Eine graphentheoretische Herleitung und Implementierung des Netzwerk-Simplex-Algorithmus

Max Kanold

21. Oktober 2018

Inhaltsverzeichnis

1 Einführung

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 von [2, Dantzig, 1951] und [3, Orden, 1956] vereinfachte Netzwerk-Simplex-Algorithmus. Auch für diesen gibt es exponentielle Instanzen (siehe Kapitel 4), in der Praxis wird er trotzdem vielfach verwendet.

TODO: Die Einführung wird ohnehin ausgebaut. Irgendwie erwähne ich auch unauffällig, dass in dieser Bachelorarbeit die Laufzeit des Algorithmus als Iterationsanzahl definiert ist, obwohl in der Praxis natürlich die Gesamtgeschwindigkeit zählt.

2 Netzwerk-Simplex-Algorithmus

Zunächst führen wir in Abschnitt 2.1 das Transportproblem und dessen Verallgemeinerung auf beschränkte Kapazitäten ein. Gleichzeitig geben wir die Einschränkungen an, auf denen insbesondere der Programmierteil dieser Bachelorarbeit fußt. Die Beschreibung des Netzwerk-Simplex-Algorithmus in Abschnitt 2.2 orientiert sich zuerst an [1, S. 291 ff.] zur Lösung des Transportproblems, danach wird in Abschnitt 2.3 der Algorithmus anhand von [1, S. 353 ff.] auf den allgemeinen Fall erweitert.

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 positivem b-Wert werden als Quellen, solche mit negativem als Senken bezeichnet. Knoten mit neutralem b-Wert werden Transitknoten genannt.

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:

$$\begin{aligned} &\text{(i)} && \forall e \in E : f(e) \leq u(e) \\ &\text{(ii)} && \forall v \in V : \sum_{(w,v)\in E} f((w,v)) - \sum_{(v,w)\in E} f((v,w)) + b(v) = 0 \end{aligned} \tag{2.1}$$

Die *Kosten* von f betragen $c(f) = \sum_{e\in E} f(e) \cdot c(e)$.

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 b auf $\mathbb{Z}$ sowie c und u auf $\mathbb{N}_0$ abbilden, um Gleitkommazahlungenauigkeit beim Programmieren zu vermeiden. Durch eine entsprechende Skalierung des Problems können die Funktionen nach $\mathbb{R}$ bzw. $\mathbb{R}_{\geq 0}$ hinreichend genug angenähert werden. Es wird davon ausgegangen, dass $\sum_{v\in V(G)} b(v) = 0$ ist, Angebot und Nachfrage also ausgeglichen sind. Der Fall eines übermäßigen Angebotes kann durch eine Dummy-Senke abgefangen 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. Des Weiteren ist in der konkreten Implementierung E keine Multimenge; es sind keine parallelen Kanten vorgesehen. Alle Netzwerke werden als zusammenhängend angenommen, da die Zusammenhangskomponenten einer Instanz einzeln gelöst werden können. Der programmierte Algorithmus verlangt keinen Zusammenhang.

Sämtliche Bilder von Graphen im Verlauf dieser Bachelorarbeit sind nach folgender Legende zu lesen. Teilweise werden nur einzelne Werte angezeigt, Ausnahme ist die Kapazität, die nie ohne Flusswerte auftritt.

- b-Wert: schwarz im Knoten
- Fluss: blau auf Kante
- Kapazität: schwarz auf Kante, mit Slash (/) vom Fluss abgetrennt
- Kosten: rot auf Kante, falls notwendig mit Komma (,) abgetrennt

2.2 Der grundlegende Algorithmus

Um die Funktionsweise des Netzwerk-Simplex-Algorithmus zu definieren, benötigen wir zuerst einige grundlegende Definitionen der Graphentheorie. Im Nachfolgenden sind nur die wichtigsten in Kürze aufgeführt, für eine vollständige Einführung sei der geneigte Leser auf [4] verwiesen.

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 $N = (G, b, c, u)$ eine Instanz des Transportproblems. Ein aufspannender Baum T von G und ein maximaler Fluss f auf N bilden eine **zulässige Baumlösung** (T, f), wenn für alle Kanten $e \in E(G) \backslash E(T)$ außerhalb des Baumes $f(e) = 0$ gilt.

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

Beweis. Sei $(T = (V, E), f)$ eine zulässige Baumlösung einer Instanz des Transportproblems. 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 Gleichung (2.1) 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 $T' = (V' \subsetneq V, E' \subsetneq E)$ bleibt weiterhin ein aufspannender Baum und $f_{|E'}$ ein maximaler Fluss. Durch Iteration ist f wohldefiniert und eindeutig. □

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.

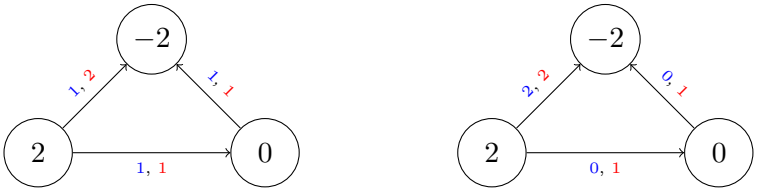

Abbildung 2.1: Links ein maximaler Fluss ohne zulässige Baumlösung, rechts ein maximaler Fluss mit uneindeutiger zulässiger Baumlösung.

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. 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 $C \subseteq G$, erhöhen also für alle Kanten $e \in E(C)$ den Fluss um einen festen Betrag $\delta \in \mathbb{N}_0$, ohne Kapazitätsschranken zu verletzen, erhalten wir einen neuen maximalen Fluss f' in N. Seine Kosten betragen $c(f') = c(f) + \delta \cdot c(C)$.

Lemma 8. *Sei N eine Instanz des Transportproblems, $R_{N,f}$ ihr 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 $\bar{e} \in E(C)$, sodass die korrespondierende Kante e einen Fluss von $f'(e) = 0$ besitzt.*

Beweis. Sei $N = (G, b, c, u)$ ein Instanz des Transportproblems mit maximalen Fluss f und C ein negativer Kreis in $R_{N,f}$. Da $c : E(G) \to \mathbb{N}_0$ 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 $\delta := \min_{e \in E(C)}\{\bar{u}(e) - \bar{f}(e)\}$ größtmöglich gewählt. Da alle Kanten $e \in E(G)$ 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 $H_f = (V', \{e \in E' | f(e) \neq 0\})$ 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 N eine Instanz des Transportproblems mit einem maximalen Fluss f. 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 Instanz des Transportproblems 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, setzen wir $\hat{f} := f_i$ und sind fertig. Ansonsten gibt es einen ungerichteten Kreis $C \subseteq G_{f_i}$. 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 jede lösbare Instanz des Transportproblems existiert eine zulässige Baumlösung (T, f), sodass der maximale Fluss f minimale Kosten hat.* □

Notation. Sei $N = (G, b, c, u)$ ein Netzwerk mit maximalen Fluss f, T ein aufspannender Baum von G, $R_{N,f} = (\bar{G}, \bar{f}, b, \bar{c}, \bar{u})$ der Residualgraph und $e \in E(\bar{G}) \backslash E(T)$ 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 $\bar{G}$ zu $C_{T,e}$, der e enthält.

Sei $N = (G, b, c, u)$ eine Instanz des Transportproblems. 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 $\bar{C}_{T,e}$ mit $e \in E(G) \backslash E(T)$. 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 $e \neq e' \in E(C_{T,e})$ 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 e' werden in Abschnitt 2.2.2 und Abschnitt 2.2.1 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. Lemma 11 ist etwas abstrakter gehalten, damit es noch mal in Abschnitt 2.3 Verwendung finden kann.

Notation. Sei $N = (G, b, c, u)$ ein Netzwerk mit einem maximalen Fluss f, T ein Wald von G und $v, w \in V(G)$ zwei beliebige 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. Sollten v und w in T nicht verbunden sein, entspricht der Weg dem leeren Graph. Insbesondere ist dieser Weg unabhängig von f und für die Kosten gilt $c(w \xrightarrow{T} v) = -c(v \xrightarrow{T} w)$.

Lemma 11. *Sei $N = (G, b, c, u)$ ein Netzwerk mit einer zulässigen Baumlösung (T, f) und C ein ungerichteter Kreis in G, sodass für einen korrespondierenden gerichteten Kreis $\bar{C}$ in $R_{N,f}$ gilt:*

(i) $c(\bar{C}) < 0$

(ii) $\forall e \in E(C)\backslash E(T)$: *die korrespondierende Kante* $\bar{e} \in E(\bar{C})$ *darf vom Netzwerk-Simplex-Algorithmus mit zulässiger Baumlösung* (T, f) *als eingehende Kante gewählt werden*

Dann existiert ein vom Algorithmus wählbares $\bar{e} \in \bar{C}$*, sodass* $c(\bar{C}_{T,\bar{e}}) < 0$.

Beweis. Seien N ein Netzwerk, (T, f) eine zulässige Baumlösung und C mit $\bar{C}$ zwei Kreise wie gefordert. Sei $E_{\bar{C},\neg T} \subseteq E(\bar{C})$ die Menge der korrespondierenden Kanten in $\bar{C}$ von den Nichtbaumkanten $E(C)\backslash E(T)$ und $E_{\bar{C},T} = E(\bar{C})\backslash E_{\bar{C},\neg T}$ dessen Komplement. Betrachten wir zunächst den Fall, dass $E_{\bar{C},T} = \emptyset$:

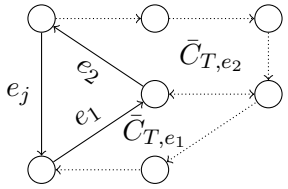

Abbildung 2.2: Die durchgezogenen Kanten bilden den negativen Kreis $\bar{C}$. Jede Kante $(v, w) \in E(\bar{C})$ wird über $w \xrightarrow{T} v$ zu einem Kreis ergänzt.

Nach Eigenschaft (ii) gibt es für alle Kanten $e_1, \ldots, e_c \in E_{\bar{C},\neg T} = E(\bar{C})$ einen Kreis $\bar{C}_{T,e_i}$. Insbesondere existiert eine Kante e_j, sodass die Kosten von $\hat{C} := \bar{C}_{T,e_j}$ folgendermaßen definiert sind:

$$c(\hat{C}) = c(\bar{C}) - \sum_{i \neq j} c(\bar{C}_{T,e_i}) \tag{2.2}$$

Die Existenz von $\hat{C}$ wird von Abb. 2.2 veranschaulicht und liegt darin begründet, dass T sonst nicht kreisfrei wäre. Sollte einer der Kreise $\bar{C}_{T,e_i}$ für $i \neq j$ negative Kosten aufweisen, ist die Aussage direkt gezeigt. Andernfalls besitzt $\hat{C}$ nach Gleichung (2.2) negative Kosten und ist der gesuchte Kreis.

Kommen wir zum Fall $E_{\bar{C},T} \neq \emptyset$. Da T ein Baum ist, ist $E_{\bar{C},\neg T} \neq \emptyset$. Sollte $|E_{\bar{C},\neg T}| = 1$ sein, ist $\bar{C}$ der gesuchte Kreis. Ansonsten werden wir $\bar{C}$ iterativ derart zu einem Kreis $\hat{C}$ verändern, dass $E_{\hat{C},\neg T} \subseteq E_{\bar{C},\neg T}$ und $|E_{\hat{C},\neg T}| = 1$. Besitzt $\hat{C}$ negative Kosten, sind wir wieder fertig, andernfalls werden wir zwischendurch bereits einen negativen Kreis $\bar{C}_{T,e}$ mit $e \in E_{\bar{C},\neg T}$ gefunden haben.

Betrachte einen Iterationsschritt mit negativen Kreis $\bar{C}$ mit $|E_{\bar{C},\neg T}| > 1$. Sei $e = (x, y) \in E_{\bar{C},\neg T}$ eine beliebige Nichtbaumkante. Wir betrachten nun den Kreis $\tilde{C} := \bar{C}_{T,e} = y \xrightarrow{T} x \cup \{(x, y)\}$ und den Kantenzug $W := x \xrightarrow{T} y \xrightarrow{\bar{C}-\{e\}} x$. Letzterer zerfällt bereinigt um in beide Richtungen begangene Kanten in kantendisjunkte Kreise $C_1, \ldots, C_c$ mit $c(W) = \sum_{i=1}^{c} c(C_i)$. O. B. d. A. sei die Kante e in C_1 enthalten.

Sollte $c(\tilde{C}) < 0$ sein, ist $\tilde{C}$ nach Eigenschaft (ii) der gesuchte Kreis. Ist einer der Kreise $C_2, \ldots, C_c$ negativ, ersetzen wir $\bar{C}$ durch diesen Kreis. Andernfalls gilt für den

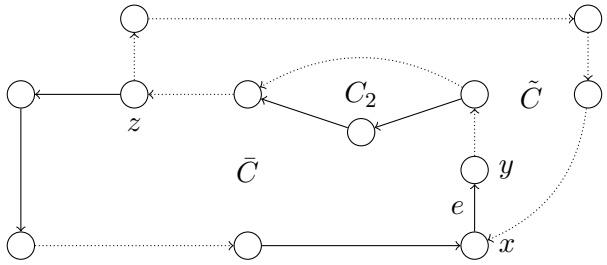

Abbildung 2.3: Der neue Kreis C_1 entsteht hier, indem wir von x beginnend entgegen $\tilde{C}$ bis z gehen und dann dem bisherigen Kreisverlauf $\bar{C}$ folgen.

Kreis C_1:

$$|E_{C_1,\neg T}| < |E_{\bar{C},\neg T}|$$
$$c(C_1) = c(\bar{C}) - c(\tilde{C}) - \sum_{i=2}^{c} c(C_i) \leq c(\bar{C}) < 0$$

Damit können wir $\bar{C} := C_1$ setzen. In letzteren beiden Fällen ist $|E_{\bar{C},\neg T}|$ im Vergleich zum Anfang der Iteration echt kleiner geworden. Wir iterieren weiter, bis $|E_{\bar{C},\neg T}| = 1$ ist. Dann ist $\hat{C} := \bar{C}$ der gesuchte Kreis. □

Theorem 12. *Sei N eine Instanz des Transportproblems 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. Dazu finden wir einen gerichteten negativen Kreis, für den wir Lemma 11 anwenden können.

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 $x \neq y \in E(G)$, sodass für die beiden Wege $\hat{w} := x \xrightarrow{\hat{T}} y$ und $w := x \xrightarrow{T} y$ gilt: $c(\hat{w}) < c(w)$ und alle Kanten $\hat{e} \in E(\hat{w})$ besitzen einen Fluss $f(\hat{e}) \neq 0$. Der geschlossene Kantenzug $W := x \xrightarrow{\hat{T}} y \xrightarrow{T} x$ besitzt damit Kosten $c(W) = c(\hat{w}) - c(w) < 0$ und es gilt $E(\hat{w}) \subseteq E(G)$, womit jede Kante $\hat{e} \in E(\hat{w}) \backslash E(T)$ vom Netzwerk-Simplex-Algorithmus gewählt werden darf. Nun sorgen wir dafür, dass aus dem geschlossenen Kantenzug mit negativen Kosten ein negativer Kreis wird.

Wie Abb. 2.4 veranschaulicht, kann W 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. Da $c(W) < 0$, existiert ein Kreis C_j mit negativen Kosten. C_j erfüllt also Eigenschaft (i) und nach obiger Betrachtung Eigenschaft (ii) von Lemma 11, damit können wir Lemma 11 für N, (T, f) und $\bar{C} := C_j$ anwenden. □

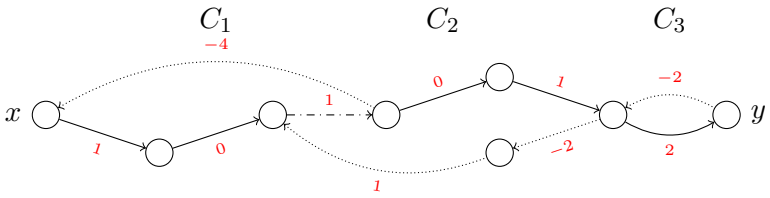

Abbildung 2.4: Zerlegung eines negativen Kantenzugs $W = x \xrightarrow{\hat{T}} y \xrightarrow{T} x$

Korollar 13. *Sei N eine Instanz des Transportproblems mit einer zulässigen Initialbaumlösung (T_0, f_0). Determiniert der Netzwerk-Simplex-Algorithmus, so ist er korrekt.* □

Bislang haben wir nicht gezeigt, dass der Algorithmus immer terminiert. Dies wäre offensichtlich, wenn bei jeder Iteration von (T, f) zu (T', f') die Kosten sinken würden, also $c(f') < c(f)$ wäre. Es gibt jedoch sogenannte *degenerierte Iterationen*, in denen ein negativer Kreis $\bar{C}_{T,e}$ um $\delta = 0$ augmentiert wird. Entfernt man danach eine Kante $e \neq e' \in E(C_{T,e})$, verändert sich nur der Baum. Der nächste Abschnitt beschäftigt sich damit, wie sichergestellt werden kann, dass der Algorithmus zumindest jeden Baum höchstens einmal betrachtet.

2.2.1 Degenerierte Iterationen

Definition 14. Wird in einer Iteration der Phase 2 des Netzwerk-Simplex-Algorithmus ein negativer Kreis $\bar{C}_{T,e}$ um $\delta = 0$ augmentiert, so bezeichnen wir dies als **degenerierte Iteration**.

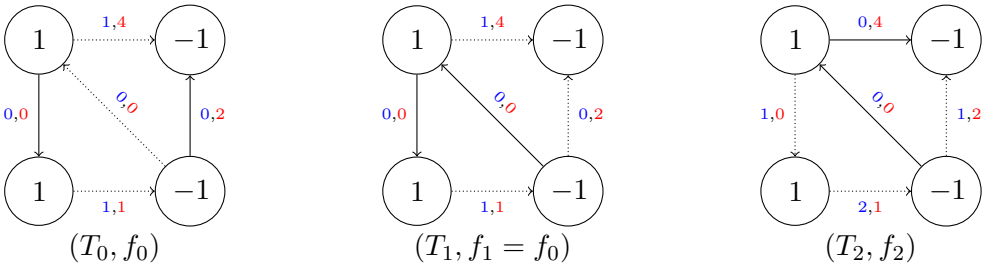

Abbildung 2.5: Die Kanten des Baumes T_i sind gestrichelt, der gewählte negative Kreis ist jeweils eindeutig. Die Startlösung kann erst nach einer degenerierten Iteration verbessert werden.

Degenerierte Iterationen entstehen, wenn bei einer zulässigen Baumlösung (T, f) nicht alle Kanten von T Fluss aufweisen, also $T \neq T_f$ ist. (T, f) wird dann auch als *degeneriert* bezeichnet. In einer ungünstigen Konstellation von der deterministischen Wahl der hinzugefügten Kante e und entfernten Kante e' kann es zum *Cycling* kommen,

also zu einem Kreisschluss von Bäumen, die wiederholt iteriert werden. Dies tritt sehr selten auf, für ein künstlich konstruiertes Beispiel siehe [1, S. 303].

[5, Cunningham, 1976] führte eine Methode ein, mit der Cycling verhindert werden kann, ohne dass die Wahl der hinzugefügten Kante e eingeschränkt wird. Dafür benötigen wir folgende Definition:

Definition 15. Sei $N = (G, b, c, u)$ eine Instanz des Transportproblems. Ein aufspannender Baum T von G, ein maximaler Fluss f auf N und ein Wurzelknoten $r \in V(T) = V(G)$ bilden eine **stark zulässige Baumlösung** $(T, f)_r$, wenn (T, f) eine zulässige Baumlösung ist und zusätzlich jede Kante $e = (v, w) \in E(T)$ mit $f(e) = 0$ von der Wurzel weggerichtet, also e im Weg $r \xrightarrow{T} w$ enthalten ist.

TODO: An dieser Stelle folgt ein Absatz, der erklärt, wie genau die ausgehende Kante abhängig vom Kreis und dem Wurzelknoten gewählt werden muss. Es folgt das Lemma, dass dann der Algorithmus terminiert, für den Beweis wird auf die Literatur verwiesen. Dieser Teil fehlt momentan noch, da ich das noch mit meinem programmierten Algorithmus abgleichen muss. Außerdem stimmten die Informationen im nächsten Absatz nicht bzw. sind unvollständig.

Die Laufzeit des Netzwerk-Simplex-Algorithmus ist bislang ungeklärt. Für bestimmte Varianten wurden exponentielle Instanzen gefunden, die auf *Stalling* basieren, also einer exponentiellen Anzahl degenerierter Iterationen. Mit diesen werden wir uns in Kapitel 4 näher befassen. Meine eigene, experimentelle Suche nach schlechten Instanzen findet sich in Kapitel 5. Zunächst vervollständigen wir den Algorithmus um die Wahl des negativen Kreises $\bar{C}_{T,e}$ und die Erzeugung einer initialen Baumlösung.

2.2.2 Pivotalgorithmen

Sei $N = (G, b, c, u)$ ein Netzwerk mit zulässiger Baumlösung (T, f). Algorithmen, die aus der Menge $\bar{C}_T = \{\bar{C}_{T,e} | c(\bar{C}_{T,e}) < 0)\}$ aller möglichen Iterationen eine auswählen, heißen *Pivotalgorithmen*. In der Praxis wird der Pivotalgorithmus nur auf einer Teilmenge von $\bar{C}_T$ ausgeführt, um Rechenzeit zu sparen. In dieser Bachelorarbeit werden nur drei naheliegende, einfache Pivotalgorithmen betrachtet.

Maximum Value

Der erste Ansatz ist es, den negativsten Kreis zu wählen, sprich

$$\tilde{C} := \underset{\bar{C}_{T,e} \in \bar{C}_T}{\arg\min} \{c(\bar{C}_{T,e})\}$$

Diesen Weg werden wir mit *MaxVal* bezeichnen.

Maximum Revenue

Die Kostenverringerung nach der Augmentierung beträgt $\delta \cdot c(\bar{C}_{T,e})$, ist also von δ abhängt. Der Pivotalgorithmus *MaxRev* maximiert diesen Wert:

$$\tilde{C} := \underset{\bar{C}_{T,e} \in \bar{C}_T}{\arg\min}\{\delta_e \cdot c(\bar{C}_{T,e})\} \qquad \delta_e := \min_{e' \in E(\bar{C}_{T,e})} \{\bar{u}(e') - \bar{f}(e')\}$$

Nach Lemma 8 ist jedes δ_e endlich. Sollten nur degenerierte Iterationen zur Auswahl stehen, ist $c(\tilde{C}) = 0$. In dem Fall wendet meine konkrete Implementierung MaxVal an; hier sind aber auch andere Strategien denkbar.

Random

Ein überraschend effektiver Ansatz ist es, $\tilde{C} \in \bar{C}_T$ zufällig zu wählen. Gerade für diesen mit *Random* bezeichneten Weg ist es schwierig, untere oder obere Schranken zu beweisen.

2.2.3 Initialisierung

Der letzte verbleibende Schritt, um einen vollständigen Algorithmus zur Lösung des Transportproblems zu erlangen, ist das Finden einer initialen, stark zulässigen Baumlösung $(T_0, f_0)_r$. Hierfür wenden wir einen Trick an:

Sei $N = (G, b, c, u)$ eine Instanz des Transportproblems. Wir fügen dem Netzwerk einen zusätzlichen Transitknoten a mit $b(a) = 0$ hinzu, den man als künstlichen *(artificial)* Knoten bezeichnet. Für alle Quellen $v \in V(G) : b(v) > 0$ ergänzen wir eine künstliche Kante (v, a), für alle Senken und Transitknoten $a \neq w \in V(G) : b(w) \leq 0$ eine künstliche Kante (a, w). Sei G' der entstehende Graph.

Sämtliche künstlichen Kanten e_v haben eine unbegrenzte Kapazität, die wir mit $u(e_v) = \frac{1}{2} \cdot \sum_{v \in V(G)} |b(v)| + 1$ abschätzen. Die Kosten sind ebenfalls unendlich und können mit $c(e_v) = |V(G)| \cdot \max_{e \in E(G)}\{c(e)\} + 1$ abgeschätzt werden, da jeder Weg in G geringere Kosten aufweist. Leicht finden wir nun die stark zulässige Baumlösung $(T_0, f_0)_a$, wobei $T_0 = (V(G), \{e_v | v \in V(G)\})$ ist. Nach Lemma 6 ist f_0 eindeutig. Diese Art der Initialisierung werden wir im Folgenden mit *HC* für *High-Cost-Initialisierung* abkürzen.

Lemma 16. *Sei $N = (G, b, c, u)$ eine Instanz des Transportproblems und (T', f) die Lösung des Netzwerk-Simplex-Algorithmus mit* HC *auf dem erweiterten Graphen G'. Die Instanz N ist genau dann lösbar, wenn G'_f keine künstlichen Kanten enthält.*

Beweis.

„⇒“ Sei $\hat{f}$ ein maximaler Fluss von N und (T', f) die Lösung des Algorithmus. $\hat{f}$ ist auch ein maximaler Fluss des Netzwerkes (G', b, c, u); damit gilt nach Theorem 9 und Korollar 13 $c(f) \leq c(\hat{f}) < \infty$. Da die Kosten einer künstlichen Kante äquivalent zu unendlich sind, kann keine davon in G'_f enthalten sein.

„⇐“ Sei (T', f) die Lösung des Algorithmus. Wenn G'_f keine künstlichen Kanten enthält, ist $f_{|E(G)}$ ein maximaler Fluss von N, womit diese Instanz wiederum lösbar ist. □

Betrachtet man Instanzen mit vielen Knoten oder hohen Kosten der teuersten Kante, so kann es passieren, dass die Kosten der künstlichen Kanten den darstellbaren Bereich der gängigen Datentypen sprengen. Unter anderem aus diesem Grund gibt es die *Low-Cost-Initialisierung*, bei uns kurz *LC*.

Bei dieser wird die gegebene Instanz $N = (G, b, c, u)$ des Transportproblems zunächst zu $N' = (G, b, c', u)$ abgewandelt. Dabei ist c' die Nullfunktion, also $c(e) = 0$ für jede Kante $e \in E(G)$. Für N' verwenden wir den Netzwerk-Simplex-Algorithmus mit *HC* und erhalten eine stark zulässige Baumlösung $(T', f')_a$, wobei die Kosten einer künstlichen Kante $c(e_v) = |V(G)| \cdot 0 + 1 = 1$ betragen.

Offensichtlicherweise ist N genau dann lösbar, wenn N' lösbar ist. Wenn $c(f') > 0$ ist, so ist N nicht lösbar und wir sind fertig. Andernfalls iterieren wir $(T', f')_a$ degeneriert, bis a ein Blatt von T' ist. Dies wird in Abschnitt 3.5 detailliert erläutert.

Damit bekommen wir die zulässige Baumlösung $(T = T' - a, f = f'_{|E(G)})$ für N' und damit für N. Als neuen Wurzelknoten können wir den eindeutigen Nachbar v_a von a in T' wählen; damit bekommen wir die stark zulässige Baumlösung $(T, f)_{v_a}$.

2.3 Erweiterung auf beschränkte Kapazitäten

Wir werden nun den Netzwerk-Simplex-Algorithmus auf das Min-Cost-Flow-Problem erweitern. Die Funktionsweise bleibt dieselbe, nur die betrachteten zulässigen Baumlösungen verändern sich leicht:

Definition 17. 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 $\forall e \in E(G) \backslash E(T) : f(e) = 0 \vee f(e) = u(e)$.

Anmerkung. Kanten $e \in E(G)$, die maximal durchflossen sind, bezeichnen wir auch als *saturiert*.

Definition 18. Sei $N = (G, b, c, u)$ ein Netzwerk. Ein aufspannender Baum T von G, ein maximaler Fluss f auf N und ein Wurzelknoten $r \in V(T) = V(G)$ bilden eine **stark zulässige Baumlösung** $(T, f)_r$, wenn (T, f) eine zulässige Baumlösung ist und folgendes gilt:

$$\forall e = (v, w) \in E(T) : (f(e) = 0 \Rightarrow e \in E(r \xrightarrow{T} w)) \quad \wedge$$
$$(f(e) = u(e) \Rightarrow e \in E(w \xrightarrow{T} r))$$

Anmerkung. An dieser Stelle sollten sämtliche Kanten $e \in E(G)$ mit $u(e) = 0$ aus dem Graphen entfernt werden; sie sind für das Problem ohnehin irrelevant.

Abbildung 2.6: Derselbe, durch gestrichelte Linien dargestellte Baum mit unterschiedlichen maximalen Flüssen. Der Wurzelknoten ist die Senke.

Im Falle beschränkter Kapazitäten erlauben wir für eine zulässige Baumlösung (T, f) auch Fluss außerhalb von T, insofern er die Kapazität der Kante voll ausnutzt. Abb. 2.6 veranschaulicht, warum dies notwendig ist. Solche saturierten Kanten $e \in E(G)\backslash E(T)$ sind als eingehende Kante keine sinnvolle Wahl mehr, stattdessen kann der Algorithmus über den Kreis $\bar{C}_{T,\bar{e}}$ der Residualkante $\bar{e}$ augmentieren, sofern er negativ ist. Für stark zulässige Baumlösungen müssen nun zusätzlich alle saturierten Kanten des Baumes zum Wurzelknoten hingerichtet sein.

Die in Lemma 6 bewiesene Eindeutigkeit von f gilt nun nicht mehr (siehe Abb. 2.6), eine Eigenschaft, die wir bei der Initialisierung nutzten. Der Leser kann sich an dieser Stelle davon überzeugen, dass diese trotzdem komplett identisch durchführbar ist.

Wir werden jetzt Lemma 8, Theoreme 9 und 12 und Korollar 10 auf das Min-Cost-Flow-Problem erweitern.

Lemma 19. *Sei N ein Netzwerk, $R_{N,f}$ sein 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 $\bar{e} \in E(C)$, deren korrespondierende Kante e einen Fluss von $f'(e) = 0$ besitzt, oder eine Kante $e \in E(C) \cap E(G)$ mit $f'(e) = u(e)$.*

Beweis. Sei $N = (G, b, c, u)$ ein Netzwerk mit maximalen Fluss f und C ein negativer Kreis in $R_{N,f}$. δ ist analog zum Beweis von Lemma 8 endlich.

Sei nun $\tilde{e} := \arg\min_{e \in E(C)}\{\bar{u}(e) - \bar{f}(e)\}$ und $\delta := \bar{u}(\tilde{e}) - \bar{f}(\tilde{e})$ größtmöglich gewählt. Ist $\tilde{e}$ eine Residualkante, so besitzt die korrespondierende Kante e einen Fluss von

$$f'(e) = f(e) - \delta = f(e) - u_f(\bar{e}) = f(e) - f(e) = 0.$$

Andernfalls ist $\tilde{e} \in E(C) \cap E(G)$ mit einem Fluss von

$$f'(\tilde{e}) = f(\tilde{e}) + \delta = f(\tilde{e}) + \bar{u}(\tilde{e}) - \bar{f}(\tilde{e}) = \bar{u}(\tilde{e}) = u(\tilde{e}). \qquad \square$$

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 $H^f = H_f\backslash\{e \in E' | f(e) = u(e)\}$ bezeichnen wir den Graph aller durchflossenen, aber nicht saturierten Kanten.

Theorem 20. *Sei N ein Netzwerk mit einem maximalen Fluss f. 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. Diesmal werden wir f zu einem maximalen Fluss $\hat{f}$ umwandeln, sodass $G^{\hat{f}}$ ein Wald ist. Der Beweis verläuft nun komplett analog zum Beweis von Theorem 9, statt Lemma 8 benutzen wir Lemma 19. □

Korollar 21. *Für jede lösbare Instanz des Min-Cost-Flow-Problems existiert eine zulässige Baumlösung (T, f), sodass der maximale Fluss f minimale Kosten hat.* □

Theorem 22. *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 wiederum zeigen, dass ein negativer Kreis $\bar{C}_{T,e}$ existiert, wenn die betrachtete Baumlösung 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 $c(f) > c(\hat{f})$. Im Beweis zu Theorem 12 haben wir implizit genutzt, dass Kapazitätsfunktion unbeschränkt ist, indem wir die Kanten im negativen Kantenzug W, die aus $\hat{T}$ stammen, als vom Algorithmus wählbar deklariert haben. Dies gilt weiterhin, da wir einen von f nicht saturierten Weg in $\hat{T}$ wählen können. Sofern $c(f_{|E(T)}) \neq c(\hat{f}_{|E(\hat{T})})$ gilt, sind wir also fertig.

Sei nun $c(f_{|E(T)}) = c(\hat{f}_{|E(\hat{T})})$. Sei $S := E(G_f) \backslash (E(G^f) \cup E(T))$ die Menge aller saturierten Nichtbaumkanten von f. Da $c(\hat{f}) < c(f)$ gilt und die Flusskosten auf den Bäumen identisch sind, transportiert $\hat{f}$ irgendwo Fluss billiger als $f_{|S}$. Somit existiert ein negativer Kantenzug W, der nur aus Kanten $\hat{e} \in G_{\hat{f}}$ mit $f(\hat{e}) < u(\hat{e})$ – denn der günstigere Weg kann nicht voll von f abgedeckt sein – und aus Residualkanten $\bar{e}$ zu $e \in S$ besteht. Analog zu Theorem 12 gewinnen wir aus W einen negativen Kreis C, der nach Konstruktion Eigenschaft (i) und Eigenschaft (ii) von Lemma 11 erfüllt. Mit Lemma 11 für N, (T, f) und $\bar{C} := C$ ist die Aussage bewiesen. □

Korollar 23. *Der Netzwerk-Simplex-Algorithmus mit einer Initialisierung nach Abschnitt 2.2.3, einem beliebigen Pivotalgorithmus und gemäß Abschnitt 2.2.1 auf stark zulässige Baumlösungen beschränkt löst das Min-Cost-Flow-Problem korrekt.* □

3 Implementierung

Der Programmierteil dieser Bachelorarbeit implementiert den Netzwerk-Simplex-Algorithmus wie in Kapitel 2 beschrieben, wobei einzelne Strukturen verändert wurden. Der Code ist in C++ geschrieben und hält sich an den C++11-Standard. Es folgt eine vollständige Übersicht der Codebestandteile, für das Gesamtkonstrukt siehe TODO Anhang.

3.1 Netzwerke

Zur Umsetzung der Graphenstruktur habe ich eine Klasse `class Network` geschrieben, die auf `struct Node` sowie `class Edge` basiert und den Fluss händelt. Damit kann `Network` beispielsweise sicherstellen, dass nie Kanten entfernt werden, die aktuell durchflossen sind.

3.1.1 Knoten

`struct Node` hat außer einem Konstruktor keinerlei Funktionen, weswegen es nicht in einer eigenen Klasse ausgelagert, sondern in `network.h` enthalten ist.

```
struct Node {
  size_t id;
  intmax_t b_value;
  //defined by (the other) nodeID
  //always incoming and outgoing due to residual edges
  std::set<size_t> neighbours;

  //constructor
  Node (size_t _id, intmax_t _b_value)
    : id(_id), b_value(_b_value) {};
};
```

Ein Knoten wird eindeutig durch seine `size_t id` identifiziert, weitere Eigenschaften sind sein b-Wert aus $\mathbb{Z}$ und seine Nachbarknoten. Letztere Menge differenziert nicht zwischen Nachbarn aus eingehenden und ausgehenden Knoten, da diese Unterscheidung im Residualgraphen entfällt.

3.1.2 Kanten

Die Klasse `class Edge` enthält alle Informationen über Kanten: Kosten, Kapazität, Fluss und die jeweiligen Endknoten. Residualkanten werden durch das Flag `isResidual`

gekennzeichnet. An dieser Stelle fällt auf, dass die Kapazität nicht nachträglich verändert werden kann, obwohl dies für Residualkanten notwendig wäre. Ich habe mich in meiner Implementierung dafür entschieden, auf Residualkanten Fluss zu erlauben und damit eine veränderliche Kapazität zu simulieren. Dies wird in Abschnitt 3.1.3 näher erläutert.

```
class Edge {
public:
  //edges are initialized with a flow of zero
  Edge (size_t node0, size_t node1, intmax_t cost,
        intmax_t capacity, bool isResidual = false);

  //returns true when flow change was successful
  bool changeFlow (intmax_t value);
  bool changeFlowPossible (intmax_t value);
  //toggles cost betweeen itself and 0
  void toggleCost ();

private:
  size_t node0, node1;
  intmax_t cost, capacity, flow, toggledCost;
  bool isResidual, isToggled = false;
};
```

Die Funktion `changeFlow` stellt über die Hilfsfunktion `changeFlowPossible` sicher, dass bei einer Flussveränderung keine Kapazitätsschranken verletzt werden. Mittels `toggleCost` können die Kosten einer Kante zwischen Null und ihrem eigentlichen Wert variiert werden. Somit lässt sich leicht die in Abschnitt 2.2.3 beschriebene Low-Cost-Initialisierung umsetzen, siehe dazu Abschnitt 3.5.

3.1.3 Graph

Werfen wir einen Blick darauf, wie die Knoten und Kanten in einem Netzwerk gespeichert werden. Ich habe mich dafür entschieden, die Objekte in einer `std::map` abzulegen. Die Knoten werden durch ihre `size_t id` identifiziert, Kanten durch Anfangsknoten, Endknoten und die `bool isResidual`. Ein `std::set` benötigt einen Vergleichsoperator, daher habe ich für die Kanten `custComp` definiert, das nach `size_t node0`, `size_t node1` und zuletzt `bool isResidual` vergleicht.

```
class Network {
private:
  intmax_t flow = 0, cost = 0;
  std::vector<size_t> sources, sinks, transit;

  std::map<std::tuple<size_t, size_t, bool>,
           Edge, custComp> edges;
  std::map<size_t, Node, std::less<size_t>> nodes;
```

```
};
```

Ein Netzwerk kennt außerdem seinen `intmax_t flow`, also die Anzahl an Fluss, die von den Quellen zu den Senken transportiert wird, und die dafür anfallenden Kosten `intmax_t cost`. Für den Algorithmus ist ersteres nicht relevant, da wir nur auf maximalen Flüssen arbeiten.

Der Graph G mit einem Fluss f und der dazugehörige Residualgraph $G_{N,f}$ werden nicht als zwei verschiedene Objekte der Netzwerk-Klasse geführt, sondern in einer Instanz gemeinsam gespeichert. Um dies zu bewerkstelligen, wird bei jeder Veränderung an einer Kante gleichzeitig die eindeutige Residualkante aktualisiert. Zur Veranschaulichung folgt der Quellcode von `Network::addEdge` ohne die Prüfung der Zulässigkeit.

```
bool Network::addEdge(Edge e) {
  std::tuple<size_t, size_t, bool> key
    = std::make_tuple(e.node0, e.node1, e.isResidual);

  //insert both the edge and the residual edge
  edges.insert(std::make_pair(key, e));
  e.invert();
  edges.insert(std::make_pair(invertKey(key), e));

  nodes.find(node0)->second.neighbours.insert(node1);
  nodes.find(node1)->second.neighbours.insert(node0);
  return true;
}
```

Dabei vertauscht `invertKey` die beiden Knoten und negiert `isResidual`. Die oben nicht erwähnte Funktion `Edge::invert` spiegelt die Veränderungen von `invertKey` und negiert die Kosten gemäß Definition 7. Die Kapazität wird beibehalten, stattdessen wird der Fluss auf `flow = capacity - flow;` gesetzt. Der größte Wert, um den die Residualkante $\bar{e}$ augmentiert werden kann, ist also gleich $f(e)$. Sofern wir für f nur Nicht-Residualkanten betrachten, ist diese Implementierung äquivalent zur Definition. Dieser alternative Ansatz fordert noch ein wenig Aufmerksamkeit bei der Klassenfunktion `Network::changeFlow`,[1] danach können wir bei der Umsetzung des Algorithmus größtenteils ignorieren, ob eine Kante residual ist.

Das Netzwerk wird leer oder mit einer Anzahl `size_t noOfNodes` von Transitknoten initialisiert und durch Hinzufügen von Knoten und Kanten vervollständigt. Sofern keine Knoten gelöscht werden, ist die `size_t id` bei null beginnend lückenlos aufsteigend.

```
class Network {
public:
  Network(size_t noOfNodes);

  bool addEdge(Edge e);
```

[1] Für alle Kanten e des Kreises wenden wir `e.changeFlow(f)` sowie `e.invert().changeFlow(-f)` an und aktualisieren die Kosten auf `this->cost += e.cost*f;`.

```
  //returns nodeID
  size_t addNode(intmax_t b_value = 0);

  //fails and returns 0 if there's flow left
  bool deleteEdge(size_t node0, size_t node1);
  //fails if there's flow on an edge to this node left
  bool deleteNode(size_t nodeID);

  //takes a path from a source to a sink
  //doesn't use residual edges
  bool addFlow(std::vector<size_t>& path, intmax_t flow);
  //takes a circle
  bool changeFlow(Circle& c, intmax_t flow);
};
```

Die Funktionen `deleteEdge` und `deleteNode` schlagen fehl, wenn auf den zu löschenden Kanten noch Fluss ist. So kann unter anderem durch Löschen des künstlichen Knotens überprüft werden, ob die Instanz lösbar ist. Für Phase 1 des Algorithmus nutzen wir die Funktion `addFlow`, die einen Weg von einer Quelle zu einer Senke verlangt und somit sicherstellt, dass $\sum_{v \in V(G)} b(v)$ bei unveränderter Knotenmenge gleichbleibt.

`changeFlow` wird in jeder Iteration der zweiten Phase des Netzwerk-Simplex-Algorithmus aufgerufen und augmentiert den übergebenen Kreis `Circle c` um einen beliebigen Wert, sofern dies möglich ist. Die hier genutzte `class Circle` werden wir uns nun genauer anschauen.

3.2 Die Klasse *Circle*

Magie des updaten

3.3 Stark zulässige Baumlösungen

Öhm, ja ...

3.4 Pivotalgorithmen

Sehr kurz? Probably nur Code

3.5 Initialisierung

Beweis LC

4 Exponentielle Instanzen aus der Literatur

5 Experimentelle Ergebnisse

Meh.

6 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] 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.

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

[4] S. Hougardy and J. Vygen, *Algorithmische Mathematik*. Springer-Lehrbuch, Berlin/Heidelberg: Springer Spektrum, 2015.

[5] W. H. Cunningham, "A network simplex method," *Mathematical Programming*, vol. 11, no. 1, pp. 105–116, 1976.