Introduction to Algorithms

Chapter 5: Randomized algorithms

Manuel

Fall 2017

Outline

So far...

Along chapter 2, section 3 (2.39) we studied:

- How to model computers
- What an algorithm is
- Various classes of problems

So far...

Along chapter 2, section 3 (2.39) we studied:

- How to model computers
- What an algorithm is
- Various classes of problems

Computation on a deterministic Turing machine depends on:

- The state of the machine
- The symbol read on the tape

For any situation at most one action is performed

Non-deterministic Turing machine

Computation on a non-deterministic Turing machine:

- The machine has a state and a symbol is read on the tape
- The machine branches into many copies
- The machine transitions into one of the copies

When the machine is run more than once:

- Different paths are chosen
- Computation cannot be exactly reproduced

Non-deterministic Turing machine

Computation on a non-deterministic Turing machine:

- The machine has a state and a symbol is read on the tape
- The machine branches into many copies
- The machine transitions into one of the copies

When the machine is run more than once:

- Different paths are chosen
- Computation cannot be exactly reproduced

Remark.
A non-deterministic Turing machine can be simulated by a deterministic Turing machine with three tapes.

Deterministic vs. probabilistic

Two main types of algorithms:

- Deterministic: follow a fixed sequence of steps; output is completely determined by the input
- Probabilistic: add some randomness to the process
  - Monte Carlo algorithm: returns either `True` or unknown; increasing the number of test decreases the probability of having a "false positive"
  - Las Vegas algorithms: always returns a correct result; the running time is however random; the time complexity cannot be precisely evaluated

Example

Quick sort:

- Worst case: $\mathcal{O}(n^2)$ vs. average case $\mathcal{O}(n \log n)$
- Fixed pivot: if the list to sort is originally structured worst complexity is likely to apply
- Random pivot: even if the list to sort is originally structured choosing a random pivot is equivalent to sorting with a fixed pivot on a randomly ordered list
- Running time: random depending on the choice of the pivot

Example

Primality testing:

- AKS: deterministic in time $\tilde{\mathcal{O}}(n \log^{12} n)$
- Miller-Rabbin:
  - Complexity: $\tilde{\mathcal{O}}(k \log^2 n)$, with k the number of witnesses
  - Returns composite or unknown
  - Unknown means prime with probability $1 - 4^{-k}$

Formalization

Definitions

1. A non-deterministic Turing machine which chooses a random transitions according to some probability distribution is called a *probabilistic Turing machine*.
2. A language L is in the Bounded-error Probabilistic Polynomial time complexity class (BPP) if and only if there is a probabilistic Turing machine M such that
   - For any input, M runs in polynomial time;
   - For all x in L, M returns 1 with probability larger than $1/2 + \varepsilon,\ \varepsilon > 0$;
   - For all x not in L, M returns 1 with probability less than $1/2 - \varepsilon,\ \varepsilon > 0$;

Remark.
In practice instead of taking $1/2 + \varepsilon$, $2/3$ and $1/3$ are often chosen.

Probabilistic or deterministic

Reasons for using probabilistic algorithms:

- No polynomial time deterministic algorithm is known
- Available polynomial time deterministic algorithms are slower

Open questions:

- Is $\mathcal{P} =$BPP?
- Are probabilistic algorithm more powerful than deterministic onces?
- What is the relative power of probabilistic and deterministic computations?

Outline

The polynomial identity testing problem

Problem (Polynomial Identity Testing (PIT))

Let P be a multivariate polynomial over some field. Decide whether P is identically zero.

Remark.
Identically zero means that after expanding P it reduces to zero. In particular this is a different problem from the Evaluate to Zero Everywhere (EZE) problem where one want to decide if an n-multivariate polynomial evaluates to zero for all x_i, $1 \leq i \leq n$.

Univariate case

The univariate case can be easily solved:

- The polynomial is given as a sum of monomials:
  - Reduce the sum
  - Check all the monomials and return `True` if they all equal zero
- The polynomial P of degree d is in a more complex form:
  - Arbitrarily choose $d+1$ points (e.g. $0, \cdots, d$)
  - Evaluate P at those $d+1$ points
  - Conclude that $P = 0$ if and only if they all evaluate to zero

Univariate case

The univariate case can be easily solved:

- The polynomial is given as a sum of monomials:
  - Reduce the sum
  - Check all the monomials and return `True` if they all equal zero
- The polynomial P of degree d is in a more complex form:
  - Arbitrarily choose $d+1$ points (e.g. $0, \cdots, d$)
  - Evaluate P at those $d+1$ points
  - Conclude that $P = 0$ if and only if they all evaluate to zero

Why is this not working anymore in the case of a multivariate polynomial?

Schwartz-Zippel Lemma

Definition

1. A *monomial* is an expression of the form $\alpha \prod_{i=1}^{n} x_i^{\beta_i}$, where α is an element from a base field, the x_i are n variables, and the β_i are positive integers.
2. The *total degree of a monomial* is $\sum_i \beta_i$.
3. The *total degree of a polynomial* is the largest total degree among all the monomials composing the polynomial.

Schwartz-Zippel Lemma

Definition

1. A *monomial* is an expression of the form $\alpha \prod_{i=1}^{n} x_i^{\beta_i}$, where α is an element from a base field, the x_i are n variables, and the β_i are positive integers.
2. The *total degree of a monomial* is $\sum_i \beta_i$.
3. The *total degree of a polynomial* is the largest total degree among all the monomials composing the polynomial.

Lemma (Schwartz-Zippel)

Let P be a n-multivariate polynomial of total degree d, that is not identically zero, over a field $\mathbb{F}$. For $y_1, \cdots, y_n$, chosen uniformly and independently from a finite set $S \subset \mathbb{F}$,

$$\Pr\left[P(y_1, \cdots, y_n) = 0\right] \leq \frac{d}{|S|}.$$

Schwartz-Zippel Lemma

Proof.

We proceed by induction on the number of variables n.

Base case: For $n = 1$ the results is clear as a univariate polynomial of degree d has at most d roots.

Induction step: we assume that the result is true for an $(n-1)$-multivariate polynomial and prove it is also true in the case of n variables.

Let k be the largest power of X_1 in any monomial composing P.

Then

$$P(X_1, \cdots, X_n) = \sum_{i=0}^{k} X_1^i Q_i(X_2, \cdots, x_n).$$

By construction, Q_k is not identically zero, its total degree is at most $d - k$, and it has $n - 1$ variables. Therefore by the induction hypothesis we get

$$\Pr\left[Q_k(y_2, \cdots, y_n) = 0\right] \leq \frac{d-k}{|S|}.$$

Schwartz-Zippel Lemma

Proof (continued).

For $y_2, \cdots, y_n$ in $\mathbb{F}$, we call $\mathcal{E}_1$ the event $Q_k(y_2, .., y_n) = 0$. Selecting $y_2, \cdots, y_n$ such that $\mathcal{E}_1$ does not occur, we define $R(X_1)$ to be the polynomial

$$R(X_1) = \sum_{i=0}^{k} X_1^i Q_i(y_2, \cdots, y_n) = P(X_1, y_2, \cdots, y_n).$$

Clearly $R(X_1)$ is not identically zero since $\mathcal{E}_1$ did not occur, meaning that X_1^k has a non zero coefficient. Therefore

$$\Pr\left[R(y_1) = 0 \mid \neg\mathcal{E}_1\right] \leq \frac{k}{|S|}.$$

Let $\mathcal{E}_2$ be the event $R(y_1) = 0$, which can also be stated as $P(y_1, \cdots, y_n) = 0$. In order to prove the lemma it remains to bound $\Pr[\mathcal{E}_2]$.

Schwartz-Zippel Lemma

Proof (continued).

As we have already bounded $\Pr[\mathcal{E}_2 \mid \neg\mathcal{E}_1]$ and $\Pr[\mathcal{E}_1]$ we can rewrite $\Pr[\mathcal{E}_2]$ as

$$\begin{aligned}\Pr[\mathcal{E}_2] &= \Pr[\mathcal{E}_2 \wedge \mathcal{E}_1] + \Pr[\mathcal{E}_2 \wedge \neg\mathcal{E}_1] \\ &= \Pr[\mathcal{E}_2 \wedge \mathcal{E}_1] + \Pr[\mathcal{E}_2 \mid \neg\mathcal{E}_1] \Pr[\neg\mathcal{E}_1] \\ &\leq \Pr[\mathcal{E}_1] + \Pr[\mathcal{E}_2 \mid \neg\mathcal{E}_1] \\ &\leq \frac{d-k}{|S|} + \frac{k}{|S|} = \frac{d}{|S|}\end{aligned}$$

Hence, by the induction principle Schwartz-Zippel lemma holds.

□

Remark.

Schwartz-Zippel lemma (5.13) says that when evaluating P at a random point there is a very low probability of finding a root. It however does not mean that a polynomial over $\mathbb{R}$ has finitely many roots. For instance $P(X_1, X_2) = X_1$ has infinitely many roots.

Solving PIT

Algorithm.

Input : $P(X_1, \cdots, X_n)$ of degree d and a field $\mathbb{F}$, l and k
Output: not zero or probably zero

```
1 Select a subset S ⊂ 𝔽 of size ≥ ld;
2 for i ← 1 to k do
3 |   (y_1, ··· , y_n) ← rand(S);
4 |   if P(y_1, ··· , y_n) ≠ 0 then return not zero;
5 end for
6 return probably zero;
```

Solving PIT

Algorithm.

Input : $P(X_1, \cdots, X_n)$ of degree d and a field $\mathbb{F}$, l and k
Output: not zero or probably zero

```
1 Select a subset S ⊂ 𝔽 of size ≥ ld;
2 for i ← 1 to k do
3 |   (y_1, ··· , y_n) ← rand(S);
4 |   if P(y_1, ··· , y_n) ≠ 0 then return not zero;
5 end for
6 return probably zero;
```

Theorem

Let $P(X_1, \cdots, X_n)$ be a polynomial. If $P(X_1, \cdots, X_n)$ is not identically zero, then algorithm 5.17 returns the correct result with probability at least $1 - 1/l^k$.

Solving PIT

Proof.
For each iteration of the loop the probability of testing a root is at most $1/I$. Each choice of element in S is independent of the previous onces, such that the probability of passing all the tests while $P(X_1, \cdots, X_n)$ is not zero is at most $1/I^k$.

Hence the probability of returning "probably zero" while $P(X_1, \cdots, X_n)$ is not identically zero is at least $1 - 1/I^k$.

□

Remark.
If the field $\mathbb{F}$ has less elements than the number of roots of $P(X_1, .., X_n)$ then Schwartz-Zippel lemma (5.13) is of no use. It is however possible to overcome this problem by applying algorithm 5.17 to the polynomial $P(X_1, .., X_n)$ over an extension field of $\mathbb{F}$.

Outline

The Min-Cut Problem

Problem (Min-Cut)

Given a connected multi-graph G, determine the minimum number of edges that must be removed such that G becomes disconnected.

Common applications:

- Split a problem for parallel programming
- Optimize a divide and conquer strategy
- Segment an image into regions of similar color/texture

Karger's min-cut algorithm

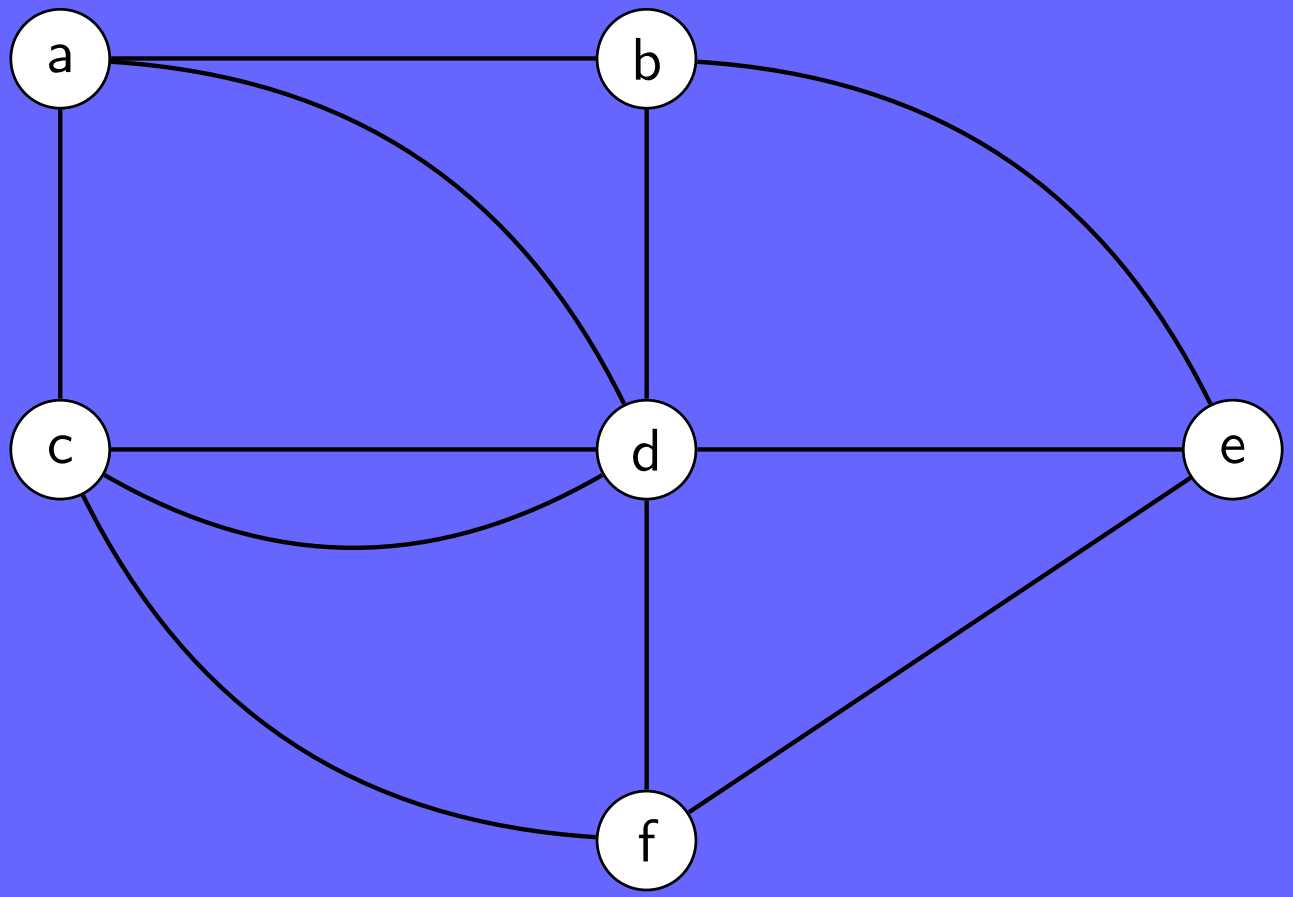
Algorithm. (*Karger*)

Input : a graph $G = \langle V, E \rangle$
Output : an upper bound on the min-cut

```
1  Function Contract(G, e):
2  |   remove all edges between e.s and e.t;
3  |   foreach edge e' containing e.s do
4  |   |   disconnect e' from e.s and reconnect it to e.t;
5  |   end foreach
6  |   S_{e.t} ← S_{e.t} ∪ S_{e.s}; remove vertex e.s from V;
7  |   return G;
8  end
9  foreach v ∈ G.V do S_v = {v};
10 while G has more than two vertices do
11 |   e ←rand(G.E); G ← Contract(G, e);   /* edge e = (e.s, e.t)
   |    */
12 end while
13 return |G.E|;
```

Exercise

Apply Karger's algorithm to the following graph G and find two different final values for $|G.E|$.



Edge contraction

The process described in the function Contract of Karger's algorithm (5.21) is called *edge contraction*.

As highlighted in the previous exercise (5.22) running Karger's algorithm can lead to various values. The question is then to figure out how to retrieve the right one with high probability. In order to do this we first evaluate the cost of a contraction.

Lemma

Let $G = \langle V, E \rangle$ be a multigraph. A single contraction in Karger's algorithm takes $\mathcal{O}(|V|^2)$ time.

Proof.

Assuming G is represented using a linked list at each vertex, a contraction consists in merging two lists, while ensuring no loop subsists. Therefore the list is of size $\mathcal{O}(|V|^2)$. □

Probability of finding a min-cut

Lemma

Let $G = \langle V, E \rangle$ be a multigraph, and (S, T) be a min-cut. Then

$$\Pr\left[\text{Karger's algorithm ends with } (S, T)\right] \geq \frac{1}{\binom{|V|}{2}}.$$

Proof.
First observe that if an edge (u, v) is contracted then only the cuts containing both u and v are unchanged. Therefore for Karger's algorithm to succeed the minimum cut (S, T) must remain untouched over all the random edge selections. Denoting the number of vertices by n, $n-2$ contractions are to be performed in order to have only two vertices left. Naming those edges $\{e_1, \cdots, e_{n-2}\}$, the goal is to determine the probability to chose a proper edge at each iteration of the algorithm.

Probability of finding a min-cut

Proof (continued).

Let k denote the size of a minimum cut in G. Then the minimum degree of any vertex in G is at least k, otherwise a cut of size less than k could be exhibited by disconnecting a vertex of degree less than k. Therefore G has at least $nk/2$ edges.

After a contraction the new graph has one less vertex but the degree of all the vertices remains at least k. So after i steps there are $n - i$ vertices and at least $(n - i)k/2$ edges.

Since any vertex has degree at least k the probability of selecting a "bad edge", assuming none has been chosen before, is $2/(n - i)$. Hence we can determine the probability of never choosing a "bad edge" during the whole process and thus end with (S, T).

Probability of finding a min-cut

Proof (continued).

This probability is given by

$$\begin{aligned}
\Pr\left[\text{find } (S,T)\right] &= \Pr\left[e_1, \cdots, e_{n-2} \notin (S,T)\right] \\
&= \Pr\left[e_1 \notin (S,T)\right] \prod_{i=1}^{n-3} \Pr\left[e_{i+1} \notin (S,T) \mid e_1, \cdots, e_i \notin (S,T)\right] \\
&\geq \prod_{i=0}^{n-3} \left(1 - \frac{2}{n-i}\right) \\
&= \frac{2}{n(n-1)} = \frac{1}{\binom{n}{2}}.
\end{aligned}$$

□

Probability of finding a min-cut

Since the probability of finding a min-cut is least $1/\binom{n}{2}$ it suffices to run the algorithm $l\binom{n}{2}$, for some value l. The probability of a run to succeed is then at least

$$1 - \left(1 - \frac{1}{\binom{n}{2}}\right)^{l\binom{n}{2}} \geq 1 - e^{-l}.$$

Therefore an appropriate choice for l is $c \ln n$, which leads to an error probability of at most $1/n^c$. Hence the total running time of Karger's algorithm is $\mathcal{O}(n^4 \log n)$.

Although this approach works well, it remains slow. The main reason is the random choice of the edge: at the beginning the multigraph features many edges and the probability of selecting an edge from a minimum cut is low. However as the process advances the probability of contracting an edge in the minimum cut grows.

Karger-Stein's algorithm

Algorithm. (*Karger-Stein*)

Input : A graph $G = \langle V, E \rangle$
Output : a mini-cut in G

```
1  Function FastCut(G):
2      if |G.V| > 6 then
3          G1 ← G; G2 ← G;
4          t ← 1 + |G.V|/√2;
5          while |G1.V| ≥ t do e ←rand(G1.E); G1 ←
             Contract(G1, e);
6          while |G2.V| ≥ t do e ←rand(G2.E); G2 ←
             Contract(G2, e);
7          FastCut(G1); FastCut(G2);
8      else
9          find the min-cut by enumeration
10     end if
11     return |G1.E| ≤ |G2.E| ? |G1.E| : |G2.E|
12 end
```

Karger-Stein's algorithm

Theorem

Given a multigraph with n vertices, Karger-Stein's algorithm discovers a minimum cut in time $\mathcal{O}(n^2 \log^3 n)$, with high probability.

Sketch of proof.

First note that $6 \ll |V|$ and as such finding a minimum cut by enumeration only impacts the final complexity by a constant factor.

Given a cut, observe that the probability that it survives down to t vertices is at least $\binom{t}{2}/\binom{n}{2}$. Thus for $t = n/\sqrt{2}$ the probability of success is larger than $1/2$.

Since Karger-Stein's algorithm follows a divide and conquer strategy, its complexity can be expressed by a recurrence relation.

Karger-Stein's algorithm

Sketch of proof (continued).

Then recalling that as a single edge contraction costs $\mathcal{O}(n^2)$ (lemma 5.23) we get

$$T(n) = 2\left(n^2 + T\left(\frac{n}{\sqrt{2}}\right)\right).$$

Hence by the master theorem (2.31) we conclude that the running time of Karger-Stein's algorithm is $\mathcal{O}(n^2 \log n)$.

We now consider the success probability. First as we start with n vertices and go down to $t = n/\sqrt{2}$, the success probability is $(t/n)^2 \approx 1/2$. Then at the next recursion level the graph shrinks from $n/\sqrt{2}$ to $n/2$ vertices, which means an overall success probability of about $1/4$.

Karger-Stein's algorithm

Sketch of proof (continued).

More generally assume the minimum cut to still be in the graph and let $P(t)$ be the probability that a call to the algorithm with t vertices successfully computes it. Then G_i, $1 \leq i \leq 2$ still contains it with probability larger than a half. Therefore the probability that a recursive call succeeds is $1/2P(t/\sqrt{2})$. And since two recursive call are performed

$$P(t) = 1 - \left(1 - \frac{1}{2}P\left(\frac{t}{\sqrt{2}}\right)\right)^2.$$

Solving this recurrence relation yields $P(n) = \Omega(1/\log n)$. This means that Karger-Stein's algorithm needs to be run about $\log^2 n$ times in order to have an error probability of at most $\mathcal{O}(1/n)$ of preserving the minimum cut. This gives a final complexity of $\mathcal{O}(n^2 \log^3 n)$.

Final remarks

Randomized algorithms:

- Bring much flexibility compared to deterministic ones
- Are often faster that deterministic ones
- Introduce imprecision on the output or on the complexity
- Require a good knowledge of probability theory
- Have proof that are often complex, even if the algorithm can be simply expressed

Key points

- What is a randomized algorithm?
- What are the two main types of randomized algorithm?
- How to solve the PIT problem?
- Why using a randomized algorithm for the Min-Cut problem?

Thank you!