Ausarbeitung zum Seminarvortrag "Probabilistische Algorithmen, Wahrscheinlichkeitsverstärkung und Recycling von Zufallszahlen"

von

Heinz-Günter Kuper (143642)

0. Motivation

Der Zufall spielt eine wichtige Rolle in fast allen Bereichen der Informatik. Randomisierte bzw. probabilistische Algorithmen sind in der Regel schneller und einfacher als der deterministischer Algorithmus für das gleiche Problem. Außerdem sind probabilistische Algorithmen besonders von Nutzen wo eine Wahrscheinlichkeitsverteilung im Spiel ist, z.B. im Falle eines kryptografischen Protokolls, das eine Zufallsprimzahl oder eine Liste von Zufallsprimzahlen verlangt, oder eine Simulation, die versucht eine probabilistische oder einen chaotischen Prozeß (z.B. das Wetter) zu modellieren.

Aber probabilistische Algorithmen benötigen (stochastisch unabhängige und gleichverteilte) Zufallszahlen bzw. –bits (d.h. binäre Zufallsentscheidungen), deren Erzeugung ein aufwendiger Prozeß ist. Will man die Fehlerwahrscheinlichkeit des Algorithmus' zusätzlich minimieren muß man sogar noch mehr Zufallszahlen verwenden. Also versuchen Informatiker die Anzahl von tatsächlichen Zufallsbits zu minimieren, indem sie Pseudozufallsgeneratoren (engl. *pseudo-random generators*) verwenden. Diese funktionieren folgendermaßen: eine kurze Zufallsstartzahl (engl. *random seed*) wird einem deterministischen Prozeß unterzogen, wobei eine längere pseudozufällige Ausgabe erzeugt wird, in der Hoffnung, dass diese längere Ausgabe die Wirkung einer echten Zufallsausgabe nachahmen kann. Das üblichste Verfahren ist die Verwendung eines linearkongruenz Generators (engl. *linear-congruential generator*).

Um die Rechenzeit noch zu verkleinern, wird hier eine Methode vorgestellt, wobei die Zufallszahlen, die in einem probabilistischen Algorithmus schon verwendet wurden, "wiederaufbereitet" (engl. *recycled*) werden zur erneuten Verwendung. Auf diese Weise kann die Zahl der Zufallszahlen, die notwendig sind um eine gewisse Fehlerwahrscheinlichkeit zu erreichen, drastisch reduziert werden.

1. Beispiel eines probabilistischen Algorithmus'

Einer der ersten probabilistischen Algorithmen ist der von Michael Rabin (basiert auf Arbeit von Gary Miller) zum Feststellen in polynomialer Zeit, ob eine gegebene natürliche Zahl prim ist. Der Rabin-Miller Algorithmus funktioniert folgendermaßen:

Gegeben sei (x,n), wobei $n \in$ N der Testkandidat ist und x zufällig aus $[1, n\text{-}1]$ gewählt wird. Sei $n - 1 = 2^q m$, wobei m eine ungerade natürliche Zahl ist. Wenn eine der folgenden Bedingungen

- $x^m \equiv 1$ (mod n) oder
- es existiert eine natürliche Zahl i in $[0, q\text{-}1]$, so dass $x^{m2^i} \equiv -1 (\bmod n)$

gelte, dann ist das Ergebnis unbestimmt, sonst ist n teilbar.

Der Erfolg des Algorithmus' beruht auf der folgenden Eigenschaft: wenn eine dieser Bedingungen für irgendein x gelte, dann ist n teilbar und wenn n schließlich teilbar sei, dann erfüllen mindestens die Hälfte der natürlichen Zahlen zwischen 1 und n eine der Bedingungen.

Der Algorithmus funktioniert wie folgt:

```
INPUT n;
Wähle durch Zufall m natürliche Zahlen, x1, ... , xm,
     wobei 1 ≤ xj < n;
FOR j := 1 TO m DO
     Teste ob eine der Bedingungen gelte für irgendein xj;
     IF ja, THEN ACCEPT END;
END;
REJECT;
END
```

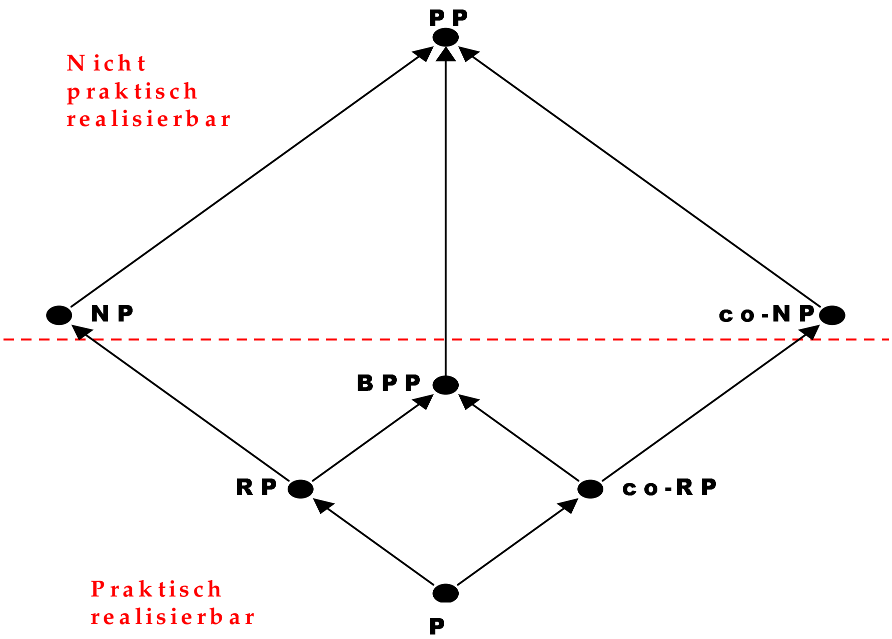
Sollte der Algorithmus akzeptieren, muß n teilbar sein, da ein x gefunden worden ist wofür eine der Bedingungen gilt. Umgekehrt, sollte der Algorithmus die Eingabe verwerfen, dann ist n höchstwahrscheinlich prim, da n nur teilbar ist wenn alle der gewählten x_j keinen Teiler für n feststellen. Hinsichtlich der Wahrscheinlichkeitsverteilung, muß dies der Fall sein für die Hälfte der Zufallszahlen x_j und dadurch ist die Fehlerwahrscheinlichkeit kleiner als $(1/2)^m$.

2. Einführung in die probabilistischen Algorithmen

Ein probabilistischer Algorithmus trifft seine Entscheidungen abhängig vom Zufall. Im Sinne einer Turingmaschine bedeutet dies, dass jede Konfiguration mehrere (aber nur endlich viele) Nachfolgekonfigurationen hat, die aber mit gleicher Wahrscheinlichkeit angenommen werden können. Durch die Einführung der Wahrscheinlichkeit wird sowohl die Entscheidung des Algorithmus' (nämlich: *akzeptiere* oder *verwerfe*, bzw. 1 oder 0) bei einer Eingabe x, als auch die Rechenzeit zu Zufallsvariablen.

M.a.W., ein probabilistischer Algorithmus ist ein deterministisches Verfahren, das allerdings Entscheidungen innerhalb einer festen Wahrscheinlichkeitsverteilung trifft.

Dass die BPP (*bounded-error probabilistic polynomial time*), RP (*randomised polynomial time*) und co-RP Algorithmen praktisch realisierbar sind (d.h. brauchbare Ergebnisse liefern innerhalb eines akteptablen Zeitraums), kann man dem folgenden Inklusionsdiagramm entnehmen.

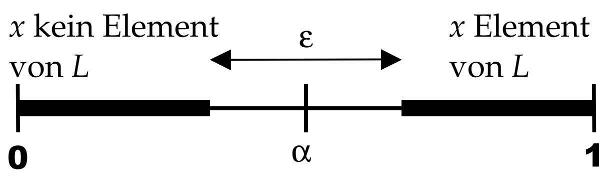


1.1. Arbeitsweise eines probabilistischen Algorithmus'

Wesentlich für die Arbeitsweise eines probabilistischen Algorithmus' ist die Angabe einer rationalen Schwellwert-Wahrscheinlichkeit $\alpha \in [0, 1]$. Ein Wort x gilt als akzeptiert, wenn es die Akzeptierwahrscheinlichket α übersteigt.

Das Konzept der "Wahrscheinlichkeitslücke" zwischen akpzeptierten und verworfenen Wörtern ist wichtig für probabilistische Algorithmen in den BPP, RP und ZPP (*zero error probabilistic polynomial time*) Klassen. Statt zu akzeptieren mit einer Wahrscheinlichkeit $\geq \alpha$ bei $x \in$ L und $< \alpha$ bei $x \notin$ L, verwendet der probabilistische Algorithmus eine Wahrscheinlichkeitslücke $\varepsilon > 0$, so dass Wörter jetzt nur als akzeptiert gelten mit einer Wahrscheinlichkeit $\geq (\alpha + \varepsilon/2)$ bei $x \in L$ und mit einer Wahrscheinlichkeit $\leq (\alpha - \varepsilon/2)$ bei $x \notin L$.

Für die Klasse BPP sieht dies wie folgt aus:



Folglich gehört eine Sprache L zur Klasse RP, falls es einen BPP-Algorithmus für L gibt, wobei für die zugehörigen Konstanten α und ε gilt: $\varepsilon = 2\alpha$. Das heißt, die

Turingmaschine akzeptiert entweder mit einer Wahrscheinlichkeit $\geq \varepsilon$ bei Eingaben $x \in L$ oder mit Wahrscheinlichkeit 0 bei $x \notin L$. Dieser Akzeptier-Situation entspricht einer nichtdeterministischen Turingmaschine.

Eine Sprache L liegt in der Klasse ZPP genau dann, wenn $L \in$ RP und $\overline{L} \in$ RP gilt (d.h. ZPP = RP $\cap$ co-RP).

1.2. Wahrscheinlichkeitsverstärkung und Majoritätsvotum

Ein probabilistischer Algorithmus benutzt also Zufallszahlen, um ein Ergebnis zu liefern, das wiederum von dem Zufall abhängt, jedoch fällt dieses Ergebnis innerhalb einer bestimmten (Fehler-)Wahrscheinlichkeitsverteilung. Man reduziert die Fehlerwahrscheinlichkeit durch ein Majoritätsvotum (d.h. "Mehrheitsverstärkung" bzw. "Wahrscheinlichkeitsverstärkung").

Die Wahrscheinlichkeitsverstärkung bedeutet im Wesentlichen das Vergrößern der Wahrscheinlichkeitslücke, um dadurch eine besseren Fehlerwahrscheinlichkeit zu erreichen. Ein probabilistischer Algorithmus kann mit nur polynomialem Zeitaufwand so umgeformt werden, dass Wörter in der Sprache mit viel höherer Wahrscheinlichkeit akzeptiert werden und Wörter, die nicht zur Sprache gehören, fast nie akzeptiert werden. Der Ansatz ist untershiedlich für RP und BPP Algorithmen.

Eine Wahrscheinlichkeitsverstärkung für RP-Algorithmen ist leichter zu erreichen als für BPP-Algorithmen. Die Ausgabe "akzeptieren" eines RP-Algorithmus' ist immer korrekt und daher sind nur Ausgaben mit "verwerfen" fragwürdig. Läßt man einen RP-Algorithmus mehrere Male auf dieselbe Eingabe durchlaufen, mit jeweils neuen Zufallszahlen (d.h. die Einzelergebnisse des Algorithmus' sind unabhängig), kann das Gesamtergebnis "akzeptieren" lauten wenn mindestens eines der Einzelergebnisse "akzeptieren" lautet. Je mehr Einzelergebnisse, desto größe die Wahrscheinlichkeitsverstärkung.

Die Ausgabe eines BPP-Algorithmus' ist in beiden Fällen (d.h. "akzeptieren" und "verwerfen") zu mißtrauen. Trotzdem liefert jedes Einzelergebnis eine gewissen Evidenz, die sich bei mehreren Versuchen verstärken kann. Sei M ein BPP-Algorithmus mit Schwellenwert α und Wahrscheinlichkeitslücke ε. Der folgende Algorithmus vergrößert die Wahrscheinlichkeitslücke und so erzielt eine Wahrscheinlichkeitsverstärkung:

```
INPUT x;
s := 0;
FOR i := 1 TO t DO
     Simuliere M auf x; das Ergebnis sei y;
     IF y = "akzeptieren" THEN s := s + 1 END;
END;
IF s > αt THEN ACCEPT ELSE REJECT END
```

Dieses Verfahren wird als Bernoulli-Experiment aufgefasst: Es wird t mal unabhängig ein Zufallsexperiment ausgeführt, das eine gewisse Erfolgswahrscheinlichkeit p hat (wobei $p \geq \alpha + \varepsilon/2$, falls $x \in L$ und $p \leq \alpha - \varepsilon/2$, falls $x \notin L$). Die Wahrscheinlichkeit, dass dieser Algorithmus, bei Eingabe $x \in L$ ein falsches Ergebnis (d.h. "verwerfen") berechnet, ist höchstens

$$\sum_{i=0}^{\alpha t} \binom{t}{i} \bullet \gamma^{i} \bullet (1-\gamma)^{t-i}$$

wobei $\gamma = \alpha + \varepsilon/2$.

Wie im einfacheren Fall der Klasse RP ergibt sich somit ein linearer Zusammenhang zwischen t und n, wenn das Ziel eine exponential-kleine Fehlerschranke (2^{-n}) ist. Es gibt also einen BPP-Algorithmus M für jede Sprache $L \in$ BPP mit $\alpha = ½$ und ein Polynom p mit einer Spanne $1 - 2^{-p(n)}$ (wobei n die Länge der Eingabe ist). M.a.W. hat dieser Algorithmus bei jeder Eingabe x eine Fehlerwahrscheinlichkeit kleiner als $2^{-p(|x|)}$.

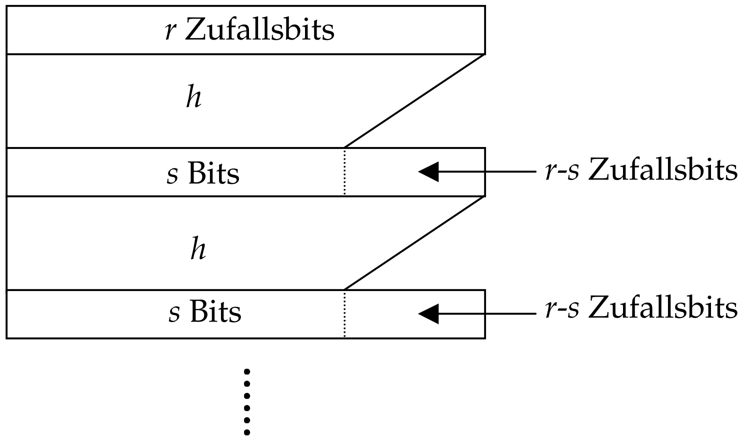
Allgemein ausgedrückt, muß man einen probabilistischen Algorithmus $O(k)$ mal wiederholen um die Wahrscheinlichkeitslücke $\varepsilon > 0$ auf $1 - 2^{-\Omega(k)}$ zu vergrößern und dadurch die Fehlerwahrscheinlichkeit auf $2^{-\Omega(k)}$ zu verkleinern. Angenommen der Algorithmus braucht r Zufallsbits, so braucht man $O(rk)$ Zufallsbits um mit einer Wahrscheinlichkeitsverstärkung zu entscheiden.

3. Das Wiederaufbereiten (*Recycling*) von Zufallszahlen

Da die Erzeugung von Zufallsbits mit einem gewissen Rechenaufwand verbunden ist, ist der Versuch die Fehlerwahrscheinlichkeit von probabilistischen Algorithmen — die ohnehin Zufallsbits für ihre Berechnungen brauchen — durch Wahrscheinlichkeitsverstärkung, dass wiederum mehr Zufallsbits fordert, nicht unproblematisch. Das Wiederaufbereiten von Zufallsbits durch die Methode des universalen Hashverfahrens (engl. *universal hashing*) stellt eine Möglichkeit da, diese Kosten so weit wie möglich zu minimieren.

3.1. Hashfunktionsklassen

Mittels einer geeigneten Hashfunktion h kann man aus r Zufallsbits (nach deren Verwendung) $s < r$ Zufallsbits wiederaufbereiten. Dementsprechend braucht man nur $r - s$ neuen echten Zufallsbits um die vom Algorithmus benötigten Anzahl von Zufallsbits zu erzeugen. Dieser Prozeß kann daraufhin iteriert werden:



Diese r-Bit langen Pseudozufallszahlen (die tatsächlich nur jeweils aus s Pseudozufallsbits und r - s echten, gleichverteilten Zufallsbits bestehen) können dann anstelle der ursprünglichen Folge von k vielen unabhängig gewählten Zufallszahlen verwendet werden. Die Zahl der echten Zufallsbits, die diese Methode verlangt, kann wie folgt abgeschätzt werden:

- r Bits für die erste Zufalls(start)zahl
- jeweils $r - s$ Bits für jede weitere (Pseudo-)Zufallszahl
- $O(r)$ vielen Bits für die Auswahl der Hashfunktion h aus einer universalen Klasse von Hashfunktionen

Insgesamt brauchen wir $r + (r - s) \bullet (k - 1) + O(r)$ viele Zufallsbits. Da $r - s = O(k)$ gewählt werden kann, ergibt sich dann eine Gesamtzahl von $O(r + k^2)$ viele verwendete Zufallsbits, was eine deutliche Verbesserung gegenüber den ursprünglichen $O(rk)$ Zufallsbits ist.

Es wird sich noch zeigen, ob diese Methode ein ähnliches Wahrscheinlichkeitsverhalten zeigt im Vergleich zu den Ergebnissen, die man durch echte Zufallszahlen erwarten kann. Für diese Analyse brauchen wir die folgende Definitionen und das Leftover Hash Lemma.

<u>Definitionen:</u>

1. Eine Klasse H von Hashfunktionen von $\{0, 1\}^r$ nach $\{0, 1\}^s$ ($s < r$) heißt *universal*, falls für alle $x, y \in \{0, 1\}^r$ mit $x \neq y$ gilt:

$$\Pr[h(x) = h(y)] = 1/2^s$$

   Hierbei wird h zufällig und gleichverteilt aus H gewählt. (Die Wahrscheinlichkeit für eine Kollision entspricht also gerade derjenigen Wahrscheinlichkeit, die man erhält, wenn $h(x)$ und $h(y)$ uabhängig und gleichverteilt aus $\{0, 1\}^s$ gezogen werden).

2. Eine Klasse H von Hashfunktionen von $\{0, 1\}^r$ nach $\{0, 1\}^s$ heißt *fast universal*, falls für alle $x, y \in \{0, 1\}^r$ mit $x \neq y$ gilt:

$$\Pr[h(x) = h(y)] \leq 1/2^s + 1/2^r$$

Hierbei wird h zufällig und gleichverteilt aus H gewählt.

Wir spezifizieren eine Hashfunktion der Klasse H durch Angabe zweier Zahlen $a, b \in [0, p\text{-}1]$, wobei p eine Primzahl der Bitlänge $O(r)$ ist und $p \geq 2^r$. Die zugehörige (Linearkongruenz-)Hashfunktion $h = h_{p,a,b}$ wird wie folgt definiert:

$$h_{p,a,b}(x) = ((ax + b) \bmod p) \bmod 2^s$$

D.h. $h(x)$ berechnet die s niederwertigsten Bits der Zahl $(ax + b) \bmod p$. Für beliebige $x \neq y$ und zufällige a, b sind die Werte $((ax + b) \bmod p) \times ((ay + b) \bmod p)$ gleichverteilt in $[0, p\text{-}1] \times [0, p\text{-}1]$, also wäre diese Hashfunktionsklasse universal (angenommen $GF(p)$ ist ein Körper). Die Beschränkung dieser Klasse auf die letzten s Bits bedeutet jedoch, dass diese Klasse fast universal ist.

3.2. Leftover Hash Lemma

Als nächstes untersuchen wir die Wahrscheinlichkeitsverteilungen dieser Hashfunktionsklasse.

Definitionen:

1. Seien D, D' Wahrscheinlichkeitsverteilungen auf einer endlichen Menge S. Wir sagen, D und D' sind sich *ähnlich* (engl. *statistically indistinguishable*) bis auf eine Schranke ε, falls für jede Teilmenge $X \subseteq S$ gilt:

$$|D(X) - D'(X)| \leq \varepsilon$$

Hiebei ist $D(X)$ die Wahrscheinlichkeit für X unter der Verteilung D, also $D(X) = \Sigma_{x\in X} \Pr_D(x)$.

2. Eine Verteilung D heißt *fast gleichverteilt* (engl. *quasi-random*) bis auf die Schranke ε, falls sich D und die Gleichverteilung auf S ähnlich sind bis auf die Schranke ε.

3. Die *Kollisionswahrscheinlichkeit* einer Verteilung D ist die Wahrscheinlichkeit, dass zwei Elemente x und y, die unabhängig unter der Verteilung D gezogen werden, gleich sind.

Die obige Defintionen werden gebraucht, um das folgende Lemma zu beweisen. Dieses Lemma besagt, dass wenn die Funktion $h \in H$ zufällig gewählt wird und das Argument x von h aus einer unbekannten, großen Grundmenge auch zufällig gezogen wird, dann erreicht man eine akzeptable Wahrscheinlichkeitsverteilung für eine (fast) universale Hashfunktionsklasse.

Lemma ("Leftover Hash Lemma"):

> Sei X eine beliebige Teilmenge von $\{0, 1\}^r$ mit $|X| \geq 2^l$. Sei H eine fast universale Klasse von Hashfunktionen von $\{0, 1\}^r$ nach $\{0, 1\}^s$. Es gelte $r \geq l > s$ und es sei $e = (l - s)/2$. Dann ist die Verteilung von $(h, h(x))$ fast gleichverteilt (auf der Menge $H \times \{0, 1\}^s$) bis auf die Schranke $1/2^e$.

Der Beweis des Lemmas erfolgt in zwei Teilen. Zuerst wird gezeigt, dass die Kollisionswahrscheinlichkeit der Verteilung $(h, h(x))$ fast die der Verteilung auf $H \times \{0, 1\}^s$ gleichkommt. Als nächstes wird gezeigt, wenn eine Verteilung auf einer Menge S eine Kollisionswahrscheinlichkeit hätte, die fast $1/|S|$ entspricht, dann wäre die Verteilung fast gleichverteilt auf S.

1. Teil:

Die Kollisionswahrscheinlichkeit der Verteilung $(h, h(x))$ ist höchstens
$(1 + 2/2^e)/(|H|2^s)$.

Beweis vom 1. Teil:

Seien h_1, $h_2 \in H$ und x_1, $x_2 \in X$ jeweils zufällig unter Gleichverteilung und unabhängig gewählt. Wir schätzen die Wahrscheinlichkeit

$$\Pr[(h_1, h_1(x_1)) = (h_2, h_2(x_2))] \Rightarrow \Pr[h_1 = h_2 \wedge h_1(x_1) = h_2(x_2)]$$

nach oben. Dies ist gerade

$$\frac{1}{|H|} \bullet \Pr[h(x_1) = h(x_2)]$$

wobei $h \in H$ und x_1, $x_2 \in X$ zufällig gewählt werden. Falls $x_1 \neq x_2$, so ist diese Wahrscheinlichkeit (wegen der Definition von fast universal) höchstens

$$1/2^s + 1/2^r$$

Da $|X| \geq 2^l$, ist die Wahrscheinlichkeit für $x_1 = x_2$ höchstens $1/2^l$. Damit kann die gesuchte Wahrscheinlichkeit nach oben abgeschätzt werden durch

$$\frac{1}{|H|} \bullet (1/2^s + 1/2^r + 1/2^l) \leq (1 + 2/2^{2e}) \bullet \frac{1}{|H|2^s}$$

2. Teil:

Sei D eine Verteilung auf einer endlichen Menge S. Wenn die Kollisionswahrscheinlichkeit von D höchstens $(1 + 2\delta^2)/|S|$ ist, dann ist D fast gleichverteilt auf S mit Schranke δ.

Beweis vom 2. Teil:

Der Beweis erfolgt durch Widerspruch. Angenommen, D ist nicht fast gleichverteilt mit Schranke δ. Dann gibt es eine Teilmenge $Y \subseteq S$ mit

$$D(Y) > \frac{|Y|}{|S|} + \delta$$

Sei $\beta > \delta$, so dass

$$D(Y) = \frac{|Y|}{|S|} + \beta$$

Die Kollisionswahrscheinlichkeit von D ist bei zufälliger und unabhängiger Wahl von d_1, d_2 gemäß D durch $\Pr[d_1 = d_2]$ gegeben. Wir zerlegen den Wahrscheinlichkeitsraum anhand von Y:

- $\Pr[d_1 = d_2] \geq \frac{1}{|Y|}, \quad d_1, d_2 \in Y$
- $\Pr[d_1 = d_2] \geq \frac{1}{|S| - |Y|}, \quad d_1, d_2 \notin Y$

Nach dem Einsezten von $|Y|/|S| + \beta$ für $D(Y)$ und Vereinfachung, erhält man eine Kollisionswahrscheinlichkeit von mindestens

$$\frac{D(Y)^2}{|Y|} + \frac{(1 - D(Y))^2}{|S| - |Y|} = \frac{1}{|S|} + \frac{\beta^2}{|Y|} + \frac{\beta^2}{|S| - |Y|}$$

Dieser Ausdruck wird minimal für $|Y| = |S|/2$. Für diesen Fall ergibt sich eine Kollisionswahrscheinlichkeit von mindestens

$$\frac{1 + 4\beta^2}{|S|} > \frac{1 + 4\delta^2}{|S|}$$

was in Widerspruch zur Voraussetzung steht.

Die Behauptung des Leftover Hash Lemmas folgt aus den 1. und 2. Teilen, was zu zeigen war.

□

3.3. Wahrscheinlichkeitsverhalten

Nun wenden wir die oben gewonnene Erkenntnisse an um das Wahrscheinlichkeitsverhalten des universalen Hashverfahrens zu untersuchen.

Sei M eine Turingmaschine, die einen BPP-Algorithmus für die Sprache L implementiert. M braucht jeweils r Zufallsbits pro n-Bit langen Eingabe x. Die Fehlerwahrscheinlichkeit ist wie folgt:

$$x \in L \Rightarrow \begin{cases} 2/3 \leq \Pr[M(x)=1] \leq 3/4 \\ 1/4 \leq \Pr[M(x)=0] \leq 1/3 \end{cases}$$

$$x \notin L \Rightarrow \begin{cases} 1/4 \leq \Pr[M(x)=1] \leq 1/3 \\ 2/3 \leq \Pr[M(x)=0] \leq 3/4 \end{cases}$$

Sei M' eine Turingmaschine, die einen Algorithmus mittels des oben beschriebenen Hashverfahrens implementiert. M' erzeugt Pseudozufallszahlen $x_1, x_2, \ldots, x_k$ (dabei ist x_1 eine echte Zufallszahl, sogar die Zufallsstartzahl), die als Zufallsbits bei k Simulationen von M benutzt werden, wobei M' schließlich nach der Majorität der erhaltenen Antworten entscheidet.

Die Bitfolge $x_1x_2...x_k$, die durch die Konkatenation der Pseudozufallszahlen erhalten wird, ist nicht (fast) gleichverteilt in $\{0, 1\}^{rk}$, da nur $O(r + k^2)$ viele echte Zufallsbits für ihre Herstellung verwendet wurden. M.a.W. die meisten 0-1-Folgen der Länge rk können gar nicht auftreten.

Von Hauptinteresse ist das Wahrscheinlichkeitsverhalten von M zu simulieren und die von M' berechnete Ergebnisse zu untersuchen. Dies tun wir durch Betrachtung der Verteilung der Bitfolge $b(x_1)b(x_2)...b(x_k)$, wobei

$$b(x_i) = \begin{cases} 1, M \; bei \; Zufallszahl \; x_i \; akzeptiert \\ 0, M \; bei \; Zufallszahl \; x_i \; verwirft \end{cases}$$

Die Verteilung der so erhaltenen Bitfolge muß der 'idealen' Verteilung von M annähern, die man erhalten würde hätte man M k-mal auf derselben Eingabe mit neuen Zufallszahlen für jede Wiederholung simuliert und die Ergebnisse als Bitfolge $b_1b_2...b_k$ notierte.

Sei o.B.d.A. die betrachtete Eingabe Element von L. D.h.

$$\frac{2}{3} \leq \Pr[b_i = 1] = p \leq \frac{3}{4}$$

Weiterhin sei ε eine kleine Zahl, so dass sich die echte Folge $b_1b_2...b_k$ und die simulierte Folge $b(x_1)b(x_2)...b(x_k)$ bis auf die Schranke ε ähneln. Das Majoritätsvotumverfahren hat eine Fehlerwahrscheinlichkeit von $2^{-\Omega(k)}$ beim Verwenden von echten Zufallszahlen. Benutzt man Pseudozufallszahlen, muß die Fehlerschranke $\varepsilon = 2^{-\Omega(k)}$ sein um eine Fehlerwahrscheinlichkeit von $2^{-\Omega(k)}$ zu erreichen. Dies wird wie folgt begründet:

Sei X die Menge der 0-1-Folgen der Länge k mit maximal $k/2$ Einsen in der Folge. Sei D eine Verteilung, die der (p, 1-p) Verteilung der Länge k ähnlich ist bis auf die Schranke ε und sei E die ideale (p, 1-p) Verteilung. Wir schätzen ab:

$$\begin{aligned}\Pr_D(X) &\leq \Pr_E(X) + \varepsilon \\ &= \sum_{i=0}^{k/2} \binom{k}{i} p^i (1-p)^{k-i} + \varepsilon \\ &\leq 2^{-\Omega(k)} + \varepsilon\end{aligned}$$

Somit muß $\varepsilon = 2^{-\Omega(k)}$ gelten.

<u>Behauptung:</u>

Für alle $i \geq 0$ ähnelt die Verteilung von $b(x_1)$ ... $b(x_i)hx_{i+1}$ (wobei x_j die "Pseudozufallszahlen" sind) der folgenden (idealen) Verteilung bis auf eine Schranke ε_i:

- Die ersten i Bits sind jeweils (unabhängig voneinander) mit Wahrscheinlichkeit p als Eins und mit Wahrscheinlichkeit $1 - p$ als Null gewählt;
- h wird unter Gleichverteilung aus H gezogen;
- dann folgt ein Zufallswort, das unter Gleichverteilung aus $\{0, 1\}^r$ gezogen wird.

D.h.

$$\underbrace{b_1 b_2 b_3 \ldots\ldots b_i}_{\substack{\Pr[b_i=1]=p, \\ \Pr[b_i=0]=1-p}} \underbrace{h}_{\in H} \underbrace{x}_{\in \{0,1\}^r}$$

Der Beweis dieser Behauptung folgt per Induktion.

<u>Beweis:</u>

<u>1.Fall: $i = 0$</u>

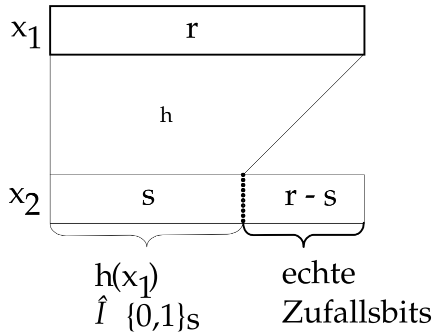
Wir betrachten die Verteilung hx_1. Da h gleichverteilt aus H gewählt wird und x_1 die echte Zufallsstartzahl ist, haben wir hier mit einer echten Gleichverteilung zu tun. Also ist $\varepsilon_0 = 0$.

<u>2.Fall: $i = 1$</u>

Dieser Fall ist sehr lehrreich für den Induktionsschritt.
Sei x_1 zufällig aus $\{0, 1\}^r$ gezogen. Um zu zeigen, dass die Verteilung von $b(x_1)hx_2$ einer idealen Verteilung bis auf ε_1 ähnlich ist, genügt es zu zeigen, dass $b(x_1)hh(x_1)$ einer Verteilung ähnelt, wo

- das erste Bit gemäß der (p, $1 - p$) Verteilung gezogen wird
- sowie h zufällig aus H

- und die letzten s Bits zufällig (jeweils unabhängig voneinander) aus $\{0, 1\}^s$ gezogen werden.



Da x_1 zufällig ist, ist $b(x_1)$ gemäß $(p, 1-p)$ verteilt. Unter der Bedingung $b(x_1) = a$, $a \in \{0, 1\}$, müssen wir zeigen, dass $hh(x_1)$ annähernd auf $H \times \{0, 1\}^s$ gleichverteilt ist. Wir bedienen uns des Leftover Hash Lemmas.

Unter der gegebenen Bedingungen stehen uns noch $p \bullet 2^r$ bzw. $(1-p) \bullet 2^r$ – also mindestens $2^r/4$ – Wahlmöglichkeiten zur Verfügung. Die Voraussetzung des Lemmas ist erfüllt durch setzen von $l = r - 2$. Aus dem Lemma ergibt sich die Schranke

$$\begin{aligned} \varepsilon_1 &= 2^{-e} \\ &= 2^{-(l-s)/2} \\ &= 2^{-(r-s-2)/2} \end{aligned}$$

<u>Induktionsschritt</u>:

Nach Voraussetzung gelte die Behauptung für $i - 1$, d.h. die Verteilung von $b(x_1)\ldots b(x_{i-1})hx_i$ ähnelt einer idealen Verteilung bis auf eine Schranke ε_{i-1}. Der Beweis der Induktionsbehauptung verlangt den Vergleich der Verteilung von $b(x_1)\ldots b(x_i)hx_{i+1}$ mit der idealen Verteilung $b_1\ldots b_i hz$.

Es genügt zu zeigen, dass $b(x_1)\ldots b(x_i)hh(x_i)$ der Verteilung von $b_1\ldots b_i hv$ ähnelt, wobei

- die b_j unabhängig $(p, 1-p)$ verteilt sind,
- h gleichverteilt aus H
- und v gleichverteilt aus $\{0, 1\}^s$ gewählt werden.

Wir brauchen zusätzlich die folgende Behauptung:
Seien F, G, und H Zufallsvariablen, so dass die Verteilung von F derjenigen von G bis auf eine Schranke δ_1 ähnelt. Ferner sei t eine Transformation, so dass die Verteilung von $t(G)$ derjenigen von H bis auf eine Schranke δ_2 ähnelt. Dann ähneln sich H und $t(F)$ bis auf die Schranke $\delta_1 + \delta_2$:

$$\begin{tikzcd}
t(F) \arrow[r, dash, dashed, "\delta_1+\delta_2"] & H \arrow[r, dash, dashed, "\delta_2"] & t(G) \\
F \arrow[u] \arrow[rr, dash, dashed, "\delta_1"'] & & G \arrow[u]
\end{tikzcd}$$

Wir setzen nun

$F = b(x_1)...b(x_{i-1})hx_i$	(simulierte Verteilung)
$G = b_1...b_{i-1}hz$	(ideale Verteilung)
$H = b_1...b_{i-1}b_ihv$	(ideale Verteilung)

und definieren die Transformation t durch

$$t(b_1...b_{i-1}hz) = b_1...b_{i-1}b(z)hh(z)$$

Dann ergibt sich

$$t(G) = b_1...b_{i-1}b(v)hh(v)$$
$$t(F) = b(x_1)...b(x_{i-1})b(x_i)hh(x_i)$$

F und G sind sich nach Induktionsvoraussetzung bis auf die Schranke ε_{i-1} ähnlich. Ferner sind sich $t(G)$ und H wie im Fall $i = 1$ bis auf die Schranke ε_1 ähnlich. Es folgt, dass sich H und $t(F)$ bis auf die Schranke $\varepsilon_{i-1} + \varepsilon_1$ ähnlich sind:

$$\begin{tikzcd}
t(F) \arrow[r, dash, dashed, "\varepsilon_{i-1}+\varepsilon_1"] & H \arrow[r, dash, dashed, "\varepsilon_1"] & t(G) \\
F \arrow[u] \arrow[rr, dash, dashed, "\varepsilon_{i-1}"'] & & G \arrow[u]
\end{tikzcd}$$

Setzen wir

$$\varepsilon_i = \varepsilon_{i-1} + \varepsilon_1 = i\varepsilon_1$$

dann verlangen wir wieder zu unserer Induktionsbehauptung. Aus der Behauptung mit $i = k$ folgt, dass die Bits $b(x_1), \ldots , b(x_k)$ einer Verteilung, die jedes Bit mit Wahrscheinlichkeit p auf 1 und mit Wahrscheinlichket $1 - p$ auf 0 setzt, bis auf eine Schranke

$$k\varepsilon_1 = \frac{k}{2^{(r-s-2)/2}}$$

ähnelt.

Wir legen nun s fest, so dass

$$\frac{(r-s-2)}{2} = k \Rightarrow s = r - 2k - 2$$

Dann erhalten wir die Schranke

$$\frac{k}{2^k} = 2^{-\Omega(k)},$$

die genügt um die gewünschte Wahrscheinlichkeitsverstärkung zu erreichen.

□

4. Literaturhinweise

Balcázar, J.L.; Díaz, J.; Gabarró, J.
Structural Complexity I
Springer-Verlag: Berlin, 1988.

Higgins, Brian C.
The Rabin-Miller Probabilistic Primality Test: Some Results on the Number of Non-witnesses to Compositeness
Proceedings of the annual Spring Meeting of the Allegheny Mountain Section of the Mathematical Association of America (ALLEMO), Vol. 1, Spring 1996.
http://www.ma.iup.edu/MAA/proceedings/vol1/higgins.pdf

Impagliazzo, R.; Zuckerman, D.
How to Recycle Random Bits
Symposium on Foundations of Computer Science, IEEE 1989.
http://www.cs.utexas.edu/users/diz/recycle.ps

Schöning, U.
Algorithmen – kurz gefasst
Spektrum Akademischer Verlag: Heidelberg, 1997.

Schöning, U.
Perlen der Theoretischen Informatik
B.I.-Wissenschaftsverlag: Mannheim, 1995.