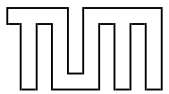
Fortgeschrittene Netzwerk- und Graph-Algorithmen

Prof. Dr. Hanjo Täubig

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

Wintersemester 2010/11



Local centers in knoten-ungewichteten Graphen

- Sei k eine natürliche Zahl, so dass

$$D_e(v_1, 0) = \ldots = D_e(v_k, 0) > D_e(v_{k+1}, 0) \qquad (1 \leq k < n)$$

- Die Definitionen von V_i und $\bar{v}_i$ implizieren, dass die Knoten $\bar{v}_1, \ldots, \bar{v}_k$ nicht definiert sind (Diese Knoten 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

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

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

Local centers in knoten-ungewichteten Graphen

Im folgenden Algorithmus von Kariv und Hakimi repräsentieren die Variablen t^* und r ein local center bzw. den local radius auf Kante $e = (v_r, v_s)$.

Knoten v_i und $\bar{v}_i$ werden repräsentiert durch Variable v^* bzw. $\bar{v}^*$.

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 $D_e(v^*, 0) \leq D_e(\hat{v}, \ell(e))$, dann $t^* \leftarrow 0$, $r \leftarrow D_e(v^*, 0)$; sonst $t^* \leftarrow \ell(e)$, $r \leftarrow D_e(\hat{v}, \ell(e))$.

   Wenn $v^* = \hat{v}$, dann STOP.

   (Für alle $v \in V$ und $t \in [0, \ell(e)]$: $D_e(v, t) \leq D_e(v^*, t)$, also $D_e(v^*, t) = D_e(t)$ und t^* und r sind ein local center und der local 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 $D_e(v,0) = D_e(v_1,0)$:
   $i \leftarrow i+1$
   Sei v^* der i-te Knoten in Liste $L(v_r)$.
   Wenn $D_e(v^*,0) \neq D_e(v_m,0)$ dann gehe zu Schritt 4;
   sonst falls $D_e(v^*,\ell(e)) > D_e(v_m,\ell(e))$: $v_m \leftarrow v^*$. Wiederhole Schritt 3.
   (Wegen $D_e(v_i,0) = D_e(v_1,0)$ ist $i < n$.)
4. $\bar{v}^* \leftarrow v_m$
   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 $D_e(v,0) = D_e(v_i,0)$, sowie das entsprechende v_m:

   $i \leftarrow i+1$

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

   Falls $D_e(v^*,0) \neq D_e(v_m,0)$, dann gehe zu Schritt 6;
   sonst falls $D_e(v^*,\ell(e)) > D_e(v_m,\ell(e))$: $v_m \leftarrow v^*$.
   Wiederhole Schritt 5.
   (Wegen $D_e(v_i,0) = D_e(v_{i-1},0)$ ist $i < n$.)

Local centers in knoten-ungewichteten Graphen

6. Behandlung des Punkts t_m:

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

   Sonst sei t_m der Schnittpunkt.
   Falls $D_e(v_m, t_m) < r$, dann $t^* \leftarrow t_m$, $r \leftarrow D_e(v_m, t_m)$.

7. Gehe zum nächsten Knoten:

   Wenn $D_e(v_m, \ell(e)) > D_e(\bar{v}^*, \ell(e))$ 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 $D_e(v^*, t)$ und $D_e(\bar{v}^*, t)$ nicht schneiden oder sich in einem ganzen Segment überlagern, dann STOP.
   Sonst sei t_m der Schnittpunkt.
   Falls $D_e(v^*, t_m) \geq r$, dann STOP;
   sonst $t^* \leftarrow t_m$, $r \leftarrow D_e(v^*, t_m)$ 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 $T(x^*)$.

$$
\begin{aligned}
D(T(x^*)) &= 2 \max_{v \in V} d_{T(x^*)}(x^*, v) && (1)\\
&= 2 \max_{v \in V} d_G(x^*, v) && (2)\\
&\leq 2 \max_{v \in V} d_G(y^*(T), v) && (3)\\
&\leq 2 \max_{v \in V} d_T(y^*(T), v) = D(T) && (4)
\end{aligned}
$$

□

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

$\Rightarrow$ $d(i,j)$ ist definiert für alle Knotenpaare (i,j)

- keine parallelen Kanten

Idee:

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

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
- $\alpha_u = 0, \quad \tau_u = \emptyset, \quad \beta_u = \infty$

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

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

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

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

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

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

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
$\Rightarrow$ 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