Pseudo-Random Generatoren

Heute: Wie gelangen wir an Zufallsbits für randomisierte Algorithmen?

Möglichkeit A: „Echte" Zufallsbits. Können erzeugt werden aus

→ radioaktiven Zerfallsprozessen

→ atmosphärischem Rauschen

→ thermisches Rauschen eines Widerstands

→ Nutzereingaben (z.B. Zeit zwischen zwei Tastaturanschlägen, Mausbewegungen

→ Intel's RdRand (?)

Probleme: → schwierig und aufwendig große Bitsequenzen zu erzeugen

→ Reproduzierbarkeit eines Zufallsexperiments

Möglichkeit B: Pseudo-Random Generatoren

→ Konstruiere aus einer „zufälligen Saat" eine lange zufällig erscheinende Bitfolge.

Probleme: Bitsequenzen von Pseudo-Random Geratoren können potentiell gut erraten werden. → schlecht für sicherheitsrelevante Anwendungen, aber trotzdem wichtig für sehr viele Algorithmen.

Heute: Was sind „gute" Pseudo-Random Generatoren? Wie sehen diese aus?

Ein Pseudo-Random Generator „streckt" also einen wirklichen Zufallsstring. Der gestreckte String soll sich wie ein echter Zufallsstring verhalten. Insbesondere sollten sich Random und Pseudo-Random Generatoren in effizient durchführbaren Tests nicht unterscheiden.

Def. Ein <u>statistischer Test</u> ist ein randomisierter Algorithmus, der eine Eingabe akzeptiert oder verwirft. Er heißt effizient, wenn er polynomielle worst-case Laufzeit hat.

Def. Sei p ein echt monoton wachsendes Polynom mit $p(n) > n$. Ein <u>Generator</u> $G: \{0,1\}^* \to \{0,1\}^*$ mit Streckung p produziert aus binären Eingaben der Länge n binäre Ausgaben der Länge $p(n)$. G heißt effizient, wenn die Berechnung in polynomieller Zeit erfolgt.

Def. Sei G ein Generator mit Streckung p und T ein stat. Test. Wir setzen

$$g_n = Pr[T \text{ akzeptiert } G(x) \mid |x| = n] \quad \text{und}$$
$$r_n = Pr[T \text{ akzeptiert } y \mid |y| = p(n)].$$

Wir sagen <u>G besteht den Test T</u>, wenn es für jedes $k \in \mathbb{N}$ eine Schranke N_k gibt, so dass

$$\forall n \geq N_k \quad |g_n - r_n| \leq n^{-k}.$$

Wir gehen dabei bei den Wahrscheinlichkeiten von einer Gleichverteilung über alle entsprechenden Inputstrings aus.

<u>Def</u> Ein (effizienter) Generator G heißt ein (effizienter) Pseudo-Random Generator, wenn G jeden effizienten statistischen Test besteht.

Warum beschränken wir uns hier auf effiziente Tests?

→ Es gibt keine Pseudo-Random Generatoren, die alle Tests bestehen:

Sei G ein Generator, der Eingaben der Länge n auf auf Ausgaben der Länge $p(n) > n$ strecke. Wir definieren T mit

$$T(x) = \begin{cases} 1 & G \text{ produziert } x \text{ als Ausgabe} \\ 0 & \text{sonst.} \end{cases}$$

Es ist: $\Pr[G(x) \in T \mid |x| = n] = 1$, während

$\Pr[y \in T \mid |y| = p(n)] \leq \frac{1}{2}$. Wieso? → Hausaufgabe.

<u>Beispiel</u>: Der Generator

$$x_0 = s, \quad x_{k+1} = (a \cdot x_k + b) \bmod m$$

für eine Saat s und Koeffizienten a und b ist <u>kein</u> Pseudo-Random Generator, wird aber häufig benutzt.

(Boyar, 1989)

<u>Satz</u> Sei G ein effizienter Pseudo-Random Generator und q ein beliebiges Polynom, das echt monoton wächst. Dann gibt es ~~wenn~~ einen {effizienten} Pseudo-Random Generator G', der n Bits auf $q(n)$ Bits streckt.

Beweis: Wir nehmen an, G streckt n Bits auf $n+1$ Bits.
Sollte G um mehr strecken, ignoriere die zusätzlichen Bits.
Wir definieren

$$G^*(x) = G^{q(n)-n}(x)$$

G^* erreicht die gewünschte Streckung und ist effizient, falls G effizient ist. Angenommen, G^* ist kein Pseudo-Random Generator. Dann gibt es einen effizienten Test T^*, den G^* nicht besteht. Sei

$$g_n^* = \Pr[T^* \text{ ~~akz~~ akzeptiert } G^*(x) \mid |x| = n]$$

$$r_n = \Pr[T^* \text{ akzeptiert } y \mid |y| = q(n)].$$

Da G^* durchfällt gibt es ein k und unendlich viele n mit

$$|g_n^* - r_n| > \frac{1}{n^k}$$

Wir definieren

$$p_i = \Pr[T^* \text{ akzeptiert } G^{q(n)-(n+i)}(y) \mid |y| = n+i] \quad \forall i: 0 \le i \le q(n)-n$$

$$\Rightarrow p_0 = g_n^* \quad \text{und} \quad p_{q(n)-n} = r_n.$$

$$\Rightarrow \exists i \quad \text{mit} \quad |p_i - p_{i+1}| > \frac{1}{q(n)} \cdot \frac{1}{n^k}.$$

Jetzt definieren wir einen Test T, der G nicht akzeptiert:

- Akzeptiere ein Wort x der Länge $n+i+1$ genau dann, wenn T^* das Wort $G^{q(n)-(n+i+1)}(x)$ akzeptiert.

$$\Rightarrow \Pr[T \text{ akzeptiert } G(y) \mid |y| = n+i] = p_i$$

und $Pr[T \text{ akzeptiert } x \mid |x| = n, i+1] = p_{i+1}$.

und $|p_i - p_{i+1}| > \frac{n^{-k}}{q(n)} > n^{-k'}$ für ausreichend großes k'. □

Satz Wenn $P = NP$, dann gibt es keine effizienten Pseudo-Random Generatoren.

Beweis: Wir konstruieren den folgenden Test:

Die Sprache $L = \{ G(x) \mid x \in \{0,1\}^* \}$ ist in NP:

Für eine Eingabe y, rate x, berechne $G(x)$ und akzeptiere, falls $y = G(x)$.

Da $NP = P$ angenommen wurde, liegt L auch in P.

$\Rightarrow \exists$ Ein Test T, der in Polynomialzeit überprüft, ob $y \in L$ also ob $y = G(x)$ für ein x. Dieser Test akzeptiert die generierten Strings mit Wahrscheinlichkeit 1 und alle Strings höchstens mit Wahrscheinlichkeit $\frac{1}{2}$.

Corrollar: Wenn es einen effizienten Pseudo-Random-Generator gibt, gilt $P \neq NP$.

$\Rightarrow$ Nachweis ist schwierig zu führen.

Es gibt aber aussichtsreiche Kandidaten:

Blum-Micali Generator: Wähle eine Primzahl p mit $p = 2q+1$, q prim und g ein zufälliger Generator der ~~zyklischen~~ Gruppe $\mathbb{Z}_p^*$

Für Saat s_0 sei $s_{i+1} = g^{s_i} \bmod p$

und $b(s_i) = \begin{cases} 1 & s_i < \frac{p}{2} \\ 0 & \text{sonst} \end{cases}$

$(b_1, b_2, \dots, b_e)$ ist Ausgabe $G(s_0)$

Exkurs: Die ~~Gruppe~~ Multiplikative Gruppe modulo n (kurz: $\mathbb{Z}_n^*$)

Def: Die Menge $\mathbb{Z}_n^*$ besteht aus allen Elementen von $\mathbb{Z}_n = \{1, \dots, n-1\}$, die teilerfremd zu n sind.

Wir bezeichnen die Anzahl der Elemente in $\mathbb{Z}_n^*$ mit der eulerschen φ - Funktion $\varphi(n)$.

Also: $$\mathbb{Z}_n^* = \{ k \in \mathbb{Z}_n \mid ggT(k,n) = 1 \}$$

Man kann zeigen: Die Menge $\mathbb{Z}_n^*$ bildet mit der Multiplikation modulo n als Verknüpfung und der 1 als neutralem Element eine abelsche Gruppe.

Bsp: $\mathbb{Z}_7^* = \{1, 2, 3, 4, 5, 6\}$

$\mathbb{Z}_{10}^* = \{1, 3, 7, 9\}$

Fakt: Wenn n eine Primzahl ist, gilt $\varphi(n) = n-1$. Wenn n das Produkt zweier versch. Primzahlen p, q ist, also $n = p \cdot q$, dann gilt
$$\varphi(n) = (p-1)(q-1).$$

Die zyklische Gruppe:

Def: Eine zyklische Gruppe wird von einem Element erzeugt. Das heißt, G ist zyklisch, wenn es ein $a \in G$ gibt mit
$$G = \{ a^i : i \geq 0 \} = \langle a \rangle.$$ a heißt Generator.

Bsp: $\mathbb{Z}_{10}^* = \langle 3 \rangle$

$\mathbb{Z}_7^* = \langle 5 \rangle.$

Satz: Wenn p eine Primzahl ist, ist $\mathbb{Z}_p^*$ zyklisch.

Eine Zahl p heißt starke Primzahl, wenn sie von der Form $p = 2q + 1$ mit q prim ist.

Satz: Sei p eine starke Primzahl und $q = \frac{p-1}{2}$. Dann ist $g \in \mathbb{Z}_p^*$ ein Generator genau dann wenn $g^q \neq 1$ und $g^2 \neq 1$.

Das heißt, zu testen, ob ein gegebenes g ein Generator ist, ist leicht. Aber wie finden wir diese effizient?

Satz: Sei p prim mit $p = 2q+1$, q prim und $1 \leq i \leq 2q$. Falls g ein Generator von $\mathbb{Z}_p^*$ ist und i nicht teilbar von q und 2, dann ist g^i auch ein Generator von $\mathbb{Z}_p^*$.

$\Rightarrow$ Effizientes Finden eines Generators: (Hausaufgabe)

Wähle zufällig $g \in \mathbb{Z}_p^*$. Teste ob $g^q \neq 1$ und $g^2 \neq 1$.

Falls nein, wähle zufällig ein neues g. Fehlerwahrscheinlichkeit ist $\left(\frac{q+1}{2q}\right)^\ell$ bei ℓ Versuchen. ($\approx 2^{-\ell}$ für großes q)