COMP3630/6360: Theory of Computation
Semester 1, 2022
The Australian National University

Probabilistic Computation and Approximation

The Conundrum

Problem.

We need to solve an NP-complete problem. What can we do?

Approaches

If a problem is NP-complete (or worse . . .)

- Hope that we need to deal with small instances only
- Hope that we don't need to deal with all instances
  - instances we need to deal with might have extra structure
- Don't insist on the *best* solution
- Don't insist on getting it right all the time.

VERTEX-COVER

If $G = (V, E)$ is an undirected graph, a *vertex cover* of G is a subset $C \subseteq V$ where every edge touches one of the nodes in C:

$$\forall (x, y) \in E \, (x \in C \vee y \in C)$$

Theorem 1.1

The problem

$$\textit{VERTEX-COVER} = \{ \langle G, k \rangle \mid G \textit{ has a } k\textit{-node vertex cover} \}$$

is **NP***-complete.*

Proof.

We show $\textit{3SAT} \leq_{\mathbf{P}} \textit{VERTEX-COVER}$ by transforming 3cnf-formulas into undirected graphs with 2 nodes per variable and 3 nodes per clause. [Details: see Sipser] □

> *An* **NP***-completeness proof is typically the first act of the analysis of a computational problem by the methods of the theory of algorithms and complexity, not the last. Once* **NP***-completeness has been established, we are motivated to explore possibilities that are less ambitious than solving the problem exactly, efficiently, every time.*

(Papadimitriou 1994, Page 299)

Optimisation Problems

In *optimisation problems* we seek the best solution among a collection of possible solutions.

Example 1.2

On input $\langle G \rangle$, where G is an undirected graph, find a smallest vertex cover.
This is **NP**-hard, too. (We did not formalise this notion.)

Approximation Algorithms

$\textsc{Apx}$ = "On input $\langle (V, E) \rangle$, where (V, E) is an undirected graph:

1. Set $M := \emptyset$.
2. While there exists an edge $(x, y) \in E \cap \overline{M}^2$
   1. $M := M \cup \{x, y\}$ — add both vertices to the cover
   2. $V := V \setminus \{x, y\}; E := E \cap (V \times V)$ — remove the vertices
3. Output $\langle M \rangle$".

arguably generates *some* vertex cover for (V, E) in polynomial time. But how close is it to an optimal one?

Theorem 1.3

Apx produces a vertex cover no more than twice as large as a smallest one.

Proof.

Correctness: With every pair of nodes removed in the body of the main loop, we add both nodes to the cover. This implies that when we then remove these nodes from the graph, all edges removed touch a node in the cover.

Approximation: For each pair of nodes removed by an iteration of Apx's main loop, a smallest vertex cover would contain at least one of the removed nodes, otherwise the edge between them wouldn't be covered. □

k-Optimality

Definition 1.4

An approximation algorithm for a minimisation problem is *k-optimal* if it always finds a solution that is at most k times the size of an optimal one.

An approximation algorithm for a maximisation problem is *k-optimal* if it always finds a solution that is at least $\frac{1}{k}$ times the size of an optimal one.

Example 1.5

We've just shown that $\textsc{Apx}$ is 2-optimal for *VERTEX-COVER*.

Traveling Salesman Problem

Given n cities $1, \ldots, n$, and a nonnegative symmetric integer distance $d_{i,j} = d_{j,i}$ between any two cities i and j, we're asked to find the *shortest tour* of the cities—that is, a permutation π such that $\sum_{i=1}^{n} d_{\pi(i),\pi((i \bmod n)+1)}$ is minimal. This problem is called *TSP*.

Theorem 1.6

Unless $\mathbf{P} = \mathbf{NP}$*, there is no* $k \in \mathbb{N}$ *for which there exists a* k*-optimal approximation algorithm for TSP.*

Proof.

Suppose to the contrary that T is k-optimal for *TSP*. Then we can use T to solve *HAMCYCLE* (undirected *HAMPATH* with a closing edge) in **P** by the machine

$H =$ "On input $\langle (V, E) \rangle$, where (V, E) is an undirected graph:

1. Run T on a *TSP* instance with $|V|$ nodes and distances $d_{i,j} = \begin{cases} 1 & \text{if } (i,j) \in E \\ k \cdot |V| & \text{otherwise} \end{cases}$
2. If T returns a total cost of $|V|$ then *accept*, otherwise *reject*."

□

Remark

This looks bad. If however all distances satisfy the *triangle inequality*, $d_{i,j} + d_{j,k} \geq d_{i,k}$, there are $\frac{3}{2}$-optimal approximation algorithms.

Probabilistic Algorithms

Definition 1.7

A *probabilistic TM* (*PTM*) M is a type of NTM in which each non-deterministic step is called a *coin-flip step* and has two legal next moves.
The *probability* of branch b of M's computation on input w is

$$\Pr[b] = \frac{1}{2^k}$$

where k is the number of coin-flip steps that occur on b.
The probability that M accepts/rejects w is

$$\Pr[M \text{ accepts } w] = \sum_{b \text{ accepts } w} \Pr[b]$$

$$\Pr[M \text{ rejects } w] = 1 - \Pr[M \text{ accepts } w]$$

Definition 1.8

For $\epsilon \in [0, \frac{1}{2})$ PTM M *recognises* A *with error probability* ϵ if

1. $w \in A$ implies $\Pr[M \text{ accepts } w] \geq 1 - \epsilon$, and
2. $w \notin A$ implies $\Pr[M \text{ rejects } w] \geq 1 - \epsilon$.

BPP is the class of languages that are recognised by polynomial time PTMs with an error probability of $\frac{1}{3}$.

Amplification Lemma

Problem.

The definition of **BPP** is robust w.r.t. the choice of error probability; $\frac{1}{3}$ is just one convenient possiblity in $[0, \frac{1}{2})$.

Lemma 1.9

Let $\epsilon \in [0, \frac{1}{2})$ and $k > 0$.
If polynomial time PTM M recognises A with error probability ϵ then there is another polynomial time PTM M' that also recognises A with error probability $\frac{1}{2^{n^k}}$.

Proof.

Run M more than once on the input word and take a majority vote among the outcomes. □

Primes

Since 2002, we know that

$$\mathit{PRIMES} = \{\ n \mid\ n \text{ is a prime number in binary } \}$$

is in **P** (e.g. $\mathcal{O}(\log^{10.5} n)$), contrary to what many had believed till then.

We'll study a (relatively simpler) proof of $\mathit{PRIMES} \in \mathbf{BPP}$.

This requires a tiny bit of good old-fashioned number theory.

Let $\mathbb{Z}_p = \{0, \ldots, p-1\}$ and $\mathbb{Z}_p^+ = \mathbb{Z}_p \setminus \{0\}$.

Theorem 2.1 (Fermat's little theorem, 1640)

If p is a prime and $a \in \mathbb{Z}_p^+$ then $a^{p-1} \equiv 1 \bmod p$.

Proof from wikipedia.org.

Consider all the possible strings of p symbols, using an alphabet Σ with a different symbols. The total number of such strings is $|\Sigma|^p = a^p$.
The *bracelet* of such a word $w_1 \dots w_p$ is the set of words
$\left\{ w_{(1+k)\%p} \dots w_{(p+k)\%p} \mid k \in \mathbb{Z}_p \right\}$.

The bracelets form a partition of Σ^p. The bracelet of a word b^p for $b \in \Sigma$ is the singleton set $\{b^p\}$. All other bracelets have size p.

It follows that $a^p - a = a(a^{p-1} - 1)$ is divisible by p. Since a and p are co-prime, $a^{p-1} - 1$ must also be divisible by p. The claim follows. $\square$

Definition 2.2

p passes the *Fermat test at* a if $a^{p-1} \equiv 1 \bmod p$.

A *pseudoprime* is a number p that passes Fermat tests for all smaller co-primes a.

If p isn't pseudoprime it fails at least half of its Fermat tests.

A probabilistic algorithm for pseudoprimality would, on input p,

1. Select $a_1, \ldots, a_k \in \mathbb{Z}_p^+$ randomly.
2. Compute $a_i^{p-1} \,\%\, p$ for each i.
3. If all computed values are 1, accept, otherwise, reject.

Parameter k affects the error probability: it is at most $\frac{1}{2^k}$.

Pseudo-primes that aren't primes are called Carmichael numbers and quite rare, e.g. 561, 1105, 1729, 2465,...

What can we do about them?

Say that *q is a square root of* 1 *modulo p* if $q^2 \equiv 1 \bmod p$.

If p is prime then only ± 1 are square roots of 1 modulo p.

For many composite numbers, including all the Carmichael numbers, 1 has 4 or more such square roots.

Examples 2.3

$\pm 1, \pm 8$ are the square roots of 1 modulo 21.

$\pm 67, \pm 188, \pm 254$ are the square roots of 1 modulo 561.

Could we test for these square roots?

After passing the Fermat test $a^{p-1} \equiv 1 \bmod p$ we deduce that the number

$$a^{(p-1)/2} \,\%\, p$$

is a square root of 1 modulo p. We continue to half the exponent as long as it is even, checking whether the first different value is -1. If it is a different number, then 1 has a square root modulo p that is different from ± 1, hence p is no prime.

BPP Algorithm for *PRIMES*

PRIME = "On input p:

1. If p is even, *accept* if $p = 2$; otherwise, *reject*.
2. Select $a_1, \ldots, a_k \in \mathbb{Z}_p^+$ randomly.
3. For each $1 \leq i \leq k$:
   1. Compute $a_i^{p-1} \,\%\, p$ and *reject* if different from 1.
   2. Let $s, t, h \in \mathbb{N}$ such that $s \cdot t = p - 1$ where s is odd and $t = 2^h$.
   3. Compute the $b_{i,j} = a_i^{s \cdot 2^j} \,\%\, p$ for $0 \leq j \leq h$.
   4. *Reject* if for the greatest j with $b_{i,j} \neq 1$ also $b_{i,j} \neq -1$.
4. *Accept*."

If p is a prime number then $\Pr[\text{PRIME accepts } p] = 1$.

If p is an even composite number then $\Pr[\text{PRIME accepts } p] = 0$.

If p is an odd composite number then $\Pr[\text{PRIME accepts } p] \leq \frac{1}{2^k}$.

For details see [Sipser2006]. We also use that modular exponentiation is in **P** for

Theorem 2.4

PRIMES $\in$ **BPP**.

RP

PRIME has an interesting property: its error is *one-sided*. Any rejected number must be composite (i.e. the error probability when rejecting is 0) whereas there is a small probability that an accepted number is not prime.

Definition 2.5

RP is the class of languages that are recognisable by probabilistic polynomial time TMs where inputs in the language are accepted with a probability of at least $\frac{1}{2}$ and inputs not in the language are rejected with a probability of 1.

Corollary 2.6

$COMPOSITES = \overline{PRIMES} \in \mathbf{RP}$.

Monte-Carlo

Definition 3.1

Call a probabilistic TM *Monte-Carlo* if it accepts any given word w with probability of either 0 (i.e. rejection is guaranteed) or $\geq \frac{1}{2}$.

Corollary 3.2

RP *is the set of languages of* **P***-time Monte-Carlo TMs.*

It is an open problem whether $\mathbf{RP} \stackrel{?}{=} \mathbf{coRP}$.

Corollary 3.3

1. $\mathbf{RP} \subseteq \mathbf{BPP}$
2. $\mathbf{RP} \subseteq \mathbf{NP}$ *(and hence* $\mathbf{coRP} \subseteq \mathbf{coNP}$*)*

Las Vegas

So far we've always categorised time complexities based on the worst-case running time. Differences may arise if we move to *expected* running time.

Definition 3.4

Call a probabilistic TM *Las Vegas* if it accepts and rejects with certainty but its running time may vary with the probabilitic choices made.

Obviously, every (deterministic) TM is a Las Vegas TM.

ZPP

Definition 3.5

Let **ZPP** (for *zero-error*, *probabilistic*, *polynomial*) be the class of languages recognised by (expected) **P**-time Las Vegas TMs.

Example 3.6

That $PRIMES \in \mathbf{ZPP}$ has been known since 1987, predating the seminal proof of $PRIMES \in \mathbf{P}$.

Corollary 3.7

1. $\mathbf{P} \subseteq \mathbf{ZPP}$
2. $A \in \mathbf{ZPP} \Rightarrow \overline{A} \in \mathbf{ZPP}$ *(so we needn't define* **coZPP***)*

Proof.

1. Let $A \in \mathbf{P}$. By definition of $\mathbf{P}$ there exists $k \in \mathbb{N}$ and a TM M such that $L(M) = A$ and, for all words w, the running time of M on w never exceeds $|w|^k$. There is only one computation path per input word to consider, so the weighted average over all computation paths is equal to the running time on the unique computation path. Thus we may use M as an expected $\mathbf{P}$-time Las Vegas TM recognising A.
2. Las Vegas machines never lie (but they may take longer to answer than you're prepared to wait). Therefore we can flip q_{accept} and q_{reject} to recognise complements without affecting expected $\mathbf{P}$-time. □

ZPP vs. **RP**

Theorem 3.8

$\mathbf{ZPP} = \mathbf{RP} \cap \mathbf{coRP}$

Compare this to the open problem of $\mathbf{P} \stackrel{?}{=} \mathbf{NP} \cap \mathbf{coNP}$.

Proof of $ZPP \supseteq \mathbf{RP} \cap \mathbf{coRP}$: Let $A \in \mathbf{RP} \cap \mathbf{coRP}$. Let M and M' be Monte Carlo TMs with $L(M) = A = \overline{L(M')}$.

Las Vegas TM $N =$ "On input w:

1. Run M on w. If M accepts, *accept*.
2. Run M' on w. If M' accepts, *reject*.
3. Repeat."

Clearly N never gives a wrong answer. But what is its expected running time? If both M and M' are running in at most n^k time then each iteration takes about $2n^k$ steps. The probabilty of halting in an iteration is $> \frac{1}{2}$. The expected running time of N is thus clearly polynomial:

$$\sum_{i>0} \Pr[N \text{ halts after iteration } i] \cdot N\text{'s running time for } i \text{ iterations}$$

$$\leq \quad \sum_{i>0} \left(\frac{1}{2^i} \cdot i2n^k \right) \quad = \quad 2n^k \sum_{i>0} \frac{i}{2^i} \quad \leq \quad 4n^k$$

Proof of $ZPP \subseteq \mathbf{RP} \cap \mathbf{coRP}$ uses Markov's inequality:

Theorem 3.9

Any non-negative random variable X satisfies $\Pr(X \geq k\mathrm{E}[X]) \leq \frac{1}{k}$.

Let N be an expected n^k-time Las Vegas TM with $L(A)$.

Let $A \in \mathbf{ZPP}$. Let Monte Carlo TM $M =$ "On input w:

1. Run N on w for at least $2|w|^k$ steps. If it halts, give its answer; otherwise, *reject*."

If $w \notin A$ then N will reject with certainty.

By Markov's inequality the chance that it will yield an answer before we stop it is $\geq \frac{1}{2}$.

This means the chance we'll give the wrong answer on $w \in A$ by timing out, is $\leq \frac{1}{2}$.
Hence $A \in \mathbf{RP}$.

To show $A \in \mathbf{coRP}$ use the same M but *accept* when timing out.

Outlook

Quantum Complexity.

- How does it work? How does it change? *Lots* of open questions . . .

Average Time Complexity. (Levin, 1986)

- what's a good definition? What classes do we have?

Fixed Parameter Complexity

- often, just some aspects of a problem contribute to complexity
- separate out easy / hard parameters

Complexity of Logical Formalisms

- mainly automated reasoning