Netzwerkalgorithmen

Sommersemester 2020

Stefan Näher
Universität Trier
naeher@uni-trier.de

Vorlesung 12

Successive Shortes Paths

Berechne Folge von optimalen Pseudoflows und dazugehörigen Potentialen

$$(x_0, \pi_0),\ (x_1, \pi_1),\ \ldots,\ (x_\ell, \pi_\ell)\ \text{ mit}$$

1. $x_0 = 0$ und $\pi_0 = 0$

2. $c_{ij}^{\pi_k} \geq 0$ für alle (i, j) in $G(x_k)$ für $0 \leq k \leq \ell$.

3. x_ℓ ist ein Fluss.

SUCCESSIVE_SHORTEST_PATHS

1. $x \leftarrow 0$
2. $\pi \leftarrow 0$
3. Wähle in jedem Schritt einen Excess-Knoten s und einen Defizit-Knoten t d.h. $e(s) > 0$ und $e(t) < 0$
4. Berechne Kürzeste-Wege Distanzen $d(i)$ von s in $G(x)$ bzgl. der Kosten c_{ij}^{π}
5. $\pi \leftarrow \pi - d$
6. Erhöhe x entlang eines kürzesten Pfades P von s nach t um

$$\delta = \min\left(e(s), -e(t), \min_{(i,j)\in P} r_{ij}\right)$$

Nach Zeile 5: $c_{ij}^{\pi} \geq 0$ für alle $(i,j) \in G(x)$ und $c_{ij}^{\pi} = 0$ für alle $(i,j) \in P$

Laufzeitanalyse

1. Da stets $c_{ij}^{\pi} \geq 0$ gilt, können die Distanzen d in Zeile 4 des Algorithmus mit dem Algorithmus von Dijsktra berechnet werden.

2. Jede Iteration des Algorithmus vermindert den Gesamtüberschuss $\sum_{e(i)>0} e(i)$ um mindestens 1 Einheit.

3. Also ist die maximale Anzahl von Iterationen (d.h. Ausführungen von Dijkstra)

$$\leq \sum_{b(i)>0} b(i) \leq n \cdot U$$

4. Damit ist die Gesamtlaufzeit $\mathcal{O}\left(n \cdot U \cdot (n \log n + m)\right)$

MCF Polynomielle Algorithmen

1. **Capacity Scaling** (Emonds/Karp 1972)
   - versuche den Fluss in größeren “Portionen” zu erhöhen
   - zerlege Flüsse dazu in 2er - Potenzen

2. **Repeated Capacity Scaling** (Orlin 1984)

2. **Enhanced Capacity Scaling** (Orlin 1988)

Capacity Scaling

1. $\Delta \leftarrow 2^{\lfloor \log U \rfloor}$;
2. **while** $\Delta \geq 1$ **do**
3. SUCCESSIVE_SHORTES_PATHS(Δ); // Δ-Phase
4. $\Delta \leftarrow \Delta/2$;
5. **od**

SUCCESSIVE_SHORTES_PATHS(Δ)
ist eine Variante des Successive-Shortest-Paths Algorithmus, die auf dem Δ-Restnetzwerk $G(x, \Delta)$ arbeitet, d.h. nur Kanten (i, j) mit $r_{ij} \geq \Delta$ betrachtet.

Laufzeitanalyse

1. Die Zahl der Δ-Phasen (Ausführungen der Hauptschleife) ist $\log U$.

2. Am Anfang jeder Δ-Phase gilt

$$\forall i \in V : e(i) < 2\Delta \text{ oder } \forall i \in V : e(i) > -2\Delta$$

   und damit ist der Gesamtüberschuss maximal $2n\Delta$, so dass in einer Δ-Phase maximal $2n$ Shortest-Paths Erhöhungen stattfinden können.

3. Eine Shortest-Path Erhöhung (Dijkstra) hat Laufzeit $\mathcal{O}(n \log n + m)$.

Damit ergibt sich eine Gesamtlaufzeit von

$$\mathcal{O}\left(n \log U \cdot (n \log n + m)\right)$$

Repeated Capacity Scaling

Eigenschaften des Capacity Scaling Algorithms (ohne Beweis)

1. Falls $x_{ij} > 4n\Delta$ am Anfang einer Δ-Phase, dann gilt:
   - $x'_{ij} > 0$ für jeden optimalen Fluss x'
   - $c^{\pi'}_{ij} = 0$ für alle optimalen Potentiale π'
   - das MCF-Problem P' mit $c' = c^{\pi'}$ hat die gleich Lösung wie P
   - für alle optimalen Paare (x', π') von P' gilt $\pi'(i) = \pi'(j)$

2. Nach $\log(6n^2) = \mathcal{O}(\log n)$ Phasen gilt für mindestens eine Kante (i, j) $x_{ij} > 4n\Delta$

Repeated Capacity Scaling Algorithmus

1. Führe den Capacity Scaling Algorithm aus

2. Falls $x_{ij} > 4n\Delta$
   kontrahiere (i, j) zu einem Knoten k ($\longrightarrow$ MCF-Problem P')
   löse das Problem P' rekursiv
   expandiere k wieder zu (i, j)

Analyse

1. Zahl der Phasen $= \mathcal{O}(n \log n)$
   da jeweils nach $\mathcal{O}(\log n)$ Phasen mindestens eine Kante kontrahiert und damit die Zahl der Knoten um mindestens 1 vermindert wird.

2. In jeder Phase finden maximal $2n$ Erhöhungen statt (Capacity Scaling).

Also werden insgesamt $\mathcal{O}(n^2 \log n)$ Shortest-Paths Erhöhungen (Dijkstra) ausgeführt und die Gesamtlaufzeit ist

$$\mathcal{O}\left(n^2 \log n \cdot (n \log n + m)\right)$$

stark polynomielle Laufzeit

Enhanced Capacity Scaling

1. Kontraktionen nur konzeptuell.

2. Die neuen MCF-Problem muss man nicht immer wieder von Anfang an lösen.

Dadurch reduziert sich die Anzahl der Shortest-Paths Erhöhungen (Dijkstra) auf $O(n \log n)$ und damit die Gesamtlaufzeit auf

$$\mathcal{O}\left(n \log n \cdot (n \log n + m)\right)$$

Bester Algorithmus für das MCF-Problem in der Theorie.