Some applications of pairwise independence

Research Seminar in Cryptography

Margus Niitsoo

December 11, 2007

Abstract

This report provides a brief review of lectures given by Michael Luby and Avi Wigderson in 1995 on the topic of pairwise independence and derandomization. We show that $BPP \subset \Delta_2$ with a proof given by Sipser in 1983 and then show different ways of recycling random bits given by Chor and Goldreich (1989), Nisan (1992), Karp, Pippenger and Sipser (1986) and by Ajtai, Komloś and Szemerédi (1987). The article is mainly intended to show the power and applications of pairwise independent distributions in both complexity theory and in generating and recycling pseudorandom bits.

1 Pairwise independence and tools of the trade

We look at a set of random variables $\mathbf{Z} = \{Z_u : u \in U\}$ where U is an index set for the variables with $|U| = N$. Each Z_u takes values from a fixed

Table 1: The simplest pairwise independent distribution different from uniform.

Z_a	Z_b	Z_c
0	0	0
0	1	1
1	0	1
1	1	0

set T (so $Z_u \in T$). We call the distribution of the set $\mathbf{Z}$ *pairwise independent* if for every $x, y \in U$ where $x \neq y$ and for every $\alpha, \beta \in T$ we have $\Pr[Z_x = \alpha \wedge Z_y = \beta] = \Pr[Z_x = \alpha] \Pr[Z_y = \beta]$. It is important to note that pairwise independence does not imply complete independence. To demonstrate that, we take an example where $T = \{0, 1\}$ and $U = \{a, b, c\}$ and give a joint distribution for Z_a, Z_b, Z_c in table 1. In this case the values of two variables always uniquely determine the third, so the third is definitely dependent on the other two, yet it does not depend on either of them separately. It is indeed easy to verify that if Z_a, Z_b, Z_c have that distribution then the set $\mathbf{Z}$ is pairwise independent.

Although complete independence is usually better in terms of its properties, it has one major drawback - if we want to have a positive probability for M different valuations for every Z_u, $u \in U$ then there need to be M^N different valuations for the vector $(Z_u : u \in U)$ that also have a positive probability. Since we are considering computer science applications, we like our considered sets to be as small as possible (so we could possibly enumerate over all their members or at least over a significant part of them).

This is where the pairwise independent distributions really excel — one can usually construct pairwise independent distributions that have a base far smaller than their fully independent counterparts. However, we first make a few simplifications. We fix $U = \{0,1\}^m$ and $T = \{0,1\}^n$ and concern ourselves with distributions for T^U that give a uniform distribution for every coordinate separately (so for every Z_u, $u \in U$ and $\alpha \in T$ we have $\Pr[Z_u = \alpha] = \frac{1}{2^n}$).

It turns out that such distributions are rather easy to construct for the case $m = n$. Take $\mathcal{S} = \{0,1\}^n \times \{0,1\}^n$ and define $h_s(x) = a * x + b$ for $s = (a,b) \in \mathcal{S}$ with multiplication and addition as defined on the Galois field $GF[2^n]$. Then uniformly choose $s \in \mathcal{S}$ and define $Z_u = h_s(u)$. It can be shown that this method of choosing values for Z_u gives a uniform and pairwise independent distribution for every Z_u, $u \in U$. The functions h_s can be viewed in familiar terms as *hash functions* mapping from $\{0,1\}^n$ uniformly to $\{0,1\}^n$. The same construction can easily be extended to the case where the mapping is to $\{0,1\}^m$ with $m < n$ by simply disregarding the last $n - m$ bits of the value. It turns out that pairwise independence holds for that generalization as well.

We note that we thus use only $|\mathcal{S}| = 2^{2n}$ different valuations for the Z vector which is a lot less than $(2^n)^{2^n}$ required for a completely independent distribution. That decrease in size is what makes this construction so powerful. We now proceed to showing how it is used in some rather remarkable applications.

2 A simple application

We look at a weak version of a problem called MAXCUT: given a graph $G = (V, E)$, find a two-valued vertex colouring $\chi : V \to \{0, 1\}$ that gives $c(\chi) = |\{(x, y) \in E : \chi(x) \neq \chi(y)\}| \geq \frac{|E|}{2}$.

To find an algorithm we first note, that if $\chi(x)$ is chosen randomly and independent of all the other $\chi(y)$ with both 0 and 1 having the probability $\frac{1}{2}$, then

$$\begin{aligned} E[c(\chi)] &= \sum_{\chi} \Pr[\chi] \sum_{(x,y)\in E} \Pr[\chi(x) \neq \chi(y)] = \\ &= \sum_{(x,y)\in E} \sum_{\chi} \Pr[\chi] \Pr[\chi(x) \neq \chi(y)] = \sum_{(x,y)\in E} \frac{1}{2} = \frac{|E}{2} \ . \end{aligned}$$

This guarantees that a solution to our problem must exist, as there has to be one case that is at least as good as or better than the average.

However, the same argument applies if the distribution we choose χ from only pairwise independent distribution so we also get $E[c(h_s)] = \frac{|E|}{2}$ when $s \in \mathcal{S}$ is chosen randomly from the uniform distribution and $\mathcal{S}$ is the family of pairwise independent functions mapping V to $\{0, 1\}$. To get such a family, we can take $\mathcal{S} : \{0, 1\}^{\lceil \lg |V| \rceil} \times \{0, 1\}$ and use the construction given in the previous section. In that case $|\mathcal{S}| = O(|V|^2)$ so we can just try all the different h_s in polynomial time and one of them has to match the criterion.

This type of derandomization is probably one of the simplest applications possible for pairwise independence - we take a randomized algorithm that samples an exponentially large uniform distribution and replace it with a polynomial-sized pairwise independent one, then enumerate over the whole

thing to get an exact result. Since sometimes actual uniformity is needed, the approach cannot be called universal, but it works in a surprisingly wide range of cases.

3 Complexity classes RP and BPP

Before we can proceed to other uses, we first need to introduce some complexity theory. We have a language $\mathcal{L} \subset \{0,1\}^n$ and are concerned with algorithms (formalized as Turing machines) that can determine whether $x \in \mathcal{L}$.

We say that $\mathcal{L} \in P$ (stands for Polynomial) if there exists a Turing Machine M that takes one input $x \in \{0,1\}^n$ and works in polynomial time of $||x|| = n$ returning 1 if and only if $x \in \mathcal{L}$.

We say that $\mathcal{L} \in NP$ (stands for Non-deterministic Polynomial) if there exists a TM $M : \{0,1\}^n \times \{0,1\}^r \to \{0,1\}$ such that $M(x,y) = 1$ iff y is a 'witness' to x belonging to $\mathcal{L}$. There is an additional constraint that for each $x \in \mathcal{L}$ there has to be at least one witness $y \in \{0,1\}^r$ such that $M(x,y) = 1$ and that no such witnesses can exist if $x \notin \mathcal{L}$.

Both previously mentioned classes are usually considered deterministic as the TM for P just produces a correct answer and the TM for NP can be made to do that as well if we enumerate over all the values of $y \in \{0,1\}^r$. We now introduce two complexity classes corresponding to randomized algorithms.

We say that $\mathcal{L} \in RP$ (stands for Randomized Polynomial) $\mathcal{L} \in NP$ with

the additional constraint that there is a fixed constant $c_{yes} > 0$ such that for every $x \in \mathcal{L}$ the set of witnesses $W_x = \{y \in \{0,1\}^r : M(x,y) = 1\}$ has at least $c_{yes}2^r$ elements or equivalently $\mu(W_x) = \frac{|W_x|}{|\{0,1\}^r|} \geq c_{yes}$.

What makes this class better than NP is that as r grows without bounds, $\frac{|W_x|}{2^r}$ may get as close to 0 as one wants in NP languages whilst the fraction is always guaranteed to be at least c_{yes} for RP problems. In practice this gives us a randomized algorithm for deciding $x \in \mathcal{L}$: choose $y \in \{0,1\}^r$ randomly and decide $x \in \mathcal{L}$ iff $M(x,y) = 1$. This guarantees that we always decide correctly if $x \notin \mathcal{L}$ and if y is chosen uniformly and $x \in \mathcal{L}$ then we decide correctly with probability c_{yes}. Therefore RP roughly corresponds to the class of polynomial-time randomized algorithms with one-sided error (with only first type error).

It should also be noted that the probability c_{yes} can be increased by choosing many different y and deciding $x \in \mathcal{L}$ if any one of them is a witness. Choosing k different values of y increases the chance of right answer $c'_{yes} = (1 - c_{yes})^k$ so we can reduce the error probability to as small as we want by taking enough samples.

We say that $\mathcal{L} \in BPP$ (stands for Bounded error Probabilistic Polynomial) if there exist fixed $c_{no} < c_{yes}$ such that for $x \in \mathcal{L}$ we have $\mu(W_x) \geq c_{yes}$ and for $x \notin \mathcal{L}$ we have $\mu(W_x) \leq c_{no}$.

The randomized algorithm to use for BPP TM-s is the same as that for RP - just choose one y randomly and decide $x \in \mathcal{L}$ iff y happens to be a witness. BPP thus corresponds to the class of randomized algorithms with

both-sided error.

Just like with RP we can also reduce the error probability by taking $k > 1$ samples and then testing if the number of witnesses among them is greater than $k\frac{c_{yes}+c_{no}}{2}$. In this case exact bounds are harder to derive, but using Chebychev inequality it is rather easy to prove that $c'_{yes} \geq 1 - \frac{1}{k(c_{yes}-c_{no})^2}$ and $c'_{no} \geq \frac{1}{k(c_{yes}-c_{no})^2}$ so the error rate does indeed come down on both sides as k increases. Tighter bounds can be achieved by using the properties of the binomial distribution. Namely,

$$c'_{yes} \geq 1 - e^{-0.5k(c_{yes}-c_{no})^2} \quad \text{and} \quad c'_{no} \leq e^{-0.5k(c_{yes}-c_{no})^2}$$

by Hoeffding's inequality [3]. Therefore the error rate drops exponentially.

4 A proof that $BPP \subset \Delta_2$

It is obvious from the definition that $RP \subset NP$. It is however much harder to see how BPP relates to the polynomial hierarchy. The following is due to Sipser [2] where he shows that $BPP \subset \Delta_2$.

We say that $\mathcal{L} \in \Sigma_2$ if $\mathcal{L}$ can be described in the form

$$x \in \mathcal{L} \Leftrightarrow \exists z : \forall w : q_{\mathcal{L}}(x, z, w) = 1 \ ,$$

where $q_{\mathcal{L}}$ works in polynomial time on all three inputs and both z and w are of length (measured in bits) polynomial in the size of x.
Analogously we say that $\mathcal{L} \in \Pi_2$ if $\mathcal{L}$ can be described in the form

$$x \in \mathcal{L} \Leftrightarrow \forall z : \exists w : q_{\mathcal{L}}(x, z, w) = 1$$

subject to same constraints as before.
We define $\Delta_2 = \Sigma_2 \cap \Pi_2$.

We show that $BPP \subset \Pi_2$. Since $\mathcal{L} \in BPP$ implies $\overline{\mathcal{L}} \in BPP$ (as BPP is symmetric), that also implies $BPP \subset \Sigma_2$ and thus $BPP \subset \Delta_2$. Let us assume that if $x \in \mathcal{L}$ then $|W_x| > 2^r - 1$ and if $x \notin \mathcal{L}$ then $|W_x| < 2^{(r-1)/2}$. That can be achieved on all BPP languages by taking $k = \frac{(r-1)\ln 2}{(c_{yes}-c_{no})^2}$ and using the tighter bounds given for repeated sampling with BPP. Since the algorithm now uses k witnesses instead of one, the sample witness also grows to W_x^k but that changes nothing since $r' = kr$ is still polynomial in n.

We choose a hash function $h : \{0,1\}^r \to \{0,1\}^{r-1}$ from a pairwise independent family of polynomial size H constructed as described above. If $x \in \mathcal{L}$ then we cannot map all the witnesses to different elements (as there are too many of them) so $h|_{W_x}$ is not one-to-one. However, if $x \notin \mathcal{L}$ then the expected number of collisions (or cases where two witnesses map to the same hash) is $E[c] = \sum_{w_1 \neq w_2 \in W_x} \Pr[h(w_1) = h(w_2)] = \binom{n}{2}\frac{1}{t}$ where the probability is taken over the different hash functions in H, t is the size of the table and n is the number of witnesses. Since $t > n^2$ we get $E[c] < \frac{1}{2}$ so the probability of h being one-to-one is greater than $\frac{1}{2}$ and thus different form zero. Thus there has to be at least one one-to-one hash function in the pairwise independent hash function family. This gives us

$$x \notin \mathcal{L} \Leftrightarrow \exists h \in H : \forall y, y' \in W_x : y \neq y' \Rightarrow h(y) \neq h(y') \ ,$$

or equivalently

$$x \in \mathcal{L} \Leftrightarrow \forall h \in H : \exists y, y' \in W_x : y' \neq y, h(y) = h(y') \ ,$$

which is a Π_2 form for BPP.

5 Recycling randomness

As we all know, good random is hard to come by. So hard in fact that when you get a few truly random bits, you want to get the most you can out of them. This is where recycling randomness comes in. Essentially, all the methods described just take some random bits as seeds and generate some extra pseudorandom bits based on them.

As we have brought in the complexity theory machinery, we are going to use it here as well. To be exact, we are going to demonstrate the recycling efficiency by applying the methods to take multiple samples from the witness space for the problems in classes BPP and RP and showing how that brings down the error bounds. There are 4 different methods covered:

Generator	Random bits	Error
Chor-Goldreich	$O(r)$	$O(\frac{1}{k})$
Nisan	$O(r \lg k)$	$2^{-O(k)}$
KPS	$O(r)$	$O(k^{-0.1})$
AKS	$r + O(k)$	$2^{-O(k)}$

where r is the size of a potential witness in bits and k is a parameter determined by the precision needed from the algorithm. The bounds can usually be proven for both BPP and RP but for technical simplicity the latter is usually preferred for proofs in the original article. We now go on to describe the algorithms.

6 The Chor-Goldreich generator

Let us assume we have a language $\mathcal{L}$ from BPP such that $c_{yes} \geq \frac{3}{4}$ and $c_{no} \leq \frac{1}{4}$. We randomly choose a hash function $h : \{0,1\}^r \to \{0,1\}^r$ from our pairwise independent hash family. The function can be determined by $2r$ bits as we showed in the first section. Algorithm returns $x \in \mathcal{L}$ if at least $\frac{k}{2}$ of $h(1), \ldots, h(k)$ belong to W_x and $x \notin \mathcal{L}$ otherwise.

We now prove that the probability for misclassifying $x \in \{0,1\}^n$ this way is at most $4/k$. We look at each of $h(i)$ as a discrete random variable Z_i that has a value of 1 if $h(i) \in W_x$ and a value of 0 otherwise. By our choice of h all the Z_i are identically distributed and pairwise independent, with a mean of $\mu(W_x)$ and a variance of $\sigma^2 = \mu(1-\mu) \leq \frac{1}{4}$. Using the Chebychev inequality and the fact that both mean and variance are additive we get

$$\Pr[|\sum_{i=1}^{k} Z_i - \mu k| > \frac{k}{4}] \leq (\frac{4}{k})^2 k\sigma^2 \leq \frac{4}{k} \ .$$

If $x \in \mathcal{L}$ then $\mu(W_x) \geq c_{yes} \geq \frac{3}{4}$ and the probability of misclassifying is

$$\begin{aligned} \Pr[\sum_{i=1}^{k} Z_i < \frac{k}{2}] &= \Pr[\mu k - \sum_{i=1}^{k} Z_i > \mu k - \frac{k}{2}] \\ &\leq \Pr[\mu k - \sum_{i=1}^{k} Z_i > \frac{k}{4}] = \Pr[|\sum_{i=1}^{k} Z_i - \mu k| > \frac{k}{4}] \leq \frac{4}{k} \ , \end{aligned}$$

where the next-to-last equality holds because $\sum Z_i - \mu k > \frac{k}{4}$ would mean $\sum Z_i > \mu k + \frac{k}{4} \geq k$ which is impossible. Analogous analysis applies for $x \notin \mathcal{L}$ so the probability of misclassifying is indeed at most $\frac{4}{k}$.

As most of the bound proofs for the other algorithms are rather technical and not really enlightening, we omit most of them and only give descriptions

of how the algorithms generate their needed rk pseudorandom bits.

7 The Nisan generator

Let $l = \lg k$ and choose y from $\{0,1\}^r$ (as a seed) and $h_1, \ldots, h_l$ randomly from our pairwise independent hash function family mapping $\{0,1\}^r$ to $\{0,1\}^r$. To do all that we need $r * (2l+1) = O(r \lg k)$ random bits. We now construct $2^l = k$ potential witnesses by considering all subsets $I = i_1, \ldots, i_m \subset 1, \ldots, l$ where $i_1 < i2 < \cdots < i_m$ and defining for each of them $y_I = h_{i_1}(h_{i_2}(\ldots(h_{i_m}(y))\ldots))$ and $y_\emptyset = y$. * Then every y_I is checked for being a witness and the descision is again made based on the number of witnesses found using the standard methods described in the second chapter.

The proof of complexity in this case is rather simple, although technical. We consider the complexity class RP and make the descision $x \in \mathcal{L}$ if at least one witness is found. To prove the bounds,the authors first show that for all $A, B \subset \{0,1\}^r$ and for all but an $\epsilon = 2^{-r/3}$ fraction of our hash family,

$$|\Pr_{y \in \{0,1\}^r}[y \in A, h(y) \in B] - \Pr_{y,z \in \{0,1\}^r}[y \in A, z \in B]| < \epsilon \ ,$$

which is the so called Hash Mixing lemma. This can be used to show that increasing l by one roughly squares the error probability (plus-minus a few ϵ). It is so because increasing l by one doubles the number of potential witnesses and the lemma allows us to show that the new witnesses created with h_{l+1} behave in roughly the same way as the ones we had before but are independent of them. Therefore, after $l = \lg k$ squaring steps the error is

*The construction was originally described in terms of recursion and trees, which is more convenient for proof but somewhat harder to grasp intuitively.

$(1 - c_{yes})^{2^l} = (1 - c_{yes})^k = 2^{-k \lg(1/(1-c_{yes}))}$

8 Expander graphs and the Karp-Pippenger-Sipser generator

We look at d-regular graphs (so each vertex in the graph has a degree of d) with n vertices $G(V, E)$. If we look at the adjacency matrices of such graphs, we find that their largest eigenvalue is always d and is realized by an eigenvector of $(1, \ldots, 1)$. What interests us in these graphs is the second largest eigenvalue λ. The reason for that is the so-called Expander Mixing Lemma, which states, that for all $A, B \subset V$,

$$|\Pr_{x,s}[x \in A, e_s(x) \in B] - \Pr_{x,y}[x \in A, y \in B]| \leq \frac{\lambda}{d} \ ,$$

where x is randomly chosen from amongst all vertices in both cases, s is chosen from the set $\{1, \ldots, d\}$ and $e_i(x)$ notes the other end vertex of the i-th edge leaving x and in the second case y is chosen uniformly and independently of x from the vertex set.

What the Expander Mixing Lemma really says is that given a graph with sufficiently large λ, it is hard to distinguish a truly random choice of x and y from just choosing x randomly, then choose a random edge e connected to it and taking y to be the second endpoint of one of that edge. From now on, we call such graphs with a small λ expanders.

Given any n and d it is possible to construct an expander with n' vertices that is d'-regular where $n \leq n' \leq 2n$ and $d \leq d' \leq 2d'$ and with $\lambda \leq d'^{9/10}$ so

the probability difference bound in the EML comes to $d'^{-1/10}$. This construction is what the KPS algorithm uses. Namely, it builds a 2^r node k-regular graph (presumably an implicit construction as building it out in full would take exponential time). Then, it chooses a random node y (using r random bits) and tests all its neighbours $y_1, \ldots, y_k$ as possible witnesses. It is quite straightforward to see that the EML implies that the probability of all of them not being witnesses for an $x \in \mathcal{L}RP$ is less than $2k^{-1/10}$.

9 The Ajtai-Komlós-Szemerédi generator

The AKS generator also uses the expander graph, but in a more subtle way — namely, instead of taking all the neighbours of a single node, it chooses nodes traversed along a random walk in the Expander graph.

To be more specific, we fix $n = 2^r$ and choose d such that an expander graph with $\lambda \leq \frac{d}{4}$ can be constructed. Then choose a random node $z \in V$ (using r random bits) and k numbers $i_1, \ldots, i_k$ uniformly from the set $\{1, \ldots, d\}$. Then take all the nodes y_j traversed by starting from z and taking the i_j-th edge to the next node on the j-th step (y_1 is the first node from z but not z itself). This generator gives far better results than the KPS as the bound for the error probability for a language in RP can be shown to be $2^{-\Theta(k)}$.

10 Derandomization (conclusion)

The previous sections show us how to bring randomized polynomial time languages in BPP into the polynomial hierarchy and how to generate good pseudorandom witness candidates for the said algorithms using very few random bits. What remains is to put the two together.

As we noted for the $BPP \subset \Delta_2$ proof, we can bring c_{yes} and c_{no} to as close to 1 and 0 as we want as long as we increase the witness space size from r to rk for large enough k. The generators introduced thereafter allow us to bring the witness space size down again, because we can generate the required k witnesses from far less than rk bits and still achieve nearly the same bounds on the probabilities. The witness space is still exponential, but the exponent achieved this way can be considerably smaller. Assuming the complexity-theoretic construction of a deterministic problem $\mathcal{L}' \in \Delta_2$ from a nondeterministic $\mathcal{L} \in BPP$ might have some practical implications (which might not be a valid assumption but is a good way of tying the ends together), the generators given here might directly be used in derandomization within that same construction.

References

[1] Luby, M. , Wigderson, A. "Pairwise Independence and Derandomization", Foundations and Trends® in Theoretical Computer Science Vol 1-4 (2005)

[2] Sipser, M. "A complexity theoretic approach to randomness", STOC pp 330-335, 1983

[3] Binomial distribution - Wikipedia, the free encyclopedia http://en.wikipedia.org/wiki/Binomial_distribution