Advanced Algorithmics

Strategies for Tackling Hard Problems

Sebastian Wild

Markus Nebel

Lecture 12

2017-05-29

Error Bounds Matter

Remark 4.17 (Success Probability)

From the point of view of complexities, the success probability bounds are flexible:

- $\mathcal{BPP}$ only requires success probability $\frac{1}{2} + \varepsilon$, but using *Majority Voting*, we can also obtain any fixed success probability $\delta \in (\frac{1}{2}, 1)$, so we could also define $\mathcal{BPP}$ to require, say, $\Pr\big[A(x) = [x \in L]\big] \geq \frac{2}{3}$.
- Similarly for $\mathcal{ZPP}$, we can use probability amplification on Las Vegas algorithms to obtain any success probability $\delta \in (\frac{1}{2}, 1)$.

But recall: this is *not* true for unbounded errors and class $\mathcal{PP}$.
In fact, we have the following result.

Theorem 4.18 (PP can simulate nondeterminism)

$\mathcal{NP} \cup \text{co-}\mathcal{NP} \;\subseteq\; \mathcal{PP}$. ◀

$\rightsquigarrow$ Useful algorithms must avoid unbounded errors.

Proof: $\mathcal{PP}$ allows for poly-time

↳ use any poly-time reduction as preprocessing

Show: $SAT \in \mathcal{PP}$ suffices $\mathcal{NP} \subseteq \mathcal{PP}$

$TAUT \in \mathcal{PP}$ similar $co\mathcal{NP} \subseteq \mathcal{PP}$

Given φ of length n with k variables

A (1) Generate random assignment $\alpha \in \{0,1\}^k$ uniformly from $\{0,1\}^k$ $\Big\}$ k rb

(2) If α satisfies φ $\rightsquigarrow$ accept $\Big\}$ linear in n

(3) Otherwise accept iff $\mathcal{B}(p) = 1$

where $p = \frac{1}{2} - \frac{1}{2^{k+1}} = \frac{2^k - 1}{2^{k+1}} < \frac{1}{2}$ $\Big\}$ $k+1$ rb

Time: linear A poly-time $\underline{\mathrm{E}}$-MC

Correctness ; $L(A) = SAT$

- $\varphi \in SAT \rightsquigarrow \Pr[\alpha(\varphi) = 1] \geq \frac{1}{2^k}$ (at least 1 sat. assign.)

$$\Pr[A(\varphi) = 0] = \Pr[\alpha(\varphi) = 0] \cdot (1-p)$$

$$\leq \left(1 - \frac{1}{2^k}\right) \cdot \left(\frac{1}{2} + \frac{1}{2^{k+1}}\right)$$

$$= \frac{1}{2} - \frac{1}{2^{2k+1}} < \frac{1}{2}$$

$$\Pr[A(\varphi) = 1] > \frac{1}{2} \quad ✓$$

- $\varphi \notin SAT \rightsquigarrow \Pr[\alpha(\varphi) = 0] = 1$

$$\Pr[A(\varphi) = 0] = 1 \cdot (1-p) > \frac{1}{2} \quad ✓$$

$\rightarrow$ A indeed UB-MC $\rightsquigarrow$ $SAT \in \mathcal{PP}$

$\square$

One-sided errors

In many cases, errors of MC algorithm are only *one-sided*.

Example: (simplistic) randomized algorithm for SAT
Guess assignment, output $[\phi \text{ satisfied}]$.
(NB: This is not a MC algorithm, since we cannot give a fixed error bound!)

Observation: No false positives; unsatisfiable ϕ always yield 0.
. . . does this help?

Definition 4.19 (One-sided error Monte Carlo algorithms)

A randomized algorithm A for language L (i.e., for $f(x) = [x \in L]$) is a one-sided-error Monte-Carlo (OSE-MC) algorithm if we have

1. $\Pr[A(x) = 1] \ge \frac{1}{2}$ for all $x \in L$, and
2. $\Pr[A(x) = 0] = 1$ for all $x \notin L$.

Definition 4.20 (RP, co-RP)

The classes $\mathcal{RP}$ and co-$\mathcal{RP}$ are the sets of all languages L with a poly-time OSE-MC algorithm for L resp. $\overline{L}$.

Theorem 4.21 (Complementation feasible → errors avoidable)

$\mathcal{RP} \cap \text{co-}\mathcal{RP} \;=\; \mathcal{ZPP}$.

Note the similarly to the open problem $\boxed{\mathcal{NP} \cap \text{co-}\mathcal{NP} \stackrel{?}{=} \mathcal{P};}$
...a first hint that randomization might not help too much?

PRIMES $\in$ co-NP (divisor certifies $n \notin$ Primes)

PRIMES $\in$ NP (fancy certificate for primes)

" $\in$ P (proven, not used)

Proof Thm 4.21

"$\supseteq$" trivial: LV algo $\rightsquigarrow$ one-sided error method

A LV $\qquad A(x) = ? \rightsquigarrow 0$ for RP

$\qquad\qquad\qquad\qquad 1$ co-RP

"$\subseteq$" $L \in RP \cap co\text{-}RP \rightsquigarrow A, \bar{A}$ OSE-MC poly-time for L and $\bar{L}$

B: Run $A(x), \bar{A}(x)$

- $\underline{A(x) = 1}, \bar{A}(x) = 0 \quad \to$ accept
- $A(x) = 0, \underline{\bar{A}(x) = 1} \quad \to$ reject
- $A(x) = 0, \bar{A}(x) = 0 \quad \to \; ? \qquad$ prob $\leq \frac{1}{2}$

($A(x) = 1 = \bar{A}(x)$ $\lightning$ cannot happen)

B: LV for L, poly-time $\rightsquigarrow L \in ZPP$ $\qquad \square$.

Derandomization

Trivial observation: If $Random_A(n) \le c \operatorname{ld} n$, there are only $2^{Random_A(n)} = n^c$ different computations.

$\rightsquigarrow$ We can simply execute all of them sequentially in poly-time!

- limited independence

We can extend this to more randomized bits using *pseudorandom generators*, i.e., algorithms that use a limited amount of real randomness and compute from this a much longer sequence of bits that look random (pseudorandom) to *every* efficient algorithm.

It is not proven that such a method exists, but under widely believed assumptions on circuit complexity lower bounds, there is such a pseudorandom generator that allows to derandomize $\mathcal{BPP}$.

$\rightsquigarrow$ Current belief is $\mathcal{BPP} = \mathcal{P}$ ... and hence $\mathcal{BPP} = \mathcal{RP} = \text{co-}\mathcal{RP} = \mathcal{ZPP} = \mathcal{P}$ (!)

For solving hard problems in theory, randomization does not help at all!

(or: no sufficiently strong lower bound techniques known!)

4.5 Examples of Randomized Algorithms

⇝ Focus on practical benefits of randomization

Randomized approaches can be grouped into categories:

1. Coping with adversarial inputs
   Randomized Quicksort, randomized BSTs, Treaps, skip lists
2. Abundance of Witnesses many certificates ⇝ guess one and check
   Solovay-Strassen primality test
3. Fingerprinting reduce universe and accept collisions
   universal hashing
4. Random Sampling know "good" structures exist, can draw one with sufficient prob.
   Perfect hashing
5. LP Relaxation & Randomized Rounding
   Set-Cover Approximation (next chapter)

① Coping with worst-case inputs

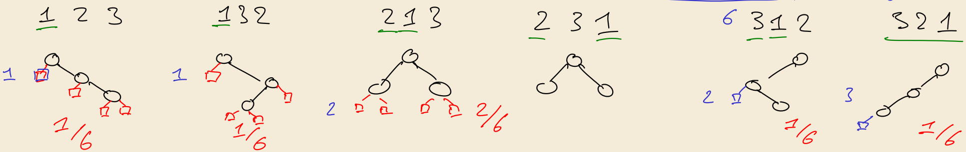
Naturally Grown BSTs

First: Ignore adversaries.

Naturally grown / Random BST: all $n!$ insertion orders *equally likely*.

Example $n = 3$

$$\frac{1+1+2+2+2+3}{6} = \frac{11}{6}$$



Lemma 4.22 (Random insertion yields random BST)

Let $n \geq 0$ be arbitrary and let T_n be a random BST over n keys. Inserting an element equally likely in one of the $n+1$ *gaps* in T_n (external leaves) results in a new BST T_{n+1} that has the same shape as a random BST of $n+1$ keys.

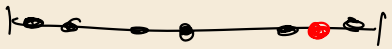
insertion order in T_{n+1} $\cong$ permutation of $[n+1]$

$\cong$ permutation of $[n]$ plus the "last value" in $[n+1]$

$(1\ 2\ 3\ 6\ 5\ 4,\ 3) \rightsquigarrow 1\ 2\ 4\ 7\ 6\ 5\ 3$

Corollary 4.23

A BST built by inserting n i.i.d. $\mathcal{U}(0,1)$ r.v. has the shape of a random BST.

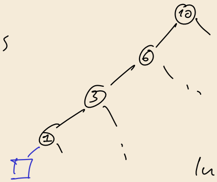


Theorem 4.24 (Expected Depth of leftmost leaf)

The *expected depth* (number of edges from root) of the leftmost external leaf (leaf for $-\infty$) in a random BST on $n \geq 1$ nodes is H_n.

$= \sum_{i=1}^{n} \frac{1}{i}$ $\qquad H_0 = 0 \qquad H_3 = 1 + \frac{1}{2} + \frac{1}{3} = \frac{6+3+2}{6} = \frac{11}{6}$

Proofs



depth = # left-right minima

labels on path = left-to-right minima

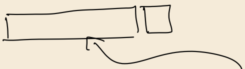
Induction on n to show $\mathbb{E}[\#\text{left-to-right mins}] = H_n$

$n=0 \qquad 0=0$

$n=1 \qquad 1=1$

left-to-right min $\iff$ min

n



$\Pr[\text{min}] = \frac{1}{n}$

$\mathbb{E}[\#\text{left-to-right min}] \underset{IH}{=} H_{n-1} + \frac{1}{n} = H_n \quad \square.$

BSTs have $O(\log n)$ height w.h.p.

(because Quicksort's recursion trees $\cong$ random BSTs)