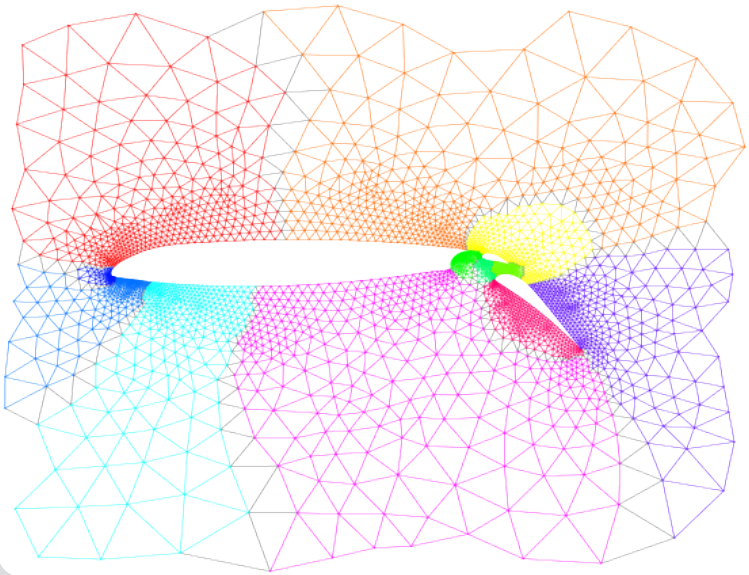
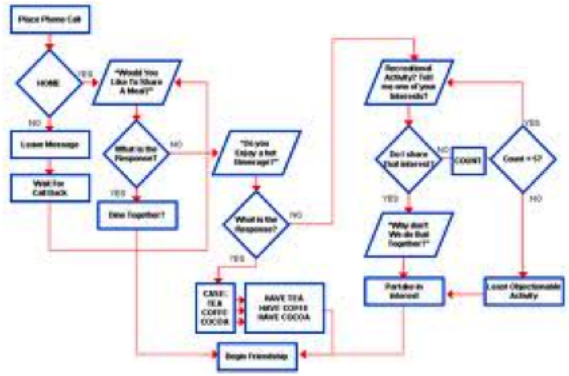
Karlsruher Institut für Technologie

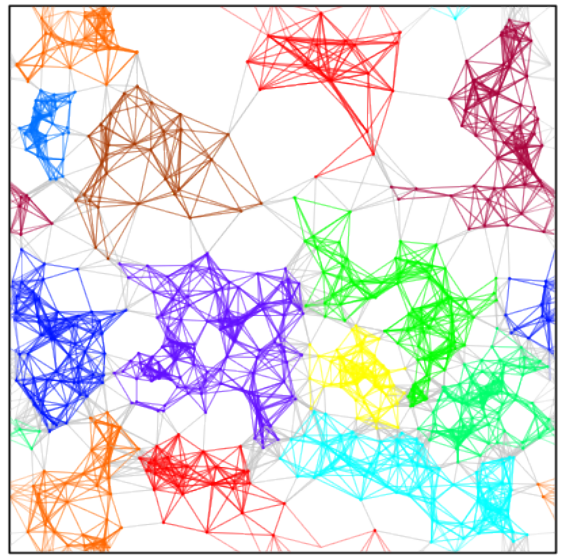
Algorithmische Methoden zur Netzwerkanalyse

Prof. Dr. Henning Meyerhenke

Institut für Theoretische Informatik







Vorlesung 5

Karlsruher Institut für Technologie

Programm des Tages:

- Präsenzübung vom Blatt 2
- Durchmesser und seine Berechnung

Inhalt

Abstand und kürzeste Wege

Definition (Pfadlänge)

Sei ein gerichteter und gewichteter Graph $G = (V, E)$ mit der Gewichtsfunktion $w : E \mapsto R$ gegeben.

Das Gewicht (oder die Länge) eines Pfades $p =< v_0, v_1, \ldots, v_k >$ ist die Summe der Gewichte seiner Kanten: $w(p) := \sum_{i=1}^{k} w(v_{i-1}, v_i)$

Definition (Kürzester Pfad)

Sei $G = (V, E)$ wie oben.

Das Gewicht eines kürzesten Pfades p zwischen $u, v \in V$ ist definiert als:

$$d(u, v) = \begin{cases} \min\{w(p) : u \text{ erreicht } v \text{ über } p\} \\ \infty \text{ sonst} \end{cases}$$

Ein kürzester Pfad zwischen $u, v \in V$ ist ein Pfad p mit $w(p) = d(u, v)$.

Abstand und Durchmesser

Definition

Für einen Multigraphen $G = (V, E)$ definieren wir:

$$\begin{aligned} \mathrm{ecc}_G(v) &= \max\{d_G(v, w) : w \in V\} && \text{(Exzentrizität von } v\text{)} \\ \mathrm{rad}(G) &= \min\{\mathrm{ecc}_G(v) : v \in V\} && \text{(Radius von } G\text{)} \\ \mathrm{diam}(G) &= \max\{\mathrm{ecc}_G(v) : v \in V\} && \text{(Durchmesser von } G\text{)} \end{aligned}$$

Beispiel an der Tafel!

Das Phänomen der kleinen Welt

- In vielen Netzwerken ist der Abstand zwischen den Knoten sehr klein ($O(\log n)$)
- Viele Experimente belegen das für unterschiedliche Netzwerke
- Mathematisch: Mittlere Distanz bzw. Durchmesser ist klein
- Implikationen für Algorithmen?

Frage: Wie berechnet man den Durchmesser (schnell)?

- Trivialer Ansatz 1: APSP, Maximum feststellen
- Aufwand kubisch bzw. $O(MM(n) \cdot \text{polylog}(n))$
- Trivialer Ansatz 2 (in ungewichteten Graphen): BFS von jedem Knoten
- Aufwand: $O(n \cdot (n+m))$

Durchmesserabschätzung

in großen dünnen ungerichteten Graphen

- Ziel: Durchmesserabschätzung für dünn besetzte Graphen mit Millionen von Knoten
- Es gibt exakte Methoden, die schneller sind als die trivialen Ansätze
- Aber: Diese haben aber (fast) quadratischen Aufwand, zu viel für große Graphen

$\Rightarrow$ Abschätzung mit weniger Aufwand,
Näherung reicht in den meisten Fällen ohnehin!

Literaturhinweise

- C. Magnien, M. Latapy, M. Habib: Fast Computation of Empirically Tight Bounds for the Diameter of Massive Graphs. Journal of Experimental Algorithmics, Volume 13, February 2009.
- P. Crescenzi, R. Grossi, M. Habib, L. Lanzi, A. Marino: On computing the diameter of real-world undirected graphs. Theor. Comput. Sci. 514: 84-95 (2013).

Durchmesserabschätzung

Untere und obere Schranken

- Offensichtlich (aber nur in ungerichteten Graphen):

$$\mathrm{ecc}_G(v) \leq \mathrm{diam}(G) \leq 2 \cdot \mathrm{ecc}_G(v) \tag{1}$$

- Die Güte der Schranken hängt natürlich stark vom Knoten v ab!

Lemma

Für Bäume (und bestimmte andere Graphklassen) gilt:
Wird v so gewählt, dass $d_G(u, v) = \mathrm{ecc}_G(u)$ für einen Knoten u, dann

$$\mathrm{diam}(G) = \mathrm{ecc}_G(v). \tag{2}$$

$\Rightarrow$ In solchen Graphen genügen zwei BFS!

- Bei irgendeinem Knoten u starten.
- Dann wiederholen bei einem Knoten v, der maximale Distanz von u hat.

Durchmesserabschätzung

Untere und obere Schranken

- Im allgemeinen Fall stimmen Schranke und Durchmesser nicht überein.
- Aber: (Häufig) Bessere untere Schranke als vorher durch zwei (oder mehr) BFS!

Weitere Verbesserungen möglich:

Lemma

Sei T ein Spannbaum von G. Dann gilt:

$$\operatorname{diam}(G) \leq \operatorname{diam}(T). \tag{3}$$

Durchmesserabschätzung

Untere und obere Schranken

Lemma

Sei T ein Spannbaum von G. Dann gilt:

$$\operatorname{diam}(G) \leq \operatorname{diam}(T).$$

Frage 1: Warum gilt das Lemma?

Frage 2: Wo kriegen wir den Spannbaum her?

Antwort 2: BFS-Baum

Schranken kombinieren

- Alle Schranken sind mit BFS zu berechnen
- Abwechselnd untere und obere Schranken verbessern
- Bei zufälligem Knoten starten oder Heuristik anwenden

Überlegungen zur Implementierung

Iterieren der Schranken

- In ungerichteten Graphen: $\text{ecc}_G(v) \leq \text{diam}(G) \leq 2 \cdot \text{ecc}_G(v)$ (1)
- Für Bäume (und bestimmte andere Graphklassen) gilt:
  Wird v so gewählt, dass $d_G(u, v) = \text{ecc}_G(u)$ für einen Knoten u, dann

$$\text{diam}(G) = \text{ecc}_G(v) \qquad (2)$$

- Sei T ein Spannbaum von G. Dann gilt:

$$\text{diam}(G) \leq \text{diam}(T) \qquad (3)$$

Pseudocode

Algorithm 1 Abschätzung des Durchmessers

```
1:  function EstimateDiameter(G = (V, E), k)
2:      while not converged do
3:          /* try to improve lower bound */
4:          Compute ecc_G(v) for suitable (e. g. random) v with BFS
5:          Update D̲ according to ecc_G(v)
6:          /* try to improve upper bound */
7:          Compute BFS tree T rooted at unused high-degree node u
8:          Compute diam(T)
9:          Update D̄ according to diam(T)
10:     end while
11:     return (D̄, D̲)
12: end function
```

Überlegungen zur Implementierung

- Iterieren der Schranken (1) und (3)
- Frage: Konvergiert die Iteration auf jeden Fall, wenn sie lange genug läuft?
- **Gegenbeispiel:** Kreis aus n Knoten **(siehe Tafel!)**
- Abbruch, wenn der Unterschied $\overline{D} - \underline{D}$ kleiner als Schwellwert ist
- Alternativer Abbruch:
  $(\overline{D} - \underline{D}) / \underline{D} < p$ für einen anderen Schwellwert p.
- Aufwand pro Iteration: $\theta(n + m)$

Experimentelle Ergebnisse

[Magnien et al., JEA Vol. 13, Feb 2009]

	t.l.b.		d.s.l.b.		h.d.t.u.b.		r.t.u.b.		t.u.b.	
INET	29		**31**		**34**		34		38	
	998	0.0001	2	0.49	26	0.1002	23	0.0633	127	0.0011
P2P	8		**9**		**10**		10		10	
	210	0.0094	1	0.7005	1	0.039	120	0.0032	3237	0.0001
WEB	26		**32**		**33**		33		34	
	816	0.001	1	0.985	34	0.0015	46	0.0025	1572	0.0005
IP	9		**9**		**9**		9		10	
	5331	0.0001	1	0.989	4	0.0543	12	0.0346	6284	0.0001

Table 1: Best bounds obtained for our four graphs by iterating each computation method: trivial lower bound (t.l.b.), double sweep lower bound (d.s.l.b.), trivial upper bound (t.u.b.), random tree upper bound (r.t.u.b.), and highest degree tree upper bound (h.d.t.u.b.). In each case, we also give the number of iterations we had to perform to obtain this bound for the first time (left), as well as the frequency with which we obtained this bound (*i.e.* the number of times we obtained it divided by the number of iterations we ran) (right).

Fazit

- Durchmesser ist längster kürzester Weg über alle Knotenpaare gesehen
- Durchmesser wichtiges Maß in der Netzwerkanalyse (Phänomen der kleinen Welt)
- Exakte Berechnung hat mindestens quadratische Laufzeit

- Abschätzen des Durchmessers für große Graphen
- BFS ist wesentliche Komponente
- Einfach!
- Algorithmus iFub von Crescenzi et al. in der Praxis noch schneller!

Inhalt

Paarweise Abstände

Einführung und Motivation

- Durchmesser kann nun abgeschätzt werden
- Exakte Berechnung z. B. durch paarweise Abstände möglich

- Paarweise Abstände auch anderweitig hilfreich
- Paarweise kürzeste Wege ebenfalls von Interesse

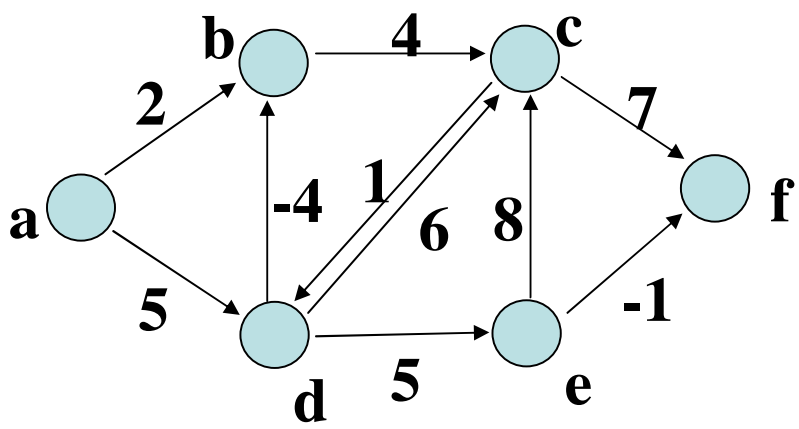
Frage: Welche Beispielanwendungen fallen Ihnen ein?

Paarweise Abstände

Beispiel

- **Eingabe:** Gewichteter Graph $G = (V, E)$
- **Ausgabe:** Für jedes Knotenpaar $(u, v) \in V \times V$: Distanz von u nach v

	a	b	c	d	e	f
a	0	1	5	5	10	9
b	∞	0	4	5	10	9
c	∞	-3	0	1	6	5
d	∞	-4	0	0	5	4
e	∞	5	8	9	0	-1
f	∞	∞	∞	∞	∞	0



Floyd-Warshall-Algorithmus

- Matrix $W = (w_{ij})$, die Graph repräsentiert:

$$w_{ij} = \begin{cases} 0 & i = j \\ \omega(i,j) & i \neq j \text{ und } (i,j) \in E \\ \infty & i \neq j \text{ und } (i,j) \notin E \end{cases}$$

- $d_{ij}^{(k)}$: Länge eines kürzesten i-j-Pfads mit Knoten aus $\{1, \ldots, k\}$

```
1:  function FLOYD-WARSHALL(W, n)
2:      D^(0) ← W
3:      for k ← 1..n do
4:          for i ← 1..n do
5:              for j ← 1..n do
6:                  d_ij^(k) ← min{d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1)}
7:              end for
8:          end for
9:      end for
10:     return D^(n)
```

Floyd-Warshall-Algorithmus

Anmerkungen

```
1:  function FLOYD-WARSHALL(W, n)
2:      D^(0) ← W
3:      for k ← 1..n do
4:          for i ← 1..n do
5:              for j ← 1..n do
6:                  d_ij^(k) ← min{ d_ij^(k−1), d_ik^(k−1) + d_kj^(k−1) }
7:              end for
8:          end for
9:      end for
10:     return D^(n)
11: end function
```

- Kubische Laufzeit
- Kürzeste Wege lassen sich ohne großen Aufwand mitberechnen
- **Hausaufgabe:** Ausführen am Beispiel

Paarweise Abstände und Matrixmultiplikation

- Wesentliche Zeile im FW-Algo: $d_{ij}^{(k)} \leftarrow \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}$
- Seien $D, W \in \mathbb{R}^{n \times n}$. Dann:

$$(D \cdot W)_{ij} = \sum_{k=1}^{n} D_{ik} \cdot W_{kj}$$

- Ersetze nun $\cdot$ durch $+$ und $+$ durch min. Dann gilt für $D, W \in (\mathbb{R} \cup \{\infty\})^{n \times n}$:

$$(D * W)_{ij} = \min_{1 \le k \le n} \{D_{ik} + W_{kj}\}$$

- **Idee:** Berechne D^{n-1} durch fortgesetzte Matrixmultiplikation
- $D^{(1)} = W, \quad D^{(2)} = W^2 = W * W, \quad D^{(4)} = W^4 = W^2 * W^2, \quad \ldots$
- $D^{(2^{\lceil \log(n-1) \rceil})} = W^{2^{\lceil \log(n-1) \rceil}} = W^{2^{\lceil \log(n-1) \rceil - 1}} * W^{2^{\lceil \log(n-1) \rceil - 1}}$

Algorithmus

Pseudocode

```
1: function MMAPD(weight matrix W)
2:     D^(1) ← W
3:     m ← 1
4:     while (m < n − 1) do
5:         D^(2m) ← MATRIX-MULTIPLY(D^(m), D^(m))
6:         m ← 2m
7:     end while
8:     return D^(m)
9: end function
```

```
1: function MATRIXMULTIPLY(matrices A, B)
2:     for i ← 1..n do
3:         for j ← 1..n do
4:             C_ij ← ∞
5:             for k ← 1..n do
6:                 C_ij = min{C_ij, A_ik + B_kj}
7:             end for
8:         end for
9:     end for
10:    return C
11: end function
```

Algorithmus
Diskussion

- Laufzeit des Algorithmus MMAPD: $\mathcal{O}(\text{MatMult}(n) \cdot \log n)$
- MatMult(n): Zeit für Multiplikation von zwei $n \times n$-Matrizen

- Nachteil: Keine implizite Darstellung der kürzesten Wege
- Dazu wären alle Matrizen $D^{(1)}, \ldots, D^{(m)}$ notwendig
- Dann stiege die Laufzeit auf $\mathcal{O}(\text{MatMult}(n) \cdot n)$:-(

- **Bald:** Schnelle praxistaugliche Algorithmen für Distanzanfragen in komplexen Netzwerken