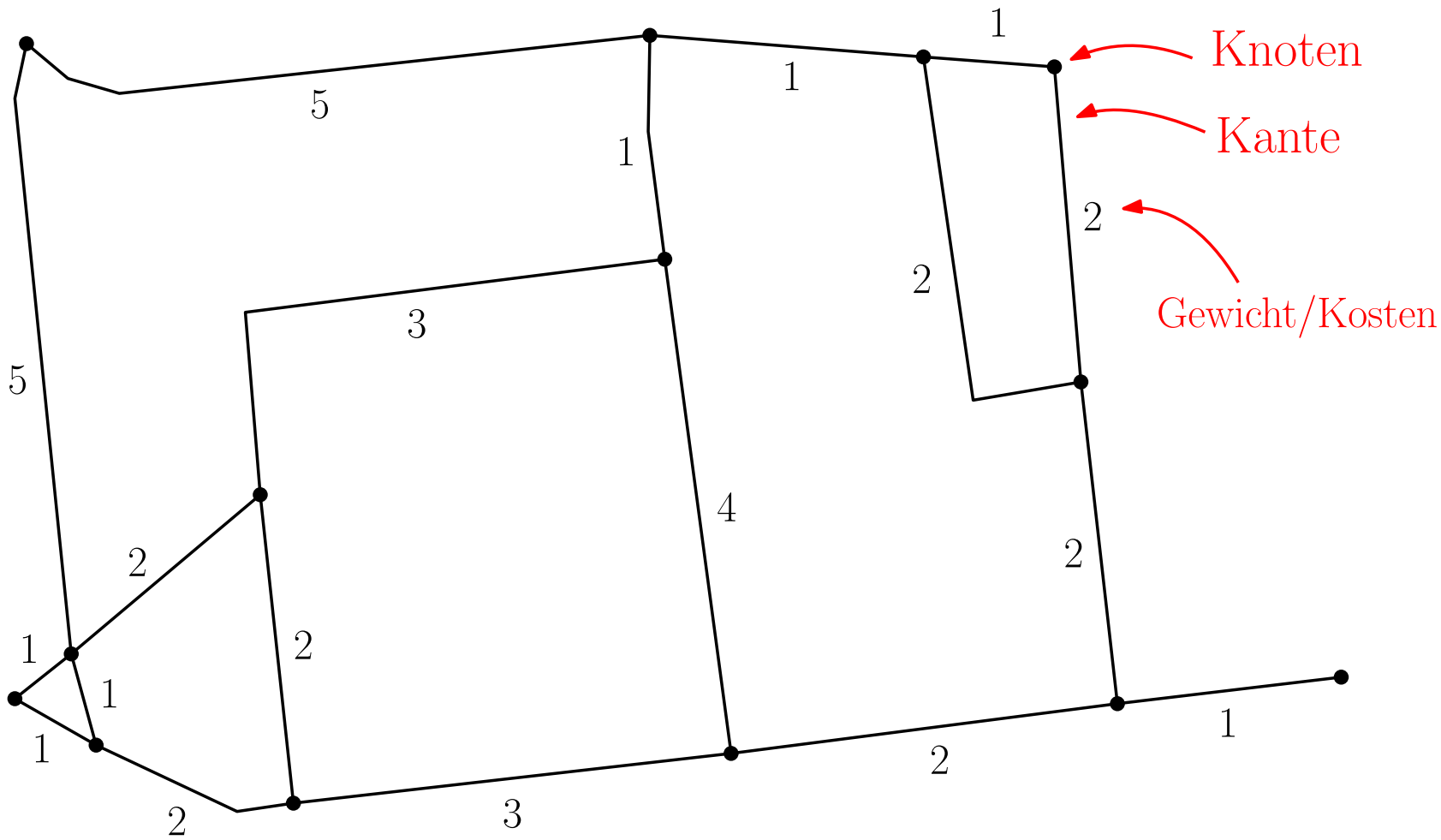
Landkarten


Campus
Campus D
Universität Mensa
Campus D
Universität Mensa
Aula
A2.2
Campus A
Campus A
A3.1
KHG
Ausländer-Café
A5.1
C1.1
C1.2
C1.3
C2.1
C2.2
Campus C
C4.5
C4.2
C4.3
C4.4
AStA
A5 3
C3 1
Forum
Präsidium / Verwaltung
Universität des Saarlandes
Campus A
Campus A
Campus A
Campus C
Campus C
Torque
Bank1Saar
Unique
Postbank
Bock & Seip
EDEKA Kappes
P
A1.5
Physiktur

Landkarten


Campus
Campus D
Universität Mensa
Campus D
Universität Mensa
Aula
A2.2
A3.1
KHG
Ausländer-Café
A5.1
C1.1
C2.1
C2.2
C1.2
C1.3
C4.5
C4.3
C4.2
C4.4
AStA
A5.3
C3.1
Campus A
Campus A
Campus A
Campus A
Campus A
Campus C
Campus C
Campus
Forum
Präsidium / Verwaltung
Universität des Saarlandes
Torque
Bank1Saar
Unique
Postbank
Bock & Seip
EDEKA Kappes
P
A1.5
A1.4
Physiktur

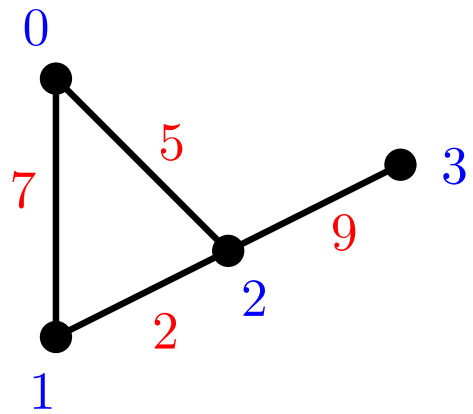
Graphen



Graphen im Speicher

Knoten:

Knotennummer:	0	1	2	3
Erste Kante:	0	2	4	7
Letzte Kante + 1:	2	4	7	8



Kanten:

Kantennummer:	0	1	2	3	4	5	6	7
Startknoten:	0	0	1	1	2	2	2	3
Zielknoten:	1	2	0	2	0	1	3	2
Distanz:	7	5	7	2	5	2	9	9

Ein paar Daten

- USA: 24 Millionen Knoten, 58 Millionen Kanten
- Fahrtzeiten durch Erfahrungswerte/Höchstgeschwindigkeit
- Fahrtzeiten ändern sich durch Staus/Bauarbeiten

Grundlegendes

- "Kürzeste Wege" äquivalent zu "Schnellste Wege"
- Kürzeste Wege und kürzeste Distanzen grob gleich schwer
- Praxis ist komplizierter als Theorie
  - Schnellste Route $\neq$ Beste Route
  - Staus
  - Abbiege- und Fahrbeschränkungen
  - ...

Geschichte

Algorithmus	Jahr	Mikrosekunden (10^{-6}s)
Naiv	–	"sehr langsam"
Dijkstra	1959	2000000
Reach	2004	1000
Transitknoten	2007	2
Hub Labels	2011	0.2

Zeiten für zufälliges Start und Ziel auf Straßennetzwerk der USA.

Naiver Algorithmus

Grundidee

Teilstücke von kürzesten Wegen sind auch kürzeste Wege!

Algorithmus

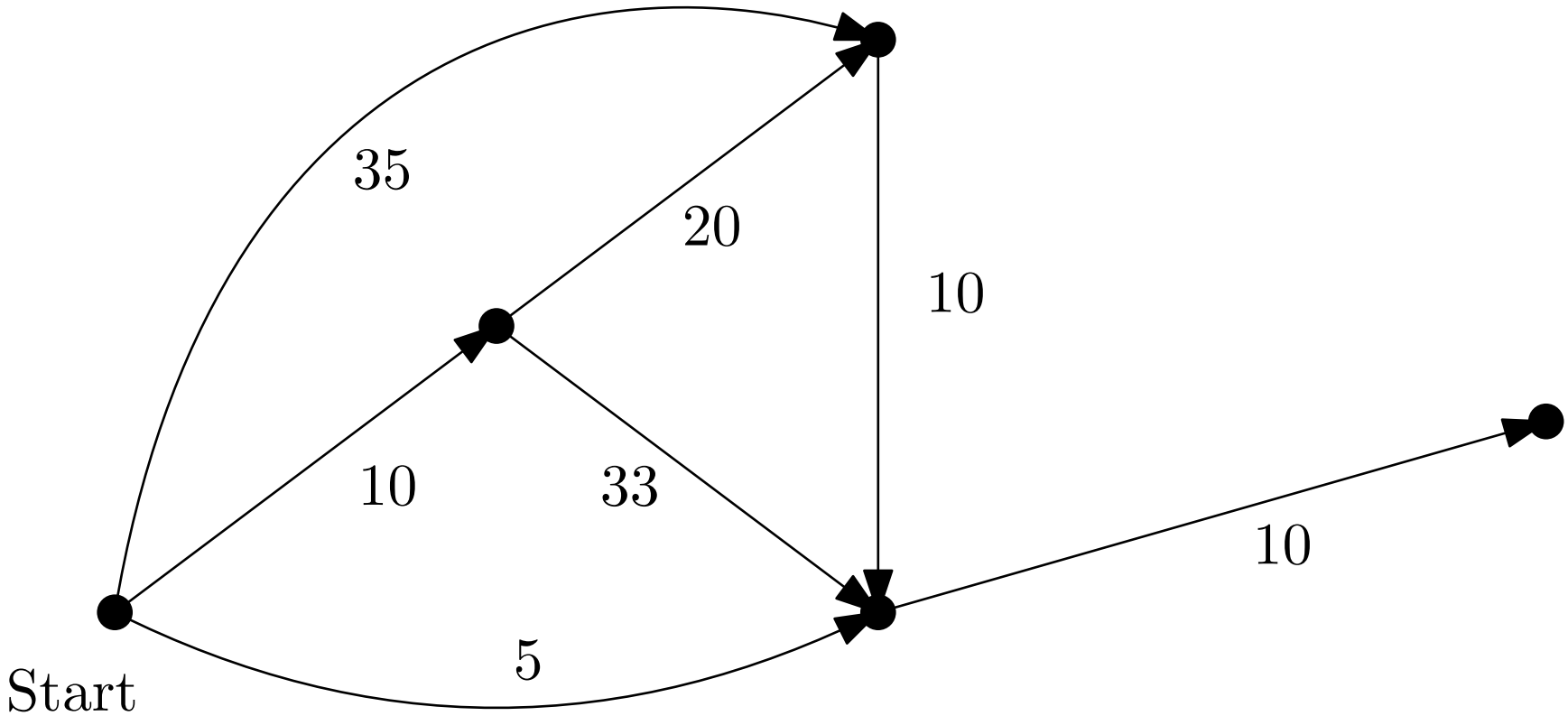
Gegeben: Ein Straßengraph und Startknoten s

Gesucht: Kürzester Weg von s zu allen Knoten

Algorithmus

1. Fahrtzeit von s zu s ist 0 und zu anderen Knoten ∞
2. Finde Kante (a, b) sodass:
   Fahrtzeit von s nach $b >$
   Fahrtzeit von s nach a + Länge der Straße/Kante von a nach b
3. Fahrtzeit von s nach $b =$
   Fahrtzeit von s nach a + Länge der Straße/Kante von a nach b
4. Gehe zu 2. falls noch eine Fahrtzeit verbessert werden kann

Beispiel



→ Ausführung siehe Tafel

Pseudocode

- $dist[v]$ = beste bisher gefundene Distanz von s nach v
- $zeit(u, v)$ = Fahrtzeit über Straße/Kante von u nach v

Algorithmus: Naiver Algorithmus von s

1: $dist[s] \leftarrow 0$
2: $dist[v] \leftarrow \infty$ für alle $v \neq s$
3: **while** gibt Kante (u, v) mit $dist[u] + zeit(u, v) < dist[v]$ **do**
4: $\quad dist[v] \leftarrow dist[u] + zeit(u, v)$

Pseudocode

- $dist[v] =$ beste bisher gefundene Distanz von s nach v
- $zeit(u, v) =$ Fahrtzeit über Straße/Kante von u nach v

Algorithmus: Naiver Algorithmus von s

1: $dist[s] \leftarrow 0$
2: $dist[v] \leftarrow \infty$ für alle $v \neq s$
3: **while** gibt Kante (u, v) mit $dist[u] + zeit(u, v) < dist[v]$ **do**
4: $\quad dist[v] \leftarrow dist[u] + zeit(u, v)$

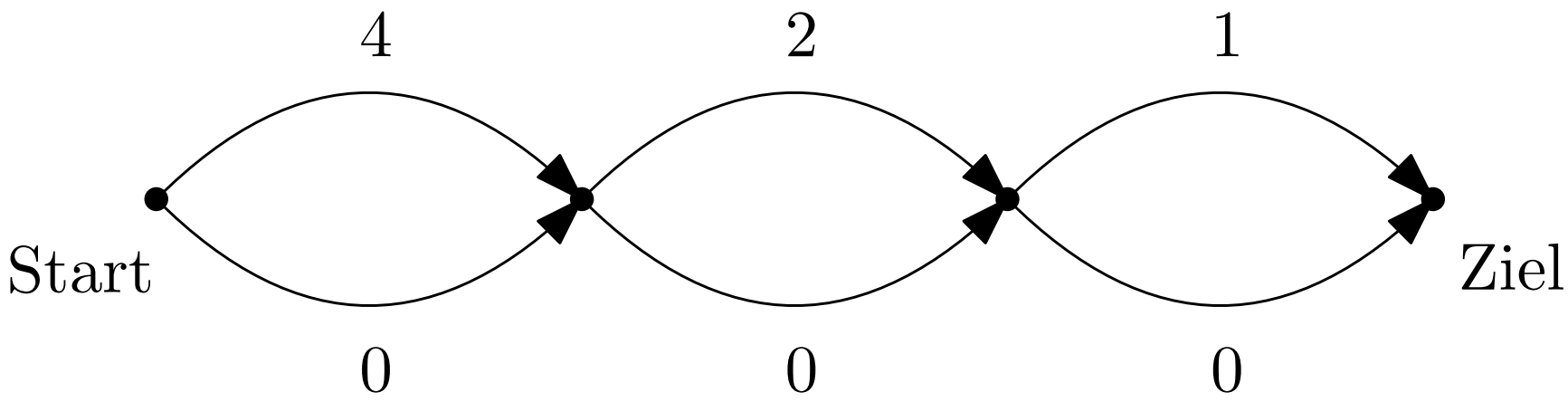
Korrektheit

- *Distanz nicht zu klein:* Wegen Zuweisung am Anfang und später in der Schleife. In beiden Fällen kann die kürzeste Distanz nicht unterschritten werden.
- *Distanz nicht zu groß:* Wenn von s nach v nicht die kürzeste Fahrtzeit gefunden wurde, muss für einen Vorgänger auf dem kürzesten Pfad eine Verbesserung möglich sein. Also wäre die Schleife noch mindestens einmal durchlaufen worden.

Korrektheit

- *Distanz nicht zu klein:* Wegen Zuweisung am Anfang und später in der Schleife. In beiden Fällen kann die kürzeste Distanz nicht unterschritten werden.
- *Distanz nicht zu groß:* Wenn von s nach v nicht die kürzeste Fahrtzeit gefunden wurde, muss für einen Vorgänger auf dem kürzesten Pfad eine Verbesserung möglich sein. Also wäre die Schleife noch mindestens einmal durchlaufen worden.

Laufzeit



Anzahl der möglichen Verbesserungen der Zieldistanz:

- Bei diesem Beispiel: 8
- Gleiches Beispiel mit n Knoten: 2^n

$\rightarrow$ Exponentielle Laufzeit bei schlechter Wahl der Kanten!

Dijkstras Algorithmus

Grundidee

Kanten im naiven Algorithmus geschickt wählen!

Algorithmus

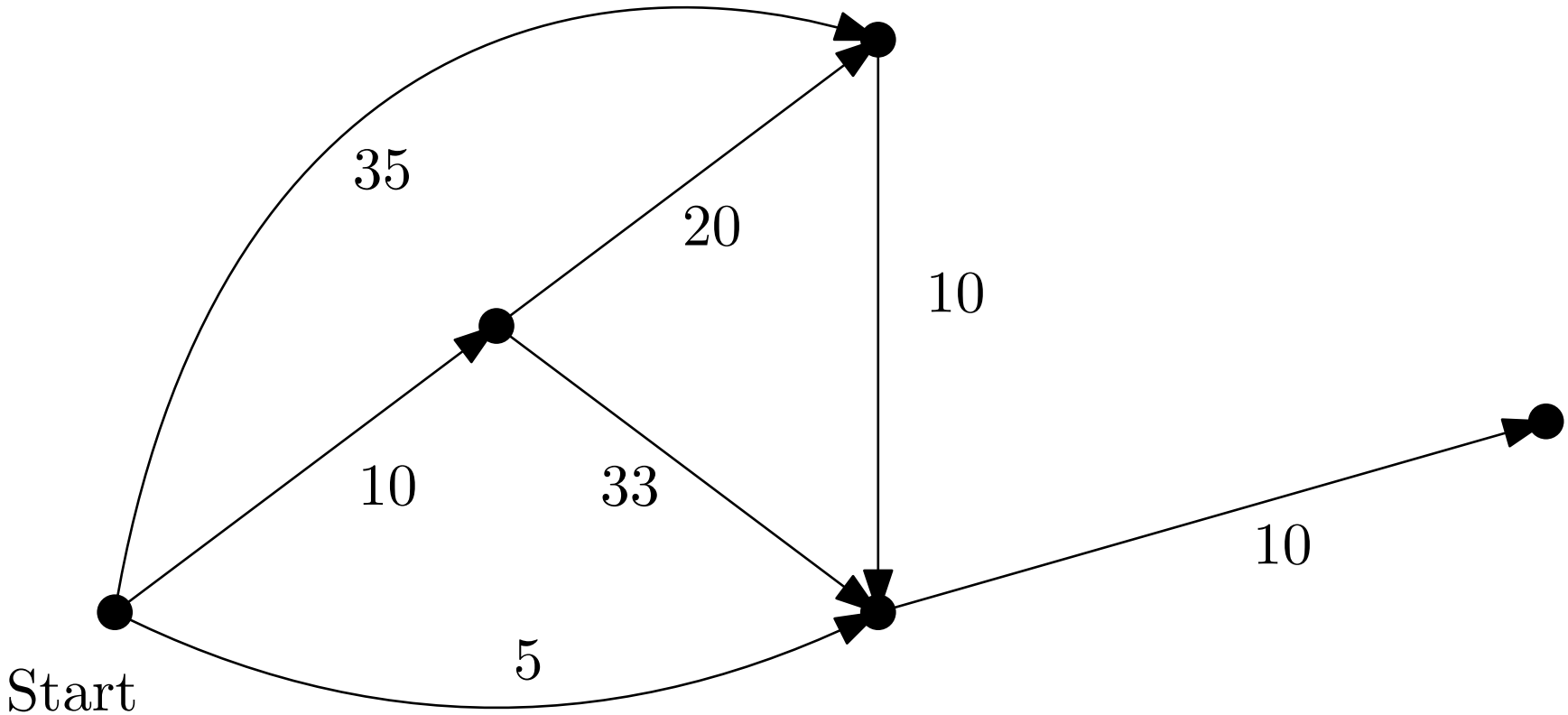
Gegeben: Ein Straßengraph und Startknoten s

Gesucht: Kürzester Weg von s zu allen Knoten

Algorithmus

1. Fahrtzeit von s zu s ist 0 und zu anderen Knoten ∞
2. Finde Straße/Kante von a nach b für die gilt:
   - *Fahrtzeit von s nach a + Länge der Straße/Kante a nach b* $\neq$ Fahrtzeit von s nach b
   - *Fahrtzeit von s nach a + Länge der Straße/Kante a nach b* ist minimal (unter allen in Punkt 2 betrachteten Kanten)
3. Fahrtzeit von s nach b = *Fahrtzeit von s nach a + Länge der Straße/Kante von a nach b*
4. Gehe zu 2. falls noch eine Fahrtzeit verbessert werden kann

Beispiel



→ Ausführung siehe Tafel

Pseudocode

- $dist[v]$ = beste bisher gefundene Distanz von s nach v
- $zeit(u, v)$ = Fahrtzeit über Straße/Kante von u nach v

Algorithmus: Dijkstras Algorithmus von s

```
1: dist[s] ← 0
2: dist[v] ← ∞ für alle v ≠ s
3: Färbe alle Knoten rot
4: while es gibt einen roten Knoten do
5:     u ← der rote Knoten mit dem kleinsten Wert dist[u]
6:     Färbe u schwarz
7:     for alle Kanten (u, v) do
8:         if dist[u] + zeit(u, v) < dist[v] then
9:             dist[v] ← dist[u] + zeit(u, v)
```

Pseudocode

- $dist[v] =$ beste bisher gefundene Distanz von s nach v
- $zeit(u, v) =$ Fahrtzeit über Straße/Kante von u nach v

Algorithmus: Dijkstras Algorithmus von s

```
1: dist[s] ← 0
2: dist[v] ← ∞ für alle v ≠ s
3: Färbe alle Knoten rot
4: while es gibt einen roten Knoten do
5:     u ← der rote Knoten mit dem kleinsten Wert dist[u]
6:     Färbe u schwarz
7:     for alle Kanten (u, v) do
8:         if dist[u] + zeit(u, v) < dist[v] then
9:             dist[v] ← dist[u] + zeit(u, v)
```

Korrektheit

- Ähnlich wie beim naiven Algorithmus, aber komplizierter
- Beobachtung: Jeder Knoten wird nur einmal schwarz gefärbt!
- Idee: Wenn ein Knoten schwarz gefärbt wird, dann kennen wir die kürzeste Distanz zu ihm!

Laufzeit

- $n =$ Anzahl der Knoten, $m =$ Anzahl der Kanten
- Anzahl der Schritte/Schleifendurchläufe in rot

Algorithmus: Dijkstras Algorithmus von s

1: $dist[s] \leftarrow 0$
2: $dist[v] \leftarrow \infty$ für alle $v \neq s$ ▷ n
3: Färbe alle Knoten rot ▷ n
4: **while** es gibt einen roten Knoten **do** ▷ n
5: $u \leftarrow$ der rote Knoten mit dem kleinsten Wert $dist[u]$ ▷ n
6: Färbe u schwarz
7: **for** alle Kanten (u, v) **do** ▷ m
8: **if** $dist[u] + zeit(u, v) < dist[v]$ **then**
9: $dist[v] \leftarrow dist[u] + zeit(u, v)$

$\rightarrow$ Laufzeit: $2n + n \cdot n + m \approx n^2$ (optimiert: $m + n \log n$)

Laufzeit

- $n =$ Anzahl der Knoten, $m =$ Anzahl der Kanten
- Anzahl der Schritte/Schleifendurchläufe in rot

Algorithmus: Dijkstras Algorithmus von s

1: $dist[s] \leftarrow 0$
2: $dist[v] \leftarrow \infty$ für alle $v \neq s$ ▷ n
3: Färbe alle Knoten rot ▷ n
4: **while** es gibt einen roten Knoten **do** ▷ n
5: $u \leftarrow$ der rote Knoten mit dem kleinsten Wert $dist[u]$ ▷ n
6: Färbe u schwarz
7: **for** alle Kanten (u, v) **do** ▷ m
8: **if** $dist[u] + zeit(u, v) < dist[v]$ **then**
9: $dist[v] \leftarrow dist[u] + zeit(u, v)$

$\rightarrow$ Laufzeit: $2n + n \cdot n + m \approx n^2$ (optimiert: $m + n \log n$)

Laufzeit

- $n =$ Anzahl der Knoten, $m =$ Anzahl der Kanten
- Anzahl der Schritte/Schleifendurchläufe in rot

Algorithmus: Dijkstras Algorithmus von s

1: $dist[s] \leftarrow 0$
2: $dist[v] \leftarrow \infty$ für alle $v \neq s$ ▷ n
3: Färbe alle Knoten rot ▷ n
4: **while** es gibt einen roten Knoten **do** ▷ n
5: $u \leftarrow$ der rote Knoten mit dem kleinsten Wert $dist[u]$ ▷ n
6: Färbe u schwarz
7: **for** alle Kanten (u, v) **do** ▷ m
8: **if** $dist[u] + zeit(u, v) < dist[v]$ **then**
9: $dist[v] \leftarrow dist[u] + zeit(u, v)$

$\rightarrow$ Laufzeit: $2n + n \cdot n + m \approx n^2$ (optimiert: $m + n \log n$)

Transitknoten

Verbesserungsmöglichkeiten

Idee: Alles vorberechnen (ein Graph, viele Anfragen)

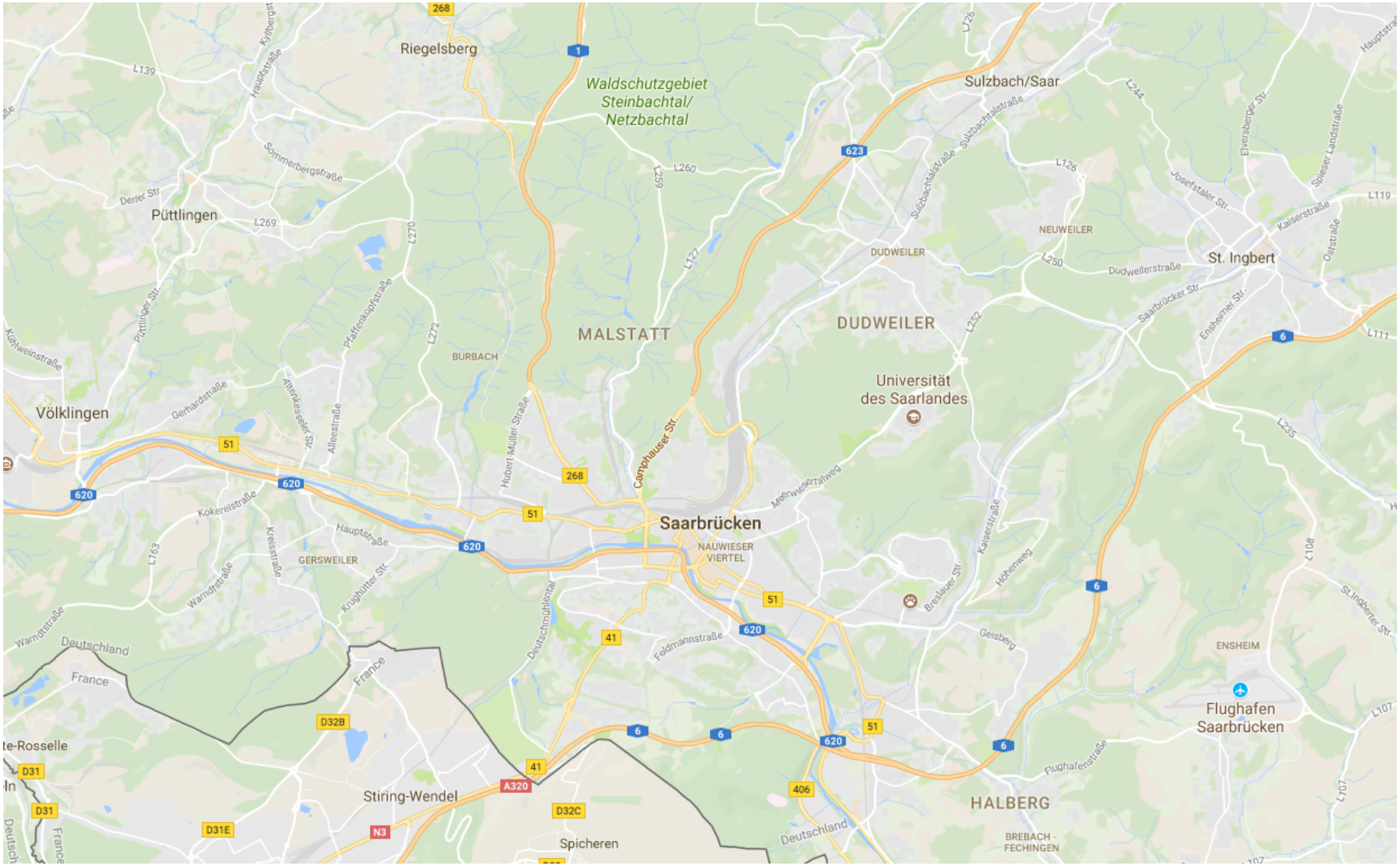
Problem: Alle Distanzen vorberechnen und speichern braucht

- ca. 2 Jahre
- ca. 1.2 Millionen Gigabyte

Besser:

- Nur wichtige Distanzen vorberechnen
- Hierarchie des Straßennetzwerkes nutzen

Hierarchie


Riegelsberg
Waldschutzgebiet Steinbachtal/ Netzbachtal
Sulzbach/Saar
Püttlingen
MALSTATT
DUDWEILER
NEUWEILER
St. Ingbert
BURBACH
Völklingen
Universität des Saarlandes
Saarbrücken
NAUWIESER VIERTEL
GERSWEILER
Deutschland
France
Stiring-Wendel
Spicheren
ENSHEIM
Flughafen Saarbrücken
HALBERG
BREBACH - FECHINGEN

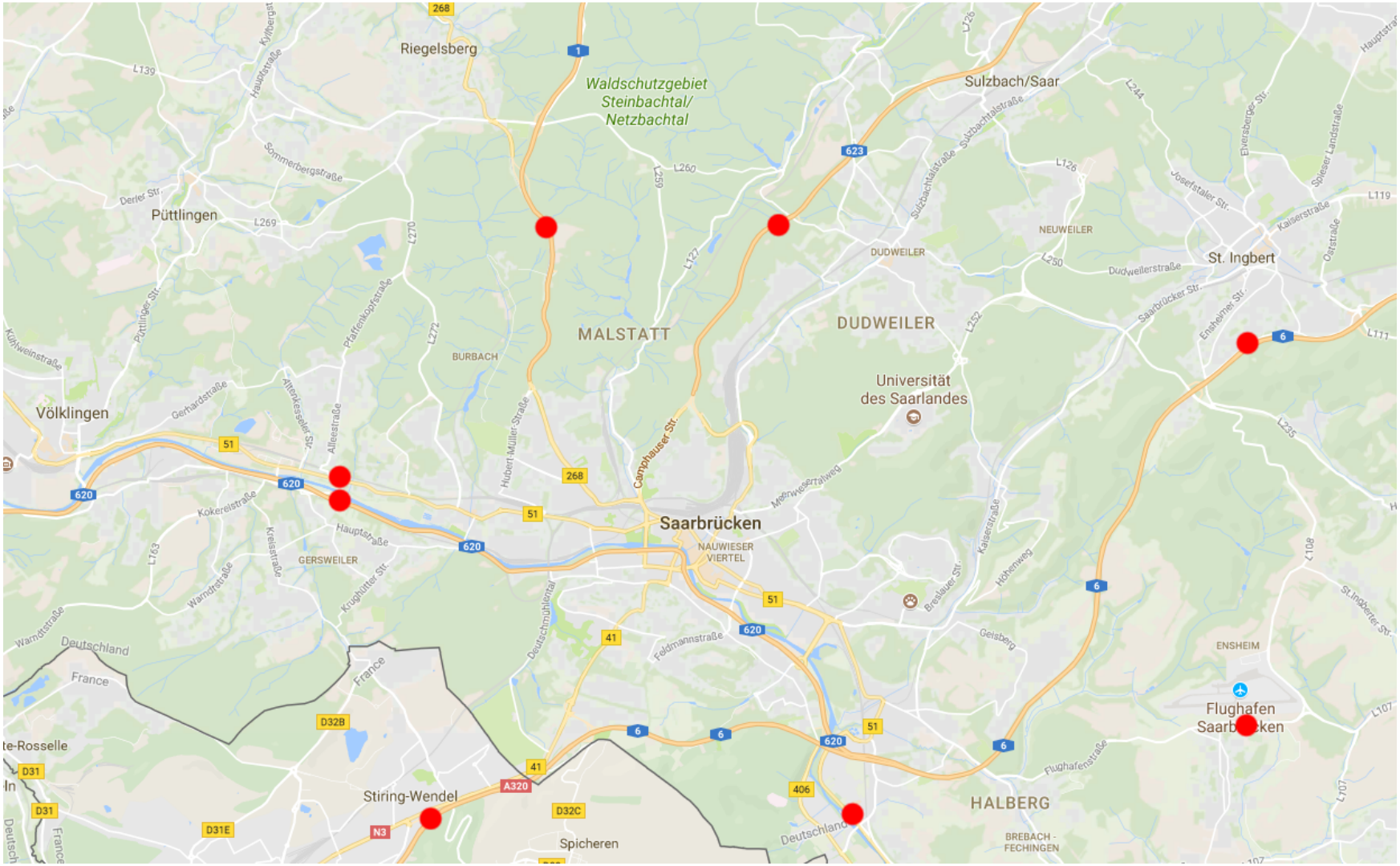
Grundidee

- *Beobachtung:* Alle Menschen aus einem bestimmten Bereich fahren bei langen Strecken über die gleichen Knoten. Wir nennen diese Knoten Transitknoten.
- *Beobachtung:* Es gibt nur wenige *Transitknoten* aufgrund der Hierarchie von Straßennetzwerken.
- *Idee:* Distanzen von und zu *Transitknoten* vorberechnen.
- *Dann:* Diese Distanzen als Tabelle verwenden

Algorithmus

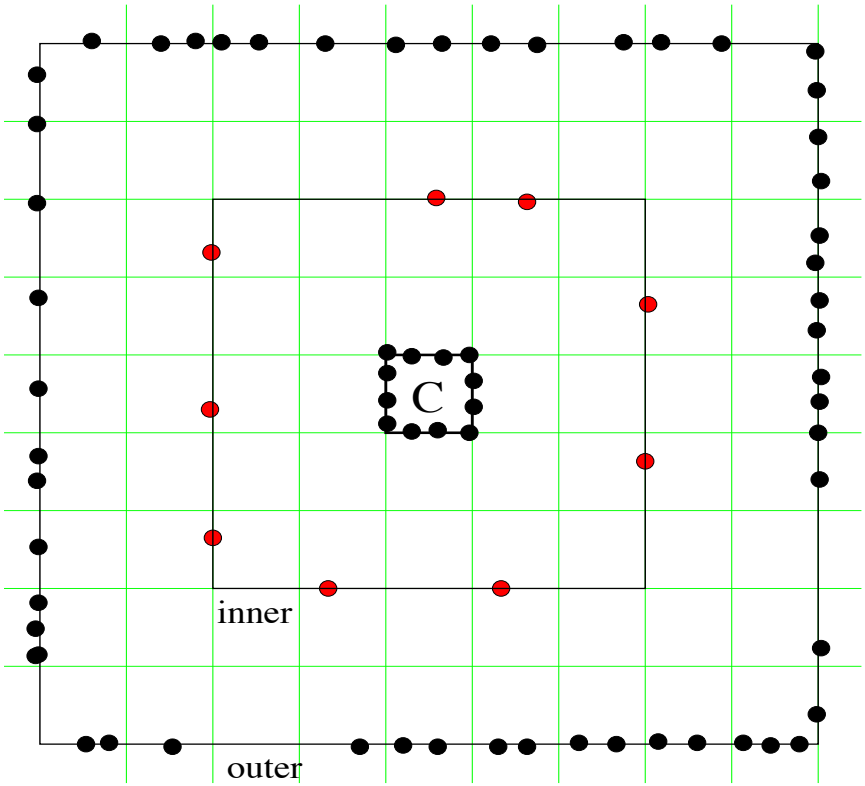
- *Kurze Wege:* Mit Dijkstras Algorithmus berechnen (schnell für kurze Distanzen!)
- *Lange Wege:* Wir müssen drei Distanzen ermitteln:
  - Start zu all seinen Transitknoten
  - Ziel zu all seinen Transitknoten ("rückwärts fahrend")
  - Distanzen zwischen allen paaren von Start- und Ziel-Transitknoten

Transitknoten Nauwieser Viertel



Berechnung der Transitknoten

- Ziehe Gitter über das Straßennetzwerk
- Für Zelle *C* die Wege nach "outer" ausrechnen
- Knoten aus "inner" auf diesen Pfaden sind die Transitknoten



Vorberechnung für USA

- Anzahl der Knoten im Graph: $24 \cdot 10^6$
- Anzahl der Transitknoten: 10^4
- Transitknoten pro Zelle: 10
- Anzahl Distanzen Knoten zu Transitknoten: $24 \cdot 10^7$
- Anzahl paarweise Distanzen zwischen Transitknoten: 10^8

$\rightarrow$ Das sind ein paar GB und passt also auf einen normalen PC!

Nachbemerkungen

Weitere Probleme

- Öffentlicher Nah- und Fernverkehr (schwieriger!)
- Alternative Routen (Menschen wollen Auswahl!)
- Straßennetzwerk scannen (Hausnummern, Schilder, . . .)
- Adressen rekonstruieren (genaue Anfahrt)
- Staus ermitteln (durch Smartphones!)
- Selbstfahrende Autos (Spuren, Vorfahrt, . . .)

Zusammenfassung

- Straßennetzwerke werden als Graphen dargestellt
- Dijkstras Algorithmus ist alt aber noch relevant
- Moderne Algorithmen nutzen Hierarchie von Straßengraphen
- Viele andere Problemstellungen bezüglich Navigation


OCEANI GERMANICI
PARS.
HARLINGERLAND
AVRIKERLAND
GRONINGERLAND
PARS COM: OLDENBVRGENSIS
SADERLAND.
Dat Brack
Emden
TYPVS
FRISIÆ
ORIENTALIS.
Auctore
Vbbone Emmio.