

Humboldt-Universität zu Berlin

Seminarausarbeitung



Der Nisan-Wigderson-Generator

im Seminar Komplexitätstheorie
bei Prof. Dr. Köbler
am Lehrstuhl Komplexität und Kryptografie
Betreuer: Sebastian Kuhnert

Nils Alberti

Inhaltsverzeichnis

| | | |
|----------|--------------------------------------|-----------|
| 1 | Einleitung | 3 |
| 2 | Grundlagen | 5 |
| 2.1 | Pseudozufall | 5 |
| 2.2 | Pseudo-Random-Generator | 5 |
| 2.3 | Harte Funktionen | 5 |
| 2.4 | Kombinatorische Designs | 7 |
| 3 | Der Nisan-Wigderson Generator | 9 |
| 4 | Fazit | 11 |

1 Einleitung

Bei der Verwendung von Zufallsbits stellt sich das Problem, dass im Computer rein deterministische Operationen ablaufen; zufällige Bits müssen also von außen zugeführt werden und können dann deterministisch verarbeitet werden. Da dies sehr aufwendig ist und u. a. in der Kryptographie sehr viele Zufallsbits benötigt werden, kam die Idee auf, aus einer kleinen Menge echter Zufallsbits deterministisch eine viel größere Anzahl zu generieren, die dann zwar nicht mehr zufällig, dafür aber nach bestimmten Kriterien von echten Zufallsbits nicht zu unterscheiden sind. Damit war die Idee der Pseudo-Zufalls-Generatoren (Pseudo-Random-Generator (PRG)) geboren.

In der Komplexitätstheorie werden mathematische Probleme anhand ihrer Eigenschaften in Klassen eingeteilt. Sehr bekannt sind die Klassen \mathcal{P} , für die Probleme, die in polynomieller Zeit auf einer deterministischen Turingmaschine gelöst werden können und \mathcal{NP} , für die Probleme, die auf einer Nichtdeterministischen Turingmaschine in polynomieller Zeit gelöst werden können. Es ist nicht bekannt, ob $\mathcal{P} = \mathcal{NP}$ gilt. Allgemein wird angenommen, dass $\mathcal{P} \neq \mathcal{NP}$ ist. Es gibt neben \mathcal{P} und \mathcal{NP} eine große Anzahl weiterer Klassen unter anderem randomisierte Komplexitätsklassen: Die bekanntesten sind zum einen die Klasse \mathcal{BPP} , die alle Probleme beinhaltet, die in polynomieller Zeit durch eine randomisierte Turingmaschine, die in höchstens $1/3$ der Fälle das falsche Ergebnis ausgibt gelöst werden können. Zum anderen die Klassen \mathcal{RP} , bzw. $\mathcal{CO} - \mathcal{RP}$, die alle Probleme beinhalten, die durch eine randomisierte Turingmaschine in polynomieller Zeit gelöst werden können, bei nur einem einseitigen Fehler. D. h. wenn bei \mathcal{RP} für eine Eingabe die richtige Ausgabe *false* wäre, dann wird diese auch ausgegeben, wenn sie aber *true* lautet, dann wird *true* nur mit einer konstanten Wahrscheinlichkeit ausgegeben.

Andrew Yao zeigte, dass PRG zur Derandomisierung von propabilistischen Komplexitätsklassen verwendet werden können. [Yao, 1982] Die Idee ist die folgende: Wenn es einen PRG gibt, der aus einer kleinen Bitfolge von echten Zufallsbits, dem sog. seed, eine Bitfolge exponentieller Länger erzeugen kann, die in polynomieller Zeit nicht von echten Zufallsbits unterschieden werden kann, dann könnte man die erzeugten Bits für einen randomisierten, polynomiellen Algorithmus verwenden. Da der Algorithmus sie wegen seiner beschränkten Laufzeit nicht von echten Zufallsbits unterscheiden kann, können seine Berechnungen auch nicht grundlegend anders sein wie bei der Verwendung von solchen.

Da die Bitfolge, die als Eingabe für den Algorithmus dient auf exponentielle

Länge „gestreckt“ ist, es also nur polynomiell viele Eingaben gibt, kann jede mögliche Bitfolge als Eingabe durchprobiert und am Ende eine Majoritätsentscheidung getroffen werden. Die Laufzeit des Algorithmus bleibt polynomiell. Somit wäre der Zufallsanteil im Algorithmus eliminiert und es wäre ein rein deterministischer Algorithmus, der asymptotisch nicht mehr Zeit benötigt und immer eine richtige Entscheidung trifft.

Die Forschung zu PRGs war lange auch auf den Anwendungszweck fokussiert, also die schnelle Erzeugung möglichst vieler Zufallsbits. So beinhaltete die ursprüngliche Definition von PRGs, dass die Laufzeit polynomiell in der Länge der „echten“ Zufallsbits beschränkt ist. Es bedurfte der bahnbrechenden Idee von Nisan und Wigderson, nicht mehr auf kryptographische PRGs zurückzugreifen, sondern einen PRG für genau den Zweck der Derandomisierung von Komplexitätsklassen zu designen, um hier einen Durchbruch zu erzielen. [Nisan and Wigderson, 1994]

2 Grundlagen

2.1 Pseudozufall

Definition: Eine Verteilung R über $\{0, 1\}^m$ heißt (S, ϵ) -pseudozufällig, mit $S \in \mathcal{N}$ und $\epsilon > 0$, falls für jeden Schaltkreis C der Größe höchstens S gilt:

$$|Pr[C(R) = 1] - Pr[C(U_m) = 1]| < \epsilon$$

mit U_m für die Gleichverteilung über $\{0, 1\}^m$.

Mit anderen Worten, ein Bitstring der Länge m ist (S, ϵ) -pseudozufällig, wenn jeder Schaltkreis der Größe S diesen String von einem echten Zufallstring nur mit einer Wahrscheinlichkeit kleiner ϵ unterscheiden kann.

2.2 Pseudo-Random-Generator

Nisan und Wigderson definieren einen Pseudo-Random-Generator viel weitergefasst, als es beispielsweise Yao [Yao, 1982] tat. Sie gestehen ihm nicht nur exponentielle Laufzeit zu, sondern konzipieren ihn auch so allgemein, dass er sich auch für ganz andere Komplexitätsklassen wie RAC^0 , die Klasse, die alle randomisierten Schaltkreise konstanter Tiefe beinhaltet, eignet.

Definition: Sei $S : \mathcal{N} \rightarrow \mathcal{N}$ eine in polynomieller Zeit berechenbare, monotone Funktion. Dann heißt eine Funktion $G : \{0, 1\}^* \rightarrow \{0, 1\}^*$ $S(l)$ -Pseudozufallsgenerator, wenn gilt:

Für jedes $z \in \{0, 1\}^l$ gilt, $|G(z)| = S(l)$ und $G(z)$ kann in Zeit 2^{cl} für eine Konstante c berechnet werden.

Für jedes $l \in \mathcal{N}$ gilt, $G(U_l)$ ist eine $(S(l)^3, 1/10)$ pseudozufällige Verteilung (die Wahl der Konstanten ist beliebig).

Wenn es einen 2^{cl} -PRG für eine beliebige Konstante $\epsilon > 0$ gibt, dann gilt $\mathcal{P} = \mathcal{BPP}$.

2.3 Harte Funktionen

Zur Konstruktion eines Pseudozufallsgenerators benötigen wir eine Funktion, die wir in polynomieller Zeit berechnen können, aber aus deren Ausgabe die ursprüngliche Eingabe nicht in polynomieller Zeit zurückberechnet werden kann. Wenn wir eine

solche Funktion kennen, dann können wir mit ihrer Hilfe echte Zufallsbits strecken und wissen, dass das Ergebnis nicht in polynomieller Zeit von echten Zufallsbits unterschieden werden kann.

Solche Funktionen sind z. B. aus der Kryptographie bekannt, wo die Sicherheit asymmetrischer Verschlüsselungsverfahren gerade darauf beruht, dass die Funktion schnell berechnet werden kann, für die Rückrichtung aber kein effizientes Verfahren bekannt ist. Die Multiplikation ist ein Beispiel, während die Berechnung eines Produkts zweier Zahlen in $O(n^2)$ möglich ist, ist kein Algorithmus bekannt, der in polynomieller Laufzeit eine gegebenen Zahl faktorisiert, d. h. ihre zwei Teiler ermittelt. Allerdings ist bisher nicht der Beweis gelungen, dass auch wirklich solche „harten“ Funktionen existieren.

Definition(worst-case Hardness): Sei $f : \{0, 1\}^* \rightarrow \{0, 1\}$ eine boolesche Funktion. Sei $H_{wrs}(f)$ eine Funktion, die jede Zahl $n \in \mathcal{N}$ auf die größte Zahl $S \in \mathcal{N}$ abbildet für die gilt, dass jeder boolesche Schaltkreis der Größe höchstens S für mindestens eine Eingabe aus $\{0, 1\}^n$ das falsche Ergebnis liefert. $H_{wrs}(f)$ heißt die worst-case Härte von f .

Definition(average-case Hardness): Die average-case Härte ist eine Funktion von \mathcal{N} nach \mathcal{N} , die jedes $n \in \mathcal{N}$ auf die größte Zahl S abbildet, so dass für jeden booleschen Schaltkreis C der Größe höchstens S gilt:

$$Pr[C(x) = f(x)] < \frac{1}{2} + \frac{1}{S}$$

mit $x \in U_m$.

Wir nutzen später die average-case hardness für den Beweis, dass der Nisan-Wigderson Generator ein Pseudozufallsgenerator ist, sofern eine harte Funktion existiert.

Wir wissen jetzt, unter welcher Voraussetzung wir aus Zufallsbits Pseudozufallsbits erzeugen können, die nicht in polynomieller Zeit als solche erkannt werden können. Wir benötigen jetzt noch einen Weg um aus dem seed eine exponentielle Anzahl von Bits zu generieren.

2.4 Kombinatorische Designs

Kryptographische Pseudozufallsgeneratoren arbeiteten in der Regel nach dem Prinzip, dass das Bit x_{i+1} das Ergebnis einer Operation auf x_i ist. Wir wollen aus dem seed eine größere Anzahl an zufällig aussehenden Bits erzeugen indem wir für ein Ausgabebit auf Teilmengen des seeds eine harte Funktion anwenden. Da wir deterministisch arbeiten, können zwei Bits der Ausgabe nicht von den selben Eingabe-Bits abhängen. Wir müssen also einen Weg finden, um aus dem seed eine potentiell exponentielle Anzahl von Teilmengen zu generieren, auf die wir dann jeweils eine harte Funktion anwenden können. Wir benötigen deshalb eine andere Vorgehensweise.

Die Idee von Nisan und Wigderson war die Verwendung von kombinatorischen Designs. Die Idee ist die, aus einer Grundmenge eine (größtmögliche) Anzahl an Teilmengen einer festen Größe zu generieren, die sich jeweils nur in einer beschränkten Anzahl von Elementen unterscheiden.

Definition: Sei d die maximale Anzahl an Überschneidungen, l die Länge des seeds und n die Größe der gewählten Mengen. Dann heißt eine Familie von Mengen $I = \{I_1, \dots, I_m\}$ mit $I_i \in \{1, \dots, l\}$ ein (l, n, d) -Design, falls gilt

- (i) $\forall i : |I_i| = n$
- (ii) $\forall i \neq j : |I_i \cap I_j| \leq d$

Beispiel: Für die Grundmenge $\{1, 2, 3, 4, 5\}$ ist ein $(5, 3, 2)$ -Design gegeben durch das Mengensystem: $\{\{1, 2, 3\}, \{1, 2, 4\}, \{1, 2, 5\}, \{1, 3, 4\}, \{1, 3, 5\}, \{1, 4, 5\}, \{2, 3, 4\}, \{2, 3, 5\}, \{2, 4, 5\}, \{3, 4, 5\}\}$. Ein $(5, 3, 1)$ -Design wäre z. B. gegeben durch $\{\{1, 2, 3\}, \{1, 4, 5\}\}$ oder $\{\{1, 2, 4\}, \{3, 4, 5\}\}$.

Eine greedy-Strategie reicht aus, um eine exponentielle Anzahl an Teilmengen zu generieren: Wir starten mit der leeren Menge und solange wir noch nicht eine ausreichende Anzahl an Mengen haben durchsuchen wir alle n -Teilmengen des seeds und nehmen eine Menge I auf, wenn sie die Design-Bedingung $|I \cap I_j| \leq d$ für jedes $j \in \{1, \dots, m\}$ erfüllt.

Satz: Diese greedy-Strategie berechnet in Zeit $O(2^{O(l)})$ aus einem seed l mit $l > \frac{10 \cdot n^2}{d}$ mit $n > d$ ein (l, n, d) -Design I mit $2^{d/10}$ Mengen.

Beweis: Die Laufzeit ist $\text{poly}(m) \cdot O(2^l) = O(2^{O(l)})$, also wie gefordert polynomiell

in der Länge der Ausgabe. Wir müssen also nur noch zeigen, dass falls $l = \frac{10 \cdot n^2}{d}$ gilt und eine Familie von weniger als $2^{d/10}$ n -elementiger Mengen durch den Algorithmus erzeugt wurden, dass dann eine Menge $I \in [l]$ der Größe n existiert, für die gilt $|I_i \cap I_j| \leq d$. Mit anderen Worten, es lässt sich solange noch nicht $2^{d/10}$ n -elementige Mengen erzeugt wurden, immer noch mindestens eine hinzufügen.

Wir zeigen dies, indem wir I zufällig wählen. Jedes Element $x \in [l]$ wird mit Wahrscheinlichkeit $2 \cdot n/l$ zu I hinzugefügt. Da die erwartete Größe von I $2n$ ist und die der Schnittmenge von I und I_j $2n^2/l < d/5$ erhalten wir mit der Chernoff-Schranke

$$\begin{aligned} Pr[|I| \geq n] &\geq \frac{9}{10} \\ \forall j \in [m] : Pr[|I \cap I_j| \geq d] &\leq \frac{1}{2^{1-d/10}} \end{aligned}$$

Da es insgesamt höchstens $m \leq 2^{d/10}$ Mengen gibt können wir folgern, dass die Wahrscheinlichkeit, dass I mindestens n Elemente enthält und höchstens d Übereinstimmungen mit einer anderen Teilmenge hat größer als 0.4 ist.

□

Kombinatorische Designs ermöglichen es also, aus einer Grundmenge eine exponentielle Anzahl von Teilmengen zu generieren, die sich nur in höchstens k Bits überschneiden.

3 Der Nisan-Wigderson Generator

In diesem Abschnitt wollen wir nun alles zusammenführen und die Konstruktion des NW Generators darstellen. Für den Beweis, dass er auch ein Pseudozufallsgenerator ist, benötigen wir noch ein Theorem von Yao [Yao, 1982].

Theorem (Yao): Sei Y eine Verteilung über $\{0, 1\}^m$. Angenommen es existiert ein $S > 10 \cdot n$ und ein $\epsilon > 0$, sodass für jeden Schaltkreis C der Größe höchstens $2 \cdot S$ und $i \in \{1, \dots, m\}$

$$Pr_{r \in Y}[C(r_1, \dots, r_{i-1}) = r_i] \leq \frac{1}{2} + \frac{\epsilon}{m}$$

gilt, dann ist Y (S, ϵ) -pseudozufällig.

Definition: (NW Generator): Sei $\mathcal{I} = \{I_1, \dots, I_m\}$ eine Familie von Teilmengen des seeds mit $|I_j| = n$ für jedes j und sei $f : \{0, 1\}^n \rightarrow \{0, 1\}$. Der (\mathcal{I}, f) -NW Generator ist die Funktion $NW_{\mathcal{I}}^f : \{0, 1\}^l \rightarrow \{0, 1\}^m$, die jedes $z \in \{0, 1\}^l$ auf

$$NW_{\mathcal{I}}^f(z) = f(z_{I_1}) \circ f(z_{I_2}) \circ \dots \circ f(z_{I_m})$$

abbildet. z_{I_i} bezeichnet die Einschränkung des seeds auf die Koordinaten von I_i .

Theorem (Nisan-Wigderson): Falls \mathcal{I} ein (l, n, d) -Design ist mit $|I| = 2^{d/10}$ und für $f : \{0, 1\}^n \rightarrow \{0, 1\}$ gilt $H_{avg}(f) > 2^{2^d}$, dann ist die Verteilung $NW_{\mathcal{I}}^f(U_l)$ $(\frac{H_{avg}(f)}{10}, \frac{1}{10})$ -pseudozufällig.

Beweis: Sei $S = H_{avg}(f)$. Mit dem Theorem von Yao reicht es aus zu zeigen, dass für jedes $i \in [2^{d/10}]$ kein Schaltkreis C der Größe $S/2$ existiert für den

$$Pr_{R=NW_{\mathcal{I}}^f(Z \sim U_l)}[C(R_1, \dots, R_{i-1}) = R_i] \geq \frac{1}{2} + \frac{1}{10 \cdot 2^{d/10}}$$

gilt.

Nehmen wir das Gegenteil an, also, dass dies für einen beliebigen Schaltkreis C und ein beliebiges i gilt. Mit der Definition von $NW_{\mathcal{I}}^f$ erhalten wir

$$Pr_{Z \sim U_l}[C(f(Z_{I_1}), \dots, f(Z_{I_{i-1}})) = f(Z_{I_i})] \geq \frac{1}{2} + \frac{1}{10 \cdot 2^{d/10}}$$

Mit anderen Worten, wir nehmen an, dass ein Schaltkreis C und ein i existieren, so dass der Schaltkreis C auf der Eingabe $f(Z_{I_1}), \dots, f(Z_{I_{i-1}})$ den Wert von $f(Z_{I_i})$ mit größerer Wahrscheinlichkeit als $\frac{1}{2} + \frac{1}{10 \cdot 2^{d/10}}$ hervorsagen kann. Wenn wir diese Aussage zum Widerspruch führen können, dann ist, wegen dem Satz von Yao, die

erzeugte Verteilung pseudozufällig.

Wir bezeichnen mit Z_1 und Z_2 die zwei unabhängigen Variablen, die den Koordinaten von Z in I_i und $[l] \setminus I_i$ entsprechen.

Wir erhalten

$$Pr_{Z_1 \sim U_n, Z_2 \sim U_{l-n}}[C(f_1(Z_1, Z_2), \dots, f_{i-1}(Z_1, Z_2)) = f_i(Z_1)] \geq \frac{1}{2} + \frac{1}{10 \cdot 2^{d/10}}, (*)$$

wobei f_k die Anwendung von f auf die Koordinaten von Z_1 ist, die zu I_k korrespondieren und zu denen von Z_2 , die zu $[l] \setminus I_k$ korrespondieren.

Mit einem Durchschnittsargument folgt, dass wenn diese Ungleichung gilt, dass dann auch ein fester String $z_2 \in \{0, 1\}^{l-n}$ existieren muss für den gilt

$$Pr_{Z_1 \sim U_n}[C(f_1(Z_1, z_2), \dots, f_{i-1}(Z_1, z_2)) = f_i(Z_1)] \geq \frac{1}{2} + \frac{1}{10 \cdot 2^{d/10}}.$$

Alle $f_j(Z_1, z_2)$ hängen für $j < i$ nur noch von Z_1 ab, wenn sie uns genug Informationen über Z_1 gäben, dann ließe sich ein Schaltkreis C' bauen, der als Eingabe sämtliche $f_j(Z_1, z_2)$ mit $j < i$ erhält und $f_i(Z_1)$ mit höherer Wahrscheinlichkeit als $\frac{1}{2} + \frac{1}{10 \cdot 2^{d/10}}$ vorhersagen würde.

Wegen $|I_j \cap I_i| \leq d$ für $j \neq i$ hängt die Funktion $Z_1 \rightarrow f_j(Z_1, z_2)$ von höchstens d Koordinaten von Z_1 ab. Somit kann diese Funktion durch einen Schaltkreis der Größe $d \cdot 2^d$ berechnet werden. Das bedeutet, wenn die Ungleichung (*) gilt, dann existiert ein Schaltkreis C'' der Größe $2^d/10 \cdot d \cdot 2^d + \frac{S}{2} < S$ für den gilt

$$Pr_{Z_1 \sim U_n}[C''(Z_1) = f_i(Z_1)] \geq \frac{1}{2} + \frac{1}{10 \cdot 2^{d/10}} > \frac{1}{2} + \frac{1}{S}.$$

Wegen $S := H_{avg}(f)$ steht dies im Widerspruch zur Definition der Average Case Hardness.

□

4 Fazit

Indem Nisan und Wigderson die von Yao aufgezeigte theoretische Möglichkeit der Nutzung eines Pseudo-Random-Generators zur Derandomisierung probabilistischer Komplexitätsklassen konkretisierten, legten sie einen Meilenstein in der Komplexitätstheorie. Durch ihre Arbeit zeigten sie ganz neue Wege zur Lösung der $\mathcal{P} = \mathcal{BPP}$ Frage auf und erhellten den Zusammenhang zwischen ganz unterschiedlichen Gebieten, wie die der PRG, Expander-Graphen und randomisierten Komplexitätsklassen. Auf Nisan und Wigdersons Arbeit folgte eine große Anzahl weiterer Arbeiten, die auf der hier beschriebenen Idee aufbauten. Wer sich mit diesem Gebiet der Komplexitätstheorie weiter beschäftigen möchte, dem sei das Buch Computational Complexity: A Modern Approach von Sanjeev Arora und Boaz Barak empfohlen. [Arora and Barak, 2009]

Literatur

- S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.
- N. Nisan and A. Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49:149–167, 1994.
- A. C. Yao. Theory and application of trapdoor functions. *IEEE Symposium on Foundations of Computer Science*, 1982.