# The Chinese Remainder Theorem

- Let $n = n_1 n_2 \cdots n_k$, where $n_i$ are pairwise relatively prime.
- For any integers $a_1, a_2, \ldots, a_k$, the set of simultaneous equations

$$\begin{aligned} x &= a_1 \bmod n_1, \\ x &= a_2 \bmod n_2, \\ &\vdots \\ x &= a_k \bmod n_k, \end{aligned}$$

has a unique solution modulo $n$ for the unknown $x$.

# Fermat's "Little" Theorem[a]

**Lemma 61** *For all $0 < a < p$, $a^{p-1} = 1 \bmod p$.*

- Recall $\Phi(p) = \{ 1, 2, \ldots, p-1 \}$.
- Consider $a\Phi(p) = \{ am \bmod p : m \in \Phi(p) \}$.
- $a\Phi(p) = \Phi(p)$.
  - $a\Phi(p) \subseteq \Phi(p)$ as a remainder must be between 1 and $p-1$.
  - Suppose $am \equiv am' \bmod p$ for $m > m'$, where $m, m' \in \Phi(p)$.
  - That means $a(m - m') = 0 \bmod p$, and $p$ divides $a$ or $m - m'$, which is impossible.

---

[a] Pierre de Fermat (1601–1665).

# The Proof (concluded)

- Multiply all the numbers in $\Phi(p)$ to yield $(p-1)!$.
- Multiply all the numbers in $a\Phi(p)$ to yield $a^{p-1}(p-1)!$.
- As $a\Phi(p) = \Phi(p)$, we have

$$a^{p-1}(p-1)! \equiv (p-1)! \bmod p.$$

- Finally, $a^{p-1} = 1 \bmod p$ because $p \not| (p-1)!$.

# The Fermat-Euler Theorem[a]

**Corollary 62** *For all* $a \in \Phi(n)$, $a^{\phi(n)} = 1 \bmod n$.

- The proof is similar to that of Lemma 61 (p. 487).
- Consider $a\Phi(n) = \{\, am \bmod n : m \in \Phi(n) \,\}$.
- $a\Phi(n) = \Phi(n)$.
  - $a\Phi(n) \subseteq \Phi(n)$ as a remainder must be between 0 and $n - 1$ and relatively prime to $n$.
  - Suppose $am \equiv am' \bmod n$ for $m' < m < n$, where $m, m' \in \Phi(n)$.
  - That means $a(m - m') = 0 \bmod n$, and $n$ divides $a$ or $m - m'$, which is impossible.

---

[a] Proof by Mr. Wei-Cheng Cheng (`R93922108`, `D95922011`) on November 24, 2004.

# The Proof (concluded)[a]

- Multiply all the numbers in $\Phi(n)$ to yield $\prod_{m\in\Phi(n)} m$.
- Multiply all the numbers in $a\Phi(n)$ to yield $a^{\phi(n)}\prod_{m\in\Phi(n)} m$.
- As $a\Phi(n) = \Phi(n)$,

$$\prod_{m\in\Phi(n)} m \equiv a^{\phi(n)} \left( \prod_{m\in\Phi(n)} m \right) \bmod n.$$

- Finally, $a^{\phi(n)} = 1 \bmod n$ because $n \not| \prod_{m\in\Phi(n)} m$.

---

[a] Some typographical errors corrected by Mr. Jung-Ying Chen (`D95723006`) on November 18, 2008.

# An Example

- As $12 = 2^2 \times 3$,

$$\phi(12) = 12 \times \left(1 - \frac{1}{2}\right)\left(1 - \frac{1}{3}\right) = 4.$$

- In fact, $\Phi(12) = \{\, 1, 5, 7, 11 \,\}$.

- For example,

$$5^4 = 625 = 1 \bmod 12.$$

# Exponents

- The **exponent** of $m \in \Phi(p)$ is the least $k \in \mathbb{Z}^+$ such that
$$m^k = 1 \bmod p.$$
- Every residue $s \in \Phi(p)$ has an exponent.
  - $1, s, s^2, s^3, \ldots$ eventually repeats itself modulo $p$, say $s^i \equiv s^j \bmod p$, $i < j$, which means $s^{j-i} = 1 \bmod p$.
- If the exponent of $m$ is $k$ and $m^\ell = 1 \bmod p$, then $k \mid \ell$.
  - Otherwise, $\ell = qk + a$ for $0 < a < k$, and $m^\ell = m^{qk+a} \equiv m^a \equiv 1 \bmod p$, a contradiction.

**Lemma 63** *Any nonzero polynomial of degree $k$ has at most $k$ distinct roots modulo $p$.*

# Exponents and Primitive Roots

- From Fermat's "little" theorem (p. 487), all exponents divide $p - 1$.
- A primitive root of $p$ is thus a number with exponent $p - 1$.
- Let $R(k)$ denote the total number of residues in $\Phi(p) = \{ 1, 2, \ldots, p - 1 \}$ that have exponent $k$.
- We already knew that $R(k) = 0$ for $k \not| (p - 1)$.
- So
$$\sum_{k \,|\, (p-1)} R(k) = p - 1$$
as every number has an exponent.

# Size of $R(k)$

- Any $a \in \Phi(p)$ of exponent $k$ satisfies

$$x^k = 1 \bmod p.$$

- By Lemma 63 (p. 492) there are at most $k$ residues of exponent $k$, i.e., $R(k) \leq k$.
- Let $s$ be a residue of exponent $k$.
- $1, s, s^2, \ldots, s^{k-1}$ are distinct modulo $p$.
  - Otherwise, $s^i \equiv s^j \bmod p$ with $i < j$.
  - Then $s^{j-i} = 1 \bmod p$ with $j - i < k$, a contradiction.
- As all these $k$ distinct numbers satisfy $x^k = 1 \bmod p$, they comprise *all* the solutions of $x^k = 1 \bmod p$.

## Size of $R(k)$ (continued)

- But do all of them have exponent $k$ (i.e., $R(k) = k$)?
- And if not (i.e., $R(k) < k$), how many of them do?
- Pick $s^\ell$, where $\ell < k$.
- Suppose $\ell \notin \Phi(k)$ with $\gcd(\ell, k) = d > 1$.
- Then
$$(s^\ell)^{k/d} = (s^k)^{\ell/d} = 1 \bmod p.$$
- Therefore, $s^\ell$ has exponent at most $k/d < k$.
- So $s^\ell$ has exponent $k$ *only if* $\ell \in \Phi(k)$.
- We conclude that
$$R(k) \leq \phi(k).$$

# Size of $R(k)$ (concluded)

- Because all $p-1$ residues have an exponent,

$$p-1=\sum_{k\,|\,(p-1)} R(k) \leq \sum_{k\,|\,(p-1)} \phi(k)=p-1$$

  by Lemma 60 (p. 481).

- Hence

$$R(k)=\begin{cases} \phi(k) & \text{when } k\,|\,(p-1) \\ 0 & \text{otherwise} \end{cases}$$

- In particular, $R(p-1)=\phi(p-1)>0$, and $p$ has at least one primitive root.

- This proves one direction of Theorem 55 (p. 466).

# A Few Calculations

- Let $p = 13$.
- From p. 489 $\phi(p-1) = 4$.
- Hence $R(12) = 4$.
- Indeed, there are 4 primitive roots of $p$.
- As
$$\Phi(p-1) = \{ 1, 5, 7, 11 \},$$
the primitive roots are
$$g^1, g^5, g^7, g^{11},$$
where $g$ is *any* primitive root.

# 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 SAT $\in$ P, then FSAT has a polynomial-time algorithm.
- SAT is a subroutine (black box) that returns "yes" or "no" on the satisfiability of the input.

# An Algorithm for FSAT Using SAT

```
$t := \epsilon$; {Truth assignment.}
if $\phi \in$ SAT then
    for $i = 1, 2, \ldots, n$ do
        if $\phi[\, x_i = \texttt{true} \,] \in$ SAT then
            $t := t \cup \{\, x_i = \texttt{true} \,\}$;
            $\phi := \phi[\, x_i = \texttt{true} \,]$;
        else
            $t := t \cup \{\, x_i = \texttt{false} \,\}$;
            $\phi := \phi[\, x_i = \texttt{false} \,]$;
        end if
    end for
    return $t$;
else
    return "no";
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).
- Note that this reduction from FSAT to SAT is not a Karp reduction.[b]
- Instead, it calls SAT multiple times as a subroutine, and its answers guide the search on the computation tree.

---

[a] Contributed by Ms. Eva Ou (`R93922132`) on November 24, 2004.

[b] Recall p. 261 and p. 265.

# 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) $\in$ P, then TSP has a polynomial-time algorithm.

# An Algorithm for TSP Using TSP (D)

```
Perform a binary search over interval $[0, 2^{|x|}]$ by calling
    TSP (D) to obtain the shortest distance, $C$;
for $i, j = 1, 2, \ldots, n$ do
    Call TSP (D) with $B = C$ and $d_{ij} = C + 1$;
    if "no" then
        Restore $d_{ij}$ to its old value; {Edge $[i, j]$ is critical.}
    end if
end for
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.

## 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).

# *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 *deterministic* algorithms but for which very efficient randomized algorithms exist.
  - Extraction of square roots, for instance.
- There are problems where randomization is *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?[a]
- Coin flips are occasionally used in politics.[b]

---

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

[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”[a]

1. Primality testing.[b]
2. Graph connectivity using random walks.[c]
3. Polynomial identity testing.[d]
4. Algorithms for approximate counting.[e]

---

[a] Trevisan (2006).

[b] Rabin (1976); Solovay & Strassen (1977).

[c] Aleliunas, Karp, Lipton, Lovász, & Rackoff (1979).

[d] Schwartz (1980); Zippel (1979).

[e] Sinclair & Jerrum (1989).

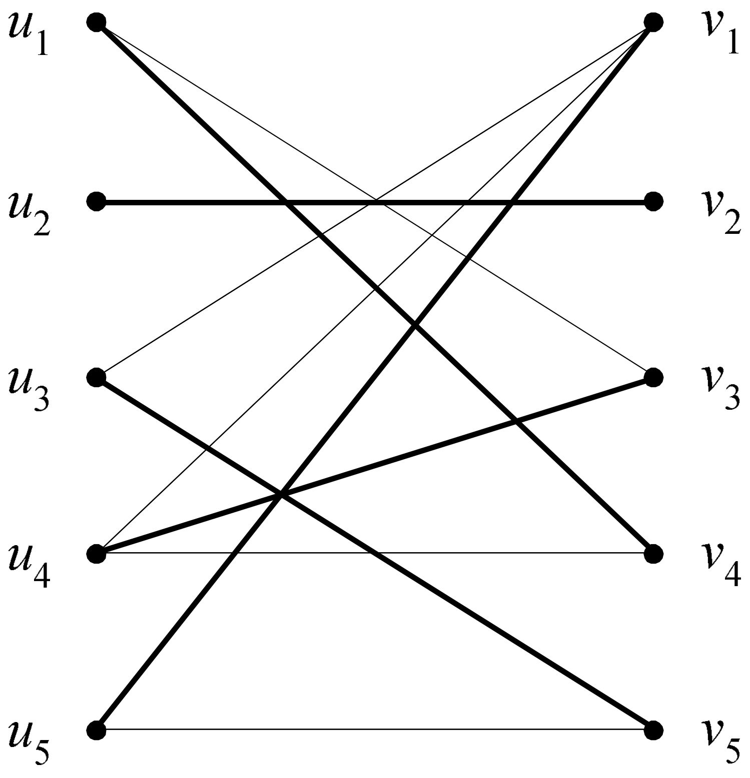
# Bipartite Perfect Matching

- 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:

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

# Symbolic Determinants (continued)

- The matrix for the bipartite graph $G$ on p. 513 is[a]

$$A^G = \begin{bmatrix} 0 & 0 & x_{13} & x_{14} & 0 \\ 0 & x_{22} & 0 & 0 & 0 \\ x_{31} & 0 & 0 & 0 & x_{35} \\ x_{41} & 0 & x_{43} & x_{44} & 0 \\ x_{51} & 0 & 0 & 0 & x_{55} \end{bmatrix}. \qquad (7)$$

---

[a] The idea is similar to the Tanner graph in coding theory by Tanner (1981).

## Symbolic Determinants (concluded)

- The **determinant** of $A^G$ is

$$\det(A^G) = \sum_{\pi} \text{sgn}(\pi) \prod_{i=1}^{n} A^G_{i,\pi(i)}. \tag{8}$$

  - $\pi$ 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.

---

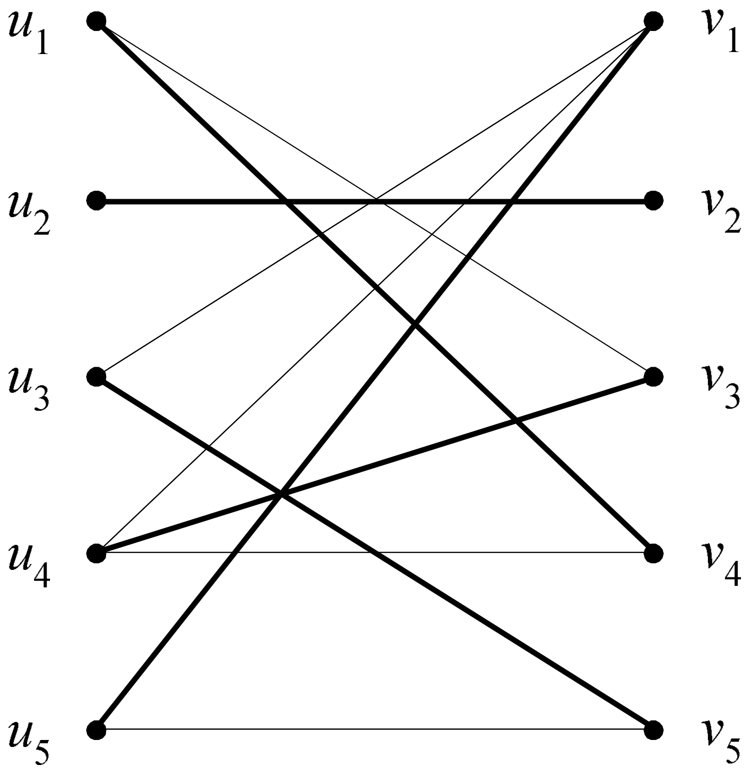
[a] Equivalently, $\text{sgn}(\pi) = 1$ if the number of $(i, j)$s such that $i < j$ and $\pi(i) > \pi(j)$ is even. Contributed by Mr. Hwan-Jeu Yu (`D95922028`) on May 1, 2008.

## Determinant and Bipartite Perfect Matching

- In $\sum_\pi \operatorname{sgn}(\pi) \prod_{i=1}^n A^G_{i,\pi(i)}$, note the following:
  - Each summand corresponds to a possible perfect matching $\pi$.
  - Nonzero summands $\prod_{i=1}^n A^G_{i,\pi(i)}$ are distinct monomials and *will not cancel*.
- $\det(A^G)$ is essentially an exhaustive enumeration.

**Proposition 64 (Edmonds, 1967)** *$G$ has a perfect matching if and only if* $\det(A^G)$ *is not identically zero.*

# Perfect Matching and Determinant (p. 513)



# Perfect Matching and Determinant (concluded)

- The matrix is (p. 515)

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

- $\det(A^G) = -x_{14}x_{22}x_{35}x_{43}x_{51} + x_{13}x_{22}x_{35}x_{44}x_{51} + x_{14}x_{22}x_{31}x_{43}x_{55} - x_{13}x_{22}x_{31}x_{44}x_{55}$.
- 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 65 (Schwartz, 1980)** *Let* $p(x_1, x_2, \ldots, x_m) \not\equiv 0$ *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

$$\frac{mdM^{m-1}}{M^m} = \frac{md}{M}. \tag{9}$$

- So suppose $p(x_1, x_2, \ldots, x_m) \not\equiv 0$.
- Then a random

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

  has a probability of $\leq md/M$ 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 $p(x_1, x_2, \ldots, x_m) \not\equiv 0$.

```
Choose $i_1, \ldots, i_m$ from $\{ 0, 1, \ldots, M-1 \}$ randomly;
if $p(i_1, i_2, \ldots, i_m) \neq 0$ then
    return "$p$ is not identically zero";
else
    return "$p$ is (probably) identically zero";
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 $p(x_1, x_2, \ldots, x_m) \not\equiv 0$.
  - The algorithm will answer incorrectly with probability at most $md/M$ by Eq. (9) on p. 522.
- We next return to the original problem of bipartite perfect matching.

# A Randomized Bipartite Perfect Matching Algorithm[a]

```
Choose $n^2$ integers $i_{11}, \ldots, i_{nn}$ from $\{ 0, 1, \ldots, 2n^2 - 1 \}$ randomly; {So $M = 2n^2$.}
Calculate $\det(A^G(i_{11}, \ldots, i_{nn}))$ by Gaussian elimination;
if $\det(A^G(i_{11}, \ldots, i_{nn})) \neq 0$ then
    return "$G$ has a perfect matching";
else
    return "$G$ has (probably) no perfect matchings";
end if
```

---

[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. (9) on p. 522.
- 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 are calculating

$$\text{prob}[\,\text{algorithm answers “no”}\,|\,G \text{ has no perfect matchings}\,],$$

$$\text{prob}[\,\text{algorithm answers “yes”}\,|\,G \text{ has a perfect matching}\,].$$

- We are *not* calculating[b]

$$\text{prob}[\,G \text{ has no perfect matchings}\,|\,\text{algorithm answers “no”}\,],$$

$$\text{prob}[\,G \text{ has a perfect matching}\,|\,\text{algorithm answers “yes”}\,].$$

---

[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
$$n!\left(2n^2\right)^n.$$
- 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.

# An Intriguing Question[a]

- Is there an $(i_{11}, \ldots, i_{nn})$ that will always give correct answers for the algorithm on p. 525?
- A theorem on p. 621 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.

---

[a] Thanks to a lively class discussion on November 24, 2004.

# Randomization vs. Nondeterminism[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.

---

[a] Contributed by Mr. Olivier Valery (`D01922033`) and Mr. Hasan Alhasan (`D01922034`) on November 27, 2012.

# Monte Carlo Algorithms[a]

- The randomized bipartite perfect matching algorithm is called a **Monte Carlo algorithm** in the sense that
  - If the algorithm finds that a matching exists, it is always correct (no **false positives**; no **type 1 errors**).
  - If the algorithm answers in the negative, then it may make an error (**false negatives**; **type 2 errors**).

---

[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]

  prob[ algorithm answers "no" | $G$ has a perfect matching ]

  not

  prob[ $G$ 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, 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[a]

**Lemma 66** *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 $x = i$.

$$\begin{aligned} E[\, x\,] &= \sum_i i p_i = \sum_{i < kE[\, x\,]} i p_i + \sum_{i \geq kE[\, x\,]} i p_i \\ &\geq \sum_{i \geq kE[\, x\,]} i p_i \geq kE[\, x\,] \sum_{i \geq kE[\, x\,]} p_i \\ &\geq kE[\, x\,] \times \text{prob}[x \geq kE[\, x\,]]. \end{aligned}$$

---

[a] Andrei Andreyevich Markov (1856–1922).

Andrei Andreyevich Markov (1856–1922)

## FSAT for $k$-SAT Formulas (p. 500)

- 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

```
Start with an *arbitrary* truth assignment $T$;
for $i = 1, 2, \ldots, r$ do
    if $T \models \phi$ then
        return "$\phi$ is satisfiable with $T$";
    else
        Let $c$ be an unsatisfied clause in $\phi$ under $T$; {All of
        its literals are false under $T$.}
        Pick any $x$ of these literals *at random*;
        Modify $T$ to make $x$ true;
    end if
end for
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 $r = 3n$,[a] much better than $O(2^n)$.[b]
- 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.[c]

---

[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[a]

**Theorem 67** *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.

---

[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:

$$\begin{aligned} t(0) &= 0, & (10)\\ t(i) &\leq \frac{t(i-1)+t(i+1)}{2}+1, \quad 0<i<n, & (11)\\ t(n) &\leq t(n-1)+1. & (12) \end{aligned}$$

- 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]

$$\begin{aligned} & 2t(1) + 2t(2) + \cdots + 2t(n-1) + t(n) \\ \leq\ & t(0) + t(1) + 2t(2) + \cdots + 2t(n-2) + 2t(n-1) + t(n) \\ & +2(n-1) + 1. \end{aligned}$$

- Simplify it to yield

$$t(1) \leq 2n - 1. \tag{13}$$

---

[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

$$\begin{aligned} & 2t(2) + \cdots + 2t(n-1) + t(n) \\ \leq\ & t(1) + t(2) + 2t(3) + \cdots + 2t(n-2) + 2t(n-1) + t(n) \\ & +2(n-2) + 1. \end{aligned}$$

- Simplify it to yield

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

by Eq. (13) on p. 544.

## 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 $0 \leq i \leq n$.

## The Proof (concluded)

- So the expected number of steps is at most $n^2$.
- The algorithm picks $r = 2n^2$.
  - This amounts to invoking the Markov inequality (p. 535) with $k = 2$, resulting in a 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

  $$\leq 2^{-m}.$$