Netzwerkalgorithmen

Prof. Dr. Stefan Näher
verschriftet von Erik Boritsch

12. April 2010

Inhaltsverzeichnis

Inhalt Algorithmen für Netwerk(fluss)probleme

Literatur Network Flows Ahuja, Magmonti, Orlin (Prentice Hall, 1993)
LEDA, A Platform for Combinatorial and Geometric Computing Mehlhorn/Näher.

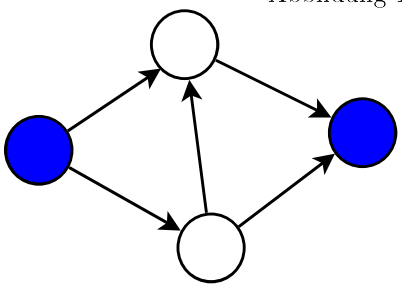
0.1 Netzwerke

Netzwerke: Graphen + Zusatzinformationen G=(V,E)

Beispiel Kantenlabel C: E $\rightarrow \mathbb{R} \Rightarrow$ kürzeste (billigste) Flussprobleme $\rightarrow$Transportnetzwerk

1. Gerichteter Graph G=(V,E)

Abbildung 1: Gerichteter Graph



2. Kapazität auf den Kanten $u : E \rightarrow \mathbb{R}_0^+$ $v \xrightarrow[\vec{u(e)}]{c} w$
   u(e): Wieviele Einheiten pro Zeiteinheit kann e transportieren

   Einheiten Kg, Anzahl

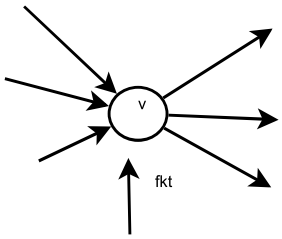
   Zeiteinheit 1 Sec. ....

   Fluss über die Kante e: f(e)

   $(f \cdot E \rightarrow \mathbb{R}_0^+)$, $0 \leq f(e) \leq \overset{\text{Kapazität}}{u(e)}$ (Kapazitätsbedingung)

3. Massenbalance an einem Knoten v
   Summe der eingehenden Flüsse = Summe der ausgehenden Flüsse

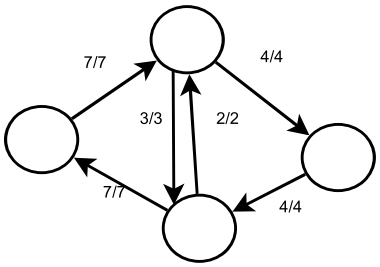
Abbildung 2: Massenbalance

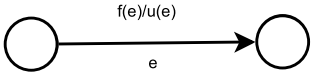


$$\sum_{(u,v)\in E} f((u,v)) = \sum_{(u,v)\in E} f((v,w))$$

Bemerkung Falls Massenbalance für jeden Knoten erfüllt: → Zirkulation

Abbildung 3: Beispiel





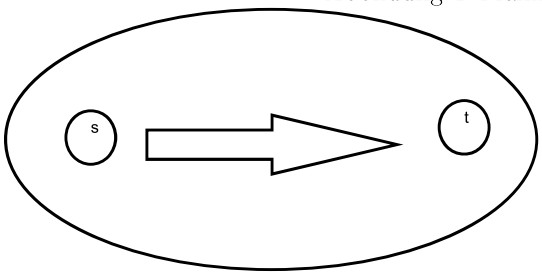
Beispiel

1. $\forall e \in E : f(e) = 0$ ist mögliche Zirkulation, Massenbalance trivial erfüllt √

0.2 Maximale Flüsse (Maxflow)

Es gibt 2 Knoten $s, t \in V$, die Massenvalance verletzen.
s: Quelle (source)
t: Zeil (drain), $s \neq t$

Abbildung 4: Maxflow



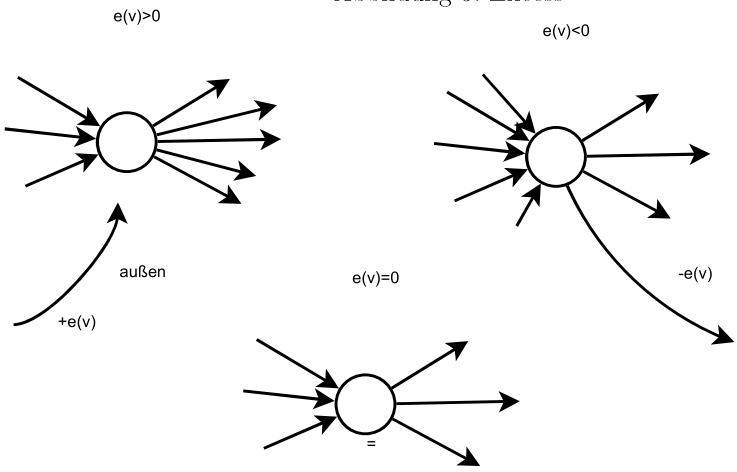
Definiere $e : V \to \mathbb{R}$

$$e(v) = \sum_{(v,w)\in E} f(v,w) - \sum_{(u,v)} f(u,v)$$

$$\text{Excess } e(v) \begin{cases} > 0 & \text{Einspeiseknoten} \\ = 0 & \text{Transportknoten} \\ < 0 & \text{Entnahmeknoten} \end{cases}$$

Massenbalance erfüllt $\Leftrightarrow e(v) = 0$

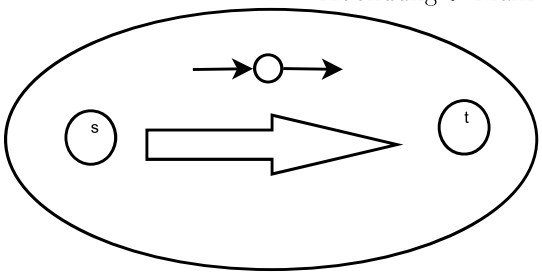
Abbildung 5: Excess



Zirkilation $\forall v \in V : e(v) = 0$

Verallgemeinte Massenbalance $\sum_{v\in V} e(v) = 0$

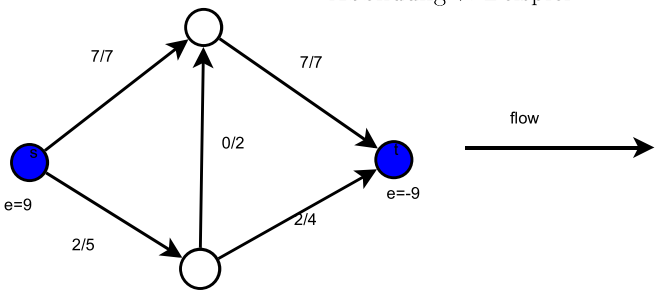
Abbildung 6: MaxFlow



Finde Flussfunktion $f : E \to \mathbb{R}_0^+$

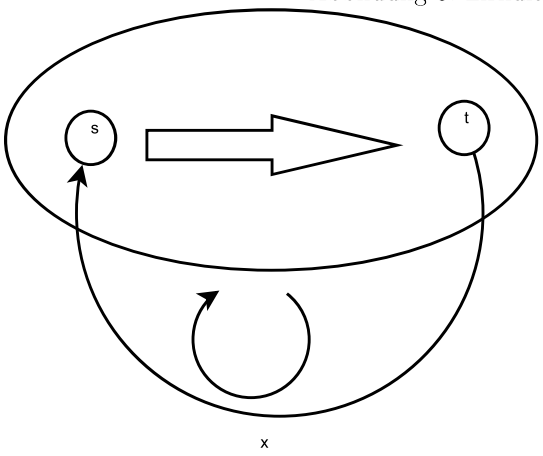
1. $\forall x \in E : 0 \leq f(x) \leq u(x)$ Kapazitätsbedingung
2. $\sum_{v \in V} e(v) = 0$ Massenbalance
3. $\forall v \in V \backslash \{s, t\} : e(v) = 0$
4. $e(s) = -e(t)$ maximal

Abbildung 7: Beispiel



Beispiel Maxflow → Zirkulation

Abbildung 8: Zirkulation



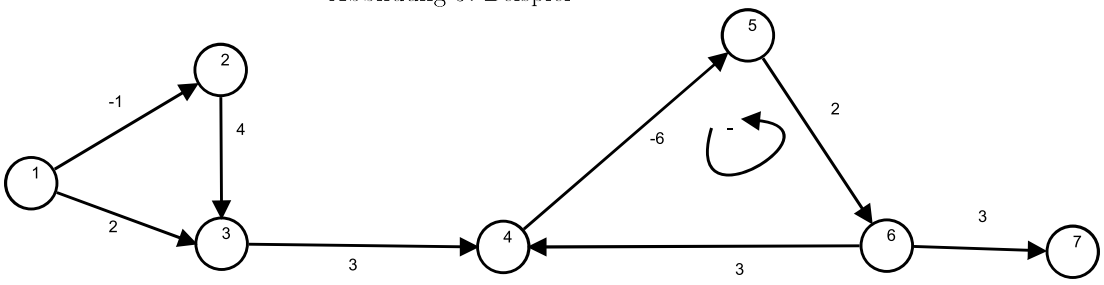
e(v)=0 für alle $v \in V$, $u(x) = \infty$
Suche Zirkulation $f : E \to \mathbb{R}_0^+$, die f(x) maximiert
19. April 2010

Aufgabe: Finde Fluss. der die Supply, Demand, Kapazizäz erfüllt → Feasible Flow, mit minimalen Gesamtkosten

1 Kürzeste (billigste) Wege

Eingabe: G=(V,E) gerichteter Graph, Netzwerk $c : E \to \mathbb{R}$Kanten Kosten (Funktion)

Abbildung 9: Beispiel



Beispiel

Kosten eines Pfades $c((v,w) \to c(v,w)$

Sei $P = v_0, v_1, \ldots, v_l$ ein Pvad von v nach w (d.h. $v_0 = v, v_l = w$, $(v_i, v_{i+1}) \in E$ für $i = 0, \ldots, l-1$)

Kosten $c(P) = \sum_{i=0}^{l-1} (v_i, v_{i+1})$ Summe der Knatenkosten

Definiere $dist(v,w) = inf\,\{C(P) | \text{P ist Pfad von v nach w}\}$

inf: Infinum: $\{+\infty, -\infty\}$

$+\infty$: Wenn kein Pfad existiert

$-\infty$: Wenn Pfade beliebig klenen Kosten existierten $\to$ neg. Zyklen

Im Beispiel $dist(1,2) = -1$

$dist(1,3) = 2$

$dist(4,1) = +\infty$ (kein Pfad)

$dist(3,7) = -\infty$ (Pfad $3,4,5,6(4,5,6)^i, 7$, $i \geq 0$) hat Kosten $-1-i+1 = -i$

Spezialfall Kürzeste Wege von einem festen Knoten s aus $\to$ (single source shortest paths)

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

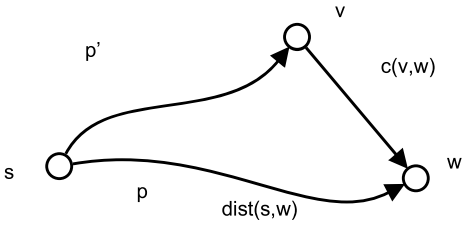
Ausgabe $dist(s,v)$ für alle $v \in V$

Beobachtung Die dist-Funktion (DIST[v] = dist(s,v)) erfüllt die Dreiecksungleichung ($\triangle$)

Sei $(v,w) \in E$ gel. Kante, dann filt:

$dist(s,w) \leq dist(s,v) + c(v,w)$

Abbildung 10: Beispiel



P kürzester Pfad von s nach w

P' kürzester Pfad von s nach v

Annahme $\underbrace{dist(s,v) + c(u,v)}$ $< dist(s,w)$

Pfad P' +(u,w) wäre billiger als P: Widerspruch

Beobachtung $\Rightarrow$ Algorithmus

Verwende Feld DIST[v] für temporäre $dist(s,v)$ Werte. Am Schluss: $DIST[v] = dist(s,v)$

1.1 Grundalgorithmus

```
forall v ∈ V do DIST[v] <- +∞ od;
DIST[s] <- 0;
while ∃ Kante (u,v) mit DIST[v] > DIST[u]+c(u,v); //△-Ungl. verletzt!
do
    wähle eine solche Kante (u,v);
    DIST[v] <- DIST[u]+c(u,v);
od
```

Beweise durch Induktion Beweis durch Induktion über Schleife, dass Falls DIST[v] $< \infty$, dann existiert Pfad von s nach v mit Kosten DIST[v]

Grundalgorithmus kann exponentielle Laufzeit haben (Übung).

Verbesserung

1. Speichere alle Knoten, aus denen Kanten ausgehen können, die Dreiecksungleichung verletzen in einer Menge U.
2. Wähle $u \in U$ und teste alle ausgehenden Kanten auf Verletzung der Dreiecksungleichung.
3. Immer wenn DIST[v] vermindert wird, dann nimm v in U auf.
4. Am Anfang $U = \{s\}$

1.2 Verfeinerter Algorithmus

```
forall v ∈ V do DIST[v] <- ∞ od;
DIST[s] <- 0;
U <- {s};
while U ≠ ∅ do
    wähle beliebigen Knoten u ∈ U; //Zeile 5
    U <- U\{u};
    forall v ∈ V mit (u,v) ∈ E do
        d <- DIST[u] + c(u,v);
        if DIST[v] > d then
            DIST[v] <- d;
            U <- U∪{v}
        fi
    od
od
```

Eigenschaften

1. Es gilt stets: $DIST[v] \geq dist(s, v)$ (da DIST-Werte immer Kosten eines existierenden Pfades oder $= \infty$ sind.
2. Am Ende DIST[v] = dist(s,v)

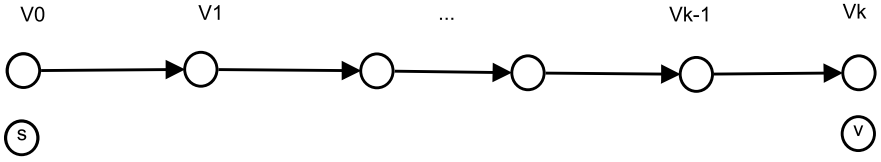
Lemma Falls G keinen negativen Kreis enthält, gilt:

1. Falls $v \notin U$, dann gilt für alle ausgehenden Kanten (v,w); $\underbrace{DIST[v] + c(v, w) \geq DIST[w]}_{\triangle\text{–Ungl.}\checkmark}$
2. Sei $s = v_0, \ldots, v_k = v$ ein billigster Pfad von s nach v. Falls DIST[v] > dist(s,v), dann gibt es ein v_i, $0 \leq i \leq k-1$ mit
   (a) $DIST[v_i] = dist(s, v_i)$
   (b) $v_i \in U$
3. Es existiert immer $u \in U$ mit DIST[u] = dist(s,u) → Perfekte Wahl.
4. Wenn in Zeile 5 stets eine perfekte Wahl ausgeführt wird (nach Teil 3.), dann wird die while-Schleife für jeden Knoten u höchstens einmal ausgeführt.

26. April 2010
Beweis → Übung

Beweis vom Teil 2. sei P billigster Pfad, von s nach v und DIST[v] < dist(s,v)

Abbildung 11:



Dann gilt: für ein i, $0 \leq i \leq k-1$

1. $v_i \in U$
2. $DIST[v_i] = dist(s_i, v_i)$
   Daraus folgen Teile 3. und 4.
3. zu jedem Zeitpunkt existiert ein $u \in U$ mit DIST[u]=dist(s,u) (perfekte Wahl)

   Beweis

Betrachte $u \in U$

Fall 1 DIST-Wert korrekt $\surd$

Fall 2 Sei P billigster Pfad von s nach v $\overset{\text{Teil 2.}}{\Rightarrow} \exists u \in U$ auf P mit korrektem DIST-Wert

4. Wenn in Zeile 5 stets eine perfekte Wahl getroffen, dann wird die while-Schleife für jeden Knoten u höchstens einmal ausgeführt.

   Beweis Knoten wird nur dann nach U aufgenommen, wenn sein DIST-Wert vermindert wird.

1.3 Varianten des Algorithmus für verschiedene Problem-Typen

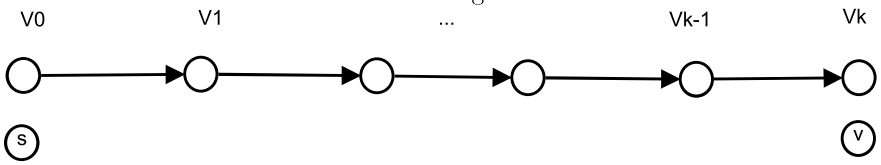
1.3.1 Keine negativen Zyklen

1. Azyklische Netzwerke → topologische Sortierung.

Zeile 5 wähle $u \in U$ mit kleinster topologischen Nummer.

Behauptung Es gilt DIST[u]=dist(s,u)

Abbildung 12:



Beweis (indirekt)

Annahme DIST[u]>dist(s,u)

Lemma Teil 2. ⇒ Auf billigstem Pfad von s nach u existiert ein $v \in U$mit DIST[v]=dist(s,v)

Da $v \overset{+}{\longrightarrow} u$ gilt topnum(v) < topnum(u) und $u \in U$: Widerspruch zur Wahl von u.

Implementierung mit Laufzeit O(n+m) *1. Algorithmus*

1. Topologische Sortierung → $topnum[v] \in \{1, \ldots, n\}$, so dass für alle $(u, v) \in E$: $topnum[v] < topnum[w]$ Zeit: $O(n + m)$