14 Das Sandhaufen-Modell

In diesem Kapitel werden wir einen bestimmten Zellularautomaten betrachten. Neben einigen modellspezifischen Punkten

Das „Sandpile Model“ wurde von Bak, Tang und Wiesenfeld 1987 ursprünglich als ein einfaches Beispiel für ein System eingeführt, das *selbstorganisierte Kritikalität* (engl. *self-organized criticality*) aufweist. Eine genauere Auseinandersetzung mit diesem Begriff soll nicht Gegenstanddieser Vorlesung sein; Interessierte seien auf das Buch von Jensen 1998 verwiesen.

14.1 Der Zellularautomat und das Modell außen herum

14.1 Definition Dem Modell von Bak, Tang und Wiesenfeld liegt der folgende einfache Zellularautomat zu Grunde:

- zweidimensionaler Raum mit $R = \{0, \dots, k\} \times \{0, \dots, l\}$
- Von-Neumann-Nachbarschaft $N = H_1^{(2)}$ mit Radius 1
- Zustandsmenge $Q = \{0, 1, 2, 3, 4, 5, 6, 7\}$
- Die lokale Überführungsfunktion könnte man als Formel etwa wie folgt hinschreiben[1]:

$$\delta(\ell) = -4 \cdot [\ell(0) \geqslant 4] + \sum_{y \in N, y \neq 0} [\ell(y) \geqslant 4]$$

Für eine Beschreibung kann man sich vorstellen, dass jede Zelle x in einer Konfiguration c sozusagen c_x „Sandkörner“ enthält.

- Eine Zelle x mit $c_x \geqslant 4$ heißt *kritisch*. Jede kritische Zelle gibt in einem Schritt vier Sandkörner ab. Wir sagen auch, dass die Zelle *feuert*. Je ein Sandkorn wird in jede der vier Himmelsrichtungen weitergegeben.
- Unkritische Zellen tun „von sich aus“ nicht, sie können aber von Nachbarzellen Sandkörner erhalten.

Für die Zellen am Rand bedeutet dies, dass sie aus manchen Richtungen (nämlich von nicht existierenden Nachbarn) niemals Sandkörner erhalten. Andererseits „fallen die Körner aus der Konfiguration heraus“, die eine Randzelle in eine Richtung abgibt, in der sich keine Nachbarzelle mehr befindet. ◇

Man mache sich als Übung klar, dass hier eine sinnvolle Definition vorliegt: Für jede lokale Konfiguration ℓ ist $\delta(\ell)$ wieder ein Wert aus Q.

14.2 Beispiel. Für den einfachen Fall eines 2×2-Feldes würde sich zum Beispiel folgender Konfigurationsübergang ergeben:

[1] Ein Boolescher Ausdruck in eckigen Klammern sei 1, wenn die Bedingung erfüllt ist, und 0 sonst.

4	2
5	1

$\mapsto$

1	3
2	2

Eine Konfiguration, in der alle Zellen unkritisch sind, heiße auch *stabil*, weil in diesem Fall $\Delta(c) = c$ ist.

14.3 Lemma. Für jede Konfiguration c gilt: Nach einer (von c abhängigen) endlichen Zahl t von Schritten ist $\Delta^t(c)$ stabil.

14.4 Beweis Angenommen, die Aussage wäre falsch und es gäbe eine Konfiguration c, für die alle $\Delta^t(c)$ nicht stabil wären. Dann würde also in jeder Konfiguration $\Delta^t(c)$ mindestens eine Zelle feuern. Also gäbe es mindestens eine Zelle x, die unendlich oft feuern würde.

Dann würde aber auch die rechte Nachbarzelle $r(x)$ unendlich oft feuern, weil sie von ihrem linken Nachbarn unendlich viele Sandkörner erhielte. Folglich würde auch deren rechte Nachbarzelle $r(r(x))$ von $r(x)$ unendlich oft feuern, usw., so dass also auch die Randzelle rechts von Zelle x unendlich oft feuern würde.

Also würde von dort unendlich oft ein Sandkorn über den Rand aus dem Zellularautomaten herausfallen. Das ist aber unmöglich, da von Anfang an nur endlich viele Sandkörner, nämlich $\sum_{x \in R} c_x$ viele, vorhanden sind. ∎

14.5 Wenn man in einer stabilen Konfiguration c einer Zelle x von außen zusätzlich ein Sandkorn hinzufügt, dann kann daraus eine instabile Konfiguration entstehen, aus der nach endlich vielen Schritten wieder eine stabile ensteht. Diese wollen wir mit $A_x(c)$ bezeichnen.

Man sagt auch, dass durch das Hinzufügen des Sandkorns eine *Lawine* entstehe. Zur Charakterisierung deren Größe kann man verschiedene Maßzahlen verwenden:

- die Anzahl der ZA-Schritte bis zum Erreichen von $A_x(C)$,
- die Anzahl der Zellen, die bis zum Erreichen von $A_x(C)$ mindestens einmal (wie oft auch immer) instabil sind oder
- die Anzahl der Ereignisse, dass eine Zelle einmal feuert.

Im folgenden wird die letzte Möglichkeit zu Grund gelegt.

14.6 Definition Das Sandpile Model sieht vor, dass ausgehend von einer stabilen Konfiguration c immer wieder zufällig gleichverteilt eine Zelle x ausgewählt wird, ihr ein Sandkorn hinzugefügt wird und dann der Zellularautomat läuft, bis $A_x(c)$ erreicht ist. Mit dieser Konfiguration wird dann ebenso weiter verfahren, usw. Dabei wird jedes Mal die Größe der Lawine vom Werfen des Sandkorns bis zum Erreichen der nächsten stabilen Konfiguration beobachtet. ◇

14.7 Das Bemerkenswerte am Sandpile Model ist die Tatsache, dass die Häufigkeitsverteilung der verschiedenen Lawinengrößen relativ gut übereinstimmt mit der Häufigkeitsverteilung der Stärken von Erdbeben.

Die Erdbeben-Verteilung wird durch das sogenannte *Gutenberg-Richter-Gesetz* beschrieben:

$$\log N \approx a - b \log E$$

Dabei ist E die bei einem Erdbeben freigesetzte Energie und N die Anzahl solcher Erdbeben. Der Zusammenhang zwischen E und N wird also durch ein Potenzgesetz beschrieben (d. h. er ist polynomiell). Der Wert für b schwankt zwar regional ein bisschen, im wesentlich gilt aber $b \approx 1$.

Die Stärke eines Erdbebens auf der Richter-Skala ist übrigens proportional zum Zehner-Logarithmus der Auslenkung der Erdoberfläche bei dem Beben. Ist die Stärke um 1 größer, ist die Auslenkung 10 mal so groß. Die freigesetzte Energie bei einem Erdbeben mit um 1 höherer Stärke ist übrigens sogar 32 mal größer.

14.2 Einfache Simulation auf dem Rechner

Wir schreiben im folgenden $x \pm M = \{x \pm m \mid m \in M\}$.

Der „Standardalgorithmus“ für die Simulation eines Zellularautomaten benutzt zwei Speicherbereiche c und c', die globale Konfigurationen enthalten. In c ist der aktuelle Zustand aller Zellen abgelegt. Es wird über alle Zellen $x \in R$ iteriert, die Zustände der Zellen $x + N$ aus c gelesen und der neue Zustand von x an der richtigen Stelle in c' gespeichert. Am Ende werden die Rollen von c und c' vertauscht, damit man für die Simulation des nächsten globalen Konfigurationsüberganges vorbereitet ist.

14.8 Algorithmus. Idealisiert lässt sich die Standardsimulation eines Konfigurationsüberganges etwa so notieren:

foreach $x \in R$ **do**
    $c'_x \leftarrow \delta(c_{x+N})$
od
$c \leftrightarrow c'$

Dabei drückt man sich allerdings noch um diverse Implementierungsdetails, z. B. den Zugriff auf Nachbarn einer Zelle x. Zur Klärung betrachten wir zwei Möglichkeiten, die der Einfachheit halber anhand eines eindimensionalen Zellularautomaten demonstriert werden. Die lokale Überführungsfunktion sei $\delta(\ell) = \ell(-1) \cdot \ell(0) \cdot \ell(1)$.

14.9 Algorithmus. In dem folgenden Ausschnitt aus einem C-Programm werden für c und c' die Namen `c1` und `c2` benutzt. Es sei `L` die Größe der (endlichen) Konfigurationen.

Am bequemsten, aber *falsch*, wäre es zu schreiben:

```
int c1[L];   int c2[L];   int x;

for (x=0; x<L; x++) {
   c2[x] = c1[x-1]*c1[x]*c1[x+1];
}
```

Das Problem sind die Indexausdrücke `x-1`, falls `x` den Wert 0 hat, und `x+1`, falls `x` den Wert $L-1$ hat. Die entsprechenden Feldzugriffe wären nicht definiert. Für die weitere Diskussion wollen wir davon ausgehen, dass zyklische Randbedingungen realisiert werden sollen, d. h. „linker“ Nachbar von Zelle 0 soll Zelle $L-1$ sein und umgekehrt soll rechter Nachbar der Zelle $L-1$ die Zelle 0 sein.

Zwei mögliche Reparaturen des obigen Fehlers bestehen darin, die Behandlung der Randzellen über eine `if`-Anweisung zu implementieren oder alle Indexrechnungen modulo `L` zu machen. Beides hat den Nachteil, für die Bearbeitung jeder Zelle zusätzliche Rechenzeit zu benötigen. Dies vermeidet die Methode der „Geisterzellen“. Man benutzt Konfigurationspuffer,

die Platz für zwei zusätzliche Zustände bieten. Die eigentlich Konfiguration wird jetzt in den Feldkomponenten 1 bis L gespeichert, und Zellen 0 und $L+1$ übernehmen die Rolle der oben noch fehlenden Nachbarn:

```
int c1[L+2];   int c2[L+2];   int x;

c1[0] = c1[L-1];
c1[L+1] = c1[1];
for (x=1; x<=L; x++) {
   c2[x] = c1[x-1]*c1[x]*c1[x+1];
}
```

14.10 Für eine effiziente Simulation sind natürlich noch weitere Aspekte von Bedeutung.

- Zum Beispiel kann man statt statisch deklarierter Felder auch Zeigervariablen benutzen (die auf dynamisch allozierte Speicherbereiche verweisen). Dann ist der Rollentausch von c und c' durch einfachen Zeigertausch realisierbar. Man wird im allgemeinen *nicht* alle Zustände kopieren wollen.
- Für die Berechnung der neuen Zustände kommen auch ganz verschiedene Vorgehensweisen in Frage. Neben arithmetischen Ausdrücken wie oben trifft man auch Funktionsaufrufe oder aber (hoffentlich schnelles) Nachschlagen in einer Tabelle.
- Bei der Form, in der die Zustände abgespeichert werden, haben je nachdem auch verschiedene Möglichkeiten ihre Vorteile. Statt wie oben nur je einen ganzen Zustand in einem ganzen Speicherwort abzulegen, kann man manchmal auch mehrere in ein Wort packen.

  Beim sogenannten *Multi Spin Coding* benutzt man

  - ein erstes Maschinenwort, um die ersten Bits einer Reihe von Zellen zu speichern,
  - ein zweites Maschinenwort, um die zweiten Bits einer Reihe von Zellen zu speichern,
  - usw.

Für Details verweisen wir auf das Buch von Weimar 1997.

14.3 Schnelle Simulation bei wenigen Zustandsänderungen

14.11 Nur dann kann eine Zelle möglicherweise ihren Zustand echt ändern, wenn sich im Schritt zuvor der Zustand mindestens einer Nachbarzelle geändert hat. Wenn bei einem Zellularautomaten in einem Schritt nur wenigen Zellen ihren Zustand echt ändern, dann können das im nächsten Schritt nur die (immer noch wenigen) Nachbarzellen dieser Zellen tun. In solch einem Fall ist die eben skizzierte Simulationsmethode ineffizient, weil die meisten Zellen ihren Zustand gar nicht ändern.

Und das kann die Simulation auch wissen. Der folgende Algorithmus versucht, schlauer zu sein. Er wurde immer wieder in der ein oder anderen ähnlichen Form wieder entdeckt, zum Bespiel von Walter und Worsch 2004.

Die Idee besteht darin, eine Menge $A \subseteq R$ von Zellen zu verwalten, so dass man sicher sein kann, dass, wann immer $x \notin A$ ist, für Zelle x kein neuer Zustand berechnet werden muss (weil c' ihn, nämlich den alten, schon enthält).

14.12 Algorithmus. Der nachfolgende Text ist bereits mit Zusicherungen versehen, auf die wir beim Nachweis der Korrektheit zurückkommen werden. Außerdem sei $N_0 = N \cup \{0\}$.

$A' \leftarrow \emptyset$
$\boxed{Z1}$ ——— $\langle\ \forall x \in R : x \notin A \Longrightarrow c'_x = c_x \wedge c_x = \delta(c_{x+N})\ \rangle$
foreach $x \in A$ **do**
  $c'_x \leftarrow \delta(c_{x+N})$
  if $c'_x \neq c_x$ **then**
    foreach $y \in x - N_0$ **do**
      $A' \leftarrow A' \cup \{y\}$
    od
  fi
od
$\boxed{Z2}$ ——— $\langle\ c' = \Delta(c)$, i.e. $\forall x \in R : c'_x = \Delta(c)_x\ \rangle$
$\boxed{Z3}$ ——— $\langle\ \forall x \in R : x \notin A' \Longrightarrow c'_x = c_x \wedge c'_x = \delta(c'_{x+N})\ \rangle$
$c \leftrightarrow c'$
$A \leftarrow A'$
$\boxed{Z4}$ ——— $\langle\ \forall x \in R : x \notin A \Longrightarrow c'_x = c_x \wedge c_x = \delta(c_{x+N})\ \rangle$

14.13 Lemma. Der obige Algorithmus ist korrekt.

14.14 Beweis Wir nehmen an, Zusicherung Z1 ist wahr, und betrachten ein beliebiges $x \in R$ unmittelbar bevor die Schleife „**foreach** $x \in A$ **do** ... **od**“ betreten wird.

Fall 1: $x \in A$**.** Dann wird der Schleifenrumpf für x ausgeführt. Offensichtlich erhält man $c'_x = \delta(c_{x+N}) = \Delta(c)_x$ wie in Zusicherung Z2 behauptet.

Fall 2: $x \notin A$**.** In diesem Fall ist schon wegen Zusicherung Z1 klar, dass $c'_x = \delta(c_{x+N}) = \Delta(c)_x$ gilt.

Also gilt Zusicherung Z2 in jedem Fall.

Betrachten wir nun die Menge A', nachdem die Schleife „**foreach** $x \in A$ **do** ... **od**“ beendet ist, und ein beliebiges $y \in R$:

Fall 1: $y \in A'$**.** In diesem Fall ist nichts zu beweisen.

Fall 2: $y \notin A'$**.** Dies kann nur passieren, wenn $\neg\exists x \in R : y \in x - N_0 \wedge c'_x \neq c_x$. Das ist äquivalent zu $\neg\exists x \in R : x \in y + N_0 \wedge c'_x \neq c_x$, und das zu $\forall x \in R : x \notin y + N_0 \vee c'_x = c_x$.

Wegen $y \in y + N_0$ ist $c'_y = c_y$.

Außerdem gilt $c'_x = c_x$ für alle $x \in y + N$, d. h. $c'_{y+N} = c_{y+N}$, und daher $\delta(c'_{y+N}) = \delta(c_{y+N})$. Von Zusicherung Z2 wissen wir schon, dass $\Delta(c)_y = c'_y$, also $\delta(c'_{y+N}) = c'_y$.

Also ist Zusicherung Z3 wahr.

Aus Zusicherung Z3 erhält man mit den Zuweisungen $c \leftrightarrow c'$ und $A \leftarrow A'$ schnell Zusicherung Z4.

Zusicherung Z2 besagt, dass der Algorithmus tatsächlich $c' = \Delta(c)$ berechnet.

Man beachte, dass die 0 in der Schleife „**foreach** $y \in x - N_0$ **do** ... **od**“ wirklich wichtig ist. Falls eine Nachbarschaft nicht 0 enthält, würde man andernfalls im allgemeinen falsche Ergebnisse erhalten. ∎

14.15 Benutzt man obigen Algorithmus in einer Schleife, um mehrere globale Schritte eines Zellularautomaten zu simulieren, dann ist Zusicherung Z1 eine wichtige Schleifeninvariante, denn Z4 stimmt mit Z1 überein.

14.4 Zwei Eigenschaften des Sandhaufenmodells

Die im folgenden dargestellten experimentellen Daten wurden für die Simulation eines Quadrates mit 1000×1000 Zellen ermittelt, in das wiederholt ein zusätzliches Sandkorn in eine zufällig gleichverteilt ausgewählte Zelle geworfen wurde.

14.16 Als erstes gehen wir von der Konfiguration aus, in der alle Zellen im Zustand 0 sind. Abbildung 14.1 zeigt im oberen Teil den relativen Anteil der Zustände 0 bis 3 an den nach jeweils 1000 Körnern erreichten stabilen Konfigurationen. Wie man sieht, haben die Kurven nach ungefähr 2.13 mal Zellenzahl vielen Körnern einen „Knick“ und sind von da an zumindest näherungsweise konstant. Größere Simulationen bestätigen diese Beobachtung.

Im unteren Teil sind die absoluten Rechenzeiten. Aus Gründen, auf die wir später noch zu sprechen kommen werden, korrelieren sie bei der implementierten Simulationsmethode sehr stark mit der Summe der Größen der Lawinen für je 1000 Sandkörner. Die Simulationszeiten steigen in relativ kurzer Zeit stark an, und zwar in etwa zu dem Zeitpunkt, zu dem auch die relativen Häufigkeiten der Zustände „konstant“ werden.

14.17 Offensichtlich werden nach einiger Zeit Konfigurationen erreicht, die sich qualitativ deutlich von den vorher durchlaufenen unterscheiden. Auf eine Eigenschaft dieser Konfigurationen kommen wir gleich noch zu sprechen. Physiker sprechen davon, dass ein „Zustand“ *selbstorganisierter Kritikalität* (engl. *self-organized criticality (SOC)*) erreicht sei. Was ist das?

Jensen 1998 vermerkt hierzu:

> „There does not exist a clear-cut and generally accepted definition of what SOC is.“

Man kann aber sagen, dass sich das System *ohne Justieren irgendwelcher Parameter* so entwickelt, dass irgendwann globale Zustände durchlaufen werden, die eine (oder mehrere) charakteristische Eigenschaft(en) haben. Eine Eigenschaft, die wir eben schon gesehen haben, sind die mehr oder weniger konstanten Anteile der Zustände $0, \ldots, 3$ an den Konfigurationen.

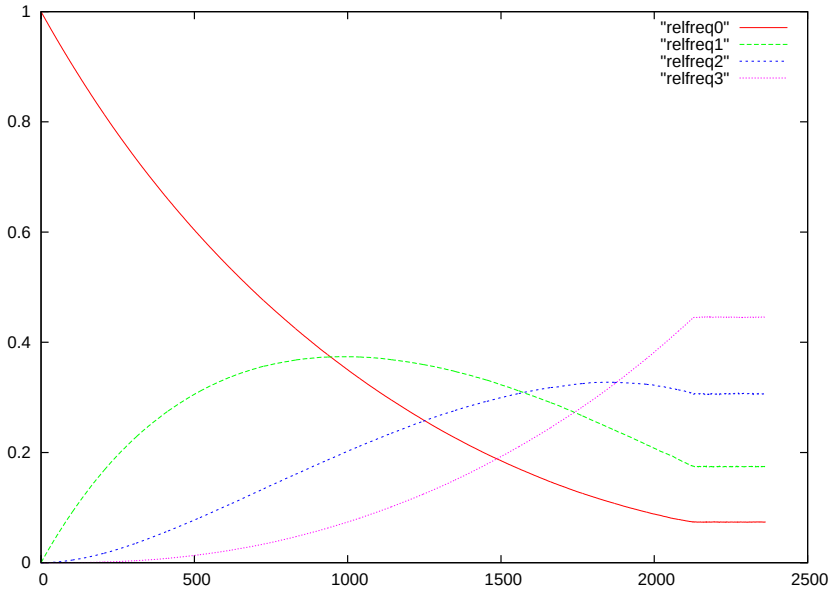
14.18 Aber Achtung: Wir behaupten nicht, dass alle Konfigurationen mit diesen Anteilen von Zuständen nach Erreichen von SOC auftreten. Das ist *nicht* so. Das sieht man zum Beispiel, wenn man die folgende Untersuchung macht.

14.19 Als zweites wollen wir uns nun die Häufigkeiten von Lawinen verschiedener Größe ansehen. Abbildung 14.2 zeigt mit doppelt logarithmischen Skalen die Häufigkeitsverteilung verschieden großer Lawinen. Die Anzahl der Lawinen der Größe 0 ist an der Stelle 0.11 eingetragen. Ermittelt wurden diese Daten ausgehend von einer Konfiguration, bei der man davon ausgehen kann, dass selbstorganisierte Kritikalität erreicht war.

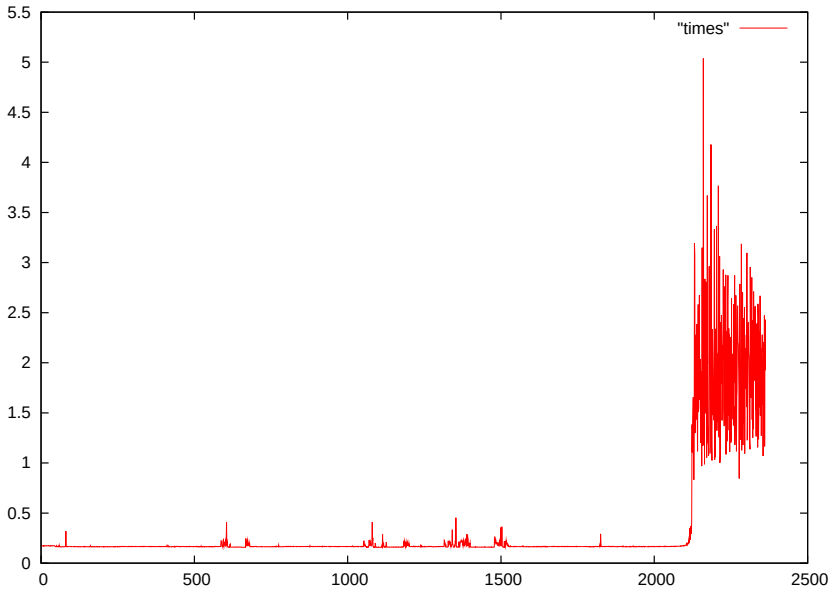
Man sieht, dass – zumindest in einem gewissen Bereich – die Messpunkte in der Nähe einer Geraden mit Steigung -1 liegen. Diese Übereinstimmung mit dem Gutenberg-Richter-Gesetz ist ein Grund, weshalb sich Geophysiker für das Sandhaufenmodell interessieren.

Zusammenfassung

- Das Sandhaufenmodell von Bak, Tang und Wiesenfeld 1987 zeigt sogenannte selbstorganisierte Kritikalität.



(a) relative Häufigkeiten der Zustände $0, 1, 2, 3$ in den stabilen Konfigurationen



(b) Simulationszeiten für je 1000 Sandkörner

Abbildung 14.1: Einschwingphase bei leerer Anfangskonfiguration; jeder Datenpunkt steht für 1000 Sandkörner; Feldgröße 1000×1000

- Es ist möglicherweise ein Modell für gewisse Phänomene im Zusammenhang mit Erdbeben.
- Für Zellularautomaten, bei denen häufig nur wenige Zellen ihren Zustand ändern, muss man für effiziente Simulation den Standardalgorithmus durch etwas besseres ersetzen.

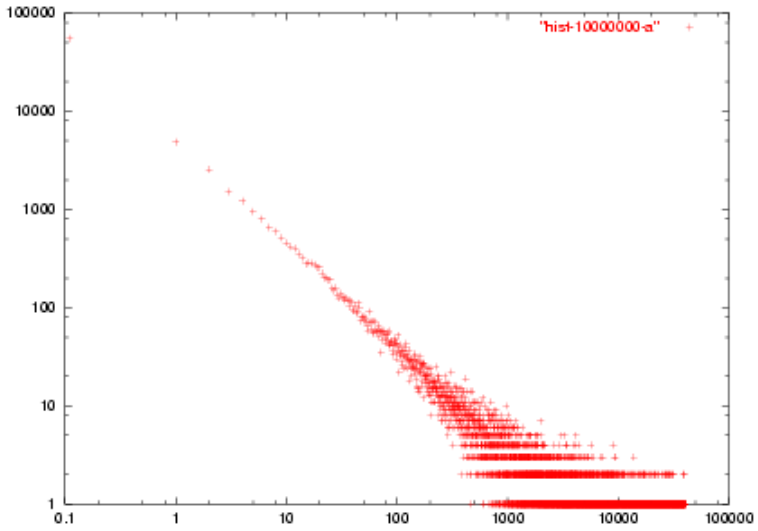


Abbildung 14.2: Häufigkeit von Lawinen

Literatur

Bak, Per, Chao Tang und Kurt Wiesenfeld (1987). «Self-organized criticality: An explanation of the 1/f noise». In: *Physical Review Letters* 59.4, S. 381–384 (siehe S. 109, 114).

Jensen, Henrik Jeldtoft (1998). *Self-Organized Criticality*. Cambridge University Press. ISBN: 0-521-48371-9 (siehe S. 109, 114).

Walter, Richard und Thomas Worsch (2004). «Efficient Simulation of CA with Few Activities». In: *6th International Conference on Cellular Automata for Research and Industry, ACRI 2004*. Hrsg. von Peter M. A. Sloot, Bastien Chopard und Alfons G. Hoekstra. Bd. 3305. LNCS. Springer, S. 101–110 (siehe S. 112).

Weimar, Jörg Richard (1997). *Simulation with Cellular Automata*. Berlin: Logos-Verlag (siehe S. 112).

