Function Problems

- Decision problems are yes/no problems (SAT, TSP (D), etc.).
- **Function problems** require a solution (a satisfying truth assignment, a best TSP tour, etc.).
- Optimization problems are clearly function problems.
- What is the relation between function and decision problems?
- Which one is harder?
Function Problems Cannot Be Easier than Decision Problems

- If we know how to generate a solution, we can solve the corresponding decision problem.
  - If you can find a satisfying truth assignment efficiently, then SAT is in P.
  - If you can find the best TSP tour efficiently, then TSP (D) is in P.
- But we shall see that decision problems can be as hard as the corresponding function problems. immediately.
FSAT

- FSAT is this function problem:
  - Let $\phi(x_1, x_2, \ldots, x_n)$ be a boolean expression.
  - If $\phi$ is satisfiable, then return a satisfying truth assignment.
  - Otherwise, return “no.”

- We next show that if \textsc{sat} $\in \mathcal{P}$, then \textsc{fsat} has a polynomial-time algorithm.

- \textsc{SAT} is a subroutine (black box) that returns “yes” or “no” on the satisfiability of the input.
An Algorithm for FSAT Using SAT

1: \( t := \epsilon; \) \{Truth assignment.\}
2: if \( \phi \in \text{SAT} \) then
3: \hspace{1em} for \( i = 1, 2, \ldots, n \) do
4: \hspace{2em} if \( \phi[x_i = \text{true}] \in \text{SAT} \) then
5: \hspace{3em} \( t := t \cup \{ x_i = \text{true} \}; \)
6: \hspace{3em} \( \phi := \phi[x_i = \text{true}]; \)
7: \hspace{2em} else
8: \hspace{3em} \( t := t \cup \{ x_i = \text{false} \}; \)
9: \hspace{3em} \( \phi := \phi[x_i = \text{false}]; \)
10: \hspace{1em} end if
11: end for
12: return \( t; \)
13: else
14: return “no”;
15: end if
Analysis

• If SAT can be solved in polynomial time, so can FSAT.
  – There are $\leq n + 1$ calls to the algorithm for SAT.$^a$
  – Boolean expressions shorter than $\phi$ are used in each call to the algorithm for SAT.

• Hence SAT and FSAT are equally hard (or easy).

$^a$Contributed by Ms. Eva Ou (R93922132) on November 24, 2004.
Analysis (concluded)

• Note that this reduction from FSAT to SAT is not a Karp reduction.\(^a\)
  
  – Will the set of NP-complete problems differ under different reductions?\(^b\)

• Instead, it calls SAT multiple times as a subroutine, and its answers guide the search on the computation tree.

\(^a\)Recall p. 275 and p. 280.

\(^b\)Contributed by Mr. Yu-Ming Lu (R06723032, D08922008) and Mr. Han-Ting Chen (R10922073) on December 9, 2021.
TSP and TSP (D) Revisited

- We are given \( n \) cities 1, 2, \ldots, \( n \) and integer distances \( d_{ij} = d_{ji} \) between any two cities \( i \) and \( j \).
- TSP (D) asks if there is a tour with a total distance at most \( B \).
- TSP asks for a tour with the shortest total distance.
  - The shortest total distance is at most \( \sum_{i,j} d_{ij} \).
  * Recall that the input string contains \( d_{11}, \ldots, d_{nn} \).
- Thus the shortest total distance is less than \( 2|x| \) in magnitude, where \( x \) is the input (why?).
- We next show that if TSP (D) \( \in \mathbb{P} \), then TSP has a polynomial-time algorithm.
An Algorithm for TSP Using TSP (D)

1: Perform a binary search over interval \([0, 2|x|]\) by calling TSP (D) to obtain the shortest distance, \(C\);

2: \textbf{for} \(i, j = 1, 2, \ldots, n\) \textbf{do}

3: \hspace{1em} Call TSP (D) with \(B = C\) and \(d_{ij} = C + 1\);

4: \hspace{1em} \textbf{if} “no” \textbf{then}

5: \hspace{2em} Restore \(d_{ij}\) to its old value; \{Edge \([i, j]\) is critical.\}

6: \hspace{1em} \textbf{end if}

7: \textbf{end for}

8: \textbf{return} the tour with edges whose \(d_{ij} \leq C\);
Analysis

• An edge which is not on any remaining optimal tours will be eliminated, with its $d_{ij}$ set to $C + 1$.

• So the algorithm ends with $n$ edges which are not eliminated (why?).

• This is true even if there are multiple optimal tours!\(^a\)

\(^a\)Thanks to a lively class discussion on November 12, 2013 and December 9, 2021.
Analysis (concluded)

• There are $O(|x| + n^2)$ calls to the algorithm for TSP (D).

• Each call has an input length of $O(|x|)$.

• So if TSP (D) can be solved in polynomial time, so can TSP.

• Hence TSP (D) and TSP are equally hard (or easy).\(^a\)

---

\(^a\)How about counting the number of optimal TSP tours? This is related to \#P-completeness (p. 874). Contributed by Mr. Vincent Hwang (R10922138) on December 9, 2021.
Randomized Computation
I know that half my advertising works,
I just don’t know which half.
— John Wanamaker

I know that half my advertising is a waste of money,
I just don’t know which half!
— McGraw-Hill ad.
Randomized Algorithms\(^{a}\)

- Randomized algorithms flip unbiased coins.

- There are important problems for which there are no known efficient \textit{deterministic} algorithms but for which very efficient randomized algorithms exist.
  - Extraction of square roots, for instance.

- There are problems where randomization is \textit{necessary}.
  - Secure protocols.

- Randomized version can be more efficient.
  - Parallel algorithms for maximal independent set.\(^{b}\)

\(^{a}\)Rabin (1976); Solovay & Strassen (1977).

\(^{b}\)“Maximal” (a local maximum) not “maximum” (a global maximum).
Randomized Algorithms (concluded)

- Are randomized algorithms algorithms?\textsuperscript{a}

- Coin flips are occasionally used in politics.\textsuperscript{b}

\textsuperscript{a}Pascal, “Truth is so delicate that one has only to depart the least bit from it to fall into error.”

\textsuperscript{b}In the 2016 Iowa Democratic caucuses, e.g. (see http://edition.cnn.com/2016/02/02/politics/hillary-clinton-coin-flip-iowa-bernie-sanders/index.html).
“Four Most Important Randomized Algorithms”\textsuperscript{a}

1. Primality testing.\textsuperscript{b}

2. Graph connectivity using random walks.\textsuperscript{c}

3. Polynomial identity testing.\textsuperscript{d}

4. Algorithms for approximate counting.\textsuperscript{e}

\textsuperscript{a}Trevisan (2006).

\textsuperscript{b}Rabin (1976); Solovay & Strassen (1977).

\textsuperscript{c}Aleliunas, Karp, Lipton, Lovász, & Rackoff (1979).

\textsuperscript{d}Schwartz (1980); Zippel (1979).

\textsuperscript{e}Sinclair & Jerrum (1989).
Bipartite Perfect Matching

• We are given a bipartite graph \( G = (U, V, E) \).
  - \( U = \{ u_1, u_2, \ldots, u_n \} \).
  - \( V = \{ v_1, v_2, \ldots, v_n \} \).
  - \( E \subseteq U \times V \).

• We are asked if there is a perfect matching.
  - A permutation \( \pi \) of \( \{ 1, 2, \ldots, n \} \) such that
    \[
    (u_i, v_{\pi(i)}) \in E
    \]
    for all \( i \in \{ 1, 2, \ldots, n \} \).

• A perfect matching contains \( n \) edges.
A Perfect Matching in a Bipartite Graph
Symbolic Determinants

- We are given a bipartite graph $G$.

- Construct the $n \times n$ matrix $A^G$ whose $(i, j)$th entry $A^G_{ij}$ is a symbolic variable $x_{ij}$ if $(u_i, v_j) \in E$ and 0 otherwise:

$$A^G_{ij} = \begin{cases} 
 x_{ij}, & \text{if } (u_i, v_j) \in E, \\
 0, & \text{otherwise}.
\end{cases}$$
Symbolic Determinants (continued)

- The matrix for the bipartite graph $G$ on p. 533 is\(^a\)

$$A^G = \begin{bmatrix}
0 & 0 & x_{13} & x_{14} & 0 \\
0 & x_{22} & 0 & 0 & 0 \\
x_{31} & 0 & 0 & 0 & x_{35} \\
x_{41} & 0 & x_{43} & x_{44} & 0 \\
x_{51} & 0 & 0 & 0 & x_{55}
\end{bmatrix}.$$

\(^a\)The idea is similar to the Tanner (1981) graph in coding theory.
Symbolic Determinants (concluded)

• The determinant of $A^G$ is

$$\det(A^G) = \sum_{\pi} \text{sgn}(\pi) \prod_{i=1}^{n} A_{i,\pi(i)}^G.$$  \hspace{1cm} (9)

- $\pi$ ranges over all permutations of $n$ elements.
- $\text{sgn}(\pi)$ is 1 if $\pi$ is the product of an even number of transpositions and $-1$ otherwise.\(^a\)

• $\det(A^G)$ contains $n!$ terms, many of which may be 0s.

\(^a\)Equivalently, $\text{sgn}(\pi) = 1$ if the number of $(i, j)$s such that $i < j$ and $\pi(i) > \pi(j)$ is even. Contributed by Mr. Hwan-Jeu Yu (D95922028) on May 1, 2008.
Determinant and Bipartite Perfect Matching

• In $\sum_{\pi} \text{sgn}(\pi) \prod_{i=1}^{n} A_{i,\pi(i)}^{G}$, note the following:
  - Each summand corresponds to a possible perfect matching $\pi$.
  - Nonzero summands $\prod_{i=1}^{n} A_{i,\pi(i)}^{G}$ are distinct monomials and will not cancel.

• $\det(A^{G})$ is essentially an exhaustive enumeration.

**Proposition 65 (Edmonds, 1967)** $G$ has a perfect matching if and only if $\det(A^{G})$ is not identically zero.
Perfect Matching and Determinant (p. 533)
Perfect Matching and Determinant (concluded)

- The matrix is (p. 535)

\[
A^G = \begin{bmatrix}
0 & 0 & x_{13} & x_{14} & 0 \\
0 & x_{22} & 0 & 0 & 0 \\
x_{31} & 0 & 0 & 0 & x_{35} \\
x_{41} & 0 & x_{43} & x_{44} & 0 \\
x_{51} & 0 & 0 & 0 & x_{55}
\end{bmatrix}
\]

- \(\det(A^G) = -x_{14}x_{22}x_{35}x_{43}x_{51} + x_{13}x_{22}x_{35}x_{44}x_{51} + x_{14}x_{22}x_{31}x_{43}x_{55} - x_{13}x_{22}x_{31}x_{44}x_{55}\). 

- Each nonzero term denotes a perfect matching, and vice versa.
How To Test If a Polynomial Is Identically Zero?

- \( \det(A^G) \) is a polynomial in \( n^2 \) variables.
- It has, potentially, exponentially many terms.
- Expanding the determinant polynomial is thus infeasible.
- If \( \det(A^G) \equiv 0 \), then it remains zero if we substitute arbitrary integers for the variables \( x_{11}, \ldots, x_{nn} \).
- When \( \det(A^G) \not\equiv 0 \), what is the likelihood of obtaining a zero?
Number of Roots of a Polynomial

Lemma 66 (Schwartz, 1980) Let \( p(x_1, x_2, \ldots, x_m) \neq 0 \) be a polynomial in \( m \) variables each of degree at most \( d \). Let \( M \in \mathbb{Z}^+ \). Then the number of \( m \)-tuples

\[
(x_1, x_2, \ldots, x_m) \in \{0, 1, \ldots, M - 1\}^m
\]

such that \( p(x_1, x_2, \ldots, x_m) = 0 \) is

\[
\leq mdM^{m-1}.
\]

• By induction on \( m \) (consult the textbook).
Density Attack

• The density of roots in the domain is at most

\[
\frac{mdM^{m-1}}{M^m} = \frac{md}{M}.
\]  

(10)

• So suppose \( p(x_1, x_2, \ldots, x_m) \neq 0 \).

• Then a random

\[(x_1, x_2, \ldots, x_m) \in \{0, 1, \ldots, M-1\}^m\]

has a probability of \( \leq \frac{md}{M} \) of being a root of \( p \).

• Note that \( M \) is under our control!

  – One can raise \( M \) to lower the error probability, e.g.
Density Attack (concluded)

Here is a sampling algorithm to test if \( p(x_1, x_2, \ldots, x_m) \neq 0 \).

1: Choose \( i_1, \ldots, i_m \) from \( \{0, 1, \ldots, M - 1\} \) randomly;
2: \textbf{if} \( p(i_1, i_2, \ldots, i_m) \neq 0 \) \textbf{then}
3: \hspace{1em} \textbf{return} “\( p \) is not identically zero”;
4: \textbf{else}
5: \hspace{1em} \textbf{return} “\( p \) is (probably) identically zero”;
6: \textbf{end if}
Analysis

- If \( p(x_1, x_2, \ldots, x_m) \equiv 0 \), the algorithm will always be correct as \( p(i_1, i_2, \ldots, i_m) = 0 \).

- Suppose \( p(x_1, x_2, \ldots, x_m) \neq 0 \).
  - The algorithm will answer incorrectly with probability at most \( md/M \) by Eq. (10) on p. 542.

- We next return to the original problem of bipartite perfect matching.
A Randomized Bipartite Perfect Matching Algorithm\textsuperscript{a}

1: Choose $n^2$ integers $i_{11}, \ldots, i_{nn}$ from $\{0, 1, \ldots, 2n^2 - 1\}$ randomly; \{So $M = 2n^2$.\}

2: Calculate $\det(A^G(i_{11}, \ldots, i_{nn}))$ by Gaussian elimination;

3: \textbf{if} $\det(A^G(i_{11}, \ldots, i_{nn})) \neq 0$ \textbf{then}

4: \hspace{1em} \textbf{return} “$G$ has a perfect matching”;

5: \hspace{1em} \textbf{else}

6: \hspace{2em} \textbf{return} “$G$ has (probably) no perfect matchings”;

7: \textbf{end if}

\textsuperscript{a}Lovász (1979). According to Paul Erdős, Lovász wrote his first significant paper “at the ripe old age of 17.”
Analysis

• If $G$ has no perfect matchings, the algorithm will always be correct as $\det(A^G(i_{11}, \ldots, i_{nn})) = 0$.

• Suppose $G$ has a perfect matching.
  – The algorithm will answer incorrectly with probability at most $md/M = 0.5$ with $m = n^2$, $d = 1$ and $M = 2n^2$ in Eq. (10) on p. 542.

• Run the algorithm *independently* $k$ times.

• Output “$G$ has no perfect matchings” if and only if all say “(probably) no perfect matchings.”

• The error probability is now reduced to at most $2^{-k}$. 
Lószló Lovász (1948–)
Remarks$^a$

- Note that we calculated
  \[
  \text{prob[algorithm answers “no” | } G \text{ has no perfect matchings]},
  \]
  \[
  \text{prob[algorithm answers “yes” | } G \text{ has a perfect matching]}. 
  \]
  - And they are 1 and $\geq 1/2$, respectively.

- We did not calculate$^b$

  \[
  \text{prob[ } G \text{ has no perfect matchings | algorithm answers “no”]},
  \]
  \[
  \text{prob[ } G \text{ has a perfect matching | algorithm answers “yes”]}. 
  \]

---

$^a$Thanks to a lively class discussion on May 1, 2008.

$^b$Numerical Recipes in C (1988), “statistics is not a branch of mathematics!” Similar issues arise in MAP (maximum a posteriori) estimates and ML (maximum likelihood) estimates.
But How Large Can \( \det(A^G(i_{11}, \ldots, i_{nn})) \) Be?

- It is at most\(^a\)

\[
n! (2n^2)^n.
\]

- Stirling’s formula says \( n! \sim \sqrt{2\pi n} (n/e)^n \).

- Hence

\[
\log_2 \det(A^G(i_{11}, \ldots, i_{nn})) = O(n \log_2 n)
\]

bits are sufficient for representing the determinant.

- We skip the details about how to make sure that all intermediate results are of polynomial size.

\(^a\)In fact, it can be lowered to \( 2^{O(\log^2 n)} \) (Csanky, 1975)!
An Intriguing Question$^a$

- Is there an $(i_{11}, \ldots, i_{nn})$ that will always give correct answers for the algorithm on p. 545?

- A theorem on p. 642 shows that such an $(i_{11}, \ldots, i_{nn})$ exists!
  - Whether it can be found efficiently is another matter.

- Once $(i_{11}, \ldots, i_{nn})$ is available, the algorithm can be made deterministic.
  - Is it an algorithm for bipartite perfect matching?$^b$

$^a$Thanks to a lively class discussion on November 24, 2004.

$^b$We have one algorithm for each $n$ — unless there is an algorithm to generate such $(i_{11}, \ldots, i_{nn})$ for all $n$. Contributed by Mr. Han-Ting Chen (R10922073) on December 9, 2021.
Randomization vs. Nondeterminism\textsuperscript{a}

- What are the differences between randomized algorithms and nondeterministic algorithms?
- Think of a randomized algorithm as a nondeterministic one but with a probability associated with every guess/branch.
- So each computation path of a randomized algorithm has a probability associated with it.

\textsuperscript{a}Contributed by Mr. Olivier Valery (D01922033) and Mr. Hasan Alhasan (D01922034) on November 27, 2012.
Monte Carlo Algorithms\textsuperscript{a}

- The randomized bipartite perfect matching algorithm is called a \textbf{Monte Carlo algorithm} in the sense that
  - If the algorithm finds that a matching exists, it is always correct (no \textit{false positives}; no \textit{type I errors}).
  - If the algorithm answers in the negative, then it may make an error (\textit{false negatives}; \textit{type II errors}).

\* And the error probability must be small.

\textsuperscript{a}Metropolis & Ulam (1949).
Monte Carlo Algorithms (continued)

- The algorithm makes a false negative with probability \( \leq 0.5 \).\(^a\)

- Again, this probability refers to\(^b\)

\[
\text{prob[algorithm answers “no” | } G \text{ has a perfect matching]}
\]

not

\[
\text{prob[ } G \text{ has a perfect matching | algorithm answers “no” }].
\]

\(^a\)Equivalently, among the coin flip sequences, at most half of them lead to the wrong answer.

\(^b\)In general, \(\text{prob[algorithm answers “no” | input is a yes instance]}.\)
Monte Carlo Algorithms (concluded)

• This probability 0.5 is not over the space of all graphs or determinants, but over the algorithm’s own coin flips.
  – It holds for any bipartite graph.

• In contrast, to calculate

\[ \text{prob}[G \text{ has a perfect matching} | \text{algorithm answers “no”}] \]

we will need the distribution of \( G \).

• But it is an empirical statement that is very hard to verify.
The Markov Inequality\textsuperscript{a}

**Lemma 67** Let $x$ be a random variable taking nonnegative integer values. Then for any $k > 0$,

$$\text{prob}[x \geq kE[x]] \leq 1/k.$$  

- Let $p_i$ denote the probability that $x = i$.

\[
E[x] = \sum_i ip_i = \sum_{i < kE[x]} ip_i + \sum_{i \geq kE[x]} ip_i \\
\geq \sum_{i \geq kE[x]} ip_i \geq kE[x] \sum_{i \geq kE[x]} p_i \\
\geq kE[x] \times \text{prob}[x \geq kE[x]].
\]

\textsuperscript{a}Andrei Andreyevich Markov (1856–1922).
Andrei Andreyevich Markov (1856–1922)
FSAT for $k$-SAT Formulas (p. 519)

- Let $\phi(x_1, x_2, \ldots, x_n)$ be a $k$-SAT formula.

- If $\phi$ is satisfiable, then return a satisfying truth assignment.

- Otherwise, return “no.”

- We next propose a randomized algorithm for this problem.
A Random Walk Algorithm for $\phi$ in CNF Form

1: Start with an arbitrary truth assignment $T$;
2: for $i = 1, 2, \ldots, r$ do
3:  if $T \models \phi$ then
4:  return “$\phi$ is satisfiable with $T$”;
5:  else
6:  Let $c$ be an unsatisfied clause in $\phi$ under $T$; \{All of its literals are false under $T$.\}
7:  Pick any $x$ of these literals at random;
8:  Modify $T$ to make $x$ true;
9:  end if
10: end for
11: return “$\phi$ is unsatisfiable”;
3SAT vs. 2SAT Again

- Note that if $\phi$ is unsatisfiable, the algorithm will answer “unsatisfiable.”

- The random walk algorithm needs expected exponential time for 3SAT.
  - In fact, it runs in expected $O((1.333 \cdots + \epsilon)^n)$ time with $r = 3n$, much better than $O(2^n)$.

- We will show immediately that it works well for 2SAT.

- The state of the art as of 2014 is expected $O(1.30704^n)$ time for 3SAT and expected $O(1.46899^n)$ time for 4SAT.

---

*a* Use this setting per run of the algorithm.

*b* Schöning (1999). Makino, Tamaki, & Yamamoto (2011) improve the bound to deterministic $O(1.3303^n)$.

*c* Hertli (2014).
Random Walk Works for 2SAT\textsuperscript{a}

**Theorem 68** Suppose the random walk algorithm with $r = 2n^2$ is applied to any satisfiable 2SAT problem with $n$ variables. Then a satisfying truth assignment will be discovered with probability at least 0.5.

- Let $\hat{T}$ be a truth assignment such that $\hat{T} \models \phi$.
- Assume our starting $T$ differs from $\hat{T}$ in $i$ values.
  - Their Hamming distance is $i$.
- Recall $T$ is arbitrary.

\textsuperscript{a}Papadimitriou (1991).
The Proof

• Let $t(i)$ denote the expected number of repetitions of the flipping step\(^a\) until a satisfying truth assignment is found.

• It can be shown that $t(i)$ is finite.

• $t(0) = 0$ because it means that $T = \hat{T}$ and hence $T \models \phi$.

• If $T \neq \hat{T}$ or any other satisfying truth assignment, then we need to flip the coin at least once.

• We flip a coin to pick among the 2 literals of a clause not satisfied by the present $T$.

• At least one of the 2 literals is true under $\hat{T}$ because $\hat{T}$ satisfies all clauses.

\(^a\)That is, Statement 7.
The Proof (continued)

• So we have at least a 50% chance of moving closer to $\hat{T}$.

• Thus

$$t(i) \leq \frac{t(i - 1) + t(i + 1)}{2} + 1$$

for $0 < i < n$.

  – Inequality is used because, for example, $T$ may differ from $\hat{T}$ in both literals.

• It must also hold that

$$t(n) \leq t(n - 1) + 1$$

because at $i = n$, we can only decrease $i$. 
The Proof (continued)

• Now, put the necessary relations together:

\[
\begin{align*}
t(0) &= 0, \\
t(i) &\leq \frac{t(i-1) + t(i+1)}{2} + 1, \quad 0 < i < n, \\
t(n) &\leq t(n-1) + 1.
\end{align*}
\]

• Technically, this is a one-dimensional random walk with an absorbing barrier at \(i = 0\) and a reflecting barrier at \(i = n\) (if we replace “\(\leq\)” with “\(=\)”).\(^a\)

\(^a\)The proof in the textbook does exactly that. But a student pointed out difficulties with this proof technique on December 8, 2004. So our proof here uses the original inequalities.
The Proof (continued)

• Add up the relations for 
  \[ 2t(1), 2t(2), 2t(3), \ldots, 2t(n - 1), t(n) \] to obtain\(^a\)

  \[
  2t(1) + 2t(2) + \cdots + 2t(n - 1) + t(n) \\
  \leq t(0) + t(1) + 2t(2) + \cdots + 2t(n - 2) + 2t(n - 1) + t(n) \\
  + 2(n - 1) + 1.
  \]

• Simplify it to yield

  \[ t(1) \leq 2n - 1. \] (14)

\(^a\)Adding up the relations for \(t(1), t(2), t(3), \ldots, t(n - 1)\) will also work, thanks to Mr. Yen-Wu Ti (D91922010).
The Proof (continued)

• Add up the relations for $2t(2), 2t(3), \ldots, 2t(n - 1), t(n)$ to obtain

$$2t(2) + \cdots + 2t(n - 1) + t(n) \leq t(1) + t(2) + 2t(3) + \cdots + 2t(n - 2) + 2t(n - 1) + t(n) + 2(n - 2) + 1.$$ 

• Simplify it to yield

$$t(2) \leq t(1) + 2n - 3 \leq 2n - 1 + 2n - 3 = 4n - 4$$

by Eq. (14) on p. 564.
The Proof (continued)

• Continuing the process, we shall obtain

\[ t(i) \leq 2in - i^2. \]

• The worst upper bound happens when \( i = n \), in which case

\[ t(n) \leq n^2. \]

• We conclude that

\[ t(i) \leq t(n) \leq n^2 \]

for \( 0 \leq i \leq n. \)

\(^a\)See also Feller (1968).
The Proof (concluded)

• So the expected number of steps is at most $n^2$.

• The algorithm picks $r = 2n^2$.

• Apply the Markov inequality (p. 555) with $k = 2$ to yield the desired probability of 0.5.

• The proof does not yield a polynomial bound for 3SAT.\(^a\)

\(^a\)Contributed by Mr. Cheng-Yu Lee (R95922035) on November 8, 2006.
Boosting the Performance

- We can pick $r = 2mn^2$ to have an error probability of
  \[ \leq \frac{1}{2m} \]
  by Markov’s inequality.

- Alternatively, with the same running time, we can run the “$r = 2n^2$” algorithm $m$ times.

- The error probability is now reduced to
  \[ \leq 2^{-m}. \]
Primality Tests

- PRIMES asks if a number $N$ is a prime.
- The classic algorithm tests if $k | N$ for $k = 2, 3, \ldots, \sqrt{N}$.
- But it runs in $\Omega(2^{(\log_2 N)/2})$ steps.
- COMPOSITENESS asks if a number is composite.
The Fermat Test for Primality

Fermat’s “little” theorem (p. 505) suggests the following primality test for any given number $N$:

1. Pick a number $a$ randomly from $\{1, 2, \ldots, N - 1\}$;
2. if $a^{N-1} \not\equiv 1 \mod N$ then
3. return “$N$ is composite”;
4. else
5. return “$N$ is (probably) a prime”;
6. end if
The Fermat Test for Primality (continued)

- **Carmichael numbers** are composite numbers that will pass the Fermat test for all \( a \in \{1, 2, \ldots, N - 1\} \).\(^a\)
  - The Fermat test will return “\( N \) is a prime” for all Carmichael numbers \( N \).

- If there are finitely many Carmichael numbers, store them for matches before running the Fermat test.

- Unfortunately, there are infinitely many such numbers.\(^b\)
  - The number of Carmichael numbers less than \( N \) exceeds \( N^{2/7} \) for \( N \) large enough.

\(^a\)Carmichael (1910). Lo (1994) mentions an investment strategy based on such numbers!

\(^b\)Alford, Granville, & Pomerance (1992).
The Fermat Test for Primality (concluded)

- The Fermat test will fail all of them.
- So the Fermat test is an *incorrect* algorithm for PRIMES.
- Even suppose $N$ is not a Carmichael number but remains composite.
- We need many $a \in \{1, 2, \ldots, N - 1\}$ such that $a^{N-1} \not\equiv 1 \mod N$.
- Otherwise, the correct answer will come only with a vanishing probability (say $1/N$).\(^a\)

\(^a\)Contributed by Mr. Vincent Hwang (R10922138) on December 9, 2021.