Kapitel 5
Anfragemethoden auf Verkehrsnetzwerke

Aus dem Skript zur Vorlesung: Neue Trends zur Suche in modernen Datenbanksystemen
Wintersemester 2013/14, LMU München

5.1 Motivation

- Naheliegend: Distanz zwischen zwei räumlichen Punkten über Euklidische Distanz berechnen (Luftlinie).
- In vielen Anwendungen, z.B. im Straßenverkehr, existieren Einschränkungen bzgl. möglicher Bewegungsrichtungen.
- In Verkehrsnetzwerken wird als Distanz zwischen zwei Punkten i.d.R. die Länge des kürzesten Pfades (Netzwerkdistanz) zwischen den Punkten angenommen
- (Beispiel: Distanzangaben bei der Navigation)
- Herausforderung:

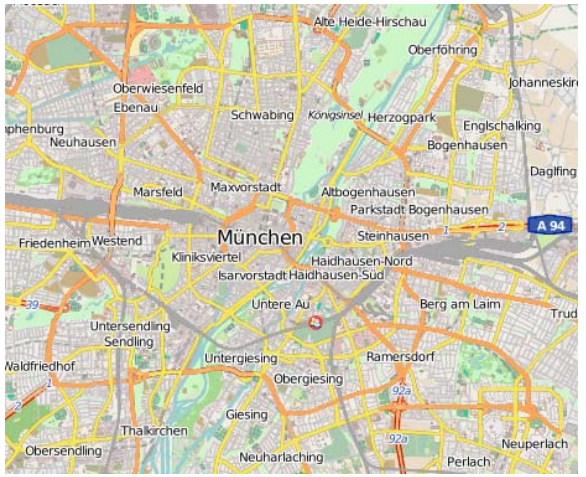
  Berechnung der Distanz

  (bzw. Nachbarschaft/Ähnlichkeit)

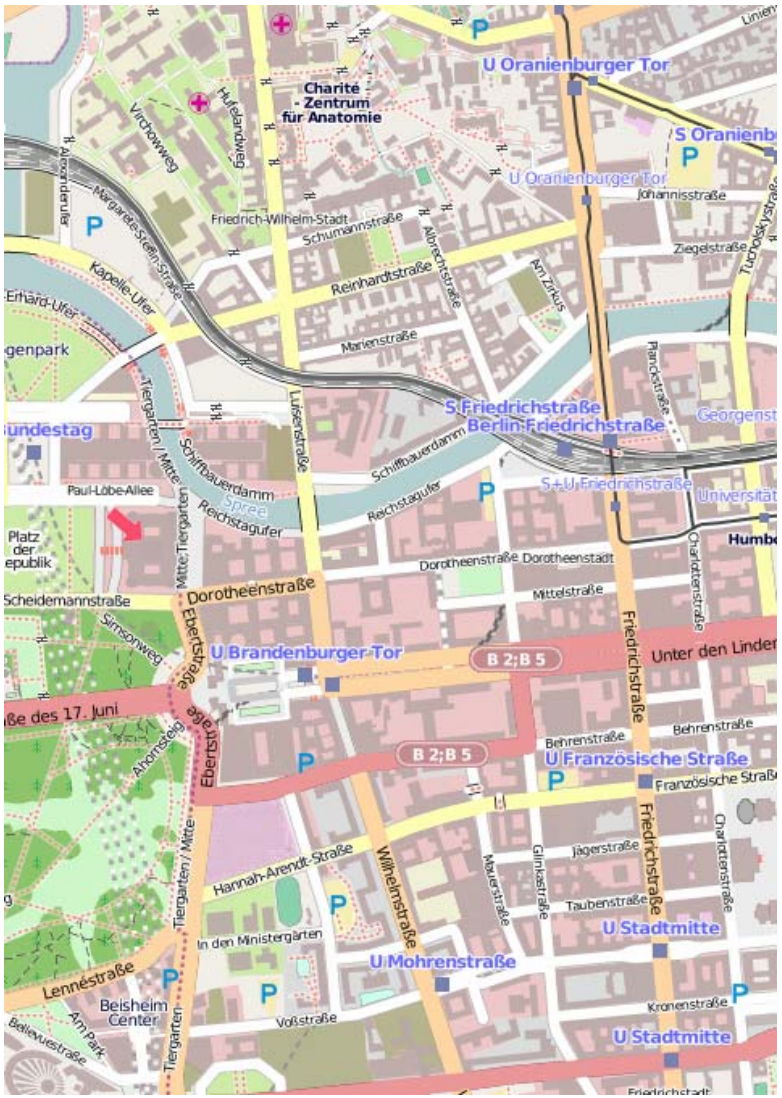
  ist sehr teuer

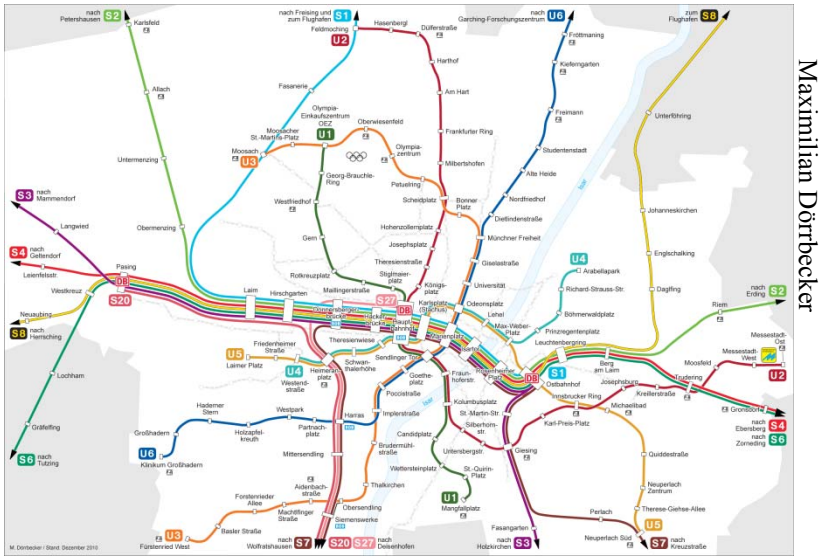
  => Minimierung der

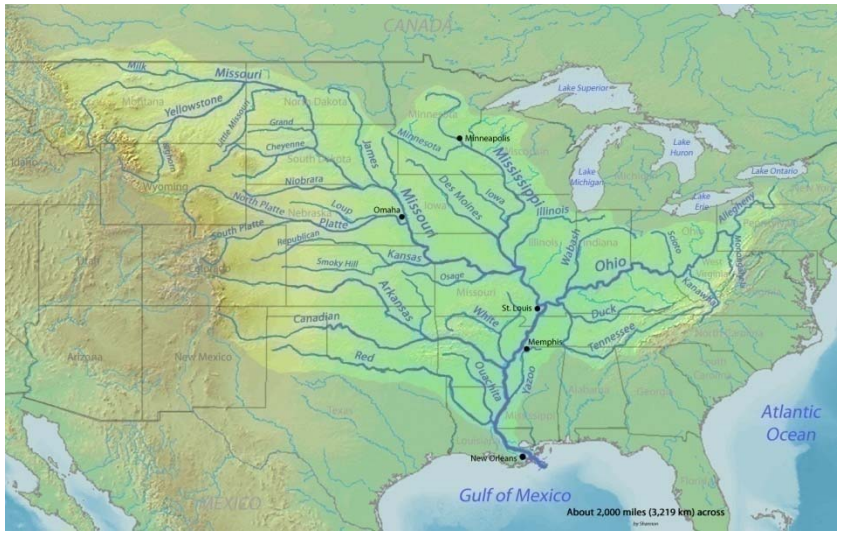
  Distanzvergleiche



Straßen- , Fluss- und Schienennetzerke







Maximilian Dörrbecker

– Beispiel-Anfragen

- Schienennetzwerke:
  - "Finde die Stationen auf der ICE-Strecke von München nach Berlin.
  - "Finde alle Stationen, die direkt vom Marienplatz erreicht werden können."
  - "Finde die Linien, die Starnberg und Universität verbinden."
  - "Was ist der vorletzte Halt der U-Bahn nach Messestadt-West?"

- Flussnetzwerke:
  - "Was sind die Namen aller direkten und indirekten Zuflüsse der Donau?"
  - "Was sind alle direkten Zuflüsse des Rheins?"
  - "Welche Gewässer wären von einem Chemieunfall in der Breg betroffen?"

- Straßennetzwerke:
  - "Finde den kürzesten Pfad von der LMU zur HU Berlin."
  - "Finde das näheste Computergeschäft nach zu laufender Strecke."
  - "Finde den kürzesten Pfad, um mehrere Shops zu beliefern."
  - "Verweise Kunden auf den nähesten Kundendienst."

5.2 Modellierung von Verkehrsnetzwerken

Drei-Ebenen-Modell:

1. Konzeptionelles Datenmodell: Graphen

2. Logisches Datenmodell (beschreibt das DB-Schema)
   - Datentypen
     - Graph, Vertex, Edge, Path, …
   - Operationen (Queries)
     - is connected(.,.), shortest-path(), weitere Nachbarschaftsanfragen, ...

3. Physisches Datenmodell (beschreibt wie die Daten abgespeichert werden)
   - Hauptspeicherbasierte Darstellung
   - Festplattenbasierte Darstellung

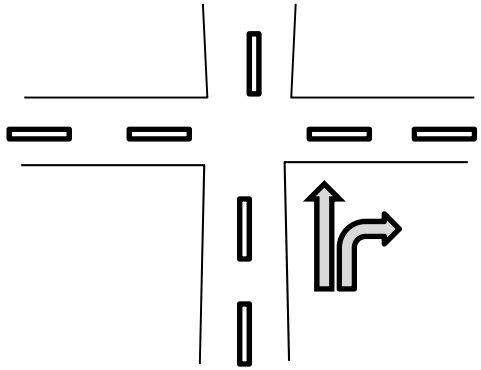
5.2.1 Konzeptuelles Modell

- Straßennetze können als Graph G=(V,E), bestehend aus einer Menge von Knoten V sowie einer Menge von (gerichteten) Kanten $E \subset \{(v_i, v_j) \mid v_i, v_j \in V\}$ modelliert werden

- Repräsentation 1: (üblich)
  - *Knoten $v_i \in V$: Kreuzungen, Ende einer Straße, weitere relevante Punkte (z.B. Änderung der Geschwindigkeitsbeschränkung)*
  - *(Gerichtete) Kante $e_i \in E$: Straßenstück zwischen zwei Knoten, modelliert topologische Information*

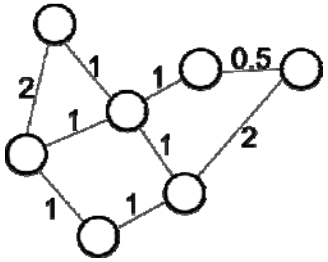
- *Repräsentation 2:*
  - *Knoten $v_i \in V$: Straßen*
  - *Kanten $e_i \in E$: Kreuzungen zwischen Straßen*

Wann sinnvoll?

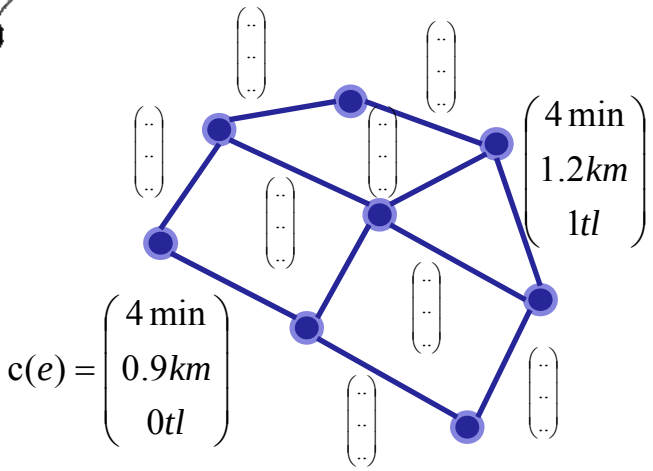


Mit Rep. 1 möglich?

- Kanten haben Gewichte (Kosten), z. B.
  - Reellwertig
    - Entfernung zwischen zwei Knoten
    - Fahrtdauer
    - Benzinverbrauch
    - gewichtete Kombination mehrerer Merkmale



  - Vektorielle Kombination mehrerer reellwertiger Gewichte [KRS10]
    - Zusammenfassung verschiedener Gewichte zu einem Vektor



  - Funktionen über die Zeit
    - Z.B. die Fahrtdauer is abhängig von der Tageszeit

5.2.2 Logisches Datenmodell – Datentypen

– Beschreibt das Schema einer Räumlichen Netzwerk Datenbank (Spatial Network Database SNDB)

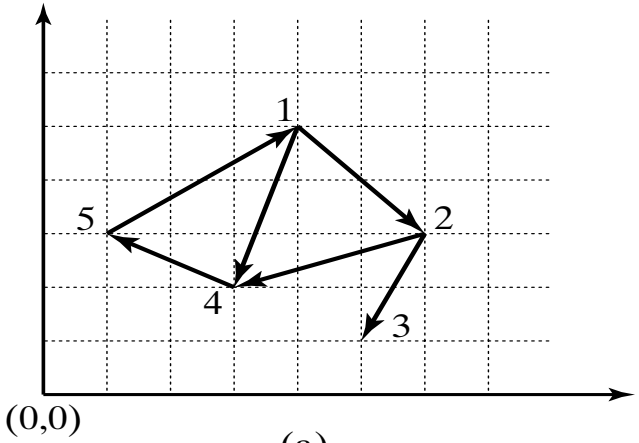
- Vertex (Knoten), Attribute:
  - label
  - isVisited
  - location (Koordinaten => räumliche Graphen)
- DirectedEdge (gerichtete Kante), Attribute:
  - startNode
  - endNode
  - label
- Graph, Attribute:
  - Set<Vertex>
  - Set<DirectedEdge>
- Path: Attribute
  - Sequence<Vertex>

5.2.3 Physisches Datenmodell

- Hauptspeicherbasiert:
  - Adjazenzmatrix: M[A, B] = 1 gdw. edge(vertex A, vertex B) existiert
  - Adjazenzlisten: Bildet vertex A auf eine Liste von Nachfolgern von A ab
  - Beispiele: Abbildung (a), (b) und (c) auf der nächsten Folie

- Festplattenbasiert
  - Normalisiert -- Tabellen, eine für Knoten, die andere für Kanten
  - Denormalisiert – Tabelle für Knoten + Adjazenzlisten
  - Beispiele: Siehe Abbildung (a), (d) und (e) auf der nächsten Folie

– Beispiel



(a)

Destination

source	1	2	3	4	5
1	0	1	0	1	0
2	0	0	1	1	0
3	0	0	0	0	0
4	0	0	0	0	1
5	1	0	0	0	0

(b) Adjacency-matrix

1	→	2	4
2	→	3	4
3	→	N	
4	→	5	
5	→	1	

(c) Adjacency-List

Node (R)

id	x	y
1	4.0	5.0
2	6.0	3.0
3	5.0	1.0
4	3.0	2.0
5	1.0	3.0

Edge (S)

source	dest	distance
1	2	$\sqrt{8}$
1	4	$\sqrt{10}$
2	3	$\sqrt{5}$
2	4	$\sqrt{10}$
4	5	$\sqrt{5}$
5	1	$\sqrt{18}$

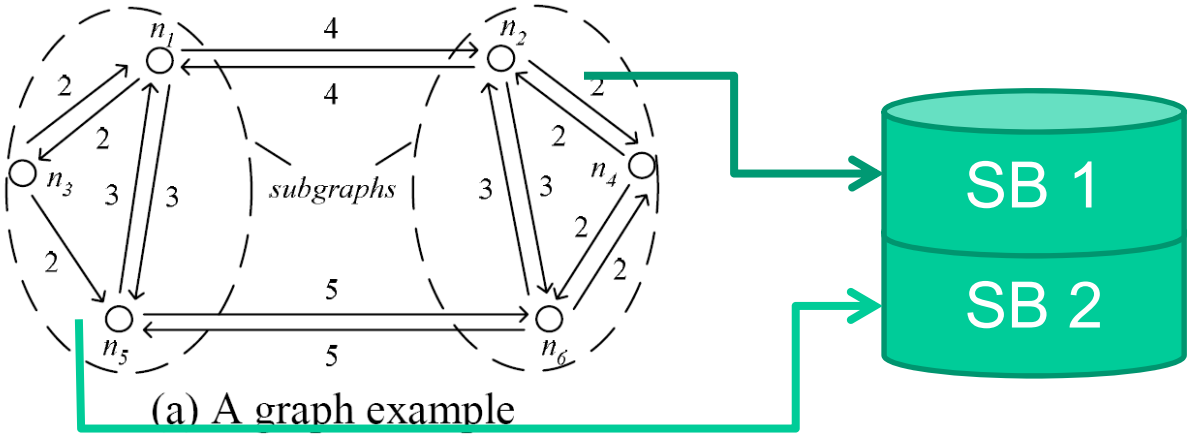
(d) Node and Edge Relations

id	x	y	Successors	Predecessors
1	4.0	5.0	(2,4)	(5)
2	6.0	3.0	(3,4)	(1)
3	5.0	1.0	( )	(2)
4	3.0	2.0	(5)	(1,2)
5	1.0	3.0	(1)	(4)

(e) Denormalized Node Table

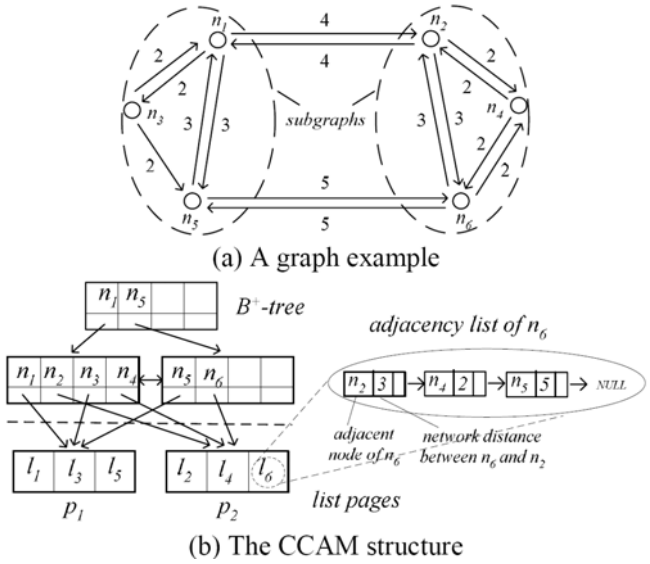
5.2.4 Effiziente Organisation des Straßen-Netzwerkgraphen

- Adjazenzlisten geeignet für Räumliche Netwerkgraphen
- Ziel: Adaptierung der Adjazenzliste für Sekundärspeicherorganisation (Festplatte)
  - Effizienter Zugriff auf räumlich benachbarte Netzwerkelemente
  - Minimierung der I/O Kosten bei Anfragen
- Idee: Gruppiere Listen von adjazenten Knoten in gleichen Speicherblöcken (Seiten)



– Connectivity-Clustered Access Method (CCAM)

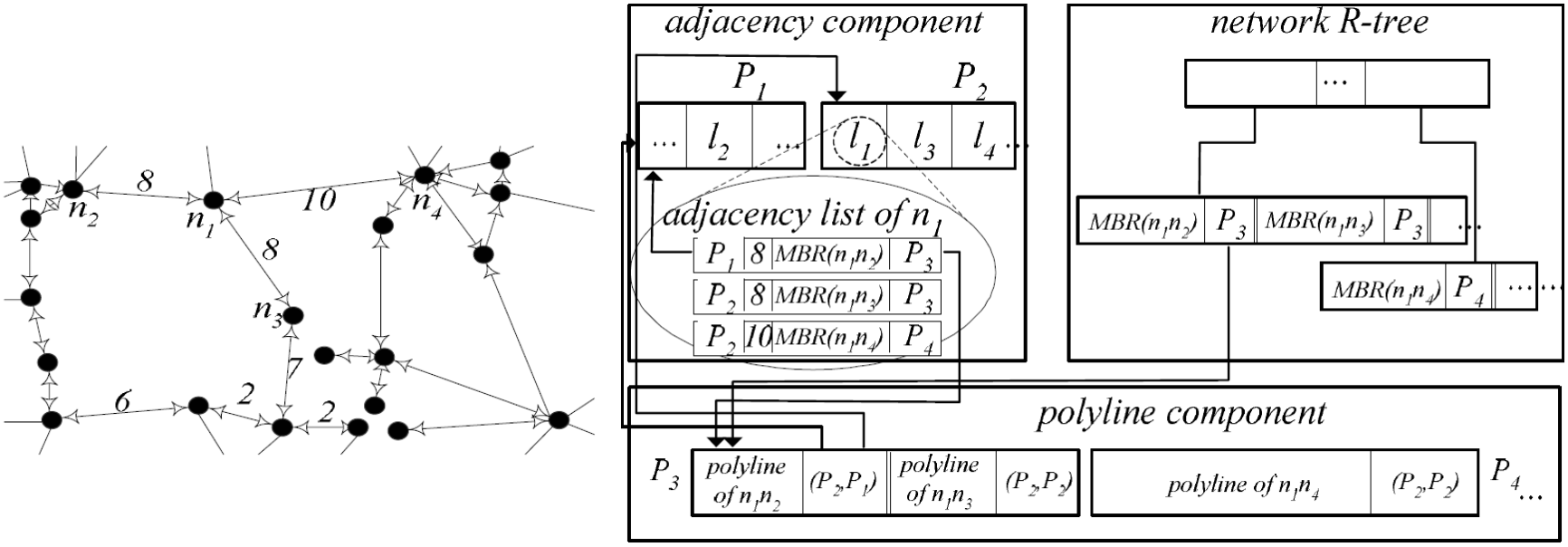
- Einbettung der Netzwerkknoten in den ein-dimensionalen Raum
  - Einbettung über raumfüllende Kurve, z.B. Z-Kurve
  - Z-Kurve erhält räumliche Nachbarschaft, d.h. räumlich nah beieinander liegende Knoten im Originalraum liegen auch nahe beieinander im eingebetteten Raum
- Adjazenzlisten nah beieinanderliegender Knoten werden in einer Seite (*list page*) abgespeichert
- List pages werden über einen B+-Baum auf den entsprechenden Knoten-Ids indexiert
- Eigenschaften:
  - Unterstützt (theoretisch) Anfragen bzgl. der topologischen Verbundenheit im Netzwerkgraph, z.B. shortest path
  - (Klassische) räumliche Anfragen mit Bezug zum Euklidischen Raum nicht gut unterstützt (z.B. bzgl. Polyline-Kanten)



(a) A graph example

(b) The CCAM structure

– SNDB-Index-Architektur [PZMT03]
  - Integriert Informationen über Raum und Verbundenheit
  - Unterstützt (konventionelle) räumliche Anfragen als auch netzwerktopologische Anfragen
  - 3-Komponenten-Architektur:
    – Adjazenz-Komponente (adjacency component)
    – Räumliche Komponente (network R-tree)
    – Polyline Komponente (polyline component)



- Folgende Funktionen werden im Folgenden benötigt. Sie basieren auf den vorgestellten Datenstrukturen, sollen aber nicht näher erläutert werden.
  - *check_entity(segment, point)*: gibt *true* zurück, falls *point* auf *segment* liegt
  - *find_segment(point)*: gibt das Segment zurück, auf dem *point* liegt. Liegt *point* auf mehreren Segmenten, wird das zuerst gefundene Segment zurückgegeben
  - *find_entities(segment)*: Gibt alle Punkte zurück, die auf *segment* liegen
  - *compute_ND(point1, point2)*: Berechnet Netzwerkdistanz zweier beliebiger Punkte *point1* und *point2*