Randomness and Computation

or, “Randomized Algorithms”

Heng Guo

(Based on slides by M. Cryan)

School of Informatics
University of Edinburgh

Lectures and tutorials

Lectures:

- 11:10-12:00 Tuesday, room 2.3 of the Lister Centre;
- 11:10-12:00 Friday, room 1.3 of the Lister Centre.

Tutorials (choose one):

- 12:10-13:00 Tuesday, G203 Teaching Room 2, Doorway 3, Medical School;
- 12:10-13:00 Wednesday, room 1.4 of the Lister Centre.

Lecturers

- Heng Guo, IF 5.05A (Weeks 1-5)
  Email: `hguo@inf.ed.ac.uk`
  Webpage: `http://homepages.inf.ed.ac.uk/hguo/`
- Mary Cryan, IF 5.18 (Weeks 6-10)
  Email: `mcryan@inf.ed.ac.uk`
  Webpage: `http://homepages.inf.ed.ac.uk/mcryan/`

Randomness and Computation

- Interested in what we can compute (exactly, approximately) when we have the option of "tossing coins" in our computation.
- Of course, when introduce some randomness, we no longer have a deterministic algorithm. An algorithm which exploits random choices will *either* show variation in the answer computed *or* in the time-taken to return an answer. Or both!
- Though we will have variation in running-times and/or the answer returned, we will always aim to calculate the *expected* running-times, *expected value returned*. Or possibly we will prove *bounds* on running-times and/or values returned.

Syllabus

Introduction, Las Vegas and Monte Carlo algorithms (Simple Examples: checking identities, fingerprinting)

Moments, Deviations and Tail Inequalities (Balls and Bins, Coupon Collecting, stable marriage, routing)

Randomization in Sequential Computation (Data Structures, Graph Algorithms)

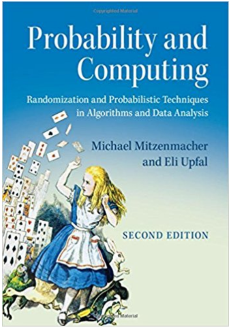
Randomization in Parallel and Distributed Computation (algebraic techniques, matching, sorting, independent sets)

The Probabilistic Method (threshold phenomena in random graphs, Lovász Local Lemma)

Derandomisation (Use of conditional expectation to derandomise some algorithms)

Random Walks and Markov Chains (hitting and cover times, Markov chain Monte Carlo, mixing times)

Textbook (essential for the course)



Probability and Computing - Randomized Algorithms and Probabilistic Analysis, by Michael Mitzenmacher and Eli Upfal; Cambridge University Press, 2017 (2nd ed).

- Blackwell's on South Bridge has a number of copies of the 2nd ed.
- You are welcome to work with edition 1 if you can find a cheaper copy.

Course Webpage

Slides will be provided for each lecture (and notes sometimes if appropriate) on the course webpage:

`http://www.inf.ed.ac.uk/teaching/courses/rc/`

Recordings of lectures (slides and voice) will be on Learn.

You will need the book too!

Pre-requisites

Good news: no formal requirement.

However, strong maths is necessary, especially *Discrete Maths* and *confidence in proving things*.

Please take the "Self test" on the course webpage to assess whether you are prepared for this course.

I expect you to have covered an "Algorithms class" in the past, and to have done well in it (can waive that if your Maths is very strong).

If you're not sure, come and speak to me.

Math you should know

You should know:

- what it means to *prove* a theorem (induction, proof by contradiction, etc ...) and to be confident in your ability to do this.
- The definitions of the main categories of asymptotic operators $O(\cdot)$, $\Omega(\cdot)$, $\Theta(\cdot)$, and how to reason about them.
- How to multiply matrices or polynomials, also basic linear algebra.
- Some probability theory, definition of expectation (1st moment) and variance (related to 2nd moment), linearity of expectation, simple probabilistic distributions and how they behave.
- Some graph theory.

Your own work (formative assessment)

- 4-5 tutorial sheets

  5 tutorials sprinkled through semester, weeks 4, 6, 7, 8, 10

- Office hours

  Heng Guo: By appointment

  Mary Cryan: 10:30-11 Tuesday, or 12-12:30 Friday

- Coursework 1 (due Thursday of week 5)

  The first coursework for RC will be read and commented-on by us; however it will not be "for credit". It is to give you experience solving problems and doing small proofs.

Coursework (summative assessment)

We have 2 Courseworks (problem-solving and proofs), and both will be marked to give you feedback. Coursework 1 is "just for feedback", and Coursework 2 will be worth 20% of the course mark. Details are:

- Coursework 1. "Feedback-only"
  - OUT Thurs, 30th Jan (Thurs week 3)
  - DUE **4pm** Thurs, 13th Feb (Thurs week 5)
  - FEEDBACK by Thurs, 27th Feb (Thurs week 6)
- Coursework 2. "Worth 20%"
  - OUT Tue, 3rd March (Tue week 7)
  - DUE **4pm** Tue, 17th March (Tue week 9)
  - FEEDBACK by Tue, 31st March (Tue week 11)

Feedback given will include marks to individual sub-parts of questions, comments on scripts to explain why marks were lost, plus a description of common errors.

Common marking scheme

Marking follows the University's Common Marking Scheme (Links can be found online, too long to fit here).

Key points:

- A: $>$70
- Fail: $<$40

Verifying polynomial identities

Suppose we are given two polynomials $F(x)$ and $G(x)$, where $F(x)$ is expressed as a product of d "monomials" and $G(x)$ is given as an expansion of x^i terms, with degree at most d.

How much time does it take to verify *whether $F(x) \stackrel{?}{\equiv} G(x)$*

(A "monomial" is a term of the form $(x - a)$, for some value a.)

For example,

- $F(x) = (x-1)(x+2)(x-3)(x+4)(x-5)(x+6)$;
- $G(x) = x^6 - 7x^3 + 720$.

Simple "multiply out" algorithm on $F(x)$ (uses no randomness) gives the answer in $\Theta(d^2)$ time. (Each addition or multiplication is one step.)

Other (deterministic) algorithm uses FFT to "multiply out" in $\Theta(d \cdot \lg^2(d))$.

We will use randomness to test equivalence *without multiplying out $F(x)$*.

Testing polynomial identities using random sampling

- We will choose a value for x_0 *uniformly at random* from the set of integers $\{1, \ldots, 100d\}$.
- Then we will calculate $F(x_0)$. For each monomial we do 1 addition and 1 multiplication.

  Overall this takes at most d additions and d multiplications.
- We also calculate $G(x_0)$. We first do d multiplications to get all of x_0, x_0^2, x_0^3, ..., x_0^d. Then we multiply each term with its coefficient and add everything up.

  Overall this takes at most d additions and $2d$ multiplications.
- Next compare the two numbers ... answering "yes" if they are the same, "no" otherwise.

uniformly at random (uar) - every item has the same chance.

Monte Carlo algorithm

This is a *Monte Carlo* algorithm (coined by Stan Ulam), meaning that with some prob. it will give a wrong answer.

The error is *one-sided*.

- If $F(x)$ does equal $G(x)$, "yes" is always returned.
- If $F(x) \not\equiv G(x)$, "no" is returned with probability $\frac{99}{100}$ (failure probability $\leq \frac{1}{100}$).

Testing polynomial identities

The probability of the algorithm giving a wrong answer ("yes" when it should be "no") equals

$$\frac{|\{x : F(x) = G(x)\} \cap \{1, \ldots, 100d\}|}{100d} \leq \frac{|\{x : F(x) = G(x)\}|}{100d}$$

If $F(x) \not\equiv G(x)$, the set $\{x : F(x) = G(x)\}$ is equal to the set of roots of $(F - G)(x)$, namely, $\{x : (F - G)(x) = 0\}$.

$\#$ of roots of a polynomial $\leq$ its degree, and the degree of $F - G$ is at most d.

So error probability $\leq \frac{d}{100d} = \frac{1}{100}$.

Reducing the error probability

One option to improve error rate is to increase the size of the sample set - eg, by sampling a random integer from $\{1, \ldots, 1000d\}$, error probability would drop to $\frac{1}{1000}$ ... this improvement is not "free" though, it's more work to sample from larger sets (not officially costed by us).

Alternatively, suppose we run *two* random trials to test $F(x) \stackrel{?}{\equiv} G(x)$, first drawing x_1 uar from $\{1, \ldots, 100d\}$ and testing $F(x_1) \stackrel{?}{=} G(x_1)$, next drawing x_2 uar from $\{1, \ldots, 100d\}$ and testing whether $F(x_2) \stackrel{?}{=} G(x_2)$.

We return "yes" if *both* calculations give matching values, otherwise we return "no".

Refining the verification of polynomial identities

Observation

This refined algorithm again gives one-sided error*:*

- *If* $F(x) \equiv G(x)$*, certainly we will see that* $F(x_1)$ *matches* $G(x_1)$, and *that* $F(x_2)$ *matches* $G(x_2)$ *(answer "yes").*
- *If* $F(x)$ *and* $G(x)$ *are non-identical, we will show the algorithm returns "no" most of the time, with failure probability at most* $\left(\frac{1}{100}\right)^2$*.*

Refining the verification of polynomial identities (analysis)

Two options for "repeated sampling" from $\{1, \ldots, 100d\}$ (or any discrete set): *with replacement* or *without replacement*.

with replacement: We draw the random value x_2 uniformly at random from $\{1, \ldots, 100d\}$ (including x_1 as an option).
For this case, the two *events* of "generating x_1" and "generating x_2" are *mutually independent*.

Definition (1.3)

The two events A and B are said to be *mutually independent* if and only if

$$\Pr[A \cap B] = \Pr[A] \cdot \Pr[B].$$

Refining the verification of polynomial identities (analysis)

with replacement (cont'd): Recall that if $F(x) \not\equiv G(x)$, then $(F - G)(x)$ has *at most* d roots; hence there are *at most* d values in $\{1, \ldots, 100d\}$ that could give matching values for $F(x)$, $G(x)$.

If H_1 is the event that "a root of $(F - G)(x)$" is generated on this first trial, then $\Pr[H_1] \leq d/100d = (1/100)$.

But sampling with replacement, the outcomes of the 2nd trial are *independent* of what happened before. So H_2 (the probability of generating a root of $(F - G)(x)$ on the 2nd trial) is *independent* of H_1. Also it happens to have identical probability.

The probability that *both* experiments would draw a root of $(F - G)(x)$ is (by Defn 1.3) equal to

$$\Pr[H_1] \cdot \Pr[H_2] \leq \frac{1}{100} \cdot \frac{1}{100},$$

which is $1/100^2 = 1/10000$.

Refining the verification of polynomial identities (analysis)

without replacement: We have already tested x_1 and found $F(x_1), G(x_1)$ to match (else we'd finish, with "no"). For H_2 we will draw a value from $\{1, \ldots, 100d\} \setminus \{x_1\}$.

Events H_1 and H_2 are no longer independent, H_2 is *conditional on* H_1.

Definition (1.4)

The *conditional probability* of event A conditional on event B having happened is

$$\Pr(A \mid B) \ = \frac{\Pr[A \cap B]}{\Pr[B]}.$$

Refining the verification of polynomial identities (analysis)

without replacement (cont'd): In applying Definition 1.4, E is H_1 and F is H_2. We want to calculate $\Pr[H_1 \cap H_2]$ (two samples both giving a false match). This is $\Pr[H_1] \cdot \Pr[H_2 \mid H_1]$.

We know $\Pr[H_1] \leq \frac{1}{100}$.

For $\Pr[H_2 \mid H_1]$, note that since H_1 occurred (and the integer removed was a match), we have one less root ($d' - 1$ instead of d', say) remaining in the set

$$\{1, \ldots, 100d\} \setminus \{x_1\}.$$

Hence $\Pr[H_2 \mid H_1] = \frac{d'-1}{100d-1}$. Then

$$\Pr[H_1 \cap H_2] \; = \; \Pr[H_1] \cdot \Pr[H_2 \mid H_1] \; \leq \; \frac{d'}{100d} \cdot \frac{d'-1}{100d-1} \; < \; \frac{1}{100^2},$$

where we use $d' \leq d$ to show $\frac{d'-1}{100d-1} < \frac{1}{100}$.

Refining the verification of polynomial identities (wrapup)

Can similarly consider carrying out k different trials of values sampled from $\{1, \ldots, 100d\}$.

- Will be able to show "one-sided error" of at most $1/100^k$.
- The probability of failure (returning "yes" when $F(x), G(x)$ are non-identical) is always a bit better in the "without replacement" case).
- This iterated testing algorithm will take $\Theta(k \cdot d)$ time.
- No point doing more than d iterations (why?)

Reading Assignment

Start reading Chapter 1 of “Probability and Computing” in preparation for lecture 2.