10 Anführerauswahl (Leader Election)

Das Problem, um das es in diesem Kapitel geht, findet sich in der Literatur unter verschiedenen Namen (siehe Abschnitt 10.1). Am treffendsten ist wohl der der *Anführerauswahl* bzw. *Leader Election* für d-dimensionale Muster. Im Unterschied zum FSSP ist für dieses Problem immer noch unklar, was im allgemeinen die Optimalzeit ist. Die beste bekannte untere Schranke ist $\Theta(\mathit{diam})$, wobei *diam* der Durchmesser des Musters ist, die beste bekannte obere Schranke $\Theta(\mathit{diam} \cdot \log(\mathit{diam}))$ und stammt aus der Arbeit von Stratmann und Worsch (2002). Nachfolgend findet sich im wesentlichen eine Übersetzung dieser Arbeit, der wie folgt aufgebaut ist: In Abschnitt 10.1 sind die grundlegenden Begriffe zusammengefasst. Abschnitt 10.2 gibt einen Überblick über den Algorithmus. Die technischen Details sind in Abschnitt 10.3 beschrieben und die Laufzeit des Algorithmus wird in Abschnitt 10.4 abgeschätzt.

10.1 Grundlagen für das Problem der Anführerauswahl

10.1 Definition Es sei N eine Nachbarschaft mit $-N = N$, d. h. für alle $\mathbf{n} \in N$ ist auch $-\mathbf{n} \in N$.

Im folgenden schreiben wir Koordinaten von Zellen in der Form $\mathbf{x} = (x_1, \ldots, x_d)$ etc.

Wir sagen, zwei Zellen $\mathbf{x}, \mathbf{y} \in T$ seien N-*benachbart*, in Zeichen $\mathbf{x} \sim_N \mathbf{y}$, falls $\mathbf{x} - \mathbf{y} \in N$ ist. Eine Folge $(\mathbf{x}^0, \mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^{k-1}, \mathbf{x}^k)$ heißt ein N-*Pfad*, falls für alle $0 \leqslant i < k$ gilt: $\mathbf{x}^i \sim_N \mathbf{x}^{i+1}$. Ein N-Pfad *verläuft in* $T \subset R$, falls für alle $0 \leqslant i \leqslant k$ gilt: $\mathbf{x}^i \in T$.

Ein Muster $m : T \to Q'$ heißt N-*zusammenhängend*, wenn es für alle $\mathbf{x}, \mathbf{y} \in T$ einen N-Pfad von $\mathbf{x}$ nach $\mathbf{y}$ gibt, der nur in T verläuft. ◇

10.2 Problem. (M-Konzentration, Anführerauswahl, Queen Bee, Leader Election) Es seien drei Zustände „schwarz“ ■, „weiß“ □ und „rot“ ■ gegeben, und es sei M eine Menge „weißer Muster“, also Muster der Form $m : T \to \{\square\}$. Gesucht ist ein Zellularautomat mit Zustandsmenge $Q \supseteq \{\blacksquare, \square, \blacksquare\}$ und ■ als totem Ruhezustand, der für jedes $H_1^{(d)}$-zusammenhängende Muster $m \in M$ mit Träger T die zugehörige Musterkonfiguration c_m überführt in die Musterkonfiguration eines Musters $\hat{m}$ mit gleichem Träger und $|\hat{m}^{-1}(\blacksquare)| = 1$, in der keine Zelle mehr ihren Zustand ändert.

In den späteren Abschnitten werden auch noch die folgenden Begriffe verwendet:

10.3 Den Träger eines Musters (siehe Kapitel 3) bzw. einer Musterkonfiguration c bezeichnen wir mit $\mathit{supp}(c)$.

10.4 Definition Der *Durchmesser diam* eines Musters ist das Maximum der Längen kürzester Pfade von $\mathbf{x}$ nach $\mathbf{y}$ innerhalb des Musters, wobei über alle Paare von Zellen $(\mathbf{x}, \mathbf{y}) \in T^2$ maximiert wird. ◇

Zur Geschichte des Problems

Dimension 1. Für eindimensionale ZA ist das Problem trivial in konstanter Zeit lösbar: Das Muster ist immer einfach eine Folge weißer Zellen, von denen genau eine, nämlich die am linken Ende einen schwarzen linken Nachbarn hat. Sie wird in ersten Schritt rot.

Dimension 2. Für zweidimensionale ZA wurde das Problem zuerst von Smith III (1971) unter dem Namen *Queen Bee Problem (QBP)* untersucht. Er beschrieb einen ZA der das Problem in Zeit $\Theta(peri)$ löst, wobei *peri* die Länge des außeren Randes des Musters ist. Die Beschreibung des Algorithmus ist allerdings sehr skizzenhaft. Vollmar (1979) betrachtete deshalb eine vereinfachte Variante des QBP und beschrieb eine Lösung die ebenfalls Zeit $\Theta(peri)$ benötigt.

Etliche Jahre später haben Beckers und Worsch (1998) und Beckers und Worsch (2001) und Mazoyer, Nichitiu und Rémila (1999) unabhängig voneinander ZA beschrieben, die ebenfalls auf dem Rand des Musters arbeiten, aber verschiedene Techniken für den Vergleich von Positionen bestimmter Zellen verwenden. Auch diese Algorithmen benötigen Zeit $\Theta(peri)$.

Dimension $d \geqslant 3$. Algorithmen, die nicht nur in 2 sondern auch in $d \geqslant 3$ Dimensionen funktionieren, sind schwieriger zu finden. Der naheliegende Trick, die Dimensionalität zu reduzieren, klappt nicht. Zwar hat die Oberfläche Dimensionalität $d-1$, aber z. B. die 2-dimensionale Oberfläche eines 3-dimensionalen Musters hat keinen Rand, so dass die obigen Algorithmen nicht anwendbar sind.

Der erste ZA für das allgemeine Problem in d Dimensionen geht auf Hemmerling (1979) zurück. Unter Benutzung eines Algorithmus für die Suche in endlichen Labyrinthen gibt es eine Lösung des von ihm so genannten *Konzentrationsproblems* an, die allerdings Laufzeit $\Theta(vol^5)$ hat, wobei *vol* die Gesamtzahl der zu Muster gehörenden Zellen ist. Für $d = 2$ ist dieser ZA also deutlich langsamer als die oben erwähnten.

Mit Hilfe einer Verallgemeinerung von Mazoyers Technik konnten Nichitiu und Rémila (1999) einen Algorithmus für d-dimensionale Muster angeben, der Laufzeit $diam^2$ hat.

Stratmann (2000) gab in seiner Studienarbeit einen Algorithmus an, der nur noch eine Anzahl Schritte in $O(diam \cdot (\log diam)^{d+1})$ benötigte, was schon relativ nahe an der einzigen bekannte, naheliegenden unteren Schranke von *diam* liegt.

Im Folgenden wird eine verbesserte Version dieses Algorithmus beschrieben, die nur noch $diam \cdot \log diam$ Schritte benötigt. Sie stammt aus der Arbeit von Stratmann und Worsch (2002), die hier in wesentlichen Teilen wiedergegeben wird.

10.2 Idee des Algorithmus

In diesem Abschnitt beschreiben wir die wesentlichen Bestandteile des Algorithmus. Unterabschnitt

10.2.1 führt noch weitere Begriffe ein.

10.2.2 beschreibt die Graphstruktur, auf der der Algorithmus arbeitet.

10.2.3 beschreibt die Zähler, die ausgiebig benutzt werden.

10.2.4 erläutert, wie Kollisionen von Zählern gehandhabt werden.

10.2.5 beschreibt, wie Zyklen aus dem Graphen entfernt werden.

10.2.6 fasst den Algorithmus zusammen.

10.2.1 Einige Begriffe

10.5 DEFINITION Für Zellen $\mathbf{x} = (x_1, \ldots, x_d)$ und $\mathbf{y} = (y_1, \ldots, y_d)$ definieren wir die Relation $\sqsubset$ und sagen, dass $(x_1, \ldots, x_d) \sqsubset (y_1, \ldots, y_d)$ gelte, falls es in j gibt, so dass $x_j < y_j$ gilt und $x_i = y_i$ für alle $i < j$. Wir schreiben $\mathbf{x} \sqsubseteq \mathbf{y}$, falls $\mathbf{x} \sqsubset \mathbf{y}$ oder $\mathbf{x} = \mathbf{y}$. ◇

Offensichtlich ist $\sqsubseteq$ eine totale Ordnung.

Die grundsätzliche Idee vieler ZA für das Leader Election Problem besteht darin, eine leicht (in konstanter Zeit) überprüfbare Eigenschaft zu benutzen, um *Kandidatenzellen* auszuzeichnen. Diese Zellen werden dann bezüglich der Ordnung $\sqsubset$ verglichen und die Kandidateneigenschaft der jeweils kleineren Zell gelöscht. Am Ende bleibt genau ein Kandidat übrig. Der Algorithmus entdeckt, wann das der Fall ist, und „ernennt" diese Zelle zum Anführer, d. h. sie geht in den Zustand ■ über.

10.6 DEFINITION Eine Zelle $\mathbf{x} \in \mathit{supp}(c)$ ist ein(e) *Kandidat(enzelle)* wenn ihr *Kandidatenbit* (Teil des Zustandes) gesetzt ist. Zu Beginn setzt eine Zelle ihr Kandidatenbit genau dann, wenn in ihrer M_1-Nachbarschaft keine andere Zelle $\mathbf{y} \in \mathit{supp}(c)$ ist, für die $\mathbf{x} \sqsubset \mathbf{y}$ gilt. (Das kann jede Zelle lokal entscheiden!) Die *beste* Zelle ist der Kandidat, der größer (bezüglich $\sqsubset$) ist als alle anderen Kandidatenzellen. ◇

Im Laufe des Algorithmus für die Anführerauswahl wird jede Kandidatenzelle außer der besten ihr Kandidatenbit zu irgendeinem Zeitpunkt zurücksetzen. Um das zu ermöglichen, werden die Positionen der Kandidatenzellen miteinander verglichen. Dafür startet jede von ihnen eine „Welle", die versucht, das gesamte Muster zu fluten. Wenn zwei Wellen kollidieren, werden Vergleiche durchgeführt. Sind die Wellen bei verschiedenen Kandidaten gestartet, dann wird die, die vom kleineren Kandidaten kommt, gelöscht und die andere bewegt sich weiter. Wenn eine Welle auf eine Kandidatenzelle trifft, wird deren Kandidatenbit zurückgesetzt.

Um herauszufinden, wann nur noch die beste Kandidatenzelle übrig ist, wird das Muster geschrumpft. Zellen an dessen Grenze, die keine Kandidaten sind, werden aus dem Muster entfernt. Sobald eine Kandidatenzelle in ihrer Nachbarschaft ansonsten nur noch schwarze und eliminierte Zelle hat, weiß sie, dass sie die beste ist.

Dies ist eine grobe Skizze des Algorithmus. Besser kann man ihn an Hand eines Graphen beschreiben, der im folgenden eingeführt wird.

10.2.2 Die Graphenstruktur des Musters

Der Algorithmus zur Anführerauswahl wird auf einem ungerichteten Graphen arbeiten, der nach und nach modifiziert wird.

10.7 ALGORITHMUS. (INITIALISIERUNG DES GRAPHEN) Der Graph G wird aufgrund des Eingabemusters auf naheliegende Art initialisiert:

I1. Jeder □-Zelle entspricht ein Knoten von G.

I2. Zwischen zwei Knoten gibt es genau dann eine Kante, wenn die korrespondierenden Zellen H_1-Nachbarn sind.

Die Struktur des Graphen wird im Zellularautomaten dadurch repräsentiert, dass die Tatsache, dass ein Knoten des Graphen vorliegt, und zu welchen Nachbarn Kanten führen in einer Zelle gespeichert werden.

10.8 Zu Beginn ist der Graph zusammenhängend, da man mit einem H_1-zusammenhängenden Muster beginnt.

Im Laufe der Berechnung wird der Graph modifiziert. Die algorithmische Idee besteht darin, durch Entfernung von Kanten zu einem aufspannenden Baum zu kommen. Dessen Blätter werden dann nach und nach entfernt, bis nur noch die Wurzel übrig bleibt. Sie wird dann zum Anführer „ernannt“. Es wird sichergestellt werden, dass der Graph immer zusammenhängend bleibt.

Bevor wir das erläutern können, muss ein Konzept für Zähler eingeführt werden. Dabei bauen wir auf den in Kapitel 6 vorgestellten Ideen auf.

10.2.3 Zähler

In diesem und dem anschließenden Unterabschnitt wird die Benutzung von Zählern auf einem nicht allzu niedrigen Niveau beschrieben. Weitere (Implementierungs-)Details finden sich in Abschnitt 10.3.

10.9 Die grundlegende Idee von Vollmar (1977) eines sich bewegenden Zählers haben wir schon in Kapitel 6 kennengelernt. Zur Erinnerung zeigt Abbildung 10.1 noch einmal ein Beispiel. Für den Algorithmus zur Anführerauswahl wird das Zähler-Konzept verallgemeinert und erweitert:

- Die Zähler bewegen sich nicht mit konstanter Geschwindigkeit 1 sondern machen nur ab und zu einen Schritt von einer Zelle zur nächsten.
- Die Zähler werden erweitert, so dass auch negative ganze Zahlen gespeichert werden können. Ein *Raumzähler* für eine Dimension i wird um 1 erhöht, wenn er sich in einer Richtung um eine Zelle weiter bewegt, und er wird um 1 erniedrigt, wenn er sich in die entgegengesetzte Richtung bewegt.
- Für jede Dimension gibt es einen Raumzähler. Wenn sich die Zähler zusammen von einer Zelle zur nächsten bewegen, wird also der für die entsprechende Dimension erhöht oder erniedrigt, während sich alle anderen nicht ändern.
- Ein *Schrittzähler* ist ein Zähler, der in *jedem Schritt erhöht* wird, unabhängig davon, in welche Richtung er sich bewegt. Die Länge eines Schrittzählers ist daher eine obere Schranke für die Länge jedes Raumzählers, der sich entlang des gleichen Pfades bewegt hat.
- Ein *voller Zähler* besteht aus einem Schrittzähler und d Raumzählern, je einem für jede Dimension. Sie bewegen sich synchron, wobei die niedrigstwertigen Bits voranlaufen und sich immer alle in der gleichen Zelle befinden. Jede Zelle benötigt daher $2(d+1)$ Register für die Summen- und die Übertragsbits. Da die Längen der Zähler verschieden sein können, kann es passieren, dass in einer Zelle die Summen- und die Übertragsbits einiger Zähler besetzt sind (mit 0 oder 1) und die anderer Zähler leer. Letztere werden als 0 interpretiert, falls nötig.

1	2	3	4	5	6	7	8	9	10
(1)									
0									
	0)								
	(1								
	(1	1)							
	0	0							
		(1	0)						
		0	1						
			0	1)					
			(1	0					
			(1	0	0)				
			0	0	1				
				(1	1	1)			
				0	0	0			
					(1	1	0)		
					0	0	1		
						(1	0	1)	
						0	1	0	
							0	0	0)
							(1	0	1

Abbildung 10.1: Raum-Zeit-Diagramm eines ZA, in dem ein Zähler nach rechts läuft.

10.10 Im Algorithmus für die Anführerauswahl wird jede Kandidatenzelle des Ausgangsmusters einen vollen Zähler starten, dessen Zähler alle mit 0 initialisiert sind, und der entlang aller wegführenden Kanten losläuft.

Die Zähler bewegen sich entlang der Kanten des Graphen. Wann immer sich das niedrigstwertige Bit eines Zählers weiterbewegt, tut es das entlang aller Kanten, die mit dem aktuellen Knoten inzidieren, ausgenommen die Kante, über die die Bits zu dem Knoten kamen. Ist der Knotengrad größer oder gleich 3, so muss sich der volle Zähler in zwei oder mehr (maximal $2d - 1$) aufspalten, die den Kanten gleichzeitig folgen. Eine gewisse Zeit lang werden korrespondierende Zähler noch ein (schrumpfendes) gemeinsames Ende haben, während die Anfänge bereits getrennte Wege gegangen sind.

10.11 Es wird darauf geachtet werden, dass hieraus keine Probleme entstehen, indem dafür gesorgt wird, dass sich immer alle Zähler, die irgendwo im Graphen unterwegs sind, synchron weiterbewegen. Daher werden auch alle Aktionen wie z. B. das Auslesen eines Zählerinhaltes (siehe nächster Unterabschnitt) synchron starten, und es wird auch darauf geachtet werden, dass sie überall gleich lange dauern.

10.12 Definition Zwei Zähler heißen genau dann *verwandt*, wenn sie in der gleichen Kandidatenzelle gestartet sind. ◇

Wenn ein Zähler in einer Sackgasse des Graphen endet, werden seine Bits der Reihe nach „absorbiert“.

10.13 Die niedrigstwertigen Bits verwandter Zähler kann man als Wellenfront auffassen, die in der zugehörigen Kandidatenzelle gestartet ist. Dies ist eine schwache Analogie zu dem Algorithmus von Nichitiu und Rémila 1999. Diese Autoren benutzen aber keine binäre, sondern eine Art unäre Zahldarstellung, weshalb z. B. der Vergleich zweier Werte viel länger dauert als bei unserem Algorithmus.

Da es im Allgemeinen mehrere Kandidatenzellen gibt und der Graph immer zusammenhängend bleibt, werden sich über kurz oder lang sowohl verwandte als auch nicht verwandte volle Zähler treffen.

10.2.4 Kollisionen von Zählern

10.14 Zähler können *kollidieren*. Es gibt zwei Arten von Kollisionen. Die niedrigstwertigen Bits von Zählern

C1. können *eine Kollisionszelle* (aus verschiedenen Richtungen) zur gleichen Zeit betreten, oder

C2. sie können zu einem Zeitpunkt *zwei* H_1*-benachbarte Kollisionszellen* betreten.

Es kann passieren, dass zwei kollidierende Zähler verwandt sind. In einem solchen Fall hat man einen Zyklus im Graphen entdeckt.

Eine Kollisionszelle kann leicht feststellen, von welchem Typ die Kollision ist. Wir betrachten zunächst den Fall C1, und reduzieren anschließend Fall C2 darauf.

Angenommen, zwei volle Zähler, die in Kandidatenzellen $\mathbf{x}^j$ ($j \in \{1,2\}$) gestartet sind, kollidieren in Zelle $\mathbf{y}$ irgendwo im Muster. Die beiden Mengen von d Raumzählern kann mal als d-dimensionale „Offsets“ $\mathbf{z}^j$ interpretieren, so dass $\mathbf{y} = \mathbf{x}^j + \mathbf{z}^j$, also $\mathbf{y} - \mathbf{z}^j = \mathbf{x}^j$. Folglich gilt $\mathbf{x}^1 \sqsubseteq \mathbf{x}^2$ genau dann, wenn $\mathbf{z}^2 \sqsubseteq \mathbf{z}^1$. Letztere Bedingung kann leicht überprüft werden.

10.15 Algorithmus. (Zählervergleich)

for all $i \in \{1,\ldots,d\}$ **do**
  $r_i \leftarrow \operatorname{sign}(z_i^1 - z_i^2)$
od
$m \leftarrow \min(\{d+1\} \cup \{i \mid r_i \neq 0\})$

Ist $m = d+1$, dann sind alle $r_i = 0$, d. h. $\mathbf{x}^1 = \mathbf{x}^2$. Andernfalls ist m das kleinste i, so dass $r_i \neq 0$: Ist dann $r_m = 1$ (resp. $r_m = -1$), so ist $\mathbf{x}^1 \sqsubset \mathbf{x}^2$ (resp. $\mathbf{x}^2 \sqsubset \mathbf{x}^1$).

Man beachte, dass hier noch nichts darüber gesagt wurde, wie die einzelnen vorzeichenbehafteten (!) Raumzähler z_i^j repräsentiert werden. Die genaue Art der Bestimmung von $\operatorname{sign}(z_i^1 - z_i^2)$ kann daher erst später beschrieben werden.

Eine detaillierte Beschreibung dieses Algorithmus findet man in Unterabschnitt 10.3.5. Die Tatsache, dass die Länge des Schrittzählers (der überall den gleichen Wert hat!) eine obere Schranke für die Länge der Raumzähler ist, kann man ausnutzen um sicherzustellen, dass der Zeitbedarf für den Zählervergleich stets ein konstantes Vielfaches der Länge der Schrittzähler ist.

Man kann den zweiten Typ von Kollisionen (C2) auf den ersten zurückführen. Um Asymmetrien zu vermeiden stellen wir sogar sicher, dass in *beiden* Kollisionszellen die Zähler aus beiden Richtungen verglichen werden. Dazu habe jede Zelle zusätzlich sogenannte *td-Register*

(von „transfer/delay“), um die Summenbits jedes Raum- und des Schrittzählers für einen Takt zu speichern.

Im Fall C2 führt jede Kollisionszelle in jedem Schritt die folgenden Aktionen durch:

- Die Bits der „eigenen“ Zähler werden in den td-Registern gespeichert und
- die vorangegangenen Inhalte der td-Register werden mit den Inhalten der td-Register der benachbarten Kollisionszelle verglichen.

Es kann passieren, dass in manchen Zellen Kollisionen vom Typ C1 geschehen und in anderen solche vom Typ C2. Im zweiten Fall dauern die Vergleiche einen Schritt länger. Um trotzdem volle Synchronität zu gewährleisten, werden C1-Kollisionen künstlich um einen Schritt verzögert.

In einem d-dimensionalen ZA kann es passieren, dass in einer Zelle bis zu 2d Wellen kollidieren.

10.16 Algorithmus. (Auflösung von Kollisionskonflikten) Diese Konflikte werden wie folgt aufgelöst:

- Eine beliebige Folge aller möglichen Paare $(\mathbf{a}, \mathbf{b})$ von Richtungen wird für den Algorithmus fixiert.
- Nachdem sich alle Zähler eine Zelle weiterbewegt haben, wird *immer für alle* Paare $(\mathbf{a}, \mathbf{b})$ etwas getan:
  - Falls aus den beiden Richtungen Zähler ankommen und kollidieren, werden die notwendigen Aktionen wie oben beschrieben durchgeführt.
  - Falls aus den beiden Richtungen keine Zähler ankommen, wird die gleiche Anzahl Schritte, die für die Vergleiche benötigt worden wäre, damit verbracht nichts zu tun. Da diese Zeit als Vielfaches der Länge des Schrittzählers gewählt werden kenn, kann sie leicht gemessen werden, indem man ein Signal entlang des Schrittzählers hin und her schickt.

Da die Zahl der möglichen Paare von Richtungen konstant ist (sie hängt nur von d ab), verlangsamt dieses Vorgehen den Algorithmus nur um einen konstanten Faktor. Auf ähnliche Weise kann und muss man den Fall behandeln, dass eine Mischung von C1- und C2-Kollisionen vorliegt.

Als nächstes beschreiben wir, wie Kollisionen von Zählern können benutzt werden, um im Graphen Zyklen zu finden.

10.2.5 Transformation des Graphen

Wenn zwei verwandte Zähler kollidieren, hat man einen Zyklus im Graphen entdeckt.

10.17 Algorithmus. (Aufbrechen von Zyklen) Ist eine Kollision verwandter Zähler vom Typ C2, dann wird die Kante zwischen den Kollisionszellen markiert und damit für Löschung vorgemerkt. Ist die Kollision vom Typ C1, wird eine der Kanten, die mit der Kollisionszelle inzidiert und entlang der einer der Zähler ankam, markiert und damit für Löschung vorgemerkt.

Nachdem in Algorithmus 10.16 alle Paare $(\mathbf{a}, \mathbf{b})$ bearbeitet worden sind, werden alle markierten Kanten tatsächlich gelöscht.

Damit kann der Prozess der Graphtransformation nun vollständig beschrieben werden.

10.18 Algorithmus. (Löschungen im Graphen) Knoten und Kanten werden wie folgt gelöscht:

D1. Ein Knoten wird gelöscht, sobald die letzte mit ihm inzidierende Kante gelöscht wurde.

D2. Eine Kante wird gelöscht, wenn

E1. sie die letzte Kante ist, die mit einem Knoten inzidiert, und der Knoten kein Kandidat ist, oder

E2. wenn Algorithmus 10.17 sie entfernt, um einen Zyklus aufzubrechen.

Man beachte, dass nicht alle Kanten, die zu einem Zyklus gehören, als solche entdeckt und gelöscht werden.

10.19 Algorithmus. (Erweiterungen des Graphen) Es gibt auch den Fall, dass ein Graph wieder erweitert wird. Das wird dann gemacht, wenn eine Zelle feststellt, dass die niedrigstwertigen Bits eines vollen Zählers in einer Nachbarzelle angekommen sind. In diesem Fall werden alle gelöschten Kanten (und Knoten) dem Graphen wieder hinzugefügt.

Dieser Algorithmus stellt sicher, dass jeder volle Zähler, in jeder Zelle, in der er ankommt, den vollen Graphen sieht. Offensichtlich können beide Graphtransformationen in einem ZA implementiert werden.

Es ist wichtig, dass während der Löschungen der Zusammenhang des Graphen nicht zerstört wird. Andernfalls könnte es passieren, dass der Algorithmus zur Anführerauswahl in jeder Zusammenhangskomponente einen Anführer auswählt, d. h. in dem Eingabemuster würden *mehrere* Anführer ausgewählt, was die Problemspezifikation verletzten würde.

Zähler, die in der besten Zelle starten, heißen im folgenden auch *beste Zähler*.

10.20 Lemma. Die besten vollen Zähler bauen einen Baum mit folgenden Eigenschaften auf:

- Er spannt den gesamten ursprünglichen Graphen auf.
- Jeder Pfad in dem Baum von Wurzel **r** zu irgend einem Knoten **x** ist ein kürzester Pfad im ursprünglichen Graphen von **r** nach **x**.

10.21 Beweis Wenn ein bester voller Zähler eine Zelle erreicht, sieht er dort immer den gesamten ursprünglichen Graphen.

- Man betrachte einen beliebigen Zyklus im ursprünglichen Graphen. Es muss gezeigt werden, dass er aufgebrochen wird. Mindestens einer besten vollen Zähler wird zu irgendeinem Zeitpunkt einen Knoten des Zyklus erreichen (unter Umständen auch mehrere (Zähler und Zellen)). Von dort wird der oder werden die Zähler sich wo möglich verzweigen entlang aller Kanten. Insbesondere werden auch beste Zähler entlang des Zyklus laufen, sich irgendwo auf dem Zyklus treffen und ihn dort aufbrechen.
- Ein Knoten habe genau dann *Abstand* k, wenn k die Länge eines kürzesten Pfades im Graphen zwischen **r** und **x** ist. Wir werden nun mittels vollständiger Induktion zeigen, dass für jeden Knoten **x** mit Abstand k der Pfad im Baum von **r** nach **x** Länge k hat.

  Für $k = 0$ und $k = 1$ ist das klar.

  Induktionsannahme: Die Behauptung ist für ein k richtig.

Sei nun $\mathbf{x}$ ein Knoten mit Abstand $k+1$. Wir betrachten einen Knoten $\mathbf{y}$ mit Abstand k und einer Kante e zu $\mathbf{x}$. Nach Induktionsannahme hat der Pfad von $\mathbf{r}$ zu $\mathbf{y}$ im Baum die Länge k. Da $\mathbf{x}$ Abstand $k+1$ hat, muss jeder beste volle Zähler $k+1$ Zellen weit laufen, bevor er $\mathbf{x}$ erreicht. Aber $\mathbf{y}$ wird bereits nach k Schritten erreicht. Also kommen die Zähler in $\mathbf{x}$ und $\mathbf{y}$ nicht gleichzeitig an, d. h. sie kollidieren nicht in benachbarten Zellen und folglich wird e nicht wegen einer Kollision vom Typ C2 entfernt. Würde e wegen einer Kollision vom Typ C1 entfernt, dann gäbe es einen anderen kürzesten Pfad nach $\mathbf{x}$ über eine andere Zelle $\mathbf{y}'$ (mit Abstand k) und eine Kante e', die dann ihrerseits nicht entfernt sondern zum Baum hinzugenommen wird. Folglich ist im Baum der Pfad von $\mathbf{r}$ zu $\mathbf{x}$ ein Pfad von $\mathbf{r}$ nach $\mathbf{y}$ bzw. $\mathbf{y}'$ gefolgt von einer weiteren Kante e bzw. e', hat also Länge $k+1$.

■

10.2.6 Grundidee des Algorithmus

Es sind nun alle Voraussetzungen gegeben, um den vollständigen Algorithmus zu skizzieren.

10.22 Algorithmus. (grober Überblick) Parallel werden die beiden folgenden Algorithmen abgearbeitet, bis im aktuellen Muster nur noch eine Zelle übrig ist.

Schrumpfen des Graphen: Sobald mit einem Knoten des Graphen nur noch eine Kante inzidiert, wird er entfernt, sofern er keine Kandidatenzelle ist.

Zähler: In jeder Kandidatenzelle wird ein voller Zähler gestartet:

W1. Er bewegt sich „in alle Richtungen“.

W2. Solange der Zähler existiert, wird er sich nach jeweils einer Anzahl von Zeitschritten proportional zu seiner Länge um eine Zelle weiterbewegen. Also bewegt er sich mit einer Geschwindigkeit echt größer als 0.

W3. Wenn ein Zähler im Graphen das Ende einer Sackgasse erreicht, wird er dort gelöscht.

W4. Wenn zwei Zähler kollidieren, werden die relativen Positionen Ihrer Ursprünge miteinander verglichen. Falls sie verschieden sind, wird der Zähler, der vom „kleineren“ Ursprung kommt, gelöscht. Falls die Ursprünge gleich sind, wurde ein Zyklus entdeckt, der aufgebrochen wird.

W5. Wenn ein Zähler auf eine Kandidatenzelle trifft, wird das Kandidatenbit zurückgesetzt. (Ab dann kann (und wird auch irgendwann) die Zelle am Schrumpfungsprozess teilnehmen.)

Am Ende wird die Zelle, die in ihrer Nachbarschaft nur Zellen beobachtet, die ■ sind oder deren Knoten gelöscht wurde, als Anführer ausgewählt.

Wir diskutieren nun mehrere Behauptungen Algorithmus 10.22 betreffend. Sie sind alle zu verstehen als Aussagen für jedes beliebige endliche zusammenhängende Anfangsmuster.

10.23 Lemma. 1. Beste Zähler löschen sich nie gegenseitig aus.

2. Beste Zähler erreichen jede Kandidatenzelle außer der besten.

3. Alle Kandidatenzellen außer der besten werden irgendwann ihr Kandidatenbit zurücksetzen.

4. Jede Zelle außer der besten wird irgendwann ihren Knoten des Graphen löschen.

10.24 Beweis 1. Wegen Punkt W4. des Zähleralgorithmus.

2. Folgt aus (1) wegen W1. und W2. des Zähleralgorithmus.

3. Folgt aus (2) wegen W5. des Zähleralgorithmus.

4. Folgt aus (3) wegen des Algorithmus zum Graphschrumpfen.

■

10.25 Lemma. Kein Zähler wird jemals die beste Zelle erreichen und folglich wird die beste Zelle niemals ihr Kandidatenbit zurücksetzen und daher nie gelöscht werden.

10.26 Beweis Angenommen, es gäbe ein Anfangsmuster, bei dem die beste Zelle von einem Zähler W besucht wird. Sei P der Pfad, entlang dem der Zähler zur besten Zelle läuft. Dann wird gemäß Punkt W1. ein bester Zähler von der besten Zelle in umgekehrter Richtung entlang P laufen. Folglich werden sich die beiden Zähler treffen. Nach Definition bester Zähler und wegen Punkt W4. wird W nicht überleben. Widerspruch. ■

Aus obigen Lemmata folgt nun sofort:

10.27 Korollar. Algorithmus 10.22 löst das Problem der Anführerauswahl.

10.3 Implementierungsdetails

In diesem Abschnitt werden noch einige Implementierungsdetails für die diversen Zähler vorgestellt. In einigen Zeichnungen werden wir so tun, als hätte man es mit Konfigurationen eines eindimensionalen ZA zu tun. Tatsächlich handelt es sich aber um eindimensionale Pfade, die in einer d-dimensionalen Konfiguration verlaufen. Im wirklichen ZA steht in jeder Zelle des Pfades die Information zur Verfügung, in welcher Richtung sich die nächste befindet.

10.3.1 Repräsentation von Zahlen

Ein Zähler speichert einen Wert, der unbeschränkt wachsen kann. Deshalb werden im allgemeinen mehrere Zellen zur Speicherung benötigt; siehe Abbildung 10.1. In jeder Zelle benutzen wir ein „Summenbit“ b_i und ein „Übertragsbit“ c_i. Wenn wir unspezifisch von einem „Bit“ sprechen, dann meinen wir ein Summenbit. Das zusätzliche Symbol (wird benutzt, um das höchstwertige Bit zu kennzeichnen. Ein solches wird also in der Form (1 oder (0 gespeichert.

Jede ganze Zahl wird als ein Paar aus Vorzeichen und Absolutwert gespeichert. An das niedrigstwertige Bit ist ein Symbol + oder - angehängt, das das Vorzeichen signalisiert und außerdem das niedrigstwertige Bit von allen anderen unterscheidet. Für den Wert 0 benutzen wir ± als „Vorzeichensymbol“. Das Vorzeichensymbol ersetzt das) aus Abbildung 10.1.

Die Zahl -5 würde also in 3 aufeinanderfolgenden Zellen als Folge (101- von Summenbits gespeichert, falls alle Übertragsbits 0 sind. Wir später noch genauer erläutert, wollen wir führende Nullen vermeiden; z. B. wäre (000± eine illegale Repräsentation von 0 (wiederum, falls

Übertragsbits 0 sind). Nur (0± ist erlaubt. Für die Speicherung von Summenbits werden also die elf Symbole

$$\{\,\texttt{(1, (0, 1, 0, 1+, 1-, 0+, 0-, (1+, (1-, (0±}\,\}\,.$$

benutzt.

Sind im allgemeinen Summenbits $b_{k-1}\cdots b_0$ (umgeben von (und Vorzeichensymbol) und Übertragsbits $c_{k-1}\cdots c_0$ gegeben, dann wird dadurch der Absolutwert

$$\sum_{i=0}^{k-1} 2^i(b_i + 2c_i)\,. \tag{10.1}$$

repräsentiert.

Solange alle $c_i \in \{0,1\}$ sind, ist das nichts Ungewohntes. Da wir Zähler aber auch erniedrigen können wollen, werden wir für die Übertragsbits Werte aus dem Bereich $\{-1,0,1\}$ erlauben. Dies könnte im allgemeinen zu Problemen führen. Da Zahlen immer als Vorzeichen und Absolutwert gespeichert werden, muss verhindert werden, dass in unserer Anwendung Formel (10.1) jemals negativ wird. Wie man das erreichen kann, wird als nächstes beschrieben.

10.3.2 Bewegung von Zählern

Wenn wir von der Bewegung von Zählern sprechen, dann meinen wir im folgenden genauer die Bewegung der Summenbits und gehen der Einfachheit halber davon aus, dass die Übertragsbits Null sind (sofern nicht ausdrücklich etwas anderes gesagt wird).

Eine Zelle, die dem niedrigstwertigen Bit b_0 eines Zählers benachbert ist, kann diesen veranlassen, sich eine Zelle weiter zu bewegen. Das niedrigstwertige Bit bewegt sich zuerst, die anderen folgen der Reihe nach. Man kann sich vorstellen, dass der Nachbar ein Signal M initialisiert, das von einem Summenbit zum nächsten weitergegeben wird, indem M und b_{i+1} miteinander vertauscht werden. Das höchstwertige Bit vernichtet das M-Signal. Abbildung 10.2 zeigt ein entsprechendes Raum-Zeit-Diagramm.

	(1	0	1+	M	
	(1	0	M	1+	
	(1	M	0	1+	
		(1	0	1+	

Abbildung 10.2: Raum-Zeit-Diagramm für die Bewegung der Summenbits eines Zählers nach rechts. Die Raumkoordinate verläuft horizontal, die Zeitachse verläuft nach unten. Das M-Signal wird vom höchstwertigen Bit vernichtet.

In Abhängigkeit von der Richtung, in der sich ein Raumzähler bewegt, bleibt sein Wert gleich oder er wird um 1 erhöht oder erniedrigt.

1. Der Wert bleibt der gleiche, wenn sich der Zähler in Dimension j bewegt, aber für das Zählen der Schritte in einer anderen Dimension $i \neq j$ zuständig ist. In diesem Fall initialisiert die Zelle, in der das M-Signal startet, das Übertragsbit mit 0. Das ist der Fall, der in Abbildung 10.2 dargestellt ist.

2. Wenn sich der Zähler in die positive Richtung der Dimension bewegt, für die er zuständig ist, muss er um 1 erhöht werden. Das erreicht man, indem man das Übertragsbit wie folgt initialisiert:
   - Wenn der Zählerwert nichtnegativ ist, wird 1 als Übertragsbit benutzt.
   - Wenn der Zählerwert negativ ist, wird -1 als Übertragsbit benutzt.
3. Muss der Zähler um 1 erniedrigt werden, wird analog zu eben vorgegangen (wobei man „(nicht)negativ“ durch „(nicht)positiv“ ersetzt und die Übertragsbits 1 und -1 vertauscht).

Wie man in den Fällen 2 und 3 mit dem Zähler und den Übertragsbits umgeht, wird als nächstes beschrieben.

10.3.3 Manipulation von Zählern und Übertragsbits

Man kann die bekannten Regeln für Halbaddierer leicht für den Fall $c_i = -1$ erweitern. Wenn eine Zelle ein Übertragsbit c_i gespeichert hat und Summenbit b_{i+1} von seinem Nachbarn erhält, berechnet es neue Werte für Summen- und Übertragsbit wie Abbildung 10.3 dargestellt:

	b_{i+1} \ c_i	-1	0	1
	0	1	0	1
	1	0	1	0

	b_{i+1} \ c_i	-1	0	1
	0	-1	0	0
	1	0	0	1

Abbildung 10.3: Die Regeln für die Aktualisierung von Summenbit (links) und Übertragsbit (rechts) einer Zelle i.

Abbildung 10.4 zeigt ein Beispiel, in dem auch eine illegale Zahlrepräsentation mit einer führenden Null erzeugt wird. In diesem Beispiel wird angenommen, dass in drei unmittelbar aufeinander folgenden Schritten vom Zähler jeweils eine Weiterbewegung verlangt wird. Das wird hier aber nur zu Demonstrationszwecken so gemacht:

10.28 Im Algorithmus zur Anführerauswahl werden Zähler nur alle $c \log s$ Schritte angestoßen, wobei $\log s$ die Länge des Schrittzähler ist und c eine positive Konstante, die wir frei wählen können. Folglich wird ein Zähler für $c \geqslant 4$ höchsten alls 4 Schritte zu einer weiteren Bewegung angestoßen.

Die obige Beschreibung ist nur richtig, wenn das Vorzeichen, das beim niedrigstwertigen Bit gespeichert ist, immer korrekt ist. Ein Wert ungleich Null kann bei Addition oder Subtraktion von 1 sein Vorzeichen (+ oder -) behalten oder zu $\pm$ wechseln. Letzterer Fall erfordert zusätzliche Maßnahmen.

10.3.4 Das richtige Vorzeichen

Um das beser zu sehen betrachten wir die Fälle, in denen die Zähler `(101+` und `(001+` um 1 erniedrigt werden. Nehmen wir an, sie würden beide 3 Schritte machen. Gemäß der bisherigen

	(1	0	0+	M		
				-1		
	(1	0	M	1+	M	
				-1		
	(1	M	0	M	1+	M
				-1		
		(1	M	1	M	1+
				-1		
			(1	M	1	1+
				-1		
				(0	1	1+
				0		

Abbildung 10.4: Raum-Zeit-Diagram, das die Bewegung eines Raumzählers zeigt. Für jede Zelle sind ein Register für das Summenbit (oben) und ein Register für das Übertragsbit (unten) dargestellt. Ein Übertragsbit von 0 ist meist nicht aufgeschrieben. Das erste M-Signal startet eine Erniedrigung des Zählers, das zweite und dritte nicht.

Beschreibung wären die Zählerinhalte hinterher (100+ bzw. (000+. Das ist im zweiten Fall falsch, es sollte (0± sein.

Der einfachste Ausweg besteht darin, den Algorithmus wie folgt leicht zu verändern. Da sich die Zähler nicht „sehr oft“ bewegen (siehe Bemerkung 10.28), kann man leicht sicherstellen, dass immer, wenn eine führende Null erzeugt wird, diese umgehend gelöscht und die (Markierung zum nächst niedrigeren Bit verschoben wird. Folglich kann 0± nur entstehen, wenn (1+ erniedrigt oder (1- erhöht wird. In beiden Fällen kann man die lokale Überführungsfunktion leicht so implementieren, dass das Richtige passiert.

10.3.5 Vergleich voller Zähler

In diesem Abschnitt gehen wir davon aus, dass zwei volle Zähler in einer Kollisionszelle aufeinander treffen. Der Vergleich zweier voller Zähler erfordert den Vergleich einander entsprechender Raumzähler für jede Dimension i und die anschließende Bestimmung der kleinsten Dimension, in der die Zählerstände verschieden sind. Aus technischer Sicht stellen sich hier drei Aufgaben:

- die “Extraktion” der Zählerwerte aus den Raumzählern (nachdem alle Überträge abgearbeitet sind),
- die “Extraktion” des Wertes aus dem Schrittzähler und
- den Vergleich der Werte der Raumzähler.

Die Extraktion der Zählerstände wird gestartet, indem ein E-Signal von der Kollisionszelle in Richtung der höchstwertigen Bits der Zähler geschickt wird; dafür verwenden wir ein zusätzliches Register, das Extraktionsregister. Wann immer ein E-Signal eine Zelle verlässt, wird das Extraktionsregister mit dem Summenbit gefüllt. Nach dem E-Signal schickt die Kollisionszelle

in jedem vierten Schritt ein M-Signal (siehe Abschnitt 10.3.2). Die Zählerzellen benutzen das für die Extraktionsregister wie für die Zählerregister: die Werte werden in Richtung Kollisionszelle geschoben.

Letztere benutzt d Register $r_1, \ldots, r_d$, um die Ergebnisse der Raumzählervergleiche zu bestimmen, wie es in der folgenden Modifikation von Algorithmus 10.15 beschrieben ist:

10.29 Algorithmus.

for all $i \in \{1, \ldots, d\}$ **do** $r_i \leftarrow 0$ **od**
while ⟨*most significant bit of step counter did not arrive*⟩ **do**
  for all $i \in \{1, \ldots, d\}$ **in parallel do**
    $z_i^1 \leftarrow$ ⟨*next bit of counter* i *from wave 1*⟩
    $z_i^2 \leftarrow$ ⟨*next bit of counter* i *from wave 2*⟩
    if $z_i^1 \neq z_i^2$ **then**
      $r_i \leftarrow \text{sign}(z_i^1 - z_i^2)$
    fi
  od
  $m \leftarrow \min(\{d+1\} \cup \{i \mid r_i \neq 0\})$
od

Sind Raumzähler kürzer als der Schrittzähler, so werden in in obigem Algorithmus fehlende führende Nullen als solche interpretiert.

Ist am Ende $m = d + 1$, dann waren die Raumzählerwerte für jede Dimension gleich. Andernfalls kann man r_m für die Entscheidung benutzen, welche Welle von der besseren Zelle kam; siehe Algorithmus 10.15.

10.3.6 Bugfix

Die eben beschriebene (und so leider auch publizierte) Behandlung von Zählern ist fehlerhaft. Es kann sein, dass nach der teilweisen Aufspaltung eines Zähler in verschiedene Richtungen auf dem einen Pfad eine Erhöhung nötig ist und auf dem anderen nicht. Die Erhöhung kann aber dazu führen, dass Zählerbits, die im noch nicht aufgespaltenen, also identischen, Teil liegen unterschiedliche Werte speichern müssten.

Die Abhilfe besteht darin, die Überträge *nicht* zu den höherwertigen Bits durchzureichen. Statt dessen speichert man sie in den Zellen, in den sie entstehen. Es genügt, sie dann zu verarbeiten, wenn Zähler zum Vergleichen an die „Wellenfront“ geschoben werden. Das geht genauso wie das bei einfachen Wanderzählern in Kapitel 6 demonstriert wurde.

10.4 Laufzeit des Algorithmus

10.30 Satz. *Die Laufzeit des obigen Algorithmus zur Anführerauswahl ist* $O(diam \cdot \log(diam))$.

10.31 Beweis Der Abstand zwischen der besten Zelle und irgendeiner anderen Zelle des Anfangsmusters ist $\leqslant diam$. Nach Lemma 10.20 wird keine Zelle einen größeren Abstand von der besten Zelle haben, wenn die besten Zähler den aufspannenden Baum erzeugt haben.

Folglich muss kein Zähler mehr als $O(diam)$ Zellen durchlaufen. Die Länge voller Zähler ist durch $O(\log diam)$ beschränkt. Deswegen bewegen sie sich mindestens alle $O(\log diam)$ Zeitschritte um eine Zelle weiter. Folglich ist nach spätestens $O(diam \cdot \log(diam))$ Zeitschritten der beste aufspannende Baum erzeugt und das Schrumpfen des Graphen beginnt.

Während dieses Prozesses wird nach jeweils einer konstanten Anzahl von Schritten die maximale Entfernung der Wurzel von irgendeinem Blatt um 1 verringert. Also bleibt nach $O(diam)$ Schritten nur noch die beste Zelle übrig.

Die Summe der Laufzeiten ist offensichtlich in $O(diam \cdot \log(diam))$. ■

Zusammenfassung

- In d-dimensionalen Zellularautomaten kann man das Leader-Election-Problem in beliebigen Mustern (auch mit Löchern) in Zeit $(\Theta(diam \cdot \log(diam)))$ lösen.
- Für Muster ohne Löcher kann man mit Zeit $(\Theta(diam))$ auskommen (Nichitiu, Papazian und Rémila 2004).
- Die beste bekannte untere Schranke für den Zeitbedarf $O(diam)$. Es ist ein nach wie offenes Problem, ob man dies nach oben verbessern kann oder Laufzeiten von Algorithmen nach unten.

Literatur

Beckers, Andreas und Thomas Worsch (1998). «A perimeter-time CA for the queen-bee problem». In: *Cellular Automata: Research towards Industry*. Hrsg. von Stefania Bandini, Roberto Serra und Furio Suggi Liverani. London: Springer, S. 15–25 (siehe S. 74).

— (2001). «A perimeter-time CA for the queen-bee problem». In: *Parallel Computing* 27.5, S. 555–569 (siehe S. 74).

Hemmerling, Armin (1979). «Concentration of Multidimensional Tape-Bounded Systems of Turing Automata and Cellular Spaces». In: *International Conference on Fundamentals of Computation Theory (FCT '79)*. Hrsg. von L. Budach. Berlin: Akademie-Verlag, S. 167–174 (siehe S. 74).

Mazoyer, Jacques, Codrin Nichitiu und Eric Rémila (1999). «Compass permits leader election». In: *Proceedings of SODA'99*, S. 947–948 (siehe S. 74).

Nichitiu, Codrin, Christophe Papazian und Eric Rémila (2004). «Leader election in plane cellular automata, only with left-right global convention». In: *Theoretical Computer Science* 319.1–3, S. 367–384 (siehe S. 87).

Nichitiu, Codrin und Eric Rémila (1999). «Leader Election by d-Dimensional Cellular Automata». In: *Automata, Languages and Programming, 26th International Colloquium (ICALP)*. Bd. 1644. Lecture Notes in Computer Science. Springer-Verlag, S. 565–574 (siehe S. 74, 78).

Smith III, Alvy Ray (1971). «Two-dimensional formal languages and pattern recognititon by cellular automata». In: *Proceedings of the Twelfth Annual IEEE Symposium on Switching and Automata Theory*, S. 144–152 (siehe S. 74).

Stratmann, Michael (2000). «Eine Lösung für das Queen-Bee-Problem in Zeit $O(diam(\log diam)^{dim+1})$». Studienarbeit. Fakultät für Informatik, Universität Karlsruhe (siehe S. 74).

Stratmann, Michael und Thomas Worsch (2002). «Leader Election in d-dimensional CA in time $diam \cdot \log(diam)$». In: *Future Generation Computer Systems* 18.7, S. 939–950 (siehe S. 73, 74).

Vollmar, Roland (1977). «On two modified problems of synchronization in cellular automata». In: *Acta Cybernetica* 3, S. 293–300 (siehe S. 76).

— (1979). *Algorithmen in Zellularautomaten*. Stuttgart: Teubner (siehe S. 74).

