

Universität Stuttgart

Fakultät Informatik

Prüfer: Prof. Dr. rer. nat. Volker Diekert
Betreuer: Prof. Dr. rer. nat. Volker Diekert
begonnen am 18.11.1998
beendet am 29.3.1999
CR-Klassifikation: C.2.4, F.2.2, G.3

Diplomarbeit Nr. 1727

Verteilte Algorithmen zur Koordinatorwahl in Netzwerken

Holger Austinat

Hiermit versichere ich, die vorliegende Diplomarbeit selbständig und nur mit den angegebenen Hilfsmitteln durchgeführt zu haben.

Stuttgart, den 29. März 1999

Inhaltsverzeichnis

1	Einführung	1
1.1	Aufgabenstellung	1
1.2	Verteilte Algorithmen	1
1.3	Koordinatorwahl	2
1.3.1	Definition	2
1.3.2	Motivation	2
1.4	Komplexitätsmaße	3
1.5	Annahmen	4
1.5.1	Asynchrone Kommunikation	4
1.5.2	Fehlerfreiheit	4
1.5.3	Prozeßnamen	5
1.5.4	Netzwerktopologie	5
1.5.5	Initiierung eines Algorithmus	5
1.6	Pseudocode-Konventionen	5
1.7	Überblick	7
2	Koordinatorwahl in unidirektionalen Ringen	9
2.1	Korrektheit eines Algorithmus zur Koordinatorwahl	10
2.2	Der Algorithmus von Le Lann	11
2.3	Der Algorithmus von Chang und Roberts	11
2.3.1	Worst-Case-Nachrichtenkomplexität	12
2.3.2	Mittlere Nachrichtenkomplexität	12
2.4	Der Algorithmus von Peterson und Dolev et al.	14
2.5	Der modifizierte Algorithmus von Peterson	16

2.5.1	Korrektheit	17
2.5.2	Nachrichtenkomplexität	18
2.6	Der Algorithmus von Higham und Przytycka	19
2.6.1	Der Grundalgorithmus	20
2.6.2	Der vollständige Algorithmus	21
2.6.3	Nachrichtenkomplexität des vollständigen Algorithmus	23
3	Untere Schranken für die Koordinatorwahl in Ringen	29
4	Anonyme Netzwerke	35
4.1	Berechenbarkeit in anonymen Ringen	35
4.2	Koordinatorwahl in anonymen Ringen	38
4.3	Rendezvous in allgemeinen Graphen	40
4.4	Analyse des Rendezvous	41
4.5	Gegenbeispiel zu Behauptung 14 bei Métivier et al.	46
4.5.1	Erfolgreiche Anfragen unter G	47
4.5.2	Erfolgreiche Anfragen unter G'	48
4.5.3	Vergleich der Erfolgswahrscheinlichkeiten	49
4.6	Wahrscheinlichkeit eines Rendezvous in beliebigen Graphen	49
5	Graph-Faktorisierung ist NP-vollständig	57
5.1	Partitionierung ist NP-vollständig	57
5.2	Graph-Faktorisierung ist NP-vollständig	61
	Literaturverzeichnis	64

Kapitel 1

Einführung

1.1 Aufgabenstellung

Diese Diplomarbeit befaßt sich mit verteilten Algorithmen zur Bestimmung eines Koordinators in einem Netzwerk (und einigen eng verwandten Problemen).

Betrachtet werden sowohl Netzwerke, in denen jeder Prozessor durch einen eindeutigen Namen identifiziert werden kann, also auch sog. anonyme Netzwerke, in denen die einzelnen Prozessoren nicht durch Namen unterschieden werden können. Dies erfordert die Untersuchung sowohl deterministischer als auch probabilistischer Algorithmen. Ferner werden die Unlösbarkeit gewisser Aufgabenstellungen bzw. untere Schranken für deren Aufwand aufgezeigt.

Diese Arbeit soll außerdem dazu beitragen, die sehr unterschiedlichen Darstellungen in der Primärliteratur zu vereinheitlichen.

1.2 Verteilte Algorithmen

Verteilte Algorithmen sind Algorithmen, bei denen mehrere Prozesse parallel an der Lösung einer Aufgabe arbeiten. Dabei ist es aus theoretischer Sicht völlig unerheblich, ob es sich um ein Computernetzwerk (bestehend aus mehreren Rechnern, die über Netzwerkverbindungen wie Ethernet miteinander kommunizieren), einen Parallelrechner (bei dem mehrere Prozessoren parallel arbeiten und über Nachrichten oder einen gemeinsamen Speicherbereich kommunizieren) oder ein Multitasking-Betriebssystem auf einem Einprozessorcomputer handelt.

All diese Systeme werden durch einen ungerichteten Graphen $G = (V, E)$ mit Knotenmenge V und Kantenmenge E beschrieben. Die Knotenmenge entspricht dabei den Prozessen bzw. Prozessoren (im folgenden wird ausschließlich der Begriff Prozesse verwendet), und eine Kante $e = \{u, v\} \in E$ gibt an, daß die beiden Prozesse u und v durch einen Kommunikationskanal miteinander verbunden sind und damit Nachrichten direkt austauschen können. Ist die Kommunikation nur in einer Richtung möglich, so wird dies durch einen gerichteten Graphen modelliert. Die Kanten werden dann mit $\langle u, v \rangle$ bezeichnet (mit der Bedeutung, daß u an v eine Nachricht schicken kann, nicht aber umgekehrt).

1.3 Koordinatorwahl

1.3.1 Definition

Die vorliegende Diplomarbeit befaßt sich hauptsächlich mit dem Problem der Koordinatorwahl.

Definition 1.1 (Koordinatorwahl) Ein Algorithmus A löst das Problem der *Koordinatorwahl* in einem Netzwerk $G = (V, E)$, wenn A genau einen Prozeß $p \in V$ auswählt und dieser von seiner Wahl weiß. \square

Natürlich sollten die anderen Prozesse im Netzwerk ebenfalls von dieser Wahl wissen. Geschieht dies nicht automatisch durch Ablauf des Algorithmus, dann muß der gewählte Koordinator seine Wahl propagieren, indem er eine Nachricht im Netzwerk verteilt (sog. *Broadcasting*, z.B. mittels eines Tiefensuchalgorithmus).

In der englischsprachigen Originalliteratur werden für die Koordinatorwahl mehrere Begriffe verwendet: *Leader-Election*, *Extrema-Finding* und *Ranking*. Diese Probleme sind alle eng miteinander verwandt:

- Das *Leader-Election*-Problem entspricht obiger Definition der Koordinatorwahl.
- Das *Extrema-Finding*-Problem hat hingegen zum Ziel, den Prozeß mit dem größten bzw. kleinsten Bezeichner auszuwählen. Ausgehend von einem nicht-extremalen Koordinator läßt sich ein Extremum aber leicht mit einem Tiefensuchalgorithmus finden.
- Das *Ranking*-Problem will sämtliche Prozesse in eine Rangfolge stellen (anstatt nur einen herausragenden Prozeß zu ermitteln). Dies kann ebenfalls mit einem Tiefensuchalgorithmus bewerkstelligt werden, wenn erst einmal ein Koordinator gewählt ist.

Da sich die anderen Probleme leicht auf das Problem der Koordinatorwahl reduzieren lassen, werden sie in dieser Arbeit nicht gesondert behandelt. Die beschriebenen Algorithmen terminieren in der Regel in dem Zustand, in dem der Koordinator von seiner Wahl weiß, diese aber noch nicht im Netzwerk propagiert hat.

1.3.2 Motivation

Zwei Fälle sind denkbar, in denen ein Koordinator (d.h. ein herausragender Prozeß) nötig ist:

1. Verteilte Algorithmen, in denen ein Prozeß zentrale Aufgaben übernehmen muß (z.B. die Verteilung der anstehenden Arbeit an die anderen Prozesse im System). Dieser Prozeß behält seine herausragende Position in der Regel über die gesamte Laufzeit des Algorithmus (oder sogar über die gesamte Lebensdauer des Systems).

2. Systeme, in denen sich die Prozesse eine Ressource teilen, auf die ein paralleler Zugriff vermieden werden muß. Zu diesem Zweck können die Prozesse in einem (virtuellen) Ring angeordnet werden, in dem eine sog. Kontrollmarke (*control token*) kreist. Dann hat nur der Prozeß, der sich im Besitz der Kontrollmarke befindet, Zugriff auf die Ressource. Hier wechselt die herausragende Position also ständig (womit der Begriff Koordinator nicht ganz zutreffend ist).

Jetzt kann es aber passieren, daß der zentrale Prozeß bzw. der Prozeß, der gerade die Kontrollmarke besitzt, ausfällt oder die Verbindung zu ihm unterbrochen wird. Dann befindet sich das System in einem blockierten Zustand. Dies läßt sich in beiden Fällen feststellen:

1. Der zentrale Prozeß muß in regelmäßigen Abständen Kontrollnachrichten an seine Nachbarn schicken. Außerdem muß eine obere Schranke für die Dauer des Nachrichtentransports bekannt sein. Dann können die Nachbarprozesse feststellen, daß der Koordinator ausgefallen ist, wenn sie eine gewisse Zeit lang keine Kontrollnachricht von ihm erhalten haben.
2. Jeder Prozeß verschickt in regelmäßigen Abständen eine Nachricht mit seinem eigenen Bezeichner im Ring, die von den anderen Prozessen weitergeleitet wird. Erhält er diese Nachricht zurück, bevor er die Kontrollmarke erhalten hat, muß der Prozeß mit der Kontrollmarke ausgefallen sein. (Wir nehmen an, daß Nachrichten in der Reihenfolge empfangen werden, in der sie verschickt wurden; vgl. Abschnitt 1.5.1).

An diesem Punkt setzen die Algorithmen zur Koordinatorwahl auf. Sie dienen dazu, den blockierten Zustand wieder aufzuheben, indem sie einen neuen Prozeß ermitteln, der die zentralen Aufgaben übernimmt bzw. die Kontrollmarke erhält.

1.4 Komplexitätsmaße

Die Komplexitätsmaße herkömmlicher Algorithmen (Zeit- und Platzbedarf) lassen sich nicht ohne weiteres auf verteilte Algorithmen übertragen. Für diese sind hingegen folgende Abschätzungen üblich (siehe z.B. [Lamport and Lynch, 1990]):

- **Nachrichtenkomplexität**

Das wichtigste Komplexitätsmaß ist die *Nachrichtenkomplexität*. Sie entspricht der Anzahl der Nachrichtentransporte im gesamten System, also der Gesamtzahl verschickter Nachrichten.

- **Zeitkomplexität**

Die Angabe einer *Zeitkomplexität* ist nur unter zwei Annahmen sinnvoll:

1. Die Zeit des Nachrichtentransports übersteigt die Zeit lokaler Berechnungen deutlich, womit letztere vernachlässigt werden können.
2. Die Dauer eines Nachrichtentransports ist durch eine Konstante τ begrenzt.

Unter diesen Voraussetzungen kann die maximal benötigte Zeit als Vielfaches von τ ausgedrückt werden. Zur Bestimmung der Zeitkomplexität ist die längste Nachrichtenkette zu bestimmen, deren Länge multipliziert mit τ dann den gewünschten Wert ergibt.

- **Bitkomplexität**

Verschiedene Algorithmen benötigen in der Regel unterschiedlich viel Informationen pro Nachricht. Oftmals wird eine geringere Nachrichtenkomplexität nur durch eine größere Anzahl verschiedener Nachrichten erreicht. Von daher erscheint es sinnvoll, das Produkt aus Nachrichtenkomplexität und Nachrichtengröße (in Bit) als Komplexitätsmaß zu verwenden.

1.5 Annahmen

1.5.1 Asynchrone Kommunikation

In dieser Arbeit werden ausschließlich Algorithmen betrachtet, die auf einem *asynchronen* Kommunikationssystem beruhen. Schickt ein Prozeß p eine Nachricht an einen anderen Prozeß q , so wird diese vom Kommunikationssystem so lange gepuffert, bis q sämtliche zuvor erhaltenen Nachrichten abgearbeitet hat. Der Nachrichtentempuffer ist als FIFO-Queue (*first in first out*) organisiert, d.h. die Reihenfolge ankommender Nachrichten wird stets beibehalten. Außerdem nehmen wir an, daß der Nachrichtentempuffer eine unbegrenzte Kapazität besitzt.

Im Gegensatz dazu kann bei einem *synchronen* Kommunikationssystem ein Sendebefehl erst dann ausgeführt werden, wenn der Zielprozeß zum Empfang bereit ist — Sender und Empfänger werden stets synchronisiert. Später werden wir sehen, daß ein synchrones Kommunikationssystem mächtiger als ein asynchrones ist (Abschnitt 4.1).

Die Korrektheitsbeweise vieler Algorithmen beruhen auf einer Ausführung in Runden oder Phasen. Da die Ausführung der Algorithmen lediglich von der Art und Reihenfolge eingehender Nachrichten abhängt (weswegen diese auch als *nachrichtengetrieben* bezeichnet werden), ist jede Ausführung eines asynchronen Algorithmus gleichwertig — insbesondere also auch die Ausführung in Runden.

Unter einem asynchronen Kommunikationssystem kann eine synchrone Ausführung simuliert werden, die zur Unterscheidung als *pseudosynchron* bezeichnet wird. Dazu wird die Ausführung in Runden unterteilt, in denen jeder Prozeß maximal eine Nachricht an jeden seiner Nachbarn sendet. Zu Beginn einer Runde verschickt jeder Prozeß außerdem Synchronisationsnachrichten an jeden seiner Nachbarn. Er fährt erst mit der nächsten Runde fort, wenn er seinerseits von jedem seiner Nachbarn je eine Synchronisationsnachricht erhalten hat. An dieser Stelle sei ausdrücklich darauf hingewiesen, daß diese Simulation nur der Erleichterung von Berechenbarkeitsbeweisen dient und nicht die Gleichwertigkeit zu synchronen Kommunikationssystemen bewirkt (deswegen auch die Begriffswahl *pseudosynchron*).

1.5.2 Fehlerfreiheit

Es wird angenommen, daß das System fehlerfrei arbeitet. Genauer heißt das:

- Prozesse fallen nicht aus.
- Nachrichten gehen weder verloren, noch ändert sich ihr Inhalt.

- Auf den Nachrichtenkanälen findet keine Vertauschung von Nachrichten statt (FIFO–Eigenschaft).
- Nachrichten kommen nach endlicher Zeit am Zielprozeß an (d.h. es existiert eine obere Schranke für die maximale Dauer einer Nachrichtenübermittlung).

Für eine allgemeine Einführung in fehlertolerante verteilte Systeme und Algorithmen sei auf [Tel, 1994, Kapitel 12–15] verwiesen. Eine Behandlung der Koordinatorwahl in nicht fehlerfreien Systemen findet sich z.B. in [Garcia-Molina, 1982].

1.5.3 Prozeßnamen

Es werden zwei Typen von Netzwerken unterschieden: *benannte Netzwerke*, bei denen jeder Prozeß durch einen eindeutigen Namen identifiziert ist (der im folgenden ID genannt wird), und *anonyme Netzwerke*, bei denen die einzelnen Prozesse nicht durch Namen unterscheidbar sind. Wie wir sehen werden, hat dieser Unterschied deutliche Auswirkungen auf die Berechenbarkeit verschiedener Funktionen und die Art der verwendeten Algorithmen.

1.5.4 Netzwerktopologie

Die Koordinatorwahl wird in dieser Diplomarbeit ausschließlich auf Ringen betrachtet. Es gibt zwar einige Artikel, in denen die Koordinatorwahl auf andere Topologien untersucht wird (wenn auch bei weitem nicht so viele wie für Ringe), doch deren Beschreibung würde Rahmen dieser Diplomarbeit sprengen.

Allgemeine Graphen werden nur bei der Behandlung des Rendezvous–Verfahrens in Abschnitt 4.3 und beim NP–Vollständigkeitsbeweis in Kapitel 5 betrachtet.

1.5.5 Initiierung eines Algorithmus

Die Anzahl der Prozesse, die einen verteilten Algorithmus starten, ist ungewiß. Es wird dabei immer vorteilhaft sein, wenn diese Anzahl so klein wie möglich ist. Dann ist es nämlich möglich, daß ein Nicht–Initiator eine Nachricht erhält, die ihn bereits von den weiteren Berechnungen ausschließt — dieser Prozeß muß folglich keine eigene Nachricht mehr schicken, die evtl. weitere unnötige Berechnungen zur Folge hätte.

Von daher wird der „Worst Case“ immer dann eintreten, wenn alle Prozesse als Initiatoren auftreten. In sämtlichen Algorithmen in dieser Arbeit wird folglich die Annahme getroffen, daß die Initiierung durch alle Prozesse stattfindet.

1.6 Pseudocode–Konventionen

Die Algorithmen in dieser Arbeit sind in einem einheitlichen Pseudocode dargestellt. Das Grundgerüst eines Algorithmus hat folgendes Aussehen:

```

(* Algorithmus für Prozeß id *)

(* Globale Variablen *)
var
  ⋮

(* Hauptprogramm *)
begin
  ⋮
end

receive <message> from process
  ⋮
end

```

Bei Algorithmen in benannten Netzwerken enthält die erste Zeile den Prozeßnamen, der stets *id* ist. Dann folgen globale Variablendeklarationen, die die Form

$$\langle type \rangle \langle name \rangle \mathbf{init} \langle startvalue \rangle$$

haben, also z.B. „**int** *number* **init** 0“ für eine Integer-Variablen *number* mit Startwert 0. Es folgt das Hauptprogramm, das in den meisten Fällen lediglich aus einem **send**-Befehl besteht. Schließlich kommen ein oder mehrere **receive**-Prozeduren, die bei Ankunft einer passenden Nachricht ausgeführt werden. Lokale Variablen in den **receive**-Prozeduren werden in der Regel nicht explizit deklariert. Zwei **receive**-Befehle eines Prozesses werden nie parallel ausgeführt, d.h. die nächste empfangene Nachricht wird erst dann bearbeitet, wenn die vorherige **receive**-Prozedur komplett abgearbeitet ist.

Die Kommunikationsbefehle haben folgende Form:

$$\mathbf{send} \langle type, par_1, par_2, \dots \rangle \mathbf{to} \textit{process}$$

$$\mathbf{receive} \langle type, par_1, par_2, \dots \rangle \mathbf{from} \textit{process}$$

Jede Nachricht hat also einen Typ (in dieser Arbeit stets ein Name) und eine (feste) Anzahl von Parametern (die auch 0 sein darf). Beim **send**-Befehl wird in *process* der Zielprozeß angegeben, beim **receive**-Befehl der Sendeprozeß in der Variablen *process* übermittelt. Von dieser Grundform wird teilweise abgewichen:

- In unidirektionalen Ringen hat jeder Prozeß jeweils genau einen Nachbarn, von dem er eine Nachricht empfangen bzw. zu dem er eine Nachricht schicken kann. Die Angabe des Prozeßnamens ist deshalb überflüssig (d.h. **to/from process** entfällt).
- Gibt es nur eine Art von Nachrichten, wird *type* ebenfalls weggelassen.

- In anonymen Netzwerken können die Nachrichten nicht direkt an Prozesse adressiert werden, da diese keinen eindeutigen Namen besitzen. Deshalb tritt an die Stelle der Prozeßnamen eine lokale Numerierung der Kommunikationskanäle.

Die restlichen Konstrukte entsprechen denen gängiger imperativer Programmiersprachen und sollten selbsterklärend sein:

- **if ... then ... [elsif ...]* [else ...] fi**
- **for ... to ... do ... next**
- Zuweisung: $var := value$
- Vergleich: $value_1 \circ value_2$ mit $\circ \in \{=, \neq, <, \leq, >, \geq, \dots\}$
- Boolesche Operatoren: **and, or, not, even/odd** (Argument gerade/ungerade?)
- **max**(e_1, e_2, \dots), **min**(e_1, e_2, \dots) (Maximum/Minimum von ≥ 2 Elementen)

1.7 Überblick

Der Rest dieser Diplomarbeit gliedert sich in 4 Kapitel.

In Kapitel 2 werden Algorithmen diskutiert, die die Koordinatorwahl in unidirektionalen, benannten Ringen lösen.

In Kapitel 3 werden zwei untere Schranken für die Koordinatorwahl in Ringen bewiesen.

In Kapitel 4 werden anonyme Netzwerke untersucht. Dieses Kapitel enthält allgemeine Ergebnisse über die Berechenbarkeit in Ringen, einen probabilistischen Algorithmus zur Lösung der Koordinatorwahl und eine ausführliche Diskussion eines Rendezvous-Verfahrens für allgemeine Graphen.

In Kapitel 5 wird schließlich die NP-Vollständigkeit der Graph-Faktorisierung bewiesen. (Als Nebenergebnis fällt dabei die NP-Vollständigkeit der Partitionierung einer Menge von Gewichten in gleich schwere Teilmengen an).

Kapitel 2

Koordinatorwahl in unidirektionalen Ringen

In diesem Kapitel werden Algorithmen vorgestellt, die das Problem der Koordinatorwahl in unidirektionalen Ringen lösen. Die Algorithmen werden in der Reihenfolge ihrer Veröffentlichung beschrieben.

Das Problem der Koordinatorwahl (in unidirektionalen Ringen) wurde 1977 von Gérard Le Lann aufgebracht [Le Lann, 1977]. Er gab zugleich eine sehr einfache Lösung, bei der stets $\mathcal{O}(n^2)$ Nachrichten verschickt werden.

1979 veröffentlichten Ernest Chang und Rosemary Roberts eine modifizierte Version von Le Lanns Algorithmus, der im Mittel mit $\mathcal{O}(n \log n)$ Nachrichten auskommt [Chang and Roberts, 1979]. An der Worst-Case-Komplexität von $\mathcal{O}(n^2)$ ändert sich hingegen nichts.

Eine Zeitlang war es ungewiß, ob die Worst-Case-Komplexität verbessert werden könnte — die Vermutung kam auf, daß $\Omega(n^2)$ die untere Schranke für dieses Problem sein könnte. Erst gute 3 Jahre später veröffentlichten Gary Peterson sowie Danny Dolev, Maria Klawe und Michael Rodeh unabhängig voneinander einen Algorithmus, der auch im ungünstigsten Fall mit $\mathcal{O}(n \log n)$ Nachrichten auskommt [Peterson, 1982, Dolev et al., 1982].

Nachdem Jan Pachl, Ephraim Korach und Doron Rotem 1984 $\Omega(n \log n)$ als untere Schranke für die Koordinatorwahl in unidirektionalen Ringen beweisen konnten ([Pachl et al., 1984]; siehe Kapitel 3), blieb lediglich Spielraum für eine kleinere Konstante. Diese wurde von Peterson und Dolev et al. in mehreren Schritten von anfänglich 2 auf 1,356.. gedrückt. Erst knappe 10 Jahre später wurde die Konstante von Lisa Higham und Teresa Przytycka auf den bislang besten bekannten Wert von 1,270.. verbessert [Higham and Przytycka, 1993].

Im vorliegenden Kapitel werden folgende Annahmen getroffen:

- Jeder Prozeß hat einen eindeutigen Namen (im folgenden ID genannt), den außer ihm selbst keiner kennt. Der Einfachheit halber wird angenommen, daß die IDs ganze Zahlen aus dem Bereich $[1, N]$, $N \geq n$ sind, wobei die maximal mögliche ID N den Prozessen nicht bekannt ist.

- Abgesehen von der eindeutigen ID ist der Algorithmus für alle Prozesse identisch.
- Die Kommunikationskanäle arbeiten nach dem FIFO-Prinzip (first in, first out), d.h. die Nachrichten kommen stets in der Reihenfolge an, in der sie gesendet wurden.
- Alle Prozesse und das Kommunikationssystem arbeiten fehlerfrei, es gibt also keine Prozeßausfälle oder verlorene oder veränderte Nachrichten.
- Die Prozesse laufen asynchron und haben keinen Zugriff auf eine globale Uhr.

2.1 Korrektheit eines Algorithmus zur Koordinatorwahl

Die in den Originalarbeiten gegebenen Korrektheitsbeweise beruhen größtenteils auf Ad-hoc-Argumenten, die sich stark am jeweiligen Algorithmus orientieren. Higham und Przytycka haben in ihrem Artikel hingegen drei Kriterien aufgestellt, die eine systematische Beweisführung ermöglichen [Higham and Przytycka, 1993]; je nach Sichtweise beziehen sich diese Kriterien auf Nachrichten oder Prozesse (letztere werden in aktive und passive Prozesse unterteilt, da sie nicht gelöscht werden können):

- **Sicherheit** (*safety*)
Der Algorithmus darf zu keinem Zeitpunkt sämtliche Nachrichten löschen (bzw. sämtliche Prozesse passiv werden lassen).
- **Fortschritt** (*progress*)
Der Algorithmus muß die Anzahl verbleibender Nachrichten (bzw. die Anzahl aktiver Prozesse) stetig verringern.
- **Korrekte Terminierung** (*correct termination*)
Die Koordinatorwahl ist eindeutig, d.h. es darf keine zwei Prozesse geben, die gleichzeitig zum Koordinator gewählt werden.

Offensichtlich genügt der Nachweis dieser 3 Bedingungen für die Korrektheit eines Algorithmus zur Koordinatorwahl. Die Fortschrittsbedingung besagt, daß die Anzahl der Nachrichten (bzw. die Anzahl aktiver Prozesse) streng monoton abnimmt; die Sicherheitsbedingung besagt, daß aber trotzdem zu keinem Zeitpunkt alle Nachrichten gelöscht (bzw. alle aktiven Prozesse passiv) werden; und die Terminierungsbedingung sichert letztendlich zu, daß nie zwei Prozesse gleichzeitig zu Koordinatoren gewählt werden können.

Die Korrektheitsbeweise der Algorithmen von Chang und Roberts, Peterson sowie Dolev et al. werden unter diesen Gesichtspunkten reinterpretiert (obige drei Kriterien werden nicht auf den Algorithmus von Le Lann angewandt, da dies den Korrektheitsbeweis unnötig verkomplizieren würde).

Es sei noch angemerkt, daß die Algorithmen in diesem Kapitel an dem Punkt enden, an dem der Koordinator von seiner Wahl weiß, die anderen Prozesse aber evtl. noch keine Kenntnis davon haben (sie wissen nur, daß sie selbst nicht gewählt sind). Exakte Komplexitätsabschätzungen müssen in solchen Fällen noch n weitere Nachrichten für die Propagierung des Koordinators berücksichtigen.

2.2 Der Algorithmus von Le Lann

Der erste Algorithmus zur Koordinatorwahl wurde von Le Lann vorgeschlagen [Le Lann, 1977]. Bei ihm schickt jeder Prozeß zunächst eine Nachricht mit seiner ID an seinen Nachbarn. Anschließend leitet er ankommende Nachrichten ohne Modifikation so lange weiter, bis er seine eigene Nachricht (also die mit seiner eigenen ID) wieder erhält. Der Prozeß wird genau dann zum Koordinator, wenn seine ID größer als alle anderen weitergeleiteten IDs ist (siehe Alg. 2.1).

<pre> (* Algorithmus für Prozeß <i>id</i> *) (* Globale Variablen *) var int leader init <i>id</i> state ∈ {ACTIVE, PASSIVE, LEADER} init ACTIVE (* Hauptprogramm *) begin send ⟨<i>id</i>⟩ end </pre>	<pre> receive ⟨<i>i</i>⟩ if <i>i</i> = <i>id</i> then if leader = <i>id</i> then state := LEADER else state := PASSIVE fi else if <i>i</i> > leader then leader := <i>i</i> fi send ⟨<i>i</i>⟩ fi end </pre>
---	---

Algorithmus 2.1: Der Algorithmus von Le Lann.

Aufgrund der FIFO-Eigenschaft ist gesichert, daß vor dem Empfang der eigenen Nachricht eine Nachricht jedes anderen Prozesses angekommen sein muß. Da nur ein Prozeß die maximale ID besitzt, ist die Korrektheit dieses Algorithmus offensichtlich.

Die Nachrichtenkomplexität läßt sich ebenso einfach bestimmen. Jeder Prozeß sendet eine Nachricht mit seiner eigenen ID und leitet für jeden der $n - 1$ anderen Prozesse jeweils eine Nachricht weiter. Pro Prozeß werden also genau n Nachrichten gesendet, insgesamt also $n^2 = \mathcal{O}(n^2)$.

2.3 Der Algorithmus von Chang und Roberts

Chang und Roberts modifizierten Le Lanns Algorithmus leicht und erzielten damit eine bessere mittlere Nachrichtenkomplexität [Chang and Roberts, 1979]. Sie machten die Beobachtung, daß es offensichtlich keinen Sinn hat, ein Nachricht mit kleinerer ID als der eigenen weiterzuleiten. Mit dieser Änderung entsteht Alg. 2.2.

Zur Korrektheit überprüfen wir die drei Bedingungen Sicherheit, Fortschritt und korrekte Terminierung:

- *Sicherheit:* Es werden nie alle Nachrichten gelöscht, da die Nachricht mit maximaler ID stets weitergeleitet wird.
- *Fortschritt:* Betrachtet man die Ausführung des Algorithmus in Runden, so wird in jeder dieser Runde mindestens eine Nachricht gelöscht (nämlich die, die der Prozeß mit maximaler ID erhält).

<pre> (* Algorithmus für Prozeß <i>id</i> *) (* Globale Variablen *) var int leader init <i>id</i> state ∈ {ACTIVE, PASSIVE, LEADER} init ACTIVE (* Hauptprogramm *) begin send ⟨<i>id</i>⟩ end </pre>	<pre> receive ⟨<i>i</i>⟩ if <i>i</i> = <i>id</i> then state := LEADER elsif <i>i</i> > <i>id</i> then state := PASSIVE send ⟨<i>i</i>⟩ leader := <i>i</i> fi end </pre>
---	--

Algorithmus 2.2: Der Algorithmus von Chang und Roberts.

- *Korrekte Terminierung:* Die einzige Nachricht, die wieder ihren Initiator erreicht, ist die mit maximaler ID. Alle anderen werden spätestens bei Passieren des Prozesses mit maximaler ID verworfen.

2.3.1 Worst-Case-Nachrichtenkomplexität

Offensichtlich gilt die gleiche obere Schranke wie für Le Lanns Algorithmus, also $\mathcal{O}(n^2)$. Das folgende Beispiel zeigt, daß diese Schranke auch tatsächlich erreicht werden kann:

Beispiel: Sei R ein Ring mit n Prozessen $\{1, \dots, n\}$, deren IDs absteigend angeordnet sind, also z.B. $id(1) = n, \dots, id(n) = 1$ (vgl. Abb. 2.1(a)). Dann wird die Nachricht $\langle n \rangle$ genau n mal weitergeleitet, die Nachricht $\langle n-1 \rangle$ genau $n-1$ mal, usw., und schließlich die Nachricht $\langle 1 \rangle$ genau 1 mal. Damit ergibt sich eine Nachrichtenkomplexität von

$$\sum_{i=1}^n i = \frac{n(n+1)}{2} = \mathcal{O}(n^2).$$

■

Im ungünstigsten Fall hat der Algorithmus von Chang und Roberts gegenüber dem von Le Lann also kaum Vorteile.

2.3.2 Mittlere Nachrichtenkomplexität

Ein weiteres Beispiel belegt hingegen, daß es auch deutlich günstigere Fälle gibt:

Beispiel: Sind die IDs in R nicht absteigend, sondern aufsteigend angeordnet ($id(1) = 1, \dots, id(n) = n$, vgl. Abb. 2.1(b)), so werden alle Nachrichten außer $\langle n \rangle$ bereits beim nächsten Prozeß verworfen (jeweils 1 verschickte Nachricht). Die Nachricht $\langle n \rangle$ wird einmal im Ring herumgereicht (insgesamt n Nachrichten). Die Nachrichtenkomplexität beträgt somit

$$\left(\sum_{i=1}^{n-1} 1 \right) + n = 2n - 1 = \mathcal{O}(n).$$

■

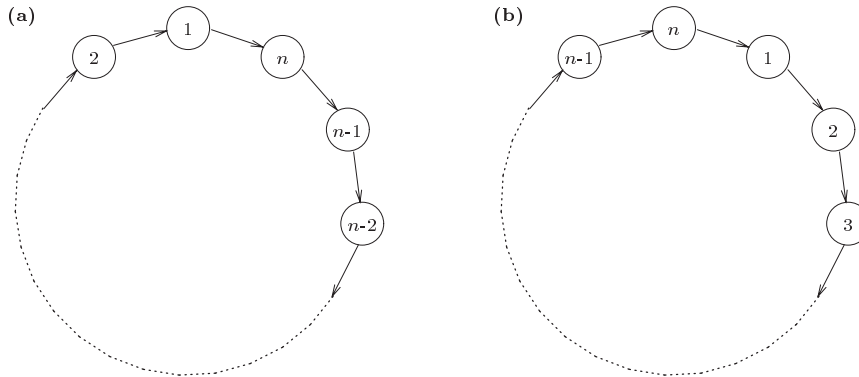


Abbildung 2.1: (a) Ungünstigster und (b) günstigster Fall für den Algorithmus von Chang und Roberts.

Dieses Beispiel läßt die Vermutung aufkommen, daß eine Verbesserung der mittleren Nachrichtenkomplexität gegenüber Le Lann's Algorithmus vorliegt. Um diese zu ermitteln, muß die Anzahl der Nachrichten über alle möglichen Verteilungen (Permutationen) der IDs gemittelt werden.

Die von Chang und Roberts durchgeführte Berechnung der mittleren Nachrichtenkomplexität ist relativ lang und mühsam, wenn sie ausführlich dargestellt werden soll. Ich gebe deshalb einen Beweis aus [Tel, 1994] wieder, der auf einem Vorschlag Friedemann Matterns beruht.

Satz 2.1 Die mittlere Nachrichtenkomplexität des Chang–Roberts–Algorithmus (Alg. 2.2) beträgt $\mathcal{O}(n \log n)$.

Beweis: Sei p_{max} der Prozeß mit der größten ID und p_i der Prozeß, der i Schritte vor p_{max} liegt. Die Nachricht von p_i kann offensichtlich maximal i mal weitergeleitet werden, da sie spätestens von p_{max} verworfen wird. Uns interessiert nun die Wahrscheinlichkeit $Pr(i, k)$, mit der p_i genau k mal weitergeleitet wird.

Die Nachricht $\langle p_i \rangle$ wird *mindestens* k mal weitergeleitet, wenn die nachfolgenden $k-1$ Prozesse $p_{i-1}, \dots, p_{i-k+1}$ alle eine kleinere ID haben, wenn also p_i die maximale ID der Prozesse p_i, \dots, p_{i-k+1} besitzt. Die Wahrscheinlichkeit dafür ist $1/k$.

Die Nachricht $\langle p_i \rangle$ wird *genau* k mal weitergeleitet, wenn $\langle p_i \rangle$ mindestens k mal, aber *nicht* mindestens $k+1$ mal weitergeleitet wird, also mit Wahrscheinlichkeit $Pr(i, k) = \frac{1}{k} - \frac{1}{k+1}$ ($k < i$). Für $k = i$ wird die Nachricht mit Wahrscheinlichkeit $\frac{1}{i}$ mindestens i mal weitergeleitet und mit Wahrscheinlichkeit 0 mindestens $i+1$ mal, also mit Wahrscheinlichkeit $\frac{1}{i}$ *genau* i mal.

Damit können wir den Erwartungswert E_i dafür bestimmen, wie häufig $\langle p_i \rangle$ weitergeleitet wird:

$$\begin{aligned}
 E_i &= \left[\sum_{k=1}^{i-1} k \cdot Pr(i, k) \right] + i \cdot \frac{1}{i} \\
 &= \left[\sum_{k=1}^{i-1} k \cdot \left(\frac{1}{k} - \frac{1}{k+1} \right) \right] + 1 \\
 &= \left[\sum_{k=1}^{i-1} \frac{1}{k+1} \right] + 1 = \left[\sum_{k=2}^i \frac{1}{k} \right] + 1 = \left[\sum_{k=1}^i \frac{1}{k} \right] = H_i,
 \end{aligned} \tag{2.1}$$

wobei H_i die i -te **Harmonische Zahl** ist, deren Wert $H_i = \ln n + \mathcal{O}(1)$ beträgt (siehe z.B. [Knuth, 1997, S. 75]).

Der Erwartungswert für die Gesamtzahl verschickter Nachrichten ergibt sich durch Summierung über alle i ($E_n = n$, da $\langle p_{max} \rangle$ immer genau n mal weitergeleitet wird):

$$\begin{aligned} E &= \sum_{i=1}^n E_i = \left[\sum_{i=1}^{n-1} H_i \right] + n \stackrel{(*)}{=} [n \cdot H_{n-1} - (n-1)] + n \\ &= n \cdot H_{n-1} + 1 = n \left(H_{n-1} + \frac{1}{n} \right) = n \cdot H_n \end{aligned} \quad (2.2)$$

Die Gleichung $(*)$ gilt wegen:

$$\begin{aligned} \sum_{i=1}^n H_i &= \sum_{i=1}^n \sum_{k=1}^i \frac{1}{k} = n \cdot \frac{1}{1} + (n-1) \cdot \frac{1}{2} + \dots + 1 \cdot \frac{1}{n} \\ &= \sum_{j=1}^n (n+1-j) \cdot \frac{1}{j} = \sum_{j=1}^n \frac{n+1}{j} - \sum_{j=1}^n 1 = (n+1) \cdot H_n - n \end{aligned}$$

Der Erwartungswert für die Gesamtzahl aller Nachrichten (gleichbedeutend mit der mittleren Nachrichtenkomplexität) ist also $E = n \cdot H_n = \mathcal{O}(n \ln n)$ (mit $\ln n = \log_2 n / \log_2 e = 0,6931 \dots \log_2 n$). ■

2.4 Der Algorithmus von Peterson und Dolev et al.

James Burns sowie Daniel Hirschberg und James Sinclair stellten 1980 zwei Algorithmen für bidirektionale Ringe vor, die auch im „Worst Case“ mit $\mathcal{O}(n \log n)$ Nachrichten auskommen [Burns, 1980, Hirschberg and Sinclair, 1980]. Letztere äußerten dabei die Vermutung, daß $\Omega(n^2)$ die untere Schranke für die Koordinatorwahl in unidirektionalen Ringen sei. (Algorithmen zur Koordinatorwahl in bidirektionalen Ringen werden in der vorliegenden Diplomarbeit nicht beschrieben.)

Dies wurde 1982 durch Gary Peterson sowie Danny Dolev, Maria Klawe und Michael Rodeh widerlegt [Peterson, 1982, Dolev et al., 1982]. Sie entwickelten unabhängig voneinander den gleichen Grundalgorithmus, der in diesem Abschnitt beschrieben wird. Es folgten mehrere Modifikationen, die den konstanten Faktor (vor dem $n \log n$) immer kleiner werden ließen. Eine von Peterson weiterentwickelte Version wird im nächsten Abschnitt beschrieben.

Die oben erwähnten bidirektionalen Algorithmen dienten als Vorlage für die unidirektionale Version. Bei den bidirektionalen Algorithmen vergleicht ein Prozeß seine ID mit denen seiner linken und rechten Nachbarn und bleibt nur dann aktiv, wenn er die größte dieser drei IDs besitzt. Ansonsten wird er passiv und nimmt an der weiteren Wahl nicht mehr teil (bis auf das Weiterleiten von Nachrichten).

Eine direkte Übernahme dieser Idee für unidirektionale Algorithmen ist nicht möglich, da die Kommunikation nur in einer Richtung erfolgen kann. Mit folgendem Trick ist dies aber doch machbar: statt eines Vergleichs mit den linken und rechten Nachbarn wird ein Vergleich mit den beiden nächsten linken Nachbarn durchgeführt (vgl. Abb. 2.2). Da keine Rückmeldung über das Ergebnis dieses Vergleichs

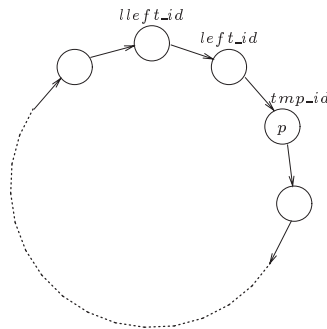


Abbildung 2.2: Die Situation aus Sicht des Prozesses p mit ID tmp_id . Er erhält nacheinander die IDs seiner beiden linken Nachbarn, $left_id$ und $lleft_id$, und bleibt genau dann aktiv, wenn $left_id$ das Maximum dieser drei IDs ist.

möglich ist, nimmt der rechte der drei Prozesse die ID des mittleren Prozesses an, wenn diese maximal ist. Ansonsten wechselt er in den passiven Zustand (in dem eingehende Nachrichten lediglich weitergeleitet werden). Dieser Grundalgorithmus ist in Alg. 2.3 dargestellt.

Wichtig ist also, daß der Algorithmus mit temporären IDs arbeitet (im Algorithmus mit tmp_id bezeichnet), die keine feste Bindung an Prozesse besitzen.

Man kann sich die Abarbeitung des Algorithmus in Runden vorstellen, die jeweils in eine A- und B-Phase unterteilt sind. In jeder Phase wird eine Nachricht verschickt. In der A-Phase enthält diese die eigene (temporäre) ID, in der B-Phase wird einfach die in der A-Phase erhaltene Nachricht weitergeleitet (also die ID des linken Nachbarn). Die Variable $phase$ dient zur Unterscheidung beider Phasen. (Um die korrekte Terminierung sicherzustellen, darf die ID des linken Nachbarn nur weitergeleitet werden, wenn diese größer als die eigene ID ist; ist dies nicht der Fall, wird in der B-Phase erneut die eigene ID verschickt.)

Die drei Bedingungen für die Korrektheit lassen sich leicht nachprüfen:

- *Sicherheit:* Die maximale ID bleibt immer erhalten, da sie stets ein lokales Maximum ist.
- *Fortschritt:* Bei ≥ 3 aktiven Prozessen wird deren Anzahl pro Runde mindestens halbiert, da keine zwei lokalen Maxima direkt nebeneinander liegen können.

Sind nur noch zwei aktive Prozesse p und q (ohne Einschränkung $id(p) > id(q)$) übrig, dann erhalten beide die ID des jeweils anderen als erste Nachricht. Als zweite Nachricht erhält p seine eigene Nachricht zurück, q jedoch erneut $\langle id(p) \rangle$. In dieser Situation ist der Fortschritt streng genommen nicht mehr gegeben, da beide Prozesse (mit temporärer ID $id(p)$) aktiv bleiben. Der Prozeß p wird jedoch eindeutig zum Koordinator gewählt, und q kann bei der anschließenden Propagierung der Wahl zu einem passiven Prozeß werden.

- *Korrekte Terminierung:* Bei ≥ 3 aktiven Prozessen wechselt kein Prozeß in den Zustand LEADER, da jeder Prozeß zwei Nachrichten von anderen Prozessen erhält. Bei zwei aktiven Prozessen wurde oben gezeigt, daß einer von beiden eindeutig zum Koordinator gewählt wird. Bei lediglich einem aktiven Prozeß erhält dieser seine erste Nachricht direkt wieder und wird zum Koordinator.

```

(* Algorithmus für Prozeß id *)

(* Globale Variablen *)
var
  int tmp_id          init id
  phase ∈ {A, B}    init A
  int left_id       init  $-\infty$ 
  state ∈ {ACTIVE, PASSIVE, LEADER}
                    init ACTIVE

(* Hauptprogramm *)
begin
  send ⟨id⟩
end

receive ⟨i⟩
  if state = PASSIVE then
    send ⟨i⟩
  else (* state = ACTIVE *)
    if tmp_id = i then
      state := LEADER
    elsif phase = A then
      left_id := i
      send (max(tmp_id, left_id))
      phase := B
    else (* phase = B *)
      max := max(tmp_id, left_id, i)
      if left_id = max then
        tmp_id := left_id
        send ⟨tmp_id⟩
      else
        state := PASSIVE
      fi
    fi
  fi
end

```

Algorithmus 2.3: Der Algorithmus von Peterson und Dolev et al.

Zur Abschätzung der Nachrichtenkomplexität sind nur zwei Feststellungen nötig:

1. In jeder Runde wechselt mindestens die Hälfte der verbliebenen aktiven Prozesse in den passiven Zustand (da keine zwei lokalen Maxima nebeneinander liegen können). Es gibt also maximal $\lceil \log n \rceil$ Runden.
2. Pro Runde werden exakt $2n$ Nachrichten versandt, denn jeder aktive Prozeß verschickt genau zwei Nachrichten, und jeder passive Prozeß leitet genau zwei Nachrichten weiter.

Die Gesamtzahl verschickter Nachrichten ist also $2n \cdot \lceil \log n \rceil = \mathcal{O}(n \log n)$. Es sei noch angemerkt, daß die Nachrichtengröße bei diesem Algorithmus mit $\log N$ optimal ist (N ist die maximale ID).

2.5 Der modifizierte Algorithmus von Peterson

Anfang der 80er Jahre konkurrierten Peterson und Dolev et al. um den schnellsten Algorithmus zur Koordinatorwahl in unidirektionalen Ringen. Nachdem beide unabhängig voneinander obigen Grundalgorithmus mit einer Nachrichtenkomplexität von $2n \log n + \mathcal{O}(n)$ entwickelten, wurde die Konstante in mehreren Schritten weiter gedrückt.

Zunächst führten Dolev et al. mehrere Modifikationen des Grundalgorithmus durch, nach denen $1,5n \log n + \mathcal{O}(n)$ Nachrichten ausreichen [Dolev et al., 1982, Abschn. 3]. Peterson widmete sich daraufhin der Weiterentwicklung des Grundalgorithmus, womit er eine Nachrichtenkomplexität von $1,440..n \log n + \mathcal{O}(n)$ erhielt [Peterson, 1982, Abschn. 4]. Dolev et al. wandten schließlich ihre ursprünglichen Modifikationen auf Petersons neuen Algorithmus an und erhielten damit einen Algorithmus, der lediglich $1,356..n \log n + \mathcal{O}(n)$ Nachrichten benötigt [Dolev et al., 1982, Abschn. 4].

(Diese letzte Verbesserung von Dolev et al. ist allerdings nur mit einer signifikant zunehmenden Nachrichtengröße möglich.)

Im vorliegenden Abschnitt wird der von Peterson weiterentwickelte Algorithmus beschrieben, dem folgende Idee zugrunde liegt: Ein Prozeß kann nach Erhalt der ersten Nachricht eventuell schon entscheiden, daß er diese Runde (unabhängig vom Inhalt der zweiten Nachricht) gar nicht mehr überstehen kann. Er kann sich also das Senden seiner zweiten Nachricht sparen. Der modifizierte Algorithmus ist in Alg. 2.4 dargestellt.

<pre> (* Algorithmus für Prozeß <i>id</i> *) (* Globale Variablen *) var int <i>tmp_id</i> init <i>id</i> phase ∈ {A, B} init A state ∈ {ACTIVE, PASSIVE, LEADER} init ACTIVE (* Hauptprogramm *) begin send ⟨<i>id</i>⟩ end receive ⟨<i>i</i>⟩ if state = PASSIVE then send ⟨<i>i</i>⟩ else (* state = ACTIVE *) if <i>tmp_id</i> = <i>i</i> then </pre>	<pre> state := LEADER elseif phase = A then if <i>i</i> > <i>tmp_id</i> then state := PASSIVE else send ⟨<i>tmp_id</i>⟩ phase := B fi else (* phase = B *) if <i>i</i> < <i>tmp_id</i> then state := PASSIVE else <i>tmp_id</i> := <i>i</i> send ⟨<i>tmp_id</i>⟩ phase := A fi fi end </pre>
---	--

Algorithmus 2.4: Der modifizierte Algorithmus von Peterson.

2.5.1 Korrektheit

- *Sicherheit:* Sei p der Prozeß mit maximaler ID id_{max} . Wenn es noch mindestens zwei aktive Prozesse zu Beginn der A-Phase gibt, dann erhält p eine Nachricht mit kleinerer ID und bleibt aktiv. Wenn p der einzige aktive Prozeß ist, wird er zum Koordinator.

Zu Beginn der B-Phase gibt es ebenfalls zwei Möglichkeiten: wenn p der einzige verbliebene aktive Prozeß ist, wird er zum Koordinator. Ansonsten erhält sein rechter aktiver Nachbarprozeß q die Nachricht $\langle id_{max} \rangle$, bleibt aktiv und übernimmt diesen Wert als temporäre ID.

Die maximale ID bleibt also stets erhalten.

- *Fortschritt:* Sei p der Prozeß mit maximaler ID. In der A-Phase wird sein rechter aktiver Nachbar passiv, da er eine Nachricht mit größerer ID erhält. In der B-Phase wird p selbst passiv, da er eine Nachricht mit kleinerer ID erhält. Die Anzahl aktiver Prozesse nimmt also in jeder Phase ab.
- *Korrekte Terminierung:* Aus der Begründung der Sicherheit folgt, daß die maximale ID nicht dupliziert wird. Außerdem kann eine ID kleiner id_{max} nicht im Ring zirkulieren, da sie in beiden Phasen spätestens beim Prozeß mit maximaler ID verworfen würde. Folglich kann $\langle id_{max} \rangle$ als einzige Nachricht wieder bei ihrem Ausgangsprozeß ankommen.

2.5.2 Nachrichtenkomplexität

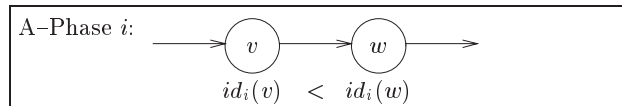
Satz 2.2 Auf einem Ring der Größe n benötigt der modifizierte Algorithmus von Peterson maximal $1,440 \cdot n \log n + \mathcal{O}(n)$ Nachrichten.

Wir numerieren die Phasen des Algorithmus in umgekehrter Reihenfolge von p bis 1 durch; in Phase p sind also noch alle Prozesse aktiv, in Phase 1 nur noch einer. Wir benötigen zunächst folgendes Lemma:

Lemma 2.3 Die Anzahl aktiver Prozesse zu Beginn einer Phase i ist kleiner oder gleich der Anzahl der Prozesse, die in der vorhergehenden Phase $i + 1$ von aktiven zu passiven Prozessen wurden.

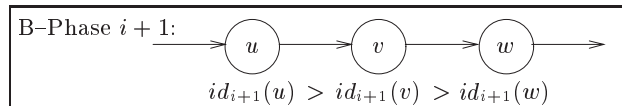
Beweis: Wir beweisen folgende Behauptung, die das Lemma impliziert: ein Prozeß kann in Phase i nur dann aktiv bleiben, wenn in der vorherigen Phase $i + 1$ sein linker aktiver Nachbar passiv wurde.

- Betrachten wir zunächst eine A-Phase i . Sei w ein Prozeß, der am Ende von Phase i aktiv ist, und v sein linker aktiver Nachbar zu Beginn der Phase i . Es liegt also folgende Situation vor:



Damit w am Ende von Phase i aktiv ist, muß $id_i(w) > id_i(v)$ gelten ($id_i(w)$ ist die ID des Prozesses w zu Beginn der Phase i).

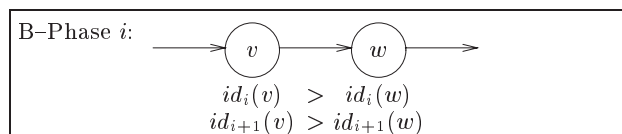
Nehmen wir nun an, daß v auch in der B-Phase $i + 1$ der erste aktive Nachbar links von w war, und u der linke aktive Nachbar von v :



Damit w in dieser B-Phase aktiv bleibt, muß die empfangene Nachricht eine ID größer als $id_{i+1}(w)$ haben. w empfängt seine Nachricht von v , also muß gelten: $id_{i+1}(v) > id_{i+1}(w)$. Gleiches gilt auch für v : damit v aktiv bleibt, muß $id_{i+1}(u) > id_{i+1}(v)$ gelten.

Nach Ablauf der B-Phase nehmen die Prozesse, die aktiv bleiben, die ID der empfangenen Nachricht an. Es gilt also $id_i(v) = id_{i+1}(u)$ und $id_i(w) = id_{i+1}(v)$, und damit auch $id_i(v) > id_i(w)$, ein Widerspruch zu obiger Wahl von w . Also kann v nicht der linke aktive Nachbar von w in Phase $i + 1$ gewesen sein.

- Betrachten wir nun eine B-Phase i . Wiederum sei w ein Prozeß, der am Ende dieser Phase aktiv ist, und v sein linker aktiver Nachbar:



Damit w diese B-Phase übersteht, muß $id_i(w) < id_i(v)$ sein. Da während der A-Phase $i + 1$ die IDs nicht geändert werden, muß auch hier $id_{i+1}(w) < id_{i+1}(v)$ gegolten haben. Dann wäre aber w bereits in Phase $i + 1$ zu einem passiven Prozeß geworden, was einen Widerspruch zur Wahl von w darstellt. Als kann auch hier v nicht der linke aktive Nachbar von w in Phase $i + 1$ gewesen sein.

Zu jedem aktiven Prozeß in Phase i gibt es somit mindestens einen Prozeß, der in Phase $i + 1$ passiv wurde. ■

Beweis (von Satz 2.2): Nun ist es einfach, die Gesamtzahl verschickter Nachrichten zu bestimmen. Sei A_k die Anzahl aktiver Prozesse zu Beginn von Phase i (es gilt also $A_p = n$ und $A_1 = 1$). Das Lemma gibt uns folgende Beziehung:

$$A_k \leq A_{k+2} - A_{k+1} \quad \Leftrightarrow \quad A_{k+2} \geq A_k + A_{k+1}. \quad (2.3)$$

Das ist offensichtlich eine Fibonacci-Entwicklung mit $A_k \geq F_{k+1}$ (mit $F_0 := 0$, $F_1 := 1$ und $F_i := F_{i-1} + F_{i-2} \forall i \geq 2$). Für die n -te Fibonacci-Zahl gilt folgende Beziehung (siehe z.B. [Knuth, 1997, Seite 83]):

$$F_n = \phi^n / \sqrt{5} + \mathcal{O}(1), \quad (2.4)$$

wobei $\phi = \frac{1}{2}(1 + \sqrt{5})$ die **goldene Zahl** ist. Daraus folgt:

$$\begin{aligned} A_p = F_{p+1} &= n \\ \Leftrightarrow \phi^{p+1} / \sqrt{5} + \mathcal{O}(1) &= n \\ \Leftrightarrow p + 1 &= \log_\phi(\sqrt{5}n - \mathcal{O}(1)) \\ \Leftrightarrow p &= \log_\phi n + \log_\phi \sqrt{5} + \mathcal{O}(1) \\ \Leftrightarrow p &= \log_2 n / \log_2 \phi + \mathcal{O}(1) \\ \Leftrightarrow p &= 1,4404.. \log n + \mathcal{O}(1) \end{aligned}$$

■

2.6 Der Algorithmus von Higham und Przytycka

Etwa 10 Jahre lang wurde auf diesem Gebiet kein Fortschritt mehr erzielt, bis 1993 Higham und Przytycka einen neuen Algorithmus zur Koordinatorwahl vorstellten, der die Konstante von 1,356.. auf 1,270.. verbessert [Higham and Przytycka, 1993].

Interessanterweise wird bei diesem Algorithmus *nicht* der Prozeß mit maximaler oder minimaler ID zum Koordinator gewählt; ist der Koordinator aber erstmal eindeutig bestimmt, kann ein Extremum mit $\leq 2n$ zusätzlichen Nachrichten ermittelt und im gesamten Ring propagiert werden.

Aus der Grundidee des Algorithmus wird unmittelbar deutlich, daß die Koordinatorwahl auch ohne Auffinden eines Extremums möglich ist. Es werden abwechselnd Sequenzen aufsteigender bzw. absteigender IDs gesucht, von denen jeweils nur die letzte ID eine Phase übersteht. In der Regel wird dadurch kein Prozeß mit extremer ID zum Koordinator gewählt.

2.6.1 Der Grundalgorithmus

Wenden wir uns zunächst der Grundversion des Algorithmus zu, wie er in Alg. 2.5 dargestellt ist. Nachrichten haben die Form $\langle id, rnd \rangle$, wobei id eine Prozeß-ID und rnd eine Rundennummer ist (beginnend bei 1). Die Variablen $last_id$ und $last_rnd$ dienen dazu, die Werte des letzten **send**-Befehls zu speichern. Wichtig ist, daß der Algorithmus nicht aus Sicht der Prozesse, sondern aus der der Nachrichten beschrieben wird.

Zu Beginn schickt jeder Prozeß seine ID (und Rundennummer 1) an seinen Nachbarn. Beim Empfang einer Nachricht geschieht folgendes:

- Erhält ein Prozeß eine Nachricht mit einer Rundennummer, die ungleich seiner zuletzt gesandten Rundennummer ist, so leitet er diese einfach weiter.
- Erhält ein Prozeß eine Nachricht mit gleicher Rundennummer, so wird diese weitergeleitet, wenn die Rundennummer gerade (bzw. ungerade) und die ID größer (bzw. kleiner) als die zuletzt gesendete ID ist. Außerdem wird die Rundennummer um eins inkrementiert. Diese Inkrementierung des Rundenzählers wird als *Beförderung in die nächste Runde* bezeichnet.

Stimmt außerdem noch die ID überein, wird der Prozeß zum Koordinator.

Wenn wir nun die Ausführung dieses Algorithmus in Runden betrachten (in denen jede Nachricht die gleiche Rundennummer hat), dann können wir das Verhalten noch genauer beschreiben. In „geraden“ Runden bestehen nur die Nachrichten mit der größten ID in einer Sequenz aufsteigender IDs. In „ungeraden“ Runden ist das Verhalten symmetrisch, d.h. nur die kleinsten IDs in Sequenzen absteigender IDs bleiben bestehen. Damit sind die drei Kriterien für die Korrektheit leicht nachzuprüfen:

- *Sicherheit*: Alle Nachrichten können nicht auf einmal gelöscht werden, da in „geraden“ („ungeraden“) Runden zumindest die maximale (minimale) ID bestehen bleibt.
- *Fortschritt*: Die Anzahl der Nachrichten nimmt von Runde zu Runde ab, da in einem Ring keine zirkuläre Folge aufsteigender bzw. absteigender IDs möglich ist.
- *Korrekte Terminierung*: Der Algorithmus terminiert, wenn ein Prozeß p seine zuletzt gesandte Nachricht unverändert (d.h. mit gleicher Rundennummer) wiedererhält. Dies ist aber nur möglich, wenn er der einzige verbliebene Prozeß ist, denn gäbe es einen weiteren Prozeß q mit gleicher Rundennummer, so würde dieser p 's Nachricht entweder verwerfen oder in die nächste Runde befördern.

Die Untersuchung der Nachrichtenkomplexität wird nicht für den Grundalgorithmus, sondern nur für den im folgenden beschriebenen vollständigen Algorithmus durchgeführt.

```

(* Algorithmus für Prozeß  $id$  *)

(* Globale Variablen *)
var
  int  $last\_id$           init  $id$ 
  int  $last\_rnd$          init 1
   $state \in \{ACTIVE, LEADER\}$  init ACTIVE

(* Hauptprogramm *)
begin
  send  $\langle last\_id, last\_rnd \rangle$ 
end

receive  $\langle i, rnd \rangle$ 
  if  $\langle i, rnd \rangle = \langle last\_id, last\_rnd \rangle$  then
     $state := LEADER$ 

  elsif [even( $rnd$ ) and ( $i > last\_id$ )] or
    [odd( $rnd$ ) and ( $i < last\_id$ )] or
    [ $rnd \neq last\_rnd$ ] then

    (* Beförderung in nächste Runde? *)
    if  $rnd = last\_rnd$  then
       $rnd := rnd + 1$ 
    fi

    send  $\langle i, rnd \rangle$ 
     $last\_id := i$ 
     $last\_rnd := rnd$ 
  fi
end

```

Algorithmus 2.5: Der Grundalgorithmus von Higham und Przytycka.

2.6.2 Der vollständige Algorithmus

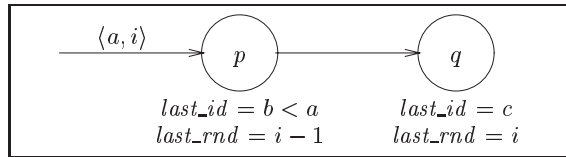
Um eine geringere Nachrichtenkomplexität als Dolev et al. zu erhalten, sind zwei weitere Änderungen des Grundalgorithmus nötig, die beide eine vorzeitige Beförderung bewirken. Vorzeitige Beförderung heißt, daß eine Nachricht in die nächste Runde befördert wird, obwohl die Bedingung für die Beförderung eigentlich nicht erfüllt ist.

Zwei Methoden zur vorzeitigen Beförderung werden verwendet:

- *Vorzeitige Beförderung durch einen Zeugen*

Betrachten wir eine Nachricht $\langle a, i \rangle$ (mit gerader Rundennummer i), die auf einen Prozeß p mit Rundennummer $i - 1$ und ID $b < a$ trifft. Die Behauptung ist nun: $\langle a, i \rangle$ kann bereits von p befördert werden, denn die Beförderung würde spätestens beim nächsten Prozeß q mit Rundennummer i durchgeführt (anders ausgedrückt: q hat eine ID $< a$).

Es liegt folgende Situation vor:



Die letzte Nachricht von p war $\langle b, i - 1 \rangle$. Sei z der erste Prozeß mit Rundennummer $i - 1$, auf den diese Nachricht trifft. Dann gibt es zwei Möglichkeiten:

1. Die ID von z ist größer b : Dann wird z die Nachricht $\langle b, i - 1 \rangle$ in Runde i befördern ($i - 1$ ist ungerade) und seine Variable $last_id$ auf b setzen. Wegen $a > b$ entspricht z in diesem Fall dem Prozeß q aus der Behauptung.
2. Die ID von z ist kleiner b : Dann wird $\langle b, i - 1 \rangle$ von z verworfen. Sei $\langle d_1, i - 1 \rangle$ die letzte Nachricht von z (mit $d_1 < b$); diese wird beim nächsten Prozeß mit Rundennummer $i - 1$ und ID d_2 verworfen, falls $d_2 < d_1$; usw.

Es gibt also eine Sequenz von Prozessen mit absteigenden IDs $d_1 > d_2 > \dots > d_k$ (und Rundennummer $i - 1$), bei deren Erreichen die Nachricht jeweils verworfen wird. Diese Sequenz muß aber mit einem Prozeß mit ID $d_{k+1} > d_k$ enden, der die ankommende Nachricht in Runde i befördert und d_k als ID annimmt. Offensichtlich gilt: $b > d_k$ und damit erst recht $a > d_k$. Somit entspricht der Prozeß mit ID d_{k+1} dem Prozeß q aus obiger Behauptung.

Die „normale“ Art der Beförderung, wie sie im Grundalgorithmus durchgeführt wird, kommt durch die vorzeitige Beförderung durch einen Zeugen gar nicht mehr zum Einsatz. Angenommen, ein Prozeß q mit Rundennummer i und ID c würde eine Nachricht $\langle a, i \rangle$ befördern (da i gerade ist, muß $a > c$ sein). Dann kann aber auf jeden Fall der Prozeß als Zeuge auftreten, der die Nachricht $\langle c, i - 2 \rangle$ in Runde $i - 1$ befördert hat. Aus diesem Grund wird die „normale“ Beförderungsbedingung im vollständigen Algorithmus nicht mehr benutzt.

- *Vorzeitige Beförderung durch Entfernung*

Eine Nachricht in Runde i wird vorzeitig befördert, wenn sie F_{i+2} Schritte zurückgelegt hat (F_i ist die i -te Fibonacci-Zahl). Würde keine vorzeitige Beförderung durch Entfernung stattfinden, so könnte es passieren, daß die vorzeitigen Beförderungen durch Zeugen in einem sehr kleinen Teil des Ringes stattfinden, was evtl. keine entscheidende Einsparung von Nachrichten zur Folge hätte. (Der Grund für die Wahl von F_{i+2} wird aus dem nachfolgenden Beweis der Nachrichtenkomplexität deutlich.)

Durch diesen Zähler werden die Nachrichten um eine dritte Komponente erweitert. Sie besitzen jetzt die Form $\langle id, rnd, cnt \rangle$ mit Prozeß-ID id , Rundennummer rnd und Entfernungszähler cnt .

Beide Methoden vorzeitiger Beförderung könnten in „geraden“ und „ungeraden“ Runden angewandt werden. Higham und Przytycka verwenden die vorzeitige Beförderung durch einen Zeugen nur in „geraden“, die durch Entfernung nur in ungeraden Runden. Der so modifizierte Grundalgorithmus ist in Alg. 2.6 dargestellt.

Die Korrektheit läßt sich analog zum Grundalgorithmus nachweisen.

- *Sicherheit*: Analog zum Grundalgorithmus.

```

(* Algorithmus für Prozeß  $id$  *)

(* Globale Variablen *)
var
  int  $last\_id$           init  $id$ 
  int  $last\_rnd$          init 1
   $state \in \{ACTIVE, LEADER\}$  init ACTIVE

(* Hauptprogramm *)
begin
  send  $\langle last\_id, last\_rnd, F_{last\_rnd+2} \rangle$ 
end

receive  $\langle i, rnd, cnt \rangle$ 
  if  $\langle i, rnd \rangle = \langle last\_id, last\_rnd \rangle$  then
     $state := LEADER$ 

  elsif [even( $rnd$ ) and ( $i > last\_id$ )] or
    [odd( $rnd$ ) and ( $i < last\_id$ )] or
    [ $rnd \neq last\_rnd$ ] then

    (* [vorzeitige] Beförderung in nächste Runde? *)
    if [even( $rnd$ ) and ( $last\_rnd = rnd-1$ ) and ( $i > last\_id$ )] or (* Zeuge *)
      [odd( $rnd$ ) and ( $cnt = 0$ )] or (* Entfernung *)
      [odd( $rnd$ ) and ( $last\_rnd = rnd$ ) and ( $i < last\_id$ )] then (* „normal“ *)
         $rnd := rnd + 1$ 
         $cnt := F_{rnd+2}$ 
      fi

    send  $\langle i, rnd, cnt-1 \rangle$ 
     $last\_id := i$ 
     $last\_rnd := rnd$ 
  fi
end

```

Algorithmus 2.6: Der vollständige Algorithmus von Higham und Przytycka.

- *Fortschritt:* Angenommen, $k \geq 2$ Nachrichten würden im Ring zirkulieren. Nach endlicher Zeit wird der Zähler jeder dieser Nachrichten einen Wert größer oder gleich der Ringgröße n haben. Spätestens dann ist aber gewährleistet, daß eine Nachricht den nächsten Prozeß erreicht, der ebenfalls eine Nachricht initiiert hat. Aufgrund der zirkulären Anordnung der Prozesse muß dann zumindest die Nachricht mit maximaler ID (in „ungeraden“ Runden) bzw. minimaler ID (in „geraden“ Runden) verworfen werden.
- *Korrekte Terminierung:* Analog zum Grundalgorithmus.

2.6.3 Nachrichtenkomplexität des vollständigen Algorithmus

Um die Komplexität dieses Algorithmus abschätzen zu können, müssen zunächst einige Begriffe eingeführt werden:

- $host_i(a)$ ist der Prozeß, der die Nachricht $\langle a, i-1 \rangle$ in Runde i befördert.
- $destroyer_i(a)$ ist der Prozeß, der die Nachricht $\langle a, i \rangle$ verwirft.

- $\delta(p, q)$ ist der Abstand zweier Prozesse p und q ($\delta(p, p) = 0$).
- $\text{succ}_i(a)$ ist die ID der Nachricht in Runde i , die unmittelbar auf $\langle a, i \rangle$ folgt.

Der größte Teil der Komplexitätsanalyse beruht auf folgendem Lemma, dessen Beweis am Schluß folgt.

Lemma 2.4 *Sei i eine ungerade Rundennummer.*

(a) *Wenn eine Nachricht $\langle a, i \rangle$ in Runde $i + 1$ befördert wird, dann gilt:*

$$\delta(\text{host}_i(a), \text{host}_{i+1}(a)) \geq F_{i+1}.$$

(b) *Wenn eine Nachricht $\langle a, i \rangle$ in Runde i verworfen wird, dann gilt:*

$$\delta(\text{host}_i(a), \text{destroyer}_i(a)) \geq F_i.$$

Offensichtlich wandert im Grundalgorithmus jede Nachricht $\langle a, i \rangle$ von $\text{host}_i(a)$ bis $\text{host}_i(\text{succ}_i(a))$. In jeder Runde werden also genau n Nachrichten verschickt. Der vollständige Algorithmus hat hingegen die Möglichkeit, einige dieser n Nachrichten durch vorzeitige Beförderung einzusparen. Wir sagen: $\langle a, i \rangle$ spart k Nachrichten ein, wenn sie lediglich $\delta(\text{host}_i(a), \text{host}_i(\text{succ}_i(a))) - k$ mal bis zur Beförderung weitergeleitet wird.

Korollar 2.5 *Sei i gerade. Dann spart jede Nachricht, die in Runde $i + 1$ befördert wird, mindestens F_i Nachrichten in Runde i ein.*

Beweis: Sei i gerade, $\langle a, i \rangle$ eine Nachricht, die in Runde $i + 1$ befördert wird, und $b = \text{succ}_i(a)$ die ID der direkten Nachfolgenachricht. Da i gerade ist und $\langle a, i \rangle$ befördert wird, muß $a > b$ gelten. Nach der Bemerkung, die weiter oben bei der Beschreibung der vorzeitigen Beförderung durch einen Zeugen gemacht wurde, wird $\langle a, i \rangle$ spätestens bei $\text{host}_{i-1}(b)$ vorzeitig befördert. Es werden also $\delta(\text{host}_{i-1}(b), \text{host}_i(b))$ Nachrichten eingespart, was nach Lemma 2.4 $\geq F_i$ Nachrichten sind. ■

Lemma 2.6 *Der vollständige Algorithmus benötigt $\leq \log_\phi n + \mathcal{O}(1)$ Runden bis zur Terminierung, wobei $\phi = \frac{1}{2}(1 + \sqrt{5})$ die goldene Zahl ist.*

Beweis: Sei i eine ungerade Rundennummer. Dann wird eine Nachricht in Runde i (nach Lemma 2.4) mindestens F_i mal weitergeleitet. Da eine Nachricht $\langle a, i \rangle$ höchstens bis zu $\text{host}_i(\text{succ}_i(a))$ wandern kann, kann es maximal n/F_i Nachrichten in Runde $i + 1$ geben. Der Algorithmus terminiert, wenn die Anzahl der Nachrichten auf 1 gesunken ist. Mit $F_i = \phi^i / \sqrt{5} + \mathcal{O}(1)$ gilt:

$$\begin{aligned} n/F_i &\leq 1 \\ \Leftrightarrow n &\leq F_i \\ \Leftrightarrow n &\leq \phi^i / \sqrt{5} + \mathcal{O}(1) \\ \Leftrightarrow n\sqrt{5} - \mathcal{O}(1) &\leq \phi^i \\ \Leftrightarrow \log_\phi n + \mathcal{O}(1) &\leq i \end{aligned}$$

■

Satz 2.7 *Auf einem Ring der Größe n benötigt der vollständige Algorithmus von Higham und Przytycka maximal $1,270 \cdot n \log n + \mathcal{O}(n)$ Nachrichten.*

Beweis: Wir schätzen die Anzahl der Nachrichten ab, die in zwei aufeinanderfolgenden Runden i und $i + 1$ (i gerade) verschickt werden. Zwei solche Runden werden als Block bezeichnet.

Sei x die Anzahl der Nachrichten in Runde $i + 1$. Nach Korollar 2.5 werden $\leq n - xF_i$ Nachrichten in Runde i verschickt.

In Runde $i + 1$ sind zwei Abschätzungen möglich: (1) offensichtlich werden insgesamt nie mehr als n Nachrichten verschickt, und (2) werden durch die Beförderung durch Entfernung pro Nachricht nie mehr als $F_{(i+1)+2}$ Weiterleitungen durchgeführt, insgesamt also $\leq xF_{i+3}$. In Runde $i + 1$ werden also $\leq \min\{n, xF_{i+3}\}$ Nachrichten verschickt.

Für einen Block (bestehend aus den Runden i und $i + 1$) ist die maximale Anzahl verschickter Nachrichten also $\leq \min\{2n - xF_i, n + x(F_{i+3} - F_i)\}$. Dieses Minimum wird für $x = n/F_{i+3}$ maximiert, d.h. in einem Block werden $\leq 2n - n(F_i/F_{i+3})$ Nachrichten verschickt.

Benötigt wird eine Abschätzung für F_i/F_{i+3} . Für die i -te Fibonacci-Zahl gilt (siehe z.B. [Knuth, 1997, Seite 83]):

$$F_i = \frac{1}{\sqrt{5}}(\phi^i - \hat{\phi}^i) \quad \left(\phi = \frac{1 + \sqrt{5}}{2}, \hat{\phi} = \frac{1 - \sqrt{5}}{2} \right)$$

Außerdem ist $\hat{\phi} = -1/\phi$, wie man leicht nachrechnen kann. Dann gilt (i gerade):

$$\begin{aligned} \frac{F_i}{F_{i+3}} &= \frac{\frac{1}{\sqrt{5}}(\phi^i - \hat{\phi}^i)}{\frac{1}{\sqrt{5}}(\phi^{i+3} - \hat{\phi}^{i+3})} \\ &= \frac{\phi^i - \phi^{-i}}{\phi^{i+3} + \phi^{-(i+3)}} \\ &= \frac{(\phi^i - \phi^{-i})(\phi^{i+3} - \phi^{-(i+3)})}{(\phi^{i+3} + \phi^{-(i+3)})(\phi^{i+3} - \phi^{-(i+3)})} \\ &= \frac{\phi^{2i+3} - \phi^3 - \phi^{-3} + \phi^{-(2i+3)}}{\phi^{2i+6} - \phi^{-(2i+6)}} \\ &> \frac{\phi^{2i+3} - \phi^3 - \phi^{-3}}{\phi^{2i+6}} \\ &= \phi^{-3} - \phi^{-2i} \underbrace{(\phi^{-3} - \phi^{-9})}_{<1} \\ &> \phi^{-3} - \phi^{-2i} \end{aligned}$$

Die Anzahl der Nachrichten in einem Block ist also

$$\begin{aligned} &\leq 2n - n \frac{F_i}{F_{i+3}} \\ &< n(2 - \phi^{-3} + \phi^{-2i}) \\ &= n\left(2 - \frac{8}{(1 + \sqrt{5})^3} + \phi^{-2i}\right) \\ &= n(4 - \sqrt{5} + \phi^{-2i}). \end{aligned}$$

Lemma 2.6 gibt uns eine obere Schranke für die Anzahl der Runden. Damit läßt sich die Anzahl aller Nachrichten als Summe über alle Blöcke bestimmen:

$$\begin{aligned}
& \sum_{i=\{2,4,\dots,\log_\phi n\}} n(4 - \sqrt{5} + \phi^{-2i}) + \mathcal{O}(n) \\
= & \frac{\log_\phi n}{2}(4 - \sqrt{5})n + \underbrace{\sum_{i=\{2,4,\dots,\log_\phi n\}} \phi^{-2i}}_{< 1/(1-\phi)} + \mathcal{O}(n) \\
= & \frac{\log_\phi n}{2}(4 - \sqrt{5})n + \mathcal{O}(n) \\
= & \frac{4 - \sqrt{5}}{2} n \frac{\log_2 n}{\log_2 \phi} + \mathcal{O}(n) \\
= & 1,2704 \cdot n \log n + \mathcal{O}(n)
\end{aligned}$$

■

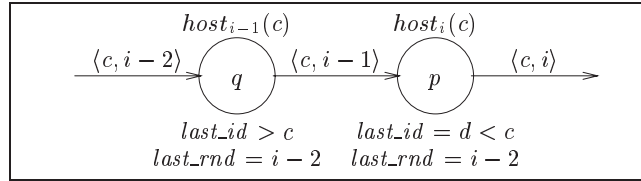
Schließlich muß noch Lemma 2.4 bewiesen werden.

Beweis (von Lemma 2.4): Wir beweisen dieses Lemma durch Induktion über ungerade Rundennummern. Der Induktionsanfang für Runde 1 gilt offensichtlich, da $F_1 = F_2 = 1$ ist und jede Nachricht mindestens einmal verschickt wird.

Für den Induktionsschluß machen wir zunächst zwei Bemerkungen. Sei i eine ungerade Rundennummer:

Bemerkung 1: Ein Zeuge, der eine Nachricht von Runde $i - 1$ in Runde i befördert, hat auch eine Nachricht in Runde $i - 2$ befördert.

Beweis: Sei $\langle c, i - 1 \rangle$ eine Nachricht, die von einem Prozeß p mit Rundennummer $i - 2$ und ID $d < c$ in Runde i befördert wird:



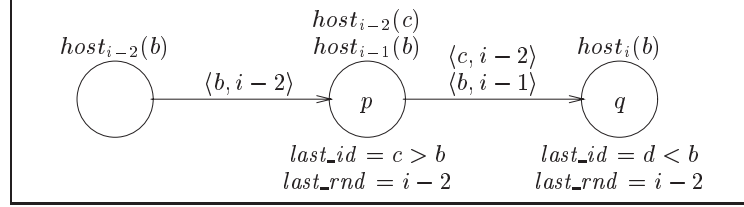
Alle Prozesse mit Rundennummer $i - 2$ und ID d müssen dem Prozeß $\text{host}_{i-2}(d)$ folgen, und zwischen zwei solchen Prozessen kann es keine anderen Prozesse mit Rundennummer $i - 2$ geben.

Die Nachricht $\langle c, i - 2 \rangle$ kann nur von einem Prozeß q mit gleicher Rundennummer in Runde $i - 1$ befördert worden sein (da $i - 2$ ungerade ist). Der Prozeß $\text{host}_{i-2}(d)$ scheidet aber aus, da $c > d$ ist. Folglich muß $\langle c, i - 1 \rangle$ vom ersten Prozeß mit Rundennummer $i - 2$ und ID d (also von $\text{host}_{i-2}(d)$ selbst) befördert worden sein, womit Bemerkung 1 bewiesen ist.

Bemerkung 2: Für jede Nachricht $\langle b, i - 1 \rangle$, die in Runde i befördert wird, gilt: $\delta(\text{host}_{i-2}(b), \text{host}_i(b)) \geq F_i$.

Beweis: Eine Nachricht $\langle b, i - 2 \rangle$ hat zwei Möglichkeiten, in Runde $i - 1$ befördert zu werden:

1. Beförderung durch Entfernung, wenn die Nachricht F_i Schritte ohne Beförderung zurückgelegt hat. In diesem Fall ist Bemerkung 2 trivial erfüllt.
2. „Normale“ Beförderung durch einen Prozeß $p = host_{i-1}(b)$ mit gleicher Rundennummer ($i - 2$) und größerer ID $c > b$. Diese Situation ist in folgender Abbildung veranschaulicht:



Nach Induktionsannahme gilt $\delta(host_{i-2}(b), host_{i-1}(b)) \geq F_{i-1}$. Da $\langle b, i - 1 \rangle$ auch in Runde i befördert wird, muß es dafür nach $host_{i-1}(b)$ einen Zeugen q mit Rundennummer $i - 2$ und ID $d < b$ geben.

Aus $d < b < c$ folgt $c > d$. Die Nachricht $\langle c, i - 2 \rangle$ kann maximal den Prozeß q erreichen, denn spätestens hier würde sie verworfen. Also gilt

$$\delta(host_{i-1}(b), host_i(b)) \geq \delta(host_{i-2}(c), destroyer_{i-2}(c)) \geq F_{i-2}$$

(die zweite Ungleichung gilt nach Induktionsannahme). Insgesamt ergibt sich also

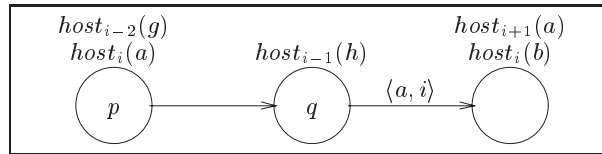
$$\delta(host_{i-2}(b), host_i(b)) \geq F_{i-2} + F_{i-1} = F_i.$$

Seien $\langle a, i \rangle$ und $\langle b, i \rangle$ zwei direkt aufeinanderfolgende Nachrichten. Wir unterscheiden zwei Fälle:

1. $\langle a, i \rangle$ überlebt Runde i .

Wenn $\langle a, i \rangle$ durch einen Zeugen durch Entfernung (nach F_{i+2} Schritten) befördert wird, ist das Lemma trivial erfüllt.

Ansonsten wandert $\langle a, i \rangle$ bis zu $host_i(b)$ und wird hier in Runde $i+1$ befördert (es muß $a < b$ gelten). Die folgende Abbildung illustriert diesen Fall:



In der vorherigen Runde $i - 1$ (gerade) muß es einen Zeugen p gegeben haben, der $\langle a, i - 1 \rangle$ in Runde i befördert hat. Nach Bemerkung 1 hat dieser Zeuge aber auch eine Nachricht $\langle g, i - 3 \rangle$ in Runde $i - 2$ befördert. Da $\langle a, i - 1 \rangle$ befördert wurde, muß $a > g$ gelten.

Betrachte nun den ersten Prozeß nach $host_{i-2}(g)$, der eine Nachricht in Runde $i - 1$ befördert; dieser sei $q = host_{i-1}(h)$. Entweder befördert q die Nachricht $\langle g, i - 2 \rangle$ in Runde $i - 1$ (dann gilt $h = g$), oder eine andere Nachricht $\langle h, i - 2 \rangle$ wird befördert. Dann gibt es zwei Möglichkeiten:

- (a) $host_{i-2}(h)$ liegt rechts von $host_{i-2}(g)$: Dann muß $\langle g, i - 2 \rangle$ bei $host_{i-2}(h)$ verworfen werden, denn sonst wäre q nicht der erste Prozeß, der eine Nachricht in Runde $i - 1$ befördert. Also gilt $g > h$.

(b) $host_{i-2}(h)$ liegt links von $host_{i-2}(g)$: Dieser Fall ist nicht möglich, denn $\langle h, i-2 \rangle$ würde von p entweder verworfen oder in Runde $i-1$ befördert.

Also liegt $host_{i-2}(h)$ nicht links von $host_{i-2}(g)$, und es gilt $h \leq g < a < b$, insbesondere also $h < b$. Dann gilt:

$$\begin{aligned} & \delta(host_i(a), host_{i+1}(a)) \\ &= \delta(host_{i-2}(g), host_i(b)) \\ &= \delta(host_{i-2}(g), host_{i-1}(h)) + \delta(host_{i-1}(h), host_i(b)) \\ &\geq \delta(host_{i-2}(h), host_{i-1}(h)) + \delta(host_{i-2}(b), host_i(b)) \end{aligned}$$

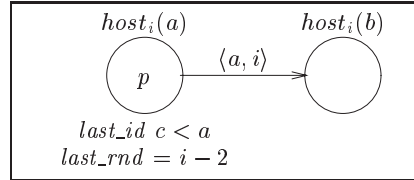
Die Ungleichung für den linken Summanden gilt, da $host_{i-2}(h)$ nicht links von $host_{i-2}(g)$ liegen kann (wie oben gezeigt wurde). Die Ungleichung für den rechten Summanden ist ebenfalls korrekt, denn würde $host_{i-1}(b)$ links von $host_{i-1}(h)$ liegen, würde die Nachricht $\langle b, i-1 \rangle$ wegen geradem $i-1$ und gleicher Rundennummer wie q von q verworfen.

Nun gelten $\delta(host_{i-2}(h), host_{i-1}(h)) \geq F_{i-1}$ (nach Induktionsannahme) und $\delta(host_{i-2}(b), host_i(b)) \geq F_i$ (nach Bemerkung 2), insgesamt also

$$\delta(host_i(a), host_{i+1}(a)) \geq F_{i-1} + F_i = F_{i+1}$$

2. $\langle a, i \rangle$ wird in Runde i verworfen.

Da $\langle a, i \rangle$ bei $host_i(b)$ verworfen wird und i ungerade ist, muß $a > b$ sein. In der vorherigen Runde $i-1$ muß es einen Zeugen p gegeben haben, der $\langle a, i-1 \rangle$ in Runde i befördert hat. Es liegt also folgende Situation vor:



Der Prozeß $host_{i-2}(b)$ wäre ein möglicher Kandidat für p . Aus dem Beweis von Bemerkung 1 folgt außerdem, daß kein anderer Kandidat näher an $host_i(b)$ liegen kann. Damit gilt:

$$\begin{aligned} & \delta(host_i(a), destroyer_i(a)) \\ &= \delta(host_i(a), host_i(b)) \\ &\geq \delta(host_{i-2}(b), host_i(b)) \\ &\geq F_i, \end{aligned}$$

wobei die letzte Ungleichung aus Bemerkung 2 folgt.

Damit ist Lemma 2.4 bewiesen. ■

Kapitel 3

Zwei untere Schranken für die Koordinatorwahl in unidirektionalen Ringen

In diesem Abschnitt wird ein Beweis von Jan Pachl, Ephraim Korach und Doron Rotem wiedergegeben, der zwei untere Schranken für die Koordinatorwahl in unidirektionalen Ringen liefert. Die Schranke für die mittlere Nachrichtenkomplexität ist exakt, die für die „Worst Case“-Komplexität bis auf einen Faktor kleiner 2 genau [Pachl et al., 1984].

Es gelten die in Kapitel 2 auf Seite 9 gemachten Annahmen zum Verhalten von Prozessen und dem Kommunikationssystem.

Das Verhalten eines Algorithmus zur Koordinatorwahl wird durch sog. Spuren beschrieben, die von Nachrichten hinterlassen werden. Eine Spur kann (zumindest theoretisch) die Information aller Prozesse enthalten, die auf ihrem Weg liegen. Die folgenden beiden Definition konkretisieren dieses Konzept:

Definition 3.1 (Sequenz) Eine nichtleere Liste natürlicher Zahlen $\langle s_1, \dots, s_k \rangle$, $k \geq 1$, $s_i \in \mathbb{N}$ heißt **Sequenz**. Die Menge aller Sequenzen, in denen kein Eintrag doppelt vorkommt, ist $\mathcal{D} := \{ \langle s_1, \dots, s_k \rangle \mid k \geq 1, s_i \in \mathbb{N}, \forall i \neq j : s_i \neq s_j \}$.

Seien $s = \langle s_1, \dots, s_k \rangle$, $t = \langle t_1, \dots, t_l \rangle$, r und u Sequenzen. Dann wird definiert:

- (a) Die **Länge** einer Sequenz s wird mit $\text{len}(s) = k$ bezeichnet.
- (b) Zwei Sequenzen s und t sind **gleich**, wenn $\text{len}(s) = \text{len}(t)$ und $\forall 1 \leq i \leq \text{len}(s) : s_i = t_i$.
- (c) $st = \langle s_1, \dots, s_k, t_1, \dots, t_l \rangle$ ist die **Konkatenation** der Sequenzen s und t (mit $\text{len}(st) = \text{len}(s) + \text{len}(t) = k + l$).
- (d) u ist eine **Teilsequenz** von s , wenn $s = rut$.
- (e) u ist **Präfix** von s , wenn $s = ut$.
- (f) u ist **Suffix** von s , wenn $s = tu$.

- (g) s und t sind **disjunkt**, wenn $\forall i \in s \forall j \in t : i \neq j$.
- (h) $\mathcal{C}(s) = \{\langle s_i, \dots, s_k, s_1, \dots, s_{i-1} \rangle \mid 1 \leq i \leq k\}$ ist die **Menge zyklischer Vertauschungen** von s . (Offensichtlich gilt: $|\mathcal{C}(s)| = \text{len}(s)$.)

□

Im folgenden wird nur noch die Menge \mathcal{D} verwendet, d.h. alle Elemente der betrachteten Sequenzen sind verschieden.

Definition 3.2 (Spur) Eine **Spur** ist entweder eine Sequenz $s = \langle s_1 \rangle$, wenn der Sender die ID s_1 hat und bisher noch keine Nachricht erhalten hat, oder eine Sequenz $s = \langle s_1, \dots, s_k \rangle$, wenn der Sender die ID s_k hat und zuletzt $\langle s_1, \dots, s_{k-1} \rangle$ empfangen hat. □

Definition 3.3 Seien $s \in \mathcal{D}$ und $\mathcal{E} \subseteq \mathcal{D}$. Dann wird definiert:

- (a) $N(s, \mathcal{E}) := \#\{t \in \mathcal{E} \mid \exists r \in \mathcal{C}(s) : t \text{ ist Präfix von } r\}$,
d.h. $N(s, \mathcal{E})$ ist die Anzahl der Sequenzen in \mathcal{E} , die Präfix einer zyklischen Vertauschung von s sind.
- (b) $N_k(s, \mathcal{E}) := \#\{t \in \mathcal{E} \mid \exists r \in \mathcal{C}(s) : t \text{ ist Präfix von } r \wedge \text{len}(t) = k\}$.

□

Damit gilt offensichtlich: $N_k(s, \mathcal{E}) = 0 \forall k > \text{len}(s)$ und $\sum_{k=1}^{\text{len}(s)} N_k(s, \mathcal{E}) = N(s, \mathcal{E})$.
Schließlich benötigen wir noch die Definition sog. erschöpfender Mengen:

Definition 3.4 (erschöpfend) $\mathcal{E} \subseteq \mathcal{D}$ heißt **erschöpfend**, wenn gilt:

- (a) **Präfixeigenschaft**
 $\forall s, t \in \mathcal{D} : st \in \mathcal{E} \Rightarrow s \in \mathcal{E}$
- (b) **Zyklische Vertauschungseigenschaft**
 $\forall s \in \mathcal{D} : \mathcal{C}(s) \cap \mathcal{E} \neq \emptyset$

□

Eine erschöpfende Menge \mathcal{E} entspricht der Menge aller Spuren, die ein Algorithmus zur Koordinatorwahl in beliebigen unidirektionalen Ringen erzeugen kann.

Beispiel: Die Menge

$$\mathcal{E} = \{\langle s_1, \dots, s_k \rangle \mid k \geq 1, s_1 = \max_{1 \leq i \leq k} \{s_i\}\}$$

ist erschöpfend. Die Elemente dieses \mathcal{E} entsprechen den Spuren, die der Algorithmus von Chang und Roberts (siehe Abschnitt 2.3) hinterläßt: eine Nachricht wird von einem Prozeß nur dann weitergeleitet, wenn die übermittelte ID größer als die eigene (und damit größer als alle IDs auf dem dazwischenliegenden Weg) ist.

Im eigentlichen Algorithmus wird natürlich nur die Nachricht $\langle s_1 \rangle$ weitergeleitet, theoretisch könnte aber auch die komplette Spur $\langle s_1, \dots, s_k \rangle$ gesendet werden. ■

Das folgende Lemma besagt, daß ein Prozeß nur lokale Entscheidungen treffen kann, die vom nicht bekannten Teil des Ringes völlig unabhängig sind. Damit folgt dann der zentrale Satz, der erschöpfende Mengen mit Algorithmen zur Koordinatorwahl gleichsetzt.

Lemma 3.5 *Seien $s, t \in \mathcal{D}$ zwei Sequenzen und $u \in \mathcal{D}$ eine Teilsequenz von s sowie von t . Sei \mathbf{A} ein Algorithmus zur Koordinatorwahl. Wenn \mathbf{A} auf dem Ring mit Beschriftung s eine Spur u hinterläßt, dann tut er dies auch auf dem Ring mit Beschriftung t .*

Beweis: Nach den getroffenen Annahmen hängt die Reihenfolge der Nachrichten ausschließlich vom Inhalt der Nachrichten ab, nicht jedoch von zeitlichen Verzögerungen des Kommunikationssystems ab (die Algorithmen sind nachrichtengetrieben). Außerdem erhält jeder Prozeß Nachrichten von lediglich einem anderen Prozeß. Die Reihenfolge eintreffender Nachrichten auf einem Ring s ist also bei jeder Ausführung von \mathbf{A} gleich.

Daraus folgt aber direkt, daß eine Sequenz u , die Teilsequenz beider Ringe s und t ist, entweder in beiden oder in keinem der beiden Ringe eine Spur u hinterlassen muß. ■

Satz 3.6 *Für jeden Algorithmus zur Koordinatorwahl \mathbf{A} gibt es eine erschöpfende Menge $\mathcal{E}(\mathbf{A}) \subseteq \mathcal{D}$, sodaß \mathbf{A} mindestens $N(s, \mathcal{E}(\mathbf{A}))$ Nachrichten verschickt, wenn er auf einem Ring mit Beschriftung $s \in \mathcal{D}$ ausgeführt wird.*

Beweis: Sei $\mathcal{E}(\mathbf{A})$ als die Menge aller t definiert, für die \mathbf{A} bei Ausführung auf einem Ring mit Beschriftung t eine Spur t hinterläßt. Es ist zunächst zu zeigen, daß $\mathcal{E}(\mathbf{A})$ die beiden Bedingungen einer erschöpfende Menge erfüllt.

- *Präfixeigenschaft:* Für jedes $s \in \mathcal{E}(\mathbf{A})$ und für jede Zerlegung $s = tu$ ($\text{len}(t) \geq 1$) muß auch $t \in \mathcal{E}(\mathbf{A})$ sein. Aus der Definition der Spur folgt, daß der Spur s eine Spur t vorausgegangen sein muß. Aus Lemma 3.5 folgt dann aber auch, daß \mathbf{A} bei Ausführung auf jedem Ring, der t als Teilsequenz enthält, insbesondere also auch auf einem Ring mit Beschriftung t , eine Spur t hinterläßt. Dann ist aber (nach obiger Definition von $\mathcal{E}(\mathbf{A})$) t in der erschöpfenden Menge enthalten.
- *Zyklische Vertauschungseigenschaft:* Wir betrachten einen Ring mit Beschriftung $s = \langle s_1, \dots, s_k \rangle$. Der Algorithmus \mathbf{A} erzeugt mindestens eine Spur t der Länge k , denn sonst wäre keine eindeutige Koordinatorwahl möglich, da kein Prozeß alle IDs kennen würde. Für diese Spur gilt aber $t \in \mathcal{C}(s)$, und damit auch $\mathcal{E}(\mathbf{A}) \cap \mathcal{C}(s) \neq \emptyset$.

Nun bleibt noch zu zeigen, daß \mathbf{A} in einem Ring mit Beschriftung s mindestens $N(s, \mathcal{E}(\mathbf{A}))$ Nachrichten verschickt. Nach der Definition für $N(s, \mathcal{E}(\mathbf{A}))$ ist es ausreichend zu zeigen, daß für jedes $t \in \mathcal{E}(\mathbf{A})$, das Präfix eines $r \in \mathcal{C}(s)$ ist, eine Spur t hinterlassen wird.

Das ist aber leicht einzusehen. Sei t eine Teilsequenz des Ringes s (und damit Präfix eines $r \in \mathcal{C}(s)$). Wenn $t \in \mathcal{E}(\mathbf{A})$ ist, dann wird (nach Definition von $\mathcal{E}(\mathbf{A})$ und Lemma 3.5) eine Spur t auf jedem Ring hinterlassen, dessen Beschriftung t als Teilsequenz enthält, insbesondere also auch auf dem Ring s . ■

Definition 3.7 ($worst_{\mathbf{A}}(n)$, $avg_{\mathbf{A}}(n)$) Die maximale (bzw. durchschnittliche) Anzahl von Nachrichten, die ein Algorithmus \mathbf{A} zur Koordinatorwahl auf einem unidirektionalen Ring mit n Prozessen verschickt, wird mit $worst_{\mathbf{A}}(n)$ (bzw. $avg_{\mathbf{A}}(n)$) bezeichnet. \square

Aus Satz 3.6 folgt damit unmittelbar:

Korollar 3.8 Sei $\Pi(n)$ die Menge aller Permutationen von n Elementen. Dann gilt:

$$(a) \quad avg_{\mathbf{A}}(n) \geq \frac{1}{n!} \sum_{s \in \Pi(n)} N(s, \mathcal{E}(\mathbf{A}))$$

$$(b) \quad worst_{\mathbf{A}}(n) \geq \max_{s \in \Pi(n)} N(s, \mathcal{E}(\mathbf{A}))$$

Nun muß versucht werden, diese Ausdrücke möglichst genau abzuschätzen. Beginnen wir mit $avg_{\mathbf{A}}(n)$.

$$\begin{aligned} avg_{\mathbf{A}}(n) &\geq \frac{1}{n!} \sum_{s \in \Pi(n)} N(s, \mathcal{E}(\mathbf{A})) \\ &= \frac{1}{n!} \sum_{s \in \Pi(n)} \sum_{k=1}^n N_k(s, \mathcal{E}(\mathbf{A})) \\ &= \frac{1}{n!} \sum_{k=1}^n \sum_{s \in \Pi(n)} N_k(s, \mathcal{E}(\mathbf{A})) \end{aligned} \tag{3.1}$$

Wenn k und s fest sind, gibt es maximal n Sequenzen $t \in \mathcal{E}(\mathbf{A})$ mit Länge k , die Präfix eines $r \in \mathcal{C}(s)$ sind (da $|\mathcal{C}(s)| = n$). Summiert über alle Permutationen (bei immer noch festem k) gibt es $n \cdot n!$ solcher Präfixe der Länge k .

Diese Präfixe lassen sich in Gruppen der Größe k zusammenfassen, deren Elemente sich jeweils nur durch zyklische Vertauschungen unterscheiden. Für die Erfüllung der zyklischen Vertauschungseigenschaft von $\mathcal{E}(\mathbf{A})$ reicht es aus, wenn jeweils eines dieser k Präfixe in $\mathcal{E}(\mathbf{A})$ enthalten ist. Somit gilt:

$$\begin{aligned} avg_{\mathbf{A}}(n) &\geq \frac{1}{n!} \sum_{k=1}^n \frac{n \cdot n!}{k} \\ &= n \sum_{k=1}^n \frac{1}{k} \\ &= nH_n, \end{aligned} \tag{3.2}$$

wobei H_n die n -te Harmonische Zahl ist.

Aus dieser Abschätzung und dem Algorithmus von Chang und Roberts, der im Mittel nH_n Nachrichten verschickt (siehe Abschnitt 2.3), folgt:

Satz 3.9 Die durchschnittliche Anzahl verschickter Nachrichten eines Algorithmus zur Koordinatorwahl auf einem unidirektionalen Ring mit n Prozessen beträgt mindestens nH_n .

Für den ungünstigsten Fall ist keine exakte untere Schranke bekannt. Hier gilt aber:

Satz 3.10 *Die untere Schranke für die „Worst Case“-Nachrichtenkomplexität seine Algorithmus zur Koordinatorwahl auf einem unidirektionalen Ring mit n Prozessen ist*

$$0,6931..n \log_2 n + \mathcal{O}(n) \leq \min_{\mathbf{A}} \{worst_{\mathbf{A}}(n)\} \leq 1,270..n \log_2 n + \mathcal{O}(n).$$

Beweis: Die Abschätzung nach unten folgt aus Satz 3.9 und aus der Abschätzung der Harmonischen Zahlen (siehe z.B. [Knuth, 1997, Seite 75]):

$$H_n = \ln n + \mathcal{O}(n) = \frac{\log_2 n}{\log_2 e} + \mathcal{O}(n) = 0.6931.. \log_2 n + \mathcal{O}(n)$$

Die Abschätzung nach oben ist durch den bisher besten bekannten Algorithmus von Higham und Przytycka gegeben (siehe Abschnitt 2.6). ■

Kapitel 4

Anonyme Netzwerke

In diesem Kapitel sollen einige Aspekte anonymer Netzwerke untersucht werden. In einem anonymen Netzwerk sind die einzelnen Prozesse nicht durch Namen bzw. IDs unterscheidbar. Wie wir im folgenden sehen werden, hat dies entscheidende Auswirkungen auf die Berechenbarkeit vieler Funktionen.

4.1 Berechenbarkeit in anonymen Ringen

In diesem Abschnitt werden einige allgemeine Ergebnisse wiedergegeben, die Aussagen über die Berechenbarkeit in anonymen Ringen machen. Dieser Abschnitt beruht weitestgehend auf zwei Artikeln von Hagit Attiya, Marc Snir und Manfred Warmuth [Attiya et al., 1988, Attiya and Snir, 1991].

Zunächst müssen einige Begriffe eingeführt werden. Sei R ein Ring mit bidirektionalen Kommunikationskanälen, d.h. $R = (V, E)$ mit $V = \{1, \dots, n\}$ und $E = \{\{i, (i \bmod n) + 1\} \mid 1 \leq i \leq n\}$. (Im folgenden wird die umständliche Modulo-Schreibweise weggelassen; verwendete Prozeßnummern sind als stets modulo n zu verstehen.)

Zu jedem Prozeß i bezeichnen $left(i) = i - 1$ bzw. $right(i) = i + 1$ den linken bzw. rechten Nachbarprozeß von i . Die Prozesse sind nicht gleich orientiert; jeder einzelne Prozeß kann zwar seine zwei Nachbarprozesse unterscheiden, doch diese lokale Numerierung ist global nicht einheitlich. Die Orientierung eines Prozesses i wird mit $D(i)$ (*direction*) bezeichnet:

- $D(i) = 0$ *gdw.* i ist im Uhrzeigersinn orientiert, also $left(i) = i - 1$ und $right(i) = i + 1$.
- $D(i) = 1$ *gdw.* i ist gegen den Uhrzeigersinn orientiert, also $left(i) = i + 1$ und $right(i) = i - 1$.

Mit $\overline{D(i)}$ wird die komplementäre Richtung bezeichnet, also $\overline{D(i)} = 1 - D(i)$. Die Eingabe und Ausgabe eines Prozesses i wird mit $I(i)$ (*input*) bzw. $O(i)$ (*output*) bezeichnet, Orientierung, Eingabe und Ausgabe für den gesamten Ring mit $D = \langle D(1), \dots, D(n) \rangle$, $I = \langle I(1), \dots, I(n) \rangle$ und $O = \langle O(1), \dots, O(n) \rangle$.

Wichtig ist die Unterscheidung der Informationen, die ein Prozeß selbst kennt, und der Informationen, die zur Beschreibung des Systems verwendet werden.

- *Ein Prozeß* kennt nur seinen Eingabewert und eine lokale Numerierung seiner beiden Kommunikationskanäle. Ihm ist weder sein Name (bzw. Index) noch seine Orientierung bekannt.
- *Zur Beschreibung des Systems* werden die Prozeßindizes, die Eingabewerte und die Prozeßorientierungen verwendet. Die Startkonfiguration eines Ringes ist damit durch $R = \langle D(1), I(1), \dots, D(n), I(n) \rangle$ gegeben.

Definition 4.1 (*k*-Umgebung) Sei $R = (V, E)$ ein Ring. Dann besteht die *k*-Umgebung eines Prozesses i aus den Orientierungen und Eingabewerten der $2k+1$ Prozesse $i-k, \dots, i+k$ (relativ zu i 's Orientierung).

Die *k*-Umgebung von i ist

$$\langle D(i-k), I(i-k), \dots, D(i+k), I(i+k) \rangle$$

für $D(i) = 0$ bzw.

$$\langle \overline{D(i+k)}, I(i+k), \dots, \overline{D(i-k)}, I(i-k) \rangle$$

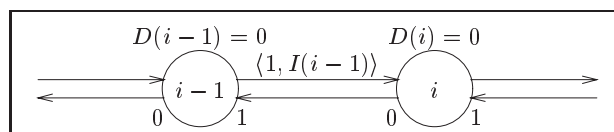
für $D(i) = 1$. □

Die *k*-Umgebung enthält offensichtlich die maximale Information, die ein Prozeß i haben kann, nachdem er je k Nachrichten über beide Kommunikationskanäle erhalten hat. (Für allgemeine Graphen existiert eine ähnliche Definition, die einer *Sicht*; siehe [Yamashita and Kameda, 1996]).

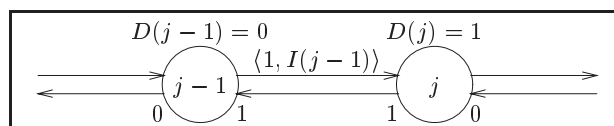
Beispiel: Wir betrachten zwei Prozesse i und j mit $D(i) = 0$ und $D(j) = 1$. Da den Prozessen die globale Orientierung nicht bekannt ist, nimmt jeder an, er sei im Uhrzeigersinn orientiert. Die lokale Numerierung der Kommunikationskanäle (im folgenden „Ports“ genannt) wird mit 0 und 1 bezeichnet.

Nun erhalten i und j je eine Nachricht ihres linken Nachbarn $i-1$ bzw. $j-1$ mit $D(i-1) = D(j-1) = 0$. Eine Nachricht von Prozeß k hat dabei stets die Form $\langle port(k), I(k) \rangle$ mit lokaler Portnummer $port(k)$ und Eingabewert $I(k)$.

- Prozeß i erhält über Port 0 eine Nachricht $\langle 1, I(i-1) \rangle$ (siehe Abbildung unten). Damit weiß er, daß $i-1$ gleich orientiert ist und nimmt als Umgebung $\langle 0, I(i-1), 0, I(i) \rangle$ an, was offensichtlich $\langle D(i-1), I(i-1), D(i), I(i) \rangle$ entspricht.



- Betrachten wir nun den Prozeß j mit $D(j) = 1$ (siehe Abbildung unten). Aus ihrer eigenen Sicht unterscheiden sich die Prozesse i und j zunächst nicht (beide nehmen an, sie seien im Uhrzeigersinn orientiert). Wenn j jetzt aber eine Nachricht $\langle 1, I(j-1) \rangle$ von seinem linken Nachbarn erhält, dann wird dies als Nachricht von rechts mit entgegengesetzter Orientierung interpretiert. Die Umgebung von j ist damit $\langle 0, I(j), 1, I(j-1) \rangle$, was $\langle \overline{D(j)}, I(j), \overline{D(j-1)}, I(j-1) \rangle$ entspricht.



■

Nach den Bemerkungen in Abschnitt 1.5.1 können wir annehmen, daß die Ausführung der Algorithmen in (pseudosynchronen) Runden erfolgt. Das nächste Lemma konkretisiert dann obige Bemerkung, daß ein Prozeß nach je k Nachrichten von beiden Seiten nicht mehr als seine k -Umgebung kennen kann.

Lemma 4.2 *Seien R_1 und R_2 zwei Startkonfigurationen für einen Ring. Sei p ein Prozeß, der in R_1 und R_2 die gleiche k -Umgebung besitzt. Dann befindet sich p nach k Runden unter R_1 oder R_2 im selben Zustand.*

Als nächstes wird gezeigt, daß keine Funktion auf Ringen unbekannter Größe berechnet werden kann.

Definition 4.3 (Funktion) Eine *Funktion* f ist eine Abbildung, die jeder Startkonfiguration R genau eine Lösung $f(R)$ zuordnet. Ein Algorithmus A berechnet eine Funktion f , wenn jeder Prozeß ausgehend von einer Startkonfiguration R die Ausgabe $f(R)$ produziert. \square

Satz 4.4 *Sei $f : \Sigma \rightarrow \{0,1\}$ eine nicht-konstante Funktion. Dann gibt es keinen Algorithmus, der f auf einem Ring beliebiger, aber nicht bekannter Größe berechnet.*

Beweis: Wir nehmen zunächst an, es gäbe einen Algorithmus A , der f auf einem im Uhrzeigersinn orientierten Ring berechnet. Wir betrachten zwei Eingaben I_1 und I_2 mit $f(I_1) = 0$ und $f(I_2) = 1$. Sei t so gewählt, daß A nach spätestens t Runden bei Eingabe I_1 bzw. I_2 terminiert.

Wir konstruieren nun folgende Eingabe $I = I_1^{2t+1} I_2^{2t+1}$ (d.h. $2t+1$ mal I_1 gefolgt von $2t+1$ mal I_2). Im ersten Teil dieses Ringes gibt es einen Prozeß, der die gleiche t -Umgebung wie ein Prozeß im Ring mit Eingabe I_1 hat. Analoges gilt für den zweiten Teil des neuen Ringes. Dann wird aber einer der beiden Prozesse 0, der andere hingegen 1 als Ausgabe generieren.

Es gibt also (im Widerspruch zu obiger Annahme) Ringkonfigurationen, unter denen A keine korrekte Ausgabe berechnet. \blacksquare

Für die verteilte Berechnung jeder nicht-konstanten Funktion in anonymen Netzwerken ist also zumindest eine obere Schranke für die Anzahl der Prozesse nötig. Viele Algorithmen benötigen sogar die exakte Prozeßanzahl n . Dazu gehören beispielsweise die Berechnung der Summe aller Eingabewerte und die XOR-Funktion.

In dieser Arbeit interessiert uns aber das Problem der Koordinatorwahl, das ohne Kenntnis der exakten Ringgröße nicht zu lösen ist. Zunächst muß die Definition einer Funktion verallgemeinert werden:

Definition 4.5 (Problem) Ein Problem P ist eine Abbildung, die jeder Startkonfiguration eine Menge von korrekten Ergebnisvektoren zuordnet (d.h. der Ergebniswert der einzelnen Prozesse darf sich unterscheiden). Ein Algorithmus löst ein Problem P , wenn er ausgehend von einer Startkonfiguration R eines der korrekten Ergebnisse $P(R)$ berechnet. \square

Die korrekte Ergebnismenge für die Koordinatorwahl ist (unabhängig von der Eingabe) $\{ \langle 0^i 1 0^{n-i-1} \rangle \mid 0 \leq i \leq n-1 \}$, d.h. genau ein Prozeß erzeugt als Ausgabe 1 und ist damit zum Koordinator gewählt.

Lemma 4.6 *Es gibt keinen deterministischen Algorithmus für die Koordinatorwahl in einem Ring, der ohne Kenntnis der Ringgröße n auskommt.*

Beweis: Für die Koordinatorwahl ist die Eingabe unwichtig — sie entspricht damit einer Eingabe, die für jeden Prozeß des Ringes identisch ist.

Dann sind offensichtlich für jedes $k \geq 0$ die k -Umgebungen für alle Prozesse identisch. Angenommen, A sei ein Algorithmus zur Koordinatorwahl, der auf einem Ring der Größe n terminiert. Dann gilt: entweder wurden alle Prozesse oder überhaupt kein Prozeß zum Koordinator gewählt. Also erfüllt A (im Widerspruch zur Annahme) nicht die Eigenschaft eines Algorithmus zur Koordinatorwahl, daß er nämlich genau einen Prozeß zum Koordinator ernennt. ■

An dieser Stelle ist eine Anmerkung zu synchronen Kommunikationssystemen angebracht, denn hier ist das Problem der Koordinatorwahl zumindest in manchen Fällen lösbar. Antoni Mazurkiewicz hat beispielsweise gezeigt, daß die Koordinatorwahl in einem Ring genau dann möglich ist, wenn die Ringgröße eine Primzahl ist [Mazurkiewicz, 1988].

Wir betrachten ein einfacheres Beispiel, nämlich ein System mit lediglich zwei Prozessen. Jeder Prozeß versucht, dem anderen eine Nachricht zu schicken. Durch das Kommunikationssystem wird aber nur eine davon mit dem Empfangsbefehl des anderen synchronisiert. Gewinner ist der Prozeß, dessen Sendebefehl erfolgreich war. Dieses Schema läßt sich auf Cliques beliebiger Größe ausweiten [Angluin, 1980]. Das synchrone Kommunikationssystem ist dabei hilfreich, weil es als „externer Richter“ fungieren kann.

4.2 Koordinatorwahl in anonymen Ringen

In diesem Abschnitt wird ein Algorithmus von Alon Itai und Michael Rodeh beschrieben, der die Koordinatorwahl in anonymen Ringen mit Wahrscheinlichkeit 1 löst [Itai and Rodeh, 1981]. Er gehört zu der Klasse der Las Vegas-Algorithmen.

Definition 4.7 (Las Vegas-Algorithmus) Ein Algorithmus A ist ein *Las Vegas-Algorithmus*, wenn er folgende zwei Bedingungen erfüllt:

1. A terminiert mit Wahrscheinlichkeit > 0 .
2. A ist partiell korrekt (d.h. terminiert A , dann liefert er das korrekte Ergebnis).

□

(Anmerkung: im Gegensatz dazu ist ein *Monte Carlo-Algorithmus* ein Algorithmus, der stets terminiert, aber nur mit Wahrscheinlichkeit > 0 partiell korrekt ist.)

Der Algorithmus von Itai und Rodeh ist eine Adaption des Algorithmus von Chang und Roberts (siehe Abschnitt 2.3) an die veränderten Bedingungen anonymer Netzwerke. Die Ringgröße n sei allen Prozessen bekannt.

- Da die Prozesse keine IDs besitzen, wählt zunächst jeder Prozeß zufällig eine ID im Intervall $[1, n]$ aus.
- Nun kann es aber passieren, daß zwei oder mehr Prozesse die maximale ID besitzen. Um zu vermeiden, daß mehrere Prozesse in den Zustand LEADER wechseln, wird den Nachrichten ein Zähler mitgegeben. Ein Prozeß terminiert nur dann, wenn er eine Nachricht mit seiner eigenen ID und Zählerstand n erhält.

- Der Zähler allein schließt aber immer noch nicht aus, daß mehrere Prozesse gleichzeitig terminieren können. Den Nachrichten wird deshalb ein weiterer Parameter *unique* hinzugefügt, der angibt, ob ein Wert im Ring einmalig ist. Zunächst wird *unique=TRUE* gesetzt. Trifft die Nachricht jedoch auf einen Prozeß mit gleicher ID, setzt dieser *unique=FALSE*. Ein Prozeß wechselt nur dann in den Zustand LEADER, wenn er eine Nachricht mit seiner ID, Zählerstand *n* und *unique=TRUE* erhält.
- Erhält ein Prozeß eine Nachricht mit seiner ID, Zählerstand *n* und *unique=FALSE*, dann weiß er, daß er zwar die maximale ID besitzt, diese aber nicht einmalig ist. Jeder der Prozesse, die diesen Zustand erreichen, wählt dann erneut eine ID im Intervall $[1, n]$ und beginnt den Algorithmus von neuem (startet also die nächste Runde). Die Prozesse, die bereits zuvor eine größere als die eigene ID erhalten haben, nehmen an der Koordinatorwahl nicht mehr teil (abgesehen vom Weiterleiten eingehender Nachrichten).

Der so modifizierte Algorithmus ist in Alg. 4.1 dargestellt.

<pre>(* Globale Variablen *) var int id init 0 state ∈ {ACTIVE, PASSIVE, LEADER} init ACTIVE (* Hauptprogramm *) begin id := random(1, ..., n) send ⟨id, 1, TRUE⟩ end receive ⟨i, count, unique⟩ if state = PASSIVE then send ⟨i, count+1, unique⟩</pre>	<pre>else (* state = ACTIVE *) if i = id then if count = n then if unique = TRUE then state := LEADER else (* unique = FALSE *) (* starte nächste Runde *) id := random(1, ..., n) send ⟨id, 1, TRUE⟩ fi else (* count < n *) send ⟨i, count+1, FALSE⟩ fi elsif i > id then state := PASSIVE send ⟨i, count+1, unique⟩ fi end</pre>
---	---

Algorithmus 4.1: Der Algorithmus von Itai und Rodeh.

Für den Nachweis der Korrektheit wenden wir wieder die drei Kriterien Sicherheit, Fortschritt und korrekte Terminierung an (siehe Abschnitt 2.1).

- *Sicherheit:* Ein aktiver Prozeß mit maximaler ID (unter den aktiven Prozessen) bleibt auf jeden Fall aktiv, denn (1) passive Prozesse sowie aktive Prozesse mit kleinerer ID leiten seine Nachricht weiter, (2) aktive Prozesse mit gleicher ID setzen zwar *unique=FALSE*, leiten die Nachricht aber ebenfalls weiter. Also erhält ein solcher Prozeß seine Nachricht auf jeden Fall wieder zurück.
- *Fortschritt:* Der Fortschritt ist nicht mehr deterministisch zu begründen, sondern nur über Wahrscheinlichkeiten. Nehmen wir an, daß es zu Beginn einer Runde $i \geq 2$ noch *k* aktive Prozesse gibt (die in Runde $i-1$ alle die gleiche minimale ID gewählt hatten). Jeder dieser *k* Prozesse wählt zunächst eine neue

ID. Die Wahrscheinlichkeit dafür, daß alle k wiederum die gleiche ID wählen und die Anzahl aktiver Prozesse zu Beginn von Runde $i+1$ nicht abgenommen hat, ist $n \cdot \frac{1}{n^k} = \frac{1}{n^{k-1}}$. Die Wahrscheinlichkeit, daß es zu Beginn von Runde $i+l$ immer noch k Prozesse gibt, ist demnach $\frac{1}{n^{l(k-1)}}$. Offensichtlich strebt diese Wahrscheinlichkeit für wachsendes l gegen 0.

Anders ausgedrückt ist die Wahrscheinlichkeit, daß die Anzahl der Prozesse nach l Runden abgenommen hat, $(1 - \frac{1}{n^{l(k-1)}})$ und strebt somit gegen 1. Damit ist auch gezeigt, daß Alg. 4.1 mit Wahrscheinlichkeit 1 terminiert.

- *Korrekte Terminierung:* Ein Prozeß wird nur dann zum Koordinator gewählt, wenn er eine Nachricht $\langle i, n, \text{TRUE} \rangle$ erhält, deren ID i seiner eigenen ID entspricht. Dann ist er aber der einzige aktive Prozeß mit maximaler ID im Ring, denn ein anderer Prozeß mit gleicher ID würde `unique=FALSE` setzen, und ein Prozeß mit größerer ID würde die Nachricht ganz verwerfen.

4.3 Rendezvous in allgemeinen Graphen

In beliebigen Graphen ist neben der Koordinatorwahl ein weiteres Problem von großer Bedeutung: die Berechnung von Spannbäumen.

Definition 4.8 (Spannbaum) Sei $G = (V, E)$ ein ungerichteter, zusammenhängender Graph. Ein Graph $G' = (V, E')$ heißt **Spannbaum** von G , wenn G' zusammenhängend und $|E'| = |V| - 1$ ist. \square

Der Spannbaum eines Graphen gibt Wege vor, über die ein Knoten alle anderen Knoten des Graphen erreichen kann. Dies ist z.B. sinnvoll, damit der Koordinator Informationen effizient im gesamten Netzwerk verteilen kann.

Ein Koordinator kann leicht einen Spannbaum berechnen. Der Koordinator startet dazu einen Tiefensuchalgorithmus, der jeden Knoten des Graphen G besucht. Kanten des Spannbaumes sind dann diejenigen Kanten von G , über die ein Knoten das erste Mal erreicht wurde (über die der Knoten also seine erste Nachricht erhielt).

Wie man leicht sieht, gilt die Umkehrung nicht. Wir betrachten dazu einen Graphen, der aus zwei miteinander verbundenen Knoten besteht (diese Verbindung ist trivialerweise die einzige Kante des Spannbaumes). Zu Beginn befinden sich beide Prozesse im selben Zustand. Jede weitere Operation (interner Befehl, Send- oder Empfangsbefehl) wird aber von beiden Prozessen gleichermaßen ausgeführt, womit beide Prozesse stets in den gleichen Zustand wechseln. Es kann also keinen deterministischen Algorithmus geben, der die inhärente Symmetrie bricht.

Yves Métivier, Nasser Saheb und Pierre-André Wacrenier haben einen Algorithmus zur Spannbauumberechnung entwickelt, dessen Beschreibung und Korrektheitsbeweis aber bereits in der Originalarbeit ca. 15 Seiten in Anspruch nehmen und damit den Rahmen dieser Diplomarbeit sprengen würde [Métivier et al., 1996]. Interessant ist aber der erste Schritt des Spannbaum-Algorithmus: ein probabilistisches Verfahren für ein Rendezvous. (Auch hier ist leicht einzusehen, daß es kein deterministisches Rendezvous-Verfahren geben kann.)

Ein probabilistisches Rendezvous-Verfahren dient zur Auflösung der initialen Symmetrie, die in anonymen Netzwerken stets vorliegt. Es ist damit ein Grundbaustein, der für viele komplexere Algorithmen eingesetzt werden kann. Ziel eines

Rendezvous-Verfahrens ist es, daß sich mindestens zwei benachbarte Prozesse finden, die in einem anschließenden Duell einen Sieger und einen Verlierer ermitteln (das Duell wird weiter unten beschrieben). Damit ist zumindest für ein Knotenpaar die inhärente Symmetrie gebrochen. Ist der zugrundeliegende Graph nicht faktorierbar, dann impliziert dies unmittelbar die Auflösung der Symmetrie für den gesamten Graphen (*faktorierbar* heißt, das es eine Zerlegung des Graphen in gleich große Teilgraphen gibt; siehe Def. 5.4 auf Seite 61; in Kapitel 5 wird außerdem gezeigt, daß die Graph-Faktorisierung NP-vollständig ist).

Alg. 4.2 zeigt das Rendezvous-Verfahren von Métivier et al. Hierbei wählt jeder Prozeß p zufällig einen seiner Nachbarprozesse q aus und schickt diesem die Nachricht $\langle \text{YOU} \rangle$. Erhält p daraufhin eine Nachricht $\langle \text{YOU} \rangle$ von q , findet ein Rendezvous zwischen p und q statt. Erhält er diese Nachricht hingegen von einem Prozeß $r \neq q$, so schickt er r eine Nachricht $\langle \text{NOT-YOU} \rangle$ zurück. Erhält p eine Nachricht $\langle \text{NOT-YOU} \rangle$, beginnt der Auswahlprozeß von vorne.

<pre>(* Alg. für Prozeß mit <i>deg</i> Nachbarn *) (* Globale Variablen *) var int <i>neighbour</i> init 0 (* Hauptprogramm *) begin <i>neighbour</i> := random(1, ..., <i>deg</i>) send $\langle \text{YOU} \rangle$ to <i>neighbour</i> end</pre>	<pre>receive $\langle \text{YOU} \rangle$ from <i>i</i> if $i = \textit{neighbour}$ then (* Rendezvous mit <i>neighbour</i> *) else send $\langle \text{NOT-YOU} \rangle$ to <i>i</i> fi end receive $\langle \text{NOT-YOU} \rangle$ from <i>i</i> (* neuen Nachbarn wählen *) <i>neighbour</i> := random(1, ..., <i>deg</i>) send $\langle \text{YOU} \rangle$ to <i>neighbour</i> end</pre>
---	--

Algorithmus 4.2: Das Rendezvous-Verfahren von Métivier et al.

Hat ein Rendezvous zwischen p und q stattgefunden, muß zwischen diesen beiden Prozessen ein Gewinner ermittelt werden. Dazu wird ein sog. *Duell* durchgeführt, bei dem beide Prozesse jeweils eine Münze werfen (d.h. eine Zufallszahl $\in \{0, 1\}$ wählen) und dies so lange wiederholen, bis einer der beiden eine 1, der andere hingegen eine 0 gewählt hat; der Prozeß mit einer 1 ist dann der Sieger des Duells. Der Erwartungswert für die Anzahl der benötigten Runden ist 2. (Das Duell kann dadurch beschleunigt werden, daß beide Prozesse eine Zahl aus dem Intervall $[1, N]$ wählen, von denen die größere gewinnt; der Erwartungswert für die Anzahl der Runden ist dann $N/(N - 1)$.)

4.4 Analyse des Rendezvous

Zur Analyse des Rendezvous müssen zunächst einige Begriffe eingeführt werden, die das Konzept des Rendezvous konkretisieren.

Definition 4.9 (Anfragefunktion, Rendezvous) Sei $G = (V, E)$ ein ungerichteter, zusammenhängender Graph mit $|V| = n$. Dann wird definiert:

- Eine **Anfragefunktion** (oder schlicht **Anfrage**) auf G ist eine Funktion $c : V \rightarrow V$ mit $c(v) \in \textit{adj}(v) \forall v \in V$. ($\textit{adj}(v) := \{u \in V \mid \{v, u\} \in E\}$ bezeichnet die Menge der Nachbarknoten von v .)

- Die Menge aller Anfragefunktionen über G wird mit $\mathcal{C}(G)$ bezeichnet.
- Zwei Knoten $u \in V$ und $v \in V$ haben ein **Rendezvous**, wenn $c(u) = v$ und $c(v) = u$.
- Eine Anfrage heißt **erfolgreich**, wenn mindestens ein Rendezvous existiert. Ansonsten heißt sie **nicht erfolgreich** (oder **Fehlanfrage**).

□

Unter der Annahme, daß alle Nachbarknoten mit gleicher Wahrscheinlichkeit ausgewählt werden, gilt:

- Die Wahrscheinlichkeit eines Rendezvous zwischen $u \in V$ und $v \in V$ ist

$$p(\{u, v\} \text{ ist ein Rendezvous}) = \frac{1}{\deg(u)} \cdot \frac{1}{\deg(v)} =: q(\{u, v\}).$$

($\deg(v)$ ist der Grad des Knoten v , also $\deg(v) = |\text{adj}(v)|$.)

- Die Wahrscheinlichkeit, daß ein Knoten $u \in V$ ein Rendezvous hat, ist

$$\begin{aligned} p(u \text{ hat ein Rendezvous}) &= \sum_{v \in \text{adj}(u)} q(\{u, v\}) \\ &= \sum_{v \in \text{adj}(u)} \frac{1}{\deg(u)\deg(v)} \\ &= \frac{1}{\deg(u)} \sum_{v \in \text{adj}(u)} \frac{1}{\deg(v)}. \end{aligned}$$

Diese Wahrscheinlichkeit nimmt offensichtlich ab, wenn der Grad von u selbst oder einem $v \in \text{adj}(u)$ abnimmt.

- Die Wahrscheinlichkeit einer einzelnen Anfrage $c \in \mathcal{C}(G)$ ist

$$\alpha(G) := \prod_{v \in V} \frac{1}{\deg(v)}.$$

- Sei $N(G)$ die Anzahl möglicher Fehlanfragen unter G . Dann ist die Wahrscheinlichkeit einer Fehlanfrage

$$f(G) := \alpha(G) \cdot N(G),$$

die einer erfolgreichen Anfrage (= Erfolgswahrscheinlichkeit)

$$s(G) := 1 - \alpha(G) \cdot N(G).$$

Beispiel: Einige Beispiele sollen diese Wahrscheinlichkeiten illustrieren.

- Sei $G = (V, E)$ ein Baum. Dann gilt $s(G) = 1$, d.h. eine Anfrage ist stets erfolgreich.

Um dies einzusehen, betrachten wir eine Anfrage als gerichtete Kanten im Graphen G : ein Funktionswert $c(u) = v$ entspricht einer gerichteten Kante $\langle u, v \rangle$. Nun hat ein Baum mit n Knoten bekanntermaßen $n - 1$ (ungerichtete) Kanten; durch die Anfrage hingegen werden n (gerichtete) Kanten definiert. Also muß es mindestens ein adjazentes Knotenpaar u, v geben, auf das zwei gerichtete Kanten fallen; u und v bilden also ein Rendezvous.

- Sei $G = (V, E)$ ein Ring mit n Knoten. Die Wahrscheinlichkeit einer erfolgreichen Anfrage beträgt dann $s(G) = (2^n - 2)/2^n$.

Dieser Wert gilt, da es bei n Knoten mit jeweils 2 Nachbarn 2^n verschiedene Anfragefunktionen gibt, von denen aber lediglich zwei zu einer Fehlanfrage führen: die, in der alle Knoten ihren Nachbarn im bzw. gegen den Uhrzeigersinn gewählt haben.

■

Um die Wahrscheinlichkeit einer erfolgreichen Anfrage für einen beliebigen Graphen G zu ermitteln, wird eine Anfrage mit einem Matching M im Graphen G gleichgesetzt. Eine Kante $e = \{u, v\}$ des Matchings M entspricht dabei einem Rendezvous zwischen u und v . Das Analogon zu einer Fehlanfrage ist ein „Matching“ mit Kardinalität 0.

Wir erweitern die Definition von q (Wahrscheinlichkeit eines Rendezvous) auf Matchings M : $q(M) := \prod_{e \in M} q(e)$. Desweiteren sei \mathcal{M}_k die Menge aller Matchings mit Kardinalität k (im folgenden auch als k -Matchings bezeichnet). Dann wird definiert:

$$q_k := \sum_{M \in \mathcal{M}_k} q(M) \quad (k = 0 \leq k \leq \lfloor n/2 \rfloor). \quad (4.1)$$

Es sei darauf hingewiesen, daß q_k *nicht* die Wahrscheinlichkeit dafür ist, daß es ein k -Matching gibt, da Anfragen in der Regel mehrfach gezählt werden (z.B. gilt $q_0 = 1$). Um die Wahrscheinlichkeit eines k -Matchings zu bestimmen, muß das Inklusions-Exklusions-Prinzip angewandt werden. Damit ergibt sich die Wahrscheinlichkeit P_k eines k -Matchings zu

$$P_k = \sum_{i=0}^{\lfloor n/2 \rfloor - k} (-1)^i q_{k+i} = q_k - q_{k+1} + q_{k+2} \pm \dots \pm q_{\lfloor n/2 \rfloor}. \quad (4.2)$$

und insbesondere die Erfolgswahrscheinlichkeit im Graph G zu

$$s(G) = P_1 = \sum_{i=0}^{\lfloor n/2 \rfloor - 1} (-1)^i q_{i+1} = q_1 - q_2 + q_3 \pm \dots \pm q_{\lfloor n/2 \rfloor}. \quad (4.3)$$

Damit läßt sich die Erfolgswahrscheinlichkeit für einen vollständigen Graphen abschätzen.

Satz 4.10 Sei $G = (V, E)$ der vollständige Graph mit n Knoten (also $G = K_n$). Dann gilt für die Erfolgswahrscheinlichkeit: $s(G) > 3/8$.

Beweis: Nach Gl. (4.3) gilt:

$$s(G) = q_1 - q_2 + q_3 \pm \dots \pm q_{\lfloor n/2 \rfloor} \geq q_1 - q_2 = \sum_{M \in \mathcal{M}_1} q(M) - \sum_{M \in \mathcal{M}_2} q(M)$$

Die Ungleichung gilt, da die Folge der q_i 's monoton fallend ist. Die beiden Summen lassen sich leicht abschätzen, da G ein vollständiger Graph ist:

$$\begin{aligned} & \sum_{M \in \mathcal{M}_1} q(M) - \sum_{M \in \mathcal{M}_2} q(M) \\ = & \sum_{e \in E} q(e) - \sum_{e, f \in E, e \cap f = \emptyset} q(e)q(f) \\ = & \sum_{e \in E} \frac{1}{(n-1)^2} - \sum_{e, f \in E, e \cap f = \emptyset} \frac{1}{(n-1)^4} \\ = & \binom{n}{2} \frac{1}{(n-1)^2} - \binom{n}{2} \binom{n-2}{2} \frac{1}{2!} \frac{1}{(n-1)^4} \\ = & \frac{n(n-1)}{2} \frac{1}{(n-1)^2} - \frac{n(n-1)(n-2)(n-3)}{8} \frac{1}{(n-1)^4} \\ = & \frac{n}{2(n-1)} - \frac{n(n-2)(n-3)}{8(n-1)^3} \\ = & \frac{n}{2(n-1)} \left(1 - \frac{(n-2)(n-3)}{4(n-1)^2} \right) \\ > & \frac{1}{2} \left(1 - \frac{1}{4} \right) = \frac{3}{8} \end{aligned}$$

■

Wie wir später (in Abschnitt 4.6) sehen werden, ist dies bereits eine relativ gute Abschätzung. Mit dieser Wahrscheinlichkeit läßt sich der Erwartungswert für die Anzahl der Runden unmittelbar angeben.

Korollar 4.11 *Der Erwartungswert für die Anzahl der Anfragen, die bis zum Erfolg nötig sind, ist $< \frac{8}{3}$.*

Beweis: Aufeinanderfolgende Anfragen entsprechen einem Bernoulli-Experiment. Für ein solches Experiment X mit $p(X) = p$ ist bekannt, daß der Erwartungswert $1/p$ beträgt. Dies kann aber auch leicht nachgerechnet werden (mit $q = 1 - p$):

$$\begin{aligned} E(X) &= p \cdot 1 + qp \cdot 2 + q^2p \cdot 3 + \dots \\ &= \frac{p}{q} \cdot \sum_{i=1}^{\infty} i \cdot q^i \\ &= \frac{p}{q} \cdot \frac{q}{(1-q)^2} = \frac{p}{p^2} = \frac{1}{p} \end{aligned}$$

■

Métivier et al. stellen nun die Behauptung auf, daß die Hinzunahme einer Kante die Erfolgswahrscheinlichkeit mindert [Métivier et al., 1996, Seite 25, Behauptung 14]. Diese Aussage stimmt aber nicht und wird im folgenden widerlegt.

Behauptung 4.12 (Behauptung 14 in [Métivier et al., 1996]) *Sei $G = (V, E)$ ein ungerichteter, zusammenhängender (nicht-vollständiger) Graph. Seien $a \in V$ und $b \in V$ zwei Knoten mit $\{a, b\} \notin E$. Dann gilt: $s(G) > s(G' = (V, E \cup \{a, b\}))$.*

Diese Behauptung würde unmittelbar folgende Vermutung implizieren:

Vermutung 4.13 *Der vollständige Graph K_n hat unter allen Graphen mit n Knoten die geringste Erfolgswahrscheinlichkeit.*

Diese (bis heute unbewiesene) Vermutung wird weiter unten ausführlich diskutiert (siehe Abschnitt 4.6).

Der ursprüngliche „Beweis“ obiger Behauptung in [Métivier et al., 1996] konnte an mehreren Stellen nicht nachvollzogen werden. In einer neueren (bisher unveröffentlichten) Version ist er in eine verständlichere Form gebracht worden, die hier wiedergegeben wird. Der Illustration des darin gemachten Denkfehlers könnte sich als hilfreich erweisen, die eigentlich interessierende Vermutung („der vollständige Graph hat die geringste Erfolgswahrscheinlichkeit“) zu beweisen.

„**Beweis**“ (Beweis zu Behauptung 14 bei [Métivier et al., 1996]) Die Wahrscheinlichkeit einer beliebigen Anfrage unter G bzw. G' ist $\alpha := \alpha(G)$ bzw. $\alpha' := \alpha(G')$, wobei gilt:

$$\alpha' = \frac{\deg(a)\deg(b)}{(\deg(a) + 1)(\deg(b) + 1)} \cdot \alpha$$

Der jetzt folgende Schritt ist nicht mehr korrekt:

Sei $c : V \rightarrow V$ eine beliebige Fehlanfrage unter G . Dann sind folgende Anfragen ebenfalls erfolglos:

- $c^{(i)}(x) = c(x) \quad \forall x \in V.$
- $c^{(ii)}(x) = \begin{cases} c(x) & ; x \in V \setminus \{a\} \\ b & ; x = a \end{cases}$
- $c^{(iii)}(x) = \begin{cases} c(x) & ; x \in V \setminus \{a, c(a)\} \\ b & ; x = a \\ a & ; x = c(a) \end{cases}$
- $c^{(iv)}(x) = \begin{cases} c(x) & ; x \in V \setminus \{b\} \\ a & ; x = b \end{cases}$
- $c^{(v)}(x) = \begin{cases} c(x) & ; x \in V \setminus \{b, c(b)\} \\ a & ; x = b \\ b & ; x = c(b) \end{cases}$

Im ersten Moment erscheinen diese 5 Anfragen allesamt plausibel, was sie isoliert betrachtet auch sind. Das Problem liegt aber darin, daß die Konstruktionen (ii) und (iii), ausgehend von unterschiedlichen Anfragefunktionen c_1 und c_2 , die gleiche Anfrage $c_1^{(ii)} = c_2^{(iii)}$ erzeugen können (gleiches gilt für die Konstruktionen (iv) und (v)). Dies ist in Abb. 4.1 veranschaulicht.

Die im ursprünglichen Beweis gemachte Abschätzung für $f(G')$ läßt sich damit nicht mehr durchführen. ■

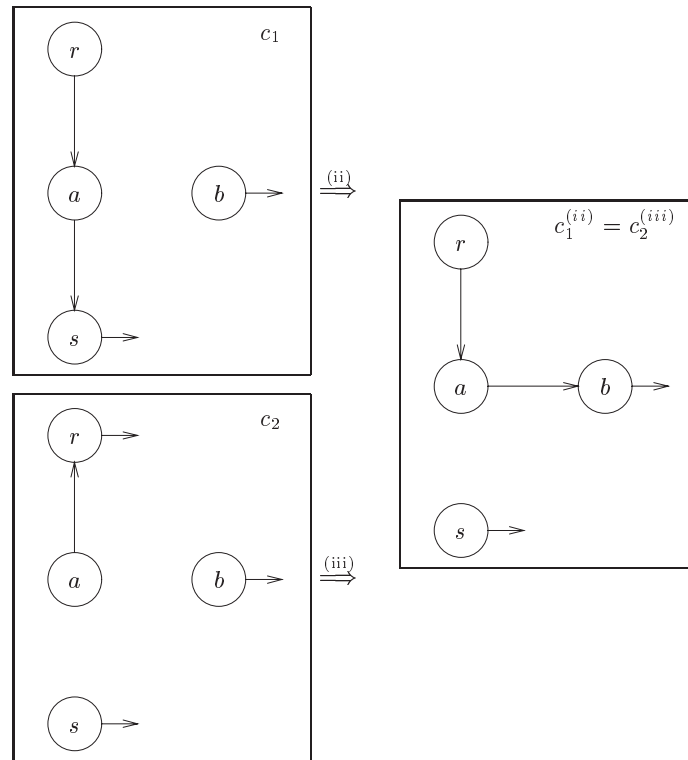


Abbildung 4.1: Veranschaulichung des Fehlers im Beweis zu Behauptung 14 bei Métivier et al. Die beiden Anfragen c_1 und c_2 auf der linken Seite sind verschieden, werden jedoch durch Anwendung der Konstruktionen (ii) bzw. (iii) zur selben Anfragefunktion $c_1^{(ii)} = c_2^{(iii)}$.

4.5 Gegenbeispiel zu Behauptung 14 bei Métivier et al.

Ein falscher Beweis hat natürlich noch keinen großen Aussagewert. Das folgende Gegenbeispiel jedoch widerlegt Beh. 4.12.

Wir betrachten einen Graph $G = (V, E)$ mit $V = \{1, 2, \dots, 7\}$ und $E = \{1, 3\} \cup \{2, 4\} \cup \{\{i, j\} \mid 3 \leq i < j \leq 7\}$. Er besteht also aus einer 5er-Clique und zwei einzelnen Knoten, die jeweils nur eine Verbindung zur Clique besitzen. Dieser Graph ist in Abb. 4.2 dargestellt.

Der Graph G' entsteht durch Hinzunahme der Kante $\{1, 2\}$. Die Anzahl verschiedener Anfragefunktionen unter G bzw. G' läßt sich leicht aus den jeweiligen Knotengraden bestimmen:

- Unter G gibt es $1^2 \cdot 4^3 \cdot 5^2 = 64 \cdot 25 = 1600$ verschiedene Anfragen.
- In G' erhöht sich der Grad zweier Knoten von 1 auf 2; daraus ergeben sich $2^2/1^2 \cdot 1600 = 6400$ verschiedene Anfragen.

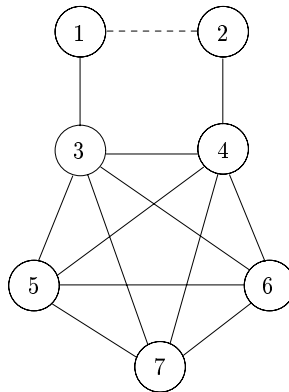


Abbildung 4.2: Gegenbeispiel zur Behauptung 14 bei Métivier et al. Die Kante, die bei G' hinzugenommen wird, ist gestrichelt gezeichnet.

4.5.1 Erfolgreiche Anfragen unter G

Der Graph G besteht aus zwei Knoten (1 und 2), deren Funktionswert für alle Anfragen feststeht ($c(1) = 3$ und $c(2) = 4$). Die restlichen 5 Knoten bilden eine Clique. Um die Anzahl erfolgreicher Anfragen in G zu ermitteln, werden folgende Fälle unterschieden:

- [G-1] $c(3) = 1$ und $c(4) = 2$: Diese Anfragen sind alle erfolgreich (Rendezvous $\{1, 3\}$ und $\{2, 4\}$). In diesem Fall gibt es $4^3 = \mathbf{64}$ verschiedene Anfragen (die Funktionswerte der Knoten 1 bis 4 liegen fest; die Knoten 5 bis 7 können frei unter jeweils 4 Nachbarknoten wählen).
- [G-2] $c(3) = 1$ und $c(4) \neq 2$: Diese Anfragen sind ebenfalls alle erfolgreich (Rendezvous $\{1, 3\}$). In diesem Fall gibt es $4^4 = \mathbf{256}$ verschiedene Anfragen (die Funktionswerte der Knoten 1, 3 und 4 liegen fest; Knoten 2 darf einen der Nachbarn 3 bis 7 wählen, nicht aber Knoten 2; und schließlich dürfen die Knoten 5 bis 7 wieder beliebig unter ihren 4 Nachbarknoten wählen).
- [G-3] $c(3) \neq 1$ und $c(4) = 2$: Symmetrisch zu [G-2].
- [G-4] $c(3) \neq 1$ und $c(4) \neq 2$: In diesem Fall entspricht die Anzahl erfolgreicher Anfragen genau der Anzahl erfolgreicher Anfragen in einer 5er-Clique. Diese kann mit dem Inklusions-Exklusions-Prinzip ermittelt werden:

- (a) Es gibt $\binom{5}{2}$ Knotenpaare, die ein Rendezvous haben können. Die jeweils verbleibenden 3 Knoten können unter ihren 4 Nachbarn frei wählen. Also gibt es

$$\binom{5}{2} \cdot 4^3 = 10 \cdot 64 = 640$$

Anfragen mit mindestens einem Rendezvous.

- (b) Die Anfragen mit zwei Rendezvous sind darin allerdings doppelt gezählt und wieder subtrahiert werden. Es gibt $\binom{5}{2} \cdot \binom{3}{2} \cdot \frac{1}{2!}$ Möglichkeiten, zwei Knotenpaare auszuwählen. Der fünfte Knoten hat wieder die freie Wahl

unter seinen 4 Nachbarn. Die Anzahl der Anfragen mit 2 Rendezvous ist damit

$$\binom{5}{2} \cdot \binom{3}{2} \cdot \frac{1}{2!} \cdot 4 = 10 \cdot 3 \cdot \frac{1}{2} \cdot 4 = 60.$$

(c) Anfragen mit 3 oder mehr Rendezvous sind bei 5 Knoten nicht möglich.

Die Anzahl erfolgreicher Anfragen in einer 5er-Clique ist also $640 - 60 = 580$.

Wenn wir nun alle 4 Fälle aufsummieren, erhalten wir insgesamt $64 + 2 \cdot 256 + 580 = 1156$ erfolgreiche Anfragen unter G .

4.5.2 Erfolgreiche Anfragen unter G'

Nun bestimmen wir die Anzahl erfolgreicher Anfragen unter G' , indem wir wieder 4 Fälle unterscheiden, diesmal abhängig von den Funktionswerten $c(1)$ und $c(2)$:

[G' -1] $c(1) = 3$ und $c(2) = 4$: Dieser Fall, in dem die zusätzliche Kante $\{1, 2\}$ nicht benutzt wird, entspricht dem Graphen G . Es gibt also **1156** erfolgreiche Anfragen.

[G' -2] $c(1) = 2$ und $c(2) = 1$: In diesem Fall sind alle **1600** Anfragen erfolgreich, denn $\{1, 2\}$ ist offensichtlich ein Rendezvous.

[G' -3] $c(1) = 3$ und $c(2) = 1$: Dieser Fall unterscheidet sich von Anfragen unter G nur durch den Funktionswert von Knoten 2. Erfolgreiche Anfragen der Fälle [G -1], [G -2] und [G -4] bleiben erfolgreich, da jeweils ein von $\{2, 4\}$ verschiedenes Knotenpaar für das Rendezvous verantwortlich war.

Zu untersuchen bleibt der Fall [G -3], bei dem ein Rendezvous über $\{2, 4\}$ stattfand, das aber nicht das einzige gewesen sein muß. Die Frage ist also: wie viele der erfolgreichen Anfragen unter [G -3] haben neben $\{2, 4\}$ ein weiteres Rendezvous?

Die Funktionswerte der Knoten 1 und 2 liegen (durch den aktuellen Unterfall) fest. Da wir uns auf den Fall [G -3] beschränken, muß außerdem $c(3) \neq 1$ und $c(4) = 2$ gelten. Also besitzen nur die Knoten 3, 5, 6 und 7 Freiheitsgrade (mit jeweils 4 möglichen Funktionswerten).

Wiederum wenden wir das Inklusions-Exklusions-Prinzip an. Unter diesen 4 Knoten gibt es

$$\binom{4}{2} \cdot 4^2 = 6 \cdot 16 = 96$$

Anfragen mit mindestens einem Rendezvous und

$$\binom{4}{2} \cdot \binom{2}{2} \cdot \frac{1}{2!} = 6 \cdot 1 \cdot \frac{1}{2} = 3.$$

Anfragen mit 2 Rendezvous, insgesamt also $96 - 3 = 93$ Anfragen.

Anders ausgedrückt heißt das: von den 256 erfolgreichen Anfragen unter [G -3] sind $256 - 93 = 163$ nur durch ein einzelnes Rendezvous $\{2, 4\}$ erfolgreich. Diese Anfragen werden im aktuellen Fall zu Fehlanfragen.

Da alle erfolgreichen Anfragen der anderen 3 Fälle erfolgreich bleiben, gibt es insgesamt $1156 - 163 = 993$ erfolgreiche Anfragen.

[G' -4] $c(1) = 2$ und $c(2) = 4$: Symmetrisch zu [G' -3].

Wiederum werden alle 4 Fälle aufsummiert. Dies ergibt $1156 + 1600 + 2 \cdot 993 = 4742$ erfolgreich Anfragen unter G' .

4.5.3 Vergleich der Erfolgswahrscheinlichkeiten

Nun ist es einfach, die Erfolgswahrscheinlichkeiten für G bzw. G' auszurechnen:

- $s(G) = 1156/1600 = 0.7225$
- $s(G') = 4742/6400 = 0.7409..$

Die Wahrscheinlichkeit einer erfolgreichen Anfrage hat also — im Widerspruch zu Beh. 4.12 — durch Hinzunahme einer Kante deutlich zugenommen.

Intuitiv kann man sich dieses Ergebnis wie folgt klar machen: unter G ist die Wahrscheinlichkeit relativ gering, daß einer der Knoten 1 bzw. 2 ein Rendezvous hat, denn sein jeweiliger Nachbarknoten 3 bzw. 4 hat sehr viele Nachbarn in der 5er-Clique; wird jetzt die Kante $\{1, 2\}$ hinzugenommen, wird also nur eine kleine Anzahl erfolgreicher Anfragen erfolglos, wohingegen $1/4$ aller Anfragen unter G' über die neue Kante erfolgreich sind.

Die Zunahme der Erfolgswahrscheinlichkeit dürfte also noch drastischer ausfallen, wenn die 5er-Clique durch eine k -Clique mit sehr großem k ersetzt wird. Weiter oben wurde die Vermutung geäußert, daß die Erfolgswahrscheinlichkeit eines vollständigen Graphen bei zunehmender Knotenzahl gegen einen Wert $< 0,4$ strebt. Die Hinzunahme zweier Knoten (analog zum Gegenbeispiel) zum vollständigen Graphen dürfte recht insignifikant sein; werden diese beiden aber verbunden, wird die Erfolgswahrscheinlichkeit von $\approx 0,4$ auf $\approx 0,65$ steigen.

4.6 Wahrscheinlichkeit eines Rendezvous in beliebigen Graphen

Die Frage, ob der vollständige Graph unter allen Graphen mit n Knoten die geringste Rendezvous-Wahrscheinlichkeit besitzt, ist nach wie vor offen. Obiges Gegenbeispiel hat gezeigt, daß die Adjunktion einer beliebigen Kante nicht zwingend zu einer geringeren Erfolgswahrscheinlichkeit führt.

Es bestand aber die Hoffnung, daß sich folgende Vermutung bewahrheiten könnte:

Vermutung 4.14 *Für jeden ungerichteten, zusammenhängenden, nicht-vollständigen Graphen $G = (V, E)$ existiert eine Kante $e \notin E$ so, daß für $G' = (V, E \cup e)$ gilt: $s(G) > s(G')$.*

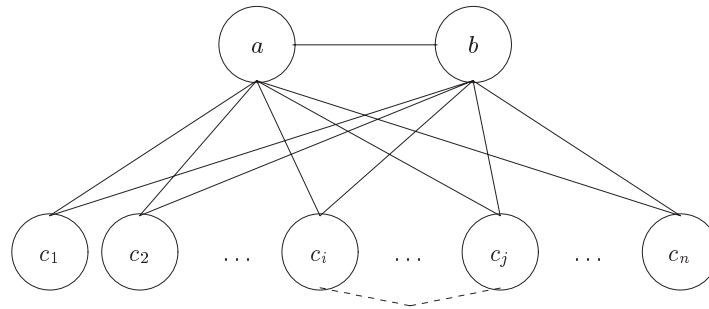


Abbildung 4.3: Mike Robsons Gegenbeispiel zur Vermutung, daß stets eine Kante existiert, deren Hinzunahme die Erfolgswahrscheinlichkeit verringert. (Die hinzugenommene Kante ist gestrichelt gezeichnet.)

Würde diese Vermutung gelten, könnte man wiederum durch ein induktives Argument beweisen, daß unter allen Graphen mit n Knoten der vollständige Graph die geringste Erfolgswahrscheinlichkeit besitzt. Das folgende Gegenbeispiel von Mike Robson (persönliche Kommunikation) macht diese Hoffnung aber zunichte.

Sei $G = (V, E)$ ein Graph mit $V = \{a, b, c_1, \dots, c_n\}$ und

$$E = \{a, b\} \cup \{a, c_i\} \mid 1 \leq i \leq n \cup \{b, c_i\} \mid 1 \leq i \leq n$$

(siehe Abb. 4.3). Wir nehmen an, daß n sehr groß ist.

Eine Abschätzung für die Erfolgswahrscheinlichkeit läßt sich leicht durchführen (eine genaue Berechnung der Wahrscheinlichkeiten ist nicht nötig, da die nicht berücksichtigten Wahrscheinlichkeiten durch ausreichend große Wahl von n beliebig klein werden). Der Knoten a wird mit sehr großer Wahrscheinlichkeit einen Knoten c_a ($1 \leq a \leq n$) und nicht b auswählen (gleiches gilt für b). Seien also c_a bzw. c_b die Knoten, die a bzw. b auswählen. Mit einer Wahrscheinlichkeit von $\approx 1/2$ wählt c_a aber b aus, sodaß kein Rendezvous $\{a, c_a\}$ entsteht (gleiches gilt wiederum für $\{b, c_b\}$). Insgesamt findet also mit Wahrscheinlichkeit $\approx 1/2 \cdot 1/2 = 1/4$ kein Rendezvous statt (die Wahrscheinlichkeit für ein Rendezvous beträgt $\approx 3/4$).

Eine weitere Kante kann nur zwischen zwei Knoten c_i und c_j ($i \neq j$) eingefügt werden. Bei großem n hat dies auf die Wahrscheinlichkeit eines Rendezvous $\{a, c_a\}$ bzw. $\{b, c_b\}$ praktisch keinen Einfluß. Ein Rendezvous $\{c_i, c_j\}$ findet aber mit Wahrscheinlichkeit $1/9$ statt. Dieses ist genau dann das einzige Rendezvous in G , wenn sowohl a als auch b kein Rendezvous haben; dann trägt die Kante $\{c_i, c_j\}$ aber eine Wahrscheinlichkeit von $\approx 1/4 \cdot 1/9 = 1/36$ zum Erfolg bei.

Also erhöht die Hinzunahme jeder beliebigen Kante zu G die Erfolgswahrscheinlichkeit signifikant. Obige Vermutung ist demnach widerlegt und gibt uns keine Möglichkeit, etwas über den vollständigen Graphen auszusagen.

Von Mike Robson existiert jedoch eine Abschätzung für die Rendezvous-Wahrscheinlichkeit in beliebigen Graphen (persönliche Kommunikation).

Satz 4.15 Sei $G = (V, E)$ ein ungerichteter, zusammenhängender Graph. Dann ist die Wahrscheinlichkeit eines Rendezvous in G größer $1 - 1/\sqrt{e} = 0,3934\dots$

Beweis (von Mike Robson): Seien $G = (V, E)$ ein Graph mit $V = \{v_1, \dots, v_n\}$ und $E = \{e_1, \dots, e_m\}$ sowie deg_{max} der maximale Knotengrad im Graphen G . Wenn x ein Knoten oder eine Kante ist, dann bezeichnet $Rdv(x)$ bzw. $\neg Rdv(x)$ die Tatsache, daß x ein Rendezvous bzw. kein Rendezvous hat.

Dann ist die Wahrscheinlichkeit dafür, daß ein Knoten $v \in V$ ein Rendezvous hat,

$$\text{Prob}(Rdv(v \in V)) \geq \frac{1}{deg_{max}}, \quad (4.4)$$

denn v wählt einen seiner Nachbarknoten w aus, der ihn wiederum mit Wahrscheinlichkeit $1/deg(w) \geq 1/deg_{max}$ auswählt.

Der Erwartungswert für die Anzahl der Knoten, die an einem Rendezvous beteiligt sind, ist damit offensichtlich

$$E(\# \text{ Knoten mit Rendezvous}) = \frac{n}{deg_{max}}. \quad (4.5)$$

Die Anzahl der Kanten, über die ein Rendezvous stattfindet, ist halb so groß wie die Anzahl der Knoten mit Rendezvous. Der Erwartungswert dafür beträgt also

$$E(\# \text{ Kanten mit Rendezvous}) = \frac{n}{2deg_{max}} > \frac{1}{2}. \quad (4.6)$$

Dieser Erwartungswert läßt sich aber auch wie folgt ausdrücken:

$$E(\# \text{ Kanten mit Rendezvous}) = \sum_{e \in E} p(e) \cdot 1, \quad (4.7)$$

wobei $p(e)$ die Wahrscheinlichkeit dafür ist, daß die Kante $e \in E$ ein Rendezvous hat. Damit gilt:

$$\sum_{e \in E} p(e) > \frac{1}{2}. \quad (4.8)$$

Sei nun $p'(e_i)$ die bedingte Wahrscheinlichkeit dafür, daß $e_i \in E$ ein Rendezvous hat unter der Voraussetzung, daß keine der Kanten e_1, \dots, e_{i-1} ein Rendezvous hatte:

$$p'(e_i) = \text{Prob}(Rdv(e_i) \mid \neg Rdv(e_1) \wedge \dots \wedge \neg Rdv(e_{i-1})) \quad (4.9)$$

Der entscheidende Schritt ist nun die Feststellung, daß $p'(e) \geq p(e)$ für all $e \in E$ gilt. Sei $e_i = \{u, v\}$ die betrachtete Kante,

$$C' = \{c : V \setminus \{u, v\} \rightarrow V \mid c(v) \in adj(v) \forall v \in V \setminus \{u, v\}\} \quad (4.10)$$

die Menge der Anfragefunktionen auf $V \setminus \{u, v\}$ und

$$H = \neg Rdv(e_1) \wedge \dots \wedge \neg Rdv(e_{i-1}) \quad (4.11)$$

die Hypothese, daß keine der Kanten e_1, \dots, e_{i-1} ein Rendezvous hat. Dann gilt:

$$\begin{aligned} p'(e_i) &= \text{Prob}(\text{Rdv}(e_i) \mid H) \\ &= \sum_{c \in \mathcal{C}'} \text{Prob}(c \mid H) \cdot \text{Prob}(\text{Rdv}(e_i) \mid H \wedge c) \end{aligned} \quad (4.12)$$

Desweiteren seien für jedes $c \in \mathcal{C}'$

$$\begin{aligned} u_c &= \#\{e \in \{e_1, \dots, e_{i-1} \mid \exists w \in V : \{u, w\} \in E \wedge c(w) = u\}\} \text{ und} \\ v_c &= \#\{e \in \{e_1, \dots, e_{i-1} \mid \exists w \in V : \{v, w\} \in E \wedge c(w) = v\}\} \end{aligned}$$

die Anzahl der Nachbarknoten von u (bzw. v) mit Funktionswert u (bzw. v) unter der Anfrage c . Dann gilt:

$$\text{Prob}(\text{Rdv}(e_i) \mid H \wedge c) = \frac{1}{(\text{deg}(u) - u_c)(\text{deg}(v) - v_c)}, \quad (4.13)$$

und für Gl. (4.12)

$$\begin{aligned} p'(e_i) &= \sum_{c \in \mathcal{C}'} \text{Prob}(c \mid H) \cdot \text{Prob}(\text{Rdv}(e_i) \mid H \wedge c) \\ &= \sum_{c \in \mathcal{C}'} \frac{\text{Prob}(c \mid H)}{(\text{deg}(u) - u_c)(\text{deg}(v) - v_c)} \\ &\geq \sum_{c \in \mathcal{C}'} \frac{\text{Prob}(c \mid H)}{\text{deg}(u)\text{deg}(v)} \\ &= \sum_{c \in \mathcal{C}'} \text{Prob}(c \mid H) \cdot p(e_i) \\ &= p(e_i) \end{aligned} \quad (4.14)$$

Damit läßt sich jetzt die Wahrscheinlichkeit dafür abschätzen, daß in G kein Rendezvous stattfindet:

$$\begin{aligned} &\text{Prob}(G \text{ hat kein Rendezvous}) \\ &= \text{Prob}(\neg \text{Rdv}(e_1)) \cdot \text{Prob}(\neg \text{Rdv}(e_2) \mid \neg \text{Rdv}(e_1)) \\ &\quad \dots \cdot \text{Prob}(\neg \text{Rdv}(e_m) \mid \neg \text{Rdv}(e_1) \wedge \dots \wedge \neg \text{Rdv}(e_{m-1})) \\ &= \prod_{i=1}^m (1 - p'(e_i)) \\ &\leq \prod_{i=1}^m (1 - p(e_i)) \end{aligned} \quad (4.15)$$

Gesucht ist also eine obere Schranke für Gl. (4.15) unter der durch Gl. (4.8) gegebenen Nebenbedingung.

Das Maximum in Gl. (4.15) wird genau dann angenommen, wenn alle $p(e_i)$'s den gleichen Wert haben. Wegen Gl. (4.8) gilt dann aber $p(e_i) > 1/(2m)$ und

$$\prod_{i=1}^m (1 - p(e_i)) < \prod_{i=1}^m \left(1 - \frac{1}{2m}\right) = \left(1 - \frac{1}{2m}\right)^m < \frac{1}{\sqrt{e}}. \quad (4.16)$$

Der Grenzwert von $(1 - 1/2m)^m$ läßt sich leicht bestimmen:

$$\begin{aligned} \left(1 - \frac{1}{2m}\right)^m &= \left(\frac{2m-1}{2m}\right)^m = \left(\frac{2m}{2m-1}\right)^{-m} = \left(1 + \frac{1}{2m-1}\right)^{-m} \\ &= \left(1 + \frac{1}{x}\right)^{-(x+1)/2} \quad (\text{mit } x = 2m - 1) \\ &= \left[\left(1 + \frac{1}{x}\right)^{x+1}\right]^{-1/2} \\ &\rightarrow e^{-1/2} = \frac{1}{\sqrt{e}} \quad \text{für } x \rightarrow \infty \end{aligned}$$

Schließlich ist noch die Ungleichung selbst abzuschätzen. Die Behauptung lautet:

$$\begin{aligned} \left(1 - \frac{1}{2m}\right)^m &= \left[\left(1 + \frac{1}{x}\right)^{x+1}\right]^{-1/2} < \frac{1}{\sqrt{e}} = e^{-1/2} \\ \Leftrightarrow \left(1 + \frac{1}{x}\right)^{x+1} &< e \end{aligned} \quad (4.17)$$

Beweis:

- Für $x = 1$ ist $(1 + 1/x)^{x+1} = 4 > e = 2,7182\dots$
- Die Folge der $(1 + 1/x)^{x+1}$ ist streng monoton fallend:

$$\begin{aligned} \left(1 + \frac{1}{x-1}\right)^x &> \left(1 + \frac{1}{x}\right)^{x+1} \\ \Leftrightarrow \left(\frac{x}{x-1}\right)^x &> \left(\frac{x+1}{x}\right)^{x+1} \\ \Leftrightarrow \frac{x^x}{(x-1)^x} &> \frac{(x+1)^{x+1}}{x^{x+1}} \\ \Leftrightarrow \frac{x^x}{(x-1)^x} &> \frac{(x+1)^x}{x^x} \cdot \frac{x+1}{x} \\ \Leftrightarrow \frac{(x^2)^x}{(x^2-1)^x} &> \frac{x+1}{x} \\ \Leftrightarrow \left(1 + \frac{1}{x^2-1}\right)^x &> 1 + \frac{1}{x} \end{aligned}$$

Der letzte Schritt folgt aus der Bernoullischen Ungleichung, nach der $(1+a)^b > 1 + b \cdot a$ für $a > 0$ gilt. Mit $a = 1/(x^2 - 1)$ und $b = x$ gilt:

$$\left(1 + \frac{1}{x^2-1}\right)^x > 1 + \frac{x}{x^2-1} > 1 + \frac{x}{x^2} = 1 + \frac{1}{x}$$

- Der Grenzwert von $(1 + 1/x)^{x+1}$ ist allgemein bekannt:

$$\lim_{x \rightarrow \infty} \left(1 + \frac{1}{x}\right)^{x+1} = \lim_{x \rightarrow \infty} \left(1 + \frac{1}{x}\right)^x = e$$

Aus diesen drei Fakten (Startwert $> e$, monoton fallende Folge, Grenzwert e) folgt die Behauptung unmittelbar. ■

Damit ist Satz 4.15 bewiesen. ■

Mit Hilfe des Inklusions–Exklusions–Prinzips wurden die Erfolgswahrscheinlichkeiten für vollständige Graphen bis 100.000 Knoten ermittelt. Diese sind als Schaubild in Abb. 4.4 dargestellt.

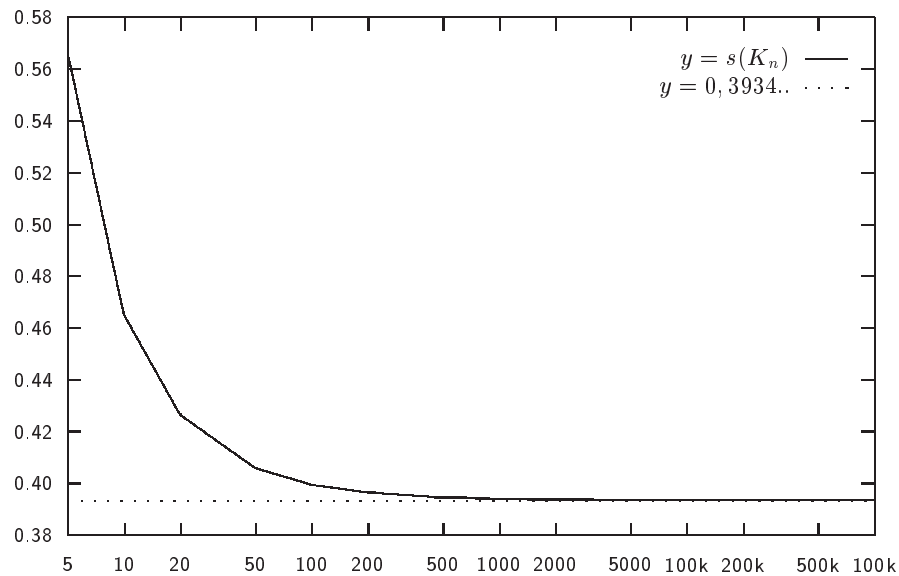


Abbildung 4.4: Erfolgswahrscheinlichkeit $s(K_n)$ für ein Rendezvous im vollständigen Graphen K_n , ermittelt mit dem Inklusions–Exklusions–Prinzip (logarithmische Skala; x–Achse= n , y–Achse= $s(K_n)$). Die gestrichelte Linie ist die untere Schranke von $1 - 1/\sqrt{e} = 0,3934\dots$

Danach ist zu vermuten, daß die Schranke von $1 - 1/\sqrt{e}$ der exakte Grenzwert für vollständige Graphen ist. (Beispielsweise liegt die exakte Wahrscheinlichkeit eines Rendezvous in $K_{100.000}$ um weniger als $6,1 \cdot 10^{-6}$ über dieser Schranke.) Tatsächlich kann man obigen Beweis unter der Annahme eines vollständigen Graphen K_n mit sehr großem n nochmals nachvollziehen und wird feststellen, daß die jeweiligen Abschätzungen den Grenzwerten entsprechen:

- Der maximale Grad im vollständigen Graphen ist $n - 1$. Damit ist der Erwartungswert für die Anzahl der Kanten mit Rendezvous $n/2(n - 1) \rightarrow 1/2$ (Gl. (4.6)).
- Gleiches gilt für die Summe aller $p(e)$ (Gl. (4.8)).

- Die Werte u_c und v_c werden beliebig klein, denn die Wahrscheinlichkeit dafür, daß eine der Kanten e_1, \dots, e_{i-1} adjazent zu u bzw. v ist und noch zusätzlich u bzw. v als Funktionswert gewählt wird, nimmt mit wachsendem n ab (Gl. (4.14)).
- Schließlich ist der Fall, unter dem die Rendezvouswahrscheinlichkeit in G maximiert wird, durch einen vollständigen Graphen automatisch gegeben: alle Kantenwahrscheinlichkeiten sind gleich (Gl. (4.16)).

Damit ist gezeigt, daß $\lim_{n \rightarrow \infty} s(K_n) = (1 - 1/\sqrt{e})$ ist und kein Graph G existiert, dessen Erfolgswahrscheinlichkeit unter diesem Grenzwert liegt. Vermutung 4.13 ist aber immer noch offen: es wäre nach wie vor denkbar, daß für ein festes n ein Graph G mit n Knoten existiert, für den $s(G) < s(K_n)$ gilt.

Kapitel 5

Graph–Faktorisierung ist NP–vollständig

5.1 Partitionierung ist NP–vollständig

Definition 5.1 (MPART) Sei $W = \{w_1, \dots, w_n\}$, $w_i \in \mathbb{N}$ eine Menge von Gewichten, die in unärer Codierung gegeben sind. Gesucht ist eine Partitionierung von W in $1 < m < n$ Mengen W_i so, daß gilt:

$$\sum_{w \in W_j} w = \frac{1}{m} \sum_{w \in W} w \quad \forall 1 \leq j \leq m.$$

Dieses Problem wird als MPART (Multi–Partition) bezeichnet. \square

Satz 5.2 *MPART ist NP–vollständig.*

Für den Beweis dieses Satzes muß zunächst das Problem des 3–dimensionalen Matchings eingeführt werden.

Definition 5.3 (3DM) Seien $X = \{x_1, \dots, x_q\}$, $Y = \{y_1, \dots, y_q\}$ und $Z = \{z_1, \dots, z_q\}$ drei gleichmächtige Mengen und $Q \subseteq X \times Y \times Z$ gegeben. Gesucht ist eine Menge $M \subseteq Q$ so, daß gilt:

1. $|M| = q$
2. $\Pi_X(M) = X$, $\Pi_Y(M) = Y$, $\Pi_Z(M) = Z$, wobei $\Pi_{X/Y/Z}(M)$ die Projektion von M auf die $X/Y/Z$ –Komponente ist (z.B. $\Pi_X(M) := \{x \mid \langle x, y, z \rangle \in M\}$).

Dieses Problem heißt 3–dimensionales Matching und wird als 3DM bezeichnet. \square

Beweis (Beweisidee von Klaus Reinhardt, Universität Tübingen): Für den Beweis dieses Satzes wird 3DM auf MPART reduziert.

Ohne Beschränkung können wir annehmen, daß $q + 1$ eine Primzahl ist (dies kann durch Hinzunahme weiterer Elemente zu X , Y und Z sowie trivialer Tripel zu Q und M sichergestellt werden; das Auffinden der nächsten Primzahl ist wegen der unären Codierung von q und wegen Bertrands Postulat, nach dem es eine Primzahl $< 2q$ gibt, leicht möglich).

Aus der Eingabe konstruieren wir nun 3 verschiedene Gewichte, die mit den Farben rot, blau und schwarz bezeichnet werden.

Rote Gewichte

Sei $b = 2q + 1$. Die roten Gewichte werden dann als Codierung der Eingabetripel $v \in Q$ zur Basis b definiert:

$$r(v = \langle x_i, y_j, z_k \rangle) := i + jb + kb^2 + b^3 \quad (5.1)$$

Die obere Schranke für die Summe aller $r(v)$, $v \in Q$ ist

$$R_Q := \sum_{v \in Q} r(v) \leq \frac{q(q+1)}{2} q^2 (1 + b + b^2) + q^3 b^3 < b^6, \quad (5.2)$$

denn für jede der q^2 Kombination von j und k kann i die Werte $1, \dots, q$ annehmen (analog für j bzw. k).

Die Summe aller $r(v)$, $v \in M$ für ein 3-dimensionales Matching M ist

$$R_M := \sum_{v \in M} r(v) = \frac{q(q+1)}{2} (1 + b + b^2) + qb^3 < b^4. \quad (5.3)$$

Blaue Gewichte

Sei $p = 3d + e$ eine Primzahl mit $d = 3b^6$ und $e \in \{1, 2\}$ (intuitiv gilt: b ist klein, d ist groß). Definiere für jedes $1 \leq l \leq q$ drei blaue Gewichte:

$$\begin{aligned} b_1(l) &:= d - l \\ b_2(l) &:= d - lb \\ b_3(l) &:= d - lb^2 - b^3 + e. \end{aligned} \quad (5.4)$$

Die Summe aller blauen Gewichte ist

$$\begin{aligned} B &:= \sum_{l=1}^q [b_1(l) + b_2(l) + b_3(l)] \\ &= \sum_{l=1}^q [3d - l(1 + b + b^2) - b^3 + e] \\ &= q(3d + e) - \frac{q(q+1)}{2} (1 + b + b^2) - qb^3 \\ &= pq - R_M \end{aligned} \quad (5.5)$$

Schwarzes Gewicht

Schließlich definieren wir ein **schwarzes Gewicht** s so, daß die Summe aus s und allen in M nicht verwendeten roten Gewichten p ergibt, also

$$s := p - R_Q + R_M \quad (5.6)$$

Summe aller Gewichte

Die roten, blauen und das schwarze Gewicht summieren sich zu

$$\begin{aligned} & R_Q + B + s \\ = & R_Q + (pq - R_M) + (p - R_Q + R_M) \\ = & pq + p \\ = & p(q + 1) \end{aligned} \quad (5.7)$$

Die Behauptung lautet nun: Es existiert ein 3D-Matching $M \subseteq Q$ genau dann, wenn eine nicht-triviale Partitionierung der roten, blauen und schwarzen Gewichte existiert.

3D-Matching \Rightarrow Partitionierung

Nehmen wir also an, daß ein 3D-Matching $M \subseteq Q$ existiert. Wir konstruieren daraus $q + 1$ Mengen wie folgt: jede Menge W_1, \dots, W_q erhält genau ein codiertes Element $w = \langle x_i, y_j, z_k \rangle \in M$ und wird mit den drei blauen Gewichten $b_1(i)$, $b_2(j)$ und $b_3(k)$ aufgefüllt. In der Summe ergibt das

$$r(w) + (d - i) + (d - jb) + (d - kb^2 - b^3 + e) = 3d + e = p.$$

Die Menge W_{q+1} erhält das schwarze Gewicht s und alle verbleibenden blauen Gewichte. Das Gewicht s wurde gerade so gewählt, daß diese Summe p ergibt:

$$s + (R_Q - R_M) = (p - R_Q + R_M) + (R_Q - R_M) = p \quad (5.8)$$

Damit existiert eine Partitionierung aller Gewichte in $q + 1$ Mengen mit einer Gewichtssumme von jeweils p .

Partitionierung \Rightarrow 3D-Matching

Wir nehmen nun an, daß eine Partitionierung der farbigen Gewichte in Mengen mit gleicher Gewichtssumme existiert. Die Summe aller Gewichte ist $p(q + 1)$, und da p und $q + 1$ Primzahlen sind, bestehen nur zwei Möglichkeiten einer Partitionierung. Eine Partitionierung in p Mengen mit Gewichtssumme $q + 1$ scheidet aus, da $s = p - R_Q + R_M \geq 3d + e - R_Q > 2d > q + 1$ ist.

Also bleibt nur eine Partitionierung in $q + 1$ Mengen mit Gewichtssumme p . Ohne Einschränkung nehmen wir an, daß s in W_{q+1} enthalten ist.

Nach dem Schubkastenprinzip kann nun die genaue Verteilung aller Gewichte ermittelt werden:

- Jede Menge $W_i \neq W_{q+1}$ muß genau 3 blaue Gewichte enthalten.
 - Die Summe 4 blauer Gewichte ist

$$4 \cdot \text{„blau“} > 4(d - qb^2 - b^3 + e) > (3d + e) + d - 8b^3 > p,$$
 da $d = 3b^6 \geq 8b^3$ für $b \geq 2$.

- 2 blaue Gewichte sind nicht möglich, da sonst ein blaues Gewicht zu der Menge mit s (also zu W_{q+1}) hinzugefügt werden müßte. Dies ergäbe aber eine Summe von

$$\begin{aligned} s + \text{„blau“} &\geq (p - R_Q + R_M) + (d - qb^2 - b^3 + e) \\ &> (p - b^6) + (d - 2b^3) \\ &\geq p, \end{aligned}$$

$$\text{da } d = 3b^6 \geq b^6 + 2b^3.$$

Daraus folgt, daß die Mengen W_1, \dots, W_q jeweils genau 3 blaue Gewichte enthalten, W_{q+1} hingegen keines.

- *Jede Menge $W_i \neq W_{q+1}$ muß genau ein rotes Gewicht enthalten.*

Angenommen, eine Menge W_i würde (neben den drei blauen Gewichten) zwei rote Gewichte enthalten. Dann müßte diese Menge mindestens zwei b_3 -Gewichte enthalten, denn bei lediglich einem b_3 -Gewicht wäre die Summe

$$\begin{aligned} &(2 \cdot \text{„rot“}) + (1 \cdot \text{„}b_3\text{“}) + (2 \cdot \text{„}b_1 \text{ oder } b_2\text{“}) \\ &> 2(b^2 + b^3) + (d - qb^2 - b^3 + e) + 2(d - qb) \\ &= \underbrace{(3d + e)}_{=p} + \underbrace{b^3 - qb^2}_{>1, \text{ da } b>q} + \underbrace{2b^2 - 2qb}_{>1, \text{ da } b>q} \\ &> p. \end{aligned}$$

Wären also 2 rote Gewichte in W_i , dann müßte diese Menge auch mindestens zwei b_3 -Gewichte enthalten. Dann gäbe es aber eine andere Menge W_j ($j \neq i$), die kein b_3 -Gewicht enthält. Da sich die drei blauen Gewichte nicht zu p summieren, müßte auch W_j mindestens ein rotes Gewicht enthalten:

$$\begin{aligned} \sum_{w \in W_j} &= (\text{„rot“}) + (3 \cdot \text{„}b_1 \text{ oder } b_2\text{“}) \\ &> (2 + b^3) + 3(d - qb) \\ &= \underbrace{(3d + 2)}_{\geq p} + \underbrace{(b^3 - 3qb)}_{>1} \\ &> p. \end{aligned}$$

Also muß jede Menge W_1, \dots, W_q genau ein rotes Gewicht enthalten. Aus der letzten Rechnung folgt außerdem, daß jede Menge genau ein b_3 -Gewicht enthalten muß.

- *Das rote Gewicht legt die Wahl der drei blauen Gewichte fest.*

Sei $r(v = \langle x_i, y_j, z_k \rangle) = i + jb + kb^2 + b^3$ das rote Gewicht. Im letzten Schritt wurde gezeigt, daß es genau ein b_3 -Gewicht geben muß; dieses sei $b_3(l)$. Damit gilt für $l < k$:

$$\begin{aligned} &(i + jb + kb^2 + b^3) + (d - lb^2 - b^3 + e) + (2 \cdot \text{„}b_1 \text{ oder } b_2\text{“}) \\ &= (d + e) + (i + jb + (k - l)b^2) + (2 \cdot \text{„}b_1 \text{ oder } b_2\text{“}) \\ &\geq (3d + e) + (i + jb + (k - l)b^2) + (-2qb) \\ &= p + i + (j - 2q)b + (k - l)b^2 \\ &> p - 2qb + b^2 \\ &= p + b(b - 2q) \\ &= p + b \\ &> p, \end{aligned}$$

und für $l > k$:

$$\begin{aligned}
& (i + jb + kb^2 + b^3) + (d - lb^2 - b^3 + e) + (2 \cdot \text{„}b_1 \text{ oder } b_2\text{“}) \\
&= (d + e) + (i + jb + (k - l)b^2) + (2 \cdot \text{„}b_1 \text{ oder } b_2\text{“}) \\
&< (3d + e) + (i + jb + (k - l)b^2) \\
&\leq p + (q + qb - b^2) \\
&= p + q + (2q^2 + q) - (4q^2 + 4q + 1) \\
&= p + (-2q^2 - 2q - 1) \\
&< p
\end{aligned}$$

Es bleibt also nur $k = l$ (und somit das blaue Gewicht $b_3(k)$).

Nehmen wir nun an, die beiden anderen blauen Gewichte seien beides b_1 -Gewichte, also $b_1(x)$ und $b_1(y)$. Dies ergäbe eine Summe von

$$(3d + e) + (i + jb) - (x + y) \geq p + (1 + b) - 2q = p + 2 > p.$$

Zwei b_1 -Gewichte sind also nicht möglich.

Zwei b_2 -Gewichte sind ebensowenig möglich, denn wenn eine Menge W_i zwei b_2 -Gewichte enthalten würde, blieben für mindestens eine andere Menge W_j ($i \neq j$) nur noch zwei b_1 -Gewichte übrig.

Jede Menge muß demnach jeweils genau ein b_1 -, b_2 - und b_3 -Gewicht enthalten. Diese seien $b_1(x)$, $b_2(y)$ und $b_3(k)$ (letzteres liegt ja bereits fest). Damit erhalten wir die Summe

$$(3d + e) + (i - x) + (j - y)b.$$

Da $|i - x| < q < b$, muß $j = y$ und folglich auch $i = x$ sein. Das rote Gewicht legt die drei blauen Gewichte also eindeutig fest.

Insgesamt gilt damit: Wenn es eine Partitionierung der Gewichte gibt, dann ist durch die blauen Gewichte (bzw. durch die Koeffizienten der roten Gewichte) in den Mengen W_1, \dots, W_q ein 3D-Matching definiert.

■

5.2 Graph-Faktorisierung ist NP-vollständig

Definition 5.4 (faktorisiert) Sei $G = (V, E)$ ein zusammenhängender Graph. Der Graph heißt **faktorisiert**, wenn sich die Knotenmenge V in $1 < m < |V|$ Teilmengen V_1, V_2, \dots, V_m partitionieren läßt, sodaß für alle $1 \leq i \leq m$ gilt: $|V_i| = |V|/m$ und der induzierte Teilgraph $G_i = (V_i, E|_{V_i})$ ist zusammenhängend.

Dieses Problem wird als GRAPH-FACT (Graph-Faktorisierung) bezeichnet. □

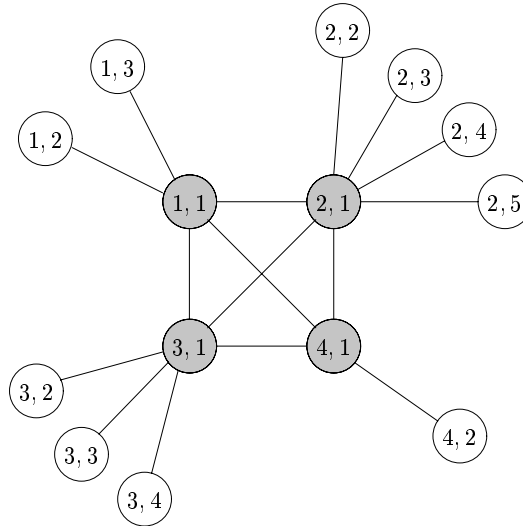
Satz 5.5 Das Problem GRAPH-FACT ist NP-vollständig.

Beweis: Wir reduzieren MPART auf GRAPH-FACT. Sei also eine Menge von Gewichten $W = \{w_1, \dots, w_n\}$, $w_i \in \mathbb{N}$ in unärer Codierung gegeben. Wir konstruieren daraus einen Graph $G = (V, E)$ mit

- $V := \bigcup_{1 \leq i \leq n} \langle i, 1 \rangle \cup \dots \cup \langle i, w_i \rangle$
- $E := \{ (\langle i, 1 \rangle, \langle i, j \rangle) \mid 1 \leq i \leq n, 2 \leq j \leq w_i \}$ (Sterne mit Zentrum i_1)
 $\cup \{ (\langle i, 1 \rangle, \langle j, 1 \rangle) \mid 1 \leq i, j \leq n, i \neq j \}$ (vollst. Graph der Zentren)

Jedes Gewicht w_i wird also in einen Stern mit Zentrum $\langle i, 1 \rangle$ und $w_i - 1$ Strahlen überführt, und alle Sternzentren werden zu einem vollständigen Graphen verbunden.

Beispiel: Sei $W = \{3, 5, 4, 2\}$. Diese Gewichte werden in folgenden Graphen überführt (die grauen Knoten sind die Sternzentren):



■

Die Behauptung ist nun, daß eine Partitionierung der Menge W genau dann existiert, wenn G faktorierbar ist.

MPART \Rightarrow GRAPH-FACT

Sei W_1, \dots, W_m eine Partitionierung von W in m Teilmengen mit gleicher Gewichtssumme. Dies entspricht folgender Faktorisierung des Graphen: Jeder Teilgraph G_i besteht genau aus den Sternen, die den Gewichten in W_i entsprechen. Sterne sind in sich zusammenhängend, und jeder Stern ist mit jedem anderen über sein Zentrum verbunden. Somit ist jeder der entstehenden Teilgraphen zusammenhängend, und alle G_i haben die gleiche Größe (entsprechend der Gewichtssumme von W_i).

GRAPH-FACT \Rightarrow MPART

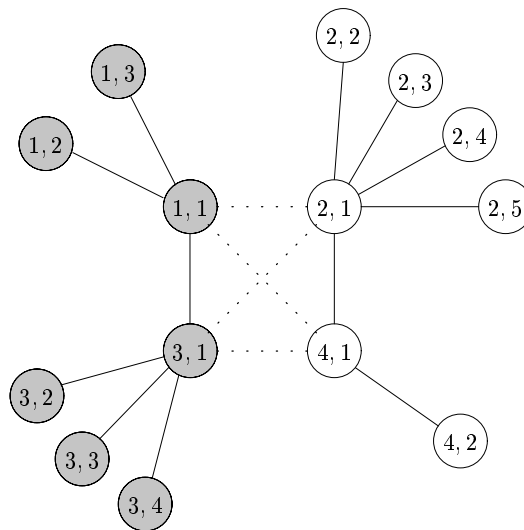
Sei nun G_1, \dots, G_m eine Faktorisierung von G in m zusammenhängende Teilgraphen mit gleicher Knotenanzahl. Jetzt wird behauptet, daß jeder Stern komplett in einem der G_i liegen muß:

- Sei $v \in V_i$ und $v = s_1$ das Zentrum eines Sterns. Dann müssen aber auch alle Randknoten s_2, \dots, s_{w_s} des Sterns in W_i sein, da sie nur zu s_1 eine Verbindung haben und jeder Teilgraph G_i mindestens 2 Knoten besitzt.
- Sei $v \in V_i$ und $v = s_k$ ($2 \leq k \leq w_s$) ein Randknoten eines Sterns. Da jeder Teilgraph G_i mindestens 2 Knoten besitzt, muß das Zentrum s_1 dieses Sterns ebenfalls zu V_i gehören. Nach obiger Argumentation folgt dann aber, daß auch alle anderen Randknoten $s_2, \dots, s_{k-1}, s_{k+1}, \dots, s_{w_s}$ zu V_i gehören müssen.

Jeder Teilgraph G_i besteht also ausschließlich aus kompletten Sternen. Doch dies läßt sich direkt in die den Sternen entsprechenden Gewichte übertragen, um eine Partitionierung von W zu erhalten.

■

Beispiel: Führen wir obiges Beispiel fort. Die Gewichte lassen sich in zwei Teilmengen $W_1 = \{3, 4\}$ und $W_2 = \{5, 2\}$ unterteilen. Dies entspricht folgender Partitionierung (W_1 graue Knoten, W_2 weiße Knoten, durchgezogene Linien sind Kanten der induzierten Graphen):



■

Von Mike Robson (Universität Bordeaux) existiert ein direkter Beweis für die NP-Vollständigkeit der Graph-Faktorisierung, der eine leicht modifizierte Version von SAT auf FACTORIZATION reduziert [Robson, 1998]. Er kommt zu einem noch stärkeren Ergebnis: die Graph-Faktorisierung bleibt auch für planare Graphen mit maximalem Knotengrad 3 NP-vollständig.

Literaturverzeichnis

- [Angluin, 1980] Dana Angluin. Local and Global Properties in Networks of Processors. In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing*, pages 82–93, Los Angeles, California.
- [Attiya and Snir, 1991] Hagit Attiya and Marc Snir. Better Computing on the Anonymous Ring. *Journal of Algorithms*, 12:204–238.
- [Attiya et al., 1988] Hagit Attiya, Marc Snir, and Manfred K. Warmuth. Computing on an Anonymous Ring. *Journal of the ACM*, 35(4):845–875.
- [Burns, 1980] James E. Burns. A Formal Model for Message Passing Systems. Technical Report 91, Computer Science Department, Indiana University, Bloomington.
- [Chang and Roberts, 1979] Ernest Chang and Rosemary Roberts. An Improved Algorithm for Decentralized Extrema-finding in Circular Configurations of Processes. *Communications of the ACM*, 22(5):281–283.
- [Dolev et al., 1982] Danny Dolev, Maria Klawe, and Michael Rodeh. An $\mathcal{O}(n \log n)$ Unidirectional Distributed Algorithm for Extrema Finding in a Circle. *Journal of Algorithms*, 3:245–260.
- [Garcia-Molina, 1982] Hector Garcia-Molina. Elections in a Distributed Computing System. *IEEE Transactions on Computers*, C-31(1):48–59.
- [Higham and Przytycka, 1993] Lisa Higham and Teresa Przytycka. A Simple, Efficient Algorithm for Maximum Finding on Rings. In Schiper, A., editor, *7th International Workshop on Distributed Algorithms*, volume 725 of *Lecture Notes in Computer Science*, pages 249–263.
- [Hirschberg and Sinclair, 1980] Daniel S. Hirschberg and James B. Sinclair. Decentralized Extrema-Finding in Circular Configurations of Processors. *Communications of the ACM*, 23(11):627–628.
- [Itai and Rodeh, 1981] Alon Itai and Michael Rodeh. Symmetry Breaking in Distributed Networks. In *Proceedings of the 22nd Annual Symposium on Foundations of Computer Science*, pages 150–158, Nashville, Tennessee.
- [Knuth, 1997] Donald E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming*. Addison Wesley Longman, 3rd edition.
- [Lamport and Lynch, 1990] Leslie B. Lamport and Nancy A. Lynch. Distributed Computing: Models and Methods. In van Leeuwen, J., editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 18, pages 1155–1199. Elsevier Science Publishers B.V.

- [Le Lann, 1977] Gérard Le Lann. Distributed Systems—Towards a Formal Approach. In Gilchrist, B., editor, *Information Processing (IFIP)*, pages 155–160. North-Holland, Amsterdam.
- [Mazurkiewicz, 1988] Antoni Mazurkiewicz. Solvability of the Asynchronous Ranking Problem. *Information Processing Letters*, 28:221–224.
- [Métivier et al., 1996] Yves Métivier, Nasser Saheb, and Pierre-André Wacrenier. A Distributed Algorithm for Computing a Spanning Tree. Technical Report 1152–96, Laboratoire Bordelais de Recherche en Informatique (LaBRI), Université de Bordeaux.
- [Pachl et al., 1984] Jan K. Pachl, Ephraim Korach, and Doron Rotem. Lower Bounds for Distributed Maximum-Finding Algorithms. *Journal of the ACM*, 31:905–918.
- [Peterson, 1982] Gary L. Peterson. An $\mathcal{O}(n \log n)$ Unidirectional Algorithm for the Circular Extrema Problem. *ACM Transactions on Programming Languages and Systems*, 4(4):758–762.
- [Robson, 1998] Mike Robson. The problem of deciding whether a graph has a nontrivial factorization is NP-complete. Unveröffentlichte Arbeit.
- [Tel, 1994] Gerard Tel. *Introduction to Distributed Algorithms*. Cambridge University Press.
- [Yamashita and Kameda, 1996] Masafumi Yamashita and Tsunehiko Kameda. Computing on Anonymous Networks: Part I—Characterizing the Solvable Cases. *IEEE Transactions on Parallel and Distributed Systems*, 7(1):69–89.