A Few Calculations

- From p. 362, we know $\phi(p - 1) = 4$.
- Hence $R(12) = 4$.
- And there are 4 primitives roots of $p$.
- As $\Phi(p - 1) = \{1, 5, 7, 11\}$, the primitive roots are $g^1, g^5, g^7, g^{11}$ for any primitive root $g$.

The Other Direction of Theorem 47 (p. 346)

- We must show $p$ is a prime only if there is a number $r$ (called primitive root) such that
  1. $r^{p-1} \equiv 1 \pmod{p}$, and
  2. $r^{(p-1)/q} \not\equiv 1 \pmod{p}$ for all prime divisors $q$ of $p - 1$.
- Suppose $p$ is not a prime.
- We proceed to show that no primitive roots exist.
- Suppose $r^{p-1} \equiv 1 \pmod{p}$ (note $\gcd(r, p) = 1$).
- We will show that the 2nd condition must be violated.

The Proof (concluded)

- $r^{\phi(p)} \equiv 1 \pmod{p}$ by the Fermat-Euler theorem (p. 362).
- Because $p$ is not a prime, $\phi(p) < p - 1$.
- Let $k$ be the smallest integer such that $r^k \equiv 1 \pmod{p}$.
- As $k \leq \phi(p)$, $k < p - 1$.
- Let $q$ be a prime divisor of $(p - 1)/k > 1$.
- Then $k|(p - 1)/q$.
- Therefore, by virtue of the definition of $k$,
  $$r^{(p-1)/q} \equiv 1 \pmod{p}.$$  
- But this violates the 2nd condition.

Function Problems

- Decisions problem are yes/no problems ($\text{SAT}$, $\text{TSP}$ (D), etc.).
- Function problems require a solution (a satisfying truth assignment, a best $\text{TSP}$ 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 decision problems can be as hard as the corresponding function problems.

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.

An Algorithm for FSAT Using SAT

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

Analysis

- There are \( \leq n + 1 \) calls to the algorithm for SAT\(^a\).
- Shorter boolean expressions than \( \phi \) are used in each call to the algorithm for SAT.
- So if SAT can be solved in polynomial time, so can FSAT.
- Hence SAT and FSAT are equally hard (or easy).

\(^a\)Contributed by Ms. Eva Ou (R93922132) on November 24, 2004.
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 \( j \).
- The TSP asks for a tour with the shortest total distance (not just the shortest total distance, as earlier).
  - The shortest total distance must be at most \( 2|\lambda| \), where \( \lambda \) is the input.
- TSP (D) asks if there is a tour with a total distance at most \( B \).
- We next show that if TSP (D) \( \in P \), then TSP has a polynomial-time algorithm.

Analysis

- An edge that is not on any optimal tour will be eliminated, with its \( d_{ij} \) set to \( C + 1 \).
- An edge which is not on all remaining optimal tours will also be eliminated.
- So the algorithm ends with \( n \) edges which are not eliminated (why?).
- There are \( O(|\lambda| + n^2) \) calls to the algorithm for TSP (D).
- So if TSP (D) can be solved in polynomial time, so can TSP.
- Hence TSP (D) and TSP are equally hard (or easy).

An Algorithm for TSP Using TSP (D)

1. Perform a binary search over interval \( [0, 2|\lambda|] \) by calling TSP (D) to obtain the shortest distance \( C \);
2. for \( i, j = 1, 2, \ldots, n \) do
3.   Call TSP (D) with \( B = C \) and \( d_{ij} = C + 1 \);
4.   if “no” then
5.      Restore \( d_{ij} \) to old value; \{Edge \( [i, j] \) is critical.\}
6.   end if
7. end for
8. return the tour with edges whose \( d_{ij} \leq C \);

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

• 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 algorithm for maximal independent set.
• Are randomized algorithms algorithms?

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 $u_i \in U$.

A Perfect Matching
Symbolic Determinants

- Given a bipartite graph $G$, construct the $n \times n$ matrix $A^G$ whose $(i,j)$th entry $A^G_{ij}$ is a variable $x_{ij}$ if $(u_i,v_j) \in E$ and zero otherwise.
- The determinant of $A^G$ is
  \[
  \det(A^G) = \sum_\pi \text{sgn}(\pi) \prod_{i=1}^n A^G_{i,\pi(i)}. \tag{5}
  \]
  - $\pi$ 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.

Determinant and Bipartite Perfect Matching

- In $\sum_\pi \text{sgn}(\pi) \prod_{i=1}^n A^G_{i,\pi(i)}$, note the following:
  - Each summand corresponds to a possible perfect matching $\pi$.
  - As all variables appear only once, all of these summands are different monomials and will not cancel.
- It is essentially an exhaustive enumeration.

**Proposition 56 (Edmonds (1967))** $G$ has a perfect matching if and only if $\det(A^G)$ is not identically zero.

The Perfect Matching in the Determinant

- The matrix is
  \[
  A^G = \begin{bmatrix}
  0 & 0 & x_{13} & x_{14} & 0 \\
  0 & 0 & 0 & 0 & 0 \\
  x_{31} & 0 & 0 & 0 & x_{35} \\
  x_{41} & 0 & x_{43} & x_{44} & 0 \\
  x_{51} & 0 & 0 & 0 & x_{55}
  \end{bmatrix}.
  \]
- $\det(A^G) = -x_{14}x_{22}x_{35}x_{43}x_{51} + x_{13}x_{22}x_{35}x_{44}x_{51} + x_{14}x_{22}x_{31}x_{43}x_{55} - x_{13}x_{22}x_{31}x_{44}x_{55}$, each denoting a perfect matching.
How To Test If a Polynomial Is Identically Zero?

- $\det(A^G)$ is a polynomial in $n^2$ variables.
- There are exponentially many terms in $\det(A^G)$.
- Expanding the determinant polynomial is not feasible.
  - Too many terms.
- Observation: If $\det(A^G)$ is identically zero, then it remains zero if we substitute arbitrary integers for the variables $x_{11}, \ldots, x_{nn}$.
- What is the likelihood of obtaining a zero when $\det(A^G)$ is not identically zero?

Density Attack

- The density of roots in the domain is at most
  $$\frac{mdM^{m-1}}{M^m} = \frac{md}{M}.$$  
- So suppose $p(x_1, x_2, \ldots, x_m) \not\equiv 0$.
- Then a random
  $$(x_1, x_2, \ldots, x_n) \in \{0, 1, \ldots, M-1\}^n$$
  has a probability of $\leq md/M$ of being a root of $p$.

Number of Roots of a Polynomial

Lemma 57 (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 (concluded)

Here is a sampling algorithm to test if $p(x_1, x_2, \ldots, x_m) \not\equiv 0$.
1: Choose $i_1, \ldots, i_m$ from $\{0, 1, \ldots, M-1\}$ randomly;
2: if $p(i_1, i_2, \ldots, i_m) \not\equiv 0$ then
3: return “$p$ is not identically zero”;
4: else
5: return “$p$ is identically zero”;
6: end if
A Randomized Bipartite Perfect Matching Algorithm\textsuperscript{a}

We now return to the original problem of bipartite perfect matching.

1: Choose $n^2$ integers $i_{11}, \ldots, i_{nn}$ from $\{0, 1, \ldots, b - 1\}$ randomly;
1: Calculate $\det(A^G(i_{11}, \ldots, i_{nn}))$ by Gaussian elimination;
2: if $\det(A^G(i_{11}, \ldots, i_{nn})) \neq 0$ then
3: return "$G$ has a perfect matching";
4: else
5: return "$G$ has no perfect matchings";
6: end if

\textsuperscript{a}Lovász (1979).

Perfect Matching for General Graphs

- Page 382 is about bipartite perfect matching
- Now we are given a graph $G = (V, E)$.
  - $V = \{v_1, v_2, \ldots, v_{2n}\}$.
- We are asked if there is a perfect matching.
  - A permutation $\pi$ of $\{1, 2, \ldots, 2n\}$ such that
    $$(v_i, v_{\pi(i)}) \in E$$
    for all $v_i \in V$.

Analysis

- Pick $b = 2n^2$.
- If $G$ has no perfect matchings, the algorithm will always be correct.
- Suppose $G$ has a perfect matching.
  - The algorithm will answer incorrectly with probability at most $n^2d/b = 0.5$ because $d = 1$.
  - Run the algorithm independently $k$ times and output "$G$ has no perfect matchings" if they all say no.
  - The error probability is now reduced to at most $2^{-k}$.
- Is there an $(i_{11}, \ldots, i_{nn})$ that will always give correct answers for all bipartite graphs of $2n$ nodes?\textsuperscript{a}

\textsuperscript{a}Thanks to a lively class discussion on November 24, 2004.

The Tutte Matrix\textsuperscript{a}

- Given a graph $G = (V, E)$, construct the $2n \times 2n$ Tutte matrix $T^G$ such that
  $$T^G_{ij} = \begin{cases} 
  x_{ij} & \text{if } (v_i, v_j) \in E \text{ and } i < j, \\
  -x_{ij} & \text{if } (v_i, v_j) \in E \text{ and } i > j, \\
  0 & \text{othersie.}
  \end{cases}$$
- The Tutte matrix is a skew-symmetric symbolic matrix.
- Similar to Proposition 56 (p. 385):
  \textbf{Proposition 58} $G$ has a perfect matching if and only if $\det(T^G)$ is not identically zero.

\textsuperscript{a}William Thomas Tutte (1917–2002).
Monte Carlo Algorithms

- 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).
  - If the algorithm answers in the negative, then it may make an error (false negative).
- The algorithm makes a false negative with probability \( \leq 0.5 \).
- This probability is not over the space of all graphs or determinants, but over the algorithm's own coin flips.
  - It holds for any bipartite graph.

\(^*\text{Metropolis and Ulam (1949).}\)

---

The Markov Inequality

**Lemma 59** Let \( x \) be a random variable taking nonnegative integer values. Then for any \( k > 0 \),

\[
\Pr[x \geq kE[x]] \leq \frac{1}{k}.
\]

- Let \( p_i \) denote the probability that \( x = i \).

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

\(^*\text{Andrei Andreyevich Markov (1856–1922).}\)

---

An Application of Markov’s Inequality

- Algorithm \( C \) runs in expected time \( T(n) \) and always gives the right answer.
- Consider an algorithm that runs \( C \) for time \( kT(n) \) and rejects the input if \( C \) does not stop within the time bound.
- By Markov’s inequality, this new algorithm runs in time \( kT(n) \) and gives the wrong answer with probability \( \leq 1/k \).
- By running this algorithm \( m \) times, we reduce the error probability to \( \leq k^{-m} \).

---

An Application of Markov’s Inequality (concluded)

- Suppose, instead, we run the algorithm for the same running time \( mkT(n) \) once and rejects the input if it does not stop within the time bound.
- By Markov’s inequality, this new algorithm gives the wrong answer with probability \( \leq 1/(mk) \).
- This is a far cry from the previous algorithm’s error probability of \( \leq k^{-m} \).
- The loss comes from the fact that Markov’s inequality does not take advantage of any specific feature of the random variable.
FSAT for \( k \)-SAT Formulas (p. 373)

- 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 unsatisfiable 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 not refute it.
- 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 \( O(2^n) \).\(^a\)
- We will show immediately that it works well for 2SAT.
- The state of the art is expected \( O(1.324^n) \) time for 3SAT and expected \( O(1.474^n) \) time for 4SAT.\(^b\)

\(^a\)Schöning (1999).
\(^b\)Kwama and Tamaki (2004).