

Pseudozufallszahlengeneratoren

Gerrit Blöss

Fachbereich Informatik der Universität Hamburg
Vogt-Kölln-Straße 30
Hamburg, Germany

5bloess@informatik.uni-hamburg.de

ABSTRACT

Für viele verschiedene Anwendungsbereiche werden Zufallszahlen benötigt. Oft reichen mit mathematischen Verfahren erzeugte Pseudozufallszahlen aus. In diesem Paper werden die Kongruenzalgorithmen beschrieben, die ein sehr grundlegendes Vorgehen zur Erzeugung von Pseudozufallszahlen sind. Außerdem wird auf einfache Verfahren zum Testen der Qualität von generierten Zufallszahlen eingegangen. Zusätzlich dazu werden mögliche Unsicherheiten im Zusammenhang mit der Erzeugung von Pseudozufallszahlen an Hand eines konkreten Beispiels erläutert.

Allgemeine Begriffe

Pseudozufallszahlen, Pseudozufallszahlengeneratoren

Keywords

Pseudozufallszahlen, Pseudozufallszahlengeneratoren, Kongruenzgeneratoren, Fibonacci-Algorithmus, Chi-Quadrat-Test, Runs-Test, Run-up-Test, Run-down-Test, Mersenne Twister

1. EINLEITUNG

Für viele verschiedene Anwendungszwecke besteht ein Bedarf an zufälligen Zahlenfolgen. Dabei unterscheiden sich die Anforderungen an die tatsächliche „Zufälligkeit“. Kryptographische Anwendungen (wie z.B. in das in [1] beschriebene RSA-Verfahren) benötigen zur Schlüsselerzeugung möglichst „echte“ Zufallszahlen, um eine ausreichend hohe Sicherheit zu gewährleisten. Teilweise wird hier die Erzeugung der Zahlen durch Mausbewegungen oder Tasteneingaben des Benutzers beeinflusst, oder es werden physikalische Generatoren verwendet, die auf unvorhersagbaren Vorgängen wie schwankendem Widerstand oder radioaktiver Emission basieren, um zu verhindern, dass sich die Zahlenfolgen nach einem bestimmten Muster richten.

In anderen Bereichen ist der Anspruch an die Zufälligkeit der Zahlen nicht so hoch. Exemplarisch seien hier Optimierungsverfahren erwähnt, wie sie in [2] beschrieben werden. Es bietet sich die Verwendung von Pseudozufallszahlengeneratoren an. Die wohl einfachste Form der hierfür herangezogenen Algorithmen sind die Kongruenzalgorithmen, die im folgenden Abschnitt beschrieben werden.

Zur Bewertung der Zufälligkeit der von einem Algorithmus erzeugten Zahlen gibt es einige Tests, sowohl theoretische als auch empirische. In Abschnitt 3 wird auf zwei empirische Testverfahren näher eingegangen, den χ^2 -Test und den Runs-Test.

Leider wird in manchen kritischen Fällen die Erzeugung der Zufallszahlen falsch gehandhabt, oder es wird auf vorhandene Algorithmen und Bibliotheken zugegriffen, die unzureichende Sicherheit bieten. Deswegen lege ich im vierten Abschnitt anhand eines konkreten Beispiels dar, welche Sicherheitslücken sich bei bestimmten Pseudozufallszahlengeneratoren bzw. deren Fehleinsatz auftun können.

In Abschnitt 5 gebe ich eine kurze Zusammenfassung, sowie eine kurze Beschreibung der Eigenschaften des Mersenne Twister, eines neueren Algorithmus zur Zufallszahlenerzeugung.

2. KONGRUENZALGORITHMEN

Alle Kongruenzalgorithmen basieren auf der modularen Arithmetik und profitieren von dem Umstand, dass zwei verschiedene Zahlen bezüglich einer anderen Zahl kongruent sein können. Die Modulo-Rechnung „verwischt“ gewissermaßen die Spuren der Erzeugung.

2.1 Lineare Kongruenzgeneratoren

Lineare Kongruenzalgorithmen sind (nach [3]) von der Form:

$$(2-1) \quad x_{i+1} \equiv a \cdot x_i + c \pmod{M}$$

mit $x_i, a, c, M \in \mathbb{N}$ und $x_0 \in \{0, \dots, M-1\}$.

Dabei wird x_0 als Saat bezeichnet. Aus der Gleichung folgt, dass $0 \leq x_{i+1} < M$. Um die rekursive Berechnung zu umgehen, kann man Gleichung (2-1) umformen:

$$(2-2) \quad x_i \equiv a^i \cdot x_0 + \frac{c \cdot (a^i - 1)}{(a - 1)} \pmod{M}$$

Jede von einem Algorithmus dieser Form erzeugte Zahlenfolge wird sich auf Grund der Kongruenzbeziehung irgendwann wiederholen. Für $a = 3, c = 0, M = 8$ und $x_0 = 7$ erhält man $x_1 = 5$ und $x_2 = 7$. Schon ab x_3 wiederholt sich diese Folge, denn $x_3 = 5 = x_1$. [3] Die Anzahl der Iterationen, nach denen sich eine Wiederholung einstellt, nennt man Periode. Ein guter Zufallszahlengenerator zeichnet sich durch eine möglichst große Periode aus. In der Regel wählt daher man für M einen hohen Wert. Um die Ausführungsgeschwindigkeit zu erhöhen, bietet sich für M ein Wert an, der auf dem Zielsystem der größten in einem Prozessorregister darstellbaren Zahl entspricht, bei einem 32-Bit-Prozessor also sinnvollerweise 2^{32} ; a und c sollten so gewählt werden, dass man neben einer großen Periode auch eine vertretbare Ausführungszeit sowie eine möglichst zufällige statistische Verteilung der Zahlen erhält.

In der Regel wird von einem Algorithmus ein Wert zwischen 0 und 1 zurückgegeben, d.h. – mit Bezug auf (2-1) – wird der erzeugte Wert x_{i+1} durch M geteilt und das Ergebnis aus dieser Rechnung zurückgegeben.

Texas Instruments verwendeten (laut [4]) für ihren Taschenrechner TI-59 den folgenden Zufallszahlengenerator:

$$(2-3) \quad x_{i+1} \equiv 24298 \cdot x_i + 99991 \pmod{199017}$$

2.2 Multiplikative Kongruenzgeneratoren

Multiplikative Kongruenzgeneratoren basieren auf Gleichung (2-1), wobei jedoch $c = 0$ gilt. Aus dem Wegfall der additiven Komponente ergibt sich auch der Name. Durch den Verzicht auf die Addition wird die Geschwindigkeit bei der Zahlenerzeugung erhöht.

zeugung leicht erhöht, ohne die Qualität der gewonnenen Zahlen zu stark zu mindern. [5]

2.3 Additive Kongruenzgeneratoren

Bei additiven Kongruenzalgorithmen, die ebenfalls auf (2-1) basieren, gilt $a = 1$, d.h. bei jeder Iteration wird ein fester Wert c aufaddiert, um die jeweils nächste Zahl zu erhalten. In der Regel wird diese Form der Zufallszahlenerzeugung vermieden, da die gewonnenen Zahlenreihen durch verschiedene Tests, insbesondere auch den in 3.2 beschriebenen Runs-Test, leicht als nicht-zufällig zu identifizieren sind. [5]

2.4 Fibonacci-Algorithmus

Zur Zufallszahlenerzeugung kann auch die Reihe der Fibonacci-Zahlen verwendet werden:

$$(2-4) \quad x_{i+1} = x_i + x_{i-d} \text{ mod } M$$

Für die tatsächlichen Fibonacci-Zahlen wird $d = 1$ gesetzt, zur Variation des Algorithmus sind aber auch andere Werte möglich. Fibonacci-Generatoren haben sich aber als für die Erzeugung von Zufallszahlen als wenig geeignet erwiesen. [5]

3. TESTVERFAHREN

Wie erwähnt, gibt es einige Verfahren, um die Qualität der mit Algorithmen erzeugten Zahlreihen in Bezug auf deren Zufälligkeit, Verteilung und Unabhängigkeit zu prüfen. Dabei gibt es sowohl theoretische als auch empirische Verfahren. Zwei empirische Verfahren sollen im Folgenden näher erläutert werden.

3.1 χ^2 -Test (Chi-Quadrat-Test)

Der χ^2 -Test ist ein Test, mit dem sich die Gleichverteilung einer Menge von Zahlen überprüfen lässt. Die Vorgehensweise hierbei ist schnell erklärt: Man teilt die Menge in k Kategorien auf. Jede Zahl x_i aus der Menge muss einer der Kategorien zugeordnet werden. Man kann (nach [5]) z.B. vorgeben, dass x_i der Kategorie s zugewiesen wird, wenn

$$(3-1) \quad \frac{s-1}{k} < x_i \leq \frac{s}{k}$$

Es sei n die Anzahl der Zahlen in der Menge, p_s die Wahrscheinlichkeit, dass die Zahl x_i der Kategorie s zugeordnet wird und y_s die Anzahl der Zahlen, die der Kategorie s tatsächlich zugeordnet wurden. Dann gilt:

$$(3-2) \quad \chi^2 = \sum_{s=1}^k \frac{(y_s - n \cdot p_s)^2}{n \cdot p_s}$$

$y_s - n \cdot p_s$ ist das Maß für die absolute Abweichung der Anzahl der einer Kategorie zugeteilten Werte von der erwarteten Anzahl. Das Quadrieren dieses Werts ist nötig, da sich sonst die Abweichungen gegeneinander aufheben würden. Durch Teilen durch $n \cdot p_s$ erhält man die relative Abweichung der tatsächlichen von der erwarteten Anzahl der Werte in einer Kategorie.

Ein niedrigerer Wert von χ^2 steht für eine bessere Gleichverteilung der Werte. Das heißt jedoch nicht unbedingt, dass ein Generator, dessen Zufallszahlen generell einen niedrigen χ^2 -Wert aufweisen, ohne weiteres empfehlenswert ist. Vorstellbar wäre ein Generator der folgenden Form:

$$(3-3) \quad x_{i+1} \equiv x_i + 1 \pmod{5}$$

Obwohl die Anforderung der Gleichverteilung hier erfüllt wäre, ist leicht einzusehen, dass die mit diesem Algorithmus erzeugten Zahlen als „Zufallszahlen“ vollkommen unbrauchbar wären. Aus diesem Grund wird der χ^2 -Test nicht nur direkt auf

die erzeugten Zahlenreihen angewandt, sondern auch auf andere Werte im Zusammenhang mit dem zu prüfenden Zufallszahlengenerator. Näheres hierzu findet sich in Abschnitt 3.2.

3.2 Runs-Test

Mit einem gewöhnlichen Run-Test kann man die mit einem Zufallszahlengenerator erzeugten Zahlen auf ihre Unabhängigkeit hin überprüfen. Dabei wird ein Run als eine Folge von identischen Ereignissen definiert, und die Länge des Runs ist die Häufigkeit, mit der dieses Ereignis hintereinander auftritt. Normalerweise verwendet man für den Test von Pseudozufallszahlengeneratoren den Run-up-Test und den Run-down-Test. Beim Run-up-Test muss für eine beliebige Folge von Zahlen diese Bedingung erfüllt sein:

$$(3-4) \quad x_{i+1} \geq x_i$$

Beispielsweise würde die Zahlenfolge (3, 3, 6, 7, 8) einen Run der Länge 5 bilden.

Für den Run-down-Test gilt entsprechend die Bedingung:

$$(3-5) \quad x_{i+1} < x_i$$

Ein Zufallszahlengenerator ist dabei als umso besser anzusehen, je mehr die Anzahl und Länge der Runs in einer von ihm erzeugten Folge den statistischen Erwartungen entsprechen.

Knuth schlägt in [5] vor, dass man eine Zahl x_{i+1} , die einer einen vorigen Run abschließenden Zahl x_i folgt, nicht betrachtet, sondern den nächsten Run erst bei x_{i+2} beginnen lässt, weil dadurch die Run-Längen voneinander unabhängig sind. Dann hat man die Möglichkeit, mit den Anzahlen für Runs verschiedener Längen und ihren Wahrscheinlichkeiten den in 3.1 beschriebenen χ^2 -Test durchzuführen.

Nach [5] gilt für die Auftrittswahrscheinlichkeit p von Runs der Länge r in einer beliebigen Zahlenfolge:

$$(3-6) \quad p = \frac{r}{(r+1)!}$$

Durch die sinnvolle Kombination von Runs-Test und χ^2 -Test steigt also auch die Qualität der Aussage über die (scheinbare) Zufälligkeit der mit einem Pseudozufallszahlengenerator erzeugten Zahlen.

Bei der Anwendung auf (3-3) zeigt sich schnell die mindere Qualität der mit diesem Algorithmus erzeugten Zahlen, da ausschließlich aufsteigende Runs der Länge 5 auftreten.

4. SICHERHEIT UND SICHERHEITSLÜCKEN

In [6] beschreiben die Autoren, wie sie durch Ausnutzung eines mangelhaften Zufallszahlengenerators bei einem Online-Poker-Spiel in der Lage waren, durch die Kenntnis einiger weniger sichtbarer Karten alle anderen unsichtbaren Karten zuzuordnen zu können, d.h. sie konnten in Erfahrung bringen, welcher Spieler welche Karte auf der Hand hielt.¹

Der Anbieter des Spiels hatte die Beschreibung des Zufallszahlengenerators öffentlich gemacht, damit sich alle Spielteilnehmer vergewissern konnten, dass die Kartenverteilung so gut wie zufällig ist. Das Verfahren hatte jedoch einige elementare Schwächen:

¹ Es handelte sich um die in Europa weniger bekannte Poker-Variante „Texas Hold'em“, bei der die Spieler ihre nichtöffentlichen Karten mit einigen öffentlichen Karten kombinieren können. Eine nähere Beschreibung findet sich in [6].

4.1 Mangelhafte Verteilung

Auf die Einzelheiten des Algorithmus soll hier nicht eingegangen werden (dafür siehe [6]); es ergab sich aus der Art und Weise, wie die 52 Karten eines Blatts gemischt wurden, dass einige Varianten wesentlich häufiger vorkamen. Bei einer Anzahl von $52!$ (etwa 2^{226}) Möglichkeiten, die Karten anzuordnen, mag dies noch nicht dramatisch erscheinen, in Verbindung mit der unter 4.2 beschriebenen Problematik jedoch erwies sich dieser Umstand als fatal.

4.2 Anzahl der erzeugbaren Blattanordnungen

Wie erwähnt, gibt es ca. 2^{226} Möglichkeiten, ein Blatt von 52 Karten anzuordnen. Doch da der verwendete Zufallszahlenalgorithmus, der die Karten anordnete, eine Saat im Bereich von bis zu 2^{32} verwendete, war auf Grund der Determiniertheit der erzeugten Zahlen natürlich auch die Anzahl der möglichen Anordnungen des Blatts auf 2^{32} begrenzt.

Hinzu kam der Umstand, dass im verwendeten Algorithmus als Saat stets die Anzahl der seit Mitternacht verstrichenen Millisekunden gewählt wurde. Ein Tag hat 86.400.000 (zum Vergleich: in etwa 2^{27}) Millisekunden; dadurch wurde die Anzahl der möglichen Blattvariationen weiter reduziert.

Doch die Tatsache, dass die Uhrzeit als Saat verwendet wurde, ließ noch eine stärkere Reduktion der Möglichkeiten zu: Indem Arkin und seine Kollegen ihre Systemzeit mit der Systemzeit des Spielers synchronisierten, konnten sie anhand des Zeitpunkts des Spielstarts für die Anzahl der seit Mitternacht verstrichenen Sekunden einen Wert innerhalb sehr enger Grenzen vermuten – damit beschränkten sie die Anzahl der möglichen Anordnungen auf etwa 200.000. Nun konnten sie mit Kenntnis der öffentlichen Karten alle in Frage kommenden Kombinationen überprüfen und mit Hilfe eines entsprechenden Programms durch stures Durchprobieren feststellen, welche Karten ihre Mitspieler auf der Hand hielten.

5. ZUSAMMENFASSUNG UND FAZIT

Bei der Erzeugung von Pseudozufallszahlen muss stets in Betracht gezogen werden, welchem Anspruch die Zahlen genügen müssen. Ein Einblick in hierfür verfügbare Methoden wurde in Abschnitt 3 gegeben. Obwohl die in Abschnitt 2.1 beschriebenen Algorithmen bei bestimmten in die Variablen eingesetzten Werten sehr gute Ergebnisse erzielen, kann eine unsachgemäße Verwendung jedoch erschreckende Folgen nach sich ziehen, wie in Abschnitt 4 dargelegt wurde.

Obwohl Kongruenzgeneratoren für die meisten Einsatzgebiete mehr als ausreichend sind, könnten sie in einigen Anwendungsbereichen vom 1998 von Matsumoto und Nishimura in [7] beschriebenen „Mersenne Twister“ ersetzt werden. Dieser Algorithmus besitzt gegenüber fast allen bisher bekannten Algorithmen einige entscheidende. Er hat eine extrem große Periode von $2^{19937}-1$. Dieser Wert ist eine Mersenne-Primzahl, d.h. eine Primzahl $M_p = 2^p-1$, wobei p prim ist; daher auch der Name „Mersenne Twister“ für den Generator. „Twister“ bezieht sich darauf, dass die Pseudozufallszahlen durch das Vertauschen (engl. „to twist“) von Bits von intern verwendeten Zahlen erzeugt werden. Eine genauere Beschreibung dieses Algorithmus würden den Rahmen dieses Papers sprengen, nähere Informationen werden in [7] gegeben. Neben der großen Periode weisen mit dem Algorithmus erzeugte Zahlen auch eine sehr gute Gleichverteilung auf, und der Algorithmus ist im Vergleich zu anderen sehr schnell. Zwar lassen sich die Sequenzen mit diversen Tests als nicht-zufällig identifizieren, dennoch ist Nichtvorhersagbarkeit gewährleistet.

6. LITERATUR

- [1] Soht, F. *Verschlüsselung mit dem RSA-Verfahren*. Universität Hamburg, Deutschland, 2005. (In diesem Band.)
- [2] Heppner, C. *Tabu-Search – Übersicht / Einführung in eine moderne Meta-Heuristik*. Universität Hamburg, Deutschland, 2005. (In diesem Band.)
- [3] Gess, J. L. *A Survey of Random Number Generation Theory*. Naval Air Development Center, Warminster, Pennsylvania, USA, 1971.
- [4] Götz, S., Reichel, H.-C., Müller, R., Hanisch, G. *Mathematik-Lehrbuch 6*. öbvhp Verlagsgesellschaft, Wien, 2005.
- [5] Knuth, D. E. *The Art of Computer Programming, 3rd edition, volume 2: Seminumerical Algorithms*. Addison-Wesley, Reading, Massachusetts, USA, 1997.
- [6] Arkin, B., Hill, F., Schmid, M. et. al. *How we Learned to Cheat in Online Poker: A Study in Software Security*. Software Security Group, 1999. <http://itmanagement.earthweb.com/entdev/article.php/616221>, am: 22.01.2006
- [7] Matsumoto, M. und Nishimura, T. Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. In: *ACM Transactions on Modeling and Computer Simulations, Vol. 8, 3–30*, 199