Applications of Randomization

- Other applications of the technique include verifying polynomial identities.
- For instance, let $P_1(x)$, $P_2(x)$ be two polynomials in a field F.
- The polynomial product verification problem is to check whether $P_1(x) \cdot P_2(x) = P_3(x)$ for a given $P_3(x)$.
- It holds that there exists an O(n log n) time algorithm to multiply two polynomials, where n is themaximum degree of P_1 and P_2.
- We design a verification algorithm that is faster than O(n log n).

Applications of Randomization

- Let $S \subset F$ be a subset of size at least $2n + 1$.
- The main idea of the verification procedure is that if indeed $P_3(x)$ equals $P_1(x) \cdot P_2(x)$, then, also $P_3(r) = P_1(r) \cdot P_2(r)$ for r chosen uniformly at random from S.
- Further, evaluating a polynomial at a given input can be done in $O(n)$ time.
- So, we can declare that $P_3(x)$ equals $P_1(x) \cdot P_2(x)$ unless $P_3(r) \neq P_1(r) \cdot P_2(r)$.
- The algorithm makes a mistake only when indeed $P_3(x) \neq P_1(x) \cdot P_2(x)$ but the choice of r fails to detect this.

Applications of Randomization

- To estimate the probability that the algorithm makes a mistake, let $Q(x) := P_3(x) - P_1(x) \cdot P_2(x)$.
- The degree of $Q(x)$ is at most $2n$.
- Suppose that $P_3(x) \neq P_1(x) \cdot P_2(x)$.
- Then, $Q(x)$ is a nonzero polynomial.
- So the test fails if $Q(r) = 0$ (but $P_3(x) \neq P_1(x) \cdot P_2(x)$).
- However, the polynomial $Q(x)$ of degree at most $2n$ can have at most $2n$ roots.
- So, the probability that $Q(r) = 0$ is at most $2n/|S|$, which is also the probability of error.
- As earlier, the probability of failure can be made polynomially small in n by using repeated trails or choosing a larger S, or both.

Applications of Randomization

- One may wonder whether it is at all worthwhile to have elaborate verification algorithms for things as simple as polynomial product verification.
- Such techniques however are more applicable when polynomials are not available explicitly.

Finger-Printing

- The two algorithms that we considered today have the property that for inputs that are identical, the algorithm does not make any error.
- But in inputs that are not identical, the algorithm makes an error that is upper bounded by at least a constant.
- Repeated runs of the same algorithm can catch the error, hence the error can be made arbitrarily small.

Finger-Printing

- On inputs that are not identical, the algorithm makes an error that is upper bounded by at least a constant.
- Repeated runs of the same algorithm can catch the error, hence the error can be made arbitrarily small.
- Consider A to be a finger-printing algorithm.
- Let us run A on input x, y for t iterations.
- The t outputs are, say, o_1, o_2, ..., o_t.
- If any of these t outputs are NO, then we can return NO as the answer.
- If all t are YES, then we return YES as the answer.
- Given that x = y,

Finger-Printing

- Repeated runs of the same algorithm can catch the error, hence the error can be made arbitrarily small.
- Consider A to be a finger-printing algorithm.
- Let us run A on input x, y for t iterations.
- The t outputs are, say, $o_1, o_2, \ldots, o_t$.
- If any of these t outputs are NO, then we can return NO as the answer.
- If all t are YES, then we return YES as the answer.
- Given that $x = y$, $Pr(A(x,y) = YES) = 1$.
- Given that $x \neq y$, $Pr(A(x,y) = YES) \leq (1/2)^t$.
- So, if $t = O(\log n)$, then the error probability is $O(1/n^c)$.

RP and co-RP

- The above algorithms are called as co-RP algorithms.
- RP stands for Randomized Polynomial.
- Definition: The class RP consists of languages L such that there exists a randomized algorithm running in worst case polynomial time such that for any input x:
  - $x \in L \Rightarrow \Pr(A \text{ accepts } x) \geq 1/2$.
  - $x \notin L \Rightarrow \Pr(A \text{ accepts } x) = 0$

- The complement of the class RP is the class co-RP.
- Note that any RP or co-RP algorithm can err only on one side, either for x in L, or for x not in L.

RP and co-RP

- In definition: The class RP consists of languages L such that there exists a randomized algorithm running in worst case polynomial time such that for any input x:
  - $x \in L \Rightarrow Pr(A \text{ accepts } x) \geq 1/2$.
  - $x \notin L \Rightarrow Pr(A \text{ accepts } x) = 0$

- The complement of the class RP is the class co-RP.
- In definition: The class co-RP consists of languages L such that there exists a randomized algorithm running in worst case polynomial time such that for any input x:
  - $x \in L \Rightarrow Pr(A \text{ accepts } x) = 0$.
  - $x \notin L \Rightarrow Pr(A \text{ accepts } x) \leq 1/2$.

RP and co-RP

- In definition: The class RP consists of languages L such that there exists a randomized algorithm running in worst case polynomial time such that for any input x:
  - $x \in L \Rightarrow \Pr(A \text{ accepts } x) \geq 1/2$.
  - $x \notin L \Rightarrow \Pr(A \text{ accepts } x) = 0$

- In definition: The class co-RP consists of languages L such that there exists a randomized algorithm running in worst case polynomial time such that for any input x:
  - $x \in L \Rightarrow \Pr(A \text{ accepts } x) = 0$.
  - $x \notin L \Rightarrow \Pr(A \text{ accepts } x) \leq 1/2$

- Note that any RP or co-RP algorithm can err only on one side, either for x in L, or for x not in L.

RP and co-RP

- In definition: The class RP consists of languages L such that there exists a randomized algorithm running in worst case polynomial time such that for any input x:
  - $x \in L \Rightarrow \Pr(A \text{ accepts } x) \geq 1/2$.
  - $x \notin L \Rightarrow \Pr(A \text{ accepts } x) = 0$

What about Randomized QuickSort

- The randomized quick sort algorithm does not make any errors in its output.
- So, clearly, it does not fit into either of RP or co-RP.
- While there are no errors in its output, we recall that its run time may vary.
- Such algorithms are called ZP algorithms.
  - Stands for Zero Error Expected Polynomial, time
- The class ZP consists of languages L such that there is a randomized algorithm A that always outputs the correct answer while running in expected polynomial time.
- Another name for ZP algorithms is Las Vegas algorithms.

Proof by Existence

- Many times you want to show that a particular combinatorial object exists.
- May be very inefficient to build possibly because of a huge space and a small target of interest.
  - Like finding a needle in a haystack.
- This is where randomization can come to the resuce.

Proof by Existence

- Two useful statements:

1) If a random variable has a finite expected value $E[X] = a$, then certainly there exists a realisation of X with value $\geq a$ and a realisation of X with value $\leq a$.
2) If a random object drawn from some universe of objects has a certain property with non-zero probability then there must exist an object with that property in this universe.

Proof by Existence

- Two useful statements:

1) If a random variable has expected value $E[X] = a$, then certainly there exists a realisation of X with value $\geq a$ and a realisation of X with value $\leq a$.

2) If a random object drawn from some universe of objects has a certain property with non-zero probability then there must exist an object with that property in this universe.

- Each of these statements looks simple on their own, but they are remarkably powerful in Computer Science.

Proof by Existence

- We will start with a simple example.
- Consider an undirected graph G = (V, E).
- We want to find a subgraph G' of G that:

  1) has the largest number of edges of G, and

  2) G' is bipartite.
- We will first show the existence of a G' with |E(G')| at least |E(G)|/2.

Proof by Existence

- Consider an undirected graph G = (V, E).
- We want to find a subgraph G' of G that:

  1) has the largest number of edges of G, and

  2) G' is bipartite.
- We will first show the existence of a G' with |E(G')| at least |E(G)|/2.
- The random experiment we perform is to assign a bit 0 or 1, denoted b(v), to each vertex v of G ind. and uar.
- Put all vertices in G'.
- An edge uv is in G' if and only if b(u) is different from b(v).

Proof by Existence

- The random experiment we perform is to assign a bit 0 or 1, denoted b(v), to each vertex v of G ind. and uar.
- Put all vertices in G'.
- An edge uv is in G' if and only if b(u) is different from b(v).
- In other words, G' = (V_0 U V_1, V_0xV_1 int. E(G)).
- Let us now bound |E(G')|.
- Let X_{uv} be a random variable that indicates the event {uv in E(G')}.
- $E[X_{uv}] =$

Proof by Existence

- The random experiment we perform is to assign a bit 0 or 1, denoted b(v), to each vertex v of G ind. and uar.
- Put all vertices in G'.
- An edge uv is in G' if and only if b(u) is different from b(v).
- In other words, G' = (V_0 U V_1, V_0xV_1 int. E(G)).
- Let us now bound |E(G')|.
- Let X_{uv} be a random variable that indicates the event {uv in E(G')}.
- E[X_{uv}] = Pr({uv in E(G')}) = 1/2.

Proof by Existence

- The random experiment we perform is to assign a bit 0 or 1, denoted b(v), to each vertex v of G ind. and uar.
- An edge uv is in G' if and only if b(u) is different from b(v).
- Let us now bound |E(G')|.
- Let X_{uv} be a random variable that indicates the event {uv in E(G')}.
- $E[X_{uv}] = \Pr(\{uv \text{ in } E(G')\}) = 1/2$.
- Let $X = \sum_{uv \text{ in } E(G)} X_{uv}$.
- E[X] =

Proof by Existence

- The random experiment we perform is to assign a bit 0 or 1, denoted b(v), to each vertex v of G ind. and uar.
- An edge uv is in G' if and only if b(u) is different from b(v).
- Let us now bound |E(G')|.
- Let X_{uv} be a random variable that indicates the event {uv in E(G')}.
- $E[X_{uv}] = Pr(\{uv \text{ in } E(G')\}) = 1/2$.
- Let $X = \sum_{uv \text{ in } E(G)} X_{uv}$.
- $E[X] = E[\sum_{uv \text{ in } E(G)} X_{uv}] = \sum_{uv \text{ in } E(G)} E[X_{uv}] = |E(G)|/2$.

Proof by Existence

- The random experiment we perform is to assign a bit 0 or 1, denoted b(v), to each vertex v of G ind. and uar.
- Let X_{uv} be a random variable that indicates the event {uv in E(G')}.
- $E[X_{uv}] = Pr(\{uv \text{ in } E(G')\}) = 1/2$.
- Let $X = \Sigma_{uv \text{ in } E(G)} X_{uv}$.
- $E[X] = E[\Sigma_{uv \text{ in } E(G)} X_{uv}] = \Sigma_{uv \text{ in } E(G)} E[X_{uv}] = |E(G)|/2$.
- By Statement (1) earlier, there must exist an assignment of b() to vertices such that G' has at least half the edges of G.