Experimentelle Ergebnisse zum Network-Simplex-Algorithmus

Max Kanold

27. September 2018

Inhaltsverzeichnis

Kapitel 1

Einführung

TODO unbeschränkt/beschränkt je einzelne Kapitel, Einführungstexte umsortieren.

Bla. Zum Beispiel in Abschnitt 2.3 habe ich programmiert.

Kapitel 2

Network-Simplex-Algorithmus

Das Simplex-Verfahren, zu welchem eine Einführung in [1] gefunden werden kann, löst Lineare Programme in der Praxis sehr schnell, obwohl die Worst-Case-Laufzeit nicht polynomiell ist. Jedes Netzwerkproblem lässt sich als Lineares Programm darstellen und somit durch das Simplex-Verfahren lösen, durch die konkrete Struktur solcher Probleme genügt jedoch der vereinfachte Network-Simplex-Algorithmus. Auch für diesen gibt es exponentielle Instanzen (siehe [2]), in der Praxis wird er trotzdem vielfach verwendet.
TODO gibt es?

2.1 Min-Cost-Flow-Problem

Definition 1. Ein **Netzwerk** ist ein Tupel (G, b, c, u), wobei $G = (V, E)$ ein gerichteter Graph, $b : V \to \mathbb{R}$ eine b-Wert-Funktion, $c : E \to \mathbb{R}$ eine Kostenfunktion und $u : E \to \mathbb{R}_{\geq 0} \cup \infty$ eine Kapazitätsfunktion seien.

Anmerkung. Knoten mit positiven b-Wert werden als Quellen, solche mit negativen als Senken bezeichnet.
Ein ungerichteter Graph kann durch das Verwandeln jeder Kante $\{v, w\}$ in zwei Kanten (v, w) und (w, v) zu einem gerichteten modifiziert werden.

Definition 2. Ein **maximaler Fluss** auf einem Netzwerk $(G = (V, E), b, c, u)$ ist eine Abbildung $f : E \to \mathbb{R}_{\geq 0}$, die folgende Eigenschaften erfüllt:

(i) $\forall e \in E : f(e) \leq u(e)$

(ii) $\forall v \in V : \sum_{(w,v)\in E} f((w, v)) - \sum_{(v,w)\in E} f((v, w)) + b(v) = 0$

Der **Wert** von f ist $v(f) = \frac{1}{2} \cdot \sum_{v\in V} |b(v)|$
und die **Kosten** von f sind $c(f) = \sum_{e\in E} f(e) \cdot c(e)$.

Beim *Min-Cost-Flow-Problem* wird unter allen maximalen Flüssen einer mit minimalen Kosten gesucht. Sind die Kapazitäten unbeschränkt, so wird es als *Transportproblem* bezeichnet.

In dieser Bachelorarbeit wird angenommen, dass G zusammenhängend ist, andernfalls betrachte man jede Zusammenhangskomponente einzeln. Wir nehmen weiterhin an, dass u und c auf $\mathbb{N}$ sowie b auf $\mathbb{Z}$ abbildet, um Gleitkommazahlungenauigkeit beim Programmieren zu vermeiden. Durch eine entsprechende Skalierung des Problems können die Funktionen nach $\mathbb{R}_{\geq 0}$ bzw. $\mathbb{R}$ hinreichend genug angenähert werden. Durch die eingeschränkte Kostenfunktion hat kein maximaler Fluss negative Kosten; es gibt keine unbeschränkten Instanzen. Unbeschränkte Kapazitäten können somit o. B. d. A. durch $\frac{1}{2} \cdot \sum_{v \in V} |b(v)| + 1$ abgeschätzt werden.

Zusätzlich wird davon ausgegangen, dass $\sum_{v \in V(G)} b(v) = 0$ ist, Angebot und Nachfrage also ausgeglichen sind. Des Weiteren ist in der konkreten Implementierung E keine Multimenge; es sind keine parallelen Kanten vorgesehen.

2.2 Algorithmus

[3, Dantzig, 1951] und [4, Orden, 1956] vereinfachten das Simplex-Verfahren zum Netzwerk-Simplex-Algorithmus; die folgende Beschreibung orientiert sich zuerst an [1, S. 291 ff.] zur Lösung des Transportproblems, danach wird der Algorithmus anhand von [1, S. 353 ff.] auf den allgemeinen, durch Kapazitäten beschränkten Fall erweitert.

Definition 3. Der einem gerichteten Graphen $G = (V, E)$ **zugrundeliegende ungerichtete Graph** $G' = (V, E')$ ist definiert durch:

$$\{v, w\} \in E' \iff (v, w) \in E \vee (w, v) \in E$$

Anmerkung. Nach dieser Definition gibt es keine Bijektion zwischen gerichteten und den zugrundeliegenden ungerichteten Graphen; dafür vermeidet man parallele Kanten.

Definition 4. Ein **Baum** T ist ein ungerichteter, zusammenhängender und kreisfreier Graph. Ein **Wald** ist ein Graph, bei dem jede Zusammenhangskomponente ein Baum ist.
Ein Teilgraph $T = (V', E')$ eines ungerichteten Graphen $G = (V, E)$ heißt **aufspannender Baum**, wenn T ein Baum und $V' = V$ ist.

Anmerkung. Sprechen wir bei einem gerichteten Graphen G über einen Wald bzw. (aufspannenden) Baum, so bezieht sich das stets auf einen Teilgraphen T von G, dessen zugrundeliegender ungerichteter Graph ein Wald bzw. (aufspannender) Baum des G zugrundeliegenden ungerichteten Graphen ist.

Definition 5. Sei $N = (G, b, c, u)$ ein Netzwerk. Ein aufspannender Baum T von G und ein maximaler Fluss f auf N bilden eine **zulässige Baumlösung** (T, f), wenn $\forall e \in E(G) \backslash E(T) : f(e) = 0$.

Lemma 6. *Jede zulässige Baumlösung (T, f) ist eindeutig durch den aufspannenden Baum T definiert.*

Beweis. Sei $(T = (V, E), f)$ eine zulässige Baumlösung eines Netzwerkes. Für jedes Blatt von $l \in V$ sei $e_l \in E$ die eindeutige Kante in T zwischen den Knoten k und l. Für alle Blätter l ist der Wert von $f(e_l)$ nach Definition 2 Punkt (ii) eindeutig.

Wir entfernen nun alle mit Fluss belegten Kanten e_l sowie die nun isolierten Knoten l und aktualisieren den b-Wert von k. Der dabei entstehende Graph $T' = (V' \subseteq V, E' \subseteq E)$ bleibt weiterhin ein aufspannender Baum und $f_{|E'}$ ein maximaler Fluss. Durch Iteration ist f wohldefiniert und eindeutig. $\square$

Wie Abb. 2.1 veranschaulicht, gibt es maximale Flüsse, zu denen wir keine zulässige Baumlösung finden können. Auch wenn wir uns auf die maximalen Flüsse beschränken, zu denen es zulässige Baumlösungen gibt, gilt die Gegenrichtung nach Abb. 2.1 nicht.

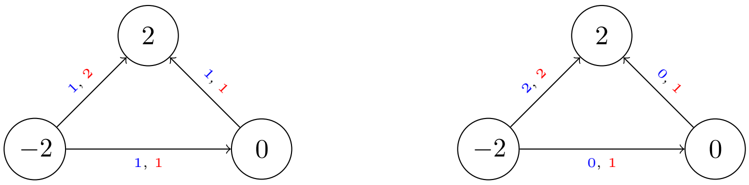

Abbildung 2.1: Links ein maximaler Fluss ohne zulässige Baumlösung, rechts ein maximaler Fluss mit uneindeutiger zulässiger Baumlösung.
Legende. b-Wert: schwarz im Knoten, Fluss: blau auf Kante, Kosten: rot auf Kante

Wie wir in Korollar 10 zeigen werden, existiert eine zulässige Baumlösung (T, f), sodass $c(f)$ minimal ist. Die dem Algorithmus zugrundeliegende Idee ist es, über die Bäume zulässiger Baumlösungen mit sinkenden Kosten zu iterieren, bis diese minimal sind. Der Übergang basiert dabei auf dem Augmentieren negativer Kreise, eine Methode, für die das Konzept des Residualgraphen hilfreich ist.

Definition 7. Sei $N = (G = (V, E), b, c, u)$ ein Netzwerk mit einem maximalen Fluss f. Die **Residualkante** $\bar{e}$ einer Kante $e = (v, w) \in E$ verläuft von w nach v. Sei $\bar{E} = \{\bar{e} | e \in E\}$ die Menge aller Residualkanten. Die *Residualkapazität* $u_f : \bar{E} \to \mathbb{Z}$ ist bestimmt durch $u_f(\bar{e}) = f(e)$, die *Residualkosten* $c_f : \bar{E} \to \mathbb{Z}$ durch $c_f(\bar{e}) = -c(e)$.

Der **Residualgraph** R ist ein Tupel $R_{N,f} = (\bar{G} = (V, E \amalg \bar{E}), \bar{f}, b, \bar{c}, \bar{u})$ mit dem gerichteten Multigraphen $\bar{G}$ aus der bisherigen Knotenmenge und der disjunkten Vereinigung von Kanten und Residualkanten, dem maximalen Fluss $\bar{f}$ mit $\bar{f}(e) = f(e)$ für alle $e \in E$ und $\bar{f}(\bar{e}) = 0$ für alle $\bar{e} \in \bar{E}$, der b-Wert-Funktion b wie gehabt, der Kostenfunktion $\bar{c} = c \amalg c_f : E \amalg \bar{E} \to \mathbb{Z}$ und der Kapazitätsfunktion $\bar{u} = u \amalg u_f : E \amalg \bar{E} \to \mathbb{Z}$.

Erhöht man in einem Residualgraphen $R_{N,f}$ den Fluss einer Residualkante $\bar{e}$ um $0 \leq \delta \leq u_f(\bar{e})$, so entspricht das der Flussreduktion der dazugehörigen Kante e in N um δ. Nach obiger Definition ist sichergestellt, dass $f(e) - \delta \geq 0$.

Augmentieren wir in einem Residualgraphen $R_{N,f}$ einen gerichteten Kreis $C \subseteq E \amalg \bar{E}$, erhöhen also für alle Kanten $e \in C$ den Fluss um einen festen Betrag $\delta \in \mathbb{N}$, ohne die Kapazitätsschranken zu verletzen, erhalten wir einen neuen maximalen Fluss f' in N. Seine Kosten betragen $c(f') = c(f) + \delta \cdot (\sum_{e \in C} \bar{c}(e))$.

TODO Kosten eines Kreises definieren?

Lemma 8. *Sei $R_{N,f}$ ein Residualgraph und C ein Kreis in $R_{N,f}$ mit negativen Kosten. Der größte Wert δ, um den C augmentiert werden kann, ist endlich, und nach der Augmentierung um δ zum neuen maximalen Fluss f' existiert eine*

Residualkante $\bar{e} \in C$, sodass die korrespondierende Kante e einen Fluss von $f'(e) = 0$ besitzt.

Beweis. Sei $N = ((V, E), b, c, u)$ ein Netzwerk mit maximalen Fluss f und C ein negativer Kreis in $R_{N,f}$. Da $c : E \to \mathbb{N}$ in die natürlichen Zahlen abbildet, enthält jeder negative Kreis in $R_{N,f}$ mindestens eine Residualkante $\bar{e}$. Damit ist $0 \leq \delta \leq u_f(\bar{e}) < \infty$.

Sei nun $\delta = \min_{e \in C}\{\bar{u}(e) - \bar{f}(e)\}$ maximal gewählt. Da alle Kanten $e \in E$ unbeschränkte Kapazität haben und ihr Fluss $f(e)$ endlich ist, gibt es eine Residualkante $\bar{e}$ mit $u_f(\bar{e}) = \delta$. Augmentiert man C um δ zu einem maximalen Fluss f', ist der neue Fluss auf der korrespondierenden Kante $f'(e) = f(e) - \delta = f(e) - u_f(\bar{e}) = f(e) - f(e) = 0$. □

Notation. Sei f ein maximaler Fluss für ein Netzwerk $(G = (V, E), b, c, u)$ und $H = (V' \subseteq V, E' \subseteq E)$ ein Teilgraph. Mit $H_f = (\hat{V}, \hat{E} = \{e \in E' | f(e) \neq 0\})$ bezeichnen wir den Graph aller durchflossenen Kanten.

Dank Lemma 8 wissen wir, dass nach maximaler Augmentierung eines negativen Kreises C alle seine durchflossenen Kanten C_f einen Wald bilden. Damit können wir nun zeigen, dass eine zulässige Baumlösung (T, f) mit einem maximalen Fluss f minimaler Kosten existiert.

Theorem 9. *Sei f ein maximaler Fluss. Es existiert ein maximaler Fluss $\hat{f}$, sodass $c(\hat{f}) \leq c(f)$ ist und eine zulässige Baumlösung $(\hat{T}, \hat{f})$ existiert.*

Beweis. Sei $N = (G, b, c, u)$ ein Netzwerk mit maximalen Fluss f. Wir werden f zu einem maximalen Fluss $\hat{f}$ umwandeln, sodass $G_{\hat{f}}$ ein Wald ist. Für $\hat{f}$ finden wir dann leicht eine zulässige Baumlösung. Wenn wir für die endlich vielen maximalen Zwischenflüsse $f = f_0, f_1, f_2, \ldots, f_n = \hat{f}$ sicherstellen, dass $c(f_{i+1}) \leq c(f_i)$ ist, gilt auch $c(\hat{f}) \leq c(f)$.

Betrachte einen maximalen Fluss f_i. Wenn G_{f_i} ein Wald ist, gilt $\hat{f} = f_i$ und wir sind fertig. Ansonsten gibt es einen ungerichteten Kreis C in G_{f_i}. Betrachte die beiden dazugehörigen, gerichteten, kantendisjunkten Kreise C_1 und C_2 in R_{N,f_i}. Nach Definition 7 gilt $c(C_1) = -c(C_2)$.

Fall 1: $c(C_1) = 0$

Mindestens einer der beiden Kreise enthält eine Residualkante, sei dies o. B. d. A. C_1. Augmentiere nun C_1 analog zum Beweis von Lemma 8 größtmöglich zu einem maximalen Fluss f_{i+1}. Damit ist $C \nsubseteq G_{f_{i+1}}$ und $c(f_{i+1}) = c(f_i)$.

Fall 2: $c(C_1) \neq 0$

O. B. d. A. sei $c(C_1) < 0$. Nach Lemma 8 erhalten wir einen maximalen Fluss $f_{i+1} = f'$, sodass $C \nsubseteq G_{f_{i+1}}$ und $c(f_{i+1}) = c(f_i) + \delta \cdot c(C_1) < c(f_i)$.

Damit ist $G_{f_{i+1}} \subsetneq G_{f_i}$ sowie $c(f_{i+1}) \leq c(f_i)$. Nach endlich vielen Iterationen erhalten wir somit $\hat{f}$. □

Korollar 10. *Für ein Netzwerk N existiert eine zulässige Baumlösung (T, f), sodass der maximale Fluss f minimale Kosten hat.* □

Notation. Sei $N = (G, b, c, u)$ ein Netzwerk, T ein aufspannender Baum von G und $e \in E(G) \backslash E(T)$ eine weitere Kante. Mit $C_{T,e}$ bezeichnen wir den eindeutigen Teilgraph von $T \cup \{e\}$, dessen zugrundeliegender Graph ein Kreis ist, und mit $\bar{C}_{T,e}$ den eindeutigen gerichteten Kreis in $R_{N,f}$ zu $C_{T,e}$, der e enthält.

Sei $N = (G, b, c, u)$ ein beliebiges Netzwerk. Wie beim Simplex-Algorithmus gibt es beim Netzwerk-Simplex-Algorithmus zwei Phasen. In der ersten wird eine initiale zulässige Baumlösung (T_0, f_0) auf N erzeugt; die beiden etablierten Vorgehensweisen werden in Abschnitt 2.2.3 beschrieben. Die Problematik einer Instanz ohne Lösung wird ebenfalls dort behandelt. Die zweite Phase iteriert folgende Vorgehensweise:

Betrachte Iteration i mit zulässiger Baumlösung (T_i, f_i). Wähle einen negativen Kreis $\bar{C}_{T,e}$. Existiert kein solcher, beende den Algorithmus. Andernfalls augmentiere $\bar{C}_{T,e}$ größtmöglich zum maximalen Fluss f_{i+1}; jetzt existiert nach Lemma 8 eine Kante $e \neq e' \in C_{T,e}$ mit $f_{i+1}(e') = 0$. Die neue zulässige Baumlösung sei dann $(T_{i+1} = T_i \backslash \{e'\} \cup \{e\}, f_{i+1})$. Die verschiedenen Möglichkeiten zur Wahl von e und TODO wirklich? e' werden in Abschnitt 2.2.1 und Abschnitt 2.2.2 näher beleuchtet.

Wir werden nun beweisen, dass der Algorithmus eine optimale Lösung des Transportproblems gefunden hat, wenn Phase 2 in Ermangelung einer geeigneten Kante e beendet wird.

Notation. Sei $N = (G, b, c, u)$ ein Netzwerk mit einem maximalen Fluss f, T ein aufspannender Baum von G und $v \neq w \in V(G)$ zwei beliebige verschiedene Knoten. Mit $v \xrightarrow{T} w$ bezeichnen wir den eindeutigen gerichteten Weg von v nach w in $R_{N,f}$, der nur über Kanten $e \in E(T)$ oder deren Residualkanten $\bar{e}$ verläuft. Insbesondere ist dieser Weg unabhängig von f und für die Kosten gilt $c(w \xrightarrow{T} v) = -c(v \xrightarrow{T} w)$.

Theorem 11. *Sei N ein Netzwerk mit einer zulässigen Baumlösung (T, f). Existiert kein negativer Kreis $\bar{C}_{T,e}$, so ist f eine optimale Lösung.*

Beweis. Wir werden zeigen, dass ein negativer Kreis $\bar{C}_{T,e}$ existiert, wenn die betrachtete zulässige Baumlösung (T, f) nicht optimal ist.

Sei $N = (G, b, c, u)$ ein Netzwerk mit einer optimalen zulässigen Baumlösung $(\hat{T}, \hat{f})$ und einer zulässigen Baumlösung (T, f), sodass $c(f) > c(\hat{f})$. Da $\hat{f}$ günstiger ist, existieren zwei verschiedene Knoten $x \neq y \in E(G)$, deren Weg $\hat{w} := x \xrightarrow{\hat{T}} y$ günstiger ist als $w := x \xrightarrow{T} y$. Der geschlossene Kantenzug $W := x \xrightarrow{\hat{T}} y \xrightarrow{T} x$ besitzt damit Kosten $c(W) = c(\hat{w}) - c(w) < 0$. Zunächst sorgen wir dafür, dass aus dem geschlossenen Kantenzug mit negativen Kosten ein negativer Kreis wird.

W kann in kantendisjunkte Kreise $C_1, \ldots, C_k$ zerlegt werden, womit auch die Kosten $c(W) = \sum_{i=1}^{k} c(C_i)$ aufgeteilt werden. Für jeden Kreis C_i gibt es zwei Knoten $x_i \neq y_i$ mit $C_i = x_i \xrightarrow{\hat{T}} y_i \xrightarrow{T} x_i$. Da $c(W) < 0$, existiert ein Kreis C_j mit negativen Kosten. Wir werden im Folgenden nur noch C_j betrachten, also können wir $C := C_j$, $x := x_j$ und $y := y_j$ setzen.

□

2.2.1 Pivotalgorithmen

Obv.

2.2.2 Degenerierte Iterationen

Cycling und Stalling

2.2.3 Initialisierung

Geht nicht früher, weil LC degenerierte Iterationen braucht. Außerdem unlösbare Instanzen.

2.3 Implementation

Hier beginnt mein schönes Werk ...

2.3.1 Spezielle Konstrukte

... und hier endet es.

Die Klasse Circle

Kreise halt.[1]

Der Rest halt

Kleinkram.

Kapitel 3

Experimentelle Ergebnisse

Meh.

Kapitel 4

Ausblick

La la la.

Literaturverzeichnis

[1] V. Chvátal, *Linear Programming*, pp. 291 ff. Series of books in the mathematical sciences, W. H. Freeman, 16 ed., 2002.

[2] N. Zadeh, "A bad network problem for the simplex method and other minimum cost flow algorithms," *Mathematical Programming*, vol. 5, no. 1, pp. 255–266, 1973.

[3] G. B. Dantzig, "Application of the simplex method to a transportation problem," in *Activity Analysis of Production and Allocation* (T. C. Koopmans, ed.), ch. XXIII, pp. 359–373, New York: Wiley, 1951.

[4] A. Orden, "The transhipment problem," *Management Science*, vol. 2, no. 3, pp. 276–285, 1956.