Call-Control in Ringnetzwerken

Michael Markl

Seminar
„Algorithmen und Datenstrukturen“

Sommersemester 2018
Universität Augsburg

1 Einführung

Bei der Optimierung von Kommunikationssystemen geht es oft darum, konkrete Probleme auf abstrakte, mathematische Problemstellungen zurückzuführen, und man gelangt so in vielen Fällen zu Problemen in Graphen. Eines dieser Probleme beschreibt Call-Control (auch Call-Admission-Control), bei dem in einem Netzwerk eine Menge von Anfragen (sog. calls) gestellt werden und jede Anfrage gemäß einer bestimmten Zielsetzung akzeptiert oder verworfen werden muss. Es soll insbesondere sichergestellt werden, dass durch das Durchführen der akzeptierten Anfragen die Bandbreiten, die das Netzwerk zur Verfügung stellt, nicht überstrapaziert werden. Eine weitere Zielsetzung könnte dann beispielsweise sein, die Anzahl der akzeptierten Anfragen zu maximieren.

Da es meist schwierig ist, über willkürliche Graphen weitreichende Aussagen zu treffen, schränkt man sich oft auf bestimmte Typen eines Graphen ein. So beschränkt sich der Artikel „Call Control in Rings“ [1] der vier Autoren Adamy, Ambühl, Anand und Erlebach auf Netzwerke, die sich als Ring- oder Kettengraph darstellen lassen.

1.1 Problemdefinition und Bezeichnungen

Im Speziellen wird hier nur das sogenannte offline Szenario behandelt. Das bedeutet, dass die Menge von Anfragen bereits zu Beginn feststeht und keine weiteren Anfragen während der Laufzeit hinzukommen.

Zunächst führen wir einige wichtige Begriffe wie den eines (ungerichteten) Netzwerks ein:

Definition 1.1 (Netzwerk)**.** Sei (V, E) ein ungerichteter Graph mit Knoten V und Kanten E, und $c : E \to \mathbb{N}$ eine Kapazitätsfunktion. Das Tupel (V, E, c) heißt (ungerichtetes) Netzwerk und eine Kante e besitzt darin die Kapazität $c(e)$.

Die Anfragen werden nun als Pfade in einem Netzwerk betrachtet. Dabei verbraucht ein Pfad eine Kapazitätseinheit einer Kante, falls er die Kante enthält. Für die Zielsetzung, die Anzahl der akzeptierten Anfragen zu maximieren, nennen wir das entstehende Problem Call-Control. Sind die Pfade gewichtet und wollen wir das Gesamtgewicht maximieren, reden wir von Weighted-Call-Control.

Definition 1.2 (Call-Control und Weighted-Call-Control)**.** Sei (V, E, c) ein ungerichtetes Netzwerk und P eine (Multi-)Menge von $m \in \mathbb{N}$ Pfaden in (V, E, c). Eine Menge $Q \subseteq P$ heißt *zulässig*, falls für jede Kante $e \in E$ die Anzahl aller Pfade in Q, die e enthalten, höchstens $c(e)$ ist.

Call-Control besteht darin, eine zulässige Menge Q maximaler Mächtigkeit zu finden. Eine solche Menge nennen wir eine *optimale Lösung* des Call-Control-Problems.

Gibt es außerdem eine Gewichtsfunktion $w\colon P \to \mathbb{R}_+$, die jedem Pfad in P ein positives Gewicht zuweist, heißt das Problem, eine zulässige Menge Q mit maximalem Gesamtgewicht $w(Q) \coloneqq \sum_{p \in Q} w(p)$ zu finden, *Weighted-Call-Control* und Q heißt *optimale Lösung* des Weighted-Call-Control-Problems.

Eine *Kette* ist ein ungerichteter Graph (V, E), der nur aus den Kanten und Knoten eines Pfades besteht, also einem Weg mit den Kanten $E = \{(v_0, v_1), \ldots, (v_{n-2}, v_{n-1})\}$, sodass $v_i \neq v_j$ für $i \neq j$. Dabei stellen wir uns vor, dass eine Kette entlang einer horizontalen Linie aufgezeichnet ist und die Knoten bei 0 beginnend von links nach rechts durchnummeriert sind. Entsprechend sagen wir, eine Kante bzw. ein Knoten liege *vor* einer anderen Kante bzw. einem anderen Knoten, falls diese(r) weiter links liegt. Jeder Teilpfad p kann in einer Kette durch zwei Endknoten identifiziert werden, einem Anfangsknoten s_p und einem Zielknoten t_p, wobei der Anfangsknoten vor dem Zielknoten liegt.

Ein *Ring* ist ein ungerichteter Graph, der nur aus den Kanten und Knoten eines Kreises besteht, d.h. aus einem Weg mit den Kanten $E = \{(v_0, v_1), \ldots, (v_{n-1}, v_n)\}$, sodass $v_0 = v_n$ und $v_i \neq v_j$ für alle anderen Paare i, j mit $i \neq j$. Betrachten wir einen Ring mit den Knoten $V = \{0, \ldots, n-1\}$ aufgezeichnet auf einer Ebene, bei dem die Knoten nach dem Uhrzeigersinn nummeriert wurden, identifizieren wir wieder jeden Pfad p im Ring durch einen Anfangsknoten s_p und einen Zielknoten t_p, wobei der Pfad alle Kanten vom Anfangsknoten aus im Uhrzeigersinn bis zum Zielknoten enthält.

1.2 Motivation und Anwendungen

Oft trifft man in praktischen Anwendungen zwar auf die online-Variante des Problems, bei der jede Anfrage schon bei Eintreffen und ohne Wissen über künftige Anfragen akzeptiert oder abgelehnt werden muss; jedoch sind die Erkenntnisse über das offline-Problem auch hier relevant: So ist es beispielsweise während der Konzeption eines Netzwerks hilfreich, die größtmögliche Belastbarkeit unter bestimmten Bedingungen untersuchen zu können. Daneben gibt es auch Netzwerke, die für Anfragen Bandbreiten-Reservierungen im Voraus erlauben und diese dann in einem vorgeschriebenen Zeitintervall abarbeiten, was das offline-Problem in jedem dieser Intervalle abbildet. Außerdem kann die optimale Lösung des offline-Problems auch als Vergleichswert bei der Bewertung von Lösungen des online-Problems genutzt werden.

Eine direkte Anwendung bietet die zyklische Ablaufplanung. Hier ist beispielsweise eine Menge an Aufgaben gegeben, die je in einem bestimmten Zeitintervall innerhalb eines Tages erledigt werden sollen. Diese Zeitintervalle dürfen sich beliebig überschneiden oder sich sogar bis in den nächsten Tag erstrecken - sie müssen aber immer kürzer als 24 Stunden andauern. Der Tag wird nun in so viele Intervalle I_i aufgeteilt, sodass das Zeitintervall jeder Aufgabe als Vereinigung aufeinanderfolgender I_i dargestellt werden kann. Außerdem stehen jeden Tag im Zeitintervall I_i genau c_i Maschinen zur Verfügung und jede Maschine kann an jeder Aufgabe arbeiten, wobei zu jeder Zeit an einer Aufgabe maximal eine Maschine arbeitet. Vereinfachend wird angenommen, dass es egal ist, welche Maschine tatsächlich an einer Aufgabe arbeitet, und dass eine Übergabe einer laufenden Aufgabe von Maschine zu Maschine, z.B. beim Wechsel eines Zeitintervalls,

ohne Weiteres möglich ist. Jedoch muss im vorgegebenen Zeitintervall zu jeder Zeit an einer Aufgabe gearbeitet werden, um sie zu erledigen. Ziel ist es nun einen Ablaufplan zu finden, der die Anzahl an Aufgaben, die an einem Tag bearbeitet werden, maximiert. Dieser Ablaufplan soll dann fortlaufend über mehrere Tage hinweg eingesetzt werden können. Dazu kann Call-Control verwendet werden: Das Ringnetzwerk ist durch die Kanten I_i mit den Kapazitäten c_i der verfügbaren Maschinen zu jedem Zeitintervall gegeben. Die dazugehörigen Aufgaben stellen die Anfragen im Netzwerk dar, die als Pfade auf dem Ring die Zeitintervalle der Aufgaben abbilden. Löst man nun diese Instanz des Call-Control-Problems, erhält man eine größtmögliche Menge an Pfaden, die keine Kapazitäten verletzt, und dadurch auch die größtmögliche Menge an Aufgaben, die an einem Tag erledigt werden können.

2 Optimale Lösung des Call-Control-Problems in Ketten

Dieser Abschnitt analysiert den einfacheren Fall des Problems, nämlich den in einer Kette: Sei (V, E) eine Kette mit Kapazitäten $c : E \to \mathbb{N}$ und $V = \{0, \dots, n\}$, sodass für $i \in \{0, \dots, n-1\}$ die Kante e_i die Knoten i und $i+1$ verbindet. Außerdem sei eine Menge P von Pfaden in (V, E) gegeben.

Definition 2.1 (Gierige Ordnung)**.** Auf einer Menge P von Pfaden in einer Kette nennen wir eine Totalordnung $\leq_G$ sowie die zugehörige strenge Totalordnung Ordnung $<_G$ *gierig*, falls sie die Pfade nach ihren Zielknoten aufsteigend ordnet, das heißt, falls $\forall p, q \in P\colon t_p < t_q \Rightarrow p <_G q$.

Im Folgenden bezeichne $\leq_G$ eine feste gierige Ordnung auf P. Die Berechnung der optimalen Lösung für Call-Control in der Kette erfolgt mit dem *gierigen Verfahren*: Dieses nimmt an, dass die Pfade P bereits in gieriger Ordnung $\leq_G$ sortiert sind, und verwaltet eine Menge der von ihm akzeptierten Pfade, die zu Beginn leer ist. Es bearbeitet alle Pfade in der gegebenen Sortierung und fügt einen Pfad zu den akzeptierten Pfaden hinzu, falls die akzeptierten Pfade dadurch keine der Kantenkapazitäten verletzen, das heißt, falls die akzeptierten Pfade eine zulässige Menge bleiben. Ansonsten wird der Pfad abgelehnt. Nachdem alle Pfade bearbeitet wurden, endet das Verfahren mit den akzeptierten Pfaden als Rückgabe.

Jede Menge $A \subseteq P$ von k Pfaden kann als eine Menge $\{a_1, \dots, a_k\}$ dargestellt werden mit $a_1 <_G \cdots <_G a_k$. Für solche $A = \{a_1, \dots, a_k\}$ und $B = \{b_1, \dots, b_k\}$ schreibe man dann $A \leq_G B$, falls $\forall i \in \{1, \dots, k\}\colon a_i \leq_G b_i$. Dabei nennen wir eine zulässige Menge A *minimal*, falls $A \leq_G B$ für jede gleich-mächtige, zulässige Menge B.

Lemma 2.1 (Optimalität des gierigen Verfahrens)**.** *Existiert bei einer Menge P von Pfaden einer Kette eine zulässige Teilmenge mit $k \in \mathbb{N}$ Pfaden, so ist die Menge der in gieriger Ordnung $\leq_G$ kleinsten k Pfade, die das gierige Verfahren berechnet, eine minimale Menge.*

Beweis. Sei Q_0 eine zulässige Menge mit k Pfaden. Wir nennen G die Menge der in $\leq_G$ kleinsten k Pfade, die das gierige Verfahren berechnet, und transformieren Q_0 schrittweise zu $Q_k = G$. Dabei soll für $i \in \{1, \dots k\}$ gelten, dass Q_i zulässig ist sowie $Q_i \leq_G Q_{i-1}$. Dadurch ist auch G zulässig und durch die Transitivität von $\leq_G$ auf Mengen von Pfaden folgt $G \leq_G Q_0$, also insbesondere die Minimalität von G.

Als weitere Invariante nehmen wir auf, dass die ersten i Pfade (in gieriger Ordnung) von Q_i

mit denen von G übereinstimmen. Für $i = 0$ gelten alle Voraussetzungen. Nehmen wir also an, Q_{i-1} sei zulässig und stimme auf den ersten $i-1$ Pfaden mit G überein. Den i-ten Pfad von G nennen wir $p = (s_p, t_p)$.

Für den Fall $p \in Q_{i-1}$ ist p auch bereits der i-te Pfad von Q_{i-1}, da das Verfahren nach gieriger Ordnung vorgeht und Q_{i-1} mit G auf den ersten $i-1$ Pfaden übereinstimmt. Setzt man $Q_i \coloneqq Q_{i-1}$, bleiben alle Invarianten erhalten.

Sonst gilt $p \notin Q_{i-1}$ und die Menge der Pfade $q \in Q_{i-1}$ mit $q >_G p$ ist nicht leer. Von diesen Pfaden sei q ein solcher mit kleinstem Startknoten s_q, der in Q_{i-1} an j-ter Stelle steht (insb. $j \geq i$). Setzt man nun $Q_i \coloneqq (Q_{i-1} \cup \{p\}) \setminus \{q\}$, so stimmen Q_i und G an den ersten i Stellen wieder überein und alle Pfade, die in Q_{i-1} an den Stellen i bis $j-1$ stehen, rutschen in Q_i eine Stelle weiter an die Positionen $i+1$ bis j. Insbesondere gilt also $Q_i \leq_G Q_{i-1}$. Es bleibt zu zeigen, dass Q_i wieder zulässig ist, wozu die Kapazitäten der einzelnen Kanten betrachtet wird: Die Kanten, die links der beiden Anfangsknoten s_q und s_p stehen, sind nicht betroffen, da aufgrund der Minimalität von s_q dort nach q keine weiteren Kapazitäten benötigt werden. Ist s_q kleiner als s_p, so sparen die Kanten zwischen s_q und s_p sogar eine Kapazität ein. Ist andersrum s_p kleiner als s_q, verletzt Q_i auf den Zwischenkanten trotzdem keine Kapazitäten, da hier nach p aufgrund der Minimalität von s_q keine weiteren Pfade eine Kapazität verbrauchen und p vom Verfahren akzeptiert wurde, weil bis zum Pfad p keine Kapazität verletzt wurde. Die Kanten, bei denen sich p und q überschneiden, spielen offensichtlich keine Rolle und die Kanten vom Zielknoten von p bis zum Zielknoten von q benötigen wieder eine Kapazität weniger. □

Daraus folgt, dass das gierige Verfahren das Call-Control-Problem in Ketten optimal löst: Sei Q eine optimale Lösung, also eine möglichst große, zulässig Teilmenge von Pfaden in P, so liefert das gierige Verfahren nach Lemma 2.1 eine gleichmächtige, sogar minimale Lösung.

Eine einfache Implementierung dieses Verfahrens könnte in $\mathcal{O}(m \cdot n)$ Zeit erfolgen, wobei m die Anzahl der gegebenen Pfade und n die Anzahl der Kanten ist, indem einfach für jeden Pfad einzeln an jeder Kante überprüft wird, ob durch die Hinzunahme die Kapazität überschritten wird. Jedoch lässt sich dieser Zeitaufwand sogar auf $\mathcal{O}(n+m)$ verringern, wie wir in den nächsten Abschnitten erkennen werden. Dabei werden wir zunächst einen Algorithmus kennenlernen, bei dem vorausgesetzt wird, dass alle Kanten die gleiche Kapazität haben, und diesen danach an unser Ausgangsproblem anpassen.

2.1 Gieriger Algorithmus für gleiche Kapazitäten von Carlisle und Lloyd

Sei wieder (V, E) eine Kette und P eine Menge von m Pfaden in (V, E), wobei alle Kanten in E nun die feste Kapazität $C \in \mathbb{N}$ besitzen. Dabei können wir ohne Beschränkung der Allgemeinheit von $C \leq m$ ausgehen (sonst könnten wir einfach alle Pfade ohne Gefahr akzeptieren). Der Algorithmus, den Carlisle und Lloyd in [2] beschreiben, hat eine etwas abgewandelte Problemformulierung: Es wird statt von Pfaden in einer Kette von Intervallen auf der Zahlengeraden gesprochen. Das Verfahren soll nun möglichst viele dieser Intervalle in C verschiedene Farben färben, wobei sich zwei Intervalle derselben Farbe nie überschneiden dürfen. Wie man schnell erkennt, entspricht die größte Menge der gefärbten Intervalle im Ursprungsproblem gerade der größten Teilmenge der Pfade, in der sich maximal C Pfade an einer Kante überschneiden dürfen, bei der also jede Kante die Kapazität C hat. Statt also die Intervalle zu färben, reden wir im

Folgenden davon, die Pfade entsprechend zu färben.

Der Algorithmus führt zunächst C weitere sogenannte *virtuelle Pfade* ein, die in gieriger Ordnung vor den eigentlichen Pfaden P stehen und jeweils in einer der C Farben gefärbt sind. Dann werden alle Pfade in gieriger Ordnung verarbeitet. Dabei wird zu jeder Farbe c der aktuelle *Anführer* gespeichert, das heißt, der in gieriger Ordnung größte, bereits verarbeitete Pfad mit der Farbe c. Die virtuellen Pfade stellen dabei die initialen Anführer der Farben dar. Der Algorithmus sucht jetzt für jeden Pfad p den größten Anführer, der sich nicht mit p überschneidet. Ein solcher Pfad heißt *optimaler Anführer von p*. Findet er einen solchen, wird p akzeptiert, in die Farbe seines optimalen Anführers gefärbt und damit neuer Anführer der Farbe. Findet er keinen, gibt es keine freie Farbe, das heißt das Akzeptieren des Pfades würde eine Kapazität verletzen und der Pfad wird abgelehnt. Eine solche Färbung mit $C = 2$, kann man in Abbildung 1 sehen.

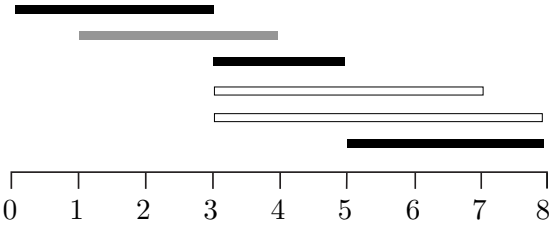


Abb. 1: Eine Färbung mit zwei Farben; weiße Pfade sind ungefärbt.

Speichert man allerdings zu jeder Farbe einfach den jeweiligen Anführer direkt ab, würde das Berechnen eines optimalen Anführers für jeden Pfad mindestens $\mathcal{O}(\log C)$ Zeit benötigen und damit würden wir das Ziel einer gesamt-linearen Laufzeit $\mathcal{O}(m)$ verpassen.

Stattdessen verwenden Carlisle und Lloyd in ihrem Algorithmus eine spezielle Union-Find-Struktur, die sogenannte Static-Tree-Set-Union-Struktur aus [3], bei der die möglichen Vereinigungen bereits zu Beginn feststehen und in einem Baum dargestellt werden können, wodurch m union- und find-Aufrufe bei n Elementen in einer Laufzeit von $\mathcal{O}(m + n)$ erfolgen können. Außerdem geht man beim Algorithmus davon aus, dass eine bereits aufsteigend sortierte Liste der Endknoten aller Pfade existiert. Diese Liste hat also $2 \cdot m$ Einträge, d.h. ein Punkt kann für verschiedene Pfade öfters vorkommen, dabei sind doppelte Einträge entsprechend der gierigen Ordnung der zugehörigen Pfade sortiert.

Genauer fügt der Algorithmus in gieriger Ordnung an erster Stelle (noch vor den virtuellen Pfaden) einen weiteren *fiktiven Anführer* f ein, der ermöglicht, nicht-akzeptierte Pfade in der Datenstruktur zu behandeln. Zu Beginn ist dann jeder Pfad in einer eigenen Gruppe der Union-Find-Instanz. Wir erhalten folgende Invarianten: Der Repräsentant jeder Gruppe ist immer der kleinste Pfad (in gieriger Ordnung) innerhalb der Gruppe. Ist ein Pfad noch nicht verarbeitet, so ist er in einer Einzelgruppe - die virtuellen Pfade und der fiktive Anführer zählen jedoch bereits als verarbeitet. Jede andere, sogenannte *aktive Gruppe* enthält nur (in gieriger Ordnung) aufeinanderfolgende Pfade. Der Repräsentant einer aktiven Gruppe ist dabei entweder ein Anführer einer Farbe oder der fiktive Anführer. Dementsprechend gibt es zu jeder Zeit genau $C + 1$ aktive Gruppen, wobei die Gruppe des fiktiven Anführers immer die Gruppe mit den in gieriger Ordnung kleinsten Pfaden bleibt.

Zunächst berechnet der Algorithmus zu jedem Pfad $p \in P$ den *bevorzugten Anführer von p*, also den größten Pfad in gieriger Ordnung, dessen Zielknoten nicht nach dem Anfangsknoten s_p von p steht. Das kann man mit einfachem Durchlaufen der Liste aller Endknoten in P erreichen,

also in $\mathcal{O}(n)$ Zeit. Es ist offensichtlich, dass kein optimaler Anführer von p in gieriger Ordnung größer als der bevorzugte Anführer von p sein kann.

In Abbildung 2 sieht man die entstehende Ausgangssituation für ein Beispiel mit 6 Pfaden.

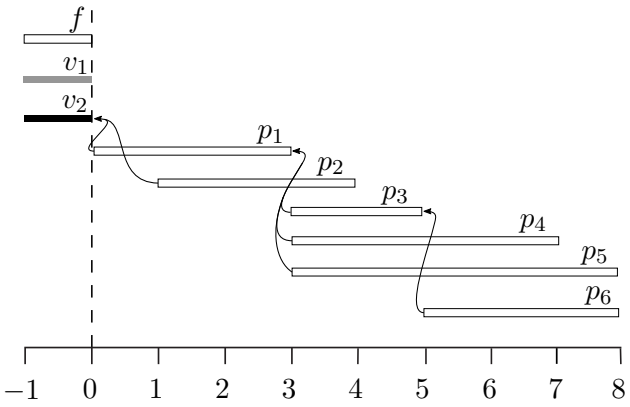


Abb. 2: Die Ausgangssituation: Mit den Pfeilen wird für jeden Pfad der bevorzugte Anführer markiert; v_1 und v_2 sind die virtuellen Pfade und f ist der fiktive Anführer.

Der Algorithmus bearbeitet danach jeden Pfad $p \in P$ nach gieriger Reihenfolge, wobei er jeweils wie folgt vorgeht. Es wird der bevorzugte Anführer q von p betrachtet: Ist *find*(q), also der Repräsentant der Gruppe von q, der fiktive Anführer f, so sind alle Pfade in P, die kleinergleich dem bevorzugten Anführer q sind und damit Kandidaten für einen optimalen Anführer von p wären, auch in der Gruppe von f. Daher kann es keinen optimalen Anführer zu p geben, da alle Anführer die Repräsentanten der anderen C aktiven Gruppen sind und daher in gieriger Ordnung nach dem bevorzugten Anführer q kommen. Daher wird p abgelehnt und die Gruppe von p mit der Gruppe des Vorgängers von p in gieriger Ordnung vereinigt. Die Invarianten bleiben erhalten, da wieder $C + 1$ Gruppen unter den verarbeiteten Pfaden existieren mit den gleichen Anführern.

Ist *find*(q) jedoch ein Anführer einer Farbe, so ist es mit Sicherheit der optimale Anführer von p, da es der größte Anführer ist, der noch kleinergleich dem bevorzugten Anführer von p ist (da die Pfade in den Gruppen aufeinanderfolgend sind). Also wird p akzeptiert, in dieser Farbe gefärbt und damit neuer Anführer der Farbe. Da *find*(q) nun kein Anführer mehr ist, wird seine Gruppe aufgelöst und mit der Gruppe des Vorgängers von *find*(q) vereinigt. Wieder erhalten wir alle Invarianten.

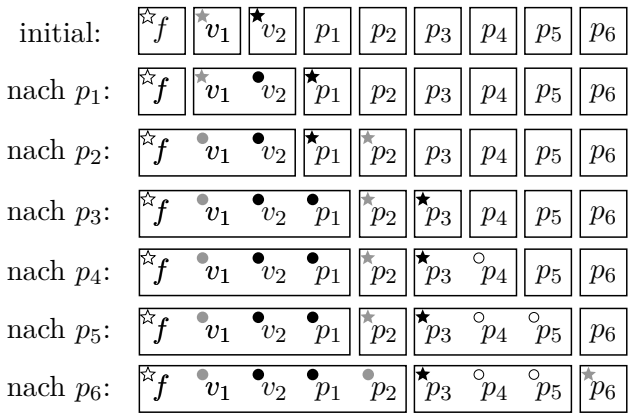


Abb. 3: Der Verlauf der Union-Find-Instanz für das Beispiel aus Abb. 2

In Abbildung 3 sieht man den Verlauf der Union-Find-Instanz für das Beispiel aus Abbildung 2. Die i-te Zeile zeigt dabei den Zustand der Union-Find-Instanz nach Bearbeiten des i-ten Pfades. Jedes Rechteck stellt darin eine Gruppe der Instanz dar. Der Repräsentant einer Gruppe steht ganz links, wobei Anführer durch Sterne und Färbungen durch Punkte bzw. Sterne in schwarz, grau oder weiß (für nicht-akzeptiert) dargestellt werden. Insbesondere wird erkennbar, dass wir nur Vereinigungen entlang einer Kette machen, was uns ermöglicht, die effizientere Static-Tree-Set-Union-Struktur aus [3] zu verwenden. Eine genauere Analyse dieses Verfahrens und dessen Laufzeit $\mathcal{O}(m)$ beschreiben Carlisle und Llyod in [2].

2.2 Gieriger Algorithmus mit willkürlichen Kapazitäten

Nachdem wir einen effizienten Algorithmus für identische Kapazitäten gefunden haben, wollen wir nun das eigentliche Problem mit willkürlichen Kapazitäten $c : E \mapsto \mathbb{N}$ durch geschicktes Anpassen des Algorithmus aus Abschnitt 2.1 lösen. Sei $C := \max_{e \in E} c(e)$ die maximale Kapazität aller Kanten. Die Idee ist es, den Algorithmus für Kanten mit identischer Kapazität C anzuwenden und dabei für jede Kante e die $C - c(e)$ hinzugewonnen Kapazitäten mit Platzhalterpfaden zu befüllen. Werden alle Platzhalterpfade vom Algorithmus akzeptiert, so können wir sicher sein, dass wir nach Entfernen der Platzhalterpfade eine optimale Lösung für willkürliche Kapazitäten gefunden haben. Wir müssen also insbesondere dafür sorgen, dass all unsere Platzhalter eingefärbt werden. Eine wie im vorigen Abschnitt aufsteigend sortierte Liste der Endknoten aller Pfade berechnen wir nun allerdings selbst mit einem Aufwand von $\mathcal{O}(n + m)$, z.B. durch Sortieren durch Zählen. Dabei speichern wir zu jedem Endknotens in der Liste eine Referenz auf seinen zugehörigen Pfad ab. Insbesondere können wir dann für jeden Eintrag in der Liste entscheiden, ob der Knoten ein Anfangs- oder ein Zielknoten seines zugehörigen Pfades ist.

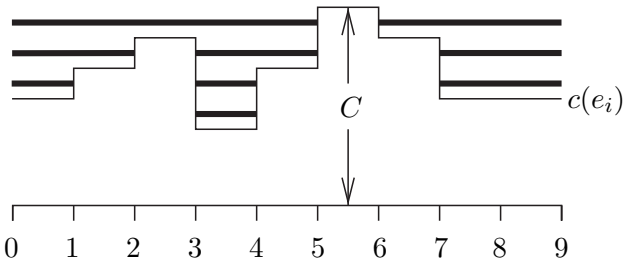


Abb. 4: Platzhalterpfade in schwarz dargestellt (mod. nach [1])

Wir beschäftigen uns zunächst näher mit dem Hinzufügen der Platzhalter: Dazu wird die Liste aller Knoten der Kette durchlaufen und gleichzeitig berechnet, wie viele Platzhalterpfade an jedem Knoten beginnen bzw. enden müssen (siehe Abbildung 4): Beim ersten Knoten v_0 müssen $C - c(e_1)$ Platzhalterpfade beginnen und wir schreiben $(C - c(e_1))$ mal v_0 in einen leeren Keller K. Ist $c(e_i) > c(e_{i+1})$, beginnen beim Knoten v_i weitere Platzhalter und v_i wird genau $(c(e_i) - c(e_{i+1}))$ mal in K gelegt. Ist $c(e_i) < c(e_{i+1})$, so müssen Platzhalterpfade bei v_i enden und es werden $c(e_{i+1}) - c(e_i)$ Elemente aus K geholt, die jeweils als Anfangsknoten mit v_i als Zielknoten einen der Platzhalterpfade bilden. Beim letzten Knoten v_{n-1} bilden dann alle verbleibenden Knoten aus K als Anfangsknoten mit v_{n-1} als Zielknoten die restlichen Platzhalter.

Da wir allerdings insgesamt eine Laufzeit von $\mathcal{O}(n + m)$ zum Ziel haben, können wir das beschriebene Verfahren nicht einfach übernehmen: Bei starken Schwankungen der einzelnen Ka-

pazitäten kann es hier passieren, dass bis zu $\Omega(m \cdot n)$ Platzhalter hinzugefügt werden müssen, was unsere Ziellaufzeit übersteigt (siehe Abbildung 5). Jedoch wird schnell ersichtlich, dass in solchen Fällen viele Platzhalter unnötigerweise platziert werden: Hat beispielsweise die Nachfolgekante eines Knoten v eine sehr hohe Kapazität, wobei die Vorgängerkante nur eine sehr niedrige Kapazität besitzt, so müssten sehr viele Pfade den Anfangsknoten v haben, um die Kapazität der Nachfolgekante auszunutzen. Da wir die Pfade bereits kennen, können wir also in vielen Fällen die Kapazität von solchen Nachfolgekanten verringern und damit Schwankungen in den Kapazitäten etwas ausgleichen.

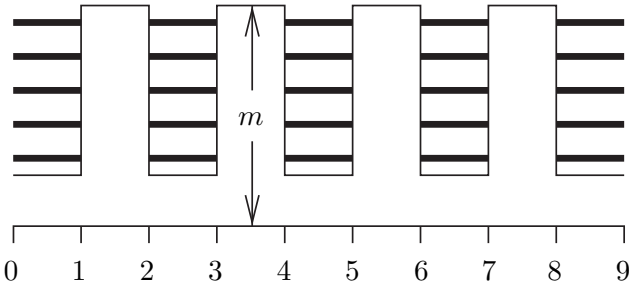


Abb. 5: Bei jeder zweiten Kante werden $C - 1 = \Omega(m)$ Platzhalter eingefügt (mod. nach [1])

Um diese Erkenntnis umzusetzen, definieren wir eine neue Kapazitätsfunktion c' mit

$$c'(e_i) = \begin{cases} \min(c(e_0), n_0) & \text{für } i = 0 \\ \min(c(e_i), c'(e_{i-1}) + n_i) & \text{für } i \geq 1 \end{cases}$$

wobei n_i die Anzahl der Pfade in P mit Anfangsknoten v_i ist. Diese können wir in $\mathcal{O}(n + m)$ Zeit berechnen: Dazu gehen wir einmal durch die sortierte Liste der $2m$ Endknoten aller Pfade und zählen dabei für jeden Knoten v_i der Kette alle Anfangsknoten v in der Liste (überspringe Zielknoten) mit $v = v_i$. Dies liefert n_i, womit wir auch $c'(e_i)$ berechnen können. Durch das Verwenden von c' statt c wird das Problem auch nicht verändert, da nur Kapazitäten entfernt werden, die die Pfade P ohnehin nicht nutzen könnten. Insbesondere ist also jede zulässige Menge auf den Kapazitäten c auch eine zulässige Menge auf den Kapazitäten c'. Das nächste Lemma soll nun aufklären, ob unser Vorgehen die Anzahl an Platzhaltern ausreichend verringern konnte:

Lemma 2.2. *Mit den neuen Kapazitäten c' werden nur noch $\mathcal{O}(m)$ Platzhalterpfade erzeugt.*

Beweis. Wir betrachten den Gesamtzuwachs der Kapazitäten $I = \sum_{i=0}^{n-2} I_i$ als Summe aller Zuwächse I_i am Knoten v_i mit $I_0 = c'(e_0)$ und $I_i = \max\{c'(e_i) - c'(e_{i-1}), 0\}$ für $i \geq 1$. Ein Anstieg von c' in e_i um $I_i \in \mathbb{N}_0$ bedeutet, dass $n_i \geq I_i$, da per Definition $c'(e_i) \leq c'(e_{i-1}) + n_i$. Insbesondere ist also I beschränkt durch die Anzahl an Pfaden m. Ein Platzhalterpfad wird außerdem nur dann erzeugt, wenn der Algorithmus auf einen Zuwachs in den Kapazitäten (höchstens I) oder auf das Ende der Kette (höchstens C) stößt. Das heißt es werden höchstens $I + C = \mathcal{O}(m)$ Platzhalterpfade erzeugt. □

Insbesondere gelingt also die Anpassung der Kapazitäten sowie das Auffüllen mit Platzhalterpfaden in $\mathcal{O}(n + m)$ Zeit. Nun müssen wir uns nur noch der Herausforderung stellen, beim Lösen des konstruierten Problems mit identischen Kapazitäten alle Platzhalter zu akzeptieren. Die Idee dabei ist es, die Platzhalterpfade so früh wie möglich zu akzeptieren, und erst danach die Originalpfade zu betrachten.

Dazu wird eine Liste L der Endknoten aller Pfade - jetzt mit Platzhalterpfaden - angelegt, wobei wir wieder zu jedem Knoten eine Referenz auf den zugehörigen Pfad mitspeichern und damit entscheiden können, ob ein Anfangs- oder Zielknoten vorliegt. Die Einträge der Liste sollen nach Knoten aufsteigend sortiert sein und bei identischen Knoten sollen Zielknoten vor Anfangsknoten geordnet sein. Jeder Knoten kommt also genau so oft vor, wie es Pfade mit ihm als einer der beiden Endknoten gibt. Die gierige Ordnung $\leq_G$ erhalten wir, indem wir in der Liste jeden Zielknoten eines Pfades durch den Pfad selbst ersetzen und die restlichen Knoten verwerfen.

Wir benutzen mit dem fiktiven Anführer und den C virtuellen Pfaden wieder dieselbe Hilfspfade wie in Abschnitt 2.1, wobei diese in $\leq_G$ wieder vor den restlichen Pfaden stehen. Für die Berechnung der bevorzugten Anführer jedes Pfades p (also dem in $\leq_G$ größten Pfad $q = (s_q, t_q)$ mit $t_q \leq s_p$) benötigen wir mit $\leq_G$ ebenfalls einen Zeitaufwand von $\mathcal{O}(m)$. Auch die Union-Find-Struktur bleibt die gleiche und jeder Pfad startet in seiner eigenen Gruppe. Nun iteriert der Algorithmus durch die Liste L der Endknoten und verarbeitet dabei Platzhalterpfade beim Antreffen vom zugehörigen Anfangsknoten und Originalpfade beim Antreffen vom zugehörigen Zielknoten. Die eigentliche Verarbeitung eines Pfades p hat sich aber nicht geändert: Es wird der bevorzugte Anführer q von p betrachtet; falls *find*(q) der fiktive Anführer ist, wird p abgelehnt und die Gruppe von p mit der Gruppe des Vorgängers von p vereinigt; andernfalls wird p akzeptiert und in der Farbe von *find*(q) gefärbt und die Gruppe von *find*(q) mit der Gruppe des Vorgängers von *find*(q) vereinigt.

Das nächste Lemma soll nun zeigen, dass das Auffüllen mit Platzhaltern zusammen mit diesem Verfahren eine korrekte Implementierung des gierigen Verfahrens darstellt:

Lemma 2.3. *Der beschriebene Algorithmus ist eine korrekte Implementierung des gierigen Verfahrens.*

Beweis. Wir bezeichnen den oben beschriebenen Algorithmus nun mit U. Für U haben wir wieder folgende Invariante für die Union-Find-Gruppen: Jede Gruppe mit verarbeiteten Pfaden enthält nur in $\leq_G$ aufeinanderfolgende Pfade und der Repräsentant einer solchen Gruppe ist entweder ein Anführer einer Farbe oder der fiktive Anführer. Andere Gruppen sind Einzelgruppen.

Es ist leicht zu sehen, dass U wie im vorherigen Abschnitt alle Pfade in C Farben färbt, wobei sich zwei Pfade derselben Farbe nie schneiden können. Um zu zeigen, dass U alle Platzhalterpfade färbt, nehmen wir an, U würde einen Platzhalterpfad p nicht färben. Sei p derjenige Platzhalter mit kleinstem Anfangsknoten, der von U nicht gefärbt wird, und sei q sein bevorzugter Anführer. Das bedeutet, dass unmittelbar vor der Bearbeitung von p der Repräsentant *find*(q) der fiktive Anführer war. Insbesondere gab es unter den C gefärbten Anführern keinen optimalen Anführer von p, sondern alle gefärbten Anführer haben zu der Zeit einen Zielknoten, der rechts vom Anfangsknoten von p liegt (sonst gäbe es einen optimalen Anführer). Da p bereits am Anfangsknoten bearbeitet wird und alle Originalpfade erst am Zielknoten, kann kein Originalpfad einer dieser gefärbten Anführer sein, sondern es sind allesamt Platzhalterpfade, die bereits vor p gefärbt worden sein müssen. Dementsprechend liegen auf der ersten Kante von p mindestens $C+1$ Platzhalterpfade, was einen Widerspruch zur Konstruktion der Platzhalterpfade darstellt.

Insbesondere berechnet U wieder eine zulässige Teilmenge der Pfade. Jetzt müssen wir noch

zeigen, dass genau die Originalpfade akzeptiert werden, die auch das gierige Verfahren akzeptiert.

Dazu gehen wir vom Gegenteil aus und betrachten den ersten Originalpfad p in gieriger Ordnung, bei dem die Entscheidung unterschiedlich ausgefallen ist. Da das gierige Verfahren immer eine minimale Lösung liefert, muss p vom gierigen Verfahren akzeptiert und von U abgelehnt worden sein, da die Menge, die U berechnet, ebenfalls zulässig ist. Sei A die Menge der Pfade, die unmittelbar vor der Bearbeitung von p (von beiden Verfahren) akzeptiert wurden. Da der gierige Algorithmus zusätzlich p akzeptiert hat, wissen wir, dass $A \cup \{p\}$ zulässig ist, was wir im Folgenden zum Widerspruch führen: Da U den Pfad p am Zielknoten verarbeitete (und p damit der aktuell größte Pfad in gieriger Ordnung war), mussten sich zu dieser Zeit alle Anführer von Farben mit p überschnitten haben. Dabei sei l der in gieriger Ordnung kleinste Anführer einer Farbe und e die letzte Kante seines Pfades. Von dieser wissen wir sicher, dass sie sich mit p überschneidet.

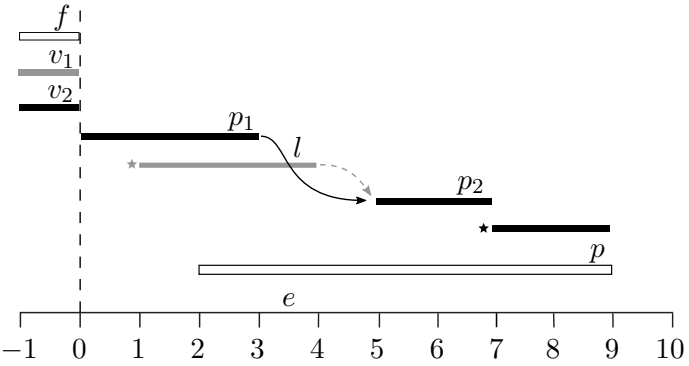


Abb. 6: c-gefärbte Pfade sind schwarz. Mit einem Stern gekennzeichnete Pfade sind Anführer ihrer Farbe. Der durchgezogene Pfeil zeigt den tatsächlichen Verlauf der Farbe an. p wurde vom gierigen Verfahren akzeptiert und von U abgelehnt.

Angenommen, es existiere eine Farbe c, sodass kein c-gefärbter Pfad e enthält. Da l der kleinste Anführer ist, existiert ein c-gefärbter Pfad p_1 links von e (z.B. der c-gefärbte virtuelle Pfad) und ein c-gefärbter Pfad p_2 rechts von e (z.B. der c-gefärbte Anführer). Wählt man, wie in Abbildung 6 beispielhaft skizziert, diese jeweils möglichst nah an e, so erkennt man einen Widerspruch: Als p_2 mit p_1 als den scheinbar optimalen Anführer in c gefärbt wurde, war tatsächlich l ein besserer Anführer von p_2, da sein Zielknoten weiter rechts ist. Insbesondere hätte U p_2 also nicht in c gefärbt.

Daher besitzen alle C Farben einen Pfad, der e enthält, insgesamt gibt es also $C + 1$ Pfade in $A \cup \{p\}$, die e enthalten. Das ist allerdings ein Widerspruch dazu, dass $A \cup \{p\}$ eine zulässige Menge ist. □

Daraus folgt nun:

Theorem 2.1. *Das gierige Verfahren berechnet eine Lösung für Call-Control in Kettennetzwerken mit beliebigen Kapazitäten und kann in einer Laufzeit von $\mathcal{O}(n+m)$ implementiert werden, wobei n die Anzahl der Knoten der Kette und m die Anzahl der gegebenen Pfade ist.*

3 Optimale Lösung des Call-Control-Problems in Ringen

Nachdem wir das Call-Control-Problem in Ketten optimal lösen können, betrachten wir nun das Problem in Ringen. Sei also (V, E) ein Ring mit $V = \{0, \ldots, n-1\}$ und $E = \{e_0, \ldots, e_{n-1}\}$, wobei e_i die Knoten i und $(i+1 \bmod n)$ verbindet. Außerdem haben wir eine Menge $P = \{p_1, \ldots, p_m\}$ an Pfaden gegeben, unter denen jeder Pfad p einen Anfangsknoten s_p mit einem Zielknoten t_p verbindet mit $s_p \neq t_p$. Dabei enthält der Pfad alle Kanten, auf die man trifft, wenn man sich vom Anfangsknoten im Uhrzeigersinn - das heißt bis auf mod n mit aufsteigenden Knotennamen - zum Zielknoten bewegt. Alle Knoten eines Pfades bis auf den Anfangs- und Zielknoten heißen *innere Knoten*.

Wir können außerdem ohne Einschränkung von $n \leq 2m$ ausgehen, da wir solche Knoten entfernen können, die nicht Anfangs- oder Zielknoten eines Pfades sind. Dabei ersetzen wir die zwei Kanten, die am zu entfernenden Knoten anliegen, durch eine einzige Kante, dessen Kapazität das Minimum der vorherigen beiden ist.

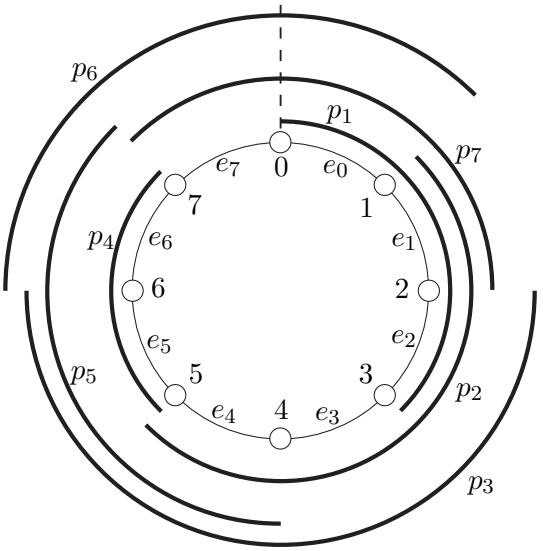


Abb. 7: Ein Ringnetzwerk mit 8 Kanten (innen) und 7 Pfaden (außen) (mod. nach [1])

Das Call-Control-Problem beschreibt hier wieder das Problem, eine möglichst große zulässige Teilmenge $Q \subseteq P$ zu finden. Dazu teilen wir die Menge der Pfade P in zwei disjunkte Teilmengen P_1 und P_2. P_1 bildet die Menge der Pfade in P, die den Knoten 0 nicht als inneren Knoten haben; P_2 beinhaltet die restlichen Pfade, also $P \backslash P_1$. Diese Partition können wir ganz einfach berechnen: Wir sagen nun, dass jeder Pfad p, dessen Zielknoten 0 ist, eigentlich den Zielknoten n hat. Ist dann $s_p < t_p$, so ist 0 kein innerer Knoten von p und p wird P_1 zugeordnet, sonst P_2.

Abbildung 7 zeigt ein Beispiel eines Rings mit 8 Knoten sowie die Menge der sieben gegebenen Pfade (außen). Dabei stellen die Pfade $p_1, \ldots, p_5$ die Menge P_1 dar, p_6 und p_7 die Menge P_2.

Nun transformieren wir den Ring in eine Kette mit $2n$ Knoten: Wir fertigen zwei Kopien der Kanten $e_1, \ldots, e_n$ aus unserem Ring an und bilden eine Kette, indem wir die beiden Kopien aneinanderhängen. Die dazu gehörigen Knotennamen sind dabei $0, \ldots, 2n-1$, wobei wir die Knoten $n, \ldots, 2n-1$ durch $0', \ldots, (n-1)'$ als die zweite Kopie der Knoten kennzeichnen. Die Pfade werden diesem Schema angepasst, sodass Pfade in P_1 nur Kanten der ersten Kopie belegen, wohingegen jeder Pfad $p \in P_2$ aus zwei Teilpfaden besteht: Dem *Kopf*, der von s_p bis nach $0'$ auf der ersten Kopie der Kanten liegt, und dem *Schwanz*, der von $0'$ bis t'_p auf der zweiten Kopie

liegt. In Abbildung 8 wurde der Ring aus aus Abbildung 7 in die Kettenstruktur umgewandelt.

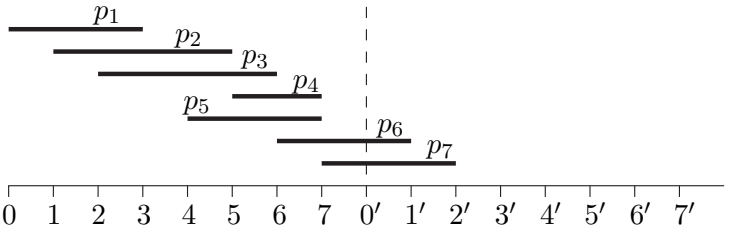


Abb. 8: Die aus dem Ring in Abb. 7 konstruierte Kette (mod. nach [1])

Wir definieren im Weiteren für eine Menge Q von Pfaden die *Belastung* $L_1(Q, e_i)$ der Kante e_i in der ersten Kopie als die Anzahl der Pfade von Q, die die erste Kopie der Kante e_i beinhalten. Analog dazu sei $L_2(Q, e_i)$ die Belastung von e_i in der zweiten Kopie.

Definition 3.1 (Profil)**.** Sei Q eine Menge von Pfaden. Die monoton fallende Folge

$$\pi_Q \coloneqq (L_2(Q, e_0), \ldots, L_2(Q, e_{n-1}))$$

der Belastungen der n Kanten in den zweiten Kopien heißt *Profil von* Q. Außerdem bezeichne $\pi_Q(e_i) \coloneqq L_2(Q, e_i)$ und für zwei Profile π und π' schreiben wir $\pi \leq \pi'$, falls $\pi(e_i) \leq \pi'(e_i)$ für alle $i \in \{0, \ldots, n-1\}$.

Wir bezeichnen eine Menge Q von Pfaden im Ring als *kettenzulässig*, falls die Pfade von Q in der oben konstruierten Kette keine Kapazitäten überschreiten, d.h. falls $L_1(Q, e) \leq c(e)$ und $L_2(Q, e) \leq c(e)$ für alle Kanten e. Gilt darüber hinaus $L_1(Q, e) + \pi(e) \leq c(e)$ für alle Kanten e, so heißt Q *kettenzulässig zum Startprofil* π. Anschaulich betrachtet, werden hier von den Kanten der ersten Kopie einige Kapazitäten bereits vom Startprofil belegt. Betrachtet man jetzt nochmals die Konstruktion der Kette, kann man leicht erkennen, dass eine Menge Q von Pfaden zulässig (im Ring) ist genau dann, wenn Q kettenzulässig zum eigens generierten Startprofil π_Q ist, welches ja gerade den Schwänzen der überstehenden Pfade entspricht.

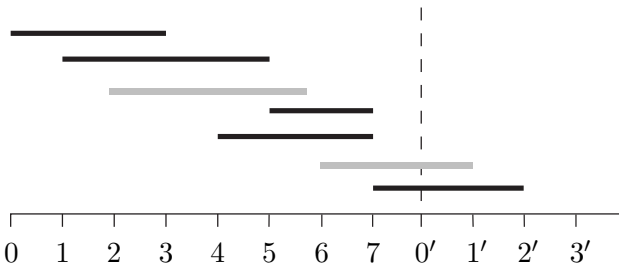
3.1 Der Algorithmus

Nachdem wir nun alle notwendigen Vorbereitung getroffen haben, können wir mit dem eigentlichen Algorithmus beginnen. Dabei gehen wir wie folgt vor: Wir suchen mit binärer Suche nach dem größten $k \in \{0, \ldots, m\}$, sodass eine zulässige Teilmenge $Q_k \subseteq P$ mit k Pfaden existiert, und geben schließlich Q_k aus.

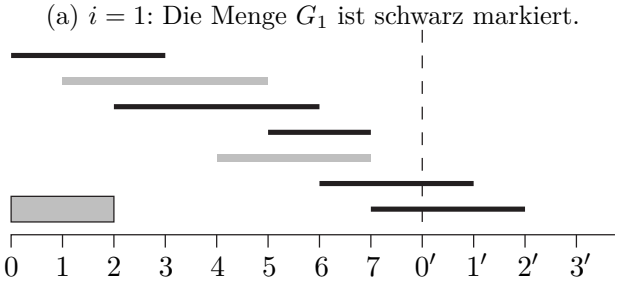
Die Prozedur für das Finden einer zulässigen k-elementigen Menge von Pfaden bauen wir dabei wie folgt auf. Wir starten mit dem leeren Profil π_0, das überall 0 ist. In jeder Runde $i \in \mathbb{N}$ initialisieren wir die Kapazitäten der beiden Kopien jeder Kante e mit $c(e)$, wobei wir jeweils in der ersten Kopie von $c(e)$ noch $\pi_{i-1}(e)$ Kapazitäten abziehen. Das Profil π_{i-1} belegt also bereits einige Kanten in der ersten Kopie und eine Menge Q ist mit diesen Kapazitäten kettenzulässig genau dann, wenn sie auf den ursprünglichen Kapazitäten kettenzulässig zum Startprofil π_{i-1} ist. Nun bestimmen wir mit dem gierigen Verfahren eine größtmögliche Menge G, die in der resultierenden Kette zulässig ist. Enthält G weniger als k Pfade, so bricht die Prozedur ab mit der Antwort, dass eine zulässige k-elementige Menge nicht existiert. Ansonsten sei G_i die Menge der ersten k Pfade von G in gieriger Ordnung und π_i das dadurch erzeugte

Profil. Ist nun $\pi_i = \pi_{i-1}$, so meldet die Prozedur einen Erfolg mit G_i als Rückgabe, sonst wird die nächste Runde berechnet.

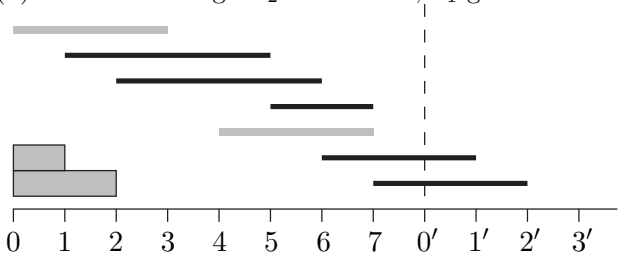
Ein Beispiel kann man in Abbildung 9 sehen. Hier wird überprüft, ob eine 5-elementige zulässige Teilmenge an Pfaden existiert, wobei das Beispiel aus Abbildung 8 weitergeführt wird. Nach drei Schritten hält der Algorithmus mit einer zulässigen Menge G_3, da, wie man sieht, das Profil π_2 dem Profil π_{G_3} von G_3 selbst entspricht.



(a) $i = 1$: Die Menge G_1 ist schwarz markiert.



(b) $i = 2$: Die Menge G_2 ist schwarz, π_1 grau markiert.



(c) $i = 3$: Die Menge G_3 ist schwarz, π_2 grau markiert. G_3 ist zulässig.

Abb. 9: Die Prozedur mit 3 Runden anhand des Beispiels aus Abb. 7 für $k = 5$, wobei alle Kanten die Kapazität 2 haben (mod. nach [1])

3.2 Korrektheit des Algorithmus

Um die Korrektheit des gesamten Algorithmus einzusehen, beweisen wir in den nächsten Lemmata, dass die Prozedur zum Finden einer zulässigen k-elementigen Menge korrekt ist. Dabei benutzen wir die vom Algorithmus generierten Mengen G_i, sowie dessen Profile π_i.

Lemma 3.1. *Die Folge der Profile ist monoton wachsend, d.h. $\pi_i \leq \pi_{i+1}$ für alle $i \in \mathbb{N}_0$.*

Beweis. Durch Induktion: Für $i = 0$ gilt die Aussage, da π_0 überall 0 ist. Angenommen, es gilt $\pi_{i-1} \leq \pi_i$. Nach Lemma 2.1 ist G_i eine minimale Menge mit k Pfaden, die auf der Kette mit den durch π_i reduzierten Kapazitäten zulässig ist. Da G_{i-1} zulässig zum Startprofil π_{i-1} ist und $\pi_{i-1} \leq \pi_i$, ist G_{i-1} auch zulässig zum Startprofil π_i und aufgrund der Minimalität von G_i ist insbesondere $G_i \leq_G G_{i-1}$. Daraus folgt direkt $\pi_i \leq \pi_{i+1}$. □

Als Kondition dafür, dass unser Algorithmus hält, benötigen wir noch, dass die generierten Profile nicht ewig wachsen können und, falls eine Lösung existiert, ab irgendeiner Runde konstant bleiben.

Lemma 3.2. *Falls eine im Ring zulässige Lösung Q^* mit k Pfaden existiert, dann ist π_{Q^*} eine obere Grenze der generierten Profile π_i.*

Beweis. Sei also Q^* eine zulässige Lösung mit k Pfaden. Wir zeigen wieder per Induktion für $i \in \mathbb{N}_0$, dass $\pi_i \leq \pi_{Q^*}$. Für $i = 0$ ist das offensichtlich, da π_0 überall 0 ist. Angenommen, $\pi_i \leq \pi_{Q^*}$. Wie wir oben gesehen haben, ist Q^* kettenzulässig zum Startprofil π_{Q^*}, da es zulässig im Ring ist. Da G_i mittels des gierigen Verfahren zum Startprofil π_i ermittelt wird und Q^* ebenfalls kettenzulässig zum Startprofil π_i ist (da $\pi_i \leq \pi_{Q^*}$), gilt nach Lemma 2.1, dass $G_i \leq_G Q^*$. □

Nun können wir die Korrektheit der Prozedur vervollständigen:

Lemma 3.3. *Die Prozedur liefert die korrekte Rückgabe in höchstens $n \cdot c(e_0)$ Runden.*

Beweis. Für die Korrektheit unterscheiden wir, ob eine k-elementige zulässige Menge existiert. Nehmen wir an, es existiert keine solche Menge und der Algorithmus bricht nicht ab, sondern liefert eine Menge Q_i als Rückgabe. Dann müssen die letzten beiden Profile π_{i-1} und $\pi_i = \pi_{Q_i}$ identisch gewesen sein. Da Q_i aber durch das gierige Verfahren so berechnet wurde, dass es kettenzulässig zum Startprofil π_{i-1} ist, ist es kettenzulässig zum eigenen Startprofil, insbesondere eine zulässige Menge im Ring, was im Widerspruch zur Annahme steht.

Angenommen es existiere eine zulässige Menge Q^* mit k Elementen. Dann ist die Folge der generierten Profile $(\pi_i)_i$ nach Lemma 3.2 durch π_{Q^*} nach oben beschränkt. Daher existiert zu jedem dieser Profile eine kettenzulässig k-elementige Menge (nämlich Q^*), weswegen das gierige Verfahren nach Lemma 2.1 eine mindestens k-elementige Menge zurückgibt. Daher kann der Algorithmus nicht ohne Rückgabe abbrechen. Da die Folge $(\pi_i)_i$ nach Lemma 3.1 außerdem monoton wachsend ist, konvergiert sie nach maximal $\sum_{j=0}^{n-1} \pi_{Q^*}(e_j)$ Runden. Also existiert ein minimales $i \in \mathbb{N}$ mit $\pi_i = \pi_{i-1}$ und der Algorithmus gibt Q_i zurück. Da $\pi_{i-1} = \pi_{Q_i}$, ist Q_i kettenzulässig zum eigenen Startprofil und damit eine zulässige Menge. Weil ein Profil eine monoton fallende Folge ist, bekommen wir zusätzlich die Abschätzung von maximal $\sum_{j=0}^{n-1} \pi_{Q^*}(e_j) \leq n \cdot \pi_{Q^*}(e_0) \leq n \cdot c(e_0)$ Runden. □

Unsere Ergebnisse können wir nun im folgenden Theorem zusammenfassen:

Theorem 3.1. *Das Call-Control-Problem in Ringen kann in $\mathcal{O}(m \cdot n \cdot c_{\min} \cdot \log m)$ Zeit gelöst werden, wobei n die Anzahl der Knoten, m die Anzahl der Pfade und $c_{\min}$ die kleinste Kantenkapazität ist.*

Beweis. Jede Runde des Entscheidungsalgorithmus ruft einmal das gierige Verfahren auf. Dieser hat nach Theorem 2.1 eine Laufzeit von $\mathcal{O}(n+m)$ und, da o.E. $n \leq 2m$, gilt $\mathcal{O}(n+m) = \mathcal{O}(m)$. Nach Lemma 3.3 können wir in $\mathcal{O}(n \cdot c(e_0))$ Runden eine zulässige k-elementige Menge finden. Dies kann auf $\mathcal{O}(n \cdot c_{\min})$ reduziert werden, indem die Kanten und Knoten des Rings so benannt werden, dass die Kante e_0 die geringste Kapazität besitzt (dies gelingt in $\mathcal{O}(n+m)$ Zeit). Mit binärer Suche benötigen wir schließlich noch $\mathcal{O}(\log m)$ Entscheidungsrunden, um das größte $k \in \{0, \ldots, m\}$ mit einer zulässigen k-elementigen Menge zu finden. □

4 Weighted-Call-Control in Ketten und Ringen

Nachdem wir nun optimale Lösungen für das Call-Control-Problem in Ketten und Ringen gefunden haben, modifizieren wir in diesem Kapitel die Problemstellung ein wenig und kümmern uns nun um das Weighted-Call-Control-Problem. So geben wir jedem der Pfade p ein bestimmtes Gewicht $w(p) \in \mathbb{R}_+$ und versuchen statt der Anzahl der akzeptierten Pfade Q das Gesamtgewicht $w(Q) \coloneqq \sum_{p \in Q} w(p)$ zu maximieren, ohne dabei die Kapazität einer Kante zu verletzen. Dabei verbraucht weiterhin jeder Pfad eine Kapazitätseinheit einer Kante, falls die Kante Teil des Pfades ist.

Wir werden dazu für das Problem in Ketten einen optimalen Algorithmus finden, jedoch für das Problem in Ringen nur einen Approximationsalgorithmus von Güte 2 kennenlernen.

4.1 Weighted-Call-Control-Problem in Ketten

Für den Fall, dass alle Kanten e wieder identische Kapazitäten $c(e) = C \in \mathbb{N}$ haben, stellen Carlisle und Lloyd in [2] eine optimale Lösung für das äquivalente Problem des C-Färbens von gewichteten Intervallen vor. Dabei wird das Problem auf ein Minimum-Cost Flow Problem reduziert, und dadurch eine Laufzeit von $\mathcal{O}(C \cdot S(m))$ erreicht, wobei m die Anzahl der Intervalle (hier: Pfade) und $S(m)$ die Laufzeit eines Kürzesten-Pfade-Algorithmus in gerichteten Graphen mit positiven Kantengewichten und m Kanten ist. Außerdem wird wieder angenommen, dass bereits eine sortierte Liste aller Endknoten zur Verfügung steht.

Eine Instanz der Größe C des Minimum-Cost Flow Problem besteht dabei aus einem gerichteten Graphen (V, E) mit zwei ausgezeichneten Knoten s und t, wobei jede Kante e des Graphen die Kosten $\mathit{cost}(e)$ und die Kapazität $\mathit{cap}(e) \geq 0$ besitzt. Eine Funktion $f \colon E \to \mathbb{R}$ heißt *Fluss*, falls $0 \leq f(e) \leq \mathit{cap}(e)$ für alle Kanten e und in jeden Knoten v (ausgenommen s und t) genauso viel hineinfließt wie auch wieder herausfließt, d.h. die Summe des Flusses aller eingehenden Kanten von v mit der der ausgehenden Kanten übereinstimmt. Bei s beträgt der ausgehende und bei t der eingehende Fluss dabei jeweils C. Gesucht ist nun ein Fluss, dessen Kosten $\sum_{e \in E} f(e) \cdot \mathit{cost}(e)$ minimal sind.

Übertragen auf das Weighted-Call-Control-Problem, skizzieren wir für das Kettennetzwerk (V, E, c) und der Menge P der gegebenen Pfade in (V, E) im Folgenden die Reduktion unseres Problems. Die Knoten der Kette (V, E) seien wieder wie gewohnt von links nach rechts durchnummeriert. Sei $v_0 < \cdots < v_r$ die sortierte Liste der Endknoten aller Pfade ohne Duplikate. Wir bauen den gerichteten Graphen als gerichteten Pfad mit den Knoten $\{s = v_0, \ldots, v_r = t\}$ und erstellen für $1 \leq i \leq r$ eine Kante (v_{i-1}, v_i) mit Kapazität C und Kosten 0. Zusätzlich erstellen wir für jeden Pfad $p \in P$ eine sogenannte *Pfadkante* (s_p, t_p) mit s_p Anfangs- und t_p Zielknoten von p, wobei die Kosten der Kante dem negativen Gewicht des Pfades und die Kapazität der Kante 1 entsprechen. In einem Fluss f repräsentieren dann die Pfadkanten e mit $f(e) = 1$ die akzeptierten Pfade.

Die Intuition hinter der Korrektheit dieser Problemformulierung besteht darin, den Fluss nicht als Ganzes, sondern den Fluss der einzelnen C Einheiten zu betrachten. Dadurch erkennt man schnell, dass maximal C Flusseinheiten sich überschneidende Pfade belegen können. Für eine genauere Beschreibung und Analyse des Verfahren sowie dessen Laufzeit betrachte man [2]. Wir wollen uns dagegen dem Problem mit willkürlichen Kapazitäten widmen.

Theorem 4.1. *Das gewichtete Call-Control-Problem einer Kette (V, E) mit willkürlichen Kapazitäten kann in $\mathcal{O}(n + m \cdot S(m))$ Zeitaufwand optimal gelöst werden. Dabei ist $n = |V|$, $m = |E|$ und $S(m)$ die Laufzeit eines Kürzesten-Pfade-Algorithmus in gerichteten Graphen mit positiven Kantengewichten und m Kanten.*

Beweis. Wir führen auch dieses Problem wieder auf ein Problem mit identischen Kapazitäten zurück. Dabei ist $C = \max_{e \in E} c(e)$ die maximale Kantenkapazität. Wie in Abschnitt 2.2 müssen die zusätzlichen $C - c(e)$ Kapazitäten aller Kanten e durch Platzhalterpfade aufgefüllt werden. Erhalten wir eine optimale Lösung, in der alle Platzhalterpfade akzeptiert wurden, so ist die Menge der akzeptierten Originalpfade wieder eine optimale Lösung für das Ursprungsproblem.

Um dafür zu sorgen, dass alle Platzhalterpfade p in jedem Fall akzeptiert werden, setzen wir $w(p) \coloneqq G+1$, wobei G die Summe der Gewichte aller Originalpfade ist. Nun hat der Algorithmus für identische Kapazitäten keine andere Wahl mehr, als jeden Platzhalterpfad zu akzeptieren, da jede Menge an Originalpfaden, die er stattdessen akzeptieren könnte, ein geringeres Gewicht hat und es nach Konstruktion genügend Kapazitäten gibt, um jeden Platzhalterpfad zu akzeptieren.

Für die Sortierung der Endknoten aller Pfade benötigen wir noch $\mathcal{O}(n+m)$ Zeit und erhalten insgesamt eine Laufzeit von $\mathcal{O}(n + m \cdot S(m))$. □

4.2 Approximationsalgorithmus von Güte 2 für Ringe

In diesem Abschnitt betrachten wir einen simplen Approximationsalgorithmus von Güte 2 für das Weighted-Call-Control-Problem in Ringen, d.h. einen Algorithmus, der die optimale Lösung bis auf den Faktor 2 annähert. Dabei gehen wir wie folgt vor:

Wir suchen die Kante e des Rings mit der geringsten Kapazität $c_{\min}$ und betrachten die beiden Pfadmengen $Q^*_{\overline{e}}$ und Q^*_e. Wir berechnen $Q^*_{\overline{e}}$ mit Hilfe von Abschnitt 4.1 als die optimale Lösung des Gewichteten Call-Control-Problem der Kette, die durch Weglassen der Kante e entsteht, mit allen Pfaden, die e nicht enthalten. Dagegen stellt Q^*_e die Menge der $c_{\min}$ Pfade größten Gewichts, die e enthalten, dar. Zwischen $Q^*_{\overline{e}}$ und Q^*_e wird nun die Menge mit größerem Gesamtgewicht als Approximationslösung von Güte 2 gewählt.

Theorem 4.2. *Der beschriebene Algorithmus ist ein Approximationsalgorithmus von Güte 2 für das Weighted-Call-Control-Problem in Ringen.*

Beweis. Es beschreibe e die Kante aus dem Algorithmus mit der geringsten Kapazität. Zunächst sehen wir, dass $Q^*_{\overline{e}}$ und Q^*_e zulässige Mengen darstellen. Das ist klar, da $Q^*_{\overline{e}}$ sogar eine zulässige Menge für die Kette ohne e ist und Q^*_e nur $c_{\min}$ Pfade enthält, weshalb in beiden Fällen keine Kapazitäten verletzt werden.

Sei OPT eine optimale Lösung für Weighted-Call-Control im Ring. OPT ist die disjunkte Vereinigung der beiden Mengen $\mathrm{OPT}_e \coloneqq \{p \in \mathrm{OPT} \mid p \text{ enthält } e\}$ und $\mathrm{OPT}_{\overline{e}} \coloneqq \{p \in \mathrm{OPT} \mid p \text{ enthält } e \text{ nicht}\}$. Es gilt insbesondere, dass $w(Q^*_{\overline{e}}) \geq w(\mathrm{OPT}_{\overline{e}})$ und $w(Q^*_e) \geq w(\mathrm{OPT}_e)$, da $Q_{\overline{e}}$ und Q_e in ihren Bereichen maximal gewählt wurden. Daher folgt für die berechnete Lösung Q^*:

$$2 \cdot w(Q^*) = 2 \cdot \max\{w(Q^*_e), w(Q^*_{\overline{e}})\} \geq w(\mathrm{OPT}_e) + w(\mathrm{OPT}_{\overline{e}}) = w(\mathrm{OPT}).$$

□

5 Fazit

Die vier Autoren des Artikels „Call Control in Rings“ haben es geschafft, das Call-Control-Problem in Ketten mit einem Algorithmus in linearer Zeit zu lösen. Diesen Algorithmus konnte man dann beim Problem in Ringen elegant einsetzen, um so zu einem Algorithmus zu gelangen, der mit polynomiellem Zeitaufwand eine optimale Lösung des Call-Control-Problems in Ringen berechnet. Dies ist vor allem aus diesem Grund interessant, da ein ähnlich wirkendes Problem, bei dem es darum geht den größtmöglichen k-färbbaren Teilgraphen in Kreisbogengraphen zu berechnen, *NP*-hart ist, wie in [4] nachgewiesen wurde. Ein Kreisbogengraph (V, E) mit Knotenmenge V ist dabei wie folgt definiert: Existiert eine Familie von Kreisbögen $(K_x)_{x \in V}$ um einen gemeinsamen Punkt mit festem Radius und gilt $(x, y) \in E \Leftrightarrow K_x \cap K_y \neq \emptyset$, so ist (V, E) ein Kreisbogengraph.

Für das Weighted-Call-Control-Problem wurde anhand der Vorarbeit in [2] ein effizienter Algorithmus in Ketten sowie ein darauf aufbauender Approximationsalgorithmus von Güte 2 für das Problem in Ringnetzwerken entwickelt. Außerdem zeigen die Autoren in [1], dass eine beliebig nahe Approximation in polynomieller Zeit gefunden werden kann. Dazu wird ein sogenanntes Polynomial-time approximation scheme (kurz PTAS) entwickelt, bei dem ein Parameter $\epsilon \in \mathbb{R}_+$ gegeben ist und der Algorithmus eine Lösung berechnet, dessen Gesamtgewicht um einen Faktor von $1 - e$ von dem der optimalen Lösung abweicht. Außerdem ist das Weighted-Call-Control-Problem in Ringen sogar mindestens so schwer wie das exact-matching-Problem in einem bipartiten Graphen (siehe [5]), also einem Graphen, der in zwei unabhängige disjunkte Teilgraphen U und V geteilt werden kann, sodass jeder Knoten in U mit mindestens einem Knoten in V und andersherum verbunden ist. Von diesem ist jedoch immer noch unbekannt, ob es in P liegt oder ob es ein *NP*-hartes Problem darstellt.

Literatur

[1] Udo Adamy, Christoph Ambühl, R. Sai Anand, and Thomas Erlebach. Call control in rings. *Algorithmica*, 47:217–238, 2007.

[2] Martin C. Carlisle and Errol. L. Lloyd. On the k-coloring of intervals. *Discrete Applied Mathematics*, 59:225–235, 1995.

[3] Harold N. Gabow and Robert Endre Tarjan. A linear-time algorithm for a special case of disjoint set union. *Journal of Computer and System Sciences*, 30:209–221, 1985.

[4] Michael R. Garey, David S. Johnson, Gary L. Miller, and Christos H. Papadimitriou. The complexity of coloring circular arcs and chords. *SIAM Journal on Algebraic Discrete Methods*, 1:216–227, 1980.

[5] D. S. Hochbaum and A. Levin. *Cyclical scheduling and multi-shift scheduling: complexity and approximation algorithms.* UC Berkeley, June 2003.