Karlsruher Institut für Technologie

Praktikum Routenplanung

Vorbesprechung, Wintersemester 2011/2012

Julian Dibbelt und Thomas Pajor | 2. November 2011

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

Organisatorisches

Praktikum

- Bearbeitung in 2er-Gruppen
- Betreuer: Julian Dibbelt und Thomas Pajor
- Credits: 6 ETCS (4 SWS)
- Email: dibbelt@kit.edu / pajor@kit.edu
- Sprechstunde in Raum 318 und 322 (Gebäude 50.34)
- Bei Fragen kommt einfach vorbei!

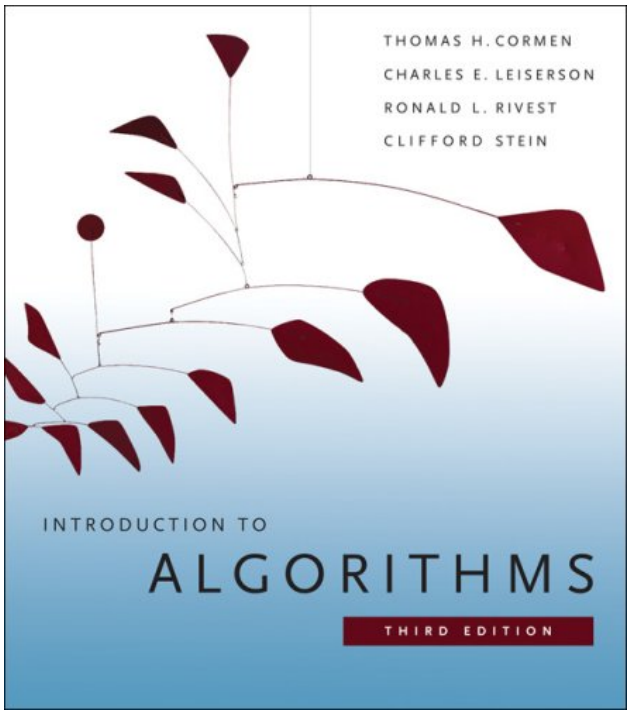
Homepage:
`http://i11www.iti.uni-karlsruhe.de/teaching/winter2011/algorithmengineeringpraktikum/index`

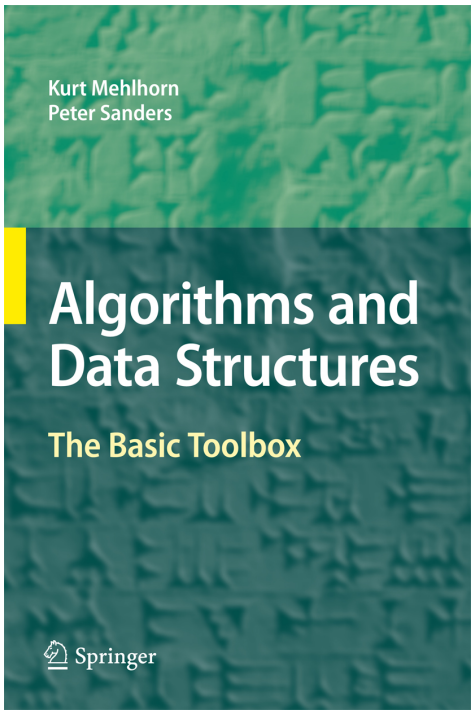
Vorkenntnisse

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

Material

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





Ablauf

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)

Beschleunigungstechniken

Beobachtungen:

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

Idee:

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

Beschleunigungstechniken

Beobachtungen:

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

Idee:

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

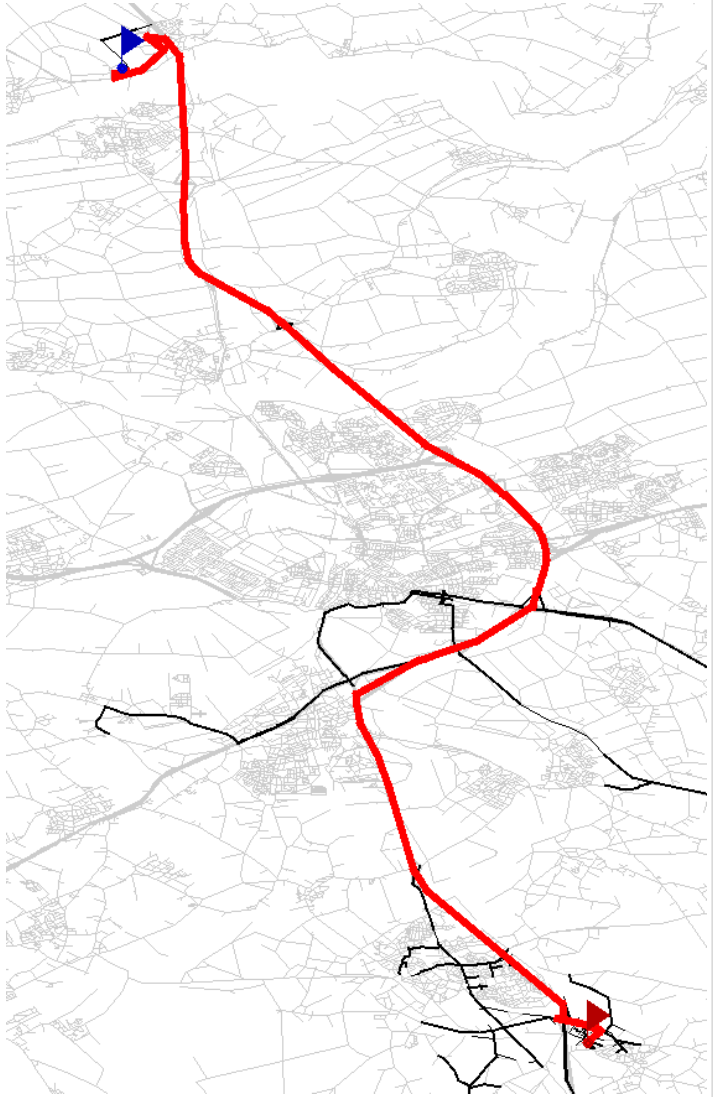
Beschleunigungstechniken

Beobachtungen:

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

Idee:

- Zwei-Phasen Algorithmus:
  - offline: berechne Zusatzinformation während Vorberechnung
  - online: beschleunige Berechung mit diesen Zusatzinformationen
- drei Kriterien:
  - wenig Zusatzinformation $\mathcal{O}(n)$
  - kurze Vorberechnung (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: Google Maps

Unser Anspruch:

- Anfragen sollen beweisbar korrekt sein

$\Rightarrow$ weniger Ausnahmeregelungen

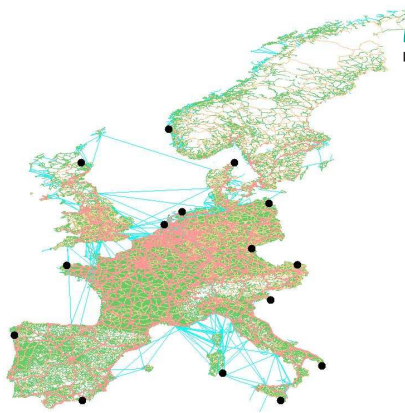
$\Rightarrow$ schneller (!)

- Verfahren sollen nach und nach in der Industrie eingesetzt werden

Ergebnisse

Eingabe: Straßennetzwerk von Westeuropa

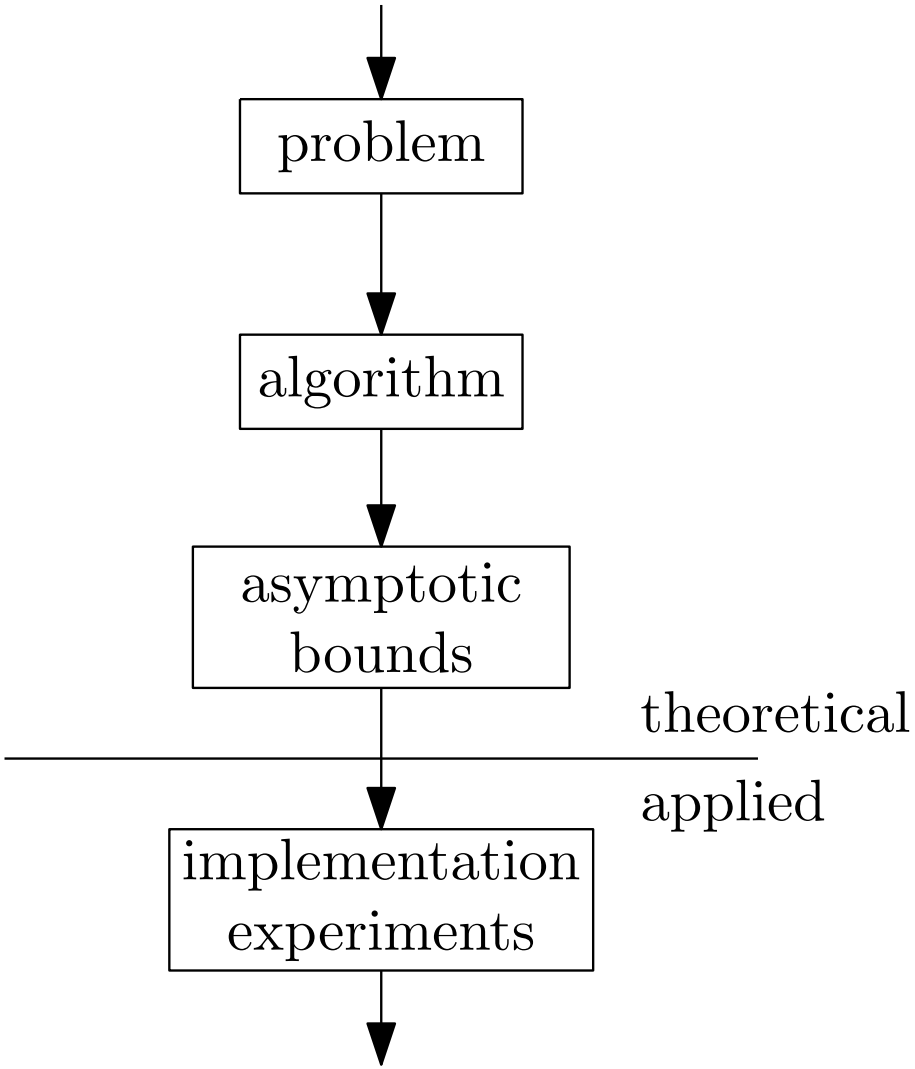
- 18 Mio. Knoten
- 42 Mio. Kanten



		Vorberechnung		Anfrage	
	Jahr	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
Highway Hierarchies	2005	0:13	48	0.61	8 448
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.

* Damalige Variante deutlich langsamer, wir sehen nachher, warum

Klassischer Ansatz



Lücke Theorie vs. Praxis

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

hier:

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

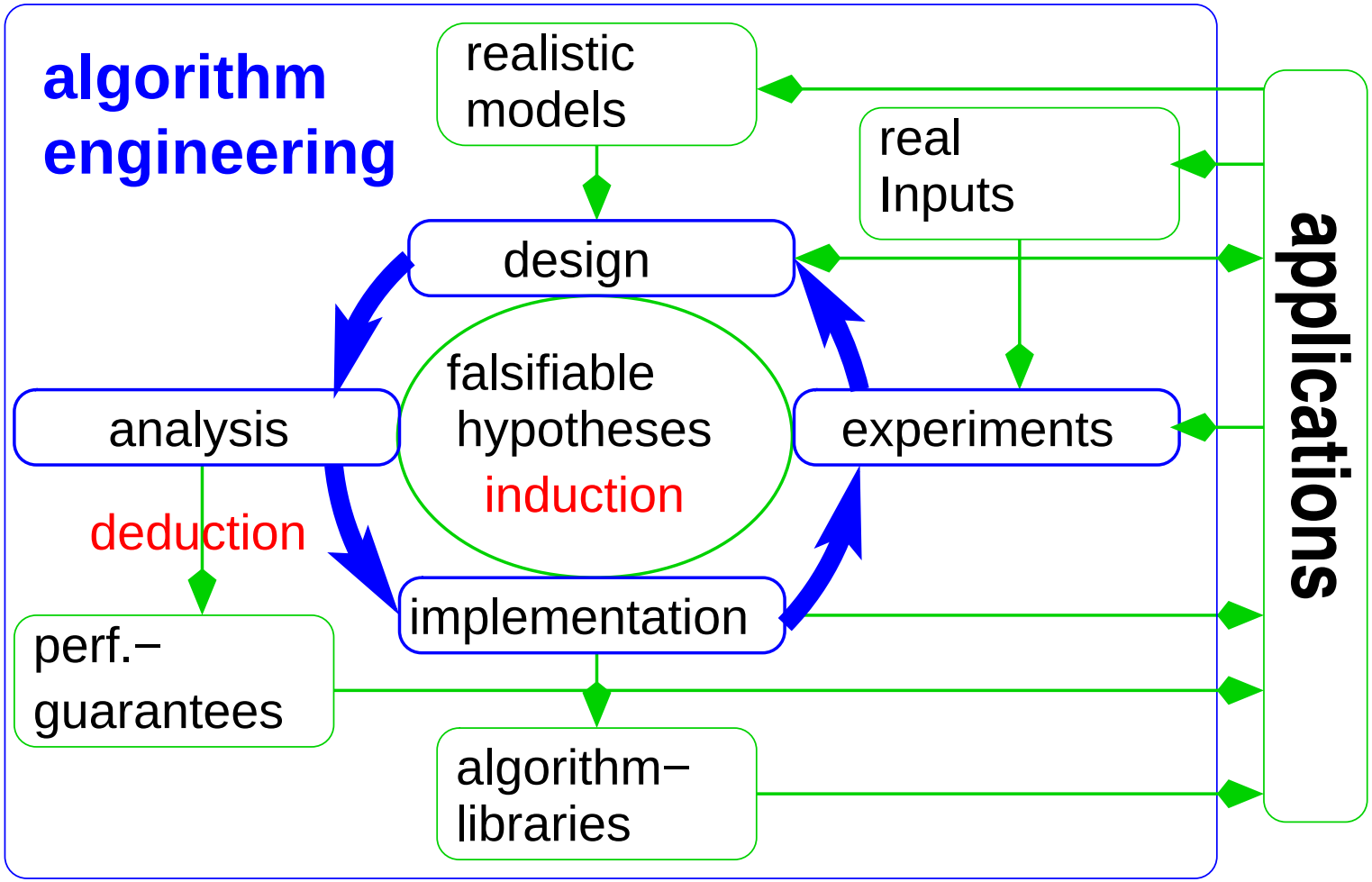
Lücke Theorie vs. Praxis

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

hier:

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

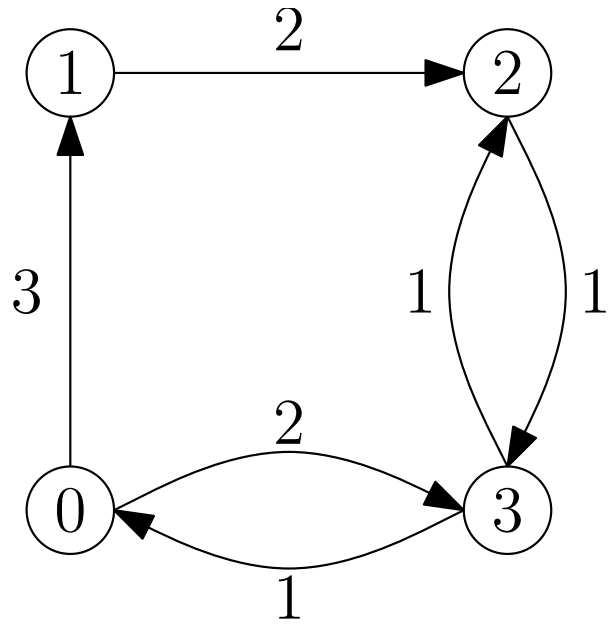
Algorithm Engineering



Graph-Repräsentationen

Drei klassische Ansätze:

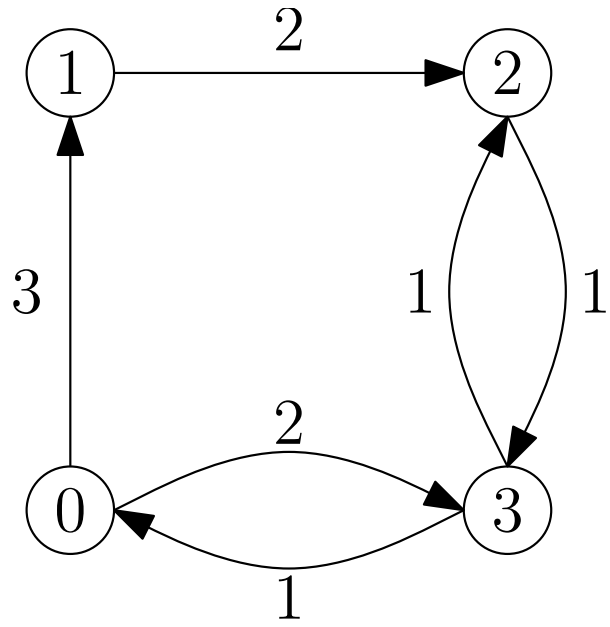
- Adjazenzmatrix
- Adjazenzlisten
- Adjazenzarray



Graph-Repräsentationen

Drei klassische Ansätze:

- Adjazenzmatrix
- Adjazenzlisten
- Adjazenzarray

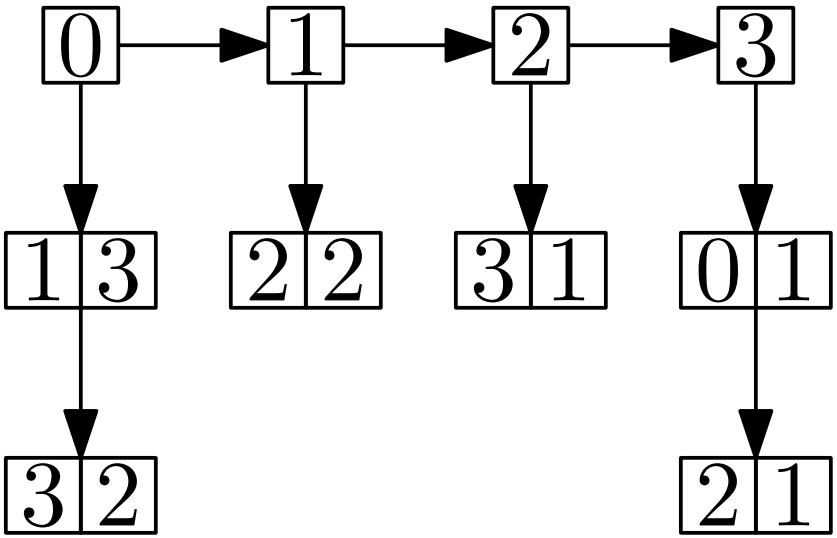


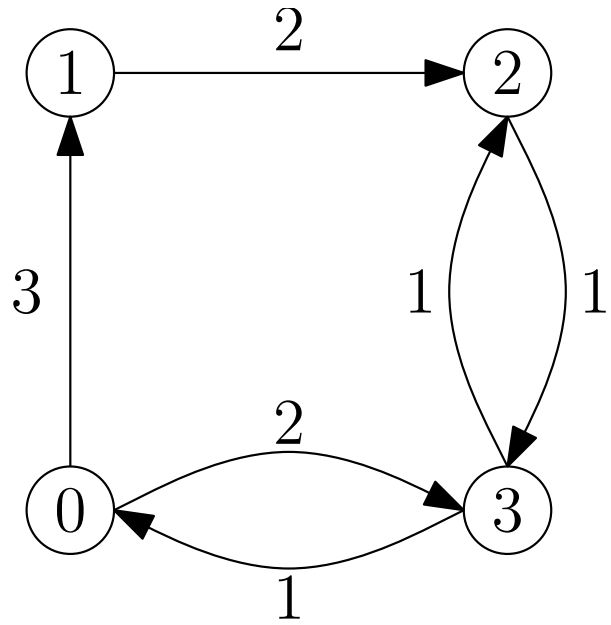
	0	1	2	3
0	–	3	–	2
1	–	–	2	–
2	–	–	–	1
3	1	–	1	–

Graph-Repräsentationen

Drei klassische Ansätze:

- Adjazenzmatrix
- Adjazenzlisten
- Adjazenzarray

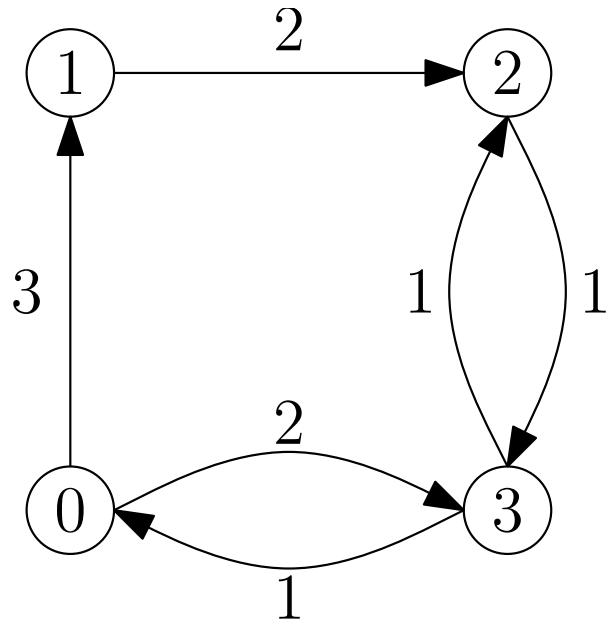


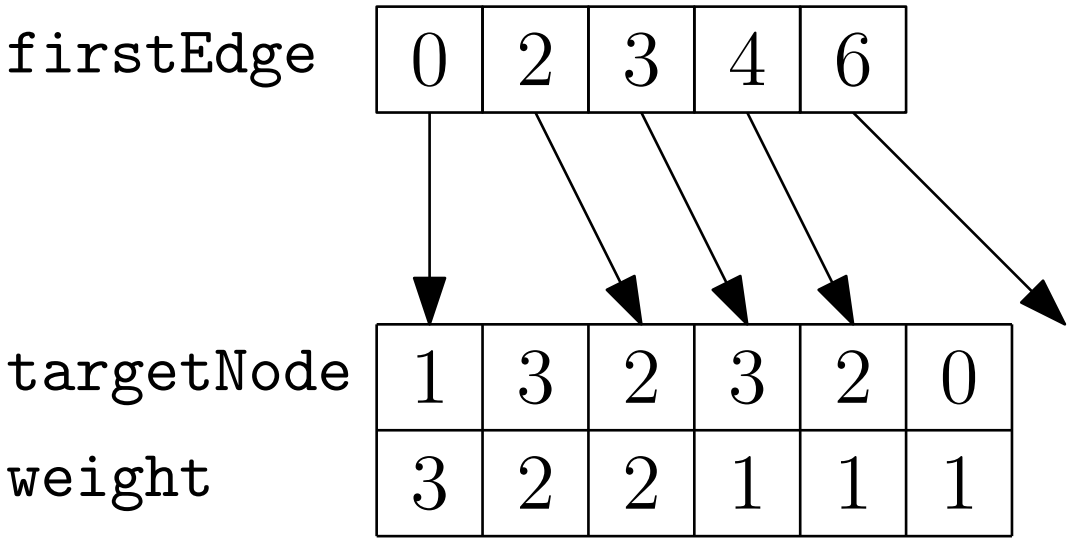


Graph-Repräsentationen

Drei klassische Ansätze:

- Adjazenzmatrix
- Adjazenzlisten
- Adjazenzarray





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 nich 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:

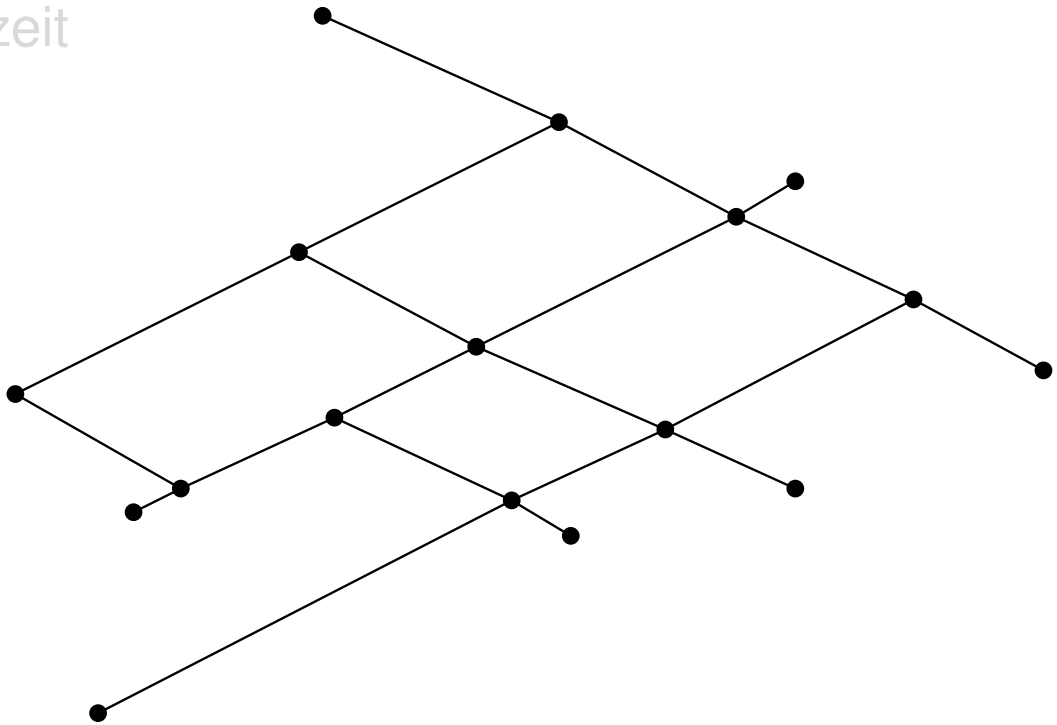
- Was brauchen wir?
- Was muss nich supereffizient sein?
- erstmal Modelle anschauen!

Modellierung (Straßengraphen)

Modellierung (Straßengraphen)

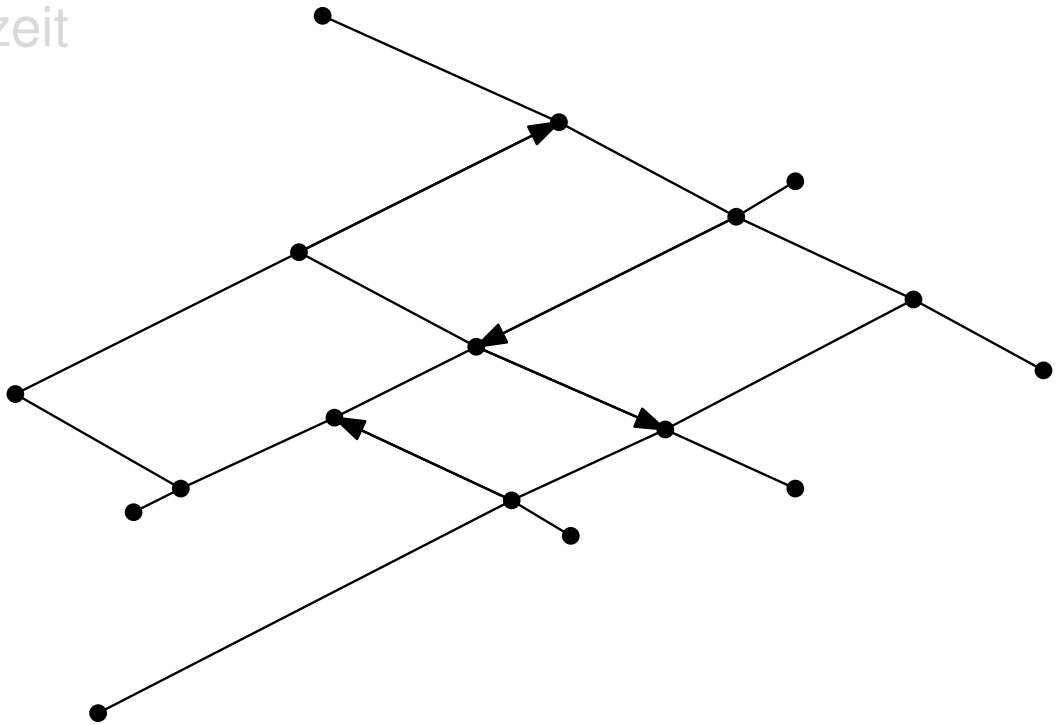
Modellierung (Straßengraphen)

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



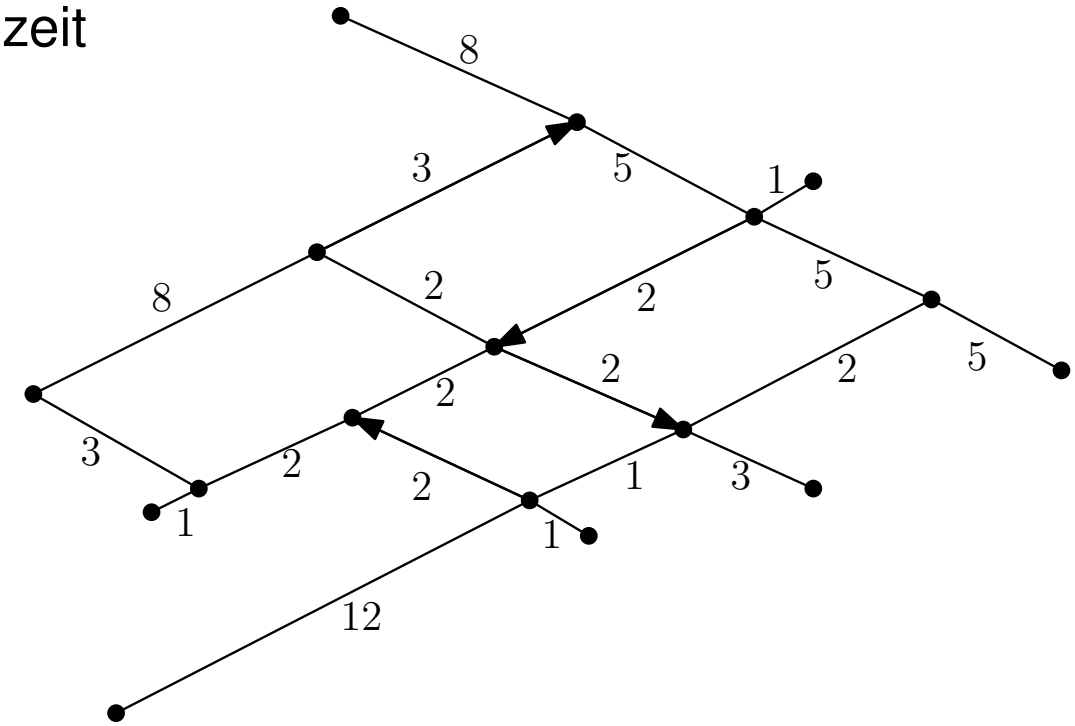
Modellierung (Straßengraphen)

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



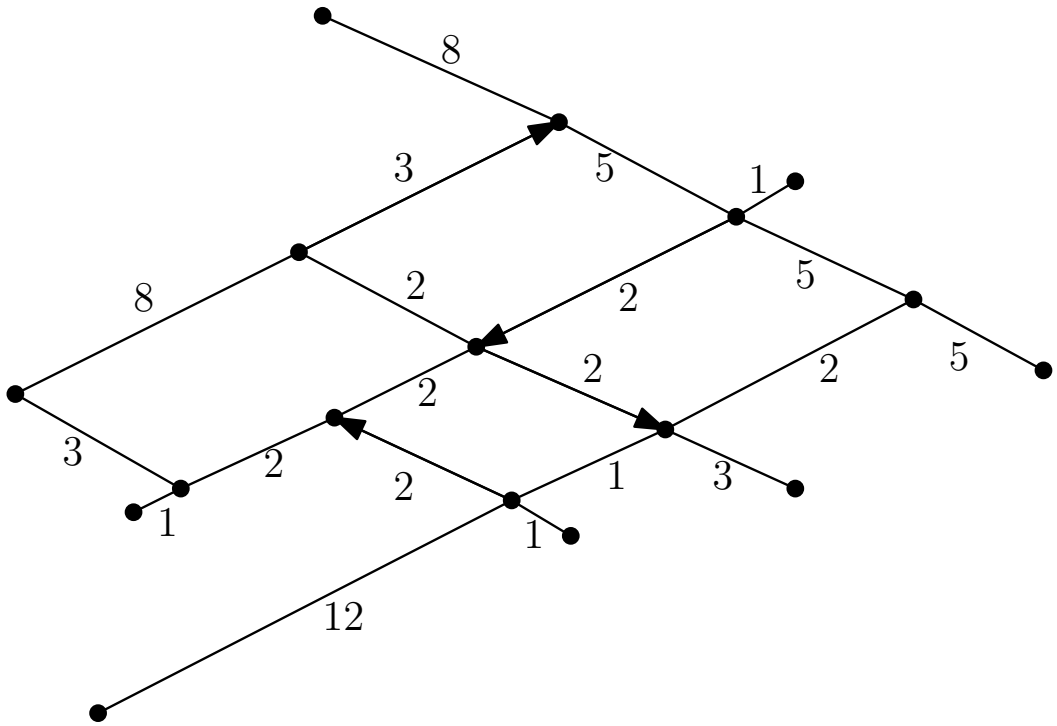
Modellierung (Straßengraphen)

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



Modellierung (Straßengraphen)

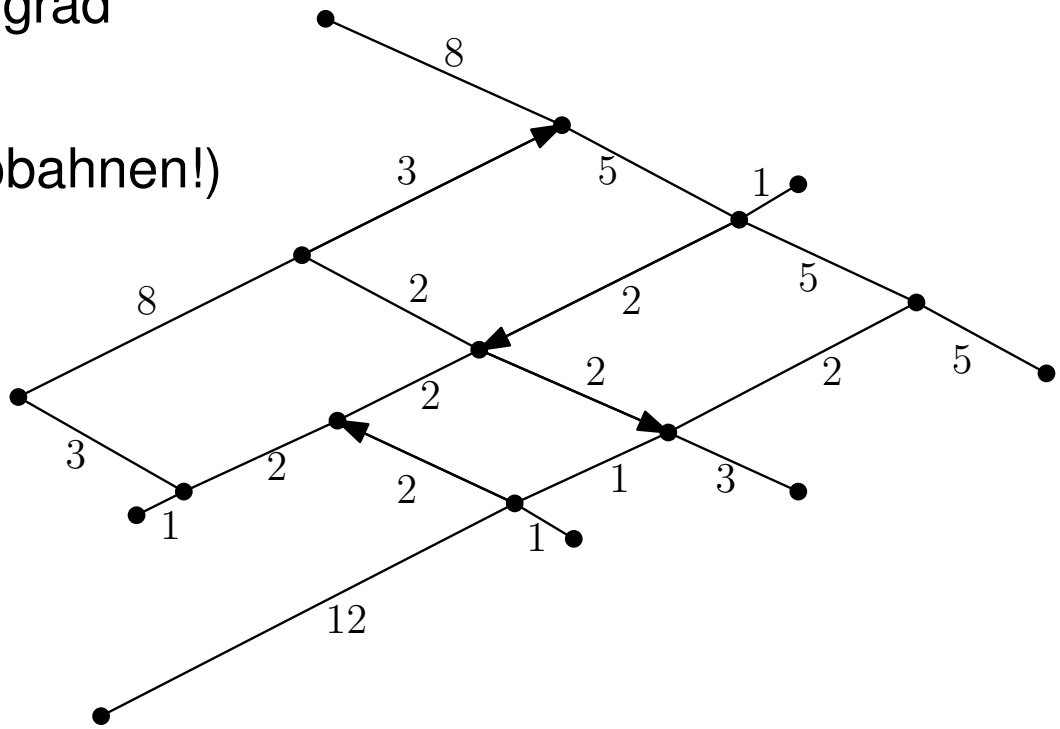
Eigenschaften (sammeln):



Modellierung (Straßengraphen)

Eigenschaften:

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



Eigenschaften der Graphen

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

Ablauf des Praktikums

Ablauf

1. Zuteilung der Themen an die Gruppen (heute)
2. Einlesen und Bearbeiten v. erstem Aufgabenblatt (alle Gruppen)
3. Bearbeiten zweites Aufgabenblatt (alle Gruppen)
4. Kurzvorträge, ca. 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

Zeitplan

Terminfindung

Woche	Num.	Aufgabe bis hierhin
31.10.–04.11.	1	Vorbesprechung
07.11.–10.11.	2	Abgabe Blatt 1 / Einlesen
21.11.–25.11.	4	Abgabe Blatt 2
05.12.–09.12.	6	Zwischenvorträge (je 5–10 Minuten)
ca. 13.02.–17.02.	ca. 15	Endvorträge + Demo (je 20 Minuten)
Ende Februar		Draft-Version von Ausarbeitung
ca. Semesterende		Abgabe Ausarbeitung

An diesen Terminen ist Anwesenheitspflicht.

Zeitplan

Terminfindung

Woche	Num.	Aufgabe bis hierhin
31.10.–04.11.	1	Vorbesprechung
07.11.–10.11.	2	Abgabe Blatt 1 / Einlesen
21.11.–25.11.	4	Abgabe Blatt 2
05.12.–09.12.	6	Zwischenvorträge (je 5–10 Minuten)
ca. 13.02.–17.02.	ca. 15	Endvorträge + Demo (je 20 Minuten)
Ende Februar		Draft-Version von Ausarbeitung
ca. Semesterende		Abgabe Ausarbeitung

An diesen Terminen ist Anwesenheitspflicht.

Themenübersicht

Customizable Route Planning

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 ($\leq 10\,\text{ms}$)

Customizable Route Planning

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.

Customizable Route Planning

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.

PHAST

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?

PHAST

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
- 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.

PHAST

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
- 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.

HLDB

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.

HLDB

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.

RAPTOR

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, . . .

RAPTOR

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.
Unpublished.

RAPTOR

Karlsruher Institut für Technologie

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.
Unpublished.

Zuteilung der Themen

Bitte gruppiert euch in 2er Gruppen und wählt.

Themen

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

Accounts

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.

Schnell, schneller, am schnellsten...

Karlsruher Institut für Technologie

Maxime

Bei der Routenplanung müssen

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

$\Rightarrow$ Wir benutzen C++ mit einem (sehr) schlanken Framework!

Slim-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)

Überblick

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.

Graphen

Umgehen mit Graphen...

Grundlegende Datentypen

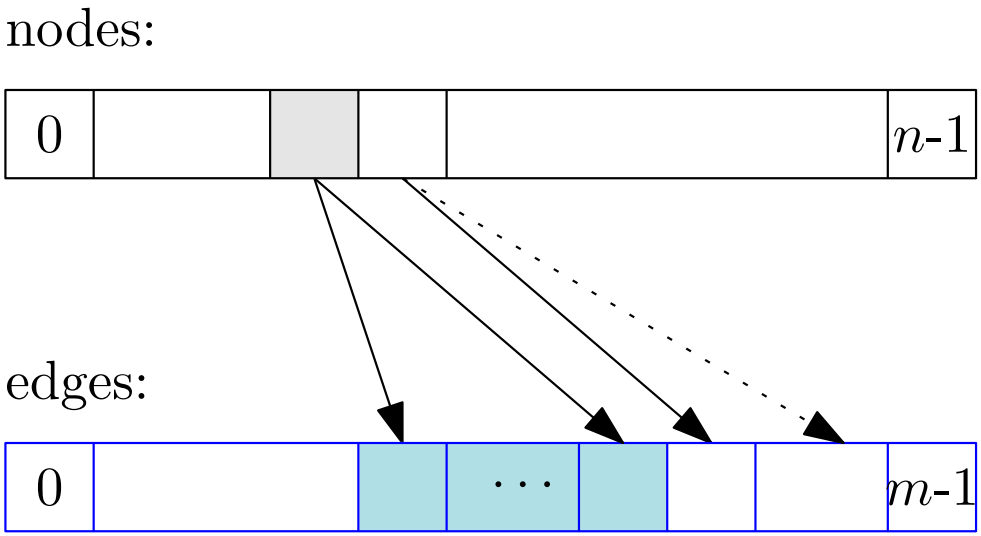
Grundlegende Datentypen aus unserem Framework sind

- `typedef unsigned int types::NodeID;`
  Index in dem Knotenvektor `nodes`
- `typedef unsigned int types::EdgeID;`
  Index in dem Kantenvektor `edges`
- `typedef int types::EdgeWeight;`
  Kantengewichte

- `types::INFTY`
  Spezieller `EdgeWeight`-Wert für ∞.

Graphen

Die Graphen sind als Adjazenzarray implementiert.

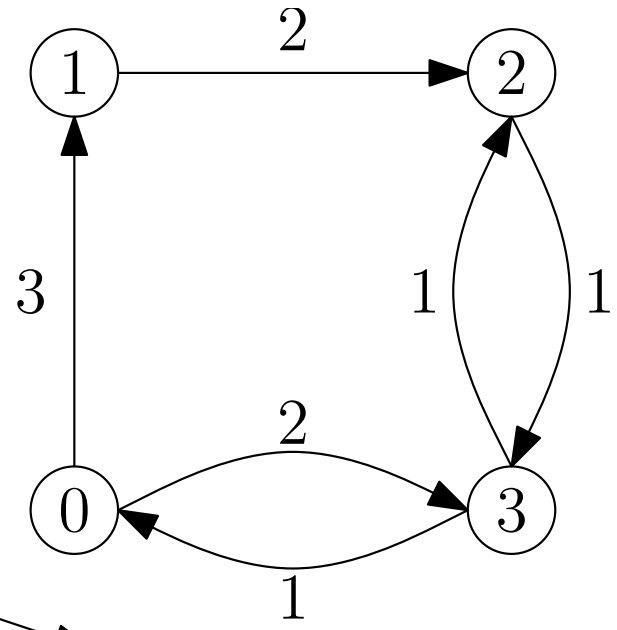



```
struct basicNode {
  EdgeID first_edge();
  EdgeId last_edge();
}

struct indEdge {
  NodeID node();
  bool forward();
  bool backward();
  EdgeWeight weight();
}
```

Graphen – Beispiel

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
`backwardFlag`	–	–	√	√	–	√	√	–	√	√
`forwardFlag`	√	√	–	–	√	–	√	√	–	√

Graph-Struktur – Implementierung

Wichtige Bestandteile der Graph-Klasse:

```
template<typename PropertiesType, typename NodeType,
typename EdgeType, typename ConnectionType>
class dynamicGraph {
    // ...
    vector<NodeType> nodes;
    vector<EdgeType> edges;
};
```

Der einfache Graph ergibt sich damit durch

```
typedef dynamicGraph<basicProperties, basicNode,
 indEdge, basicConnection> roadGraph;
```

Der Rest ist für uns (erstmal) nicht wichtig!

Größe des Graphen

Folgende Funktionen können benutzt werden um Größen zu ermitteln

- `get_num_nodes()`
  Größe des `nodes`-vectors -1.
- `get_num_edges()`
  Größe des `edges`-vectors

Achtung: `get_num_edges()` entspricht in der Regel nicht der tatsächlichen Anzahl Kanten im Graph! `get_num_arcs()` liefert die Anzahl Vorwärtskanten.

Iterieren auf Graphen

Folgende Konstrukte existieren zur Iteration:

- `FORALL_NODES(G, n)`
  Iteriert über alle Knoten in `G`
- `graphs::roadGraph::edgeType *e(NULL);`
  `FORALL_EDGES(G, u, e)`
  Iteriert über *alle* Kanten, `u` ist dabei der Quellknoten der Kante
- `graphs::roadGraph::edgeType *e(NULL);`
  `FORALL_INCIDENT_EDGES(G, u, e)`
  Iteriert über *alle* zu `u` inzidenten Kanten in `G`

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

Vorwärts und Rückwärtskanten

Problem: Nur über *ausgehende* Kanten iterieren.

```
graphs::roadGraph G;      types::NodeID u = 42;

// Iteriere ueber ausgehende Kanten von u...
graphs::roadGraph::edgeType *e(NULL);
FORALL_INCIDENT_EDGES(G, u, e) {

  if (!e->forward())
    continue;

  std::cout << "Kante mit Gewicht "
            << e->weight() << std::endl;
}
```

Nur Rückwärtskanten: analog

Viel Erfolg!