# Some applications of pairwise independence

Margus Niitsoo

University of Tartu

Research seminar in cryptography

# Outline

1. **Pairwise independent distributions**
2. **A simple application**
3. $BPP \subset \Delta_2$
4. **Recycling randomness**

# Introduction

We study certain special distributions on a set of $n$ different random variables

**Example**

| $Z_a$ | $Z_b$ | $Z_c$ | Pr |
|---|---|---|---|
| 0 | 0 | 0 | 0.25 |
| 0 | 1 | 1 | 0.25 |
| 1 | 0 | 1 | 0.25 |
| 1 | 1 | 0 | 0.25 |

It ressembles a uniform distribution. However...

# Hash function formalization

- We mostly use the formalization of a random hash function $h_r : \mathbf{Z}_{2^n} \to \mathbf{Z}_{2^m}$ where $h_r(x)$ and $h_r(y)$ are independent if $x \neq y$.
- Such functions are easy to construct taking $h_{(a,b)}(x) = ax + b$ where $(a, b) \in \mathbf{Z}_{2^n} \times \mathbf{Z}_{2^n}$ is chosen uniformly.
- We only need $2n$ random bits instead of $n2^n$ that way.

# Direct application

Main idea - take a randomized algorithm on a uniform distribution and see if it works if we substitute with a pairwise independent one.

# Toy example problem

**Example**

- Problem: given a graph $G = (V, E)$, find a two-valued vertex colouring $\chi : V \to \{0, 1\}$ that gives $c(\chi) = |\{(x, y) \in E : \chi(x) \neq \chi(y)\}| \geq \frac{|E|}{2}$.
- Solution: take $\chi$ to be a hash function chosen from a pairwise independent distribution and try all the possibilities.

# Complexity classes - terminology

- Language $\mathcal{L} \subset \{0,1\}^n$ and a potential sentence $x \in \{0,1\}^n$
- Polynomial-time predicate $P : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$
- Set of witnesses for $W_x = y \in \{0,1\}^m | P(x,y) = 1$

# Complexity classes

- $P$: $m = 0$.
- $NP$: $|W_x| > 0$ for $x \in \mathcal{L}$ and 0 elsewhere.
- $RP$: $\frac{|W_x|}{2^m} > c_{yes}$ for $x \in \mathcal{L}$ and 0 elsewhere.
- $BPP$: $\frac{|W_x|}{2^m} > c_{yes}$ for $x \in \mathcal{L}$ and $\frac{|W_x|}{2^m} < c_{no}$ elsewhere.

# Why?

- Motivation behind *RP* and *BPP*
- Relation to deterministic algorithms?

# Polynomial Hierarchy

- $\Sigma_2$ - $\exists z : \forall w : P(x, z, w) = 1$
- $\Pi_2$ - $\forall z : \exists w : P(x, z, w) = 1$
- $\Delta_2 = \Sigma_2 \cap \Pi_2$

# The result

- Turns out that $BPP \subset \Delta_2$
- Proof first shows (via hash functions) that $BPP \subset \Pi_2$ and then applies symmetry considerations.

# Recycling WHAT?

- Recycling: (tr. v) To put or pass through a cycle again, as for further treatment.
- Always recycle, because good randomness is hard to come by.

# What is really going on?

- Essentially, we are examining different pseudorandom generators that take a (truly) random seed and use it to generate more seemingly random bits.
- We use the framework of choosing potential witnesses for $x \in \mathcal{L} \in RP$

# The Generators

We describe 4 different generators

| Generator | Random bits | Error |
|---|---|---|
| Chor-Goldreich | $O(r)$ | $O(\frac{1}{k})$ |
| Nisan | $O(r \lg k)$ | $2^{-O(k)}$ |
| Karp-Pippenger-Sipser | $O(r)$ | $O(k^{-0.1})$ |
| Ajtai-Komlós-Szemerédi | $r + O(k)$ | $2^{-O(k)}$ |

Error is essentially the chance of not finding a witness given there are at least an $\epsilon$ fraction of them.

# Chor-Goldreich Generator

Randomly choose $r = (a, b)$, then use values $h_r(0), \ldots, h_r(k)$

# Nisan Generator

- Take $l = \lg k$, choose $h_1, \ldots, h_l : \{0,1\}^r \rightarrow \{0,1\}^r$ uniformly from PID. Also choose a seed value $y \in \{0,1\}^r$.
- Let $I = \{i_1, \ldots, i_m\} \subset \{1, .., l\}$ (where $i_1 < \cdots < i_m$) and take $y_I = h_{i_1}(\ldots h_{i_m}(y) \ldots)$.
- Use all such $y_I$.

# Expanders, KPS and AKS Generators

- An Expander is a $d$-regular graph whose adjacency matrix has a small enough second largest eigenvalue.
- What it really means is that the neighbours of a randomly chosen node are usually nearly uniformly distributed.
- Both random generators build an expander with the vertex set $V = \{0, 1\}^r$.
- Karp-Pippinger-Sipser generator uses the random seed to choose a vertex and then uses its neighbouring vertices.
- Ajtai-Komlós-Szemerédi generator performs a random walk starting from a random vertex and uses the vertices encountered.

# Thank you for listening

Comments? Questions?