

Fortgeschrittene Netzwerk- und Graph-Algorithmen

Dr. Hanjo Täubig

Lehrstuhl für Effiziente Algorithmen
(Prof. Dr. Ernst W. Mayr)
Institut für Informatik
Technische Universität München

Wintersemester 2012/13



Übersicht

- 1 Grundlagen
- 2 Zentralitätsindizes
- 3 Wiederholung: Kürzeste Wege
- 4 Algorithmen für Zentralitätsindizes
- 5 Lokale Dichte
- 6 Zusammenhang

Vorlesungsdaten

- Modul: IN2158
- Bereich:
Informatik III (Theoretische Informatik)
Bioinformatik (Bereich Informatik)
- Semesterwochenstunden:
4 SWS Vorlesung + 2 SWS Übung
- ECTS: 8 Punkte
- Vorlesungszeiten:
Dienstag 16:15 – 17:45 Uhr (MI 00.08.038)
Donnerstag 12:15 – 13:45 Uhr (MI 00.13.009A)
- Übung:
Dienstag 12:15 – 13:45 Uhr (MI 03.10.011)

Dozent

- Hanjo Täubig
(Lehrstuhl für Effiziente Algorithmen / Prof. Mayr)
- eMail: taeubig@in.tum.de
- Web: <http://www14.in.tum.de/personen/taeubig/>
- Telefon: 089 / 289-17740
- Raum: 03.09.039
- Sprechstunde: Mittwoch 13–14 Uhr
(oder nach Vereinbarung)

Voraussetzungen

Stoff des Informatik Grundstudiums

- Diskrete Strukturen
- Grundlagen: Algorithmen und Datenstrukturen
- Einführung in die Theoretische Informatik

vorteilhaft:

- Effiziente Algorithmen und Datenstrukturen I/II

Inhalt

Schwerpunkte:

- Zentralitätsindizes und zugehörige Algorithmen
- Dichte in (Teil-)Graphen
- Fortgeschrittene Algorithmen für Zusammenhangsprobleme
- Assignment Problem / Ungarische Methode

Weitere mögliche Themengebiete:

- Graphfärbung
- Clustering
- Netzwerk-Statistik
- Netzwerk-Vergleich
- Algebraische Methoden
- Spektrale Analyse
- Robustheit

Literatur

Die Vorlesung orientiert sich an folgendem Buch:

U. Brandes, Th. Erlebach (Eds.):

Network Analysis – Methodological Foundations

Lecture Notes in Computer Science Vol. 3418,

Springer, 2005.

Webzugriff:

<http://www.springerlink.com/openurl.asp?genre=issue&issn=0302-9743&volume=3418>

(Automatische Proxy-Konfiguration: <http://pac.lrz-muenchen.de/>)

Weitere Literatur

- K. Mehlhorn, P. Sanders:
Algorithms and Data Structures: The Basic Toolbox
- J. Bang-Jensen, G. Gutin:
Digraphs: Theory, Algorithms and Applications
- V. Turau:
Algorithmische Graphentheorie
- J. Aldous, R. Wilson:
Graphs and Applications – An Introductory Approach
- A. Dolan, J. Aldous:
Networks and Algorithms – An Introductory Approach

Übersicht

- 1 Grundlagen
 - Netzwerke und Graphen
 - Graphrepräsentation
 - Wiederholung bekannter Algorithmen
- 2 Zentralitätsindizes
- 3 Wiederholung: Kürzeste Wege
- 4 Algorithmen für Zentralitätsindizes
- 5 Lokale Dichte
- 6 Zusammenhang

Netzwerk

Objekt bestehend aus

- **Elementen** und
- Interaktionen bzw. **Verbindungen** zwischen den Elementen

eher *informales* Konzept

- keine exakte Definition
- Elemente und insbesondere ihre Verbindungen können ganz unterschiedlichen Charakter haben
- manchmal manifestiert in **real** existierenden Dingen, manchmal nur gedacht (**virtuell**)

Beispiele für Netzwerke

- Kommunikationsnetze: Internet, Telefonnetz
- Verkehrsnetze: Straßen-, Schienen-, Flug-, Nahverkehrsnetz
- Versorgungsnetzwerke: Strom, Wasser, Gas, Erdöl
- wirtschaftliche Netzwerke: Geld- und Warenströme, Handel
- biochemische Netzwerke: Metabolische und Interaktionsnetzwerke
- biologische Netzwerke: Gehirn, Ökosysteme
- soziale / berufliche Netzwerke: virtuell oder explizit (Communities)
- Publikationsnetzwerke: Zitationsnetzwerk, Koautor-Netzwerk

Graph

formales / abstraktes Objekt bestehend aus

- Menge von **Knoten** V (engl. vertices, nodes)
- Menge von **Kanten** E (engl. edges, lines, links), die jeweils ein Paar von Knoten verbinden
- Menge von Eigenschaften der Knoten und / oder Kanten

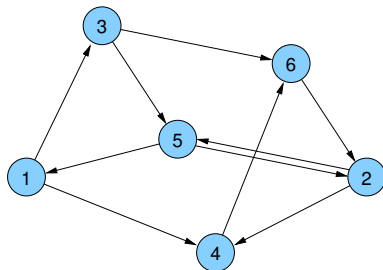
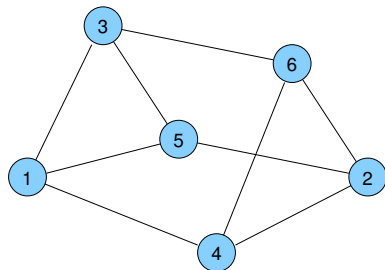
Notation:

- $G = (V, E)$
manchmal auch $G = (V, E, w)$ im Fall gewichteter Graphen
- Anzahl der Knoten: $n = |V|$
Anzahl der Kanten: $m = |E|$

Gerichtete und ungerichtete Graphen

Kanten bzw. Graphen

- ungerichtet: $E \subseteq \{\{v, w\} : v \in V, w \in V\}$
(ungeordnetes Paar von Knoten bzw. 2-elementige Teilmenge)
- gerichtet: $E \subseteq \{(v, w) : v \in V, w \in V\}$, also $E \subseteq V \times V$
(geordnetes Paar von Knoten)



Gerichtete und ungerichtete Graphen

Anwendungen:

- Ungerichtete Graphen:

symmetrische Beziehungen (z.B. $\{v, w\} \in E$ genau dann, wenn Person v und Person w verwandt sind)

- Gerichtete Graphen:

asymmetrische Beziehungen (z.B. $(v, w) \in E$ genau dann, wenn Person v Person w mag)

kreisfreie Beziehungen (z.B. $(v, w) \in E$ genau dann, wenn Person v Vorgesetzter von Person w ist)

Modellierung von ungerichteten durch gerichtete Graphen:

- Ersetzung jeder ungerichteten Kante durch je zwei antiparallele gerichtete Kanten

Nachbarn: Adjazenz, Inzidenz, Grad

Sind zwei Knoten v und w durch eine Kante e verbunden, dann nennt man

- v und w **adjazent** bzw. benachbart
- v und e **inzident** (ebenso w und e)

Anzahl der Nachbarn eines Knotens v : **Grad** $\deg(v)$

bei gerichteten Graphen:

- Eingangsgrad: $\deg^-(v) = |\{(w, v) \in E\}|$
- Ausgangsgrad: $\deg^+(v) = |\{(v, w) \in E\}|$

Annahmen

- Graph (also Anzahl der Knoten und Kanten) ist **endlich**
- Graph ist **einfach**, d.h. E ist eine Menge und keine Multimenge (anderenfalls heißt G Multigraph)
- Graph enthält **keine Schleifen** (Kanten von v nach v)

Gewichtete Graphen

In Abhängigkeit vom betrachteten Problem wird Kanten und / oder Knoten oft eine Eigenschaft (z.B. eine Farbe oder ein numerischer Wert, das **Gewicht**) zugeordnet (evt. auch mehrere), z.B.

- Distanzen (in Längen- oder Zeiteinheiten)
- Kosten
- Kapazitäten / Bandbreite
- Ähnlichkeiten
- Verkehrsdichte

Wir nennen den Graphen dann

- **knotengewichtet** bzw.
- **kantengewichtet**

Beispiel: $w : E \mapsto \mathbb{R}$

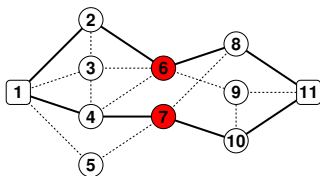
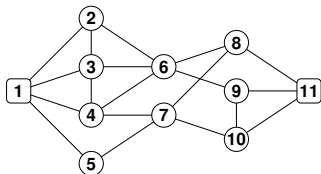
Schreibweise: $w(e)$ für das Gewicht einer Kante $e \in E$

Wege, Pfade und Kreise

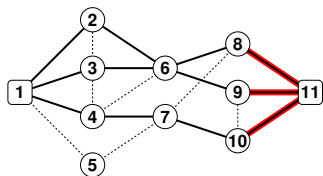
- **Weg** (engl. *walk*) in einem Graphen $G = (V, E)$:
alternierende Folge von Knoten und Kanten $x_0, e_1, \dots, e_k, x_k$, so dass
 - ▶ $\forall i \in [0, k] : x_i \in V$ und
 - ▶ $\forall i \in [1, k] : e_i = \{x_{i-1}, x_i\}$ bzw. $e_i = (x_{i-1}, x_i) \in E$.
- **Länge** eines Weges: Anzahl der enthaltenen Kanten
- **Pfad**: Weg, der (in sich) kantendisjunkt ist, also: $e_i \neq e_j$ für $i \neq j$
- **einfacher Pfad**: Pfad, der (in sich) knotendisjunkt ist, also: $x_i \neq x_j$ für $i \neq j$
- Ein Weg heißt **Kreis** (engl. *cycle*), falls $x_0 = x_k$.

Disjunkte s - t -Pfade

andere Bedeutung von *knoten-/kantendisjunkt* bei Betrachtung **mehrerer** Pfade zwischen gleichen Anfangs- und Endknoten



2 knotendisjunkte 1-11-Pfade



3 kantendisjunkte 1-11-Pfade

Graphrepräsentation

Darstellung von Graphen im Computer?

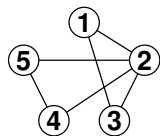
Vor- und Nachteile bei z.B. folgenden Fragen:

- Sind zwei gegebene Knoten v und w **adjazent**?
- Was sind die **Nachbarn** eines Knotens?
- Welche Knoten sind (direkte oder indirekte) **Vorgänger** bzw. **Nachfolger** eines Knotens v in einem gerichteten Graphen?
- Wie aufwendig ist das **Einfügen** oder **Löschen** eines Knotens bzw. einer Kante?

Graphrepräsentationen

- Kantenliste
- Adjazenzmatrix
- Inzidenzmatrix
- Adjazenzarray
- Adjazenzliste
- implizit

Kantenliste



$\{1, 2\}, \{1, 3\}, \{2, 3\}, \{2, 4\}, \{2, 5\}, \{4, 5\}$

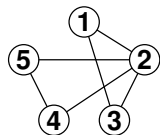
Vorteil:

- Speicherbedarf $\mathcal{O}(m + n)$
- Einfügen von Knoten und Kanten in $\mathcal{O}(1)$
- Löschen von Kanten per Handle in $\mathcal{O}(1)$

Nachteil:

- $G.\text{find}(\text{Key } i, \text{Key } j)$: im worst case $\Theta(m)$
- $G.\text{remove}(\text{Key } i, \text{Key } j)$: im worst case $\Theta(m)$
- Nachbarn nur in $\mathcal{O}(m)$ feststellbar

Adjazenzmatrix



$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

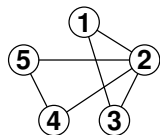
Vorteil:

- in $\mathcal{O}(1)$ feststellbar, ob zwei Knoten Nachbarn sind
- ebenso Einfügen und Löschen von Kanten

Nachteil:

- kostet $\Theta(n^2)$ Speicher, auch bei Graphen mit $o(n^2)$ Kanten
- Finden aller Nachbarn eines Knotens kostet $\mathcal{O}(n)$
- Hinzufügen neuer Knoten kostet $\Theta(n^2)$

Inzidenzmatrix

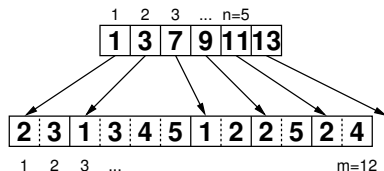
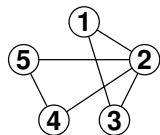


$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Nachteil:

- kostet $\Theta(mn)$ Speicher

Adjazenzarray



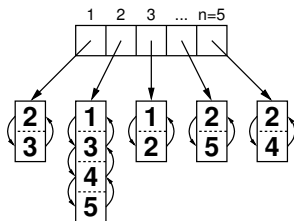
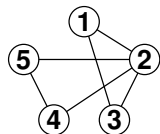
Vorteil:

- Speicherbedarf:
 gerichtete Graphen: $n + m + \Theta(1)$
 (hier noch kompakter als Kantenliste mit $2m$)
 ungerichtete Graphen: $n + 2m + \Theta(1)$

Nachteil:

- Einfügen und Löschen von Kanten ist schwierig,
 deshalb nur für *statische* Graphen geeignet

Adjazenzliste



Unterschiedliche Varianten:
einfach/doppelt verkettet, linear/zirkulär

Vorteil:

- Einfügen von Kanten in $\mathcal{O}(d)$ oder $\mathcal{O}(1)$
- Löschen von Kanten in $\mathcal{O}(d)$ (per Handle in $\mathcal{O}(1)$)
- mit unbounded arrays etwas cache-effizienter

Nachteil:

- Zeigerstrukturen verbrauchen relativ viel Platz und Zugriffszeit

Adjazenzliste + Hashtabelle

- speichere Adjazenzliste (Liste von adjazenten Nachbarn bzw. inzidenten Kanten zu jedem Knoten)
- speichere Hashtabelle, die zwei Knoten auf einen Zeiger abbildet, der dann auf die ggf. vorhandene Kante verweist

Zeitaufwand:

- $G.\text{find}(\text{Key } i, \text{Key } j)$: $\mathcal{O}(1)$ (worst case)
- $G.\text{insert}(\text{Edge } e)$: $\mathcal{O}(1)$ (im Mittel)
- $G.\text{remove}(\text{Key } i, \text{Key } j)$: $\mathcal{O}(1)$ (im Mittel)
- Speicheraufwand: $\mathcal{O}(n + m)$

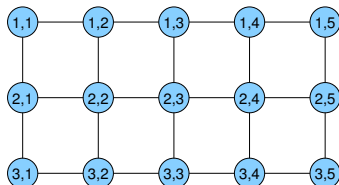
Implizite Repräsentation

Beispiel: **Gitter-Graph** (grid graph)

- definiert durch zwei Parameter k und ℓ

$$V = \{1, \dots, k\} \times \{1, \dots, \ell\}$$

$$E = \{((i, j), (i, j')) \in V^2 : |j - j'| = 1\} \cup \\ \{((i, j), (i', j)) \in V^2 : |i - i'| = 1\}$$



- Kantengewichte können in 2 zweidimensionalen Arrays gespeichert werden: eins für waagerechte und eins für senkrechte Kanten

Implizite Repräsentation

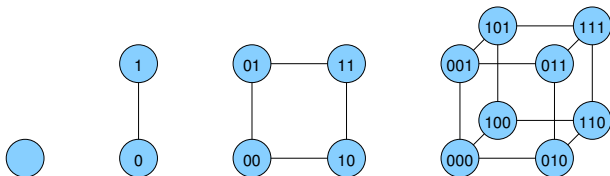
Beispiel: **Hyperwürfel** (hypercube)

- definiert durch einen Parameter d (Dimension)

$$V = \{0, \dots, 2^d - 1\} \quad n = 2^d$$

$$E = \{\{v, w\} : v \text{ und } w \text{ unterscheiden sich in genau einem Bit}\}$$

$$m = d \cdot 2^{d-1}$$



Weiteres Beispiel: Cube-Connected Cycles

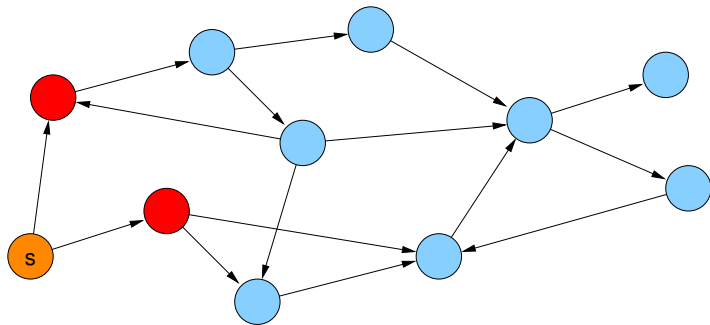
Graphtraversierung

Problem:

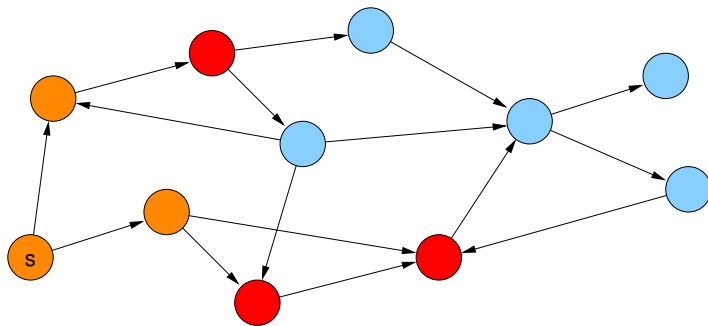
Wie kann man die Knoten eines Graphen **systematisch durchlaufen**?

- Breitensuche (breadth-first search, bfs)
 - ▶ schichtenweise Traversierung in aufsteigendem Abstand vom Ursprungsknoten
- Tiefensuche (depth-first search, dfs)
 - ▶ Topologische Sortierung (bei DAGs)
 - ▶ Zweifachzusammenhangskomponenten
 - ▶ Starke Zusammenhangskomponenten

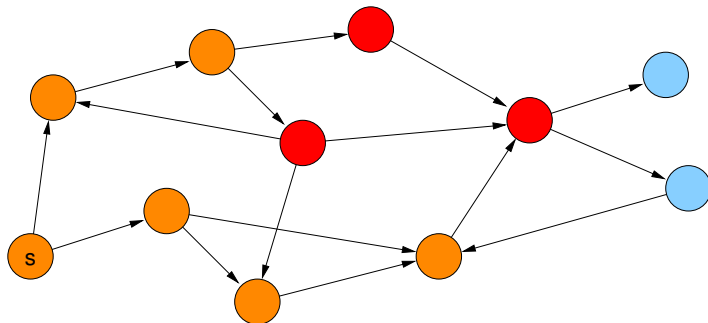
Breitensuche



Breitensuche

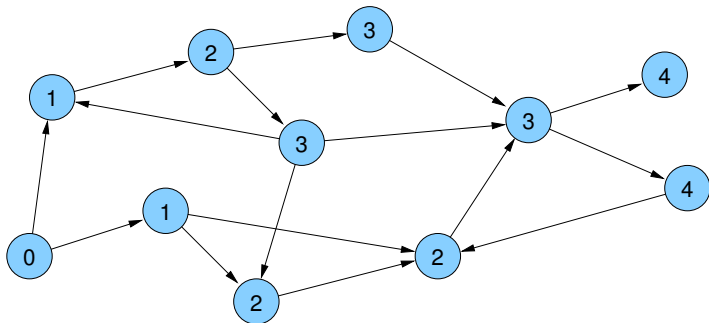


Breitensuche



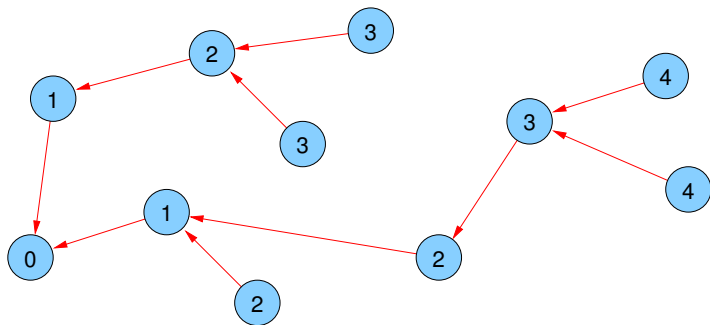
Breitensuche

- Anwendung: Single Source Shortest Path (SSSP) Problem in **ungewichteten** Graphen
- $d(v)$: Distanz von Knoten v zu s ($d(s) = 0$)



Breitensuche

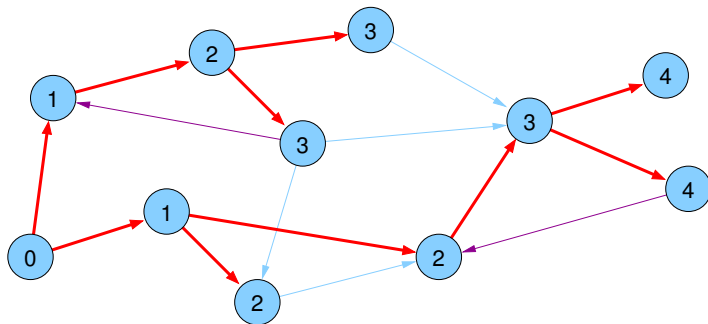
- **parent**(v): Knoten, von dem v entdeckt wurde
- **parent** wird beim ersten Besuch von v gesetzt (\Rightarrow eindeutig)



Breitensuche

Kantentypen:

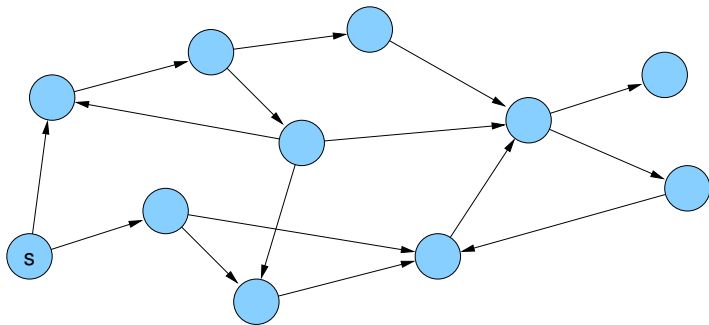
- **Baumkanten:** zum Kind
- **Rückwärtskanten:** zu einem Vorfahren
- **Kreuzkanten:** sonstige



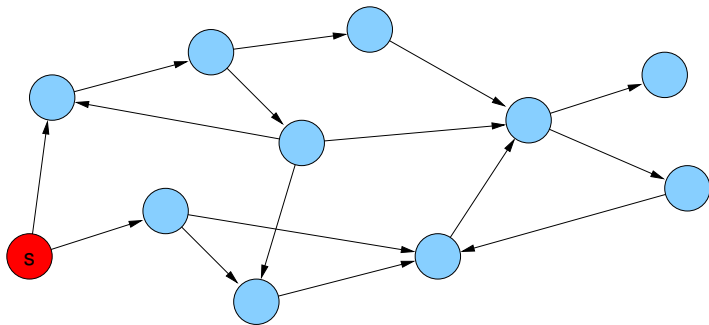
Breitensuche

```
void BFS(Node s) {
    d[s] = 0;
    parent[s] = s;
    List<Node> q = ⟨s⟩;
    while (!q.empty()) {
        u = q.popFront();
        foreach ((u, v) ∈ E) {
            if (parent[v] == null) {
                q.pushBack(v);
                d[v] = d[u]+1;
                parent[v] = u;
            }
        }
    }
}
```

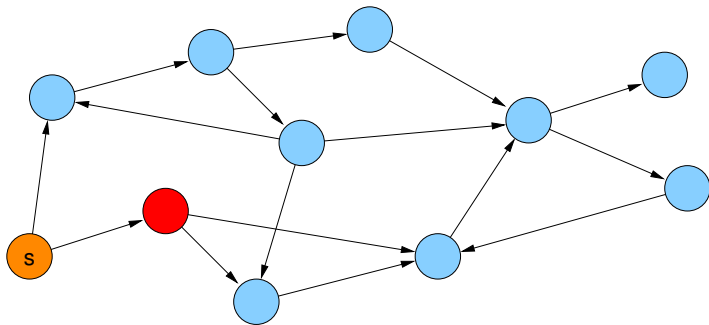
Tiefensuche



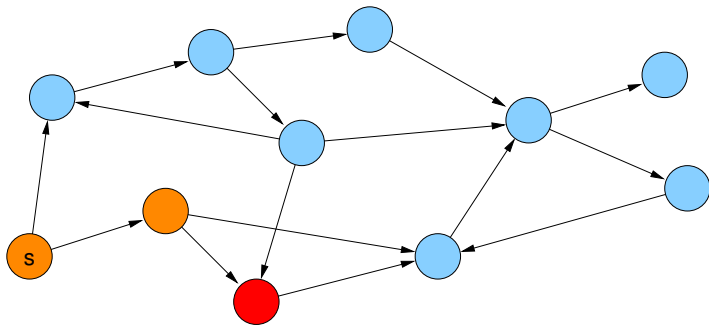
Tiefensuche



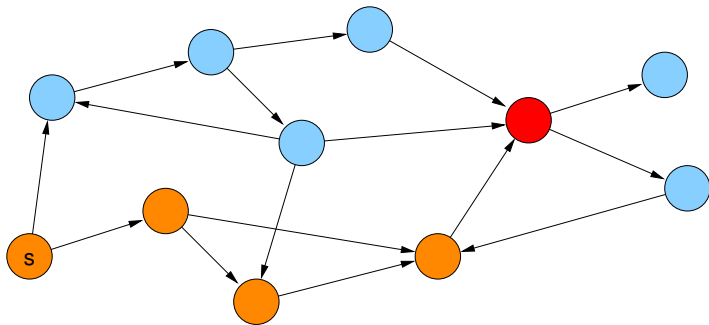
Tiefensuche



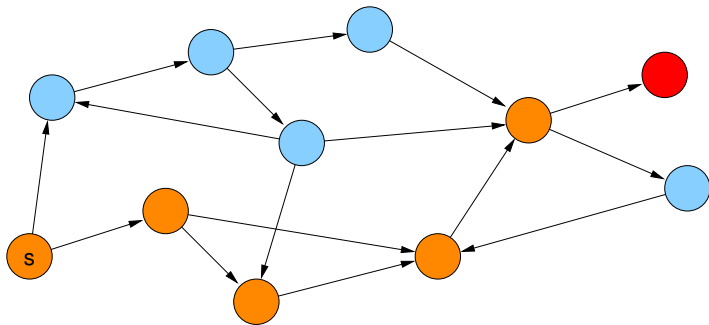
Tiefensuche



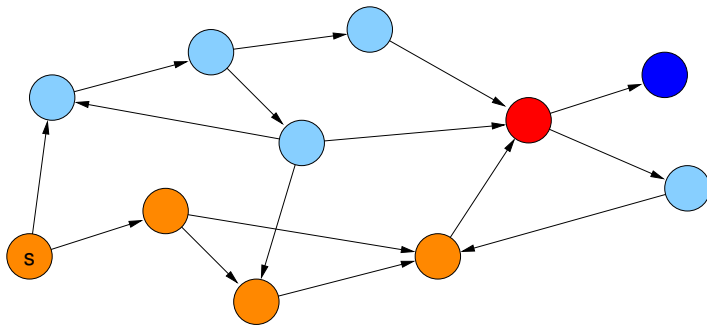
Tiefensuche



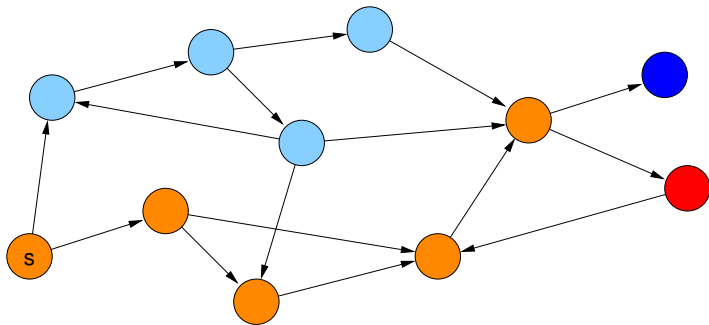
Tiefensuche



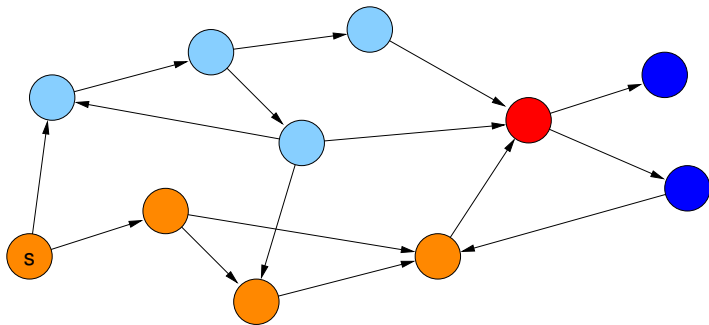
Tiefensuche



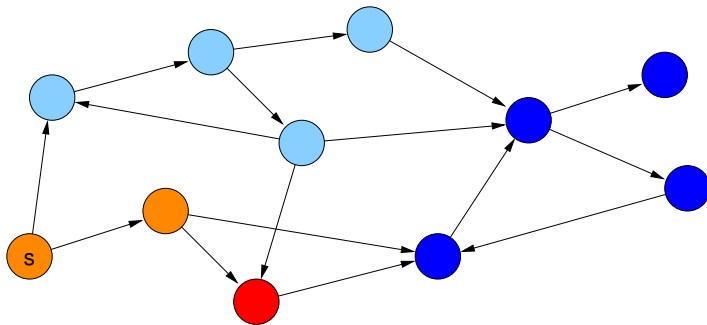
Tiefensuche



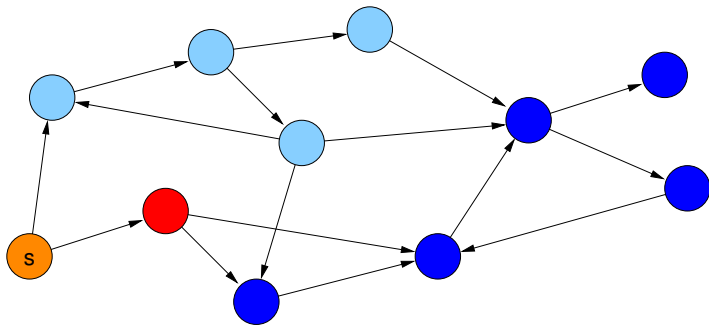
Tiefensuche



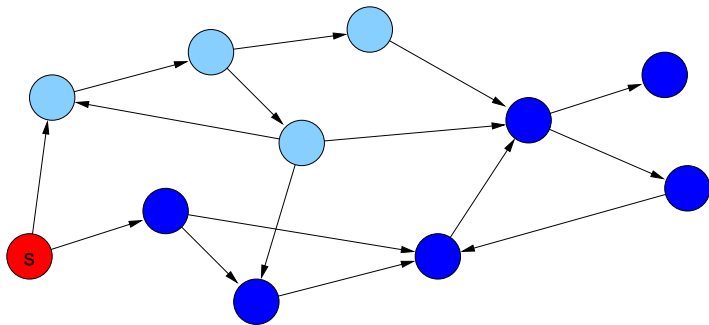
Tiefensuche



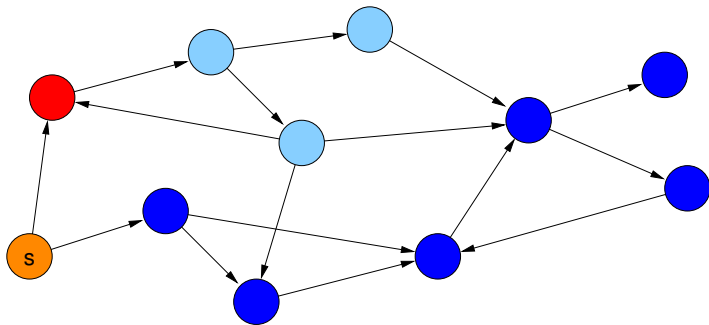
Tiefensuche



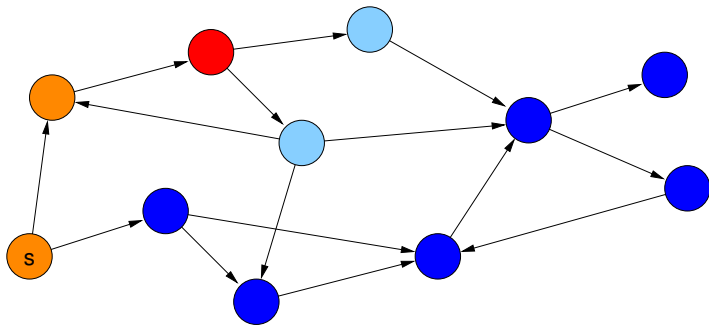
Tiefensuche



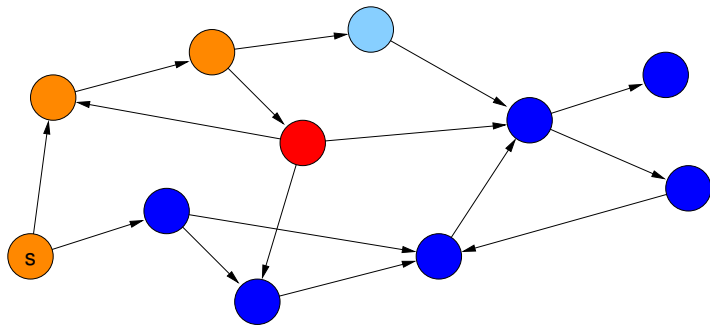
Tiefensuche



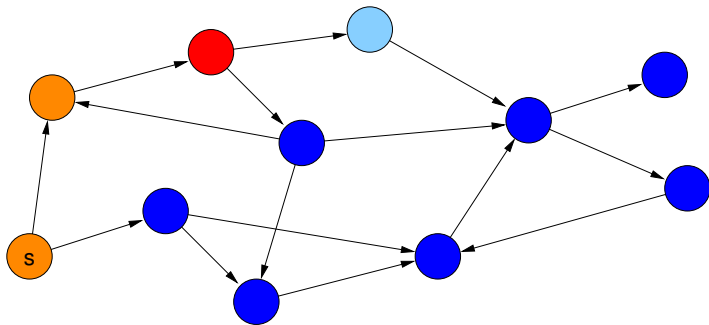
Tiefensuche



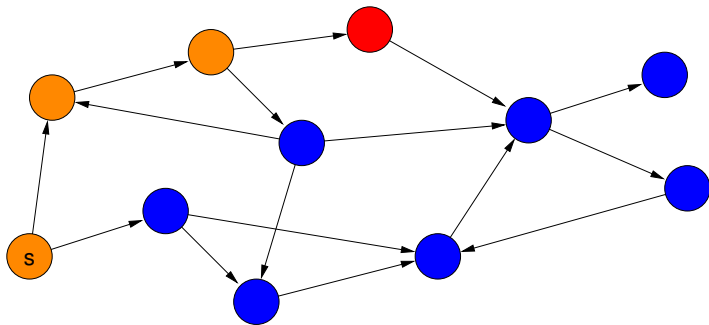
Tiefensuche



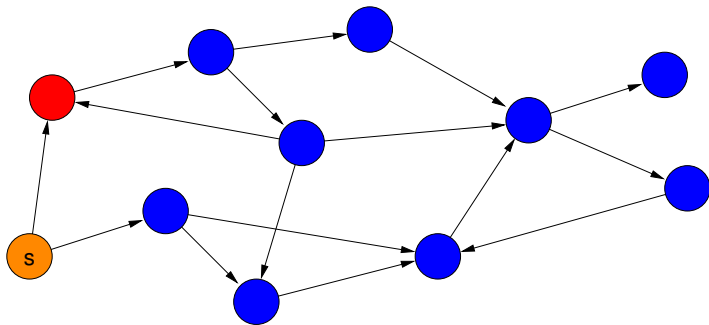
Tiefensuche



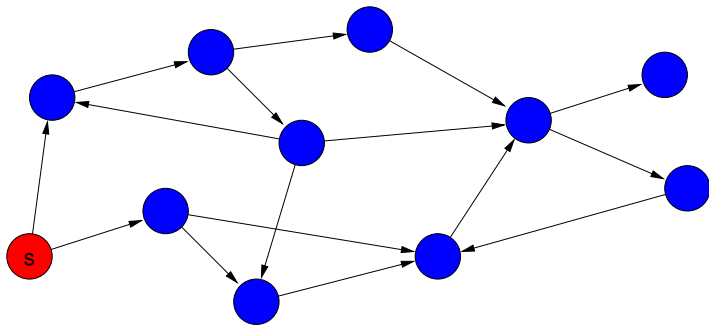
Tiefensuche



Tiefensuche



Tiefensuche



Tiefensuche

Übergeordnete Methode (falls nicht alle Knoten erreicht werden)

```
unmark all nodes;
```

```
init();
```

```
foreach ( $s \in V$ )
```

```
  if (s is not marked) {
```

```
    mark s;
```

```
    root(s);
```

```
    DFS(s,s);
```

```
  }
```


Tiefensuche

```
void DFS(Node u, Node v) {  
    foreach ((v, w) ∈ E)  
        if (w is marked)  
            traverseNonTreeEdge(v,w);  
        else {  
            traverseTreeEdge(v,w);  
            mark w;  
            DFS(v,w);  
        }  
    backtrack(u,v);  
}
```

Tiefensuche

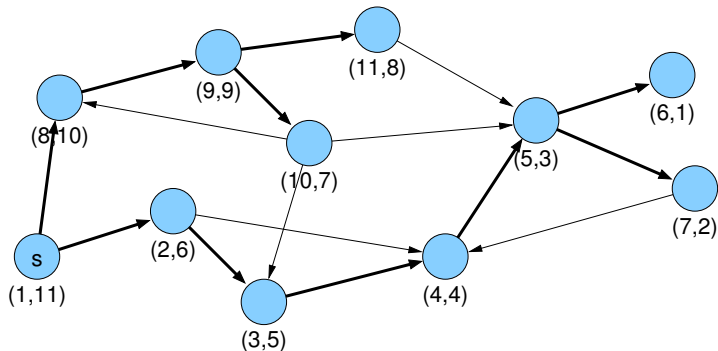
Variablen:

- `int[] dfsNum;` // Explorationsreihenfolge
- `int[] finishNum;` // Fertigstellungsreihenfolge
- `int dfsCount, finishCount;` // Zähler

Methoden:

- `void init() { dfsCount = 0; finishCount = 0; }`
- `void root(Node s) { dfsCount++; dfsNum[s] = dfsCount; }`
- `void traverseTreeEdge(Node v, Node w)`
`{ dfsCount++; dfsNum[w] = dfsCount; }`
- `void traverseNonTreeEdge(Node v, Node w) { }`
- `void backtrack(Node u, Node v)`
`{ finishCount++; finishNum[v] = finishCount; }`

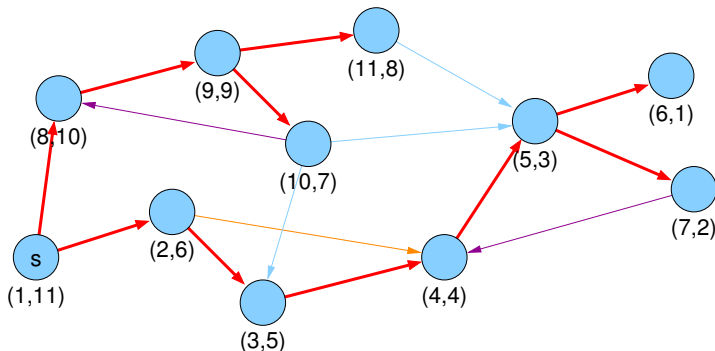
Tiefensuche



DFS-Nummerierung

Kantentypen:

- **Baumkanten:** zum Kind
- **Vorwärtskanten:** zu einem Nachfahren
- **Rückwärtskanten:** zu einem Vorfahren
- **Kreuzkanten:** sonstige



DFS-Nummerierung

Beobachtung für Kante (v, w) :

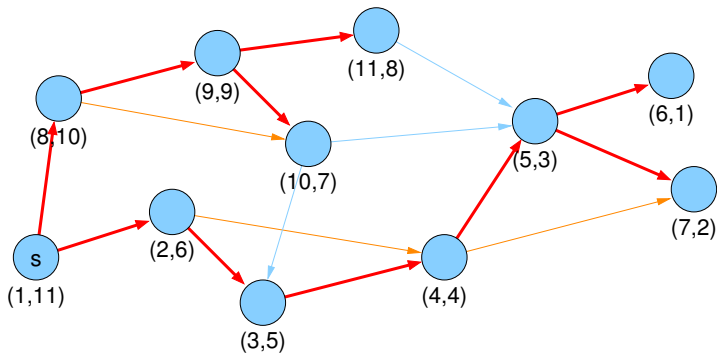
Kantentyp	$\text{dfsNum}[v] < \text{dfsNum}[w]$	$\text{finishNum}[v] > \text{finishNum}[w]$
Baum & Vorwärts	ja	ja
Rückwärts	nein	nein (umgekehrt)
Kreuz	nein	ja

Bemerkung: DFS-Vorwärtskanten gibt es nur in gerichteten Graphen

DFS-Nummerierung

Anwendung:

- Erkennung von azyklischen gerichteten Graphen (engl. directed acyclic graph / DAG)



- keine gerichteten Kreise

DFS-Nummerierung

Lemma

Folgende Aussagen sind äquivalent:

- 1 *Graph G ist ein DAG.*
- 2 *DFS in G enthält keine Rückwärtskante.*
- 3 $\forall (v, w) \in E : \text{finishNum}[v] > \text{finishNum}[w]$

Beweis.

- (2) \Rightarrow (3): wenn (2), dann gibt es nur Baum-, Vorwärts- und Kreuzkanten
Für alle gilt (3)
- (3) \Rightarrow (2): für Rückwärtskanten gilt sogar die umgekehrte Relation $\text{finishNum}[v] < \text{finishNum}[w]$
wenn (3), dann kann es also keine Rückwärtskanten geben (2)

...

DFS-Nummerierung

Lemma

Folgende Aussagen sind äquivalent:

- 1 Graph G ist ein DAG.
- 2 DFS in G enthält keine Rückwärtskante.
- 3 $\forall (v, w) \in E : \text{finishNum}[v] > \text{finishNum}[w]$

Beweis.

- $\neg(2) \Rightarrow \neg(1)$: wenn Rückwärtskante (v, w) existiert, gibt es einen gerichteten Kreis ab Knoten w (und G ist kein DAG)
- $\neg(1) \Rightarrow \neg(2)$: wenn es einen gerichteten Kreis gibt, ist mindestens eine von der DFS besuchte Kante dieses Kreises eine Rückwärtskante (Kante zu einem schon besuchten Knoten, dieser muss Vorfahr sein)



Zusammenhang in Graphen

Definition

Ein ungerichteter Graph heißt **zusammenhängend**, wenn es von jedem Knoten einen Pfad zu jedem anderen Knoten gibt.

Ein maximaler zusammenhängender induzierter Teilgraph wird als **Zusammenhangskomponente** bezeichnet.

Die Zusammenhangskomponenten eines ungerichteten Graphen können mit DFS oder BFS in $\mathcal{O}(n + m)$ bestimmt werden.

Knoten-Zweifachzusammenhang

Definition

Ein ungerichteter Graph $G = (V, E)$ heißt **2-fach zusammenhängend** (oder genauer gesagt *2-knotenzusammenhängend*), falls

- $|V| > 2$ und
- für jeden Knoten $v \in V$ der Graph $G - \{v\}$ zusammenhängend ist.

Artikulationsknoten und Blöcke

Definition

Ein Knoten v eines Graphen G heißt **Artikulationsknoten** (engl. *cut-vertex*), wenn sich die Anzahl der Zusammenhangskomponenten von G durch das Entfernen von v erhöht.

Definition

Die **Zweifachzusammenhangskomponenten** eines Graphen sind die maximalen Teilgraphen, die 2-fach zusammenhängend sind.

Ein **Block** ist ein maximaler zusammenhängender Teilgraph, der keinen Artikulationsknoten enthält.

D.h. die Menge der Blöcke besteht aus den Zweifachzusammenhangskomponenten, den Brücken (engl. *cut edges*), sowie den isolierten Knoten.

Blöcke und DFS

Modifizierte DFS nach R. E. Tarjan:

- $\text{dfsNum}[v]$: DFS-Nummer von v
- $\text{low}[v]$: kleinste Nummer $\text{dfsNum}[w]$ eines Knotens w , der von v aus über beliebig viele (≥ 0) Baumkanten abwärts, evt. gefolgt von einer einzigen Rückwärtskante erreicht werden kann

- $\text{low}[v]$: Minimum von
 - ▶ $\text{dfsNum}[v]$
 - ▶ $\text{low}[w]$, wobei w ein Kind von v im DFS-Baum ist (Baumkante)
 - ▶ $\text{dfsNum}[w]$, wobei $\{v, w\}$ eine Rückwärtskante ist

Blöcke und DFS

Lemma

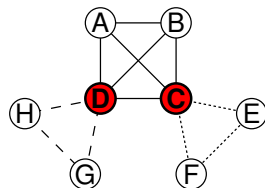
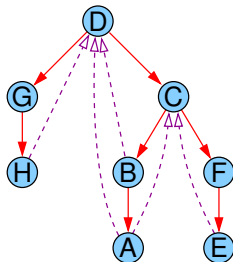
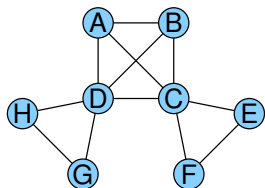
Sei $G = (V, E)$ ein ungerichteter, zusammenhängender Graph und T ein DFS-Baum in G .

Ein Knoten $a \in V$ ist genau dann ein Artikulationsknoten, wenn

- a die Wurzel von T ist und mindestens 2 Kinder hat, oder
- a nicht die Wurzel von T ist und es ein Kind b von a mit $low[b] \geq dfsNum[a]$ gibt.

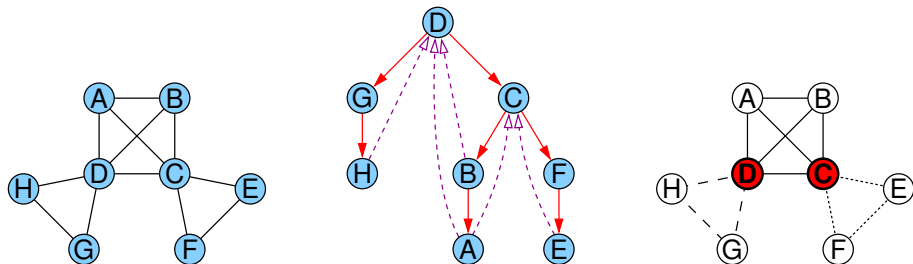
Artikulationsknoten und Blöcke per DFS

- bei Aufruf der DFS für Knoten v wird $\text{dfsNum}[v]$ bestimmt und $\text{low}[v]$ mit $\text{dfsNum}[v]$ initialisiert
- nach Besuch eines Nachbarknotens w : Update von $\text{low}[v]$ durch Vergleich mit
 - ▶ $\text{low}[w]$ nach Rückkehr vom rekursiven Aufruf, falls (v, w) eine Baumkante war
 - ▶ $\text{dfsNum}[w]$, falls (v, w) eine Rückwärtskante war



Artikulationsknoten und Blöcke per DFS

- Kanten werden auf einem anfangs leeren Stack gesammelt
- Baumkanten kommen vor dem rekursiven Aufruf auf den Stack
- Rückwärtskanten werden direkt auf den Stack gelegt
- nach Rückkehr von einem rekursiven Aufruf werden im Fall $low[w] \geq dfsNum[v]$ die obersten Kanten vom Stack bis einschließlich Baumkante $\{v, w\}$ entfernt (bilden nächsten Block)



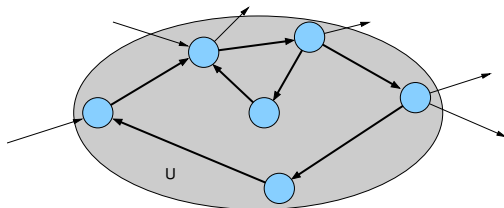
Starke Zusammenhangskomponenten

Definition

Sei $G = (V, E)$ ein gerichteter Graph.

Knotenteilmenge $U \subseteq V$ heißt **stark zusammenhängend** genau dann, wenn für alle $u, v \in U$ ein gerichteter Pfad von u nach v in G existiert.

Für Knotenteilmenge $U \subseteq V$ heißt der induzierte Teilgraph $G[U]$ **starke Zusammenhangskomponente** von G , wenn U stark zusammenhängend und (inklusions-)maximal ist.



Starke Zusammenhangskomponenten

Beobachtungen:

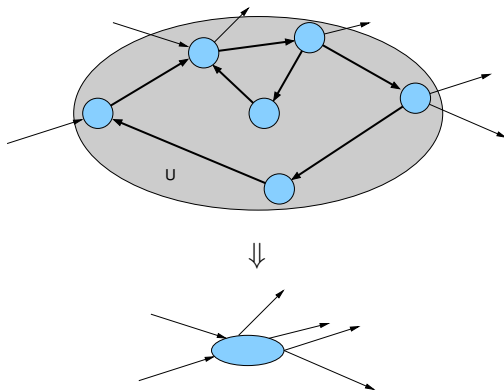
- Knoten $x, y \in V$ sind stark zusammenhängend, falls beide Knoten auf einem gemeinsamen gerichteten Kreis liegen (oder $x = y$).
- Die starken Zusammenhangskomponenten bilden eine Partition der Knotenmenge.

(im Gegensatz zu 2-Zhk. bei ungerichteten Graphen, wo nur die Kantenmenge partitioniert wird, sich aber zwei verschiedene 2-Zhk. in einem Knoten überlappen können)

Starke Zusammenhangskomponenten

Beobachtung:

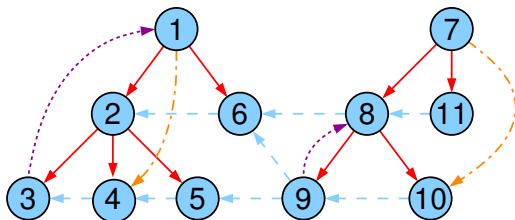
Schrumpft man alle starken Zusammenhangskomponenten zu einzelnen (Super-)Knoten, ergibt sich ein DAG.



Starke Zhk. und DFS / Variante 1

Modifizierte DFS nach R. E. Tarjan

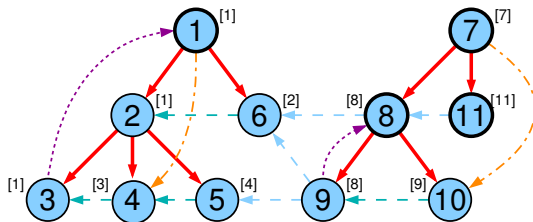
- **dfsNum**[v]: DFS-Nummer von v
- In einer starken Zhk. heißt der Knoten mit kleinster DFS-Nummer Wurzel der starken Zhk.
- **low**[v]: minimales **dfsNum**[w] eines Knotens w , der von v aus über beliebig viele (≥ 0) Baumkanten abwärts, evt. gefolgt von einer einzigen Rückwärtskante oder einer Querkante zu einer Zhk., deren Wurzel echter Vorfahre von v ist, erreicht werden kann



Starke Zhk. und DFS / Variante 1

Modifizierte DFS nach R. E. Tarjan

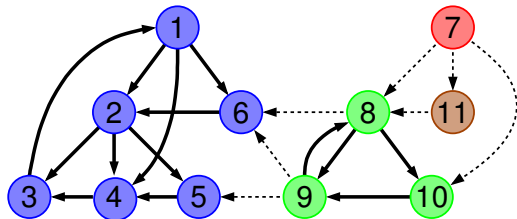
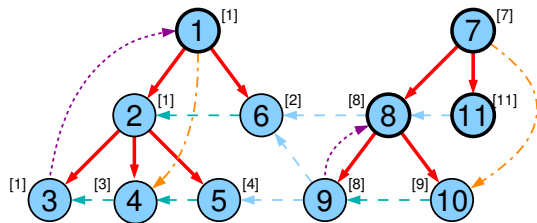
- $\text{low}[v]$: Minimum von
 - ▶ $\text{dfsNum}[v]$
 - ▶ $\text{low}[w]$, wobei w ein Kind von v im DFS-Baum ist (**Baumkante**)
 - ▶ $\text{dfsNum}[w]$, wobei $\{v, w\}$ eine **Rückwärtskante** ist
 - ▶ $\text{dfsNum}[w]$, wobei $\{v, w\}$ eine **Querkante** ist und die Wurzel der starken Zusammenhangskomponente von w ist Vorfahre von v



Starke Zhk. und DFS / Variante 1

Modifizierte DFS nach R. E. Tarjan

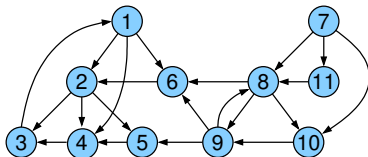
- Knoten v ist genau dann Wurzel einer starken Zusammenhangskomponente, wenn $\text{dfsNum}[v] = \text{low}[v]$



Starke Zhk. und DFS / Variante 2

Idee:

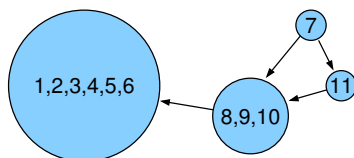
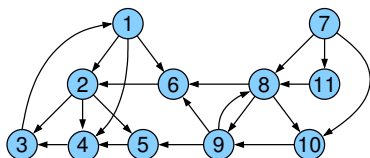
- beginne mit Graph ohne Kanten, jeder Knoten ist eigene SCC
 - füge nach und nach einzelne Kanten ein
- ⇒ aktueller (current) Graph $G_c = (V, E_c)$
- Update der starken Zusammenhangskomponenten (SCCs)



Starke Zhk. und DFS / Variante 2

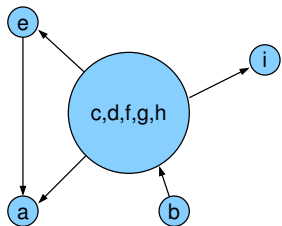
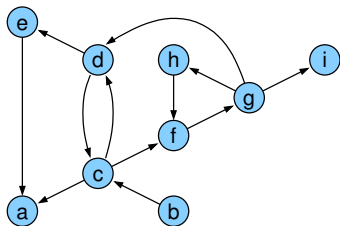
Idee:

- betrachte geschrumpften (shrunken) Graph G_c^s :
Knoten entsprechen SCCs von G_c , Kante (C, D) genau dann, wenn es Knoten $u \in C$ und $v \in D$ mit $(u, v) \in E_c$ gibt
- geschrumpfter Graph G_c^s ist ein DAG
- Ziel: Aktualisierung des geschrumpften Graphen beim Einfügen



Starke Zhk. und DFS / Variante 2

Geschrumpfter Graph
(Beispiel aus Mehlhorn / Sanders)



Starke Zhk. und DFS / Variante 2

Update des geschrumpften Graphen nach Einfügen einer Kante:

3 Möglichkeiten:

- beide Endpunkte gehören zu derselben SCC
⇒ geschrumpfter Graph unverändert
- Kante verbindet Knoten aus zwei verschiedenen SCCs, aber schließt keinen Kreis
⇒ SCCs im geschrumpften Graph unverändert, aber eine Kante wird im geschrumpften Graph eingefügt (falls nicht schon vorhanden)
- Kante verbindet Knoten aus zwei verschiedenen SCCs und schließt einen oder mehrere Kreise
⇒ alle SCCs, die auf einem der Kreise liegen, werden zu einer einzigen SCC verschmolzen

Starke Zhk. und DFS / Variante 2

Prinzip:

- Tiefensuche

V_c schon markierte (entdeckte) Knoten

E_c schon gefundene Kanten

- 3 Arten von SCC: unentdeckt, offen, geschlossen

- unentdeckte Knoten haben Ein- / Ausgangsgrad Null in G_c

⇒ zunächst bildet jeder Knoten eine eigene **unentdeckte** SCC, andere SCCs enthalten nur markierte Knoten

- SCCs mit mindestens einem aktiven Knoten (ohne finishNum) heißen **offen**

- SCC heißt **geschlossen**, falls sie nur fertige Knoten (mit finishNum) enthält

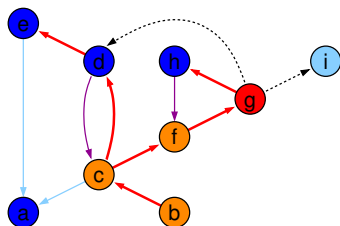
- Knoten in offenen / geschlossenen SCCs heißen offen / geschlossen

Starke Zhk. und DFS / Variante 2

- Knoten in geschlossenen SCCs sind immer fertig (mit finishNum)
- Knoten in offenen SCCs können fertig oder noch aktiv (ohne finishNum) sein
- **Repräsentant** einer SCC: Knoten mit kleinster dfsNum

Starke Zhk. und DFS / Variante 2

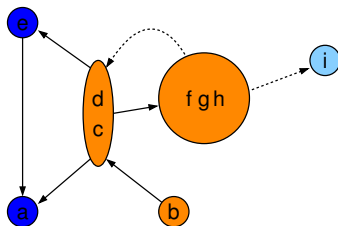
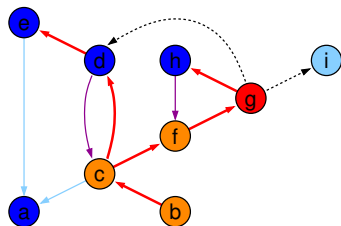
DFS-Snapshot:



- erste DFS startete bei Knoten a , zweite bei b
- aktueller Knoten ist g , auf dem Rekursionsstack liegen b, c, f, g
- (g, d) und (g, i) wurden noch nicht exploriert
- (d, c) und (h, f) sind Rückwärtskanten
- (c, a) und (e, a) sind Querkanten
- $(b, c), (c, d), (d, e), (c, f), (f, g)$ und (g, h) sind Baumkanten

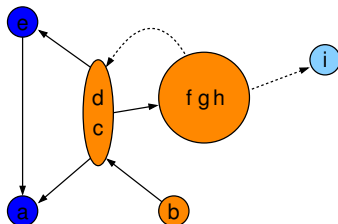
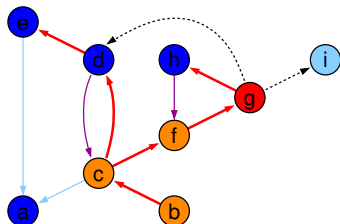
Starke Zhk. und DFS / Variante 2

DFS-Snapshot mit geschrumpftem Graph:



- unentdeckt: $\{i\}$ offen: $\{b\}$, $\{c, d\}$, $\{f, g, h\}$ geschlossen: $\{a\}$, $\{e\}$
- offene SCCs bilden Pfad im geschrumpften Graph
- aktueller Knoten gehört zur letzten SCC
- offene Knoten wurden in Reihenfolge b, c, d, f, g, h erreicht und werden von den Repräsentanten b, c und f genau in die offenen SCCs partitioniert

Starke Zhk. und DFS / Variante 2



Beobachtungen (Invarianten für G_c):

- 1 Pfade aus **geschlossenen** Knoten / SCCs führen immer zu **geschlossenen** Knoten / SCCs
- 2 Pfad zum aktuellen Knoten enthält die **Repräsentanten** aller **offenen** SCCs
offene Komponenten bilden **Pfad** im geschrumpften Graph
- 3 Knoten der offenen SCCs in Reihenfolge der DFS-Nummern werden durch Repräsentanten in die offenen SCCs **partitioniert**

Starke Zhk. und DFS / Variante 2

Geschlossene SCCs von G_c sind auch SCCs in G :

- Sei v geschlossener Knoten und S / S_c seine SCC in G / G_c .
- zu zeigen: $S = S_c$
- G_c ist Subgraph von G , also $S_c \subseteq S$
- somit zu zeigen: $S \subseteq S_c$

- Sei w ein Knoten in S .

$\Rightarrow \exists$ Kreis C durch v und w .

- Invariante 1: alle Knoten von C sind geschlossen und somit erledigt (alle ausgehenden Kanten exploriert)
- C ist in G_c enthalten, also $w \in S_c$
- damit gilt $S \subseteq S_c$, also $S = S_c$

Starke Zhk. und DFS / Variante 2

Vorgehen:

- Invarianten 2 und 3 helfen bei Verwaltung der offenen SCCs
- Knoten in offenen SCCs auf Stack **oNodes**
(in Reihenfolge steigender dfsNum)
- Repräsentanten der offenen SCCs auf Stack **oReps**
- zu Beginn Invarianten gültig (alles leer)
- vor Markierung einer neuen Wurzel sind alle markierten Knoten erledigt, also keine offenen SCCs, beide Stacks leer
dann: neue offene SCC für neue Wurzel s ,
 s kommt auf beide Stacks

Starke Zhk. und DFS / Variante 2

Prinzip: betrachte Kante $e = (v, w)$

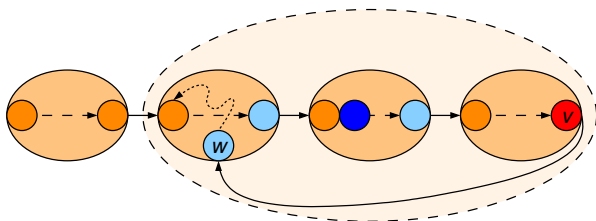
- Kante zu unbekanntem Knoten w (Baumkante):
neue eigene offene SCC für w (w kommt auf `oNodes` und `oReps`)
- Kante zu Knoten w in geschlossener SCC (Nicht-Baumkante):
von w gibt es keinen Weg zu v , sonst wäre die SCC von w noch nicht geschlossen (geschlossene SCCs sind bereits komplett), also SCCs unverändert
- Kante zu Knoten w in offener SCC (Nicht-Baumkante):
falls v und w in unterschiedlichen SCCs liegen, müssen diese mit allen SCCs dazwischen zu einer einzigen SCC verschmolzen werden (durch Löschen der Repräsentanten)

Wenn Knoten keine ausgehenden Kanten mehr hat:

- Knoten fertig
- wenn Knoten Repräsentant seiner SCC ist, dann SCC schließen

Starke Zhk. und DFS / Variante 2

Vereinigung offener SCCs im Kreisfall:



- offene SCC entsprechen Ovalen, Knoten sortiert nach dfsNum
- alle Repräsentanten offener SCCs liegen auf Baumpfad zum aktuellen Knoten v in SCC S_k
- Nicht-Baumkante (v, w) endet an Knoten w in offener SCC S_i mit Repräsentant r_i
- Pfad von w nach r_i muss existieren (innerhalb SCC S_i)

⇒ Kante (v, w) vereinigt S_i, \dots, S_k

Starke Zhk. und DFS / Variante 2

- `init()` {
 `component = new int[n];`
 `oReps = <>;`
 `oNodes = <>;`
 `dfsCount = 0;`
}

- `root(Node w) / traverseTreeEdge(Node v, Node w)` {
 `oReps.push(w);` // Repräsentant einer neuen ZHK
 `oNodes.push(w);` // neuer offener Knoten
 `dfsCount++;`
 `dfsNum[w] = dfsCount;`
}

Starke Zhk. und DFS / Variante 2

- **traverseNonTreeEdge**(Node v, Node w) {
 if ($w \in \text{oNodes}$) // verschmelze SCCs
 while ($\text{dfsNum}[w] < \text{dfsNum}[\text{oReps.top}()]$)
 oReps.pop();
}

- **backtrack**(Node u, Node v) {
 if ($v == \text{oReps.top}()$) { // v Repräsentant?
 oReps.pop(); // ja: entferne v
 do { // und offene Knoten bis v
 w = oNodes.pop();
 component[w] = v;
 } while ($w \neq v$);
 }
}

Starke Zhk. und DFS / Variante 2

Zeit: $\mathcal{O}(n + m)$

Begründung:

- **init, root:** $\mathcal{O}(1)$
- **traverseTreeEdge:** $(n - 1) \times \mathcal{O}(1)$
- **backtrack, traverseNonTreeEdge:**
da jeder Knoten höchstens einmal in `oReps` und `oNodes` landet,
insgesamt $\mathcal{O}(n + m)$
- **DFS-Gerüst:** $\mathcal{O}(n + m)$
- **gesamt:** $\mathcal{O}(n + m)$

Übersicht

1 Grundlagen

2 Zentralitätsindizes

- Beispiele
- Grad- und Distanz-basierte Zentralitäten
- Kürzeste Pfade und Zentralität
- Abgeleitete Kantenzentralitäten
- Vitalität
- Elektrischer Fluss

3 Wiederholung: Kürzeste Wege

4 Algorithmen für Zentralitätsindizes

5 Lokale Dichte

Zentralitätsindizes

- Manche Elemente in Netzwerken (Knoten / Kanten in Graphen) sind wichtiger, zentraler oder einflussreicher als andere.
- Zentralitätsmaße bzw. -indizes (kurz Zentralitäten) quantifizieren diese Eigenschaften durch skalare Werte.
- Es gibt jedoch **kein universelles Zentralitätsmaß**, das jeder Anwendung gerecht wird.

⇒ 'richtiges' Maß hängt vom Kontext der Anwendung ab

Beispiel: Wahl eines Klassensprechers

Variante 1:

- Knoten entsprechen Personen
 - Kante von Knoten A nach B , wenn Person A für Person B stimmt
-
- Person ist zentraler, je höher die Anzahl der erhaltenen Stimmen ist
- ⇒ in-degree centrality

Beispiel: Wahl eines Klassensprechers

Variante 2:

- Knoten entsprechen Personen
 - Kante von Knoten A nach B , wenn Person A Person B überzeugt hat, für seinen/ihren Favoriten zustimmen
 - Einfluss-Netzwerk
-
- Person ist zentraler, je mehr diese Person gebraucht wird, um die Meinung anderer zu transportieren
- ⇒ **betweenness** centrality

Beispiel: Wahl eines Klassensprechers

Variante 3:

- Knoten entsprechen Personen
 - Kante von Knoten A nach B , wenn Person A mit Person B befreundet ist

 - Person ist zentraler, je mehr Freunde diese Person hat und je zentraler diese Freunde sind
- ⇒ **feedback** centrality

Kantenzentralität

Ebenso kann man auch die Wichtigkeit / Zentralität von Beziehungen in Netzwerken (**Kanten** in Graphen) betrachten.

Beispiel: Internet

- Backbone: Verbindungen zwischen den Kontinenten gibt es wenige und sie müssen eine große Kapazität haben

Arten von Kantenzentralität / Beispiele:

- Beteiligung einer Kante an kürzesten Wegen usw.
⇒ betweenness edge centrality
- Veränderung von Netzwerk-Parametern durch Löschen der Kante
⇒ edge vitality (Bsp. flow betweenness vitality)

Definition: Zentralitätsindex

Abgesehen von der Intuition

- Wichtigkeit,
- Prestige,
- Einfluss,
- Kontrolle,
- Unentbehrlichkeit

gibt es keine allgemeingültige (formale) Definition von Zentralität.

Mindestanforderung:

Das Maß darf nur von der Struktur des Graphen abhängen.

Werte sollen zumindest eine **Ordnung** der Elemente liefern, auch wenn die numerischen Werte, Abstände oder Verhältnisse evt. nicht viel aussagen.

Graph-Isomorphie

Definition (Isomorphie)

Zwei Graphen $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$ sind **isomorph** ($G_1 \simeq G_2$), falls es eine Bijektion $\phi : V_1 \rightarrow V_2$ gibt, so dass

$$(u, v) \in E_1 \quad \Leftrightarrow \quad (\phi(u), \phi(v)) \in E_2$$

Definition (Struktureller Index)

Sei $G = (V, E)$ ein gewichteter (gerichteter oder ungerichteter) Graph und sei X die Knotenmenge (V) oder die Kantenmenge (E).

Eine reellwertige Funktion s heißt genau dann **struktureller Index**, wenn die folgende Bedingung erfüllt ist:

$$\forall x \in X : \quad G \simeq H \quad \Rightarrow \quad s_G(x) = s_H(\phi(x))$$

Grad-Zentralität

Grad-Zentralität (degree centrality):

$$c_D(v) = \deg(v)$$

- **lokales Maß**,
nur von der direkten Nachbarschaft des Knotens abhängig
- mögliche Verallgemeinerung:
Anzahl der indirekten Nachbarn mit Abstand $\leq k$

Exzentrizität

Anwendung: Positionierung eines Hospitals oder einer Feuerwehrstation

Ziel: **Minimierung** der **maximal** notwendigen Anfahrtzeit
(minimax location problem)

Exzentrizität eines Knotens $v \in V$:

$$e(v) = \max_{w \in V} \{d(v, w)\}$$

Zentralitätsmaß:

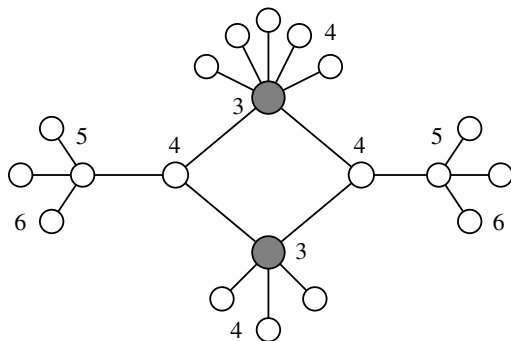
$$c_E(v) = \frac{1}{e(v)} = \frac{1}{\max_{w \in V} \{d(v, w)\}}$$

Optimale Standorte:

alle Knoten v mit minimalem Wert $e(v)$

Zentrum $\mathcal{C}(G)$

Exzentrizität / Beispiel



(Quelle: Brandes/Erlebach (Eds.): Network Analysis)

Die markierten Knoten haben die kleinste Exzentrizität (bzw. die größte Zentralität) und bilden das Zentrum des Graphen.

Nähe / Closeness

Anwendung: Positionierung eines Einkaufszentrums
(minisum location / median / service facility location problem)

Ziel:
Minimierung der Summe der Entfernungen zu den anderen Knoten
(und damit auch der durchschnittlichen Entfernung):

$$\sum_{v \in V} d(u, v)$$

(im Kontext von Kommunikationsnetzen auch *transmission number*)

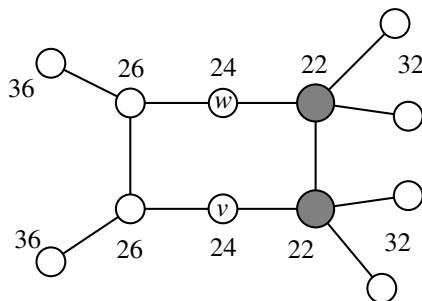
Zentralitätsmaß:

$$c_C(v) = \frac{1}{\sum_{w \in V} d(v, w)}$$

Optimale Standorte:
alle Knoten v mit minimaler Distanzsumme

Median $\mathcal{M}(G)$

Closeness / Beispiel



(Quelle: Brandes/Erlebach (Eds.): Network Analysis)

Graue Knoten sind optimal bezüglich Closeness (Median),
aber v und w sind zentraler bezüglich Exzentrität (Zentrum)

Radiality

ähnlich zu Closeness

$$c_R(v) = \frac{\sum_{w \in V} d_{\max}(G) + 1 - d(v, w)}{n - 1}$$

$d_{\max}(G)$: Durchmesser des Graphen

(größte Distanz zweier Knoten, nicht Länge des längsten Pfads!)

Zentroid-Wert

Konkurrenzsituation:

- Knoten repräsentieren Kunden, die beim nächstgelegenen Geschäft einkaufen
- Annahme: **zwei Anbieter**, der zweite Anbieter berücksichtigt bei der Standortwahl den Standort des ersten Anbieters

Fragen:

- Welchen Standort muss der erste Anbieter wählen, damit er durch den zweiten Anbieter möglichst wenig Kunden verliert?
- Ist es vorteilhaft als Erster auswählen zu können?

Zentroid-Wert

$\gamma_u(v)$: die Anzahl der Knoten, deren Distanz zu u kleiner ist als zu v :

$$\gamma_u(v) = |\{w \in V : d(w, u) < d(w, v)\}|$$

Wähle Knoten u , Gegenspieler wählt Knoten v

Resultierende Anzahl Kunden:

$$\Rightarrow \gamma_u(v) + \frac{n - \gamma_u(v) - \gamma_v(u)}{2} = \frac{n + \gamma_u(v) - \gamma_v(u)}{2}$$

\Rightarrow Gegenspieler minimiert $f(u, v) = \gamma_u(v) - \gamma_v(u)$

Zentralitätsmaß:

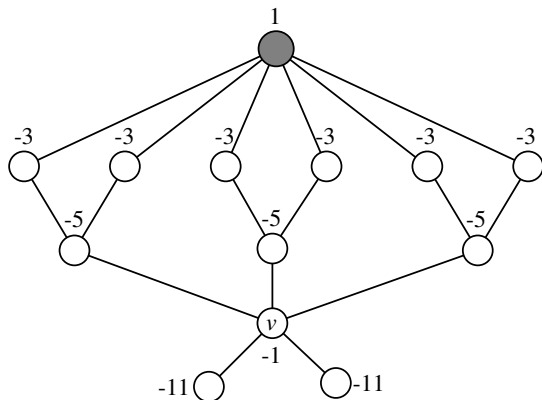
$$c_F(u) = \min_{v \in (V \setminus u)} \{f(u, v)\}$$

Optimale Standorte:

alle Knoten v mit maximalem Zentroidwert

Zentroid $\mathcal{Z}(G)$

Zentroid-Wert / Beispiel 1



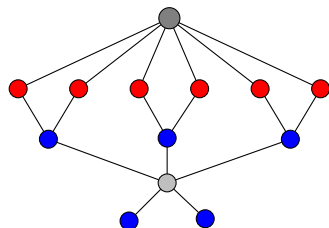
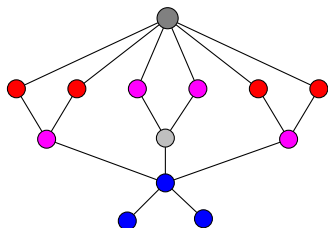
(Quelle: Brandes/Erlebach (Eds.): Network Analysis)

Der graue Knoten hat maximalen Zentroid-Wert,
aber v ist der Knoten mit maximaler Closeness.

Zentroid-Wert / Beispiel 1

Beispiel-Lösungen:

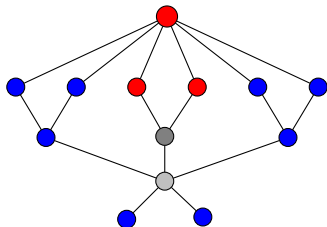
optimale Auswahl des 1. Knotens (dunkelgrau)



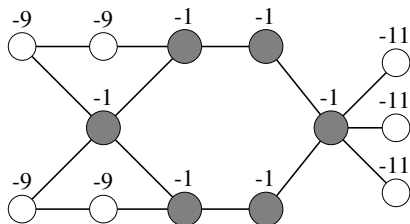
- dunkelgrau: eigene Wahl (zuerst)
- hellgrau: Wahl des Konkurrenten
- rot: näher zu dunkelgrau
- blau: näher zu hellgrau
- violett: gleiche Distanz zu dunkelgrau / hellgrau

Zentroid-Wert / Beispiel 1

Beispiel-Lösungen:
nicht-optimale Auswahl des 1. Knotens (dunkelgrau)



Zentroid-Wert / Beispiel 2



(Quelle: Brandes/Erlebach (Eds.): Network Analysis)

Die grauen Knoten haben maximalen Zentroid-Wert,
aber selbst sie haben einen negativen Wert.
⇒ Es ist hier vorteilhaft, erst als Zweiter zu wählen.

Bemerkungen

- Exzentrizität, Closeness und Zentroid-Wert sind strukturelle Indizes.
- Die Knoten maximaler Zentralität unterscheiden sich bei den verschiedenen Maßen.
- Im Gegensatz zu Exzentrizität und Closeness kann der Zentroid-Wert negativ sein.

Baum-Zentrum

Satz (Jordan, 1869)

In jedem Baum T besteht das Zentrum $\mathcal{C}(T)$ aus genau einem oder aus zwei benachbarten Knoten.

Beweis.

- Baum aus höchstens 2 Knoten $\Rightarrow \mathcal{C}(T) = V$
 - Für jeden Knoten v von T können nur Blätter exzentrisch sein (maximale Entfernung haben), keine inneren Knoten.
 - Betrachte Baum T' , der durch Entfernen aller Blätter aus T entsteht
- \Rightarrow Exzentrizität jedes Knotens in T' ist um genau Eins kleiner als in T
- beide Bäume haben **gleiches Zentrum** (falls T' nicht leer ist)
 - Wiederholung führt zwangsläufig zu einem Baum, der aus genau einem oder zwei adjazenten Knoten besteht (dem Zentrum)



Berechnung des Baum-Zentrums

Vorangegangener Beweis impliziert einen einfachen Algorithmus zur Berechnung des Zentrums eines Baums, der nicht einmal die Berechnung der Exzentrizität der einzelnen Knoten erfordert.

Eigenschaft des Graph-Zentrums

Satz

In jedem zusammenhängenden ungerichteten Graph G befinden sich alle Knoten des Graph-Zentrums $\mathcal{C}(G)$ im selben Block.

Eigenschaft des Graph-Zentrums

Beweis.

- Annahme: kein Block in G enthält alle Knoten des Zentrums $\mathcal{C}(G)$
 $\Rightarrow G$ enthält einen Artikulationsknoten u , so dass $G - u$ in nicht verbundene Teilgraphen G_1 und G_2 zerfällt, die jeweils mindestens einen Knoten ($\neq u$) aus $\mathcal{C}(G)$ enthalten.
- Sei v ein exzentrischer Knoten von u und P ein entsprechender kürzester Pfad (der Länge $e(u)$) zwischen u und v .
o.B.d.A. sei $v \in G_2$
 $\Rightarrow \exists$ Knoten $w \in \mathcal{C}(G)$ in G_1 ($w \neq u$). w liegt nicht auf P .
 $\Rightarrow e(w) \geq d(w, u) + d(u, v) \geq 1 + e(u)$
 \Rightarrow wegen $e(w) > e(u)$ gehört w nicht zu $\mathcal{C}(G)$ (Widerspruch)



Graph-Median

$$s(G) = \min_{v \in V} \left\{ \sum_{w \in V} d(v, w) \right\}$$

Median:

$$\mathcal{M}(G) = \left\{ v \in V : \sum_{w \in V} d(v, w) = s(G) \right\}$$

Satz

In jedem zusammenhängenden ungerichteten Graph G befinden sich alle Knoten des Graph-Medians $\mathcal{M}(G)$ im selben Block.

Beweis: Übungsaufgabe

Folgerung

Der Median eines Baums besteht entweder aus einem einzelnen Knoten oder aus zwei adjazenten Knoten.

Graph-Zentroid

$$f(G) = \max_{v \in V} \{c_F(v)\}$$

Zentroid:

$$\mathcal{Z}(G) = \{v \in V : c_F(v) = f(G)\}$$

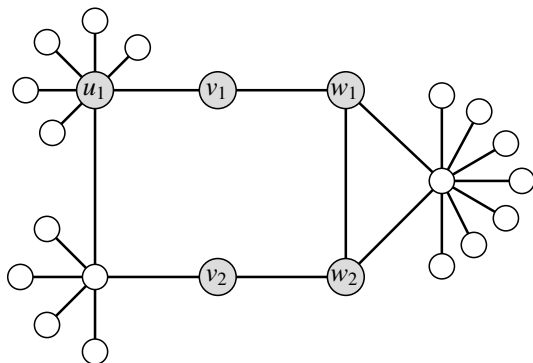
Satz (Slater, 1975)

Für jeden Baum sind Median und Zentroid identisch.

Satz (Smart & Slater, 1999)

In jedem zusammenhängenden ungerichteten Graphen liegen Median und Zentroid im gleichen Block.

Beispiel



$$\mathcal{C}(G) = \{v_1, v_2\},$$

$$\mathcal{M}(G) = \{u_1\},$$

$$\mathcal{Z}(G) = \{w_1, w_2\}$$

Unterschiedlichkeit der Maße

Satz

Für drei beliebige zusammenhängende ungerichtete Graphen H_1 , H_2 und H_3 und eine beliebige natürliche Zahl $k \geq 4$ existiert ein ungerichteter zusammenhängender Graph G , so dass

- $G[\mathcal{C}(G)] \simeq H_1$,
- $G[\mathcal{M}(G)] \simeq H_2$,
- $G[\mathcal{Z}(G)] \simeq H_3$, und
- die Distanz zwischen je zwei der zentralen Mengen ist mindestens k .

Stress-Zentralität

- Anzahl kürzester Pfade zwischen Knoten s und t , die $v \in V$ bzw. $e \in E$ enthalten: $\sigma_{st}(v)$ bzw. $\sigma_{st}(e)$
- Stress-Zentralität:

$$c_S(v) = \sum_{s \neq v \in V} \sum_{t \neq v \in V} \sigma_{st}(v)$$

$$c_S(e) = \sum_{s \in V} \sum_{t \in V} \sigma_{st}(e)$$

- Intuition: Kommunikationsfluss durch Knoten bzw. Kanten auf (allen) kürzesten Wegen zwischen allen Knotenpaaren

Knoten- und Kanten-Stresszentralität

Lemma

In einem gerichteten Graphen $G = (V, E)$ sind die Stresszentralitäten von Knoten und Kanten wie folgt miteinander verknüpft:

$$c_S(v) = \frac{1}{2} \left(\sum_{e \in \Gamma(v)} c_S(e) - \sum_{s \in V, s \neq v} \sigma_{sv} - \sum_{t \in V, t \neq v} \sigma_{vt} \right)$$

$\Gamma(v)$: Menge der Kanten mit Endknoten v

σ_{xy} : Anzahl kürzester Wege von x nach y

Knoten- und Kanten-Stresszentralität

Beweis.

- Betrachte einen kürzesten Pfad zwischen $s, t \in V$.
- ⇒ trägt genau 1 zum Stress jedes inneren Knotens und jeder Kante bei
- Innere Knoten sind jeweils zu zwei Kanten des Pfads benachbart.
- Anfangs- und Endknoten sind immer zu einer Kante des Pfads benachbart. (Während dieser kürzeste Pfad dann für die Kante zählt, soll er beim Anfangs-/Endknoten nicht mitgezählt werden.)
- Wie oft ist v Anfangs- bzw. Endknoten eines kürzesten Pfades?

$$\sum_{s \in V, s \neq v} \sigma_{sv} + \sum_{t \in V, t \neq v} \sigma_{vt}$$



Shortest-Path Betweenness

- eine Art relative Stress-Zentralität
- Anzahl kürzester Pfade zwischen $s, t \in V$: σ_{st}
- Anzahl kürzester Pfade zwischen s, t , die v enthalten: $\sigma_{st}(v)$
- Anteil der kürzesten Pfade zwischen s und t , die v enthalten:

$$\delta_{st}(v) = \frac{\sigma_{st}(v)}{\sigma_{st}}$$

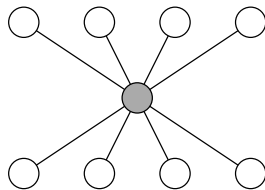
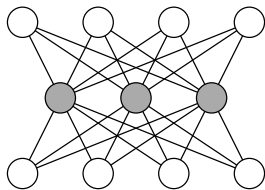
⇒ interpretiert als Anteil oder Wahrscheinlichkeit der Kommunikation

- Betweenness-Zentralität:

$$c_B(v) = \sum_{s \neq v \in V} \sum_{t \neq v \in V} \delta_{st}(v)$$

Vorteile von Betweenness

- Vorteil gegenüber Closeness:
funktioniert auch bei nicht zusammenhängenden Graphen
- Vorteil gegenüber Stress (siehe Abb.): Knoten der mittleren Schicht haben in beiden Graphen gleiche Stresszentralität
links: mehrere Alternativen
rechts: mittlerer Knoten für gesamte Kommunikation notwendig



- Normierung: Teilen durch Anzahl Paare
 $(n-1)(n-2)$ bzw. $(n-1)(n-2)/2$

Betweenness für Kanten

- Anteil der Kante e am Informationsfluss (auf allen kürzesten Pfaden) zwischen den Knoten s und t

$$\delta_{st}(e) = \frac{\sigma_{st}(e)}{\sigma_{st}}$$

- Betweenness-Zentralität der Kante e :

$$c_B(e) = \sum_{s \in V} \sum_{t \in V} \delta_{st}(e)$$

Knoten- und Kanten-Betweenness-Zentralität

Lemma

In einem gerichteten Graphen $G = (V, E)$ ist die (Kürzeste-Pfade-)Betweenness-Zentralität für Knoten und Kanten wie folgt miteinander verknüpft:

$$c_B(v) = \left(\sum_{e \in \Gamma^+(v)} c_B(e) \right) - (n - 1) = \left(\sum_{e \in \Gamma^-(v)} c_B(e) \right) - (n - 1)$$

$\Gamma^+(v)$: Menge der Kanten mit Startknoten v

$\Gamma^-(v)$: Menge der Kanten mit Zielknoten v

Knoten- und Kanten-Betweenness-Zentralität

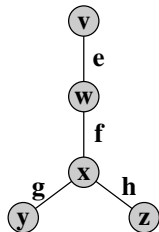
Beweis.

- Betrachte einen kürzesten Pfad zwischen $s, t \in V$.
- ⇒ trägt genau $1/\sigma_{st}$ zur Betweenness jedes inneren Knotens und jeder Kante bei
- Innere Knoten sind jeweils zu einer eingehenden und einer ausgehenden Kante des Pfads benachbart.
- Anfangs- und Endknoten sind immer zu genau einer Kante des Pfads benachbart. (Während dieser kürzeste Pfad dann $1/\sigma_{st}$ zur Betweenness der Kante beiträgt, soll er beim Anfangs-/Endknoten nicht mitgezählt werden.)
- Insgesamt summieren sich die Anteile eines Knotens als Anfangs- oder Endknoten zu $(n - 1)$, die also von der Summe der angrenzenden Kanten abzuziehen sind.

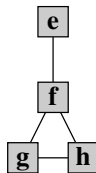


Abgeleitete Kantenzentralitäten

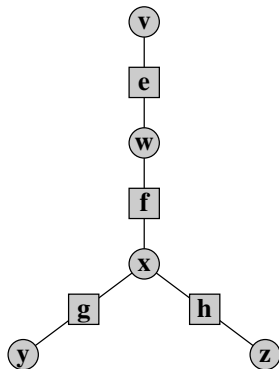
Wie kann man aus Knotenzentralitäten automatisch Definitionen von Kantenzentralität ableiten?



Graph



Kantengraph



Inzidenzgraph

Kantenzentralität als Knotenzentralität im Kantengraph

Definition

Kantengraph von Graph $G = (V, E)$ ist $G' = (E, K)$, wobei K die Menge der Kanten $e = ((x, y), (y, z))$ mit $(x, y) \in E$ und $(y, z) \in E$ ist.

Knoten im Kantengraph sind also benachbart, wenn die zugehörigen Kanten im Originalgraph einen Knoten gemeinsam haben (im gerichteten Fall als Zielknoten der einen und Startknoten der anderen Kante).

⇒ Wende Knotenzentralität auf den Kantengraph an

Nachteile:

- Größe des Kantengraphen kann quadratisch in der Größe des ursprünglichen Graphen sein,
- Kantengewichte in G werden zu Knotengewichten in G' (Nutzung unklar)
- keine natürliche Interpretation / Generalisierung

Kantenzentralität als Knotenzentralität im Inzidenzgraph

Definition

Inzidenzgraph von Graph $G = (V, E)$ ist $G'' = (V'', E'')$, wobei
 $V'' = V \cup E$ und $E'' = \{(v, e) \mid \exists w : e = (v, w) \in E\} \cup \{(e, w) \mid \exists v : e = (v, w) \in E\}$.

Im Inzidenzgraphen sind also ein 'echter Knoten' und ein 'Kantenknoten' benachbart, wenn der entsprechende Knoten und die entsprechende Kante im ursprünglichen Graphen inzident sind.

⇒ Wende Knotenzentralität auf den Inzidenzgraph an, wobei nur die Pfade zwischen 'echten Knoten' als relevant betrachtet werden

Vitalität

Vitalitätsmaße bewerten die Wichtigkeit eines Knotens oder einer Kante anhand eines Qualitätsverlusts durch das Löschen des Knotens bzw. der Kante.

Definition (Vitalitätsindex)

Sei \mathcal{G} die Menge aller einfachen ungerichteten ungewichteten Graphen $G = (V, E)$ und $f : \mathcal{G} \rightarrow \mathbb{R}$ eine reellwertige Funktion auf $G \in \mathcal{G}$.

Dann ist ein **Vitalitätsindex** $\mathcal{V}(G, x)$ definiert als die Differenz der Werte von f auf G und $G \setminus \{x\}$:

$$\mathcal{V}(G, x) = f(G) - f(G \setminus \{x\})$$

(Wir werden sehen, dass diese Definition eigentlich zu restriktiv ist.)

Flow Betweenness Vitality

- ähnlich zu Shortest Path Betweenness
- Motivation: Information in einem Kommunikationsnetzwerk muss sich nicht unbedingt auf kürzesten Pfaden bewegen.
- Maß: Abhängigkeit des maximalen Flusses zwischen zwei Knoten von der Existenz des betrachteten Knotens

Flow Betweenness Vitality

- f_{st} : Maximum-Fluss zwischen Knoten s und t unter Berücksichtigung der Kantenkapazitäten
- $f_{st}(v)$: Fluss zwischen Knoten s und t , der bei Maximum-Fluss von s nach t durch v gehen **muss**
- $\tilde{f}_{st}(v)$: Maximum-Fluss von s nach t in $G - v$

$$c_{mf}(v) = \sum_{\substack{s, t \in V \\ v \neq s, v \neq t \\ f_{st} > 0}} \frac{f_{st}(v)}{f_{st}} = \sum_{\substack{s, t \in V \\ v \neq s, v \neq t \\ f_{st} > 0}} \frac{f_{st} - \tilde{f}_{st}(v)}{f_{st}}$$

Closeness Vitality

- Wiener Index:

$$I_W(G) = \sum_{v \in V} \sum_{w \in V} d(v, w)$$

- oder in Abhängigkeit der Closeness-Zentralitäten:

$$I_W(G) = \sum_{v \in V} \frac{1}{c_C(v)}$$

- Closeness Vitality für Knoten/Kante x :

$$c_{CV}(x) = I_W(G) - I_W(G - x)$$

- Interpretation: Um wieviel steigen die Gesamtkommunikationskosten, wenn jeder Knoten mit jedem kommuniziert?

Closeness Vitality - Durchschnitt

- Durchschnittliche Distanz:

$$\bar{d}(G) = \frac{I_W(G)}{n(n-1)}$$

- Das Vitalitätsmaß auf der Basis $f(G) = \bar{d}(G)$ misst die durchschnittliche Verschlechterung der Entfernung zwischen zwei Knoten beim Entfernen eines Knotens / einer Kante x .
- Vorsicht! Beim Entfernen eines Artikulationsknotens (cut vertex) oder einer Brücke (cut edge) x ist $c_{CV}(x) = -\infty$.

Shortcut-Werte

- maximale Erhöhung eines Distanzwerts, wenn Kante $e = (v, w)$ entfernt wird
- ⇒ nur relevant für Paare, bei denen alle kürzesten Pfade über e laufen
- maximale Erhöhung tritt direkt zwischen Knoten v und w auf
 - Berechnung: mit $m = |E|$ Single Source Shortest Path Instanzen (und Vergleich mit der jeweiligen Kante)
 - später: mit $n = |V|$ SSSP-Bäumen
-
- alternativ: maximale relative Erhöhung
 - Anwendung auch auf Knotenlöschungen möglich

Bemerkung: kein echter Vitalitätsindex im Sinne der Definition

Stress-Zentralität als Vitalitätsindex

- Stress-Zentralität zählt die Anzahl der kürzesten Pfade, an denen ein Knoten oder eine Kante beteiligt ist
- Diese kürzesten Pfade gehen verloren, wenn der Knoten bzw. die Kante gelöscht wird.

⇒ kann als Vitalitätsmaß betrachtet werden

- Aber: Anzahl der kürzesten Pfade kann sich durch Löschung erhöhen (wenn sich die Distanz zwischen zwei Knoten erhöht)

⇒ Längere kürzeste Pfade müssen ignoriert werden

Stress-Zentralität als Vitalitätsindex

- $f(G \setminus \{v\})$ muss ersetzt werden durch

$$\sum_{s \in V} \sum_{t \in V} \sigma_{st} [d_G(s, t) = d_{G \setminus \{v\}}(s, t)]$$

- Ausdruck in Klammern ist 1 (wahr) oder 0 (falsch)
- $f(G \setminus \{e\})$ analog:

$$\sum_{s \in V} \sum_{t \in V} \sigma_{st} [d_G(s, t) = d_{G \setminus \{e\}}(s, t)]$$

- Also: $f(G) - „f(G \setminus x)“ = c_S(x)$

(kein echter Vitalitätsindex im Sinne der Definition, weil der Index-Wert nach der Löschung von der Distanz vor der Löschung abhängt)

Elektrisches Netzwerk

- einfacher ungerichteter zusammenhängender Graph $G = (V, E)$
- Kanten entsprechen Widerständen
- Widerstand (resistance) $r : E \mapsto \mathbb{R}$
- Leitwert (conductance) $c : E \mapsto \mathbb{R}$

$$c(e) = 1/r(e)$$

- externe Stromspeisung $b : V \rightarrow \mathbb{R}$
 - ▶ positive Werte: eingehender Strom
 - ▶ negative Werte: ausgehender Strom
- Bedingung

$$\sum_{v \in V} b(v) = 0$$

Elektrischer Fluss

Beliebige Kantenorientierung \vec{E} zur Unterscheidung der Flussrichtungen

Definition (Kirchhoffsche Gesetze)

Funktion $x : \vec{E} \mapsto \mathbb{R}$ heißt (elektrischer) Strom falls

$$\forall v \in V : \sum_{(v,w) \in \vec{E}} x(v,w) - \sum_{(w,v) \in \vec{E}} x(w,v) = b(v) \quad (\text{Knotensatz})$$

(negative Werte = Fluss entgegen der Kantenrichtung)

und für jeden Kreis C in G gilt:

$$\sum_{\vec{e} \in C} x(\vec{e}) = 0 \quad (\text{Maschensatz})$$

(Subtraktion für Kanten entgegen der Kreisrichtung)

Elektrisches Potential

Definition

Funktion $p: V \mapsto \mathbb{R}$ heißt (elektrisches) Potential falls

$$\forall (v, w) \in \vec{E}: \quad p(v) - p(w) = \frac{x(v, w)}{c(v, w)}$$

- Jedes elektrische Netzwerk $N = (G, c)$ hat für eine festgelegte Einspeisung b einen eindeutig bestimmten Strom,
- ebenso Potential (bis auf einen additiven Term), also eigentlich Potentialdifferenzen (Spannung)

Laplacian Matrix

- $L(N)$ für Netzwerk $N = (V, E, c)$:

$$L_{vw} = \begin{cases} \sum_{e \ni v} c(e) & \text{if } v = w \\ -c(e) & \text{if } e = \{v, w\} \\ 0 & \text{sonst} \end{cases}$$

- Für gegebenes Netzwerk $N = (G, c)$ und Einspeisung b kann man ein Potential durch Lösen der Gleichung $Lp = b$ finden.
- s - t -Einheitseinspeisung (unit s - t -supply) b_{st} :
 $b_{st}(s) = 1,$
 $b_{st}(t) = -1,$
 $\forall v \in V \setminus \{s, t\} : b_{st}(v) = 0$

Current-Flow Betweenness Centrality

- Anteil an allen s, t -Einheitsflüssen
- Durchsatz eines Knotens v bezüglich **einer** s - t -Einheitsspeisung b_{st} :

$$\tau_{st}(v) = \frac{1}{2} \left(-|b_{st}(v)| + \sum_{e \ni v} |x(\vec{e})| \right)$$

- $-|b_{st}(v)|$: Durchsatz für Anschlussknoten auf Null
- $\frac{1}{2}$ weil ein- und ausgehender Strom aufsummiert wird
- Current Flow Betweenness:

$$c_{CB}(v) = \frac{1}{(n-1)(n-2)} \sum_{s,t \in V} \tau_{st}(v)$$

Current-Flow Closeness Centrality

- bei kürzesten Wegen war Closeness ein Maß für die durchschnittliche Entfernung eines Knotens zu allen anderen Knoten
- hier: Entfernung im Sinne von **Potentialdifferenz**
- Current-Flow Closeness:

$$c_{cc}(v) = \frac{n-1}{\sum_{t \neq v} p_{vt}(v) - p_{vt}(t)}$$

- Brandes / Fleischer (2005):
Current-Flow Closeness = Informationszentralität

$$c_f(v)^{-1} = nM_{vv} + \text{trace}(M) - \frac{2}{n},$$

wobei $M = (L + U)^{-1}$, L : Laplacian, U : Matrix mit Einsen
 $\text{trace}(M)$: Spur von M (Summe der Diagonalelemente)

Übersicht

1 Grundlagen

2 Zentralitätsindizes

3 Wiederholung: Kürzeste Wege

- Allgemeines
- Keine Gewichte: BFS
- DAGs: Topologische Sortierung
- Nicht-negative Gewichte: Dijkstra
- Monotone Priority Queues
- Beliebige Graphen / Gewichte: Bellman-Ford
- All Pairs Shortest Paths

4 Algorithmen für Zentralitätsindizes

Kürzeste Wege

Zentrale Frage: Wie kommt man am schnellsten von A nach B?

Fälle:

- Kantenkosten 1
- DAG, beliebige Kantenkosten
- beliebiger Graph, positive Kantenkosten
- beliebiger Graph, beliebige Kantenkosten

Kürzeste-Wege-Problem

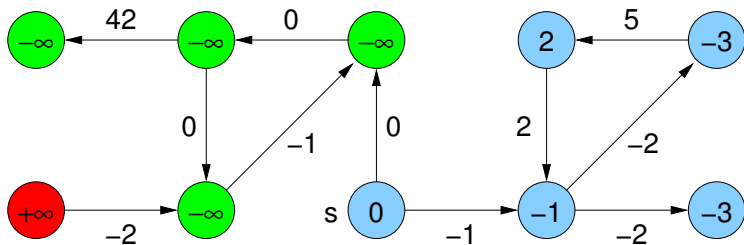
gegeben:

- gerichteter Graph $G = (V, E)$
- Kantenkosten $c : E \mapsto \mathbb{R}$

2 Varianten:

- SSSP (single source shortest paths):
kürzeste Wege von einer Quelle zu allen anderen Knoten
- APSP (all pairs shortest paths):
kürzeste Wege zwischen allen Paaren

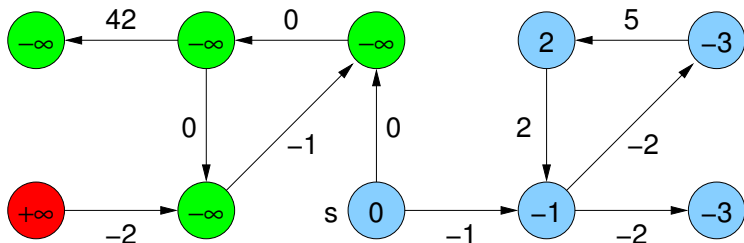
Distanzen



$\mu(s, v)$: Distanz von s nach v

$$\mu(s, v) = \begin{cases} +\infty & \text{kein Weg von } s \text{ nach } v \\ -\infty & \text{Weg beliebig kleiner Kosten von } s \text{ nach } v \\ \min\{c(p) : p \text{ ist Weg von } s \text{ nach } v\} & \end{cases}$$

Distanzen



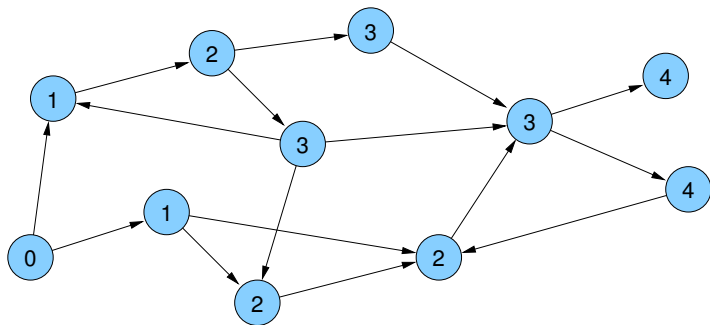
Wann sind die Kosten $-\infty$?

wenn es einen **Kreis mit negativer Gewichtssumme** gibt
(hinreichende und notwendige Bedingung)

Kürzeste Wege

Graph mit Kantenkosten 1:

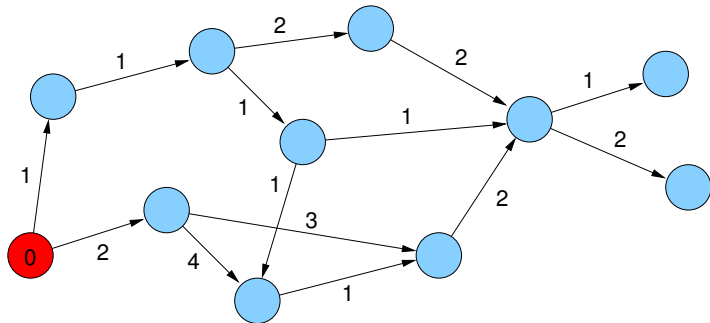
⇒ Breitensuche (BFS)



Kürzeste Wege

Beliebige Kantengewichte in DAGs

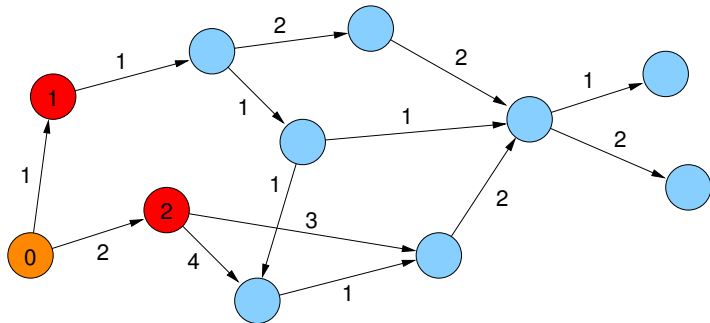
Einfache Breitensuche funktioniert nicht.



Kürzeste Wege

Beliebige Kantengewichte in DAGs

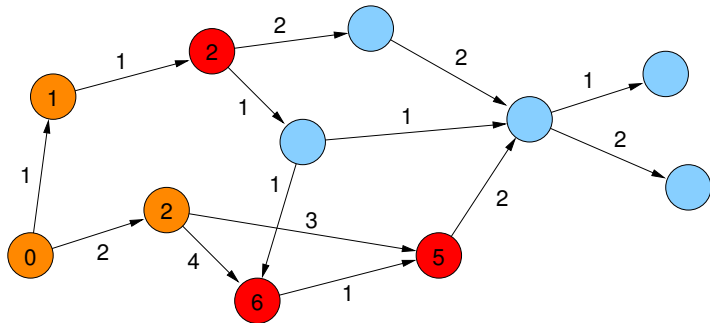
Einfache Breitensuche funktioniert nicht.



Kürzeste Wege

Beliebige Kantengewichte in DAGs

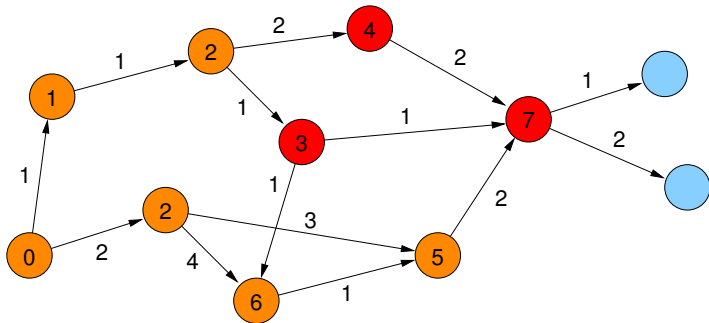
Einfache Breitensuche funktioniert nicht.



Kürzeste Wege

Beliebige Kantengewichte in DAGs

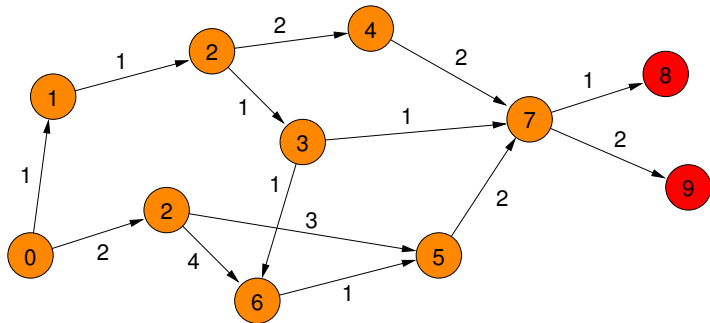
Einfache Breitensuche funktioniert nicht.



Kürzeste Wege

Beliebige Kantengewichte in DAGs

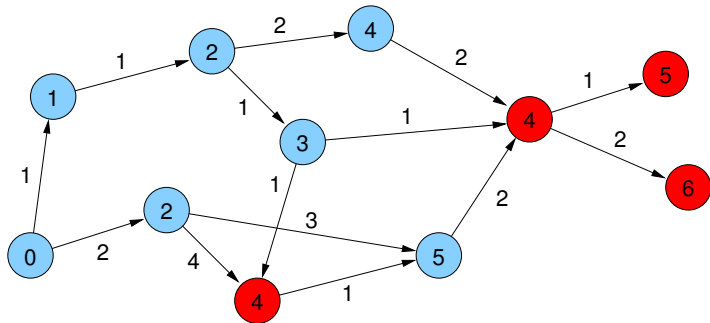
Einfache Breitensuche funktioniert nicht.



Kürzeste Wege

Beliebige Kantengewichte in DAGs

Einfache Breitensuche funktioniert nicht.

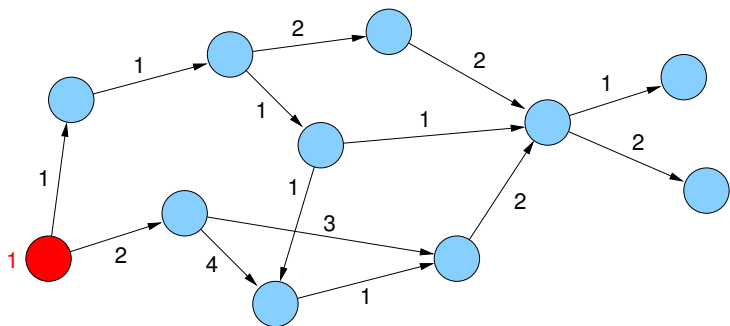


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie: in DAGs gibt es **topologische Sortierung**

(für alle Kanten $e=(v,w)$ gilt $\text{topoNum}(v) < \text{topoNum}(w)$)

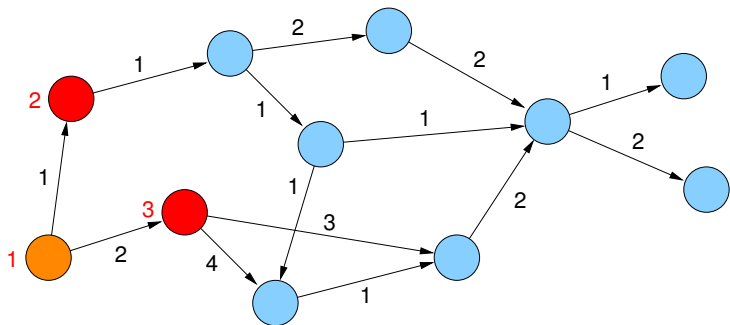


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie: in DAGs gibt es **topologische Sortierung**

(für alle Kanten $e=(v,w)$ gilt $\text{topoNum}(v) < \text{topoNum}(w)$)

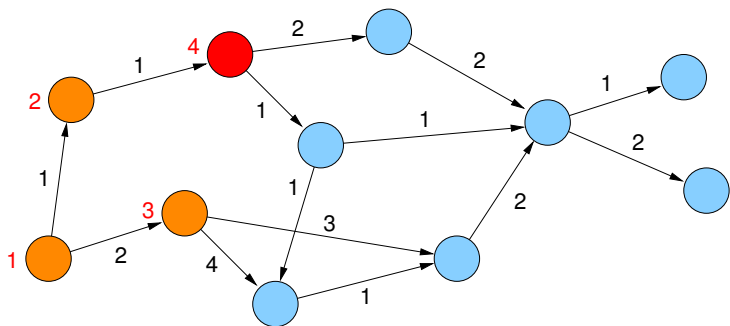


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie: in DAGs gibt es **topologische Sortierung**

(für alle Kanten $e=(v,w)$ gilt $\text{topoNum}(v) < \text{topoNum}(w)$)

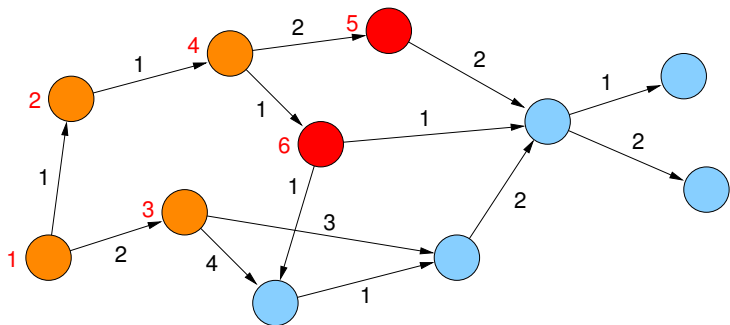


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie: in DAGs gibt es **topologische Sortierung**

(für alle Kanten $e=(v,w)$ gilt $\text{topoNum}(v) < \text{topoNum}(w)$)

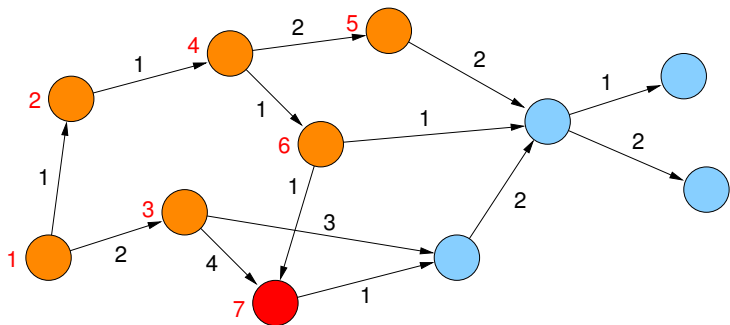


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie: in DAGs gibt es **topologische Sortierung**

(für alle Kanten $e=(v,w)$ gilt $\text{topoNum}(v) < \text{topoNum}(w)$)

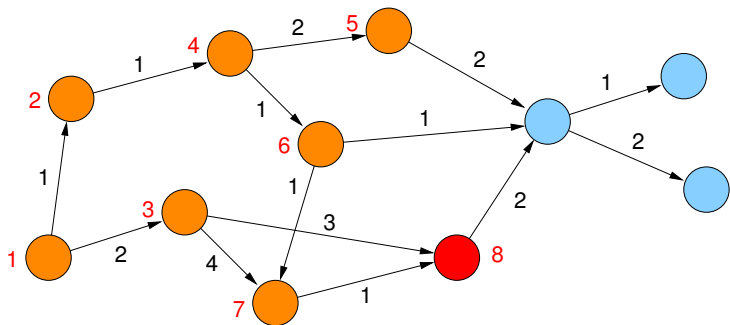


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie: in DAGs gibt es **topologische Sortierung**

(für alle Kanten $e=(v,w)$ gilt $\text{topoNum}(v) < \text{topoNum}(w)$)

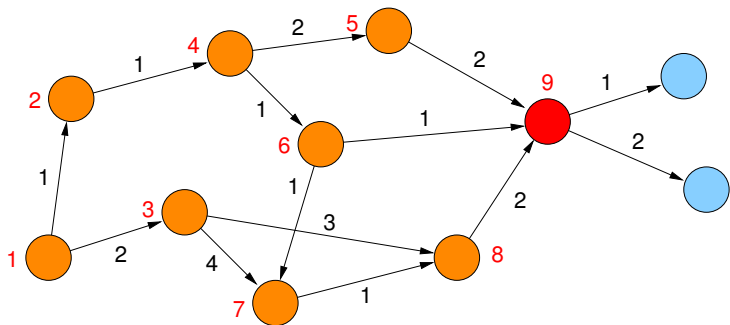


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie: in DAGs gibt es **topologische Sortierung**

(für alle Kanten $e=(v,w)$ gilt $\text{topoNum}(v) < \text{topoNum}(w)$)

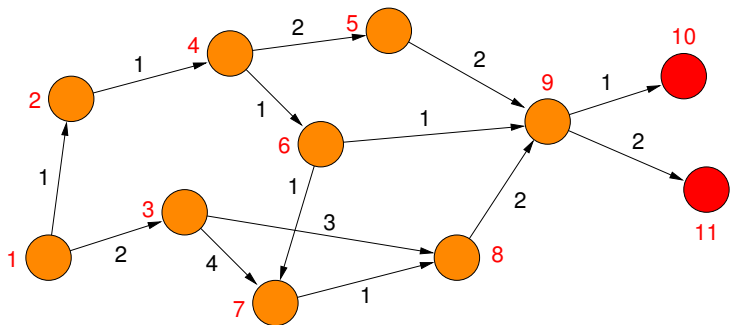


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie: in DAGs gibt es **topologische Sortierung**

(für alle Kanten $e=(v,w)$ gilt $\text{topoNum}(v) < \text{topoNum}(w)$)

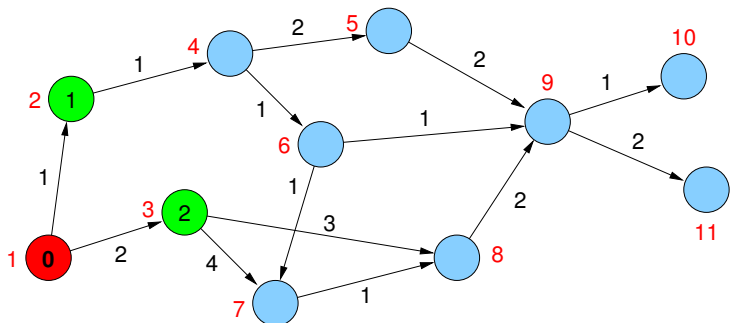


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie:

- betrachte Knoten in Reihenfolge der topologischen Sortierung
- aktualisiere Distanzwerte

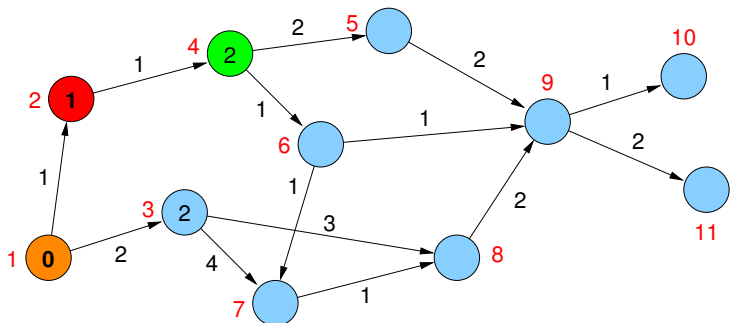


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie:

- betrachte Knoten in Reihenfolge der topologischen Sortierung
- aktualisiere Distanzwerte

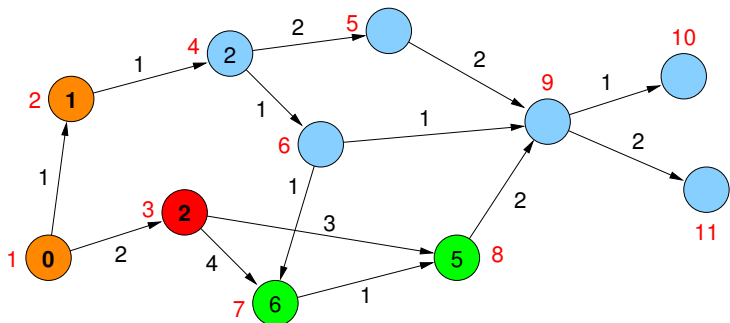


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie:

- betrachte Knoten in Reihenfolge der topologischen Sortierung
- aktualisiere Distanzwerte

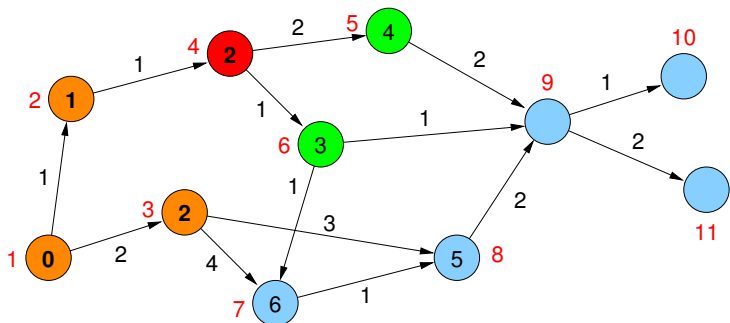


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie:

- betrachte Knoten in Reihenfolge der topologischen Sortierung
- aktualisiere Distanzwerte

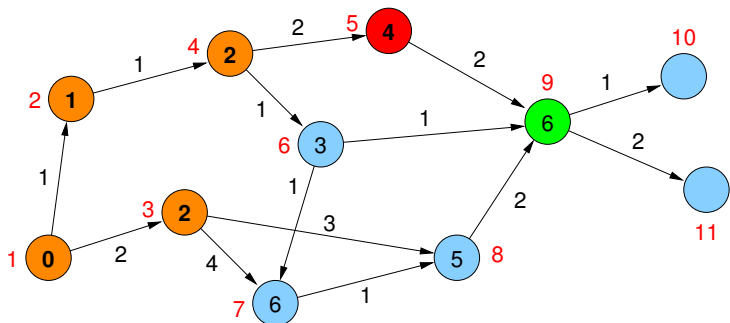


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie:

- betrachte Knoten in Reihenfolge der topologischen Sortierung
- aktualisiere Distanzwerte

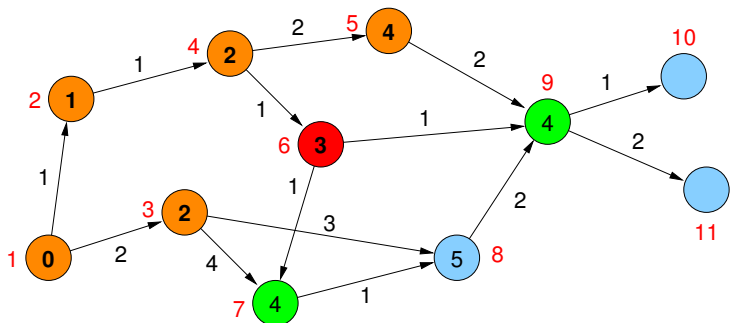


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie:

- betrachte Knoten in Reihenfolge der topologischen Sortierung
- aktualisiere Distanzwerte

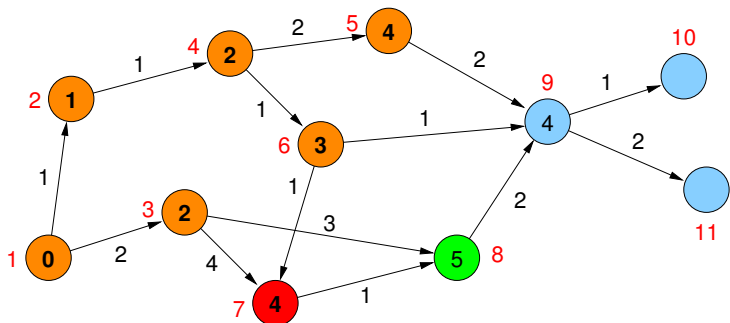


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie:

- betrachte Knoten in Reihenfolge der topologischen Sortierung
- aktualisiere Distanzwerte

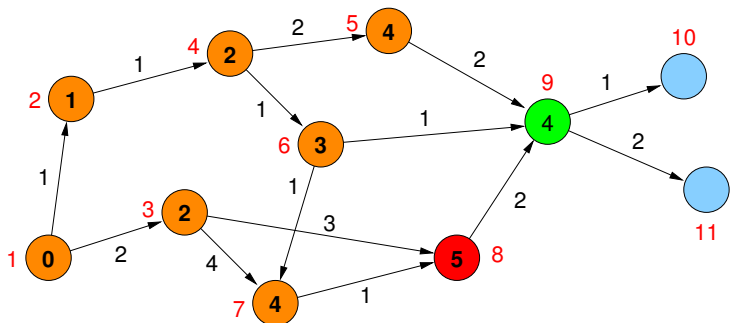


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie:

- betrachte Knoten in Reihenfolge der topologischen Sortierung
- aktualisiere Distanzwerte

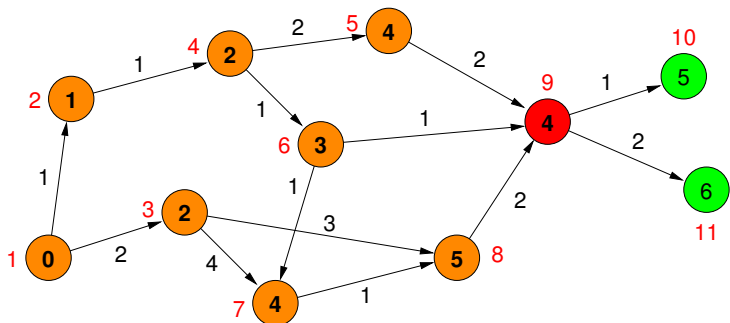


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie:

- betrachte Knoten in Reihenfolge der topologischen Sortierung
- aktualisiere Distanzwerte

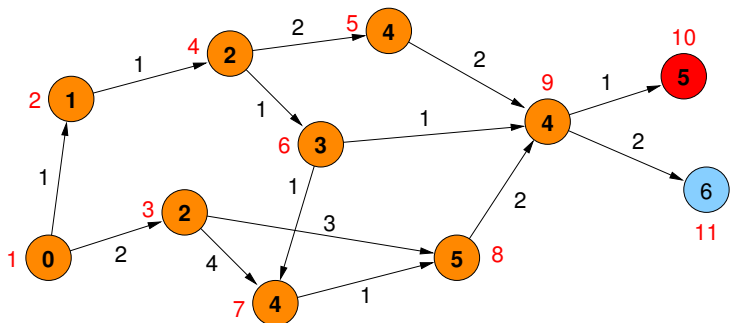


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie:

- betrachte Knoten in Reihenfolge der topologischen Sortierung
- aktualisiere Distanzwerte

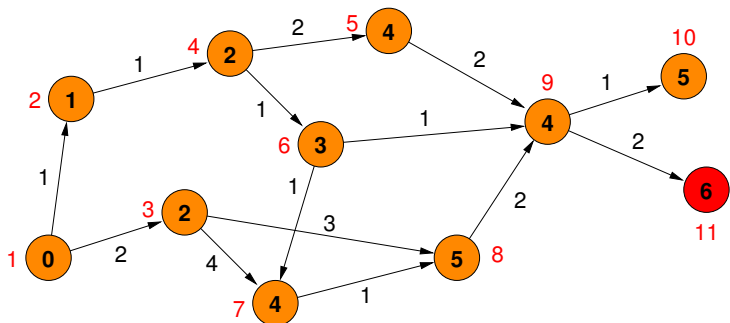


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie:

- betrachte Knoten in Reihenfolge der topologischen Sortierung
- aktualisiere Distanzwerte



Kürzeste Wege

Beliebige Kantengewichte in DAGs

Topologische Sortierung – warum funktioniert das?

- betrachte einen kürzesten Weg von s nach v
- der ganze Pfad beachtet die topologische Sortierung
- d.h., die Distanzen werden in der Reihenfolge der Knoten vom Anfang des Pfades zum Ende hin betrachtet
- damit ergibt sich für v der richtige Distanzwert

- ein Knoten x kann auch nie einen Wert erhalten, der echt kleiner als seine Distanz zu s ist
- die Kantenfolge von s zu x , die jeweils zu den Distanzwerten an den Knoten geführt hat, wäre dann ein kürzerer Pfad (Widerspruch)

Kürzeste Wege

Beliebige Kantengewichte in DAGs

Allgemeine Strategie:

- Anfang: setze $d(s) = 0$ und für alle anderen Knoten v setze $d(v) = \infty$
- besuche Knoten in einer Reihenfolge, die sicherstellt, dass **mindestens ein** kürzester Weg von s zu jedem v in der Reihenfolge seiner Knoten besucht wird
- für jeden besuchten Knoten v aktualisiere die Distanzen der Knoten w mit $(v, w) \in E$, d.h. setze

$$d(w) = \min\{ d(w), d(v) + c(v, w) \}$$

Kürzeste Wege

Topologische Sortierung

- verwende **FIFO-Queue q**
- verwalte für jeden Knoten einen **Zähler für die noch nicht markierten eingehenden Kanten**
- initialisiere q mit allen Knoten, die keine eingehende Kante haben (Quellen)
- nimm nächsten Knoten v aus q und markiere alle $(v, w) \in E$, d.h. dekrementiere Zähler für w
- falls der Zähler von w dabei Null wird, füge w in q ein
- wiederhole das, bis q leer wird

Kürzeste Wege

Topologische Sortierung

Korrektheit

- Knoten wird erst dann nummeriert, wenn alle Vorgänger nummeriert sind

Laufzeit

- für die Anfangswerte der Zähler muss der Graph einmal traversiert werden $\mathcal{O}(n + m)$
- danach wird jede Kante genau einmal betrachtet

⇒ gesamt: $\mathcal{O}(n + m)$

Test auf DAG-Eigenschaft

- topologische Sortierung erfasst genau dann **alle** Knoten, wenn der Graph ein **DAG** ist
- bei gerichteten Kreisen erhalten diese Knoten keine Nummer

Kürzeste Wege

DAG-Strategie

- 1 Topologische Sortierung der Knoten
Laufzeit $\mathcal{O}(n + m)$
- 2 Aktualisierung der Distanzen gemäß der topologischen Sortierung
Laufzeit $\mathcal{O}(n + m)$

Gesamtlaufzeit: $\mathcal{O}(n + m)$

Kürzeste Wege für beliebige Graphen mit nicht-negativen Gewichten

Gegeben:

- **beliebiger** Graph
(gerichtet oder ungerichtet, muss diesmal kein DAG sein)
- mit **nicht-negativen** Kantengewichten

⇒ keine Knoten mit Distanz $-\infty$

Problem:

- besuche Knoten eines kürzesten Weges in der richtigen Reihenfolge
- wie bei Breitensuche, jedoch diesmal auch mit Distanzen $\neq 1$

Lösung:

- besuche Knoten in der Reihenfolge der kürzesten Distanz zum Startknoten s

Kürzeste Pfade: SSSP / Dijkstra

Algorithmus 1: Dijkstra-Algorithmus

Input : $G = (V, E)$, $c : E \rightarrow \mathbb{R}$, $s \in V$

Output : Distanzen $d(s, v)$ zu allen $v \in V$

$P = \emptyset$; $T = V$;

$d(s, v) = \infty$ for all $v \in V \setminus s$;

$d(s, s) = 0$; $pred(s) = 0$;

while $P \neq V$ **do**

$v = \operatorname{argmin}_{v \in T} \{d(s, v)\}$;

$P = P \cup v$; $T = T \setminus v$;

forall the $(v, w) \in E$ **do**

if $d(s, w) > d(s, v) + c(v, w)$ **then**

$d(s, w) = d(s, v) + c(v, w)$;

$pred(w) = v$;

Algorithmus 2: Dijkstra-Algorithmus für SSSP

Input : $G = (V, E)$, $c : E \rightarrow \mathbb{R}_{\geq 0}$, $s \in V$

Output : Distanzen $d[v]$ von s zu allen $v \in V$

$d[v] = \infty$ for all $v \in V \setminus s$;

$d[s] = 0$; $pred[s] = \perp$;

$pq = \langle \rangle$; $pq.insert(s, 0)$;

while $\neg pq.empty()$ **do**

$v = pq.deleteMin()$;

forall the $(v, w) \in E$ **do**

$newDist = d[v] + c(v, w)$;

if $newDist < d[w]$ **then**

$pred[w] = v$;

if $d[w] == \infty$ **then** $pq.insert(w, newDist)$;

else $pq.decreaseKey(w, newDist)$;

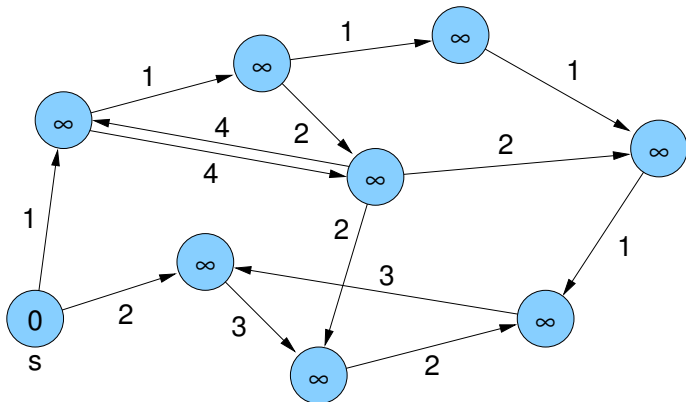
$d[w] = newDist$;

Dijkstra-Algorithmus

- setze Startwert $d(s, s) = 0$ und zunächst $d(s, v) = \infty$
- verwende **Prioritätswarteschlange**, um die Knoten zusammen mit ihren aktuellen Distanzen zu speichern
- am Anfang nur Startknoten (mit Distanz 0) in Priority Queue
- dann immer nächsten Knoten v (mit kleinster Distanz) entnehmen, endgültige Distanz dieses Knotens v steht nun fest
- betrachte alle Nachbarn von v , füge sie ggf. in die PQ ein bzw. aktualisiere deren Priorität in der PQ

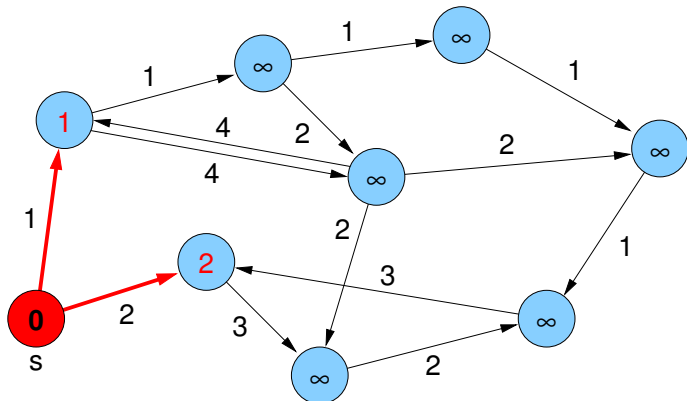
Dijkstra-Algorithmus

Beispiel:



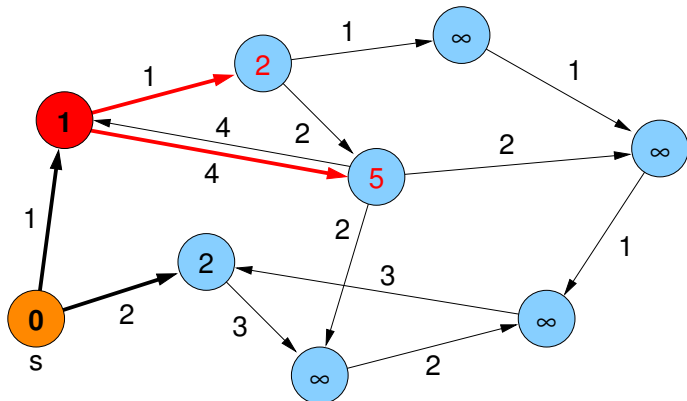
Dijkstra-Algorithmus

Beispiel:



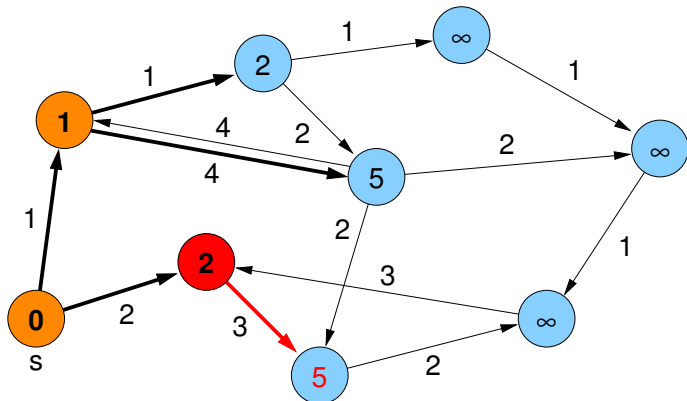
Dijkstra-Algorithmus

Beispiel:



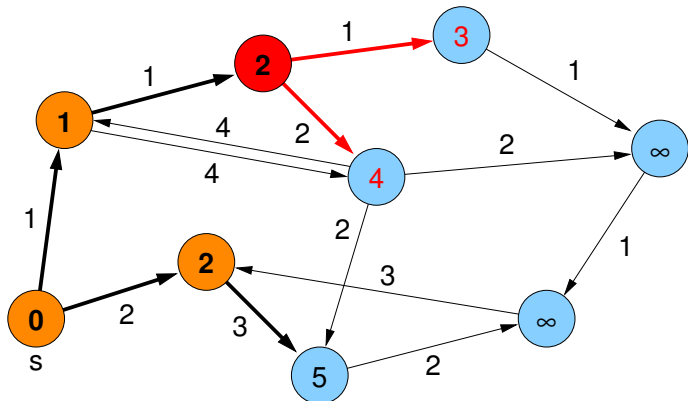
Dijkstra-Algorithmus

Beispiel:



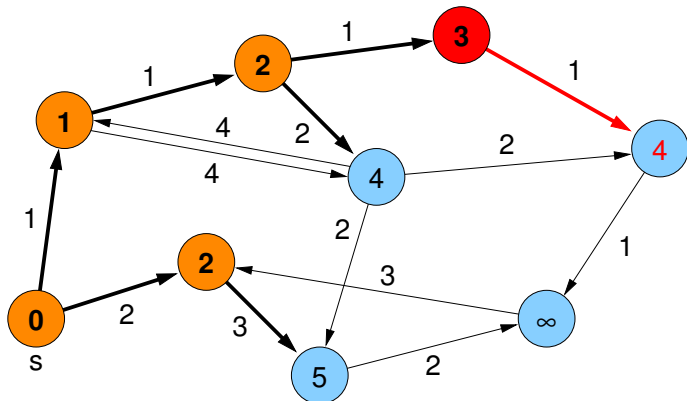
Dijkstra-Algorithmus

Beispiel:



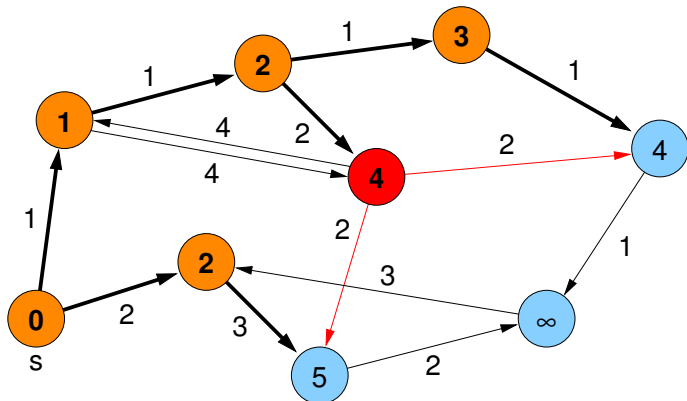
Dijkstra-Algorithmus

Beispiel:



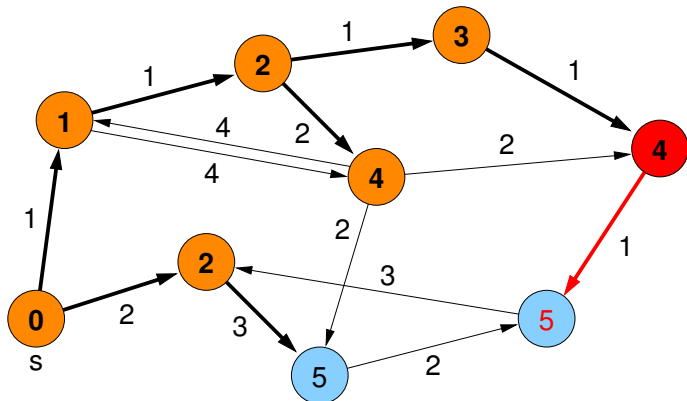
Dijkstra-Algorithmus

Beispiel:



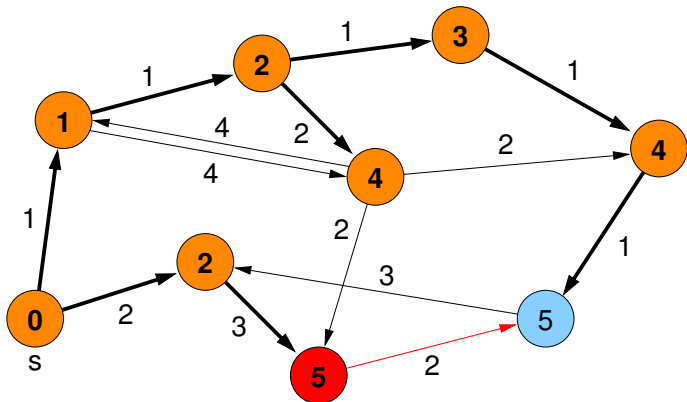
Dijkstra-Algorithmus

Beispiel:



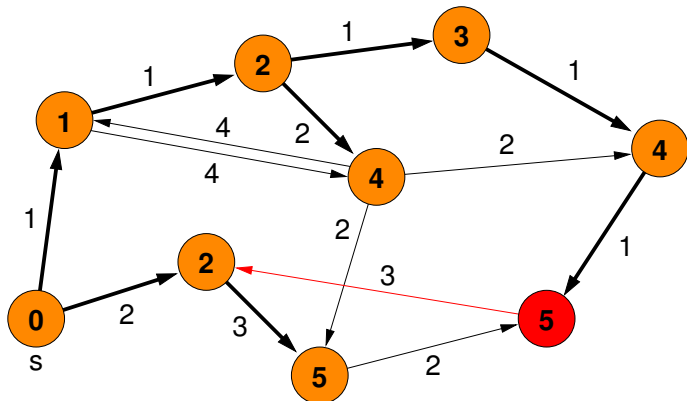
Dijkstra-Algorithmus

Beispiel:



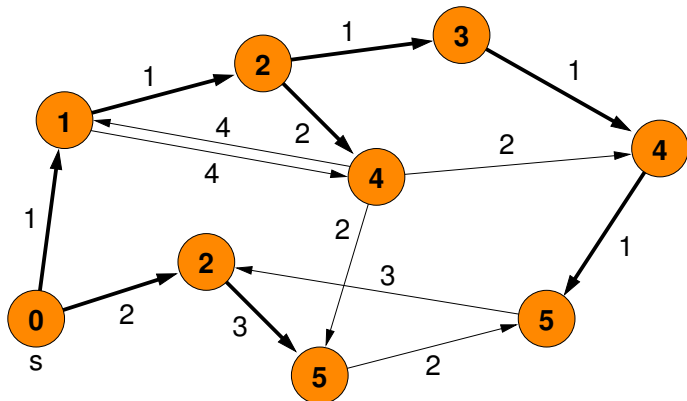
Dijkstra-Algorithmus

Beispiel:



Dijkstra-Algorithmus

Beispiel:



Dijkstra-Algorithmus

Korrektheit:

- Annahme: Algorithmus liefert für w einen **zu kleinen** Wert $d(s, w)$
- sei w der erste Knoten, für den die Distanz falsch festgelegt wird (kann nicht s sein, denn die Distanz $d(s, s)$ bleibt immer 0)
- kann nicht sein, weil $d(s, w)$ **nur dann** aktualisiert wird, wenn man über einen von s schon erreichten Knoten v mit Distanz $d(s, v)$ den Knoten w über die Kante (v, w) mit Distanz $d(s, v) + c(v, w)$ erreichen kann
- d.h. $d(s, v)$ müsste schon falsch gewesen sein (Widerspruch zur Annahme, dass w der erste Knoten mit falscher Distanz war)

Dijkstra-Algorithmus

- Annahme: Algorithmus liefert für w einen **zu großen** Wert $d(s, w)$
- sei w der Knoten mit der kleinsten (wirklichen) Distanz, für den der Wert $d(s, w)$ falsch festgelegt wird (wenn es davon mehrere gibt, der Knoten, für den die Distanz zuletzt festgelegt wird)
- kann nicht sein, weil $d(s, w)$ **immer** aktualisiert wird, wenn man über einen von s schon erreichten Knoten v mit Distanz $d(s, v)$ den Knoten w über die Kante (v, w) mit Distanz $d(s, v) + c(v, w)$ erreichen kann (dabei steht $d(s, v)$ immer schon fest, so dass auch die Länge eines kürzesten Wegs über v zu w richtig berechnet wird)
- d.h. entweder wurde auch der Wert von v zu klein berechnet (Widerspruch zur Def. von w) oder die Distanz von v wurde noch nicht festgesetzt
- weil die berechneten Distanzwerte monoton wachsen, kann letzteres nur passieren, wenn v die gleiche Distanz hat wie w (auch Widerspruch zur Def. von w)

Dijkstra-Algorithmus

- Datenstruktur: Prioritätswarteschlange
(z.B. Fibonacci Heap: amortisierte Komplexität $\mathcal{O}(1)$ für `insert` und `decreaseKey`, $\mathcal{O}(\log n)$ `deleteMin`)
- Komplexität:
 - ▶ $\mathcal{O}(n)$ `insert`
 - ▶ $\mathcal{O}(n)$ `deleteMin`
 - ▶ $\mathcal{O}(m)$ `decreaseKey` $\Rightarrow \mathcal{O}(m + n \log n)$
- aber: nur für nichtnegative Kantengewichte(!)

Monotone Priority Queues

Beobachtung:

- aktuelles Distanz-Minimum der verbleibenden Knoten ist beim Dijkstra-Algorithmus **monoton wachsend**

Monotone Priority Queue

- Folge der entnommenen Elemente hat monoton steigende Werte
- effizientere Implementierung möglich, falls Kantengewichte **ganzzahlig**

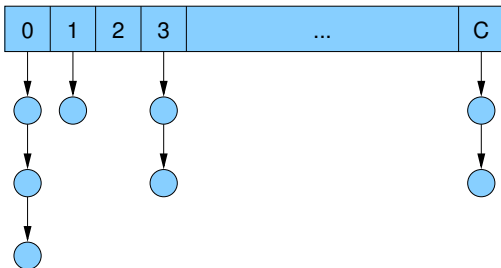
Annahme: alle **Kantengewichte** im Bereich $[0, C]$

Konsequenz für Dijkstra-Algorithmus:

⇒ enthaltene Distanzwerte immer im Bereich $[d, d + C]$

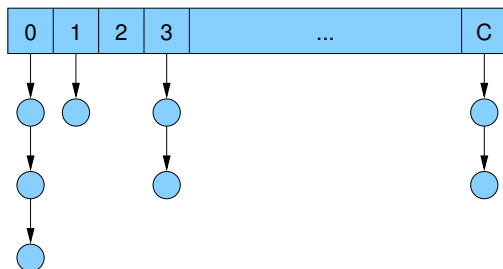
Bucket Queue

- Array **B** aus $C + 1$ Listen
- Variable d_{\min} für aktuelles Distanzminimum $\text{mod}(C + 1)$



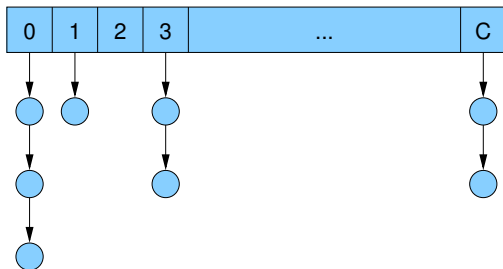
Bucket Queue

- jeder Knoten v mit aktueller Distanz $d[v]$ in Liste $B[d[v] \bmod (C + 1)]$
- alle Knoten in Liste $B[d]$ haben dieselbe Distanz, weil alle aktuellen Distanzen im Bereich $[d, d + C]$ liegen



Bucket Queue / Operationen

- **insert**(v): fügt v in Liste $B[d[v] \bmod (C + 1)]$ ein ($\mathcal{O}(1)$)
- **decreaseKey**(v): entfernt v aus momentaner Liste ($\mathcal{O}(1)$ falls Handle auf Listenelement in v gespeichert) und fügt v in Liste $B[d[v] \bmod (C + 1)]$ ein ($\mathcal{O}(1)$)
- **deleteMin**(\cdot): solange $B[d_{\min}] = \emptyset$, setze $d_{\min} = (d_{\min} + 1) \bmod (C + 1)$.
Nimm dann einen Knoten u aus $B[d_{\min}]$ heraus ($\mathcal{O}(C)$)



Dijkstra mit Bucket Queue

- insert, decreaseKey: $\mathcal{O}(1)$
- deleteMin: $\mathcal{O}(C)$
- Dijkstra: $\mathcal{O}(m + C \cdot n)$
- lässt sich mit **Radix Heaps** noch verbessern
- verwendet exponentiell wachsende Bucket-Größen
- Details in der Vorlesung Effiziente Algorithmen und Datenstrukturen
- Laufzeit ist dann $\mathcal{O}(m + n \log C)$

Kürzeste Wege für beliebige Graphen mit beliebigen Gewichten

Gegeben:

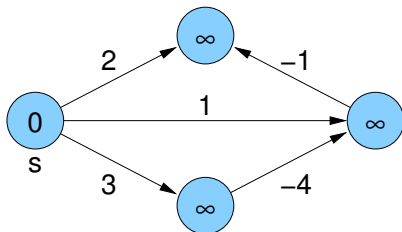
- **beliebiger** Graph mit **beliebigen** Kantengewichten
- ⇒ Anhängen einer Kante an einen Weg kann zur Verkürzung des Weges (Kantengewichtssumme) führen (wenn Kante negatives Gewicht hat)
- ⇒ es kann negative Kreise und Knoten mit Distanz $-\infty$ geben

Problem:

- besuche Knoten eines kürzesten Weges in der richtigen Reihenfolge
- Dijkstra kann nicht mehr verwendet werden, weil Knoten nicht unbedingt in der Reihenfolge der kürzesten Distanz zum Startknoten s besucht werden

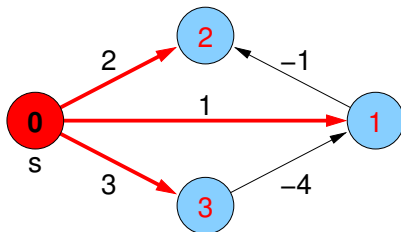
Kürzeste Wege für beliebige Graphen mit beliebigen Gewichten

Gegenbeispiel für Dijkstra-Algorithmus:



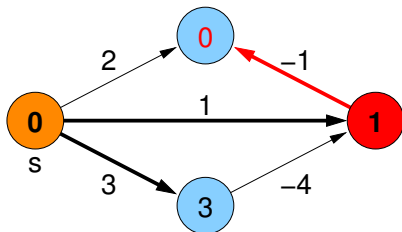
Kürzeste Wege für beliebige Graphen mit beliebigen Gewichten

Gegenbeispiel für Dijkstra-Algorithmus:



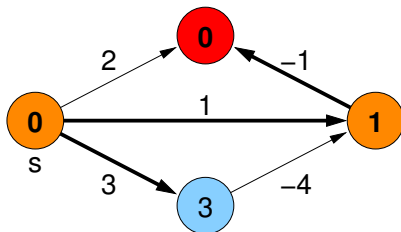
Kürzeste Wege für beliebige Graphen mit beliebigen Gewichten

Gegenbeispiel für Dijkstra-Algorithmus:



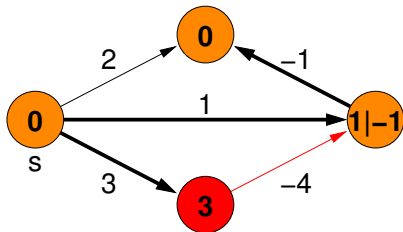
Kürzeste Wege für beliebige Graphen mit beliebigen Gewichten

Gegenbeispiel für Dijkstra-Algorithmus:



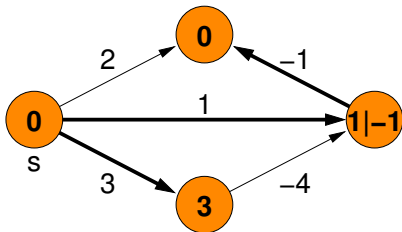
Kürzeste Wege für beliebige Graphen mit beliebigen Gewichten

Gegenbeispiel für Dijkstra-Algorithmus:



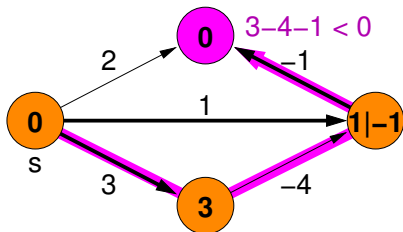
Kürzeste Wege für beliebige Graphen mit beliebigen Gewichten

Gegenbeispiel für Dijkstra-Algorithmus:



Kürzeste Wege für beliebige Graphen mit beliebigen Gewichten

Gegenbeispiel für Dijkstra-Algorithmus:



Kürzeste Wege für beliebige Graphen mit beliebigen Gewichten

Lemma

Für jeden von s erreichbaren Knoten v mit $d(s, v) > -\infty$ gibt es einen **einfachen** Pfad (ohne Kreis) von s nach v der Länge $d(s, v)$.

Beweis.

Betrachte kürzesten Weg mit Kreis(en):

- Kreis mit Kantengewichtssumme > 0 nicht enthalten:
Entfernen des Kreises würde Kosten verringern
- Kreis mit Kantengewichtssumme $= 0$:
Entfernen des Kreises lässt Kosten unverändert
- Kreis mit Kantengewichtssumme < 0 :
Distanz von s ist $-\infty$



Bellman-Ford-Algorithmus

Folgerung

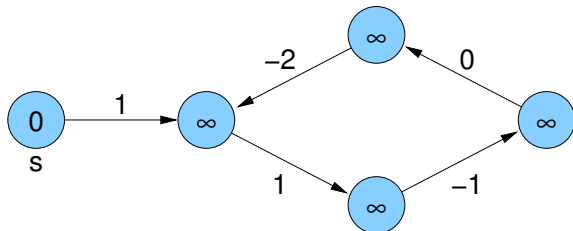
In einem Graph mit n Knoten gibt es für jeden erreichbaren Knoten v mit $d(s, v) > -\infty$ einen kürzesten Weg bestehend aus $< n$ Kanten zwischen s und v .

Strategie:

- anstatt kürzeste Pfade in Reihenfolge wachsender Gewichtssumme zu berechnen, betrachte sie in Reihenfolge steigender Kantenzahl
- durchlaufe $(n - 1)$ -mal alle Kanten im Graph und aktualisiere die Distanz
- dann alle kürzesten Wege berücksichtigt

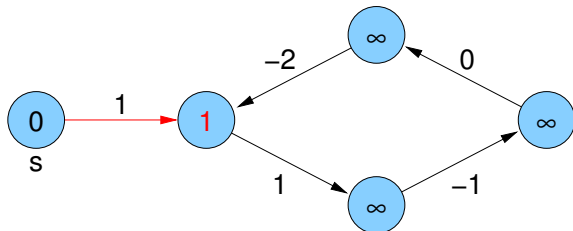
Bellman-Ford-Algorithmus

Problem: Erkennung negativer Kreise



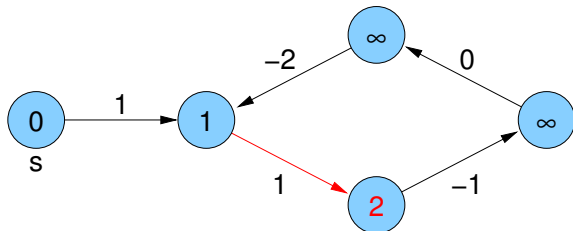
Bellman-Ford-Algorithmus

Problem: Erkennung negativer Kreise



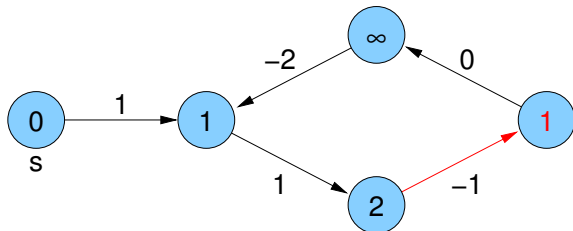
Bellman-Ford-Algorithmus

Problem: Erkennung negativer Kreise



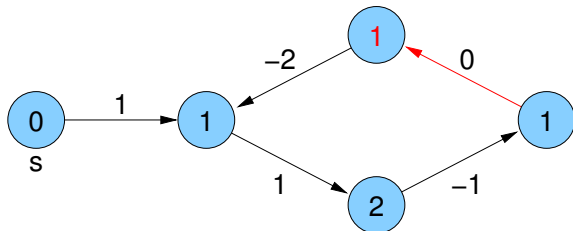
Bellman-Ford-Algorithmus

Problem: Erkennung negativer Kreise



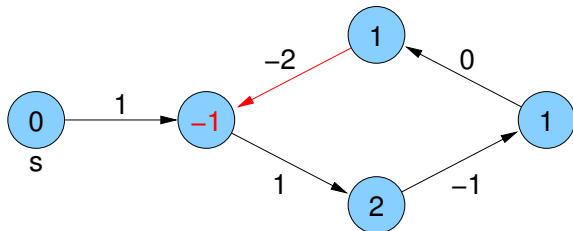
Bellman-Ford-Algorithmus

Problem: Erkennung negativer Kreise



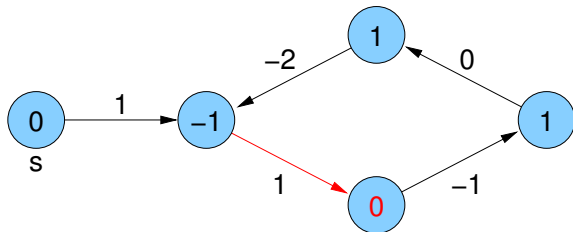
Bellman-Ford-Algorithmus

Problem: Erkennung negativer Kreise



Bellman-Ford-Algorithmus

Problem: Erkennung negativer Kreise



Bellman-Ford-Algorithmus

Keine Distanzniedrigung möglich:

- Annahme: zu einem Zeitpunkt gilt für alle Kanten (v, w)
 $d[v] + c(v, w) \geq d[w]$
- dann gilt (per Induktion) für jeden Weg p von s nach w , dass
 $d[s] + c(p) \geq d[w]$ für alle Knoten w
- falls sichergestellt, dass zu jedem Zeitpunkt für kürzesten Weg p
von s nach w gilt $d[w] \geq c(p)$, dann ist $d[w]$ zum Schluss genau die
Länge eines kürzesten Pfades von s nach w (also korrekte Distanz)

Bellman-Ford-Algorithmus

Zusammenfassung:

- **keine Distanzniedrigung** mehr möglich
($d[v] + c(v, w) \geq d[w]$ für alle w):
fertig, $d[w]$ korrekt für alle w
- **Distanzniedrigung möglich** selbst noch in n -ter Runde,
also $\exists w$ mit $d[v] + c(v, w) < d[w]$:
Es gibt einen negativen Kreis, also Knoten w mit Distanz $-\infty$.

Bellman-Ford-Algorithmus

```
void BellmanFord(Node s) {  
    foreach ( $v \in V$ )  $d[v] = \infty$ ;  
     $d[s] = 0$ ;    parent[s] = s;  
    for (int  $i = 0$ ;  $i < n - 1$ ;  $i++$ ) {    //  $n - 1$  Runden  
        foreach ( $e = (v, w) \in E$ )  
            if ( $d[v] + c(e) < d[w]$ ) {    // kürzerer Weg?  
                 $d[w] = d[v] + c(e)$ ;  
                parent[w] = v;  
            }  
    }  
  
    foreach ( $e = (v, w) \in E$ )  
        if ( $d[v] + c(e) < d[w]$ ) {    // kürzerer Weg in  $n$ -ter Runde?  
            infect(w);  
        }  
}
```

Bellman-Ford-Algorithmus

```
void infect(Node v) { //  $-\infty$ -Knoten
  if ( $d[v] > -\infty$ ) {
     $d[v] = -\infty$ ;
    foreach ( $e = (v, w) \in E$ )
      infect(w);
  }
}
```

Gesamtlaufzeit: $\mathcal{O}(m \cdot n)$

Bellman-Ford-Algorithmus

Bestimmung der **Knoten mit Distanz $-\infty$** :

- betrachte alle Knoten, die in der n -ten Phase noch Distanzverbesserung erfahren
- aus jedem Kreis mit negativem Gesamtgewicht muss mindestens ein Knoten dabei sein
- jeder von diesen Knoten aus erreichbare Knoten muss Distanz $-\infty$ bekommen
- das erledigt hier die **infect**-Funktion
- wenn ein Knoten zweimal auftritt (d.h. der Wert ist schon $-\infty$), wird die Rekursion abgebrochen

Bellman-Ford-Algorithmus

Bestimmung eines **negativen Zyklus**:

- bei den oben genannten Knoten sind vielleicht auch Knoten, die nur an negativen Kreisen über ausgehende Kanten angeschlossen sind, die selbst aber nicht Teil eines negativen Kreises sind
- Rückwärtsverfolgung der **parent**-Werte, bis sich ein Knoten wiederholt
- Kanten vom ersten bis zum zweiten Auftreten bilden **einen** negativen Zyklus

Bellman-Ford-Algorithmus

Ursprüngliche Idee der Updates vorläufiger Distanzwerte stammt von Lester R. Ford Jr.

Verbesserung (Richard E. Bellman / Edward F. Moore):

- verwalte eine **FIFO-Queue** von Knoten, zu denen ein kürzerer Pfad gefunden wurde und deren Nachbarn am anderen Ende ausgehender Kanten noch auf kürzere Wege geprüft werden müssen
- wiederhole: nimm ersten Knoten aus der Queue und prüfe für jede ausgehende Kante die Distanz des Nachbarn
falls kürzerer Weg gefunden, aktualisiere Distanzwert des Nachbarn und hänge ihn an Queue an (falls nicht schon enthalten)
- Phase besteht immer aus Bearbeitung der Knoten, die **am Anfang** des Algorithmus (bzw. der Phase) in der Queue sind
(dabei kommen während der Phase schon neue Knoten ans Ende der Queue)

Kürzeste einfache Pfade bei beliebigen Kantengewichten

Achtung!

Fakt

Die Suche nach kürzesten *einfachen* Pfaden
(also ohne Knotenwiederholungen / Kreise)
in Graphen mit beliebigen Kantengewichten
(also möglichen negativen Kreisen)
ist ein *NP-vollständiges Problem*.

(Man könnte Hamilton-Pfad-Suche damit lösen.)

All Pairs Shortest Paths

gegeben:

- Graph mit beliebigen Kantengewichten, der aber keine negativen Kreise enthält

gesucht:

- Distanzen / kürzeste Pfade zwischen allen Knotenpaaren

Naive Strategie:

- n -mal Bellman-Ford-Algorithmus (jeder Knoten einmal als Startknoten)

$$\Rightarrow \mathcal{O}(n^2 \cdot m)$$

All Pairs Shortest Paths

Bessere Strategie:

- reduziere n Aufrufe des Bellman-Ford-Algorithmus auf n Aufrufe des Dijkstra-Algorithmus

Problem:

- Dijkstra-Algorithmus funktioniert nur für nichtnegative Kantengewichte

Lösung:

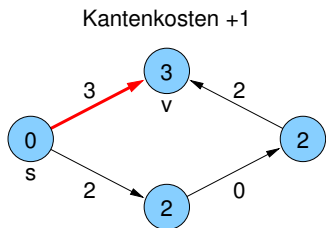
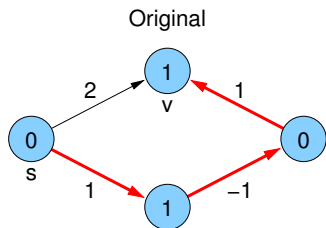
- Umwandlung in nichtnegative Kantenkosten ohne Verfälschung der kürzesten Wege

All Pairs Shortest Paths

Naive Idee:

- negative Kantengewichte eliminieren, indem auf jedes Kantengewicht der gleiche Wert c addiert wird

⇒ **verfälscht** kürzeste Pfade



All Pairs Shortest Paths

Sei $\Phi : V \mapsto \mathbb{R}$ eine Funktion, die jedem Knoten ein **Potential** zuordnet.

Modifizierte Kantenkosten von $e = (v, w)$:

$$\bar{c}(e) = \Phi(v) + c(e) - \Phi(w)$$

Lemma

Seien p und q Wege von v nach w in G .

$c(p)$ und $c(q)$ bzw. $\bar{c}(p)$ und $\bar{c}(q)$ seien die aufsummierten Kosten bzw. modifizierten Kosten der Kanten des jeweiligen Pfads.

Dann gilt für jedes Potential Φ :

$$\bar{c}(p) < \bar{c}(q) \iff c(p) < c(q)$$

All Pairs Shortest Paths

Beweis.

Sei $p = (v_1, \dots, v_k)$ beliebiger Weg und $\forall i : e_i = (v_i, v_{i+1}) \in E$

Es gilt:

$$\begin{aligned}\bar{c}(p) &= \sum_{i=1}^{k-1} \bar{c}(e_i) \\ &= \sum_{i=1}^{k-1} (\Phi(v_i) + c(e_i) - \Phi(v_{i+1})) \\ &= \Phi(v_1) + c(p) - \Phi(v_k)\end{aligned}$$

d.h. modifizierte Kosten eines Pfads hängen nur von ursprünglichen Pfadkosten und vom Potential des Anfangs- und Endknotens ab.

(Im Lemma ist $v_1 = v$ und $v_k = w$)



All Pairs Shortest Paths

Lemma

Annahme:

- Graph hat keine negativen Kreise
- alle Knoten von s aus erreichbar

Sei für alle Knoten v das Potential $\Phi(v) = d(s, v)$.

Dann gilt für alle Kanten e : $\bar{c}(e) \geq 0$

Beweis.

- für alle Knoten v gilt nach Annahme: $d(s, v) \in \mathbb{R}$ (also $\neq \pm\infty$)
- für jede Kante $e = (v, w)$ ist

$$\begin{aligned}d(s, v) + c(e) &\geq d(s, w) \\d(s, v) + c(e) - d(s, w) &\geq 0\end{aligned}$$



All Pairs Shortest Paths / Johnson-Algorithmus

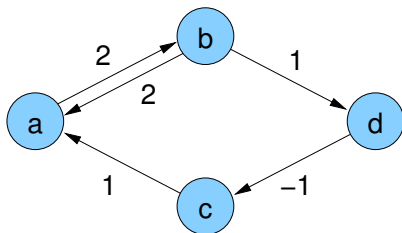
- füge **neuen Knoten s** und Kanten (s, v) für alle v hinzu mit $c(s, v) = 0$

⇒ alle Knoten erreichbar

- berechne $d(s, v)$ mit Bellman-Ford-Algorithmus
- setze $\Phi(v) = d(s, v)$ für alle v
- berechne modifizierte Kosten $\bar{c}(e)$
- berechne für alle Knoten v die Distanzen $\bar{d}(v, w)$ mittels Dijkstra-Algorithmus mit modifizierten Kantenkosten auf dem Graph ohne Knoten s
- berechne korrekte Distanzen $d(v, w) = \bar{d}(v, w) + \Phi(w) - \Phi(v)$

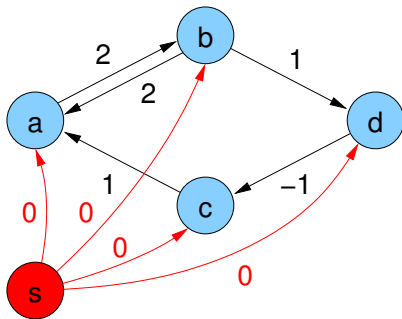
All Pairs Shortest Paths / Johnson-Algorithmus

Beispiel:



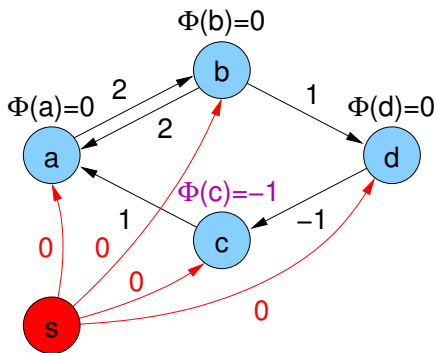
All Pairs Shortest Paths / Johnson-Algorithmus

1. künstliche Quelle s :



All Pairs Shortest Paths / Johnson-Algorithmus

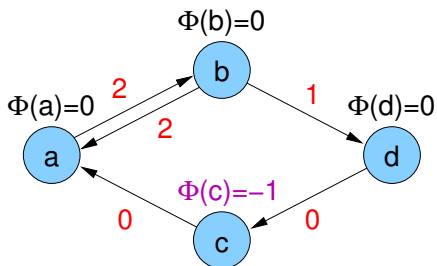
2. Bellman-Ford-Algorithmus auf s :



All Pairs Shortest Paths / Johnson-Algorithmus

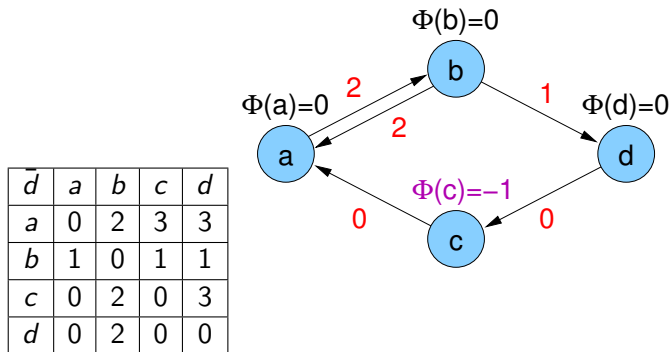
3. $\bar{c}(e)$ -Werte für alle $e = (v, w)$ berechnen:

$$\bar{c}(e) = \Phi(v) + c(e) - \Phi(w)$$



All Pairs Shortest Paths / Johnson-Algorithmus

4. Distanzen \bar{d} mit modifizierten Kantengewichten via Dijkstra:

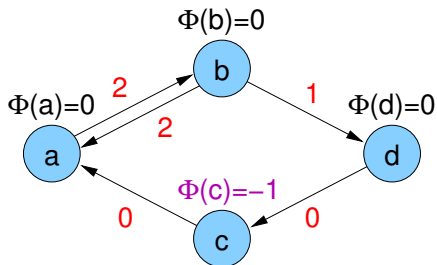


All Pairs Shortest Paths / Johnson-Algorithmus

5. korrekte Distanzen berechnen mit Formel

$$d(v, w) = \bar{d}(v, w) + \Phi(w) - \Phi(v)$$

d	a	b	c	d
a	0	2	2	3
b	1	0	0	1
c	1	3	0	4
d	0	2	-1	0



All Pairs Shortest Paths / Johnson-Algorithmus

Laufzeit:

$$\begin{aligned}T(APSP) &= \mathcal{O}(T_{\text{Bellman-Ford}}(n+1, m+n) + n \cdot T_{\text{Dijkstra}}(n, m)) \\ &= \mathcal{O}((m+n) \cdot (n+1) + n(n \log n + m)) \\ &= \mathcal{O}(m \cdot n + n^2 \log n)\end{aligned}$$

(bei Verwendung von Fibonacci Heaps)

APSP / Floyd-Warshall-Algorithmus

Grundlage:

- geht der kürzeste Weg **von u nach w über v** , dann sind auch die beiden Teile **von u nach v** und **von v nach w** kürzeste Pfade zwischen diesen Knoten
 - Annahme: alle kürzeste Wege bekannt, die nur über Zwischenknoten mit Index kleiner als k gehen
- ⇒ kürzeste Wege über Zwischenknoten mit Indizes bis einschließlich k können leicht berechnet werden:
- ▶ entweder der schon bekannte Weg über Knoten mit Indizes kleiner als k
 - ▶ oder über den Knoten mit Index k (hier im Algorithmus der Knoten v)

APSP / Floyd-Warshall-Algorithmus

Algorithmus 3: Floyd-Warshall APSP Algorithmus

Input : Graph $G = (V, E)$, $c : E \rightarrow R$

Output : Distanzen $d(u, v)$ zwischen allen $u, v \in V$

for $u, v \in V$ **do**

$d(u, v) = \infty$; $\text{pred}(u, v) = 0$;

for $v \in V$ **do** $d(v, v) = 0$;

for $\{u, v\} \in E$ **do**

$d(u, v) = c(u, v)$; $\text{pred}(u, v) = u$;

for $v \in V$ **do**

for $\{u, w\} \in V \times V$ **do**
 if $d(u, w) > d(u, v) + d(v, w)$ **then**
 $d(u, w) = d(u, v) + d(v, w)$;
 $\text{pred}(u, w) = \text{pred}(v, w)$;

APSP / Floyd-Warshall-Algorithmus

- Komplexität: $O(n^3)$
- funktioniert auch, wenn Kanten mit negativem Gewicht existieren
- Kreise negativer Länge werden nicht direkt erkannt und verfälschen das Ergebnis, sind aber indirekt am Ende an negativen Diagonaleinträgen der Distanzmatrix erkennbar

Übersicht

- 1 Grundlagen
- 2 Zentralitätsindizes
- 3 Wiederholung: Kürzeste Wege
- 4 Algorithmen für Zentralitätsindizes**
 - Berechnung der Betweenness-Zentralitäten
 - Das Absolute 1-Center Problem
 - Berechnung von Shortcut-Werten
 - Feedback Centralities
- 5 Lokale Dichte
- 6 Zusammenhang

Betweenness Centrality

$$c_B(v) = \sum_{s \in V \setminus \{v\}} \sum_{t \in V \setminus \{v\}} \delta_{st}(v) = \sum_{s \in V \setminus \{v\}} \sum_{t \in V \setminus \{v\}} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

Mögliche Berechnung:

- ① Berechne Länge und Anzahl kürzester Pfade zwischen allen Knotenpaaren
 - ② Betrachte zu jedem Knoten v alle möglichen Paare s, t und berechne den Anteil kürzester Pfade durch v
- Bedingung(Bellman-Kriterium):

$$d(s, t) = d(s, v) + d(v, t)$$

Betweenness Centrality

1 Modifiziere BFS bzw. Dijkstras Algorithmus

- ▶ Ersetze einzelne Vorgängerknoten durch eine Vorgängermenge
 $\text{pred}(s, v) = \{u \in V : \{u, v\} \in E, d(s, v) = d(s, u) + w(u, v)\}$
- ▶ Dann gilt

$$\sigma_{sv} = \sum_{u \in \text{pred}(s, v)} \sigma_{su}$$

⇒ Für *einen* Startknoten $s \in V$ kann die Anzahl kürzester Pfade zu jedem anderen Knoten

- ▶ in $\mathcal{O}(m + n \log n)$ für **gewichtete** und
- ▶ in $\mathcal{O}(m)$ für **ungewichtete** Graphen

berechnet werden.

⇒ Berechnung von σ_{st} für alle Paare $s, t \in V$:
 in $\mathcal{O}(mn + n^2 \log n)$ bzw. $\mathcal{O}(mn)$

Betweenness Centrality

- 2 Im Fall $d(s, t) = d(s, v) + d(v, t)$ gilt

$$\sigma_{st}(v) = \sigma_{sv} \cdot \sigma_{vt}$$

ansonsten ($d(s, t) < d(s, v) + d(v, t)$) ist $\sigma_{st}(v) = 0$

⇒ naive Berechnung:

$\mathcal{O}(n^2)$ pro Knoten v (Summation über alle $s \neq v \neq t$),

also insgesamt

Zeit $\mathcal{O}(n^3)$ und Platz $\mathcal{O}(n^2)$ für Speicherung aller $d(s, t)$ und σ_{st}

Betweenness Centrality (Brandes)

Abhängigkeit eines Paares (s, t) von v :

$$\delta_{st}(v) = \frac{\sigma_{st}(v)}{\sigma_{st}}$$

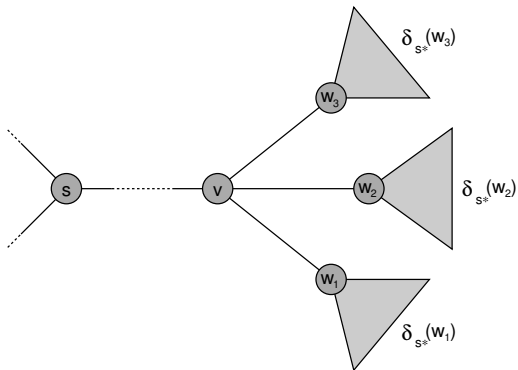
Abhängigkeit eines Startknotens s von v :

$$\delta_{s^*}(v) = \sum_{t \in V \setminus \{v\}} \delta_{st}(v)$$

Betweenness von v :

$$\Rightarrow c_B(v) = \sum_{s \in V \setminus \{v\}} \delta_{s^*}(v)$$

Betweenness Centrality (Brandes)



Betweenness Centrality (Brandes)

Lemma

Für den Fall, dass es von $s \in V$ **genau einen** kürzesten Pfad zu jedem $t \in V$ gibt, gilt

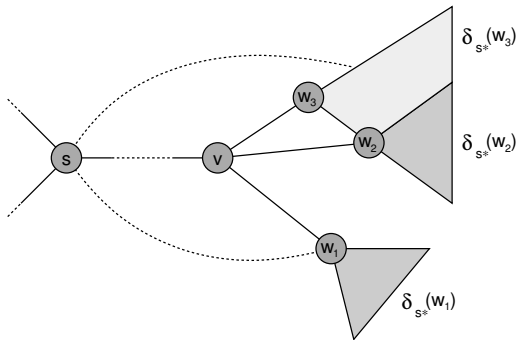
$$\delta_{s^*}(v) = \sum_{w: v \in \text{pred}(s,w)} (1 + \delta_{s^*}(w))$$

Beweis.

- Die kürzesten Pfade von s formen einen Baum.
- Also liegt v auf *allen* kürzesten Pfaden von s zu einem t oder auf *keinem*, d.h. $\delta_{st}(v)$ ist 1 oder 0.
- v liegt auf allen kürzesten Pfaden zu den Nachfolgern, sowie auf allen kürzesten Pfaden, auf denen diese Nachfolger liegen.



Betweenness Centrality (Brandes)



Im allgemeinen Fall werden die Anteile der Abhängigkeiten von den Nachfolgern über die Kanten des Kürzeste-Pfade-DAGs propagiert.

Betweenness Centrality (Brandes)

Satz

Für die Abhängigkeit $\delta_{s^*}(v)$ eines Startknotens $s \in V$ von den anderen Knoten $v \in V$ gilt:

$$\delta_{s^*}(v) = \sum_{w: v \in \text{pred}(s,w)} \frac{\sigma_{sv}}{\sigma_{sw}} (1 + \delta_{s^*}(w))$$

Betweenness Centrality (Brandes)

Beweis.

- $\delta_{st}(v) > 0$ gilt nur für die $t \in V \setminus \{s\}$, für die v auf mindestens einem kürzesten Pfad von s nach t liegt.
- Jeder solche Pfad hat exakt eine Kante $\{v, w\}$ mit $v \in \text{pred}(s, w)$ (siehe Abbildung).

Betweenness Centrality (Brandes)

Beweis.

- $\sigma_{st}(v, e)$: Anzahl kürzester s - t -Pfade, die sowohl Knoten v als auch Kante e enthalten:
- Anteil entsprechend:

$$\delta_{st}(v, e) = \frac{\sigma_{st}(v, e)}{\sigma_{st}}$$

$$\begin{aligned} \delta_{s^*}(v) &= \sum_{t \in V \setminus \{v\}} \delta_{st}(v) = \sum_{t \in V \setminus \{v\}} \left(\sum_{w: v \in \text{pred}(s, w)} \delta_{st}(v, \{v, w\}) \right) \\ &= \sum_{w: v \in \text{pred}(s, w)} \left(\sum_{t \in V \setminus \{v\}} \delta_{st}(v, \{v, w\}) \right) \end{aligned}$$

Betweenness Centrality (Brandes)

Beweis.

- Betrachte Knoten w , so dass $v \in \text{pred}(s, w)$
- σ_{sw} kürzeste Pfade von s nach w ,
davon σ_{sv} von s nach v gefolgt von Kante $\{v, w\}$
- Anteil σ_{sv}/σ_{sw} der Anzahl kürzester Pfade von s nach t über w benutzt auch die Kante $\{v, w\}$:

$$\delta_{st}(v, \{v, w\}) = \begin{cases} \frac{\sigma_{sv}}{\sigma_{sw}} & \text{falls } t = w \\ \frac{\sigma_{sv}}{\sigma_{sw}} \cdot \frac{\sigma_{st}(w)}{\sigma_{st}} & \text{falls } t \neq w \end{cases}$$

Betweenness Centrality (Brandes)

Beweis.

$$\begin{aligned}
 \delta_{s^*}(v) &= \sum_{w: v \in \text{pred}(s,w)} \left(\sum_{t \in V \setminus \{v\}} \delta_{st}(v, \{v, w\}) \right) \\
 &= \sum_{w: v \in \text{pred}(s,w)} \left(\frac{\sigma_{sv}}{\sigma_{sw}} + \sum_{t \in V \setminus \{v,w\}} \frac{\sigma_{sv}}{\sigma_{sw}} \cdot \frac{\sigma_{st}(w)}{\sigma_{st}} \right) \\
 &= \sum_{w: v \in \text{pred}(s,w)} \left(\frac{\sigma_{sv}}{\sigma_{sw}} + \frac{\sigma_{sv}}{\sigma_{sw}} \sum_{t \in V \setminus \{v,w\}} \frac{\sigma_{st}(w)}{\sigma_{st}} \right) \\
 &= \sum_{w: v \in \text{pred}(s,w)} \frac{\sigma_{sv}}{\sigma_{sw}} (1 + \delta_{s^*}(w))
 \end{aligned}$$

Betweenness Centrality (Brandes)

Beweis.

Erklärung:

- Zuerst wird die Summe in die beiden Fälle $t = w$ und $t \neq w$ aufgeteilt.
- Dann wird ausgeklammert.
- Es gilt

$$\sum_{t \in V \setminus \{v, w\}} \frac{\sigma_{st}(w)}{\sigma_{st}} = \sum_{t \in V \setminus \{w\}} \frac{\sigma_{st}(w)}{\sigma_{st}} = \delta_{s^*}(w)$$

weil aufgrund der Annahme, dass v ein Vorgänger von w bezüglich Startknoten s ist, für den Fall $t = v$ gilt, dass $\sigma_{st}(w) = \sigma_{sv}(w) = 0$



Betweenness Centrality (Brandes)

- Berechne n kürzeste-Pfade-DAGs (einen für jeden Startknoten $s \in V$)
- Berechne nacheinander für alle $s \in V$ aus dem kürzeste-Pfade-DAG von s die Abhängigkeiten $\delta_{s*}(v)$ für alle anderen Knoten $v \in V$

Vorgehen: rückwärts, von den Blättern im kürzeste-Pfade-Baum bzw. von der entferntesten Schicht im kürzeste-Pfade-DAG zum Startknoten hin

- Summiere die einzelnen Abhängigkeiten (kann schon parallel während der Berechnung aufsummiert werden, um nicht $\mathcal{O}(n^2)$ Platz zu verbrauchen)

Betweenness Centrality (Brandes)

Satz

Die Betweenness-Zentralität $c_B(v)$ für alle Knoten $v \in V$ kann

- für *ungewichtete* Graphen in $\mathcal{O}(nm)$
- für *gewichtete* Graphen in Zeit $\mathcal{O}(n(m + n \log n)) = \mathcal{O}(nm + n^2 \log n)$

berechnet werden.

Der Algorithmus benötigt dabei nur $\mathcal{O}(n + m)$ Speicherplatz.

Bemerkung:

Die anderen auf kürzesten Pfaden basierenden Zentralitäten kann man relativ einfach mit SSSP-Traversierung berechnen.

Absolute p -Center Problem

gegeben:

- ungerichteter Graph $G = (V, E)$
- nichtnegative Gewichtsfunktion $w(v)$ für Knoten $v \in V$
- positive Länge $\ell(e)$ für Kante $e \in E$

Sei $X_p = \{x_1, x_2, \dots, x_p\}$ eine Menge von p Punkten auf G (dürfen auf Knoten oder an beliebigen Kantenpositionen liegen).

$d(v, x_i)$: Länge eines kürzesten Pfades zwischen v und x_i

Distanz eines Knotens zur Positionsmenge:

$$d(v, X_p) = \min_{1 \leq i \leq p} \{d(v, x_i)\}$$

Definition: Absolute p -Center

Weiteste gewichtete Distanz eines Knotens $v \in V$ zu X_p :

$$F(X_p) = \max_{v \in V} \{w(v) \cdot d(v, X_p)\}$$

X_p^* : eine optimale Menge aus p Punkten:

$$F(X_p^*) = \min_{X_p \text{ auf } G} \{F(X_p)\}$$

Man bezeichnet X_p^* als [absolute] p -center
und $r_p = F(X_p^*)$ als [absolute] p -radius von G .

Falls X_p beschränkt ist auf (echte) Knoten des Graphen, dann spricht man vom **vertex p -center** bzw. **vertex p -radius**.

Annahmen

- Alle Knoten haben gleiches Gewicht $w(v) = c$, o.B.d.A. $c = 1$ (ungewichteter Fall).
- Graph enthält keine Schleifen oder Multikanten.
- Länge jeder Kante $e = (v_r, v_s)$ ist gleich der Distanz der inzidenten Knoten, also $\ell(e) = d(v_r, v_s)$
(ansonsten könnte man e einfach eliminieren, ohne dass sich der p -Radius ändert)
- Distanzen zwischen allen Knotenpaaren sind bekannt.
(Falls diese erst berechnet werden müssen, erhöht sich die Komplexität um den Aufwand zur Lösung des APSP-Problems.)

Hinweis: Domination Number / Set als inverses Problem

gegeben:

- $G = (V, E)$
- natürliche Zahl r

gesucht:

- kleinste natürliche Zahl p , so dass der p -Radius von G nicht größer als r ist.

Problem:

- p heißt [absolute] domination number für Radius r .
- Ein entsprechendes p -center heißt [absolute] domination set für Radius r .
- Für ungewichtete Graphen und Radius 1 ergibt sich das bekannte Problem von domination number/set (welches \mathcal{NP} -vollständig ist!)

1-center / local center

- Vereinfachung: einelementige Menge $X_1 = \{x\}$

$$F(x) = \max_{v \in V} \{w(v) \cdot d(v, x)\}$$

- Um ein 1-center zu finden, sucht man nach einem “lokalen” Zentrum auf jeder Kante.
- Ein **local center** auf Kante $e \in E$ ist eine Position x_e^* auf e , so dass

$$F(x_e^*) = \min_{x_e \text{ auf } e} \{F(x_e)\}$$

wobei $r(e) = F(x_e^*(e))$ **lokaler Radius** von G auf e heißt.

$$r_1 = r(e_j) = \min_{1 \leq i \leq |E|} \{r(e_i)\}$$

r_1 : 1-radius von G und das entsprechende $x_{e_j}^*$ ist ein 1-center.

1-center / local center

- Um ein local center auf Kante $e = (v_r, v_s)$ zu finden, betrachte für jeden Knoten $v \in V$ seine Distanz zu einem Punkt x_e auf e .
- Sei $t = t(x_e)$ die Distanz von x_e zu v_r auf Kante e .
- gewichtete Distanz von v zu $x(e)$:

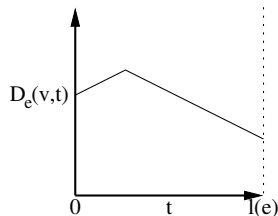
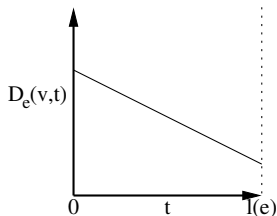
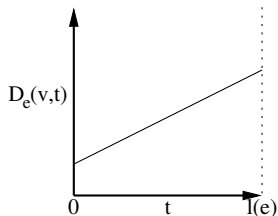
$$D_e(v, t) = w(v) \cdot \min \left\{ \begin{array}{l} t + d(v_r, v), \\ \ell(e) - t + d(v_s, v) \end{array} \right\}$$

- ungewichtete Distanz:

$$D_e(v, t) = \min \left\{ \begin{array}{l} t + d(v_r, v), \\ \ell(e) - t + d(v_s, v) \end{array} \right\}$$

- Wenn klar ist, welche Kante gemeint ist: $D(v, t)$ anstatt $D_e(v, t)$.

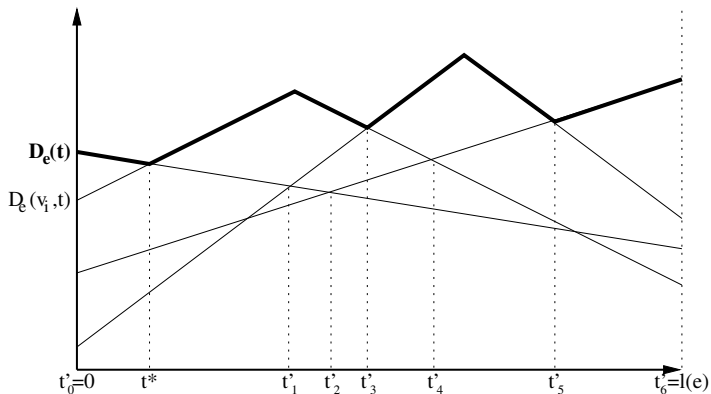
1-center / local center



- Jede solche Funktion besteht aus ein oder zwei linearen Stücken mit Anstieg $\pm w(v)$.
- Im Fall von zwei Segmenten ist der Knickpunkt ein **Maximum**.

Local center auf Kante e

$$D(t) = \max_{v \in V} \{D(v, t)\}, \quad 0 \leq t \leq \ell(e)$$



Local center auf Kante e

- Jeder Punkt x_e^* , an dem $D_e(t)$ sein globales Minimum annimmt, ist ein local center auf e .
- $t^* = t(x_e^*)$ sei der Wert von t an dieser Stelle x_e^* .
- $D_e(t^*)$ ist der local-radius auf e .
- t^* ist eine Stelle für die folgendes gilt:
 $t^* = 0$ oder $t^* = \ell(e)$ oder
 bei t^* schneiden sich zwei Funktionen $D(v_i, t)$ und $D(v_j, t)$ so, dass ihre Anstiege im Schnittpunkt gegensätzliche Vorzeichen haben.
- Wenn T^* die Menge aller Punkte ist, die diese Voraussetzung erfüllen, dann ist

$$D(t^*) = \min_{t' \in T^*} \{D(t')\}$$

⇒ Höchstens $\frac{n(n-1)}{2} + 2$ Punkte kommen als local-center von e in Frage.

1-center / local center

- Falls die Distanzmatrix bekannt ist, kann man mit den Funktionen

$$D_e(v, t) = w(v) \cdot \min \left\{ \begin{array}{l} t + d(v_r, v), \\ \ell(e) - t + d(v_s, v) \end{array} \right\}$$

$$D_e(t) = \max_{v \in V} \{D_e(v, t)\}$$

den Wert $D_e(t)$ in $\mathcal{O}(n)$ Schritten an jedem der $\mathcal{O}(n^2)$ in Frage kommenden Punkte berechnen.

- ⇒ Bei einer direkten Implementierung kann man ein local center auf e in $\mathcal{O}(n^3)$ Schritten finden.

1-center / local center

- Kariv und Hakimi haben eine Methode vorgeschlagen, die in **knoten-gewichteten** Graphen ein local-center für eine Kante e in $\mathcal{O}(n \log n)$ Schritten findet.

⇒ Berechnung eines 1-centers in $\mathcal{O}(mn \log n)$

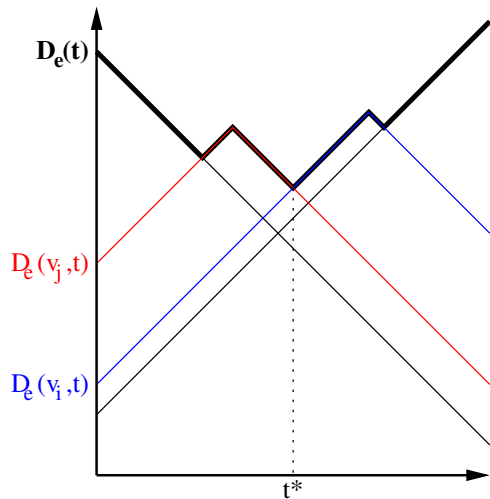
- Für ein **knoten-ungewichtetes** Netzwerk haben Kariv und Hakimi einen Algorithmus von Komplexität $\mathcal{O}(n)$ zum Finden eines local-centers für eine Kante e präsentiert.

⇒ Berechnung eines 1-centers in $\mathcal{O}(mn + n^2 \log n)$

Local centers in knoten-ungewichteten Graphen

- Ein local center x_e^* ist entweder
 - ▶ an einem der Endpunkte v_r oder v_s der Kante oder
 - ▶ an einem Punkt $t^* \in [0, \ell(e)]$, wo sich zwei Funktionen $D(v_i, t)$ und $D(v_j, t)$ schneiden.
 - Aufgrund des gleichen Gewichts für alle Knoten hat der Betrag des Anstiegs für alle Funktionen $D(v_i, t)$ den gleichen Wert.
- ⇒ Zwei beliebige solche Funktionen $D(v_i, t)$ und $D(v_j, t)$, die sich nicht in einem ganzen Liniensegment überlagern, schneiden sich höchstens in einem Punkt. Die beiden Anstiege haben an diesem Punkt umgekehrte Vorzeichen.

Local centers in knoten-ungewichteten Graphen



Local centers in knoten-ungewichteten Graphen

Definiere für jeden Knoten $v_i \in V$:

$$V_i = \{v \in V : D(v, 0) \leq D(v_i, 0)\} \quad \bar{V}_i = V \setminus V_i$$

Falls $\bar{V}_i \neq \emptyset$, definiere auch Knoten \bar{v}_i mit:

$$D(\bar{v}_i, \ell(e)) = \max_{v \in \bar{V}_i} \{D(v, \ell(e))\}$$

(Falls es mehrere Knoten gibt, die diese Bedingung erfüllen, wählen wir den mit dem kleinsten Index.)

Local centers in knoten-ungewichteten Graphen

Lemma

Sei t_0 die Stelle, an der sich zwei Funktionen $D(v_i, t)$ und $D(v_j, t)$ so schneiden, dass an der Stelle t_0 der Anstieg der Funktion $D(v_i, t)$ positiv ist, während der von $D(v_j, t)$ negativ ist.

Dann gilt

$$\begin{aligned} D(v_i, t) &< D(v_j, t) && \text{für } 0 \leq t < t_0 \\ D(v_i, t) &> D(v_j, t) && \text{für } t_0 < t \leq \ell(e) \end{aligned}$$

Folgerung

Unter den Bedingungen des vorigen Lemmas gilt

$$v_j \in \bar{V}_i \quad \text{und} \quad v_i \in V_j$$

Local centers in knoten-ungewichteten Graphen

Lemma

Ein local center auf Kante e ist entweder auf einem der Endpunkte von e , also bei $t = 0$ oder $t = \ell(e)$, oder an einer Stelle t_i , an der sich zwei Funktionen $D(v_i, t)$ und $D(\bar{v}_i, t)$ schneiden.

Beweis.

- Der Fall der Kantenendpunkte ist klar.
- Wir betrachten ein local center auf e an der Stelle t^* mit $t^* \neq 0$ und $t^* \neq \ell(e)$.
- t^* ist die Stelle, an der sich zwei Funktionen $D(v_i, t)$ und $D(v_j, t)$ so schneiden, dass an der Stelle t^* der Anstieg der Funktion $D(v_i, t)$ positiv ist, während der von $D(v_j, t)$ negativ ist.

Local centers in knoten-ungewichteten Graphen

Beweis.

$$\Rightarrow v_j \in \bar{V}_i$$

$$\Rightarrow \bar{v}_i \text{ ist definiert, denn } \bar{V}_i \neq \emptyset$$

- $D(v_i, t)$ und $D(v_j, t)$ schneiden sich an Stelle t^* und bilden dort das local center von Kante e (Maximum aller Funktionen)
- auch $D(\bar{v}_i, t)$ kann an Stelle t^* nicht größer sein als $D(v_i, t)$ bzw. $D(v_j, t)$:

$$D(\bar{v}_i, t^*) \leq D(v_j, t^*)$$

- Im Falle von Gleichheit gilt das Lemma:

$$D(\bar{v}_i, t^*) = D(v_j, t^*) = D(v_i, t^*)$$

- Annahme: $D(\bar{v}_i, t^*) < D(v_j, t^*) = D(v_i, t^*)$

Local centers in knoten-ungewichteten Graphen

Beweis.

- Falls $D(\bar{v}_i, t)$ bei t^* positiv ansteigt:

$$D(\bar{v}_i, t^*) < D(v_i, t^*) \Rightarrow D(\bar{v}_i, 0) < D(v_i, 0)$$

Widerspruch zur Definition von \bar{v}_i .

- Falls $D(\bar{v}_i, t)$ bei t^* negativ ansteigt:

$$D(\bar{v}_i, t^*) < D(v_j, t^*) \Rightarrow D(\bar{v}_i, \ell(e)) < D(v_j, \ell(e))$$

Widerspruch zur Definition von \bar{v}_i

(weil nach Folgerung des letzten Lemmas $v_j \in \bar{V}_i$).

\Rightarrow Annahme falsch, d.h. es gilt Gleichheit und damit das Lemma.



\Rightarrow höchstens $n + 2$ Kandidaten für local center auf Kante e

Schnittpunkte von $D(v_i, t)$ und $D(\bar{v}_i, t)$ bilden Maxima

Lemma

Wenn sich die Funktionen $D(v_i, t)$ und $D(\bar{v}_i, t)$ an der Stelle t_i überschneiden, dann gilt

$$D(v_i, t_i) = \max_{v \in V} \{D(v, t_i)\}$$

oder kurz

$$D(t_i) = D(v_i, t_i)$$

Local centers in knoten-ungewichteten Graphen

Beweis.

- Annahme: Lemma gilt nicht, sondern es gibt einen Knoten v' mit $D(v_i, t_i) < D(v', t_i)$.
- Da $D(v_i, t_i)$ an der Stelle t_i einen positiven Anstieg hat, impliziert die Ungleichung, dass $D(v_i, 0) < D(v', 0)$, also $v' \in \bar{V}_i$.
- Andererseits impliziert $D(v_i, t_i) < D(v', t_i)$, dass $D(\bar{v}_i, t_i) < D(v', t_i)$ (weil $D(v_i, t_i) = D(\bar{v}_i, t_i)$).
- Da $D(\bar{v}_i, t)$ an der Stelle t_i einen negativen Anstieg hat, folgt $D(\bar{v}_i, \ell(e)) < D(v', \ell(e))$.

⇒ Widerspruch zur Definition von \bar{v}_i



Local centers in knoten-ungewichteten Graphen

Folgerung (aus den letzten beiden Lemmas)

Sei $t_0 = 0$, $t_{n+1} = \ell(e)$ und sei

t_i für $i \in \{1 \dots n\}$ mit $\bar{V}_i \neq \emptyset$ die Schnittstelle von $D(v_i, t)$ und $D(\bar{v}_i, t)$.

Sei $D_0 = \max_{v \in V} \{D(v, 0)\}$, $D_{n+1} = \max_{v \in V} \{D(v, \ell(e))\}$ und für $i \in \{1 \dots n\}$, wo t_i definiert ist, sei $D_i = D(v_i, t_i)$.

Sei j ein Index, für den

$$D_j = \min_{i \in \{0 \dots n+1\}} \{D_i \mid D_i \text{ ist definiert}\}$$

Dann ist D_j der local radius auf e und das entsprechende t_j ist ein local center auf e .

Local center / Irrelevante Schnittpunkte

Lemma

Wenn für zwei Knoten v_i und v_j gilt, dass

$$\begin{aligned}D(v_i, 0) &= D(v_j, 0) \\ D(v_i, \ell(e)) &\geq D(v_j, \ell(e)),\end{aligned}$$

dann muss der Schnittpunkt t_j von $D(v_j, t)$ und $D(\bar{v}_j, t)$ aus der vorangegangenen Folgerung nicht betrachtet werden.

Local center / Irrelevante Schnittpunkte

Beweis.

- Aus den Bedingungen des Lemmas folgt für alle t ($0 \leq t \leq \ell(e)$):
 $D(v_i, t) \geq D(v_j, t)$
- Annahme: t_j ist definiert und local center auf e
 mit local radius $D(v_j, t_j) = \max_{v \in V} \{D(v, t_j)\}$
- $\Rightarrow D(v_j, t_j) \geq D(v_i, t_j) \Rightarrow D(v_i, t_j) = D(v_j, t_j).$
- $D(v_i, 0) = D(v_j, 0) \Rightarrow \bar{V}_i = \bar{V}_j \Rightarrow \bar{v}_i = \bar{v}_j$
- $\Rightarrow D(\bar{v}_i, t_j) = D(\bar{v}_j, t_j).$
- \Rightarrow Schnittpunkt t_j von $D(v_j, t)$ und $D(\bar{v}_j, t)$ ist gleichzeitig
 Schnittpunkt t_i von $D(v_i, t)$ und $D(\bar{v}_i, t)$
- \Rightarrow Es ist ausreichend, nur t_i zu betrachten und t_j zu ignorieren.



Local center / Vorberechnung

- Folgender Algorithmus basiert auf der letzten Folgerung und auf dem letzten Lemma.
- Vorberechnung:
Konstruiere für jeden Knoten $v \in V$ eine **Liste $L(v)$** , in der die Knoten des Graphen in monoton fallender Reihenfolge ihrer Distanz zu v stehen.
Knoten v ist dabei der letzte in seiner Liste $L(v)$.
- Unter der Annahme, dass die Distanzmatrix bekannt ist, dauert die Konstruktion jeder Liste $\mathcal{O}(n \log n)$ Schritte.
- Gesamtkomplexität für alle Knoten: $\mathcal{O}(n^2 \log n)$

Local center / Algorithmus

- Sei $e = (v_r, v_s)$ und sei v_i der i -te Knoten in Liste $L(v_r)$.
- Dann gilt

$$D(v_1, 0) \geq D(v_2, 0) \geq \dots \geq D(v_{n-1}, 0) > D(v_n, 0) = 0$$

In dieser Notation ist v_r (der Endpunkt von e) benannt mit v_n .

- Der Algorithmus betrachtet in der i -ten Phase die Funktionen $D(v_i, t)$ und $D(\bar{v}_i, t)$ und berechnet deren Schnittpunkt, falls es notwendig ist.
- Man beachte, dass falls für ein j und k ($1 < j \leq k < n$) gilt, dass

$$D(v_{j-1}, 0) > D(v_j, 0) = \dots = D(v_k, 0) > D(v_{k+1}, 0)$$

dann gilt aufgrund der Definitionen von V_i und \bar{v}_i , dass

$$\bar{v}_j = \bar{v}_{j+1} = \dots = \bar{v}_k.$$

Local centers in knoten-ungewichteten Graphen

- Falls v_m (mit $j \leq m \leq k$) ein Knoten mit minimalem Index ist, so dass

$$D(v_m, \ell(e)) = \max_{j \leq i \leq k} \{D(v_i, \ell(e))\}$$

dann impliziert das letzte Lemma, dass aus der ganzen Knotenmenge $\{v_j, v_{j+1}, \dots, v_k\}$ nur der Knoten v_m betrachtet werden sollte, also der Schnittpunkt t_m von $D(v_m, t)$ und $D(\bar{v}_m, t)$.

- Aufgrund der Definitionen von V_i und \bar{v}_i und aufgrund der Reihenfolge in Liste $L(v_r)$ folgt, dass Knoten \bar{v}_{k+1} genau der Knoten ist (von den beiden Knoten \bar{v}_j und v_m), für den gilt:

$$D(\bar{v}_{k+1}, \ell(e)) = \max \left\{ \begin{array}{l} D(\bar{v}_j, \ell(e)), \\ D(v_m, \ell(e)) \end{array} \right\}$$

Local centers in knoten-ungewichteten Graphen

- Sei k eine natürliche Zahl, so dass

$$D(v_1, 0) = \dots = D(v_k, 0) > D(v_{k+1}, 0) \quad (1 \leq k < n)$$

⇒ Knoten $\bar{v}_1, \dots, \bar{v}_k$ sind nicht definiert und müssen in der letzten Folgerung nicht betrachtet werden.

- Man beachte aber, dass für die gleiche Definition für v_m wie zuvor, d.h., v_m mit $1 \leq m \leq k$ ist ein Knoten mit minimalem Index, so dass

$$D(v_m, \ell(e)) = \max_{1 \leq i \leq k} \{D(v_i, \ell(e))\},$$

dann impliziert die spezielle Sortierung von $L(v_r)$, dass $\bar{v}_{k+1} = v_m$.

Local center / Algorithmus von Kariv und Hakimi

t^* und r : local center und local radius auf Kante $e = (v_r, v_s)$

v^* und \hat{v}^* : Knoten v_i und \bar{v}_i

- 1 Behandlung der Punkte $t = 0$ und $t = \ell(e)$:

Sei v^* der erste Knoten von Liste $L(v_r)$ und sei \hat{v} der erste Knoten von Liste $L(v_s)$.

Falls $D(v^*, 0) \leq D(\hat{v}, \ell(e))$,

dann $t^* \leftarrow 0$; $r \leftarrow D(v^*, 0)$;

sonst $t^* \leftarrow \ell(e)$; $r \leftarrow D(\hat{v}, \ell(e))$.

Wenn $v^* = \hat{v}$, dann STOP. (t^*/r sind local center/radius auf e)

Local centers in knoten-ungewichteten Graphen

- 2 Initialisierung der Phasen:

$$i \leftarrow 1, v_m \leftarrow v^*$$

- 3 Behandlung aller Knoten v mit $D(v, 0) = D(v_1, 0)$:

$$i \leftarrow i + 1$$

Sei v^* der i -te Knoten in Liste $L(v_r)$.

Wenn $D(v^*, 0) \neq D(v_m, 0)$ dann gehe zu Schritt 4;
sonst

$$\text{falls } D(v^*, \ell(e)) > D(v_m, \ell(e)): v_m \leftarrow v^*.$$

Wiederhole Schritt 3. (Wegen $D(v_i, 0) = D(v_1, 0)$ ist $i < n$.)

- 4 \bar{v}^* steht nun fest:

$$\bar{v}^* \leftarrow v_m$$

Wenn $i = n$, dann gehe zu Schritt 8;

sonst $v_m \leftarrow v^*$

Local centers in knoten-ungewichteten Graphen

- 5 Finde alle Knoten v mit $D(v, 0) = D(v_i, 0)$, sowie das entsprechende v_m :

$$i \leftarrow i + 1$$

Sei v^* der i -te Knoten in Liste $L(v_r)$.

Falls $D(v^*, 0) \neq D(v_m, 0)$, dann gehe zu Schritt 6;
sonst

$$\text{falls } D(v^*, \ell(e)) > D(v_m, \ell(e)): \quad v_m \leftarrow v^*.$$

Wiederhole Schritt 5. (Wegen $D(v_i, 0) = D(v_{i-1}, 0)$ ist $i < n$.)

Local centers in knoten-ungewichteten Graphen

6 Behandlung des Punkts t_m :

Wenn sich die Funktionen $D(v_m, t)$ und $D(\bar{v}^*, t)$ nicht schneiden oder sich in einem ganzen Segment überlagern, dann gehe zu Schritt 7.

Sonst sei t_m der Schnittpunkt.

Falls $D(v_m, t_m) < r$,
dann $t^* \leftarrow t_m$, $r \leftarrow D(v_m, t_m)$.

7 Gehe zum nächsten Knoten:

Wenn $D(v_m, \ell(e)) > D(\bar{v}^*, \ell(e))$, dann $\bar{v}^* \leftarrow v_m$

Wenn $i = n$, gehe zu Schritt 8;
sonst $v_m \leftarrow v^*$ und gehe zu Schritt 5.

Local centers in knoten-ungewichteten Graphen

8 Behandlung von Knoten v_n :

Wenn sich die Funktionen $D(v^*, t)$ und $D(\bar{v}^*, t)$ nicht schneiden oder sich in einem ganzen Segment überlagern, dann STOP.

Sonst sei t_m der Schnittpunkt.

Falls $D(v^*, t_m) \geq r$, dann STOP;

sonst $t^* \leftarrow t_m$, $r \leftarrow D(v^*, t_m)$ und STOP.

Unter der Annahme, dass die Distanzmatrix und die Listen $L(v_r)$ und $L(v_s)$ verfügbar sind, ist die Komplexität $\mathcal{O}(n)$.

1-centers in knoten-ungewichteten Graphen

- 1 Konstruiere für jeden Knoten $v \in V$ eine Liste $L(v)$ aller Knoten in einer Reihenfolge monoton fallender Distanz von v .
- 2 Berechne mit Hilfe des vorangegangenen Algorithmus für jede Kante e des Graphen das local center und den local radius von e .
- 3 Der minimale local radius ist der 1-radius des Graphen.
Jedes local center mit minimalem local radius ist ein 1-center des Graphen.

Schritt 1 benötigt Zeit $\mathcal{O}(n^2 \log n)$.

Schritt 2 benötigt Zeit $\mathcal{O}(mn)$.

Wenn die Distanzmatrix des Graphen bekannt ist, benötigt der gesamte Algorithmus Zeit $\mathcal{O}(mn + n^2 \log n)$.

Minimum Diameter Spanning Trees

Satz

Sei

- x^* ein absolute 1-center von G und
- $T(x^*)$ ein Shortest Path Tree, der x^* mit allen Knoten in V verbindet.

Dann ist $T(x^*)$ ein **Minimum Diameter Spanning Tree** von G
(ein Spannbaum mit minimalem Durchmesser).

Minimum Diameter Spanning Trees

Beweis.

Sei T ein beliebiger Spannbaum von G und $y^*(T)$ dessen absolute 1-center. ($y^*(T)$ ist eindeutig und für den Durchmesser von T gilt $D(T) = 2 \max_{v \in V} \{d_T(y^*(T), v)\}$).

x^* ist Mittelpunkt von jedem Durchmesser(-Pfad) von $T(x^*)$.

$$D(T(x^*)) = 2 \max_{v \in V} d_{T(x^*)}(x^*, v) \quad (1)$$

$$= 2 \max_{v \in V} d_G(x^*, v) \quad (2)$$

$$\leq 2 \max_{v \in V} d_G(y^*(T), v) \quad (3)$$

$$\leq 2 \max_{v \in V} d_T(y^*(T), v) = D(T) \quad (4)$$



Shortcut-Werte

- gegeben: gerichteter Graph $G = (V, E)$
- gesucht: shortcut-Werte aller Kanten von G

Maximale Erhöhung der Länge eines kürzesten Pfades durch Entfernen einer Kante $e = (u, v) \in E$

⇒ Berechne für jede Kante $e = (u, v) \in E$ die **Distanz von u zu v** in $G_e = (V, E \setminus \{e\})$, denn die maximale Erhöhung betrifft immer (auch) die Endknoten der Kante

- einfache Lösung: $m = |E|$ SSSP Aufrufe
- besser: nur $n = |V|$ äquivalente Aufrufe

Shortcut-Werte

Annahmen:

- keine Kreise mit negativem Gesamtgewicht
- ⇒ $d(i,j)$ ist definiert für alle Knotenpaare (i,j)
- keine parallelen Kanten

Idee:

- Ein Aufruf für Knoten u berechnet shortcut-Werte für **alle ausgehenden Kanten**

Shortcut-Werte

Vorgehen:

- Fixiere einen Startknoten u
- $\alpha_i = d(u, i)$: Distanz von u nach i
- τ_i : zweiter Knoten der kürzesten Pfade von u zu i , falls dieser Knoten eindeutig ist.
Ansonsten $\tau_i = \perp$ (impliziert zwei kürzeste Pfade der Länge α_i mit unterschiedlichen Anfangskanten).
- β_i : Länge des kürzesten Pfades von u nach i , so dass τ_i nicht zweiter Knoten
(∞ falls es keinen Pfad mehr gibt, $\beta_i = \alpha_i$ falls $\tau_i = \perp$)

Shortcut-Werte

- Betrachte α_v , τ_v und β_v für einen Nachbarn v von u , also $(u, v) \in E$.
- Dann ist die shortcut-Distanz für (u, v) gleich α_v falls $\tau_v \neq v$, also wenn Kante (u, v) nicht einziger kürzester Pfad ist
- Ansonsten, falls $\tau_v = v$, ist die neue Distanz β_v
- $\alpha_u = 0$, $\tau_u = \emptyset$, $\beta_u = \infty$

$$\alpha_j = \min_{i:(i,j) \in E} (\alpha_i + \omega(i,j))$$

- Nachbarn bezüglich eingehender Kanten, die zu einem kürzesten Pfad gehören (Vorgängermenge):

$$I_j = \{i : (i,j) \in E \text{ und } \alpha_j = \alpha_i + \omega(i,j)\}$$

Shortcut-Werte

$$\tau_j = \begin{cases} j & \text{if } I_j = \{u\}, \quad u \text{ ist Anfang, Kante } (u, j) \\ a & \text{if } \forall i \in I_j : a = \tau_i \\ & \text{(Alle Vorgänger haben erste Kante } (u, a)), \\ \perp & \text{sonst} \end{cases}$$

Im Fall $\tau_j = \perp$ gilt $\beta_j = \alpha_j$,

sonst ($\tau_j \neq \perp$) gilt

$$\beta_j = \min \left\{ \begin{array}{l} \min_{i: (i,j) \in E, \tau_i = \tau_j} \beta_i + \omega(i, j), \\ \min_{i: (i,j) \in E, \tau_i \neq \tau_j} \alpha_i + \omega(i, j) \end{array} \right\}$$

Shortcut-Werte

Betrachte den Pfad der zur Länge β_j führt, also ein kürzester Pfad p von u nach j , der nicht mit τ_j beginnt.

Wenn für den letzten Knoten i vor j in p gilt $\tau_i = \tau_j$, dann startet der Pfad p bis zu i nicht mit τ_j , und dieser Pfad wird in β_i und damit in β_j berücksichtigt.

Wenn anderenfalls $\tau_i \neq \tau_j$ für den vorletzten Knoten i von Pfad p gilt, dann beginnt einer der kürzesten Pfade von u nach i nicht mit τ_j und die Länge von p ist $\alpha_i + \omega(i, j)$.

Shortcut-Werte

Berechnung der Werte α_i , τ_i und β_i :

Im Fall positiver Gewichte hängt jeder Wert α_i nur von Werten α_j ab, die kleiner als α_i sind

⇒ Berechnung in monotoner Weise nach Dijkstra

Bei positiven Gewichten ist der kürzeste-Wege-DAG kreisfrei und die Werte τ_i können in der Reihenfolge einer topologischen Sortierung berechnet werden

(sonst stark zusammenhängende Komponenten kontrahieren)

Werte β_i hängen nur von Werten $\beta_j \leq \beta_i$ ab

⇒ Berechnung in monotoner Weise nach Dijkstra

Bei negativen Kantengewichten (aber keine negativen Kreise) Dijkstra durch Bellman-Ford Algorithmus und β_i durch Berechnung von

$\beta'_i = \beta_i - \alpha_i$ ersetzen

Katz' Status Index

Modell (Leo Katz, 1953):

- Einfluss entsteht aus **Anhängerschaft** bzw. aus indirekter Wahl
- ⇒ wenn A ein Anhänger von B ist und B ist Anhänger von C , dann ist A auch ein (indirekter) Anhänger bzw. Wähler von C

- mit zunehmender Anzahl von Zwischenschritten soll dieser Effekt jedoch abnehmen
- ⇒ Dämpfungsfaktor $\alpha > 0$

- ungewichteter, gerichteter Graph $G = (V, E)$ ohne Schleifen
- Adjazenzmatrix A
- Anzahl Wege der Länge k von j nach i ist $(A^k)_{ji}$

Katz' Status Index

Status von Knoten i :

$$c_K(i) = \sum_{k=1}^{\infty} \sum_{j=1}^n \alpha^k (A^k)_{ji}$$

(falls die unendliche Summe konvergiert)

Matrixnotation:

$$c_K = \sum_{k=1}^{\infty} \alpha^k (A^k)^T \mathbf{1}_n = \sum_{k=1}^{\infty} \alpha^k (A^T)^k \mathbf{1}_n$$

A^T : Transponierte Matrix von A

$\mathbf{1}_n$: n -dimensionaler Vektor, dessen Einträge alle 1 sind

Katz' Status Index

Für die Konvergenz muss α beschränkt werden:

Satz

Sei A die Adjazenzmatrix eines Graphen G mit größtem Eigenwert λ_1 und $\alpha > 0$, dann gilt:

$$\lambda_1 < \frac{1}{\alpha} \iff \sum_{k=1}^{\infty} \alpha^k A^k \text{ konvergiert}$$

(hier ohne Beweis)

Man kann die unendliche Summe auch so umformulieren:

$$\sum_{k=1}^{\infty} \alpha^k (A^T)^k = \left(\sum_{k=0}^{\infty} \alpha^k (A^T)^k \right) - \alpha^0 (A^T)^0 = \left(\sum_{k=0}^{\infty} \alpha^k (A^T)^k \right) - I_n$$

(I_n : $n \times n$ Einheits-/Identitätsmatrix)

Katz' Status Index

Falls die Reihe konvergent ist, erhalten wir:

$$\begin{aligned}
 \sum_{k=0}^{\infty} \alpha^k A^k &= \alpha^0 A^0 + \sum_{k=1}^{\infty} \alpha^k A^k \\
 &= I_n + \alpha A \sum_{k=1}^{\infty} \alpha^{k-1} A^{k-1} \\
 &= I_n + \alpha A \sum_{k=0}^{\infty} \alpha^k A^k
 \end{aligned}$$

$$(I_n - \alpha A) \sum_{k=0}^{\infty} \alpha^k A^k = I_n$$

$$\sum_{k=0}^{\infty} \alpha^k A^k = (I_n - \alpha A)^{-1}$$

Katz' Status Index

$$\sum_{k=1}^{\infty} \alpha^k A^k = (\mathbf{I}_n - \alpha A)^{-1} - \mathbf{I}_n$$

Unter der Annahme der Konvergenz erhält man die geschlossene Form:

$$\begin{aligned} \mathbf{c}_K &= \sum_{k=1}^{\infty} \alpha^k A^k \mathbf{1}_n = \left((\mathbf{I}_n - \alpha A)^{-1} - \mathbf{I}_n \right)^T \mathbf{1}_n \\ (\mathbf{I}_n - \alpha A)^T \mathbf{c}_K &= \left(\mathbf{I}_n - (\mathbf{I}_n - \alpha A)^T \right) \mathbf{1}_n \\ &= \alpha A^T \mathbf{1}_n \\ \left(\frac{1}{\alpha} \mathbf{I}_n - A^T \right) \mathbf{c}_K &= \mathbf{d}_{\text{in}} \end{aligned}$$

Inhomogenes lineares Gleichungssystem verdeutlicht Feedback-Charakter:
 $\mathbf{c}_K(i)$ hängt von den anderen $\mathbf{c}_K(j)$ mit $j \neq i$ ab

Eigenvektor-Zentralität von Bonacich

Phillip Bonacich, 1972:

- gegeben: ungerichteter Graph
(ungewichtet, zusammenhängend, einfach und ohne Schleifen)
- 3 Ansätze:
 - a Faktoranalyse
 - b Konvergenz einer unendlichen Reihe
 - c Lösung eines linearen Gleichungssystems
- Werte \mathbf{s}^a , \mathbf{s}^b und \mathbf{s}^c unterscheiden sich aber nur durch konstante Faktoren

Eigenvektor-Zentralität von Bonacich

Ansatz a: Faktoranalyse:

- Interpretiere den Graph als Freundschaftsnetzwerk
- Eine Kante bedeutet Freundschaft zwischen den Endknoten
- gesucht: Vektor $\mathbf{s}^a \in \mathbb{R}^n$, so dass der i -te Eintrag s_i^a das "Freundschaftspotential" von Knoten i enthält.
- $s_i^a s_j^a$ soll möglichst nah bei a_{ij} liegen
- Vektor-/Matrixschreibweise: $\mathbf{s}^a \mathbf{s}^{aT}$ soll A approximieren
- interpretiere das Problem als Minimierung des folgenden Ausdrucks (mit der Methode der kleinsten Quadrate):

$$\sum_{i=1}^n \sum_{j=1}^n (s_i^a s_j^a - a_{ij})^2$$

- \mathbf{s}^a ist der Eigenvektor zum größten Eigenwert λ_1 , aber so skaliert, dass seine Länge gleich λ_1 ist

Eigenvektor-Zentralität von Bonacich

Ansatz b: Unendliche Reihe:

Sei $\lambda_1 \neq 0$ der größte Eigenwert. Definiere

$$\mathbf{s}^{b_0} = \mathbf{1}_n \quad \text{und} \quad \mathbf{s}^{b_k} = A \frac{\mathbf{s}^{b_{k-1}}}{\lambda_1} = A^k \frac{\mathbf{s}^{b_0}}{\lambda_1^k}$$

Betrachte die Sequenz

$$\mathbf{s}^b = \lim_{k \rightarrow \infty} \mathbf{s}^{b_k} = \lim_{k \rightarrow \infty} A^k \frac{\mathbf{s}^{b_0}}{\lambda_1^k}$$

Satz

Sei $A \in \mathbb{R}^{n \times n}$ eine symmetrische Matrix mit größtem Eigenwert λ_1 , dann konvergiert

$$\lim_{k \rightarrow \infty} A^k \frac{\mathbf{s}^{b_0}}{\lambda_1^k}$$

gegen einen Eigenvektor von A mit zugehörigem Eigenwert λ_1 .

Eigenvektor-Zentralität von Bonacich

Ansatz c: Lineares Gleichungssystem:

- Definiere Zentralität jedes Knotens als Summe der Zentralitäten seiner Nachbarn:

$$s_i^c = \sum_{j=1}^n a_{ij} s_j^c \quad \text{bzw.} \quad \mathbf{s}^c = A \mathbf{s}^c \quad \Rightarrow \quad (A - I_n) \mathbf{s}^c = \mathbf{0}$$

- \Rightarrow immer mögliche Lösung: alle Zentralitäten gleich Null
- \Rightarrow hat nur eine sinnvolle Lösung, falls $\det(A - I_n) = 0$ bzw. wenn 1 ein Eigenwert von A ist
- relaxiere das Problem durch Einführung eines Faktors λ
 \Rightarrow entspricht genau dem Eigenwertproblem:

$$\lambda \mathbf{s}^c = A \mathbf{s}^c \quad \text{bzw.} \quad \mathbf{s}^c = \frac{1}{\lambda} A \mathbf{s}^c \quad \Rightarrow \quad (A - \lambda I_n) \mathbf{s}^c = \mathbf{0}$$

- Auswahl des größten Eigenwerts mit zugehörigem nichtnegativen Eigenvektor

Eigenvektor-Zentralität von Bonacich

Satz

Sei $A \in \mathbb{R}^{n \times n}$ die Adjazenzmatrix eines ungerichteten zusammenhängenden Graphen. Dann

- ist der größte Eigenwert λ_1 von A ein einfacher Eigenwert,
- sind alle Einträge des zu λ_1 gehörenden Eigenvektors ungleich Null und haben das gleiche Vorzeichen.

(siehe auch Satz von Perron und Frobenius)

Eigenvektor-Zentralität von Bonacich

- Alle 3 Varianten unterscheiden sich nur durch konstante Faktoren.

⇒ Normierte Eigenvektor-Zentralität:

$$c_{EV} = \frac{|s|}{||s||}$$

- im Fall unzusammenhängender Graphen kann man den Ansatz auf jede Zusammenhangskomponente anwenden

Hubbell Index

Charles Hubbell, 1965:

- gegeben: einfacher gewichteter gerichteter Graph (darf Schleifen enthalten, aber keine Multikanten)
- gewichtete Adjazenzmatrix W ist hier u.U. **asymmetrisch**
- Idee: (ähnlich wie bei Bonacich)
Wert eines Knotens v hängt von der Summe der Werte seiner Nachbarn w , gewichtet mit dem jeweiligen Kantengewicht ab
- Es soll also gelten:

$$\mathbf{e} = W\mathbf{e}$$

- “Exogener Input” E (externe Information zu jedem Knoten) wird eingeführt, um das System lösbar zu machen
- im Falle des Fehlens: $\mathbf{E} = \mathbf{1}$

⇒ Gleichung

$$\mathbf{s} = \mathbf{E} + W\mathbf{s}$$

Hubbell Index

$$\mathbf{s} = E + W\mathbf{s} = E + W(E + W\mathbf{s}) = E + WE + W^2\mathbf{s} = \dots$$

$$\begin{aligned}
 &= \sum_{k=0}^{\infty} W^k E = W^0 E + \sum_{k=1}^{\infty} W^k E \\
 &= I_n E + W \sum_{k=1}^{\infty} W^{k-1} E \\
 &= E + W \sum_{k=0}^{\infty} W^k E
 \end{aligned}$$

$$(I_n - W) \sum_{k=0}^{\infty} W^k E = E$$

$$\mathbf{s} = \sum_{k=0}^{\infty} W^k E = (I_n - W)^{-1} E$$

Hubbell Index

- umgeformt:

$$\mathbf{s} = (\mathbf{I}_n - \mathbf{W})^{-1} \mathbf{E}$$

⇒ System hat eine Lösung, falls $(\mathbf{I}_n - \mathbf{W})$ invertierbar ist

- Da

$$(\mathbf{I}_n - \mathbf{W})^{-1} = \sum_{k=0}^{\infty} \mathbf{W}^k$$

⇒ äquivalent zur Konvergenz der geometrischen Reihe

- Reihe konvergiert gegen $(\mathbf{I}_n - \mathbf{W})^{-1}$ gdw. der größte Eigenwert λ_1 von \mathbf{W} kleiner als 1 ist
(siehe erster Satz bei Katz' Status Index)
- Lösung des Gleichungssystems heißt Hubbell-Index

Webgraph

- gerichteter Graph $G = (V, E)$
- Knoten entsprechen Webseiten
- gerichtete Kanten entsprechen Links zwischen Webseiten

Random Surfer Model

- Modellierung des Verhaltens eines Websurfers:

Random Walk auf dem Webgraph

- in jedem Schritt überschreitet der Random Walk zufällig eine der ausgehenden Kanten des aktuellen Knotens:

$$\Pr[X_{t+1} = v \mid X_t = u] = \begin{cases} \frac{1}{d_{\text{out}}(u)}, & \text{falls } (u, v) \in E \\ 0, & \text{sonst} \end{cases}$$

- nur wohldefiniert, falls $\forall v \in V : d_{\text{out}}(v) \geq 1$
(in diesem Fall wäre die Transitionsmatrix stochastisch)
- $n \times n$ -Transitionsmatrix $T = (t_{i,j})$ mit

$$t_{i,j} = \begin{cases} 1/d_{\text{out}}(i) & \text{falls } (i,j) \in E, \\ t_{i,j} = 0 & \text{sonst} \end{cases}$$

Random Surfer Model

- Es gibt Senken (Knoten/Seiten ohne ausgehende Kanten/Links).
- ⇒ Transitionsmatrix T ist **nicht stochastisch**
- Modifikation zur stochastischen Transitionsmatrix T' :
Annahme im Fall von Senken: Surfer springt zu einer zufälligen Seite

$$t'_{i,j} = \begin{cases} \frac{1}{d_{\text{out}}(i)}, & \text{falls } (i,j) \in E \\ \frac{1}{n}, & \text{falls } d_{\text{out}}(i) = 0 \\ 0, & \text{sonst} \end{cases}$$

- T' ist aber **nicht irreduzibel**, da der Webgraph nicht stark zusammenhängend ist
- ⇒ Matrix muss modifiziert werden, damit die entsprechende Markov-Kette in eine stationäre Verteilung konvergiert

Random Surfer Model

- Random-Jump-Matrix, in der alle Einträge $\frac{1}{n}$ sind:

$$E = \frac{1}{n} \mathbf{1}_n \mathbf{1}_n^T$$

⇒ wird in gewichteter Form zu T' addiert:

$$T'' = \alpha T' + (1 - \alpha)E$$

- α wird aus dem Intervall $[0, 1)$ gewählt und entspricht der Wahrscheinlichkeit, entweder einem Link auf der Seite zu folgen (T') oder zu einer zufälligen Seite zu springen (E)
- T'' ist stochastisch
(denn T' und E sind stochastisch und in jeder Zeile wird ein α -Anteil mit einem $(1 - \alpha)$ -Anteil der Gesamtwahrscheinlichkeit 1 addiert)
- T'' ist irreduzibel (denn alle Wahrscheinlichkeiten sind > 0)

Stationäre Verteilung im Random Surfer Model

- stationäre Verteilung π'' für Transitionsmatrix T'' kann mit Power Iteration berechnet werden
- Durch Modifikation von E können die Zufallssprünge in Richtung personalisierter Surfergewichtungen verändert werden.

Power Iteration

Algorithmus 4: Power Method

Input : Matrix $A \in \mathbb{R}^{n \times n}$ und Vektor $\vec{q}^{(0)}$ mit $\|\vec{q}^{(0)}\|_2 = 1$

Output : Näherungen $\lambda^{(k)}$ und $\vec{q}^{(k)}$ für betragsmäßig größten Eigenwert λ und korrespondierenden Eigenvektor \vec{q}

$k := 1;$

repeat

$$\vec{z}^{(k)} := A\vec{q}^{(k-1)};$$

$$\vec{q}^{(k)} := \vec{z}^{(k)} / \|\vec{z}^{(k)}\|_2;$$

/* Näherung des Rayleigh-Quotienten $\lambda = \frac{\langle v, Av \rangle}{\langle v, v \rangle}$: *

$$\lambda^{(k)} := (\vec{q}^{(k)})^T A \vec{q}^{(k)};$$

$$k := k + 1;$$

until $\lambda^{(k)}$ und $\vec{q}^{(k)}$ sind akzeptable Näherungen;

Power Iteration

- konvergiert garantiert, wenn A einen dominanten Eigenwert λ_1 hat, d.h. $\forall i \in \{2, \dots, n\} : |\lambda_1| > |\lambda_i|$.
- konvergiert auch, wenn die Matrix symmetrisch ist
- Konvergenzgeschwindigkeit hängt vom Verhältnis $\frac{|\lambda_2|}{|\lambda_1|}$ ab, wobei die Eigenwerte hier nach Betrag geordnet sind: $|\lambda_1| > |\lambda_2| \geq \dots \geq |\lambda_n|$
- Approximationsfehler sinkt mit

$$\mathcal{O}\left(\left(\frac{|\lambda_2|}{|\lambda_1|}\right)^k\right)$$

- erfordert lediglich Matrix-Vektor-Multiplikationen
- ⇒ auch für große Matrizen geeignet, die nicht komplett in den Hauptspeicher passen
Eine Iteration entspricht einem Scan über die Matrix.

PageRank

(Lawrence Page, Sergey Brin, Rajeev Motwani & Terry Winograd, 1998)

- wesentlicher Bestandteil der Suchmaschine von Google
- Idee: Bewertung der Wichtigkeit einer Webseite bezüglich der **topologischen Eigenschaften** (seiner Position im Web), **unabhängig vom Inhalt** (mal abgesehen von den Links)
- ist eine Feedback-Zentralität: Wert einer Webseite hängt von der Anzahl und der Zentralität der Webseiten ab, die darauf zeigen
- Gewicht einer Seite wird gleichmäßig an die verlinkten Seiten weitergegeben

$$c_{\text{PR}}(p) = \frac{1-d}{n} + d \sum_{q \in \text{Pred}(p)} \frac{c_{\text{PR}}(q)}{d_{\text{out}}(q)}$$

d : Dämpfungsfaktor

$\text{Pred}(p)$: (Vorgänger-)Knoten mit einer Kante zu p

PageRank

- definiere Transitionsmatrix P :

$$p_{ij} = \begin{cases} \frac{1}{d_{\text{out}}(j)}, & \text{falls } (j, i) \in E \\ 0, & \text{sonst} \end{cases}$$

- oder anders: $p_{ij} = \frac{1}{d_{\text{out}}(j)} a_{ji}$ bzw. $P = (D_{\text{out}}A)^T$,
wobei D_{out} die Diagonalmatrix ist, die im i -ten Diagonaleintrag den Kehrwert des Ausgangsgrads $d_{\text{out}}(i)$ von Knoten i enthält (falls > 0)

- Matrixform:

$$\mathbf{c}_{\text{PR}} = dP\mathbf{c}_{\text{PR}} + \frac{1-d}{n}\mathbf{1}_n$$

- Lösung meist durch Power (oder Jacobi) Iteration:

$$\mathbf{c}_{\text{PR}}^k = dP\mathbf{c}_{\text{PR}}^{k-1} + \frac{1-d}{n}\mathbf{1}_n$$

PageRank

Folgender Satz garantiert eindeutige Lösung und Konvergenz für $d < 1$:

Satz

Falls $0 \leq d < 1$ dann hat die Gleichung

$$\mathbf{c}_{PR} = dP\mathbf{c}_{PR} + \frac{1-d}{n}\mathbf{1}_n$$

eine eindeutige Lösung $\mathbf{c}_{PR}^* = \frac{1-d}{n} (\mathbf{I}_n - dP)^{-1} \mathbf{1}_n$

und die Lösungen des dynamischen Systems

$$\mathbf{c}_{PR}^k = dP\mathbf{c}_{PR}^{k-1} + \frac{1-d}{n}\mathbf{1}_n$$

erfüllen

$$\lim_{k \rightarrow \infty} \mathbf{c}_{PR}^k = \mathbf{c}_{PR}^*$$

für jeden Startvektor \mathbf{c}_{PR}^0 .

PageRank

Etwas anderer Ansatz:

$$\mathbf{c}_{\text{PR}}^k = dP\mathbf{c}_{\text{PR}}^{k-1} + \frac{\|\mathbf{c}_{\text{PR}}^{k-1}\| - \|dP\mathbf{c}_{\text{PR}}^{k-1}\|}{n} \mathbf{1}_n$$

Die Lösungen dieses System konvergieren gegen $\frac{\mathbf{c}_{\text{PR}}^*}{\|\mathbf{c}_{\text{PR}}^*\|}$, die normalisierte Lösung des vorherigen Ansatzes.

Hubs & Authorities

Jon Kleinberg, 1997:

- HITS: Hyperlink-Induced Topic Search
- “A good **hub** is a page that points to many good authorities.”
- “A good **authority** is a page that is pointed to by many good hubs.”
- im Gegensatz zu PageRank berücksichtigt HITS auch den Inhalt von Webseiten
- 2 Phasen:
 - ▶ 1. Phase hängt von der Suchanfrage ab
 - ▶ 2. Phase beschäftigt sich nur mit der Linkstruktur des entsprechenden Netzwerks

Hubs & Authorities

- Suchanfrage σ
- 1. Phase berechnet $V_\sigma \in V$ mit
 - ▶ V_σ ist klein im Vergleich zu V ,
 - ▶ V_σ enthält viele Seiten, die für die Suchanfrage σ relevant sind,
 - ▶ V_σ enthält viele wichtige Authorities

Hubs & Authorities

Algorithmus 5: Hubs & Authorities, 1. Phase

Output : $V_\sigma =$ Menge relevanter Seiten

Benutze eine textbasierte Suchmaschine für Suchanfrage σ ;

Sei W_σ die Liste von Ergebnissen;

Wähle $t \in \mathbb{N}$;

Sei $W_\sigma^t \subset W_\sigma$ die Menge der t Seiten, die am höchsten bewertet wurden;

$V_\sigma := W_\sigma^t$;

forall the $i \in W_\sigma^t$ **do**

$V_\sigma := V_\sigma \cup \text{Succ}(i)$;

if $d_{in}(i) \leq r$ (*benutzerdefinierte Schranke*) **then**

$V_\sigma := V_\sigma \cup \text{Pred}(i)$;

else

 Wähle $\text{Pred}_r(i) \subseteq \text{Pred}(i)$ so, dass $|\text{Pred}_r(i)| = r$;

$V_\sigma := V_\sigma \cup \text{Pred}_r(i)$;

return V_σ ;

Hubs & Authorities

- 2. Phase berechnet die Hub/Authority-Werte der Seiten im induzierten Graphen $G[V_\sigma]$ aus der gegenseitigen Abhängigkeit zwischen Hubs und Authorities:

$\mathbf{c}_{\text{HA-H}} = A_\sigma \mathbf{c}_{\text{HA-A}}$ unter der Annahme, dass $\mathbf{c}_{\text{HA-A}}$ bekannt ist

$\mathbf{c}_{\text{HA-A}} = A_\sigma^T \mathbf{c}_{\text{HA-H}}$ unter der Annahme, dass $\mathbf{c}_{\text{HA-H}}$ bekannt ist

A_σ : Adjazenzmatrix von $G[V_\sigma]$

- Da weder $\mathbf{c}_{\text{HA-A}}$ noch $\mathbf{c}_{\text{HA-H}}$ bekannt sind, schlägt Kleinberg eine iterative Bestimmung mit Normalisierung vor.

Hubs & Authorities

Algorithmus 6: Hubs & Authorities Iteration

Output : Approximationen für $\mathbf{c}_{\text{HA-H}}$ and $\mathbf{c}_{\text{HA-A}}$

$$\mathbf{c}_{\text{HA-A}}^0 := \mathbf{1}_n;$$

for $k = 1 \dots$ **do**

$$\mathbf{c}_{\text{HA-H}}^k := A_{\sigma} \mathbf{c}_{\text{HA-A}}^{k-1};$$

$$\mathbf{c}_{\text{HA-A}}^k := A_{\sigma}^T \mathbf{c}_{\text{HA-H}}^k;$$

$$\mathbf{c}_{\text{HA-H}}^k := \frac{\mathbf{c}_{\text{HA-H}}^k}{\|\mathbf{c}_{\text{HA-H}}^k\|};$$

$$\mathbf{c}_{\text{HA-A}}^k := \frac{\mathbf{c}_{\text{HA-A}}^k}{\|\mathbf{c}_{\text{HA-A}}^k\|};$$

Hubs & Authorities

Satz

Sei A_σ die Adjazenzmatrix von $G[V_\sigma]$.

Dann sind die Grenzwerte

$$\lim_{k \rightarrow \infty} \mathbf{c}_{HA-A}^k = \mathbf{c}_{HA-A}$$

und

$$\lim_{k \rightarrow \infty} \mathbf{c}_{HA-H}^k = \mathbf{c}_{HA-H}$$

wobei \mathbf{c}_{HA-A} (\mathbf{c}_{HA-H}) der erste Eigenvektor von $A_\sigma^T A_\sigma$ ($A_\sigma A_\sigma^T$) ist.

Hubs & Authorities

⇒ iterative Methode ist nichts anderes als die Lösung der Eigenvektor-Gleichungen

$$\lambda \mathbf{c}_{\text{HA-A}} = (A_{\sigma}^T A_{\sigma}) \mathbf{c}_{\text{HA-A}}$$

$$\lambda \mathbf{c}_{\text{HA-H}} = (A_{\sigma} A_{\sigma}^T) \mathbf{c}_{\text{HA-H}}$$

für den größten Eigenwert per Power Iteration

$\mathbf{c}_{\text{HA-A}}$: enthält dann die Authority-Werte

$\mathbf{c}_{\text{HA-H}}$: enthält dann die Hub-Werte

Übersicht

- 1 Grundlagen
- 2 Zentralitätsindizes
- 3 Wiederholung: Kürzeste Wege
- 4 Algorithmen für Zentralitätsindizes
- 5 Lokale Dichte**
 - Kohäsive Gruppen
 - Cliques
 - Strukturell dichte Gruppen
 - Statistisch dichte Gruppen
- 6 Zusammenhang

Kohäsive Gruppen

Eigenschaften:

- **Gegenseitigkeit:**
Gruppenmitglieder wählen sich gegenseitig in die Gruppe und sind im graphtheoretischen Sinn benachbart
- **Kompaktheit / Erreichbarkeit:**
Gruppenmitglieder sind gegenseitig gut erreichbar (wenn auch nicht unbedingt adjazent), insbesondere
 - ▶ auf kurzen Wegen
 - ▶ auf vielen verschiedenen Wegen
- **Dichte:**
Gruppenmitglieder haben eine große Nachbarschaft innerhalb der Gruppe
- **Separation:**
Gruppenmitglieder haben mit größerer Wahrscheinlichkeit Kontakt zu einem anderen Mitglied der Gruppe als zu einem Nicht-Gruppenmitglied

Lokale Dichte

- Eine Gruppeneigenschaft heißt **lokal**, wenn sie bestimmt werden kann, indem man nur den von der Gruppe induzierten Teilgraphen betrachtet.
- ⇒ Separation ist *nicht lokal*, weil hier auch die Verbindungen zu den anderen Knoten betrachtet werden
- Einige Definitionen von kohäsiven Gruppen verlangen außer einer Eigenschaft Π auch **Maximalität** (im Sinne von Nichterweiterbarkeit), d.h. die Gruppe darf nicht in einer anderen größeren Gruppe enthalten sein.
- Maximalität verletzt die Lokalitätsbedingung
- ⇒ Betrachten diese Eigenschaften ohne Maximalitätsbedingung
- Lokalität reflektiert wichtige Eigenschaft von Gruppen:
 - ▶ Invarianz unter Veränderung des Netzwerks außerhalb der Gruppe
 - ▶ Innere Robustheit und Stabilität ist eine wichtige Gruppeneigenschaft

Cliques

Definition

Sei $G = (V, E)$ ein ungerichteter Graph.

Ein Knotenteilmenge $U \subseteq V$ heißt **Clique** genau dann, wenn der von U in G induzierte Teilgraph $G[U]$ ein vollständiger Graph ist.

Eine Clique U in G ist eine **maximale Clique**, falls es keine Clique U' mit $U \subset U'$ in G gibt.

Eine Clique U in G ist eine **Maximum-Clique**, falls es keine Clique U' mit $|U| < |U'|$ in G gibt.

Cliques – ideale kohäsive Gruppen

Cliques sind ideale kohäsive Gruppen:
(Sei U eine Clique der Kardinalität k .)

- Cliques haben größtmögliche Dichte

$$\delta(G[U]) = \bar{d}(G[U]) = \Delta(G[U]) = k - 1$$

- Cliques besitzen größtmögliche Kompaktheit

$$\text{diam}(G[U]) = 1$$

- Cliques sind bestmöglich verbunden
 U ist $(k - 1)$ -fach knotenzusammenhängend und
 $(k - 1)$ -fach kantenzusammenhängend

Satz von Turán

Satz (Turán, 1941)

Sei $G = (V, E)$ ein ungerichteter Graph mit $n = |V|$ und $m = |E|$.
Falls $m > \frac{n^2}{2} \cdot \frac{k-2}{k-1}$, dann existiert eine Clique der Größe k in G .

Spezialfall:

Satz (Mantel, 1907)

Die maximale Anzahl von Kanten in einem dreiecksfreien Graphen ist $\lfloor \frac{n^2}{4} \rfloor$.

Da die meisten (z.B. soziale) Netzwerke eher dünn sind (also $o(n^2)$ Kanten haben), müssen sie nicht unbedingt von vornherein Cliques einer bestimmten Größe > 2 enthalten.

Maximale Cliques

- Graphen enthalten mindestens eine maximale Clique, meistens sogar viele.
- Sie können sich überlappen (ohne identisch zu sein).

Satz (Moon & Moser, 1965)

Jeder ungerichtete Graph G mit n Knoten hat höchstens $3^{\lceil \frac{n}{3} \rceil}$ maximale Cliques.

Cliquen-Struktur

- Cliques sind **abgeschlossen unter Exklusion**, d.h. wenn U eine Clique in G ist und v ein Knoten aus U , dann ist $U \setminus \{v\}$ auch eine Clique.

Oder anders gesagt:

Die Cliqueneigenschaft ist eine **hereditäre** Grapheigenschaft, denn sie vererbt sich auf induzierte Teilgraphen.

- Cliques sind **geschachtelt**, d.h. jede Clique der Größe n enthält eine Clique der Größe $n - 1$ (sogar n davon).

Das folgt hier sofort aus dem Abschluss unter Exklusion.

Für andere Eigenschaften, die nicht unter Exklusion abgeschlossen sind, muss man das aber extra beweisen.

Generalisierte Cliques

Sei $G = (V, E)$ ein ungerichteter Graph, U eine Teilmenge der Knoten und $k > 0$ eine natürliche Zahl.

Generalisierte (distanz-basierte) Cliques:

- U heißt **k -Clique** g.d.w. $\forall u, v \in U : d_G(u, v) \leq k$.
- U heißt **k -Club** g.d.w. $\text{diam}(G[U]) \leq k$.
- U heißt **k -Clan** g.d.w. U ist eine maximale k -Clique und U ist ein k -Club.
- k -Cliques sind nicht lokal definiert (die Distanzen können sich aus Pfaden ergeben, die über Knoten außerhalb von U führen).
- Obwohl k -Clubs und k -Clans lokal definiert sind (abgesehen von der Maximalitätsbedingung), sind sie nur von geringerem Interesse. Distanz-basierte Cliques sind i.A. nicht abgeschlossen unter Exklusion und nicht geschachtelt.

Grundfunktionen

Für Graph $G = (V, E)$ mit $|V| = n$ und $|E| = m$ können folgende Funktionen in $\mathcal{O}(m + n)$ berechnet werden:

- Ist eine gegebene Knotenteilmenge $U \subseteq V$ eine Clique in G ?
Prüfe für jede Kante in G , ob beide Endknoten in U sind.
Zähle die Fälle und vergleiche mit $|U| \cdot (|U| - 1)/2$.
- Bestimme, ob eine gegebene Clique $U \subseteq V$ maximal ist in G .
Teste, ob es einen Knoten in $V \setminus U$ gibt, der adjazent zu allen Knoten in U ist.

Maximale Clique

Annahme: V ist eine geordnete Menge

Definiere folgende lexikographische **Ordnung** auf den Cliques ($U, U' \subseteq V$):

$U < U' \Leftrightarrow$ Der kleinste Knoten aus $U \cup U'$, der nicht in $U \cap U'$ ist, ist in U .

Aufgabe:

Bestimme die lexikographisch kleinste maximale Clique U , die eine gegebene Clique $U' \subseteq V$ enthält.

- Starte mit $U = U'$
- Iteriere über alle $v \in V \setminus U$ in aufsteigender Reihenfolge
 - ▶ Teste, ob $U \subseteq N(v)$.
 - ▶ Falls ja, dann füge v zu U hinzu.

\Rightarrow Am Ende ist U eine maximale Clique, die U' enthält.

- Laufzeit ebenfalls $\mathcal{O}(m + n)$
(falls $U \subseteq N(v)$ in $\mathcal{O}(\deg(v))$ getestet werden kann)

Cliques maximaler Kardinalität

Maximum-Clique:

Clique der größtmöglichen Kardinalität in einem gegebenen Graphen

Primitiver Algorithmus: erschöpfende Suche

- Zähle alle Kandidatensets $U \subseteq V$ auf und bestimme, ob U eine Clique ist.
- Gib die größte gefundene Clique aus.

⇒ Laufzeit: $\mathcal{O}((m+n) \cdot 2^n) \subseteq \mathcal{O}(n^2 \cdot 2^n)$

Geht das auch schneller?

Clique-(Entscheidungs-)Problem

Problem

Problem: **Clique**

Eingabe: Graph G , Größe $k \in \mathbb{N}$

Frage: Existiert eine Clique U der Größe $|U| \geq k$ in G ?

$\omega(G)$: Größe der Maximum-Clique(n) in G

Hätten wir einen Algorithmus, der **Clique** in Zeit $T(n)$ entscheidet, dann könnten wir $\omega(G)$ in Zeit $\mathcal{O}(T(n) \cdot \log n)$ mit binärer Suche berechnen.

Andererseits ergibt sich aus jedem Algorithmus zur Berechnung von $\omega(G)$ in Zeit $T(n)$ ein Algorithmus, der **Clique** in $T(n)$ entscheidet.

Ein polynomieller Algorithmus für das eine Problem würde also einen polynomiellen Algorithmus für das jeweils andere Problem implizieren.

Härte des Clique-Problems

Aber:

Satz

Clique ist *\mathcal{NP} -vollständig*, ebenso **IndependentSet**.

Beweis: Reduktion von **Satisfiability** (R. Karp, 1972)

Auch versteckte Cliques sind schwer zu finden:

Folgerung

Falls $\mathcal{P} \neq \mathcal{NP}$ gilt, gibt es keinen Polynomialzeit-Algorithmus, der eine Clique der Größe k in einem Graphen findet, der eine solche enthält.

Bemerkung:

Sogar sehr große Cliques der Kardinalität $(1 - \varepsilon) \cdot n$ (für $\varepsilon > 0$) können nicht in Polynomialzeit gefunden werden.

Maximum-Clique: besserer exponentieller Algorithmen

Satz

Eine Maximum-Clique (also eine Clique maximaler Kardinalität) kann in Zeit $\mathcal{O}^(1.3803^n)$ berechnet werden.*

(\mathcal{O}^* ignoriert polynomielle Faktoren)

Maximum-Clique: besserer exponentieller Algorithmus

$\delta(G)$: Minimalgrad in $G \Rightarrow$ in Clique U wäre $\delta(U) = |U| - 1$

Beweis.

- Falls $\delta(G) \geq n - 3$, dann
 - ▶ fehlen in G nur **einfache Pfade und Kreise** im Vergleich zum vollständigen Graphen K_n
 - ▶ betrachte zur Veranschaulichung den **komplementären** Graphen
- \Rightarrow Cliques entsprechen unabhängigen Mengen (independent sets)
 - ▶ Pfad P : größtest Independent Set hat $\lceil |V(P)|/2 \rceil$ Knoten
 - ▶ Kreis C : größtest Independent Set hat $\lfloor |V(C)|/2 \rfloor$ Knoten
 - ▶ Pfade / Kreise sind paarweise disjunkt
- \Rightarrow Man kann die einzelnen Werte einfach addieren.

\Rightarrow Maximum Clique kann in $\mathcal{O}(m + n)$ berechnet werden.

Maximum-Clique: besserer exponentieller Algorithmus

Beweis.

- Ansonsten sei v ein Knoten mit Grad $\deg_G(v) \leq n - 4$.
- Jede Maximum-Clique kann v enthalten oder nicht, ist also
 - ▶ $\{v\}$ vereinigt mit einer Maximum-Clique von $G[N(v)]$ oder
 - ▶ eine Maximum-Clique von $G[V \setminus \{v\}]$.

⇒ Rekursive Berechnung mit worst-case-Zeit

$$T(n) \leq T(n - 4) + T(n - 1) + c \cdot (m + n)$$

⇒ mit Erzeugendenfunktionen kann man zeigen, dass $T(n) \in \mathcal{O}^*(\beta^n)$ mit $\beta \approx 1.3803$, wobei β die größte reelle Nullstelle des charakteristischen Polynoms $\beta^4 - \beta^3 - 1$ ist



Approximation von Maximum-Cliques

Approximation der größten Clique mit Faktor $n/2$ ist einfach:
wähle die Endknoten einer Kante, falls es eine gibt.

Satz (Boppana & Halldórsson, 1990)

Es gibt einen Algorithmus, der bei Eingabe eines Graphen G mit n Knoten in polynomieller Zeit eine Clique ausgibt, deren Größe maximal um einen Faktor $\mathcal{O}\left(\frac{n}{(\log n)^2}\right)$ von der Maximum-Cliquengröße $\omega(G)$ abweicht.

Satz (Håstad, 1996)

Falls nicht $\mathcal{NP} = \mathcal{ZPP}$ gilt, dann existiert kein Polynomialzeitalgorithmus, der bei Eingabe eines Graphen G mit n Knoten eine Clique ausgibt, deren Größe maximal um einen Faktor $n^{1-\epsilon}$ von der Maximum-Cliquengröße $\omega(G)$ abweicht (für jedes $\epsilon > 0$).

\mathcal{ZPP} : Klasse der Probleme, die von randomisierten Algorithmen in erwarteter Polynomialzeit gelöst werden können ohne Fehler zu machen.

Suche nach Cliques fester Größe

- In einigen Fällen reicht es, nach Cliques fester Größe zu suchen
⇒ Cliquengröße wird **nicht als Teil der Eingabe** betrachtet

- Vollständige Suche: $\mathcal{O}(k^2 \cdot n^k) = \mathcal{O}(n^k)$,
wenn Cliquengröße k fest ist

Bessere Komplexität für Dreiecke

- A : Adjazenzmatrix von Graph G

- $B = A^2 = A \cdot A$

Einträge b_{ij} : Anzahl Wege der Länge 2 zwischen Knoten v_i und v_j

⇒ läßt sich durch schnellere Matrixmultiplikation ausrechnen,
z.B. in $\mathcal{O}(n^{2.376})$ (Coppersmith / Winograd, 1990)

- Existiert ein Eintrag $b_{ij} \geq 1$ mit $i \neq j$, dann gibt es einen Knoten $u \in V$, der zu v_i und zu v_j adjazent ist.

- Falls nun auch Kante $\{v_i, v_j\}$ existiert, dann enthält der Graph ein Dreieck $\{v_i, v_j, u\}$.

⇒ Checke für alle $b_{ij} > 0$, ob es Kante $\{v_i, v_j\}$ gibt

⇒ Zeit-Komplexität: $\mathcal{O}(n^\alpha)$, wobei $\alpha < 2.376$ der Exponent für Matrixmultiplikation ist

Cliquen fester Größe $k \geq 3$

Sei $\alpha(r, s, t)$ so definiert, dass man die Multiplikation einer $n^r \times n^s$ -Matrix mit einer $n^s \times n^t$ -Matrix in Zeit $\mathcal{O}(n^{\alpha(r,s,t)})$ berechnen kann.

Satz

Für jedes $k \geq 3$ existiert ein Algorithmus, der eine Clique der Größe k in einem Graphen mit n Knoten finden kann (falls eine solche existiert) und der in Zeit $\mathcal{O}(n^{\beta(k)})$ läuft, wobei $\beta(k) = \alpha(\lfloor k/3 \rfloor, \lceil (k-1)/3 \rceil, \lceil k/3 \rceil)$.

Cliques fester Größe $k \geq 3$

Beweis.

- Seien $k_1 = \lfloor k/3 \rfloor$, $k_2 = \lceil (k-1)/3 \rceil$ und $k_3 = \lceil k/3 \rceil$

$$\Rightarrow k = k_1 + k_2 + k_3$$

- Konstruiere tripartiten Hilfsgraph $\tilde{G} = (\tilde{V}, \tilde{E})$

- ▶ $\tilde{V} = \tilde{V}_1 \cup \tilde{V}_2 \cup \tilde{V}_3$

- ▶ \tilde{V}_i : alle Cliques der Größe k_i in G

- (mit dem naiven Algorithmus in $\mathcal{O}(n^{k_i})$ berechenbar)

- ▶ Knoten $U \in \tilde{V}_i$ und $U' \in \tilde{V}_j$ sind adjazent in \tilde{G} , also $(U, U') \in \tilde{E}$,
 $\Leftrightarrow i \neq j$ und $U \cup U'$ ist Clique der Größe $k_i + k_j$ in G

- ▶ Berechne dazu die partiellen Adjazenzmatrizen A_{12} , A_{13} und A_{23} , die die Kanten zwischen den jeweiligen Knotenmengen darstellen in $\mathcal{O}(n^{k_1+k_2})$, $\mathcal{O}(n^{k_1+k_3})$ bzw. $\mathcal{O}(n^{k_2+k_3})$

- Teste \tilde{G} auf Dreiecke

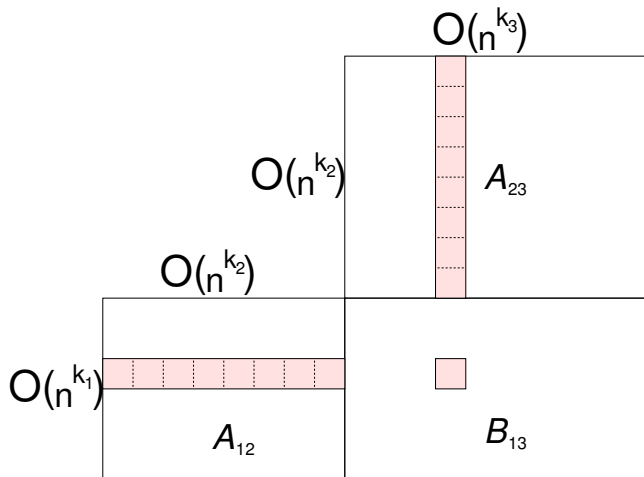
Cliques fester Größe $k \geq 3$

Beweis.

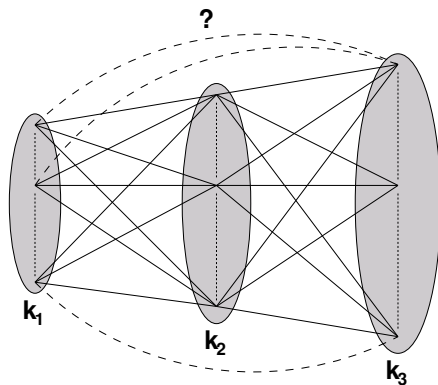
- Dreieck $\{U_1, U_2, U_3\}$ impliziert eine Clique der Größe k in G
- geht mit schneller Matrixmultiplikation, aber hier muss eine $\mathcal{O}(n^{k_1}) \times \mathcal{O}(n^{k_2})$ -Matrix (Kanten zwischen \tilde{V}_1 und \tilde{V}_2) mit einer $\mathcal{O}(n^{k_2}) \times \mathcal{O}(n^{k_3})$ -Matrix (Kanten zwischen \tilde{V}_2 und \tilde{V}_3) multipliziert werden, und zwar in Zeit $\mathcal{O}(n^{\beta(k)})$
- Berechnung der drei Matrizen A_{12} , A_{23} und A_{13} in $\mathcal{O}(n^{\max\{k_1+k_2, k_1+k_3, k_2+k_3\}}) \subseteq \mathcal{O}(n^{\lceil \frac{2k}{3} \rceil})$
(wird dominiert durch die Zeit $\mathcal{O}(n^{\beta(k)})$ für die Multiplikation der rechteckigen Matrizen, also $B_{13} = A_{12} \cdot A_{23}$)
- Für jeden echt positiven Eintrag in B_{13} wird dann wieder geprüft, ob der entsprechende Eintrag in A_{13} gleich 1 ist.



Cliques fester Größe $k \geq 3$



B_{13} wird dann mit A_{13} verknüpft

Cliques fester Größe $k \geq 3$ 

Cliques fester Größe: Beispielkomplexitäten

Cliquengröße	Vollständige Suche	Matrixmultiplikation
3	$\mathcal{O}(n^3)$	$\mathcal{O}(n^{2.376})$
4	$\mathcal{O}(n^4)$	$\mathcal{O}(n^{3.376})$
5	$\mathcal{O}(n^5)$	$\mathcal{O}(n^{4.220})$
6	$\mathcal{O}(n^6)$	$\mathcal{O}(n^{4.751})$
7	$\mathcal{O}(n^7)$	$\mathcal{O}(n^{5.751})$
8	$\mathcal{O}(n^8)$	$\mathcal{O}(n^{6.595})$

Aufzählen von Cliques

Wie kann man (maximale und Maximum-)Cliques aufzählen?

- Ausgabe ist oft exponentiell in der Eingabelänge

⇒ etwas anderer Effizienzbegriff

polynomielle Gesamtzeit:

bei Ausgabe von C Konfigurationen Begrenzung der Zeit durch ein Polynom in C und in der Eingabegröße n (output-sensitiv)

- vollständige Suche ist *nicht* in polynomieller Gesamtzeit
- Aufzählung aller maximalen Cliques geht in polynomieller Gesamtzeit (ein klassischer Algorithmus läuft z.B. erst $\mathcal{O}(n^2 C)$ Schritte ohne Ausgabe und gibt dann alle maximalen Cliques auf einmal aus)
- Aufzählung aller *Maximum*-Cliques geht nicht in polynomieller Gesamtzeit, falls $\mathcal{P} \neq \mathcal{NP}$

'Effiziente' Aufzählungsalgorithmen

Weitere Effizienzbeschreibung:

Polynomielle Verzögerung (polynomial delay):

Zeit

- bis zur Ausgabe der ersten Konfiguration,
- zwischen zwei aufeinanderfolgenden Ausgaben von Konfigurationen und
- von der Ausgabe der letzten Konfiguration bis zum Stop

ist polynomiell in der Eingabegröße

Satz

*Es gibt einen Algorithmus, der alle maximalen Cliques mit (polynomieller) **Verzögerung** $\mathcal{O}(n^3)$ und (linearem) **Platzverbrauch** $\mathcal{O}(m+n)$ aufzählt.*

Algorithmus zur Aufzählung maximaler Cliques

- Konstruiere (unvollständigen) **Binärbaum** mit n Levels, dessen Blätter sich nur auf Level n befinden
 - Jedes Level ist einem Knoten von G zugeordnet, auf Level i betrachtet man Knoten v_i .
 - Knoten von Level i des Baums entsprechen den maximalen Cliques des induzierten Teilgraphen $G[\{v_1, \dots, v_i\}]$.
- ⇒ Die Blätter sind genau die maximalen Cliques von G .
- Zur Bestimmung der Kinder auf Level $i + 1$ für eine maximale Clique U in $G[\{v_1, \dots, v_i\}]$ (auf Level i) unterscheiden wir:
 - 1 Alle Knoten von U sind adjazent zu v_{i+1} in G .
 - 2 Es existiert ein Knoten in U , der nicht zu v_{i+1} adjazent ist in G

Algorithmus zur Aufzählung maximaler Cliques

Fallunterscheidung:

- ① Alle Knoten von U sind adjazent zu v_{i+1} in G .
 - ⇒ $U \cup \{v_{i+1}\}$ ist maximale Clique in $G[\{v_1, \dots, v_{i+1}\}]$
 - ▶ Das ist die einzige Möglichkeit, eine maximale Clique in $G[\{v_1, \dots, v_{i+1}\}]$ zu bekommen, die U enthält
 - In diesem Fall hat U nur ein einziges Kind im Baum.

- ② Es existiert ein Knoten in U , der nicht zu v_{i+1} adjazent ist in G

Man kann 2 maximale Cliques in $G[\{v_1, \dots, v_{i+1}\}]$ erhalten:

 - ① U ist selbst eine maximale Clique
 - ② $(U \setminus \bar{N}(v_{i+1})) \cup \{v_{i+1}\}$ ist eine Clique,
 - wobei $\bar{N}(v_{i+1})$ alle nicht mit v_{i+1} adjazenten Knoten sind
 - ★ Wenn die Menge eine *maximale* Clique ist, hätte U zwei Kinder im Baum
 - ★ $(U \setminus \bar{N}(v_{i+1})) \cup \{v_{i+1}\}$ könnte aber Kind von mehreren sein
 - ⇒ Kind der lexikographisch kleinsten Menge U (falls maximal)

⇒ Binärbaum

(interne Knoten haben 1 oder 2 Kinder, Blätter nur in Level n)

Algorithmus zur Aufzählung maximaler Cliques

- Traversiere den Binärbaum per Tiefensuche (DFS)
- Ausgabe aller Blätter
- Gegeben Knoten U auf Level i :
 - ▶ $\text{Parent}(U, i)$:
 Vaterknoten von U ist die lexikographisch kleinste maximale Clique in $G[\{v_1, \dots, v_{i-1}\}]$, die $U \setminus \{v_i\}$ enthält
 \Rightarrow Grundfunktion, in $\mathcal{O}(m + n)$
 - ▶ $\text{LeftChild}(U, i)$:
 - ★ falls $U \subseteq N(v_{i+1})$ (1. Fall), dann $U \cup \{v_{i+1}\}$
 - ★ falls $U \not\subseteq N(v_{i+1})$ (Teil des 2. Falls), dann U
 - ★ Unterscheidung der Fälle kostet $\mathcal{O}(m + n)$ Zeit
 - ▶ $\text{RightChild}(U, i)$:
 - ★ falls $U \subseteq N(v_{i+1})$, dann existiert kein rechtes Kind
 - ★ falls $U \not\subseteq N(v_{i+1})$, dann
 $(U \setminus \bar{N}(v_{i+1})) \cup \{v_{i+1}\}$ falls es maximale Clique ist und
 $U = \text{Parent}((U \setminus \bar{N}(v_{i+1})) \cup \{v_{i+1}\}, i + 1)$
 sonst keins
 - ★ Kosten: $\mathcal{O}(m + n)$ Zeit

Algorithmus zur Aufzählung maximaler Cliques

- Längster Pfad zwischen zwei Blättern im Baum ist $2n - 2$ und geht durch $2n - 1$ Knoten
 - pro Knoten Zeitaufwand $\mathcal{O}(m + n)$
 - Jeder Unterbaum hat ein Blatt auf Level n
- ⇒ Ausgabeverzögerung $\mathcal{O}(n^3)$
-
- Wenn ein Knoten bearbeitet wird, muss nur die Menge U , das Level i , sowie ein Label zur Unterscheidung von linkem/rechten Kind gespeichert werden
- ⇒ Speicheraufwand $\mathcal{O}(m + n)$

Aufzählung in lexikographischer Reihenfolge

- Es ist \mathcal{NP} -vollständig für einen Graphen G und eine maximale Clique U von G zu entscheiden, ob es eine maximale Clique U' gibt, die lexikographisch größer ist als U .
- Falls $\mathcal{P} \neq \mathcal{NP}$, dann gibt es keinen Algorithmus, der in Polynomialzeit zu einem Graphen G und einer maximalen Clique U von G die lexikographisch nächstgrößere maximale Clique generiert.
- Falls $\mathcal{P} \neq \mathcal{NP}$, dann gibt es keinen Algorithmus, der für jeden gegebenen Graphen alle maximalen Cliques in inverser lexikographischer Reihenfolge mit polynomieller Verzögerung aufzählt.
- Überraschenderweise kann man trotzdem alle maximalen Cliques in lexikographischer Ordnung mit polynomieller Verzögerung aufzählen.

Aufzählung in lexikographischer Reihenfolge

- Idee:

Während der Generierung der aktuellen Ausgabe wird schon zusätzliche Arbeit in die Erzeugung lexikographisch größerer Cliques investiert.

⇒ Diese werden in einer Priority Queue gespeichert, die dann u.U. exponentiell viele Cliques enthält und damit auch exponentiell viel Speicherplatz braucht.

Aufzählung in lexikographischer Reihenfolge

Algorithmus 7: Alg. von Johnson, Papadimitriou und Yannakakis

$U_0 :=$ erste (lexikographisch kleinste) maximale Clique;

Füge U_0 in Priority Queue Q ein;

while Q ist nicht leer **do**

$U := \text{ExtractMin}(Q)$;

Ausgabe U ;

foreach Knoten v_j von G , der zu einem Knoten $v_i \in U$ mit $i < j$ nicht adjazent ist **do**

$U_j := U \cap \{v_1, \dots, v_j\}$;

if $(U_j - \overline{N}(v_j)) \cup \{v_j\}$ ist eine maximale Clique in $G[\{v_1, \dots, v_j\}]$

then

Sei T die lexikographisch kleinste maximale Clique, die

$(U_j - \overline{N}(v_j)) \cup \{v_j\}$ enthält;

Füge T in Q ein

Aufzählung in lexikographischer Reihenfolge

Satz

Der Algorithmus von Johnson, Papadimitriou und Yannakakis zählt alle maximalen Cliques eines gegebenen Graphen mit n Knoten in lexikographischer Reihenfolge und mit Delay $\mathcal{O}(n^3)$ auf.

Aufzählung in lexikographischer Reihenfolge

Beweis.

Korrektheit (Reihenfolge):

- Menge T ist beim Einfügen in Q (bei Betrachtung von U) lexikographisch größer als U
(denn es wird ja aus U zumindest Knoten v_i entnommen während lediglich v_j hinzukommt und es gilt $i < j$)
- ⇒ Es werden nur Mengen in der Priority Queue gespeichert, die erst nach U ausgegeben werden dürfen.
- ⇒ Die ausgegebenen maximalen Cliques sind lexikographisch aufsteigend.

Aufzählung in lexikographischer Reihenfolge

Beweis.

Vollständigkeit:

- Falls U die lexikographisch kleinste noch auszugebende Clique ist, dann ist U in Q .
- Induktionsanfang: Für $U = U_0$ ist das korrekt.
- Induktionsschritt: Sei U lexikographisch größer als U_0 .
 - ▶ Sei j der größte Index, dass $U_j = U \cap \{v_1, \dots, v_j\}$ keine maximale Clique in $G[\{v_1, \dots, v_j\}]$.
(Muss existieren, weil sonst $U = U_0$.)
 - ▶ $j < n$, weil $U = U_n$ maximale Clique des Gesamtgraphen G ist
 - ▶ Aufgrund der Maximalität von j muss gelten $v_{j+1} \in U$.
 - ▶ \exists Menge S : $U_j \cup S$ ist maximale Clique in $G[\{v_1, \dots, v_j\}]$

Aufzählung in lexikographischer Reihenfolge

Beweis.

- Induktionsschritt (Fortsetzung):

- ▶ Aufgrund der Maximalität von j ist v_{j+1} nicht adjazent zu allen Knoten in S .
- ⇒ \exists maximale Clique U' , die zwar $U_j \cup S$, aber nicht v_{j+1} enthält
- ▶ $U' \leq U$, weil sie sich in S unterscheiden
- ▶ U' wurde schon ausgegeben (Ind.voraussetzung)
- ▶ Als U' ausgegeben wurde, wurde festgestellt, dass v_{j+1} nicht adjazent ist zu einem Knoten $v_i \in U'$ mit $i < j + 1$
- ▶ $(U'_{j+1} \setminus \bar{N}(v_{j+1})) \cup \{v_{j+1}\} = U_{j+1}$
und U_{j+1} ist maximale Clique in $G[\{v_1, \dots, v_{j+1}\}]$
- ⇒ Die lexikographisch kleinste maximale Clique, die U_{j+1} enthält, wurde in Q eingefügt.

Aufzählung in lexikographischer Reihenfolge

Beweis.

- Induktionsschritt (Fortsetzung):

- ▶ Aufgrund der Maximalität von j stimmen U und T in den ersten $j + 1$ Knoten überein.
 - ▶ Annahme: $U \neq T$
Sei k der kleinste Index eines Knoten v_k , der in genau einer der beiden Mengen ist.
 - ▶ $k > j + 1$
 - ▶ Da $T \leq U$, gilt $v_k \in T$ und $v_k \notin U$.
- ⇒ U_k ist nicht maximale Clique in $G[\{v_1, \dots, v_k\}]$
(Widerspruch zur Maximalität von j)
- ⇒ $U = T$
- ⇒ U ist in der Priority Queue Q



Aufzählung in lexikographischer Reihenfolge

Komplexität:

- Extraktion der lexikographisch kleinsten maximalen Clique aus Q : $\mathcal{O}(n \log C)$
- n Berechnungen von maximalen Cliques, die eine gegebene Menge enthalten: $\mathcal{O}(m + n)$ pro Menge
- Einfügen einer maximalen Clique in Q : $\mathcal{O}(n \log C)$ pro Clique
- Da $C \leq 3^{\lceil \frac{n}{3} \rceil}$, folgt dass die Verzögerung $\mathcal{O}(n^3)$ ist.

Plex

Relaxiere Cliquesbegriff, so dass konstant viele Verbindungen zu Gruppenmitgliedern bei jedem Knoten fehlen dürfen.

Definition

Sei $G = (V, E)$ ein ungerichteter Graph und sei $k \in \{1, \dots, n - 1\}$.

Eine Knotenteilmenge $U \subseteq V$ heißt **k -Plex**, falls $\delta(G[U]) \geq |U| - k$.

Ein *maximales k -Plex* ist in keinem größeren k -Plex echt enthalten.

Ein *Maximum k -Plex* hat maximale Kardinalität unter allen k -Plexen in G .

- Jede Clique ist ein 1-Plex.
- Jedes k -Plex ist auch ein $(k + 1)$ -Plex.
- Jeder induzierte Teilgraph eines k -Plex ist auch ein k -Plex, d.h. die k -Plex-Eigenschaft ist abgeschlossen unter Exklusion.

k -Plex-Durchmesser

Satz

Sei $G = (V, E)$ ein ungerichteter Graph auf n Knoten und sei V ein k -Plex mit $k \in \{1, \dots, n-1\}$.

Dann gilt:

- 1 Wenn $k < \frac{n+2}{2}$ ist, dann gilt $\text{diam}(G) \leq 2$.

Falls zusätzlich $n \geq 4$, dann ist G zweifach kantenzusammenhängend.

- 2 Wenn $k \geq \frac{n+2}{2}$ und G zusammenhängend ist, dann gilt $\text{diam}(G) \leq 2k - n + 2$.

k -Plex-Durchmesser

Beweis.

Fall $k < \frac{n+2}{2}$:

- für adjazente Knoten u und v gilt $d(u, v) = 1$
- Seien $u, v \in V$ also nicht adjazente Knoten ($u \neq v$)
- Annahme: $d(u, v) \geq 3 \Rightarrow N(u) \cap N(v) = \emptyset$, d.h.

$$n - 2 \geq |N(u) \cup N(v)| \geq 2\delta(G) \geq 2(n - k) > \dots$$

$$\dots > 2 \left(n - \frac{n+2}{2} \right) = n - 2$$

(Widerspruch)

$$\Rightarrow d(u, v) \leq 2 \Rightarrow \text{diam}(G) \leq 2$$

k -Plex-Durchmesser

Beweis.

Fall $k < \frac{n+2}{2}$, $n \geq 4$:

- Annahme: \exists Brücke e , d.h. $G - \{e\}$ enthält zwei Zusammenhangskomponenten V_1 und V_2
 - Jeder kürzeste Pfad von einem Knoten in V_1 zu einem Knoten in V_2 muss diese Brücke benutzen.
- \Rightarrow Da $\text{diam}(G) \leq 2$, muss eine Komponente ein einzelner Knoten sein. Dieser Knoten v hat Grad $\text{deg}(v) = 1$.
- Da V ein k -Plex mit $n \geq 4$ Knoten ist, gilt für den Grad dieses Knotens v aber auch: $\text{deg}(v) \geq n - k > n - \frac{n+2}{2} = \frac{n-2}{2} \geq 1$ (Widerspruch)
- \Rightarrow Es existiert keine Brücke in G bzw. G ist **2-fach kantenzusammenhängend**.

k -Plex-Durchmesser

Beweis.

Fall $k \geq \frac{n+2}{2}$:

- Sei $\{v_0, v_1, \dots, v_r\}$ ein längster kürzester Pfad, also ein Pfad mit $d(v_0, v_r) = r = \text{diam}(G)$.
- Annahme: $r \geq 4$ (sonst ist $r < 4 = 2 \frac{n+2}{2} - n + 2 \leq 2k - n + 2$)
- Da es keinen kürzeren Pfad zwischen v_0 und v_r gibt, ist v_i zu keinem der Knoten $v_0, \dots, v_{i-2}, v_{i+2}, \dots, v_r$ auf dem Pfad adjazent, außer zu seinen Pfad-Nachbarknoten v_{i-1} und v_{i+1}
- Außerdem kann kein Knoten existieren, der gleichzeitig zu v_0 und zu v_3 adjazent ist.

$$\Rightarrow \{v_0\} \uplus \{v_2, v_3, \dots, v_r\} \uplus (N(v_3) \setminus \{v_2, v_4\}) \subseteq \overline{N}(v_0)$$

$$\Rightarrow 1 + (r - 1) + d_G(v_3) - 2 \leq k \quad \Rightarrow \quad r + (n - k) - 2 \leq k$$

$$\Rightarrow \text{diam}(G) = r \leq 2k - n + 2$$



(k-)Plex Problem

Problem

Problem: **Plex**

Eingabe: Graph G ,
Größenparameter ℓ und
Plex-Parameter k

Frage: Existiert ein k -Plex der Kardinalität $\geq \ell$ in G ?

- (variables) Entscheidungsproblem **Plex** ist \mathcal{NP} -vollständig, da das **Clique**-Problem sich darauf reduzieren lässt (mit $k = 1$)

⇒ Betrachte das Problem für feste Plex-Parameter c

c-Plex Problem

Problem

Problem: **c-Plex**

Eingabe: Graph G , Parameter $\ell \in \mathbb{N}$

Frage: Existiert ein c -Plex der Kardinalität $\geq \ell$ in G ?

Genau wie **1-Plex=Clique** sind auch alle anderen Probleme für einen festen Plex-Parameter c \mathcal{NP} -vollständig:

Satz

c-Plex ist \mathcal{NP} -vollständig für alle $c \in \mathbb{N}$.

Beweis: durch Reduktion von **Clique**, siehe z.B. Brandes/Erlebach (Eds.): Network Analysis (S.128).

Cores

Relaxiere Cliquesbegriff, so dass konstant viele Verbindungen zu Gruppenmitgliedern bei jedem Knoten mindestens vorhanden sein müssen.

Definition

Sei $G = (V, E)$ ein ungerichteter Graph und sei $k \in \{1, \dots, n - 1\}$ eine natürliche Zahl.

Dann wird ein Subset $U \subseteq V$ als **k -Core** (k -Kern) bezeichnet, falls $\delta(G[U]) \geq k$.

Ein *maximaler k -Core* ist in keinem größeren k -Core echt enthalten.

Ein *Maximum k -Core* hat maximale Kardinalität unter allen k -Cores in G .

k -Core-Eigenschaften

- Jeder Graph G ist ein $\delta(G)$ -Core, jede Clique ist ein $(n - 1)$ -Core.
- Jeder $(k + 1)$ -Core ist auch ein k -Core.
- Jeder k -Core ist ein $(n - k)$ -Plex.
- Wenn U und U' k -Cores sind, dann ist auch $(U \cup U')$ ein k -Core.

⇒ Maximale k -Cores sind einzigartig.

- Nicht jeder induzierte Teilgraph eines k -Cores ist auch wieder ein k -Core, d.h. die k -Core-Eigenschaft ist *nicht* abgeschlossen unter Exklusion.

Bsp.: Jeder Kreis ist ein 2-Core, aber jeder echte Teilgraph enthält mindestens einen Knoten vom Grad < 2 .

- k -Cores sind auch nicht geschachtelt. (Nicht jeder k -Core der Größe n enthält einen k -Core der Größe $n - 1$.)
- k -Cores müssen nicht unbedingt zusammenhängend sein.

Maximale zusammenhängende k -Cores

Fakt

Sei $G = (V, E)$ ein ungerichteter Graph und $k > 0$ eine natürliche Zahl. Wenn U und U' zwei maximale zusammenhängende k -Cores in G mit $U \neq U'$ sind, dann gibt es keine Kante zwischen U und U' .

Folgerung

- Der einzige Maximum k -Core eines Graphen ist die Vereinigung seiner maximalen zusammenhängenden k -Cores.
- Der Maximum 2-Core eines zusammenhängenden Graphen ist zusammenhängend.
(Wenn er es nicht wäre, könnte man die Teile verbinden und alle neuen Knoten hätten mindestens Grad 2.)
- Ein Graph ist genau dann ein Wald, wenn er keine 2-Cores enthält.

Der Maximum k -Core

Satz

Sei $G = (V, E)$ ein ungerichteter Graph und $k > 0$ eine natürliche Zahl. Wenn man wiederholt alle Knoten mit Grad $< k$ (und alle inzidenten Kanten) entfernt, dann entspricht die verbleibende Knotenmenge U genau dem Maximum k -Core.

Beweis.

- U ist ein k -Core, d.h. wir müssen nur noch Maximum zeigen.
 - Annahme: U ist nicht Maximum k -Core.
- ⇒ ∃ nichtleere Menge $T \subseteq V$, so dass $U \cup T$ Maximum k -Core
- Als der erste Knoten von T aus G entfernt wurde, muss er Grad $< k$ gehabt haben. Das kann aber nicht sein, da er mindestens k Nachbarn in $U \cup T$ hat und alle anderen Knoten noch im Graph enthalten waren. (Widerspruch)



Core-Zerlegung

Definition

Die Core-Zahl $\xi_G(v)$ eines Knotens $v \in V$ ist die höchste Zahl k , so dass v Teil eines k -Cores im Graphen G ist, d.h.

$$\xi_G(v) = \max\{k : \exists k\text{-Core } U \text{ in } G \text{ mit } v \in U\}$$

Algorithmus zur Berechnung arbeitet nach folgendem Prinzip:

- Jeder Graph G ist ein $\delta(G)$ -Core.
- Jeder Nachbarknoten mit geringerem Grad verringert die potentielle Core-Zahl eines Knotens.

Berechnung der Core-Zahlen

Algorithmus 8: Berechnung der Core-Zahlen

Input : Graph $G = (V, E)$

Output : Array ξ_G mit den Core-Zahlen aller Knoten in G

Berechne die Grade aller Knoten und speichere sie in D ;

Sortiere V in aufsteigender Grad-Folge D ;

foreach $v \in V$ in sortierter Reihenfolge **do**

$\xi_G(v) := D[v]$;

foreach *Nachbar* u von v **do**

if $D[u] > D[v]$ **then**

$D[u] := D[u] - 1$;

Sortiere V in aufsteigender Grad-Reihenfolge nach D

Berechnung der Core-Zahlen

Komplexität:

- naiv: $\mathcal{O}(mn \log n)$
(teuerste Operationen: Sortieren der Knoten nach dem Grad)
- besser: $\mathcal{O}(m + n)$
 - ▶ Knotengrade sind Werte aus dem Intervall $[0, n - 1]$
 \Rightarrow Sortieren mit n Buckets in $\mathcal{O}(n)$
 - ▶ Nachsortieren kann man sich sparen: man merkt sich, an welchen Indizes im Array die Knoten des nächsten Grads beginnen.
Beim Dekrementieren des Werts eines Knotens kommt der Knoten einfach an den Anfang des alten Intervalls und dann wird die Grenze um eine Stelle verschoben, so dass er nun zum Intervall des kleineren Grads gehört (in $\mathcal{O}(1)$).
 - ▶ $\mathcal{O}(n)$ für Initialisierung / Sortieren,
 $\mathcal{O}(m)$ für die Schleifendurchläufe (jede Kante wird höchstens zweimal betrachtet), also insgesamt $\mathcal{O}(m + n)$

Dichte Subgraphen

Definition

Sei $G = (V, E)$ ein ungerichteter Graph mit $n > 1$ Knoten und m Kanten.
Dichte $\rho(G)$ des Graphen G :

$$\rho(G) = \frac{m}{\binom{n}{2}}$$

Ein durch eine Knotenteilmenge $U \subseteq V$ eines Graphen $G = (V, E)$ induzierter Teilgraph heißt η -dicht (für eine reelle Zahl η mit $0 \leq \eta \leq 1$), falls gilt:

$$\rho(G[U]) \geq \eta$$

Aber: selbst (Teil-)Graphen mit hoher Dichte können isolierte Knoten enthalten!

Eigenschaften

- Cliques sind 1-dichte (Teil-)Graphen.
- Ein k -Plex hat Dichte $1 - \frac{k-1}{n-1}$
- ⇒ Für $n \rightarrow \infty$ gilt für k -Plexe $\eta \rightarrow 1$ bzw.
 $\forall k > 0, 0 \leq \eta \leq 1$: ein k -Plex der Größe mindestens $\frac{k-\eta}{1-\eta}$ ist ein η -dichter (Teil-)Graph.
- Aber: nicht jeder $1 - \frac{k-1}{n-1}$ -dichte (Teil-)Graph ist auch ein k -Plex.
- Ein k -Core ist ein $\frac{k}{n-1}$ -dichter (Teil-)Graph.
 Die Dichte von k -Cores kann sich für $n \rightarrow \infty$ beliebig nah an 0 annähern.
- η -dichte Graphen sind *nicht* abgeschlossen unter Exklusion (Knotenausschluss), sie sind aber geschachtelt.

Schachtelung von η -dichten Graphen

Satz

Sei η eine reelle Zahl mit $0 \leq \eta \leq 1$, dann gilt:

Ein η -dichter Teilgraph der Größe $\ell > 2$ eines Graphen G enthält einen η -dichten Teilgraph der Größe $\ell - 1$ in G .

Schachtelung von η -dichten Graphen

Beweis.

Sei

- U ein η -dichter Teilgraph von G mit $|U| = \ell$,
- m_U die Anzahl der Kanten in $G[U]$,
- v ein Knoten mit minimalem Grad in $G[U]$,
also $\deg_{G[U]}(v) = \delta(G[U])$

Dann

- $\delta(G[U]) \leq \bar{d}(G[U]) = \frac{2m_U}{\ell} = \rho(G[U]) \cdot (\ell - 1)$
 - Betrachte Knotenteilmenge $U' = U \setminus \{v\}$ mit Kantenanzahl
 $m_U - \delta(G[U]) \geq \rho(G[U]) \cdot \binom{\ell}{2} - \rho(G[U]) \cdot (\ell - 1) = \rho(G[U]) \cdot \binom{\ell-1}{2}$
- $\Rightarrow \rho(G[U']) \geq \rho(G[U]) \geq \eta$
- $\Rightarrow U'$ ist ein η -dichter Teilgraph der Größe $\ell - 1$.



Dichte und Wege

- Definition der Dichte entspricht der durchschnittlichen Existenz von möglichen Kanten innerhalb eines (induzierten) Teilgraphen
- Jede ungerichtete Kante entspricht zwei gerichteten Wegen der Länge 1.
- Verallgemeinerung: Dichte auf der Basis (gerichteter) **Wege (Walks)** beliebiger Länge (mit möglichen Knoten-/Kantenwiederholungen)
- Grad der Ordnung $\ell \in \mathbb{N}$ eines Knotens v ,
Anzahl der Wege der Länge ℓ in G , die in v starten: $w_\ell(v)$
- $w_0(v) = 1$, $w_1(v) = \deg(v)$
- Anzahl Wege der Länge ℓ in einem Graphen G : w_ℓ
- $w_0 = n$, $w_1 = 2m$

Anzahl Wege der Länge ℓ

Satz

Sei $G = (V, E)$ ein ungerichteter Graph.

Für alle $\ell \in \mathbb{N}$ und für alle $r \in \{0, \dots, \ell\}$ gilt:

$$w_\ell(G) = \sum_{v \in V} w_r(v) \cdot w_{\ell-r}(v)$$

Anzahl Wege der Länge ℓ

Beweis.

- Jeder Weg der Länge ℓ besteht aus (nicht unbedingt verschiedenen) Knoten v_0, \dots, v_ℓ .
 - Betrachte nun von jedem solchen Weg den Knoten v_r (für ein festgelegtes $r \in \{0, \dots, \ell\}$).
 - Es gibt $w_r(v_r)$ verschiedene Wege der Länge r , die in v_r enden, und es gibt $w_{\ell-r}(v_r)$ verschiedene Wege der Länge $\ell - r$, die in v_r beginnen.
- ⇒ Es gibt also $w_r(v_r) \cdot w_{\ell-r}(v_r)$ verschiedene Wege der Länge ℓ , die an der Stelle r den Knoten v_r besuchen.
- ⇒ Die Summe ergibt genau die Anzahl aller Wege der Länge ℓ , da die entsprechenden Wege der einzelnen Summanden sich im Knoten an der Stelle r unterscheiden und somit nichts doppelt gezählt wird.



Dichte der Ordnung ℓ

- Maximal mögliche Anzahl von Wegen der Länge ℓ in einem Graphen mit n Knoten (also im vollständigen Graphen K_n):

$$n \cdot (n - 1)^\ell$$

- Dichte** der Ordnung ℓ : (für Graphen mit $n \geq 2$)

$$\rho_\ell(G) = \frac{w_\ell(G)}{n \cdot (n - 1)^\ell}$$

$\Rightarrow \rho_1(G) = \rho(G)$, denn in $w_1(G)$ zählt jede Kante doppelt:

$$\rho_1 = \frac{w_1}{n(n-1)^1} = \frac{2m}{n(n-1)} = \frac{m}{\binom{n}{2}} = \rho$$

Monotonität der Dichte

Satz

Für alle Graphen G und alle natürlichen Zahlen $\ell \geq 2$ gilt:

$$\rho_\ell(G) \leq \rho_{\ell-1}(G)$$

Beweis.

Da gilt $w_\ell = \sum_{v \in V} w_r(v) \cdot w_{\ell-r}(v)$, gilt insbesondere für $r = 1$:

$$\begin{aligned} w_\ell &= \sum_{v \in V} w_1(v) \cdot w_{\ell-1}(v) = \sum_{v \in V} \deg(v) \cdot w_{\ell-1}(v) \\ &\leq (n-1) \cdot \sum_{v \in V} w_{\ell-1}(v) = (n-1) \cdot w_{\ell-1} \end{aligned}$$

$$\rho_\ell = \frac{w_\ell}{n(n-1)^\ell} \leq \frac{(n-1) \cdot w_{\ell-1}}{n(n-1)^\ell} = \frac{w_{\ell-1}}{n(n-1)^{\ell-1}} = \rho_{\ell-1}$$



η -dichte Teilgraphen der Ordnung ℓ

Definition

In einem Graphen $G = (V, E)$ bezeichnet man den durch eine Knotenteilmenge $U \subseteq V$ induzierten Teilgraphen genau dann als **η -dichten Teilgraphen der Ordnung ℓ** , wenn gilt

$$\rho_\ell(G[U]) \geq \eta$$

- Jeder η -dichte Teilgraph der Ordnung ℓ ist auch ein η -dichter Teilgraph der Ordnung $\ell - 1$ (siehe Monotonitätssatz).
- Die η -dichten Teilgraphen der Ordnung $\ell \geq 2$ sind (wie die η -dichten Teilgraphen) nicht abgeschlossen unter Exklusion, aber geschachtelt.
- Für eine festgelegte Dichte η werden die η -dichten Graphen wachsender Ordnung einer Clique immer ähnlicher.

Dichte unendlicher Ordnung

Definition

Die *Dichte unendlicher Ordnung* ist

$$\rho_{\infty}(G) = \lim_{\ell \rightarrow \infty} \rho_{\ell}(G)$$

Satz

Sei $G = (V, E)$ ein ungerichteter Graph

- 1 $\rho_{\infty}(G)$ ist entweder Null oder Eins.
- 2 G ist genau dann eine Clique, wenn $\rho_{\infty} = 1$.

- Die einzigen Graphen, die für ein $\eta > 0$ und für jede Ordnung ℓ η -dicht der Ordnung ℓ sind, sind die Cliques.
- Die Ordnung erlaubt eine gewisse Skalierung, wie wichtig Kompaktheit im Vergleich zu Dichte ist.

Durchschnittsgrad

- Dichte und Durchschnittsgrad hängen direkt zusammen:

$$\bar{d}(G) = \rho(G) \cdot (n - 1)$$

- **k -dichter (Teil-)Graph bzgl. Durchschnittsgrad:**
(Teil-)Graph G mit $\bar{d}(G) \geq k$
- η -dichte Graphen bzgl. prozentualer Dichte der Größe ℓ sind $\eta(\ell - 1)$ -dichte Graphen bzgl. Durchschnittsgrad.
- k -dichte Graphen bzgl. Durchschnittsgrad der Größe ℓ sind $\frac{k}{\ell-1}$ -dichte Graphen bzgl. prozentualer Dichte.
- Jeder k -Core ist ein k -dichter (Teil-)Graph, aber nicht umgekehrt.
- k -dichte Teilgraphen sind nicht abgeschlossen unter Exklusion und auch nicht geschachtelt:
Löscht man aus einem k -regulären Graph einen beliebigen Knoten, fällt der Durchschnittsgrad unter k .

Verallgemeinerung des Satzes von Turán

Extremale Graphentheorie

Wieviele Kanten kann ein Graph maximal enthalten, ohne dass ein bestimmter Teilgraph enthalten ist?

bzw.: Welche Kantenzahl erzwingt einen bestimmten Teilgraph?

Satz (Dirac, 1963)

Sei $G = (V, E)$ ein ungerichteter Graph mit n Knoten und m Kanten.

Wenn $m > \frac{n^2}{2} \cdot \frac{k-2}{k-1}$, dann enthält G einen Teilgraphen der Größe $k+r$ mit Durchschnittsgrad

$$\bar{d} \geq k + r - 1 - \frac{r}{k+r}$$

für alle $r \in \{0, \dots, k-2\}$ und $k+r \leq n$.

(Der Fall $r = 0$ entspricht dem Satz von Turán.)

Der größtmögliche Durchschnittsgrad

$$\gamma^*(G) = \max_{U \subseteq V, U \neq \emptyset} \{\overline{\deg}(G[U])\}$$

Problem

Problem: **DensestSubgraph**

Eingabe: Graph G

Ausgabe: Teilgraph mit maximalem Durchschnittsgrad

Satz

Das Problem **DensestSubgraph** kann für Graphen mit n Knoten und m Kanten in Zeit $\mathcal{O}\left(mn(\log n) \left(\log \frac{n^2}{m}\right)\right)$ gelöst werden.

Der größtmögliche Durchschnittsgrad

Beweis:

- Formulierung von **DensestSubgraph** als MaximumFlow-Problem mit Parameter $\gamma \in \mathbb{Q}^+$
- Sei $G = (V, E)$ ein ungerichteter Graph mit n Knoten und m Kanten.
- Betrachte (gerichtetes) Flussnetzwerk bestehend aus Graph $G' = (V', E')$ und Kapazitätsfunktion $u_\gamma : E' \mapsto \mathbb{Q}^+$
- Füge zu V eine Quelle s und eine Senke t hinzu:

$$V' = V \cup \{s, t\}$$

$$\begin{aligned} E' = & \{(v, w), (w, v) : \{v, w\} \in E\} \cup \\ & \{(s, v) : v \in V\} \cup \\ & \{(v, t) : v \in V\} \end{aligned}$$

Der größtmögliche Durchschnittsgrad

Neue Kanten (in G'):

- für jede ungerichtete Kante in G zwei gerichtete Kanten der Kapazität 1,
- verbinde Quelle s mit allen Knoten in V durch Kante der Kapazität m ,
- verbinde alle Knoten in V mit Senke t durch Kante der Kapazität $m + \gamma - \deg_G(v)$.

$$u_\gamma(v, w) = \begin{cases} 1 & \text{falls } \{v, w\} \in E \\ m & \text{falls } v = s \\ m + \gamma - \deg_G(v) & \text{falls } w = t \\ 0 & \text{falls } (v, w) \notin E' \end{cases}$$

Der größtmögliche Durchschnittsgrad

- Betrachte Schnitt-Kapazitäten im Netzwerk
- Partitionierung der Knotenmenge V' in zwei disjunkte Mengen S und T mit $s \in S$ und $t \in T$.
- Sei $S_+ = S \setminus \{s\}$ und $T_+ = T \setminus \{t\}$, also $S_+ \cup T_+ = V$

Der größtmögliche Durchschnittsgrad

Falls $S_+ = \emptyset$, dann ist die Kapazität des Schnitts $c(S, \bar{S}) = m|V| = mn$,
ansonsten erhält man

$$\begin{aligned}
 c(S, T) &= \sum_{v \in S, w \in T} u_\gamma(v, w) \\
 &= \sum_{w \in T_+} u_\gamma(s, w) + \sum_{v \in S_+} u_\gamma(v, t) + \sum_{v \in S_+, w \in T_+} u_\gamma(v, w) \\
 &= m|T_+| + \left(m|S_+| + \gamma|S_+| - \sum_{v \in S_+} \deg_G(v) \right) + \sum_{\substack{v \in S_+, w \in T_+ \\ \{v, w\} \in E}} 1 \\
 &= m|V| + |S_+| \left(\gamma - \frac{1}{|S_+|} \left(\sum_{v \in S_+} \deg_G(v) - \sum_{\substack{v \in S_+, w \in T_+ \\ \{v, w\} \in E}} 1 \right) \right) \\
 &= m|V| + |S_+| (\gamma - \overline{\deg}(G[S_+]))
 \end{aligned}$$

Der größtmögliche Durchschnittsgrad

- γ ist der Test-Wert für den maximalen Durchschnittsgrad.
- Wie kann man nun feststellen, ob er zu klein oder zu groß ist?

Satz

Seien S und T so gewählt, dass sie einem Minimum- s, t -Schnitt für γ entsprechen. Dann gilt:

- 1 Wenn $S_+ \neq \emptyset$, dann $\gamma \leq \gamma^*(G)$.
- 2 Wenn $S_+ = \emptyset$, dann $\gamma \geq \gamma^*(G)$.

Der größtmögliche Durchschnittsgrad

Beweis.

① $S_+ \neq \emptyset$:

Da $c(\{s\}, V' \setminus \{s\}) = m|V| \geq c(S, T)$, gilt
 $|S_+| (\gamma - \overline{\deg}(G[S_+])) \leq 0$.

Also $\gamma \leq \overline{\deg}(G[S_+]) \leq \gamma^*(G)$.

② $S_+ = \emptyset$: Annahme: $\gamma < \gamma^*(G)$

Sei $U \subseteq V$ eine nichtleere Menge mit $\overline{\deg}(G[U]) = \gamma^*(G)$.

Mit der Gleichung für $c(S, T)$ erhält man

$$c(U \cup \{s\}, \bar{U} \cup \{t\}) = mn + |U|(\gamma - \gamma^*(G)) < mn = c(S, T)$$

(Widerspruch zur Minimalität der Schnittkapazität $c(S, T)$)

$\Rightarrow \gamma \geq \gamma^*(G)$



Der größtmögliche Durchschnittsgrad

- Algorithmus benutzt binäre Suche, um den richtigen Wert für γ zu finden
- $\gamma^*(G)$ kann nur endlich viele Werte annehmen:

$$\gamma^*(G) \in \left\{ \frac{2i}{j} : i \in \{0, \dots, m\} \text{ und } j \in \{1, \dots, n\} \right\}$$

- kleinste mögliche Distanz zwischen zwei Werten der Menge ist $\frac{2}{n(n-1)}$.

Der größtmögliche Durchschnittsgrad

Algorithmus 9: DensestSubgraph mit MinCut und binärer Suche

Input : Graph $G = (V, E)$

Output : Eine Menge von k Knoten von G

Initialisiere $l := 0$, $r := m$ und $U := \emptyset$;

while $r - l \geq \frac{1}{n(n-1)}$ **do**

$\gamma := \frac{l+r}{2}$;

 Konstruiere Fluss-Netzwerk (V', E', u_γ) ;

 Finde Minimum-Schnitt (S, T) des Fluss-Netzwerks;

if $S = \{s\}$ **then**

$r := \gamma$

else

$l := \gamma$;

$U := S - \{s\}$

return U

Der größtmögliche Durchschnittsgrad

- Iteration wird $\lceil \log((m+1)n(n-1)) \rceil = \mathcal{O}(\log n)$ -mal ausgeführt
 - In jeder Iteration MinCut-Berechnung mit Push-Relabel-Algorithmus (Goldberg/Tarjan) in $\mathcal{O}(mn \log \frac{n^2}{m})$ für ein Netzwerk mit n Knoten und m Kanten (Wir haben zwar $n+2$ Knoten und $2m+2n$ Kanten, das ändert asymptotisch aber nichts.)
- ⇒ Gesamtkomplexität: $\mathcal{O}\left(mn(\log n)(\log \frac{n^2}{m})\right)$
- mit parametrischen MaxFlow-Algorithmen: $\mathcal{O}\left(mn \log \frac{n^2}{m}\right)$

Durchschnittsgrad in gerichteten Graphen

- Wie kann Dichte im Sinne von Durchschnittsgrad auf **gerichtete** Graphen angewandt werden?
 - Durchschnittlicher Eingangs- und Ausgangsgrad sind gleich.
- ⇒ keine orientierungsabhängigen Maße
- **zwei** nichtleere (nicht unbedingt disjunkte) Knotenmengen $S, T \in V$
 - $E(S, T)$: Menge der Kanten von S nach T :

$$E(S, T) = \{(u, v) : u \in S \text{ und } v \in T\}$$

Durchschnittlicher gerichteter Grad des Paares (S, T)
(Kannan & Vinay, 1999):

$$\overline{\text{deg}}_G(S, T) = \frac{|E(S, T)|}{\sqrt{|S| \cdot |T|}}$$

Durchschnittlicher gerichteter Grad

- S : Menge der Hubs, T : Menge der Authorities
- Für $S = T$: konventioneller durchschnittlicher (Eingangs-/Ausgangs-)Grad:

$$\overline{\deg}_G(S, T) = \overline{\deg}_G(G[S])$$

- Durchschnittsgrad-Maximum eines gerichteten Graphen $G = (V, E)$:

$$\gamma^* = \max_{\substack{S, T \subseteq V \\ S \neq \emptyset, T \neq \emptyset}} \{\overline{\deg}_G(S, T)\}$$

- Kann mit Linear Programming in Polynomialzeit gelöst werden (Charikar, 2000).

Dense ℓ -Subgraph

- Graph mit Durchschnittsgrad $\overline{\deg}(G)$ muss nicht unbedingt einen echten Teilgraphen mit gleichem Durchschnittsgrad enthalten.
- Maximum des Durchschnittsgrads eines Teilgraphs auf ℓ Knoten:

$$\gamma^*(G, \ell) = \max \{ \overline{\deg}(G[U]) : U \subseteq V \text{ und } |U| = \ell \}$$

Problem

Problem: **Dense- ℓ -Subgraph**

Eingabe: Graph G , Parameter $\ell \in \mathbb{N}$

Frage: Eine Knotenmenge der Kardinalität ℓ mit maximalem induzierten Durchschnittsgrad

Dense ℓ -Subgraph

- Optimierungsproblem **Dense ℓ -Subgraph** ist \mathcal{NP} -hart, denn Instanz $(G, \ell, \ell - 1)$ des zugehörigen Entscheidungsproblems entspricht der Suche nach einer Clique der Größe ℓ in G .

⇒ Wie sieht es mit Approximation aus?

Greedy-Approximation für Dense ℓ -Subgraph

Algorithmus 10: Approximation eines ℓ -Subgraph mit hohem $\overline{\deg}$

Input : Graph $G = (V, E)$ und
gerader Parameter $\ell \in \mathbb{N}$ (mit $|V| \geq \ell$)

Output : Menge von ℓ Knoten von G

Sortiere die Knoten in absteigender Reihenfolge ihrer Grade;

Sei H die Menge von $\frac{\ell}{2}$ Knoten von höchstem Grad;

Berechne $N_H(v) = |N(v) \cap H|$ für alle Knoten $v \in V \setminus H$;

Sortiere die Knoten in $V \setminus H$ in absteigender Reihenfolge der N_H -Werte;

Sei R die Menge von $\frac{\ell}{2}$ Knoten von $V \setminus H$ mit den höchsten N_H -Werten;

return $H \cup R$

Greedy-Approximation für **Dense ℓ -Subgraph**

Satz

Sei G ein Graph auf n Knoten und sei $\ell \in \mathbb{N}$ eine gerade natürliche Zahl mit $\ell \leq n$.

Sei $A(G, \ell)$ der Durchschnittsgrad des induzierten Teilgraphen, der vom vorstehenden Algorithmus ausgegeben wird.

Dann gilt:

$$\gamma^*(G, \ell) \leq \frac{2n}{\ell} \cdot A(G, \ell)$$

bzw.

$$A(G, \ell) \geq \frac{\ell}{2n} \cdot \gamma^*(G, \ell)$$

Greedy-Approximation für **Dense ℓ -Subgraph**

Beweis.

- Für Knotenteilmengen $U, U' \subseteq V$ sei $E(U, U')$ die Menge der Kanten mit einem Endpunkt in U und einem Endpunkt in U' .
- Sei $m_U = |E(G[U])|$
- Sei \deg_H der Durchschnittsgrad der $\frac{\ell}{2}$ Knoten von G mit höchstem Grad bezüglich G . Es gilt: $\deg_H \geq \gamma^*(G, \ell)$.
- Man erhält für die Anzahl der Kanten zwischen H und dem Rest $V \setminus H$:

$$|E(H, V \setminus H)| = \deg_H \cdot |H| - 2m_H = \frac{\deg_H \cdot \ell}{2} - 2m_H \geq 0$$

Greedy-Approximation für **Dense ℓ -Subgraph**

Beweis.

- Weil der Algorithmus greedy (gierig) arbeitet, muss der Anteil der Kanten nach R (i.Vgl. zu $V \setminus H$) mindestens so groß sein, wie der Anteil der Knoten:

$$\frac{|E(H, R)|}{|E(H, V \setminus H)|} \geq \frac{|R|}{|V \setminus H|} = \frac{\ell/2}{n - \ell/2} = \frac{\ell}{2n - \ell} > \frac{\ell}{2n}$$

- Also ist die Gesamtzahl der Kanten in $G[H \cup R]$ mindestens

$$\left(\frac{\deg_H \cdot \ell}{2} - 2m_H \right) \cdot \frac{\ell}{2n} + m_H \geq \frac{\deg_H \cdot \ell^2}{4n}$$



Approximation von **Dense** ℓ -Subgraph

- Die Approximationsgüte wird umso besser, je größer ℓ im Vergleich zu n ist.

- Es gibt andere Approximationsverfahren mit Güte $\mathcal{O}(\frac{n}{\ell})$, z.B. durch rekursives Löschen von Knoten mit kleinstem Grad.

Parametrisierte Dichte

- Schwelle für Dichte (density threshold) $\gamma: \mathbb{N} \rightarrow \mathbb{Q}_+$
 - ▶ γ soll in Polynomialzeit berechenbar sein und
 - ▶ $\forall \ell \in \mathbb{N}: \gamma(\ell) \leq \ell - 1$ soll gelten
- Knotenteilmenge U heißt genau dann **γ -dicht**, wenn

$$\overline{\text{deg}}(G[U]) \geq \gamma(|U|)$$

Problem

Problem: **γ -Dense Subgraph**

Eingabe: Graph G , Parameter $\ell \in \mathbb{N}$

Frage: Existiert eine γ -dichte Knotenmenge bzgl.
Durchschnittsgrad der Kardinalität ℓ in G ?

Parametrisierte Dichte

- $\gamma(\ell) = \ell - 1$ entspricht **Clique**-Problem und ist \mathcal{NP} -vollständig
- $\gamma(\ell) = 0$ ist trivial, denn jede Menge U bestehend aus ℓ Knoten ist eine Lösung, die Antwort ist also immer "ja", wenn $\ell \leq |V|$
- Welche Funktionen $\gamma(\ell)$ erlauben noch Lösbarkeit in Polynomialzeit?

Satz

Sei γ eine Schwellwertfunktion für die Dichte.

- 1 Falls $\gamma = 2 + \mathcal{O}\left(\frac{1}{\ell}\right)$, dann ist γ -dense **Subgraph** in Polynomialzeit lösbar.
- 2 Falls $\gamma = 2 + \Omega\left(\frac{1}{\ell^{1-\varepsilon}}\right)$ für ein $\varepsilon > 0$, dann ist γ -dense **Subgraph** \mathcal{NP} -vollständig.

Parametrisierte Dichte

- Einen Teilgraph mit ℓ -Knoten und Durchschnittsgrad $\overline{\deg} \geq 2$ zu finden, geht also in Polynomialzeit.
 - Aber einen Teilgraph mit ℓ -Knoten und Durchschnittsgrad $\overline{\deg} \geq 2 + \varepsilon$ (für $\varepsilon > 0$) zu finden, geht nicht in Polynomialzeit falls $\mathcal{P} \neq \mathcal{NP}$.
 - k -Cores ließen sich in Linearzeit berechnen (sogar für alle k gleichzeitig)
- ⇒ riesiger **Komplexitätsunterschied** zwischen statistischer und struktureller Dichte!

Übersicht

- 1 Grundlagen
- 2 Zentralitätsindizes
- 3 Wiederholung: Kürzeste Wege
- 4 Algorithmen für Zentralitätsindizes
- 5 Lokale Dichte
- 6 Zusammenhang**
 - Definitionen
 - Fundamentale Sätze
 - Schnitte minimaler Kantenkapazität
 - Der Stoer/Wagner-Algorithmus für globale MinCuts

Zusammenhang in Graphen / Netzwerken

- beschäftigt sich mit der Stärke der Verbindung zwischen zwei Knoten in Bezug auf die Anzahl knoten- bzw. kantendisjunkter Wege
 - “Eine Kette ist nur so stark wie ihr schwächstes Glied.”
- ⇒ Wir suchen nach den schwächsten Elementen, die beim Entfernen die Verbindung zerstören.

Definition

Ein ungerichteter Graph heißt **zusammenhängend**, wenn es von jedem Knoten einen Pfad zu jedem anderen Knoten gibt.

Ein maximaler zusammenhängender induzierter Teilgraph wird als **Zusammenhangskomponente** bezeichnet.

Knoten-Zusammenhang

Definition

Ein ungerichteter Graph $G = (V, E)$ heißt **k -knotenzusammenhängend**, falls $|V| > k$ und für jede echte Knotenteilmenge $X \subset V$ mit $|X| < k$ der Graph $G - X$ zusammenhängend ist.

Der **Knotenzusammenhang** $\kappa(G)$ des Graphen G ist die größte natürliche Zahl k , für die G k -knotenzusammenhängend ist.

Bemerkungen:

- Jeder nicht-leere Graph ist 0-knotenzusammenhängend, da es keine Teilmenge X mit $|X| < 0$ gibt.
- Obwohl es wünschenswert wäre, dass die Bezeichnung “1-knotenzusammenhängend” gleichzusetzen ist mit der Bezeichnung “zusammenhängend”, wird üblicherweise der Graph bestehend aus nur einem einzelnen Knoten zwar als zusammenhängend, aber nicht 1-zusammenhängend bezeichnet.

Kanten-Zusammenhang und k -Komponenten

Definition

Ein ungerichteter Graph $G = (V, E)$ heißt **k -kantenzusammenhängend**, falls $|V| \geq 2$ und für jede Kanteilmenge $Y \subseteq E$ mit $|Y| < k$ der Graph $G - Y$ zusammenhängend ist.

Der **Kantenzusammenhang** $\lambda(G)$ des Graphen G ist die größte natürliche Zahl k , für die G k -kantenzusammenhängend ist.

Der Kantenzusammenhang eines unzusammenhängenden Graphen sowie des Graphen bestehend aus einem einzelnen Knoten ist 0.

Definition

Die maximalen k -fach knoten-/kanten-zusammenhängenden Teilgraphen werden als **k -Knoten-/Kanten-Zusammenhangskomponenten** bezeichnet.

Zusammenhang in gerichteten Graphen

Definition

Ein gerichteter Graph ist **stark zusammenhängend**, wenn es für jeden Knoten einen gerichteten Pfad zu jedem anderen Knoten gibt.

Ein maximaler stark zusammenhängender induzierter Teilgraph wird als **starke Zusammenhangskomponente** bezeichnet.

Knoten- und Kantenzusammenhang können auf gerichtete Graphen übertragen werden, indem man in der jeweiligen Definition fordert, dass $G - X$ bzw. $G - Y$ *stark zusammenhängend* ist.

Separatoren

Definition

Sei $G = (V, E)$ ein ungerichteter Graph.

Eine Knotenteilmenge $C \subset V$ heißt **Knoten-Separator**, wenn die Anzahl der Zusammenhangskomponenten in $G - C$ größer als in G ist.

Falls zwei Knoten s und t zwar in G in der gleichen Zusammenhangskomponente sind, aber nicht in $G - C$, dann bezeichnet man C als **s - t -Knoten-Separator**.

Kanten-Separatoren und **s - t -Kanten-Separatoren** sind analog definiert.

s - t -Separatoren können auch auf gerichtete Graphen übertragen werden: eine Knoten- bzw. Kantenmenge ist dann ein s - t -Separator, wenn es keinen gerichteten Pfad mehr von s nach t gibt, nachdem die Menge aus dem Graph entfernt wurde.

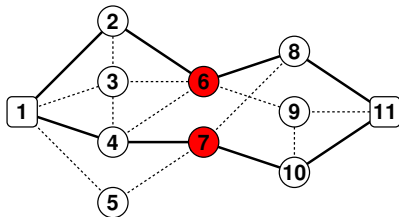
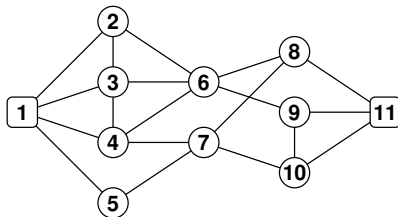
Disjunkte Pfade

Definition

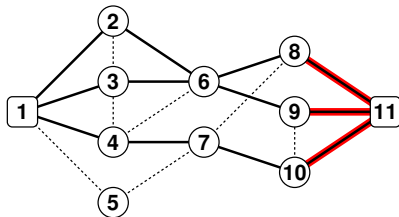
Zwei (gerichtete oder ungerichtete) Pfade von s nach t werden als **knotendisjunkte s - t -Pfade** bezeichnet, wenn sie keinen Knoten außer s und t gemeinsam haben.

Zwei Pfade werden als **kantendisjunkte Pfade** bezeichnet, wenn sie keine Kante gemeinsam haben.

Disjunkte s - t -Pfade



2 knotendisjunkte 1-11-Pfade



3 kantendisjunkte 1-11-Pfade

Lokaler Zusammenhang

Definition

Für zwei Knoten s und t eines Graphen G ist der **lokale (Knoten-)Zusammenhang** $\kappa_G(s, t)$ definiert als die minimale Anzahl von Knoten, die entfernt werden müssen, damit es keinen Weg mehr von s nach t gibt.

Für den Fall, dass zwischen s und t eine Kante existiert, können sie nicht durch das Löschen von Knoten separiert werden. Deshalb wird der lokale (Knoten-)Zusammenhang in diesem Fall $\kappa_G(s, t) = n - 1$ definiert. (Anderenfalls wäre höchstens $\kappa_G(s, t) = n - 2$ möglich.)

Der **lokale Kanten-Zusammenhang** zweier Knoten s und t ist entsprechend definiert als die minimale Anzahl von Kanten, die entfernt werden müssen, damit es keinen Weg mehr von s nach t gibt.

Lokaler Zusammenhang

Hinweis:

Für ungerichtete Graphen gilt $\kappa_G(s, t) = \kappa_G(t, s)$ und $\lambda_G(s, t) = \lambda_G(t, s)$, was für gerichtete Graphen im Allgemeinen nicht gilt.

Wiederholung: Zweifachzusammenhang

Definition

Ein **Artikulationsknoten** ist ein Knoten, der beim Entfernen aus dem Graphen die Anzahl der Zusammenhangskomponenten erhöht.

Eine **Brücke** ist eine Kante, die beim Entfernen aus dem Graphen die Anzahl der Zusammenhangskomponenten erhöht.

Eine **Zweifachzusammenhangskomponente** ist ein maximaler 2-fach (knoten-)zusammenhängender Teilgraph.

Ein **Block** ist ein maximaler zusammenhängender Teilgraph, der keinen Artikulationsknoten enthält, d.h. die Menge aller Blocks eines Graphen besteht aus den isolierten Knoten, den Brücken, sowie den Zweifachzusammenhangskomponenten.

Block-Graph und CutPoint-Graph

Definition

Der **Block-Graph** $B(G)$ eines Graphen G hat jeweils einen Knoten für jeden Block von G (außer für isolierte Knoten), wobei zwei Knoten des Block-Graphen adjazent sind, wenn die entsprechenden Blöcke in G einen (Artikulations-)Knoten gemeinsam haben.

Der **CutPoint-Graph** $C(G)$ eines Graphen G hat jeweils einen Knoten für jeden Artikulationsknoten von G , wobei zwei Knoten des CutPoint-Graphen adjazent sind, wenn die entsprechenden Artikulationsknoten in G zum gleichen Block gehören.

Satz (Harary)

Für jeden Graphen gilt:

$$B(B(G)) = C(G) \quad \text{und} \quad B(C(G)) = C(B(G))$$

Block-CutPoint-Graph

Definition

Der **Block-CutPoint-Graph** eines Graphen G ist der bipartite Graph, dessen Knotenmenge aus je einem Knoten für jeden Artikulationsknoten von G und je einem Knoten für jeden Block von G besteht, wobei ein CutVertex-Knoten mit einem Block-Knoten genau dann durch eine Kante verbunden ist, wenn der Artikulationsknoten zu dem entsprechenden Block gehört.

Satz (Harary & Prins)

*Der Block-CutPoint-Graph eines zusammenhängenden Graphen ist ein **Baum**.*

Fundamentale Ungleichung

Satz

Für jeden Graphen G gilt:

$$\kappa(G) \leq \lambda(G) \leq \delta(G)$$

Beweis.

- Spezialfall: Für unzusammenhängende Graphen sowie den Graph mit nur einem Knoten gilt $\kappa(G) = \lambda(G) = 0 \leq \delta(G)$.
- Ansonsten: Die inzidenten Kanten eines Knotens v mit $\deg(v) = \delta(G)$ bilden einen Kanten-Separator.

$$\Rightarrow \lambda(G) \leq \delta(G)$$

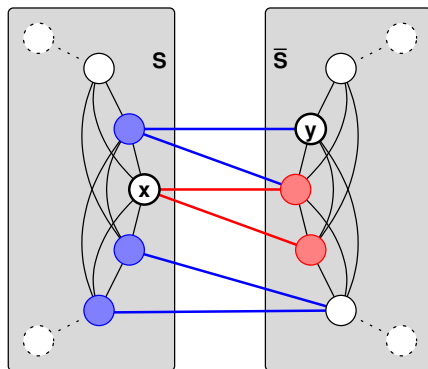
Fundamentale Ungleichung

Beweis.

- Sei $G = (V, E)$ ein Graph mit mindestens zwei Knoten. Betrachte Kanten-Separator minimaler Kardinalität. Dieser partitioniert die Knotenmenge in S und $\bar{S} = V \setminus S$.
- Falls alle Kanten zwischen S und \bar{S} vorhanden sind, gilt $\lambda(G) = |S| \cdot |\bar{S}| \geq n - 1 \geq \kappa(G)$ ($|S| \cdot |\bar{S}|$ ist am kleinsten, wenn nur 1 Knoten von den restlichen $n - 1$ abgetrennt wird)
- Anderenfalls existieren Knoten $x \in S$ und $y \in \bar{S}$ mit $\{x, y\} \notin E$, wobei die Nachbarn von x in \bar{S} zusammen mit allen Knoten aus $S \setminus \{x\}$, die Nachbarn in \bar{S} haben, einen Knoten-Separator bilden. Dieser Knoten-Separator ist höchstens so groß wie die Anzahl der Kanten von S nach \bar{S} , und er separiert mindestens x und y .



Fundamentale Ungleichung



- rote und blaue Kanten bilden zusammen einen Kantenseparator
- rote und blaue Knoten bilden einen Knotenseparator für x und y

Fundamentale Ungleichung

Der Knotenseparator hat höchstens die gleiche Kardinalität wie der Kantenseparator, denn

- zu jedem roten / blauen Knoten gibt es mindestens eine rote / blaue Kante und
- rote und blaue Kanten sind disjunkt (zu den roten ist x inzident und zu den blauen nicht)

Warum kann man nicht einfach alle Knoten in S nehmen, die einen Nachbarn in \bar{S} haben (also x anstatt der roten Knoten)?

- weil dann evt. keine Knoten von S übrigbleiben
- ⇒ dann gäbe es keine zwei Teile, sondern nur einen

Das n -Chain / n -Arc Theorem

Satz (Menger, 1927)

Seien P und Q Teilmengen der Knoten eines ungerichteten Graphen.

Dann sind folgende Größen gleich

- die **maximale Anzahl knotendisjunkter Pfade**, die Knoten von P mit Knoten von Q verbinden, und
- die **minimale Kardinalität einer Knotenmenge**, die alle Pfade zwischen Knoten in P und Knoten in Q überschneidet bzw. unterbricht.

Der Satz von Menger

Satz ("Satz von Menger")

Seien s und t zwei Knoten eines ungerichteten Graphen.

Wenn s und t nicht adjazent sind, dann ist die **maximale Anzahl knotendisjunkter s - t -Pfade** gleich der **minimalen Kardinalität eines s - t -Knoten-Separators**.

Satz (Kantenversion)

Die **maximale Anzahl kantendisjunkter s - t -Pfade** ist gleich der **minimalen Kardinalität eines s - t -Kanten-Separators**.

(Ford/Fulkerson, Dantzig/Fulkerson, Elias/Feinstein/Shannon, 1956)

Eng verwandt: MaxFlow-MinCut-Theorem

(Kantenversion von Menger's Theorem kann als Spezialfall gesehen werden, wo alle Kantengewichte den gleichen Wert haben.)

Whitney's Theorem

Satz (Whitney, 1932)

*Sei G ein nicht-trivialer Graph und k eine natürliche Zahl.
 G ist genau dann k -(knoten-)zusammenhängend, wenn alle Paare verschiedener Knoten (s, t) durch k knotendisjunkte s - t -Pfade verbunden werden können.*

Schwierig bei der Herleitung ist nur, dass der Satz von Menger fordert, dass die Knoten nicht adjazent sind.

Da diese Bedingung bei der Kantenversion nicht auftritt, folgt aus dieser sofort:

Satz

*Sei G ein nicht-trivialer Graph und k eine natürliche Zahl.
 G ist genau dann k -kanten-zusammenhängend, wenn alle Paare verschiedener Knoten (s, t) durch k kantendisjunkte s - t -Pfade verbunden werden können.*

Gemischter Knoten-/Kanten-Zusammenhang

Definition

Ein Paar (k, ℓ) heißt **Zusammenhangspaar** zweier Knoten s und t eines Graphen, falls eine Menge aus **k Knoten** und **ℓ Kanten** existiert, die jeden Weg zwischen s und t beim Entfernen zerstört, aber es keine solche Menge bestehend aus $k - 1$ Knoten und ℓ Kanten oder k Knoten und $\ell - 1$ Kanten gibt.

Satz (Beineke & Harary, 1967)

Wenn (k, ℓ) ein Zusammenhangspaar zweier Knoten s und t in einem Graphen ist, dann gibt es $k + \ell$ kantendisjunkte Pfade von s nach t , von denen k knotendisjunkte s - t -Pfade sind.

Einfache Schranken

Satz

Der (Knoten-/Kanten-)Zusammenhang in einem Graphen mit n Knoten und m Kanten ist **höchstens**

$$\begin{aligned} \lfloor \frac{2m}{n} \rfloor &: \text{ falls } m \geq n - 1 \\ 0 &: \text{ sonst} \end{aligned}$$

Der (Knoten-/Kanten-)Zusammenhang in einem Graphen mit n Knoten und m Kanten ist **mindestens**

$$\begin{aligned} m - \binom{n-1}{2} &: \text{ falls } \binom{n-1}{2} < m \leq \binom{n}{2} \\ 0 &: \text{ sonst} \end{aligned}$$

Für jeden Graphen G mit $\delta(G) \geq \lfloor \frac{n}{2} \rfloor$ gilt: $\lambda(G) = \delta(G)$.

Überlappung von k -Knoten-Komponenten

Die offensichtliche Tatsache, dass

- zwei verschiedene Zusammenhangskomponenten keinen Knoten gemeinsam haben können und
- zwei verschiedene Blöcke höchstens einen Knoten gemeinsamen haben können,

lässt sich wie folgt verallgemeinern:

Satz

Zwei verschiedene k -(Knoten-)Komponenten haben höchstens $k - 1$ Knoten gemeinsam.

Nicht-Überlappung von k -Kanten-Komponenten

Satz (Matula, 1968)

Für jede natürliche Zahl k sind die k -Kanten-Komponenten eines Graphen knotendisjunkt.

Beweis.

Übungsaufgabe ... □

Satz von Mader

Obwohl aus der fundamentalen Ungleichung $\kappa(G) \leq \lambda(G) \leq \delta(G)$ folgt, dass k -Knoten-/Kanten-Zusammenhang einen Minimalgrad $\geq k$ impliziert, ist das Gegenteil nicht unbedingt der Fall.

Ein hoher Durchschnittsgrad impliziert aber die Existenz eines relativ gut zusammenhängenden Teilgraphen:

Satz (Mader, 1972)

Jeder Graph mit Durchschnittsgrad mindestens $4k$ enthält einen k -zusammenhängenden Teilgraph.

Kanten-Schnitte

- ungerichteter gewichteter Graph $G = (V, E)$
- Def. für zwei disjunkte Knotenteilmengen $X, Y \subseteq V$, $X \cap Y = \emptyset$
 $w(X, Y)$: Gewichtssumme der Kanten von X nach Y
- für gerichtete Graphen analog, allerdings Unterscheidung wichtig:
Startknoten in X und Zielknoten in Y
- Wir wollen in der Regel Kanten-Separatoren betrachten, die minimal sind, d.h., in denen nicht mehr Kanten enthalten sind als notwendig, um zwei Knotenmengen X und Y zu trennen.
- $\bar{S} = V \setminus S$
- **Schnitt** (engl. **cut**) S bzw. (S, \bar{S}) : Knotenmenge mit $\emptyset \subset S \subset V$,
Gewicht / Kapazität: $w(S, \bar{S}) = w(S, V \setminus S)$
- ungewichtete Graphen: Gewicht des Schnitts ist Anzahl der Kanten von S nach $V \setminus S$.

Kanten-Schnitte minimalen Gewichts (Minimum Cuts)

Definition

Ein (globaler) **Minimum Cut** ist ein Schnitt mit minimalem Gewicht.

Ein Schnitt S bzw. (S, \bar{S}) ist also genau dann ein Minimum Cut, wenn für jeden Schnitt T bzw. (T, \bar{T}) gilt

$$w(S, V \setminus S) \leq w(T, V \setminus T)$$

Fakt

In einem zusammenhängenden Graphen mit echt positiven Kantengewichten zerlegt jeder Minimum Cut (S, \bar{S}) den Graph in zwei jeweils zusammenhängende Teilgraphen $G[S]$ und $G[\bar{S}]$.

Satz

In einem Graphen mit n Knoten kann es höchstens $\binom{n}{2}$ verschiedene Minimum Cuts geben.

Minimum Cuts

Lemma

Sei $(S, V \setminus S)$ ein Minimum Cut in $G = (V, E)$.
Dann gilt für jede nicht-leere Teilmenge $T \subset S$:

$$w(T, S \setminus T) \geq \frac{\lambda}{2}$$

Beweis.

- Annahme: $w(T, S \setminus T) < \frac{\lambda}{2}$
 - $w(T, V \setminus S) + w(S \setminus T, V \setminus S) = \lambda$
 - o.B.d.A.: $w(T, V \setminus S) \leq \frac{\lambda}{2}$
(sonst vertausche T und $S \setminus T$)
- $\Rightarrow w(T, V \setminus T) = w(T, S \setminus T) + w(T, V \setminus S) < \lambda$ (Widerspruch)



Minimum Cuts

Notation:

- $\bar{X} = V \setminus X$
- Im Folgenden werden wir einen Schnitt (X, \bar{X}) oft einfach nur mit X bezeichnen.

Lemma

Seien (A, \bar{A}) und (B, \bar{B}) mit $A \neq B$ zwei Minimum Cuts in $G = (V, E)$, so dass $T = A \cup B$ auch ein Minimum Cut in G ist.

Dann gilt:

$$w(A, \bar{T}) = w(B, \bar{T}) = w(A \setminus B, B) = w(A, B \setminus A) = \frac{\lambda}{2}$$

Minimum Cuts

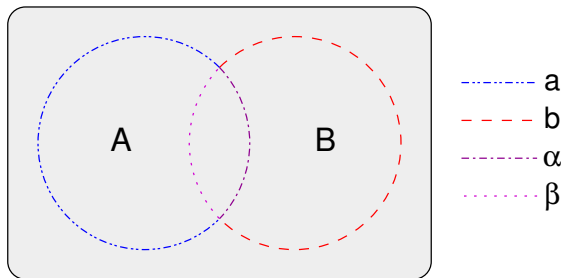


Abbildung: Schnitt zweier Minimum Cuts A und B

Minimum Cuts

Beweis.

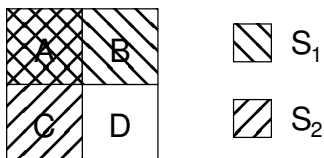
- Sei $a = w(A, \bar{T})$,
 $b = w(B, \bar{T})$,
 $\alpha = w(A, B \setminus A)$ und
 $\beta = w(B, A \setminus B)$.

$$\Rightarrow w(A, \bar{A}) = a + \alpha = \lambda,$$
$$w(B, \bar{B}) = b + \beta = \lambda$$
$$w(T, \bar{T}) = a + b = \lambda$$

- Es gilt auch:
 $w(A \setminus B, B \cup \bar{T}) = a + \beta \geq \lambda$ und $w(B \setminus A, A \cup \bar{T}) = b + \alpha \geq \lambda$.
- Dieses (Un-)Gleichungssystem hat nur eine Lösung:
 $a = \alpha = b = \beta = \frac{\lambda}{2}$.



Minimum Cuts

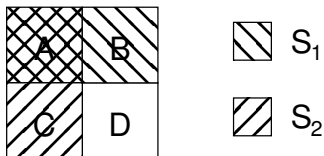


Definition

Ein Paar $\langle S_1, S_2 \rangle$ heißt **Crossing Cut**, falls S_1, S_2 Minimum Cuts sind und keine der folgenden Mengen leer ist:

- $A = S_1 \cap S_2$,
- $B = S_1 \setminus S_2$,
- $C = S_2 \setminus S_1$
- $D = \bar{S}_1 \cap \bar{S}_2$

Crossing Cuts



Lemma

Seien $\langle S_1, S_2 \rangle$ Crossing Cuts und seien Mengen wie folgt definiert:
 $A = S_1 \cap S_2$, $B = S_1 \setminus S_2$, $C = S_2 \setminus S_1$ and $D = \bar{S}_1 \cap \bar{S}_2$. Dann gilt:

- ① A, B, C und D sind Minimum Cuts.
- ② $w(A, D) = w(B, C) = 0$
- ③ $w(A, B) = w(B, D) = w(D, C) = w(C, A) = \frac{\lambda}{2}$

Crossing Cuts

Beweis.

- Da S_1 und S_2 Minimum Cuts sind, gilt:

$$\triangleright w(S_1, \bar{S}_1) = w(A, C) + w(A, D) + w(B, C) + w(B, D) = \lambda$$

$$\triangleright w(S_2, \bar{S}_2) = w(A, B) + w(A, D) + w(B, C) + w(C, D) = \lambda$$

$$\Rightarrow w(A, B) + w(A, C) + 2w(A, D) + 2w(B, C) + w(B, D) + w(C, D) = 2\lambda$$

- Da es keinen Schnitt mit Kapazität $< \lambda$ gibt, gilt:

$$w(A, \bar{A}) = w(A, B) + w(A, C) + w(A, D) \geq \lambda$$

$$w(B, \bar{B}) = w(A, B) + w(B, C) + w(B, D) \geq \lambda$$

$$w(C, \bar{C}) = w(A, C) + w(B, C) + w(C, D) \geq \lambda$$

$$w(D, \bar{D}) = w(A, D) + w(B, D) + w(C, D) \geq \lambda$$

$$\Rightarrow 2[w(A, B) + w(A, C) + w(A, D) + w(B, C) + w(B, D) + w(C, D)] \geq 4\lambda$$

$$\Rightarrow w(A, D) = w(B, C) = 0 \text{ (keine Diagonalkanten)}$$

Crossing Cuts

Beweis.

- Die waagerechte Linie in der Abbildung entspricht dem Minimum Cut $(S_1, \bar{S}_1) = (A \cup B, C \cup D) = \lambda$, die senkrechte entspricht $(S_2, \bar{S}_2) = (A \cup C, B \cup D) = \lambda$.
 - Analogie: Länge der Kanten entspricht Kapazität der geschnittenen Kanten
 - Annahme: die waagerechte und senkrechte Linie schneiden sich nicht genau in der Mitte
- ⇒ Dann definiert eine der Teilmengen $X = A, B, C$ oder D einen Schnitt $w(X, \bar{X}) < \lambda$ (Widerspruch)
- ⇒ $w(A, B) = w(B, D) = w(D, C) = w(C, A) = \frac{\lambda}{2}$ und
 $w(A, \bar{A}) = w(B, \bar{B}) = w(C, \bar{C}) = w(D, \bar{D}) = \lambda$



Zirkuläre Partitionen

Crossing Cuts in $G = (V, E)$ partitionieren V in 4 Teile, die kreisförmig durch 4 Kantenmengen vom jeweiligen Gewicht $\lambda/2$ verbunden sind.

Allgemeiner:

Definition

Eine **zirkuläre Partition** ist eine Aufteilung von V in $k \geq 3$ disjunkte Mengen V_1, V_2, \dots, V_k , so dass

- $w(V_i, V_j) = \begin{cases} \lambda/2 & \text{falls } |i - j| \equiv 1 \pmod{k} \\ 0 & \text{sonst} \end{cases}$
- und falls S ein Minimum Cut ist, dann
 - ▶ ist S oder \bar{S} eine echte Teilmenge einer Menge V_i oder
 - ▶ die zirkuläre Partition ist eine Verfeinerung der Partition, die durch den Minimum Cut S definiert wird.
(S ist die Vereinigung einiger Mengen der zirkulären Partition.)

Circular Partition Cuts

- Seien V_1, V_2, \dots, V_k die disjunkten Mengen einer zirkulären Partition.

⇒ Für alle a, b mit $1 \leq a \leq b < k$ ist die Menge

$$S = \bigcup_{i=a}^b V_i$$

ein Minimum Cut (zusammen mit $\bar{S} = V \setminus S$),
genannt **Circular Partition Cut**.

- Insbesondere ist jedes einzelne V_i ($1 \leq i \leq k$) ein Minimum Cut (erster Punkt der Definition zirkulärer Partitionen).

Zirkuläre Partitionen

- Betrachte Minimum Cut S , so dass weder S noch \bar{S} in einer Menge der zirkulären Partition enthalten ist.

Aus dem zweiten Teil der Definition zirkulärer Partitionen folgt

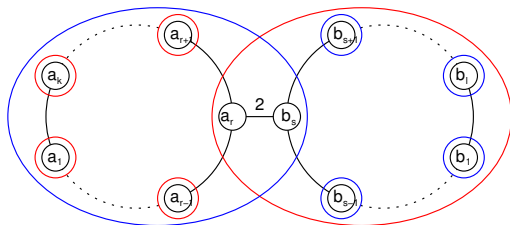
- Da S und \bar{S} zusammenhängend sein müssen, ist S oder \bar{S} gleich $\bigcup_{i=a}^b V_i$ für ein Paar a, b mit $1 \leq a < b < k$.
(Das andere ist dann die Vereinigung der restlichen Mengen V_i .)
- Für keine Menge V_i einer zirkulären Partition V_1, \dots, V_k kann es einen Minimum Cut S geben, so dass $\langle V_i, S \rangle$ ein Crossing Cut ist.

Kompatibilität zirkulärer Partitionen

Definition

Zwei verschiedene zirkuläre Partitionen $P = \{U_1, \dots, U_k\}$ und $Q = \{V_1, \dots, V_\ell\}$ sind **kompatibel**, wenn es eindeutige Indizes r und s gibt, so dass $\forall i \neq r : U_i \subseteq V_s$ und $\forall j \neq s : V_j \subseteq U_r$.

Beispiel:



$$P = \{\{a_1\}, \dots, \{a_{r-1}\}, \{a_r, b_1, \dots, b_l\}, \{a_{r+1}\}, \dots, \{a_k\}\}$$

$$Q = \{\{b_1\}, \dots, \{b_{s-1}\}, \{b_s, a_1, \dots, a_k\}, \{b_{s+1}\}, \dots, \{b_l\}\}$$

Kompatibilität zirkulärer Partitionen

Lemma

Alle verschiedenen zirkulären Partitionen sind paarweise kompatibel.

Beweis

- Betrachte zwei zirkuläre Partitionen P und Q in $G = (V, E)$.
- Alle Mengen der Partitionen sind Minimum Cuts.
- Behauptung: Jede Menge von P oder ihr Komplement ist in einer Menge von Q enthalten.
- Annahme: eine Menge $S \in P$ ist die Vereinigung von mindestens zwei, aber nicht von allen Mengen von Q .
- Genau zwei Mengen $A, B \in Q$, die in S enthalten sind, sind durch mindestens eine Kante zu den Knoten in $V \setminus S$ verbunden.
- Außerdem müssen die Mengen aus Q , die zusammen S bilden, in Q konsekutiv sein, sonst hätte der Cut zum Rest eine Kapazität $> \lambda$.

Kompatibilität zirkulärer Partitionen

Beweis

- Sei T die Menge, die man durch Ersetzen von $A \subset S$ durch das Element von Q erhält, das zu B verbunden, aber nicht in S enthalten ist.
 - T ist ein Minimum Cut (die Kanten zwischen A und $S \setminus A$ haben Kapazität $\lambda/2$, gleiches gilt für die Anbindung der neuen Menge aus Q an den Rest von Q in der anderen Richtung)
- ⇒ Dann ist $\langle S, T \rangle$ ein Crossing Cut.
- ⇒ P und Q erfüllen nicht mehr die zweite Bedingung in der Definition zirkulärer Partionen.
(Widerspruch)
- ⇒ Jede Menge von P oder ihr Komplement ist in einer Menge von Q enthalten.

Kompatibilität zirkulärer Partitionen

Beweis

- Annahme: zwei Mengen von P sind in zwei verschiedenen Mengen von Q enthalten.
 - Komplement jeder übrigen Menge von P enthält beide Mengen und kann deshalb nicht in einer einzelnen Menge von Q enthalten sein.
- ⇒ Jede übrige Menge von P ist Teilmenge einer Menge von Q .
- Gleiches gilt umgekehrt für die übrigen Mengen von Q .
- ⇒ $P = Q$ (Widerspruch)

Kompatibilität zirkulärer Partitionen

Beweis.

- Annahme: alle Mengen von P sind in einer Menge Y von Q
- $\Rightarrow Y = V$ (Widerspruch)
- \Rightarrow Es gibt mindestens eine Menge X in P , deren Komplement \bar{X} eine Teilmenge einer Menge Y aus Q ist.
- Da die Vereinigung von zwei Komplementen von Mengen aus P gleich V ist und Q mindestens drei Mengen enthält, kann nur genau ein Komplement einer Menge aus P in einer Menge von Q enthalten sein.
- Alle anderen Mengen aus P sind Teilmengen von \bar{X} und damit auch in Menge Y aus Q enthalten.
- Y ist nicht Teilmenge einer Menge aus P .
- $\Rightarrow \bar{Y} \subset X$ und alle anderen Mengen aus Q sind auch Teilmengen von X .

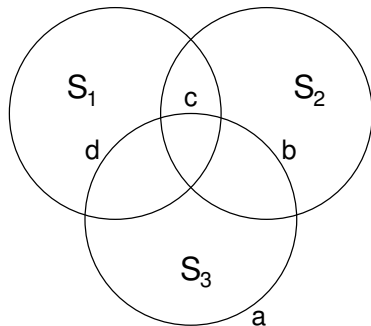


Paarweise Disjunktheit von Crossing Cuts

Lemma

Wenn S_1 , S_2 und S_3 paarweise Crossing Cuts sind, dann gilt:

$$S_1 \cap S_2 \cap S_3 = \emptyset$$



Paarweise Disjunktheit von Crossing Cuts

Beweis.

- Annahme: Lemma falsch, also Schnittmenge nicht leer

- Definiere

- ▶ $b = w((S_2 \cap S_3) \setminus S_1, S_2 \setminus (S_1 \cup S_3))$

- ▶ $c = w(S_1 \cap S_2 \cap S_3, (S_1 \cap S_2) \setminus S_3)$

- ▶ $d = w((S_1 \cap S_3) \setminus S_2, S_1 \setminus (S_2 \cup S_3))$

- Einerseits ist $S_1 \cap S_2$ ein Minimum Cut. $\Rightarrow c \geq \frac{\lambda}{2}$

- Andererseits ist $c + b = c + d = \frac{\lambda}{2}$.

$$\Rightarrow b = d = 0 \text{ und } (S_1 \cap S_3) \setminus S_2 = (S_2 \cap S_3) \setminus S_1 = \emptyset$$

- Da es keine diagonalen Kanten im Crossing Cut $\langle S_1, S_2 \rangle$ gibt (siehe früheres Lemma), sind $S_1 \cap S_2 \cap S_3$ und $S_3 \setminus (S_1 \cup S_2)$ nicht verbunden. (Widerspruch weil aufgrund der Crossing Cuts $\langle S_1, S_3 \rangle$ und $\langle S_2, S_3 \rangle$ der Schnittwert $\lambda/2$ sein müsste)



Weitere Sätze

Satz (R. Bixby, 1975)

In einem Graphen $G = (V, E)$ gibt es für jede einem Crossing Cut entsprechende Partition P von V in 4 disjunkte Mengen eine zirkuläre Partition in G , die eine Verfeinerung von P ist.

Satz (R. Bixby, 1975)

Ein Graph $G = (V, E)$ hat höchstens $\binom{|V|}{2}$ viele Minimum Cuts.

Diese Schranke ist scharf. (Sie wird vom einfachen Kreis auf n Knoten erreicht.)

Laminare Mengen

Definition

Eine Menge \mathcal{S} von Mengen heißt **laminar**, wenn für jedes Paar von Mengen $S_1, S_2 \in \mathcal{S}$ gilt, dass entweder S_1 und S_2 disjunkt sind, oder eine der beiden Menge die andere enthält.

- Jede laminare Menge \mathcal{S} , die nicht die leere Menge enthält, kann als **Wald** bzw. **Baum** repräsentiert werden.
- Jede Menge in \mathcal{S} wird durch einen eigenen Knoten repräsentiert.
- Blätter repräsentieren Mengen, die keine anderen Mengen enthalten.
- Der Vater eines zur Menge T gehörigen Knotens repräsentiert die (eindeutige) kleinste Übermenge von T in \mathcal{S} .
- Man erhält eine Menge von Bäumen, deren Wurzelknoten Mengen repräsentieren, die in keiner anderen Menge von \mathcal{S} enthalten sind.

Anzahl Knoten im Baum laminarer Mengen

- Hinzufügen eines künstlichen Wurzelknotens, der mit allen eigentlichen Wurzeln verbunden ist, liefert einen Baum.
- ⇒ Die Knoten eines Baums repräsentieren alle Mengen von \mathcal{S} , wobei die Wurzel die Grundmenge (Vereinigung aller Mengen) repräsentiert.
- Für eine gegebene **Kardinalität n der Grundmenge** erhält man eine größtmögliche Anzahl von Knoten, wenn jedes Element der Grundmenge irgendwann einzeln abgespalten wird.
- In diesem Fall hat jeder Vaterknoten mindestens 2 Kinder und es gibt genau n Blätter (jedes einzelne Element der Gesamtmenge als jeweils einelementige Menge).
- Außer den Blättern kann es dann höchstens $n - 1$ weitere Knoten geben, da jeder solche Knoten mindestens zwei Kinder vereinigt.
- Insgesamt hat jeder Baum also **höchstens $2n - 1$ Knoten**.

Zirkuläre Partitionen als laminare Mengen

- Da alle zirkulären Partitionen miteinander kompatibel sind, bilden die entsprechenden Mengen eine laminare Familie.
- Zusammen mit der Definition zirkulärer Partitionen folgt, dass es in jedem Graph mit n Knoten **höchstens $n - 2$ zirkuläre Partitionen** gibt.

Kaktus

Definition

Ein zusammenhängender Graph heißt **Kaktus**, falls jedes Paar von einfachen Kreisen höchstens einen Knoten gemeinsam hat.

Der Kaktus eines Graphen G ist ein Graph, der in kompakter Form alle MinCuts von G (durch dessen zirkuläre Partitionen) repräsentiert.

Ein Graph bestehend aus einem einzelnen Knoten heißt *trivialer Kaktus*.

Oft wird in der Definition auch verlangt, dass jede Kante zu genau einem einfachen Kreis gehört, wobei eine Doppelkante ein Kreis der Länge 2 ist.

Man kann einen solchen Kaktus erhalten, indem man jede Brücke (mit Gewicht 2 bzw. λ) durch Doppelkanten (mit jeweiligem Gewicht 1 bzw. $\lambda/2$) ersetzt.

Ein Graph ist ein Kaktus genau dann, wenn jeder Block entweder ein einfacher Kreis oder eine einzelne Kante (Brücke) ist.

Kaktus-Repräsentation von Minimum Cuts

Betrachte

- einen Graph $G = (V_G, E_G)$,
 - einen **Kaktus** $\mathcal{R} = (V_{\mathcal{R}}, E_{\mathcal{R}})$ und
 - eine **Abbildung** φ der Graphknoten in die Menge der Kaktusknoten
 $\varphi: V_G \mapsto V_{\mathcal{R}}$.
-
- Die Menge $V_{\mathcal{R}}$ kann einen Knoten x enthalten, der nicht in der Bildmenge der Abbildung φ enthalten ist, für den also V_G keinen Knoten v mit $\varphi(v) = x$ enthält.

Ein solcher Knoten wird *leerer Knoten* genannt.

Kaktus-Repräsentation von Minimum Cuts

Sei $\mathcal{C}(\mathcal{R})$ die Menge aller Minimum Cuts vom Kaktus \mathcal{R} , d.h., $\{S, V_{\mathcal{R}} \setminus S\} \in \mathcal{C}(\mathcal{R})$ gilt genau dann, wenn in \mathcal{R} zwischen S und $V_{\mathcal{R}} \setminus S$ genau zwei Kanten verlaufen, die zum gleichen Kreis in \mathcal{R} gehören.

Definition

Für eine gegebene Teilmenge $\mathcal{C}' \subseteq \mathcal{C}(G)$ von Minimum Cuts in G , nennt man ein Paar (\mathcal{R}, φ) , bestehend aus einem Kaktus \mathcal{R} und einer Knotenabbildung φ , **Kaktus-Repräsentation** für \mathcal{C}' , falls folgendes gilt:

- 1 Für jeden beliebigen Kaktus-MinCut $\{S, V_{\mathcal{R}} \setminus S\} \in \mathcal{C}(\mathcal{R})$ gehört der Cut $\{X, \bar{X}\}$ mit $X = \{u \in V_G \mid \varphi(u) \in S\}$ und $\bar{X} = \{u \in V_G \mid \varphi(u) \in V_{\mathcal{R}} \setminus S\}$ zu den Graph-MinCuts in \mathcal{C}' .
- 2 Für jeden Graph-MinCut $\{X, \bar{X}\} \in \mathcal{C}'$ existiert ein Kaktus-MinCut $\{S, V_{\mathcal{R}} \setminus S\} \in \mathcal{C}(\mathcal{R})$ mit $X = \{u \in V_G \mid \varphi(u) \in S\}$ und $\bar{X} = \{u \in V_G \mid \varphi(u) \in V_{\mathcal{R}} \setminus S\}$.

Kaktus-Repräsentation aller Minimum Cuts

Fallunterscheidung:

- 1 Graph hat **keine** zirkuläre Partition
- 2 Graph hat **genau eine** zirkuläre Partition
- 3 Graph hat **mehrere** zirkuläre Partitionen P_1, \dots, P_z

Kaktus-Repräsentation aller Minimum Cuts

1. Fall: Graph hat **keine** zirkuläre Partition

- Es kann in diesem Fall **keine Crossing Cuts** geben, denn sonst müsste es laut Lemma auch eine zirkuläre Partition geben, die eine Verfeinerung der 4 entsprechenden disjunkten Mengen ist.

⇒ Die Minimum Cuts, jeweils repräsentiert durch die Knotenmenge kleinerer Kardinalität, sind **laminar**.

Der kleinere Repräsentant ist wichtig, sonst sind nicht alle Mengen laminar.

Bsp: Knoten x, y, z verbunden durch Kanten $\{x, y\}, \{y, z\}$

⇒ MinCuts: $\{x\}|\{y, z\}, \{x, y\}|\{z\}$

⇒ $\{y, z\}$ und $\{x, y\}$ sind nicht laminar!

⇒ Die Minimum Cuts können durch einen Baum T_G repräsentiert werden.

Kaktus-Repräsentation aller Minimum Cuts

(Fortsetzung 1. Fall: Graph hat keine zirkuläre Partition)

Repräsentation der Minimum Cuts durch folgenden Baum T_G :

- Betrachte die jeweils kleinere Knotenmenge jedes Minimum Cuts und bezeichne die Menge dieser Knotenmengen mit Λ (wähle bei gleicher Kardinalität irgendeine von beiden).
- Repräsentiere jede Menge von Λ durch einen Knoten in T_G .
- Zwei Baumknoten, die zu den MinCut-Mengen A und B im Graphen gehören, sollen genau dann verbunden sein, wenn $A \subset B$ gilt und es keinen MinCut C mit $A \subset C \subset B$ gibt (der echte Obermenge von A und echte Teilmenge von B ist).
- Die Wurzeln der resultierenden Bäume repräsentieren die MinCuts in Λ , die in keiner anderen MinCut-Menge von Λ enthalten sind.
- Hinzufügen eines künstlichen Wurzelknotens und Verbinden mit den Wurzeln aller Bäume resultiert in einem Baum (T_G).

Kaktus-Repräsentation aller Minimum Cuts

(Fortsetzung 1. Fall: Graph hat keine zirkuläre Partition)

- Definiere Abbildung:
 - ▶ Jeder Knoten des Graphen G wird auf den Knoten des Baums T_G abgebildet, der zu dem MinCut mit kleinster Kardinalität gehört, der diesen Knoten enthält.
 - ▶ Jeder nicht zugeordnete Knoten wird der Wurzel zugeordnet.
 - Für jeden Minimum Cut S von G werden die Knoten von S einer Menge X von Baumknoten zugeordnet, so dass es eine Kante gibt, die beim Entfernen die Baumknoten X vom Rest des Baums trennt.
 - Andererseits zerfällt beim Entfernen einer Kante aus T_G die Menge der Baumknoten so in zwei Teile, dass die Menge der Knoten, die in dem einen Teil zugeordnet werden, die eine Seite eines Minimum Cuts bilden.
- ⇒ Wenn der Graph keine zirkulären Partitionen enthält, dann ist der Baum T_G der Kaktus C_G des Graphen G und die Anzahl seiner Knoten ist durch $2|V| - 1$ beschränkt.

Kaktus-Repräsentation aller Minimum Cuts

2. Fall: Graph hat **genau eine** zirkuläre Partition V_1, \dots, V_k .

- Die Circular Partition Cuts können durch einen Kreis mit k Knoten repräsentiert werden.
- Die Knoten jedes Partitionsteils V_i ($1 \leq i \leq k$) werden so durch einen Knoten N_i des Kreises repräsentiert, dass zwei Teile V_i und V_{i+1} (sowie V_k und V_1) durch zwei adjazente Knoten repräsentiert werden.
- Bemerkung: Für jeden Minimum Cut S , der kein Circular Partition Cut ist, ist entweder S oder \bar{S} eine echte Teilmenge eines Teils V_i (folgt direkt aus der Definition).
- Man kann den Baum $T_{(V_i, E)}$ für alle Minimum Cuts konstruieren, die Teilmenge von V_i sind, aber mit der Beschränkung, dass nur die Knoten von V_i diesem Baum zugeordnet werden.

Kaktus-Repräsentation aller Minimum Cuts

(Fortsetzung 2. Fall: Graph hat genau eine zirkuläre Partition)

- Die Wurzel von $T_{(V_i, E)}$ entspricht genau der Menge V_i .
- ⇒ Knoten N_i des Kreises kann mit der Wurzel von $T_{(V_i, E)}$ verschmolzen werden ($\forall i : 1 \leq i \leq k$).
- Dieser mit allen Bäumen verbundene Kreis ist der Kaktus C_G für G .
- Anzahl Knoten: Summe der Anzahl der Knoten aller Bäume
- ⇒ wieder beschränkt durch $2|V| - 1$
- Kreiskanten bekommen Gewicht 1 oder $\lambda/2$, Baumkanten bekommen Gewicht 2 oder λ
- ⇒ wieder Korrespondenz zwischen Minimum Cuts in G und in C_G .

Kaktus-Repräsentation aller Minimum Cuts

3. Fall: Graph hat **mehrere** zirkuläre Partitionen P_1, \dots, P_z

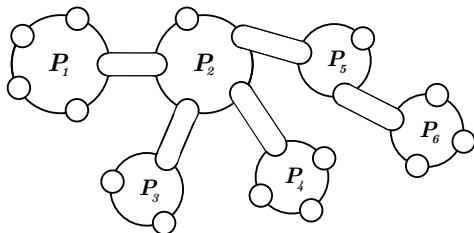
- Betrachte alle zirkulären Partitionen als Menge von Mengen
- Konstruiere den Kaktus, der die Circular Partition Cuts repräsentiert:
- Jede zirkuläre Partition wird durch einen Kreis repräsentiert.
Die Knoten des Kreises entsprechen dabei den jeweiligen disjunkten Mengen.
- Überschneidungen ergeben sich entsprechend der Definition der Kompatibilität zirkulärer Partitionen.

⇒ Es entsteht eine baumartige Struktur aus verbundenen Kreisen.
Zwei Kreise haben aber immer höchstens einen gemeinsamen Knoten.

Kaktus-Repräsentation aller Minimum Cuts

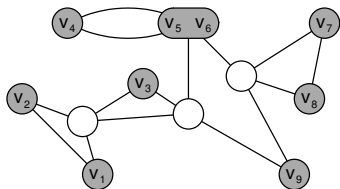
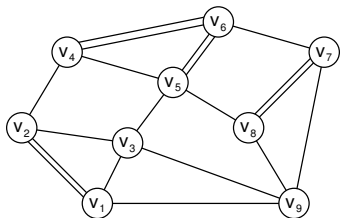
(Fortsetzung 3. Fall: Graph hat mehrere zirkuläre Partitionen P_1, \dots, P_z)

- Bsp.: Kaktus für die Circular Partition Cuts von 6 Circular Partitions



- Repräsentiere die Minimum Cuts, die nicht Teil einer zirkulären Partition sind (analog zum 2. Fall), d.h., an den Kreisknoten hängen wieder die entsprechenden Bäume.
- Man erhält den Kaktus T_C von G .
- Die Anzahl Knoten ist wieder beschränkt durch $2|V| - 1$

Beispiel für Kaktus-Repräsentation



Satz (L. Fleischer, 1998)

In Graphen mit n Knoten und m Kanten kann man eine entsprechende Kaktus-Repräsentation in Zeit $\mathcal{O}(n^3)$ bzw. $\mathcal{O}(nm \log(n^2/m))$ berechnen.

Berechnung eines globalen MinCuts

- 1994 Algorithmus publiziert von M. Stoer und F. Wagner
- benutzt keine MaxFlow-Technik
- sehr einfach
- arbeitet in $n - 1$ Phasen
- Jede einzelne Phase hat starke Ähnlichkeit zu den Algorithmen von Prim (minimale Spannbäume) bzw. Dijkstra (kürzeste Wege)
- Pro Phase Komplexität $\mathcal{O}(m + n \log n)$
- Gesamtkomplexität $\mathcal{O}(mn + n^2 \log n)$

Der Stoer/Wagner-Algorithmus

Algorithmus 11: MinCut Berechnung nach Stoer & Wagner

Input : Ungerichteter Graph $G = (V, E)$

Output : Ein MinCut C_{\min} entsprechend $\lambda(G)$

Wähle beliebigen Startknoten a ;

$C_{\min} \leftarrow$ undefiniert; $V' \leftarrow V$;

while $|V'| > 1$ **do**

$A \leftarrow \{a\}$;

while $A \neq V'$ **do**

 Füge zu A den am meisten angebotenen Knoten hinzu;

 Aktualisiere die Kapazitäten zwischen A und den Knoten in $V' \setminus A$;

$C :=$ Schnitt von V' , der den zuletzt zu A hinzugefügten Knoten vom Rest des Graphen trennt;

if $C_{\min} =$ undefiniert **or** $w(C) < w(C_{\min})$ **then**

$C_{\min} \leftarrow C$;

 Verschmelze die zwei Knoten, die zuletzt zu A hinzugefügt wurden;

return C_{\min} ;

Beschreibung des Stoer/Wagner-Algorithmus

- Wahl eines beliebigen **Startknotens** a
- Algorithmus verwaltet in jeder Phase eine **Knotenteilmenge** A
 - ▶ initialisiert mit a ,
 - ▶ wird immer wieder um einen Knoten erweitert,
 - ▶ und zwar um einen, der gerade maximale Anbindung (Summe der Kantenkapazitäten) an die aktuellen Knoten in A hat.

⇒ Maximum Adjacency Ordering (MAO)

- Nach Einfügen aller Knoten in A :
Cut, der nur den zuletzt hinzugefügten Knoten t vom Rest abtrennt, heißt '**Cut of the Phase**'.

Beschreibung des Stoer/Wagner-Algorithmus

- Nach jeder Phase kommt der 'Cut of the Phase' als globaler MinCut in Frage.
- Verschmelzung der beiden zuletzt behandelten Knoten s, t :
Wenn zwischen s und t eine Kante existiert, wird sie einfach gelöscht.
Alte Kanten $\{x, s\}$ und $\{x, t\}$ von einem anderen Knoten x werden durch eine neue Kante mit der Summe der alten Gewichte $w(s, x) + w(t, x)$ ersetzt.
- Dann: erneute MAO-Phase mit verschmolzenen Knoten ...
- 'Cut of the Phase' mit kleinster Kapazitätssumme ist globaler MinCut

Korrektheit des Stoer/Wagner-Algorithmus

Lemma

Der 'Cut of the Phase' ist ein Minimum (s, t) -Cut für die beiden zuletzt in A eingefügten Knoten s und t .

Korrektheit des Stoer/Wagner-Algorithmus

Beweis.

- Betrachte für die letzten zwei Knoten s und t einen beliebigen (also evt. auch Minimum) s - t -Cut C .
- Ein Knoten $v \neq a$ heißt **aktiv**, falls sich v und sein unmittelbarer Vorgänger bezüglich des Hinzufügens zu A auf entgegengesetzten Seiten von C befinden.
- Für einen Knoten v sei
 A_v : die Menge von Knoten, die in A sind bevor v hinzugefügt wird,
 C_v : der durch C in $A_v \cup \{v\}$ induzierte Cut.
- Sei $w(S, v)$ für eine Knoten(teil)menge S die Kapazitätssumme aller Kanten zwischen v und den Knoten in S .

Korrektheit des Stoer/Wagner-Algorithmus

Beweis.

- Beweisidee: (Induktion über die aktiven Knoten)

Für jeden aktiven Knoten v gilt:

Die Anbindung (Adjazenz) zu den Knoten, die zuvor eingefügt wurden (also A_v), überschreitet nicht das Gewicht des durch C in $A_v \cup \{v\}$ induzierten Cuts (also C_v).

- Also zu zeigen:

$$w(A_v, v) \leq w(C_v)$$

- Induktionsanfang: (1. aktiver Knoten)

Im Basisfall ist die Ungleichung erfüllt, weil für den ersten aktiven Knoten die Werte auf beiden Seiten gleich sind.

Korrektheit des Stoer/Wagner-Algorithmus

Beweis.

- Induktionsvoraussetzung:

Das Lemma stimmt für alle aktiven Knoten bis zum aktiven Knoten v .

⇒ Der Wert für den nächsten aktiven Knoten u ist

$$\begin{aligned}
 w(A_u, u) &= w(A_v, u) + w(A_u \setminus A_v, u) \\
 &\leq w(A_v, v) + w(A_u \setminus A_v, u) \\
 &\leq w(C_v) + w(A_u \setminus A_v, u) \quad (\text{Ind.voraussetzung}) \\
 &\leq w(C_u)
 \end{aligned}$$

- Die letzte Zeile folgt, weil alle Kanten zwischen $A_u \setminus A_v$ und u ihr Gewicht zu $w(C_u)$ beitragen, aber nicht zu $w(C_v)$.
- Da t durch den Cut C von seinem unmittelbaren Vorgänger s getrennt wird, ist t immer ein aktiver Knoten.
- Es folgt $w(A_t, t) \leq w(C_t)$ (mit $C_t = C$).



Korrektheit des Stoer/Wagner-Algorithmus

Satz

Ein 'Cut of the Phase' mit minimaler Kantenkapazität ist ein (globaler) MinCut des gegebenen Graphen.

Korrektheit des Stoer/Wagner-Algorithmus

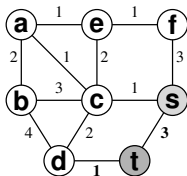
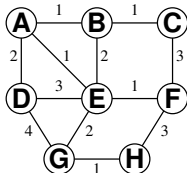
Beweis.

- Für $|V| = 2$ trivial: bei zwei Knoten existiert nur *ein* Cut.
 - Für $|V| > 2$ Fallunterscheidung:
Betrachte eine Phase mit letzten Knoten s und t
 - 1 Entweder der Graph hat einen MinCut, der gleichzeitig ein (lokaler) Minimum s - t -Cut ist.
⇒ Cut of the Phase ist nach Lemma globaler MinCut des Graphen
 - 2 Andernfalls hat der Graph einen globalen MinCut, bei dem s und t nicht separiert werden (sich also auf der gleichen Seite befinden).
Dann wird der globale MinCut durch das Verschmelzen von s und t nicht verändert.
- ⇒ (Induktion über die Anzahl der Knoten)
'Cut of the Phase' mit minimaler Gewichtssumme ist globaler MinCut.



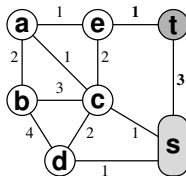
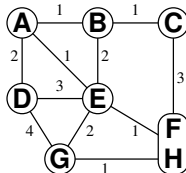
Beispiel für den Stoer/Wagner-Algorithmus

Phase 1



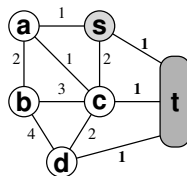
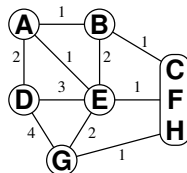
4

Phase 2



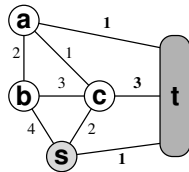
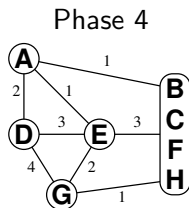
4

Phase 3

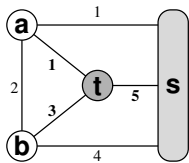
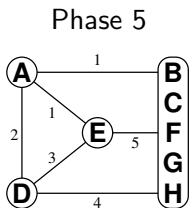


3

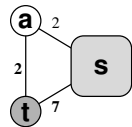
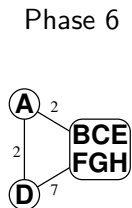
Beispiel für den Stoer/Wagner-Algorithmus



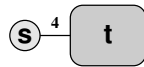
5



9



9



4

Komplexität des Stoer/Wagner-Algorithmus

- Adjazenzwerte für verbleibende Knoten ($V \setminus A$) eines MAOs können in einer Priority Queue (Maximum-Variante) gespeichert werden.
- Der nächste Knoten, der zu A hinzugefügt werden soll, wird per `deleteMax`-Operation bestimmt.
- Beim Hinzufügen eines Knotens zu A wird `increaseKey` für alle in $V \setminus A$ verbleibenden Nachbarn des Knotens aufgerufen.
- Bei Verwendung von Fibonacci-Heaps ergibt sich pro Phase (MAO) Komplexität $\mathcal{O}(m + n \log n)$
(wie bei den Algorithmen von Prim bzw. Dijkstra)
- Gesamtkomplexität für $n - 1$ Phasen: $\mathcal{O}(mn + n^2 \log n)$

Randomisierter MinCut-Algorithmus von Karger

- Gesucht: globaler Minimum Cut in einem Multi-Graphen
- D. Karger (1992): Vorschlag eines sehr einfachen Algorithmus ohne augmentierende Pfade und Flussberechnungen
- Ansatz: **randomisierter (Monte-Carlo-)Algorithmus**, der
 - ▶ mit gewisser (Erfolgs-)Wahrscheinlichkeit den Minimum Cut liefert und
 - ▶ mit gewisser (Fehler-)Wahrscheinlichkeit einen beliebigen anderen Cut

Randomisierter MinCut-Algorithmus von Karger

- Algorithmus wählt beliebige Kante $e = \{u, v\}$ von G (gleichverteilt, d.h. jede Kante wird mit gleicher Wahrscheinlichkeit gezogen)
- gewählte Kante wird kontrahiert, d.h. Multi-Graph G' wird erzeugt, bei dem Knoten u und v zu einem neuen Knoten w verschmolzen sind
- Kanten zwischen u und v verschwinden.
- Andere Kanten bleiben erhalten. Wenn genau ein Endknoten entweder u oder v ist, dann endet dafür eine neue Kante am (Super-)Knoten w .
- Hinweis: auch wenn G keine Multikanten enthält, kann G' welche enthalten.
- Das Verfahren wird rekursiv auf G' angewendet.

Randomisierter MinCut-Algorithmus von Karger

- Der Algorithmus endet, wenn nur noch zwei (Super-)Knoten vorhanden sind.
- Jeder der beiden Knoten enthält eine gewisse Menge der Knoten des ursprünglichen Graphen.
- Diese Partition definiert einen Cut, der vom Algorithmus ausgegeben wird.

Randomisierter MinCut-Algorithmus von Karger

Satz

Der Monte-Carlo-Algorithmus von Karger gibt mit Wahrscheinlichkeit $\geq 1/\binom{n}{2}$ den globalen Minimum Cut des Multigraphen zurück.

- Nachdem es exponentiell viele Cuts in G gibt (und höchstens quadratisch viele MinCuts), würde man vermuten, dass die Wahrscheinlichkeit, den globalen MinCut zu finden, exponentiell klein ist.

⇒ Was also favorisiert die kleinen Cuts?

Randomisierter MinCut-Algorithmus: Analyse

Beweis.

- Betrachte globalen MinCut (A, B) in G .
- Sei die Schnitt-Kardinalität k , d.h. es gibt genau k Kanten $(F \subseteq E)$, die einen Endpunkt in A und einen in B haben.
- gesucht: untere Schranke für die Wahrscheinlichkeit, dass der Algorithmus (A, B) zurück gibt.
- Was kann dabei schiefgehen?
- Eine Kante aus F könnte kontrahiert werden.
- Dann würden zwei Knoten kontrahiert, die auf unterschiedlichen Seiten des Cuts (A, B) liegen.
- Der Algorithmus könnte nicht mehr (A, B) ausgeben.
- Wenn eine beliebige Kante aus $E \setminus F$ kontrahiert wird, besteht noch die Chance auf die Ausgabe (A, B) .

Randomisierter MinCut-Algorithmus: Analyse

Beweis.

- also gesucht: obere Schranke für die Wahrscheinlichkeit, dass eine Kante aus F kontrahiert wird.
 - Wir brauchen eine untere Schranke für die Kardinalität von E .
 - Wenn der globale MinCut den Wert $\lambda(G) = k$ hat, gilt für den Minimalgrad des Graphen $\delta(G) \geq \lambda(G) = k$.
 - Es gilt also $|E| = m \geq kn/2$.
- ⇒ Die Wahrscheinlichkeit, dass eine Kante aus F kontrahiert wird, ist

$$\frac{k}{m} \leq \frac{k}{kn/2} = \frac{2}{n}$$

- Betrachte jetzt die Situation nach j Iterationen.
- Es gibt $n - j$ (Super-)Knoten im aktuellen G' .

Randomisierter MinCut-Algorithmus: Analyse

Beweis.

- Annahme: es wurde noch keine Kante aus F kontrahiert.
 - Da jeder Cut in G' auch ein Cut in G ist, sind zu jedem Superknoten in G' mindestens k Kanten inzident.
 - D.h. G' hat mindestens $k(n-j)/2$ Kanten.
- ⇒ Die Wahrscheinlichkeit, dass eine Kante aus F in der nächsten Iteration $j+1$ kontrahiert wird, ist

$$\frac{k}{k(n-j)/2} = \frac{2}{n-j}$$

- Cut (A, B) wird ausgegeben, wenn in Iteration $1, \dots, n-2$ keine Kante aus F kontrahiert wird.

Randomisierter MinCut-Algorithmus: Analyse

Beweis.

- Sei \mathcal{E}_j das Ereignis, dass in Iteration j keine Kante aus F kontrahiert wird. Dann gilt also: $\Pr[\mathcal{E}_1] \geq 1 - 2/n$ und

$$\Pr[\mathcal{E}_{j+1} | \mathcal{E}_1 \cap \mathcal{E}_2 \cap \dots \cap \mathcal{E}_j] \geq 1 - 2/(n - j)$$

- gesucht: untere Schranke für $\Pr[\mathcal{E}_1 \cap \mathcal{E}_2 \cap \dots \cap \mathcal{E}_{n-2}]$ bzw. $\Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 | \mathcal{E}_1] \cdot \dots \cdot \Pr[\mathcal{E}_{n-2} | \mathcal{E}_1 \cap \mathcal{E}_2 \cap \dots \cap \mathcal{E}_{n-3}]$

$$\begin{aligned} &\geq \left(1 - \frac{2}{n}\right) \left(1 - \frac{2}{n-1}\right) \dots \left(1 - \frac{2}{n-j}\right) \dots \left(1 - \frac{2}{3}\right) \\ &= \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdot \dots \cdot \frac{2}{4} \cdot \frac{1}{3} \\ &= \frac{2 \cdot 1}{n(n-1)} = \binom{n}{2}^{-1} \end{aligned}$$



Randomisierter MinCut-Algorithmus: Analyse

- Wahrscheinlichkeit, dass der randomisierte MinCut-Algorithmus nicht den Cut (A, B) ausgibt, ist höchstens $1 - 1/\binom{n}{2}$
- Nach $\binom{n}{2}$ Läufen mit unabhängigen Zufallsentscheidungen ist die Wahrscheinlichkeit, dass nie der MinCut gefunden wurde, höchstens

$$\left(1 - 1/\binom{n}{2}\right)^{\binom{n}{2}} \leq \frac{1}{e}$$

- Wenn $c \binom{n}{2} \log n$ Läufe gestartet werden, sinkt die Fehlerwahrscheinlichkeit auf $\leq e^{-c \log n} = 1/n^c$.
- Laufzeit ist in dieser einfachen Form noch relativ hoch, verglichen mit dem besten deterministischen Algorithmus

Randomisierter MinCut-Algorithmus: Analyse

- Ansatz ist unbalanciert: Fehlerwahrscheinlichkeit erhöht sich zum Ende eines Laufs
- Wenn man den Algorithmus nur t Schritte laufen lässt, ist die Wahrscheinlichkeit, dass keine Kante aus F kontrahiert wird

$\Pr[\mathcal{E}_1 \cap \mathcal{E}_2 \cap \dots \cap \mathcal{E}_t]$ bzw.

$\Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2|\mathcal{E}_1] \cdot \dots \cdot \Pr[\mathcal{E}_t|\mathcal{E}_1 \cap \mathcal{E}_2 \cap \dots \cap \mathcal{E}_{t-1}]$

$$\begin{aligned}
 &\geq \left(1 - \frac{2}{n}\right) \left(1 - \frac{2}{n-1}\right) \dots \left(1 - \frac{2}{n-(t-1)}\right) \\
 &= \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdot \dots \cdot \frac{n-t}{n-t+2} \cdot \frac{n-t-1}{n-t+1} \\
 &= \frac{(n-t) \cdot (n-t-1)}{n(n-1)} \in \Omega\left(\frac{(n-t)^2}{n^2}\right)
 \end{aligned}$$

Randomisierter MinCut-Algorithmus: Analyse

Verbesserte Variante:

- Lasse die einfache Variante mehrmals laufen
z.B. 2-mal bis $\lceil n/\sqrt{2} + 1 \rceil$ Knoten
oder 4-mal bis $n/2$ Knoten
- Setze jeden der Läufe rekursiv mit der modifizierten Variante fort.
- Laufzeit: $\mathcal{O}(n^2 \log n)$ mit Erfolgswahrscheinlichkeit $\Omega(1/\log n)$
- $\mathcal{O}(\log n)$ -malige Wiederholung des modifizierten Algorithmus sorgt für konstante Erfolgswahrscheinlichkeit.
- Um eine kleine Fehlerwahrscheinlichkeit ϵ zu erreichen, kann man $\mathcal{O}(\log n \log \epsilon^{-1})$ Wiederholungen durchführen, was zu einer Gesamtlaufzeit von $\mathcal{O}(n^2 \log^2 n \log \epsilon^{-1})$ führt.

All-Pairs MaxFlow / MinCut

- gegeben:
Graph $G = (V, E)$ mit n Knoten,
 p ausgewählte Knoten
 - gesucht:
(lokale) MaxFlow/MinCut-Werte für alle Paare aus den p gegebenen Knoten
- ⇒ kann einfach durch Lösen von $\binom{p}{2} = \frac{1}{2}p(p-1)$ MaxFlow-Problemen berechnet werden.
- kann in ungerichteten Graphen aber auch mit nur $p-1$ MaxFlow-Berechnungen gelöst werden.

Equivalent Flow Trees

Sei G also ungerichtet und sei $v_{ij} = v_{ji}$ der Wert eines MaxFlows zwischen den Knoten i und j .

Lemma

- ① Für alle Knoten $i, j, k \in \{1, \dots, n\}$ gilt:

$$v_{ik} \geq \min\{v_{ij}, v_{jk}\}$$

- ② Es gibt einen Baum T auf den Knoten 1 bis n , so dass für alle Paare von Knoten i, j gilt:

$$v_{ij} = \min\{v_{ij_1}, v_{j_1j_2}, \dots, v_{j_kj}\},$$

wobei $i - j_1 - \dots - j_k - j$ der (eindeutige) Pfad von Knoten i zu Knoten j in T ist.

Equivalent Flow Trees

Beweis.

- ① Sei (X, \bar{X}) ein i, k -MinCut, d.h. $v_{ik} = w(X, \bar{X})$.
 - ▶ Falls $j \in \bar{X}$, dann gilt $v_{ij} \leq w(X, \bar{X}) = v_{ik}$.
 - ▶ Falls $j \in X$, dann gilt $v_{jk} \leq w(X, \bar{X}) = v_{ik}$.

- ② Betrachte K_n (vollständiger Graph mit n Knoten), wobei jede Kante (i, j) Gewicht v_{ij} hat.
 Sei T darin ein Spannbaum maximalen Gewichts.
 - ▶ Induktiv folgt aus dem vorangegangenen Punkt, dass für jedes Knotenpaar (i, j) gilt: $v_{ij} \geq \min\{v_{ij_1}, v_{j_1j_2}, \dots, v_{j_kj}\}$, wobei $i - j_1 - \dots - j_k - j$ der (eindeutige) Pfad von Knoten i zu Knoten j in T ist.
 - ▶ Angenommen für ein Paar (i, j) gilt echte Ungleichheit.
 Dann gibt es eine Kante (i', j') auf dem Pfad zwischen i und j mit $v_{ij} > v_{i'j'}$.
 Das würde bedeuten, dass der Baum, der aus T durch Tausch von (i', j') gegen (i, j) entsteht, ein größeres Gewicht hätte. (Widerspruch)



Algorithmus von Gomory und Hu

- Die Existenz eines **Equivalent Flow Trees** hat natürlich gleichzeitig die Konsequenz, dass unter den $\binom{n}{2}$ MaxFlow- bzw. MinCut-Werten v_{ij} nur höchstens $n - 1$ verschiedene Werte existieren können (nämlich die Kantengewichte von T).
- Außerdem kommt man so zu der Vermutung, dass $n - 1$ MaxFlow-Berechnungen genügen, um einen solchen Equivalent Flow Tree T zu konstruieren und damit alle v_{ij} -Werte zu berechnen. Ein Algorithmus von Gomory und Hu erreicht dies tatsächlich.

Algorithmus von Gomory und Hu

- Gegeben zwei Knoten i, j und ein i, j -MinCut (X, \bar{X}) .
- Definiere den **kontrahierten Graph** G^c als den Graph, den man aus G erhält, indem man die Knoten in \bar{X} zu einem einzelnen (speziellen) Knoten $u_{\bar{X}}$ zusammenfasst und für jeden Knoten $v \in X$ alle über den Schnitt führenden Kanten durch eine einzelne Kante $\{v, u_{\bar{X}}\}$ mit der entsprechend summierten Gesamtkapazität ersetzt.

Lemma

Für jedes Paar von (normalen) Knoten i', j' in G^c (d.h. $i', j' \in X$) hat der MaxFlow bzw. MinCut zwischen i' und j' in G^c den gleichen Wert wie der MaxFlow/MinCut im Original-Graphen.

(Entsprechendes gilt natürlich für Knoten $i', j' \in \bar{X}$, wenn man X in G kontrahiert.)

Algorithmus von Gomory und Hu

Beweis

- Sei (X, \bar{X}) ein i, j -MinCut und (Y, \bar{Y}) ein i', j' -MinCut in G .
- Definiere Knotenmengen
 $A = X \cap Y, \bar{A} = X \cap \bar{Y}, B = \bar{X} \cap Y, \bar{B} = \bar{X} \cap \bar{Y}$
- Wir können annehmen, dass $i' \in A, j' \in \bar{A}, i \in A$
- 1. Fall: $j \in B$

$$w(X, \bar{X}) = w(A, B) + w(A, \bar{B}) + w(\bar{A}, B) + w(\bar{A}, \bar{B})$$

$$w(Y, \bar{Y}) = w(A, \bar{A}) + w(A, \bar{B}) + w(\bar{A}, B) + w(B, \bar{B})$$

Algorithmus von Gomory und Hu

Beweis.

- (Y, \bar{Y}) ist i', j' -MinCut und $\text{Cut}(A \cup B \cup \bar{B}, \bar{A})$ trennt i' und j' .
- $\Rightarrow w(Y, \bar{Y}) - w(A \cup B \cup \bar{B}, \bar{A}) = w(A, \bar{B}) + w(B, \bar{B}) - w(\bar{A}, \bar{B}) \leq 0$
- (X, \bar{X}) ist i, j -MinCut und $\text{Cut}(A \cup \bar{A} \cup \bar{B}, B)$ trennt i und j .
- $\Rightarrow w(X, \bar{X}) - w(A \cup \bar{A} \cup \bar{B}, B) = w(A, \bar{B}) + w(\bar{A}, \bar{B}) - w(B, \bar{B}) \leq 0$
- Summe beider Ungleichungen ergibt $w(A, \bar{B}) \leq 0$, also $w(A, \bar{B}) = 0$
- Es folgt $w(B, \bar{B}) - w(\bar{A}, \bar{B}) = 0$.
- Also $(A \cup B \cup \bar{B}, \bar{A}) = (A \cup \bar{X}, \bar{A})$ ist auch (i', j') -MinCut.
- 2. Fall: $j \in \bar{B}$
ähnlich: $(A, \bar{A} \cup \bar{X})$ ist (i', j') -MinCut.
- \Rightarrow Es gibt immer einen (i', j') -MinCut, so dass \bar{X} auf einer Seite ist.



Algorithmus von Gomory und Hu

Corollary

Für jedes Paar von Knoten $i', j' \in X$ (oder \bar{X}) gibt es einen Minimum i', j' -Cut, so dass sich alle Knoten von \bar{X} (bzw. X) auf der gleichen Seite des Schnitts befinden.

- Für zwei Knoten i, j und einen i, j -MinCut (X, \bar{X}) repräsentiert man die aktuelle Situation durch einen Baum mit zwei Knoten (die für X und \bar{X} stehen) und einer Kante mit dem Gewicht v_{ij} . Die Kante repräsentiert dabei den Schnitt (X, \bar{X}) .
- Sei A die Menge der p zu betrachtenden Knoten, für die die MaxFlow/MinCut-Werte berechnet werden sollen.
- Die ersten beiden Knoten i und j werden aus A gewählt.

Algorithmus von Gomory und Hu

Definition

Ein Baum T wird als **Semi-Cut Tree** bezeichnet, falls er folgende Eigenschaften besitzt:

- 1 Jeder Knoten U von T entspricht einer Teilmenge der Knoten von G und enthält mindestens einen Knoten der Menge A .
- 2 Jede Kante (U, V) ist mit einem Label v versehen, so dass es Knoten $i, j \in A$ mit $i \in U$ und $j \in V$ gibt und der MaxFlow/MinCut zwischen i und j den Wert v hat.
- 3 Jede Kante (U, V) repräsentiert einen i, j -MinCut mit $i, j \in A$ und i ist in einem Knoten des Teilbaums auf der Seite von U enthalten und j ist in einem Knoten des Teilbaums auf der Seite von V enthalten (und die zwei Teile des Cut bestehen aus der jeweiligen Knotenmenge).

Wenn jeder Knoten eines Semi-Cut Trees T genau einen Knoten der Menge A enthält, dann bezeichnet man T als **Cut Tree für A** .

Algorithmus von Gomory und Hu

Satz

Sei T ein Cut Tree für A .

Dann gilt für jedes Paar $i, j \in A$: $v_{ij} = \min\{v_1, \dots, v_{k+1}\}$, wobei v_1 bis v_{k+1} die Kantengewichte in T auf dem Pfad vom Knoten, der i enthält, zum Knoten, der j enthält, sind.

Beweis.

- Sei $i - j_1 - \dots - j_k - j$ die Folge von A -Knoten, die dem Pfad in T vom Knoten mit i zum Knoten mit j entsprechen.
- Aufgrund der Eigenschaften des Cut Trees sind die Kantenlabels einfach $v_{ij_1}, \dots, v_{j_k j}$.
- Aufgrund des Lemmas gilt: $v_{ij} \geq \min\{v_{ij_1}, \dots, v_{j_k j}\}$.
- Andererseits entspricht jede Kante einem i, j -Schnitt, wobei die Kapazität dem Kantenlabel entspricht. Es gilt also:
 $v_{ij} \leq \min\{v_{ij_1}, \dots, v_{j_k j}\}$ und damit $v_{ij} = \min\{v_{ij_1}, \dots, v_{j_k j}\}$.

Algorithmus von Gomory und Hu

Satz

Ein Cut Tree für A existiert und kann mit Hilfe von nur $p - 1$ MaxFlow-Berechnungen konstruiert werden.

Beweis.

- Angenommen, wir haben einen Semi-Cut Tree T für A und T hat noch einen Knoten U , der zwei Knoten $i, j \in A$ enthält.
- Aufgrund des Lemmas hat der MaxFlow zwischen i und j in G genau den gleichen Wert wie der MaxFlow zwischen i und j in dem (kontrahierten) Graph G^c , der entsteht, wenn man die Knotenmengen in jedem zu U verbundenen Teilbaum zu einem einzelnen (speziellen) Knoten kontrahiert und für jeden (Original-)Knoten $v \in U$ die Kanten, die in die jeweiligen Teilbäume führen, durch einzelne Kanten zu den entsprechenden neuen (speziellen) Knoten mit aufsummiertem Kantengewicht ersetzt.

Algorithmus von Gomory und Hu

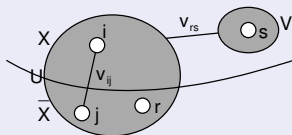
Beweis.

- Sei (X, \bar{X}) ein i, j -MinCut in G^c (notwendigerweise mit Kapazität v_{ij}).
- Konstruiere einen Baum T' wie folgt:
 - ▶ Spalte U in zwei Knoten X und \bar{X} .
 - ▶ Verbinde die Knoten X und \bar{X} durch eine Kante mit Label v_{ij} und verbinde jeden Nachbarn V von U entweder zu X oder zu \bar{X} , in Abhängigkeit von der Seite des Cuts, auf der sich der zu V 's Teilbaum gehörende spezielle Knoten in G^c befunden hat (mit Original-Kantenlabel).
- Die Behauptung ist nun, dass T' auch wieder ein Semi-Cut Tree für A ist.

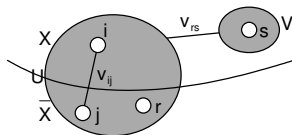
Algorithmus von Gomory und Hu

Beweis.

- Die einzige nicht-triviale zu überprüfende Eigenschaft ist der zweite Teil der Definition.
- Sei also V irgendein Nachbar von U in T und entspreche das Label der Kante (U, V) einem MaxFlow/MinCut-Wert zwischen $r \in U$ und $s \in V$ ($r, s \in A$).
- Sei o.B.d.A. $j, r \in \bar{X}$, dann können folgende zwei Fälle auftreten:
 - V wird mit \bar{X} verbunden.
Die Eigenschaften des Semi-Cut Trees bleiben erhalten.
 - V wird mit X verbunden.



Algorithmus von Gomory und Hu



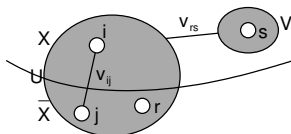
Beweis.

- ▶ Das Label v_{ij} für die Kante $\{X, \bar{X}\}$ entspricht der Definition.
- ▶ Betrachte nun das Label (v_{rs}) der Kante (X, V) .

Es wird gezeigt, dass $v_{is} = v_{rs}$:

- ★ $v_{is} \geq \min\{v_{ij}, v_{jr}, v_{rs}\}$ (Lemma).
- ★ Da i und s auf der gleichen Seite des Cuts sind, können die MaxFlow/MinCut-Werte zwischen Knoten in \bar{X} nicht den Wert von v_{is} beeinflussen (v_{jr} ist also egal) und es gilt $v_{is} \geq \min\{v_{ij}, v_{rs}\}$.

Algorithmus von Gomory und Hu



Beweis.

- ★ Andererseits muss v_{is} durch den zur Kante (U, V) in T gehörigen MinCut beschränkt sein, d.h. $v_{is} \leq v_{rs}$.
 - ★ (X, \bar{X}) hat den Wert v_{ij} und ist auch ein i - r -Cut, also $v_{ij} \geq v_{ir} \geq \min\{v_{is}, v_{rs}\}$
 - ★ (X, \bar{X}) ist auch ein r - s -Cut, also $v_{ij} \geq v_{rs}$
 - ★ Mit $v_{rs} \geq v_{is} \geq \min\{v_{ij}, v_{rs}\} = v_{rs}$ folgt daraus $v_{is} = v_{rs}$ und die Kante (X, V) entspricht dem MaxFlow zwischen i und s (mit Wert und Schnitt-Mengen wie im Original-Graph).
- Eine wiederholte Aufspaltung liefert den Cut Tree mit Hilfe von $p - 1$ MaxFlow-Berechnungen.



Unit Capacity Networks

Definition

- Ein Graph wird als **Unit Capacity Network** (oder *0-1 Network*) bezeichnet, falls die Kapazität aller Kanten gleich 1 ist.
- Ein Unit Capacity Network ist vom **Typ 1**, falls es keine parallelen Kanten hat.
- Es ist vom **Typ 2**, falls für jeden Knoten v ($v \neq s$, $v \neq t$) entweder der Eingangsgrad $d^-(v)$ oder der Ausgangsgrad $d^+(v)$ gleich 1 ist.

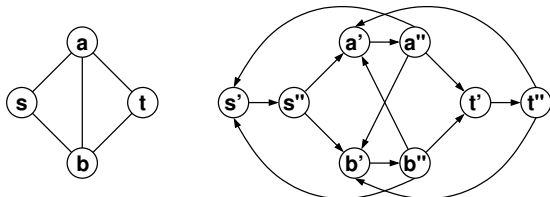
Unit Capacity Networks

Lemma

- Ein MaxFlow/MinCut kann für ein Unit Capacity Network (mit Dinitz' Algorithmus) in Zeit $\mathcal{O}(m^{3/2})$ berechnet werden.
- Für Unit Capacity Networks vom Typ 1 ist die Zeitkomplexität von Dinitz' Algorithmus $\mathcal{O}(n^{2/3}m)$.
- Für Unit Capacity Networks vom Typ 2 ist die Zeitkomplexität von Dinitz' Algorithmus $\mathcal{O}(n^{1/2}m)$.

(Beweis: siehe Shimon Even, Graph Algorithms, 1979)

Ungerichtete ungewichtete Graphen



- Gegeben: ungerichteter (ungewichteter) Graph $G = (V, E)$ mit n Knoten und m Kanten
- Konstruiere gerichteten Graph $\bar{G} = (\bar{V}, \bar{E})$ mit $|\bar{V}| = 2n$ und $|\bar{E}| = 2m + n$ wie folgt:
 - ▶ Ersetze jeden Knoten $v \in V$ durch zwei Knoten $v', v'' \in \bar{V}$, verbunden durch eine (interne) Kante $e_v = (v', v'') \in \bar{E}$.
 - ▶ Ersetze jede Kante $e = (u, v) \in E$ durch zwei (externe) Kanten $e' = (u'', v')$ und $e'' = (v'', u') \in \bar{E}$.

Ungerichtete ungewichtete Graphen

- $\kappa(s, t)$ wird nun berechnet als MaxFlow in \bar{G} von Quelle s'' zu Senke t' mit Unit Capacity-Kanten
- Hinweis: $c(e_v) = 1$, $c(e') = c(e'') = \infty$ führt übrigens zum gleichen Ergebnis.
- Für jedes Paar $v', v'' \in \bar{V}$, das einen internen Knoten $v \in V$ repräsentiert, ist die interne Kante (v', v'') die einzige von v' ausgehende Kante und die einzige eingehende Kante von v'' . Der Graph \bar{G} ist also ein UCN vom Typ 2.
- Nach dem Lemma kann die Berechnung des MaxFlow bzw. des lokalen Knotenzusammenhangs in Zeit $\mathcal{O}(\sqrt{nm})$ erfolgen.

Ungerichtete ungewichtete Graphen

Trivialer Algorithmus zur Bestimmung von $\kappa(G)$:

- Bestimme Minimum aller lokalen Knotenzusammenhangszahlen.
- Für die Endknoten jeder Kante (s, t) in G gilt:

$$\kappa_G(s, t) = n - 1$$

- Anzahl notwendiger MaxFlow-Berechnungen:

$$\frac{n(n-1)}{2} - m$$

Ungerichtete ungewichtete Graphen

Besserer Algorithmus zur Bestimmung von $\kappa(G)$:

- Betrachte minimalen Knoten-Separator $S \subset V$, der eine 'linke' Knotenteilmenge $L \subset V$ von einer 'rechten' Teilmenge $R \subset V$ separiert.
- Man könnte $\kappa(G)$ berechnen, indem man einen Knoten s in einer Teilmenge (L oder R) fixiert und die lokalen Zusammenhangszahlen $\kappa_G(s, t)$ für alle Knoten $t \in V \setminus \{s\}$ berechnet, wobei einer dieser Knoten auf der anderen Seite des Schnitts liegen muss.
- Problem: wie wählt man einen Knoten s , so dass s nicht zu jedem Minimum Vertex Separator gehört?
- Da $\kappa(G) \leq \delta(G)$, könnte man $\delta(G) + 1$ Knoten für s versuchen. Einer davon kann nicht Teil aller Minimum Knoten-Separatoren sein.
- Der Algorithmus hat Komplexität $\mathcal{O}((\delta + 1) \cdot (n - 1) \cdot \sqrt{nm}) = \mathcal{O}(\delta n^{3/2} m)$

Knotenzusammenhang (Even & Tarjan)

Algorithmus 12: Knotenzusammenhang κ (Even & Tarjan)

Input : Ungerichteter Graph $G = (V, E)$

Output : $\kappa(G)$

$\kappa_{\min} \leftarrow n - 1;$

$i \leftarrow 1;$

while $i \leq \kappa_{\min}$ **do**

for $j \leftarrow i + 1$ **to** n **do**

if $i > \kappa_{\min}$ **then**

break;

else if $\{v_i, v_j\} \notin E$ **then**

 Berechne $\kappa_G(v_i, v_j)$ mit MaxFlow-Prozedur;

$\kappa_{\min} \leftarrow \min\{\kappa_{\min}, \kappa_G(v_i, v_j)\};$

$i \leftarrow i + 1;$

return $\kappa_{\min};$

Knotenzusammenhang (Even & Tarjan)

Even/Tarjan-Algorithmus zur Berechnung des (globalen) Knotenzusammenhangs κ

- stoppt die Berechnung der lokalen Knotenzusammenhangszahlen $\kappa_G(v_i, v_j)$, falls das Minimum unter die Anzahl der momentan betrachteten Knoten i fällt
 - betrachtet höchstens $\kappa + 1$ Knoten in der Schleife für Variable i
 - Jeder Knoten hat mindestens $\delta(G)$ Nachbarn, also höchstens $n - \delta - 1$ Nicht-Nachbarn.
- ⇒ maximal $\mathcal{O}((n - \delta - 1)(\kappa + 1))$ Aufrufe für die Berechnung des lokalen Zusammenhangs (MaxFlow für zwei gegebene Knoten)
- ⇒ Da $\kappa \leq \delta \leq \bar{d} = 2m/n$ wird der richtige Wert spätestens in Aufruf $2m/n + 1$ gefunden.
- ⇒ Komplexität: $\mathcal{O}(\sqrt{nm}^2)$

Knotenzusammenhang (Esfahanian & Hakimi)

Verbesserung von Esfahanian & Hakimi:

Lemma

Wenn ein Knoten v zu allen Knoten-Separatoren minimaler Kardinalität gehört, dann gibt es für jeden Minimum Vertex-Cut S zwei Knoten $\ell \in L_S$ und $r \in R_S$, die zu v adjazent sind.

Knotenzusammenhang (Esfahanian & Hakimi)

Beweis.

- Annahme: v ist an allen Minimum Vertex-Cutsets beteiligt.
- Betrachte die beiden (getrennten) Teile L und R des Restgraphen, der nach dem Löschen verbleibt.
- Jede der beiden Seiten muss einen Nachbarn von v enthalten, sonst wäre v nicht nötig, um die Teile zu trennen (und die Knotenmenge wäre damit kein minimaler Separator)
- Jede Seite, die mehr als einen Knoten enthält, muss sogar zwei Nachbarn von v enthalten, da man sonst durch Ersetzen von v durch den einzigen Nachbarn einen MinCut ohne v konstruieren könnte (Widerspruch zur Annahme).



Knotenzusammenhang (Esfahanian & Hakimi)

Algorithmus 13: Knotenzusammenhang κ (Esfahanian & Hakimi)

Input : Ungerichteter Graph $G = (V, E)$

Output : $\kappa(G)$

$\kappa_{\min} \leftarrow n - 1;$

Wähle $v \in V$ mit minimalem Grad, also $d(v) = \delta(G);$

Seien die Nachbarn $N(v) = \{v_1, v_2, \dots, v_\delta\};$

foreach Nicht-Nachbar $w \in V \setminus (N(v) \cup \{v\})$ **do**

 Berechne $\kappa_G(v, w)$ mit MaxFlow-Prozedur;

$\kappa_{\min} \leftarrow \min\{\kappa_{\min}, \kappa_G(v, w)\};$

$i \leftarrow 1;$

while $i \leq \kappa_{\min}$ **do**

for $j \leftarrow i + 1$ **to** $\delta - 1$ **do**

if $i \geq \delta - 2$ **or** $i > \kappa_{\min}$ **then**

return $\kappa_{\min};$

else if $\{v_i, v_j\} \notin E$ **then**

 Berechne $\kappa_G(v_i, v_j)$ mit MaxFlow-Prozedur;

$\kappa_{\min} \leftarrow \min\{\kappa_{\min}, \kappa_G(v_i, v_j)\};$

Knotenzusammenhang (Esfahanian & Hakimi)

- erste Schleife:
 - ▶ Anzahl der Nicht-Nachbarn kann wieder höchstens $n - \delta - 1$ sein
⇒ höchstens $n - \delta - 1$ MaxFlow-Aufrufe
- zweite Schleife: $\kappa(2\delta - \kappa - 3)/2$
- Gesamtkomplexität: $n - \delta - 1 + \kappa(2\delta - \kappa - 3)/2$

Kantenzusammenhangsalgorithmen

- **Kantenzusammenhang** λ kann in ungerichteten ungewichteten Graphen ebenfalls mit der MaxFlow-Prozedur berechnet werden.
- Ersetze dafür jede ungerichtete Kante durch zwei antiparallele gerichtete Kanten mit Kapazität 1
- Berechne dann lokalen Zusammenhang zwischen der entsprechenden Quelle s und der Senke t .

⇒ Resultierendes Netzwerk ist Unit Capacity Network vom Typ 1

⇒ Komplexität zur Berechnung des lokalen Zusammenhangs für ein Knotenpaar: $\mathcal{O}(m \cdot \min\{m^{1/2}, n^{2/3}\})$

Einfache Kantenzusammenhangsalgorithmen

- Trivialer Algorithmus: mit $n(n - 1)/2$ MaxFlow-Aufrufen den lokalen Kantenzusammenhang aller Knotenpaare berechnen
- Besser: einen Knoten s festzuhalten und dann für alle anderen Knoten t die lokalen Zusammenhangszahlen $\lambda(s, t)$ berechnen

Mindestens einer dieser Knoten muss auf der anderen Seite eines MinCuts liegen.

Deshalb ist das Minimum aller dieser $(n - 1)$ lokalen Zusammenhangszahlen $\lambda(s, t)$ gleich dem (globalen) Kantenzusammenhang λ des Graphen.

Gesamtkomplexität: $\mathcal{O}(nm \cdot \min\{n^{2/3}, m^{1/2}\})$

λ -Covering

- Der Algorithmus funktioniert auch, wenn man statt der ganzen Knotenmenge nur eine Teilmenge verwendet, die zwei Knoten s, t enthält, deren lokaler Zusammenhang $\lambda(s, t)$ gleich dem globalen Zusammenhang λ ist.
 - Eine solche Teilmenge heißt λ -Covering.
- ⇒ Versuche, die Kardinalität dieser Knotenmenge zu reduzieren.

Komponentengröße

Lemma

Sei S ein Minimum Edge-Cut eines Graphen $G = (V, E)$ und sei $L, R \subset V$ eine Partition der Knotenmenge, so dass L and R durch S separiert werden.

Wenn $\lambda(G) < \delta(G)$, dann besteht jede Komponente von $G - S$ aus mehr als $\delta(G)$ Knoten, d.h. es gilt $|L| > \delta(G)$ und $|R| > \delta(G)$.

Komponentengröße

Beweis.

- Seien l_1, \dots, l_k die Elemente von L und sei $E[L] = E(G[L])$ die Menge der durch L induzierten Kanten in G .
- Es gilt:

$$\begin{aligned}\delta_G \cdot k &\leq \sum_{i=1}^k d_G(l_i) \\ &\leq 2 \cdot |E[L]| + |S| \\ &\leq 2 \cdot \frac{k(k-1)}{2} + |S| \\ &< k(k-1) + \delta_G\end{aligned}$$

- Aus $\delta_G \cdot (k-1) < k(k-1)$ folgt $|L| = k > 1$ und $|L| = k > \delta_G$ (sowie $|R| > \delta(G)$).



Komponentengröße

Folgerung

Wenn gilt $\lambda_G < \delta_G$, dann enthält jede Komponente von $G - S$ einen Knoten, der mit keiner Kante in S inzident ist.

Kantenzusammenhang λ

Lemma

Sei $\lambda_G < \delta_G$ und sei T ein **Spannbaum** von G .

Dann enthalten alle Komponenten von $G - S$ mindestens einen Knoten, der kein Blatt in T ist, d.h. die inneren Knoten von T bilden ein λ -Cover.

Beweis.

- Nehmen wir an, dass es nicht so wäre (d.h. alle Knoten in L sind Blätter von T).
- Also für keine Kante von T sind beide Endknoten in L , d.h. $|L| = |S|$.
- Aus dem vorhergehenden Lemma ($\lambda_G < \delta_G \Rightarrow |L| > \delta_G$ und $|R| > \delta_G$) folgt, dass $\lambda_G = |S| = |L| > \delta_G$ (Widerspruch zur Annahme).



Spannbaum-Berechnung (Esfahanian & Hakimi)

Algorithmus von Esfahanian & Hakimi:

- Berechne Spannbaum des gegebenen Graphen.
- Wähle beliebigen inneren Knoten v des Baums.
- Berechne für jeden anderen Knoten w , der kein Blatt ist, den lokalen Kantenzusammenhang $\lambda(v, w)$.
- Das Minimum dieser Wertemenge, zusammen mit δ_G , ergibt genau den Kantenzusammenhang λ_G .
- Ein Baum mit möglichst vielen Blättern wäre vorteilhaft, aber die Konstruktion eines optimalen Baums ist \mathcal{NP} -hart.

Spannbaum-Berechnung (Esfahanian & Hakimi)

Esfahanian & Hakimi:

- Algorithmus zur Berechnung eines Spannbaums T von G , so dass beide Mengen, L und R eines minimalen Kantenseparators mindestens ein Blatt von T enthalten, und nach dem letzten Lemma mindestens einen inneren Knoten.

Spannbaum-Berechnung (Esfahanian & Hakimi)

Algorithmus 14: Spannbaum-Berechnung (Esfahanian & Hakimi)

Input : Ungerichteter Graph $G = (V, E)$

Output : Spannbaum T mit einem Blatt und einem inneren Knoten in L
bzw. R

Wähle $v \in V$;

$T \leftarrow$ alle Kanten inzident mit v ;

while $|E(T)| < n - 1$ **do**

 Wähle ein Blatt w in T , so dass für alle Blätter r in T gilt:
 $|N(w) \cap (V - V(T))| \geq |N(r) \cap (V - V(T))|$;
 $T \leftarrow T \cup \{(w, x) \in E : x \in (V - V(T))\}$

return T ;

Kantenzusammenhang λ (Esfahanian & Hakimi)

Algorithmus 15: Kantenzusammenhang λ (Esfahanian & Hakimi)

Input : Ungerichteter Graph $G = (V, E)$

Output : $\lambda(G)$

Konstruiere einen Spannbaum T (voriger Algorithmus);

Sei P die kleinere der beiden Mengen

(entweder die Blätter oder die inneren Knoten von T);

Wähle einen Knoten $u \in P$;

$c \leftarrow \min\{\lambda_G(u, v) : v \in P \setminus \{u\}\}$;

$\lambda \leftarrow \min(\delta(G), c)$;

return λ ;

Kantenzusammenhang λ (Esfahanian & Hakimi)

- Da P als kleinere der beiden Mengen (Blätter / innere Knoten) gewählt wird, benötigt der Algorithmus höchstens $n/2$ Berechnungen für lokale Zusammenhangszahlen.

⇒ Gesamtzeitkomplexität: $\mathcal{O}(\lambda mn)$

Kantenzusammenhang λ (Matula)

Definition

Ein **Dominating Set** für einen Graphen $G = (V, E)$ ist eine Knotenteilmenge V' von V , so dass jeder Knoten, der nicht in V' ist, mit mindestens einem Knoten in V' über eine Kante verbunden ist.

Lemma

Im Fall $\lambda(G) < \delta(G)$ ist jedes Dominating Set von G auch ein λ -covering von G .

Kantenzusammenhang λ (Matula)

Verbesserter Algorithmus von Matula:

- Analog zur Spannbaum-Methode, wird λ hier berechnet, indem man
 - ▶ ein Dominating Set D von G berechnet,
 - ▶ einen beliebigen Knoten $u \in D$ auswählt und
 - ▶ den lokalen Kantenzusammenhang $\lambda(u, v)$ für alle anderen Knoten $v \in D \setminus \{u\}$ berechnet.
- Das Minimum der Wertemenge (wieder zusammen mit δ_G) ist der Kantenzusammenhang.
- Obwohl die Berechnung eines Dominating Sets minimaler Kardinalität \mathcal{NP} -hart ist, kann man zeigen, dass der Algorithmus in Zeit $\mathcal{O}(mn)$ läuft, wenn man das Dominating Set wie im folgenden Algorithmus konstruiert.

Kantenzusammenhang λ (Matula)

Algorithmus 16: Berechnung eines Dominating Sets (Matula)

Input : Ungerichteter Graph $G = (V, E)$

Output : Dominating Set D

Wähle $v \in V$;

$D \leftarrow \{v\}$;

while $V \setminus (D \cup N(D)) \neq \emptyset$ **do**

 Wähle einen Knoten $w \in V \setminus (D \cup N(D))$;

$D \leftarrow D \cup \{w\}$;

return D ;

k-Kantenzusammenhangskomponenten

David W. Matula: *k*-Components, Clusters, and Slicings in Graphs

- gegeben: ungewichteter, ungerichteter Graph $G = (V, E)$
- Wdh.: Eine **k-Kanten-(Zusammenhangs-)Komponente** von G ist ein maximaler k -kanten-zusammenhängender Teilgraph von G .
- Da wir in diesem Abschnitt nur über *Kantenzusammenhangskomponenten* sprechen, werden diese hier oft einfach als **k-Komponenten** bezeichnet (auch wenn sich das Wort sonst auf *Knotenzusammenhang* bezieht)
- Teilgraphen bestehend aus einem einzelnen isolierten Knoten ($k = 0$) nennen wir **triviale** Komponenten, aber diese werden hier nicht als k -Komponenten angesehen.

Vereinigung von Teilgraphen

Lemma

Seien G_1, G_2, \dots, G_ℓ Teilgraphen von G , so dass ihre Vereinigung $\bigcup_{i=1}^{\ell} G_i$ (einfach) *zusammenhängend* ist.

Dann gilt:

$$\lambda \left(\bigcup_{i=1}^{\ell} G_i \right) \geq \min_{1 \leq i \leq \ell} \{ \lambda(G_i) \}$$

Vereinigung von Teilgraphen

Beweis.

- Wenn $\bigcup_{i=1}^{\ell} G_i$ aus einem einzigen Knoten besteht, gilt die Behauptung (denn beide Seiten sind Null).
 - Sei anderenfalls $C = (A, \bar{A})$ ein MinCut von $\bigcup_{i=1}^{\ell} G_i$.
 - MinCut C muss mindestens eine Kante enthalten, da $\bigcup_{i=1}^{\ell} G_i$ zusammenhängend ist und mindestens 2 Knoten hat.
 - Falls C eine Kante eines Teilgraphen G_j enthält, enthält sowohl A als auch \bar{A} jeweils mindestens einen Knoten aus $V(G_j)$, d.h. C muss einen Cut für G_j enthalten.
- ⇒ Für mindestens ein $j \in \{1, \dots, \ell\}$ gilt: $\lambda\left(\bigcup_{i=1}^{\ell} G_i\right) \geq \lambda(G_j)$



Vereinigung von Teilgraphen

Für die Teilgraphen G_1, G_2, \dots, G_ℓ von G muss jeder Cut des **induzierten** Teilgraphen $G \left[\bigcup_{i=1}^{\ell} V(G_i) \right]$ einen Cut der einfachen Vereinigung der Teilgraphen $\bigcup_{i=1}^{\ell} G_i$ enthalten. Also gilt:

$$\lambda \left(G \left[\bigcup_{i=1}^{\ell} V(G_i) \right] \right) \geq \lambda \left(\bigcup_{i=1}^{\ell} G_i \right)$$

Folgerung

Wenn G_1, G_2, \dots, G_ℓ Teilgraphen von G sind, so dass $\bigcup_{i=1}^{\ell} G_i$ zusammenhängend ist, dann gilt:

$$\lambda \left(G \left[\bigcup_{i=1}^{\ell} V(G_i) \right] \right) \geq \min_{1 \leq i \leq \ell} \{ \lambda(G_i) \}$$

Vereinigung von Teilgraphen

- Achtung:

In der Folgerung kann die Bedingung, dass die einfache Vereinigung $\bigcup_{i=1}^{\ell} G_i$ zusammenhängend ist, nicht durch die abgeschwächte Forderung, dass der induzierte Teilgraph der vereinigten Knotenmengen $G \left[\bigcup_{i=1}^{\ell} V(G_i) \right]$ zusammenhängend ist, ersetzt werden!

Bsp.: wenn G_1 und G_2 zwei disjunkte Kreise sind, die in G durch eine einzelne Kante verbunden sind, ist die Ungleichung in der Folgerung nicht erfüllt.

- Eine weitere Konsequenz des Lemmas ist die Tatsache, dass die k -Kanten-Komponenten jedes Graphen disjunkt sind.

Die Kohäsion / Zusammenhangsfunktion

Definition

Für jedes Element (Knoten oder Kante) $x \in V(G) \cup E(G)$ eines Graphen G ist die **Kohäsion (cohesiveness)** bzw. die **Zusammenhangsfunktion $h(x)$** definiert als maximaler Wert des Kantenzusammenhangs von allen Teilgraphen von G , die x enthalten.

Das Maximum $\sigma(G)$ aller Kohäsionswerte des Graphen G wird als **Stärke (strength)** des Graphen bezeichnet, also

$$\sigma(G) = \max\{\lambda(G') : G' \text{ ist Teilgraph von } G\}.$$

Die Kohäsionsmatrix

- Die Zusammenhangsfunktion kann durch die (symmetrische) **Kohäsionsmatrix** dargestellt werden.
- Zeilen und Spalten werden durch die Knoten des Graphen indiziert
- Eintrag für Position v_i, v_j ist jeweils die Kohäsion der Kante $\{v_i, v_j\}$, falls sie existiert, und sonst Null
- Die Kohäsion eines Knotens ist dann das Maximum der entsprechenden Zeile oder Spalte.
- Die Stärke σ von G ist das Maximum aller Matrixeinträge.

Die Kohäsionsmatrix

- Für Knoten $v \in V(G)$ gilt: $0 \leq h(v) \leq \deg(v)$

- Falls $\{v_i, v_j\} \in E(G)$, gilt:

$$1 \leq h(\{v_i, v_j\}) \leq \min\{\deg(v_i), \deg(v_j)\}$$

- $h(x) = 0$ gilt also nur, wenn x ein isolierter Knoten ist.
- $\sigma(G) = 0$ gilt nur, wenn G keine Kante enthält.
- $\sigma(G) = 1$ gilt genau dann, wenn G ein Wald mit mindestens einer Kante ist, denn jeder Kreis würde $\sigma(G) \geq 2$ implizieren.

Eindeutige Komponentenzuordnung

- Für $x \in V(G) \cup E(G)$ und $h(x) \geq 1$ muss es für jedes $k \in \{1, \dots, h(x)\}$ einen k -kanten-zusammenhängenden Teilgraph geben, der x enthält.
- Insbesondere muss es auch einen maximalen solchen Teilgraph (also eine k -Kanten-Komponente) geben.
- Da die k -Kanten-Komponenten sich nicht überschneiden ist dieser maximale Teilgraph (die Komponente) eindeutig.

Folgerung

Für jeden Graphen G , jedes Element $x \in V(G) \cup E(G)$ und jede Zusammenhangszahl $k \in \{1, \dots, h(x)\}$ existiert eine eindeutige k -Kanten-Zusammenhangskomponente in G , die x enthält.

Eindeutige Komponentenzuordnung

- Für jedes Element x mit $h(x) \geq 1$ gilt:
Unter allen Teilgraphen, die x enthalten und die maximalen Kantenzusammenhang (also $h(x)$) haben, hat die eindeutige $h(x)$ -Komponente, die x enthält, die meisten Knoten.
Sie heißt **$h(x)$ -Komponente selektiert durch x** (Symbol: H_x).
- Die Kohäsion eines Elements kann aus dem Wissen über einen beliebigen Teilgraph maximalen Kantenzusammenhangs, der das Element enthält, abgeleitet werden.
- Aus der Kenntnis der k -Kanten-Komponenten von G für alle k kann man $h(x)$ für jedes Element x bestimmen.
- Aber man kann umgekehrt auch mit Hilfe der Zusammenhangsfunktion die Komponente H_x bestimmen.

Komponentenbestimmung

Satz

Sei x ein Element des Graphen G mit $h(x) \geq 1$.

Sei M_x ein maximaler zusammenhängender Teilgraph von G , der x enthält und dessen Elemente alle Kohäsion mindestens $h(x)$ haben.

Dann gilt $M_x = H_x$.

Beweis.

- Für $x \in V(G) \cup E(G)$ mit $h(x) \geq 1$ sei M_x definiert wie in dem Satz.
- Dann haben für jedes $y \in V(M_x) \cup E(M_x)$ alle Elemente von H_y Kohäsion mindestens $h(y) \geq h(x)$, so dass also gilt $H_y \cup M_x = M_x$, also $H_y \subseteq M_x$

Komponentenbestimmung

Beweis.

- Da jedes Element von M_x in einem H_y ist, gilt:

$$M_x = \bigcup \{H_y : y \in V(M_x) \cup E(M_x)\}$$

- Nach dem Lemma gilt daher

$$\lambda(M_x) \geq h(x)$$

- Da M_x selbst ein Teilgraph von G ist, der x enthält, gilt

$$\lambda(M_x) = h(x)$$

- Damit ist M_x ein $h(x)$ -kanten-zusammenhängender Teilgraph von G und muss in der $h(x)$ -Komponente selektiert durch x enthalten sein, also in H_x .

Komponentenbestimmung

Beweis.

- Da wegen $M_x = \bigcup \{H_y : y \in V(M_x) \cup E(M_x)\}$ der Teilgraph H_x in M_x enthalten sein muss, gilt $M_x = H_x$.
- Der im Satz definierte Teilgraph M_x ist damit eindeutig und kann bestimmt werden, indem man ausgehend von x alle Elemente anhängt, die von x über einen Pfad erreichbar sind, dessen Elemente alle Kohäsion mindestens $h(x)$ aufweisen.



Folgerung

Für jeden Graph G und eine beliebige Zahl $k \in \{1, \dots, \sigma(G)\}$ bilden die Knoten und Kanten von G mit Kohäsion mindestens k einen Graph, dessen Komponenten die k -Komponenten von G sind.

Komponentenbestimmung

- Für jeden Graph G sind die $h(x)$ -Komponenten selektiert durch $x \in V(G) \cup E(G)$ von besonderem Interesse. Es wird nun gezeigt, dass in dieser Menge alle k -Kanten-Komponenten von G (für alle $k \in \{1, \dots, \sigma(G)\}$) enthalten sind.

Folgerung

Wenn G' eine k -Komponente (für ein $k \geq 1$) des Graphen G ist, dann gibt es ein $x \in V(G) \cup E(G)$, so dass $G' = H_x$ ist.

Komponentenbestimmung

Beweis.

- Sei G' eine k -Komponente von G .
- Dann gilt $1 \leq k \leq \lambda(G')$ und G' ist damit auch eine $\lambda(G')$ -Komponente von G .
- Wähle $x \in V(G') \cup E(G')$ so, dass $h(x)$ minimal ist und sei M_x definiert wie im Satz. (Man beachte, dass G' in M_x als Teilgraph enthalten sein muss.)
- $M_x = H_x$ ist eine $h(x)$ -Komponente und deshalb k -kanten-zusammenhängend (da $k \leq \lambda(G') \leq h(x)$).
- Da G' ein maximaler k -kanten-zusammenhängender Teilgraph ist, gilt $G' = H_x$.



Zusammenhangsvielfalt

Definition

Für einen Graphen G sei die **Zusammenhangskomponenten**vielfalt $\eta(G)$ definiert als

$$\eta(G) = |\{H : H \text{ ist eine } k\text{-Komponente von } G \text{ für ein } k \geq 1\}|$$

- Aus dem vorangegangenen Korollar folgt $\eta(G) \leq |V(G)| + |E(G)|$.
- Noch genauer (isolierte Knoten sind keine k -Komponenten):

Satz

Für jeden Graph G gilt:

$$\eta(G) \leq \left\lfloor \frac{|V|}{2} \right\rfloor$$

Zusammenhangsvielfalt

Beweis.

- Für den Beweis wird eine stärkere Ungleichung für zusammenhängende Graphen gezeigt:

$$\eta(G) \leq \left\lfloor \frac{|V(G)| + 1 - \lambda(G)}{2} \right\rfloor$$

- klar für zusammenhängende Graphen auf 1 oder 2 Knoten
- Induktion: angenommen G ist ein zusammenhängender Graph auf $n \geq 3$ Knoten und die Ungleichung gilt für zusammenhängende Graphen auf weniger als n Knoten.
- Falls $\lambda(G) = \sigma(G)$, dann gilt $\eta(G) = 1$ und somit auch die Ungleichung.

Zusammenhangsvielfalt

Beweis.

- Sonst sei G' der Teilgraph von G , den man aus G durch Löschen der Elemente mit Kohäsion $\lambda(G)$ erhält, d.h.
 G' ist ein Graph, dessen Komponenten die $(\lambda(G) + 1)$ -Komponenten von G sind.
 - Die Kanten von G mit Kohäsion $\lambda(G)$ müssen einen Cut von G beinhalten.
- ⇒ G' ist nicht zusammenhängend oder hat weniger Knoten als G .
- In beiden Fällen hat jede der Komponenten G'_1, G'_2, \dots, G'_j von G' weniger Knoten als G .
- ⇒ Ungleichung lässt sich per Induktionsvoraussetzung auf alle G'_i mit $i \in \{1, \dots, j\}$ anwenden.

Zusammenhangsvielfalt

Beweis.

- Ebenso gilt $\lambda(G'_i) \geq \lambda(G) + 1$ für $i \in \{1, \dots, j\}$.
- Eine k -Komponente von G' muss nun eine k -Komponente einer Komponente von G sein und umgekehrt.
- Deshalb gilt

$$\begin{aligned}\eta(G') &= \sum_{i=1}^j \eta(G'_i) \leq \sum_{i=1}^j \left\lfloor \frac{|V(G'_i)| + 1 - \lambda(G'_i)}{2} \right\rfloor \\ &\leq \left\lfloor \frac{|V(G')| - j\lambda(G)}{2} \right\rfloor\end{aligned}$$

- Es gilt $|V(G')| \leq |V(G)| - 1$ oder G' ist nicht zusammenhängend, so dass $j \geq 2$ und $j\lambda(G) \geq \lambda(G) + 1$.

Zusammenhangsvielfalt

Beweis.

- In jedem Fall gilt:

$$\eta(G') \leq \left\lfloor \frac{|V(G)| - 1 - \lambda(G)}{2} \right\rfloor$$

- G selbst ist eine k -Komponente von G für $k = \lambda(G)$, und $\eta(G) = \eta(G') + 1$.
- Damit gilt:

$$\eta(G) \leq \left\lfloor \frac{|V(G)| + 1 - \lambda(G)}{2} \right\rfloor$$

Zusammenhangsvielfalt

Beweis.

Ungleichung aus dem Satz:

- Die Ungleichung aus dem Satz ist für triviale Graphen klar.
- Für nichttriviale *zusammenhängende* Graphen folgt sie aus der verschärften Form.
- Für beliebige Graphen folgt sie aus der Summation über die *nichttrivialen* Komponenten.



Cluster

- Zusammenhangsfunktion jedes Graphen hat ein Plateau über jeder $\sigma(G)$ -Komponente und evt. auch noch an anderen Stellen.
- Plateaus sind Gebiete von (lokal) optimalem Zusammenhang, d.h., k -Komponenten, die völlig disjunkt zu $(k + 1)$ -Komponenten sind.

Definition

Jeder isolierte Knoten und für $k \geq 1$ jede k -Komponente von G , die keine $(k + 1)$ -Komponente enthält, ist ein **Cluster** von G .

- Graph G ist ein Cluster, falls $\sigma(G) = \lambda(G)$.
- Verschiedene Cluster können keine gemeinsamen Knoten enthalten (folgt aus Disjunktheit von k -Komponenten).
- Manche Knoten sind in keinem Cluster.
- Cluster lassen sich aus der Zusammenhangsfunktion bestimmen

Subcluster

Definition

Ein induzierter Teilgraph $K[A]$ eines Clusters K des Graphen G mit $\lambda(K[A]) = \lambda(K)$ heißt **Subcluster** von G .

- Subcluster repräsentieren also induzierte Teilgraphen von lokal maximalem Kantenzusammenhang, die nicht unbedingt inklusions-maximal sind.
- Wenn $G[A]$ und $G[B]$ Subcluster von G sind, für die gilt $A \cap B \neq \emptyset$, dann ist $A \cup B$ ein Subcluster von G .

Cluster und Subcluster

- Die Subcluster eines Graphen bilden unter der Relation “ist echter Teilgraph von” eine partielle Ordnung mit Clustern als maximalen Elementen.
- Die partielle Ordnung der Subcluster in einem bestimmten Cluster ist ein beschränkter Verband.
- Da alle Cluster eines Graphen disjunkt sind, ist die vollständige partielle Ordnung eine Vereinigung von disjunkten beschränkten Verbänden.

Schnitt von Subclustern

Satz

Seien $G[A]$ und $G[B]$ Subcluster eines Clusters K in G , so dass $A \cap B \neq \emptyset$.

Wenn es einen *MinCut* von $G[A] \cup G[B]$ gibt, der

- die Knoten in $A \setminus B$ von den Knoten in $B \setminus A$ separiert und
- der mindestens eine Kante aus $G[A \cap B]$ enthält,

dann ist $G[A \cap B]$ ein Subcluster von K in G .

Schnitt von Subclustern

Beweis.

- Annahme: $C = (W, \bar{W})$ ist MinCut von $G[A] \cup G[B]$ mit $A \setminus B \subset W$ und $B \setminus A \subset \bar{W}$ und mindestens eine Kante von C ist in $G[A \cap B]$
- $\Rightarrow C$ ist auch ein Cut für $G[A]$ und für $G[B]$
- $\lambda(G[A] \cup G[B]) \geq \min\{\lambda(G[A]), \lambda(G[B])\}$ (siehe Lemma über zusammenhängende Vereinigung von Teilgraphen)
- $\Rightarrow \lambda(G[A] \cup G[B]) = \lambda(G[A]) = \lambda(G[B]) = \sigma(K)$
(da $G[A]$ und $G[B]$ Subcluster sind)
- $\Rightarrow |C| = \sigma(K)$ und $C \subset E(G[A \cap B])$ (da jeder Cut von $G[A]$ und $G[B]$ mindestens $\sigma(K)$ Kanten enthält)

Schnitt von Subclustern

Beweis.

- Sei $D = A \cap B$, $X = W \cap D$, $\bar{X} = \bar{W} \cap D$.
- $C = (W, \bar{W})$ ist auch Cut von $G[D]$
- Beweisidee: zeigen, dass jeder Cut von $G[D]$ mindestens $\sigma(K)$ Kanten enthält
- Sei $C_y = (Y, \bar{Y})$ ein Cut von $G[D]$
- Falls $Y \subset X$, dann $C_y = (Y, \bar{Y}) = (Y, \bar{Y} \cup \bar{W})$ ist auch Cut von $G[B]$, also $|C_y| \geq \sigma(K)$
- Ebenso, wenn $Y \subset \bar{X}$, $\bar{Y} \subset X$ oder $\bar{Y} \subset \bar{X}$, dann ist C_y Cut von $G[A]$ oder $G[B]$ und damit $|C_y| \geq \sigma(K)$
- Ansonsten sind die folgenden vier Knotenmengen nicht leer:
 $D_1 = X \cap Y$, $D_2 = X \cap \bar{Y}$, $D_3 = \bar{X} \cap Y$, $D_4 = \bar{X} \cap \bar{Y}$
- $\bigcup_{i=1}^4 D_i = D = A \cap B$

Schnitt von Subclustern

Beweis.

- Da $G[B]$ Subcluster von K ist, enthält der Cut zwischen D_1 und $B \setminus D_1$ in $G[B]$ mindestens $\sigma(K)$ Kanten die alle zwischen D_1 einerseits und D_2, D_3 und D_4 andererseits verlaufen:

$$|(D_1, B \setminus D_1)| = |(D_1, D_2)| + |(D_1, D_3)| + |(D_1, D_4)| \geq \sigma(K)$$

Ebenso für D_2 und $B \setminus D_2$ in $G[B]$:

$$|(D_2, B \setminus D_2)| = |(D_1, D_2)| + |(D_2, D_3)| + |(D_2, D_4)| \geq \sigma(K)$$

- Addition ergibt:

$$2|(D_1, D_2)| + |(D_1, D_3)| + |(D_1, D_4)| + |(D_2, D_3)| + |(D_2, D_4)| \geq 2\sigma(K)$$

$$\Rightarrow 2|(D_1, D_2)| + |(D_1 \cup D_2, D_3 \cup D_4)| \geq 2\sigma(K)$$

Schnitt von Subclustern

Beweis.

- $(D_1 \cup D_2, D_3 \cup D_4) = (X, \bar{X}) = C$
 - also $|(D_1, D_2)| \geq \sigma(K)/2$ und ebenso $|(D_3, D_4)| \geq \sigma(K)/2$
- $\Rightarrow |C_y| = |(Y, \bar{Y})| \geq |(D_1, D_2)| + |(D_3, D_4)| \geq \sigma(K)$
- \Rightarrow Jeder Cut von $G[A \cap B]$ hat mindestens $\sigma(K)$ Kanten
- $\Rightarrow G[A \cap B]$ ist Subcluster von Cluster K des Graphen G



Slicings

Definition

Die geordnete Partition der Kanten des Graphen G , $Z = (C_1, C_2, \dots, C_m)$, ist ein **Slicing** von G falls für jedes Element C_i gilt:

$$C_i \text{ ist ein Cut } (A_i, \bar{A}_i) \text{ von } \begin{cases} G & \text{für } i = 1 \\ G - \bigcup_{j=1}^{i-1} C_j & \text{für } i \in \{2, \dots, m\} \end{cases}$$

Ein Element C_i des Slicings heißt auch **Cut des Slicings**.

Anmerkung: m ist hier nicht die Kantenanzahl.

Minimale und Narrow Slicings

Definition

Ein Slicing $Z = (C_1, C_2, \dots, C_m)$ von G , für das es keine echte Unterpartition gibt, die ein Slicing von G ist, ist ein **minimales Slicing** von G .

- Jeder Cut C_i eines minimalen Slicings muss ein minimaler Cut einer Komponente von $G - \bigcup_{j=1}^{i-1} C_j$ sein.

Definition

Z ist ein **Narrow Slicing** von G , falls jeder Cut C_i ein Minimum Cut einer Komponente von $G - \bigcup_{j=1}^{i-1} C_j$ ist.

Minimale und Narrow Slicings

Dynamische Interpretation:

- Slicing ist Sequenz von nicht-leeren Cuts, die G in isolierte Knoten teilen
- Minimale / Narrow Slicings verwenden nur Minimale / Minimum Cuts in jedem Schritt.
- Jedes Narrow Slicing ist auch ein Minimales Slicing (was umgekehrt nicht gilt).

Slicings

- nützlich: Slicing als sukzessive Zerlegung der Knotenmenge
- i -ter Cut $C_i = (A_i, \bar{A}_i)_i$ ist Cut von $G - \bigcup_{j=1}^{i-1} C_j$
- Da $A_i \cup \bar{A}_i = V(G - \bigcup_{j=1}^{i-1} C_j) = V(G)$, betrifft $(A_i, \bar{A}_i)_i$ die gleiche Partition der Knoten von G wie der Cut (A_i, \bar{A}_i) von G .
- Es gilt

$$(A_i, \bar{A}_i)_i = (A_i, \bar{A}_i) - \bigcup_{j=1}^{i-1} C_j$$

- Also enthält $(A_i, \bar{A}_i)_i$ nicht unbedingt alle Kanten des Cuts (A_i, \bar{A}_i) von G für $i > 1$.
- Beachte: Kantenmenge (A_i, \bar{A}_i) hängt implizit von G ab, Kantenmenge $(A_i, \bar{A}_i)_i$ hängt implizit von G und Z ab. Die Knotenpartition ist jedoch explizit und bei beiden gleich!

Knotenpartitionen

- Repräsentation der Cuts des Slicings als Knotenpartition von $V(G)$ bietet eine nützliche Charakterisierung von jedem Graphen $G - \bigcup_{j=1}^{i-1} C_j$ als Vereinigung von induzierten Teilgraphen von G
- Da C_1 die Knotenmenge A_1 von \bar{A}_1 separiert, und damit $G - C_1 = G[A_1] \cup G[\bar{A}_1]$, gilt somit:

$$G - (C_1 \cup C_2) = G[A_1 \cap A_2] \cup G[A_1 \cap \bar{A}_2] \cup G[\bar{A}_1 \cap A_2] \cup G[\bar{A}_1 \cap \bar{A}_2]$$

Satz

Induktiv folgt: Sei $Z = (C_1, C_2, \dots, C_m)$ ein Slicing von G mit $C_i = (A_i, \bar{A}_i)_i$ für $i \in \{1, \dots, m\}$. Dann gilt für $j \in \{1, \dots, m\}$:

$$G - \bigcup_{k=1}^j C_k = \bigcup \left\{ G \left[\bigcap_{i \in s} A_i \cap \bigcap_{i \notin s} \bar{A}_i \right] : s \subset \{1, 2, \dots, j\} \right\}$$

Knotenpartitionen

- Jeder Graph G kann geschrieben werden als $G = G[P_1] \cup G[P_2] \cup \dots \cup G[P_n]$, wobei jedes $G[P_i]$ eine Komponente von G ist.
- Sei dann $\{P_1, P_2, \dots, P_n\}$ die Komponenten-Knoten-Partition von G .
- Falls G nicht zusammenhängend ist, können die induzierten Teilgraphen auf der rechten Seite im letzten Satz unzusammenhängend sein.
- Sei $Z = (C_1, C_2, \dots, C_m)$ ein Slicing von G , wobei G die Komponenten-Knoten-Partition $\{P_1, P_2, \dots, P_n\}$ hat. Dann ist die Komponenten-Knoten-Partition von $G - \bigcup_{k=1}^j C_k =$

$$\bigcup \left\{ G \left[P_\ell \cap \bigcap_{i \in s} A_i \cap \bigcap_{i \notin s} \bar{A}_i \right] : 1 \leq \ell \leq n, s \subset \{1, 2, \dots, j\} \right\}$$

Knotenpartitionen

- Beachte: die Komponenten-Knoten-Partition von $G - \bigcup_{k=1}^j C_k$ ist eine Subpartition der Komponenten-Knoten-Partition von $G - \bigcup_{k=1}^{j-1} C_k$ für $j \in \{1, \dots, m\}$.
- ⇒ Slicing $Z = (C_1, C_2, \dots, C_m)$ bewirkt eine geschachtelte Sequenz von $m + 1$ Knotensubpartitionen von der Komponenten-Knoten-Partition $\{P_1, P_2, \dots, P_n\}$ bis hinunter zur minimalen Partition bestehend aus lauter einzelnen Knoten.
- Cut C_i des Slicings $Z = (C_1, C_2, \dots, C_m)$ kann einige Komponenten von $G - \bigcup_{j=1}^{i-1} C_j$ intakt lassen.
- ⇒ sinnvoll: spezifizieren, welche Komponenten durch C_i wirklich zertrennt werden

Zertrennte Teilgraphen des Slicings

- Die Subgraphen G_1, G_2, \dots, G_m von G sind die **durch das Slicing $Z = (C_1, C_2, \dots, C_m)$ zertrennten Teilgraphen**, wenn jedes G_i genau die Komponenten von $G - \bigcup_{j=1}^{i-1} C_j$ enthält, deren Knoten durch C_i separiert werden.
- Jeder vom Slicing Z zertrennte Teilgraph G_i ist dann eine Vereinigung von induzierten Teilgraphen von G .
- Da ein minimales Slicing nur sukzessive Cuts von individuellen Komponenten beinhaltet, gilt folgendes Korollar:

Knotenpartitionen

Folgerung

Die durch ein *minimales Slicing* $Z = (C_1, C_2, \dots, C_m)$ zertrennten Teilgraphen sind alle zusammenhängende induzierte Teilgraphen von G .

- ⇒ Die geschachtelte Sequenz von Komponenten-Knoten-Partitionen für ein minimales Slicing ist derart, dass jede Subpartition aus der vorhergehenden durch Aufspaltung von **genau einem** Teil hervorgeht.
- ⇒ entspricht demzufolge einer maximalen Kette im Verband der Komponenten-Knoten-Partitionen

Länge von Slicings

Definition

Sei die **Länge** $\ell(Z)$ eines Slicings Z des Graphen G die Anzahl der Cuts des Slicings (also die Kardinalität der Kantenpartition Z).

- Jeder Cut des Slicings erhöht die Anzahl der Zusammenhangskomponenten im Graphen.
- Diese Erhöhung ist genau dann immer gleich Eins, wenn die Cuts minimal sind (bzw. das Slicing ein minimales Slicing ist).

Satz

Für jeden Graphen G mit $|E(G)| \geq 1$ gilt:

$$\max\{\ell(Z) : Z \text{ ist Slicing von } G\} = |V(G)| - \#\text{Komponenten}(G)$$

und dieses Maximum wird genau von den minimalen Slicings erreicht.

Länge von Slicings

- Länge eines Slicings Z von Graph G kann 1 sein, falls G bipartit ist.
- Allgemein:
Minimaler Wert für die Länge eines Slicings eines Graphen hängt von der minimalen Zahl k ab, so dass G ein k -partiter Graph ist, also von der sogenannten **chromatischen Zahl** $\chi(G)$ des Graphen G .

Definition

Ein Graph ist k -partit, wenn die Knotenmenge $V(G)$ in k Mengen V_1, \dots, V_k partitioniert werden kann, so dass jeder induzierte Graph $G[V_i]$ ($i \in \{1, \dots, k\}$) keine Kante enthält.

Länge von Slicings

Satz

Für jeden Graph G mit $|E(G)| \geq 1$ gilt:

$$\min\{\ell(Z) : Z \text{ ist Slicing von } G\} = \lceil \log_2 \chi(G) \rceil$$

Beweis.

- Sei $Z = (C_1, C_2, \dots, C_m)$ ein Slicing von G mit $C_i = (A_i, \bar{A}_i)_i$ und $A_i \cup \bar{A}_i = V$.
- Für $v \in V$ und $i \in \{1, \dots, m\}$ sei $b_i(v) = \begin{cases} 1 & \text{falls } v \in A_i \\ 0 & \text{falls } v \notin A_i \end{cases}$
- Partitioniere Knotenmenge V in die 2^m Teilmengen V_0, \dots, V_{2^m-1} , indem $v \in V$ der Teilmenge V_k mit $k = \sum_{i=1}^m b_i(v) \cdot 2^{i-1}$ zugeteilt wird.

Länge von Slicings

Beweis.

- Falls $\{u, w\} \in E$, dann ist $\{u, w\} \in C_i$ für ein i , so dass also Knoten u und w durch Cut C_i getrennt werden.
- $\Rightarrow b_i(u) \neq b_i(w)$
- $\Rightarrow u$ und w sind in verschiedenen Teilmengen V_k
- \Rightarrow Die nichtleeren unter den Teilmengen V_0, \dots, V_{2^m-1} bilden eine Partition von V in unabhängige Mengen (independent sets), also $2^m \geq \chi(G)$ bzw. $\ell(Z) = m \geq \lceil \log_2 \chi(G) \rceil$.
- Damit wurde erstmal eine untere Schranke gezeigt.
 - Als nächstes wird gezeigt, dass der Wert $\lceil \log_2 \chi(G) \rceil$ auch tatsächlich durch ein Slicing Z^* erreicht werden kann.

Länge von Slicings

Beweis.

- Sei $V_0, \dots, V_{\chi(G)-1}$ eine Partition von V in $\chi(G)$ unabhängige Mengen (independent sets) und sei $m = \lceil \log_2 \chi(G) \rceil$.
- Sei A_i die Vereinigung derjenigen V_k , deren Binärdarstellung den Term 2^{i-1} enthält, also

$$A_i = \bigcup \{ V_k : \lfloor k/2^{i-1} \rfloor \equiv 1 \pmod{2} \} \quad \text{für } i = \{1, \dots, m\}$$

und $\bar{A}_i = V \setminus A_i$

- Sei wie zuvor $C_1 = (A_1, \bar{A}_1)$ und $C_i = (A_i, \bar{A}_i)_i = (A_i, \bar{A}_i) - \bigcup_{j=1}^{i-1} C_j$ für $i = 2, \dots, m$.
- Für jedes $i = 1, 2, \dots, m$ muss es eine Kante $\{u, w\} \in E$ mit $u \in V_0$, $w \in V_{2^i-1}$ geben, da sonst $V_0 \cup V_{2^i-1}$ eine unabhängige Menge ist (Widerspruch zur chromatischen Zahl $\chi(G)$)

Länge von Slicings

Beweis.

- Also gilt: $u \in \bar{A}_j$ für $j = 1, \dots, m$,
sowie $w \in \bar{A}_j$ für $j = 1, \dots, i-1$ und $w \in A_i$
- $\Rightarrow \{u, w\} \in (A_i, \bar{A}_i)_i = C_i$
- \Rightarrow kein C_i ist leer
- Für $\{u, w\} \in E$ gilt außerdem $u \in V_k$ und $w \in V_j$ für ein Paar $k \neq j$.
- Sei i ein Index, so dass die Binärdarstellungen von k und j sich im Term 2^{i-1} unterscheiden.
- $\Rightarrow u \in A_i$ und $w \in \bar{A}_i$ oder $u \in \bar{A}_i$ und $w \in A_i$
- $\Rightarrow \{u, w\} \in (A_i, \bar{A}_i) = \bigcup_{n=1}^i C_n$
- $\Rightarrow E \subseteq \bigcup_{i=1}^m C_i$ und $Z^* = (C_1, C_2, \dots, C_m)$ ist Slicing von G mit $\ell(Z^*) = m = \lceil \log_2 \chi(G) \rceil$



Weite von Slicings

Definition

Die **Weite** eines Slicings Z des Graphen G sei

$$w(Z) = \max\{|C| : C \text{ ist ein Cut des Slicings } Z\}$$

Jeder Cut C von Z mit $|C| = w(Z)$ heißt **Wide Cut** des Slicings Z .

- Ein MinCut bezieht sich auf den Graphen, während ein Wide Cut sich auf ein bestimmtes Slicing bezieht.

Weite von Slicings

- Da jeder Cut $C_i = (A_i, \bar{A}_i)_i$ eines Slicings Z von G in einem Cut (A_i, \bar{A}_i) von G enthalten ist, und da jedes Slicing mit einem beliebigem Cut von G beginnen kann, muss die maximale Weite eines Cuts gleich der maximalen Anzahl von Kanten in einem beliebigen Cut sein.

Satz

Für jeden Graphen mit $|E(G)| \geq 1$ gilt:

$$\max\{w(Z) : Z \text{ ist Slicing von } G\} = \max\{|C| : C \text{ ist Cut von } G\}$$

Weite von Slicings

- Die minimale Weite eines Slicings ist ähnlich verwandt zu MinCuts, aber nicht vom ganzen Graphen G .

Satz

Für jeden Graphen mit $|E(G)| \geq 1$ gilt:

$$\min\{w(Z) : Z \text{ ist Slicing von } G\} = \sigma(G)$$

⇒ Dualität zwischen Slicings und Subgraphen:

$$\min_Z \max_C \{|C| : C \text{ ist Cut des Slicings } Z \text{ von } G\} = \max_{G'} \min_C \{|C| : C \text{ ist Cut des Teilgraphen } G' \text{ von } G\}$$

Weite von Slicings

Beweis.

(Ungleichung in beiden Richtungen)

$$\min\{w(Z) : Z \text{ ist Slicing von } G\} \geq \max\{\lambda(G') : G' \text{ ist Teilgraph von } G\} = \sigma(G):$$

- Sei K ein Cluster von G mit $\lambda(K) = \sigma(K)$.
- Dann gilt für jedes Slicing Z von G , dass es einen ersten Cut C_i gibt, der Knoten von K separiert.
- Dann muss C_i einen Cut für K enthalten, so dass $|C_i| \geq \lambda(K) = \sigma(G)$.
- Also gilt $w(Z) \geq \sigma(Z)$ für jedes Slicing Z .

Weite von Slicings

Beweis.

$$\min\{w(Z) : Z \text{ ist Slicing von } G\} \leq \max\{\lambda(G') : G' \text{ ist Teilgraph von } G\} = \sigma(G):$$

- Sei $Z^* = (C_1, \dots, C_m)$ ein Narrow Slicing von G mit Wide Cut C_i .
- Sei G_i der Teilgraph von G , der durch C_i zertrennt wird.
- Da ein Narrow Slicing nur Minimum Cuts benutzt, folgt $\lambda(G_i) = w(Z^*)$.



Weite von Slicings

- Die Weite eines Slicings muss größer oder gleich der durchschnittlichen Anzahl der Kanten in den Cuts des Slicings sein.
- Aus dem letzten Satz und dem Satz über die maximale Länge eines Slicings folgt damit die folgende untere Schranke für die Stärke $\sigma(G)$:

Folgerung

Für jeden Graphen mit $|E(G)| \geq 1$ gilt:

$$\sigma(G) \geq \frac{|E(G)|}{|V(G)| - 1} > |E(G)| / |V(G)|$$

Narrow Slicings

- Im Beweis des letzten Satzes war jedes Narrow Slicing ausreichend, um die Ungleichung der zweiten Richtung zu beweisen. Deshalb gelten folgende Korollare:

Folgerung

Für jedes Narrow Slicing Z^ von G gilt: $w(Z^*) = \sigma(G)$.*

Folgerung

Jeder Wide Cut eines Narrow Slicings von G ist ein Minimum Cut eines Subclusters von G .

Narrow Slicings und k -Komponenten

Die nächsten zwei Folgerungen zeigen, dass man ein Narrow Slicing benutzen kann, um die k -Komponenten ($k \geq 1$) und die Zusammenhangsfunktion zu berechnen.

Folgerung

Seien $G[A_1], G[A_2], \dots, G[A_m]$ die Teilgraphen, die durch das Narrow Slicing $Z = (C_1, C_2, \dots, C_m)$ getrennt werden.

Dann ist jede k -Komponente von G gleich einem $G[A_i]$ für ein $i \in \{1, \dots, m\}$.

Weiterhin ist ein $G[A_i]$ ($i \in \{1, \dots, m\}$) genau dann eine k -Komponente von G , wenn

$$A_j \supset A_i \text{ für } j < i \Rightarrow |C_j| < |C_i|.$$

Und solch ein $G[A_i]$ ist genau dann ein Cluster von G , wenn auch gilt

$$A_j \subset A_i \text{ für } j > i \Rightarrow |C_j| \leq |C_i|.$$

Narrow Slicings und k -Komponenten

Beweis.

- Seien $G[A_1], \dots, G[A_m]$ die Teilgraphen, die durch das Narrow Slicing $Z = (C_1, C_2, \dots, C_m)$ getrennt werden, so dass $\lambda(G[A_i]) = |C_i|$ ($i \in \{1, \dots, m\}$).
 - Sei H eine k -Komponente von G für ein $k \in \{1, \dots, \sigma(G)\}$.
 - Dann muss der erste Cut C_n aus Z , der Knoten aus H separiert, einen Cut von H enthalten, also $|C_n| = \lambda(G[A_n]) \geq \lambda(H)$.
 - Da H zusammenhängend ist, muss H ein Teilgraph von $G[A_n]$ sein, weil C_n der erste Cut ist, der Knoten aus H separiert.
- $\Rightarrow G[A_n]$ ist ein $\lambda(H)$ -kanten-zusammenhängender Graph ($\lambda(H) \leq \lambda(G[A_n])$).
- Da H ein maximaler $\lambda(H)$ -kanten-zusammenhängender Graph (per Def.) sowie ein Teilgraph von $G[A_n]$ ist, gilt $H = G[A_n]$.

Narrow Slicings und k -Komponenten

Beweis.

- $G[A_i]$ ($i \in \{1, \dots, m\}$) ist in einer $\lambda(G[A_i])$ -Komponente H von G enthalten, und aufgrund des vorangegangenen Arguments gilt $H = G[A_j]$ für ein $j \leq i$.
 - Also ist $G[A_i]$ selbst eine $\lambda(G[A_i])$ -Komponente von G genau dann, wenn $A_j \supset A_i$ für $j < i$ impliziert, dass $|C_j| < |C_i|$.
 - Falls weiterhin $G[A_i]$ eine $\lambda(G[A_i])$ -Komponente ist, dann handelt es sich genau dann um ein Cluster, wenn $A_j \subset A_i$ für $j > i$ impliziert, dass $|C_j| \leq |C_i|$.
- ⇒ Die Teilgraphen, die durch ein beliebiges Narrow Slicing von G zertrennt werden, enthalten alle k -Komponenten von G , und damit alle Cluster von G , die keine isolierten Knoten sind.



Narrow Slicings und Kohäsionsfunktion

Folgerung

Seien $G[A_1], G[A_2], \dots, G[A_m]$ die Teilgraphen, die durch das Narrow Slicing $Z = (C_1, C_2, \dots, C_m)$ getrennt werden.

Dann gilt für jedes Graphenelement $x \in V(G) \cup E(G)$:

$$h(x) = \begin{cases} 0, & \text{falls } x \text{ ein isolierter Knoten in } G \text{ ist,} \\ \max_i \{|C_i| : x \in A_i \cup E(G[A_i])\}, & \text{sonst} \end{cases}$$

- Wenn $G[A_1], \dots, G[A_m]$ die Subgraphen sind, die durch ein Narrow Slicing $Z = (C_1, \dots, C_m)$ von G zertrennt werden, dann müssen nicht alle m Subgraphen $G[A_i]$ k -Komponenten sein.
- Insbesondere, wenn $G[A_i]$ ein Cluster mit $\lambda(G[A_i]) \geq 2$ ist, dann sind mindestens $\lambda(G[A_i]) - 1$ der Teilgraphen $G[A_j]$ mit $j > i$ echte Teilgraphen von $G[A_i]$ und können demzufolge keine k -Komponenten von G für ein $k \geq 1$ sein.

Berechnung der Zusammenhangsfunktion

- Der Algorithmus basiert auf der sequentiellen Bestimmung von globalen Minimum Cuts für höchstens $|V(G)| - 1$ induzierte Teilgraphen von G .

Narrow Slicing Algorithmus

Algorithmus 17: Berechnung eines Narrow Slicings (Matula)

Input : Graph $G = (V, E)$ mit N Komponenten und mindestens einer Kante

Output : Ein Narrow Slicing Z

Repräsentiere G als Vereinigung seiner Komponenten $G = \bigcup_{j=1}^N G[P_{1,j}]$;

$i \leftarrow 1$;

while $i < |V(G)| - N$ **do**

Wähle $G_i = G[P_{i,k}]$ für ein k , so dass $|P_{i,k}| \geq 2$;

Finde einen MinCut $C_i = (A, \bar{A})$ von G_i . (Beachte $A \cup \bar{A} = V(G_i)$);

Definiere neue Komponenten-Knoten-Partition $\{P_{i+1,j}\}$ ($j \in \{1, \dots, N+i\}$):

$$P_{i+1,j} = \begin{cases} P_{i,j} & \text{für } 1 \leq j < k, \\ A & \text{für } j = k, \\ \bar{A} & \text{für } j = k + 1, \\ P_{i,j-1} & \text{für } k + 1 < j \leq N + i, \end{cases}$$

$$\text{mit } G - \bigcup_{n=1}^i C_n = \bigcup_{j=1}^{N+i} G[P_{i+1,j}];$$

$i \leftarrow i + 1$;

$G - \bigcup_{n=1}^{|V(G)|-N} C_n$ besteht jetzt nur noch aus isolierten Knoten und

$Z = (C_1, \dots, C_{|V(G)|-N})$ ist ein Narrow Slicing;

Narrow Slicing Algorithmus

- Ursprünglich wurde von Matula vorgeschlagen, den MinCut mit Hilfe des Cut Trees von Gomory/Hu zu berechnen.

Hier kann man aber auch andere Algorithmen einsetzen, z.B. den Stoer/Wagner-Algorithmus.

- Der Narrow Slicing Algorithmus bestimmt explizit ein Narrow Slicing $Z = (C_1, \dots, C_{|V(G)|-N})$, eine Liste $G_1, \dots, G_{|V(G)|-N}$ von induzierten Graphen, die durch das Slicing zertrennt werden, sowie die durch Z verursachte geschachtelte Sequenz von Komponenten-Knoten-Partitionen.
- Daraus können dann entsprechend den früheren Sätzen sehr einfach die Zusammenhangsfunktion, die k -Komponenten und die Cluster von G bestimmt werden.

Narrow Slicing Algorithmus

- $G_i = G[P_{i,k}]$ hat höchstens $|V(G)| + 2 - N - i$ Knoten.
- Also müssten bei Verwendung der Cut Tree Methode höchstens

$$\sum_{i=1}^{|V(G)|-N} (|V(G)| + 1 - N - i) = \frac{1}{2}(|V(G)| - N)(|V(G)| - N + 1)$$

Flussprobleme einfacher Art gelöst werden.

- Für einen zusammenhängenden Graphen wären das höchstens $\binom{|V(G)|}{2}$ Flussprobleme.
- Matula gibt die Komplexität grob mit n^4 bis n^5 an.
- Das entspricht auch ungefähr der Größenordnung, die man bei Verwendung des Stoer/Wagner-Algorithmusses errechnen würde ($\mathcal{O}(m_i n_i + n_i^2 \log n_i)$ pro Teilproblem mit n_i Knoten und m_i Kanten, summiert für $n_i = n - 1, \dots, 1$)