

Praktikum Routenplanung

Vorbesprechung, Wintersemester 2012/2013

Moritz Baum, Julian Dibbelt, Thomas Pajor, Ben Strasser | 17. Oktober 2012

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



Praktikum

- Bearbeitung in 2–3er-Gruppen
- Betreuer: Moritz Baum, Julian Dibbelt, Thomas Pajor, Ben Strasser
- Credits: 6 ETCS (4 SWS)
- Email: {moritz.baum, dibbelt, pajor, ben.strasser}@kit.edu
- Sprechstunde in Raum 318, 322 und 246 (im „Keller“)
- Bei Fragen kommt einfach vorbei

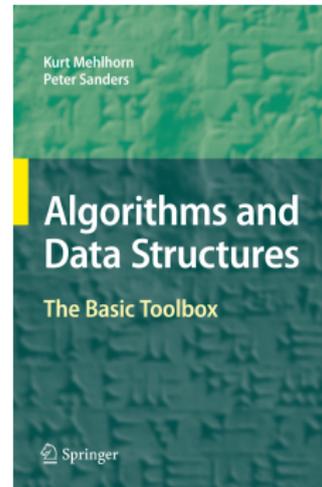
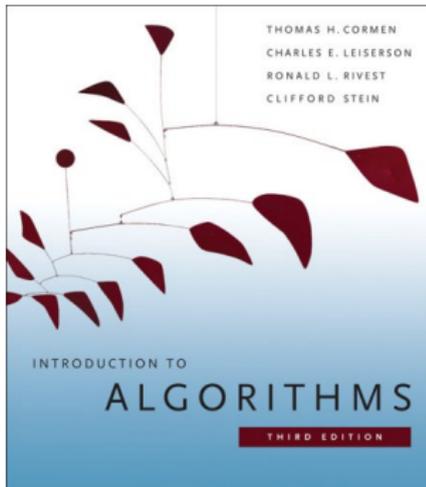
Homepage:

<http://i11www.itl.uni-karlsruhe.de/teaching/winter2012/algorithmengineeringpraktikum/index>

- Informatik I/II oder Algorithmen I
- Algorithmentechnik oder Algorithmen II (muss aber nicht sein)
- Algorithmen für Routenplanung
- ein bisschen Rechnerarchitektur
- Gute Kenntnisse von C++

Material

- Folien
- Einführungsblätter (Übung)
- wissenschaftliche Aufsätze
- Betreuer ;-)
- Basiskenntnisse:



- 1 Organisatorisches
- 2 Einführung
- 3 Zeitplan
- 4 Themen
- 5 Framework

0. Motivation

Worum geht es bei der Routenplanung?



Problemstellung

Gesucht:

- finde die **beste** Verbindung in einem Transportnetzwerk

Idee:

- Netzwerk als Graphen $G = (V, E)$
- Kantengewichte sind **Reisezeiten**
- **kürzeste** Wege in G entsprechen **schnellsten** Verbindungen
- klassisches Problem (Dijkstra)

Probleme:

- Transportnetzwerke sind **groß**
- Dijkstra zu **langsam** (> 1 Sekunde)



Problemstellung

Gesucht:

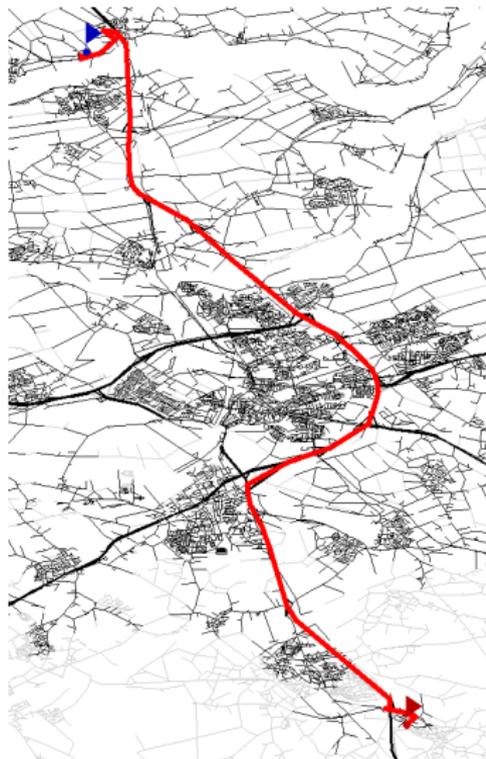
- finde die **beste** Verbindung in einem Transportnetzwerk

Idee:

- Netzwerk als Graphen $G = (V, E)$
- Kantengewichte sind **Reisezeiten**
- **kürzeste** Wege in G entsprechen **schnellsten** Verbindungen
- klassisches Problem (Dijkstra)

Probleme:

- Transportnetzwerke sind **groß**
- Dijkstra zu **langsam** (> 1 Sekunde)



Problemstellung

Gesucht:

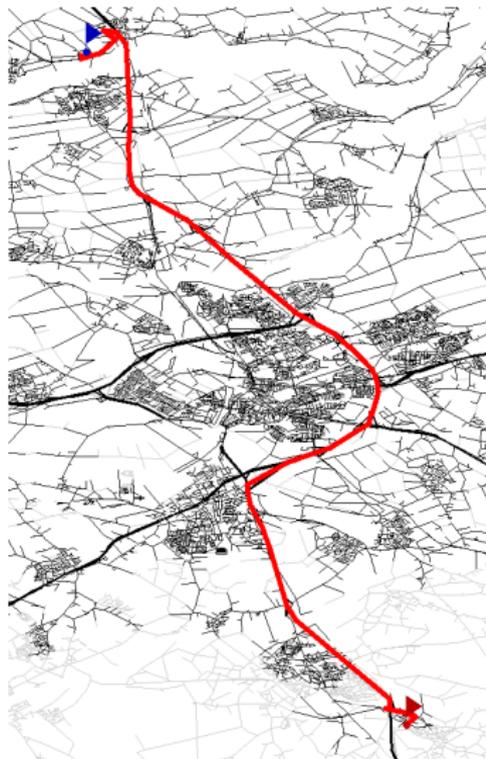
- finde die **beste** Verbindung in einem Transportnetzwerk

Idee:

- Netzwerk als Graphen $G = (V, E)$
- Kantengewichte sind **Reisezeiten**
- **kürzeste** Wege in G entsprechen **schnellsten** Verbindungen
- klassisches Problem (Dijkstra)

Probleme:

- Transportnetzwerke sind **groß**
- Dijkstra zu **langsam** (> 1 Sekunde)

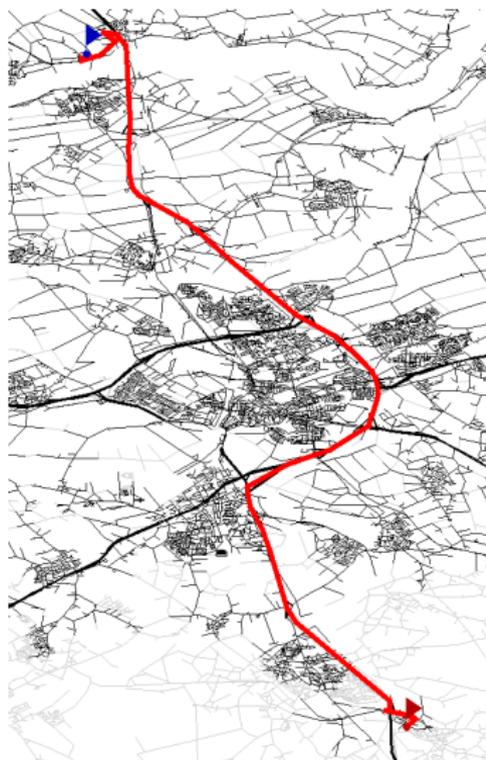


Beobachtungen:

- viele Anfragen in (statischem) Netzwerk
- manche Berechnungen scheinen **unnötig**

Idee:

- Zwei-Phasen Algorithmus:
 - offline: berechne Zusatzinformation während **Vorbereitung**
 - online: **beschleunige** Berechnung mit diesen Zusatzinformationen
- drei Kriterien:
 - wenig Zusatzinformation $\mathcal{O}(n)$
 - kurze Vorbereitung (im Bereich Stunden/Minuten)
 - hohe Beschleunigung

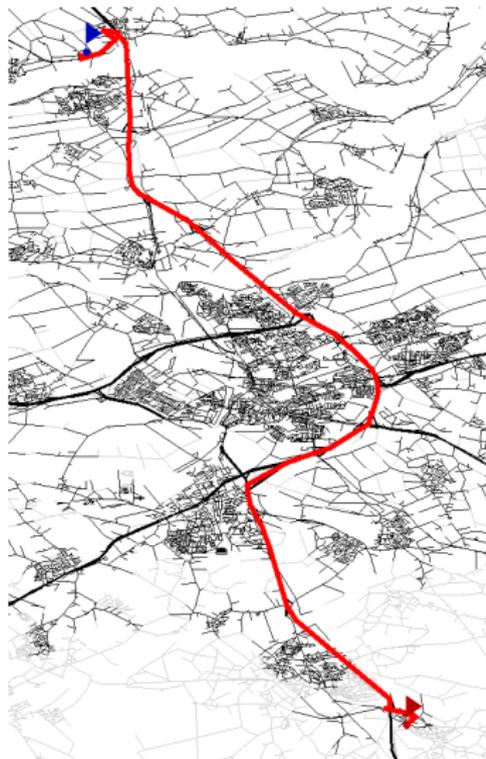


Beobachtungen:

- viele Anfragen in (statischem) Netzwerk
- manche Berechnungen scheinen **unnötig**

Idee:

- Zwei-Phasen Algorithmus:
 - offline: berechne Zusatzinformation während **Vorbereitung**
 - online: **beschleunige** Berechnung mit diesen Zusatzinformationen
- drei Kriterien:
 - wenig Zusatzinformation $\mathcal{O}(n)$
 - kurze Vorbereitung (im Bereich Stunden/Minuten)
 - hohe Beschleunigung

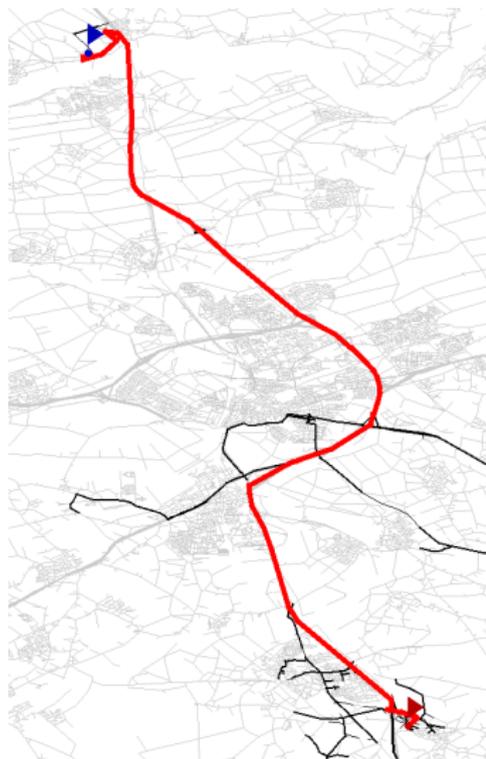


Beobachtungen:

- viele Anfragen in (statischem) Netzwerk
- manche Berechnungen scheinen **unnötig**

Idee:

- Zwei-Phasen Algorithmus:
 - offline: berechne Zusatzinformation während **Vorbereitung**
 - online: **beschleunige** Berechnung mit diesen Zusatzinformationen
- drei Kriterien:
 - wenig Zusatzinformation $\mathcal{O}(n)$
 - kurze Vorbereitung (im Bereich Stunden/Minuten)
 - hohe Beschleunigung



Unterschiede zur Industrie

Industrie:

- Falk, TomTom, bahn.de, usw.
- alles **heuristische** Verfahren
 - betrachte nur noch “wichtige” Kanten wenn mehr als x Kilometer von Start weg
 - Kombination mit A*-Suche
 - langsam!
- Ausnahme: Bing Maps, Google Maps



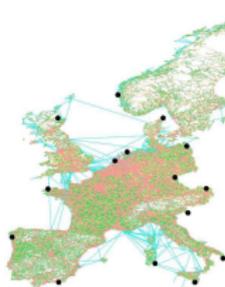
Unser Anspruch:

- Anfragen sollen **beweisbar** korrekt sein
- ⇒ weniger Ausnahmeregelungen
- ⇒ schneller (!)
- Verfahren sollen nach und nach in der Industrie eingesetzt werden

Ergebnisse

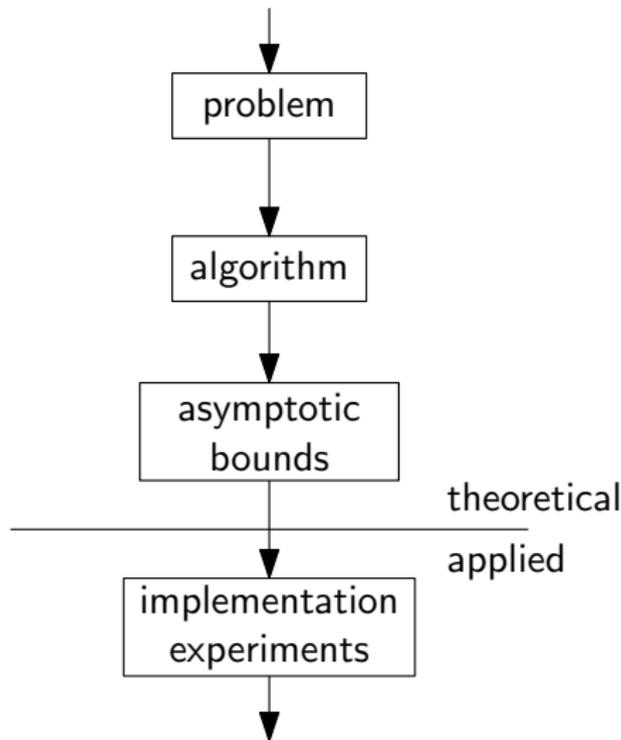
Eingabe: Straßennetzwerk von Westeuropa

- 18 Mio. Knoten
- 42 Mio. Kanten



	Jahr	VORBERECHNUNG		ANFRAGE	
		Zeit [h:m]	Platz [byte/n]	Zeit [ms]	Beschl.
Dijkstra	1959*	0:00	0	5 153.0	0
Arc-Flags	2004	17:08	19	1.6	3 221
Transit-Node Routing	2006	1:15	226	0.0043	1.2 Mio.
Contraction Hier.	2008	0:29	0	0.19	27 121
CH + Arc-Flags	2008	1:39	12	0.017	ca. 300 000
TNR + AF	2008	3:49	312	0.0019	ca. 3 Mio.
Hub Labels	2011	≈ 4:30	1 241	≈ 0.0005	ca. 11 Mio.

* Damalige Variante deutlich langsamer, wir sehen nachher, warum



Lücke Theorie vs. Praxis

Theorie	vs.	Praxis
einfach einfach	Problem-Modell Maschinenmodell	komplex komplex
komplex fortgeschritten	Algorithmen Datenstrukturen	einfach einfach
worst-case asymptotisch	Komplexitäts-Messung Effizienz	typische Eingaben konstante Faktoren

hier:

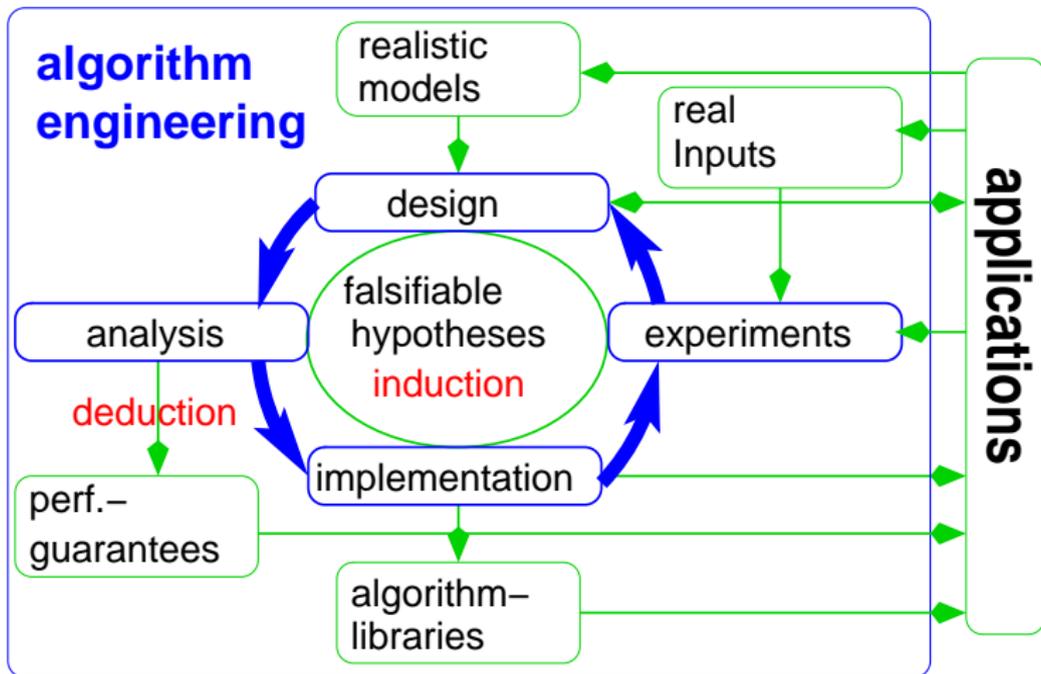
- sehr anwendungsnahes Gebiet
- Eingaben sind **echte** Daten
 - Straßengraphen
 - Eisenbahn (Fahrpläne)
 - Flugpläne

Lücke Theorie vs. Praxis

Theorie	vs.	Praxis
einfach einfach	Problem-Modell Maschinenmodell	komplex komplex
komplex fortgeschritten	Algorithmen Datenstrukturen	einfach einfach
worst-case asymptotisch	Komplexitäts-Messung Effizienz	typische Eingaben konstante Faktoren

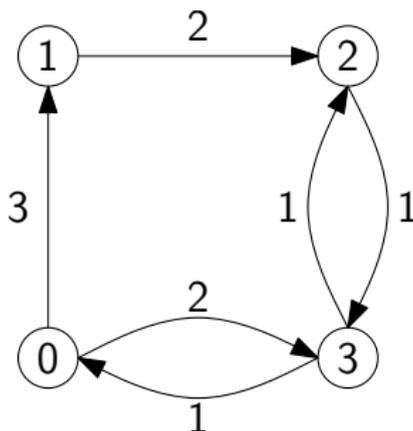
hier:

- sehr anwendungsnahes Gebiet
- Eingaben sind **echte** Daten
 - Straßengraphen
 - Eisenbahn (Fahrpläne)
 - Flugpläne



Drei klassische Ansätze:

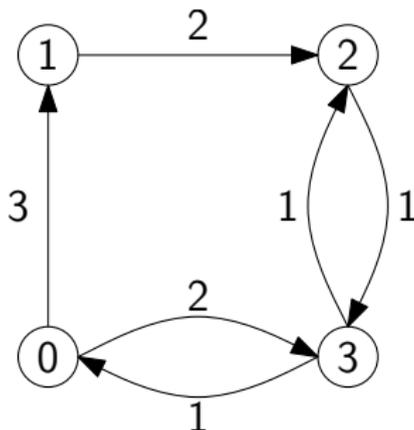
- Adjazenzmatrix
- Adjazenzlisten
- Adjazenzarray



Drei klassische Ansätze:

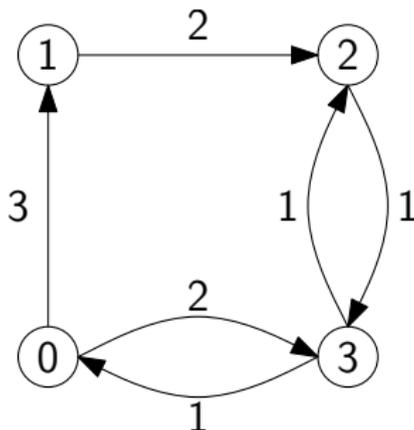
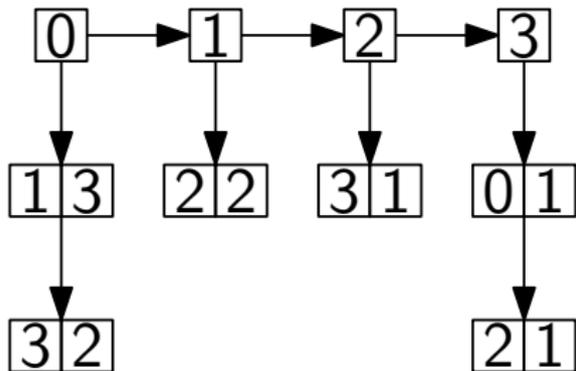
- Adjazenzmatrix
- Adjazenzlisten
- Adjazenzarray

	0	1	2	3
0	—	3	—	2
1	—	—	2	—
2	—	—	—	1
3	1	—	1	—



Drei klassische Ansätze:

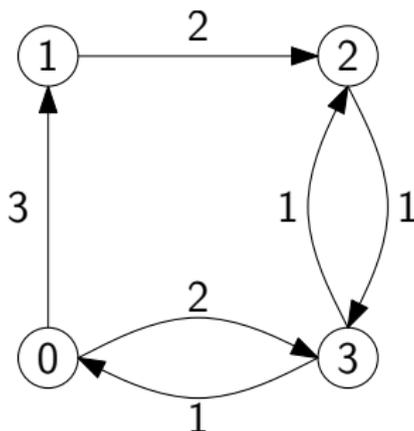
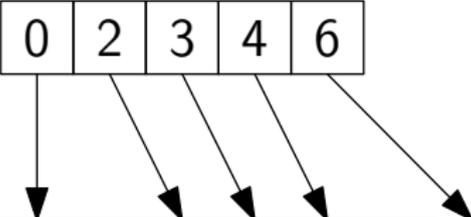
- Adjazenzmatrix
- Adjazenzlisten
- Adjazenzarray



Drei klassische Ansätze:

- Adjazenzmatrix
- Adjazenzlisten
- Adjazenzarray

firstEdge	0	2	3	4	6
targetNode	1	3	2	3	2
weight	3	2	2	1	1



Was brauchen wir?

Eigenschaften:	Matrix	Liste	Array
Speicher	$\mathcal{O}(n^2)$	$\mathcal{O}(n + m)$	$\mathcal{O}(n + m)$
Ausgehende Kanten iterieren	$\mathcal{O}(n)$	$\mathcal{O}(\deg u)$	$\mathcal{O}(\deg u)$
Kantenzugriff (u, v)	$\mathcal{O}(1)$	$\mathcal{O}(\deg u)$	$\mathcal{O}(n + \deg u)$
Effizienz	+	-	+
Updates (topologisch)	+	+	-
Updates (Gewicht)	+	+	+

Fragen:

- Was brauchen wir?
- Was muss nicht supereffizient sein?
- erstmal Modelle anschauen!

Was brauchen wir?

Eigenschaften:	Matrix	Liste	Array
Speicher	$\mathcal{O}(n^2)$	$\mathcal{O}(n + m)$	$\mathcal{O}(n + m)$
Ausgehende Kanten iterieren	$\mathcal{O}(n)$	$\mathcal{O}(\deg u)$	$\mathcal{O}(\deg u)$
Kantenzugriff (u, v)	$\mathcal{O}(1)$	$\mathcal{O}(\deg u)$	$\mathcal{O}(n + \deg u)$
Effizienz	+	-	+
Updates (topologisch)	+	+	-
Updates (Gewicht)	+	+	+

Fragen:

- Was brauchen wir?
- Was muss nicht supereffizient sein?
- erstmal Modelle anschauen!

Modellierung (Straßengraphen)

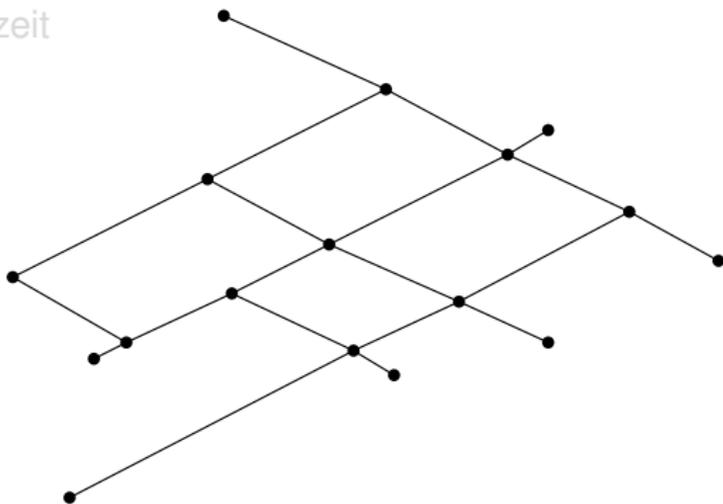


Modellierung (Straßengraphen)



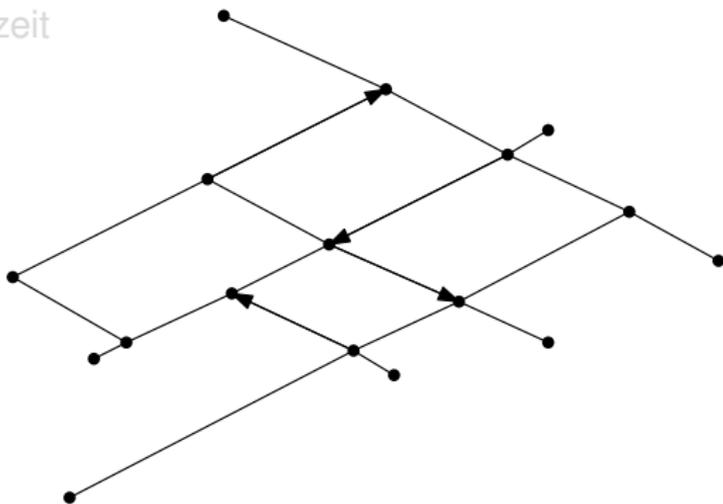
Modellierung (Straßengraphen)

- Knoten sind Kreuzungen
- Kanten sind Straßen
- Einbahnstraßen
- Metrik ist Reisezeit



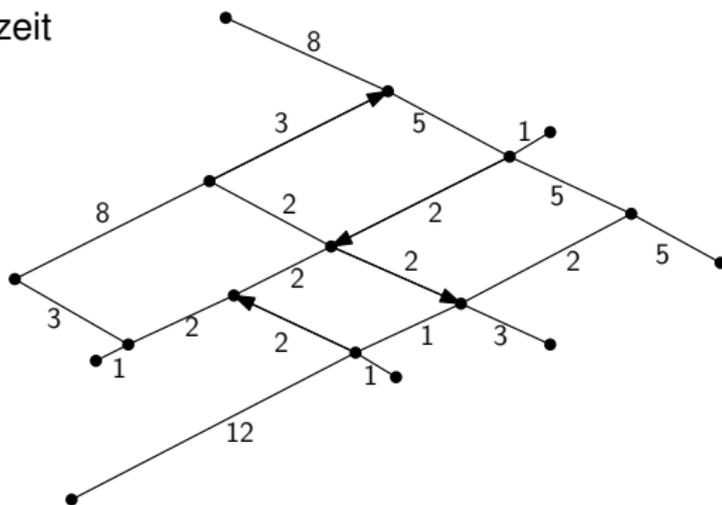
Modellierung (Straßengraphen)

- Knoten sind Kreuzungen
- Kanten sind Straßen
- Einbahnstraßen
- Metrik ist Reisezeit



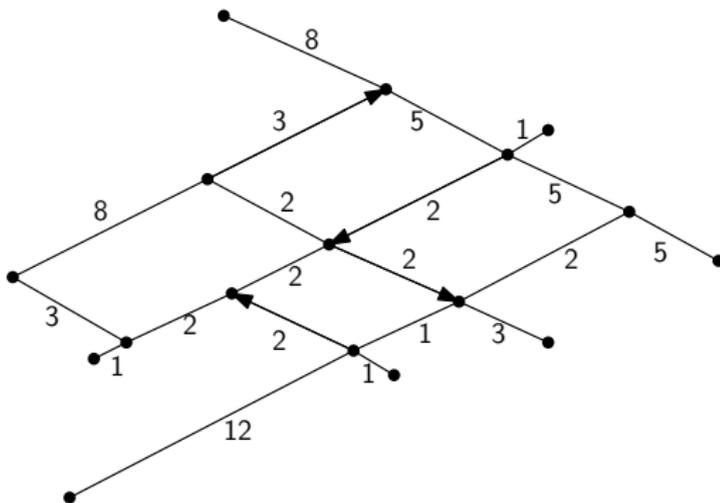
Modellierung (Straßengraphen)

- Knoten sind Kreuzungen
- Kanten sind Straßen
- Einbahnstraßen
- Metrik ist Reisezeit



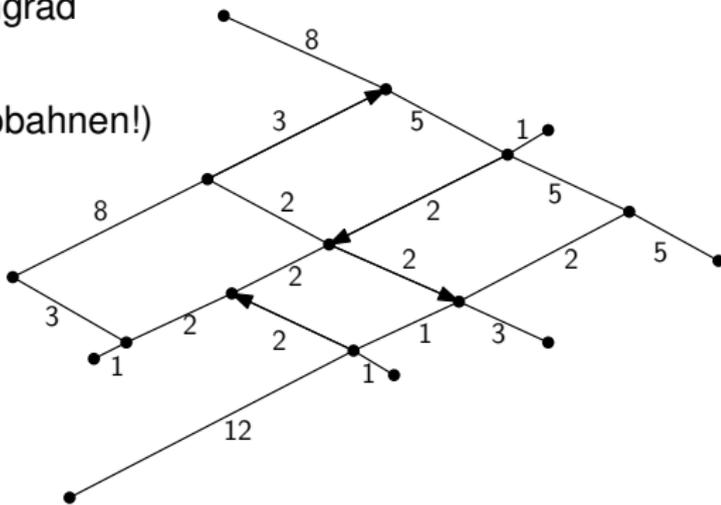
Modellierung (Straßengraphen)

Eigenschaften (sammeln):



Eigenschaften:

- dünn
- (fast) ungerichtet
- geringer Knotengrad
- Kantenzüge
- Hierarchie (Autobahnen!)



- dünn (!)
- gerichtet
- geringer Knotengrad
- meist verborgene Hierarchie (Autobahnen, ICE)
- Einbettung vorhanden (fast planar?)
- Kantengewichte nicht-negativ

Ablauf

- 1 Zuteilung der Themen an die Gruppen (heute)
- 2 Einlesen und Bearbeiten erstes Aufgabenblatt (alle Teilnehmer)
- 3 Bearbeiten zweites Aufgabenblatt (alle Teilnehmer)
- 4 Kurzvorträge, ca. 5–10 min (Vorstellung der Themen)
- 5 Implementierung des Themas
- 6 Endvorträge, ca. 20 min
- 7 Abgabe der Ausarbeitung (ca 10 Seiten)

Abgabe / Vorführen der Aufgabenblätter im Rechnerpool

Terminfindung

Woche	Num.	Aufgabe bis hierhin
Heute 17.10.	1	Vorbesprechung
22.10.–26.10.	2	Abgabe Blatt 1 / Einlesen
05.11.–09.11.	4	Abgabe Blatt 2
19.11.–23.11.	6	Zwischenvorträge (je 5–10 Minuten)
ca. 04.02.–08.02.	ca. 17	Endvorträge + Demo (je 20 Minuten)
Ende Februar		Draft-Version von Ausarbeitung
ca. Semesterende		Abgabe Ausarbeitung

An diesen Terminen ist Anwesenheitspflicht.

Themenübersicht



Problemstellung

Schnelles Berechnen von Kürzesten Wegen in Straßennetzwerken mit beliebigen Metriken.

Motivation

- Speed-Up Technik die mit beliebigen Metriken umgehen kann
Zeit, Fußgänger, keine Autobahnen, Höhenbeschränkungen, etc
- Vorberechnung pro Metrik soll sehr schnell sein
Ein paar Sekunden für den gesamten Graphen
- Extrem schnelle lokale Updates
- Echtzeit Staudaten
- Schnelle Queryzeiten (≤ 10 ms)

Aufgaben

- Implementierung der Metric Customization Phase
(Basierend auf gegebener Partitionierung des Graphen)
- Handhabung von verschiedenen Metriken (Updates)
- Implementierung des Query-Algorithmus
- Implementierung von Shortcut-Unpacking
- Visualisierung von Routen
- Durchführen von Experimenten

D. Delling, A. V. Goldberg, T. Pajor, R. F. Werneck:

Customizable Route Planning.

In: *Proceedings of the 10th International Symposium on Experimental Algorithms (SEA'11)*, volume 6630 of Lecture Notes in Computer Science, pages 376-387. Springer, 2011.

Aufgaben

- Implementierung der Metric Customization Phase
(Basierend auf gegebener Partitionierung des Graphen)
- Handhabung von verschiedenen Metriken (Updates)
- Implementierung des Query-Algorithmus
- Implementierung von Shortcut-Unpacking
- Visualisierung von Routen
- Durchführen von Experimenten

D. Delling, A. V. Goldberg, T. Pajor, R. F. Werneck:

Customizable Route Planning.

In: *Proceedings of the 10th International Symposium on Experimental Algorithms (SEA'11)*, volume 6630 of Lecture Notes in Computer Science, pages 376-387. Springer, 2011.

Problemstellung

Effiziente Berechnung von sinnvollen Alternativrouten in Straßennetzwerken.

Motivation

- Schnellste Route ist meistens eindeutig
- Manche Benutzer bevorzugen abweichende Routen
- Finden von guten Alternativrouten zur schnellsten Route
- Einschränkung auf 1-Via-Pfade
- Anforderungen:
Wenig Gemeinsamkeit, keine unsinnigen Umwege, nahe am Optimum
- Schnelle Berechnung durch Beschleunigungstechniken

Aufgaben

- Berechnen von 1-via-Alternativrouten mit bidirektionaler Suche
- Filtern der Alternativrouten nach Gütekriterien
Sharing, Stretch, Local Optimality
- Beschleunigung der Berechnung mit Contraction Hierarchies
Zu gegebener Contraction Hierarchy
- Vorberechnung guter Via-Kandidaten mit Hilfe einer Partition
Zu gegebener Partition
- Ausnutzung der vorberechneten Via-Kandidaten
- Visualisierung der Routen und Experimente

I. Abaraham, D. Delling, A. V. Goldberg, R. F. Werneck:
Alternative Routes in Road Networks.

In: *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA '10)*, 2010.

Aufgaben

- Berechnen von 1-via-Alternativrouten mit bidirektionaler Suche
- Filtern der Alternativrouten nach Gütekriterien
Sharing, Stretch, Local Optimality
- Beschleunigung der Berechnung mit Contraction Hierarchies
Zu gegebener Contraction Hierarchy
- Vorberechnung guter Via-Kandidaten mit Hilfe einer Partition
Zu gegebener Partition
- Ausnutzung der vorberechneten Via-Kandidaten
- Visualisierung der Routen und Experimente

I. Abaraham, D. Delling, A. V. Goldberg, R. F. Werneck:

Alternative Routes in Road Networks.

In: *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA '10)*, 2010.

Problemstellung

Schnelle Berechnung von one-to-all kürzeste-Wege Bäumen auf Straßennetzwerken.

Motivation

- Dijkstra's Algorithmus berechnet Abstand zu *allen* Knoten in G .
- Schnelle Speed-up Techniken nur für Punkt-zu-Punkt Anfragen.
- Aber: One-to-all Anfragen oft Bestandteil von Vorberechnung.
- Wie lässt sich die Hardware-Architektur geschickt ausnutzen um one-to-all Anfragen zu beschleunigen?

Aufgaben

- Berechnung der Level in einer Contraction Hierarchy
Bei gegebener Contraction Hierarchy
- Implementierung der PHAST-Query
- Implementierung von (Hardware-)Optimierungen
Knotenordnung, Parallelisierung, SSE-Instruktionen
- Berechnen von Arc-Flags mittels PHAST
- Implementierung der Arc-Flags Query
- *Alternative*: Implementieren der PHAST-Query auf GPU
- Visualisierung der Routen und Experimente

D. Delling, A. V. Goldberg, A. Nowatzyk, R. F. Werneck:

PHAST: Hardware-Accelerated Shortest Path Trees.

In: *25th International Parallel and Distributed Processing Symposium (IPDPS'11)*, pages 921-931. IEEE Computer Society, 2011.

Aufgaben

- Berechnung der Level in einer Contraction Hierarchy
Bei gegebener Contraction Hierarchy
- Implementierung der PHAST-Query
- Implementierung von (Hardware-)Optimierungen
Knotenordnung, Parallelisierung, SSE-Instruktionen
- Berechnen von Arc-Flags mittels PHAST
- Implementierung der Arc-Flags Query
- *Alternative*: Implementieren der PHAST-Query auf GPU
- Visualisierung der Routen und Experimente

D. Delling, A. V. Goldberg, A. Nowatzyk, R. F. Werneck:

PHAST: Hardware-Accelerated Shortest Path Trees.

In: *25th International Parallel and Distributed Processing Symposium (IPDPS'11)*, pages 921-931. IEEE Computer Society, 2011.

Problemstellung

Sehr schnelle Routenplanung mit Hilfe einer Datenbank (SQL).

Motivation

- Datenbanken sehr verbreitet. . .
- . . . und leicht zu warten
- Dijkstra's Algorithmus unpraktikabel in SQL
- Simple Implementierung des Anfragealgorithmus (Einfache SQL-Anfrage)
- Erweiterte Anfragen, wie z. B. POI-Anfragen.

Aufgaben

- Implementierung der Vorberechnung eines Labeling-Algorithmus
Zu gegebener Contraction Hierarchy
- Implementierung von Verbesserungen für die Vorberechnung
- Implementierung der HL-Query in C++
- Import der Daten in eine Datenbank
- Implementierung der HL-Query in SQL
- Implementierung einer POI-Query in SQL
- Visualisierung der Labels und Experimente

I. Abraham, D. Delling, A. V. Goldberg, R. F. Werneck.

A Hub-Based Labeling Algorithm for Shortest Paths on Road Networks.

In: *Proceedings of the 10th International Symposium on Experimental Algorithms (SEA'11)*, volume 6630 of Lecture Notes in Computer Science, pages 230-241. Springer, 2011.

Aufgaben

- Implementierung der Vorberechnung eines Labeling-Algorithmus
Zu gegebener Contraction Hierarchy
- Implementierung von Verbesserungen für die Vorberechnung
- Implementierung der HL-Query in C++
- Import der Daten in eine Datenbank
- Implementierung der HL-Query in SQL
- Implementierung einer POI-Query in SQL
- Visualisierung der Labels und Experimente

I. Abraham, D. Delling, A. V. Goldberg, R. F. Werneck.

A Hub-Based Labeling Algorithm for Shortest Paths on Road Networks.

In: *Proceedings of the 10th International Symposium on Experimental Algorithms (SEA'11)*, volume 6630 of Lecture Notes in Computer Science, pages 230-241. Springer, 2011.

Problemstellung

Schnelle Berechnung von multi-kriteriellen (Reisezeit, Anzahl Umstiege, etc.) Anfragen in öffentlichen Verkehrsnetzen.

Motivation

- Netzwerke sind *zeitabhängig*
- Bestehen aus Stops, Routen, Trips, ...
- Modellierung als Graphen zu kompliziert/langsam
- Optimierung von Ankunftszeit alleine nicht ausreichend
- Dynamische Aspekte: Verspätungen, Ausfälle, ...

Aufgaben

- Gegeben: Fahrplan als Datenstruktur
- Implementierung von RAPTOR für Ankunftszeit und Umstiege
- Erweiterung von RAPTOR auf Kriterium *Tarifzonen*
- Parallelisierung von RAPTOR auf Multi-Core
- Ausgabe der Routen und Visualisierung

D. Delling, T. Pajor, R. F. Werneck:

Round-Based Public Transit Routing.

In: *Proceedings of the 14th Meeting on Algorithm Engineering and Experiments (ALENEX'12)*, pages 130-140. SIAM, 2012.

Aufgaben

- Gegeben: Fahrplan als Datenstruktur
- Implementierung von RAPTOR für Ankunftszeit und Umstiege
- Erweiterung von RAPTOR auf Kriterium *Tarifzonen*
- Parallelisierung von RAPTOR auf Multi-Core
- Ausgabe der Routen und Visualisierung

D. Delling, T. Pajor, R. F. Werneck:

Round-Based Public Transit Routing.

In: *Proceedings of the 14th Meeting on Algorithm Engineering and Experiments (ALENEX'12)*, pages 130-140. SIAM, 2012.

Bitte gruppiert euch in 2–3er Gruppen und wählt.

Themen

- Customizable Route Planning
Routenplanung in Straßennetzwerken mit beliebigen Metriken
- Alternativrouten
Effiziente Berechnung von Alternativrouten.
- PHAST
Schnelle Berechnung von one-to-all kürzesten Wegen
- HLDB
Routenplanung mit SQL-Datenbanken
- RAPTOR
Multi-kriterielle Fahrplanauskunft

Rechner-Login

- Ausfüllen von Antragsblatt
- Wir geben per Email bescheid, sobald angelegt

SVN-Zugang

- Wir benutzen SVN zur Versionskontrolle
(siehe erstes Blatt)
- Zugangsdaten per Email an die Gruppen
- Bitte checkt regelmäßig euren Fortschritt ein!

Austeilung, Einführungsblatt



Framework

Für das erste Blatt.



Maxime

Bei der Routenplanung müssen

- große Datenmengen (Graphen)
- schnell und ressourcenschonend verarbeitet werden.

⇒ Wir benutzen C++ mit einem (sehr) schlanken Framework!

Das Framework. . .

- ist selbst geschrieben (und *sehr* klein)
- benutzt lediglich ein paar Komponenten der STL

Vorteile:

- sehr schnell
- kein (kaum) Overhead
- für euch überschaubar :-)

Nachteile:

- Unflexibel (aber für unsere Zwecke ausreichend)

Das Framework besteht aus folgenden Bestandteilen:

- **Graph-Klasse**
Verwaltet den Graphen
- **Priority-Queue**
Zum Beispiel für Dijkstra's Algorithmus
- **Query-Algorithmen**
Zum Beispiel Dijkstra's Algorithmus

... und noch ein bisschen IO.

Umgehen mit Graphen...

Grundlegende Datentypen aus unserem Framework sind

- `int`
Für jegliche Ids (z. B. Knoten-Ids)
- `enum EdgeDirection {`
 `FORWARD_DIRECTION = 0,`
 `BACKWARD_DIRECTION = 1`
};
Kanten-Richtung.
- `int`
Kantengewichte
- `Types::INFTY`
Spezieller `EdgeWeight`-Wert für ∞ .

Die Graphen sind als Adjazenzarray implementiert.

vertices:



edges:

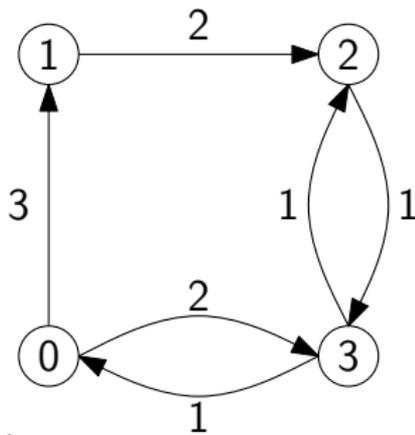


```
struct BasicVertex {  
    int FirstEdgeIndex();  
    int LastEdgeIndex();  
}
```

```
struct BasicEdge {  
    int OtherVertexIndex();  
    bool Forward();  
    bool Backward();  
    int Weight();  
}
```

Adjazenzarray an einem kleinen Beispiel

firstEdge	0	3	5	7	10					
targetNode	1	3	3	0	2	1	3	0	0	2
weight	3	2	1	3	2	2	1	1	2	1
isIncoming	-	-	✓	✓	-	✓	✓	-	✓	✓
isOutgoing	✓	✓	-	-	✓	-	✓	✓	-	✓



Graph-Struktur – Implementierung

Wichtige Bestandteile der Graph-Klasse:

```
template<typename vertexType ,  
typename edgeType , typename connectionType>  
class DynamicGraph {  
    // ...  
    vector<vertexType> vertices ;  
    vector<edgeType> edges ;  
};
```

Der einfache Graph ergibt sich damit durch

```
typedef DynamicGraph<BasicVertex , BasicEdge ,  
BasicConnection> RoadGraph ;
```

Der Rest ist für uns (erstmal) nicht wichtig!

Folgende Funktionen können benutzt werden um Größen zu ermitteln

- `NumVertices()`
Größe des `vertices-vectors` - 1.
- `NumEdges()`
Größe des `edges-vectors`.
- `NumDirectedEdges(Types::FORWARD_DIRECTION)`
Anzahl Kanten mit Vorwärts-Flag.
- `NumDirectedEdges(Types::BACKWARD_DIRECTION)`
Anzahl Kanten mit Rückwärts-Flag.

Achtung: `NumEdges()` entspricht in der Regel nicht der tatsächlichen Anzahl Kanten im Graphen!

Folgende Konstrukte existieren zur Iteration:

- `FORALL_VERTICES(G, u)`
Iteriert über alle Knoten in G
- `FORALL_EDGES(G, u, edge)`
Iteriert über *alle* Kanten, u ist dabei der Quellknoten der Kante. `edge` ist ein Zeiger auf die Kanten-Struct.
- `FORALL_INCIDENT_EDGES(G, u, edge)`
Iteriert über *alle* zu u inzidenten Kanten in G . `edge` ist ein Zeiger auf die Kanten-Struct.

Benutzung wie normale Schleifenkonstrukte in C++ (z. B. `for`)

Vorwärts und Rückwärtskanten

Problem: Nur über *ausgehende* Kanten iterieren.

```
Graph::RoadGraph graph;  
int u = 42;
```

```
// Iteriere ueber ausgehende Kanten von u...  
FORALL_INCIDENT_EDGES(graph, u, edge) {  
  
    if (!edge->Forward())  
        continue;  
  
    std::cout << "Kante mit Gewicht "  
                << edge->Weight() << std::endl;  
}
```

Nur Rückwärtskanten: analog.

Viel Erfolg!

