

Weitere Konstruktionsmethode für explizite Disperser:

## Methode 2: Expanderbasierte Disperser

Verwende **guten Expandergraphen** (informell):

- Nur **linear viele Kanten** insgesamt.
- Jede nicht ganz große Menge von Knoten hat **linear viele Nachbarn** (in der Mengengröße).

Also spärlicher Graph, der trotzdem reichhaltige Verbindungsstruktur hat.

### **Erstaunlicherweise:**

Es gibt solche Graphen – sogar einfach konstruierbare!

## Beispiel: Gabber-Galil-Expander

Bipartiter Multigraph  $G = (U, V, E)$ ,  $U = V = \mathbb{Z}_m^2$ .

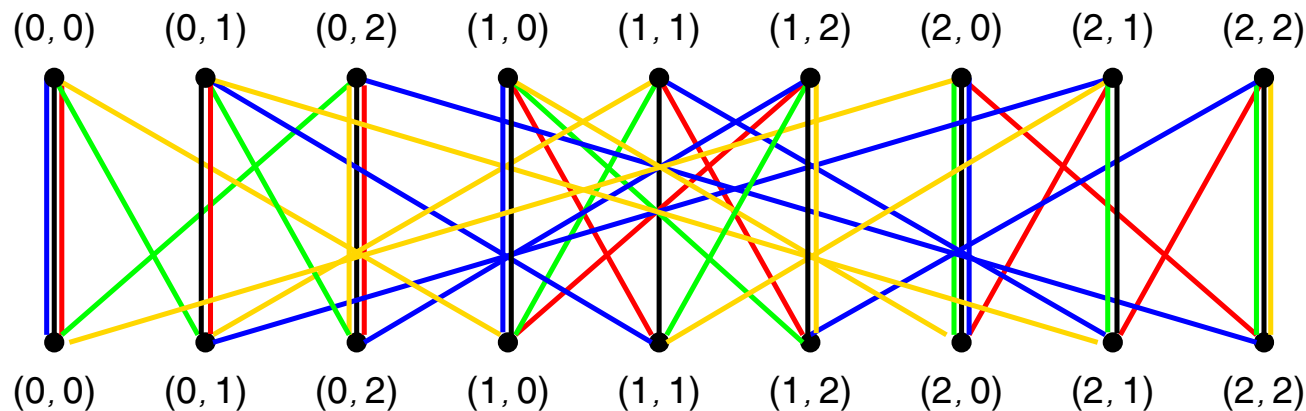
Kanten gegeben durch 5 Permutationen  $\mathbb{Z}_m^2 \rightarrow \mathbb{Z}_m^2$ ,  
beschrieben durch affin-lineare Abbildungen:

$T_1$ : Identität,  $T_2$ :  $\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ ,  $T_3$ :  $\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ ,

$T_4$ :  $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ ,  $T_5$ :  $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ .

Damit hat jeder Knoten Grad 5.

### Beispiel für $m = 3$ :



## Beispiel: Klassischer Expander-Disperser

Benutze expliziten,  $d$ -regulären,

guten Expander  $G = (V, E)$  mit  $V = \{0, 1\}^r$ .

(Z. B. Gabber-Galil-Expander, dann  $r$  ungerade benötigt.)

Nummeriere ausgehende Kanten eines Knotens mit  $1, 2, \dots, d$ .

Der Einfachheit halber zunächst direkt Verwendung für  
Zufallsbiteinsparung bei Probability-Amplification.

### **Simulation von Algorithmus $A$ , der $r$ Zufallsbits benutzt:**

- Wähle  $v_0 \in V$  zufällig gleichverteilt.
- Gehe von  $v_0$  aus  $\ell$  Schritte im Expander, für jeden Schritt Nachfolger zufällig gleichverteilt unter  $d$  vorhandenen.
- Simuliere  $A$  auf Zufallsbitstrings, die zu den auf dem Weg liegenden Knoten  $v_0, v_1, \dots, v_\ell \in V = \{0, 1\}^r$  gehören.

Beschreibung als Disperser-Graph  $G' = (U', V', E')$ :

$$U' = \{(v_0, i_1, \dots, i_\ell) \mid v_0 \in V, i_1, \dots, i_\ell \in \{1, \dots, d\}\}, V' = V,$$

$$E' = \{((v_0, i_1, \dots, i_\ell), v) \mid v \text{ liegt auf durch } (v_0, i_1, \dots, i_\ell) \text{ beschriebenenem Weg}\}.$$

### **Analyse:**

- Anzahl echter Zufallsbits:  $r + \ell \lceil \log d \rceil$ .
- $(T, 1/2)$ -(Majoritäts-)Disperser für  $T = 2^{-c\ell} |U|$  mit geeigneter Konstante  $c > 0$ . (Ohne Beweis.)

### **Ergebnis ( $d = 5, \ell = r$ ):**

Algo.  $A$  mit eins. Fehler  $1/2$ , Zeit  $t$ ,  $r$  Zufallsbits

↓ Satz 17.3.3

Algo.  $B$  mit eins. Fehler  $2^{-\Omega(r)}$ , Zeit  $\text{poly}(t)$ ,  $4r$  Zufallsbits.

## 17.4 Verbessern von schlechten Zufallsquellen

„Klassische“ Methoden aus der Mathematik:

Zufallsquelle mit **bekannter, fester Verteilung  $\mathcal{D}$**

→ Gleichverteilung.

Oft aber Verteilung der Zufallsquelle unbekannt.

**Hier:**

- Zufallsquelle gegeben, die „etwas Zufall“ produziert:  
Z. B.  $n$  „fast zufällige“ Bits mit Min-Entropie  $n/2$ .  
Beachte: Viele verschiedene Verteilungen möglich.
- Will damit randomisierte Algorithmen betreiben.  
Dafür Ausgabe der Zufallsquelle vorab „aufpeppen“.

Lösung mit Konstruktion aus Abschnitt 17.3:

### Satz 17.4.1:

Sei randomisierter Algorithmus  $A$  mit einseitigem Fehler  $\varepsilon$ , Rechenzeit  $t$  und  $r$  Zufallsbits gegeben.

Sei  $G = (U, V, E)$  mit  $U = \{0, 1\}^R$   $V = \{0, 1\}^r$  expliziter  $(T, \varepsilon)$ -Disperser, bei dem jeder Knoten in  $U$  höchstens  $d$  Nachbarn in  $V$  hat.

Betrachte Simulationsalgorithmus  $B$ , der Knoten  $u \in U$  zufällig gemäß **Verteilung  $\mathcal{D}$  mit Min-Entropie mindestens  $\log T + k$**  wählt, Algorithmus  $A$  auf Zufallsbitstrings  $v \in \Gamma(u)$  simuliert und akzeptiert, falls mindestens einer der  $A$ -Läufe akzeptiert.

Dann hat der Algorithmus  $B$  Fehlerwskt. kleiner als  $2^{-k}$  und Rechenzeit  $\text{poly}(d, R, t)$ .

## Beweis:

Satz 17.3.3: Falls  $B$  mit  $R$  echt zufälligen Bits betrieben, dann Fehlerwskt kleiner als  $T/2^R$ . Damit für feste Ja-Eingabe weniger als  $(T/2^R) \cdot 2^R = T$  Zufallsbitstrings, für die  $B$  nicht akzeptiert.

Betreibe nun  $B$  mit  $R$  „fast zufälligen“ Bits aus Quelle mit Verteilung  $\mathcal{D}$ , Min-Entropie  $\geq \log T + k$ . Dann hat jeder Zufallsbitstring, also auch jeder, für den  $B$  nicht akzeptiert, Wahrscheinlichkeit höchstens

$$2^{-(\log T + k)} = 2^{-k} \cdot (1/T).$$

Fehlerwskt. insgesamt (Vereinigungsschranke) kleiner als

$$2^{-k} \cdot (1/T) \cdot T = 2^{-k}$$

□

Analoge Aussage für zweiseitigen Fehler und Majoritäts-Disperser.

**Mit expanderbasiertem Disperser aus Abschnitt 17.3:**

In beliebigem RP- oder BPP-Algorithmus echte Zufallsbits ersetzbar durch  $R$  Bits aus schlechter Zufallsquelle mit Min-Entropie  $\alpha R$  für Konstante  $\alpha$  mit  $0 < \alpha < 1$ .

**Bestes bekanntes Ergebnis:**

In beliebigem RP- oder BPP-Algorithmus echte Zufallsbits ersetzbar durch  $R$  Bits aus schlechter Zufallsquelle mit Min-Entropie  $\Omega(R^\beta)$  für Konstante  $\beta$  mit  $0 < \beta < 1$ .

Untere Schranke mit Abzählargument:  $R^{\Omega(1)}$ .

## 17.5 Derandomisierung von zeitbeschränkten Algorithmen

Naive Derandomisierung von randomisiertem Algorithmus:  
Für  $r$  Zufallsbits Durchprobieren aller  $2^r$  Belegungen.  
Zeigt, dass  $BPP \subseteq EXP$ .

Auch Derandomisierung durch Schaltkreise einfach:

### Satz 17.5.1:

Jeder BPP-Algorithmus kann durch Folge von Schaltkreisen polynomieller Größe simuliert werden.

## Beweis:

Betrachte BPP-Algorithmus  $A$  mit Eingabelänge  $n$ .

Folgende Eigenschaften vorab sicherstellen:

- Algorithmus benutzt maximal  $r$  Zufallsbits, die als zusätzliche Eingabe zu Anfang auf dem Band erwartet werden (analog zu Rate-Verifikations-NTM).
- Fehlerwskt. kleiner als  $2^{-n}$ .

Satz 14.2.1 liefert äquivalenten Schaltkreis polynomieller Größe mit  $n$  „normalen“ Eingabebits und  $r$  Zufallseingabebits.

Betrachte Fehlermatrix  $F$  mit „normalen“ Eingaben  $x \in \{0, 1\}^n$  als Zeilen und Zufallsbitstrings  $y \in \{0, 1\}^r$  als Spalten:  
Eintrag  $F(x, y) = 1$ , falls Algorithmus  $A$  für Eingabe  $x$  und Zufallsbits  $y$  falsch rechnet,  $F(x, y) = 0$  sonst.

In jeder Zeile weniger als  $2^{-n} \cdot 2^r$  Einsen, insgesamt weniger als  $2^r$  Einsen. Also existiert Spalte  $y_0$  ohne Einsen.

Wähle  $y_0$  als feste Belegung der Zufallsbits, dann deterministischer Schaltkreis, der immer korrekt rechnet.  $\square$

Aktueller Wissenstand schließt  $BPP = EXP$  nicht aus.

Aus **Existenz von Einwegfunktionen** folgt neben  $P \neq NP$  auch  $BPP \subseteq SUBEXP$ , wobei SUBEXP Klasse aller Entscheidungsprobleme, die Algorithmus mit subexponentieller Rechenzeit haben.

**Ziel:** Nichttriviale (uniforme) Derandomisierung von BPP-Algorithmen unter schwächeren Voraussetzungen.

Im Folgenden zwei Meilensteine der Entwicklung von Derandomisierungsergebnissen.

### **Grundlegende Idee:**

Untere (Schaltkreis-)Schranken  $\rightarrow$  Pseudozufall.

### **Erinnerung an Kapitel 14:**

Für Entscheidungsproblem  $L$  zugehörige boolesche Funktionenfolge  $f_n^L : \{0, 1\}^n \rightarrow \{0, 1\}$  definiert mit  $(f_n^L)^{-1}(1) = L \cap \{0, 1\}^n$ .

### **Satz 17.5.2 (Babai, Fortnow, Nisan und Wigderson 1993):**

Falls es ein Entscheidungsproblem  $L \in \text{EXP}$  gibt, sodass für alle  $n$  die Funktion  $f_n^L$  superpolynomielle Schaltkreisgröße erfordert, dann gilt  $\text{BPP} \subseteq \text{SUBEXP}$ .

Sei E Klasse aller Entscheidungsprobleme, für die es einen Algorithmus mit Rechenzeit  $2^{O(n)}$  für Eingabelänge  $n$  gibt. (Also nur lineare Funktion im Exponenten erlaubt, nicht Polynom wie bei EXP.)

**Satz 17.5.3 (Impagliazzo und Wigderson 1997):**

Falls es ein Entscheidungsproblem  $L \in E$  und eine Konstante  $c > 0$  gibt, sodass für alle  $n$  die Funktion  $f_n^L$  Schaltkreisgröße  $2^{cn}$  erfordert, dann gilt  $P = BPP$ .

Im Folgenden zumindest Zusammenhang zu Pseudozufallszahlengeneratoren genauer herausarbeiten.

Erste Idee: Derandomisierung auf effizientes Schätzen der Anzahl von 1-Eingaben für Schaltkreise zurückführen.

### Circuit Acceptance Probability Estimation (CAPE):

**Eingabe:** Boolescher Schaltkreis  $C$  mit Eingabelänge  $n$  und Größe  $n$ .

**Versprechen:** Entweder akzeptiert  $C$  mindestens  $(3/4)$ -Anteil aller Eingaben oder höchstens  $(1/4)$ -Anteil.

**Frage:** Welcher der beiden Fälle liegt vor?

Falls Versprechen nicht erfüllt, dürfen Algorithmen **beliebige Ausgabe** produzieren (wie bei Kommunikationsprotokollen für Relationen, Kapitel 15).

**Lemma 17.5.4:** Falls es polynomiellen Algorithmus für CAPE gibt, dann auch für jedes Problem in BPP, also  $P = BPP$ .

**Beweisskizze:**

Betrachte Problem  $L$  mit BPP-Algorithmus  $A$ .

Für gegebene Eingabe  $x \in \{0, 1\}^n$ :

- Konstruiere aus  $A$  effizient simulierenden Schaltkreis  $C$  für Eingabelänge  $n$ , analog zum Beweis von Satz 17.5.1.
- Konstruiere Schaltkreis  $C_x$  durch Konstantsetzen der Eingabebits in  $C$ .

Dann gilt:  $C_x \in CAPE \Leftrightarrow x \in L$ .

Größe von  $C_x$  durch Einfügen von Dummybausteinen oder Dummyeingabebits passend für CAPE machen. □

## **Trivialer BPP-Algorithmus für CAPE:**

Gegebenen Schaltkreis  $C$  auf zufälliger Eingabe auswerten, Ausgabe des Schaltkreises als Ausgabe für CAPE verwenden. Ausgabe ist bei erfülltem Versprechen mit Wskt. mindestens  $3/4$  korrekt.

### **Ziel:**

Zufall in diesem Algorithmus durch Pseudozufallsgenerator ersetzen. Lemma 17.5.4 liefert dann Derandomisierung von beliebigem BPP-Algorithmus.

## Intuition:

Benutze Pseudozufallsgeneratoren, deren Ausgabe von kleinen SKs nicht von echtem Zufall zu unterscheiden.

### Definition 17.5.5:

- Sei Multimenge  $T \subseteq \{0, 1\}^r$  gegeben, sodass für beliebigen Schaltkreis  $C$  mit  $r$  Eingabebits der Größe  $r$  gilt:

$$|\Pr_{x \in T}\{C(x) = 1\} - \Pr_{x \in \{0,1\}^r}\{C(x) = 1\}| \leq \varepsilon.$$

Dann nenne  $T$  **Testmenge mit Fehler  $\varepsilon$  für Schaltkreise der Größe  $r$ .**

- **Pseudozufallsgenerator für Schaltkreise der Größe  $r$**  ist ein deterministischer Algorithmus, der die Elemente einer solchen Testmenge aufzählt.

### Beobachtung 17.5.6:

Pseudozufallsgenerator für Schaltkreise der Größe  $r$  mit Rechenzeit  $t(r)$  existiert: Dann BPP-Algorithmus simulierbar durch deterministischen Algorithmus mit Rechenzeit  $O(t(\text{poly}(n)) \cdot \text{poly}(n))$  bei Eingabelänge  $n$ .

Insbesondere für  $t(r) = \text{poly}(r)$ :  $P = BPP$ .

### Beweis:

Erzeuge sukzessive Elemente der Testmenge und generiere und simuliere für jedes den Schaltkreis gemäß des Beweises von Lemma 17.5.4. □

## Konstruktion eines Pseudozufallsgenerators für kleine Schaltkreise

„Low-End-Generator“, der hinter Beweis von Satz 17.5.2 steckt. Zunächst ohne Effizienzbetrachtung.

Boolesche Funktion  $f: \{0, 1\}^n \rightarrow \{0, 1\}$  gegeben, die keine Schaltkreise polynomieller Größe hat.

### **Erster Schritt:**

Erzeuge aus  $f$  Funktion  $\tilde{f}$ , sodass es für beliebiges Polynom  $s(n)$  keine Schaltkreise der Größe  $s(n)$  gibt, die  $\tilde{f}$  korrekt auf  $\left(\frac{1}{2} + \frac{1}{s(n)}\right)$ -Anteil der Eingaben berechnen.

(Ideen: Codierung der Wertetabelle, mehrfache Kopien.)

## Zweiter Schritt:

Zeige, dass clever ausgewählte Ausschnitte aus der Wertetabelle von  $\tilde{f}$  pseudozufällig für kleine Schaltkreise sind.

Dazu: Falls dies nicht erfüllt, kann  $\tilde{f}$  gut approximiert werden von kleinem Schaltkreis, im Widerspruch zum Ergebnis des ersten Schritts.

Außerdem zu zeigen, dass beide Schritte in Zeit  $2^{O(n^\varepsilon)}$  für beliebige Konstante  $\varepsilon > 0$  durchführbar.

Dann insgesamt Pseudozufallsgenerator für kleine Schaltkreise mit subexponentieller Rechenzeit und  $\text{BPP} \subseteq \text{SUBEXP}$ .

## 17.6 Derandomisierung von platzbeschränkten Algorithmen

Wissen bereits (Kapitel 13), dass  $ZPL = RL = NL$ .

**Wesentliches Ziel:**  $BPL = L$  oder zumindest  $BP_H L = L$ .

### Satz 17.6.1 (Nisan und Zuckerman 1996):

Sei  $s(n) \geq \log n$  platzkonstruierbar. Jeder  $s(n)$ -platzbeschränkte Algorithmus, der nur  $\text{poly}(s(n))$  Zufallsbits verwendet, kann durch einen deterministischen,  $s(n)$ -platzbeschränkten Algorithmus simuliert werden.

### Satz 17.6.2 (Saks und Zhou 1999):

$BP_H L \subseteq DSPACE(\log^{1.5} n)$ .