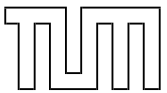
Fortgeschrittene Netzwerk- und Graph-Algorithmen

Dr. Hanjo Täubig

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

Wintersemester 2007/08



Übersicht

Eigenvektor-Zentralität von Bonacich

- Phillip Bonacich, 1972
- basiert auf Eigenvektoren der Adjazenzmatrix eines einfachen, ungerichteten, zusammenhängenden, ungewichteten Graphen
- 3 verschiedene Methoden, deren Werte sich nur um einen konstanten Faktor unterscheiden
  - $\mathbf{s}^a$ Faktor-Analyse
  - $\mathbf{s}^b$ Konvergenz einer unendlichen Reihe
  - $\mathbf{s}^c$ Lösen eines linearen Gleichungssystems (LGS)
- Modellierung:
  - Freundschaftsnetzwerk
  - Fähigkeit, Freunde zu finden
  - $\mathbf{s}^a \in \mathbb{R}^n$, so dass der i-te Eintrag $\mathbf{s}^a_i$ das Freundschaftspotential des Knotens i darstellt

Eigenvektor-Zentralität von Bonacich: Faktor-Analyse

- Produkt der Potentiale zweier Knoten i und j, also $\mathbf{s}_i^a \mathbf{s}_j^a$ soll möglichst nah beim Eintrag der Adjazenzmatrix a_{ij} liegen.

$\Rightarrow$ Minimiere

$$\sum_{i=1}^{n} \sum_{j=1}^{n} (\mathbf{s}_i^a \mathbf{s}_j^a - a_{ij})^2$$

Eigenvektor-Zentralität von Bonacich: unendliche Reihe

- Für gegebenes $\lambda_1 \neq 0$ definiere

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

Satz

Sei $A \in \mathbb{R}^{n\times n}$ eine symmetrische Matrix und λ_1 der größte Eigenwert von A, dann konvergiert

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

gegen einen Eigenvektor von A mit Eigenwert λ_1.

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

Eigenvektor-Zentralität von Bonacich: LGS

- Berechnung eines Eigenvektors eines LGS
- Definiere Zentralität als Summe der Zentralitäten der Nachbarn:

$$\mathbf{s}_i^c = \sum_{j=1}^{n} a_{ij}\mathbf{s}_j^c \quad \text{bzw.} \quad \mathbf{s}^c = A * \mathbf{s}^c$$

- Löse $\lambda\mathbf{s} = A\mathbf{s}$
- Exakt ein Eigenvektor hat von Null verschiedene Werte mit gleichem Vorzeichen

Eigenvektor-Zentralität von Bonacich: LGS

Satz

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

- *Der größte Eigenwert λ_1 ist ein einfacher Eigenwert. (Ihm ist nur ein Eigenvektor zugeordnet.)*
- *Alle Einträge des Eigenvektors zu λ_1 sind ungleich Null und haben das gleiche Vorzeichen.*

(folgt aus Theorem von Perron/Frobenius über nichtnegative quadratische Matrizen)

Eigenvektor-Zentralität: Definition

Die drei Varianten unterscheiden sich nur durch konstante Faktoren
Allgemeine Definition der Eigenvektor-Zentralität

$$\mathbf{c_{EV}} = \frac{|\mathbf{s^c}|}{||\mathbf{s^c}||}$$

Weitere Feedback-Zentralitäten

- Hubbell-Index (Charles Hubbell, 1965)
  gewichteter gerichteter Graph
  Zentralität eines Knotens hängt von der gewichteten Summe seiner Nachbarn ab
- Verhandlungszentralität von Bonacich
  ungewichtete gerichtete Graphen
  Zentralität eines Knotens hoch, wenn seine Nachbarn keine anderen Optionen haben
  ($\Rightarrow$ berücksichtigt *negative* Rückkopplung)
- WebGraph-Zentralitäten
  - PageRank (nur Topologie)
  - Hubs & Authorities (HITS)
  - SALSA (konzeptuell eine Kombination der beiden anderen)

Kürzeste Pfade: SSSP / Dijkstra

Algorithmus 1 : Dijkstra-Algorithmus (SSSP)

Input : $G = (V, E), \quad \omega : E \to \mathbb{R}, \quad s \in V$

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

$P = \emptyset, T = V;$

$d(s, v) = \infty$ for all $v \in V; \quad d(s, s) = 0; \quad pred(s) = 0;$

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

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

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

  for $w \in N(v)$ **do**

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

      $d(s, w) := d(s, v) + \omega(v, w);$

      $pred(w) := v;$

Kürzeste Pfade: SSSP / Dijkstra

- Datenstruktur: Prioritätswarteschlange
  (z.B. Fibonacci Heap: amortisierte Komplexität $\mathcal{O}(1)$ für `insert` und `decreaseKey`, $\mathcal{O}(\log n)$ `deleteMin`)
- Komplexität:
  - $n-1$ `insert`
  - $n-1$ `deleteMin`
  - $O(m)$ `decreaseKey`

  $\Rightarrow \mathcal{O}(m + n \log n)$
- aber: nur für nichtnegative Kantengewichte, sonst Bellman/Ford-Algorithmus in $\mathcal{O}(mn)$

Kürzeste Pfade: APSP / Floyd-Warshall

Algorithmus 2 : Floyd-Warshall APSP Algorithmus

Input : Graph $G = (V, E)$, edge weights $\omega : E \longrightarrow R$
Output : Shortest path distances $d(u, v)$ between all $u, v \in V$

$d(u, v) = \infty, pred(u, v) = 0$ for all $u, v \in V$;
$d(v, v) = 0$ for all $v \in V$;
$d(u, v) = \omega(u, v), pred(u, v) = u$ for all $\{u, v\} \in E$;
for $v \in V$ **do**
  for $\{u, w\} \in V \times V$ **do**
    if $d(u, w) > d(u, v) + d(v, w)$ **then**
      $d(u, w) := d(u, v) + d(v, w)$;
      $pred(u, w) := pred(v, w)$;

Kürzeste Pfade: APSP / Floyd-Warshall

- Komplexität: $\mathcal{O}(n^3)$

Betweenness Centrality

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

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 Dijkstras Algorithmus
   - Ersetze einzelnen Vorgängerknoten durch eine Menge von Vorgängern $\text{pred}(s, v)$
   - Es gilt dann

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

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 $\sigma_{st}(v) = 0$

$\Rightarrow$ $\mathcal{O}(n^2)$ pro Knoten v (Summation über alle $s \neq v \neq t$), also insgesamt $\mathcal{O}(n^3)$

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:

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

Betweenness von v:

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

Betweenness Centrality (Brandes)

Satz

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

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

Betweenness Centrality (Brandes)

Beweis.

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

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

- Es ergibt sich

$$\delta_{s*}(v) = \sum_{t \in V} \delta_{st}(v) = \sum_{t \in V} \sum_{w:\ v \in \text{pred}(s,w)} \delta_{st}(v, \{v, w\})$$

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.

Vertausche die Summationsreihenfolge:

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

□

Betweenness Centrality (Brandes)

- Berechne n kürzeste-Wege-DAGs
  (einen für jeden Startknoten $s \in V$)
- Berechne nacheinander für alle $s \in V$ aus dem kürzeste-Wege-DAG von s die Abhängigkeiten $\delta_{s*}(v)$ für alle anderen Knoten $v \in V$
  Vorgehen: rückwärts, von den Blättern im kürzeste-Wege-Baum bzw. von der entferntesten Schicht im kürzeste-Wege-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 gewichtete Graphen in Zeit*
  $\mathcal{O}(n(m + n \log n)) = \mathcal{O}(nm + n^2 \log n)$
- *für ungewichtete Graphen in $\mathcal{O}(mn)$*

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

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

Shortcut-Werte

- Ziel: Berechnung der shortcut-Werte aller Kanten in einem gerichteten Graph
- Maximale Erhöhung der Länge eines kürzesten Pfades durch Entfernen einer Kante $e = (u, v) \in E$
- Berechnung der Distanz von u zu v in $G_e = (V, E \setminus \{e\})$ für alle $e = (u, v) \in E$ (denn die maximale Erhöhung betrifft ja die Endknoten der Kante)

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

Shortcut-Werte

- Annahme: keine negativen Kreise
  ($\Rightarrow$ $d(i,j)$ ist definiert für alle Knotenpaare (i,j)),
  keine parallelen Kanten
- Idee: ein Aufruf für Knoten u berechnet shortcut-Werte für alle ausgehenden Kanten
- Vorgehen:
  - Fixiere einen Knoten 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 $\tau_i = bot$)

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 Kante (u, v) ist nicht einziger kürzester Pfad)
- Ansonsten, falls $\tau_v = v$, ist die neue Distanz β_v
- $\alpha_u = 0, \quad \tau_u = 0, \quad \beta_u = \infty$

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

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

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

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

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

Shortcut-Werte

Betrachte den Pfad p der zu 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

Mit den Rekursionen können die Werte α_i,τ_i und β_i effizient berechnet werden.
Im Fall positiver Gewichte hängt jeder Wert α_i nur von Werten α_j ab, die kleiner als α_i sind
$\Rightarrow$ 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
$\Rightarrow$ 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