Gesammelte Mitschriften zur Vorlesung
Algorithmen für verteilte Systeme

Sebastian Forster
Fachbereich Informatik
Universität Salzburg

Mit Beiträgen von Simon Bauer, Ivica Brezovic, András Czuczi, Cansu Demir, Christian Edelmayer, Florian Feik, Falco Gödde, Sebastian Landl, Mirna Mrazovic, Michael Nening und Sara Seidl

25. Mai 2023

Inhaltsverzeichnis

1 Leader Election im Ring

1.1 Problemstellung

Ein *Ring*(-Netzwerk) besteht aus n Knoten und n Kanten, wobei jeder Knoten zwei (Ports zu) Nachbarn hat. Einer der Nachbarn befindet sich „im Uhrzeigersinn“ (*Clockwise*) und der andere Nachbar befindet sich „gegen den Uhrzeigersinn“ (*Counter-Clockwise*). Ein Beispiel für so ein Netzwerk findet sich in Abb. 1.1.

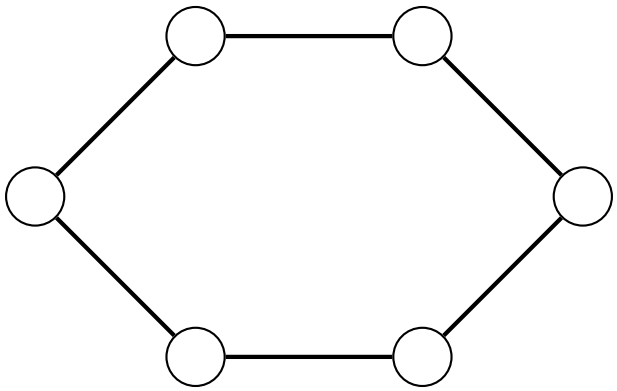


Abbildung 1.1: Beispiel für einen Ring mit 6 Knoten

Im *Leader Election* Problem müssen sich die Knoten eines Netzwerks auf einen einzigen *Leader* einigen. Alle anderen Knoten des Netzwerks werden dann *Follower* genannt. Gesucht ist also ein Algorithmus, der für jeden Knoten entscheidet, ob er der Leader ist oder ein Follower, wobei insgesamt genau ein Knoten der Leader sein darf. Der Zweck von Leader Election kann sein, dass der Leader in einem (dezentralen) Netzwerk Koordinationsaufgaben übernehmen kann.

Es mag zwar zunächst trivial erscheinen, einen Leader zu bestimmen, doch in einem dezentralen Modell gestaltet sich das nicht so einfach, wenn jeder Knoten zunächst gleichwertig ist und sich somit auch jeder Knoten als Leader eignet. Es gibt in solchen Netzwerken keinen „besseren“ oder „schlechteren“ Leader. Daher gilt, dass die Lösung nicht eindeutig ist, und insbesondere gibt es für ein Netzwerk mit n Knoten auch n verschiedene Lösungen. Die Schwierigkeit für die Knoten des Netzwerks besteht daher eigentlich darin, eine konsistente Entscheidung für eine dieser Lösungen zu treffen.

1.2 Kommunikationsmodell

In unserem Message-Passing Kommunikationsmodell gibt es n Knoten im Netzwerk. Jeder Knoten kennt die Anzahl seiner Nachbarn und kann über nummerierte Ports beliebige Nachrichten an seine Nachbarn senden und von diesen empfangen. Ein Port ist eine Verbindung von einem Knoten zu einem Nachbarknoten. Es gibt folgende zwei Arten, wie die Kommunikation stattfinden kann.

1. *Synchron:* Im *synchronen* Modell ist die Kommunikation rundenbasiert. Jede Runde besteht aus folgenden Schritten:
   (a) Nachrichten von Nachbarn empfangen (entfällt in erster Runde)
   (b) Interne Berechnungen

(c) Nachrichten an Nachbarn senden

Es gibt dabei eine globale Uhr, welche den Beginn jeder Runde vorgibt.

2. *Asynchron:* Im *asynchronen* Modell ist die Kommunikation Event-basiert. Dabei werden Knoten immer dann aktiv, wenn sie Nachrichten empfangen oder wenn für sie ein externes Initialisierungsevent[1] stattfindet. Sobald ein Knoten aktiv wird, darf er interne Berechnungen durchführen und dann Nachrichten an seine Nachbarn senden, um anschließend – bis zur Aktivierung durch das nächste Event – wieder inaktiv zu werden. Im asynchronen Modell wird nur garantiert, dass jede Nachrichtenübermittlung endliche Zeit benötigt. Es gibt jedoch insbesondere keine Garantie, dass jede Nachrichtenübermittlung gleich lange dauert.

Das synchrone Modell kann als Spezialfall des asynchronen Modells verstanden werden, in dem für alle Knoten gleichzeitig ein externes Initialisierungsevent stattfindet und jede Nachrichtenübermittlung genau eine Zeiteinheit dauert.

Wir betrachten in diesen Modellen zwei Komplexitätsmaße, nämlich *Zeitkomplexität* und *Nachrichtenkomplexität*. Die Zeitkomplexität eines Algorithmus im synchronen Modell gibt die Anzahl der *Runden* bis zur Lösung des Problems an. Die Zeitkomplexität eines Algorithmus im asynchronen Modell gibt die Anzahl der *Zeiteinheiten* von der letzten erstmaligen Aktivierung eines Knotens bis zur Lösung des Problems an, wenn jede Nachrichtenübermittlung höchstens eine Zeiteinheit benötigt. Dabei ist es wichtig zu betonen, dass mit der Zeitkomplexität interne Berechnungszeit *nicht* gemessen wird, diese wird also in unseren Analysen außen vor gelassen. Die Nachrichtenkomplexität eines Algorithmus gibt in beiden Modellen die Gesamtzahl der durch alle Knoten gesendeten Nachrichten an.

1.2.1 Weitere Varianten

Man kann zwischen einem synchronen und einem asynchronen Modell unterscheiden (siehe oben). Ebenso können Knoten anonym oder identifizierbar sein. Ein Knoten ist identifizierbar, wenn er eine eindeutige ID besitzt (oft auch UID für *unique* ID genannt). In der Regel sind solche IDs Bit-Strings, oftmals der Länge $O(\log n)$. Die IDs müssen nicht kontinuierlich vergeben sein, es kann auch Lücken geben (es müssen also nicht alle möglichen Zahlenwerte vergeben sein). Weiters kann man noch unterscheiden, ob die Anzahl der Knoten n globales Wissen ist oder nicht. Ist die Anzahl der Knoten n kein globales Wissen, dann nennt man das Modell *uniform*.

1.3 Einfacher Clockwise Algorithmus

Für den Leader-Election Algorithmus, den wir im Folgenden vorstellen verlassen wir das Setting anonymer Netzwerke und nehmen nun an, dass jeder Knoten v eine eindeutigen ID $\mathrm{ID}(v)$ hat. Wir formulieren den Algorithmus zunächst für synchrone, non-uniforme Ring-Netzwerke.

Im *(einfachen) Clockwise Algorithmus* verhält sich jeder Knoten v folgendermaßen:
Runde 1:

1 v setzt $L_v := \mathrm{ID}(v)$ (lokale Variable für kleinste bisher gesehene ID)
2 v sendet L_v an Nachbar im Uhrzeigersinn

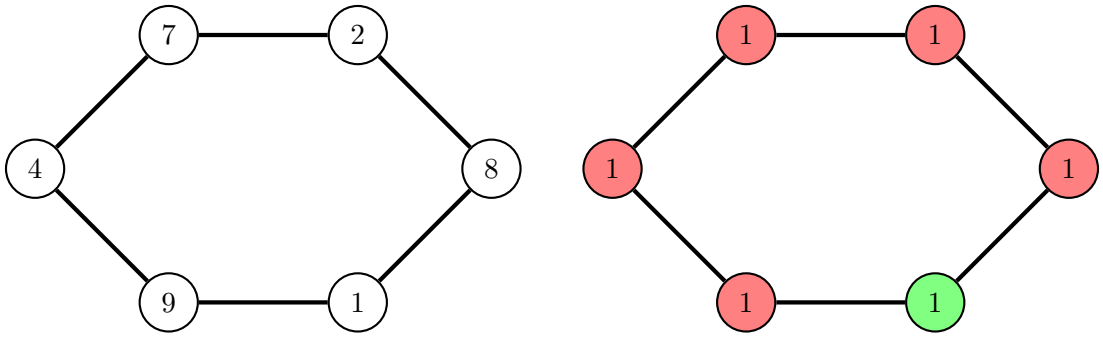
[1] Das Auftreten dieses externen Initialisierungsevents ist für mindestens einen Knoten notwendig, damit Algorithmen in diesem Modell starten können. Sobald ein Knoten durch den Empfang einer Nachricht das erste Mal aktiv wurde, ist das externe Initialisierungsevent für ihn nicht mehr notwendig.

Runde $2 \leq r \leq n$:

```
if v empfängt Nachricht M then
    if M < L_v then
        v setzt L_v := M
        v sendet L_v an Nachbar im Uhrzeigersinn
```

Runde $n + 1$:

```
if L_v = ID(v) then
    v wird zum Leader
else
    v wird zum Follower
```



(a) Initial vergebene IDs

(b) Einteilung in Leader (grün) und Follower (rot)

Abbildung 1.2: Beispiel für Leader Election im Ring

Wir betrachten das Beispiel in Abb. 1.2. Die Zahlen in Abb. 1.2a sind die IDs der Knoten. Diese sind schon vorgegeben und eindeutig. In der ersten Runde initialisieren wir für jeden Knoten die lokale Variable $L_v = \mathrm{ID}(v)$ und senden an den Nachbarn im Uhrzeigersinn eine Nachricht mit dem Inhalt L_v. In der nächsten Runde wird empfangen, wobei jeder Knoten, der einen höheren Wert als den eigenen Wert empfangen hat, den höheren (also den empfangenen) Wert übernimmt. Der Knoten links unten, welcher zuvor den Wert 9 hatte, übernimmt beispielsweise den empfangenen Wert 1. Alle Knoten, die ihren Wert verändert haben, senden den neuen Wert im Uhrzeigersinn an den Nachbarn. So läuft der Algorithmus weiter und am Ende hat jeder Knoten L_v auf 1 gesetzt und die Einteilung in Leader und Follower wird vorgenommen. Im Ergebnis in Abb. 1.2b sind die Follower rot markiert und der Leader grün. Der Knoten, welcher die ID 1 hat, wurde zum Leader, weil 1 die kleinste ID im Netzwerk ist.

1.3.1 Analyse

Theorem 1.1. *Nach $n + 1$ Runden macht der einfache Clockwise Algorithmus den Knoten mit der kleinsten ID zum Leader und alle anderen zu Followern.*

Beweis. Wir wollen im Beweis die Beobachtung, dass die kleinste ID in jeder Runde weitergeleitet wird, formalisieren. Sei z der Knoten mit der (initial) kleinsten ID. Für jedes $i \geq 0$ sei v_i der Knoten, der von z aus nach Traversieren von i Kanten im Uhrzeigersinn erreicht wird. Anders ausgedrückt ist der Knoten v_i der Knoten, welchen man von dem Knoten z aus erreichen kann, wenn man i Kanten

im Uhrzeigersinn genommen hat. Der Knoten v_0 wäre beispielsweise z selbst, da man 0 Kanten benötigt, um z von z aus zu erreichen. Der Knoten v_1 wäre der Nachbar von z im Uhrzeigersinn und so weiter. Wir verwenden nun folgende Induktionsaussage.

Induktionsaussage: Der Knoten v_i empfängt die Nachricht $\mathrm{ID}(z)$ (spätestens) in Runde $i+1$ (für jedes $0 \leq i \leq n$).

Diese Aussage kann mittels vollständiger Induktion bewiesen werden. Wir werden diesen Teil des Beweises an dieser Stelle nicht präsentieren, da er nur Standard-Argumente benötigt.

Wir beobachten nun, dass jeder von L_v angenommene Wert einer ID im Netzwerk entspricht.[2]. Sobald ein Knoten v also eine Nachricht mit dem Inhalt $\mathrm{ID}(z)$ empfängt, wird $L_v = \mathrm{ID}(z)$ gesetzt und L_v wird daraufhin keinen anderen Wert mehr annehmen.

Da es für jeden Knoten $v \neq z$ ein i mit $1 \leq i \leq n-1$ gibt, so dass $v = v_i$ gilt, folgt aus der Induktionsaussage, dass v innerhalb von $i+1 \leq n$ Runden die Nachricht $\mathrm{ID}(z)$ empfängt. Wie soeben argumentiert, gilt ab diesen Zeitpunkt $L_v = \mathrm{ID}(z) \neq \mathrm{ID}(v)$. In Runde $n+1$ gilt daher $L_v = \mathrm{ID}(v)$ nur für den Knoten $v = z$. Dieser wird daher korrekterweise zum Leader und die anderen Knoten zu Followern. □

Theorem 1.2. *Der Clockwise Algorithmus sendet insgesamt höchstens n^2 Nachrichten.*

Beweis. Wann werden im Algorithmus Nachrichten gesendet? Ein Knoten v nur direkt nachdem die Variable L_v verändert wurde. Sprich, Knoten v sendet nur, nachdem L_v initialisiert oder erhöht wurde. Jeder von L_v angenommene Wert entspricht einer ID im Netzwerk. Daher kann L_v während des Algorithmus höchstens n verschiedene Werte annehmen, da es n verschiedene Knoten gibt. Deshalb sendet jeder Knoten höchstens n Nachrichten. Weil jeder Knoten höchstens n Nachrichten sendet und es n Knoten gibt, hat der Algorithmus somit eine Nachrichtenkomplexität von n^2. □

1.4 Allgemeiner Clockwise Algorithmus

Der Clockwise Algorithmus kann auch an das asynchrone, uniforme Modell angepasst werden [Lan77, CR79]. Es gilt wieder die Annahme, dass jeder Knoten v eine eindeutige ID $\mathrm{ID}(v)$ hat. Im asynchronen Modell betrachten wir keine Runden, sondern Events. Es gibt zwei mögliche Events: Initialisierung oder Empfang einer Nachricht. Ein (externes) Initialisierungsevent ist notwendig, damit der Algorithmus gestartet wird, dabei können auch mehrere Knoten gleichzeitig initialisiert werden. Im Gegensatz zum synchronen und non-uniformen Modell ist den Knoten nun nicht bekannt, zu welchem Zeitpunkt die kleinste ID den Ring vollständig durchlaufen hat. Daher werden Knoten zu Followern, sobald sie eine kleinere ID als ihre eigene empfangen haben. Der Leader hingegen erkennt sich dadurch, dass er seine eigene ID empfängt, nachdem diese den Ring vollständig durchlaufen hat. Der Algorithmus für jeden Knoten v sieht folgendermaßen aus:

[2] Ganz formal gesehen könnte man diese Aussage wieder per Induktion beweisen

```
1  if v wird initialisiert then
2      v setzt L_v := ID(v) (lokale Variable für kleinste bisher gesehene ID)
3      v sendet L_v an Nachbar im Uhrzeigersinn
4  if v empfängt Nachricht M then
5      if v uninitialisiert then initialisiere v
6      if M < L_v then
7          v setzt L_v := M
8          v wird zum Follower (sofern nicht bereits vorher geschehen)
9          v sendet L_v an Nachbar im Uhrzeigersinn
10     if M = ID(v) then
11         v wird zum Leader
```

1.4.1 Analyse

Theorem 1.3. *Höchstens $2n - 1$ Zeiteinheiten nach der ersten Aktivierung eines Knotens bestimmt der asynchrone Clockwise Algorithmus den Knoten mit der kleinsten ID zum Leader und alle anderen zu Followern (wenn die Übermittlung jeder Nachricht höchstens eine Zeiteinheit dauert).*

Beweisskizze. Wir starten die Zeitmessung mit dem ersten Knoten, für den ein externes Initialisierungsevent stattfindet. Bei dieser ersten Aktivierung wird der Knoten auf jeden Fall eine Nachricht (seine ID) verschicken. Das heißt, er weckt den Nachbarknoten auf. Somit wird der Nachbarknoten aktiv und das gegenseitige Aktivieren setzt sich fort. Dann vergehen höchstens $n - 1$ Zeiteinheiten bis der Knoten z mit der kleinsten ID eine Nachricht empfängt und das erste Mal aktiv wird. Wenn das erste Mal der Knoten mit der kleinsten ID aktiv wurde, dann ist das Argument ähnlich wie im synchronen Fall: nach höchstens n weiteren Zeiteinheiten hat jeder Knoten $\mathrm{ID}(z)$ empfangen und sich entschieden. □

1.5 Wartezeit-Algorithmus

Es ist klar, dass jeder Leader-Election-Algorithmus im Ring $\Omega(n)$ Runden benötigt, da ansonsten relevante Informationen aus einem Teil des Netzwerks nicht alle Knoten erreichen könnte. Daher ist eine asymptotische Zeitkomplexität von $O(n)$ optimal. Für die Nachrichtenkomplexität gilt ebenfalls die triviale untere Schranke von $\Omega(n)$, da jeder Knoten mindestens einmal eine Nachricht empfangen muss. Die Lücke zwischen dieser unteren Schranke und der Nachrichtenkomplexität von $O(n^2)$ des Clockwise Algorithmus ist sehr groß. Man kann sich also die Frage stellen, ob man einen Leader mit signifikant weniger als n^2 Nachrichten bestimmen kann. Tatsächlich ist das möglich. Die Idee hinter der Lösung, die wir dazu im Folgenden präsentieren ist, das Vorhandensein von Synchronizität auszunutzen. Insbesondere wird es im Algorithmus explizit als Zustimmung interpretiert, wenn *keine* Nachricht gesendet wird.

1.5.1 Algorithmus

Für den entsprechenden Algorithmus [AW98] führt jeder Knoten v in jeder Runde die folgenden Anweisungen aus:

```
if Leader-Nachricht empfangen then
    Werde zum Follower
    Leite Leader-Nachricht im Uhrzeigersinn weiter
if #Runden = ID(v) · n + 1 und v noch kein Follower then
    Werde zum Leader
    Sende Leader-Nachricht an Nachbar im Uhrzeigersinn
```

Ein Knoten v wird also dann aktiv, wenn die aktuelle Runde gleich $\mathrm{ID}(v) \cdot n + 1$ ist. (Der Offset $+1$ ist notwendig, weil die Zählung der Runden mit 1 beginnt und die IDs mit 0.) Falls dem so ist, wird der Knoten zum Leader und sendet an seinen Nachbarn die Nachricht, dass ein Leader bestimmt wurde. Diese Nachricht wird in den folgenden Runden weitergeleitet und jeder Knoten, der die Nachricht empfängt, wird zum Follower.

1.5.2 Analyse

Wir unterteilen die Runden in Phasen der Länge n und in jeder Phase (also alle n Runden) entscheidet sich höchstens ein Knoten, Leader zu werden. Damit der Algorithmus korrekt ist muss der Leader alle anderen Knoten informiert haben bevor die nächste Phase beginnt, also innerhalb von n Runden. Das wird durch das Weiterleiten der Nachricht im Ring garantiert. Sobald sich ein Knoten entscheidet, Leader zu sein, wissen wir, dass vor Beginn der nächsten Phase alle anderen Knoten zu Followern geworden sind. Daraus folgt, dass die Anzahl der Nachrichten $O(n)$ ist. (Es gilt sogar, dass es maximal n Nachrichten sind, denn jeder Knoten sendet nur einmal.) Die Anzahl der Runden ist $\Theta(n \cdot \min_v \mathrm{ID}(v))$ und gibt im Prinzip an, wie lange wir hochzählen müssen, um eine ID, die im Netzwerk vergeben wurde, zu erreichen.[3]

Bemerkung. Bereits für IDs, die als ganze Zahlen im Bereich von 0 bis $2n - 1$ interpretiert werden können, könnte die niedrigste ID den Zahlenwert n haben. Dann benötigt der Algorithmus $\Theta(n^2)$ Runden.

1.6 Anonyme Netzwerke

In einem *anonymen Netzwerk* haben die Knoten keine IDs. Eine Unterscheidung a priori ist somit nicht möglich. Es stellt sich heraus, dass diese Einschränkung zu stark ist, um einen Leader in einem Ring zu bestimmen.

Theorem 1.4 ([Ang80])**.** *Leader Election in anonymen Netzwerken ist mit deterministischen Algorithmen unmöglich, sogar im synchronen Ring.*

Um dieses Theorem zu beweisen, muss natürlich zunächst einmal festgelegt werden, was genau unter *deterministischen Algorithmen* in unserem verteilten Modell verstanden wird. Wir definieren einen deterministischen Algorithmus wie folgt:

Ein deterministischer Algorithmus ist beschreibbar durch eine Funktion f, welche wiederholt auf jeden Knoten v angewendet wird. Als *Eingabe* für diese Funktion dient die Anzahl an Knoten n, die Anzahl an Ports von v und die gesamte Historie von v. Die Historie eines Knotens ist das nach Empfangsreihenfolge geordnete Protokoll aller bisher empfangen Nachrichten mit Aufzeichnung des Inhalts der Nachricht und des Eingangsports, wobei im synchronen Modell zusätzlich noch

[3] Zur Erinnerung: Die IDs sind in der Regel Bit-Strings und die kleinste ID im Netzwerk muss nicht unbedingt 0 sein.

die Rundenzahl bei Empfang aufgezeichnet wird. Die *Ausgabe* der Funktion sind die zu sendenden Nachrichten des Knotens an Ausgangsports und gegebenenfalls die Entscheidung, ob v Leader oder Follower wird. Anschließend sagen wir, die Funktion löst das Problem, wenn für jedes Netzwerk nach endlich vielen Anwendungen von f ein Knoten als Leader und alle anderen als Follower festgelegt wurden.

Deterministisch bedeutet hier, dass f eine „echte“ Funktion ist, die jeder möglichen Eingabe eine eindeutige Ausgabe zuordnet. Nicht-deterministisch würde bedeuten, dass es Eingaben mit mehr als einer möglichen Ausgabe gibt, wie es beispielsweise bei randomisierten Algorithmen auftreten kann.

1.6.1 Beweis von Theorem 1.4

Beweis. Wir werden die Unmöglichkeit, sprich das obige Theorem, nun beweisen. Dafür benötigen wir folgende Induktionsaussage, welche im Anschluss an den Unmöglichkeitsbeweis selbst bewiesen wird. Vorerst arbeiten wir unter der Annahme, dass die Induktionsaussage korrekt ist.

Induktionsaussage. *Für jeden deterministischen Algorithmus sind in jeder Runde jeweils die im und gegen den Uhrzeigersinn empfangenen Nachrichten für alle Knoten gleich.*

Aus der Induktionsaussage folgt nun, dass in jeder Runde alle Knoten die gleiche Historie haben. Zudem ist die Anzahl der Ports für jeden Knoten gleich und in jeder Runde ist die Eingabe der Funktion f für alle Knoten gleich. Nachdem die Funktion f deterministisch und die Eingabe für alle Knoten gleich ist, folgt, dass auch die Ausgabe der Funktion f für alle Knoten gleich ist. Wenn f eine Entscheidung trifft, dann muss diese Entscheidung für alle Knoten konsistent sein. Somit bedeutet das also: in jeder Runde macht f entweder alle Knoten zu Leadern, alle Knoten zu Followern oder trifft für keinen Knoten eine Entscheidung. Schlussendlich folgt daraus, dass f also entweder ein falsches Ergebnis liefert oder nicht terminiert, womit das vorher erwähnte Theorem bewiesen wäre (unter der Voraussetzung, dass die Induktionsaussage gilt). □

1.6.2 Beweis der Induktionsaussage

Wir werden nun die im Unmöglichkeitsbeweis aufgestellte Induktionsaussage beweisen.

Beweis der Induktionsaussage. *Induktionsbasis:* Gilt trivialerweise, da in der ersten Runde noch keine Nachrichten empfangen werden.

Induktionsschritt: Wir zeigen nun, dass die Induktionsaussage für jede Runde $k \geq 2$ gilt, unter der Annahme dass sie für alle Runden $i \leq k-1$ gilt. Aus der Gültigkeit der Induktionsaussage für alle vorherigen Runden, folgt, dass jeder Knoten am Beginn von Runde k die gleiche Historie hat. Ebenso sind alle anderen Parameter für alle Knoten gleich (Anzahl der Ports, Anzahl der Knoten n). Somit ist die Eingabe der Funktion f in Runde k für alle Knoten gleich. Aus gleicher Eingabe folgt gleiche Ausgabe, also ist die Ausgabe der Funktion f für alle Knoten gleich. Das heißt, alle Knoten senden die jeweils gleiche Nachricht im und gegen den Uhrzeigersinn in Runde k. Aus der Ring-Topologie folgt: in Runde k empfangen alle Knoten jeweils die gleiche Nachricht im und gegen den Uhrzeigersinn. Somit ist Induktionsaussage bewiesen und der vorherige Unmöglichkeitsbeweis vollständig. □

Zusammengefasst bedeutet das, dass sich nie ein Konten zum Leader oder Follower werden kann ohne dass alle anderen Knoten die gleiche Entscheidung treffen, was dem Ziel der Bestimmung

eines einzelnen Leaders widerspricht. Es sei an dieser Stelle noch einmal darauf hingewiesen, dass diese Unmöglichkeit nicht für alle Netzwerke gilt, sondern nur für anonyme Ringe.

1.7 Radius Growth Algorithmus

Bisher haben wir den Clockwise Algorithmus kennengelernt, welcher $O(n)$ Runden und $O(n^2)$ Nachrichten benötigt, um einen Leader zu bestimmen. Ebenso haben wir einen nachrichteneffizienteren Algorithmus kennen gelernt, welcher (unter gewissen Annahmen) zwar deutlich mehr Runden, aber dafür lediglich $O(n)$ Nachrichten benötigt. Wir stellen nun einen Algorithmus vor, der die geringe Rundenzahl und die geringe Nachrichtenkomplexität dieser beiden Algorithmen annähernd vereint und $O(n)$ Runden und $O(n \log n)$ Nachrichten benötigt.

Der Radius Growth Algorithmus [HS80] für Leader Election im Ring funktioniert nach folgendem Prinzip: jeder Knoten überprüft, ob er in einer gewissen Umgebung der Leader ist, was bedeutet, dass er in dieser Umgebung die kleinste ID hat. Wenn ja, so wird diese Umgebung vergrößert.

Die Ausgangssituation des Algorithmus ist, dass jeder der n Knoten als Leader infrage kommt. Im Laufe des Algorithmus werden sich dann $n - 1$ Knoten dazu entscheiden Follower zu sein und der übrige Knoten wird der Leader sein. Knoten, die noch keine Follower sind nennen wir **aktiv**.

Die Runden werden in $\lceil \log n \rceil$ aufeinanderfolgende Phasen unterteilt, wobei die i-te Phase $2^{i-1} + 1$ Runden dauert. In der ersten Runde jeder Phase schickt jeder aktive Knoten seine ID an seine Nachbarn im Ring. In allen anderen Runden, wenn ein Knoten einen Nachricht erhält, sendet er diese weiter und entscheidet sich Follower zu sein, falls die empfangene ID größer als seine eigene ist. Derjenige Knoten, der am Ende noch aktiv ist, wird zum Leader.

Erste Runde jeder Phase:

```
1 if v noch kein Follower then
2     v sendet ID(v) an Nachbarn im und gegen den Uhrzeigersinn
```

Jede andere Runde:

```
1 if v empfängt Nachricht M von Nachbar gegen (im) den Uhrzeigersinn then
2     if M < ID(v) then
3         v wird zum Follower (sofern nicht bereits vorher geschehen)
4         if nicht letzte Runde der Phase then
5             v sendet M an Nachbar im (gegen) Uhrzeigersinn
```

Zusätzlich in letzter Runde der letzten Phase:

```
1 if v ist noch kein Follower then v wird zum Leader
```

1.7.1 Korrektheit und Laufzeit

Theorem 1.5. *Der Radius Growth Algorithmus bestimmt einen eindeutigen Leader.*

Beweis. Sei z der Knoten mit der kleinsten ID. Klarerweise erhält der Knoten z nie eine Nachricht mit einer ID, die kleiner ist als seine eigene. Er wird deshalb nie zum Follower und bleibt bis zur letzten Runde der letzten Phase aktiv, weshalb er zum Leader wird. Insbesondere ist z in Phase $\ell = \lceil \log n \rceil$, der letzten Phase, noch aktiv. In dieser Phase schickt z eine Nachricht mit seiner ID im

und gegen den Uhrzeigersinn los, welche jeweils $2^{\ell-1}$ Knoten erreicht. Wegen

$$2^{\ell-1} = 2^{\lceil \log n \rceil - 1} \geq 2^{\log n - 1} = \frac{2^{\log n}}{2} = \frac{n}{2}$$

erreicht in Phase ℓ die Nachricht mit ID von z in jede Richtung mindestens $n/2$ Knoten und somit alle anderen Knoten, die spätestens dann zu Followern werden, da sie höhere ID als z haben. Somit bestimmt der Algorithmus eine gültige Einteilung in Leader und Follower. □

Theorem 1.6. *Der Radius Growth Algorithmus benötigt $O(n)$ Runden.*

Beweis. Die Phase i benötigt $2^{i-1} + 1 \leq 2^i$ Runden (großzügig abgeschätzt). Somit beträgt die Gesamtzahl an Runden: $\sum_{i=1}^{\lceil \log n \rceil} 2^i \leq 2^{\lceil \log n \rceil + 1} \leq 2^{\log n + 2} \leq 4 \cdot 2^{\log n} = 4n$. Das folgt aus der Summenformel für die geometrische Reihe. □

Lemma 1.7. *Für jeden aktiven Knoten gilt am Ende von Phase i: alle anderen Knoten in Distanz bis zu 2^{i-1} sind inaktiv.*

Die Aussage dieses Lemmas ist, dass es am Ende von Phase i im Bereich der nächsten 2^{i-1} einen eindeutigen Leader gibt.

Beweis. Wir führen einen Widerspruchsbeweis. Sei v ein Knoten, der am Ende von Phase i noch aktiv ist. Angenommen es gibt einen Knoten w mit Distanz höchstens 2^{i-1} vom zu Knoten v, der am Ende von Phase i noch aktiv ist. Dann war w auch am Anfang von Phase i aktiv.

Wir unterscheiden hier grundsätzlich drei Fälle:

- Falls $\mathrm{ID}(v) > \mathrm{ID}(w)$ gilt, erhält v in Phase i eine Nachricht mit $\mathrm{ID}(w)$ und wird daher inaktiv. Dies steht im Widerspruch zur Annahme, dass v am Ende von Phase i noch aktiv ist.
- Falls $\mathrm{ID}(v) < \mathrm{ID}(w)$ gilt, erhält w in Phase i eine Nachricht mit $\mathrm{ID}(v)$ und wird daher inaktiv. Dies steht im Widerspruch zur Annahme, dass w am Ende von Phase i noch aktiv ist.
- Der Fall $\mathrm{ID}(w) = \mathrm{ID}(v)$ kann nicht eintreten, da die IDs eindeutig sind.

□

Lemma 1.8. *Für $i \geq 2$ ist die Anzahl aktiver Knoten am Beginn von Phase i höchstens $n/2^{i-2}$.*

Beweis. Wegen Lemma 1.7 wissen wir, dass jedem aktiven Knoten eindeutig die 2^{i-2} nächsten inaktive Knoten im Uhrzeigersinn zugeordnet werden können (siehe Abb 1.3). Sei a die Anzahl an aktiven Knoten, dann gilt, dass $a \cdot 2^{i-2} \leq n$, weil a multipliziert mit der Anzahl an a zugeordneten inaktiven Knoten nicht mehr als die Gesamtzahl an Knoten im Ring sein kann. Daher gilt dann $a \leq n/2^{i-2}$.

□

Theorem 1.9. *Der Radius Growth Algorithmus sendet höchstens $O(n \log n)$ Nachrichten.*

Beweis. Zu Beginn jeder Phase schickt jeder aktive Knoten zwei Nachrichten los, eine im und eine gegen den Uhrzeigersinn. Diese Nachrichten werden in jeder Runde der Phase einmal weitergeleitet. Somit ist die Anzahl der Nachrichten in Phase $i \geq 2$ höchstens $\frac{n}{2^{i-2}} \cdot 2 \cdot 2^{i-1} = 4n$ ($\frac{n}{2^{i-2}}$ ist die maximale Anzahl an aktiven Knoten (Lemma 1.8); jeder aktive Knoten schickt 2 Nachrichten; die

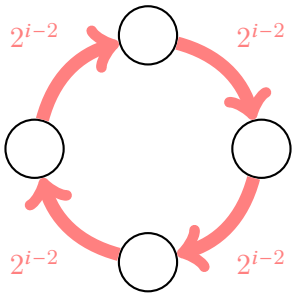


Abbildung 1.3: Visualisierung zur eindeutiger Zuordnung inaktiver Knoten zu aktiven Knoten

Anzahl der Runden der Phase i beträgt $2^{i-1}+1$ und in jeder dieser Runden, außer der letzten Runde der Phase, werden die zu Beginn erzeugten Nachrichten einmal weitergeleitet). In Phase 1 beträgt die Anzahl der Nachrichten klarerweise $2n$. In jeder Phase werden somit höchstens $4n$ Nachrichten versendet werden Da es $\lceil \log n \rceil$ viele Phasen gibt, liegt die Nachrichtenkomplexität des Radius Growth Algorithmus in $O(n \log n)$. □

1.7.2 Zusammenfassung

Es wird immer der Knoten mit der kleinsten ID zum Leader. Die Laufzeitkomplexität beträgt $O(n)$ Runden, während die Nachrichtenkomplexität $O(n \log n)$ ist. Der Radius Growth Algorithmus kann auch als asynchroner Algorithmus formuliert werden.

1.8 Randomisierter Algorithmus

Eingangs haben wir festgestellt, dass es nicht möglich ist in einem anonymen Ring mittels eines deterministischen Algorithmus einen Leader zu bestimmen. Wir werden nun zu diesem Setting zurückkehren und einen *randomisierten* Algorithmus kennen lernen, der einen Leader bestimmt.

Im anonymen Ring sind a priori keine IDs vergeben. Jedoch treffen wir die Annahme, dass die Gesamtzahl n an Knoten im Ring jedem Knoten bekannt ist. Die Grundidee des randomisierten Algorithmus [IR90, AW98] ist nun, dass wir mit IDs arbeiten, die zufällig vergeben wurden, und folgendermaßen vorgehen:

1. Jeder Knoten wählt uniform zufällige ID von 0 bis $2n-1$
2. Knoten führen einfachen Clockwise Algorithmus mit gewählten IDs aus
3. Knoten verifizieren das Ergebnis

Das Problem an der zufälligen Vergabe der IDs ist, dass es keine Garantie dafür gibt, dass keine ID mehrfach vergeben wird und es somit mehrere Knoten mit der kleinsten ID gibt. Dies würde zu einem falschen Ergebnis führen würde, weil dadurch mehr als ein einziger, eindeutiger Leader bestimmt werden würde. Um dieses Problem zu lösen, müssen die Knoten in Schritt 3 verifizieren, ob das Ergebnis, also die Einteilung in Leader und Follower korrekt ist (es also genau einen Leader und sonst nur Follower gibt), was in $O(n)$ Runden und mit $O(n)$ Nachrichten möglich ist (siehe Übungsaufgabe). Wenn das Ergebnis nicht korrekt ist, so muss wieder bei Schritt 1 angefangen werden. Diese Schritte werden so lange wiederholt, bis ein eindeutiger Leader bestimmt wurde.

1.8.1 Erfolgswahrscheinlichkeit einer Iteration

Eine Iteration des Algorithmus ist genau dann erfolgreich, wenn die kleinste vergebene ID genau einmal vergeben wurde. Davon ausgehend die Erfolgswahrscheinlichkeit exakt zu bestimmen, ist jedoch etwas schwierig. Daher wenden wir folgenden Analysetrick an: Wir wissen, dass wenn die ID 0 genau einmal vergeben wird, die entsprechende Iteration erfolgreich ist. Nun lassen wir einfach die Möglichkeit außer Acht, dass eine ID echt größer als 0 nur einmal vergeben wird und die kleinste ID ist, was jedoch kein Problem ist, weil wir damit unsere Erfolgswahrscheinlichkeit nicht überschätzen.

Wir betrachten nun für jeden Knoten $v \in V$ die Zufallsvariable

$$X_v = \begin{cases} 1 & \text{falls Knoten } v \text{ ID } 0 \text{ erhält} \\ 0 & \text{ansonsten} \end{cases}$$

sowie die Zufallsvariable $X := \sum_{v \in V} X_v$ für die Anzahl an Knoten, die ID 0 erhalten. Da jeder Knoten seine ID uniform zufällig aus dem Bereich von 0 bis $2n-1$ wählt, gilt $\Pr[X_v = 1] = \frac{1}{2n}$. Anders ausgedrückt ist X_v Bernoulli-verteilt mit Erfolgswahrscheinlichkeit $p := \frac{1}{2n}$

Da die Vergabe der ID für jeden Knoten separat erfolgt, sind die Zufallsvariablen X_v stochastisch unabhängig. Die Summe dieser Zufallsvariablen, also die Zufallsvariable X, ist deshalb binomialverteilt (mit Parametern n und p). Daher gilt $\Pr[X = k] = \binom{n}{k} p^k (1-p)^{n-k}$ für alle $0 \leq k \leq n$ Insbesondere erhalten wir für $k = 1$ die Wahrscheinlichkeit, dass ID 0 genau einmal vergeben wird. Für diese Wahrscheinlichkeit nehmen wir nun folgende Abschätzung vor:

$$\Pr[X = k] = n \cdot p(1-p)^{n-1} = \frac{1}{2} \cdot \left(1 - \frac{1}{2n}\right)^{n-1} \geq \frac{1}{2} \cdot \left(1 - \frac{n-1}{2n}\right) \geq \frac{1}{2} \cdot \left(1 - \frac{n}{2n}\right) = \frac{1}{4}.$$

Hier haben wir die *Bernoulli-Ungleichung* angewendet, die allgemein besagt, dass $(1+x)^n \geq 1 + xn$ für $x \geq -1$ und $n \geq 0$ (siehe (A.16)). Somit ist jede Iteration mit Wahrscheinlichkeit $q \geq \frac{1}{4}$ erfolgreich.

1.8.2 Korrektheit

Aufgrund des Verifikationsschrittes am Ende jeder Iteration ist klar, dass der Algorithmus ein korrektes Ergebnis liefert sobald er terminiert. Um die Korrektheit des Algorithmus zu beweisen, müssen wir also zeigen, dass er terminiert. Allerdings wäre es falsch zu behaupten, dass der Algorithmus „immer“ terminiert. Es wäre prinzipiell möglich, dass in jeder Iteration des Algorithmus die zufällige Wahl der IDs ungünstig war und der Algorithmus somit nie terminiert. Dieses Ereignis ist jedoch höchst unwahrscheinlich – es hat Wahrscheinlichkeit 0. Formal betrachtet beweisen wir die Korrektheit unseres Algorithmus also dadurch, dass wir zeigen, dass er mit Wahrscheinlichkeit 1 terminiert.

Theorem 1.10. *Der randomisierte Leader Election Algorithmus terminiert mit Wahrscheinlichkeit* 1.

Beweis. Sei A_k das Ereignis, dass der Algorithmus nach (genau) k Iterationen terminiert (für $k \geq 1$). Dementsprechend ist $\bigcup_{k \in \mathbb{N}^{\geq 1}} A_k$ das Ereignis, dass der Algorithmus irgendwann terminiert. Da die Ereignisse A_k disjunkt sind (für $k_1 \neq k_2$ kann der Algorithmus nicht sowohl nach k_1 Runden als

auch nach k_2 Runden terminieren), gilt:

$$\Pr\left[\bigcup_{k\in\mathbb{N}^{\geq 1}} A_k\right] = \sum_{k\in\mathbb{N}^{\geq 1}} A_k = \sum_{k=1}^{\infty}(1-q)^{k-1}q = q\cdot\sum_{k=1}^{\infty}(1-q)^{k-1} = q\cdot\sum_{k=0}^{\infty}(1-q)^{k}$$
$$= q\cdot\frac{1}{1-(1-q)} = q\cdot\frac{1}{q} = 1\,.$$

Der Vereinfachung $\sum_{k=1}^{\infty}(1-q)^{k-1} = \frac{1}{1-(1-q)}$ folgt aus dem Grenzwert der geometrischen Reihe (siehe (A.15)). □

1.8.3 Laufzeit

Unsere bisherige Analyse hat ergeben, dass dieser Algorithmus irgendwann terminiert. Kommen wir nun zu einer genaueren Analyse der Laufzeit des Algorithmus. Zunächst wollen wir eine Aussage darüber treffen, welchen Erwartungswert die Anzahl der Runden des Algorithmus hat. Da jede Iteration des Algorithmus $O(n)$ Runden benötigt, reicht es aus den Erwartungswert der Anzahl an Iterationen des Algorithmus zu analysieren.

Theorem 1.11. *Die erwartete Anzahl an Iterationen des randomisierten Leader Election Algorithmus ist $O(1)$.*

Beweis. Für diese Analyse definieren wir zunächst folgende Zufallsvariablen Y_i (für $i \geq 1$):

$$Y_i = \begin{cases} 1 & \text{falls } i\text{-te Iteration des Algorithmus erfolgreich ist} \\ 0 & \text{ansonsten} \end{cases}$$

Außerdem definieren wir die Zufallsvariable Z als den kleinsten Index i, für den $Y_i = 1$ gilt, also $Z = \min\{i \geq 1 \mid Y_i = 1\}$. Somit entspricht Z der Anzahl an Iterationen des Algorithmus.

Wir wissen bereits, dass wir in jeder Iteration eine konstante Erfolgswahrscheinlichkeit von $q \geq \frac{1}{4}$ haben, also $\Pr[Y_i = 1] = q$ für alle $i \geq 1$. Da die Wahl der IDs in jeder Iteration separat erfolgt, sind die Y_i's stochastisch unabhängig und deshalb ist Z geometrisch verteilt mit Parameter q. Der Erwartungswert einer solchen Zufallsvariable ist $\mathrm{Ex}[Z] = \frac{1}{q} \leq 4$ (siehe Theorem B.10). Somit folgt, dass der randomisierten Leader Election Algorithmus in Erwartung höchstens $4 = O(1)$ Runden benötigt. □

Manchmal genügt uns jedoch eine Analyse des Erwartungswerts nicht, da eine große Varianz um diesen Erwartungswert herrschen könnte und es somit viele Ausreißer nach oben geben könnte. Obwohl man also in Erwartung eine konstante Anzahl an Iterationen hat, kann die tatsächliche Anzahl an Iterationen oft dennoch deutlich größer sein.

Daher untersuchen wir nicht nur wie viele Iterationen mit konstanter Wahrscheinlichkeit notwendig sind, sondern auch wie viele Iterationen „mit hoher Wahrscheinlichkeit" benötigt werden. Somit ist unser Maß an Erfolg keine konstante Wahrscheinlichkeit, sondern eine invers polynomielle Wahrscheinlichkeit.

Theorem 1.12. *Für jedes $c \geq 1$ gilt: Mit Wahrscheinlichkeit mindestens $1 - \frac{1}{n^c}$ („mit hoher Wahrscheinlichkeit") benötigt der randomisierte Leader Election Algorithmus $O(c \log n)$ Iterationen.*

Beweis. Sei Z die Zufallsvariable, die angibt, wie viele Iterationen der Algorithmus durchführt. Um das Theorem zu beweisen, zeigen wir, dass für $k = \lceil c \log_{4/3} n \rceil = \lceil \log_{4/3} n^c \rceil$ gilt: $\Pr[Z \leq k] \geq 1 - \frac{1}{n^c}$.

Die Zufallsvariable Z ist geometrisch verteilt, mit Parameter $q \geq \frac{1}{4}$. Die Wahrscheinlichkeit des Gegenereignisses $Z > k$ lässt sich daher folgendermaßen abschätzen:

$$\Pr[Z > k] = (1-q)^k \leq \left(1 - \frac{1}{4}\right)^k = \left(\frac{3}{4}\right)^k \leq \left(\frac{3}{4}\right)^{\log_{4/3} n^c} = \left(\frac{1}{4/3}\right)^{\log_{4/3} n^c} = \frac{1}{(4/3)^{\log_{4/3} n^c}} = \frac{1}{n^c}.$$

Somit erhalten wir $\Pr[Z \leq k] = 1 - \Pr[Z > k] \geq 1 - \frac{1}{n^c}$. □

1.8.4 Allgemeines Prinzip

In unserem randomisierten Leader Election Algorithmus lässt sich ein allgemeines Schema erkennen, um einen fehlerbehafteten randomisierten Algorithmus so einzusetzen, dass immer ein richtiges Ergebnis geliefert wird.

Theorem 1.13 (Monte Carlo → Las Vegas)**.**

- *Sei $\mathcal{A}$ ein randomisierter Algorithmus für ein Problem $\mathcal{P}$, der für jede Eingabe mit Wahrscheinlichkeit q eine korrekte Lösung berechnet.*
- *Sei $\mathcal{V}$ ein Verifizierer, der für jede Ausgabe von $\mathcal{A}$ ermitteln kann, ob die Lösung korrekt ist.*
- *Dann gibt es einen Algorithmus $\mathcal{B}$, der mit Wahrscheinlichkeit 1 terminiert und dabei eine korrekte Lösung berechnet und dafür $\mathcal{A}$ und $\mathcal{V}$ in Erwartung $O(1/q)$ Mal aufruft.*

Definition 1.14. *Ein randomisierter Algorithmus $\mathcal{B}$, dessen Ergebnis immer korrekt ist, heißt* Las Vegas *Algorithmus. Ein randomisierter Algorithmus $\mathcal{A}$, dessen Ergebnis wahrscheinlich korrekt ist, heißt* Monte Carlo *Algorithmus.*

Bei einem Monte Carlo Algorithmus wirkt sich die Randomisierung darauf aus, ob der Algorithmus am Ende ein korrektes Ergebnis liefert oder nicht. Bei einem Las Vegas Algorithmus wirkt sich die Randomisierung auf die verbrauchten Ressourcen (Anzahl an benötigten Runden, Nachrichtenkomplexität, Laufzeit, Speicher) aus und ein solcher Algorithmus liefert stets ein korrektes Ergebnis. Achtung: Man kann nicht jeden Monte Carlo Algorithmus in einen Las Vegas Algorithmus überführen, da es nicht immer trivial ist, einen Verifizierer zu finden.

1.9 Fazit

Bereits bei der einfachen Aufgabe, einen Leader im Ring zu bestimmen, werden viele Prinzipien und Techniken verteilter Algorithmen deutlich. Zunächst gibt es eine Vielzahl an Variationen in der Modellierung. Diese reichen von synchronen bzw. asynchronen, über anonyme und uniforme Modelle. Zusätzlich unterscheiden wir zwischen deterministischen und randomisierten Algorithmen. Ein weiterer wichtiger Punkt sind die Komplexitätsmaße: Hier unterteilen wir zwischen der benötigten Zeit und der Anzahl der gesendeten Nachrichten. Wir haben zwei grundlegende Möglichkeiten kennengelernt, um Symmetrien zu brechen: Zum einen über über eindeutige IDs und zum anderen über Randomisierung.

1.10 Literatur

[Ang80] Dana Angluin. “Local and Global Properties in Networks of Processors”. In: *Proc. of the Annual ACM Symposium on Theory of Computing (STOC)*. 1980, S. 82–93. DOI: 10.1145/800141.804655 (siehe S. 9).

[AW98] Hagit Attiya und Jennifer L. Welch. *Distributed Computing*. McGraw-Hill, 1998 (siehe S. 8, 13).

[CR79] Ernest J. H. Chang und Rosemary Roberts. “An Improved Algorithm for Decentralized Extrema-Finding in Circular Configurations of Processes”. In: *Communications of the ACM* 22.5 (1979), S. 281–283 (siehe S. 7).

[HS80] Daniel S. Hirschberg und James B. Sinclair. “Decentralized Extrema-Finding in Circular Configurations of Processors”. In: *Communications of the ACM* 23.11 (1980), S. 627–628. DOI: 10.1145/359024.359029 (siehe S. 11).

[IR90] Alon Itai und Michael Rodeh. “Symmetry Breaking in Distributed Networks”. In: *Information and Computation* 88.1 (1990), S. 60–87. DOI: 10.1016/0890-5401(90)90004-2 (siehe S. 13).

[Lan77] Gérard Le Lann. “Distributed Systems - Towards a Formal Approach”. In: *Proc. of the IFIP*. 1977, S. 155–160 (siehe S. 7).

Dieser Abschnitt basiert zum Teil auf Vorlesungseinheiten von Robert Elsässer und Stefan Schmid.

2 Einführung CONGEST Modell

Im CONGEST Modell wird das Netzwerk durch einen ungerichteten, zusammenhängenden Graph modelliert. Die Kommunikation, also das Versenden von Nachrichten zwischen den n Knoten entlang m Kanten, findet in synchronisierten Runden statt, wobei in jeder Runde jeder Knoten pro Nachbar maximal eine Nachricht versenden kann. Im CONGEST Modell ist die Nachrichtengröße (Bandbreite) auf $O(\log n)$ Bits limitiert Außerdem besitzt jeder Knoten im Graph eine eindeutige ID bestehend aus $O(\log n)$ Bits. Diese ID ist dem Knoten selbst und seinen Nachbarn bekannt. Zusätzlich ist jedem Knoten die Anzahl der Knoten n bekannt. Formal gilt ein Algorithmus gilt als terminiert, sobald alle Knoten in einen speziellen Terminierungszustand übergegangen sind, den sie nicht mehr verlassen und in dem sie keine Nachrichten mehr senden. Wir werden im Rahmen dieser Vorlesung aus Gründen der Übersichtlichkeit den Übergang eines Knotens in den Terminierungszustand meist nicht explizit angeben, da in der Regel offensichtlich ist, wann dieser stattfinden soll.[4]

2.1 Broadcast

Für den Broadcast-Algorithmus wird angenommen, dass es genau einen Startknoten s gibt, der eine Information I der Größe $O(\log n)$ an alle Knoten im Graph senden möchte. Der Algorithmus kann folgendermaßen formuliert werden:

---

1 s sendet in Runde 1 Information I an alle Nachbarn
2 Wenn ein Knoten $v \neq s$ das erste Mal Information I empfängt: v sendet I an alle Nachbarn

---

Im Algorithmus beginnt beginnt der Startknoten s. Dieser sendet I an alle unmittelbaren Nachbarn, mit denen er verbunden ist. Diese senden I wiederum an ihre Nachbarn, bis I auch in den von s am weitesten entfernten Knoten angekommen ist. Die Knoten, die die Informationen bereits empfangen haben, ignorieren gegebenenfalls den erneuten Empfang der Information. Ein Beispiel für die Durchführung einer Runde des Algorithmus ist in Abb. 2.1 gegeben.

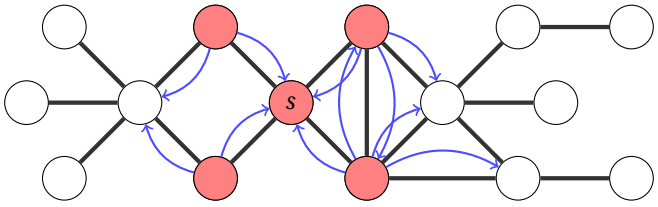


Abbildung 2.1: In diesem Beispiel wird die zweite Runde des Broadcast-Algorithmus auf dem gegebenen Netzwerk ausgeführt.

2.1.1 Korrektheit

Die für die Korrektheit zu zeigende Induktionsaussage lautet:

Für jeden Knoten $v \neq s$ gilt: Knoten v empfängt spätestens in Runde $\mathrm{dist}(s, v) + 1$ die Information I.

[4] Wenn wir beispielsweise bewiesen haben, dass ein Algorithmus nach $R(n)$ Runden das korrekte Ergebnis berechnet hat, kann jeder Knoten nach $R(n)$ Runden in den Terminierungszustand wechseln.

Die Runde, in der v spätestens Information empfängt, hängt also von der Distanz des Knotens zum Startknoten s ab.

Bemerkung: Die Distanz von s nach v ist die Länge des kürzesten Wegs von s nach v. In der Runde $r+1$ erhalten also alle Knoten, die r Kanten vom Startknoten s entfernt sind, die Information I.

Aus der Induktionsaussage folgt die Korrektheit des Algorithmus: Da das Netzwerk zusammenhängend ist, hat jeder Knoten endliche Distanz zu s. Das bedeutet, dass jeder Knoten nach einer endlichen Rundenzahl I erhält.

Um die Induktionsaussage zu beweisen, müssen wir diese für $v \neq s$ beweisen. Dabei gilt für Knoten $v \neq s$, dass $\mathrm{dist}(s,v) \geq 1$.

Induktionsbasis:

Es gilt $\mathrm{dist}(s,v) = 1$. Ein Knoten v haben, der in Distanz zu 1 zu s liegt, ist ein Nachbar von s. Nachbarn von s empfangen die Information in Runde $2 = \mathrm{dist}(s,v) + 1$.

Induktionsschritt:

Wir betrachten den Fall $\mathrm{dist}(s,v) \geq 2$. Sei u der Vorgängerknoten von v auf dem kürzestem Weg von s nach v. Es gilt für diese Knoten: $\mathrm{dist}(s,u) = \mathrm{dist}(s,v) - 1$. Somit ist die Distanz von s noch u echt kleiner ist als die Distanz von s nach v. Wir können daher für u die Induktionsaussage anwenden.

Nach Induktionsaussage hat u die Information I spätestens in Runde $\mathrm{dist}(s,u) + 1$ empfangen und dann an alle Nachbarn gesendet. Da v ein Nachbar von u ist, empfängt v die Information I spätestens in Runde $\mathrm{dist}(s,u) + 2 = \mathrm{dist}(s,v) + 1$.

2.1.2 Laufzeit

Der Algorithmus läuft so lange – hat genau so viele Runden – bis jeder Knoten die Information erhalten hat. Es muss also der am weitesten von s entfernte Knoten, nennen wir ihn v, erreicht werden. Jeder Knoten empfängt die Information spätestens nach $\mathrm{dist}(s,v) + 1$ vielen Runden. Also gilt für die maximal benötigte Anzahl an Runden: $r_{\max} = \max_v \mathrm{dist}(s,v) + 1$.

Die Exzentrizität eines Knotens u ist definiert als $\mathrm{Ecc}(u) = \max_v \mathrm{dist}(u,v)$, also die Anzahl der Kanten des längsten kürzesten Weges, der von u ausgeht. Der Durchmesser des Netzwerks ist die maximale Distanz zwischen einem Paar von Knoten im Netzwerk, also $D = \max_{u,v \in V} \mathrm{dist}(u,v)$. Er gibt also an, wie lange der kürzeste Weg ist, den es im Netzwerk gibt.

Zusammenfassend erhalten wir mit diesen beiden Definitionen $D = \max_u \mathrm{Ecc}(u)$, und daher $\mathrm{Ecc}(u) \leq D$. Der Durchmesser ist also gleich maximalen Exzentrizität irgendeines Knotens, und daher folgt, dass die Exzentrizität jedes Knotens immer kleiner gleich dem Durchmesser des Netzwerks ist. Somit beträgt die Laufzeit des Broadcast-Algorithmus $O(\mathrm{Ecc}(s)) = O(D) = O(n)$ viele Runden.

2.1.3 Nachrichtenkomplexität

Im Algorithmus wird die Information I nach erstmaligem Erhalt an alle Nachbarn weitergesendet wird. Das heißt, dass jeder Knoten genau so oft I versendet, wie er anliegende Kanten zu seinen Nachbarn hat. Dies entspricht dem sogenannten Grad des Knotens v – also $\deg(v)$. Insgesamt werden daher: $\sum_{v \in V} \deg(v) = 2m = O(m)$ viele Nachrichten verschickt; die obere Schranke ergibt sich aus der Tatsache dass die Summe der Knotengrade gleich $2m$ ist.

2.2 Spannbaum durch Breitensuche

Das Ziel ist es in einem beliebigen zusammenhängenden Graph einen Baum zu berechnen, der alle Knoten enthält. Insbesondere müssen also Eltern-Kind Beziehungen hergestellt werden ohne dass diese einen Kreis bilden. Wir erreichen unser Ziel durch die Berechnung eines Breitensuchbaums, für die eine beliebige Information I vom Startknoten s aus versendet:

---

1 s sendet in Runde 1 Information I an alle Nachbarn
2 Wenn ein Knoten $v \neq s$ das erste Mal Information I empfängt:
3 v speichert einen der Sender von I als Elternknoten
4 v sendet „Kind“-Nachricht an Elternknoten und I an alle anderen Nachbarn

---

Ein Beispiel für die Durchführung einer Runde des Algorithmus ist in Abb. 6.1 gegeben. Um für Determinismus zu sorgen kann beispielsweise in Zeile 2 der Knoten, mit der kleineren ID, bevorzugt als Elternknoten gespeichert werden. Nach dem vollständigen Durchlauf des Algorithmus ist ein impliziter Spannbaum entstanden, da jeder Knoten seinen Elternknoten und seine Kinder kennt. Dadurch ist kein Kreis entstanden, da sich alle Knoten (außer s) im Graph für genau einen Elternknoten mit geringerer Distanz zu s entschieden haben.

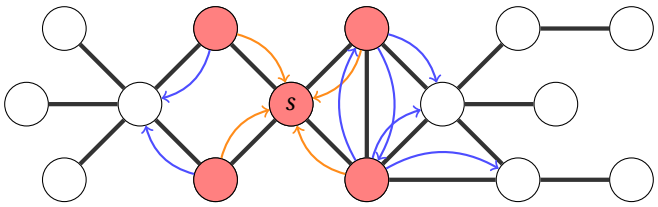


Abbildung 2.2: In diesem Beispiel wird die zweite Runde des Algorithmus zur Berechnung eines Breitensuchbaums auf dem gegebenen Netzwerk ausgeführt.

2.3 Distanz-Berechnung

Zur Distanz-Berechnung wird der Algorithmus zur Berechnung eines Breitensuchbaums etwas modifiziert bzw. erweitert. Anstelle der Kind-Nachricht senden die Elternknoten ihren Kindern die Distanz, welche sie sich zuweisen sollen.

---

1 s hat Distanz 0 zu sich selbst und sendet in Runde 1 Distanz-Information 1 an alle Nachbarn
2 Wenn ein Knoten $v \neq s$ das erste Mal eine Distanz-Information d empfängt:
3 v speichert einen der Sender von d als Elternknoten
4 v sendet „Kind“-Nachricht an Elternknoten und Distanz-Information $d+1$ an alle anderen Nachbarn

---

Durch diesen Algorithmus erhält jeder Knoten im Graph einen Elternknoten (außer s) und von genau diesem auch eine Distanz zugeordnet. Falls ein Knoten, dem bereits eine Distanz zugeordnet wurde, eine weitere, höhere empfängt, so ignoriert er diese. Ein Beispiel für die Durchführung einer Runde des Algorithmus ist in Abb. 2.3 gegeben.

2.3.1 Zusammenfassung Breitensuche

Für eine Breitensuche mit Startknoten s gelten folgende Eigenschaften:

- *Anzahl der Runden:* $O(\mathrm{Ecc}(s)) = O(D)$

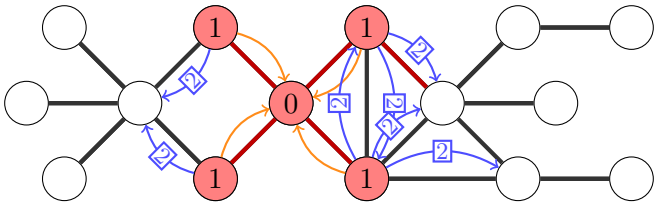


Abbildung 2.3: In diesem Beispiel wird die zweite Runde des Algorithmus zur Distanzberechnung auf dem gegebenen Netzwerk ausgeführt.

Der Algorithmus terminiert, sobald alle Knoten einen Elternknoten zugewiesen bekommen haben und alle Kind-Nachrichten bei den Elternknoten eingegangen sind. Die maximale Entfernung eines Blattes von s im Breitensuchbaum von s ist $\mathrm{Ecc}(s)$. Daher ist die Laufzeit $O(\mathrm{Ecc}(s)) = O(D)$.

- *Anzahl der Nachrichten:* $O(m)$

  Über jede Kante werden maximal zweimal eine Nachricht gesendet, also insgesamt höchstens $2m$ Nachrichten, was in $O(m)$ ist.

- *Jeder Knoten v kennt $\mathrm{dist}(s, v)$, seine Distanz zu s:*

  Dies ist das Ergebnis des verteilten Algorithmus zur Distanz-Berechnung.

- *Für den Breitensuchbaum gilt:*

  - Jeder Knoten $u \neq s$ hat einen Elternknoten v mit $\mathrm{dist}(s, v) = \mathrm{dist}(s, u) - 1$
  - Jeder Knoten kennt seinen Elternknoten und seine Kinder im Baum
  - Jeder Breitensuchbaum ist auch ein Spannbaum

2.4 Broadcast durch Upcast und Downcast

Wir betrachten folgende Ausgangssituation: Es wurde bereits ein Breitensuchbaum, ausgehend von einem Startknoten s, berechnet und ein Knoten v hat eine Information I der Größe $O(\log n)$. Das Ziel ist, allen Knoten im Graph die Informationen I zukommen zu lassen. Dies kann mit einer Kombination aus Upcast und Downcast durchgeführt werden:

```
1 Upcast: I wird über Elternknoten zu s gesendet
2 Downcast: I wird von s aus über Kinder zu allen Knoten gesendet
```

Beim Upcast sendet der Knoten v die Information I an seinen Elternknoten und dieser wieder an seinen Elternknoten usw. bis die Information den Startknoten s erreicht hat. Von dort aus wird der Downcast gestartet, indem jeder Knoten I an jedes Kind weiterleitet, bis alle Blätter erreicht wurden. Ein Beispiel für die Upcast-Phase ist in Abb. 2.4 gegeben und für die Downcast-Phase in Abb. 2.5.

Die Anzahl der Runden ist $O(\mathrm{Ecc}(s)) = O(D)$, da sowohl für den Upcast also auch für den Downcast die maximal zurückgelegte Distanz einer weitergeleiteten Nachricht $O(\mathrm{Ecc}(s))$ ist. Die obere Schranke für die Anzahl der Nachrichten beträgt $O(n)$. Im Gegensatz zum Broadcast, wo

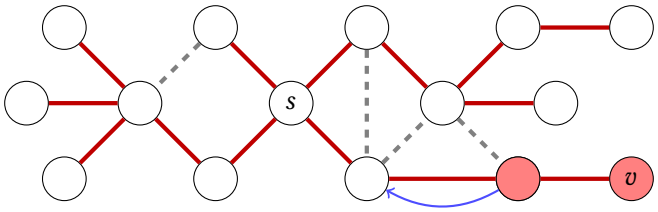


Abbildung 2.4: In diesem Beispiel wird die zweite Runde der Upcast-Phase auf dem gegebenen Netzwerk ausgeführt.

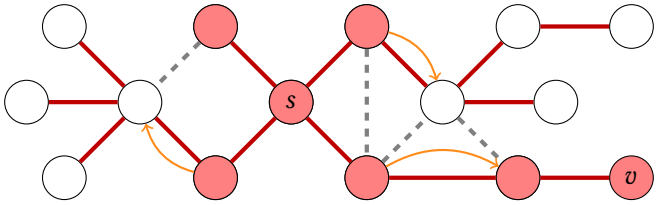


Abbildung 2.5: In diesem Beispiel wird die zweite Runde der Downcast-Phase auf dem gegebenen Netzwerk ausgeführt.

über jede Kante im Ursprungsgraph eine bis zwei Nachrichten gesendet werden, liegt beim Up- bzw. Downcast bereits ein Baum vor, für den folgende Eigenschaft gilt:

$$\text{Anzahl der Knoten} = \text{Anzahl der Kanten} - 1$$

Daher beträgt die Anzahl der Nachrichten $O(n)$. Wenn mehrere solche Broadcasts hintereinander durchgeführt werden sollen, ist es also nachrichteneffizienter, Upcasts und Downcasts im Breitensuchbaum durchzuführen als den Broadcast-Algorithmus aus Abschnitt 2.1 jedes Mal aufs Neue durchzuführen.

2.5 Pipelining

2.5.1 Multipler Downcast

Wir betrachten folgende Ausgangssituation: Der Knoten s hat k Informationen $I_1, ..., I_k$ jeweils der Größe $O(\log n)$ und ein Breitensuchbaum, ausgehend von s, wurde bereits berechnet. Das Ziel ist, allen Knoten im Graph die Informationen $I_1, ..., I_k$ zukommen zu lassen. Die Schwierigkeit dabei ist, dass wir die k Informationen nicht in einer einzigen Nachricht zusammenfassen können, da im CONGEST Modell die maximale Nachrichtengröße $O(\log n)$ ist. Ein naiver Ansatz führt einfach k-mal den regulären Broadcast-Algorithmus aus Abschnitt 2.1 durch, was zur Laufzeit $O(kD)$ führen würde. Allerdings lässt sich diese Laufzeit durch den folgenden Pipeline-Algorithmus verringern:

1 s sendet in Runde $j \leq k$ Information I_j an alle Nachbarn
2 Wenn ein Knoten $v \neq s$ eine Information I das erste Mal empfängt: v sendet I an alle Kinder

In diesem Algorithmus sendet der Startknoten s in Runde j die Information I_j aus, bis $j = k$ gilt und alle Informationen versendet wurden. Alle anderen Knoten senden jegliche Information, die sie das erste Mal empfangen haben, sofort an ihre Kinder weiter. Dadurch werden die vorhandenen Kanten besser ausgenutzt und die Informationen I_1 bis I_k erreichen früher alle Knoten als bei k sequentiellen Broadcasts.

Ein Beispiel für den Pipelining-Algorithmus ist in Abb. 2.6 gegeben: Nach 4 Runden hat Knoten v mit $\text{dist}(s, v) = 3$ die (erste) Information I_1 erhalten. (Eine Information gilt nach Erhalt erst in der

nächsten Runde als empfangen.) In der darauffolgenden Runde erhält er I_2 usw. D. h. in der Runde $3 + j$ erhält er die letzte Information I_j.

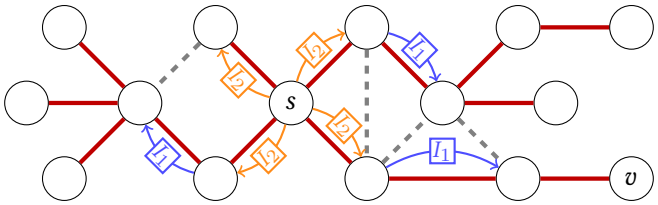


Abbildung 2.6: In diesem Beispiel wird die zweite Runde des Pipelining-Algorithmus auf dem gegebenen Netzwerk ausgeführt.

Die Induktionsaussage für den Korrektheitsbeweis lautet:

> Für jedes $1 \leq j \leq k$ und jeden Knoten $v \neq s$ gilt: Knoten v empfängt spätestens in Runde $\mathrm{dist}(s, v) + j$ die Information I_j.

Aus dieser Induktionsaussage folgt, dass die Laufzeit für den multiplen Downcast durch Pipelining $O(\mathrm{Ecc}(s) + k) = O(D + k)$ beträgt. Grob gesprochen, werden $O(D)$ Runden benötigt, um den am weitesten entfernten Knoten v (von s aus) zu erreichen und noch einmal k Runden, bis v auch die letzte Information I_k erhalten hat.

2.5.2 Convergecast

Betrachten wir nun die Situation, dass der Knoten s sicherstellen möchte, dass der von ihm initiierte Broadcast bzw. die Breitensuche auch tatsächlich abgeschlossen ist. Zwar wissen wir, dass dies nach $O(D)$ Runden der Fall ist, allerdings ist der tatsächliche Durchmesser D initial nicht bekannt. Der Knoten s hat daher keine einfache Möglichkeit zu wissen, nach wie vielen Runden die Berechnung abgeschlossen ist. Die Lösungsidee beinhaltet das Versenden von ACK Nachrichten (vergleichbar mit TCP) von jedem Knoten an s per Upcast. Eine naive Herangehensweise wäre folgende:

```
1 Führe Breitensuche mit Spannbaumberechnung von Startknoten s aus
2 Jedes Blatt: sende ACK-Nachricht an s per Upcast im Baum
```

Jedoch tritt hier das Problem auf, dass es an den Elternknoten trotz Pipelining zu hoher Congestion kommen kann. Im schlechtesten Fall beträgt die Laufzeit daher $\Theta(n)$, siehe Abb. .

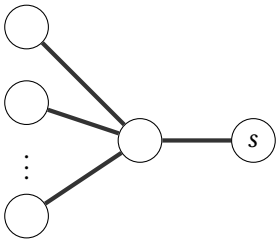


Abbildung 2.7: Beispiel für das Auftreten hoher Congestion bei naivem Convergecast

Eine Lösung des Congestion-Problems ist die Aggregation der ACK-Nachrichten, wie folgender Algorithmus beschreibt.

Führe Breitensuche mit Spannbaumberechnung von Startknoten s aus durch
Sobald Knoten ACK von allen Kindern erhalten hat: Sende ACK an Elternknoten
Blätter im Baum erkennen sich selbst als solche, wenn sie keine Kind-Nachricht erhalten
s wartet bis ACK von allen Kindern erhalten

Ein Beispiel für diesen Aggregationsalgorithmus ist in Abb. 2.8 gegeben:

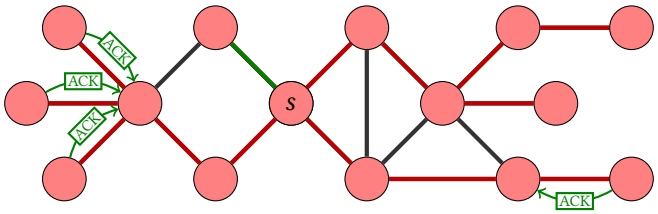


Abbildung 2.8: In diesem Beispiel wird der Aggregationsalgorithmus auf dem gegebenen Netzwerk ausgeführt.

Aus funktionaler Sicht ist das ACK-Bit eines Knotens die Und-Verknüpfung der ACK-Bits seiner Kinder. Es müssen also erst alle ACK-Nachrichten der Kinder eingegangen sein, damit die eigene ACK-Nachricht gebildet und an den Elternknoten abgeschickt werden kann. Durch die Baumstruktur ist die Bottom-Up Berechnung und das Warten auf die Funktionswerte der Kinder möglich. Die zusätzlichen ACK-Nachrichten führen maximal zu einer Verdopplung der Zeit- und Nachrichtenkomplexität. Weitere Aggregatfunktionen, bei denen diese Technik funktioniert, sind beispielsweise Summe, Minimum und Maximum.

2.6 Queuing

2.6.1 Multipler Upcast

Wir betrachten folgende Ausgangssituation: Es wurde bereits ein Breitensuchbaum, ausgehend von einem Startknoten s, berechnet und mehrere Knoten des Netzwerks haben insgesamt k Informationen $I_1, ..., I_k$, jeweils der Größe $O(\log n)$. Das Ziel ist, dem Knoten s alle Informationen $I_1, ..., I_k$ zukommen zu lassen. Dabei können die Informationen im Netzwerk verteilt sein. Im Allgemeinen ist es nicht der Fall, dass ein Knoten alle k Informationen hat. Wir führen diese Art des multiplen Upcasts mit folgendem Algorithmus durch:

Jeder Knoten hat eine Queue noch nicht weitergeleiteter Informationen
In jeder Runde, sendet jeder Knoten einer der Informationen aus seiner Queue an seinen Elternknoten

Im Algorithmus besteht am Anfang die Queue jedes Knotens aus den Informationen, die er bereits initial kennt. In jeder Runde sendet jeder Knoten eine dieser Informationen aus seiner Queue an seinen Elternknoten. Wenn der Knoten dieser Information aus der Queue an der Elternknoten gesendet hat, dann wandert die Information sofort in die Queue des Elternknotens. Das geht so weiter bis alle Queues leer sind und s alle Informationen erhalten hat. Ein Beispiel für den Queuing-Algorithmus ist in Abb. 2.9 gegeben:

2.6.2 Laufzeitschranke mit Hilfslemma

Um eine Laufzeitschranke für den Algorithmus zu beweisen, benötigen wir zunächst folgendes Hilfslemma.

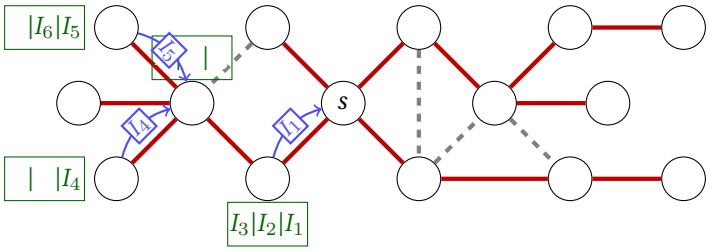


Abbildung 2.9: In diesem Beispiel wird der Queuing-Algorithmus auf dem gegebenen Netzwerk ausgeführt.

Lemma 2.1. *Für jedes $r \geq 1$ und jeden Knoten v mit $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1$ gilt: Wenn v keine Information in Runde r empfängt, dann auch nicht in Runde $r+1$.*

Dieses Lemma besagt, dass – ab einer genügend großen Rundenzahl – keine Unterbrechungen im Informationsfluss mehr auftreten können. Es genügt also, eine Schranke für den erstmaligen (und damit endgültigen) Abbruch des Informationsflusses nach diesem Zeitpunkt zu zeigen.

Theorem 2.2. *Der Algorithmus für den multiplen Upcast benötigt $O(D+k)$ Runden.*

Beweis. Wir betrachten den Knoten s und die Runde $r = \mathrm{Ecc}(s) + 1$. Trivialerweise ist die Distanz von s zu sich selbst 0, also gilt $\mathrm{dist}(s,s) = 0$. Weil wir r als $r = \mathrm{Ecc}(s) + 1$ gewählt haben, gilt $\mathrm{dist}(s,s) = 0 = \mathrm{Ecc}(s) - r + 1$. Die Ungleichung $\mathrm{dist}(s,s) \geq \mathrm{Ecc}(s) - r' + 1$ gilt somit für alle $r' \geq r$. Wir wenden als nächstes Lemma 2.1 auf jedes $r' \geq r$ an und erhalten folgende Aussage: Sobald s in einer Runde $r' \geq r$ das erste Mal keine Nachricht empfangen hat, empfängt s überhaupt keine Nachrichten mehr. Anders ausgedrückt: Nach Runde r sind keine Unterbrechungen des Informationsflusses mehr möglich. Wenn s nie mehr Informationen empfängt, dann terminiert der Algorithmus.

Nun gibt es insgesamt k Informationen im Netzwerk. Jede der k Informationen wird von jedem Knoten (und damit auch von Knoten s) höchstens einmal empfangen. Das heißt, ab der Runde r können nur noch höchstens k Runden auftreten, in denen eine Information empfangen wird. Somit ist der Gesamtzahl an Runden bis zur Terminierung höchstens $\mathrm{Ecc}(s) + k = O(D+k)$. □

Beweis von Lemma 2.1. Wir beweisen nun das Hilfslemma mittels Induktion.

Induktionsbasis: $r = 1$

Sei v ein Knoten für den $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1 = \mathrm{Ecc}(s)$ gilt. Weil $r = 1$ ist, muss gelten, dass $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s)$ ist. Da $\mathrm{Ecc}(s)$ die höchstmögliche Distanz von s aus ist, gilt außerdem $\mathrm{dist}(s,v) = \mathrm{Ecc}(s)$. Die einzigen Knoten in Distanz $\mathrm{Ecc}(s)$ zu s sind die Blätter im Breitensuchbaum von s. Da Blätter keine Informationen empfangen, gilt das Lemma für $r = 1$.

Induktionsschritt: $r - 1 \to r$

Im Induktionsschritt gilt $r \geq 2$. Wir betrachten einen Knoten v, für den die Ungleichung $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1$ gilt. Das Lemma gilt trivialerweise, falls v ein Blatt ist. Falls v kein Blatt ist, dann sei u ein beliebiges Kind von v im Breitensuchbaum. Wir nehmen an, dass v in Runde r keine Information empfängt. Aber wenn v keine Information empfängt, dann hat u in Runde $r-1$ keine Information an v gesendet. Warum hat u in Runde $r-1$ keine Information an v gesendet? Das kann nur der Fall sein, wenn u in Runde $r-1$ eine leere Queue hatte und keine Information empfangen hat. Wir

wissen weiters, weil u ein Kind von v im Breitensuchbaum ist, dass $\mathrm{dist}(s,u) = \mathrm{dist}(s,v) + 1$ gilt. Jetzt setzen wir für $\mathrm{dist}(s,v)$ die obige Ungleichung ein und erhalten:

$$\mathrm{dist}(s,u) = \mathrm{dist}(s,v) + 1 \geq \mathrm{Ecc}(s) - r + 1 + 1 = \mathrm{Ecc}(s) - (r-1) + 1\,.$$

Nun wissen wir, dass für $r-1$ die Induktionsaussage gilt und dürfen sie aufgrund der oben gezeigten Ungleichung $\mathrm{dist}(s,u) \geq \mathrm{Ecc}(s) - (r-1) + 1$ auf den Knoten u anwenden: Da u, wie oben argumentiert, in Runde $r-1$ keine Information empfangen hat, empfängt u auch in Runde $(r-1)+1 = r$ keine Information. Da ebenfalls die Queue von u in Runde r immer noch leer ist, sendet u in Runde r keine Information an v. Da u ein beliebiges Kind von v im Breitensuchbaum war, gilt diese Aussage für alle Kinder von v. Da keines seiner Kinder in Runde r eine Information an v sendet, empfängt v keine Information in Runde $r+1$. □

2.7 Literatur

[Pel00] David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM, 2000.

Dieser Abschnitt basiert zum Teil auf einer Vorlesungseinheit von Danupon Nanongkai.

3 Synchronisierung

Die Annahme, dass es in verteilten Systemen synchrone Runden gibt, ist idealisiert, da in realen Anwendung die Übermittlung von Nachrichten nicht unbedingt in gleichbleibender Zeit geschieht. Im CONGEST Modell wird diese Eigenschaft vernachlässigt, da jegliche Kommunikation synchron und ohne Verzögerung abläuft. Das Ziel ist die synchrone Algorithmen auf einem asynchronen Netzwerk mit moderaten Overheads in der Zeit- und Nachrichtenkomplexität zu simulieren.

3.1 Asynchrones Modell

Der größte Unterschied zum synchronen Modell ist der Wegfall der Runden, also den diskreten Zeitschritten. Stattdessen gibt es (wie in Abschnitt 1.2 erläutert) zwei verschiedene Events:

- Die Initialisierung eines Knotens von außerhalb des Netzwerks.
- Der Empfang einer Nachricht an einem Knoten von einem anderen Knoten.

Jedes der beiden Events aktiviert den entsprechenden Knoten und dieser kann dann interne Berechnungen durchführen um bisher empfangene Nachrichten zu verarbeiten und anschließend an jeden seiner Nachbarn eine Nachricht senden. Dabei ist die Dauer der Nachrichtenübertragung endlich, kann sich aber von Nachricht zu Nachricht unterscheiden. Für Laufzeitanalysen wird in der Regel davon ausgegangen, dass jede Nachrichtenübertragung maximal eine Zeiteinheit benötigt. Die Korrektheit eines Algorithmus muss aber für beliebige, endliche Übertragungszeiten gelten.

Bei der Ausführung synchroner Algorithmen in asynchronen Netzwerk ohne weitere Anpassungen kann es zu Inkonsistenzen kommen, wenn sich die Übertragungsdauer für die einzelnen Nachrichten unterscheidet. Ein Beispiel für die naive Ausführung des Breitensuche-Algorithmus aus Abschnitt 2.2 ist in Abb. 3.1 zu sehen. Dem grau markiertem Knoten wird die falsche Distanz zugewiesen: Die Verzögerung beim Senden der Nachricht vom Startknoten an seinen Nachbarn unten rechts groß, dass ein anderer Knoten mit Distanznachricht 2 diesen Knoten früher erreicht. Wir könnten den Algorithmus nun dahingehend anpassen, dass der Knoten beim Erhalt einer kleineren Distanz erneut eine Distanznachricht an seine Nachbarn sendet. Jedoch gelten dann die vorhandenen Garantien für Laufzeit und Nachrichtenkomplexität nicht mehr.

3.2 Synchronizer-Framework

Synchronizer sind Verfahren, die das Simulieren von synchronen Algorithmen in asynchronen Netzwerken ermöglichen.

3.2.1 Pulsgeber

Wir betrachten im Folgenden Synchronizer basierend auf Pulsgeber-Algorithmen, die lokal für jeden Knoten einen Puls generieren. Dieser Puls ist jedoch *nicht* synchron für verschiedene Knoten. Der Puls kann analog zu einer Runde im synchronen Modell betrachtet werden. Der Knoten führt also, nachdem für ihn der i-te Puls generiert wurde, die Runde i des zu simulierenden synchronen Algorithmus aus (Empfangen, Verarbeiten und Senden von Nachrichten). Eine Simulation ist dann korrekt, wenn Puls $i + 1$ immer erst dann generiert wird, wenn Nachrichten von allen Nachbarn für Puls i empfangen wurde. Die Schwierigkeit hierbei ist jedoch, dass der Knoten nicht unbedingt weiß von welchem Nachbarn er Nachrichten erhalten müsste.

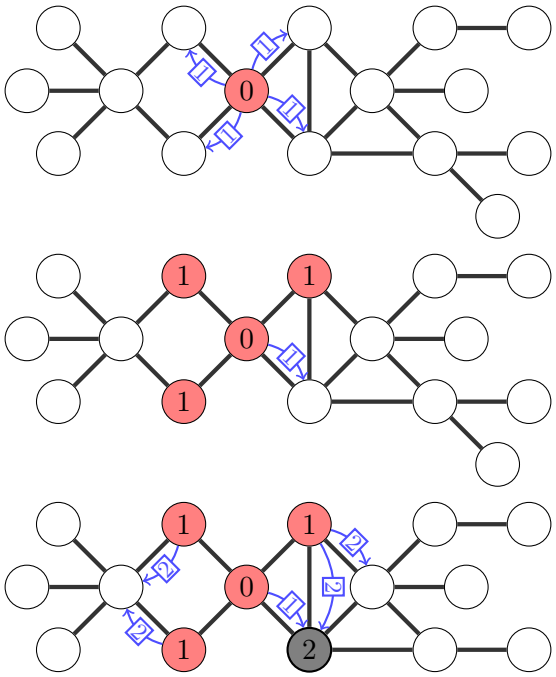


Abbildung 3.1: Beispiel für naive Ausführung des Breitensuche-Algorithmus im asynchronen Modell

3.2.2 Safety

Die Safety beschreibt den Zustand eines Knotens für den Puls i. Intuitiv gesprochen gilt ein Knoten dann als safe, wenn er die Simulation (Empfangen, Verarbeiten und Senden von Nachrichten) der aktuellen Runde i bereits abgeschlossen hat.

Definition 3.1. *Ein Knoten ist* safe *für Puls i, wenn alle seine in der Simulation von Runde i gesendeten Nachrichten die jeweiligen Nachbarn erreicht haben.*

Die Safety kann durch das Versenden von Bestätigungsnachrichten nach dem Empfangen jeder Nachricht überprüft werden. Jeder Empfänger einer Nachricht verschickt also eine Bestätigung an den Sender. Dadurch verdoppelt sich die Nachrichten- und Zeitkomplexität, da für jede gesendet Nachricht zusätzlich eine Bestätigung gesendet werden muss.

Die grundlegende Beobachtung ist nun, dass ein synchroner Algorithmus korrekt simuliert wird, wenn für jeden Knoten der Puls $i + 1$ immer erst dann generiert wird, wenn der Knoten selbst und alle seine Nachbarn safe für Puls i sind. Dies hat folgenden Grund: Wenn ein Knoten selbst und alle seine Nachbarn safe für Puls i sind, dann hat der Knoten jede Information, die für die Runde $i + 1$ benötigt wird, bereits erhalten, da er nur Nachrichten von seinen Nachbarn erhalten kann und diese keine Nachrichten mehr schicken werden, da sie bereits safe sind. Für die Überprüfung der Safety können verschiedene Strategien eingesetzt werden [Awe85].

3.3 Synchronizer α

Beim Synchronizer α entscheidet jeder Knoten, selbst ob er safe ist und in den nächsten Puls übergehen kann. Der Algorithmus funktioniert folgendermaßen:

- Sobald ein Knoten safe ist, also alle Bestätigungsnachrichten für seine gesendeten Nachrichten empfangen hat, dann sendet er eine safe-Nachricht an alle seine Nachbarn.
- Der Knoten erkennen dadurch sofort, wenn alle Nachbarn safe sind.
- Sobald ein Knoten safe ist und von allen Nachbarn eine safe-Nachricht empfangen hat, kann er mit dem nächsten Puls fortsetzen.

Synchronizer α liefert die folgenden Garantien.

Theorem 3.2. *Jeder synchrone Algorithmus, der R Runden benötigt und dabei M Nachrichten versendet, kann in asynchronen Netzwerken in Zeit $O(R)$ mit $O(M + R \cdot m)$ Nachrichten simuliert werden.*

Der größte Vorteil diese Algorithmus ist, dass er rein lokal auf jedem Knoten funktioniert. Durch das Verschicken der safe-Nachricht entsteht nur ein konstanter Overhead in der Zeitkomplexität. Die Nachrichtenkomplexität wird jedoch unter Umständen stark erhöht, da jeder Knoten zusätzlich in jedem Puls an alle Nachbarn eine safe-Nachricht schicken muss, was einen großen Nachteil dieses Algorithmus darstellt.

3.4 Synchronizer β

Beim Synchronizer β wir der Puls von einem globalen Leader generiert. Der Algorithmus funktioniert folgendermaßen:

- Ausgangssituation: Im Netzwerk wurde ein Breitensuchbaum berechnet, an dessen Wurzel sich der Leader des Netzwerks befindet.
- Der Leader sendet jeden Puls i mittels Downcast an alle Knoten im Baum.
- Jeder Knoten führt beim Erhalt des Pulses i die Runde i aus.
- Jeder Knoten gilt als safe, sobald er alle Empfangsbestätigungen erhalten hat.
- Der Leader erfährt durch aggregierten Upcast im Baum, wann alle Knoten safe sind.
- Sobald alle Knoten safe sind, generiert der Leader Puls $i + 1$.

Synchronizer β liefert die folgenden Garantien.

Theorem 3.3. *Jeder synchrone Algorithmus, der R Runden benötigt und dabei M Nachrichten versendet, kann in asynchronen Netzwerken in Zeit $O(R \cdot D)$ mit $O(M + R \cdot n)$ Nachrichten simuliert werden, wenn bereits ein Leader und ein Breitensuchbaum berechnet wurden.*

Im Gegensatz zum Synchronizer α hat der Synchronizer β eine geringere Nachrichtenkomplexität, da für jede Kante im Baum (maximal n Kanten) genau zwei zusätzliche Nachrichten verschickt werden. Der große Nachteil ist jedoch, dass sich die Laufzeit um einen Faktor $\Theta(D)$ erhöht, da der Upcast und der Downcast jeweils eine Laufzeit von $\Theta(D)$ haben. Gegenüber Synchronizer α hat Synchronizer β also vor allem bei kleinem Durchmesser des Netzwerks Vorteile. Ein zusätzlicher Nachteil von Synchronizer β ist, dass ein Breitensuchbaum und ein Leader im asynchronen Netzwerk bereits bestimmt werden mussten.

Der benötigte Breitensuchbaum kann durch verschiedene Algorithmen berechnet werden. Eine mögliche Variante wäre den *synchronen* Breitensuche-Algorithmus mit Synchronizer α zu simulieren. Der synchrone Algorithmus benötigt $O(D)$ Runden und hat eine Nachrichtenkomplexität von $O(m)$; somit kann in einem asynchronen Netzwerk ein Breitensuchbaum in $O(D)$ Zeiteinheiten mit Nachrichtenkomplexität $O(mD)$ berechnet werden. Es ist aber auch möglich direkt einen Algorithmus für asynchrone Netzwerke zu entwickeln, der $O(D^2)$ Zeiteinheiten benötigt und Nachrichtenkomplexität $O(m + nD)$ hat.

3.5 Synchronizer γ

Ziel des hybriden Synchronizers γ ist, die Vorteile der Synchronizer α und β zu vereinen. Dieser Synchronizer arbeitet mit einer speziellen Zerlegung des Netzwerks.

Definition 3.4. *Eine (ρ, μ)-Partition ist eine Zerlegung der Knoten V in nicht-leere, disjunkte Teilmengen $P_1, \ldots, P_\ell$ mit folgenden Eigenschaften:*

1. *Jede Teilmenge P_i hat ein designiertes Zentrum und die Knoten in P_i sind einem Baum mit Entfernung höchstens ρ vom Zentrum organisiert.*
2. *Es wurde eine Menge F von höchstens μ Kanten ausgewählt so dass*
   (a) *jede Kante in F benachbarte Teilmengen P_i und P_j verbindet und*
   (b) *für jedes Paar benachbarter Teilmengen mindestens eine Kante in F enthalten ist.*

Idealerweise würde F für jedes Paar benachbarter Teilmengen (d.h. Teilmengen zwischen denen eine Kante existiert) genau eine Kante enthalten, die diese Teilmengen verbindet. Der Aufwand, diese „minimale“ Teilmenge zu berechnen, ist aber für unseren Anwendungsfall nicht notwendig. Daher erlauben wir auch prinzipiell eine nicht-minimale Teilmenge F solange die Maximalgröße von F durch μ beschränkt ist. Abb. 3.2 zeigt ein Beispiel für eine (ρ, μ)-Partition Wie in verteilten Algorithmen üblich, wird es ausreichen, dass den Knoten nur lokale Information über die (ρ, μ)-Partition bekannt ist, insbesondere muss jeder Knoten wissen, ob er ein Zentrum ist oder nicht, seine Kinder und evtl. den Elternknoten im Baum kennen und für jede seiner inzidenten Kanten wissen, ob sie in der Menge F enthalten ist oder nicht.

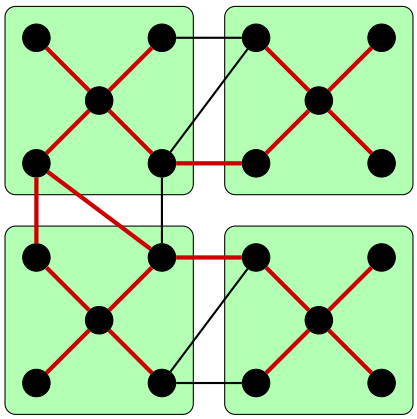


Abbildung 3.2: Beispiel für (ρ, μ)-Partition mit $\rho = 1$ und $\mu = 4$

Es gibt sehr effiziente, randomisierte Algorithmen, um eine (ρ, μ)-Partition zu berechnen

Theorem 3.5 ([EN19, FGV21]). *Im synchronen CONGEST Modell kann für gegebene jede Ganzzahl $k \geq 2$ eine (ρ, μ)-Partition mit $\rho = k - 1$ und $\mu = O(n^{1+1/k})$ in Erwartung in $O(k)$ Runden berechnet werden.*

Dieser Algorithmus kann mit Synchronizer α (siehe Theorem 3.2) auch in asynchronen Netzwerken verwendet werden.

Korollar 3.6. *Im asynchronen CONGEST Modell kann für gegebene jede Ganzzahl $k \geq 2$ eine (ρ, μ)-Partition mit $\rho = k - 1$ und $\mu = O(n^{1+1/k})$ in Erwartung in Zeit $O(k)$ mit $O(km)$ Nachrichten berechnet werden.*

Intuitiv gesprochen geht der Synchronizer γ so vor, dass der Synchronizer α zwischen den Teilmengen und der Synchronizer β innerhalb der Teilmengen angewendet wird. Insgesamt funktioniert der Algorithmus folgendermaßen:

- Synchronisierung innerhalb einer Teilmenge wie bei Synchronizer β:
  - Jedes Zentrum einer Teilmenge generiert Puls für alle Knoten der Teilmenge
  - Downcast des Pulses im Baum vom Zentrum aus zu allen Knoten der Teilmenge
  - Knoten führt aktuelle Runde nach Erhalt des Pulses aus
  - Knoten ist safe sobald er alle Empfangsbestätigungen erhalten hat
  - Teilmenge ist safe sobald alle Knoten der Teilmenge safe sind
  - Zentrum erfährt durch aggregierten Upcast im Baum, wenn Teilmenge safe ist
- Synchronisierung zwischen den Teilmengen wie bei Synchronizer α:
  - Sobald Teilmenge safe ist, werden benachbarte Teilmengen darüber informiert:
    - Downcast der Information im Baum vom Zentrum aus zu allen Knoten der Teilmenge
    - Knoten in eigener Teilmenge informieren benachbarte Teilmengen über ausgewählte Kanten zu Nachbarn in diesen Teilmengen
  - Zentrum erfährt durch aggregierten Upcast im Baum, wenn über alle ausgewählten Kanten zu benachbarten Teilmengen Information erhalten wurde, dass entsprechende Teilmenge safe ist
  - Sobald alle benachbarten Teilmengen safe sind, wird neuer Puls generiert

Achtung: Es reicht nicht, dass alle Knoten einer Teilmenge safe sind, damit die Teilmenge safe ist. Die Knoten können noch Nachbarn in anderen Teilmengen haben die noch nicht safe sind; dies war bei Synchronizer β nicht der Fall.

Der Synchronizer γ liefert folgende Garantien.

Theorem 3.7. *Jeder synchrone Algorithmus, der R Runden benötigt und dabei M Nachrichten versendet, kann in asynchronen Netzwerken in Zeit $O(R\rho)$ mit $O(M + R(\mu + n))$ Nachrichten simuliert werden, wenn bereits eine (ρ, μ)-Partition berechnet wurde.*

Der Overhead von ρ in der Zeitkomplexität kommt daher, dass zur Simulation jeder Runde eine konstante Anzahl von Up- und Downcasts durchgeführt werden muss. Die Laufzeit dafür ist proportional zum maximalen Radius ρ eines Baums der Teilmengen. Der Overhead von $\mu + n$ in der Nachrichtenkomplexität ergibt sich aus zwei Aspekten. Für jede simulierte Runde werden über jede ausgewählte Kante zwischen den Teilmengen eine konstante Anzahl an Nachrichten gesendet, also insgesamt $O(\mu)$ viele Nachrichten. Für jede simulierte Runde werden außerdem in den konstant vielen Up- und Downcasts konstant viele Nachrichten über jede Baumkante gesendet; da die Bäume keine gemeinsamen Knoten haben, haben sie insgesamt höchstens n Kanten.

Zusammen mit dem asynchronen Algorithmus zur Berechnung einer (ρ, μ)-Partition (Korollar 3.6) ergeben sich folgende Garantien für Synchronizer γ.

Korollar 3.8. *Jeder synchrone Algorithmus, der R Runden benötigt und dabei M Nachrichten versendet, kann im asynchronen Netzwerken in Zeit $O(Rk)$ mit in Erwartung $O(M + Rn^{1+1/k} + mk)$ Nachrichten simuliert werden.*

Insbesondere ergibt sich für $k = \lceil \log n \rceil$ eine Laufzeit von $O(R \log n)$ und eine Nachrichtenkomplexität von $O(M + Rn + m \log n)$.

3.6 Zusammenfassung

Wir haben drei verschiedene Synchronizer-Verfahren kennengelernt, die jeweils unterschiedliche Overheads für Zeit- und Nachrichtenkomplexität liefern. Sie unterscheiden sich konzeptuell darin, ob sie eher lokal, global oder hybrid agieren. Tabelle 1 fasst die Garantien der vorgestellten Synchronizer zusammen.

	Zeit	**Nachrichten**	**Initialisierungsaufwand**
Synchronizer α	$O(R)$	$O(M + Rm)$	–
Synchronizer β	$O(RD)$	$O(M + Rn)$	Breitensuchbaum
Synchronizer γ	$O(R\rho)$	$O(M + R(\mu + n))$	(ρ, μ)-Partition

Tabelle 1: Übersicht der vorgestellten Synchronizer

3.7 Literatur

[Awe85] Baruch Awerbuch. "Complexity of Network Synchronization". In: *Journal of the ACM* 32.4 (1985), S. 804–823. DOI: 10.1145/4221.4227 (siehe S. 28).

[EN19] Michael Elkin und Ofer Neiman. "Efficient Algorithms for Constructing Very Sparse Spanners and Emulators". In: *ACM Transactions on Algorithms* 15.1 (2019), 4:1–4:29. DOI: 10.1145/3274651 (siehe S. 31).

[FGV21] Sebastian Forster, Martin Grösbacher und Tijn de Vos. "An Improved Random Shift Algorithm for Spanners and Low Diameter Decompositions". In: *Proc. of the 25th International Conference on Principles of Distributed Systems, (OPODIS 2021)*. Bd. 217. 2021, 16:1–16:17. DOI: 10.4230/LIPIcs.OPODIS.2021.16 (siehe S. 31).

Dieser Abschnitt basiert zum Teil auf einer Vorlesungseinheit von Christoph Lenzen.

4 Berechnung eines Maximal Independent Set

Bei den bisher betrachteten Problemen (z. B. Broadcast) handelt es sich um *globale* Probleme. Algorithmen zur Lösung dieser Art von Problemen benötigen $\Omega(n)$ Runden, zumindest wenn der Durchmesser D proportional zu n ist. Im Gegensatz dazu stehen *lokale* Probleme, die in $o(n)$ Runden gelöst werden können (unabhängig vom Wert von D). Ein prominentes Beispiel solch ein lokales Problem ist die Berechnung eines Maximal Independent Set.

4.1 Definitionen

Definition 4.1 (Independent Set)**.** *In einem ungerichteten Graph $G = (V, E)$ ist ein **Independent Set** eine Teilmenge von Knoten $U \subseteq V$, in der keine zwei Knoten benachbart sind. Man spricht von einem **Maximal** Independent Set, wenn kein Knoten zu U hinzugefügt werden kann, sodass U ein Independent Set bleibt. Ein **Maximum** Independent Set U hat größtmögliche Kardinalität unter allen Independent Sets.*

Ein Beispiel für die Unterscheidung zwischen diesen beiden Definitionen wird in Abb. 4.1 gegeben.

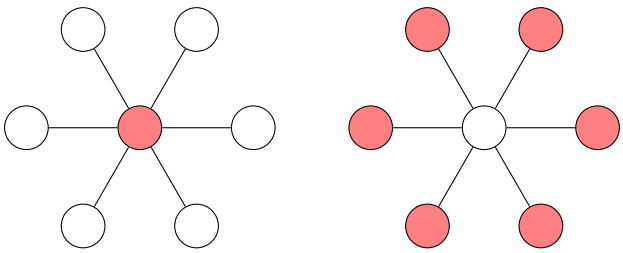


Abbildung 4.1: Die farbig markierten Knoten bilden sowohl im linken als auch im rechten Netzwerk ein Maximal Independent Set. Nur das rechte Independent Set ist jedoch ein Maximum Independent Set, weil es die größtmögliche Kardinalität hat.

4.2 Sequentieller Greedy-Algorithmus

Im Greedy-Algorithmus zur Berechnung eine Maximal Independent Sets werden alle Knoten in beliebiger Reihenfolge betrachtet. Ein betrachteter Knoten v wird nur zum Independent Set hinzugefügt, wenn v keinen Nachbarn im bisherigen Independent Set hat.

```
1 U ← ∅
2 foreach v ∈ V do
3     if ∀u ∈ U : (u, v) ∉ E then
4         U ← U ∪ {v}
```

Algorithm 1: Sequentielle Berechnung eines Maximal Independent Sets

Die Laufzeit im *RAM-Modell* beträgt bei geeigneter Implementierung $O(m + n)$. Im *CONGEST Modell* hingegen werden im schlechtesten Fall $\Omega(n)$ Runden benötigt.

4.3 Verteilter Algorithmus

Ziel eines verteilten Algorithmus ist, dass jeder Knoten weiß, ob er in U ist oder nicht. Der Algorithmus [Lub86, ABI86, MRSZ11] verläuft in Phasen und in jeder Phase führt jeder Knoten v folgende drei Schritte aus:

1. Wähle **Zufallszahl:**
   Knoten v wählt uniforme reelle Zufallszahl $r(v) \in [0, 1)$ und sendet diese an alle Nachbarknoten u

2. Triff **Entscheidung:**
   Nachdem Knoten v die Nachrichten $r(u)$ von allen Nachbarknoten empfangen hat, entscheidet sich v Teil von U zu werden falls $r(v) < r(u)$ für jeden Nachbar u.
   Wenn $v \in U$, werden alle Nachbarknoten drüber informiert und v terminiert.

3. **Empfange** Nachrichten:
   Empfängt Knoten v eine Nachricht über einen zu U hinzugefügten Nachbarn, dann terminiert v.

```
1  while v nicht terminiert do
2      Wähle uniforme reelle Zufallszahl r(v) ∈ [0, 1)
3      Sende r(v) an alle Nachbarn
4      Empfange r(u) von jedem Nachbar u
5      if r(v) < r(u) für jeden Nachbar u then
6          Füge v zur Menge U hinzu
7          Benachrichtige Nachbarn, dass v ∈ U
8          Terminiere v
9      Empfange Nachrichten über zu U hinzugefügte Nachbarn
10     if mindestens ein Nachbar wurde zu U hinzugefügt then
11         Terminiere v
```

Algorithm 2: Dezentrale Berechnung eines Maximal Independent Sets

4.3.1 Korrektheit

Wir wollen zeigen, dass dieser Algorithmus ein Maximal Independent Set berechnet. Dazu beweisen wir zuerst die *Independent Set* Eigenschaft, anschließend die *Maximalität* und schließlich die Terminierung des Algorithmus.

a) Independent Set Wir zeigen mittels einer Induktion über die einzelnen Phasen, dass die Knoten von U ein Independent Set sind und die Menge der terminierten Knoten aus U und den Nachbarn von U besteht. Der Beweis folgt direkt aus der Beschreibung des Algorithmus. Wenn die Induktionshypothese gilt, dann stimmt es, dass jeder Knoten der zu U hinzugefügt wird, keine Nachbarn in U aus der aktuellen Phase und keine Nachbarn in U aus einer früheren Phase hat. Das bedeutet jeder Knoten, den wir hinzufügen, hat keine Nachbarn in U beim Hinzufügen und wird auch später, weil wir alle seine Nachbarn darüber informieren, dass er hinzugefügt wurde, auch keine Nachbarn aus U mehr bekommen.

b) Maximalität Für jeden terminierten Knoten v gilt:

- $v \in U$ oder
- $u \in U$ wobei u ein Nachbar von v ist

Sind also alle Knoten terminiert, gilt dies für jeden Knoten v. Daher gibt es keinen Knoten v, der noch zu U hinzugefügt werden könnte ohne die Independent Set Eigenschaft zu verletzen.

c) Terminierung Die Terminierung dieses Algorithmus ist auf Grund der Zufallskomponente nur mit *Wahrscheinlichkeit* 1 garantiert.

Die Wahrscheinlichkeit, dass zwei zufällig gewählte reelle Zahlen gleich sind ist 0. Durch die Anwendung der Union Bound können wir argumentieren, dass wenn der Algorithmus n Phasen benötigt, die Wahrscheinlichkeit, dass in n dieser Phasen die Zufallszahlen unterschiedlich sind, gleich 1 ist. In jeder Phase wird zumindest der Knoten mit global kleinstem Wert der Menge U hinzugefügt, daher sind n Phasen ausreichend. Das Anwenden der Union Bound (Lemma B.1) gibt uns Terminierung mit Wahrscheinlichkeit 1.

4.3.2 Laufzeit

Überlegung: Immer wenn ein Knoten v zu U hinzugefügt wird, terminiert v sowie alle Nachbarn von v. Somit ergibt sich für jede Phase eine gewisse Reduktion der Kanten im Subgraph der noch nicht terminierten Knoten.

Wir treffen hier zunächst die *idealisierte Annahme*, dass das Senden von Zufallszahlen in $O(1)$ Runden möglich ist und somit jede Phase $O(1)$ Runden dauert.

Können wir also zeigen, dass sich die Anzahl der Kanten in jeder Phase um mindestens die Hälfte reduziert (mit hoher Wahrscheinlichkeit), dann

- werden $O(\log n)$ Phasen benötigt und
- es folgt aus obiger Annahme eine Laufzeit von $O(\log n)$.

Implementierungsdetail: Für das CONGEST Modell reicht es aus, jeweils nur $O(\log n)$ Bits der Zufallszahlen zu übertragen (s.u.).

Analyse: Betrachten wir also zu Beginn den *Erwartungswert* der entfernten Kanten in einer Phase. Wir bedienen uns folgender Analysetricks:

- Jede Kante wird als zwei gerichtete Kanten betrachtet.
- Die Anzahl der entfernten Kanten wird systematisch unterschätzt.

Zudem führen wir für die Analyse noch eine Dominanzbeziehung unter den Knoten ein:

Definition 4.2 (Dominanzbeziehung)**.** *u **dominiert** Nachbar v ($u \twoheadrightarrow v$) falls:*

1. *$r(u) < r(u')$ für alle Nachbarn u' von u und*
2. *$r(u) < r(v)$ für alle Nachbarn $v' \neq u$ von v*

Für die Wahrscheinlichkeit „u dominiert v“ gilt nun:

$$\Pr[u \twoheadrightarrow v] \geq \frac{1}{\deg(u) + \deg(v)}$$

Dies kann folgendermaßen begründet werden: Die totale Anzahl an Knoten in den Nachbarschaften von u und v, ist höchstens $\deg(u)+\deg(v)$. Das ist die obere Schranke, weil sich die Nachbarschaften auch überlappen können. Wir betrachten alle Knoten in diesen beiden Nachbarschaften und deren zugehörigen Zufallszahlen $r(\cdot)$. Nun können wir diese Knoten aufsteigend nach ihren Zufallszahlen sortieren. Das ergibt eine uniform zufällige Permutation π der Knoten die sich in den beiden Nachbarschaften befinden.

Bei zufällig gewählten Permutationen ist die Wahrscheinlichkeit, dass ein fixierter Knoten u der erste in dieser Permutation ist, gleich $1/(\text{Anzahl der Knoten in beiden Nachbarschaften})$, da dieses Ereignis für jeden Knoten gleich wahrscheinlich ist.

Wir können nun die Anzahl entfernter Kanten (u, v) mit den Dominanzbeziehungen abschätzen. Die Anzahl entfernter Kanten entspricht mindestens der Anzahl an Kanten in denen $u \twoheadrightarrow v$ gilt. Dazu definieren wir:

- F: Menge ungerichteter Kanten am Beginn der aktuellen Phase
- X: Zufallsvariable für Anzahl entfernter *gerichteter* Kanten
- $X_{(u \twoheadrightarrow v)}$: Anzahl an Nachbarkanten von v, die wegen $(u \twoheadrightarrow v)$ entfernt werden

$$X_{(u \twoheadrightarrow v)} = \begin{cases} \deg(v) & \text{falls } u \twoheadrightarrow v \\ 0 & \text{andernfalls} \end{cases}$$

Anmerkung: Die zweite Eigenschaft der Dominanzbeziehung stellt sicher, dass v wegen u entfernt wird, und nicht etwa durch einen Nachbarn w von v. Somit wird keine entfernte Kante doppelt gezählt (da jede gezählte Kante eindeutig einem $X_{(u \twoheadrightarrow v)}$ zugeordnet wird).

Mit der Linearität des Erwartungswerts und der Schranke für $\Pr[u \twoheadrightarrow v]$ von oben ergibt sich folgende Abschätzung:

$$\begin{aligned} \operatorname{Ex}[X] &\geq \sum_{\{u,v\} \in F} \left(\operatorname{Ex}[X_{(u \twoheadrightarrow v)}] + \operatorname{Ex}[X_{(v \twoheadrightarrow u)}] \right) \\ &= \sum_{\{u,v\} \in F} \left(\Pr[u \twoheadrightarrow v] \cdot \deg(v) + \Pr[v \twoheadrightarrow u] \cdot \deg(u) \right) \\ &\geq \sum_{\{u,v\} \in F} \left(\frac{\deg(v)}{\deg(u) + \deg(v)} + \frac{\deg(u)}{\deg(v) + \deg(u)} \right) \\ &= \sum_{\{u,v\} \in F} 1 = |F| \end{aligned}$$

In Erwartung ist die Anzahl der *gerichteten* Kanten die in einer Phase entfernt werden also mindestens $|F|$. Da es für jede ungerichtete Kante zwei gerichtete Kanten gibt, die nur gleichzeitig entfernt werden können, entspricht das mindestens $\frac{|F|}{2}$ *ungerichteten* Kanten, die entfernt werden.

Lemma 4.3. *In jeder Phase wird in Erwartung wird die Hälfte der Kanten aus dem durch die nicht-terminierten Knoten induzierten Subgraph entfernt.*

Der Erwartungswert allein liefert nur eine relativ eine schwache Aussage. Wir hätten gerne eine stärkere Aussage der Art, dass mit hoher Wahrscheinlichkeit relativ viele Kanten in jeder Phase entfernt werden. Diese erhalten wir durch eine kombinierte Anwendung von Markov Bound und Chernoff Bound.

Lemma 4.4. *In jeder Phase werden mit Wahrscheinlichkeit mindestens $\frac{1}{4}$ mindestens $\frac{1}{3}$ Kanten aus dem durch die nicht-terminierten Knoten induzierten Subgraph entfernt.*

Beweis. Wir werden zeigen, dass mit Wahrscheinlichkeit mindestens $\frac{1}{4}$ höchstens $\frac{2}{3}$ der Kanten *nicht* aus dem durch die nicht-terminierten Knoten induzierten Subgraph entfernt werden. Diese Aussage impliziert das Lemma: Denn wenn höchstens $\frac{2}{3}$ der Kanten *nicht* entfernt werden, müssen mindestens $\frac{1}{3}$ der Kanten entfernt werden.

Sei F die Menge der Kanten des durch die nicht-terminierten Knoten induzierten Subgraphs am Beginn der aktuellen Phase Sei Y die Zufallsvariable, die die Anzahl der Kanten angibt, die in dieser Phase *nicht* aus dem durch die nicht-terminierten Knoten induzierten Subgraph entfernt werden. Wir wissen aus Lemma 4.3, dass $\mathrm{Ex}[Y] \leq \frac{|F|}{2}$. Durch Anwendung der Markov-Ungleichung (siehe Theorem B.12) mit $\alpha = \frac{4}{3}$ folgt:

$$\Pr\left[Y \geq \frac{2}{3} \cdot |F|\right] \leq \Pr\left[Y \geq \frac{4}{3} \cdot \mathrm{Ex}[Y]\right] \leq \frac{3}{4}.$$

Für das Gegenereignis gilt nun:

$$\Pr\left[Y < \frac{2}{3} \cdot |F|\right] = 1 - \Pr\left[Y \geq \frac{2}{3} \cdot |F|\right] \geq 1 - \frac{3}{4} = \frac{1}{4}.$$

Wie oben argumentiert impliziert diese Schranke das Lemma. □

Lemma 4.5. *Mit hoher Wahrscheinlichkeit terminiert Algorithmus 10 nach $72\lceil c \log n \rceil$ Phasen.*

Beweis. Die Idee hier ist, Phasen in „gute“ bzw. „schlechte“ Phasen zu unterteilen. Wir führen dafür für jede Phase eine binäre Zufallsvariable ein:

$$Z_i = \begin{cases} 1 & \text{falls in Phase } i \text{ mindestens } \frac{1}{3} \text{ der Kanten entfernt werden} \\ 0 & \text{andernfalls} \end{cases}$$

Phase i gilt als gut, wenn $Z_i = 1$ ist, und als schlecht, wenn $Z_i = 0$ ist. Somit gibt $Z = \sum_{i=1}^{72\lceil c \log n \rceil} Z_i$ die Anzahl der guten Phasen an.

Aus der Linearität des Erwartungswerts folgt daher

$$\mathrm{Ex}[Z] = \tfrac{1}{4} \cdot 72\lceil c \ln n \rceil \geq 18c \ln n.$$

Allgemein reduziert sich die Anzahl der (ursprünglich $m \leq n^2$) Kanten nach k guten Phasen auf höchstens $(\frac{2}{3})^k m \leq (\frac{2}{3})^k n^2$. Sobald der durch die nicht-terminierten Knoten induzierte Subgraph keine Kanten mehr hat, terminiert der Algorithmus. Dies ist jedenfalls nach $k' = \log_{3/2}(n^2) + 1$ vielen guten Phasen der Fall, da $(\frac{2}{3})^{k'} n^2 < 1$ ist.

Die Gegenwahrscheinlichkeit, nämlich dass weniger als k' gute Phasen auftreten, kann nun mit Hilfe einer Chernoff Bound (siehe Theorem B.14) abgeschätzt werden. Dabei setzen wir $p = \frac{1}{4}$, $\mu = p \cdot 72\lceil c \log n \rceil = 18\lceil c \log n \rceil$ und $\delta = \frac{1}{3}$. Wir haben in Lemma 4.4 bereits gezeigt, dass $\Pr[Z_i = 1] \geq \frac{1}{4} = p$ gilt. Mit der Kette von Ungleichungen

$$k' - 1 = \log_{3/2}(n^2) \leq \tfrac{2 \ln n}{\ln(3/2)} \leq 6 \ln n \leq (1 - \tfrac{1}{3}) 18c \ln n = (1 - \tfrac{1}{3})\mu$$

können wir die Chernoff Bound nun folgendermaßen anwenden:

$$\Pr[Z < k'] = \Pr[Z \leq k' - 1] \leq \Pr\left[Z \leq (1 - \tfrac{1}{3})\mu\right] \leq \frac{1}{e^{\frac{\delta^2}{2} \cdot \mu}} = \frac{1}{e^{\mu/18}} \leq \frac{1}{e^{c \ln n}} = \frac{1}{n^c}.$$

Somit gilt $\Pr[Z \geq k'] \geq 1 - \frac{1}{n^c}$. Deshalb terminiert der Algorithmus mit Wahrscheinlichkeit mindestens $1 - \frac{1}{n^c}$ innerhalb von $72\lceil c \log n \rceil$ Phasen. □

4.3.3 Bandbreitenbeschränkung

Bisher haben wir angenommen, dass wir beliebige reelle Zufallszahlen aus dem Intervall $[0, 1)$ generieren und in einer Runde senden können. Im CONGEST Modell ist die Nachrichtengröße jedoch auf $O(\log n)$ Bits beschränkt. Wir müssen also zeigen, dass es auch mit dieser Bandbreitenbeschränkung von $O(\log n)$ möglich ist, für jede Kante $\{u, v\}$ in einer konstanten Anzahl an Runden zu entscheiden werden, ob $r(u) < r(v)$ oder $r(u) > r(v)$.

Die Idee hinter der Adaption für das CONGEST Modell, ist die reellen Zufallszahlen als unendliche Strings zufälliger Bits zu repräsentieren. Für den Vergleich zweier solcher Zahlen würde es reichen, dass zwei Knoten diese Bits nur bis zur ersten unterschiedlichen Stelle austauschen. Wir wollen zeigen, dass es mit hoher Wahrscheinlichkeit ausreicht, jeweils nur die ersten $(c + 3)\lceil \log n \rceil = O(\log n)$ Bits auszutauschen.

Die Wahrscheinlichkeit, dass die ersten $(c + 3)\lceil \log n \rceil$ Bits identisch sind, beträgt

$$\left(\frac{1}{2}\right)^{(c+3)\lceil \log n \rceil} \leq \left(\frac{1}{2}\right)^{(c+3)\log n} = \frac{1}{n^{c+3}}.$$

Die Basis $\frac{1}{2}$ ergibt sich dadurch, dass es für jedes Paar von Bits aus den vier möglichen Kombinationen $\{00, 01, 10, 11\}$ zwei Kombinationen gibt, in denen die Bits gleich sind. Die Wahrscheinlichkeit, dass für irgendeines der höchstens n^2 Paare in den höchstens n Runden des Algorithmus die ersten $(c + 3)\lceil \log n \rceil$ Bits gleich sind, beträgt wegen der Union Bound (siehe Lemma B.1) höchstens $\leq n^3 \cdot \frac{1}{n^{c+3}} = \frac{1}{n^c}$. Somit reicht eine Bandbreite von $O(\log n)$ mit hoher Wahrscheinlichkeit aus.

4.3.4 Exkurs: Robustheit der hohen Wahrscheinlichkeit

Wir erinnern uns, dass der Begriff „hohe Wahrscheinlichkeit“ folgendermaßen definiert ist.

Definition 4.6. *Ein Ereignis findet* mit hoher Wahrscheinlichkeit *statt, wenn es mit Wahrscheinlichkeit mindestens* $1 - \frac{1}{n^c}$*, für eine beliebige vorgegebene Konstante* $c \geq 1$*, stattfindet.*

In der Regel hängt bei Algorithmen, die mit hoher Wahrscheinlichkeit korrekt sind, die in der O-Notation verborgene Konstante von c ab und beeinflusst beispielsweise die Laufzeit oder die Nachrichtenkomplexität.

Nehmen wir nun an, dass es n Ereignisse $A_1, \ldots A_n$ gibt, die jeweils mit hoher Wahrscheinlichkeit eintreten. Wie wahrscheinlich ist es, dass alle gleichzeitig eintreten, also wie wahrscheinlich ist das Ereignis $\bigcap_{i=1}^{n} A_i$? Wir können folgendermaßen argumentieren, dass mit „beinahe“ hoher Wahrscheinlichkeit all diese Ereignisse gleichzeitig eintreten:

$$\Pr\left[\bigcap_{i=1}^{n} A_i\right] = 1 - \Pr\left[\bigcup_{i=1}^{n} \bar{A}_i\right] \geq 1 - \sum_{i=1}^{n} \Pr[\bar{A}_i] = 1 - \sum_{i=1}^{n} (1 - \Pr[A_i])$$
$$\geq 1 - \sum_{i=1}^{n} \frac{1}{n^c} = 1 - \frac{n}{n^c} = 1 - \frac{1}{n^{c-1}}$$

Falls nun die Wahrscheinlichkeit für jedes A_i auf $1 - \frac{1}{n^{c+1}}$ erhöht werden könnte, dann findet auch das Ereignis $\bigcap_{i=1}^{n} A_i$ mit hoher Wahrscheinlichkeit (für die vorgegebene Konstante c) statt. In algorithmischen Kontexten kann so eine Erhöhung der Wahrscheinlichkeit oft durchgeführt werden in dem der Algorithmus mit $c + 1$ anstelle von c ausgeführt wird, was in der Regel die Zeit- bzw. Nachrichtenkomplexität nur um Konstanten verändert.

4.4 Literatur

[ABI86] Noga Alon, László Babai und Alon Itai. “A Fast and Simple Randomized Parallel Algorithm for the Maximal Independent Set Problem”. In: *Journal on Algorithms* 7.4 (1986), S. 567–583. DOI: 10.1016/0196-6774(86)90019-2 (siehe S. 34).

[Lub86] Michael Luby. “A Simple Parallel Algorithm for the Maximal Independent Set Problem”. In: *SIAM Journal on Computing* 15.4 (1986), S. 1036–1053. DOI: 10.1137/0215074 (siehe S. 34).

[MRSZ11] Yves Métivier, John Michael Robson, Nasser Saheb-Djahromi und Akka Zemmari. “An optimal bit complexity randomized distributed MIS algorithm”. In: *Distributed Computing* 23.5-6 (2011), S. 331–340. DOI: 10.1007/s00446-010-0121-5 (siehe S. 34).

Dieser Abschnitt basiert zum Teil auf Vorlesungseinheiten von Christoph Lenzen und Roger Wattenhofer.

5 Graph Spanners

5.1 Problemstellung

Eine grundlegende Hürde in der Speicherung und Verarbeitung von Graphen ist deren Größe. Ein Graph mit n Knoten kann potentiell $\Omega(n^2)$ viele Kanten haben. Daher wurden zahlreiche Verfahren entwickelt, um Graphen durch andere Graphen zu komprimieren, die die gleichen Knoten haben wie das Original aber weitaus weniger Kanten, idealerweise nur $O(n)$ viele. In der Regel erfüllen solche komprimierten Graphen nur näherungsweise die gleichen Eigenschaften wie die Originale und oft gibt es einen Trade-off zwischen der Größe des komprimierten Graphs und der Qualität der Annäherung an das Original.

In diesem Abschnitt beschäftigen wir uns mit Kompressionsverfahren, die die Distanz annähernd erhalten. Die zentrale Definition dafür ist die eines Spanners.

Definition 5.1. *Ein* t-Spanner *(Spanner mit* Stretch t*) eines Graphen* $G = (V, E)$ *ist ein Subgraph* $H = (V, F)$*, für den*

$$\operatorname{dist}_H(u, v) \leq t \cdot \operatorname{dist}_G(u, v)$$

für alle Paare von Knoten $u, v \in V$ *gilt.*

Der Spanner H ist dabei ein Subgraph von G in dem Sinn, dass die Knotenmenge V der beiden Graphen gleich ist und die Kantenmenge F von H eine Teilmenge der Kantenmenge E von G ist (also $F \subseteq E$).

Wir diskutieren zunächst zwei grundlegende Eigenschaften dieser Definition. Zunächst gilt, dass der Subgraph H die Distanzen in G niemals überschätzt, da alle Pfade die in H zur Verfügung stehen auch in G zur Verfügung stehen, insbesondere auch jeder kürzeste Weg in H.

Lemma 5.2. *Für jeden Spanner* H *von* G *gilt:* $\operatorname{dist}_H(u, v) \geq \operatorname{dist}_G(u, v)$ *für jedes Paar von Knoten* $u, v \in V$*.*

Zudem können Spanner dadurch charakterisiert werden, dass die Ungleichung aus Definition 5.1 für alle gemeinsamen Endpunkte von Kanten gilt.

Lemma 5.3. *Ein Subgraph* $H = (V, F)$ *ist genau dann ein* t*-Spanner von* $G = (V, E)$ *wenn*

$$\operatorname{dist}_H(u, v) \leq t \cdot w_G(u, v)$$

für jede Kante $(u, v) \in E$ *gilt.*

Die Äquivalenz der beiden Formulierung (alle Paare von Knoten vs. gemeinsame Endpunkte von Kanten) ergibt sich nahezu direkt aus der Betrachtung aller Kanten eines kürzesten Wegs.

Im Folgenden behandeln wir Spanner von ungerichteten, ungewichtete Graphen (in denen also jede Kante Gewicht 1 hat).

5.2 Greedy Spanner

5.2.1 3-Spanner

Wir analysieren zunächst folgenden (sequentiellen) Greedy-Algorithmus [ADDJ+93] zur Berechnung eines 3-Spanners eines Graphen $G = (V, E)$.

```
1 F ← ∅
2 foreach (u, v) ∈ E do
3 |   Sei H = (V, F)
4 |   if dist_H(u, v) > 3 then
5 |   |   F ← F ∪ {(u, v)}
```

Der Algorithmus iteriert über die Kanten von G und überprüft für jede Kante, ob im bisher berechneten Spanner bereits ein Pfad der Länge höchstens 3 zwischen den Endpunkten der Kante existiert. Falls nicht, wird die Kante zum Spanner hinzugefügt. Der Algorithmus folgt also einem *Greedy*-Schema, weil für jede Kante individuell entschieden wird, ob sie notwendig ist oder nicht. Der Algorithmus berechnet *by design* einen Spanner mit Stretch 3, was wir nun formal beweisen.

Lemma 5.4. *Der vom Greedy-Algorithmus berechnete Spanner $H = (V, F)$ ist ein 3-Spanner von G.*

Beweis. Sei $(u, v) \in E$ beliebige Kante von G. Falls (u, v) in die Kantenmenge F von H aufgenommen wurde, dann gilt offensichtlich $\mathrm{dist}_H(u, v) = 1 \leq 3$. Andernfalls ($(u, v) \notin F$), betrachte jene Iteration, in der $\mathrm{dist}_H(u, v)$ überprüft wird (und die Entscheidung getroffen wird, (u, v) nicht in F aufzunehmen) und sei $H' = (V, F')$ der Zustand von H während dieser Iteration. Dann gilt nach Algorithmus $\mathrm{dist}_{H'}(u, v) \leq 3$. Da H' ein Subgraph von H ist, gilt ebenso $\mathrm{dist}_H(u, v) \leq \mathrm{dist}_{H'}(u, v)$ und somit $\mathrm{dist}_H(u, v) \leq 3$. Da (u, v) eine beliebige Kante war, ist H nach Lemma 5.3 ein 3-Spanner von G. □

Der Beweis, dass der berechnete Spanner H nur $O(n^{3/2})$ Kanten hat ist etwas aufwändiger. Dabei ist der Begriff des Girth eines Graphen nützlich.

Definition 5.5. *Der* Girth (Taillenweite) *eines Graphen ist die Länge seines kürzesten Kreises.*

Lemma 5.6. *Der Girth von $H = (V, F)$ ist größer als 4.*

Beweis. Angenommen H hat einen Kreis K der Länge höchstens 4 (d.h. mit höchstens 4 Kanten). Sei (u, v) die im Algorithmus zuletzt zu F hinzugefügte Kante des Kreises K. Die Kanten in $K \setminus \{(u, v)\}$ bilden einen Pfad von u nach v, der in der Iteration, in der (u, v) zu F hinzugefügt wird, bereits in H existiert. Vor dem Hinzufügen von (u, v) zu F gibt es keinen Pfad der Länge höchstens 3 von u nach v. Daher hat der Pfad $K \setminus \{(u, v)\}$ Länge mindestens 4 und der Kreis K somit Länge mindestens 5. Dies steht im Widerspruch zur Annahme, dass K Länge höchstens 4 hat. □

Lemma 5.7. *Jeder ungerichtete Graph mit n Knoten und Girth größer als 4 hat $O(n^{3/2})$ Kanten.*

Wir werden für den Beweis dieses Lemmas folgendes weitere Lemma benötigen, das wir direkt im Anschluss beweisen.

Lemma 5.8. *Jeder ungerichtete Graph mit n Knoten und minimalem Grad $\geq n^{1/2} + 1$ hat Girth höchstens 4.*

Beweis von Lemma 5.7. Sei G ein Graph mit Girth größer als 4 und mindestens $3n^{3/2}$ Kanten. Entferne wiederholt Knoten mit Grad kleiner als $n^{1/2} + 1$ (mit allen anliegenden Kanten) aus G, bis jeder Knoten mindestens Grad $n^{1/2} + 1$ hat. Sei $G' = (V', E')$ der resultierende Graph. Insgesamt werden höchstens $n \cdot (n^{1/2} + 1) \leq 2n^{3/2}$ Kanten entfernt, es gilt also

$$|E'| \geq |E| - 2n^{3/2} \geq 3n^{3/2} - 2n^{3/2} = n^{3/2} > 0\,.$$

Insbesondere ist G' also nicht leer und $|V| \geq 1$. Wegen $|E'| \leq |V'|^2$ gilt außerdem

$$|V'| \geq \frac{|E'|}{|V'|} \geq \frac{|E'|}{n} \geq n^{3/2}/n = n^{1/2}\,.$$

Nach Konstruktion hat G' nun einen minimalen Grad von mindestens $n^{1/2} + 1 \geq |V'|^{1/2} + 1$.

Wir können nun Lemma 5.8 auf G' anwenden: G' hat Girth mindestens 4. Da G' ein Subgraph von G ist, existiert jeder Kreis in G' auch in G. Es folgt, dass G Girth mindestens 4 hat, was im Widerspruch zu unserer initialen Annahme steht. Somit hat jeder Graph mit Girth > 4 weniger als $3n^{3/2} = O(n^{3/2})$ Kanten. □

Beweis von Lemma 5.8. Nehmen wir an, dass G Girth mindestens 5 hat. Wir betrachten einen beliebigen Knoten v und den Breitensuchbaum der Tiefe 2 von ausgehend von v.

Jede Kante eines Knotens mit Distanz höchstens 1 zu v zu einem Knoten mit Distanz höchstens 2 zu v muss eine Kante dieses Knotens zu seinem Elternknoten oder zu einem Kind im Breitensuchbaum sein, da G ansonsten einen Kreis der Länge höchstens 4 enthält (siehe Abb. 5.1). Jeder Knoten mit Distanz weniger als 2 zu v (also die inneren Knoten im Breitensuchbaum der Tiefe 2 von v) hat also mindestens $\sqrt{n}$ Kinder im Breitensuchbaum, da höchstens eine seiner anliegenden Kanten zu einem Elternknoten gehen kann.

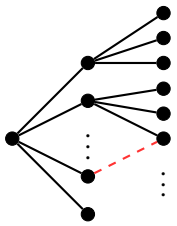


Abbildung 5.1: Breitensuchbaum der Tiefe 2 von einem beliebigen Wurzelknoten v aus. Die rot gestrichelte Kante kann nicht existieren, da sie einen Kreis der Länge 4 schließen würde.

Die Anzahl an Knoten in Distanz 2 zu v ist also mindestens $n^{1/2} \cdot n^{1/2} = (n^{1/2})^2 = n$. Gemeinsam mit v zählen wir für G also mindestens $n + 1$ Knoten. Aufgrund dieses Widerspruchs war unsere Annahme falsch und wir schließen daraus, dass G Girth höchstens 4 hat. □

Wir haben mit dem Greedy-Algorithmus also folgende Aussage gezeigt.

Theorem 5.9. *Jeder Graph G mit n Knoten hat einen 3-Spanner mit $O(n^{3/2})$ Kanten.*

5.2.2 Verallgemeinerung

Die Aussage von Theorem 5.9 lässt sich auf folgenden Trade-Off zwischen Stretch und Größe des Spanners verallgemeinern.

Theorem 5.10. *Für jede ganze Zahl $k \geq 2$ hat jeder Graph G mit n Knoten einen $(2k-1)$-Spanner mit $O(n^{1+1/k})$ Kanten.*

Für $k = 2$ ergibt sich, wie bereits von uns gezeigt, ein 3-Spanner der Größe $O(n^{3/2})$. Für $k = \lceil \log n \rceil$ ergibt sich

$$n^{1/k} \leq n^{1/\log n} = \left(2^{\log n}\right)^{1/\log n} = 2^{(1/\log n)\cdot \log n} = 2$$

und damit ist $O(n^{1+1/k}) = O(n)$. Dies entspricht, bis auf konstante Faktoren, der minimal notwendigen Anzahl von $\Omega(n)$ Kanten für verbundene Graphen.

Ein $(2k-1)$-Spanner mit $O(n^{1+1/k})$ Kanten kann ebenfalls mit dem Greedy-Algorithmus berechnet werden; wir müssen nur tolerierte Länge des Pfads im Spanner von 3 auf $2k-1$ verallgemeinern.

```
1 F ← ∅
2 foreach (u, v) ∈ E do
3     Sei H = (V, F)
4     if dist_H(u, v) > 2k − 1 then
5         F ← F ∪ {(u, v)}
```

Wir skizzieren nun wie die vorherigen Argumente angepasst werden müssen, um Theorem 5.10 zu beweisen.

Um zu beweisen, dass der Stretch $2k-1$ beträgt, kann der gleiche Beweis wie in Lemma 5.4 geführt werden, es muss nur die Zahl 3 durch die Zahl $2k-1$ ersetzt werden. Für den Beweis, dass H die gewünschte Größe hat gehen wir ähnlich vor.

Lemma 5.11. *Der Girth von $H = (V, F)$ ist größer als $2k$.*

Beweis. Angenommen H hat einen Kreis K der Länge höchstens $2k$ (d.h. mit höchstens 4 Kanten). Sei (u, v) die im Algorithmus zuletzt zu F hinzugefügte Kante des Kreises K. Die Kanten in $K \setminus \{(u, v)\}$ bilden einen Pfad von u nach v, der in der Iteration, in der (u, v) zu F hinzugefügt wird, bereits in H existiert. Vor dem Hinzufügen von (u, v) zu F gibt es keinen Pfad der Länge höchstens $2k-1$ von u nach v. Daher hat der Pfad $K \setminus \{(u, v)\}$ Länge mindestens $2k$ und der Kreis K somit Länge mindestens $2k+1$. Dies steht im Widerspruch zur Annahme, dass K Länge höchstens $2k$ hat. □

Lemma 5.12. *Jeder ungerichtete Graph mit n Knoten und Girth größer als $2k$ hat $O(n^{1+1/k})$ Kanten.*

Um dieses Lemma zu beweisen muss ebenfalls nur der Beweis von Lemma 5.7 mit anderen Zahlen geführt werden. Auch der Beweis des Hilfslemmas muss nur leicht adaptiert werden.

Lemma 5.13. *Jeder ungerichtete Graph mit n Knoten und minimalem Grad $\geq n^{1/k} + 1$ hat Girth höchstens $2k$.*

Beweis. Nehmen wir an, dass G Girth mindestens $2k+1$ hat. Wir betrachten einen beliebigen Knoten v und den Breitensuchbaum der Tiefe k von ausgehend von v.

Jede Kante eines Knotens mit Distanz höchstens $k-1$ zu v zu einem Knoten mit Distanz höchstens k zu v muss eine Kante dieses Knotens zu seinem Elternknoten oder zu einem Kind im Breitensuchbaum sein, da G ansonsten einen Kreis der Länge höchstens $2k$ enthält. Jeder Knoten mit Distanz weniger als k zu v (also die inneren Knoten im Breitensuchbaum der Tiefe k von v) hat also mindestens $n^{1/k}$ Kinder im Breitensuchbaum, da höchstens eine seiner anliegenden Kanten zu einem Elternknoten gehen kann.

Die Anzahl an Knoten in Distanz 2 zu v ist also mindestens $(n^{1/k})^k = n$. Gemeinsam mit v zählen wir für G also mindestens $n+1$ Knoten. Aufgrund dieses Widerspruchs war unsere Annahme falsch und wir schließen daraus, dass G Girth höchstens $2k$ hat. □

5.2.3 Optimalitätsvermutung

Der Trade-off zwischen Stretch und Größe des durch den Greedy-Algorithmus berechneten Spanners geht zurück auf graphentheoretische Überlegungen zwischen der Größe von Graphen und ihren

Girth. Die *Girth-Vermutung* von Paul Erdős [Erd63] besagt, dass es für jedes $k \geq 2$ und jedes genügend große n gibt einen Graph mit n Knoten und $\Omega(n^{1+1/k})$ Kanten gibt, der keinen Kreis der Länge höchstens $2k$ besitzt. Diese Vermutung wurde bisher nur für kleine Werte von k bewiesen und es ist ein offenes Problem in der Kombinatorik, ob die Verallgemeinerung für beliebige k zutrifft. Falls die Girth-Vermutung wahr ist, dann ist der $2k - 1/O(n^{1+1/k})$ Trade-off – bis auf Konstanten in der Größe – optimal.

Lemma 5.14. *Wenn die Girth Vermutung zutrifft, dann gibt es für alle genügend große n einen Graph G mit n Knoten, so dass jeder $(2k-1)$-Spanner von G mindestens $\Omega(n^{1+1/k})$ Kanten besitzt.*

Beweis. Sei G ein Graph wie in der Girth Vermutung, d.h., G hat $\Omega(n^{1+1/k})$ Kanten und Girth mehr als $2k$. Angenommen es gibt einen nicht-trivialen $(2k-1)$-Spanner H von G. Nicht-trivial heißt es gibt mindestens eine Kante (u, v) in G, die nicht in H vorkommt. Da H ein $(2k-1)$-Spanner von G ist gibt es einen Pfad P der Länge höchstens $2k-1$ von u nach v in H. Nun ist aber $P \cup \{(u, v)\}$ ein Kreis der Länge höchstens $2k$ in G, was der Annahme widerspricht, dass der Girth von G größer als $2k$. Somit ist G der einzige $(2k-1)$-Spanner von G. Es ist also zutreffend zu sagen, dass jeder $(2k-1)$-Spanner von G so viele Kanten wie G hat, nämlich $\Omega(n^{1+1/k})$ viele. □

Einen besseren Trade-off zwischen Stretch und asymptotischer Größe von Spannern zu finden würde also darauf hinauslaufen, die Girth-Vermutung zu widerlegen, was für sich selbst ein Durchbruch in der Kombinatorik wäre.

5.3 Spanner-Berechnung durch Clustering

Wie wir gesehen haben berechnet der einfache Greedy-Algorithmus bereits einen nützlichen Spanner. Allerdings ist dieser Greedy-Algorithmus hochgradig sequentiell, da die Kanten des Graphs nicht unabhängig voneinander geprüft werden können. Für parallele und verteilte Rechenmodelle (wie das CONGEST Modell) wurden daher andere Algorithmen entwickelt, die Spanner mit dem vermutlich optimalen Trade-off berechnen bzw. möglichst nahe an diesen Trade-off herankommen.

Wir gehen nun näher darauf ein, wie Spanner effizient im CONGEST Modell berechnet werden können. Ein bekannter Algorithmus von Baswana und Sen [BS07] hat folgende Garantien.

Theorem 5.15 ([BS07])**.** *Für jedes $k \geq 2$ kann ein $(2k-1)$-Spanner eines ungewichteten Graphen mit $O(n^{1+1/k} + kn)$ Kanten in Erwartung in $O(k^2)$ Runden im CONGEST Modell berechnet werden.*

Dieser Algorithmus kann auf gewichtete Graphen erweitert werden. Wir werden im Folgenden den Algorithmus für den Spezialfall $k = 2$ vorstellen; es wird also ein 3-Spanner mit $O(n^{3/2})$ Kanten berechnet. Das Ziel ist, dass am Ende der Berechnung jeder Knoten weiß, welche seiner anliegenden Kanten zum Spanner gehören.

Der Algorithmus von Baswana und Sen basiert auf der Berechnung von Clustern, die sich formal folgendermaßen definieren lassen.

Definition 5.16. *Ein Cluster ist eine Menge zusammenhängender Knoten. Ein Clustering ist eine Menge nicht-überlappender Cluster.*

Der Algorithmus zur Berechnung des 3-Spanners besteht grundsätzlich aus drei Teilen. Zunächst wird zufällig eine Teilmenge Z der Knoten bestimmt, die Zentren. Danach werden Stern-Cluster um die Zentren als Mittelpunkt gebildet, wobei die jeweiligen Kanten zum Mittelpunkt des Sterns zum Spanner hinzugefügt werden. Abschließend werden noch zusätzliche Kanten zum Spanner

hinzugefügt: geclusterte Knoten fügen für jedes benachbarte Cluster *eine* der Kanten in das benachbarte Cluster zum Spanner hinzu und nicht-geclusterte Knoten fügen die Kanten zu allen ihren Nachbarn zum Spanner hinzu. Wir geben nun den entsprechenden Pseudocode.

```
1  H ← (V, ∅)
2  Z ← ∅
3  foreach Knoten v ∈ V do
4      Füge v zu Z mit Wahrscheinlichkeit p = 1/√n hinzu und erstelle Cluster für v
5  foreach Knoten v ∈ V \ Z do
6      if v hat (mindestens) einen Nachbar aus Z then
7          Füge v zum Cluster eines Nachbarn aus Z in hinzu
8          Füge Kante zu diesem Nachbar zu H hinzu
9  foreach Knoten v ∈ V do
10     if v ist Teil eines Clusters then
11         Füge für jedes mit v benachbarte Cluster eine Kante zu H hinzu
12     else
13         Füge Kanten zu allen Nachbarn von v zu H hinzu
```

Ein Beispiel für ein mögliches Ergebnis des Algorithmus ist in Abb. 5.2 gegeben.

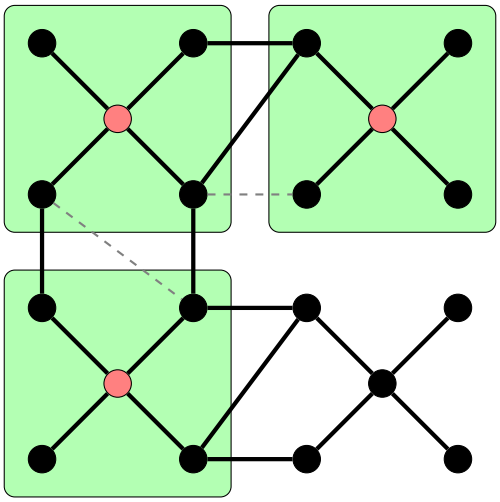


Abbildung 5.2: Beispiel für ein Ergebnis des Algorithmus. Die nicht im Spanner enthaltenen Kanten sind grau gestrichelt. Die zufällig gewählten Zentren sind rot markiert und die Cluster mit den Zentren als Mittelpunkte sind grün markiert

Im Algorithmus können Cluster über die IDs ihrer Zentren unterschieden werden. Diese IDs können in konstanter Zeit allen Knoten im Cluster sowie allen benachbarten Knoten eines Clusters bekannt gemacht werden. Daher kann der Algorithmus offensichtlich in einer konstanten Anzahl an Runden im CONGEST Modell implementiert werden. Wir analysieren nun den Stretch und die Größe des berechneten Spanners.

Lemma 5.17. *Der vom Algorithmus berechnete Spanner H hat Stretch* 3.

Beweis. Sei (u, v) eine beliebige Kante aus G. Falls einer der Endpunkte u oder v nicht geclustert ist, enthält H die Kante (u, v). Falls sowohl u als auch v geclustert sind, unterscheiden wir zwei Fälle:

entweder u und v sind im selben Cluster oder in verschiedenen Clustern. Falls u und v im selben Cluster sind, gibt es einen Pfad der Länge 2 von u nach v über das Zentrum des Clusters. Falls u und v in verschiedenen Cluster sind, dann ist das Cluster von v zu u benachbart, weil es mindestens eine Kante (nämlich (u, v)) von u in das Cluster von v gibt. Daher enthält H irgendeine Kante (u, w) so dass w ein Knoten im Cluster von v ist. Somit gibt es einen Pfad der Länge höchstens 3 von u nach v über w und das Zentrum des gemeinsamen Clusters von w und v. □

Lemma 5.18. *Der vom Algorithmus berechnete Spanner H hat in Erwartung $O(n^{3/2})$ Kanten.*

Beweis. Es gibt prinzipiell drei Typen von Kanten in H:

1. Kanten zum Zentrum des Clusters für geclusterte Knoten
2. Kanten zu benachbarten Clustern für geclusterte Knoten
3. Kanten zu Nachbarn für nicht-geclusterte Knoten

Wir zeigen nun, dass es von jedem dieser drei Typen in Erwartung $O(n^{3/2})$ Kanten gibt, woraus dann (aufgrund der Linearität des Erwartungswerts) folgt, dass H $O(n^{3/2})$ Kanten hat.

Jeder geclusterte Knoten hat eine Kante des Typs 1 und da es höchstens n geclusterte Knoten gibt, gibt es höchstens $O(n) = O(n^{3/2}$ Kanten des Typs 1. Jeder geclusterte Knoten hat höchstens so viele Kanten des Typs 2 wie es Cluster gibt. Die Anzahl der Cluster ist $|Z|$. Da die Anzahl der Zentren binomialverteilt mit Parametern p und n ist, gilt $\mathrm{Ex}[|Z|] = pn = \sqrt{n}$. Da es höchstens n geclusterte Knoten gibt, gibt es in Erwartung höchstens $O(n^{3/2}$ Kanten des Typs 2. Jeder nicht-geclusterte Knoten hat so viele Kanten des Typs 3 wie er Nachbarn hat. Iteriert man über die Nachbarn eines Knotens in beliebiger Reihenfolge, ist der Zeitpunkt des Auftretens des ersten Zentrums geometrisch verteilt mit Parameter p; in Erwartung tritt also nach $\frac{1}{p} = \sqrt{n}$ Knoten das erste Zentrum unter den Nachbarn auf. Da nicht-geclusterte Knoten keine Zentren als Nachbarn haben, haben sie also höchstens $O(\sqrt{n})$ Nachbarn in Erwartung. Da es höchstens n nicht-geclusterte Knoten gibt, gibt es in Erwartung höchstens $O(n^{3/2})$ Kanten des Typs 3. □

Zuletzt sei angemerkt, dass man das Ergebnis des Algorithmus auch als $(1, n^{3/2})$-Partition im Sinne von Definition 3.4 zur Anwendung im Synchronizer γ interpretiert werden kann.

5.4 Literatur

[ABSH+20] Abu Reyan Ahmed, Greg Bodwin, Faryad Darabi Sahneh, Keaton Hamm, Mohammad Javad Latifi Jebelli, Stephen G. Kobourov und Richard Spence. "Graph spanners: A tutorial review". In: *Computer Science Review* 37 (2020), S. 100253. DOI: 10.1016/j.cosrev.2020.100253.

[ADDJ+93] Ingo Althöfer, Gautam Das, David P. Dobkin, Deborah Joseph und José Soares. "On Sparse Spanners of Weighted Graphs". In: *Discret. Comput. Geom.* 9 (1993), S. 81–100. DOI: 10.1007/BF02189308 (siehe S. 40).

[BS07] Surender Baswana und Sandeep Sen. "A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs". In: *Random Structures and Algorithms* 30.4 (2007), S. 532–563. DOI: 10.1002/rsa.20130 (siehe S. 44).

[Erd63] Paul Erdős. "Extremal problems in graph theory". In: *Proceedings of the Symposium on Theory of Graphs and its Applications*. 1963, S. 2936 (siehe S. 44).

Dieser Abschnitt basiert zum Teil auf einer Vorlesungseinheit von Virginia Vassilevska Williams.

6 Berechnung kürzester Wege

6.1 Problemstellung

Es gibt verschiedene Algorithmen zur Berechnung von kürzesten Wegen in verteilten Modellen wie dem CONGEST Modell. Wir definieren zwei Probleme:

Single-Source Shortest Paths (SSSP): Es ist ein Startknoten s gegeben und das Ziel ist, von diesem Knoten aus die Distanz zu jedem Knoten v zu berechnen. Jeder Knoten weißt initial, ob er der Startknoten ist oder nicht. Am Ende soll jeder Knoten v die Distanz $\mathrm{dist}(s, v)$ vom Startknoten s zu v kennen. Man könnte noch zusätzlich verlangen, dass ein Baum kürzester Wege berechnet werden muss. In einem verteilten Modell hieße das, dass jeder Knoten den Vorgängerknoten auf dem kürzesten Weg von s kennen muss; dieser entspricht dem Elternknoten im Baum.

All-Pairs Shortest Paths (APSP): Das Ziel ist die Distanz $\mathrm{dist}(u, v)$ für alle Paare von Knoten u und v zu berechnen. Am Ende kennt jeder Knoten v für jeden anderen Knoten u die Distanz $\mathrm{dist}(u, v)$. Für jeden Knoten müssen also $n - 1$ verschiedene Distanzwerte berechnet werden.

6.2 Ungewichtete Graphen

6.2.1 Wiederholung: Breitensuche

In „ungewichteten“ Graphen haben alle Kante das gleiche Gewicht und wir normieren dieses Gewicht auf 1. Wir können das SSSP-Problem in solchen Netzwerken durch Breitensuche lösen: Der Startknoten s hat die Initialdistanz 0 zu sich selbst ($\mathrm{dist}(s, s) = 0$), und jeder Knoten, der erstmals eine Distanzinformation erhalten hat, sendet eine Nachricht mit einer um 1 erhöhten Distanz an alle Nachbarn.

Ein Beispiel für den Breitensuche-Algorithmus, den wir bereits kennengelernt haben, ist in den Abb. 6.1 zu finden: Anfangs hat der Startknoten Distanz 0 zu sich selbst und sendet in der ersten Runde die Distanzinformation 1 an alle seine Nachbarn. Der Startknoten wird zum Elternknoten seiner Nachbarn im Breitensuchbaum und die entsprechende Baumkante ist jene Kante, über die die Distanzinformation empfangen wurde. Die Nachbarn übernehmen diesen Distanzwert und senden in der zweiten Runde die Distanzinformation 2 an alle ihre Nachbarn, außer ihren Elternknoten im Baum. Diese Propagation von Distanzwerten wiederholt sich bis alle Knoten erreicht wurden.

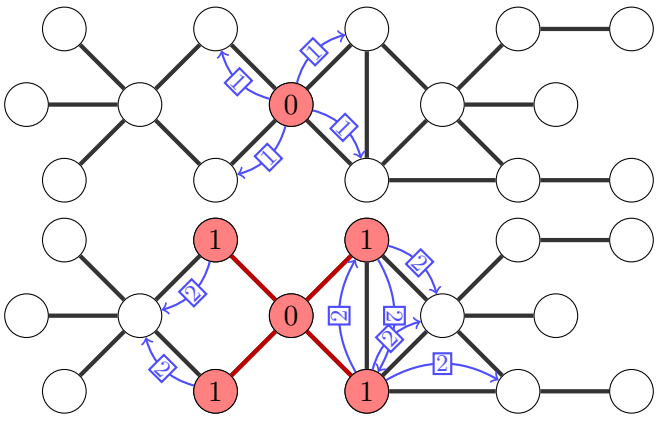


Abbildung 6.1: Beispiel für erste Runde (oben) und zweite Runde (unten) der Breitensuche im gegebenen Netzwerk

Aus den Garantien, die wir für den Breitensuche-Algorithmus bereits kennengelernt haben, folgt folgendes Theorem für das SSSP-Problem.

Theorem 6.1. *Im CONGEST Modell kann das SSSP-Problem für ungewichtete Graphen in $O(D)$ Runden gelöst werden.*

6.2.2 All-Pairs Shortest Paths

Idee: Man führt für jeden Knoten die Breitensuche in ungewichteten Graphen aus. Dadurch erhält man für jeden Knoten die Distanz zu jedem anderen Knoten.

Naive Umsetzung: Wenn ein Knoten an mehreren Breitensuchen teilnimmt, kann es passieren, dass dieser Knoten gleichzeitig mehrere Distanzinformationen von mehreren Breitensuchen empfängt und in der gleichen Runde all diese Informationen weiterleiten müsste. Dies kann man wegen der beschränkten Nachrichtengröße im CONGEST Modell aber nicht einfach dadurch umsetzen, dass all diese Informationen in eine Nachricht gepackt werden. Der naive Ansatz wäre nun die Breitensuchen schrittweise nacheinander auszuführen, wofür dann $O(nD)$ Runden benötigt werden. Im Fall der Breitensuche gibt tatsächlich aber eine effizientere Lösung: Parallelisierung durch Randomisierung.

6.2.3 Parallelisierung durch Random Delay

Wir verwenden die Random Delay Technik [LMR94, Nan14], um mehrere Instanzen der Breitensuche parallel ausführen zu können. Das Ziel ist dabei Breitensuche für $k = |U|$ verschiedene Startknoten auszuführen.

Idee: Wir nutzen aus, dass im Breitensuche-Algorithmus jeder Knoten höchstens einmal Distanzinformation an seine Nachbarn sendet.

Random Delay Algorithmus:

1 Jeder Startknoten s_i wählt uniform zufälliges $\delta_i \in \{0, \dots, k-1\}$
2 Knoten s_i startet Instanz des Algorithmus mit Verzögerung δ_i

Wir werden sehen, dass diese Art der Randomisierung dafür sorgt, dass die zu versendenden Distanzinformationen für jeden Knoten einigermaßen gut verteilt werden so dass wenige Distanzinformationen verschiedener Breitensuchen gleichzeitig gesendet werden müssen. Wir werden also folgendes Lemma beweisen.

Lemma 6.2. *Der Random Delay Algorithmus berechnet einen Breitensuchbaum für jeden Knoten in U in $O(k + D)$ Runden und benötigt Nachrichtengröße $O(\log^2 n)$ mit hoher Wahrscheinlichkeit.*

Dabei ist klar, dass bei einem Delay von höchstens $k - 1$ die letzte Breitensuche nach $O(k + D)$ Runden abgeschlossen ist. Unsere Analyse wird sich daher auf die Beschränkung der Nachrichtengröße fokussieren. Wenn man Nachrichtengröße $O(\log^2 n)$ zur Verfügung hätte, würde dann der Random Delay Algorithmus $O(k + D)$ Runden benötigen. Da wir nur Nachrichtengröße $O(\log n)$ im CONGEST Modell zur Verfügung haben, simulieren wir den Algorithmus aus Lemma 6.2 folgendermaßen: Jede Runde des ursprünglichen Algorithmus aus dem Lemma wird unterteilt in $\log n$ simulierte Runden, in denen jeweils $O(\log n)$ Bits der zu übertragenden Nachricht gesendet werden. Daraus folgt, dass (mit dieser Simulation) der Random Delay Algorithmus $O((k + D) \log n)$ Runden im CONGEST Modell benötigt.

Theorem 6.3. *Im CONGEST Modell gibt es einen Monte-Carlo Algorithmus, der das APSP Problem für ungewichtete Graphen in $O(n \log n)$ Runden mit hoher Wahrscheinlichkeit löst.*

Das Theorem folgt aus der obigen Diskussion, da $D \leq n - 1$ und $k = n - 1$ gilt. Es handelt sich hierbei um einen Monte-Carlo Algorithmus, der korrekt ist, wenn die Nachrichtengröße von $O(\log n)$ eingehalten werden kann. Wenn das nicht der Fall ist, ist unklar, was zu tun ist, und wir erlauben dem Algorithmus abzubrechen, was zu einem falschen oder unvollständigen Ergebnis führt.

6.2.4 Analyse

Wir möchten nun folgendes Lemma beweisen.

Lemma 6.4. *In jeder Runde des Random Delay Algorithmus werden mit hoher Wahrscheinlichkeit für jeden Knoten nur Nachrichten aus $O(\log n)$ Instanzen gleichzeitig gesendet.*

Wir wenden folgende Strategie zum Beweis dieses Lemmas an: Wir zeigen, dass es mit hoher Wahrscheinlichkeit nie vorkommt, dass für irgendeinen Knoten in mehr als $O(\log n)$ vielen Instanzen der Breitensuche gleichzeitig Nachrichten gesendet werden müssen. Daraus folgt die gewünschte Bandbreitenbeschränkung für Lemma 6.2, weil jede Nachricht aus den $O(\log n)$ Instanzen nur Größe $O(\log n)$ hat. Somit ist insgesamt eine Nachrichtengröße von $O(\log^2 n)$ ausreichend.

Beweis von Lemma 6.4. Wir wissen, dass in jeder Instanz des Breitensuche-Algorithmus jeder Knoten nur einmal eine Nachricht sendet. Wir bezeichnen für jeden Knoten v und jede Instanz i mit $M_{v,i}$ jene eindeutige Nachricht, die v in der i-ten Instanz der Breitensuche zu irgendeinem Zeitpunkt senden muss. Weiters bezeichnen wir mit $r_{v,i}$ jene Runde, in der diese Nachricht $M_{v,i}$ gesendet werden würde, wenn die i-te Instanz der Breitensuche ganz ohne Delays (und ohne die anderen Instanzen) ausgeführt werden würde.

Wir fixieren zunächst eine Runde r, einen Knoten v und eine Instanz i. Die Wahrscheinlichkeit, dass Nachricht $M_{v,i}$ in Runde r von Knoten v gesendet wird ist höchstens $\frac{1}{k}$, also:

$$\Pr[M_{v,i} \text{ in Runde } r \text{ von } v \text{ gesendet}] \leq \frac{1}{k}$$

Warum ist das so? Wir wissen, dass aufgrund des Delays δ_i der i-ten Instanz die Nachricht $M_{v,i}$ in Runde $\delta_i + r_{v,i}$ gesendet wird. Wenn wir nun annehmen, dass die Nachricht in Runde r gesendet wird, muss $r = \delta_i + r_{v,i}$ gelten, oder anders ausgedrückt: $\delta_i = r - r_{v,i}$. Damit diese Gleichung erfüllt ist, muss $r - r_{v,i}$ einen Wert zwischen 0 bis $k - 1$ annehmen und δ_i muss diesem Wert entsprechen. Für die Wahl von δ_i gibt es k verschiedene Möglichkeiten, die alle gleich wahrscheinlich sind, aber höchstens eine Wahl, die die Anforderung $\delta_i = r - r_{v,i}$ erfüllt. Daher ist die Wahrscheinlichkeit diese Anforderung zu erfüllen höchstens $\frac{1}{k}$.

Wir fixieren nun eine Runde r, einen Knoten v und eine Menge von Nachrichten $\mathcal{M} \subseteq \bigcup_{i=1}^{k} \{M_{v,i}\}$. Die Menge $\mathcal{M}$ ist dabei eine Teilmenge der Einzelnachrichten, die v in den k Instanzen sendet. Die Wahrscheinlichkeit für eine Nachricht $M_{v,i}$ in Runde r von v gesendet zu werden hängt nur von der Wahl des Delays δ_i ab. Da die Delays unabhängig voneinander gewählt wurden, ist die Anzahl an Nachrichten aus $\mathcal{M}$, die in Runde r von v gesendet werden binomialverteilt – der Prozess kann also durch ein Bernoulli-Experiment modelliert werden. Wie oben analysiert, ist die Wahrscheinlichkeit, dass eine einzelne Nachricht aus $\mathcal{M}$ genau in Runde r von Knoten v gesendet

wird, höchstens $\frac{1}{k}$. Für die Wahrscheinlichkeit, dass alle Nachrichten aus $\mathcal{M}$ genau in Runde r von Knoten v gesendet werden, gilt somit:

$$\Pr[\text{„alle Nachrichten aus } \mathcal{M} \text{ in Runde } r \text{ von } v \text{ gesendet“}] \leq \left(\frac{1}{k}\right)^{|\mathcal{M}|} \tag{6.1}$$

Wir möchten nun argumentieren, dass es für eine Menge von Nachrichten $\mathcal{M}$ der Größe $|\mathcal{M}| = \Omega(\log n)$ sehr unwahrscheinlich ist, dass alle Nachrichten aus $\mathcal{M}$ in irgendeiner Runde r von irgendeinem Knoten v gleichzeitig gesendet werden sollen. Allgemein gilt, dass es höchstens $\binom{k}{\ell}$ Nachrichtenmengen $\mathcal{M} \subseteq \bigcup_{i=1}^{k}\{M_{v,i}\}$ der Größe $|\mathcal{M}| = \ell$ gibt. Dies liegt daran, dass es prinzipiell $\binom{k}{\ell}$ verschiedene Möglichkeiten gibt, aus einer Menge der Größe k eine Teilmenge der Größe ℓ auszuwählen.

Sei nun $\ell' = \max\{(c+4)\log n, 2e\}$, wobei e die Eulersche Zahl ist, und sei $S_{r,v}$ folgendes Ereignis: es gibt eine Teilmenge $\mathcal{M} \subseteq \bigcup_{i=1}^{k}\{M_{v,i}\}$ der Größe $|\mathcal{M}| \geq \ell'$ so dass im Random-Delay Algorithmus alle Nachrichten aus $\mathcal{M}$ in Runde r gleichzeitig von Knoten v gesendet werden. Nun wenden wir (6.1) mit einer Union Bound über alle möglichen Teilmengen der Größen ℓ' bis k an und erhalten:

$$\Pr[S_{r,v}] \leq \sum_{\ell' \leq \ell \leq k} \binom{k}{\ell} \cdot \left(\frac{1}{k}\right)^{\ell} .$$

Durch Anwendung der allgemeingültigen Ungleichung $\binom{a}{b} \leq \left(\frac{ea}{b}\right)^{b}$ und der aus der Definition von ℓ' resultierenden Ungleichungen $\ell' \geq (c+4)\log n$ und $\ell' \geq 2e$ können wir die Wahrscheinlichkeit $\Pr[S_{r,v}]$ folgendermaßen nach oben beschränken:

$$\Pr[S_{r,v}] \leq \sum_{\ell' \leq \ell \leq k} \left(\frac{ek}{\ell}\right)^{\ell} \left(\frac{1}{k}\right)^{\ell} = \sum_{\ell' \leq \ell \leq k} \left(\frac{e}{\ell}\right)^{\ell} \leq k \left(\frac{1}{2}\right)^{(c+4)\log n} = k \cdot \frac{1}{n^{c+4}} \leq \frac{1}{n^{c+3}} .$$

Mit einer erneuten Anwendung der Union Bound erhalten wir

$$\Pr\left[\bigcup_{1 \leq r \leq 2n, v \in V} S_{r,v}\right] \leq \sum_{1 \leq r \leq 2n, v \in V} \Pr[S_{r,v}] \leq 2n^2 \cdot \frac{1}{n^{c+3}} \leq \frac{1}{n^{c}} .$$

Das Gegenereignis zu $\bigcup_{1 \leq r \leq 2n, v \in V} S_{r,v}$ ist, dass für jede Runde r und jeden Knoten v die Menge der in Runde r von Knoten v gesendeten Nachrichten höchstens Größe $\ell' = O(\log n)$ hat, und es hat mit der obigen Abschätzung Wahrscheinlichkeit mindestens $1 - \frac{1}{n^c}$. □

6.2.5 Zusammenfassung Random Delay

Mit einer leichten Modifikation des Beweises kann man die Random Delay Technik auch für Algorithmen anwenden, bei denen jeder Knoten mehr also einmal sendet. Diese Verallgemeinerung liefert folgende Garantie.

Lemma 6.5. *Sei $\mathcal{A}$ ein Algorithmus im CONGEST Modell, der $R(n) = n^{O(1)}$ Runden benötigt und in dem jeder Knoten in insgesamt $M(n) = n^{O(1)}$ Runden Nachrichten sendet. Dann können $k = n^{O(1)}$ Instanzen von $\mathcal{A}$ mit hoher Wahrscheinlichkeit in $O((R(n)+kM(n))\log n)$ Runden ausgeführt werden.*

In diesem Lemma steht $n^{O(1)}$ für ein beliebiges Polynom in n.

6.3 Gewichtete Graphen

6.3.1 Modell

Für gewichtete Graphen gibt es einige Modellierungsaspekte, die zuerst bedacht werden sollten. Jeder Kante $e = (u, v)$ wird ein Gewicht $w(u, v)$ zugeordnet, das initial sowohl dem Knoten u als auch dem Knoten v bekannt ist. im Folgenden sind die Kantengewichte immer positive, ganze Zahlen von 1 bis W. Wir nehmen der Einfachheit halber an, dass W polynomiell in n ist, also $W = n^{O(1)}$. Dadurch kann beispielsweise die Summe von bis zu n Kantengewichten in einer Nachricht der Größe $O(\log n)$ gesendet werden. Wir gehen davon aus, dass die Kantengewichte Kosten repräsentieren und keine Delays; die direkte Kommunikation mit Nachbarn dauert weiterhin nur eine Runde. Mit D bezeichnen wir weiterhin den Durchmesser des *ungewichteten* Netzwerks. Für Distanzberechnungen in gewichteten Graphen ist außerdem folgende Definition oft nützlich.

Definition 6.6. *Für jedes Paar von Knoten u und v und jedes $h \geq 0$ bezeichnet die h-*Distanz $\mathrm{dist}^h(u, v)$ *die Länge des kürzesten Wegs von u nach v mit höchstens h Kanten.*

6.3.2 Bellman-Ford Algorithmus

Im Folgenden verwenden wir eine Variante des Bellman-Ford Algorithmus zur SSSP-Berechnung. Dieser Algorithmus arbeitet folgendermaßen:

Initialisierung in Runde 1:

$$\delta_1(v) = \begin{cases} 0 & \text{falls } v = s \\ \infty & \text{andernfalls} \end{cases}$$

Update in Runde $r \geq 2$:

$$\delta_r(v) = \min_{(u,v)\in E} (\delta_{r-1}(u) + w(u, v))$$

Die Korrektheit und die Laufzeit dieses Algorithmus ergeben sich aus folgender Invariante.

Lemma 6.7 (Invariante)**.** *Nach Runde r entspricht $\delta_r(v)$ für jeden Knoten v der Länge des kürzesten Wegs von s nach v mit höchstens $r - 1$ Kanten, also $\delta_r(v) = \mathrm{dist}^{r-1}(s, v)$.*

Wir überspringen den Beweis dieser Invariante, da er nur Standardargumente benötigt, die wir bereits an mehreren Stellen verwendet haben. Die Korrektheit und Laufzeit des Algorithmus ergibt sich aus der Invariante, da der absolut kürzeste Weg immer höchstens $n - 1$ Kanten hat. Wir erhalten also folgendes Ergebnis.

Theorem 6.8. *Im CONGEST Modell kann das SSSP-Problem für gewichtete Graphen in $O(n)$ Runden gelöst werden.*

Wir betrachten ein Beispiel für die Durchführung des Bellman-Ford Algorithmus: Im Netzwerk von Abb. 6.2 kennt am Anfang der Startknoten die Distanz 0 zu sich selbst.

In der ersten Runde sendet der Startknoten seine Distanz an alle Nachbarknoten. Diese Distanz wird von den Nachbarknoten in Runde 2 empfangen und der aktualisierte Wert jedes Nachbarknotens entspricht dem Gewicht seiner Kante zum Startknoten, siehe Abb. 6.3.

Wenn diese Update-Schritte solange wiederholt werden, bis keine Änderungen mehr auftreten, dann erhält man das in Abb. 6.4 dargestellte Ergebnis. Jeder Knoten kennt dann seine Distanz zu s.

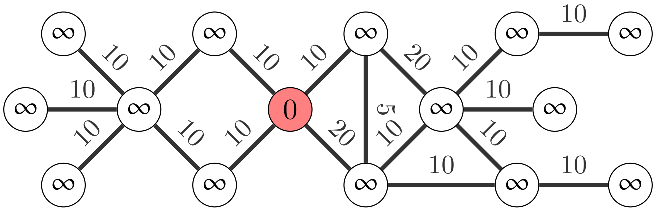


Abbildung 6.2: SSSP-Problem für gewichtete Graphen

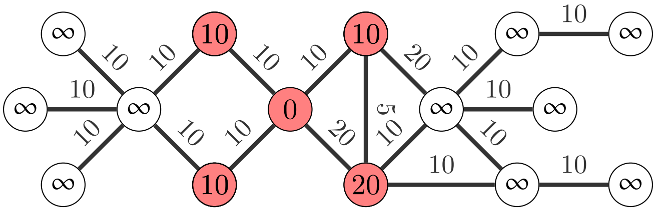


Abbildung 6.3: SSSP-Problem für gewichtete Graphen

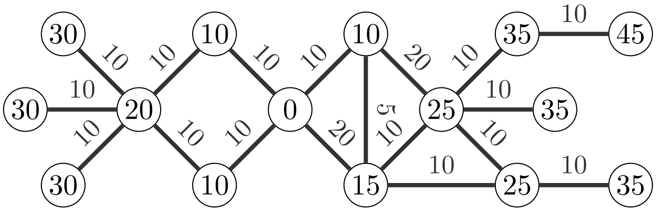


Abbildung 6.4: SSSP-Problem für gewichtete Graphen

6.3.3 All-Pairs Shortest Paths

Wir möchten das APSP Problem in gewichteten Graphen mit der gleichen Idee wie in ungewichteten Graphen lösen, nämlich in dem wir mit der Random-Delay Technik für jeden Knoten des Netzwerks eine Instanz eines SSSP-Algorithmus ausführen. Im Bellman-Ford Algorithmus sendet aber jeder Knoten in jeder Runde. Somit würde man mit der Random-Delay Technik nur eine Laufzeitschranke $O(n^2 \log n)$ erhalten. Das ist natürlich schlechter als $O(n^2)$, die Schranke die man bei sequentieller Ausführung der n Instanzen von Bellman-Ford erhält. Das Ziel ist nun, einen SSSP-Algorithmus mit einer geringerer Anzahl an Nachrichten pro Knoten zu entwerfen. Dies kann mit einer Art „gewichteter" Breitensuche erreicht werden. Dabei simulieren wird den gewichteten Graph als ungewichteten Graph, indem jede Kante konzeptuell durch einen Pfad entsprechender Länge ersetzt wird. Diese Idee kann mit folgendem Algorithmus umgesetzt werden:

---

1 s hat Distanz *$ 0 $* zu sich selbst sendet für jeden Nachbarknoten v in Runde $w(s,v)$ die Distanzinformation $w(s,v)$ an v (Empfang durch v in Runde $w(u,v)+1$)

2

3 Sobald für Knoten $u \neq s$ das erste Mal eine Distanz-Information δ empfangen wurde :

4 Sei r die aktuelle Runde

5 Für jeden Nachbarknoten v: Sende Distanzinformation $\delta + w(u,v)$ an v in Runde $r + w(u,v) - 1$ (Empfang durch v in Runde $r + w(u,v)$)

Analog zur bereits bekannten Breitensuche lässt sich für die „gewichtete" Breitensuche folgende Invariante aufstellen, deren Beweis wir wiederum überspringen.

Lemma 6.9 (Invariante)**.** *Für jeden Knoten v gilt: Die erste Distanznachricht wird in Runde* $\operatorname{dist}(s,v)+1$ *empfangen; sie hat den Inhalt* $\operatorname{dist}(s,v)$*.*

Der Offset +1 ergibt sich dadurch, dass die Rundenzählung bei 1 beginnt, aber die kleinste Distanz, nämlich $\operatorname{dist}(s,s)$ den Wert 0 hat.

Durch die Simulation der Breitensuche erhalten wir also einen SSSP-Algorithmus, in dem jeder Knoten nur einmal Nachrichten sendet, und dessen Laufzeit von der gewünschten Tiefe des Breitensuchbaums abhängt.

Lemma 6.10. *Für jedes $T \geq 0$ kann in $T+1$ Runden für jeden Knoten v ein Wert $\delta(v)$ berechnet werden, so dass $\delta(v) = \operatorname{dist}(s,v)$ falls $\operatorname{dist}(s,v) \leq T$ und $\delta(v) = \infty$ andernfalls, wobei jeder Knoten in höchstens einer Runde Nachrichten an seine Nachbarn sendet.*

Zur Berechnung der Distanz $\operatorname{dist}(s,v)$ für jeden Knoten v sind also $O(nW)$ Runden ausreichend, weil jede Kante ein Gewicht von höchstens W hat und jeder kürzeste Weg aus höchstens $n-1$ Kanten besteht. Dadurch ergibt sich ein „pseudopolynomieller" SSSP-Algorithmus, dessen Laufzeit vom Wert von W abhängt (und nicht von $\log W$, wie es von „polynomiellen" Algorithmen gefordert wird.)

Lemma 6.11. *Im CONGEST Modell kann das SSSP-Problem für gewichtete Graphen in $O(nW)$ Runden gelöst werden, wobei jeder Knoten höchstens einmal Nachrichten sendet.*

Jetzt nutzen wir die Random-Delay Technik und mit der Anwendung von Lemma 6.5 ergibt sich folgendes Theorem zur Lösung des APSP-Problems.

Theorem 6.12. *Im CONGEST Modell gibt es einen Monte-Carlo Algorithmus, der das APSP-Problem für gewichtete Graphen in $O(nW \log n)$ Runden mit hoher Wahrscheinlichkeit löst.*

6.4 Approximationsalgorithmus für Single-Source Shortest Paths

Im Folgenden werden wir das Single-Source Shortest Path Problem für gewichtete Graphen näher betrachten. Wir haben in Abschnitt 6.3.2 bereits einen einfachen Algorithmus kennengelernt, der das Problem in $O(n)$ Runden löst. Der derzeit schnellste Algorithmus für dieses Problem [CM20] benötigt mit hoher Wahrscheinlichkeit $O((\sqrt{n}D^{1/4}+D)\cdot\log^{O(1)} n)$ viele Runden. Es lässt sich ebenfalls zeigen, dass Algorithmen für das SSSP-Problem nicht beliebig gut sein können. Im Allgemeinen werden $\Omega(\sqrt{n}/\log n + D)$ Runden benötigt [PR00]. Eine annähernd „enge“ obere bzw. untere Schranke für dieses Problem zu finden ist ein großes offenes Problem.

Eine „enge“ obere/untere Schranke ist bisher nur für Approximationsalgorithmen bekannt. Wir sagen, dass ein Algorithmus eine (multiplikative) α-Approximation für das SSSP-Problem berechnet, wenn am Ende jeder Knoten v eine Distanzschätzung $\delta(s,v)$ hat, für die gilt:

$$\operatorname{dist}(s,v) \leq \delta(s,v) \leq \alpha \operatorname{dist}(s,v)\,.$$

Einerseits werden im Allgemeinen werden $\Omega(\sqrt{n/(\alpha \log n)} + D)$ Runden benötigt, um eine α-Approximation für das SSSP Problem zu berechnen [Elk06]. Andererseits kann eine $(1+\epsilon)$-Approximation für das SSSP Problem mit hoher Wahrscheinlichkeit in $O((\sqrt{n}+D)\cdot\log^{O(1)}(n)/\epsilon^{O(1)})$ Runden berechnet werden [BFKL21].

Wir werden nun einen etwas weniger effizienten Approximationsalgorithmus entwickeln, der $O(n^{2/3}\log^2(n)/\epsilon + D)$ Runden benötigt und mit hoher Wahrscheinlichkeit eine $(1+\epsilon)$-Approximation berechnet, wobei $0 < \epsilon \leq 1$ ein Parameter für die gewünschte Approximationsgüte ist. Für diesen Approximationsalgorithmus verwenden wir zwei hilfreiche Lemmas.

Lemma 6.13. *Sei h ein Parameter und sei Z (Zentren) eine Menge zu der jeder Knoten unabhängig mit Wahrscheinlichkeit $p = ((c+2)\ln n)/h$ hinzugefügt wurde. Dann gilt mit Wahrscheinlichkeit mindestens $1 - \frac{1}{n^c}$: Für jedes Knotenpaar u und v gibt es einen kürzesten Weg von u nach v, der weniger als h Kanten hat oder innerhalb der ersten h vom Startknoten verschiedenen Knoten ein Zentrum enthält.*

Lemma 6.14. *Sei Z eine Teilmenge von Knoten und h ein Parameter. In $O((|Z|+h)\log(nW)\log(n)/\epsilon)$ Runden kann für jedes $x \in Z$ und jedes $v \in V$ eine approximative Distanz $\widetilde{\operatorname{dist}}(x,v)$ berechnet werden (die v am Ende kennt), für die mit hoher Wahrscheinlichkeit gilt:*

$$\operatorname{dist}(x,v) \leq \widetilde{\operatorname{dist}}(x,v) \leq (1+\epsilon)\operatorname{dist}^h(x,v)\,.$$

Wir beweisen diese Lemmas erst später und fokussieren uns im Folgenden darauf einen Approximationsalgorithmus unter Zuhilfenahme der beiden Lemmas zu entwickeln. Die Grundidee des Algorithmus ist, die SSSP Berechnung auf ein relativ kleines Overlay-Netzwerk H mit den Zentren als Knoten zurückzuführen, also ein Netzwerk der Gestalt $H = (Z, Z \times Z)$ mit Gewichten $w_H(x,y) = \widetilde{\operatorname{dist}}(x,y)$ für alle $x, y \in Z$, wobei $\widetilde{\operatorname{dist}}(x,y)$ eine noch näher zu bestimmende Distanzapproximation bezeichnet. Dabei ist zu beachten, dass das Overlay-Netzwerk nur konzeptuell erstellt wird; die Kanten zwischen den Zentren werden *nicht* in das Kommunikationsnetzwerk eingefügt.

Wir formulieren unseren SSSP-Approximationsalgorithmus zunächst für einen beliebig wählbaren Parameter h und bestimmen später eine gute Wahl für diesen Parameter. Der Algorithmus besteht aus fünf Schritten:

1. Intern für jeden Knoten v: Füge v mit Wahrscheinlichkeit $p = ((c+2)\ln n)/h$ zu Z hinzu, füge s immer zu Z hinzu

2. Berechne, für alle Paare $x \in Z$, $v \in V$, approximative Distanzen $\widetilde{\text{dist}}(x, v)$, für die gilt: $\text{dist}(x, v) \leq \widetilde{\text{dist}}(x, v) \leq (1 + \epsilon)\,\text{dist}^h(x, v)$
3. Mache $\widetilde{\text{dist}}(x, y)$ für alle Paare $x, y \in Z$ im gesamten Netzwerk durch Up- und Downcasts bekannt
4. Intern für jeden Knoten v: Konstruiere Graph $H_v = (Z \cup \{v\}, (Z \cup \{v\})^2)$ mit Gewicht $w_{H_v}(x, y) = \widetilde{\text{dist}}(x, y)$ für jede Kante (x, y)
5. Intern für jeden Knoten v: Berechne $\delta(s, v) := \text{dist}_{H_v}(s, v)$ als Ergebnis

6.4.1 Laufzeit

Für die Laufzeitanalyse beobachten wir zunächst, dass die Knoten des Netzwerks für die Schritte 1, 2 und 5 gar nicht kommunizieren müssen. Schritt 2 benötigt laut Lemma 6.14 $O((|Z| + h)\log(nW)\log(n)/\epsilon)$ Runden und Schritt 3 benötigt $O(|Z|^2 + D)$ Runden durch eine Kombination aus multiplem Upcast (siehe Abschnitt 2.6.1) und multiplen Downcast (siehe Abschnitt 2.5.1). Die Anzahl an Zentren $|Z|$ ist in Erwartung $O(\frac{n}{h}\log n)$ und mittels der Anwendung der Chernoff-Bound (Theorem B.13) lässt sich argumentieren, dass $|Z|$ von diesem Erwartungswert mit hoher Wahrscheinlichkeit nur um einen konstanten Faktor abweicht (sofern $h \leq \frac{n}{c}$) für eine genügend große Konstante c gilt). Daher benötigt unser Algorithmus mit hoher Wahrscheinlichkeit $O((h + \frac{n}{h}\log n) \cdot \log(nW)\log(n)/\epsilon + \frac{n^2}{h^2}(\log n)^2 + D)$ viele Runden. Wir setzen für unseren Algorithmus $h = n^{2/3}$ (womit $h \leq \frac{n}{c}$ für n groß genug gilt) und erhalten somit eine Rundenkomplexität von $O((n^{2/3} + n^{1/3}\log n) \cdot \log(nW)\log(n)/\epsilon + n^{2/3}(\log n)^2 + D)$. Dies lässt sich vereinfacht darstellen als $O(n^{2/3}\log(nW)\log(n)/\epsilon + D)$.

6.4.2 Korrektheit

Wir zeigen nun, dass das vom Algorithmus berechnete Ergebnis $\delta(s, v) := \text{dist}_{H_v}(s, v)$ (für jeden Knoten $v \in V$) korrekt ist, also die gewünschte Approximationsgüte liefert.

Lemma 6.15. *Mit hoher Wahrscheinlichkeit gilt:* $\text{dist}_G(s, v) \leq \text{dist}_{H_v}(s, v) \leq (1 + \epsilon)\,\text{dist}_G(s, v)$ *für jeden Knoten* $v \in V$.

Beweis. Sei $v \in V$ beliebig. Da jeder Knoten von H_v auch in G vorkommt und jedes Gewicht einer Kante (x, y) in H_v die Ungleichung $w_{H_v}(x, y) := \widetilde{\text{dist}}(x, y) \geq \text{dist}_G(x, y)$ gilt (das Kantengewicht die Distanz in G also nicht unterschätzt) gilt klarerweise die Ungleichung $\text{dist}_{H_v}(s, v) \geq \text{dist}_G(s, v)$ für alle $v \in V$. Es bleibt zu zeigen, dass $\text{dist}_{H_v}(s, v) \leq (1 + \epsilon)\,\text{dist}_G(s, v)$ für alle $v \in V$.

Sei π der kürzeste Weg von s nach v in G. Durch die wiederholte Anwendung von Lemma 6.13 kann π so gewählt werden, dass der Pfad mit hoher Wahrscheinlichkeit jeweils nach höchstens h Kanten ein Zentrum enthält (siehe Abb. 6.5). Sei nun $x_1, x_2 \ldots, x_k$ die Sequenz der Zentren auf den inneren Knoten von π und seien $x_0 := s$ und $x_{k+1} := v$. Da $x_0, \ldots, x_k \in Z$ gilt, enthält H_v Kante (x_i, x_{i+1}) für alle $0 \leq i \leq k - 1$. Ebenso enthält H_v Kante (x_k, x_{k+1}). Daher gibt es in H_v einen Pfad $\pi' = (x_0, x_1, \ldots, x_k, x_{k+1})$ von $s = x_0$ nach $v = x_{k+1}$.

Wir können die Gewichte der Kanten auf π' folgendermaßen nach oben beschränken. Für alle $0 \leq i \leq k$ hat der Teilpfad von π von x_i nach x_{i+1} höchstens h Kanten enthält. Da jeder Teilpfad eines kürzesten Wegs ebenfalls ein kürzester Weg ist, gilt somit

$$\text{dist}^h_G(x_i, x_{i+1}) = \text{dist}_G(x_i, x_{i+1}) \tag{6.2}$$

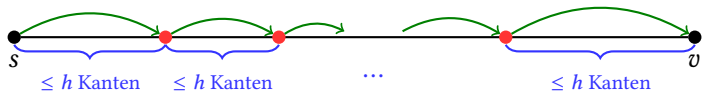


Abbildung 6.5: Unterteilung des kürzesten Wegs von s nach v so dass jeweils nach höchstens h Kanten ein Zentrum (rot) erreicht wird.

für alle $0 \leq i \leq k$. Die entsprechende Kante (x_i, x_{i+1}) in H_v hat laut Algorithmus (Schritt 4) das Gewicht $w_{H_v}(x_i, x_{i+1}) = \widetilde{\text{dist}}(x_i, x_{i+1})$. Laut Lemma 6.14 gilt $\widetilde{\text{dist}}(x, v) \leq (1+\epsilon)\,\text{dist}^h(x, v)$. Wegen (6.2) gilt somit insgesamt

$$w_{H_v}(x_i, x_{i+1}) \leq (1+\epsilon)\,\text{dist}(x, v)\,. \tag{6.3}$$

Nun kann $\text{dist}_{H_v}(s, v)$ auf folgende Weise nach oben beschränkt werden: Zunächst gilt dass der kürzeste Weg von s nach v in H_v aufgrund seiner Eigenschaft als *kürzester* Weg niemals länger ist als der von uns betrachtete Pfad π', also $\text{dist}_{H_v}(s, v) \leq w_{H_v}(\pi') := \sum_{i=0}^{k} w_{H_v}(x_i, x_{i+1})$. Wegen 6.3 gilt $\sum_{i=0}^{k} w_{H_v}(x_i, x_{i+1}) \leq (1+\epsilon)\sum_{i=0}^{k} \text{dist}_G(x, x_{i+1})$. Aufgrund der Definition von $x_0, x_1, \ldots, x_{k+1}$ als Knoten auf einem kürzesten Weg von $s = x_0$ nach $v = x_{k+1}$ in G gilt $\text{dist}_G(s, v) = \sum_{i=0}^{k} \text{dist}_G(x, x_{i+1})$ (der kürzeste Weg ist insgesamt so lang wie die Summe seiner Teilstücke). Insgesamt gilt also $\text{dist}_{H_v}(s, v) \leq (1+\epsilon)\,\text{dist}_G(s, v)$. □

6.4.3 Beweis von Lemma 6.13

Wir fixieren in unserem Beweis für jedes Paar von Knoten $u, v \in V$ *einen* der kürzesten Wege von u nach v. Sei Z die Menge der zufällig gewählten Zentren und seien $\pi_1, \pi_2, \ldots \pi_\ell$ (mit $\ell \leq n^2$) jene paarweisen kürzesten Wege mit mindestens h Kanten. Wir möchten zeigen dass mit hoher Wahrscheinlichkeit jeder dieser Pfade – abgesehen von seinem jeweiligen Startknoten – ein Zentrum enthält.

Für jeden solchen Pfad π_i ist die Anzahl an Zentren innerhalb seiner ersten h vom Startknoten verschiedenen Knoten binomialverteilt mit Parametern h und $p = ((c+2)\ln n)/h$. Sei A_i jenes Ereignis das eintritt, wenn π_i innerhalb seiner ersten h vom Startknoten verschiedenen Knoten ein Zentrum enthält. Das entsprechende Gegenereignis $\bar{A}_i$ hat die Wahrscheinlichkeit

$$\Pr[\bar{A}_i] = (1-p)^h = (1-p)^{((c+2)\ln n)/p} = \left((1-p)^{\frac{1}{p}}\right)^{\ln n^{c+2}}.$$

Unter Verwendung der Ungleichung (A.18) $\left(\left(1-\frac{1}{x}\right)^x \leq \frac{1}{e}\right)$ erhalten wir

$$\Pr[\bar{A}_j] \leq \left(\frac{1}{e}\right)^{\ln n^{c+2}} = \frac{1}{n^{c+2}}\,.$$

Wir können nun die Wahrscheinlichkeit, dass jeder der Pfade $\pi_1, \pi_2, \ldots \pi_\ell$ innerhalb seiner ersten h vom Startknoten verschiedenen Knoten ein Zentrum enthält durch das Gegenereignis abschätzen:

$$\Pr\left[\bigcap_{i=1}^{\ell} A_i\right] = 1 - \Pr\left[\bigcup_{i=1}^{\ell} \bar{A}_i\right] \geq 1 - \sum_{i=1}^{\ell} \Pr\left[\bar{A}_i\right] \geq 1 - \ell \cdot \frac{1}{n^{c+2}} \geq 1 - \frac{1}{n^c}\,.$$

6.4.4 Beweis von Lemma 6.14

Wir werden uns auf folgendes Lemma konzentrieren.

Lemma 6.16. *Eine Distanzapproximation* $\widetilde{\mathrm{dist}}(s,\cdot)$*, für die*

$$\mathrm{dist}(s,v) \leq \widetilde{\mathrm{dist}}(s,v) \leq (1+\epsilon)\,\mathrm{dist}^h(s,v)$$

für jeden Knoten v *gilt, kann in* $O(h \log(hW)/\epsilon)$ *vielen Runden berechnet werden. Dabei sendet jeder Knoten in* $O(\log(hW))$ *vielen Runden.*

Lemma 6.14 folgt dann aus Lemma 6.16 durch Anwendung der Random-Delay Technik aus Lemma 6.5.

Für unseren Beweis von Lemma 6.16 verwenden wir nun wiederholt einen Hilfsalgorithmus mit folgenden Garantien.

Lemma 6.17 ([CM20])**.** *Seien* $M > 0, T \geq 0$ *und sei* G *ein Graph mit Kantengewichten* $\geq M$*. Dann können von einem Startknoten* s *aus alle Knoten* v *mit* $\mathrm{dist}_G(s,v) \leq T$ *sowie deren Distanz* $\mathrm{dist}_G(s,v)$ *in* $O(T/M)$ *Runden berechnet werden, wobei jeder Knoten höchstens in einer Runde Nachrichten sendet.*

Der Beweis dieses Lemmas ist eine interessante Übungsaufgabe.

Im Folgenden analysieren wir für alle $i \geq 0$ des Ergebnis dieses Hilfsalgorithmus jeweils für die Parameter $M_i := \frac{\epsilon 2^i}{h}$ und $T_i := (1+\epsilon)\cdot 2^{i+1}$ im Graph $G_i^{\uparrow}$, in dem alle kleineren Gewichte aus G auf den Wert M_i aufgerundet werden, also

$$w_{G_i^{\uparrow}}(u,v) := \begin{cases} M_i & \text{falls } w_G(u,v) < M_i \\ w(u,v) & \text{andernfalls} \end{cases}$$

für alle Kanten (u,v) aus G.

Lemma 6.18. *Für alle Paare von Knoten* $x, y \in V$ *gilt:*

1. $\mathrm{dist}_{G_i^{\uparrow}}(x,y) \geq \mathrm{dist}_G(x,y)$
2. *Falls* $\mathrm{dist}_G^h(x,y) \geq 2^i$*:* $\mathrm{dist}_{G_i^{\uparrow}}(x,y) \leq (1+\epsilon)\,\mathrm{dist}_G^h(x,y)$
3. *Falls* $2^i \leq \mathrm{dist}_G^h(x,y) \leq 2^{i+1}$*:* $\mathrm{dist}_{G_i^{\uparrow}}(x,y) \leq T_i$

Beweis. Die erste Eigenschaft gilt offensichtlich, weil die Gewichte in $G_i^{\uparrow}$ mindestens so groß sind wie in G. Die letzte Eigenschaft folgt aus der zweiten Eigenschaft und aus der Definition von T_i: Wegen $\mathrm{dist}_G^h(x,y) \geq 2^i$ gilt $\mathrm{dist}_{G_i^{\uparrow}}(x,y) \leq (1+\epsilon)\,\mathrm{dist}_G^h(x,y)$ und wegen $\mathrm{dist}_G^h(x,y) \leq 2^{i+1}$ gilt $(1+\epsilon)\,\mathrm{dist}_G^h(x,y) \leq (1+\epsilon)2^i = T_i$, also insgesamt: $\mathrm{dist}_{G_i^{\uparrow}}(x,y) \leq T_i$.

Für den Beweis der zweiten Eigenschaft betrachten wir einen kürzesten Weg π von x nach y in G (also mit Gesamtgewicht $w_G(\pi) = \mathrm{dist}_G(x,y)$). Jede Kante in $G_i^{\uparrow}$ hat durch das Aufrunden einen um höchstens M_i höheren Wert als in G. Daher hat der Pfad π, der aus höchstens h Kanten besteht, in $G_i^{\uparrow}$ einen um höchstens $h \cdot M_i$ höheren Wert als in G. Da der Pfad π auch in $G_i^{\uparrow}$ existiert, hat der

kürzeste Weg in $G_i^{\uparrow}$ höchstens das Gewicht von π in $G_i^{\uparrow}$, also $\operatorname{dist}_{G_i^{\uparrow}}(x,y) \leq w_{G_i^{\uparrow}}(\pi)$. Zusammen mit der Definition von M_i und der Ungleichung $2^i \leq \operatorname{dist}_G^h(x,y)$ ergibt sich dadurch

$$\begin{aligned}\operatorname{dist}_{G_i^{\uparrow}}(x,y) \leq w_{G_i^{\uparrow}}(\pi) \leq w_G(\pi) + h \cdot M_i = \operatorname{dist}_G^h(x,y) + \epsilon 2^i &\leq \operatorname{dist}_G^h(x,y) + \epsilon \operatorname{dist}_G^h(x,y \\ &= (1+\epsilon)\operatorname{dist}_G^h(x,y)\,.\end{aligned}$$

□

Wir verwenden nun folgenden Algorithmus, um die h-Distanzen von einem gegebenen Startknoten s zu berechnen:

1. Für jedes $0 \leq i < \lfloor \log(hW) \rfloor$, berechne folgenden Wert $\widetilde{\operatorname{dist}}_i(s,v)$ für jeden Knoten $v \in V$:

$$\widetilde{\operatorname{dist}}_i(s,v) := \begin{cases} \operatorname{dist}_{G_i^{\uparrow}}(s,v) & \text{falls } \operatorname{dist}_{G_i^{\uparrow}}(s,v) \leq T_i \\ \infty & \text{andernfalls.} \end{cases}$$

2. Intern, für jeden Knoten v: Berechne $\widetilde{\operatorname{dist}}(s,v) := \min_{0 \leq i < \lfloor \log(hW) \rfloor} \widetilde{\operatorname{dist}}_i(s,v)$

Dabei erfolgt die Berechnung der Werte $\widetilde{\operatorname{dist}}_i(s,v)$ für jedes $0 \leq i < \lfloor \log(hW) \rfloor$ durch den Algorithmus aus Lemma 6.17 mit Parametern M_i und T_i in $O(T_i/M_i) = O(h/\epsilon)$ vielen Runden. Dadurch ergibt sich eine Gesamtlaufzeit von $O(h \log(hW)/\epsilon)$ vielen Runden.

Die Korrektheit dieses Algorithmus lässt sich folgendermaßen argumentieren: Sei $v \in V$. Für jedes $0 \leq i < \lfloor \log(hW) \rfloor$ können wir zwei Fälle unterscheiden: Falls $\operatorname{dist}_{G_i^{\uparrow}}(s,v) \leq T_i$ gilt, dann ist $\widetilde{\operatorname{dist}}_i(s,v) = \operatorname{dist}_{G_i^{\uparrow}}(s,v)$; da aufgrund der zweiten Eigenschaft aus Lemma 6.18 $\operatorname{dist}_{G_i^{\uparrow}}(s,v) \geq \operatorname{dist}_G(s,v)$ gilt, erhalten wir $\widetilde{\operatorname{dist}}_i(s,v) \geq \operatorname{dist}_G(s,v)$. Andernfalls ist $\widetilde{\operatorname{dist}}_i(s,v) = \infty$ und $\widetilde{\operatorname{dist}}_i(s,v) \geq \operatorname{dist}_G(s,v)$ gilt trivialerweise. Insgesamt gilt also immer $\widetilde{\operatorname{dist}}_i(s,v) \geq \operatorname{dist}_G(s,v)$. Wenn diese Ungleichung für jeden der Werte $\widetilde{\operatorname{dist}}_i(s,v)$ gilt, dann gilt sie insbesondere auch für das Minimum all dieser Werte, also $\widetilde{\operatorname{dist}}(s,v) = \min_{0 \leq i < \log(hW)} \widetilde{\operatorname{dist}}_i(s,v) \geq \operatorname{dist}_G(s,v)$.

Wähle nun j zwischen 0 und $\lfloor \log(hW) \rfloor$ so dass $2^j \leq \operatorname{dist}_G^h(s,v) < 2^{j+1}$ gilt; da $\operatorname{dist}_G^h(s,v) \leq hW$ gilt, gibt es immer ein eindeutiges j, das diese Bedingung erfüllt. Aufgrund der dritten Eigenschaft aus Lemma 6.18 gilt $\operatorname{dist}_{G_j^{\uparrow}}(x,v) \leq T_j$ und somit $\widetilde{\operatorname{dist}}_j(s,v) = \operatorname{dist}_{G_j^{\uparrow}}(s,v)$. Aufgrund der zweiten Eigenschaft aus Lemma 6.18 gilt $\operatorname{dist}_{G_j^{\uparrow}}(s,v) \leq (1+\epsilon)\operatorname{dist}_G^h(x,v)$ und somit $\widetilde{\operatorname{dist}}_j(s,v) \leq (1+\epsilon)\operatorname{dist}_G^h(x,v)$. Insgesamt erhalten wir $\widetilde{\operatorname{dist}}(s,v) = \min_{0 \leq i < \lfloor \log(hW) \rfloor} \widetilde{\operatorname{dist}}_i(s,v) \leq \widetilde{\operatorname{dist}}_j(s,v) \leq (1+\epsilon)\operatorname{dist}_G^h(x,v)$.

6.5 Literatur

[BFKL21] Ruben Becker, Sebastian Forster, Andreas Karrenbauer und Christoph Lenzen. "Near-Optimal Approximate Shortest Paths and Transshipment in Distributed and Streaming Models". In: *SIAM Journal on Computing* 50.3 (2021), S. 815–856. DOI: 10.1137/19M1286955 (siehe S. 54).

[CM20] Shiri Chechik und Doron Mukhtar. "Single-Source Shortest Paths in the CONGEST Model with Improved Bound". In: *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC '20)*. 2020, S. 464–473. DOI: 10.1145/3382734.3405729 (siehe S. 54, 57).

[Elk06] Michael Elkin. “An Unconditional Lower Bound on the Time-Approximation Trade-off for the Distributed Minimum Spanning Tree Problem”. In: *SIAM Journal on Computing* 36.2 (2006), S. 433–456. DOI: 10.1137/S0097539704441058 (siehe S. 54).

[FN18] Sebastian Forster und Danupon Nanongkai. “A Faster Distributed Single-Source Shortest Paths Algorithm”. In: *Proceedings of the 59th IEEE Annual Symposium on Foundations of Computer Science (FOCS 2018)*. 2018, S. 686–697. DOI: 10.1109/FOCS.2018.00071.

[LMR94] Frank Thomson Leighton, Bruce M. Maggs und Satish Rao. “Packet Routing and Job-Shop Scheduling in O(Congestion + Dilation) Steps”. In: *Combinatorica* 14.2 (1994), S. 167–186. DOI: 10.1007/BF01215349 (siehe S. 48).

[Nan14] Danupon Nanongkai. “Distributed approximation algorithms for weighted shortest paths”. In: *Proc. of the Symposium on Theory of Computing (STOC)*. 2014, S. 565–573. DOI: 10.1145/2591796.2591850 (siehe S. 48).

[PR00] David Peleg und Vitaly Rubinovich. “A Near-Tight Lower Bound on the Time Complexity of Distributed Minimum-Weight Spanning Tree Construction”. In: *SIAM Journal on Computing* 30.5 (2000), S. 1427–1442. DOI: 10.1137/S0097539700369740 (siehe S. 54).

7 Epidemische Informationsausbreitung

7.1 Einführung

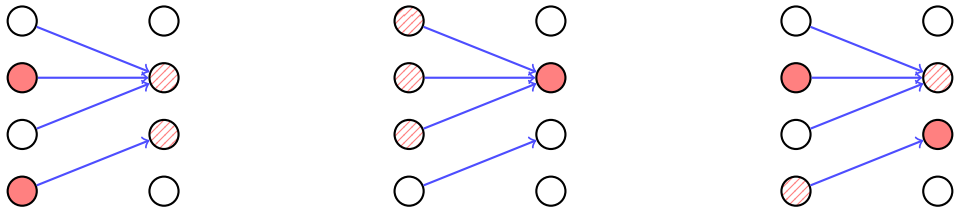
In diesem Abschnitt legen wir den Fokus auf *Point-to-Point* Netzwerke, in denen jeder Knoten pro Runde nur mit jeweils einem anderen Knoten eine Verbindung herstellen kann. Dies modelliert insbesondere Ende-zu-Ende-Verbindungen über niedere Netzwerkschichten.

Ein zentrale Fragestellung ist, wie eine Information effizient an alle Knoten des Netzwerks verbreitet werden kann.

Ein potentielles Anwendungsszenario hierfür sind replizierte Datenbanken in Peer-to-Peer Netzwerken. Hier ist das Ziel den Datenbestand an allen Knoten gleich zu halten. Sobald also ein Knoten eine neue Information erhält soll, diese an alle anderen Knoten verteilt werden.

Wir abstrahieren dieses Setting im sogenannten *Phone-Call* Modell. In diesem Modell modellieren wir das Netzwerk als einen vollständigen Graph mit n Knoten. Die Kommunikation zwischen den Knoten findet in synchronen Runden statt und in jeder Runde kann jeder Knoten einen anderen Knoten anrufen. Wir unterscheiden zwischen drei Kommunikationsformen (siehe Abb. 7.1):

1. *Push*: Der Anrufer informiert den Angerufenen
2. *Pull*: Der Angerufene informiert den Anrufer
3. *Push & Pull*: Kombination von Push und Pull



(a) Push (b) Pull (c) Push & Pull

Abbildung 7.1: Die drei Kommunikationsformen im Phone-Call Modell

Eine naheliegende Idee, um einen Broadcast im Push-Modell durchzuführen, ist, dass der i-te informierte Knoten den i-ten uninformierten Knoten anruft. Um diese Idee durchführen zu können, muss eine allen bekannte konsistente Ordnung der Knoten existieren. Abb. 7.2 visualisiert eine Runde des entsprechenden Algorithmus.

In diesem Algorithmus ruft jeder informierte Konten einen uninformierten Knoten an und jeder uninformierte Knoten wird von höchstens einem uninformierten Knoten angerufen. Daher verdoppelt sich die Anzahl der informierten Knoten mit jeder Runde und es werden somit $O(\log n)$ Runden benötigt. Es ist klar, dass im Push-Modell eine Verdopplung der Anzahl informierte Knoten mit jeder Runde der günstigste Fall ist. Auch die Nachrichtenkomplexität ist mit $O(n)$ optimal, das jeder Knoten genau einmal angerufen wird.

Ein Nachteil dieses Algorithmus ist seine mangelnde Fehlertoleranz: Wenn eine Informationsweitergabe scheitert, kann dies dazu führen dass sehr viele Knoten nicht informiert werden. Ein weiterer Nachteil ist außerdem, dass jedem Knoten alle anderen Knoten des Netzwerks explizit bekannt sein müssen. Konzeptuell ist außerdem unklar, wie das Schema auf das Pull-Modell übertragen werden könnte.

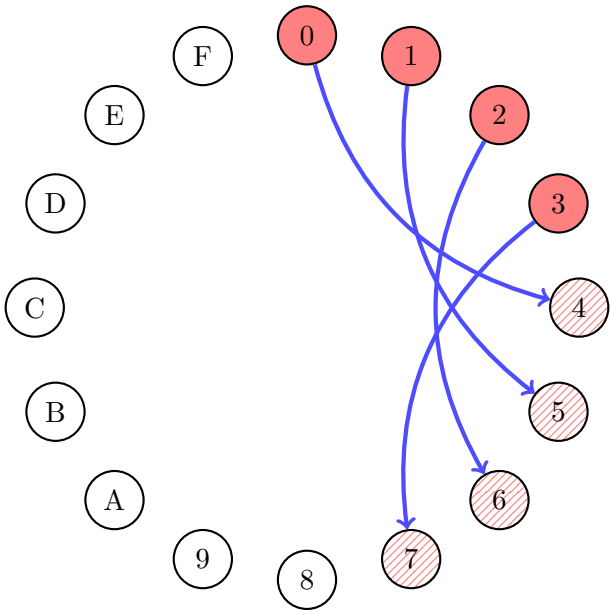


Abbildung 7.2: Beispiel einer Runde eines deterministischen Broadcast-Algorithmus für das Push-Modell. Die Knoten 0, 1, 2 und 3 sind bereits informiert, die Knoten 4, 5, 6, und 7 erhalten die Information in der aktuellen Runde.

Als Ersatz für diese Art deterministischer Broadcast-Algorithmen wurde in der Literatur das Konzept von *Epicast* vorgeschlagen [DGHI+88]. Die Idee dabei ist, dass sich Information im Netzwerk durch Zufallsauswahl epidemisch ausbreiten soll, wie ein Virus oder ein Gerücht. Der Vorteil dieser Vorgehensweise ist, dass sie zu einfachen, robusten und auch schnellen Algorithmen führt. Der Nachteil daran ist ein gewisser Nachrichtenoverhead.

Zur Analyse von Epicasts führen wir das *Random-Phone-Call* Modell ein, welches das Phone-Call Modell dahingehend modifiziert, dass in jeder Runde jeder Knoten einen *uniform zufällig* gewählten Knoten anruft (anstelle eines beliebig gewählten Knotens). Der Einfachheit halber erlauben wir den Fall, dass ein Knoten sich selbst aufruft. In jeder Runde beträgt also die Wahrscheinlichkeit, dass ein Knoten u einen Knoten v anruft $\frac{1}{n}$. Abb. zeigt ein Beispiel einer Runde im Random-Push Modell. Die zufällige Wahl des angerufenen Knotens kann prinzipiell ohne explizite Kenntnis der anderen Knoten implementiert werden. Zufällige Adressierung wird beispielsweise in vielen Peer-to-Peer-Netzwerken unterstützt.

Wie wir im Folgenden sehen werden, benötigen Epicasts im Random-Phone-Call Modell mit hoher Wahrscheinlichkeit $O(\log n)$ Runden. Da sich eine konkrete obere Schranke für die Dauer eines Epicasts angeben lässt, könnte in einer Anwendung mit einem zusätzlichen Zeitstempel überprüft werden, ob mit hoher Wahrscheinlichkeit bereits alle Knoten informiert sind und Ausbreitung der Information gestoppt werden kann. Da es in der Analyse niemals eine Rolle spielt, *welche* Knoten bereits informiert sind, sondern nur *wieviele*, ergibt sich automatisch eine gewisse Robustheit des Prozesses gegenüber Übertragungsfehlern: Pessimistisch geschätzt erhöht sich pro fehlerbehafteter Runde die Gesamtzahl benötigter Runden um eins. Unabhängig von konkreten Anwendungsszenarien modelliert das Random-Phone-Call Modell einen interessanten mathematischen Prozess, der beispielsweise auch als einfaches Modell für die Ausbreitung von Infektionen dienen kann. Es sei angemerkt, dass die Analyse sehr einfach auf ein Modell mit Weitergabewahrscheinlichkeiten erweitert werden kann und dass prinzipiell auch andere Netzwerkstrukturen als der vollständige

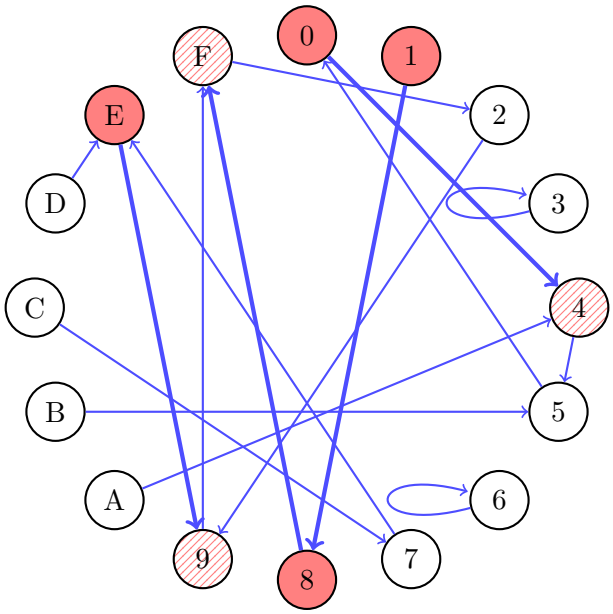


Abbildung 7.3: Beispiel einer Runde im Random-Push-Modell. Die Knoten 0, 1, 8 und E sind bereits informiert, die Knoten 4, 9 und F erhalten die Information in der aktuellen Runde.

Graph betrachtet werden können.

7.2 Analyse Random-Push Modell

Theorem 7.1 ([FG85]). *Ausgehend von einem informierten Knoten, sind im Random-Push Modell mit hoher Wahrscheinlichkeit nach $O(\log n)$ Runden alle n Knoten informiert.*

Wir widmen dem Beweis des Theorems den Rest dieses Teilabschnitts.
Wir verwenden für unsere Analyse folgende Zufallsvariablen::

- $I(t)$: Anzahl der informierten Knoten am Beginn von Runde t
- $U(t)$: Anzahl der uninformierten Knoten Beginn von Runde t
- $i(t) = \frac{|I(t)|}{n}$: relativer Anteil der informierten Knoten Beginn von Runde t
- $u(t) = \frac{|u(t)|}{n}$: relativer Anteil der uninformierten Knoten Beginn von Runde t

Es gilt also immer $I(t) + U(t) = n$, $i(t) + u(t) = 1$.

In der folgenden Analyse fixieren wir nun zunächst eine beliebige Runde t. Zunächst versuchen wir allgemeine Aussagen darüber zu treffen wie stark die Anzahl der informierten Knoten in einer Runde wächst. Dabei wollen wir jeden Knoten der in dieser Runde erstmalig informiert wird, wird seinem kleinsten informierten Anrufer zuschreiben. Zu diesem Zweck führen außerdem folgende Terminologie ein: Ein Knoten j führt in Runde t eine **erstmalige Informationsweitergabe** durch, wenn

- j am Beginn von Runde t informiert ist,
- der von j in Runde t angerufene Knoten am Beginn von Runde t uninformiert ist und

- der von j in Runde t angerufene Knoten in Runde t von keinem informierten Knoten, der kleiner ist als j, angerufen wird

Wir geben zunächst eine Schranke für die Wahrscheinlichkeit an, dass ein informierter Knoten eine erstmalige Informationsweitergabe durchführt. Im Folgenden bezeichnen wir $\mathcal{I}(t)$ als die Menge der informierten Knoten am Beginn von Runde t und, für alle $j \in \mathcal{I}(t)$, $c_j(t)$ als den Knoten, der von Knoten j in Runde t angerufen wird Wir definieren nun folgende Zufallsvariable für jeden Knoten $1 \leq j \leq I(t)$:

$$X_j(t) := \begin{cases} 1 & \text{falls } j \text{ eine erstmalige Informationsweitergabe durchführt} \\ 0 & \text{andernfalls.} \end{cases}$$

Äquivalent kann die Zufallsvariable folgendermaßen geschrieben werden:

$$X_j(t) = \begin{cases} 1 & \text{falls } c_j(t) \notin \mathcal{I}(t) \cup \{c_1(t), \ldots, c_{j-1}(t)\} \\ 0 & \text{andernfalls.} \end{cases}$$

Im Push Modell ruft der Knoten j uniform zufällig einen der n Knoten des Netzwerks an. Die Wahrscheinlichkeit, dass dieser angerufene Knoten in der Menge $\mathcal{I}(t) \cup \{c_1(t), \ldots, c_{j-1}(t)\}$ liegt ist

$$\frac{|\mathcal{I}(t) \cup \{c_1(t), \ldots, c_{j-1}(t)\}|}{n}$$

Wir beobachten nun, dass die Menge $\mathcal{I}(t) \cup \{c_1(t), \ldots, c_{j-1}(t)\}$ höchstens Größe $I(t) + j - 1 \leq 2I(t)$ haben kann. Damit erhalten wir folgende Abschätzung:

$$\begin{aligned} \Pr[X_j(t) = 1] &= 1 - \Pr[X_j(t) = 0] \\ &= 1 - \Pr[c_j(t) \in \mathcal{I}(t) \cup \{c_1(t), \ldots, c_{j-1}(t) \\ &= 1 - \frac{|\mathcal{I}(t) \cup \{c_1(t), \ldots, c_{j-1}(t)\}|}{n} \\ &\geq 1 - \frac{2I(t)}{n} = 2i(t)\,. \end{aligned}$$

Nun schätzen wir den Erwartungswert von $i(t+1)$ in Abhängigkeit von $i(t)$ ab. Sei $F(t) := I(t+1) - I(t)$ die Zufallsvariable für die Anzahl der in Runde t erstmalig informierten Knoten. Aufgrund der Linearität des Erwartungswerts und der einfachen Bestimmung des Erwartungswerts einer binären Zufallsvariable erhalten wir:

$$\begin{aligned} \mathrm{Ex}[F(t)] &= \mathrm{Ex}\left[\sum_{j=1}^{I(t)} X_j(t)\right] \\ &= \sum_{j=1}^{I(t)} \mathrm{Ex}[X_j(t)] \\ &= \sum_{j=1}^{I(t)} \Pr[X_j(t) = 1] \\ &\geq \sum_{j=1}^{I(t)} (1 - 2i(t)) \\ &= I(t) - 2i(t)I(t) \end{aligned}$$

Der Erwartungswert des relativen Anteils informierter Knoten nach Runde t kann nun folgendermaßen bestimmt werden:

$$\begin{aligned}
\mathrm{Ex}[i(t+1)] &= \mathrm{Ex}\left[i(t) + \frac{F(t)}{n}\right] \\
&= i(t) + \frac{\mathrm{Ex}[F(t)]}{n} \\
&\geq i(t) + \frac{I(t) - 2i(t)I(t)}{n} \\
&= i(t) + i(t) - 2i(t)^2 \\
&= 2i(t) - 2i(t)^2
\end{aligned}$$

Diese Schranke für den Erwartungswert liefert uns folgende Intuition für die weitere Analyse: Solange noch relativ wenige Knoten informiert wurden, ist der Term $2i(t)^2$ vernachlässigbar und der Term $2i(t)$ dominiert; die Anzahl informierter Knoten wächst dann annähernd um den Faktor 2 in jeder Runde.

Für den weiteren Beweis unterteilen wir den Prozess in zwei Phasen:

1. Die *Wachstumsphase* dauert so lange wie $I(t) \leq \frac{1}{3}n$ gilt.
2. Die *Schrumpfungsphase* beginnt sobald $I(t) > \frac{1}{3}n$ gilt.

Wir zeigen, dass beide Phasen mit hoher Wahrscheinlichkeit $O(\log n)$ Runden dauern, was impliziert, dass der gesamte Prozess mit hoher Wahrscheinlichkeit $O(\log n)$ Runden dauert.

7.2.1 Wachstumsphase

Wir zunächst zeigen, dass in jeder Runde der Wachstumsphase mit konstanter Wahrscheinlichkeit $I(t)$ um mindestens den Faktor $\frac{7}{6}$ wächst. Dies ist der Fall, wenn $F(t) > \frac{1}{6}I(t)$ ist. Da in dieser Phase $I(t) \leq \frac{1}{3}n$ – oder äquivalent $i(t) \leq \frac{1}{3}$ – gilt erhalten wir

$$\Pr[X_j(t) = 1] \geq 1 - 2i(t) \geq 1 - 2 \cdot \tfrac{1}{3} = \tfrac{1}{3}\,.$$

Wir würden nun gerne die Wahrscheinlichkeit für das Ereignis $F(t) \leq \frac{1}{6}I(t)$ (d.h. das Wachstum ist nicht stark genug) die Chernoff-Schranke aus Theorem B.14 mit $p = \frac{1}{3}$, $\mu = \frac{1}{3}I(t)$ und $\delta = \frac{1}{2}$ anwenden um folgende Abschätzung zu erhalten:

$$\begin{aligned}
\Pr[F(t) \leq \tfrac{1}{6}I(t)] &= \Pr\left[\sum_{j=1}^{I(t)} X_j(t) \leq \frac{1}{6}I(t)\right] \\
&= \Pr\left[\sum_{j=1}^{I(t)} X_j(t) \leq \left(1 - \frac{1}{2}\right) \cdot \frac{1}{3}I(t)\right] \\
&\leq \frac{1}{e^{\frac{(\frac{1}{2})^2}{2} \cdot \frac{1}{3}I(t)}} \\
&= \frac{1}{e^{\frac{1}{24}I(t)}} \leq \frac{1}{e^{\frac{1}{24}}}\,,
\end{aligned}$$

wobei die letzte Ungleichung wegen $I(t) \geq 1$ gilt. Wir erhalten somit

$$\Pr[I(t+1) > \tfrac{7}{6} I(t)] = \Pr[F(t) > \tfrac{1}{6} I(t)] = 1 - \Pr[F(t) \leq \tfrac{1}{6} I(t)] \geq 1 - \frac{1}{e^{\frac{1}{24}}} \geq \tfrac{4}{100} .$$

Analog zur Laufzeitanalyse im verteilten MIS-Algorithmus (siehe Lemma 4.5) unterteilen wir nun die Runden der Wachstumsphase in *gute* und *schlechte* Runden. Runde t gilt als gut, wenn $I(t+1) > \frac{7}{6} I(t)$ gilt, und ansonsten als schlecht. Spätestens nach $\log_{7/6} \frac{n}{3}$ guten Runden gilt $I(t) > \frac{1}{3} n$ (womit die Wachstumsphase abgeschlossen ist). Mit einer erneuten Anwendung der Chernoff-Schranke kann gezeigt werden, dass mit hoher Wahrscheinlichkeit $O(\log n)$ Runden benötigt, damit darunter $\log_{7/6} \frac{n}{3}$ gute Runden auftreten. Somit benötigt die Wachstumsphase mit hoher Wahrscheinlichkeit $O(\log n)$ Runden.

Unsere Analyse hat bisher ein zentrales Detail außer Acht gelassen: Die Chernoff-Schranke gilt nur für unabhängige Variablen, aber die Variablen $X_j(t)$ sind nicht unabhängig. Gilt beispielsweise $X_j(t) = 1$ für einen Knoten j, dann schließt das die Möglichkeit aus, dass andere Knoten denselben Knoten erstmalig informieren. Dies verringert aber letztendlich die Wahrscheinlichkeit, dass $X_j(t) = 1$ für Knoten $j' \neq j$ gilt. Wir werden aber sehen, dass es genügt den „Grad an Unabhängigkeit“ unserer Zufallsvariablen auf folgende Aussage einzuschränken:

$$\Pr[X_j(t) = 1 \mid X_1(t) = x_1, \ldots, X_{j-1}(t) = x_{j-1}] \geq 1 - 2i(t) \tag{7.1}$$

für alle potentiellen Belegungen $x_1, \ldots, x_{j-1} \in \{0, 1\}$.

Lemma 7.2 ([Doe20]). *Seien $X_1, \ldots, X_k$ beliebige binäre Zufallsvariablen und seien $Y_1, \ldots, Y_k$ unabhängige binäre Zufallsvariablen. Wenn für alle j und alle $x_1, \ldots, x_{j-1} \in \{0, 1\}$ gilt, dass*

$$\Pr[X_j = 1 \mid X_1 = x_1, \ldots, X_{j-1} = x_{j-1}] \geq \Pr[Y_j = 1]$$

dann gilt für alle $T \geq 0$

$$\Pr\left[\sum_{j=1}^{k} X_j \leq T\right] \leq \Pr\left[\sum_{j=1}^{k} Y_j \leq T\right]$$

In der Regel möchte man dieses Lemma anwenden, um anschließend $\Pr\left[\sum_{j=1}^{k} Y_j \leq T\right]$ durch eine Chernoff-Schranke abschätzen.

In unserer Analyse des Push Modells können wir für jedes $1 \leq j \leq I(t)$ eine binäre Zufallsvariable Y_j definieren, für die $Y_j = 1$ mit Wahrscheinlichkeit $1 - 2i(t)$ gilt und $Y_j = 0$ andernfalls. Mit (7.1) ergibt sich dann

$$\Pr[X_j(t) = 1 \mid X_1(t) = x_1, \ldots, X_{j-1}(t) = x_{j-1}] \geq 1 - 2i(t) = \Pr[Y_j = 1]$$

und mit Hilfe des obigen Lemmas verändert sich unsere obige Analyse nur insofern, dass wir die Chernoff-Schranke auf die Summe der $Y_j(t)$'s anstatt der Summe der $X_j(t)$'s anwenden – alle anderen Argumente bleiben genau gleich.

Es bleibt also zu zeigen, dass (7.1) gilt. Dabei lässt sich leicht erkennen, dass unsere bisherige Analyse weiterhin problemlos durchgeht: Für jede Belegung $x_1, \ldots, x_{j-1} \in \{0, 1\}$ gilt

$$\begin{aligned}
&\Pr[X_j(t) = 0 \mid X_1(t) = x_1, \ldots, X_{j-1}(t) = x_{j-1}] \\
&\quad = \Pr[c_j(t) \in I(t) \cup \{c_1(t), \ldots, c_{j-1}(t)\} \mid X_1(t) = x_1, \ldots, X_{j-1}(t) = x_{j-1}] \\
&\quad \leq \frac{I(t) + j - 1}{n} \leq \frac{2I(t)}{n} = 2i(t) .
\end{aligned}$$

7.2.2 Schrumpfungsphase

Für die Analyse der Schrumpfungsphase wechseln wir die Perspektive: Anstelle das Wachstums der Menge der informierten Knoten analysieren wir explizit die Schrumpfung der Menge der uninformierten Knoten.

Betrachten wir die Wahrscheinlichkeit, dass ein beliebiger uninformierter Knoten in Runde t nicht informiert wird. Jeder bereits informierte Knoten ruft diesen uninformierten Knoten mit Wahrscheinlichkeit $1 - \frac{1}{n}$ an, wobei insgesamt $I(t)$ solcher informierter Knoten existieren. Der gesunde Knoten wird also mit Wahrscheinlichkeit $(1 - \frac{1}{n})^{I(t)}$ nicht infiziert. Da in der Schrumpfungsphase $I(t) \geq \frac{1}{3}n$ gilt, lässt sich für diese Wahrscheinlichkeit folgende obere Schranke ableiten:

$$\begin{aligned}\left(1 - \frac{1}{n}\right)^{I(t)} &\leq \left(1 - \frac{1}{n}\right)^{\frac{1}{3}n} \\ &= \left(\left(1 - \frac{1}{n}\right)^{n}\right)^{\frac{1}{3}} \leq \left(\frac{1}{e}\right)^{\frac{1}{3}} = \frac{1}{e^{\frac{1}{3}}}\,.\end{aligned}$$

Hier haben wir die nützliche Ungleichung $(1 - \frac{1}{x})^x \leq \frac{1}{e}$ (siehe (A.18)) verwendet.

Betrachten wir nun die Wahrscheinlichkeit, dass ein (fixierter) uninformierter Knoten in $3(c + 1) \ln n$ aufeinanderfolgenden Runden nicht informiert wird. Wenn man die Informationsweitergabe als Bernoulli-Experiment auffasst, folgt, dass dies gleich der Wahrscheinlichkeit ist, dass der Knoten in *einer* Runde nicht informiert wird ($\leq \frac{1}{e^{1/3}}$), hoch die Anzahl der Runden, also:

$$\left(\left(1 - \frac{1}{n}\right)^{I(t)}\right)^{3(c+1)\ln n} \leq \left(\frac{1}{e^{\frac{1}{3}}}\right)^{3(c+1)\ln n} = \frac{1}{e^{\frac{1}{3}\cdot 3(c+1)\ln n}} = \frac{1}{n^{c+1}}\,. \tag{7.2}$$

Nun möchten wir die Wahrscheinlichkeit, dass mindestens einer der uninformierten Knoten in $3(c + 1) \ln n$ aufeinanderfolgenden Runden nicht infiziert wird, analysieren. Hierfür verwenden wir die Union Bound (Lemma B.1) und multiplizieren deshalb die Wahrscheinlichkeit aus (7.2) mit $U(t) \leq n$, um eine obere Schranke von $U(t) \cdot \frac{1}{n^{c+1}} \leq \frac{1}{n^c}$ zu erhalten. Die Wahrscheinlichkeit für das Gegenereignis, nämlich dass nach $3(c + 1) \ln n$ Runden der Schrumpfungsphase alle Knoten infiziert wurden, beträgt somit mindestens $1 - \frac{1}{n^c}$. Mit hoher Wahrscheinlichkeit dauert die Schrumpfungsphase daher $O(\log n)$ Runden

7.3 Analyse Random-Pull Modell

Wir werden das Pull Modell in dieser Vorlesung nicht explizit analysieren. Auch im Pull Modell gilt, dass mit hoher Wahrscheinlichkeit $O(\log n)$ Runden benötigt werden bis alle Knoten informiert sind. Die Analyse benötigt ähnliche Beweistechniken wie jene, die wir für das Push Modell gesehen haben.

7.4 Analyse Random-Push&Pull Modell

Sowohl für das Random-Push als auch das Random-Pull Modell ergibt sich mit hoher Wahrscheinlichkeit eine Rundenkomplexität von $O(\log n)$ und damit trivialerweise eine Nachrichtenkomplexität von $O(n \log n)$, weil in jeder Runde jeder der n Knoten eine Nachricht sendet. Außerdem ist klar, dass immer $\Omega(n)$ Nachrichten benötigt werden, da jeder Knoten mindestens einmal eine Nachricht

empfangen muss. Interessanterweise stellt sich heraus, dass sich die Nachrichtenkomplexität auf $O(n \log \log n)$ reduziert, wenn Push und Pull kombiniert werden.

Theorem 7.3 ([KSSV00]). *Ausgehend von einem informierten Knoten, sind im Random-Push&Pull Modell mit hoher Wahrscheinlichkeit nach $O(\log n)$ Runden mit insgesamt $O(n \log \log n)$ gesendeten Nachrichten alle n Knoten informiert.*

Der erste Teil des Theorems folgt direkt aus Theorem 7.1, da für die Kombination aus Push und Pull niemals mehr Runden benötigt werden als für Push alleine. Wir widmen dem Beweis der Nachrichtenkomplexität von $O(n \log \log n)$ den Rest dieses Teilabschnitts. Dabei verwenden wir die gleiche Notation und Terminologie wie in der Analyse des Random-Push Modells (Abschnitt 7.2). Da wir nur eine asymptotischen Abschätzung der Nachrichtenkomplexität durchführen, nehmen wir im Folgenden immer an, dass n groß genug ist, das heißt größer als eine Konstante n_0, die sich implizit aus unserer Analyse ergibt.

Für den weiteren Beweis unterteilen wir den Prozess in drei Phasen:

1. Die *Wachstumsphase* dauert so lange wie $I(t) \leq \frac{n}{\ln n}$ gilt.
2. Die *Schrumpfungsphase* dauert so lange wie $n - \frac{n}{\ln n} > U(t) \geq \sqrt{3cn(\ln n)^5}$ gilt.
3. Die *Schlussphase* beginnt sobald $U(t) < \sqrt{3cn(\ln n)^5}$ gilt.

Wir zeigen, dass in jeder der drei Phasen mit hoher Wahrscheinlichkeit $O(n \log \log n)$ Nachrichten gesendet werden, was impliziert, dass insgesamt mit hoher Wahrscheinlichkeit $O(n \log \log n)$ Nachrichten gesendet werden. Für die Wachstumsphase zeigen wir direkt, dass $O(n)$ Nachrichten gesendet werden. Für die Schrumpfungsphase zeigen wir, dass diese $O(\log \log n)$ Runden dauert, was trivialerweise impliziert, dass $O(n \log \log n)$ Nachrichten gesendet werden. Für die Schlussphase zeigen wir eine Rundenkomplexität von $O(1)$, die eine Nachrichtenkomplexität von $O(n)$ impliziert.

7.4.1 Wachstumsphase

Aus der Analyse des Random-Push Modells folgt, dass die Wachstumsphase mit hoher Wahrscheinlichkeit $t^* = O(\log n)$ Runden dauert. Sei $M(t)$ die Zufallsvariable für die Anzahl der in Runde t gesendeten Nachrichten. Wir definierten nun folgende drei Zufallsvariablen, basierend auf den drei grundlegenden Interaktionsarten zwischen jeweils zwei Knoten:

- $A(t)$: Anzahl informierter Knoten, die uninformierte Knoten anrufen (Push)
- $B(t)$: Anzahl uninformierter Knoten, die informierte Knoten anrufen (Pull)
- $C(t)$: Anzahl informierter Knoten, die informierte Knoten anrufen (Push&Pull)

Wir beobachten nun, dass $M(t) = A(t) + B(t) + 2C(t)$ gilt. (Der Faktor 2 rührt daher, dass beim Anruf eines informierten Knotens durch einen informierten Knoten beide Knoten jeweils eine Nachricht senden.) Trivialerweise gilt $A(t) \leq I(t)$ und $C(t) \leq I(t)$. Somit erhalten wir $M(t) \leq 3I(t) + B(t)$. Da jeder Knoten nur einmal informiert werden kann (also seinen Zustand von uninformiert auf informiert ändern kann), gilt $\sum_{t=1}^{t^*} B(t) \leq n$. Für die Anzahl der in der

Wachstumsphase gesendeten Nachrichten erhalten wir wegen $I(t) \leq \frac{n}{\ln n}$ und $t^* = O(\log n)$ mit hoher Wahrscheinlichkeit nun folgende Abschätzung:

$$\begin{aligned}\sum_{t=1}^{t^*} M(t) \leq \sum_{t=1}^{t^*} (3I(t) + B(t)) &= \sum_{t=1}^{t^*} 3I(t) + \sum_{t=1}^{t^*} B(t) \\ &\leq \frac{3n}{\ln n} \cdot t^* + n = O(n)\end{aligned}$$

7.4.2 Schrumpfungsphase

Wir zeigen zunächst, dass in dieser Phase die erwartete Anzahl uninformierter Knoten mit jeder Runde „quadratisch“ schrumpft. Allgemein beträgt die Wahrscheinlichkeit, dass ein uninformierter Knoten einen uninformierten Knoten anruft, $u(t)$. Somit ergibt sich

$$\mathrm{Ex}[U(t+1)] = u(t) \cdot U(t)$$

oder äquivalent:

$$\mathrm{Ex}[u(t+1)] = \frac{u(t) \cdot U(t)}{n} = (u(t))^2 .$$

Iteriert man diese Gleichung ergibt sich

$$\mathrm{Ex}[u(t+k)] = (u(t))^{(2^k)} .$$

Wir möchten nun herausfinden, wie oft diese Gleichung angewendet werden muss, bzw. wie groß k sein muss, damit $U(t) < 1$ (oder äquivalent: $u(t) < \frac{1}{n}$) in Erwartung gilt, also kein Knoten mehr uninformiert ist. In der Schrumpfungsphase gilt $U(t) < n - \frac{n}{\ln n}$ (also $u(t) < 1 - \frac{1}{\ln n}$). Wir beobachten nun, dass sich für $k \geq 2\log_2(\ln n)$ folgendes ergibt.

$$(u(t))^{(2^k)} < \left(1 - \frac{1}{\ln n}\right)^{(\ln n)^2} \leq \frac{1}{e^{\ln n}} = \frac{1}{n} . \tag{7.3}$$

Die Idee ist nun mittels einer Chernoff-Schranke zu zeigen, dass mit hoher Wahrscheinlichkeit in *jeder* Runde der Schrumpfungsphase ein annähernd quadratisches Schrumpfen stattfindet.

Sei zunächst für jeden uninformierten Knoten j in Runde t die binäre Zufallsvariable $X_j(t)$ gegeben durch

$$X_j(t) := \begin{cases} 1 & \text{falls in Runde } t \text{ uninformierter Knoten } j \text{ uninformierten Knoten anruft} \\ 0 & \text{andernfalls.} \end{cases}$$

Dann gilt $U(t+1) = \sum_{j=1}^{U(t)} X_j(t)$. Außerdem haben wir $\Pr[X_j(t) = 1] = u(t) \leq 1 - \frac{1}{\ln n}$, da in der Schrumpfungsphase $U(t) \leq n - \frac{n}{\ln n}$ gilt. Für die Anwendung der Chernoff-Schranke (Theorem B.13) setzen wir nun $p := 1 - \frac{1}{\ln n}$ (und somit $\mu = \left(1 - \frac{1}{\ln n}\right) \cdot U(t)$) und $\delta := \frac{1}{(\ln n)^2}$. Dann erhalten wir, zusammen mit der Schranke $U(t) \geq \sqrt{3cn(\ln n)^5}$ für die Schrumpfungsphase:

$$\begin{aligned}\Pr\left[U(t+1) \geq (1+\delta) \cdot u(t) \cdot U(t)\right] = \Pr\left[\sum_{j=1}^{U(t)} X_j(t) \geq (1+\delta) \cdot \mu\right] &\leq \frac{1}{e^{\frac{\delta^2}{3} \cdot \mu}} = \frac{1}{e^{\frac{\delta^2}{3} \cdot \frac{(U(t))^2}{n}}} \\ &\leq \frac{1}{e^{\frac{\delta^2}{3} \cdot \frac{3cn(\ln n)^5}{n}}} = \frac{1}{e^{c \ln n}} = \frac{1}{n^c} .\end{aligned}$$

Für das Gegenereignis gilt nun

$$\Pr\left[U(t+1) < (1+\delta)\cdot u(t)\cdot U(t)\right] = 1 - \Pr\left[U(t+1) \geq (1+\delta)\cdot u(t)\cdot U(t)\right] \geq 1 - \frac{1}{n^c}\,.$$

Somit ergibt sich $U(t+1) \leq (1+\delta)\cdot u(t)\dot{U}(t)$ mit hoher Wahrscheinlichkeit in jeder Runde der Schrumpfungsphase, beziehungsweise äquivalent:

$$u(t+1) \leq (1+\delta)\cdot (u(t))^2\,.$$

Durch Iteration dieser Ungleichung gilt mit hoher Wahrscheinlichkeit nach k Runden der Schrumpfungsphase:

$$u(t+k) \leq (1+\delta)^{(2^k)-1}\cdot (u(t))^{(2^k)} \leq (1+\delta)^{(2^k)}\cdot (u(t))^{(2^k)}\,.$$

Mit $k := \lceil 2\log_2(\ln n)\rceil$ und $U(t) < n - \frac{n}{\ln n}$ (also $u(t) < 1 - \frac{1}{\ln n}$) erhalten wir wegen (7.3):

$$(1+\delta)^{(2^k)}(u(t))^{(2^k)} \leq (1+\delta)^{(2^k)}\cdot\frac{1}{n}$$

Nun gilt aufgrund unserer Wahl $\delta = \frac{1}{(\ln n)^2}$, dass $k = \lceil \log_2 \frac{1}{\delta}\rceil$ und wir erhalten mit (A.17) (der Ungleichung $(1+\frac{1}{x})^x \leq e$) und $\delta \leq 1$:

$$(1+\delta)^{(2^k)} \leq (1+\delta)^{(2^{1+\log_2\frac{1}{\delta}})} = (1+\delta)^{2\cdot\frac{1}{\delta}} \leq e^2\,.$$

Da n „groß genug“ ist, dürfen wir annehmen, dass $\sqrt{3cn(\ln n)^5} \geq e^2$ gilt. Somit ergibt sich für $k := \lceil 2\log_2(\ln n)\rceil$ die Ungleichung

$$u(t+k) \leq \frac{\sqrt{3cn(\ln n)^5}}{n}\,.$$

Die Schrumpfungsphase dauert also höchstens $k = O(\log\log n)$ viele Runden.

7.4.3 Schlussphase

Um zu zeigen, dass die Schlussphase mit Wahrscheinlichkeit mindestens $1 - \frac{1}{n^c}$ aus höchstens $2c+3$ Runden besteht, analysieren wir das Gegenereignis. Da es in der Schlussphase höchstens $\sqrt{3cn(\ln n)^5}$ uninformierte Knoten gibt, ist in jeder Runde der Schlussphase die Wahrscheinlichkeit für einen uninformierter Knoten einen uninformierten Knoten anzurufen höchstens

$$q := \frac{\sqrt{3cn(\ln n)^5}}{n}$$

Die Wahrscheinlichkeit für einen uninformierten Knoten in $2c+3$ Runden der Schlussphase nur uninformierte Knoten anzurufen (und damit uninformiert bleibt) ist somit höchstens q^{2c+3}. Die Schlussphase dauert länger als $2c+3$ Runden, wenn mindesten einer der höchstens $\sqrt{3cn(\ln n)^5}$ uninformierten Knoten in den $2c+3$ Runden der Schlussphase nur uninformierte Knoten anruft. Die Wahrscheinlichkeit dieses Ereignisses lässt sich nun mit der Union-Bound abschätzen als höchstens

$$\sqrt{3cn(\ln n)^5}\cdot p^{2c+3} = \frac{\left(\sqrt{3cn(\ln n)^5}\right)^{2c+4}}{n^{2c+3}} = \frac{(3c(\ln n)^5)^{c+2}\cdot n^{c+2}}{n^{2c+3}} = \frac{(3c(\ln n)^5)^{c+2}}{n^{c+1}} \leq \frac{1}{n^c}\,.$$

Dabei gilt die letzte Ungleichung, weil wir annehmen, dass n „groß genug“ ist, insbesondere so groß, dass $n \geq (3c(\ln n)^5)^{c+2}$ gilt.

7.5 Literatur

[DGHI+88] Alan J. Demers, Daniel H. Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard E. Sturgis, Daniel C. Swinehart und Douglas B. Terry. "Epidemic Algorithms for Replicated Database Maintenance". In: *Operating Systems Review* 22.1 (1988), S. 8–32. DOI: 10.1145/43921.43922 (siehe S. 61).

[DK14] Benjamin Doerr und Marvin Künnemann. "Tight Analysis of Randomized Rumor Spreading in Complete Graphs". In: *Proceedings of the Eleventh Workshop on Analytic Algorithmics and Combinatorics (ANALCO 2014)*. 2014, S. 82–91. DOI: 10.1137/1.9781611973204.8.

[Doe20] Benjamin Doerr. "Probabilistic Tools for the Analysis of Randomized Optimization Heuristics". In: *Theory of Evolutionary Computation - Recent Developments in Discrete Optimization*. Hrsg. von Benjamin Doerr und Frank Neumann. Natural Computing Series. Springer, 2020, S. 1–87. DOI: 10.1007/978-3-030-29414-4_1 (siehe S. 65).

[FG85] Alan M. Frieze und Geoffrey R. Grimmett. "The shortest-path problem for graphs with random arc-lengths". In: *Discrete Applied Mathematics* 10.1 (1985), S. 57–77. DOI: 10.1016/0166-218X(85)90059-9 (siehe S. 62).

[KSSV00] Richard M. Karp, Christian Schindelhauer, Scott Shenker und Berthold Vöcking. "Randomized Rumor Spreading". In: *Proc. of the Symposium on Foundations of Computer Science (FOCS)*. 2000, S. 565–574. DOI: 10.1109/sfcs.2000.892324 (siehe S. 67).

Dieser Abschnitt basiert zum Teil auf Vorlesungseinheiten von Robert Elsässer und Christian Schindelhauer.

A Basiswissen Mathematik

A.1 Exponenten

Die *Exponentialfunktion* mit einer reellen Zahl $b \in \mathbb{R}^{>0} \setminus \{1\}$ als *Basis* ist die Funktion $\exp_b : \mathbb{R} \to \mathbb{R}^{>0}$, die durch die Abbildungsvorschrift $x \mapsto b^x$ gegeben ist. Für $k \in \mathbb{Z}^{>0}$ erhält man b^k durch $(k-1)$-malige Multiplikation von b mit sich selbst, formal $b^k = \prod_{i=1}^{k} b$, und $b^{1/k}$ ist die k-te Wurzel von b, also $b^{1/k} = \sqrt[k]{b}$.

Die *Eulersche Zahl* e kann auf verschiedene äquivalente Arten definiert werden, beispielsweise als Grenzwert der Folge, deren n-tes Glied durch $(1 + \frac{1}{n})^n$ gegeben ist, also $e = \lim_{n\to\infty}(1 + 1/n)^n$. Wir die Basis in der Exponentialfunktion nicht explizit angeführt, so ist in der Regel die Exponentialfunktion zur Basis e gemeint, also $\exp(x) = e^x$.

Wir benötigen in der Vorlesung folgende Rechenregeln für Exponenten (die für alle $x, y \in \mathbb{R}^{>0}$ und alle Basen $b \in \mathbb{R}^{>0} \setminus \{1\}$ gelten):

$$b^0 = 1 \tag{A.1}$$

$$b^x \cdot b^y = b^{x+y} \tag{A.2}$$

$$\frac{b^x}{b^y} = b^{x-y} \tag{A.3}$$

$$(b^x)^y = b^{x \cdot y} \tag{A.4}$$

$$b^{-x} = \frac{1}{b^x} \tag{A.5}$$

A.2 Logarithmen

Die Umkehrfunktion der Exponentialfunktion mit Basis b ist die *Logarithmusfunktion* $\log_b : \mathbb{R}^{>0} \to \mathbb{R}$ mit Basis b. Der *Logarithmus* $\log_b(x)$ von x zur Basis b ist also die eindeutige Zahl y, für die $b^y = x$ gilt. Der Logarithmus zur Basis e wird mit $\ln$ symbolisiert, also $\ln(x) = \log_e(x)$. Wird die Basis im Logarithmus nicht explizit angeführt, so ist in der Informatik in der Regel der Logarithmus zur Basis 2 gemeint.

Wir benötigen in der Vorlesung folgende Rechenregeln für Logarithmen (die für alle $x, y \in \mathbb{R}^{>0}$ und alle Basen $b, b' \in \mathbb{R}^{>0} \setminus \{1\}$ gelten):

$$b^{\log_b(x)} = x \tag{A.6}$$

$$\log_b(1) = 0 \tag{A.7}$$

$$\log_b(x \cdot y) = \log_b(x) + \log_b(y) \tag{A.8}$$

$$\log_b\left(\frac{x}{y}\right) = \log_b(x) - \log_b(y) \tag{A.9}$$

$$\log_b\left(\frac{1}{x}\right) = -\log_b(x) \tag{A.10}$$

$$\log_b(x^y) = y \cdot \log_b(x) \tag{A.11}$$

$$\log_b(x) = \frac{\log_{b'}(x)}{\log_{b'}(b)} \tag{A.12}$$

Falls b eine Konstante ist, so ist auch $\frac{1}{\log_{b'}(b)}$ eine Konstante und mit Regel (A.12) zur Basisumrechnung gilt daher $\log_b(x) = O(\log_{b'}(x))$. Aus diesem Grund wird die Basis des Logarithmus in der O-Notation meist weggelassen und insbesondere gilt $\ln(x) = O(\log(x))$.

In der Vorlesung haben wir oft das Ziel, einen Algorithmus mit Fehlerwahrscheinlichkeit p (zum Beispiel $p \leq \frac{1}{2}$) so oft zu wiederholen, dass das Ereignis, dass alle Ausführungen des Algorithmus fehlerhaft waren, eine geringere Wahrscheinlichkeit von höchstens q aufweist (zum Beispiel $q = \frac{1}{n^c}$). Bei $k = \lceil \log_p(q) \rceil$ Wiederholungen des Algorithmus ist die Wahrscheinlichkeit, dass alle Ausführungen fehlerhaft waren gleich

$$p^k = p^{\lceil \log_p(q) \rceil} \leq p^{\log_p(q)} = q\,.$$

Für $p \leq \frac{1}{2}$ und $q = \frac{1}{n^c}$ wären also $k = \lceil \log_{1/2}(\frac{1}{n^c}) \rceil = \lceil \log_2(n^c) \rceil = O(c \log n)$ Wiederholungen nötig damit $p^k \leq \frac{1}{n^c}$ gilt.

A.3 Verschiedenes

A.3.1 Rundungsoperator

Der *ceil*-Operator $\lceil \cdot \rceil$ rundet eine reelle Zahl auf die nächste ganze Zahl auf, also $\lceil x \rceil = \min\{z \in \mathbb{Z} \mid z \geq x\}$. Für diese Art der Rundung gilt die Ungleichung $x \leq \lceil x \rceil \leq x + 1$.

A.3.2 Geometrische Reihe

Die n-te Partialsumme einer geometrischen Folge mit Quotient q ist definiert als $\sum_{k=0}^{n} q^k$. Für $q \neq 1$ gilt

$$\sum_{k=0}^{n} q^k = \frac{1 - q^{n+1}}{1 - q} = \frac{q^{n+1} - 1}{q - 1} \tag{A.13}$$

und für $q = 2$ gilt insbesondere

$$\sum_{k=0}^{n} 2^k = \frac{2^{n+1} - 1}{1} \leq 2^{n+1} = O(2^n)\,. \tag{A.14}$$

Für $|q| < 1$ konvergiert der Grenzwert der Partialsummen und es gilt

$$\sum_{k=0}^{\infty} q^k = \frac{1}{1-q}\,. \tag{A.15}$$

A.3.3 Ungleichungen

Bernoullische Ungleichung Für jede reelle Zahl $x \geq -1$ und jede ganze Zahl $n \geq 0$ gilt

$$(1+x)^n \geq 1 + nx\,. \tag{A.16}$$

Eulersche Zahl Aus der Grenzwertdefinition der Eulerschen Zahl ergibt sich dass für jede reelle Zahl $x > 0$

$$\left(1 + \frac{1}{x}\right)^x \leq e \tag{A.17}$$

gilt und für jede reelle Zahl $x \geq 1$

$$\left(1 - \frac{1}{x}\right)^x \leq \frac{1}{e} \tag{A.18}$$

gilt.

B Grundlagen Wahrscheinlichkeitstheorie

B.1 Diskrete Wahrscheinlichkeitsräume

Ein *diskreter Wahrscheinlichkeitsraum* (Ω, p) besteht aus einer höchstens abzählbaren[5] *Ergebnismenge* Ω (deren Elemente Ergebnisse genannt werden) und einer Wahrscheinlichkeitsfunktion $p : \Omega \to [0,1]$, die Ergebnisse auf reelle Zahlen zwischen 0 und 1 abbildet, so dass $\sum_{\omega\in\Omega} p(\omega) = 1$. Für den Fall, dass $\Omega = \{\omega_1, \omega_2, \ldots\}$ abzählbar unendlich ist muss also $\sum_{i=1}^{\infty} p(\omega_i) = 1$ gelten.[6] Ein *Ereignis* ist eine Menge von Ergebnissen, also eine Teilmenge $A \subseteq \Omega$, wobei eine einelementige Teilmenge $\{\omega\}$ bestehend aus einem einzigen Ereignis ω *Elementarereignis* genannt wird. Die *Wahrscheinlichkeitsverteilung* $\Pr : 2^\Omega \to [0,1]$ auf (Ω, p), die Ereignisse auf reelle Zahlen zwischen 0 und 1 abbildet, ist definiert durch $\Pr[A] := \sum_{\omega\in A} p(\omega)$ für alle $A \subseteq \Omega$. Das *Gegenereignis* eines Ereignisses A ist das Ereignis $\bar{A} := \Omega \setminus A$ und aus dieser Definition folgt $\Pr[\bar{A}] = 1 - \Pr[A]$.

Beispiel. Das Werfen eines Würfels kann mit einer Ergebnismenge $\Omega = \{1,2,3,4,5,6\}$ mit sechs verschiedenen Ereignissen und der Wahrscheinlichkeitsfunktion p gegeben durch $p(1) = p(2) = p(3) = p(4) = p(5) = p(6) = \frac{1}{6}$ modelliert werden. Sei $G = \{2,4,6\}$ jenes Ereignis, das den Wurf einer geraden Zahl repräsentiert. Dann gilt $\Pr[G] = p(2) + p(4) + p(6) = 3 \cdot \frac{1}{6} = \frac{3}{6} = \frac{1}{2}$. Sei

[5] Zur Erinnerung: Eine Menge heißt abzählbar, wenn sie die gleiche Mächtigkeit hat wie die Menge der natürlichen Zahlen.

[6] Zur Erinnerung: $\sum_{i=1}^{\infty} p(\omega_i)$ bezeichnet den Grenzwert der Partialsummen der Reihe, also $\sum_{i=1}^{\infty} p(\omega_i) = \lim_{n\to\infty} \sum_{i=1}^{n} p(\omega_i)$.

$U = \{1, 3, 5\}$ jenes Ereignis, das den Wurf einer geraden Zahl repräsentiert. Da U das Gegenereignis von G ist, gilt $\Pr[U] = 1 - \Pr[G] = 1 - \frac{1}{2} = \frac{1}{2}$.

Lemma B.1 (Boolesche Ungleichung bzw. Union Bound)**.** *Seien A und B zwei Ereignisse und $C = A \cup B$ jenes Ereignis das eintritt, wenn Ereignis A eintritt oder wenn Ereignis B eintritt. Dann gilt $\Pr[C] \leq \Pr[A] + \Pr[B]$. Allgemein gilt für jede endliche, nichtleere Menge $\mathcal{A} = \{A_1, \ldots, A_n\}$ von Ereignissen: $\Pr[\bigcup_{i=1}^{n} A_i] \leq \sum_{i=1}^{n} \Pr[A_i]$.*

Beispiel. Sei beim Werfen eines Würfels $G = \{2, 4, 6\}$ jenes Ereignis, das den Wurf einer geraden Zahl repräsentiert und $Q = \{1, 4\}$ jenes Ereignis, das den Wurf einer Quadratzahl repräsentiert. Dann gilt mit der Booleschen Ungleichung $\Pr[G \cup Q] \leq \Pr[G] + \Pr[Q] \leq \frac{3}{6} + \frac{2}{6} = \frac{5}{6}$. Tatsächlich gilt $G \cup Q = \{1, 2, 4, 6\}$ und deshalb $\Pr[G \cup Q] = \frac{4}{6} = \frac{2}{3}$.

B.2 Zufallsvariablen und Erwartungswerte

Eine (reelle) *Zufallsvariable* $X : \Omega \to \mathbb{R}$ auf einem diskreten Wahrscheinlichkeitsraum (Ω, p) ist eine Funktion, die jedem Ergebnis $\omega \in \Omega$ eine reelle Zahl $X(\omega)$ zuordnet. Für jedes $x \in \mathbb{R}$ bezeichnen wir als „$X = x$“ das Ereignis $\{\omega \in \Omega | X(\omega) = x\}$. Die *Wahrscheinlichkeitsverteilung* $\Pr_X : \mathbb{R} \to [0, 1]$ einer Zufallsvariablen X ist gegeben durch $\Pr_X[x] := \Pr[X = x] = \Pr[\{\omega \in \Omega | X(\omega) = x\}]$. Der *Erwartungswert* von X ist definiert als

$$\mathrm{Ex}[X] := \sum_{\omega \in \Omega} X(\omega) \cdot p(\omega) = \sum_{x \in \mathbb{R}} x \cdot \Pr[X = x] \,.$$

wobei in der letzteren Summe jene Summenglieder von Werten x, die nicht im Wertebereich $X(\Omega)$ liegen, automatisch 0 sind.

Beispiel. Wir modellieren wieder das Werfen eines Würfels mit dem Wahrscheinlichkeitsraum (Ω, p) gegeben durch $\Omega = \{1, \ldots, 6\}$ und $p(k) = \frac{1}{6}$ für alle $k \in \Omega$. Sei X die Zufallsvariable definiert durch

$$X(x) = \begin{cases} x & \text{falls } k \in \{1, 2, 3, 4, 5, 6\} \\ 0 & \text{ansonsten} \end{cases} .$$

Dann entspricht X der geworfenen Augenzahl. Die Wahrscheinlichkeitsverteilung von X ist somit gegeben durch

$$\Pr[X = x] = \begin{cases} \frac{1}{6} & \text{falls } x \in \{1, \ldots, 6\} \\ 0 & \text{ansonsten} \end{cases}$$

Für den Erwartungswert von X gilt

$$\mathrm{Ex}[X] = 1 \cdot \Pr[X = 1] + \cdots + 6 \cdot \Pr[X = 6] = 1 \cdot \tfrac{1}{6} + \ldots 6 \cdot \tfrac{1}{6} = (1 + \cdots + 6)\tfrac{1}{6} = \tfrac{21}{6} = \tfrac{7}{2} = 3.5$$

Theorem B.2 (Linearität des Erwartungswerts)**.** *Seien X und Y zwei beliebige Zufallsvariablen und $Z := X + Y$ die Summe dieser Zufallsvariablen. Dann gilt $\mathrm{Ex}[Z] = \mathrm{Ex}[X] + \mathrm{Ex}[Y]$. Allgemein gilt für endlich viele Zufallsvariablen $X_1, \ldots, X_n$ dass $\mathrm{Ex}[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} \mathrm{Ex}[X_i]$.*

An dieser Stelle sei darauf hingewiesen, dass für die Linearität des Erwartungswerts *keine* Unabhängigkeit der Zufallsvariablen gefordert ist.

Beispiel. Wir betrachten das Werfen zweier Würfel, wobei die Zufallsvariable X die Augenzahl des ersten Wurfs und Y die Augenzahl des zweiten Wurfs bezeichnet. Die erwartete Summe der Augenzahlen beider Würfe ist wegen der Linearität des Erwartungswerts $\mathrm{Ex}[X + Y] = \mathrm{Ex}[X] + \mathrm{Ex}[Y] = 3.5 + 3.5 = 7$.

B.3 Stochastische Unabhängigkeit und bedingte Wahrscheinlichkeit

Zwei Ereignisse A und B heißen *(stochastisch) unabhängig* wenn $\Pr[A \cap B] = \Pr[A] \cdot \Pr[B]$ gilt. Allgemein ist eine endliche, nichtleere Menge $\mathcal{A} = \{A_1, \ldots, A_n\}$ von Ereignissen *(stochastisch) unabhängig*, wenn für jede nichtleere Teilmenge $I \subseteq \{1, \ldots, n\}$ gilt dass $\Pr[\bigcap_{i \in I} A_i] = \prod_{i \in I} \Pr[A_i]$.[7]

Beispiel. Sei beim Werfen eines Würfels $G = \{2, 4, 6\}$ jenes Ereignis, das den Wurf einer geraden Zahl repräsentiert und $Q = \{1, 4\}$ jenes Ereignis, das den Wurf einer Quadratzahl repräsentiert. Dann gilt $\Pr[G] = \frac{3}{6} = \frac{1}{2}$ und $\Pr[Q] = \frac{2}{6} = \frac{1}{3}$ und damit $\Pr[G] \cdot \Pr[Q] = \frac{1}{2} \cdot \frac{1}{3} = \frac{1}{6}$. Da $G \cap Q = \{4\}$ ist, gilt $\Pr[G \cap Q] = \frac{1}{6} = \Pr[G] \cdot \Pr[Q]$ und somit sind die Ereignisse G und Q unabhängig.

Diese Definitionen lassen sich direkt auf Zufallsvariablen erweitern: Zwei Zufallsvariablen X und Y heißen *(stochastisch) unabhängig*, wenn die Ereignisse $X = x$ und $Y = y$ für alle $x, y \in \mathbb{R}$ unabhängig sind, das heißt: $\Pr[X = x \cap Y = y] = \Pr[X = x] \cdot \Pr[Y = y]$ für alle $x, y \in \mathbb{R}$ gilt. Allgemein ist eine endliche, nichtleere Menge $\mathcal{X} = \{X_1, \ldots, X_n\}$ von Zufallsvariablen *(stochastisch) unabhängig*, wenn die Ereignisse $X_1 = x_1, \ldots, X_n = x_n$ für alle $x_1, \ldots, x_n \in \mathbb{R}$ unabhängig sind, das heißt:$\Pr[\bigcap_{i \in I} X_i = x_i] = \prod_{i \in I} \Pr[X_i = x_i]$ für jede nichtleere Teilmenge $I \subseteq \{1, \ldots, n\}$.

Für beliebige Ereignisse A und B mit $\Pr[B] > 0$ ist die *Wahrscheinlichkeit von A unter der Bedingung B* definiert als $\Pr[A \mid B] := \frac{\Pr[A \cap B]}{\Pr[B]}$. Somit gilt, dass zwei Ereignisse A und B mit $\Pr[B] > 0$ genau dann unabhängig sind, wenn $\Pr[A \mid B] = A$ gilt.

Beispiel. Sei beim Werfen eines Würfels $G = \{2, 4, 6\}$ jenes Ereignis, das den Wurf einer geraden Zahl repräsentiert, $Q = \{1, 4\}$ jenes Ereignis, das den Wurf einer Quadratzahl repräsentiert, und $P = \{2, 3, 5\}$ jenes Ereignis, das den Wurf einer Primzahl repräsentiert. Dann gilt $\Pr[Q \mid G] = \frac{\Pr[Q \cap G]}{\Pr[G]} = \frac{1/6}{1/2} = \frac{1}{3} = \Pr[Q]$ und $\Pr[P \mid G] = \frac{\Pr[P \cap G]}{\Pr[G]} = \frac{1/6}{1/2} = \frac{1}{3} \neq \frac{1}{2} = \Pr[P]$.

Lemma B.3 (Gesetz der totalen Wahrscheinlichkeit)**.** *Seien A und B beliebige Ereignisse und bezeichne $\bar{B}$ das Gegenereignis von B. Dann gilt* $\Pr[A] = \Pr[A \mid B] \cdot \Pr[B] + \Pr[A \mid \bar{B}] \cdot \Pr[\bar{B}]$.

B.4 Wahrscheinlichkeitsverteilungen

Definition B.4 (Uniforme Verteilung)**.** *Eine Zufallsvariable X heißt uniform verteilt (bzw. gleichverteilt) mit den Parametern $a \in \mathbb{N}$ und $b \in \mathbb{N}$, wobei $a \leq b$, ($X \sim U(a, b)$) falls*

$$\Pr[X = k] = \begin{cases} \frac{1}{n} & \text{falls } k \in \{a, a+1, \ldots, b\} \\ 0 & \text{ansonsten} \end{cases}$$

mit $n := b - a + 1$ gilt.

Eine uniform verteilte Zufallsvariable X beschreibt einen zufälligen Vorgang mit n möglichen Versuchsausgängen, die alle gleich wahrscheinlich sind.

Beispiel. Sei X die Augenzahl beim Werfen eines Würfels. Dann ist X uniform verteilt mit $a = 1$ und $b = 6$ (und somit $n = 6$) (siehe Abb. B.1).

Definition B.5 (Bernoulli-Verteilung)**.** *Eine Zufallsvariable X heißt Bernoulli-verteilt mit Parameter $p \in [0, 1]$ falls*

$$\Pr[X = k] = \begin{cases} p & \text{falls } k = 1 \\ 1 - p & \text{falls } k = 0 \\ 0 & \text{ansonsten} \end{cases}$$

[7] Die schwächere Bedingung, dass $\mathcal{A} = \{A_1, \ldots, A_n\}$ *paarweise (stochastisch) unabhängig* ist, lautet $\Pr[A_i \cap A_j] = \Pr[A_i] \cdot \Pr[A_j]$ für alle $i, j \in I$ so dass $i \neq j$.

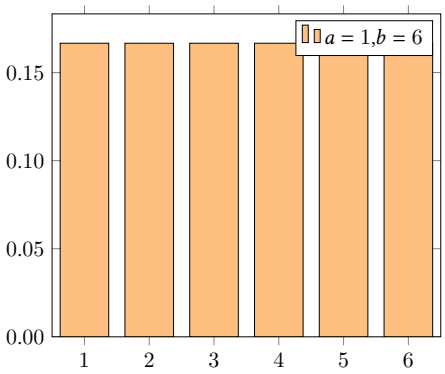


Abbildung B.1: Uniforme Verteilung mit $a = 1$ und $b = 6$ (und somit $n = 6$)

gilt. Dabei wird üblicherweise p als die Erfolgswahrscheinlichkeit bezeichnet.

Eine Bernoulli-verteilte Zufallsvariable X beschreibt einen zufälligen Vorgang, oft als Bernoulli-Versuch oder Bernoulli-Experiment bezeichnet, bei dem es nur zwei mögliche Versuchsausgänge gibt, wobei 1 als Erfolg des Versuchs und 0 als Misserfolg interpretiert wird.

Beispiel. Sei X jene Zufallsvariable, die 1 ist, wenn eine Sechs gewürfelt wurde und 0 ansonsten. Dann ist X Bernoulli-verteilt mit $p = \frac{1}{6}$ (siehe Abb. B.2).

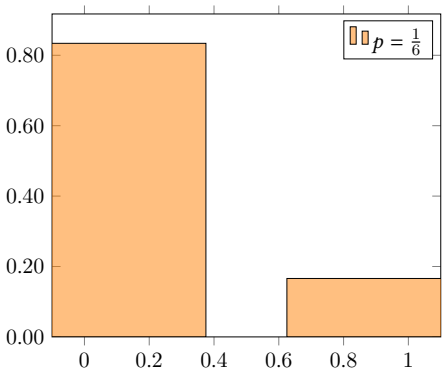


Abbildung B.2: Bernoulli-Verteilung mit $p = \frac{1}{6}$

Lemma B.6. *Sei X Bernoulli-verteilt mit Erfolgswahrscheinlichkeit p. Dann gilt* $\mathrm{Ex}[X] = p$.

Beweis. $\mathrm{Ex}[X] = 0 \cdot \Pr[X = 0] + 1 \cdot \Pr[X = 1] = \Pr[X = 1] = p$ □

Definition B.7 (Binomialverteilung)**.** *Sei $\{X_1, \ldots, X_n\}$ eine endliche Menge unabhängiger Zufallsvariablen, die jeweils Bernoulli-verteilt mit Parameter p sind, und sei $X = \sum_{i=1}^{n} X_i$. Dann heißt X heißt binomialverteilt mit den Parametern $n \in \mathbb{N}$ und $p \in [0, 1]$ ($X \sim B(n, p)$) und es gilt*

$$\Pr[X = k] = \begin{cases} \binom{n}{k} p^k (1-p)^{n-k} & \text{falls } k \in \{0, 1, \ldots, n\} \\ 0 & \text{ansonsten} \end{cases}.$$

Dabei wird üblicherweise n als die Anzahl an Versuchen und p als die Erfolgswahrscheinlichkeit bezeichnet.

Eine binomialverteilte Zufallsvariable X beschreibt die Anzahl an Erfolgen in einem Bernoulli-Prozess mit n Versuchen.

Beispiel. Sei X die Anzahl an Sechsen, die bei dreimaligem Würfeln gewürfelt werden. Dann ist X binomialverteilt mit $n = 3$ und $p = \frac{1}{6}$ (siehe Abb. B.3).

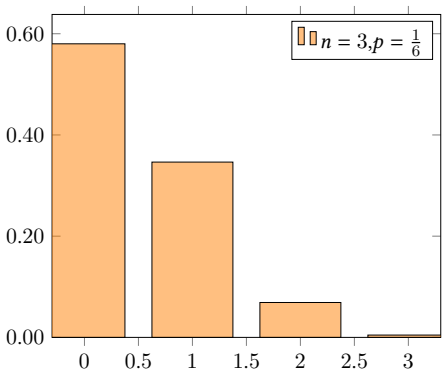


Abbildung B.3: Binomialverteilung mit $n = 3$ und $p = \frac{1}{6}$

Lemma B.8. *Sei X binomialverteilt mit den Parameter n und p. Dann gilt* $\mathrm{Ex}[X] = pn$.

Beweis. Es gilt $X = \sum_{i=1}^{n} X_i$ für Zufallsvariablen $X_1, \ldots, X_n$, die jeweils binomialverteilt mit Parameter p sind. Wegen Lemma B.6 gilt $\mathrm{Ex}[X_i] = p$ für alle $1 \leq i \leq n$. Aus der Linearität des Erwartungswerts (Lemma B.2) folgt nun $\mathrm{Ex}[X] = \mathrm{Ex}[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} \mathrm{Ex}[X_i] = \sum_{i=1}^{n} p = pn$. □

Definition B.9 (Geometrische Verteilung)**.** *Sei $\{X_1, X_2, \ldots\}$ eine abzählbar unendliche Menge unabhängiger Zufallsvariablen, die jeweils Bernoulli-verteilt mit Parameter p sind, und sei $X = \min\{i \geq 1 \mid X_i = 1\}$. Dann heißt X heißt geometrisch verteilt mit Parameter $p \in [0, 1]$ ($X \sim G(p)$) und es gilt.*

$$\Pr[X = k] = \begin{cases} p(1-p)^{k-1} & \text{falls } k \in \{1, 2, \ldots\} \\ 0 & \text{ansonsten} \end{cases}.$$

Dabei wird üblicherweise p als die Erfolgswahrscheinlichkeit bezeichnet.

Eine geometrisch verteilte Zufallsvariable X beschreibt die Anzahl an Bernoulli-Versuchen, die notwendig sind um einen Erfolg zu haben.

Beispiel. Sei X die Anzahl an Würfen, die benötigt werden um eine Sechs zu würfeln. Dann ist X geometrisch verteilt mit $p = \frac{1}{6}$ (siehe Abb. B.4).

Theorem B.10. *Sei X geometrisch verteilt mit Parameter p. Dann gilt* $\mathrm{Ex}[X] = \frac{1}{p}$.

Beweis. Die Wahrscheinlichkeit genau bei der k-ten Wiederholung erfolgreich zu sein ist $\Pr[X = k] = (1-p)^{k-1}p$, also die Wahrscheinlichkeit für $k-1$ aufeinanderfolgende Misserfolge gefolgt von einem Erfolg. Laut Definition des Erwartungswerts gilt $\mathrm{Ex}[X] = \sum_{k=1}^{\infty} k \cdot \Pr[x = k]$ (Summe über alle Ausprägungen der Zufallsvariable mal der dazugehörigen Wahrscheinlichkeit).

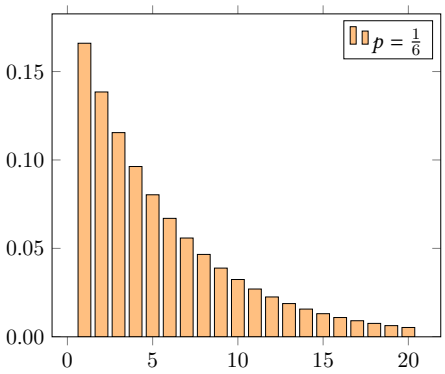


Abbildung B.4: Geometrische Verteilung mit $p = \frac{1}{6}$ (dargestellt für $k \leq 20$)

Wir führen nun folgende Umformungen durch:

$$\begin{aligned}
\mathrm{Ex}[X] = \sum_{k=1}^{\infty} k \cdot \Pr[X = k] &\overset{①}{=} \sum_{k=1}^{\infty} k(1-p)^{k-1}p \overset{②}{=} \sum_{k=0}^{\infty} (k+1)(1-p)^k p \\
&\overset{③}{=} \sum_{k=0}^{\infty} k(1-p)^k p + \sum_{k=0}^{\infty} (1-p)^k p \\
&\overset{④}{=} \sum_{k=1}^{\infty} k(1-p)^k p + \sum_{k=1}^{\infty} (1-p)^{k-1} p \\
&\overset{⑤}{=} (1-p) \sum_{k=1}^{\infty} k(1-p)^{k-1} p + \sum_{k=1}^{\infty} \Pr[X = k]
\end{aligned}$$

① Einsetzen

② Index-Shift

③ Ausmultiplizieren von $(k+1) \cdot \ldots$ und Aufteilung in zwei Summen

④ Bei der ersten Summe: für $k = 0$ ist der Summand gleich 0, somit kann man diese Summe auch bei 1 beginnen lassen
Bei der zweiten Summe: Index-Shift in die anderen Richtung

⑤ Bei der ersten Summe: Ausklammern von $1 - p$
Bei der zweiten Summe: die Definition von $\Pr[X = k]$ einsetzen

Nun entspricht die linke Summe in der letzten Zeile der Definition des Erwartungswerts und die rechte Summe ist, wie im Beweis von Theorem 1.10 argumentiert, gleich 1. Zur Erinnerung: Wir betrachten in der rechten Summe die Wahrscheinlichkeiten aller möglichen Ausprägungen von k, also die Wahrscheinlichkeiten, dass in der Runde k der erste Treffer erzielt wird, für alle Runden k. Diese Ereignisse sind disjunkt (der erste Treffer kann nicht in zwei unterschiedlichen Runden auftreten) und spannen den gesamten Ereignisraum auf (eines der Ereignisse *muss* auftreten, weil die Wartezeit eine natürliche Zahl größer gleich 1 sein muss).

Zusammenfassend gilt $E[X] = (1-q)E[X] + 1$, oder anders ausgedrückt $E[X] = \frac{1}{q}$ □

Theorem B.11 (Gedächtnislosigkeit der geometrischen Verteilung)**.** *Für jede geometrisch verteilte Zufallsvariable X und alle $m, n \geq 0$ gilt*

$$\Pr[X > m + n \mid X > m] = \Pr[X > n] .$$

Beispiel. Sei X die Anzahl an Würfen, die benötigt werden um eine Sechs zu würfeln. Dann gilt $\Pr[X > 3 \mid X > 2] = \Pr[X > 2 + 1 \mid X > 2] = \Pr[X > 1] = 1 - p = \frac{5}{6}$. Wenn also nach zwei Würfen keine Sechs gewürfelt wurde, dann ist die Wahrscheinlichkeit im nächsten Wurf eine Sechs zu würfeln gleich $\frac{1}{6}$.

B.5 Konzentrationsungleichungen

Für einige Analysen von Algorithmen ist der Erwartungswert nicht aussagekräftig genug. Es stellt sich oft die Frage, wie groß die Wahrscheinlichkeit ist, vom Erwartungswert abzuweichen. Hierbei ist es oft interessant, die Abweichung um einen *multiplikativen Faktor* abzuschätzen. Eine einfache Möglichkeit, solche Abschätzungen vorzunehmen, bietet die Markov-Ungleichung.

Theorem B.12 (Markov-Ungleichung). *) Sei X eine nicht-negative*[8] *Zufallsvariable und sei $\alpha > 0$ beliebig. Dann gilt*

$$\Pr[X \geq \alpha \cdot \mathrm{Ex}[X]] \leq \frac{1}{\alpha}.$$

Beispiel. Sei X die Augenzahl beim Werfen eines Würfels. Möchten wir abschätzen, wie wahrscheinlich es ist, mindestens vier Augen zu würfeln, liefert die Markov-Ungleichung – mit dem Wissen dass $\mathrm{Ex}[X] = 3.5$ gilt – $\Pr[X \geq 4] = \Pr[X \geq \frac{8}{7} \cdot \mathrm{Ex}[X]] \leq \frac{7}{8}$. Das korrekte Ergebnis wäre hingegen $\Pr[X \geq 4] = \frac{1}{2}$.

Eine weiteres wichtiges Werkzeug für solche Abschätzungen ist die sogenannte Chernoff-Schranke. Neben einer allgemeinen Formulierung der Chernoff-Schranke gibt es in der Literatur mehrere mehrere Spezialisierungen, die teilweise relativ starke Aussagen zulassen. Wir formulieren im Folgenden eine Chernoff-Schranke, die sich als besonders nützlich für die Analyse randomisierter Algorithmen erwiesen hat und sich insbesondere auf binomialverteilte Zufallsvariablen anwenden lässt. Abb. B.5 verdeutlicht visuell, wie sich eine binomialverteilte Zufallsvariable mit steigender Anzahl an Versuchen n um ihren Erwartungswert konzentriert.

Wir betrachten für die Chernoff-Schranke eine Zufallsvariable $X = \sum_{i=1}^{n} X_i$, die die Summe von n *unabhängigen* Bernoulli-verteilten (also binären) Zufallsvariablen X_i ist, deren Erfolgswahrscheinlichkeiten sich alle von oben (bzw. von unten) durch p beschränken lassen. Die Zufallsvariable X entspricht also der Anzahl an in den n Versuchen erzielten Erfolge. Die Chernoff-Schranke erlaubt dann eine Aussage darüber wie nahe X an $\mu := pn$ liegt. Falls die Erfolgswahrscheinlichkeiten der X_i's alle *gleich* p sind, so X binomialverteilt (mit Parametern n und p) und es gilt $\mu = \mathrm{Ex}[X]$.

Theorem B.13 (Chernoff-Schranke für oberen Rand). *Seien $X_1, \ldots, X_n \in \{0, 1\}$ unabhängige binäre Zufallsvariablen mit $\Pr[X_i = 1] \leq p$ und sei $\mu := pn$. Dann gilt Für jedes $\delta > 0$:*

$$\Pr\left[\sum_{i=1}^{n} X_i \geq (1 + \delta) \cdot \mu\right] \leq \frac{1}{e^{\frac{\min\{\delta, \delta^2\}}{3} \cdot \mu}}$$

Theorem B.14 (Chernoff-Schranke für unteren Rand). *Seien $X_1, \ldots, X_n \in \{0, 1\}$ unabhängige binäre Zufallsvariablen mit $\Pr[X_i = 1] \geq p$ und sei $\mu := pn$. Dann gilt für jedes $\delta \in [0, 1]$:*

$$\Pr\left[\sum_{i=1}^{n} X_i \leq (1 - \delta) \cdot \mu\right] \leq \frac{1}{e^{\frac{\delta^2}{2} \cdot \mu}}$$

[8] Die Zufallsvariable X ist nicht-negativ, falls für alle $x < 0$ gilt: $\Pr[X = x] = 0$.

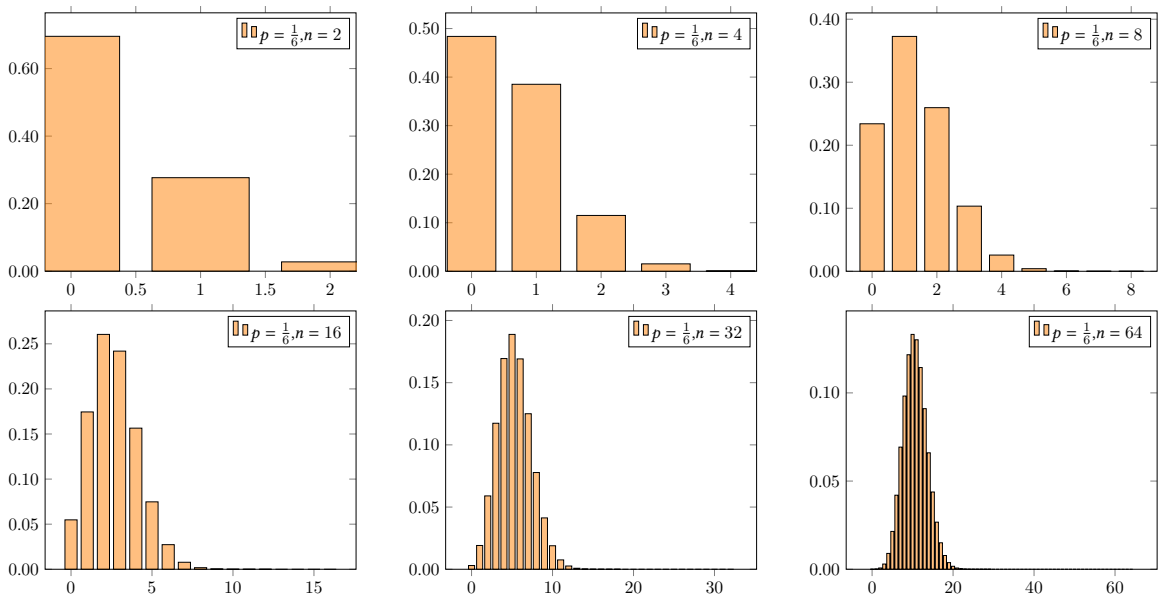


Abbildung B.5: Mit steigender Versuchsanzahl konzentrieren sich die Ausprägungen einer binomialverteilten Zufallsvariable (hier mit Erfolgswahrscheinlichkeit $p = \frac{1}{6}$) um ihren Erwartungswert