Graph Coloring

- k-COLORING asks if the nodes of a graph can be colored with $\leq k$ colors such that no two adjacent nodes have the same color.
- 2-COLORING is in P (why?).
- But 3-COLORING is NP-complete (see next page).
- k-COLORING is NP-complete for $k \geq 3$ (why?).

3-COLORING Is NP-Complete[a]

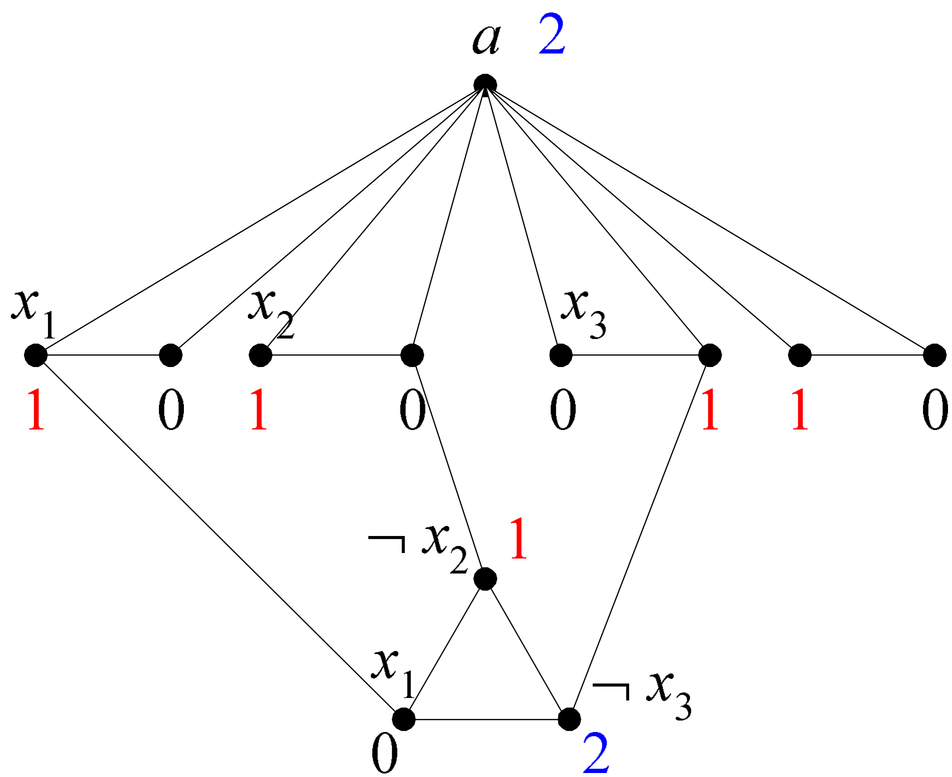
- We will reduce NAESAT to 3-COLORING.
- We are given a set of clauses $C_1, C_2, \ldots, C_m$ each with 3 literals.
- The boolean variables are $x_1, x_2, \ldots, x_n$.
- We shall construct a graph G such that it can be colored with colors $\{0, 1, 2\}$ if and only if all the clauses can be NAE-satisfied.

[a]Karp (1972).

The Proof (continued)

- Every variable x_i is involved in a triangle $[\,a, x_i, \neg x_i\,]$ with a common node a.
- Each clause $C_i = (c_{i1} \lor c_{i2} \lor c_{i3})$ is also represented by a triangle
$$[\,c_{i1}, c_{i2}, c_{i3}\,].$$
  - Node c_{ij} with the same label as one in some triangle $[\,a, x_k, \neg x_k\,]$ represent *distinct* nodes.
- There is an edge between c_{ij} and the node that represents the jth literal of C_i.

Construction for $\cdots \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge \cdots$



The Proof (continued)

Suppose the graph is 3-colorable.

- Assume without loss of generality that node a takes the color 2.
- A triangle must use up all 3 colors.
- As a result, one of x_i and $\neg x_i$ must take the color 0 and the other 1.

The Proof (continued)

- Treat 1 as `true` and 0 as `false`.[a]
  - We were dealing only with those triangles with the a node, not the clause triangles.
- The resulting truth assignment is clearly contradiction free.
- As each clause triangle contains one color 1 and one color 0, the clauses are NAE-satisfied.

[a]The opposite also works.

The Proof (continued)

Suppose the clauses are NAE-satisfiable.

- Color node a with color 2.
- Color the nodes representing literals by their truth values (color 0 for `false` and color 1 for `true`).
  - We were dealing only with those triangles with the a node, not the clause triangles.

The Proof (concluded)

- For each clause triangle:
  - Pick any two literals with opposite truth values.
  - Color the corresponding nodes with 0 if the literal is `true` and 1 if it is `false`.
  - Color the remaining node with color 2.
- The coloring is legitimate.
  - If literal w of a clause triangle has color 2, then its color will never be an issue.
  - If literal w of a clause triangle has color 1, then it must be connected up to literal w with color 0.
  - If literal w of a clause triangle has color 0, then it must be connected up to literal w with color 1.

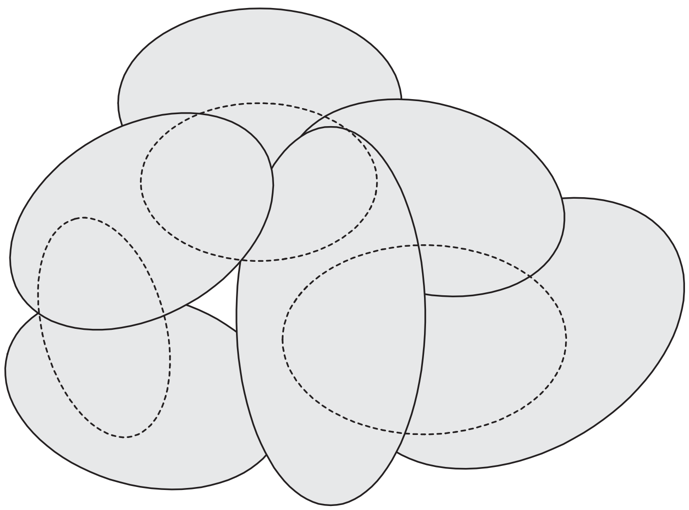
TRIPARTITE MATCHING

- We are given three sets B, G, and H, each containing n elements.
- Let $T \subseteq B \times G \times H$ be a ternary relation.
- TRIPARTITE MATCHING asks if there is a set of n triples in T, none of which has a component in common.
  - Each element in B is matched to a different element in G and different element in H.

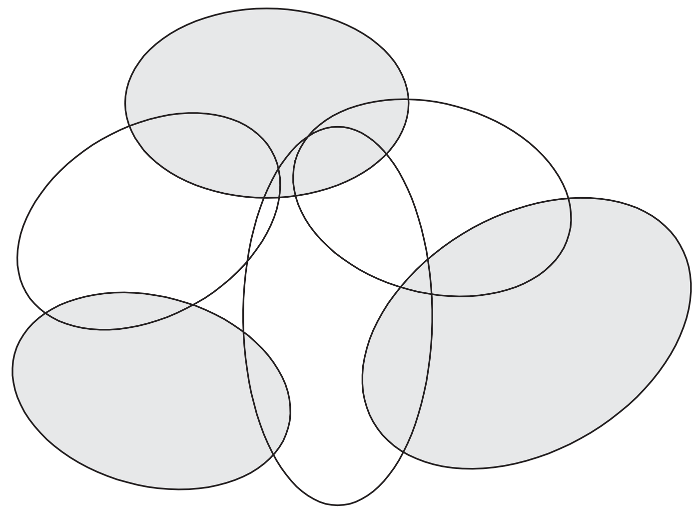
Theorem 39 (Karp (1972)) TRIPARTITE MATCHING *is NP-complete.*

Related Problems

- We are given a family $F = \{S_1, S_2, \ldots, S_n\}$ of subsets of a finite set U and a budget B.
- SET COVERING asks if there exists a set of B sets in F whose union is U.
- SET PACKING asks if there are B disjoint sets in F.
- Assume $|U| = 3m$ for some $m \in \mathbb{N}$ and $|S_i| = 3$ for all i.
- EXACT COVER BY 3-SETS asks if there are m sets in F that are disjoint and have U as their union.



SET COVERING



SET PACKING

Related Problems (concluded)

Corollary 40 SET COVERING, SET PACKING, *and* EXACT COVER BY 3-SETS *are all NP-complete.*

The KNAPSACK Problem

- There is a set of n items.
- Item i has value $v_i \in \mathbb{Z}^+$ and weight $w_i \in \mathbb{Z}^+$.
- We are given $K \in \mathbb{Z}^+$ and $W \in \mathbb{Z}^+$.
- KNAPSACK asks if there exists a subset $S \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{i \in S} w_i \leq W$ and $\sum_{i \in S} v_i \geq K$.
  - We want to achieve the maximum satisfaction within the budget.

KNAPSACK Is NP-Complete

- KNAPSACK $\in$ NP: Guess an S and verify the constraints.
- We assume $v_i = w_i$ for all i and $K = W$.
- KNAPSACK now asks if a subset of $\{v_1, v_2, \ldots, v_n\}$ adds up to exactly K.
  - Picture yourself as a radio DJ.
  - Or a person trying to control the calories intake.
- We shall reduce EXACT COVER BY 3-SETS to KNAPSACK.

The Proof (continued)

- We are given a family $F = \{S_1, S_2, \ldots, S_n\}$ of size-3 subsets of $U = \{1, 2, \ldots, 3m\}$.
- EXACT COVER BY 3-SETS asks if there are m disjoint sets in F that cover the set U.
- Think of a set as a bit vector in $\{0, 1\}^{3m}$.
  - `001100010` means the set $\{3, 4, 8\}$, and `110010000` means the set $\{1, 2, 5\}$.
- Our goal is $\overbrace{\texttt{11}\cdots\texttt{1}}^{3m}$.

The Proof (continued)

- A bit vector can also be considered as a binary *number*.
- Set union resembles addition.
  - $\texttt{001100010} + \texttt{110010000} = \texttt{111110010}$, which denotes the set $\{1, 2, 3, 4, 5, 8\}$, as desired.
- Trouble occurs when there is *carry*.
  - $\texttt{001100010} + \texttt{001110000} = \texttt{010010010}$, which denotes the set $\{2, 5, 8\}$, not the desired $\{3, 4, 5, 8\}$.

The Proof (continued)

- Carry may also lead to a situation where we obtain our solution $\texttt{11}\cdots\texttt{1}$ with more than m sets in F.
  - $\texttt{001100010} + \texttt{001110000} + \texttt{101100000} + \texttt{000001101} = \texttt{111111111}$.
  - But this "solution" $\{1, 3, 4, 5, 6, 7, 8, 9\}$ does not correspond to an exact cover.
  - And it uses 4 sets instead of the required 3.[a]
- To fix this problem, we enlarge the base just enough so that there are no carries.
- Because there are n vectors in total, we change the base from 2 to $n + 1$.

[a]Thanks to a lively class discussion on November 20, 2002.

The Proof (continued)

- Set v_i to be the $(n+1)$-ary number corresponding to the bit vector encoding S_i.
- Now in base $n+1$, if there is a set S such that $\sum_{v_i \in S} v_i = \overbrace{11 \cdots 1}^{3m}$, then every bit position must be contributed by exactly one v_i and $|S| = m$.
- Finally, set

$$K = \sum_{j=0}^{3m-1} (n+1)^j = \overbrace{11 \cdots 1}^{3m} \quad \text{(base } n+1\text{)}.$$

The Proof (continued)

- Suppose F admits an exact cover, say $\{S_1, S_2, \ldots, S_m\}$.
- Then picking $S = \{v_1, v_2, \ldots, v_m\}$ clearly results in

$$v_1 + v_2 + \cdots + v_m = \overbrace{11 \cdots 1}^{3m}.$$

  - It is important to note that the meaning of addition (+) is independent of the base.[a]
  - It is just regular addition.
  - But a S_i may give rise to different v_i's under different bases.

---

[a] Contributed by Mr. Kuan-Yu Chen (R92922047) on November 3, 2004.

The Proof (concluded)

- On the other hand, suppose there exists an S such that $\sum_{v_i \in S} v_i = \overbrace{11 \cdots 1}^{3m}$ in base $n+1$.
- The no-carry property implies that $|S| = m$ and $\{S_i : v_i \in S\}$ is an exact cover.

An Example

- Let $m = 3$, $U = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, and

$$\begin{aligned} S_1 &= \{1, 3, 4\}, \\ S_2 &= \{2, 3, 4\}, \\ S_3 &= \{2, 5, 6\}, \\ S_4 &= \{6, 7, 8\}, \\ S_5 &= \{7, 8, 9\}. \end{aligned}$$

- Note that $n = 5$, as there are 5 S_i's.

An Example (concluded)

- Our reduction produces

$$
\begin{aligned}
K &= \sum_{j=0}^{3\times 3-1} 6^j = \overbrace{11\cdots 1}^{3\times 3} \quad \text{(base 6)} = 2015539,\\
v_1 &= \texttt{101100000} = 1734048,\\
v_2 &= \texttt{011100000} = 334368,\\
v_3 &= \texttt{010011000} = 281448,\\
v_4 &= \texttt{000001110} = 258,\\
v_5 &= \texttt{000000111} = 43.
\end{aligned}
$$

- Note $v_1 + v_3 + v_5 = K$.
- Indeed, $S_1 \cup S_3 \cup S_5 = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, an exact cover by 3-sets.

BIN PACKINGS

- We are given N positive integers $a_1, a_2, \ldots, a_N$, an integer C (the capacity), and an integer B (the number of bins).
- BIN PACKING asks if these numbers can be partitioned into B subsets, each of which has total sum at most C.
- Think of packing bags at the check-out counter.

Theorem 41 BIN PACKING *is NP-complete.*

INTEGER PROGRAMMING

- INTEGER PROGRAMMING asks whether a system of linear inequalities with integer coefficients has an integer solution.
  - LINEAR PROGRAMMING asks whether a system of linear inequalities with integer coefficients has a *rational* solution.

INTEGER PROGRAMMING Is NP-Complete[a]

- SET COVERING can be expressed by the inequalities $Ax \geq \vec{1}$, $\sum_{i=1}^{n} x_i \leq B$, $0 \leq x_i \leq 1$, where
  - x_i is one if and only if S_i is in the cover.
  - A is the matrix whose columns are the bit vectors of the sets $S_1, S_2, \ldots$.
  - $\vec{1}$ is the vector of 1s.
- This shows INTEGER PROGRAMMING is NP-hard.
- Many NP-complete problems can be expressed as an INTEGER PROGRAMMING problem.

[a]Papadimitriou (1981).

Easier or Harder?[a]

- Adding restrictions on the allowable *problem instances* will not make a problem harder.
  - We are now solving a subset of problem instances.
  - The INDEPENDENT SET proof (p. 277) and the KNAPSACK proof (p. 322).
  - SAT to 2SAT (easier by p. 264).
  - CIRCUIT VALUE to MONOTONE CIRCUIT VALUE (equally hard by p. 241).

[a]Thanks to a lively class discussion on October 29, 2003.

Easier or Harder? (concluded)

- Adding restrictions on the allowable *solutions* may make a problem easier, as hard, or harder.
- It is problem dependent.
  - MIN CUT to BISECTION WIDTH (harder by p. 303).
  - LINEAR PROGRAMMING to INTEGER PROGRAMMING (harder by p. 332).
  - SAT to NAESAT (equally hard by p. 272) and MAX CUT to MAX BISECTION (equally hard by p. 301).
  - 3-COLORING to 2-COLORING (easier by p. 309).

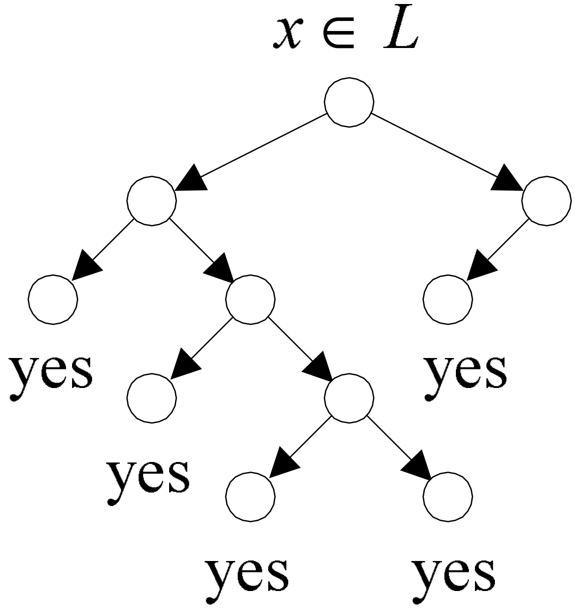
coNP and Function Problems

coNP

- By definition, coNP is the class of problems whose complement is in NP.
- NP is the class of problems that have succinct certificates (recall Proposition 30 on p. 251).
- coNP is therefore the class of problems that have succinct disqualifications:
  - A "no" instance of a problem in coNP possesses a short proof of its being a "no" instance.
  - Only "no" instances have such proofs.

coNP (continued)

- Suppose L is a coNP problem.
- There exists a polynomial-time nondeterministic algorithm M such that:
  - If $x \in L$, then $M(x) =$ "yes" for all computation paths.
  - If $x \notin L$, then $M(x) =$ "no" for some computation path.


$x \in L$
yes
yes
yes
yes
yes
$x \notin L$
no
yes
no
yes
yes

coNP (concluded)

- Clearly $\mathrm{P} \subseteq \mathrm{coNP}$.
- It is not known if

$$\mathrm{P} = \mathrm{NP} \cap \mathrm{coNP}.$$

  – Contrast this with

$$\mathrm{R} = \mathrm{RE} \cap \mathrm{coRE}$$

  (see Proposition 11 on p. 124).

Some coNP Problems

- VALIDITY $\in$ coNP.
  - If ϕ is not valid, it can be disqualified very succinctly: a truth assignment that does not satisfy it.
- SAT COMPLEMENT $\in$ coNP.
  - The disqualification is a truth assignment that satisfies it.
- HAMILTONIAN PATH COMPLEMENT $\in$ coNP.
  - The disqualification is a Hamiltonian path.
- OPTIMAL TSP (D) $\in$ coNP.[a]
  - The disqualification is a tour with a length $< B$.

[a] Asked by Mr. Che-Wei Chang (R95922093) on September 27, 2006.

An Alternative Characterization of coNP

Proposition 42 *Let $L \subseteq \Sigma^*$ be a language. Then $L \in coNP$ if and only if there is a polynomially decidable and polynomially balanced relation R such that*

$$L = \{x : \forall y \, (x, y) \in R\}.$$

(As on p. 250, we assume $|\, y \,| \leq |\, x \,|^k$ for some k.)

- $\bar{L} = \{x : (x, y) \in \neg R \text{ for some } y\}$.
- Because $\neg R$ remains polynomially balanced, $\bar{L} \in \text{NP}$ by Proposition 30 (p. 251).
- Hence $L \in \text{coNP}$ by definition.

coNP Completeness

Proposition 43 *L is NP-complete if and only if its complement $\bar{L} = \Sigma^* - L$ is coNP-complete.*

Proof ($\Rightarrow$; the $\Leftarrow$ part is symmetric)

- Let $\bar{L}'$ be any coNP language.
- Hence $L' \in$ NP.
- Let R be the reduction from L' to L.
- So $x \in L'$ if and only if $R(x) \in L$.
- So $x \in \bar{L}'$ if and only if $R(x) \in \bar{L}$.
- R is a reduction from $\bar{L}'$ to $\bar{L}$.

Some coNP-Complete Problems

- SAT COMPLEMENT is coNP-complete.
  - SAT COMPLEMENT is the complement of SAT.
- VALIDITY is coNP-complete.
  - ϕ is valid if and only if $\neg\phi$ is not satisfiable.
  - The reduction from SAT COMPLEMENT to VALIDITY is hence easy.
- HAMILTONIAN PATH COMPLEMENT is coNP-complete.

Possible Relations between P, NP, coNP

1. $\mathrm{P} = \mathrm{NP} = \mathrm{coNP}$.
2. $\mathrm{NP} = \mathrm{coNP}$ but $\mathrm{P} \neq \mathrm{NP}$.
3. $\mathrm{NP} \neq \mathrm{coNP}$ and $\mathrm{P} \neq \mathrm{NP}$.
   - This is current "consensus."

coNP Hardness and NP Hardness[a]

Proposition 44 *If a coNP-hard problem is in NP, then NP = coNP.*

- Let $L \in \text{NP}$ be coNP-hard.
- Let NTM M decide L.
- For any $L' \in \text{coNP}$, there is a reduction R from L' to L.
- $L' \in \text{NP}$ as it is decided by NTM $M(R(x))$.
  - Alternatively, NP is closed under complement.
- Hence $\text{coNP} \subseteq \text{NP}$.
- The other direction $\text{NP} \subseteq \text{coNP}$ is symmetric.

[a] Brassard (1979); Selman (1978).

coNP Hardness and NP Hardness (concluded)

Similarly,

Proposition 45 *If an NP-hard problem is in coNP, then NP = coNP.*

Hence NP-complete problems are unlikely to be in coNP and coNP-complete problems are unlikely to be in NP.

The Primality Problem

- An integer p is **prime** if $p > 1$ and all positive numbers other than 1 and p itself cannot divide it.
- PRIMES asks if an integer N is a prime number.
- Dividing N by $2, 3, \ldots, \sqrt{N}$ is *not* efficient.
  - The length of N is only $\log N$, but $\sqrt{N} = 2^{0.5 \log N}$.
- A polynomial-time algorithm for PRIMES was not found until 2002 by Agrawal, Kayal, and Saxena!
- We will focus on efficient "probabilistic" algorithms for PRIMES (used in *Mathematica*, e.g.).

```
if n = a^b for some a, b > 1 then
    return "composite";
end if
for r = 2, 3, ..., n − 1 do
    if gcd(n, r) > 1 then
        return "composite";
    end if
    if r is a prime then
        Let q be the largest prime factor of r − 1;
        if q ≥ 4√r log n and n^((r−1)/q) ≠ 1 mod r then
            break; {Exit the for-loop.}
        end if
    end if
end for{r − 1 has a prime factor q ≥ 4√r log n.}
for a = 1, 2, ..., 2√r log n do
    if (x − a)^n ≠ (x^n − a) mod (x^r − 1) in Z_n[x] then
        return "composite";
    end if
end for
return "prime"; {The only place with "prime" output.}
```

DP

- $\text{DP} \equiv \text{NP} \cap \text{coNP}$ is the class of problems that have succinct certificates and succinct disqualifications.
  - Each "yes" instance has a succinct certificate.
  - Each "no" instance has a succinct disqualification.
  - No instances have both.
- $\text{P} \subseteq \text{DP}$.
- We will see that PRIMES $\in$ DP.
  - In fact, PRIMES $\in$ P as mentioned earlier.

Primitive Roots in Finite Fields

Theorem 46 (Lucas and Lehmer (1927)) [a] *A number $p > 1$ is prime if and only if there is a number $1 < r < p$ (called the* **primitive root** *or* **generator***) such that*

1. $r^{p-1} = 1 \bmod p$, *and*
2. $r^{(p-1)/q} \neq 1 \bmod p$ *for all prime divisors q of $p-1$.*

- We will prove the theorem later.

---

[a] François Edouard Anatole Lucas (1842–1891); Derrick Henry Lehmer (1905–1991).

Pratt's Theorem

Theorem 47 (Pratt (1975)) PRIMES $\in NP \cap coNP$.

- PRIMES is in coNP because a succinct disqualification is a divisor.
- Suppose p is a prime.
- p's certificate includes the r in Theorem 46 (p. 351).
- Use recursive doubling to check if $r^{p-1} = 1 \bmod p$ in time polynomial in the length of the input, $\log_2 p$.
- We also need all *prime* divisors of $p-1$: $q_1, q_2, \ldots, q_k$.
- Checking $r^{(p-1)/q_i} \neq 1 \bmod p$ is also easy.

The Proof (concluded)

- Checking $q_1, q_2, \ldots, q_k$ are all the divisors of $p-1$ is easy.
- We still need certificates for the primality of the q_i's.
- The complete certificate is recursive and tree-like:

$$C(p) = (r; q_1, C(q_1), q_2, C(q_2), \ldots, q_k, C(q_k)).$$

- $C(p)$ can also be checked in polynomial time.
- We next prove that $C(p)$ is succinct.

The Succinctness of the Certificate

Lemma 48 *The length of $C(p)$ is at most quadratic at $5\log_2^2 p$.*

- This claim holds when $p = 2$ or $p = 3$.
- In general, $p - 1$ has $k < \log_2 p$ prime divisors $q_1 = 2, q_2, \ldots, q_k$.
- $C(p)$ requires: 2 parentheses and $2k < 2\log_2 p$ separators (length at most $2\log_2 p$ long), r (length at most $\log_2 p$), $q_1 = 2$ and its certificate 1 (length at most 5 bits), the q_i's (length at most $2\log_2 p$), and the $C(q_i)$s.

The Proof (concluded)

- $C(p)$ is succinct because

$$
\begin{aligned}
|C(p)| &\leq 5\log_2 p + 5 + 5\sum_{i=2}^{k} \log_2^2 q_i \\
&\leq 5\log_2 p + 5 + 5\left(\sum_{i=2}^{k} \log_2 q_i\right)^2 \\
&\leq 5\log_2 p + 5 + 5\log_2^2 \frac{p-1}{2} \\
&< 5\log_2 p + 5 + 5(\log_2 p - 1)^2 \\
&= 5\log_2^2 p + 10 - 5\log_2 p \leq 5\log_2^2 p
\end{aligned}
$$

for $p \geq 4$.

Basic Modular Arithmetics[a]

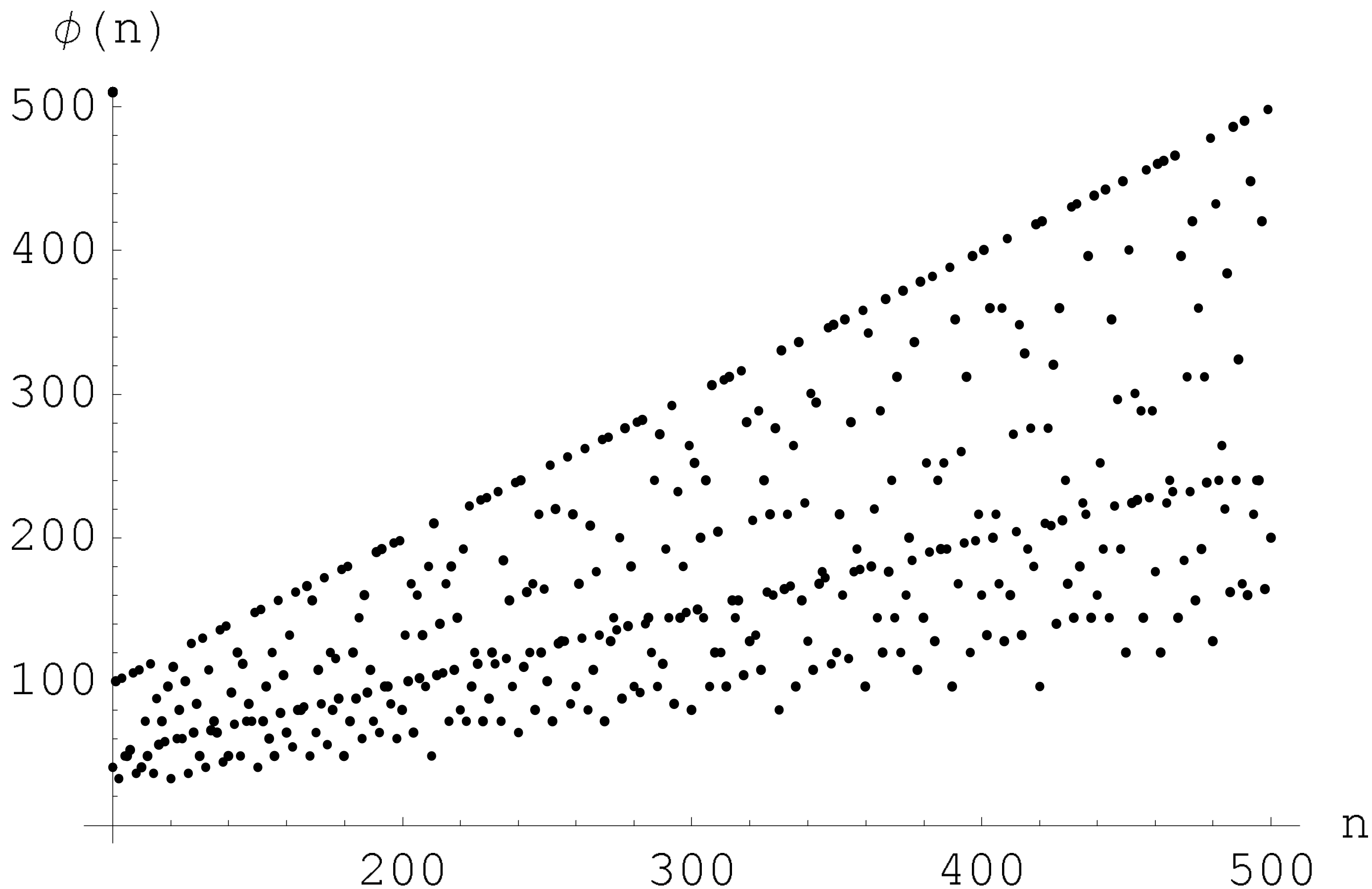
- Let $m, n \in \mathbb{Z}^+$.
- $m|n$ means m divides n and m is n's **divisor**.
- We call the numbers $0, 1, \ldots, n-1$ the **residue** modulo n.
- The **greatest common divisor** of m and n is denoted $\gcd(m, n)$.
- The r in Theorem 46 (p. 351) is a primitive root of p.
- We now prove the existence of primitive roots and then Theorem 46.

[a]Carl Friedrich Gauss.

Euler's[a] Totient or Phi Function

- Let
$$\Phi(n) = \{m : 1 \leq m < n, \gcd(m, n) = 1\}$$
be the set of all positive integers less than n that are prime to n (Z_n^* is a more popular notation).
  - $\Phi(12) = \{1, 5, 7, 11\}$.
- Define **Euler's function** of n to be $\phi(n) = |\Phi(n)|$.
- $\phi(p) = p - 1$ for prime p, and $\phi(1) = 1$ by convention.
- Euler's function is not expected to be easy to compute without knowing n's factorization.

---
[a]Leonhard Euler (1707–1783).


φ(n)
500
400
300
200
100
200
300
400
500
n

Two Properties of Euler's Function

The inclusion-exclusion principle[a] can be used to prove the following.

Lemma 49 $\phi(n) = n \prod_{p|n} (1 - \frac{1}{p})$.

- If $n = p_1^{e_1} p_2^{e_2} \cdots p_t^{e_t}$ is the prime factorization of n, then

$$\phi(n) = n \prod_{i=1}^{t} \left(1 - \frac{1}{p_i}\right).$$

Corollary 50 $\phi(mn) = \phi(m)\,\phi(n)$ *if* $\gcd(m, n) = 1$.

[a]See my *Discrete Mathematics* lecture notes.

A Key Lemma

Lemma 51 $\sum_{m|n} \phi(m) = n$.

- Let $\prod_{i=1}^{\ell} p_i^{k_i}$ be the prime factorization of n and consider

$$\prod_{i=1}^{\ell} [\, \phi(1) + \phi(p_i) + \cdots + \phi(p_i^{k_i}) \,]. \tag{4}$$

- Equation (4) equals n because $\phi(p_i^k) = p_i^k - p_i^{k-1}$ by Lemma 49.
- Expand Eq. (4) to yield $\sum_{k'_1 \leq k_1, \ldots, k'_\ell \leq k_\ell} \prod_{i=1}^{\ell} \phi(p_i^{k'_i})$.

The Proof (concluded)

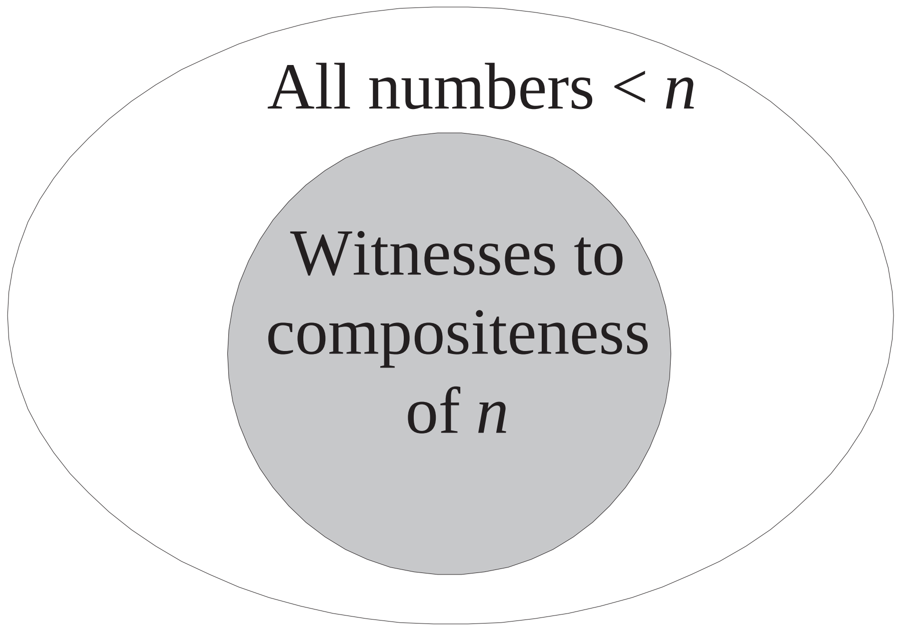
- By Corollary 50 (p. 359),

$$\prod_{i=1}^{\ell} \phi(p_i^{k_i'}) = \phi\left(\prod_{i=1}^{\ell} p_i^{k_i'}\right).$$

- Each $\prod_{i=1}^{\ell} p_i^{k_i'}$ is a unique divisor of $n = \prod_{i=1}^{\ell} p_i^{k_i}$.
- Equation (4) becomes

$$\sum_{m|n} \phi(m).$$

The Density Attack for PRIMES



- It works, but does it work well?

Factorization and Euler's Function

- The ratio of numbers $\leq n$ relatively prime to n is $\phi(n)/n$.
- When $n = pq$, where p and q are distinct primes,

$$\frac{\phi(n)}{n} = \frac{pq - p - q + 1}{pq} > 1 - \frac{1}{q} - \frac{1}{p}.$$

  – The "density attack" to factor $n = pq$ hence takes $\Omega(\sqrt{n})$ steps on average when $p \sim q = O(\sqrt{n}\,)$.
  – This running time is exponential: $\Omega(2^{0.5 \log_2 n})$.

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

- 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 0 and $p - 1$.
  - Suppose $am = am' \bmod p$ 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 $a\Phi(p) = \Phi(p)$, $(p-1)! = a^{p-1}(p-1)! \bmod p$.
- Finally, $a^{p-1} = 1 \bmod p$ because $p \not| (p-1)!$.

The Fermat-Euler Theorem[a]

Corollary 53 *For all $a \in \Phi(n)$, $a^{\phi(n)} = 1 \bmod n$.*

- The proof is similar to that of Lemma 52 (p. 365).
- 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 $am = am' \bmod n$ 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`) on November 24, 2004.

The Proof (concluded)

- 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 $a\Phi(n) = \Phi(n)$,

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

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

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, say $s^i = s^j \bmod p$, which means $s^{j-i} = 1 \bmod p$.
- If the exponent of m is k and $m^\ell = 1 \bmod p$, then $k | \ell$.
  - Otherwise, $\ell = qk + a$ for $0 < a < k$, and $m^\ell = m^{qk+a} = m^a = 1 \bmod p$, a contradiction.

Lemma 54 *Any nonzero polynomial of degree k has at most k distinct roots modulo p.*

Exponents and Primitive Roots

- From Fermat's "little" theorem, 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 $\Phi(p)$ 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 $x^k = 1 \bmod p$.
- Hence there are at most k residues of exponent k, i.e., $R(k) \leq k$, by Lemma 54 on p. 370.
- Let s be a residue of exponent k.
- $1, s, s^2, \ldots, s^{k-1}$ are all distinct modulo p.
  - Otherwise, $s^i = s^j \bmod p$ with $i < j$ and s is of exponent $j - i < k$, a contradiction.
- As all these k distinct numbers satisfy $x^k = 1 \bmod p$, they are all the solutions of $x^k = 1 \bmod p$.
- But do all of them have exponent k (i.e., $R(k) = k$)?

Size of $R(k)$ (continued)

- And if not (i.e., $R(k) < k$), how many of them do?
- Suppose $\ell < k$ and $\ell \notin \Phi(k)$ with $\gcd(\ell, k) = d > 1$.
- Then
$$(s^{\ell})^{k/d} = (s^{k})^{\ell/d} = 1 \bmod p.$$
- Therefore, s^{ℓ} has exponent at most k/d, which is less than 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 50 on p. 359.

- 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 46 (p. 351).

A Few Calculations

- Let $p = 13$.
- From p. 367, 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 46 (p. 351)

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

The Proof (concluded)

- $r^{\phi(p)} = 1 \bmod p$ by the Fermat-Euler theorem (p. 367).
- Because p is not a prime, $\phi(p) < p - 1$.
- Let k be the smallest integer such that $r^k = 1 \bmod p$.
- 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,

$$r^{(p-1)/q} = 1 \bmod p.$$

- But this violates the 2nd condition.

Function Problems

- Decisions problem 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 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 ϕ 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

```
t := ε;
if φ ∈ SAT then
    for i = 1, 2, ..., n do
        if φ[ x_i = true ] ∈ SAT then
            t := t ∪ { x_i = true };
            φ := φ[ x_i = true ];
        else
            t := t ∪ { x_i = false };
            φ := φ[ x_i = false ];
        end if
    end for
    return t;
else
    return "no";
end if
```

Analysis

- There are $\leq n + 1$ calls to the algorithm for SAT.[a]
- Shorter boolean expressions than ϕ 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 $2^{|x|}$, where x 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.

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, ..., n do
    Call TSP (D) with B = C and d_ij = C + 1;
    if "no" then
        Restore d_ij to old value; {Edge [i, j] is critical.}
    end if
end for
return the tour with edges whose d_ij ≤ C;
```

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 $O(|x| + n^2)$ 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).

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

[a]Rabin (1976); Solovay and Strassen (1977).

"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 and Strassen (1977).
[c] Aleliunas, Karp, Lipton, Lovász, and Rackoff (1979).
[d] Schwartz (1980); Zippel (1979).
[e] Sinclair and Jerrum (1989).

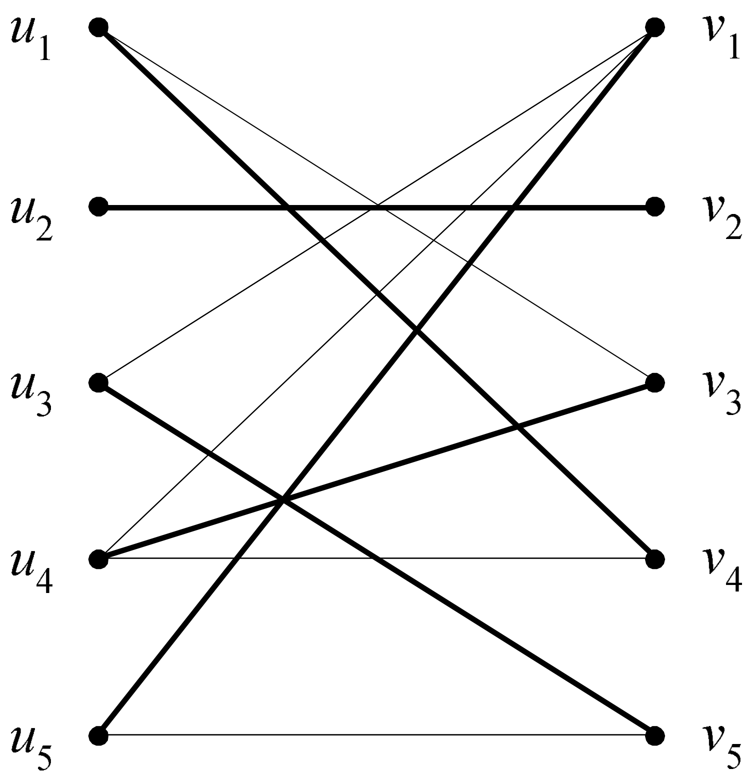
Bipartite Perfect Matching

- 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 π 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

$$\det(A^G) = \sum_{\pi} \operatorname{sgn}(\pi) \prod_{i=1}^{n} A^G_{i,\pi(i)}. \tag{5}$$

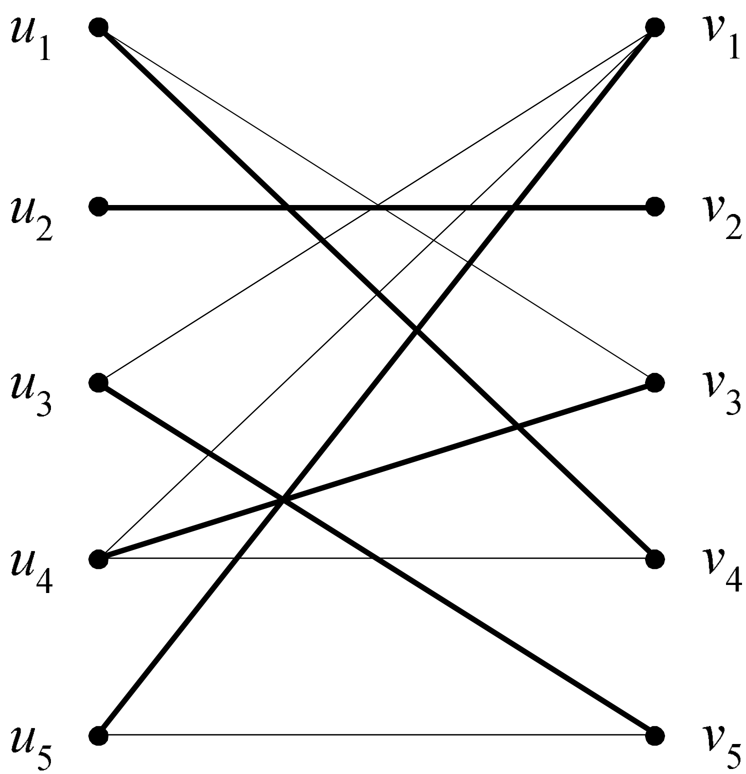
  - π ranges over all permutations of n elements.
  - $\operatorname{sgn}(\pi)$ is 1 if π is the product of an even number of transpositions and -1 otherwise.

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 prefect matching π.
  - As all variables appear only *once*, all of these summands are different monomials and will not cancel.
- It is essentially an exhaustive enumeration.

Proposition 55 (Edmonds (1967)) *G has a perfect matching if and only if* $\det(A^G)$ *is not identically zero.*

A Perfect Matching in a Bipartite Graph



The Perfect Matching in the Determinant

- The matrix is

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

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

Number of Roots of a Polynomial

Lemma 56 (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

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

- 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.

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 identically zero";
end if

A Randomized Bipartite Perfect Matching Algorithm[a]

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

Choose n^2 integers $i_{11}, \ldots, i_{nn}$ from $\{0, 1, \ldots, b-1\}$ randomly;
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 no perfect matchings";
end if

---

[a] Lovász (1979).

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 $n^2 d/b = 0.5$ 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?[a]

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

Perfect Matching for General Graphs

- Page 390 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 π of $\{1, 2, \ldots, 2n\}$ such that

$$(v_i, v_{\pi(i)}) \in E$$

    for all $v_i \in V$.

The Tutte Matrix[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 55 (p. 393):

Proposition 57 *G has a perfect matching if and only if $\det(T^G)$ is not identically zero.*

---

[a]William Thomas Tutte (1917–2002).

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**).
  - If the algorithm answers in the negative, then it may make an error (**false negative**).
- The algorithm makes a false negative with probability ≤ 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.

[a]Metropolis and Ulam (1949).

The Markov Inequality[a]

Lemma 58 *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 $x = i$.

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

[a] 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. 380)

- Let $\phi(x_1, x_2, \ldots, x_n)$ be a k-SAT formula.
- If ϕ 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 ϕ in CNF Form

Start with an *arbitrary* truth assignment T;
for $i = 1, 2, \ldots, r$ **do**
    if $T \models \phi$ **then**
        return "ϕ is satisfiable with T";
    else
        Let c be an unsatisfiable clause in ϕ 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 "ϕ is unsatisfiable";

3SAT vs. 2SAT Again

- Note that if ϕ 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 $r = 3n$,[a] much better than $O(2^n)$.[b]
- We will show immediately that it works well for 2SAT.
- The state of the art is expected $O(1.322^n)$ time for 3SAT and expected $O(1.474^n)$ time for 4SAT.[c]

[a] Use this setting per run of the algorithm.
[b] Schöning (1999).
[c] Kwama and Tamaki (2004); Rolf (2006).

Random Walk Works for 2SAT[a]

Theorem 59 *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 $\hat{T} \models \phi$.
- Let $t(i)$ denote the expected number of repetitions of the flipping step until a satisfying truth assignment is found if our starting T differs from $\hat{T}$ in i values.
  - Their Hamming distance is i.

[a]Papadimitriou (1991).

The Proof

- 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 T is not equal to any other satisfying truth assignment, then we need to flip at least once.
- We flip 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.
- So we have at least 0.5 chance of moving closer to $\hat{T}$.

The Proof (continued)

- 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)

- As we are only interested in upper bounds, we solve

$$\begin{aligned} x(0) &= 0 \\ x(n) &= x(n-1)+1 \\ x(i) &= \frac{x(i-1)+x(i+1)}{2}+1, \quad 0<i<n \end{aligned}$$

- This is one-dimensional random walk with a reflecting and an absorbing barrier.

The Proof (continued)

- Add the equations up to obtain

$$\begin{aligned}
& x(1) + x(2) + \cdots + x(n) \\
= \quad & \frac{x(0) + x(1) + 2x(2) + \cdots + 2x(n-2) + x(n-1) + x(n)}{2} \\
& +n + x(n-1).
\end{aligned}$$

- Simplify to yield

$$\frac{x(1) + x(n) - x(n-1)}{2} = n.$$

- As $x(n) - x(n-1) = 1$, we have

$$x(1) = 2n - 1.$$

The Proof (continued)

- Iteratively, we obtain

$$\begin{aligned} x(2) &= 4n - 4, \\ &\vdots \\ x(i) &= 2in - i^2. \end{aligned}$$

- The worst case happens when $i = n$, in which case

$$x(n) = n^2.$$

The Proof (concluded)

- We therefore reach the conclusion that

$$t(i) \leq x(i) \leq x(n) = n^2.$$

- So the expected number of steps is at most n^2.
- The algorithm picks a running time $2n^2$.
- This amounts to invoking the Markov inequality (p. 405) with $k = 2$, with the consequence of having 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 $\leq (2m)^{-1}$ by Markov's inequality.
- Alternatively, with the same running time, we can run the "$r = 2n^2$" algorithm m times.
- But the error probability is reduced to $\leq 2^{-m}$!
- Again, the gain comes from the fact that Markov's inequality does not take advantage of any specific feature of the random variable.
- The gain also comes from the fact that the two algorithms are different.

How about Random CNF?

- Select m clauses independently and uniformly from the set of all possible disjunctions of k distinct, non-complementary literals with n boolean variables.
- Let $m = cn$.
- The formula is satisfiable with probability approaching 1 as $n \to \infty$ if $c < c_k$ for some $c_k < 2^k \ln 2 - O(1)$.
- The formula is unsatisfiable with probability approaching 1 as $n \to \infty$ if $c > c_k$ for some $c_k > 2^k \ln 2 - O(k)$.
- The above bounds are not tight yet.

Primality Tests

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

The Density Attack for PRIMES

Pick $k \in \{2, \ldots, N-1\}$ randomly; {Assume $N > 2$.}
if $k \mid N$ **then**
  return "N is composite";
else
  return "N is a prime";
end if

Analysis[a]

- Suppose $N = PQ$, a product of 2 primes.
- The probability of success is
$$< 1 - \frac{\phi(N)}{N} = 1 - \frac{(P-1)(Q-1)}{PQ} = \frac{P+Q-1}{PQ}.$$
- In the case where $P \approx Q$, this probability becomes
$$< \frac{1}{P} + \frac{1}{Q} \approx \frac{2}{\sqrt{N}}.$$
- This probability is exponentially small.

[a] See also p. 363.

The Fermat Test for Primality

Fermat's "little" theorem on p. 365 suggests the following primality test for any given number p:

1: Pick a number a randomly from $\{1, 2, \ldots, N-1\}$;
2: **if** $a^{N-1} \neq 1 \bmod N$ **then**
3:     **return** "N is composite";
4: **else**
5:     **return** "N is probably a prime";
6: **end if**

The Fermat Test for Primality (concluded)

- Unfortunately, there are composite numbers called **Carmichael numbers** that will pass the Fermat test for *all* $a \in \{1, 2, \ldots, N-1\}$.
- There are infinitely many Carmichael numbers.[a]

[a] Alford, Granville, and Pomerance (1992).

Square Roots Modulo a Prime

- Equation $x^2 = a \bmod p$ has at most two (distinct) roots by Lemma 54 (p. 370).
  - The roots are called **square roots**.
  - Numbers a with square roots and $\gcd(a, p) = 1$ are called **quadratic residues**.
    - They are $1^2 \bmod p, 2^2 \bmod p, \ldots, (p-1)^2 \bmod p$.
- We shall show that a number either has two roots or has none, and testing which one is true is trivial.
- There are no known efficient *deterministic* algorithms to find the roots.

Euler's Test

Lemma 60 (Euler) *Let p be an odd prime and $a \neq 0 \bmod p$.*

1. *If $a^{(p-1)/2} = 1 \bmod p$, then $x^2 = a \bmod p$ has two roots.*
2. *If $a^{(p-1)/2} \neq 1 \bmod p$, then $a^{(p-1)/2} = -1 \bmod p$ and $x^2 = a \bmod p$ has no roots.*

- Let r be a primitive root of p.
- By Fermat's "little" theorem, $r^{(p-1)/2}$ is a square root of 1, so $r^{(p-1)/2} = \pm 1 \bmod p$.
- But as r is a primitive root, $r^{(p-1)/2} \neq 1 \bmod p$.
- Hence $r^{(p-1)/2} = -1 \bmod p$.

The Proof (continued)

- Suppose $a = r^{2j}$ for some $1 \leq j \leq (p-1)/2$.
- Then $a^{(p-1)/2} = r^{j(p-1)} = 1 \bmod p$ and its two *distinct* roots are $r^j, -r^j (= r^{j+(p-1)/2})$.
  - If $r^j = -r^j \bmod p$, then $2r^j = 0 \bmod p$, which implies $r^j = 0 \bmod p$, a contradiction.
- As $1 \leq j \leq (p-1)/2$, there are $(p-1)/2$ such a's.

The Proof (concluded)

- Each such a has 2 distinct square roots.
- The square roots of all the a's are distinct.
  - The square roots of different a's must be different.
- Hence the set of *square roots* is $\{1, 2, \ldots, p-1\}$.
  - I.e., $\bigcup_{1 \leq a \leq p-1} \{x : x^2 = a \bmod p\} = \{1, 2, \ldots, p-1\}$.
- If $a = r^{2j+1}$, then it has no roots because all the square roots have been taken.
- $a^{(p-1)/2} = \left[\, r^{(p-1)/2} \,\right]^{2j+1} = (-1)^{2j+1} = -1 \bmod p$.

The Legendre Symbol[a] and Quadratic Residuacity Test

- By Lemma 60 (p. 426) $a^{(p-1)/2} \bmod p = \pm 1$ for $a \neq 0 \bmod p$.
- For odd prime p, define the **Legendre symbol** $(a \mid p)$ as

$$(a \mid p) = \begin{cases} 0 & \text{if } p \mid a, \\ 1 & \text{if } a \text{ is a quadratic residue modulo } p, \\ -1 & \text{if } a \text{ is a } \textbf{quadratic nonresidue} \text{ modulo } p. \end{cases}$$

- Euler's test implies $a^{(p-1)/2} = (a \mid p) \bmod p$ for any odd prime p and any integer a.
- Note that $(ab|p) = (a|p)(b|p)$.

[a] Andrien-Marie Legendre (1752–1833).

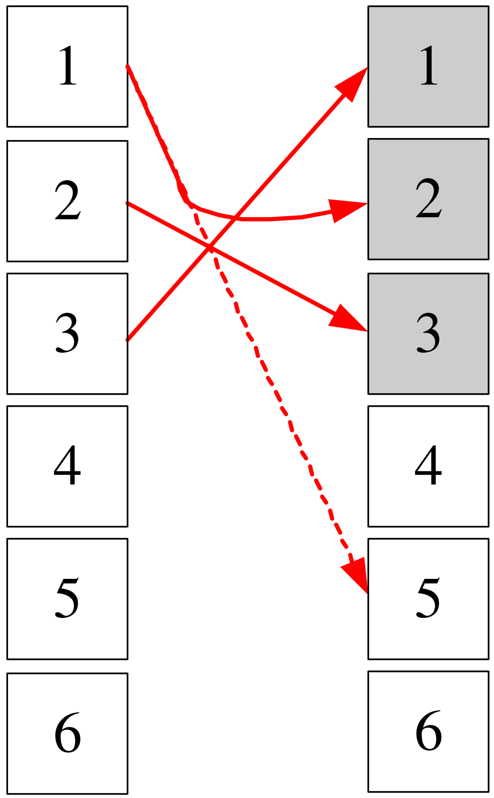
Gauss's Lemma

Lemma 61 (Gauss) *Let p and q be two odd primes. Then $(q|p) = (-1)^m$, where m is the number of residues in $R = \{iq \bmod p : 1 \leq i \leq (p-1)/2\}$ that are greater than $(p-1)/2$.*

- All residues in R are distinct.
  - If $iq = jq \bmod p$, then $p|(j-i)\,q$ or $p|q$.
- No two elements of R add up to p.
  - If $iq + jq = 0 \bmod p$, then $p|(i+j)$ or $p|q$.
  - But neither is possible.

The Proof (continued)

- Consider the set R' of residues that result from R if we replace each of the m elements $a \in R$ such that $a > (p-1)/2$ by $p - a$.
  - This is equivalent to performing $-a \bmod p$.
- All residues in R' are now at most $(p-1)/2$.
- In fact, $R' = \{1, 2, \ldots, (p-1)/2\}$ (see illustration next page).
  - Otherwise, two elements of R would add up to p, which has been shown to be impossible.



$p = 7$ and $q = 5$.

The Proof (concluded)

- Alternatively, $R' = \{\pm iq \bmod p : 1 \leq i \leq (p-1)/2\}$, where exactly m of the elements have the minus sign.
- Take the product of all elements in the two representations of R'.
- So $[(p-1)/2]! = (-1)^m q^{(p-1)/2} [(p-1)/2]! \bmod p$.
- Because $\gcd([(p-1)/2]!, p) = 1$, the above implies

$$1 = (-1)^m q^{(p-1)/2} \bmod p.$$

Legendre's Law of Quadratic Reciprocity[a]

- Let p and q be two odd primes.
- The next result says their Legendre symbols are distinct if and only if both numbers are 3 mod 4.

Lemma 62 (Legendre (1785), Gauss)

$$(p|q)(q|p) = (-1)^{\frac{p-1}{2}\frac{q-1}{2}}.$$

---

[a]First stated by Euler in 1751. Legendre (1785) did not give a correct proof. Gauss proved the theorem when he was 19. He gave at least 6 different proofs during his life. The 152nd proof appeared in 1963.

The Proof (continued)

- Sum the elements of R' in the previous proof in mod2.
- On one hand, this is just $\sum_{i=1}^{(p-1)/2} i \bmod 2$.
- On the other hand, the sum equals

$$\sum_{i=1}^{(p-1)/2} \left(qi - p \left\lfloor \frac{iq}{p} \right\rfloor \right) + mp \bmod 2$$

$$= \left(q \sum_{i=1}^{(p-1)/2} i - p \sum_{i=1}^{(p-1)/2} \left\lfloor \frac{iq}{p} \right\rfloor \right) + mp \bmod 2.$$

  - Signs are irrelevant under mod2.
  - m is as in Lemma 61 (p. 430).

The Proof (continued)

- Ignore odd multipliers to make the sum equal

$$\left(\sum_{i=1}^{(p-1)/2} i - \sum_{i=1}^{(p-1)/2} \left\lfloor \frac{iq}{p} \right\rfloor \right) + m \bmod 2.$$

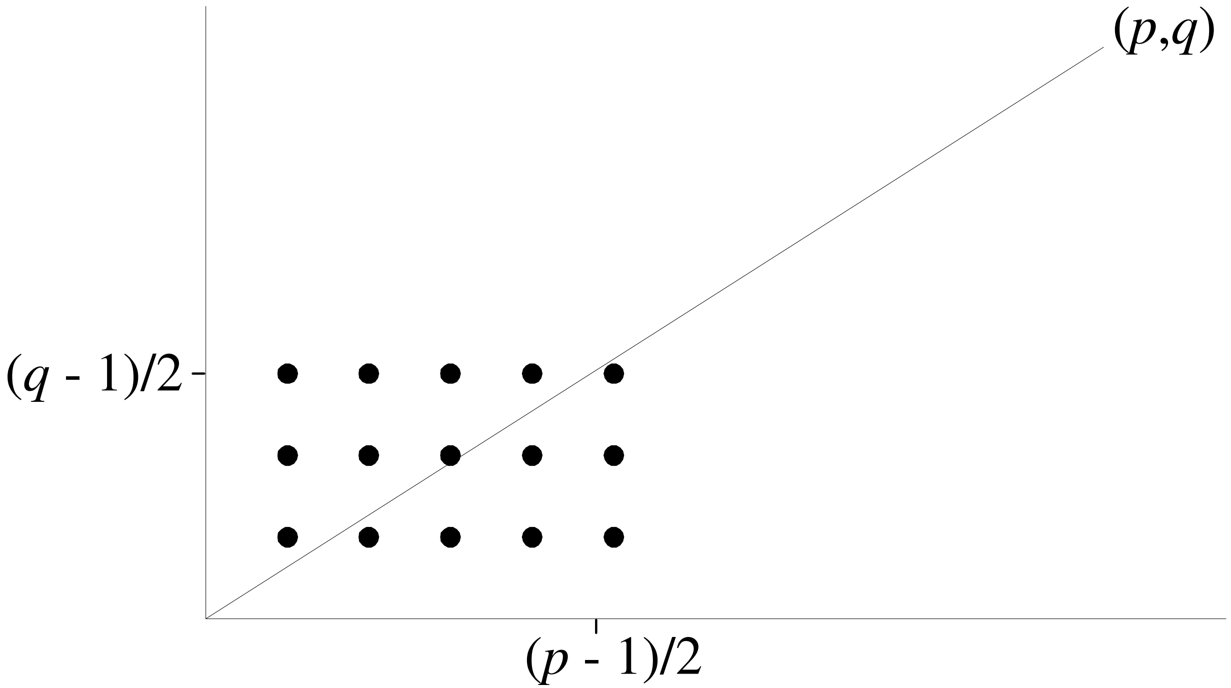
- Equate the above with $\sum_{i=1}^{(p-1)/2} i \bmod 2$ to obtain

$$m = \sum_{i=1}^{(p-1)/2} \left\lfloor \frac{iq}{p} \right\rfloor \bmod 2.$$

The Proof (concluded)

- $\sum_{i=1}^{(p-1)/2} \lfloor \frac{iq}{p} \rfloor$ is the number of integral points under the line $y = (q/p)\, x$ for $1 \leq x \leq (p-1)/2$.
- Gauss's lemma (p. 430) says $(q|p) = (-1)^m$.
- Repeat the proof with p and q reversed.
- So $(p|q) = (-1)^{m'}$, where m' is the number of integral points *above* the line $y = (q/p)\, x$ for $1 \leq y \leq (q-1)/2$.
- As a result, $(p|q)(q|p) = (-1)^{m+m'}$.
- But $m + m'$ is the total number of integral points in the $\frac{p-1}{2} \times \frac{q-1}{2}$ rectangle, which is $\frac{p-1}{2} \, \frac{q-1}{2}$.

Eisenstein's Rectangle



$p = 11$ and $q = 7$.

The Jacobi Symbol[a]

- The Legendre symbol only works for odd *prime* moduli.
- The **Jacobi symbol** $(a \mid m)$ extends it to cases where m is not prime.
- Let $m = p_1 p_2 \cdots p_k$ be the prime factorization of m.
- When $m > 1$ is odd and $\gcd(a, m) = 1$, then

$$(a|m) = \prod_{i=1}^{k} (a \mid p_i).$$

- Define $(a \mid 1) = 1$.

[a]Carl Jacobi (1804–1851).

Properties of the Jacobi Symbol

The Jacobi symbol has the following properties, for arguments for which it is defined.

1. $(ab \,|\, m) = (a \,|\, m)(b \,|\, m)$.
2. $(a \,|\, m_1 m_2) = (a \,|\, m_1)(a \,|\, m_2)$.
3. If $a = b \bmod m$, then $(a \,|\, m) = (b \,|\, m)$.
4. $(-1 \,|\, m) = (-1)^{(m-1)/2}$ (by Lemma 61 on p. 430).
5. $(2 \,|\, m) = (-1)^{(m^2-1)/8}$ (by Lemma 61 on p. 430).
6. If a and m are both odd, then $(a \,|\, m)(m \,|\, a) = (-1)^{(a-1)(m-1)/4}$.

Calculation of $(2200|999)$

Similar to the Euclidean algorithm and does *not* require factorization.

$$
\begin{aligned}
& (202|999) = (-1)^{(999^2-1)/8}(101|999) \\
= \quad & (-1)^{124750}(101|999) = (101|999) \\
= \quad & (-1)^{(100)(998)/4}(999|101) = (-1)^{24950}(999|101) \\
= \quad & (999|101) = (90|101) = (-1)^{(101^2-1)/8}(45|101) \\
= \quad & (-1)^{1275}(45|101) = -(45|101) \\
= \quad & -(-1)^{(44)(100)/4}(101|45) = -(101|45) = -(11|45) \\
= \quad & -(-1)^{(10)(44)/4}(45|11) = -(45|11) \\
= \quad & -(1|11) = -(11|1) = -1.
\end{aligned}
$$

A Result Generalizing Proposition 10.3 in the Textbook

Theorem 63 *The group of set* $\Phi(n)$ *under multiplication* mod n *has a primitive root if and only if* n *is either 1, 2, 4,* p^k*, or* $2p^k$ *for some nonnegative integer* k *and and odd prime* p*.*

This result is essential in the proof of the next lemma.

The Jacobi Symbol and Primality Test[a]

Lemma 64 *If* $(M|N) = M^{(N-1)/2} \bmod N$ *for all* $M \in \Phi(N)$, *then* N *is prime. (Assume* N *is odd.)*

- Assume $N = mp$, where p is an odd prime, $\gcd(m, p) = 1$, and $m > 1$ (not necessarily prime).
- Let $r \in \Phi(p)$ such that $(r \,|\, p) = -1$.
- The Chinese remainder theorem says that there is an $M \in \Phi(N)$ such that

$$\begin{aligned} M &= r \bmod p, \\ M &= 1 \bmod m. \end{aligned}$$

[a]Mr. Clement Hsiao (R88526067) pointed out that the textbook's proof in Lemma 11.8 is incorrect while he was a senior in January 1999.

The Proof (continued)

- By the hypothesis,

$$M^{(N-1)/2} = (M \mid N) = (M \mid p)(M \mid m) = -1 \bmod N.$$

- Hence

$$M^{(N-1)/2} = -1 \bmod m.$$

- But because $M = 1 \bmod m$,

$$M^{(N-1)/2} = 1 \bmod m,$$

  a contradiction.

The Proof (continued)

- Second, assume that $N = p^a$, where p is an odd prime and $a \geq 2$.
- By Theorem 63 (p. 442), there exists a primitive root r modulo p^a.
- From the assumption,

$$M^{N-1} = \left[M^{(N-1)/2} \right]^2 = (M|N)^2 = 1 \bmod N$$

for all $M \in \Phi(N)$.

The Proof (continued)

- As $r \in \Phi(N)$ (prove it), we have

$$r^{N-1} = 1 \bmod N.$$

- As r's exponent modulo $N = p^a$ is $\phi(N) = p^{a-1}(p-1)$,

$$p^{a-1}(p-1) \mid N-1,$$

  which implies that $p \mid N-1$.

- But this is impossible given that $p \mid N$.

The Proof (continued)

- Third, assume that $N = mp^a$, where p is an odd prime, $\gcd(m, p) = 1$, $m > 1$ (not necessarily prime), and a is even.
- The proof mimics that of the second case.
- By Theorem 63 (p. 442), there exists a primitive root r modulo p^a.
- From the assumption,

$$M^{N-1} = \left[M^{(N-1)/2} \right]^2 = (M|N)^2 = 1 \bmod N$$

for all $M \in \Phi(N)$.

The Proof (continued)

- In particular,
$$M^{N-1} = 1 \bmod p^a \tag{6}$$
for all $M \in \Phi(N)$.

- The Chinese remainder theorem says that there is an $M \in \Phi(N)$ such that
$$\begin{aligned} M &= r \bmod p^a, \\ M &= 1 \bmod m. \end{aligned}$$

- Because $M = r \bmod p^a$ and Eq. (6),
$$r^{N-1} = 1 \bmod p^a.$$

The Proof (concluded)

- As r's exponent modulo $N = p^a$ is $\phi(N) = p^{a-1}(p-1)$,
$$p^{a-1}(p-1) \mid N-1,$$
which implies that $p \mid N-1$.
- But this is impossible given that $p \mid N$.

The Number of Witnesses to Compositeness

Theorem 65 (Solovay and Strassen (1977)) *If N is an odd composite, then $(M|N) \neq M^{(N-1)/2} \bmod N$ for at least half of $M \in \Phi(N)$.*

- By Lemma 64 (p. 443) there is at least one $a \in \Phi(N)$ such that $(a|N) \neq a^{(N-1)/2} \bmod N$.
- Let $B = \{b_1, b_2, \ldots, b_k\} \subseteq \Phi(N)$ be the set of *all* distinct residues such that $(b_i|N) = b_i^{(N-1)/2} \bmod N$.
- Let $aB = \{ab_i \bmod N : i = 1, 2, \ldots, k\}$.

The Proof (concluded)

- $|aB| = k$.
  - $ab_i = ab_j \bmod N$ implies $N|a(b_i - b_j)$, which is impossible because $\gcd(a, N) = 1$ and $N > |b_i - b_j|$.
- $aB \cap B = \emptyset$ because

$$(ab_i)^{(N-1)/2} = a^{(N-1)/2} b_i^{(N-1)/2} \neq (a|N)(b_i|N) = (ab_i|N).$$

- Combining the above two results, we know

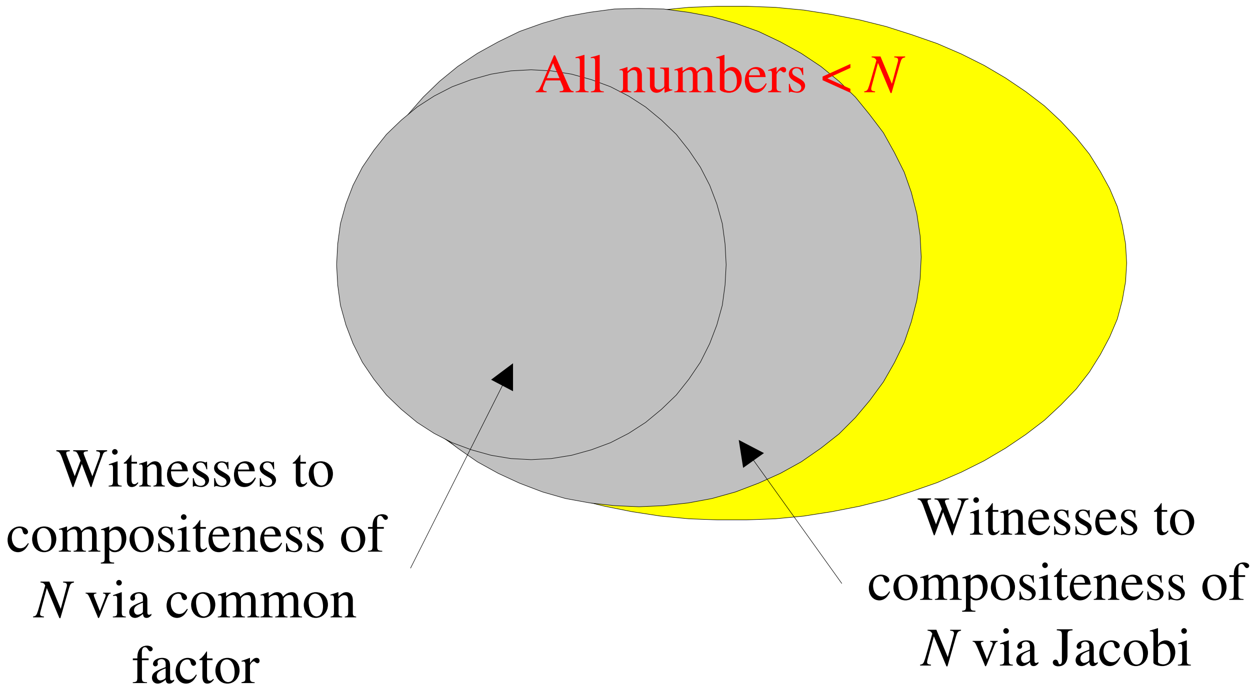
$$\frac{|\, B \,|}{\phi(N)} \leq 0.5.$$

```
if N is even but N ≠ 2 then
    return "N is composite";
else if N = 2 then
    return "N is a prime";
end if
Pick M ∈ {2, 3, ..., N − 1} randomly;
if gcd(M, N) > 1 then
    return "N is a composite";
else
    if (M|N) ≠ M^((N−1)/2) mod N then
        return "N is composite";
    else
        return "N is a prime";
    end if
end if
```

Analysis

- The algorithm certainly runs in polynomial time.
- There are no false positives (for COMPOSITENESS).
  - When the algorithm says the number is composite, it is always correct.
- The probability of a false negative is at most one half.
  - When the algorithm says the number is a prime, it may err.
  - If the input is composite, then the probability that the algorithm errs is one half.
- The error probability can be reduced but not eliminated.

The Improved Density Attack for COMPOSITENESS



Randomized Complexity Classes; RP

- Let N be a polynomial-time precise NTM that runs in time $p(n)$ and has 2 nondeterministic choices at each step.
- N is a **polynomial Monte Carlo Turing machine** for a language L if the following conditions hold:
  - If $x \in L$, then at least half of the $2^{p(n)}$ computation paths of N on x halt with "yes" where $n = |x|$.
  - If $x \notin L$, then all computation paths halt with "no."
- The class of all languages with polynomial Monte Carlo TMs is denoted **RP** (**randomized polynomial time**).[a]

[a] Adleman and Manders (1977).

Comments on RP

- Nondeterministic steps can be seen as fair coin flips.
- There are no false positive answers.
- The probability of false negatives, $1 - \epsilon$, is at most 0.5.
- But any constant between 0 and 1 can replace 0.5.
  - By repeating the algorithm $k = \lceil -\frac{1}{\log_2 1-\epsilon} \rceil$ times, the probability of false negatives becomes $(1 - \epsilon)^k \leq 0.5$.
- In fact, ϵ can be arbitrarily close to 0 as long as it is of the order $1/p(n)$ for some polynomial $p(n)$.
  - $-\frac{1}{\log_2 1-\epsilon} = O(\frac{1}{\epsilon}) = O(p(n))$.

Where RP Fits

- $P \subseteq RP \subseteq NP$.
  - A deterministic TM is like a Monte Carlo TM except that all the coin flips are ignored.
  - A Monte Carlo TM is an NTM with extra demands on the number of accepting paths.
- COMPOSITENESS $\in$ RP; PRIMES $\in$ coRP; PRIMES $\in$ RP.[a]
  - In fact, PRIMES $\in$ P.[b]
- $RP \cup coRP$ is another "plausible" notion of efficient computation.

---

[a] Adleman and Huang (1987).

[b] Agrawal, Kayal, and Saxena (2002).

ZPP[a] (Zero Probabilistic Polynomial)

- The class **ZPP** is defined as $\text{RP} \cap \text{coRP}$.
- A language in ZPP has *two* Monte Carlo algorithms, one with no false positives and the other with no false negatives.
- If we repeatedly run both Monte Carlo algorithms, *eventually* one definite answer will come (unlike RP).
  - A *positive* answer from the one without false positives.
  - A *negative* answer from the one without false negatives.

[a]Gill (1977).

The ZPP Algorithm (**Las Vegas**)

{Suppose $L \in \text{ZPP}$.}
{N_1 has no false positives, and N_2 has no false negatives.}
while `true` **do**
    if $N_1(x) =$ "yes" **then**
        return "yes";
    end if
    if $N_2(x) =$ "no" **then**
        return "no";
    end if
end while

ZPP (concluded)

- The *expected* running time for the correct answer to emerge is polynomial.
  - The probability that a run of the 2 algorithms does not generate a definite answer is 0.5.
  - Let $p(n)$ be the running time of each run.
  - The expected running time for a definite answer is

$$\sum_{i=1}^{\infty} 0.5^i i p(n) = 2p(n).$$

- Essentially, ZPP is the class of problems that can be solved without errors in expected polynomial time.

Et Tu, RP?

```
{Suppose L ∈ RP.}
{N decides L without false positives.}
while true do
    if N(x) = "yes" then
        return "yes";
    end if
    {But what to do here?}
end while
```

- You eventually get a "yes" if $x \in L$.
- But how to get a "no" when $x \notin L$?
- You have to sacrifice either correctness or bounded running time.

Large Deviations

- Suppose you have a *biased* coin.
- One side has probability $0.5 + \epsilon$ to appear and the other $0.5 - \epsilon$, for some $0 < \epsilon < 0.5$.
- But you do not know which is which.
- How to decide which side is the more likely—with high confidence?
- Answer: Flip the coin many times and pick the side that appeared the most times.
- Question: Can you quantify the confidence?

The Chernoff Bound[a]

Theorem 66 (Chernoff (1952)) *Suppose $x_1, x_2, \ldots, x_n$ are independent random variables taking the values 1 and 0 with probabilities p and $1-p$, respectively. Let $X = \sum_{i=1}^{n} x_i$. Then for all $0 \leq \theta \leq 1$,*

$$\text{prob}[\, X \geq (1+\theta)\, pn \,] \leq e^{-\theta^2 pn/3}.$$

- The probability that the deviate of a **binomial random variable** from its expected value $E[\, X \,] = E[\, \sum_{i=1}^{n} x_i \,] = pn$ decreases exponentially with the deviation.
- The Chernoff bound is asymptotically optimal.

---

[a]Herman Chernoff (1923–).

The Proof

- Let t be any positive real number.
- Then

$$\text{prob}[\, X \geq (1+\theta)\, pn \,] = \text{prob}[\, e^{tX} \geq e^{t(1+\theta)\, pn} \,].$$

- Markov's inequality (p. 405) generalized to real-valued random variables says that

$$\text{prob}\left[\, e^{tX} \geq kE[\, e^{tX} \,] \,\right] \leq 1/k.$$

- With $k = e^{t(1+\theta)\, pn} / E[\, e^{tX} \,]$, we have

$$\text{prob}[\, X \geq (1+\theta)\, pn \,] \leq e^{-t(1+\theta)\, pn} E[\, e^{tX} \,].$$

The Proof (continued)

- Because $X = \sum_{i=1}^{n} x_i$ and x_i's are independent,

$$E[\, e^{tX}\,] = (E[\, e^{tx_1}\,])^n = [\, 1 + p(e^t - 1)\,]^n.$$

- Substituting, we obtain

$$\begin{aligned} \text{prob}[\, X \geq (1+\theta)\, pn\,] &\leq e^{-t(1+\theta)\, pn} [\, 1 + p(e^t - 1)\,]^n \\ &\leq e^{-t(1+\theta)\, pn} e^{pn(e^t - 1)} \end{aligned}$$

as $(1+a)^n \leq e^{an}$ for all $a > 0$.

The Proof (concluded)

- With the choice of $t = \ln(1+\theta)$, the above becomes

$$\text{prob}[\, X \geq (1+\theta)\, pn \,] \leq e^{pn[\, \theta - (1+\theta) \ln(1+\theta) \,]}.$$

- The exponent expands to $-\frac{\theta^2}{2} + \frac{\theta^3}{6} - \frac{\theta^4}{12} + \cdots$ for $0 \leq \theta \leq 1$, which is less than

$$-\frac{\theta^2}{2} + \frac{\theta^3}{6} \leq \theta^2 \left(-\frac{1}{2} + \frac{\theta}{6} \right) \leq \theta^2 \left(-\frac{1}{2} + \frac{1}{6} \right) = -\frac{\theta^2}{3}.$$

Power of the Majority Rule

From prob$[\, X \le (1-\theta)\, pn \,] \le e^{-\frac{\theta^2}{2} pn}$ (prove it):

Corollary 67 *If* $p = (1/2) + \epsilon$ *for some* $0 \le \epsilon \le 1/2$, *then*

$$\text{prob}\left[\sum_{i=1}^{n} x_i \le n/2 \right] \le e^{-\epsilon^2 n/2}.$$

- The textbook's corollary to Lemma 11.9 seems incorrect.
- Our original problem (p. 462) hence demands $\approx 1.4k/\epsilon^2$ independent coin flips to guarantee making an error with probability at most 2^{-k} with the majority rule.

BPP[a] (Bounded Probabilistic Polynomial)

- The class **BPP** contains all languages for which there is a precise polynomial-time NTM N such that:
  - If $x \in L$, then at least $3/4$ of the computation paths of N on x lead to "yes."
  - If $x \notin L$, then at least $3/4$ of the computation paths of N on x lead to "no."
- N accepts or rejects by a *clear* majority.

---

[a] Gill (1977).

Magic 3/4?

- The number 3/4 bounds the probability of a right answer away from 1/2.
- Any constant *strictly* between 1/2 and 1 can be used without affecting the class BPP.
- In fact, 0.5 plus any inverse polynomial between 1/2 and 1,

$$0.5 + \frac{1}{p(n)},$$

can be used.

The Majority Vote Algorithm

Suppose L is decided by N by majority $(1/2) + \epsilon$.

for $i = 1, 2, \ldots, 2k + 1$ **do**
    Run N on input x;
end for
if "yes" is the majority answer **then**
    "yes";
else
    "no";
end if

Analysis

- The running time remains polynomial, being $2k + 1$ times N's running time.
- By Corollary 67 (p. 467), the probability of a false answer is at most $e^{-\epsilon^2 k}$.
- By taking $k = \lceil 2/\epsilon^2 \rceil$, the error probability is at most 1/4.
- As with the RP case, ϵ can be any inverse polynomial, because k remains polynomial in n.

Probability Amplification for BPP

- Let m be the number of random bits used by a BPP algorithm.
  - By definition, m is polynomial in n.
- With $k = \Theta(\log m)$ in the majority vote algorithm, we can lower the error probability to $\leq (3m)^{-1}$.

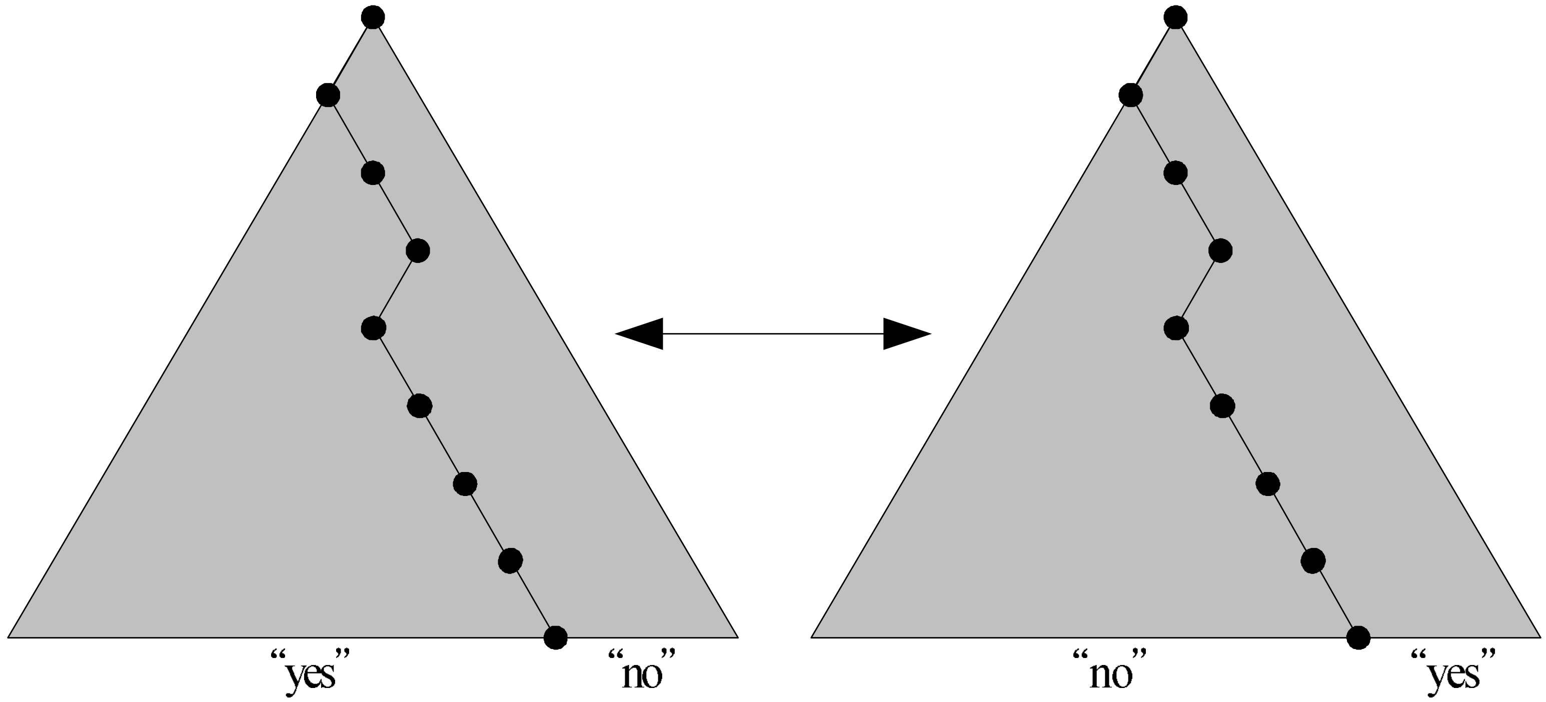
Aspects of BPP

- BPP is the most comprehensive yet plausible notion of efficient computation.
  - If a problem is in BPP, we take it to mean that the problem can be solved efficiently.
  - In this aspect, BPP has effectively replaced P.
- $(\text{RP} \cup \text{coRP}) \subseteq (\text{NP} \cup \text{coNP})$.
- $(\text{RP} \cup \text{coRP}) \subseteq \text{BPP}$.
- Whether $\text{BPP} \subseteq (\text{NP} \cup \text{coNP})$ is unknown.
- But it is unlikely that $\text{NP} \subseteq \text{BPP}$ (p. 487).

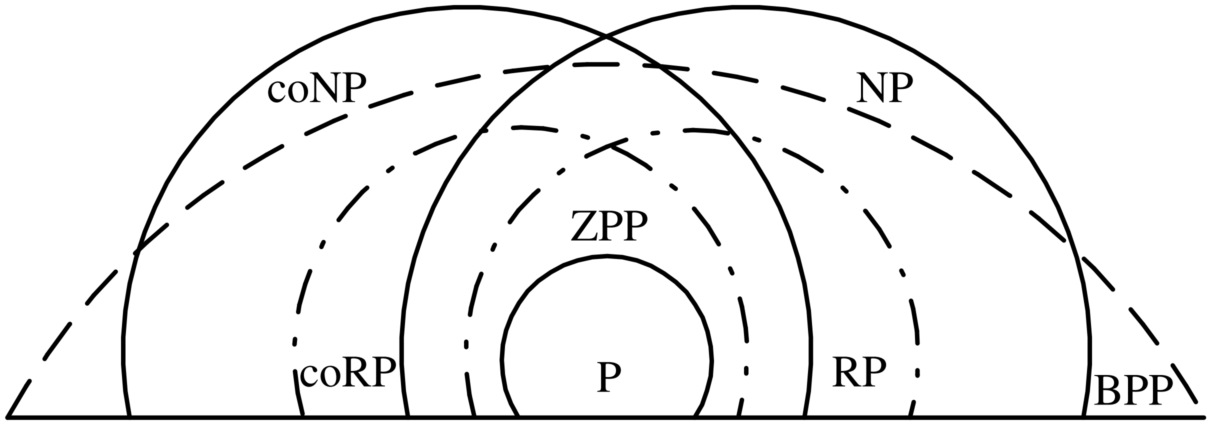
coBPP

- The definition of BPP is symmetric: acceptance by clear majority and rejection by clear majority.
- An algorithm for $L \in$ BPP becomes one for $\bar{L}$ by reversing the answer.
- So $\bar{L} \in$ BPP and BPP $\subseteq$ coBPP.
- Similarly coBPP $\subseteq$ BPP.
- Hence BPP $=$ coBPP.
- This approach does not work for RP.
- It did not work for NP either.

BPP and coBPP



“The Good, the Bad, and the Ugly”



Circuit Complexity

- Circuit complexity is based on boolean circuits instead of Turing machines.
- A boolean circuit with n inputs computes a boolean function of n variables.
- By identify `true` with 1 and `false` with 0, a boolean circuit with n inputs accepts certain strings in $\{0, 1\}^n$.
- To relate circuits with arbitrary languages, we need one circuit for each possible input length n.

Formal Definitions

- The **size** of a circuit is the number of *gates* in it.
- A **family of circuits** is an infinite sequence $\mathcal{C} = (C_0, C_1, \ldots)$ of boolean circuits, where C_n has n boolean inputs.
- $L \subseteq \{0,1\}^*$ has **polynomial circuits** if there is a family of circuits $\mathcal{C}$ such that:
  - The size of C_n is at most $p(n)$ for some fixed polynomial p.
  - For input $x \in \{0,1\}^*$, $C_{|\,x\,|}$ outputs 1 if and only if $x \in L$.
    - C_n accepts $L \cap \{0,1\}^n$.

Exponential Circuits Contain All Languages

- Theorem 14 (p. 153) implies that there are languages that cannot be solved by circuits of size $2^n/(2n)$.
- But exponential circuits can solve all problems.

Proposition 68 *All decision problems (decidable or otherwise) can be solved by a circuit of size* 2^{n+2}.

- We will show that for any language $L \subseteq \{0,1\}^*$, $L \cap \{0,1\}^n$ can be decided by a circuit of size 2^{n+2}.

The Proof (concluded)

- Define boolean function $f : \{0,1\}^n \to \{0,1\}$, where

$$f(x_1 x_2 \cdots x_n) = \begin{cases} 1 & x_1 x_2 \cdots x_n \in L, \\ 0 & x_1 x_2 \cdots x_n \notin L. \end{cases}$$

- $f(x_1 x_2 \cdots x_n) = (x_1 \land f(1 x_2 \cdots x_n)) \lor (\neg x_1 \land f(0 x_2 \cdots x_n))$.
- The circuit size $s(n)$ for $f(x_1 x_2 \cdots x_n)$ hence satisfies

$$s(n) = 4 + 2s(n-1)$$

  with $s(1) = 1$.
- Solve it to obtain $s(n) = 5 \times 2^{n-1} - 4 \leq 2^{n+2}$.

The Circuit Complexity of P

Proposition 69 *All languages in P have polynomial circuits.*

- Let $L \in \mathrm{P}$ be decided by a TM in time $p(n)$.
- By Corollary 27 (p. 239), there is a circuit with $O(p(n)^2)$ gates that accepts $L \cap \{0,1\}^n$.
- The size of the circuit depends only on L and the length of the input.
- The size of the circuit is polynomial in n.

Languages That Polynomial Circuits Accept

- Do polynomial circuits accept only languages in P?
- There are *undecidable* languages that have polynomial circuits.
  - Let $L \subseteq \{0,1\}^*$ be an undecidable language.
  - Let $U = \{1^n : \text{the binary expansion of } n \text{ is in } L\}$.[a]
  - U is also undecidable.
  - $U \cap \{1\}^n$ can be accepted by C_n that is trivially true if $1^n \in U$ and trivially false if $1^n \notin U$.
  - The family of circuits $(C_0, C_1, \ldots)$ is polynomial in size.

[a] Assume n's leading bit is always 1 without loss of generality.

A Patch

- Despite the simplicity of a circuit, the previous discussions imply the following:
  - Circuits are *not* a realistic model of computation.
  - Polynomial circuits are *not* a plausible notion of efficient computation.
- What gives?
- The *effective and efficient constructibility* of

$$C_0, C_1, \ldots .$$

Uniformity

- A family $(C_0, C_1, \ldots)$ of circuits is **uniform** if there is a $\log n$-space bounded TM which on input 1^n outputs C_n.
  - Circuits now cannot accept undecidable languages (why?).
  - The circuit family on p. 484 is not constructible by a *single* Turing machine (algorithm).
- A language has **uniformly polynomial circuits** if there is a *uniform* family of polynomial circuits that decide it.

Uniformly Polynomial Circuits and P

Theorem 70 *$L \in P$ if and only if L has uniformly polynomial circuits.*

- One direction was proved in Proposition 69 (p. 483).
- Now suppose L has uniformly polynomial circuits.
- Decide $x \in L$ in polynomial time as follows:
  - Let $n = |\, x \,|$.
  - Build C_n in $\log n$ space, hence polynomial time.
  - Evaluate the circuit with input x in polynomial time.
- Therefore $L \in$ P.

Relation to P vs. NP

- Theorem 70 implies that P $\neq$ NP if and only if NP-complete problems have no *uniformly* polynomial circuits.
- A stronger conjecture: NP-complete problems have no polynomial circuits, *uniformly or not*.
- The above is currently the preferred approach to proving the P $\neq$ NP conjecture—without success so far.
  - Theorem 14 (p. 153) states that there are boolean functions requiring $2^n/(2n)$ gates to compute.
  - In fact, almost all boolean functions do.

BPP's Circuit Complexity

Theorem 71 (Adleman (1978)) *All languages in BPP have polynomial circuits.*

- Our proof will be *nonconstructive* in that only the existence of the desired circuits is shown.
  - Something exists if its probability of existence is nonzero.
- How to efficiently generate circuit C_n given 1^n is not known.
- If the construction of C_n is efficient, then $\mathrm{P} = \mathrm{BPP}$, an unlikely result.

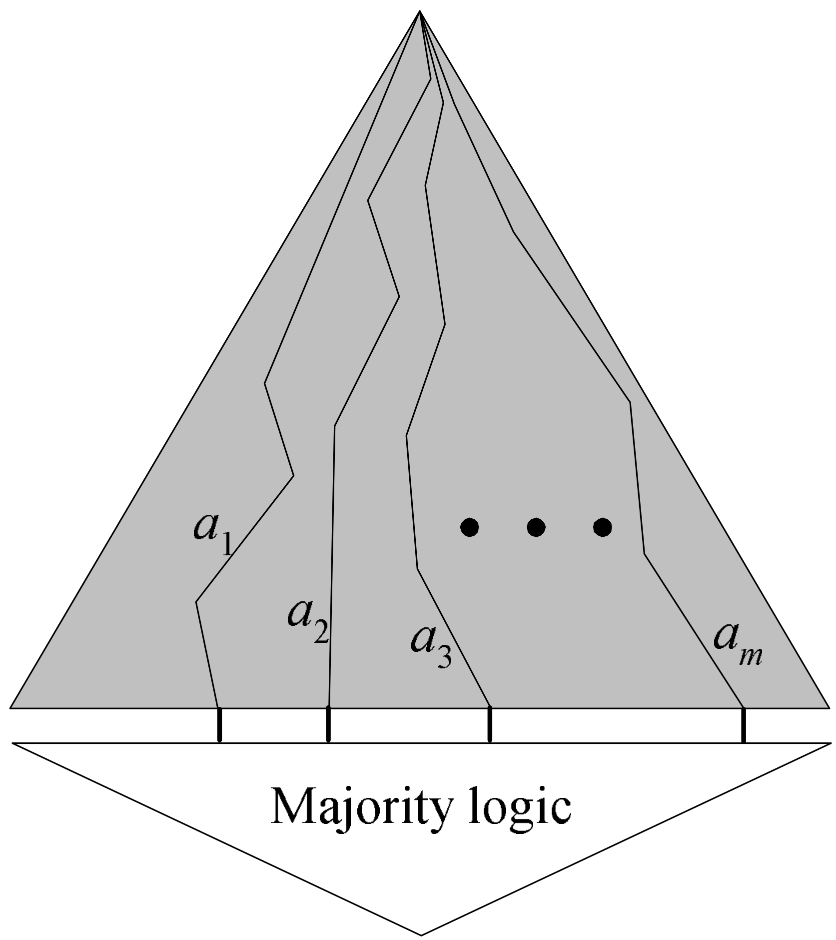
The Proof

- Let $L \in \text{BPP}$ be decided by a precise NTM N by clear majority.
- We shall prove that L has polynomial circuits $C_0, C_1, \ldots$.
- Suppose N runs in time $p(n)$, where $p(n)$ is a polynomial.
- Let $A_n = \{a_1, a_2, \ldots, a_m\}$, where $a_i \in \{0,1\}^{p(n)}$.
- Let $m = 12(n+1)$.
- Each $a_i \in A_n$ represents a sequence of nondeterministic choices—i.e., a computation path—for N.

The Proof (continued)

- Let x be an input with $|x| = n$.
- Circuit C_n simulates N on x with each sequence of choices in A_n and then takes the majority of the m outcomes.
- Because N with a_i is a polynomial-time TM, it can be simulated by polynomial circuits of size $O(p(n)^2)$.
  - See the proof of Proposition 69 (p. 483).
- The size of C_n is therefore $O(mp(n)^2) = O(np(n)^2)$, a polynomial.
- We next prove the existence of A_n making C_n correct.

The Circuit



The Proof (continued)

- Call a_i **bad** if it leads N to a false positive or a false negative answer.
- Select A_n *uniformly randomly*.
- For each $x \in \{0,1\}^n$, 1/4 of the computations of N are erroneous.
- Because the sequences in A_n are chosen randomly and independently, the expected number of bad a_i's is $m/4$.
- By the Chernoff bound (p. 464), the probability that the number of bad a_i's is $m/2$ or more is at most

$$e^{-m/12} < 2^{-(n+1)}.$$

The Proof (concluded)

- The error probability is $< 2^{-(n+1)}$ for each $x \in \{0,1\}^n$.
- The probability that there is an x such that A_n results in an incorrect answer is $< 2^n 2^{-(n+1)} = 2^{-1}$.
  - $\text{prob}[\, A \cup B \cup \cdots] \leq \text{prob}[\, A\,] + \text{prob}[\, B\,] + \cdots$.
- So with probability one half, a random A_n produces a correct C_n for *all* inputs of length n.
- Because this probability exceeds 0, an A_n that makes majority vote work for all inputs of length n exists.
- Hence a correct C_n exists.

Cryptography

Whoever wishes to keep a secret
must hide the fact that he possesses one.
— Johann Wolfgang von Goethe (1749–1832)

Cryptography

- **Alice** (A) wants to send a message to **Bob** (B) over a channel monitored by **Eve** (eavesdropper).
- The protocol should be such that the message is known only to Alice and Bob.
- The art and science of keeping messages secure is **cryptography**.

$$\text{Alice} \xrightarrow{\text{Eve}} \text{Bob}$$

Encryption and Decryption

- Alice and Bob agree on two algorithms E and D—the **encryption** and the **decryption algorithms**.
- Both E and D are known to the public in the analysis.
- Alice runs E and wants to send a message x to Bob.
- Bob operates D.
- Privacy is assured in terms of two numbers e, d, the **encryption** and **decryption keys**.
- Alice sends $y = E(e, x)$ to Bob, who then performs $D(d, y) = x$ to recover x.
- x is called **plaintext**, and y is called **ciphertext**.[a]

[a] Both "zero" and "cipher" come from the same Arab word.

Some Requirements

- D should be an inverse of E given e and d.
- D and E must both run in (probabilistic) polynomial time.
- Eve should not be able to recover x from y without knowing d.
  - As D is public, d must be kept secret.
  - e may or may not be a secret.

Degrees of Security

- **Perfect secrecy**: After a ciphertext is intercepted by the enemy, the a posteriori probabilities of the plaintext that this ciphertext represents are identical to the a priori probabilities of the same plaintext before the interception.
- Such systems are said to be **informationally secure**.
- A system is **computationally secure** if breaking it is theoretically possible but computationally infeasible.

Conditions for Perfect Secrecy[a]

- Consider a cryptosystem where:
  - The space of ciphertext is as large as that of keys.
  - Every plaintext has a nonzero probability of being used.
- It is perfectly secure if and only if the following hold.
  - A key is chosen with uniform distribution.
  - For each plaintext x and ciphertext y, there exists a unique key e such that $E(e, x) = y$.

[a] Shannon (1949).

The One-Time Pad[a]

1: Alice generates a random string r as long as x;

2: Alice sends r to Bob over a secret channel;

3: Alice sends $r \oplus x$ to Bob over a public channel;

4: Bob receives y;

5: Bob recovers $x := y \oplus r$;

---

[a] Mauborgne and Vernam (1917), Shannon (1949); allegedly used for the hotline between Russia and U.S.

Analysis

- The one-time pad uses $e = d = r$.
- This is said to be a **private-key cryptosystem**.
- Knowing x and knowing r are equivalent.
- Because r is random and private, the one-time pad achieves perfect secrecy (see also p. 501).
- The random bit string must be new for each round of communication.
  - **Cryptographically strong pseudorandom generators** require exchanging only the seed once.
- The assumption of a private channel is problematic.

Public-Key Cryptography[a]

- Suppose only d is private to Bob, whereas e is public knowledge.
- Bob generates the (e, d) pair and publishes e.
- Anybody like Alice can send $E(e, x)$ to Bob.
- Knowing d, Bob can recover x by $D(d, E(e, x)) = x$.
- The assumptions are complexity-theoretic.
  - It is computationally difficult to compute d from e.
  - It is computationally difficult to compute x from y without knowing d.

[a]Diffie and Hellman (1976).

Complexity Issues

- Given y and x, it is easy to verify whether $E(e, x) = y$.
- Hence one can always guess an x and verify.
- Cracking a public-key cryptosystem is thus in NP.
- A necessary condition for the existence of secure public-key cryptosystems is P $\neq$ NP.
- But more is needed than P $\neq$ NP.
- It is not sufficient that D is hard to compute in the worst case.
- It should be hard in "most" or "average" cases.

One-Way Functions

A function f is a **one-way function** if the following hold.[a]

1. f is one-to-one.
2. For all $x \in \Sigma^*$, $|\,x\,|^{1/k} \leq |f(x)| \leq |\,x\,|^k$ for some $k > 0$.
   - f is said to be **honest**.
3. f can be computed in polynomial time.
4. f^{-1} cannot be computed in polynomial time.
   - Exhaustive search works, but it is too slow.

---

[a] Diffie and Hellman (1976); Boppana and Lagarias (1986); Grollmann and Selman (1988); Ko (1985); Ko, Long, and Du (1986); Watanabe (1985); Young (1983).

Existence of One-Way Functions

- Even if P $\neq$ NP, there is no guarantee that one-way functions exist.
- No functions have been proved to be one-way.
- Is breaking a glass a one-way function?

UP[a]

- An NTM that has at most one accepting computation for any input is called an **unambiguous Turing machine** (**UTM**).
- UP denotes the set of languages accepted by UTMs in polynomial time.
- Obviously, $\mathrm{P} \subseteq \mathrm{UP} \subseteq \mathrm{NP}$.

[a] Valiant (1976).

SAT and UP

- SAT is not expected to be in UP (so UP $\neq$ NP).
  - Suppose SAT $\in$ UP.
  - Then there is an NTM M that has a single accepting computation path for all satisfiable boolean expressions.
  - But M runs in polynomial time.
  - Hence M does not try all truth assignments for satisfiable boolean expressions.
  - At present, this seems implausible.

UP and One-Way Functions[a]

Theorem 72 *One-way functions exist if and only if $P \neq UP$.*

- Suppose there exists a one-way function f.
- Define language
  $L_f \equiv \{ (x, y) : \exists z \text{ such that } f(z) = y \text{ and } z \leq x \}$.
  - Relation "$\leq$" orders strings of $\{ 0, 1 \}^*$ first by length and then lexicographically.
  - So $\epsilon < 0 < 1 < 00 < 01 < 10 < 11 < \cdots$.

[a] Ko (1985); Grollmann and Selman (1988).

The Proof (continued)

- $L_f \in \text{UP}$.
  - There is an UTM M that accepts L_f.
    - M on input (x, y) nondeterministically guesses a string z of length at most $|y|^k$.
    - M tests if $y = f(z)$.
    - If the answer is "yes" (this happens at most once because f is one-to-one) and $z \leq x$, M accepts.

The Proof (continued)

- $L_f \notin$ P.
  - Suppose there is a polynomial-time algorithm for L_f.
  - Then $f(x) = y$ can be inverted.
    - Given y, ask $(1^{|\,y\,|^k}, y) \in L_f$.
    - If the answer is "no," we know x does not exist as any such x must satisfy $|\,x\,| \leq |\,y\,|^k$.
    - Otherwise, ask $(1^{|\,y\,|^k - 1}, y) \in L_f, (1^{|\,y\,|^k - 2}, y) \in L_f, \ldots$ until we got a "no" for $(1^{\ell - 1}, y) \in L_f$.
    - This means $|\,x\,| = \ell$.
  - The procedure makes $O(|\,y\,|^k)$ calls to L_f.

The Proof (continued)

- (continued)
  - ∗ Now conduct a binary search to find each bit of x as follows.
    ∗ If $(01^{\ell-1}, y) \in L_f$, then $x = 0 \cdots$ and we recur by asking "$(001^{\ell-2}, y) \in L_f$?"
    ∗ If $(01^{\ell-1}, y) \notin L_f$, then $x = 1 \cdots$ and we recur by asking $(101^{\ell-2}, y) \in L_f$?"
  - The procedure makes $O(|\, y \,|^k)$ calls to L_f.
- $\mathrm{P} \neq \mathrm{UP}$ because $L_f \in \mathrm{UP} - \mathrm{P}$.

The Proof (continued)

- Now suppose $\text{P} \neq \text{UP}$ with $L \in \text{UP} - \text{P}$.
- Let L be accepted by an UTM M.
- $\text{comp}_M(y)$ denotes an accepting computation of $M(y)$.
- Define

$$f_M(x) = \begin{cases} 1y & \text{if } x = \text{comp}_M(y), \\ 0x & \text{otherwise.} \end{cases}$$

- f_M is well-defined as y is part of $\text{comp}_M(y)$ (recall p. 238) and there is at most one accepting computation for y.
- So f_M is a total function.

The Proof (concluded)

- f_M is one-way.
  - The lengths of argument and results are polynomially related as M has polynomially long computations.
  - f_M is one-to-one because $f(x) = f(x')$ means that $x = x'$ by the use of the flag and unambiguity of M.
  - f_M can be inverted on $1y$ if and only if M accepts y (i.e., if $y \in L$).
  - Were we able to invert f_M in polynomial time, then we would be able to decide L in polynomial time.

Complexity Issues

- For a language in UP, there is either 0 or 1 accepting path.
- So similar to RP, there are not likely to be UP-complete problems.
- Relating a cryptosystem with an NP-complete problem has been argued before to be not useful (p. 505).
- Theorem 72 (p. 510) shows that the relevant question is the P = UP question.
- There are stronger notions of one-way functions.

Candidates of One-Way Functions

- Modular exponentiation $f(x) = g^x \bmod p$, where g is a primitive root of p.
  - **Discrete logarithm** is hard.[a]
- The RSA[b] function $f(x) = x^e \bmod pq$ for an odd e relatively prime to $\phi(pq)$.
  - Breaking the RSA function is hard.
- Modular squaring $f(x) = x^2 \bmod pq$.
  - Determining if a number with a Jacobi symbol 1 is a quadratic residue is hard—the **quadratic residuacity assumption** (**QRA**).

---

[a] But it is in NP in some sense; Grollmann and Selman (1988).
[b] Rivest, Shamir, and Adleman (1978).

The RSA Function

- Let p, q be two distinct primes.
- The RSA function is $x^e \bmod pq$ for an odd e relatively prime to $\phi(pq)$.
  - By Lemma 49 (p. 359),
$$\phi(pq) = pq\left(1-\frac{1}{p}\right)\left(1-\frac{1}{q}\right) = pq - p - q + 1.$$
- As $\gcd(e, \phi(pq)) = 1$, there is a d such that
$$ed \equiv 1 \bmod \phi(pq),$$
which can be found by the Euclidean algorithm.

A Public-Key Cryptosystem Based on RSA

- Bob generates p and q.
- Bob publishes pq and the encryption key e, a number relatively prime to $\phi(pq)$.
  - The encryption function is $y = x^e \bmod pq$.
- Knowing $\phi(pq)$, Bob calculates d such that $ed = 1 + k\phi(pq)$ for some $k \in \mathbb{Z}$.
  - The decryption function is $y^d \bmod pq$.
  - It works because $y^d = x^{ed} = x^{1+k\phi(pq)} = x \bmod pq$ by the Fermat-Euler theorem when $\gcd(x, pq) = 1$ (p. 367).

The "Security" of the RSA Function

- Factoring pq or calculating d from (e, pq) seems hard.
  - See also p. 363.
- Breaking the last bit of RSA is as hard as breaking the RSA.[a]
- Recommended RSA key sizes:
  - 1024 bits up to 2010.
  - 2048 bits up to 2030.
  - 3072 bits up to 2031 and beyond.

---

[a] Alexi, Chor, Goldreich, and Schnorr (1988).

The "Security" of the RSA Function (concluded)

- Recall that problem A is "harder than" problem B if solving A results in solving B.
  - Factorization is "harder than" breaking the RSA.
  - Calculating Euler's phi function is "harder than" breaking the RSA.
  - Factorization is "harder than" calculating Euler's phi function (see Lemma 49 on p. 359).
- Factorization cannot be NP-hard unless NP = coNP.[a]
- So breaking the RSA is unlikely to imply P = NP.

[a] Brassard (1979).

The Secret-Key Agreement Problem

- Exchanging messages securely using a private-key cryptosystem requires Alice and Bob possessing the same key (p. 503).
- How can they agree on the same secret key when the channel is insecure?
- This is called the **secret-key agreement problem**.
- It was solved by Diffie and Hellman (1976) using one-way functions.

The Diffie-Hellman Secret-Key Agreement Protocol

1: Alice and Bob agree on a large prime p and a primitive root g of p; $\{p$ and g are public.$\}$

2: Alice chooses a large number a at random;

3: Alice computes $\alpha = g^a \bmod p$;

4: Bob chooses a large number b at random;

5: Bob computes $\beta = g^b \bmod p$;

6: Alice sends α to Bob, and Bob sends β to Alice;

7: Alice computes her key $\beta^a \bmod p$;

8: Bob computes his key $\alpha^b \bmod p$;

Analysis

- The keys computed by Alice and Bob are identical:

$$\beta^a = g^{ba} = g^{ab} = \alpha^b \bmod p.$$

- To compute the common key from p, g, α, β is known as the **Diffie-Hellman problem**.
- It is conjectured to be hard.
- If discrete logarithm is easy, then one can solve the Diffie-Hellman problem.
  - Because a and b can then be obtained by Eve.
- But the other direction is still open.

A Parallel History

- Diffie and Hellman's solution to the secret-key agreement problem led to public-key cryptography.
- At around the same time (or earlier) in Britain, the RSA public-key cryptosystem was invented first before the Diffie-Hellman secret-key agreement scheme was.
  - Ellis, Cocks, and Williamson of the Communications Electronics Security Group of the British Government Communications Head Quarters (GCHQ).

Digital Signatures[a]

- Alice wants to send Bob a *signed* document x.
- The signature must unmistakably identifies the sender.
- Both Alice and Bob have public and private keys

$$e_{\text{Alice}}, e_{\text{Bob}}, d_{\text{Alice}}, d_{\text{Bob}}.$$

- Assume the cryptosystem satisfies the commutative property

$$E(e, D(d, x)) = D(d, E(e, x)). \tag{7}$$

  - As $(x^d)^e = (x^e)^d$, the RSA system satisfies it.
  - Every cryptosystem guarantees $D(d, E(e, x)) = x$.

[a] Diffie and Hellman (1976).

Digital Signatures Based on Public-Key Systems

- Alice signs x as
$$(x, D(d_{\text{Alice}}, x)).$$
- Bob receives (x, y) and verifies the signature by checking
$$E(e_{\text{Alice}}, y) = E(e_{\text{Alice}}, D(d_{\text{Alice}}, x)) = x$$
based on Eq. (7).
- The claim of authenticity is founded on the difficulty of inverting E_{Alice} without knowing the key d_{Alice}.
- Warning: If Alice signs anything presented to her, she might inadvertently decrypt a ciphertext of hers.

Mental Poker[a]

- Suppose Alice and Bob have agreed on 3 n-bit numbers $a < b < c$, the cards.
- They want to randomly choose one card each, so that:
  - Their cards are different.
  - All 6 pairs of distinct cards are equiprobable.
  - Alice's (Bob's) card is known to Alice (Bob) but not to Bob (Alice), until Alice (Bob) announces it.
  - The person with the highest card wins the game.
  - The outcome is indisputable.
- Assume Alice and Bob will not deviate from the protocol.

[a]Shamir, Rivest, and Adleman (1981).

The Setup

- Alice and Bob agree on a large prime p;
- Each has two *secret* keys $e_{\text{Alice}}, e_{\text{Bob}}, d_{\text{Alice}}, d_{\text{Bob}}$ such that $e_{\text{Alice}} d_{\text{Alice}} = e_{\text{Bob}} d_{\text{Bob}} = 1 \bmod (p-1)$;
  - This ensures that $(x^{e_{\text{Alice}}})^{d_{\text{Alice}}} = x \bmod p$ and $(x^{e_{\text{Bob}}})^{d_{\text{Bob}}} = x \bmod p$.
- The protocol lets Bob pick Alice's card and Alice pick Bob's card.
- Cryptographic techniques make it plausible that Alice's and Bob's choices are practically random, for lack of time to break the system.

The Protocol

1: Alice encrypts the cards

$$a^{e_{\text{Alice}}} \bmod p, b^{e_{\text{Alice}}} \bmod p, c^{e_{\text{Alice}}} \bmod p$$

and sends them in random order to Bob;

1: Bob picks one of the messages $x^{e_{\text{Alice}}}$ to send to Alice;

2: Alice decodes it $(x^{e_{\text{Alice}}})^{d_{\text{Alice}}} = x \bmod p$ for her card;

3: Bob encrypts the two remaining cards $(x^{e_{\text{Alice}}})^{e_{\text{Bob}}} \bmod p, (y^{e_{\text{Alice}}})^{e_{\text{Bob}}} \bmod p$ and sends them in random order to Alice;

4: Alice picks one of the messages, $(z^{e_{\text{Alice}}})^{e_{\text{Bob}}}$, encrypts it $((z^{e_{\text{Alice}}})^{e_{\text{Bob}}})^{d_{\text{Alice}}} \bmod p$, and sends it to Bob;

5: Bob decrypts the message $(((z^{e_{\text{Alice}}})^{e_{\text{Bob}}})^{d_{\text{Alice}}})^{d_{\text{Bob}}} = z \bmod p$ for his card;

Probabilistic Encryption[a]

- The ability to forge signatures on even a vanishingly small fraction of strings of some length is a security weakness if those strings were the probable ones!
- What is required is a scheme that does not "leak" *partial* information.
- The first solution to the problems of skewed distribution and partial information was based on the QRA.

[a]Goldwasser and Micali (1982).

The Setup

- Bob publishes $n = pq$, a product of two distinct primes, and a quadratic nonresidue y with Jacobi symbol 1.
- Bob keeps secret the factorization of n.
- To send bit string $b_1 b_2 \cdots b_k$ to Bob, Alice encrypts the bits by choosing a random quadratic residue modulo n if b_i is 1 and a random quadratic nonresidue with Jacobi symbol 1 otherwise.
- A sequence of residues and nonresidues are sent.
- Knowing the factorization of n, Bob can efficiently test quadratic residuacity and thus read the message.

A Useful Lemma

Lemma 73 *Let $n = pq$ be a product of two distinct primes. Then a number $y \in Z_n^*$ is a quadratic residue modulo n if and only if $(y \,|\, p) = (y \,|\, q) = 1$.*

- The "only if" part:
  - Let x be a solution to $x^2 = y \bmod pq$.
  - Then $x^2 = y \bmod p$ and $x^2 = y \bmod q$ also hold.
  - Hence y is a quadratic modulo p and a quadratic residue modulo q.

The Proof (concluded)

- The "if" part:
  - Let $a_1^2 = y \bmod p$ and $a_2^2 = y \bmod q$.
  - Solve

$$\begin{aligned} x &= a_1 \bmod p, \\ x &= a_2 \bmod q, \end{aligned}$$

    for x with the Chinese remainder theorem.
  - As $x^2 = y \bmod p$, $x^2 = y \bmod q$, and $\gcd(p, q) = 1$, we must have $x^2 = y \bmod pq$.

The Protocol for Alice

for $i = 1, 2, \ldots, k$ **do**
    Pick $r \in Z_n^*$ randomly;
    if $b_i = 1$ **then**
        Send $r^2 \bmod n$; {Jacobi symbol is 1.}
    else
        Send $r^2 y \bmod n$; {Jacobi symbol is still 1.}
    end if
end for

The Protocol for Bob

for $i = 1, 2, \ldots, k$ **do**
  Receive r;
  if $(r \,|\, p) = 1$ and $(r \,|\, q) = 1$ **then**
    $b_i := 1$;
  else
    $b_i := 0$;
  end if
end for

Semantic Security

- This encryption scheme is probabilistic.
- There are a large number of different encryptions of a given message.
- One is chosen at random by the sender to represent the message.
- This scheme is both polynomially secure and **semantically secure**.

What Is a Proof?

- A proof convinces a party of a certain claim.
  - "Is $x^n + y^n \neq z^n$ for all $x, y, z \in \mathbb{Z}^+$ and $n > 2$?"
  - "Is graph G Hamiltonian?"
  - "Is $x^p = x \bmod p$ for prime p and $p \not| x$?"
- In mathematics, a proof is a fixed sequence of theorems.
  - Think of a written examination.
- We will extend a proof to cover a proof *process* by which the validity of the assertion is established.
  - Think of a job interview or an oral examination.

Prover and Verifier

- There are two parties to a proof.
  - The **prover** (**Peggy**).
  - The **verifier** (**Victor**).
- Given an assertion, the prover's goal is to convince the verifier of its validity (**completeness**).
- The verifier's objective is to accept only correct assertions (**soundness**).
- The verifier usually has an easier job than the prover.
- The setup is very much like the Turing test.[a]

[a] Turing (1950).

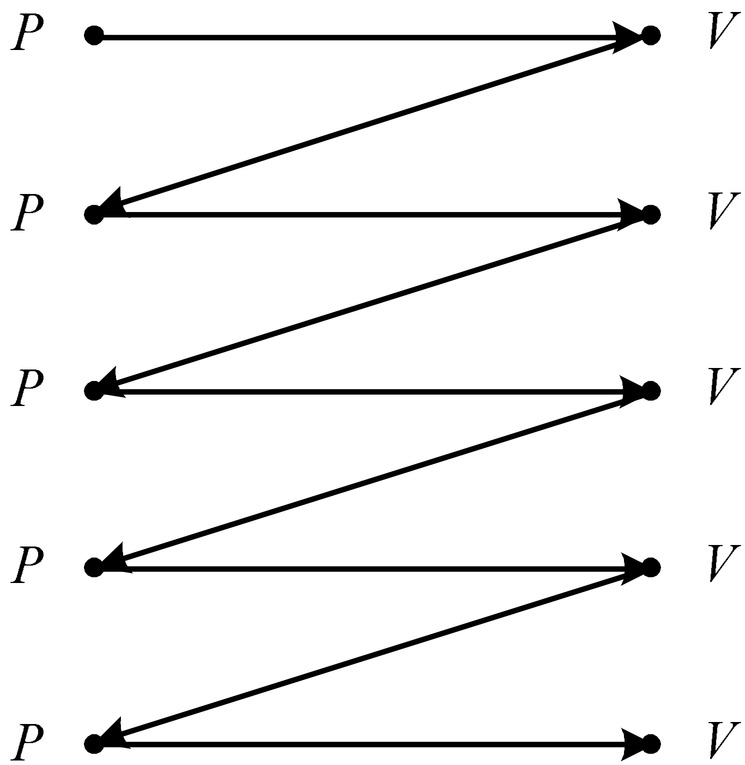
Interactive Proof Systems

- An **interactive proof** for a language L is a sequence of questions and answers between the two parties.
- At the end of the interaction, the verifier decides based on the knowledge he acquired in the proof process whether the claim is true or false.
- The verifier must be a probabilistic polynomial-time algorithm.
- The prover runs an exponential-time algorithm.
  - If the prover is not more powerful than the verifier, no interaction is needed.

Interactive Proof Systems (concluded)

- The system decides L if the following two conditions hold for any common input x.
  - If $x \in L$, then the probability that x is accepted by the verifier is at least $1 - 2^{-|x|}$.
  - If $x \notin L$, then the probability that x is accepted by the verifier with *any* prover replacing the original prover is at most $2^{-|x|}$.
- Neither the number of rounds nor the lengths of the messages can be more than a polynomial of $|x|$.

An Interactive Proof



IP[a]

- **IP** is the class of all languages decided by an interactive proof system.
- When $x \in L$, the completeness condition can be modified to require that the verifier accepts with certainty without affecting IP.[b]
- Similar things cannot be said of the soundness condition when $x \notin L$.
- Verifier's coin flips can be public.[c]

---

[a] Goldwasser, Micali, and Rackoff (1985).
[b] Goldreich, Mansour, and Sipser (1987).
[c] Goldwasser and Sipser (1989).

The Relations of IP with Other Classes

- NP $\subseteq$ IP.
  - IP becomes NP when the verifier is deterministic.
- BPP $\subseteq$ IP.
  - IP becomes BPP when the verifier ignores the prover's messages.
- IP actually coincides with PSPACE (see the textbook for a proof).[a]

[a] Shamir (1990).

Graph Isomorphism

- $V_1 = V_2 = \{1, 2, \ldots, n\}$.
- Graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are **isomorphic** if there exists a permutation π on $\{1, 2, \ldots, n\}$ so that $(u, v) \in E_1 \Leftrightarrow (\pi(u), \pi(v)) \in E_2$.
- The task is to answer if $G_1 \cong G_2$ (**isomorphic**).
- No known polynomial-time algorithms.
- The problem is in NP (hence IP).
- But it is not likely to be NP-complete.[a]

[a] Schöning (1987).

GRAPH NONISOMORPHISM

- $V_1 = V_2 = \{1, 2, \ldots, n\}$.
- Graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are **nonisomorphic** if there exist no permutations π on $\{1, 2, \ldots, n\}$ so that $(u, v) \in E_1 \Leftrightarrow (\pi(u), \pi(v)) \in E_2$.
- The task is to answer if $G_1 \ncong G_2$ (**nonisomorphic**).
- Again, no known polynomial-time algorithms.
  - It is in coNP, but how about NP or BPP?
  - It is not likely to be coNP-complete.
- Surprisingly, GRAPH NONISOMORPHISM $\in$ IP.[a]

[a] Goldreich, Micali, and Wigderson (1986).

A 2-Round Algorithm

Victor selects a random $i \in \{1, 2\}$;
Victor selects a random permutation π on $\{1, 2, \ldots, n\}$;
Victor applies π on graph G_i to obtain graph H;
Victor sends (G_1, H) to Peggy;
if $G_1 \cong H$ **then**
    Peggy sends $j = 1$ to Victor;
else
    Peggy sends $j = 2$ to Victor;
end if
if $j = i$ **then**
    Victor accepts;
else
    Victor rejects;
end if

Analysis

- Victor runs in probabilistic polynomial time.
- Suppose the two graphs are not isomorphic.
  - Peggy is able to tell which G_i is isomorphic to H.
  - So Victor always accepts.
- Suppose the two graphs are isomorphic.
  - No matter which i is picked by Victor, Peggy or any prover sees 2 identical graphs.
  - Peggy or any prover with exponential power has only probability one half of guessing i correctly.
  - So Victor erroneously accepts with probability $1/2$.
- Repeat the algorithm to obtain the desired probabilities.

Knowledge in Proofs

- Suppose I know a satisfying assignment to a satisfiable boolean expression.
- I can convince Alice of this by giving her the assignment.
- But then I give her more knowledge than necessary.
  - Alice can claim that she found the assignment!
  - Login authentication faces essentially the same issue.
  - See `www.wired.com/wired/archive/1.05/atm_pr.html` for a famous ATM fraud in the U.S.

Knowledge in Proofs (concluded)

- Digital signatures authenticate *documents* but not *individuals*.
- They hence do not solve the problem.
- Suppose I always give Alice random bits.
- Alice extracts no knowledge from me by any measure, but I prove nothing.
- Question 1: Can we design a protocol to convince Alice of (the knowledge of) a secret without revealing anything extra?
- Question 2: How to define this idea rigorously?

Zero Knowledge Proofs[a]

An interactive proof protocol (P, V) for language L has the **perfect zero-knowledge** property if:

- For every verifier V', there is an algorithm M with expected polynomial running time.
- M on any input $x \in L$ generates the same probability distribution as the one that can be observed on the communication channel of (P, V') on input x.

---

[a]Goldwasser, Micali, and Rackoff (1985).

Comments

- Zero knowledge is a property of the prover.
  - It is the robustness of the prover against attempts of the verifier to extract knowledge via interaction.
  - The verifier may deviate arbitrarily (but in polynomial time) from the predetermined program.
  - A verifier cannot use the transcript of the interaction to convince a third-party of the validity of the claim.
  - The proof is hence not transferable.

Comments (continued)

- Whatever a verifier can "learn" from the specified prover P via the communication channel could as well be computed from the verifier alone.
- The verifier does not learn anything except "$x \in L$."
- For all practical purposes "whatever" can be done after interacting with a zero-knowledge prover can be done by just believing that the claim is indeed valid.
- Zero-knowledge proofs yield no knowledge in the sense that they can be constructed by the verifier who believes the statement, and yet these proofs do convince him.

Comments (continued)

- The "paradox" is resolved by noting that it is not the transcript of the conversation that convinces the verifier.
- But the fact that this conversation was held "on line."
- There is no zero-knowledge requirement when $x \notin L$.
- *Computational* zero-knowledge proofs are based on complexity assumptions.
  - M only needs to generate a distribution that is computationally indistinguishable from the verifier's view of the interaction.

Comments (concluded)

- It is known that if one-way functions exist, then zero-knowledge proofs exist for every problem in NP.[a]
- The verifier can be restricted to the honest one (i.e., it follows the protocol).[b]
- The coins can be public.[c]

---

[a] Goldreich, Micali, and Wigderson (1986).
[b] Vadhan (2006).
[c] Vadhan (2006).

Are You Convinced?

- A newspaper commercial for hair-growing products for men.
  - A (for all practical purposes) bald man has a full head of hair after 3 months.
- A TV commercial for weight-loss products.
  - A (by any reasonable measure) overweight woman loses 10 kilograms in 10 weeks.

Quadratic Residuacity

- Let n be a product of two distinct primes.
- Assume extracting the square root of a quadratic residue modulo n is hard without knowing the factors.
- We next present a zero-knowledge proof for x being a quadratic residue.

Zero-Knowledge Proof of Quadratic Residuacity (continued)

1: **for** $m = 1, 2, \ldots, \log_2 n$ **do**
2: Peggy chooses a random $v \in Z_n^*$ and sends $y = v^2 \bmod n$ to Victor;
3: Victor chooses a random bit i and sends it to Peggy;
4: Peggy sends $z = u^i v \bmod n$, where u is a square root of x; $\{u^2 \equiv x \bmod n.\}$
5: Victor checks if $z^2 \equiv x^i y \bmod n$;
6: **end for**
7: Victor accepts x if Line 5 is confirmed every time;

Analysis

- Suppose x is a quadratic nonresidue.
  - Peggy can answer only one of the two possible challenges.
    - Reason: a is a quadratic residue if and only if xa is a quadratic nonresidue.
  - So Peggy will be caught in any given round with probability one half.

Analysis (continued)

- Suppose x is a quadratic residue.
  - Peggy can answer all challenges.
  - So Victor will accept x.
- How about the claim of zero knowledge?
- The transcript between Peggy and Victor when x is a quadratic residue can be generated without Peggy!
  - So interaction with Peggy is useless.
- Here is how.

Analysis (continued)

- Suppose x is a quadratic residue.[a]
- In each round of interaction with Peggy, the transcript is a triplet (y, i, z).
- We present an efficient Bob that generates (y, i, z) with the same probability *without* accessing Peggy.

[a]By definition, we do not need to consider the other case.

Analysis (concluded)

1: Bob chooses a random $z \in Z_n^*$;

2: Bob chooses a random bit i;

3: Bob calculates $y = z^2 x^{-i} \bmod n$;

4: Bob writes (y, i, z) into the transcript;

Comments

- Assume x is a quadratic residue.
- In both cases, for (y, i, z), y is a random quadratic residue, i is a random bit, and z is a random number.
- Bob cheats because (y, i, z) is *not* generated in the same order as in the original transcript.
  - Bob picks Victor's challenge first.
  - Bob then picks Peggy's answer.
  - Bob finally patches the transcript.

Comments (concluded)

- So it is not the transcript that convinces Victor, but that conversation with Peggy is held "on line."
- The same holds even if the transcript was generated by a cheating Victor's interaction with (honest) Peggy.
- But we skip the details.

Zero-Knowledge Proof of 3 Colorability[a]

for $i = 1, 2, \ldots, |\,E\,|^2$ **do**

  Peggy chooses a random permutation π of the 3-coloring ϕ;

  Peggy samples an encryption scheme randomly and sends $\pi(\phi(1)), \pi(\phi(2)), \ldots, \pi(\phi(|V|))$ encrypted to Victor;

  Victor chooses at random an edge $e \in E$ and sends it to Peggy for the coloring of the endpoints of e;

  if $e = (u, v) \in E$ **then**

    Peggy reveals the coloring of u and v and "proves" that they correspond to their encryption;

  else

    Peggy stops;

  end if

---

[a] Goldreich, Micali, and Wigderson (1986).

10: **if** the “proof” provided in Line 6 is not valid **then**

11: Victor rejects and stops;

12: **end if**

13: **if** $\pi(\phi(u)) = \pi(\phi(v))$ or $\pi(\phi(u)), \pi(\phi(v)) \notin \{1, 2, 3\}$ **then**

14: Victor rejects and stops;

15: **end if**

16: **end for**

17: Victor accepts;

Analysis

- If the graph is 3-colorable and both Peggy and Victor follow the protocol, then Victor always accepts.
- If the graph is not 3-colorable and Victor follows the protocol, then however Peggy plays, Victor will accept with probability $\leq (1 - m^{-1})^{m^2} \leq e^{-m}$, where $m = |\, E\,|$.
- Thus the protocol is valid.
- This protocol yields no knowledge to Victor as all he gets is a bunch of random pairs.
- The proof that the protocol is zero-knowledge to *any* verifier is intricate.

Approximability

Tackling Intractable Problems

- Many important problems are NP-complete or worse.
- **Heuristics** have been developed to attack them.
- They are **approximation algorithms**.
- How good are the approximations?
  - We are looking for theoretically *guaranteed* bounds, not "empirical" bounds.
- Are there NP problems that cannot be approximated well (assuming NP $\neq$ P)?
- Are there NP problems that cannot be approximated at all (assuming NP $\neq$ P)?

Some Definitions

- Given an **optimization problem**, each problem instance x has a set of **feasible solutions** $F(x)$.
- Each feasible solution $s \in F(x)$ has a cost $c(s) \in \mathbb{Z}^+$.
- The **optimum cost** is $\textsc{opt}(x) = \min_{s \in F(x)} c(s)$ for a minimization problem.
- It is $\textsc{opt}(x) = \max_{s \in F(x)} c(s)$ for a maximization problem.

Approximation Algorithms

- Let algorithm M on x returns a feasible solution.
- M is an **ϵ-approximation algorithm**, where $\epsilon \geq 0$, if for all x,
$$\frac{|c(M(x)) - \text{OPT}(x)|}{\max(\text{OPT}(x), c(M(x)))} \leq \epsilon.$$
  - For a minimization problem,
$$\frac{c(M(x)) - \min_{s \in F(x)} c(s)}{c(M(x))} \leq \epsilon.$$
  - For a maximization problem,
$$\frac{\max_{s \in F(x)} c(s) - c(M(x))}{\max_{s \in F(x)} c(s)} \leq \epsilon.$$

Lower and Upper Bounds

- For a minimization problem,

$$\min_{s\in F(x)} c(s) \leq c(M(x)) \leq \frac{\min_{s\in F(x)} c(s)}{1-\epsilon}.$$

  - So **approximation ratio** $\frac{\min_{s\in F(x)} c(s)}{c(M(x))} \geq 1-\epsilon$.

- For a maximization problem,

$$(1-\epsilon) \times \max_{s\in F(x)} c(s) \leq c(M(x)) \leq \max_{s\in F(x)} c(s).$$

  - So approximation ratio $\frac{c(M(x))}{\max_{s\in F(x)} c(s)} \geq 1-\epsilon$.

- The above are alternative definitions of ϵ-approximation algorithms.

Range Bounds

- ϵ takes values between 0 and 1.
- For maximization problems, an ϵ-approximation algorithm returns solutions within $[\,(1-\epsilon) \times \text{OPT}, \text{OPT}\,]$.
- For minimization problems, an ϵ-approximation algorithm returns solutions within $[\,\text{OPT}, \frac{\text{OPT}}{1-\epsilon}\,]$.
- For each NP-complete optimization problem, we shall be interested in determining the *smallest* ϵ for which there is a polynomial-time ϵ-approximation algorithm.
- Sometimes ϵ has no minimum value.

Approximation Thresholds

- The **approximation threshold** is the greatest lower bound of all $\epsilon \geq 0$ such that there is a polynomial-time ϵ-approximation algorithm.
- The approximation threshold of an optimization problem can be anywhere between 0 (approximation to any desired degree) and 1 (no approximation is possible).
- If P = NP, then all optimization problems in NP have an approximation threshold of 0.
- So we assume P $\neq$ NP for the rest of the discussion.

NODE COVER

- NODE COVER seeks the smallest $C \subseteq V$ in graph $G = (V, E)$ such that for each edge in E, at least one of its endpoints is in C.
- A heuristic to obtain a good node cover is to iteratively move a node with the highest degree to the cover.
- This turns out to produce
$$\frac{c(M(x))}{\text{OPT}(x)} = \Theta(\log n).$$
- Hence the approximation ratio is $\Theta(\log^{-1} n)$.
- It is not an ϵ-approximation algorithm for any $\epsilon < 1$.

A 0.5-Approximation Algorithm[a]

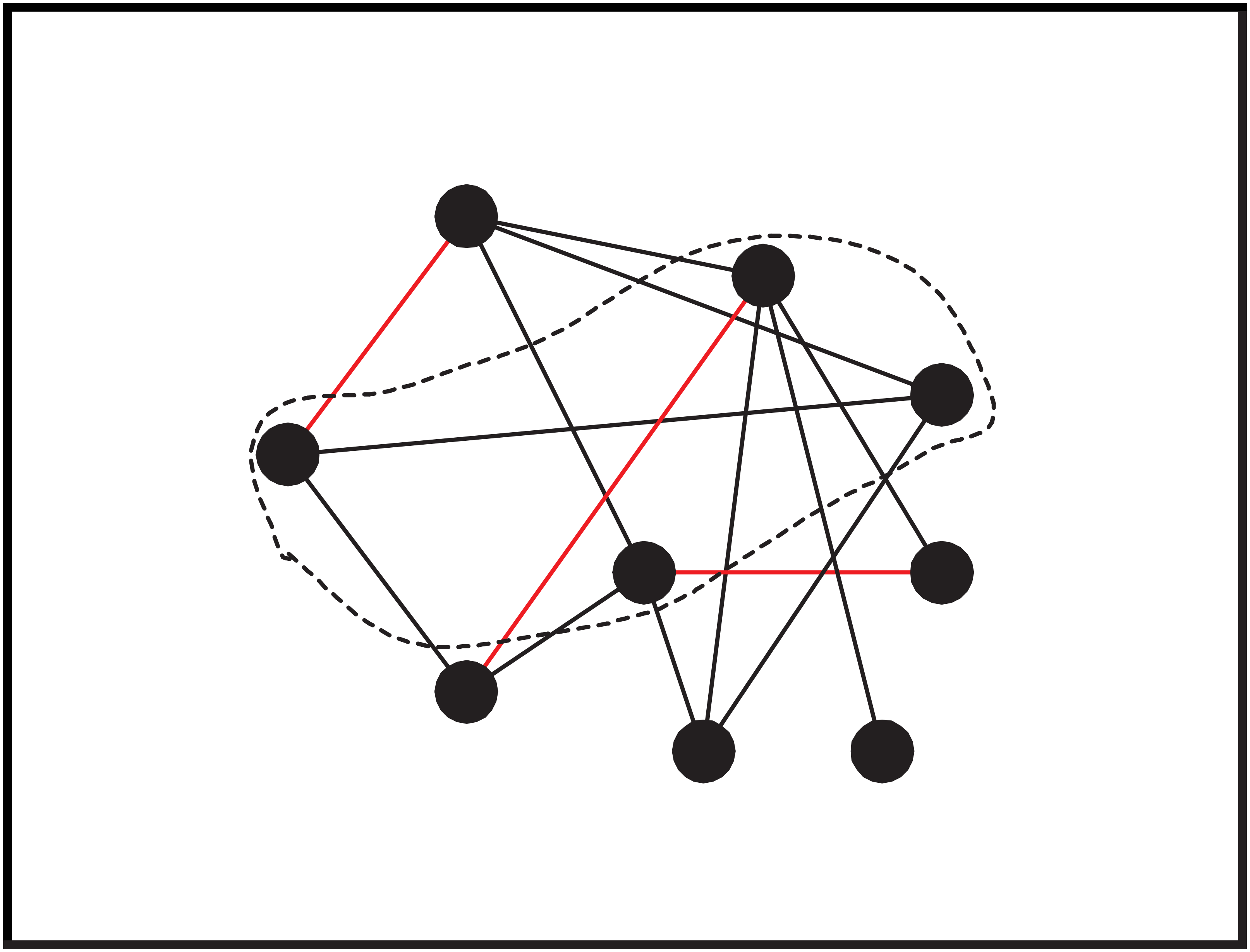
```
C := ∅;
while E ≠ ∅ do
    Delete an arbitrary edge { u, v } from E;
    Delete edges incident with u and v from E;
    Add u and v to C; {Add 2 nodes to C each time.}
end while
return C;
```

[a] Johnson (1974).

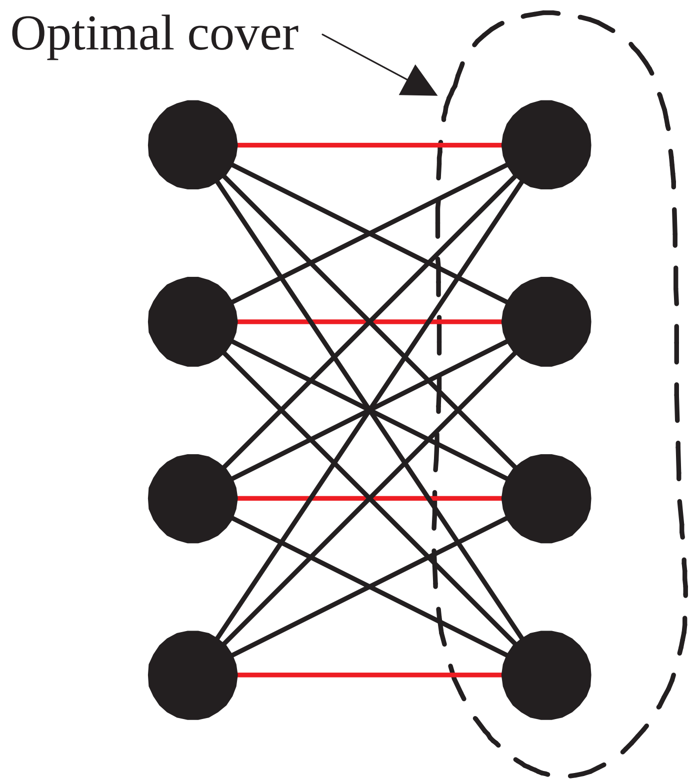
Analysis

- C contains $|C|/2$ edges.
- No two edges of C share a node.
- *Any* node cover must contain at least one node from each of these edges.
- This means that $\text{OPT}(G) \geq |C|/2$.
- So
$$\frac{\text{OPT}(G)}{|C|} \geq 1/2.$$
- The approximation threshold is ≤ 0.5.
- We remark that 0.5 is also the lower bound for any "greedy" algorithms.[a]

[a]Davis and Impagliazzo (2004).



The 0.5 Bound Is Tight for the Algorithm[a]



[a]Contributed by Mr. Jenq-Chung Li (R92922087) on December 20, 2003.

Maximum Satisfiability

- Given a set of clauses, MAXSAT seeks the truth assignment that satisfies the most.
- MAX2SAT is already NP-complete (p. 266).
- Consider the more general k-MAXGSAT for constant k.
  - Given a set of boolean expressions $\Phi = \{\phi_1, \phi_2, \ldots, \phi_m\}$ in n variables.
  - Each ϕ_i is a *general* expression involving k variables.
  - k-MAXGSAT seeks the truth assignment that satisfies the most expressions.

A Probabilistic Interpretation of an Algorithm

- Each ϕ_i involves exactly k variables and is satisfied by t_i of the 2^k truth assignments.
- A random truth assignment $\in \{0,1\}^n$ satisfies ϕ_i with probability $p(\phi_i) = t_i/2^k$.
  - $p(\phi_i)$ is easy to calculate as k is a constant.
- Hence a random truth assignment satisfies an expected number

$$p(\Phi) = \sum_{i=1}^{m} p(\phi_i)$$

of expressions ϕ_i.

The Search Procedure

- Clearly

$$p(\Phi) = \frac{1}{2} \{ p(\Phi[\, x_1 = \texttt{true}\,]) + p(\Phi[\, x_1 = \texttt{false}\,]) \}.$$

- Select the $t_1 \in \{\texttt{true}, \texttt{false}\}$ such that $p(\Phi[\, x_1 = t_1\,])$ is the larger one.

- Note that $p(\Phi[\, x_1 = t_1\,]) \geq p(\Phi)$.

- Repeat with expression $\Phi[\, x_1 = t_1\,]$ until all variables x_i have been given truth values t_i and all ϕ_i either true or false.

The Search Procedure (concluded)

- By our hill-climbing procedure,

$$
\begin{aligned}
& p(\Phi[\, x_1 = t_1, x_2 = t_2, \ldots, x_n = t_n \,]) \\
\geq \quad & \cdots \\
\geq \quad & p(\Phi[\, x_1 = t_1, x_2 = t_2 \,]) \\
\geq \quad & p(\Phi[\, x_1 = t_1 \,]) \\
\geq \quad & p(\Phi).
\end{aligned}
$$

- So at least $p(\Phi)$ expressions are satisfied by truth assignment $(t_1, t_2, \ldots, t_n)$.
- The algorithm is deterministic.

Approximation Analysis

- The optimum is at most the number of satisfiable ϕ_i—i.e., those with $p(\phi_i) > 0$.
- Hence the ratio of algorithm's output vs. the optimum is

$$\geq \frac{p(\Phi)}{\sum_{p(\phi_i)>0} 1} = \frac{\sum_i p(\phi_i)}{\sum_{p(\phi_i)>0} 1} \geq \min_{p(\phi_i)>0} p(\phi_i).$$

- The heuristic is a polynomial-time ϵ-approximation algorithm with $\epsilon = 1 - \min_{p(\phi_i)>0} p(\phi_i)$.
- Because $p(\phi_i) \geq 2^{-k}$, the heuristic is a polynomial-time ϵ-approximation algorithm with $\epsilon = 1 - 2^{-k}$.

Back to MAXSAT

- In MAXSAT, the ϕ_i's are clauses.
- Hence $p(\phi_i) \geq 1/2$, which happens when ϕ_i contains a single literal.
- And the heuristic becomes a polynomial-time ϵ-approximation algorithm with $\epsilon = 1/2$.[a]
- If the clauses have k distinct literals, $p(\phi_i) = 1 - 2^{-k}$.
- And the heuristic becomes a polynomial-time ϵ-approximation algorithm with $\epsilon = 2^{-k}$.
  - This is the best possible for $k \geq 3$ unless P = NP.

---

[a] Johnson (1974).

MAX CUT Revisited

- The NP-complete MAX CUT seeks to partition the nodes of graph $G = (V, E)$ into $(S, V - S)$ so that there are as many edges as possible between S and $V - S$ (p. 290).
- **Local search** starts from a feasible solution and performs "local" improvements until none are possible.

A 0.5-Approximation Algorithm for MAX CUT

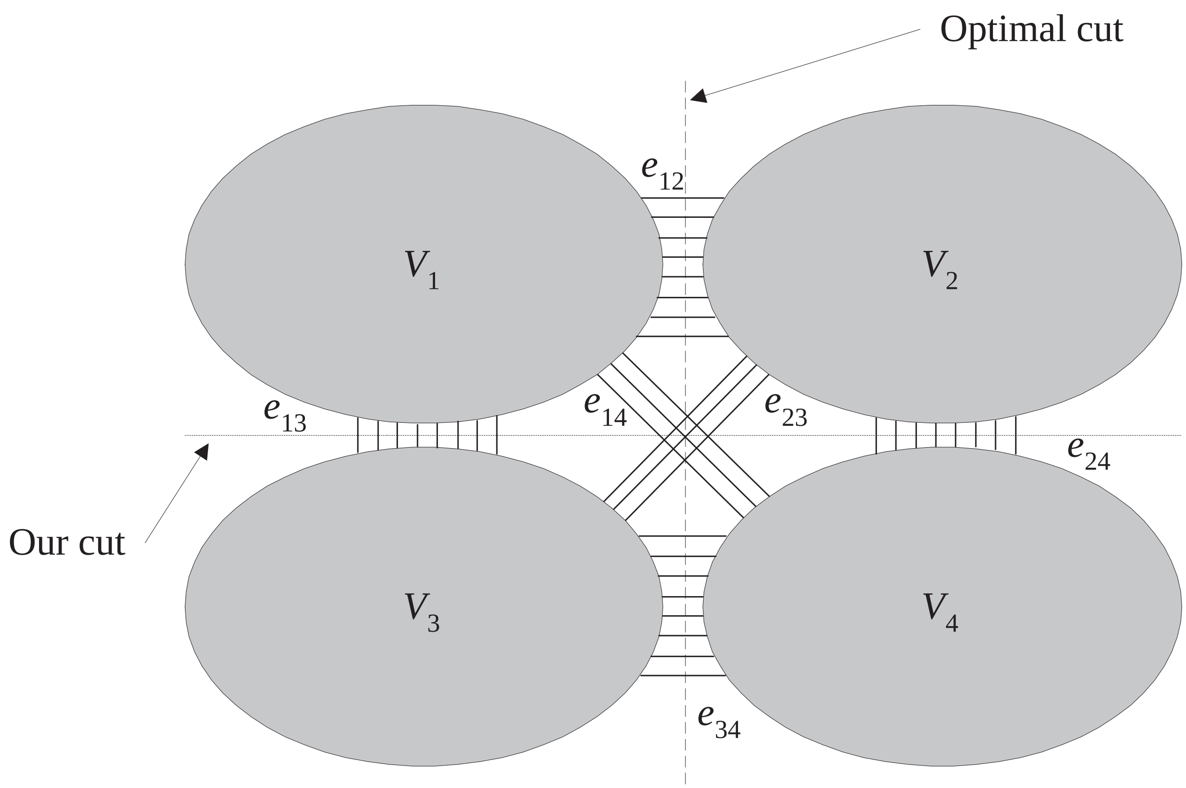
$S := \emptyset$;
while $\exists v \in V$ whose switching sides results in a larger cut **do**
    Switch the side of v;
end while
return S;

- A 0.12-approximation algorithm exists.[a]
- 0.059-approximation algorithms do not exist unless NP = ZPP.

---

[a] Goemans and Williamson (1995).

Analysis



Analysis (continued)

- Partition $V = V_1 \cup V_2 \cup V_3 \cup V_4$, where our algorithm returns $(V_1 \cup V_2, V_3 \cup V_4)$ and the optimum cut is $(V_1 \cup V_3, V_2 \cup V_4)$.
- Let e_{ij} be the number of edges between V_i and V_j.
- Because no migration of nodes can improve the algorithm's cut, for each node in V_1, its edges to $V_1 \cup V_2$ are outnumbered by those to $V_3 \cup V_4$.
- Considering all nodes in V_1 together, we have $2e_{11} + e_{12} \leq e_{13} + e_{14}$, which implies

$$e_{12} \leq e_{13} + e_{14}.$$

Analysis (concluded)

- Similarly,

$$\begin{aligned} e_{12} &\leq e_{23} + e_{24} \\ e_{34} &\leq e_{23} + e_{13} \\ e_{34} &\leq e_{14} + e_{24} \end{aligned}$$

- Adding all four inequalities, dividing both sides by 2, and adding the inequality $e_{14} + e_{23} \leq e_{14} + e_{23} + e_{13} + e_{24}$, we obtain

$$e_{12} + e_{34} + e_{14} + e_{23} \leq 2(e_{13} + e_{14} + e_{23} + e_{24}).$$

- The above says our solution is at least half the optimum.

Approximability, Unapproximability, and Between

- KNAPSACK, NODE COVER, MAXSAT, and MAX CUT have approximation thresholds less than 1.
  - KNAPSACK has a threshold of 0 (see p. 590).
  - But NODE COVER and MAXSAT have a threshold larger than 0.
- The situation is maximally pessimistic for TSP: It cannot be approximated unless P = NP (see p. 588).
  - The approximation threshold of TSP is 1.
    - The threshold is 1/3 if the TSP satisfies the triangular inequality.
  - The same holds for INDEPENDENT SET.

Unapproximability of TSP[a]

Theorem 74 *The approximation threshold of* TSP *is 1 unless P = NP.*

- Suppose there is a polynomial-time ϵ-approximation algorithm for TSP for some $\epsilon < 1$.
- We shall construct a polynomial-time algorithm for the NP-complete HAMILTONIAN CYCLE.
- Given any graph $G = (V, E)$, construct a TSP with $|V|$ cities with distances

$$d_{ij} = \begin{cases} 1, & \text{if } \{ i, j \} \in E \\ \frac{|V|}{1-\epsilon}, & \text{otherwise} \end{cases}$$

[a] Sahni and Gonzales (1976).

The Proof (concluded)

- Run the alleged approximation algorithm on this TSP.
- Suppose a tour of cost $|V|$ is returned.
  - This tour must be a Hamiltonian cycle.
- Suppose a tour with at least one edge of length $\frac{|V|}{1-\epsilon}$ is returned.
  - The total length of this tour is $> \frac{|V|}{1-\epsilon}$.
  - Because the algorithm is ϵ-approximate, the optimum is at least $1 - \epsilon$ times the returned tour's length.
  - The optimum tour has a cost exceeding $|V|$.
  - Hence G has no Hamiltonian cycles.

KNAPSACK Has an Approximation Threshold of Zero[a]

Theorem 75 *For any ϵ, there is a polynomial-time ϵ-approximation algorithm for* KNAPSACK.

- We have n weights $w_1, w_2, \ldots, w_n \in \mathbb{Z}^+$, a weight limit W, and n values $v_1, v_2, \ldots, v_n \in \mathbb{Z}^+$.[b]
- We must find an $S \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{i \in S} w_i \leq W$ and $\sum_{i \in S} v_i$ is the largest possible.
- Let

$$V = \max\{v_1, v_2, \ldots, v_n\}.$$

---

[a] Ibarra and Kim (1975).

[b] If the values are fractional, the result is slightly messier but the main conclusion remains correct. Contributed by Mr. Jr-Ben Tian (`R92922045`) on December 29, 2004.

The Proof (continued)

- For $0 \leq i \leq n$ and $0 \leq v \leq nV$, define $W(i, v)$ to be the minimum weight attainable by selecting some among the i first items, so that their value is exactly v.
- Start with $W(0, v) = \infty$ for all v.
- Then

$$W(i+1, v) = \min\{W(i, v), W(i, v - v_{i+1}) + w_{i+1}\}.$$

- Finally, pick the largest v such that $W(n, v) \leq W$.
- The running time is $O(n^2 V)$, not polynomial time.
- Key idea: Limit the number of precision bits.

The Proof (continued)

- Given the instance $x = (w_1, \ldots, w_n, W, v_1, \ldots, v_n)$, we define the approximate instance

$$x' = (w_1, \ldots, w_n, W, v'_1, \ldots, v'_n),$$

where

$$v'_i = 2^b \left\lfloor \frac{v_i}{2^b} \right\rfloor.$$

- Solving x' takes time $O(n^2 V / 2^b)$.
- The solution S' is close to the optimum solution S:

$$\sum_{i \in S} v_i \geq \sum_{i \in S'} v_i \geq \sum_{i \in S'} v'_i \geq \sum_{i \in S} v'_i \geq \sum_{i \in S} (v_i - 2^b) \geq \sum_{i \in S} v_i - n 2^b.$$

The Proof (continued)

- Hence
$$\sum_{i \in S'} v_i \geq \sum_{i \in S} v_i - n2^b.$$
- Without loss of generality, $w_i \leq W$ (otherwise item i is redundant).
- V is a lower bound on OPT.
  - Picking the item with value V alone is a legitimate choice.
- The relative error from the optimum is $\leq n2^b/V$ as
$$\frac{\sum_{i \in S} v_i - \sum_{i \in S'} v_i}{\sum_{i \in S} v_i} \leq \frac{\sum_{i \in S} v_i - \sum_{i \in S'} v_i}{V} \leq \frac{n2^b}{V}.$$

The Proof (concluded)

- Truncate the last $b = \lfloor \log_2 \frac{\epsilon V}{n} \rfloor$ bits of the values.
- The algorithm becomes ϵ-approximate (see Eq. (8) on p. 567).
- The running time is then $O(n^2 V / 2^b) = O(n^3/\epsilon)$, a polynomial in n and $1/\epsilon$.

Pseudo-Polynomial-Time Algorithms

- Consider problems with inputs that consist of a collection of integer parameters (TSP, KNAPSACK, etc.).
- An algorithm for such a problem whose running time is a polynomial of the input length and the *value* (not length) of the largest integer parameter is a **pseudo-polynomial-time algorithm**.[a]
- On p. 591, we presented a pseudo-polynomial-time algorithm for KNAPSACK that runs in time $O(n^2 V)$.
- How about TSP (D), another NP-complete problem?

---

[a] Garey and Johnson (1978).

No Pseudo-Polynomial-Time Algorithms for TSP (D)

- By definition, a pseudo-polynomial-time algorithm becomes polynomial-time if each integer parameter is limited to having a *value* polynomial in the input length.
- Corollary 38 (p. 306) showed that HAMILTONIAN PATH is reducible to TSP (D) with weights 1 and 2.
- As HAMILTONIAN PATH is NP-complete, TSP (D) cannot have pseudo-polynomial-time algorithms unless P = NP.
- TSP (D) is said to be **strongly NP-hard**.
- Many weighted versions of NP-complete problems are strongly NP-hard.

Polynomial-Time Approximation Scheme

- Algorithm M is a **polynomial-time approximation scheme** (**PTAS**) for a problem if:
  - For each $\epsilon > 0$ and instance x of the problem, M runs in time polynomial (depending on ϵ) in $|\, x \,|$.
    - Think of ϵ as a constant.
  - M is an ϵ-approximation algorithm for every $\epsilon > 0$.

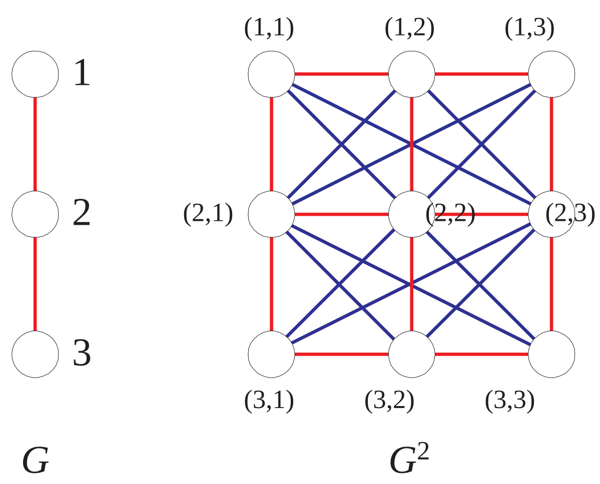
Fully Polynomial-Time Approximation Scheme

- A polynomial-time approximation scheme is **fully polynomial** (**FPTAS**) if the running time depends polynomially on $|x|$ and $1/\epsilon$.
  - Maybe the best result for a "hard" problem.
  - For instance, KNAPSACK is fully polynomial with a running time of $O(n^3/\epsilon)$ (p. 590).

Square of G

- Let $G = (V, E)$ be an undirected graph.
- G^2 has nodes $\{(v_1, v_2) : v_1, v_2 \in V\}$ and edges

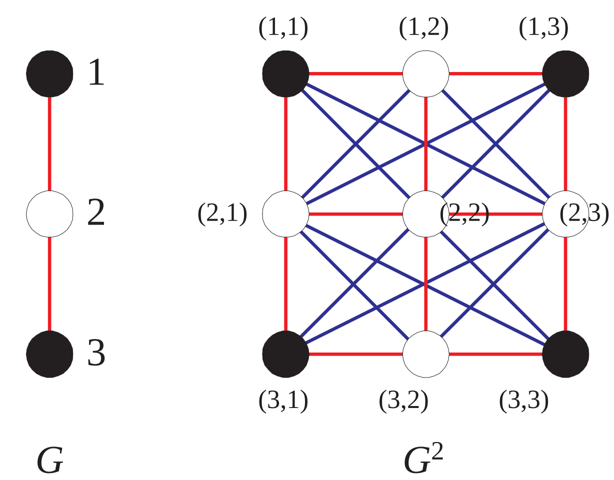
  $\{\{ (u, u'), (v, v') \} : (u = v \wedge \{ u', v' \} \in E) \vee \{ u, v \} \in E\}$.



Independent Sets of G and G^2

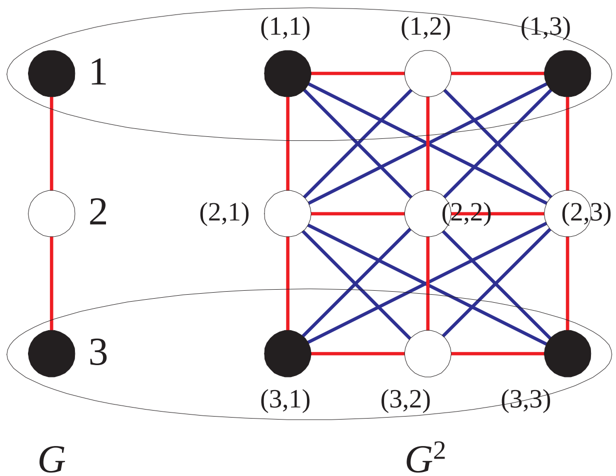
Lemma 76 *$G(V, E)$ has an independent set of size k if and only if G^2 has an independent set of size k^2.*

- Suppose G has an independent set $I \subseteq V$ of size k.
- $\{(u, v) : u, v \in I\}$ is an independent set of size k^2 of G^2.



The Proof (continued)

- Suppose G^2 has an independent set I^2 of size k^2.
- $U \equiv \{u : \exists v \in V\ (u, v) \in I^2\}$ is an independent set of G.



- $|\,U\,|$ is the number of "rows" that the nodes in I^2 occupy.

The Proof (concluded)[a]

- If $|U| \geq k$, then we are done.
- Now assume $|U| < k$.
- As the k^2 nodes in I^2 cover fewer than k "rows," there must be a "row" in possession of $> k$ nodes of I^2.
- Those $> k$ nodes will be independent in G as each "row" is a copy of G.

[a]Thanks to a lively class discussion on December 29, 2004.

Approximability of INDEPENDENT SET

- The approximation threshold of the maximum independent set is either zero or one (it is one!).

Theorem 77 *If there is a polynomial-time ϵ-approximation algorithm for* INDEPENDENT SET *for* any $0 < \epsilon < 1$, *then there is a polynomial-time approximation scheme.*

- Let G be a graph with a maximum independent set of size k.
- Suppose there is an $O(n^i)$-time ϵ-approximation algorithm for INDEPENDENT SET.

The Proof (continued)

- By Lemma 76 (p. 600), the maximum independent set of G^2 has size k^2.
- Apply the algorithm to G^2.
- The running time is $O(n^{2i})$.
- The resulting independent set has size $\geq (1-\epsilon)\, k^2$.
- By the construction in Lemma 76 (p. 600), we can obtain an independent set of size $\geq \sqrt{(1-\epsilon)\, k^2}$ for G.
- Hence there is a $(1-\sqrt{1-\epsilon})$-approximation algorithm for INDEPENDENT SET.

The Proof (concluded)

- In general, we can apply the algorithm to G^{2^ℓ} to obtain an $(1-(1-\epsilon)^{2^{-\ell}})$-approximation algorithm for INDEPENDENT SET.
- The running time is $n^{2^\ell i}$.[a]
- Now pick $\ell = \lceil \log \frac{\log(1-\epsilon)}{\log(1-\epsilon')} \rceil$.
- The running time becomes $n^{i \frac{\log(1-\epsilon)}{\log(1-\epsilon')}}$.
- It is an ϵ'-approximation algorithm for INDEPENDENT SET.

[a] It is not fully polynomial.

Comments

- INDEPENDENT SET and NODE COVER are reducible to each other (Corollary 36, p. 286).
- NODE COVER has an approximation threshold at most 0.5 (p. 573).
- But INDEPENDENT SET is unapproximable (see the textbook).
- INDEPENDENT SET limited to graphs with degree $\leq k$ is called k-DEGREE INDEPENDENT SET.
- k-DEGREE INDEPENDENT SET is approximable (see the textbook).

On P vs NP

Density[a]

The **density** of language $L \subseteq \Sigma^*$ is defined as

$$\text{dens}_L(n) = |\{x \in L : |\,x\,| \leq n\}|.$$

- If $L = \{0, 1\}^*$, then $\text{dens}_L(n) = 2^{n+1} - 1$.
- So the density function grows at most exponentially.
- For a unary language $L \subseteq \{0\}^*$,

$$\text{dens}_L(n) \leq n + 1.$$

  – Because $L \subseteq \{\epsilon, 0, 00, \ldots, \overbrace{00 \cdots 0}^{n}, \ldots\}$.

[a]Berman and Hartmanis (1977).

Sparsity

- **Sparse languages** are languages with polynomially bounded density functions.
- **Dense languages** are languages with superpolynomial density functions.

Self-Reducibility for SAT

- An algorithm exploits **self-reducibility** if it reduces the problem to the same problem with a smaller size.
- Let ϕ be a boolean expression in n variables $x_1, x_2, \ldots, x_n$.
- $t \in \{0, 1\}^j$ is a **partial** truth assignment for $x_1, x_2, \ldots, x_j$.
- $\phi[\,t\,]$ denotes the expression after substituting the truth values of t for $x_1, x_2, \ldots, x_t$ in ϕ.

An Algorithm for SAT with Self-Reduction

We call the algorithm below with empty t.

```
if | t | = n then
    return φ[ t ];
else
    return φ[ t0 ] ∨ φ[ t1 ];
end if
```

The above algorithm runs in exponential time, by visiting all the partial assignments (or nodes on a depth-n binary tree).

NP-Completeness and Density[a]

Theorem 78 *If a unary language* $U \subseteq \{0\}^*$ *is NP-complete, then* $P = NP$.

- Suppose there is a reduction R from SAT to U.
- We shall use R to guide us in finding the truth assignment that satisfies a given boolean expression ϕ with n variables if it is satisfiable.
- Specifically, we use R to prune the exponential-time exhaustive search on p. 611.
- The trick is to keep the already discovered results $\phi[t]$ in a table H.

[a]Berman (1978).

```
if |t| = n then
    return φ[t];
else
    if (R(φ[t]), v) is in table H then
        return v;
    else
        if φ[t0] = "satisfiable" or φ[t1] = "satisfiable" then
            Insert (R(φ[t]), 1) into H;
            return "satisfiable";
        else
            Insert (R(φ[t]), 0) into H;
            return "unsatisfiable";
        end if
    end if
end if
```

The Proof (continued)

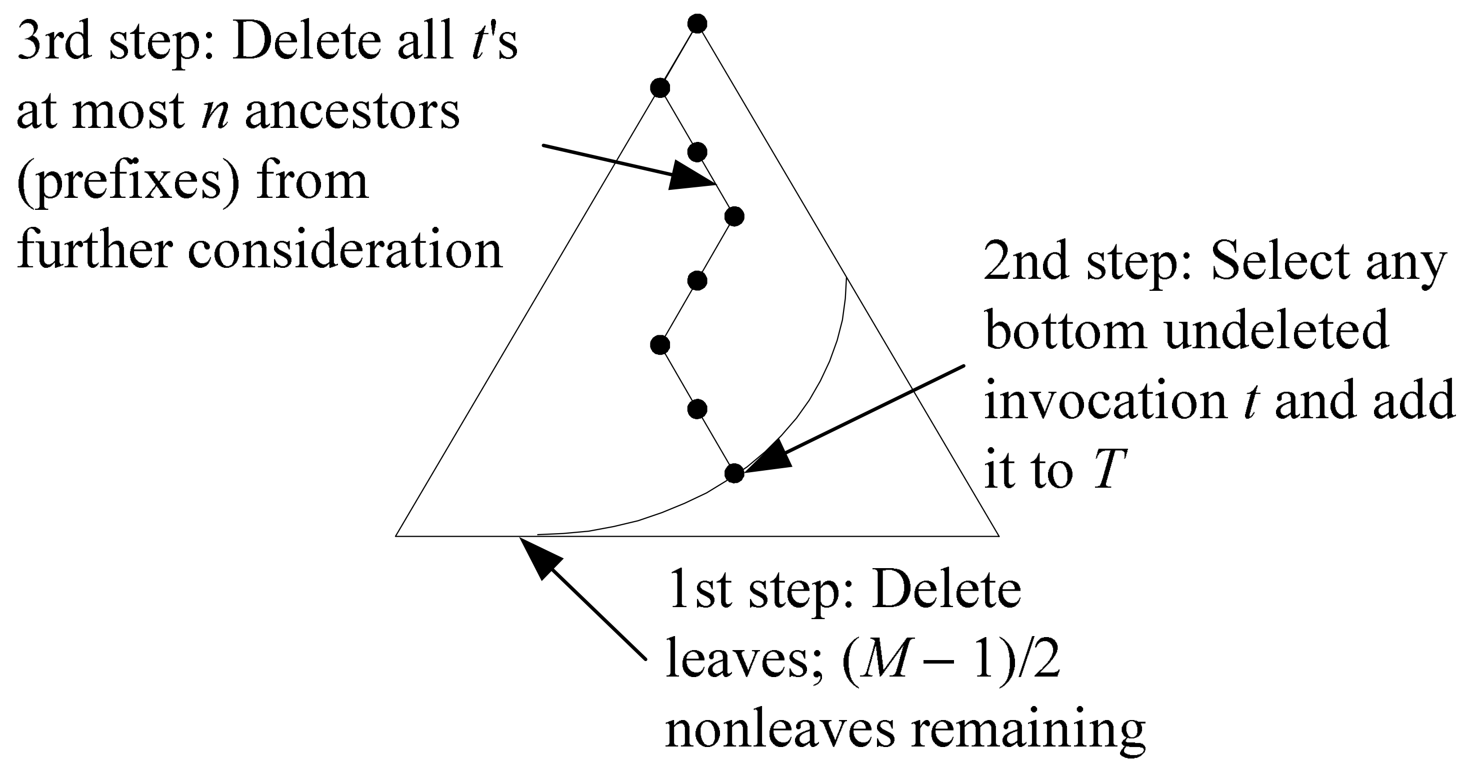
- Since R is a reduction, $R(\phi[\,t\,]) = R(\phi[\,t'\,])$ implies that $\phi[\,t\,]$ and $\phi[\,t'\,]$ must be both satisfiable or unsatisfiable.
- $R(\phi[\,t\,])$ has polynomial length $\leq p(n)$ because R runs in log space.
- As R maps to unary numbers, there are only polynomially many $p(n)$ values of $R(\phi[\,t\,])$.
- How many nodes of the complete binary tree (of invocations/truth assignments) need to be visited?
- If that number is a polynomial, the overall algorithm runs in polynomial time and we are done.

The Proof (continued)

- A search of the table takes time $O(p(n))$ in the random access memory model.
- The running time is $O(Mp(n))$, where M is the total number of invocations of the algorithm.
- The invocations of the algorithm form a binary tree of depth at most n.

The Proof (continued)

- There is a set $T = \{t_1, t_2, \ldots\}$ of invocations (partial truth assignments, i.e.) such that:
  - $|T| \geq (M-1)/(2n)$.
  - All invocations in T are **recursive** (nonleaves).
  - None of the elements of T is a prefix of another.


3rd step: Delete all t's
at most n ancestors
(prefixes) from
further consideration
2nd step: Select any
bottom undeleted
invocation t and add
it to T
1st step: Delete
leaves; (M – 1)/2
nonleaves remaining

The Proof (continued)

- All invocations $t \in T$ have different $R(\phi[\,t\,])$ values.
  - None of $s, t \in T$ is a prefix of another.
  - The invocation of one started after the invocation of the other had terminated.
  - If they had the same value, the one that was invoked second would have looked it up, and therefore would not be recursive, a contradiction.
- The existence of T implies that there are at least $(M-1)/(2n)$ different $R(\phi[\,t\,])$ values in the table.

The Proof (concluded)

- We already know that there are at most $p(n)$ such values.
- Hence $(M-1)/(2n) \leq p(n)$.
- Thus $M \leq 2np(n)+1$.
- The running time is therefore $O(Mp(n)) = O(np^2(n))$.
- We comment that this theorem holds for any sparse language, not just unary ones.[a]

[a]Mahaney (1980).

Computation That Counts

Counting Problems

- Counting problems are concerned with the number of solutions.
  - #SAT: the number of satisfying truth assignments to a boolean formula.
  - #HAMILTONIAN PATH: the number of Hamiltonian paths in a graph.
- They cannot be easier than their decision versions.
  - The decision problem has a solution if and only if the solution count is larger than 0.
- But they can be harder than their decision versions.

Decision and Counting Problems

- FP is the set of polynomial-time computable functions $f : \{0,1\}^* \to \mathbb{Z}$.
  - GCD, LCM, matrix-matrix multiplication, etc.
- If $\#\text{SAT} \in \text{FP}$, then $\text{P} = \text{NP}$.
  - Given boolean formula ϕ, calculate its number of satisfying truth assignments, k, in polynomial time.
  - Declare "$\phi \in \text{SAT}$" if and only if $k \geq 1$.
- The validity of the reverse direction is open.

A Counting Problem Harder than Its Decision Version

- Some counting problems are harder than their decision versions.
- CYCLE asks if a directed graph contains a cycle.
- #CYCLE counts the number of cycles in a directed graph.
- CYCLE is in P by a simple greedy algorithm.
- But #CYCLE is hard unless P = NP.

Counting Class #P

A function f is in #P (or $f \in$ #P) if

- There exists a polynomial-time NTM M.
- $M(x)$ has $f(x)$ accepting paths for all inputs x.
- $f(x) =$ number of accepting paths of $M(x)$.

Some #P Problems

- $f(\phi) =$ number of satisfying truth assignments to ϕ.
  - The desired NTM guesses a truth assignment T and accepts ϕ if and only if $T \models \phi$.
  - Hence $f \in \#\mathrm{P}$.
  - f is also called #SAT.
- #HAMILTONIAN PATH.
- #3-COLORING.

#P Completeness

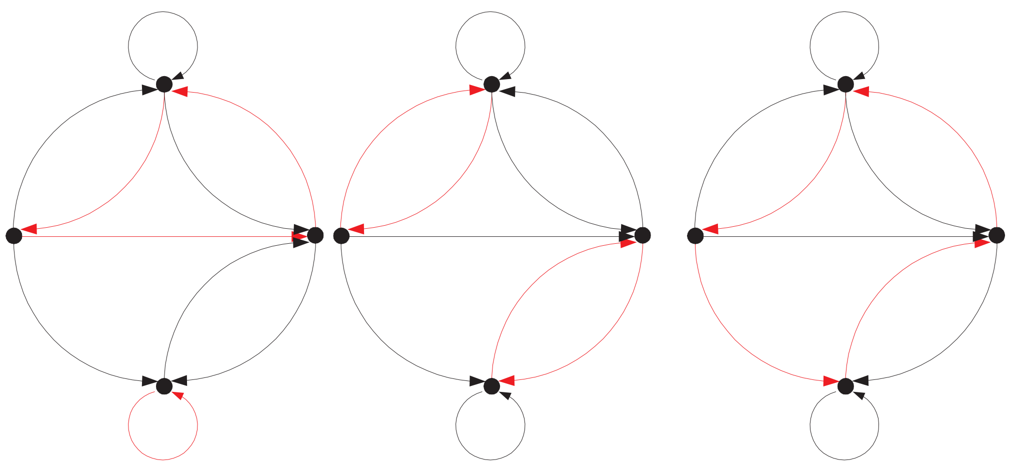
- Function f is #P-complete if
  - $f \in$ #P.
  - #P $\subseteq \mathrm{FP}^f$.
    * Every function in #P can be computed in polynomial time with access to a black box or **oracle** for f.
  - Of course, oracle f will be accessed only a polynomial number of times.
  - #P is said to be **polynomial-time Turing-reducible to** f.

#SAT Is #P-Complete

- First, it is in #P (p. 625).
- Let $f \in \#\text{P}$ compute the number of accepting paths of M.
- Cook's theorem uses a *parsimonious* reduction from M on input x to an instance ϕ of SAT (p. 247).
  - Hence the number of accepting paths of $M(x)$ equals the number of satisfying truth assignments to ϕ.
- Call the oracle #SAT with ϕ to obtain the desired answer regarding $f(x)$.

CYCLE COVER

- A set of node-disjoint cycles that cover all nodes in a directed graph is called a **cycle cover**.



- There are 3 cycle covers (in red) above.

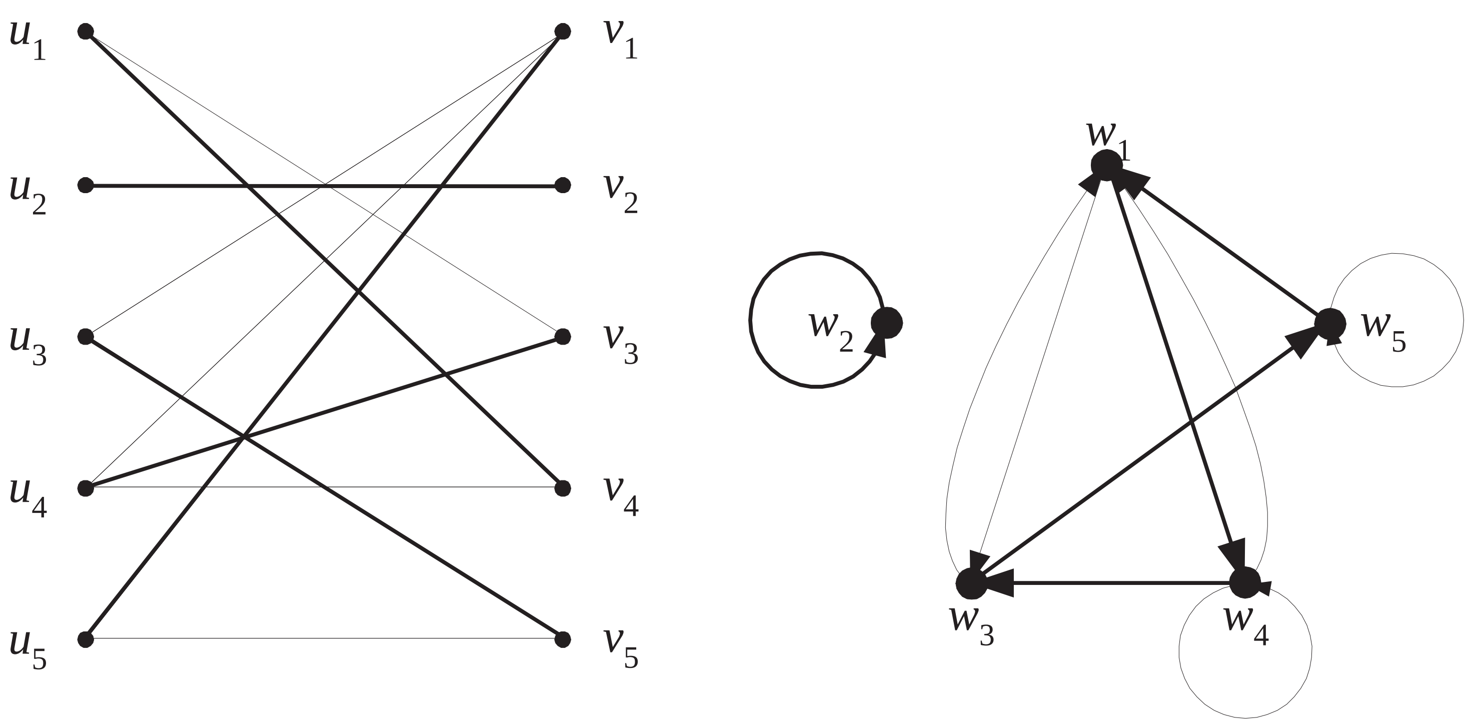
CYCLE COVER and BIPARTITE PERFECT MATCHING

Proposition 79 CYCLE COVER *and* BIPARTITE PERFECT MATCHING *(p. 390) are parsimoniously reducible to each other.*

- A polynomial-time algorithm creates a bipartite graph G' from any directed graph G.
- Moreover, the number cycle covers for G equals the number of bipartite perfect matchings for G'.
- And vice versa.

Corollary 80 CYCLE COVER $\in P$.

Illustration of the Proof



Permanent

- The **permanent** of an $n \times n$ integer matrix A is
$$\text{perm}(A) = \sum_{\pi} \prod_{i=1}^{n} A_{i,\pi(i)}.$$
  - π ranges over all permutations of n elements.
- 0/1 PERMANENT computes the permanent of a 0/1 (binary) matrix.
  - The permanent of a binary matrix is at most $n!$.
- Simpler than determinant (5) on p. 392: no signs.
- But, surprisingly, much harder to compute than determinant!

Permanent and Counting Perfect Matchings

- BIPARTITE PERFECT MATCHING is related to determinant (p. 393).
- #BIPARTITE PERFECT MATCHING is related to permanent.

Proposition 81 0/1 PERMANENT *and* BIPARTITE PERFECT MATCHING *are parsimoniously reducible to each other.*

The Proof

- Given a bipartite graph G, construct an $n \times n$ binary matrix A.
  - The (i, j)th entry A_{ij} is 1 if $(i, j) \in E$ and 0 otherwise.
- Then $\text{perm}(A) =$ number of perfect matchings in G.

Illustration of the Proof Based on p. 630 (Left)

$$A = \begin{bmatrix} 0 & 0 & 1 & \boxed{1} & 0 \\ 0 & \boxed{1} & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & \boxed{1} \\ 1 & 0 & \boxed{1} & 1 & 0 \\ \boxed{1} & 0 & 0 & 0 & 1 \end{bmatrix}.$$

- $\text{perm}(A) = 4$.
- The permutation corresponding to the perfect matching on p. 630 is marked.

Permanent and Counting Cycle Covers

Proposition 82 *0/1* PERMANENT *and* CYCLE COVER *are parsimoniously reducible to each other.*

- Let A be the adjacency matrix of the graph on p. 630 (right).
- Then $\text{perm}(A) =$ number of cycle covers.

Three Parsimoniously Equivalent Problems

From Propositions 79 (p. 629) and 81 (p. 632), we summarize:

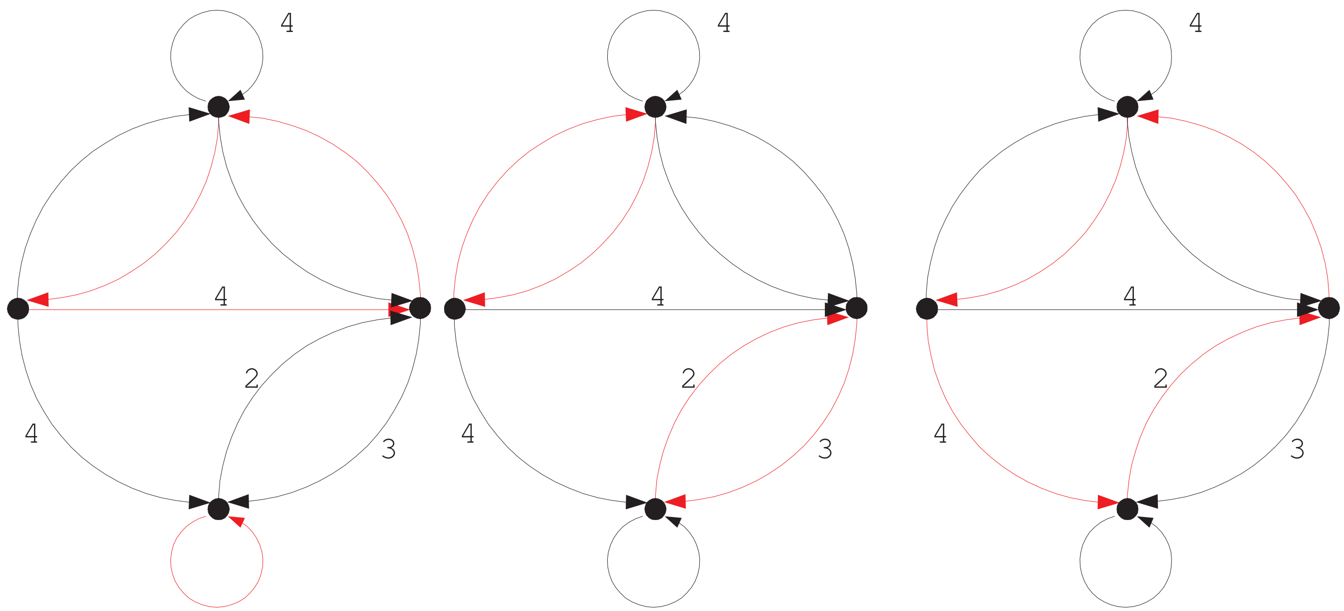
Lemma 83 *0/1* PERMANENT*,* BIPARTITE PERFECT MATCHING*, and* CYCLE COVER *are* **parsimoniously equivalent***.*

We will show that the counting versions of all three problems are in fact #P-complete.

WEIGHTED CYCLE COVER

- Consider a directed graph G with integer weights on the edges.
- The weight of a cycle cover is the product of its edge weights.
- The **cycle count** of G is sum of the weights of all cycle covers.
  - Let A be G's adjacency matrix but $A_{ij} = w_i$ if the edge (i, j) has weight w_i.
  - Then $\text{perm}(A) = G$'s cycle count (same proof as Proposition 82 on p. 635).
- #CYCLE COVER is a special case: All weights are 1.

An Example[a]



There are 3 cycle covers, and the cycle count is

$$(4 \cdot 1 \cdot 1) \cdot (1) + (1 \cdot 1) \cdot (2 \cdot 3) + (4 \cdot 2 \cdot 1 \cdot 1) = 18.$$

[a]Each edge has weight 1 unless stated otherwise.

Three #P-Complete Counting Problems

Theorem 84 (Valiant (1979)) *0/1* PERMANENT, #BIPARTITE PERFECT MATCHING, *and* #CYCLE COVER *are #P-complete.*

- By Lemma 83 (p. 636), it suffices to prove that #CYCLE COVER is #P-complete.
- #SAT is #P-complete (p. 627).
- #3SAT is #P-complete because it and #SAT are parsimoniously equivalent (p. 256).
- We shall prove that #3SAT is polynomial-time Turing-reducible to #CYCLE COVER.

The Proof (continued)

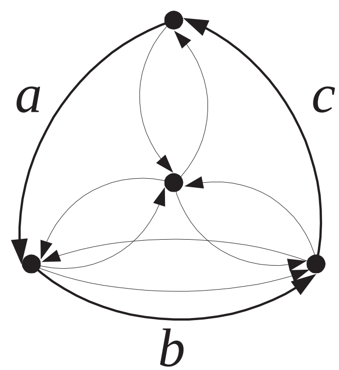
- Let ϕ be the given 3SAT formula.
  - It contains n variables and m clauses (hence $3m$ literals).
  - It has $\#\phi$ satisfying truth assignments.
- First we construct a *weighted* directed graph H with cycle count

$$\#H = 4^{3m} \times \#\phi.$$

- Then we construct an unweighted directed graph G.
- We make sure $\#H$ (hence $\#\phi$) is polynomial-time Turing-reducible to G's number of cycle covers (denoted $\#G$).

The Proof: the Clause Gadget (continued)

- Each clause is associated with a **clause gadget**.



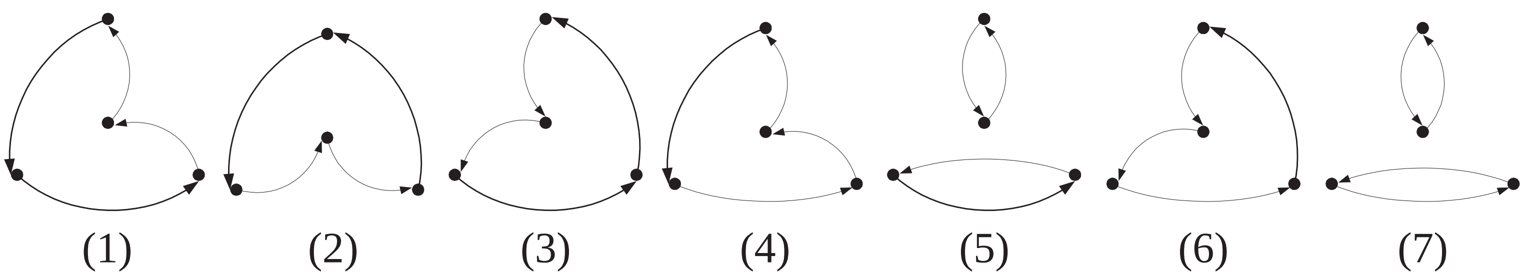
- Each edge has weight 1 unless stated otherwise.
- Each bold edge corresponds to one literal in the clause.
- There are not *parallel* lines as bold edges are schematic only (preview p. 654).

The Proof: the Clause Gadget (continued)

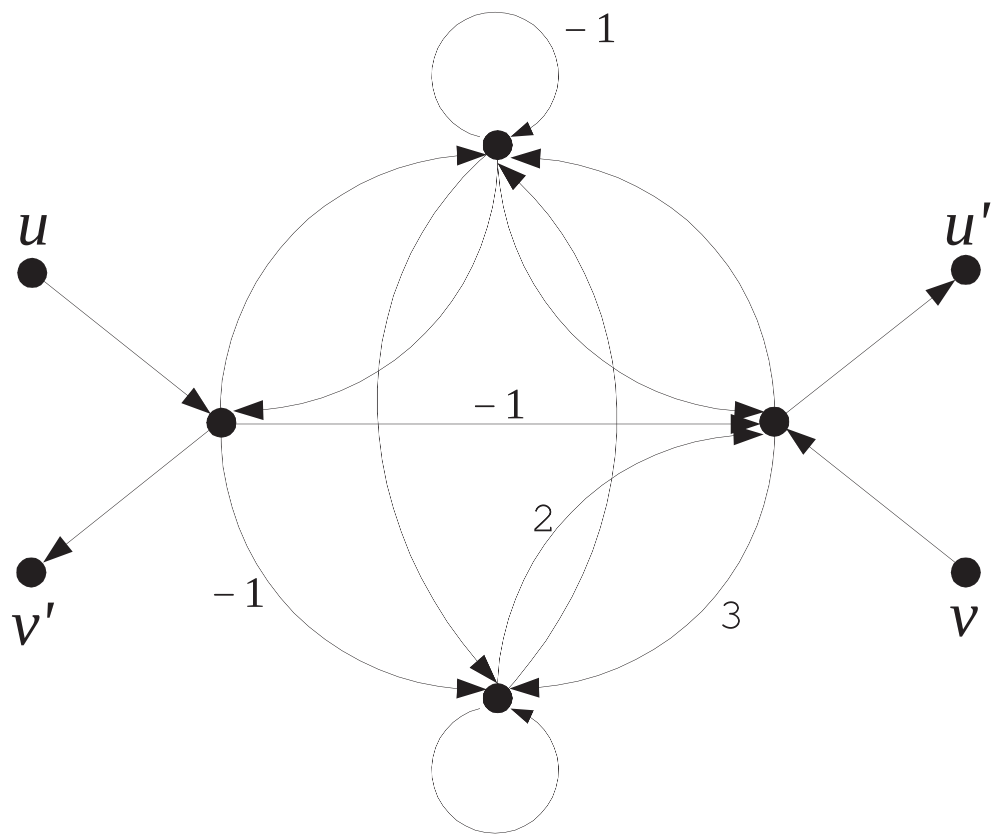
- Following a bold edge means making the literal false (0).
- A cycle cover cannot select *all* 3 bold edges.
  - The interior node would be missing.
- Every proper nonempty subset of bold edges corresponds to a unique cycle cover of weight 1 (see next page).

The Proof: the Clause Gadget (continued)

7 possible cycle covers, one for each satisfying assignment: (1) $a = 0, b = 0, c = 1$, (2) $a = 0, b = 1, c = 0$, etc.

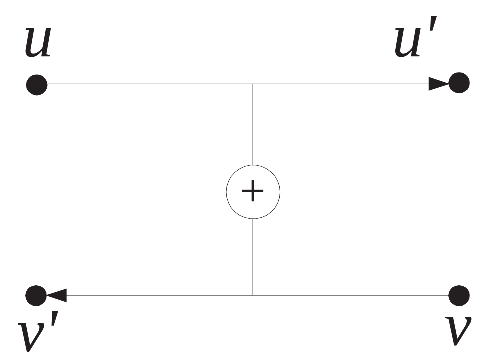


The Proof: the XOR Gadget (continued)



The Proof: Properties of the XOR Gadget (continued)

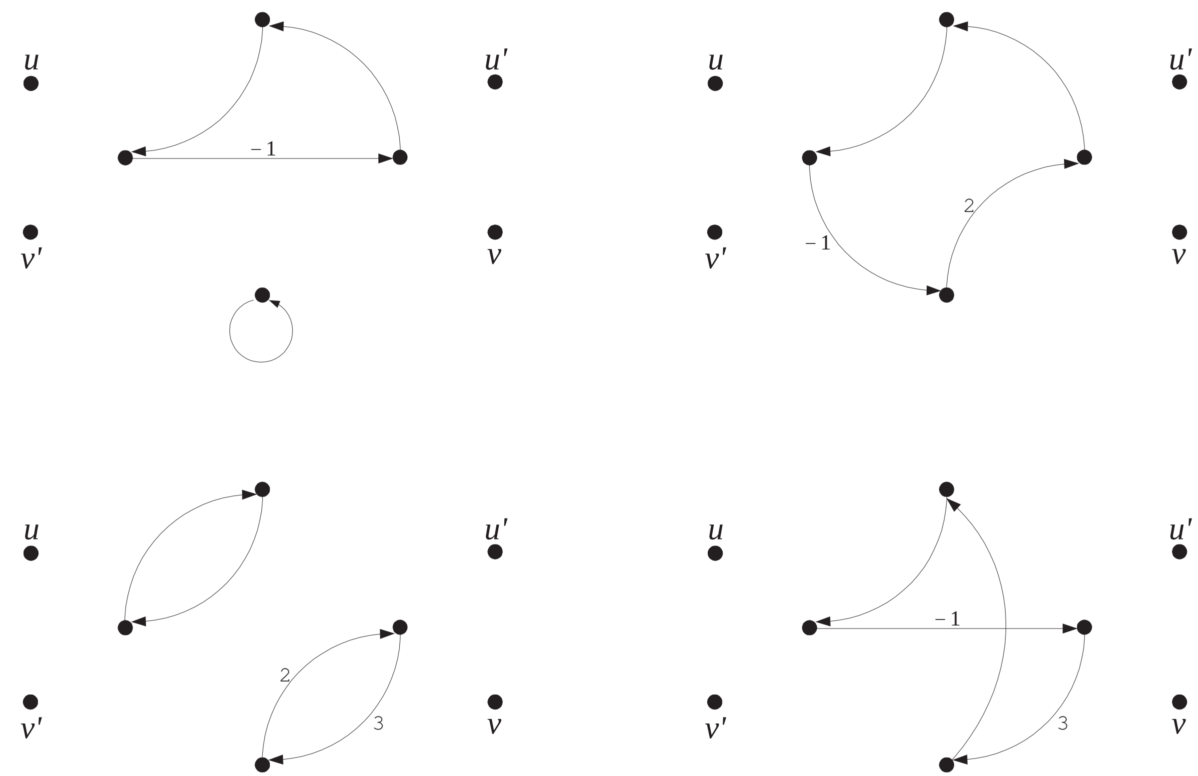
- The XOR gadget schema:



- *At most one* of the 2 schematic edges will be included in a cycle cover.
- There will be $3m$ XOR gadgets, one for each literal.

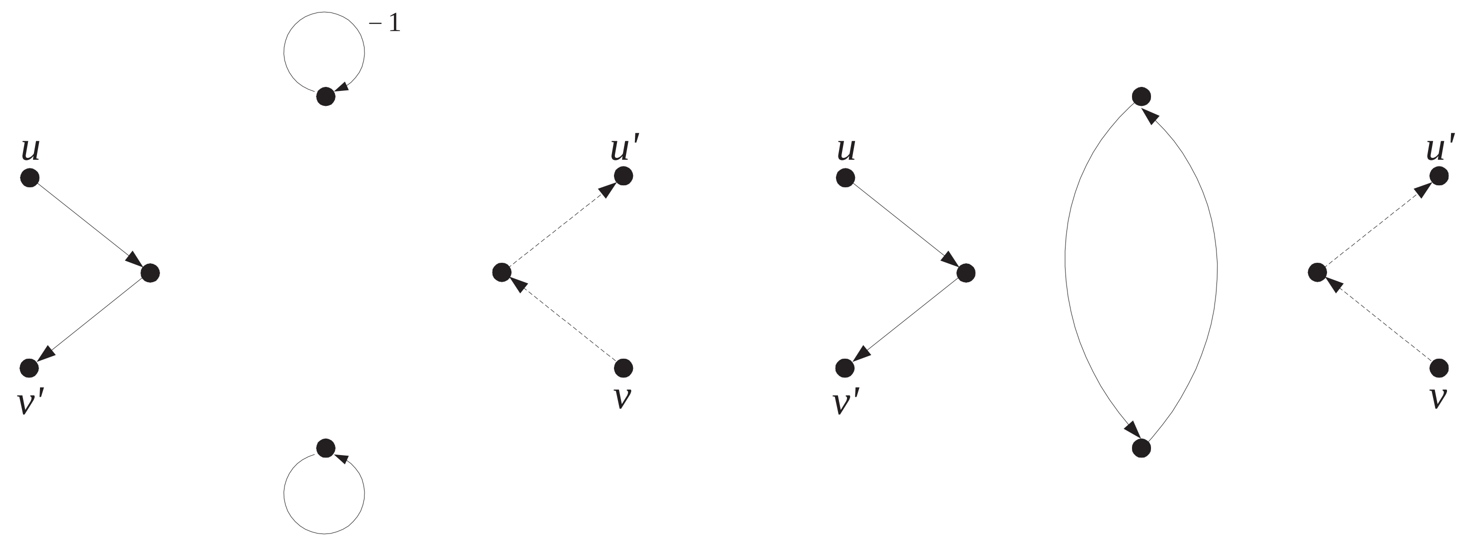
The Proof: Properties of the XOR Gadget (continued)

Total weight of $-1 - 2 + 6 - 3 = 0$ for cycle covers not entering or leaving it.



The Proof: Properties of the XOR Gadget (continued)

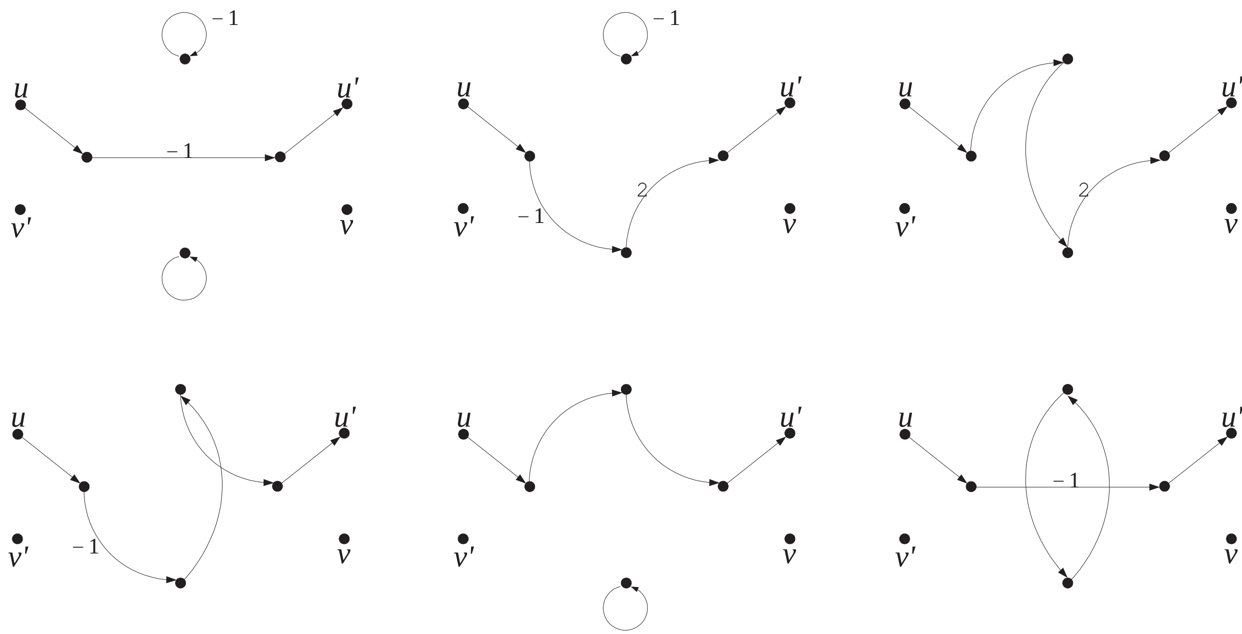
- Total weight of $-1 + 1 = 0$ for cycle covers entering at u and leaving at v'.



- Same for cycle covers entering at v and leaving at u'.

The Proof: Properties of the XOR Gadget (continued)

- Total weight of $1 + 2 + 2 - 1 + 1 - 1 = 4$ for cycle covers entering at u and leaving at u'.



- Same for cycle covers entering at v and leaving at v'.

The Proof: Summary (continued)

- Cycle covers not entering *all* of the XOR gadgets contribute 0 to the cycle count.
  - Fix an XOR gadget x not entered.
  - Now,

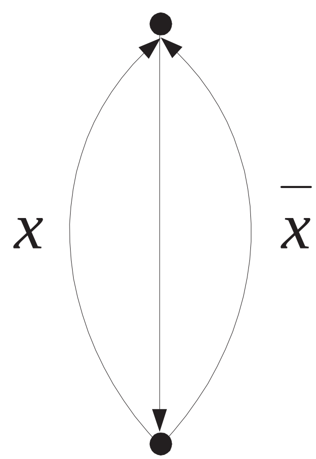
$$
\begin{aligned}
& \text{cycle count} \\
= & \sum_{\text{cycle cover } c \text{ for } H} \text{weight}(c) \\
= & \sum_{\text{cycle cover } c \text{ for } H - x} \text{weight}(c) \sum_{\text{cycle cover } c \text{ for } x} \text{weight}(x) \\
= & \sum_{\text{cycle cover } c \text{ for } H - x} \text{weight}(c) \cdot 0 \\
= & \; 0.
\end{aligned}
$$

The Proof: Summary (continued)

- Cycle covers entering *any* of the XOR gadgets and leaving illegally contribute 0 to the cycle count.
- For every XOR gadget entered and left legally, the total weight of a cycle cover is multiplied by 4.
- Hereafter we consider only cycle covers which enter every XOR gadget and leaves it legally.
  - Only these cycle covers contribute nonzero weights to the cycle count.
  - They are said to **respect** the XOR gadgets.

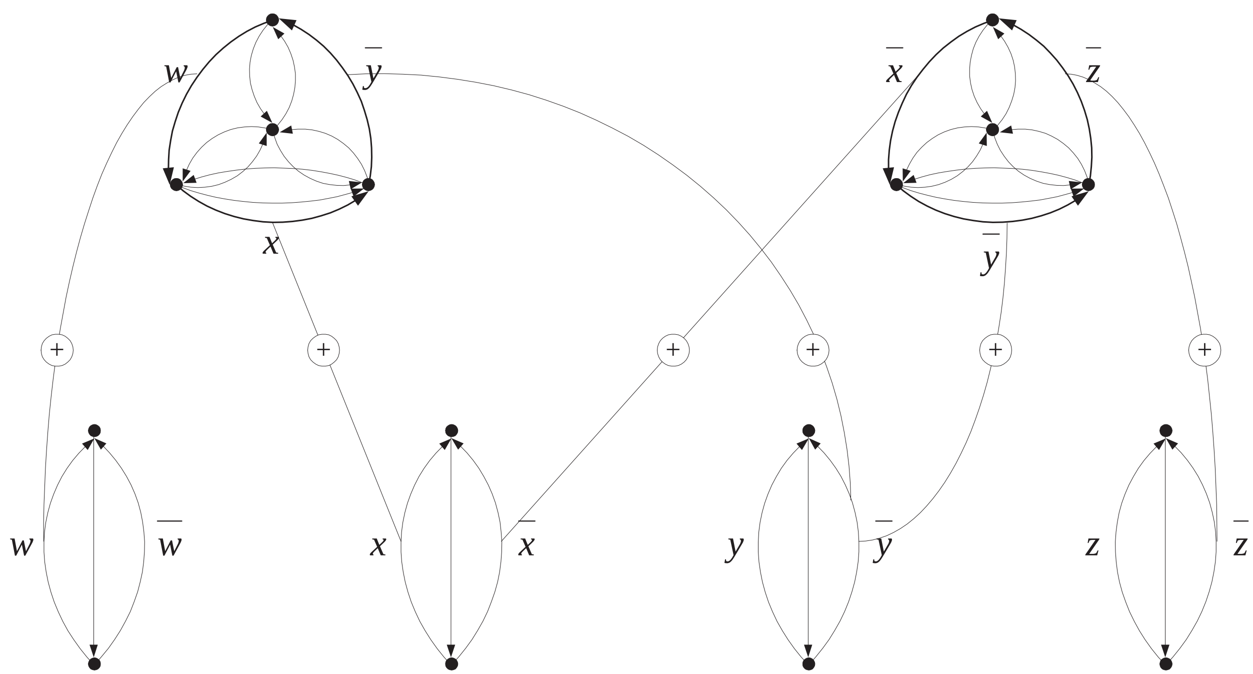
The Proof: the Choice Gadget (continued)

- One choice gadget (a schema) for each variable.

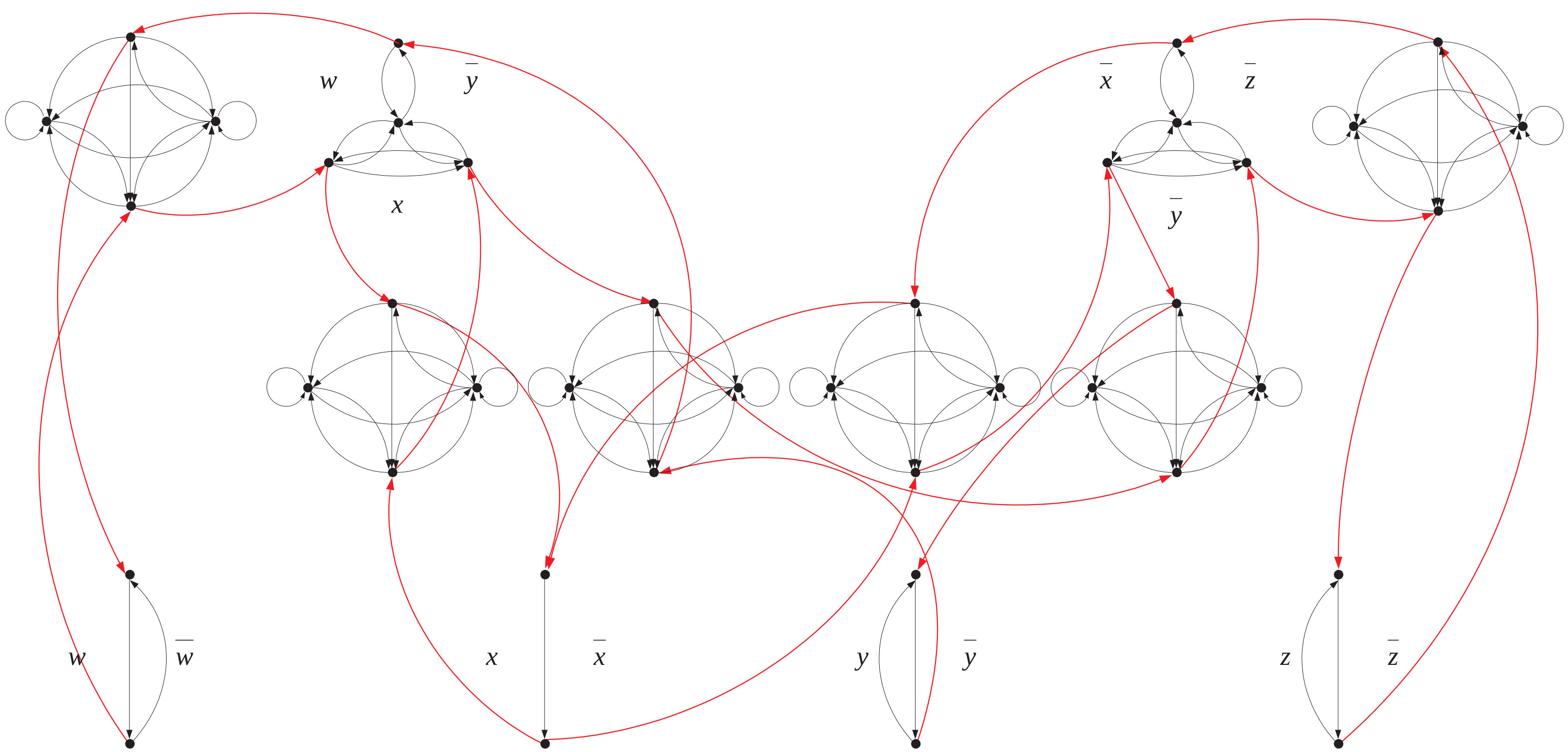


- It gives the truth assignment for the variable.
- Use it with the XOR gadget to enforce consistency.

Schema for $(w \vee x \vee \bar{y}) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})$



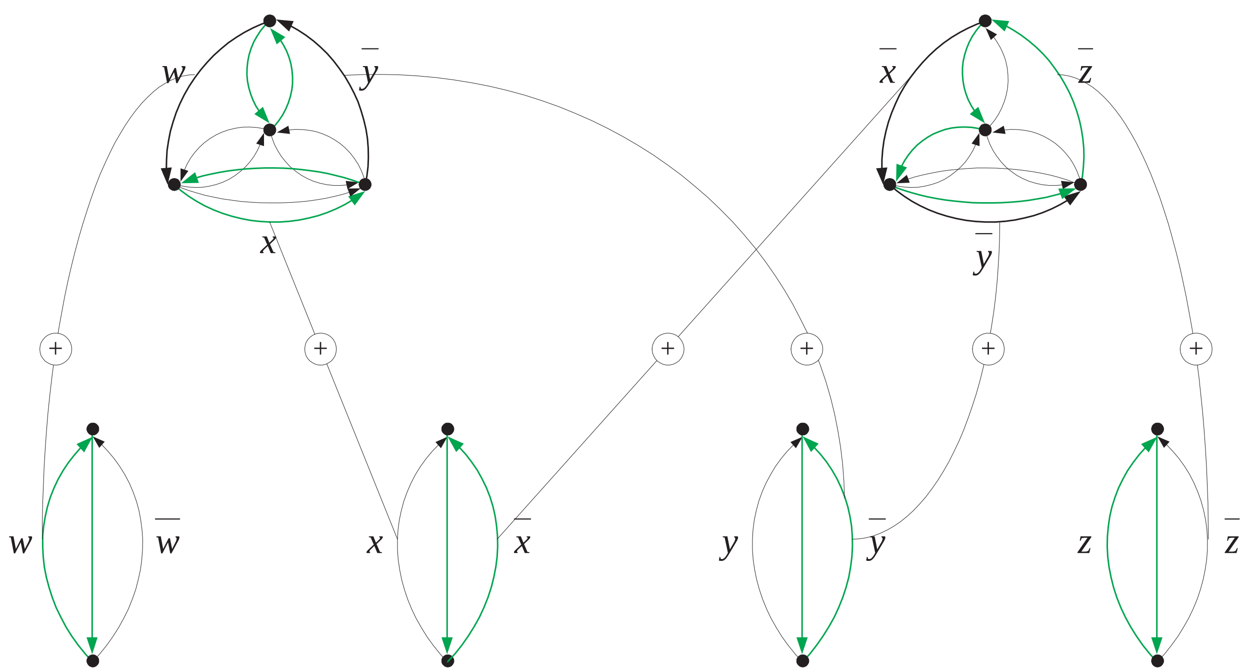
Full Graph $(w \vee x \vee \bar{y}) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})$



The Proof: a Key Observation (continued)

Each satisfying truth assignment to ϕ corresponds to a schematic cycle cover that respects the XOR gadgets.

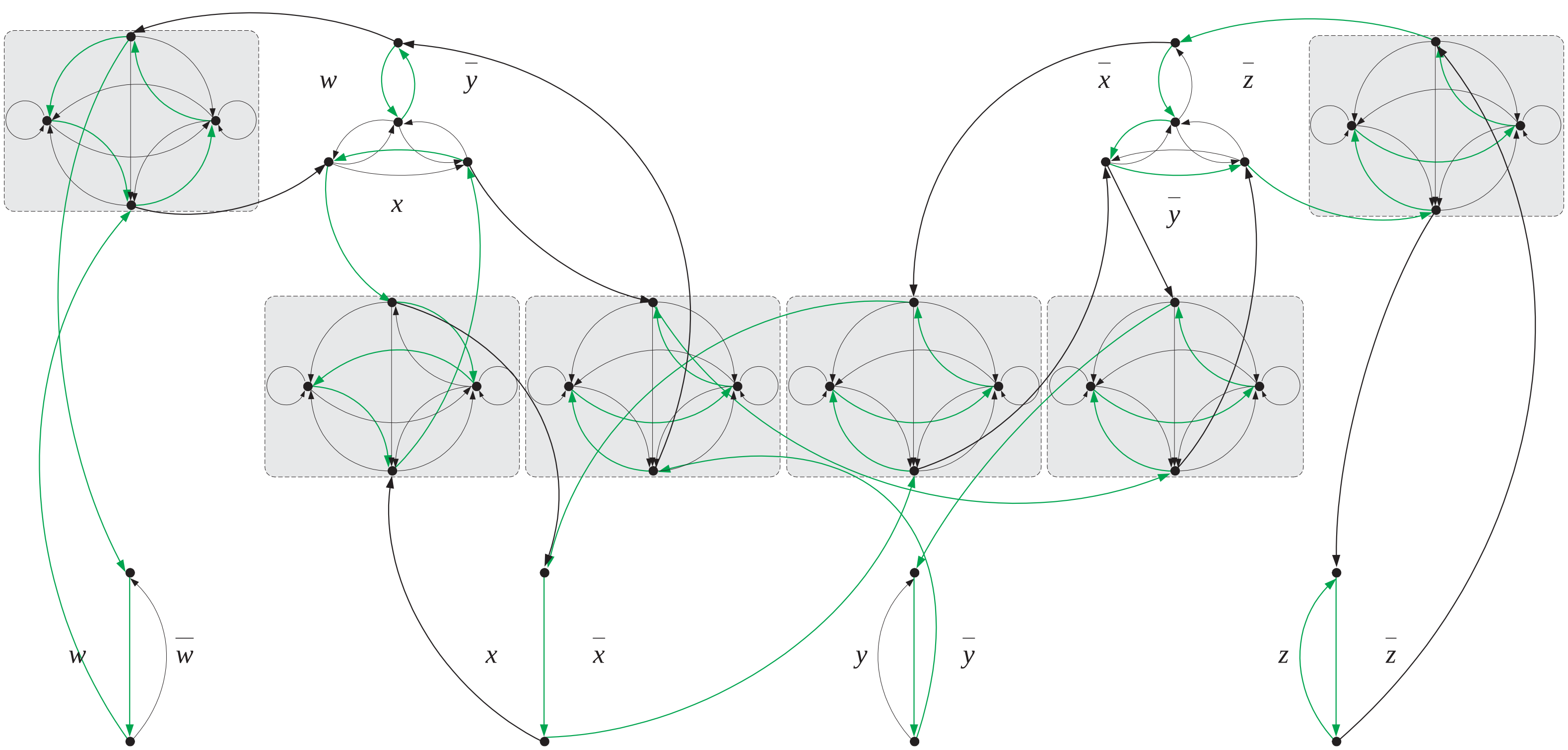
$$w = 1, x = 0, y = 0, z = 1 \Leftrightarrow \text{One Cycle Cover}$$



The Proof: a Key Corollary (continued)

- Recall that there are $3m$ XOR gadgets.
- Each satisfying truth assignment to ϕ contributes 4^{3m} to the cycle count $\#H$.
- Hence
$$\#H = 4^{3m} \times \#\phi,$$
as desired.

"$w = 1, x = 0, y = 0, z = 1$" Adds 4^6 to Cycle Count



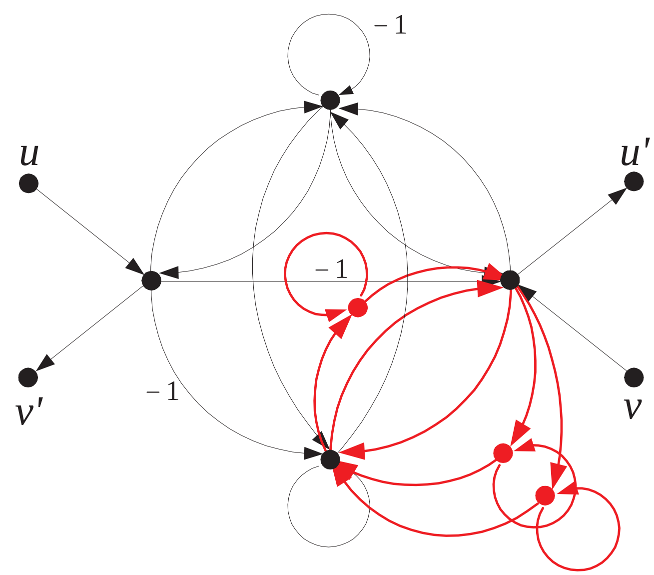
The Proof (continued)

- We are almost done.
- The weighted directed graph H needs to be *efficiently* replaced by some unweighted graph G.
- Furthermore, knowing $\#G$ should enable us to calculate $\#H$ *efficiently*.
  - This done, $\#\phi$ will have been Turing-reducible to $\#G$.[a]
- We proceed to construct this graph G.

[a] By way of $\#H$ of course.

The Proof: Construction of G (continued)

- Replace edges with weights 2 and 3 as follows (note that the graph cannot have parallel edges):



- The cycle count $\#H$ remains *unchanged*.

The Proof: Construction of G (continued)

- We move on to edges with weight -1.
- First, we count the number of nodes, M.
- Each clause gadget contains 4 nodes (p. 641), and there are m of them (one per clause).
- Each XOR gadget contains 7 nodes (p. 660), and there are $3m$ of them (one per literal).
- Each choice gadget contains 2 nodes (p. 652), and there are $n \leq 3m$ of them (one per variable).
- So

$$M \leq 4m + 21m + 6m = 31m.$$

The Proof: Construction of G (continued)

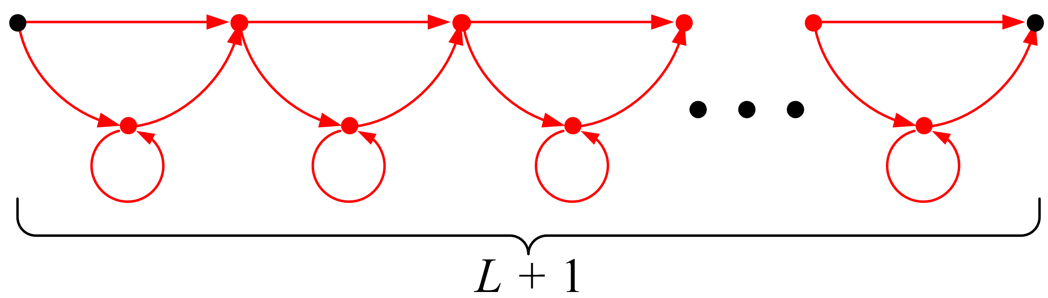
- $\#H \leq 2^L$ for some $L = O(m \log m)$.
  - The maximum absolute value of the edge weight is 1.
  - Hence each term in the permanent is at most 1.
  - There are $M! \leq (31m)!$ terms.
  - Hence

$$\begin{aligned} \#H &\leq \sqrt{2\pi(31m)} \left(\frac{31m}{e}\right)^{31m} e^{\frac{1}{12\times(31m)}} \\ &= 2^{O(m \log m)} \end{aligned} \tag{10}$$

by a refined Stirling's formula.

The Proof: Construction of G (continued)

- Replace each edge with weight -1 with the following:



- Each increases the number of cycle covers 2^{L+1}-fold.
- The desired unweighted G has been obtained.

The Proof (continued)

- $\#G$ equals $\#H$ after replacing each appearance -1 in $\#H$ with 2^{L+1}:

$$
\begin{aligned}
\#H &= \cdots + \overbrace{(-1) \cdot 1 \cdots\cdots 1}^{\text{a cycle cover}} + \cdots\,, \\
\#G &= \cdots + \overbrace{2^{L+1} \cdot 1 \cdots\cdots 1}^{\text{a cycle cover}} + \cdots\,.
\end{aligned}
$$

- Let $\#G = \sum_{i=0}^{n} a_i \times (2^{L+1})^i$, where $0 \leq a_i < 2^{L+1}$.
- As $\#H \leq 2^L$ even if we replace -1 by 1 (p. 662), each a_i equals the number of cycle covers with i edges of weight -1.

The Proof (concluded)

- We conclude that

$$\#H = a_0 - a_1 + a_2 - \cdots + (-1)^n a_n,$$

indeed easily computable from $\#G$.

- We know $\#H = 4^{3m} \times \#\phi$ (p. 657).
- So

$$\#\phi = \frac{a_0 - a_1 + a_2 - \cdots + (-1)^n a_n}{4^{3m}}.$$

  – More succinctly,

$$\#\phi = \frac{\#G \bmod (2^{L+1} + 1)}{4^{3m}}.$$

Polynomial Space

PSPACE and Games

- Given a boolean expression ϕ in CNF with boolean variables $x_1, x_2, \ldots, x_n$, is it true that $\exists x_1 \forall x_2 \cdots Q_n x_n \phi$?
- This is called **quantified satisfiability** or QSAT.
- This problem is like a two-person game: $\exists$ and $\forall$ are the two players.
- We ask then is there a winning strategy for $\exists$?
- QSAT Is PSPACE-Complete[a]

[a]Stockmeyer and Meyer (1973).

IP and PSPACE

- We next prove that coNP $\subseteq$ IP.
- Shamir in 1990 proved that IP equals PSPACE using similar ideas (p. 710).

Interactive Proof for Boolean Unsatisfiability

- Like GRAPH NONISOMORPHISM (p. 538), it is not clear how to construct a short certificate for UNSAT.
- But with interaction and randomization, we shall present an interactive proof for UNSAT.
- A 3SAT formula is a conjunction of disjunctions of at most three literals.
- For any unsatisfiable 3SAT formula $\phi(x_1, x_2, \ldots, x_n)$, there is an interactive proof for the fact that it is unsatisfiable.
- Therefore, coNP $\subseteq$ IP.

Arithmetization of Boolean Formulas

The idea is to arithmetize the boolean formula.

- T $\to$ positive integer
- F $\to 0$
- $x_i \to x_i$
- $\neg x_i \to 1 - x_i$
- $\vee \to +$
- $\wedge \to \times$
- $\phi(x_1, x_2, \ldots, x_n) \to \Phi(x_1, x_2, \ldots, x_n)$

The Arithmetized Version

- A boolean formula is transformed into a multivariate polynomial Φ.
- It is easy to verify that ϕ is unsatisfiable if and only if

$$\sum_{x_1=0,1} \sum_{x_2=0,1} \cdots \sum_{x_n=0,1} \Phi(x_1, x_2, \ldots, x_n) = 0.$$

- But the above seems to require exponential time.
- We turn to more intricate methods.

Choosing the Field

- Suppose ϕ has m clauses of length three each.
- Then $\Phi(x_1, x_2, \ldots, x_n) \leq 3^m$ for any truth assignment $(x_1, x_2, \ldots, x_n)$.
- Because there are at most 2^n truth assignments,

$$\sum_{x_1=0,1} \sum_{x_2=0,1} \cdots \sum_{x_n=0,1} \Phi(x_1, x_2, \ldots, x_n) \leq 2^n 3^m.$$

Choosing the Field (concluded)

- By choosing a prime $q > 2^n 3^m$ and working modulo this prime, proving unsatisfiability reduces to proving that

$$\sum_{x_1=0,1} \sum_{x_2=0,1} \cdots \sum_{x_n=0,1} \Phi(x_1, x_2, \ldots, x_n) \equiv 0 \bmod q. \quad (11)$$

- Working under a *finite* field allows us to uniformly select a random element in the field.

Binding Peggy

- Peggy has to find a sequence of polynomials that satisfy a number of restrictions.
- The restrictions are imposed by Victor: After receiving a polynomial from Peggy, Victor sets a new restriction for the next polynomial in the sequence.
- These restrictions guarantee that if ϕ is unsatisfiable, such a sequence can always be found.
- However, if ϕ is not unsatisfiable, any Peggy has only a small probability of finding such a sequence.
  - The probability is taken over Victor's coin tosses.

The Algorithm

Peggy and Victor both arithmetize ϕ to obtain Φ;
Peggy picks a prime $q > 2^n 3^m$ and sends it to Victor;
Victor rejects and stops if q is not a prime;
Victor sets $v_0 = 0$;
for $i = 1, 2, \ldots, n$ **do**
    Peggy calculates $P_i^*(z) = \sum_{x_{i+1}=0,1} \cdots \sum_{x_n=0,1} \Phi(r_1, \ldots, r_{i-1}, z, x_{i+1}, \ldots, x_n)$;
    Peggy sends $P_i^*(z)$ to Victor;
    Victor rejects and stops if $P_i^*(0) + P_i^*(1) \not\equiv v_{i-1} \bmod q$ or $P_i^*(z)$'s degree exceeds m; $\{P_i^*(z)$ has at most m clauses.$\}$
    Victor uniformly picks $r_i \in Z_q$ and calculates $v_i = P_i^*(r_i)$;
    Victor sends r_i to Peggy;
end for
Victor accepts iff $\Phi(r_1, r_2, \ldots, r_n) \equiv v_n \bmod q$;

Comments

- The following invariant is maintained by the algorithm:

$$P_i^*(0) + P_i^*(1) \equiv P_{i-1}^*(r_{i-1}) \bmod q \tag{12}$$

  for $1 \leq i \leq n$.

  – $P_i^*(0) + P_i^*(1)$ equals $\sum_{x_i=0,1} \cdots \sum_{x_n=0,1} \Phi(r_1, \ldots, r_{i-1}, x_i, x_{i+1}, \ldots, x_n)$ modulo q.

  – The above equals $P_{i-1}^*(r_{i-1}) \bmod q$ by definition.

Comments (concluded)

- The computation of $v_1, v_2, \ldots, v_n$ must rely on Peggy's supplied polynomials as Victor does not have the power to carry out the exponential-time calculations.
- But $\Phi(r_1, r_2, \ldots, r_n)$ in Step 12 is computed without relying on Peggy.

Completeness

- Suppose ϕ is unsatisfiable.
- For $i \geq 1$, by Eq. (12) on p. 688,

$$\begin{aligned}
& P_i^*(0) + P_i^*(1) \\
= & \sum_{x_i=0,1} \sum_{x_{i+1}=0,1} \cdots \sum_{x_n=0,1} \Phi(r_1, \ldots, r_{i-1}, x_i, x_{i+1}, \ldots, x_n) \\
= & P_{i-1}^*(r_{i-1}) \\
\equiv & v_{i-1} \bmod q.
\end{aligned}$$

Completeness (concluded)

- In particular at $i = 1$, because ϕ is unsatisfiable, we have

$$\begin{aligned} P_1^*(0) + P_1^*(1) &= \sum_{x_1=0,1} \cdots \sum_{x_n=0,1} \Phi(x_1, \ldots, x_n) \\ &\equiv v_0 \\ &= 0 \bmod q. \end{aligned}$$

- Finally, $v_n = P_n^*(r_n) = \Phi(r_1, r_2, \ldots, r_n)$.
- Because all the tests by Victor will pass, Victor will accept ϕ.

Soundness

- Suppose ϕ is not unsatisfiable.
- Victor will reject after an honest Peggy sends $P_1^*(z)$.
  - $P_1^*(z) = \sum_{x_2=0,1} \cdots \sum_{x_n=0,1} \Phi(z, x_2, \ldots, x_n)$.
  - So

$$\begin{aligned} & P_1^*(0) + P_1^*(1) \\ = & \sum_{x_1=0,1} \sum_{x_2=0,1} \cdots \sum_{x_n=0,1} \Phi(x_1, x_2, \ldots, x_n) \\ \not\equiv & \; 0 \bmod q \end{aligned}$$

    by Eq. (11) on p. 685.
  - But $v_0 = 0$.

Soundness (continued)

- We will show that if Peggy is dishonest in one round (by sending a polynomial other than $P_i^*(z)$), then with high probability she must be dishonest in the next round, too.
- In the last round (Step 12), her dishonesty is exposed.

Soundness (continued)

- Let $P_i(z)$ represent the polynomial sent by Peggy in place of $P_i^*(z)$.
- Victor calculates $v_i = P_i(r_i) \bmod p$.
- In order to deceive Victor in the next round, round $i+1$, Peggy must use $r_1, r_2, \ldots, r_i$ to find a $P_{i+1}(z)$ of degree at most m such that

$$P_{i+1}(0) + P_{i+1}(1) \equiv v_i \bmod q$$

(see Step 8 of the algorithm on p. 687).

- And so on to the end, except that Peggy has no control over Step 12.

A Key Claim

Lemma 88 *If* $P_i^*(0) + P_i^*(1) \not\equiv v_{i-1} \bmod q$, *then either Victor rejects in the ith round, or* $P_i^*(r_i) \not\equiv v_i \bmod q$ *with probability at least* $1 - (m/q)$, *where the probability is taken over Victor's choices of* r_i.

- Think of $P_i^*(r_i)$ as the v_i that Victor *should be* computing if Peggy were honest.
- But Victor actually calculates $P_i(z)$ as v_i (Peggy claims $P_i(z)$ is $P_i^*(z)$).
- So $v_i = P_i(r_i) \bmod q$.
- What Victor can do is to check for consistencies.

The Proof of Lemma 88 (continued)

- If Peggy sends a $P_i(z)$ which equals $P_i^*(z)$, then

$$P_i(0) + P_i(1) = P_i^*(0) + P_i^*(1) \not\equiv v_{i-1} \bmod q,$$

and Victor rejects immediately.

- Suppose Peggy sends a $P_i(z)$ different from $P_i^*(z)$.
- If $P_i(z)$ does not pass Victor's test

$$P_i(0) + P_i(1) \equiv v_{i-1} \bmod q, \tag{13}$$

then Victor rejects and we are done, too.

The Proof of Lemma 88 (concluded)

- Finally, assume $P_i(z)$ passes the test (13).
- $P_i(z) - P_i^*(z) \not\equiv 0$ is a polynomial of degree at most m.
- Hence equation $P_i(z) - P_i^*(z) \equiv 0 \bmod q$ has at most m roots $r \in Z_q$, i.e.,

$$P_i^*(r) \equiv v_i \bmod q.$$

- Hence Victor will pick one of these as his r_i so that

$$P_i^*(r_i) \equiv v_i \bmod q$$

with probability at most m/q.

Soundness (continued)

- Suppose Victor does not reject in any of the first n rounds.
- As ϕ is not unsatisfiable,

$$P_1^*(0) + P_1^*(1) \not\equiv v_0 \bmod q.$$

- By Lemma 88 (p. 695) and the fact that Victor does not reject, we have $P_1^*(r_1) \not\equiv v_1 \bmod q$ with probability at least $1 - (m/q)$.
- Now by Eq. (12) on p. 688,

$$P_1^*(r_1) = P_2^*(0) + P_2^*(1) \not\equiv v_1 \bmod q.$$

Soundness (concluded)

- Iterating on this procedure, we eventually arrive at

$$P_n^*(r_n) \not\equiv v_n \bmod q$$

with probability at least $(1 - m/q)^n$.

- As $P_n^*(r_n) = \Phi(r_1, r_2, \ldots, r_n)$, Victor's last test at Step 12 fails and he rejects.

- Altogether, Victor rejects with probability at least

$$[\,1 - (m/q)\,]^n > 1 - (nm/q) > 2/3 \tag{14}$$

because $q > 2^n 3^m$.

An Example

- $(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3)$.
- The above is satisfied by assigning true to x_1.
- The arithmetized formula is

  $$\Phi(x_1, x_2, x_3) = (x_1 + x_2 + x_3) \times [\, x_1 + (1 - x_2) + (1 - x_3) \,].$$

- Indeed, $\sum_{x_1=0,1} \sum_{x_2=0,1} \sum_{x_3=0,1} \Phi(x_1, x_2, x_3) = 16 \neq 0$.
- We have $n = 3$ and $m = 2$.
- A prime q that satisfies $q > 2^3 \times 3^2 = 72$ is 73.

An Example (continued)

- The table below is an execution of the algorithm in Z_{73} *when Peggy follows the protocol.*

i	$P_i^*(z)$	$P_i^*(0) + P_i^*(1)$	$= v_{i-1}$?	r_i	v_i
0					0
1	$4z^2 + 8z + 2$	16	no		

- Victor therefore rejects ϕ early on at $i = 1$.

An Example (continued)

- Now suppose Peggy does not follow the protocol.
- In order to deceive Victor, she comes up with fake polynomials $P_i(z)$ from $i = 1$.
- The table below is an execution of the algorithm.

i	$P_i(z)$	$P_i(0) + P_i(1)$	$= v_{i-1}$?	r_i	v_i
0					0
1	$8z^2 + 11z + 27$	0	yes	2	35
2	$z^2 + 8z + 13$	35	yes	3	46
3	$3z^2 + z + 21$	46	yes	r_3	$P_3(r_3)$

An Example (concluded)

- Victor has been satisfied up to round 3.
- Finally at Step 12, Victor checks if

$$\Phi(2, 3, r_3) \equiv P_3(r_3) \bmod 73.$$

- It can be verified that the only choices of $r_3 \in \{0, 1, \ldots, 72\}$ that can mislead Victor are 31 and 59.
- The probability of that happening is only 2/73.[a]

[a]Ms. Ching-Ju Lin (R92922038) on January 7, 2004, pointed out an error in an earlier calculation.

An Example

- $(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$.
- The above is unsatisfiable.
- The arithmetized formula is

$$\Phi(x_1, x_2) = (x_1 + x_2) \times (x_1 + 1 - x_2) \times (1 - x_1 + x_2) \times (2 - x_1 - x_2).$$

- Because $\Phi(x_1, x_2) = 0$ for any *boolean* assignment $\{0, 1\}^2$ to (x_1, x_2), certainly

$$\sum_{x_1 = 0,1} \sum_{x_2 = 0,1} \Phi(x_1, x_2) = 0.$$

- With $n = 2$ and $m = 4$, a prime q that satisfies $q > 2^2 \times 3^4 = 4 \times 81 = 324$ is 331.

An Example (concluded)

- The table below is an execution of the algorithm in Z_{331}.

i	$P_i^*(z)$	$P_i^*(0) + P_i^*(1)$	$= v_{i-1}$?	r_i	v_i
0					0
1	$z(z+1)(1-z)(2-z)$	0	yes	10	283
	$+(z+1)z(2-z)(1-z)$				
2	$(10+z) \times (11-z)$	283	yes	5	46
	$\times(-9+z) \times (-8-z)$				

- Victor calculates $\Phi(10, 5) \equiv 46 \bmod 331$.
- As it equals $v_2 = 46$, Victor accepts ϕ as unsatisfiable.

Objections to the Soundness Proof?[a]

- Based on the steps required of a cheating Peggy on p. 694, why must we go through so many rounds (in fact, n rounds)?
- Why not just go directly to round n:
  - Victor sends $r_1, r_2, \ldots, r_{n-1}$ to Peggy.
  - Peggy returns with a (claimed) $P_n^*(z)$.
  - Victor accepts if and only if $\Phi(r_1, r_2, \ldots, r_{n-1}, r_n) \equiv P_n^*(r_n) \bmod q$ for a random $r_n \in Z_q$.

[a]Contributed by Ms. Emily Hou (D89011) and Mr. Pai-Hsuen Chen (R90008) on January 2, 2002.

Objections to the Soundness Proof? (continued)

- Let us analyze the compressed proposal when ϕ is satisfiable.
- To succeed in foiling Victor, Peggy must find a polynomial $P_n(z)$ of degree m such that

$$\Phi(r_1, r_2, \ldots, r_{n-1}, z) \equiv P_n(z) \bmod q.$$

- But this she is able to do: Just give the verifier the polynomial $\Phi(r_1, r_2, \ldots, r_{n-1}, z)$!
- What has happened?

Objections to the Soundness Proof? (concluded)

- You need the intermediate rounds to "tie" Peggy up with a chain of claims.
- In the original algorithm on p. 687, for example, $P_n(z)$ is bound by the equality $P_n(0) + P_n(1) \equiv v_{n-1} \bmod q$ in Step 8.
- That v_{n-1} is in turn derived by an earlier polynomial $P_{n-1}(z)$, which is in turn bound by $P_{n-1}(0) + P_{n-1}(1) \equiv v_{n-2} \bmod q$, and so on.

Shamir's Theorem[a]

Theorem 89 *IP = PSPACE.*

- We first sketch the proof for IP $\subseteq$ PSPACE.
- Without loss of generality, assume:
  - If $x \in L$, then the probability that x is accepted by the verifier is at least $3/4$.
  - If $x \notin L$, then the probability that x is accepted by the verifier with *any* prover is at most $1/4$.

[a]Shamir (1990).

The Proof (continued)

- Now we track down every possible message exchange based on random choices by the verifier and all possible messages generated by the prover.
- Use recursion to calculate

$$\text{prob}[\,\text{verifier accepts } x\,] = \max_P \text{prob}[\,(V, P) \text{ accepts } x\,].$$

- If this value is at least $3/4$, then $x \in L$; otherwise, $x \notin L$.

The Proof (continued)

- To prove PSPACE $\subseteq$ IP, we next prove that QSAT is in IP.
- We do so by describing an interactive protocol that decides QSAT.
- Suppose Alice and Bob are given

$$\begin{aligned}\phi &= \forall x \exists y (x \vee y) \wedge \forall z [\, (x \wedge z) \vee (y \wedge \neg z)\,] \\ &\quad \vee \exists w [\, z \vee (y \wedge \neg w)\,].\end{aligned}$$

- As above, we assume no occurrence of a variable is separated by more than one $\forall$ from its point of quantification.

Proof of Theorem (continued)

- We also assume that $\neg$ is applied only to variables, not subexpressions.
- We now arithmetize ϕ as before except:
  - 1 means true.
  - $\neg x \to 1 - x$.
    - It is the standard representation on p. 134.
  - $\exists x \to \sum_{x=0,1}$.
  - $\forall x \to \prod_{x=0,1}$.
- Alice tries to convince Bob that this arithmetization of ϕ is nonzero.

Proof of Theorem (continued)

- Our ϕ becomes

$$\begin{aligned} A_\phi &= \prod_{x=0}^{1} \sum_{y=0}^{1} \{(x+y) \cdot \prod_{z=0}^{1} [(x \cdot z + y \cdot (1-z)) \\ &\quad + \sum_{w=0}^{1} (z + y \cdot (1-w))]\}. \end{aligned}$$

- Call it a $\sum - \prod$ expression.
- A_ϕ is a number; it equals 96 here.

Proof of Theorem (continued)

- As before, ϕ is true if and only if $A_\phi > 0$.
- In fact, more is true.
- For any ϕ and any truth assignment to its free variables:
  - If ϕ is true, then $A_\phi > 0$ under the corresponding 0-1 assignment.
  - If ϕ is false, then $A_\phi = 0$.
- So Alice only has to convince Bob that $A_\phi > 0$.

Proof of Theorem (continued)

- The trouble is that A_ϕ evaluated can be exponential in length.
- Fortunately, it can be shown that if expression A_ϕ of length n is nonzero, then there is a prime p between 2^n and 2^{3n} such that $A_\phi \neq 0 \bmod p$.
- So Alice only has to convince Bob that $A_\phi \neq 0$ under mod$p$.
- The protocol starts with Alice sending Bob p (assume $p = 13$) and its primality certificate.

Proof of Theorem (continued)

- Now Alice sends Bob $A_\phi \bmod p$, which is

$$\boldsymbol{a} = 96 \bmod 13 = 5.$$

- Each stage starts with the following:
  - A $\sum - \prod$ expression A, with a leading $\sum_x$ or $\prod_x$.
  - A's alleged value $a \bmod p$, supplied by Alice.
- If the first $\sum$ or $\prod$ is deleted, the result is a polynomial in x, called $A'(x)$.
- Bob demands from Alice the coefficients of $A'(x)$.
- Trouble occurs if the degree of $A'(x)$ is exponential in n.

Proof of Theorem (continued)

- Luckily, $\deg(A'(x)) \leq 2n$.
  - No occurrence of a variable is separated by more than one $\forall$ from its point of quantification.
  - So $A'(x)$ has only one $\prod$ symbol.
  - Other $\prod$s are over quantities not related to x, hence purely numerical.
  - Symbols other than $\prod$ can only increase the degree of $A'(x)$ by at most one.
  - For example, $\sum_y (x + y) \prod_z (x + \sum_w (x \cdot w))$.
- So Alice has no problem transmitting $A'(x)$ to Bob.

Proof of Theorem (continued)

- $A'(x) = 2x^2 + 8x + 6$.
- Bob verifies that $A'(0) \cdot A'(1) = 5 \bmod 13$.
- Indeed $A'(0) \cdot A'(1) = 6 \cdot 16 = 5 \bmod 13$.
- So far $A'(x)$ is consistent with the alleged value 5.
- Bob deletes the leading $\prod_x$.
- The *free variable* x in the resulting expression prevents it from being an evaluation problem.

Proof of Theorem (continued)

- So Bob replaces x with a random number $\bmod 13$, say 9:

$$\sum_{y=0}^{1} \left\{ (9+y) \cdot \prod_{z=0}^{1} \left[(9 \cdot z + y \cdot (1-z)) + \sum_{w=0}^{1} (z + y \cdot (1-w)) \right] \right\}.$$

- The above equals

$$\boldsymbol{a} = A'(9) = 2 \cdot 9^2 + 8 \cdot 9 + 6 = 6 \bmod 13.$$

- Bob sends 9 to Alice.

Proof of Theorem (continued)

- In the new stage, Alice evaluates

$$A'(y) = 2y^3 + y^2 + 3y$$

  after substituting $x = 9$ and sends it to Bob.
- Bob checks that $A'(0) + A'(1) = 6 \bmod 13$.
- Indeed $0 + 6 = 6 \bmod 13$.
- Bob deletes the leading $\sum_y$.
- Bob replaces y with a random number mod13, say 3:

$$(9+3) \cdot \prod_{z=0}^{1} \left\{ [\, 9 \cdot z + 3 \cdot (1 - z) \,] + \sum_{w=0}^{1} [\, z + 3 \cdot (1 - w) \,] \right\}.$$

Proof of Theorem (continued)

- The above should equal
  $A'(3) = 2 \cdot 3^2 + 3^2 + 3 \cdot 3 = 7 \bmod 13.$
- So
  $$A = \prod_{z=0}^{1} \{ [\, 9 \cdot z + 3 \cdot (1 - z) \,] + \sum_{w=0}^{1} [\, z + 3 \cdot (1 - w) \,] \}$$
  should equal
  $$\boldsymbol{a} = 12^{-1} \cdot 7 = 12 \cdot 7 = 6 \bmod 13.$$
- Bob sends 3 to Alice.

Proof of Theorem (continued)

- In the new stage, Alice evaluates

$$A'(z) = 8z + 6$$

  after substituting $y = 3$ and sends it to Bob.

- Bob checks that $A'(0) \cdot A'(1) = 6 \bmod 13$.
- Indeed $6 \cdot 14 = 6 \bmod 13$.
- Bob deletes the leading $\prod_z$.
- Bob replaces z with a random number mod13, say 7:

$$[\,9 \cdot 7 + 3 \cdot (1 - 7)\,] + \sum_{w=0}^{1} [\,7 + 3 \cdot (1 - w)\,].$$

Proof of Theorem (continued)

- The above should equal $A'(7) = 8 \cdot 7 + 6 = 10 \bmod 13$.
- So
$$A = \sum_{w=0}^{1} [\, z + 3 \cdot (1 - w) \,] \tag{15}$$
should equal
$$\boldsymbol{a} = 10 - [\, 9 \cdot 7 + 3 \cdot (1 - 7) \,] = 10 - 45 = 4 \bmod 13.$$
- Bob sends 7 to Alice.

Proof of Theorem (continued)

- In the new stage, Alice evaluates
$$A'(w) = 10 - 3w$$
after substituting $z = 7$ and sends it to Bob.
- Bob checks that $A'(0) + A'(1) = 4 \bmod 13$.
- Indeed $10 + 7 = 4 \bmod 13$.
- Now there are no more $\sum$s and $\prod$s.
- Bob checks if $A'(w)$ is indeed as claimed by using (15) with $z = 7$.
- It is, and Bob accepts $A_\phi \neq 0 \bmod 13$.

Proof of Theorem (continued)

- Clearly, if $A_\phi > 0$, the protocol convinces Bob of this.
- We next show that if $A_\phi = 0$, then Bob will be cheated with only negligible probability.

Lemma 90 *Suppose $A_\phi = 0$ and Alice claims a nonzero value $\boldsymbol{a}$. Then with probability $\geq (1 - \frac{2n}{2^n})^{i-1}$, the value of $\boldsymbol{a}$ claimed at the ith stage is wrong.*

Proof of Lemma 90 (continued)

- The first $\boldsymbol{a}$ claimed by Alice is nonzero, which is certainly wrong.
- The lemma therefore holds for $i = 1$.
- By induction, for $i > 1$, the $(i-1)$st value was wrong with probability $\geq (1 - \frac{2n}{2^n})^{i-2}$.
- Suppose it is indeed wrong.
- The polynomial $A'(x)$ produced by Alice in the ith stage must be such that $A'(0) \cdot A'(1)$ or $A'(0) + A'(1)$ equals the wrong value $\boldsymbol{a}$.

Proof of Lemma 90 (continued)

- Alice must therefore supply a wrong polynomial $A'(x)$, different from the true polynomial $C(x)$.
  - Recall that Bob uses $A'(x)$ not $C(x)$.
- $C(x) - A'(x)$ is a polynomial of degree $2n$.
- Hence it has at most $2n$ roots.
- The random number between 0 and $p - 1$ picked by Bob will be one of these roots with probability at most $2n/p$.

Proof of Lemma 90 (concluded)

- The probability that $\boldsymbol{a}$ at the ith stage is *correct* is

$$
\begin{aligned}
&\leq \left[1 - \left(1 - \frac{2n}{2^n}\right)^{i-2}\right]\left(1 - \frac{2n}{p}\right) \\
&\leq 1 - \left(1 - \frac{2n}{2^n}\right)^{i-2}\left(1 - \frac{2n}{p}\right) \\
&\leq 1 - \left(1 - \frac{2n}{2^n}\right)^{i-1}.
\end{aligned}
$$

  - Recall that $p \geq 2^n$.

Proof of Theorem (concluded)

- In the last round, Bob will catch Alice's deception with probability $(1 - \frac{2n}{2^n})^n \to 1$.
- To achieve the confidence level of $1 - 2^{-n}$ required by the definition of IP, simply repeat the protocol.

The Algorithm

Alice and Bob both arithmetize ϕ to obtain Φ;
Alice picks a prime p and sends it to Bob;
Bob rejects if p does not satisfy the desired conditions;
Alice claims $A_{\phi} = \boldsymbol{a} \bmod p$ to Bob;
Bob set $A = A_{\phi}$;
repeat
    Alice sends $A'(x)$ to Bob;
    Bob rejects if $\boldsymbol{a} \neq A'(0) \cdot A'(1) \bmod p$ when $A = \prod_x \cdots$ or $\boldsymbol{a} \neq A'(0) + A'(1) \bmod p$ when $A = \sum_x \cdots$;
    Bob picks a random number r and sends it to Alice;
    Bob calculates $\boldsymbol{a} = A'(r)$;
    Alice and Bob both set $A = A'(r)$; {Some details left out.}
until there no $\prod$ or $\sum$ left in A
Bob accepts iff $A'(x)$ is as claimed in the last stage;

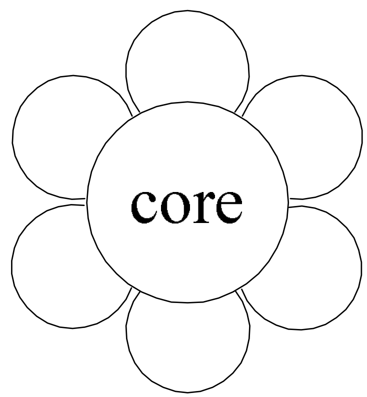
Exponential Circuit Complexity

- Almost all boolean functions require $\frac{2^n}{2n}$ gates to compute (generalized Theorem 14 on p. 153).
- Progress of using circuit complexity to prove exponential lower bounds for NP-complete problems has been slow.
  - As of January 2006, the best lower bound is $5n - o(n)$.[a]
- We next establish exponential lower bounds for depth-3 circuits.

[a] Iwama and Morizumi (2002).

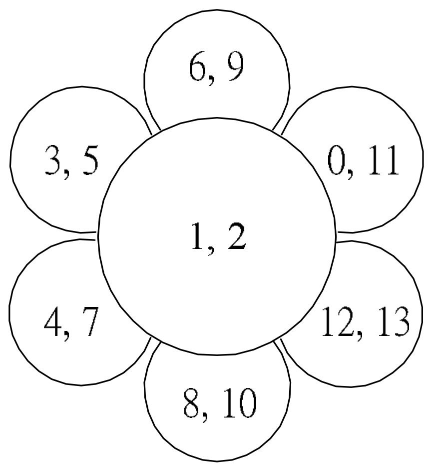
Sunflowers

- Fix $p \in \mathbb{Z}^+$ and $\ell \in \mathbb{Z}^+$.
- A **sunflower** is a family of p sets $\{P_1, P_2, \ldots, P_p\}$, called **petals**, each of cardinality at most ℓ.
- All pairs of sets in the family must have the same intersection (called the **core** of the sunflower).



A Sample Sunflower

$$\{\{1,2,3,5\},\{1,2,6,9\},\{0,1,2,11\},$$
$$\{1,2,12,13\},\{1,2,8,10\},\{1,2,4,7\}\}$$



The Erdős-Rado Lemma

Lemma 91 *Let $\mathcal{Z}$ be a family of more than $M = (p-1)^{\ell}\ell!$ nonempty sets, each of cardinality ℓ or less. Then $\mathcal{Z}$ must contain a sunflower.*

- Induction on ℓ.
- For $\ell = 1$, p different singletons form a sunflower (with an empty core).
- Suppose $\ell > 1$.
- Consider a *maximal* subset $\mathcal{D} \subseteq \mathcal{Z}$ of *disjoint* sets.
  - Every set in $\mathcal{Z} - \mathcal{D}$ intersects some set in $\mathcal{D}$.

The Proof of the Erdős-Rado Lemma (continued)

- Suppose $\mathcal{D}$ contains at least p sets.
  - $\mathcal{D}$ constitutes a sunflower with an empty core.
- Suppose $\mathcal{D}$ contains fewer than p sets.
  - Let D be the union of all sets in $\mathcal{D}$.
  - $|D| \leq (p-1)\ell$ and D intersects every set in $\mathcal{Z}$.
  - There is a $d \in D$ that intersects more than $\frac{M}{(p-1)\ell} = (p-1)^{\ell-1}(\ell-1)!$ sets in $\mathcal{Z}$.
  - Consider $\mathcal{Z}' = \{Z - \{d\} : Z \in \mathcal{Z}, d \in Z\}$.
  - $\mathcal{Z}'$ has more than $M' = (p-1)^{\ell-1}(\ell-1)!$ sets.
  - M' is just M with ℓ decreased by one.

The Proof of the Erdős-Rado Lemma (concluded)

- (continued)
  - $\mathcal{Z}'$ contains a sunflower by induction, say
$$\{P_1, P_2, \ldots, P_p\}.$$
  - Now,
$$\{P_1 \cup \{d\}, P_2 \cup \{d\}, \ldots, P_p \cup \{d\}\}$$
is a sunflower in $\mathcal{Z}$.

Comments on the Erdős-Rado Lemma

- A family of more than M sets must contain a sunflower.
- **Plucking** a sunflower entails replacing the sets in the sunflower by its core.
- By repeatedly finding a sunflower and plucking it, we can reduce a family with more than M sets to a family with at most M sets.
- If $\mathcal{Z}$ is a family of sets, the above result is denoted by pluck($\mathcal{Z}$).

An Example of Plucking

- Recall the sunflower on p. 733:

$$\begin{aligned}\mathcal{Z} &= \{\{1,2,3,5\},\{1,2,6,9\},\{0,1,2,11\},\\ &\quad \{1,2,12,13\},\{1,2,8,10\},\{1,2,4,7\}\}\end{aligned}$$

- Then

$$\text{pluck}(\mathcal{Z}) = \{\{1,2\}\}.$$

Exponential Circuit Complexity for NP-Complete Problems

- We shall prove exponential lower bounds for NP-complete problems using *monotone* circuits.
  - Monotone circuits are circuits without $\neg$ gates.
- Note that this does not settle the P vs. NP problem or any of the conjectures on p. 489.

The Power of Monotone Circuits

- Monotone circuits can only compute monotone boolean functions.
- They are powerful enough to solve a P-complete problem, MONOTONE CIRCUIT VALUE (p. 241).
- There are NP-complete problems that are not monotone; they cannot be computed by monotone circuits at all.
- There are NP-complete problems that are monotone; they can be computed by monotone circuits.
  - HAMILTONIAN PATH and CLIQUE.

$\text{CLIQUE}_{n,k}$

- $\text{CLIQUE}_{n,k}$ is the boolean function deciding whether a graph $G = (V, E)$ with n nodes has a clique of size k.
- The input gates are the $\binom{n}{2}$ entries of the adjacency matrix of G.
  - Gate g_{ij} is set to true if the associated undirected edge $\{ i, j \}$ exists.
- $\text{CLIQUE}_{n,k}$ is a monotone function.
- Thus it can be computed by a monotone circuit.
- This does not rule out that nonmonotone circuits for $\text{CLIQUE}_{n,k}$ may use fewer gates.

Crude Circuits

- One possible circuit for $\text{CLIQUE}_{n,k}$ does the following.
  1. For each $S \subseteq V$ with $|S| = k$, there is a subcircuit with $O(k^2)$ $\wedge$-gates testing whether S forms a clique.
  2. We then take an OR of the outcomes of all the $\binom{n}{k}$ subsets $S_1, S_2, \ldots, S_{\binom{n}{k}}$.
- This is a monotone circuit with $O(k^2 \binom{n}{k})$ gates, which is exponentially large unless k or $n - k$ is a constant.
- A **crude circuit** $\mathrm{CC}(X_1, X_2, \ldots, X_m)$ tests if *any* of $X_i \subseteq V$ forms a clique.
  - The above-mentioned circuit is $\mathrm{CC}(S_1, S_2, \ldots, S_{\binom{n}{k}})$.

Razborov's Theorem

Theorem 92 (Razborov (1985)) *There is a constant c such that for large enough n, all monotone circuits for* $\text{CLIQUE}_{n,k}$ *with $k = n^{1/4}$ have size at least $n^{cn^{1/8}}$.*

- We shall approximate any monotone circuit for $\text{CLIQUE}_{n,k}$ by a restricted kind of crude circuit.
- The approximation will proceed in steps: one step for each gate of the monotone circuit.
- Each step introduces few errors (false positives and false negatives).
- But the resulting crude circuit has exponentially many errors.

The Proof

- Fix $k = n^{1/4}$.
- Fix $\ell = n^{1/8}$.
- Note that
$$2\binom{\ell}{2} \leq k.$$
- p will be fixed later to be $n^{1/8} \log n$.
- Fix $M = (p-1)^{\ell} \ell!$.
  - Recall the Erdős-Rado lemma (p. 734).

The Proof (continued)

- Each crude circuit used in the approximation process is of the form $\mathrm{CC}(X_1, X_2, \ldots, X_m)$, where:
  - $X_i \subseteq V$.
  - $|X_i| \leq \ell$.
  - $m \leq M$.
- We shall show how to approximate any circuit for $\textsc{clique}_{n,k}$ by such a crude circuit, inductively.
- The induction basis is straightforward:
  - Input gate g_{ij} is the crude circuit $\mathrm{CC}(\{i, j\})$.

The Proof (continued)

- Any monotone circuit can be considered the OR or AND of two subcircuits.
- We shall show how to build approximators of the overall circuit from the approximators of the two subcircuits.
  - We are given two crude circuits $\mathrm{CC}(\mathcal{X})$ and $\mathrm{CC}(\mathcal{Y})$.
  - $\mathcal{X}$ and $\mathcal{Y}$ are two families of at most M sets of nodes, each set containing at most ℓ nodes.
  - We construct the approximate OR and the approximate AND of these subcircuits.
  - Then show both approximations introduce few errors.

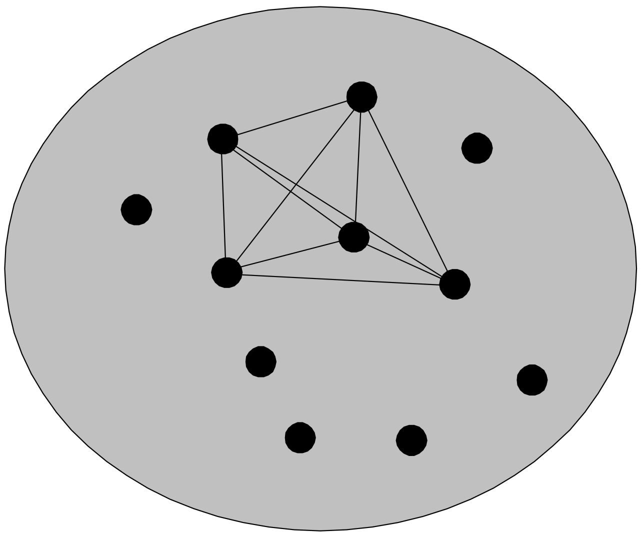
The Proof: Positive Examples

- Error analysis will be applied to only **positive examples** and **negative examples**.
- A positive example is a graph that has $\binom{k}{2}$ edges connecting k nodes in all possible ways.
- There are $\binom{n}{k}$ such graphs.
- They all should elicit a true output from $\text{CLIQUE}_{n,k}$.

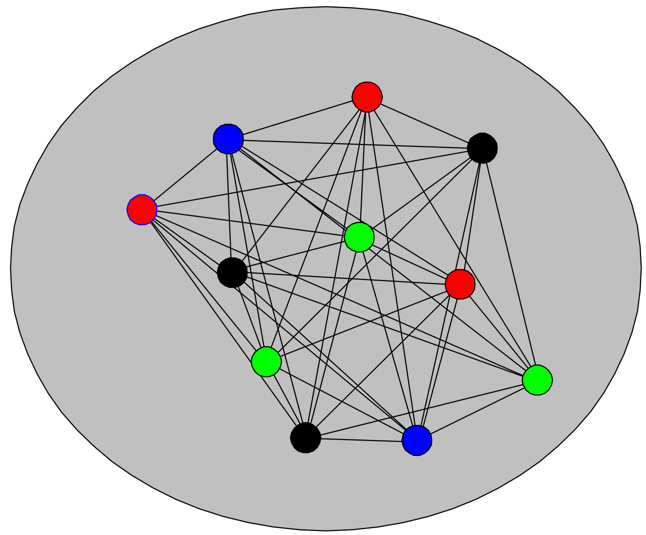
The Proof: Negative Examples

- Color the nodes with $k-1$ different colors and join by an edge any two nodes that are colored differently.
- There are $(k-1)^n$ such graphs.
- They all should elicit a false output from $\text{CLIQUE}_{n,k}$.

Positive and Negative Examples with $k = 5$



A positive example



A negative example

The Proof: OR

- $\mathrm{CC}(\mathcal{X} \cup \mathcal{Y})$ is *equivalent to* the OR of $\mathrm{CC}(\mathcal{X})$ and $\mathrm{CC}(\mathcal{Y})$.
- Violations occur when $|\mathcal{X} \cup \mathcal{Y}| > M$.
- Such violations can be eliminated by using

$$\mathrm{CC}(\mathrm{pluck}(\mathcal{X} \cup \mathcal{Y}))$$

  as the approximate OR of $\mathrm{CC}(\mathcal{X})$ and $\mathrm{CC}(\mathcal{Y})$.
- We now count the numbers of errors this approximate OR makes on the positive and negative examples.

The Proof: OR (concluded)

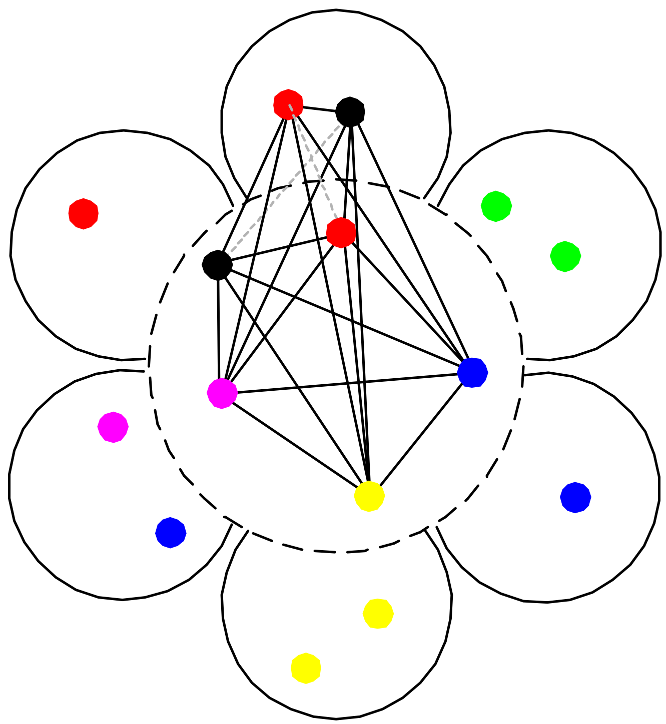
- CC(pluck($\mathcal{X} \cup \mathcal{Y}$)) *introduces* a **false positive** if a negative example makes both CC($\mathcal{X}$) and CC($\mathcal{Y}$) return false but makes CC(pluck($\mathcal{X} \cup \mathcal{Y}$)) return true.
- CC(pluck($\mathcal{X} \cup \mathcal{Y}$)) *introduces* a **false negative** if a positive example makes either CC($\mathcal{X}$) or CC($\mathcal{Y}$) return true but makes CC(pluck($\mathcal{X} \cup \mathcal{Y}$)) return false.
- How many false positives and false negatives are introduced by CC(pluck($\mathcal{X} \cup \mathcal{Y}$))?

The Number of False Positives

Lemma 93 CC(pluck($\mathcal{X} \cup \mathcal{Y}$)) *introduces at most* $\frac{M}{p-1} 2^{-p}(k-1)^n$ *false positives.*

- Assume a plucking replaces the sunflower $\{Z_1, Z_2, \ldots, Z_p\}$ with its core Z.
- A false positive is *necessarily* a coloring such that:
  - There is a pair of identically colored nodes in each petal Z_i (and so both crude circuits return false).
  - But the core contains distinctly colored nodes.
    - ∗ This implies at least one node from each same-color pair was plucked away.
- We now count the number of such colorings.

Proof of Lemma 93 (continued)



Proof of Lemma 93 (continued)

- Color nodes V at random with $k-1$ colors and let $R(X)$ denote the event that there are repeated colors in set X.
- Now $\text{prob}[R(Z_1) \wedge \cdots \wedge R(Z_p) \wedge \neg R(Z)]$ is at most

$$
\begin{aligned}
& \text{prob}[R(Z_1) \wedge \cdots \wedge R(Z_p) | \neg R(Z)] \\
= \quad & \prod_{i=1}^{p} \text{prob}[R(Z_i) | \neg R(Z)] \leq \prod_{i=1}^{p} \text{prob}[R(Z_i)]. \quad (16)
\end{aligned}
$$

  - First equality holds because $R(Z_i)$ are independent given $\neg R(Z)$ as Z contains their only common nodes.
  - Last inequality holds as the likelihood of repetitions in Z_i decreases given no repetitions in $Z \subseteq Z_i$.

Proof of Lemma 93 (continued)

- Consider two nodes in Z_i.
- The probability that they have identical color is $\frac{1}{k-1}$.
- Now $\text{prob}[\, R(Z_i)\,] \leq \frac{\binom{|Z_i|}{2}}{k-1} \leq \frac{\binom{\ell}{2}}{k-1} \leq \frac{1}{2}$.
- So the probability[a] that a random coloring is a new false positive is at most 2^{-p} by inequality (16).
- As there are $(k-1)^n$ different colorings, each plucking introduces at most $2^{-p}(k-1)^n$ false positives.

[a]Proportion, i.e.

Proof of Lemma 93 (concluded)

- Recall that $|\,\mathcal{X} \cup \mathcal{Y}\,| \leq 2M$.
- Each plucking reduces the number of sets by $p - 1$.
- Hence at most $\frac{M}{p-1}$ pluckings occur in pluck$(\mathcal{X} \cup \mathcal{Y})$.
- At most
$$\frac{M}{p-1}\, 2^{-p}(k-1)^n$$
false positives are introduced.

The Number of False Negatives

Lemma 94 CC(pluck($\mathcal{X} \cup \mathcal{Y}$)) *introduces no false negatives.*

- Each plucking replaces a set in a crude circuit by a subset.
- This makes the test less stringent.
  - For each $Y \in \mathcal{X} \cup \mathcal{Y}$, there must exist at least one $X \in \text{pluck}(\mathcal{X} \cup \mathcal{Y})$ such that $X \subseteq Y$.
  - So if $Y \in \mathcal{X} \cup \mathcal{Y}$ is a clique, then pluck($\mathcal{X} \cup \mathcal{Y}$) also contains a clique, in X.
- So plucking can only increase the number of accepted graphs.

The Proof: AND

- The approximate AND of crude circuits $\mathrm{CC}(\mathcal{X})$ and $\mathrm{CC}(\mathcal{Y})$ is

$$\mathrm{CC}(\mathrm{pluck}(\{X_i \cup Y_j : X_i \in \mathcal{X}, Y_j \in \mathcal{Y}, |X_i \cup Y_j| \leq \ell\})).$$

- We now count the numbers of errors this approximate AND makes on the positive and negative examples.

The Proof: AND (concluded)

- The approximate AND *introduces* a **false positive** if a negative example makes either $\mathrm{CC}(\mathcal{X})$ or $\mathrm{CC}(\mathcal{Y})$ return false but makes the approximate AND return true.
- The approximate AND *introduces* a **false negative** if a positive example makes both $\mathrm{CC}(\mathcal{X})$ and $\mathrm{CC}(\mathcal{Y})$ return true but makes the approximate AND return false.
- How many false positives and false negatives are introduced by the approximate AND?

The Number of False Positives

Lemma 95 *The approximate* AND *introduces at most* $M^2 2^{-p}(k-1)^n$ *false positives.*

- $\text{CC}(\{X_i \cup Y_j : X_i \in \mathcal{X}, Y_j \in \mathcal{Y}\})$ introduces no false positives.
  - If $X_i \cup Y_j$ is a clique, both X_i and Y_j must be cliques, making both $\text{CC}(\mathcal{X})$ and $\text{CC}(\mathcal{Y})$ return true.
- $\text{CC}(\{X_i \cup Y_j : X_i \in \mathcal{X}, Y_j \in \mathcal{Y}, |X_i \cup Y_j| \leq \ell\})$ introduces no false positives for the same reason as above.

Proof of Lemma 95 (concluded)

- $|\{X_i \cup Y_j : X_i \in \mathcal{X}, Y_j \in \mathcal{Y}, |X_i \cup Y_j| \leq \ell\}| \leq M^2$.
- Each plucking reduces the number of sets by $p-1$.
- So pluck$(\{X_i \cup Y_j : X_i \in \mathcal{X}, Y_j \in \mathcal{Y}, |X_i \cup Y_j| \leq \ell\})$ involves $\leq M^2/(p-1)$ pluckings.
- Each plucking introduces at most $2^{-p}(k-1)^n$ false positives by the proof of Lemma 93 (p. 752).
- The desired upper bound is

$$[\,M^2/(p-1)\,]\,2^{-p}(k-1)^n \leq M^2 2^{-p}(k-1)^n.$$

The Number of False Negatives

Lemma 96 *The approximate* AND *introduces at most* $M^2\binom{n-\ell-1}{k-\ell-1}$ *false negatives.*

- We follow the same three-step proof as before.
- $\mathrm{CC}(\{X_i \cup Y_j : X_i \in \mathcal{X}, Y_j \in \mathcal{Y}\})$ introduces no false negatives.
  - Suppose both $\mathrm{CC}(\mathcal{X})$ and $\mathrm{CC}(\mathcal{Y})$ accept a positive example with a clique of size k.
  - The clique must contain an $X_i \in \mathcal{X}$ and a $Y_j \in \mathcal{Y}$.
  - As it contains $X_i \cup Y_j$, the new circuit returns true.

Proof of Lemma 96 (concluded)

- $\mathrm{CC}(\{X_i \cup Y_j : X_i \in \mathcal{X}, Y_j \in \mathcal{Y}, |X_i \cup Y_j| \le \ell\})$ introduces $\le M^2\binom{n-\ell-1}{k-\ell-1}$ false negatives.
  - Deletion of set $Z = X_i \cup Y_j$ larger than ℓ introduces false negatives which are cliques containing Z.
  - There are $\binom{n-|Z|}{k-|Z|}$ such cliques.
  - $\binom{n-|Z|}{k-|Z|} \le \binom{n-\ell-1}{k-\ell-1}$ as $|Z| > \ell$.
  - There are at most M^2 such Zs.
- Plucking introduces no false negatives.

Two Summarizing Lemmas

From Lemmas 93 (p. 752) and 95 (p. 760), we have:

Lemma 97 *Each approximation step introduces at most* $M^2 2^{-p}(k-1)^n$ *false positives.*

From Lemmas 94 (p. 757) and 96 (p. 762), we have:

Lemma 98 *Each approximation step introduces at most* $M^2\binom{n-\ell-1}{k-\ell-1}$ *false negatives.*

The Proof (continued)

- The above two lemmas show that each approximation step introduce "few" false positives and false negatives.
- We next show that the resulting crude circuit has "a lot" of false positives or false negatives.

The Final Crude Circuit

Lemma 99 *Every final crude circuit either is identically false—thus wrong on all positive examples—or outputs true on at least half of the negative examples.*

- Suppose it is not identically false.
- By construction, it accepts at least those graphs that have a clique on some set X of nodes, with $|X| \leq \ell$, which at $n^{1/8}$ is less than $k = n^{1/4}$.
- The proof of Lemma 93 (p. 752ff) shows that at least half of the colorings assign different colors to nodes in X.
- So half of the negative examples have a clique in X and are accepted.

The Proof (continued)

- Recall the constants on p. 744: $k = n^{1/4}$, $\ell = n^{1/8}$, $p = n^{1/8} \log n$, $M = (p-1)^\ell \ell! < n^{(1/3)n^{1/8}}$ for large n.
- Suppose the final crude circuit is identically false.
  - By Lemma 98 (p. 764), each approximation step introduces at most $M^2 \binom{n-\ell-1}{k-\ell-1}$ false negatives.
  - There are $\binom{n}{k}$ positive examples.
  - The original crude circuit for $\text{CLIQUE}_{n,k}$ has at least

$$\frac{\binom{n}{k}}{M^2 \binom{n-\ell-1}{k-\ell-1}} \geq \frac{1}{M^2} \left(\frac{n-\ell}{k} \right)^\ell \geq n^{(1/12)n^{1/8}}$$

gates for large n.

The Proof (concluded)

- Suppose the final crude circuit is not identically false.
  - Lemma 99 (p. 766) says that there are at least $(k-1)^n/2$ false positives.
  - By Lemma 97 (p. 764), each approximation step introduces at most $M^2 2^{-p}(k-1)^n$ false positives
  - The original crude circuit for $\text{CLIQUE}_{n,k}$ has at least

$$\frac{(k-1)^n/2}{M^2 2^{-p}(k-1)^n} = \frac{2^{p-1}}{M^2} \geq n^{(1/3)n^{1/8}}$$

    gates.

$P \neq NP$ Proved?

- Razborov's theorem says that there is a monotone language in NP that has no polynomial monotone circuits.
- If we can prove that all monotone languages in P have polynomial monotone circuits, then $P \neq NP$.
- But Razborov proved in 1985 that some monotone languages in P have no polynomial monotone circuits!

Logarithmic Space

REACHABILITY Is NL-Complete

- REACHABILITY $\in$ NL (p. 95).
- Suppose L is decided by the $\log n$ space-bounded TM N.
- Given input x, construct in logarithmic space the polynomial-sized configuration graph G of N on input x (see Theorem 21 on p. 176).
- G has a single initial node, call it 1.
- Assume G has a single accepting node n.
- $x \in L$ if and only if the instance of REACHABILITY has a "yes" answer.

2SAT Is NL-Complete

- 2SAT $\in$ NL (p. 265).
- As NL $=$ coNL (p. 191), it suffices to reduce the coNL-complete UNREACHABILITY to 2SAT.
- Start without loss of generality an *acyclic* graph G.
- Identify each edge (x, y) with clause $\neg x \vee y$.
- Add clauses (s) and $(\neg t)$ for the start and target nodes s and t.
- The resulting 2SAT instance is satisfiable if and only if there is no path from s to t in G.

The Class RL

- REACHABILITY is for *directed* graphs.
- It is not known if UNDIRECTED REACHABILITY is in L.
- But it is in randomized logarithmic space, called **RL**.
- RL is RP in which the space bound is logarithmic.
- We shall prove that UNDIRECTED REACHABILITY $\in$ RL.[a]
- As a note, UNDIRECTED REACHABILITY $\in$ coRL.[b]

---

[a] Aleliunas, Karp, Lipton, Lovász, and Rackoff (1979).
[b] Borodin, Cook, Dymond, Ruzzo, and Tompa (1989).

Random Walks

- Let $G = (V, E)$ be an undirected graph with $1, n \in V$.
- Add self-loops $\{ i, i \}$ at each node i.
- The randomized algorithm for testing if there is a path from 1 to n is a **random walk**.

The Random Walk Framework

$x := 1$;

while $x \neq n$ **do**

  Pick y uniformly from x's neighbors (including x);

  $x := y$;

end while

Some Terminology

- v_t is the node visited by the random walk at time t.
- In particular, $v_0 = 1$.
- d_i denotes the degree of i (including the self-loops).
- Let $p_t[\,i\,] = \text{prob}[\,v_t = i\,]$.

A Convergence Result

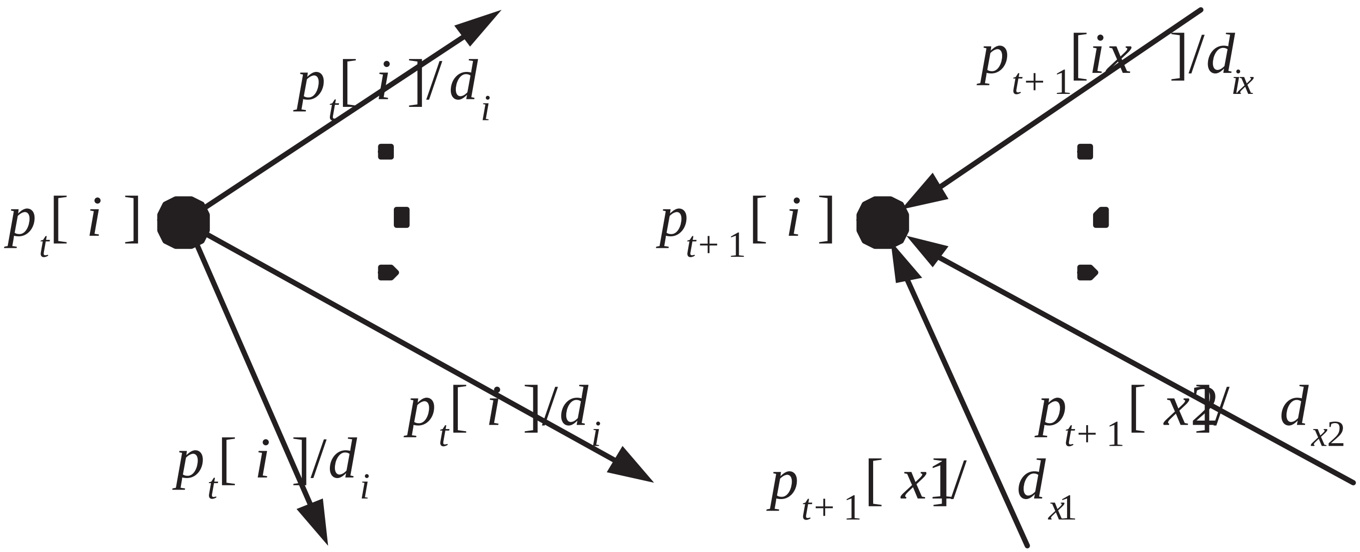
Lemma 102 *If $G = (V, E)$ is connected, then* $\lim_{t\to\infty} p_t[\,i\,] = \frac{d_i}{2\cdot|\,E\,|}$ *for all nodes i.*

- Here is the intuition.
- The random walk algorithm picks the edges uniformly randomly.
- In the limit, the algorithm will be well "mixed" and forgets about the initial node.
- Then the probability of each node being visited is proportional to its number of incident edges.
- Finally, observe that $\sum_{i=1}^{n} d_i = 2 \cdot |\,E\,|$.

Proof of Lemma 102

- Let $\delta_t[\,i\,] = p_t[\,i\,] - \frac{d_i}{2\cdot|\,E\,|}$, the deviation.
- Define $\Delta_t = \sum_{i \in V} |\delta_t[\,i\,]|$, the total absolute deviation.
- Now we calculate the $p_{t+1}[\,i\,]$'s from the $p_t[\,i\,]$'s.
- Each node divides its $p_i[\,t\,]$ into d_i equal parts and distributes them to its neighbors.
- Each node adds those portions from its neighbors (including itself) to form $p_i[\,t+1\,]$.

The Flows



Proof of Lemma 102 (continued)

- $p_t[\,i\,] = \delta_t[\,i\,] + \frac{d_i}{2 \cdot |\,E\,|}$ by definition.
- Splitting and giving the $\frac{d_i}{2 \cdot |\,E\,|}$ part does not affect $p_{t+1}[\,i\,]$ because the same $\frac{1}{2 \cdot |\,E\,|}$ is exchanged between any two neighbors.
- So we only consider the splitting of the $\delta_t[\,i\,]$ part.
- The $\delta_t[\,i\,]$'s are exchanged between adjacent nodes.

Proof of Lemma 102 (continued)

- Clearly $\sum_i \delta_{t+1}[\,i\,] = \sum_i \delta_t[\,i\,]$ because of conservation.
- But $\Delta_{t+1} = \sum_i |\delta_{t+1}[\,i\,]| \leq \sum_i |\delta_t[\,i\,]| = \Delta_t$.
  - If $\delta_t[\,i\,]$'s are all of the same sign, then $\Delta_{t+1} = \sum_i |\delta_{t+1}[\,i\,]| = \sum_i |\delta_t[\,i\,]| = \Delta_t$.
  - When $\delta_t[\,i\,]$'s of opposite signs meet at a node, that will reduce $\sum_i |\delta_{t+1}[\,i\,]|$.
- We next quantify the decrease $\Delta_t - \Delta_{t+1}$.

Proof of Lemma 102 (continued)

- There is a node i^+ with $\delta_t[\, i^+ \,] \geq \frac{\Delta_t}{2 \cdot |\, V \,|}$, and there is a node i^- with $\delta_t[\, i^- \,] \leq -\frac{\Delta_t}{2 \cdot |\, V \,|}$.
  - Recall that $\sum_i \delta_t[\, i \,] = 0$ and $\sum_{i \in V} |\delta_t[\, i \,]| = \Delta_t$.
  - So the sum of all $\delta_t[\, i \,] \geq 0$ equals $\Delta_t / 2$.
  - As there are at most $|\, V \,|$ such $\delta_t[\, i \,]$, there must be one with magnitude at least $(\Delta_t/2)/|\, V \,|$.
  - Similarly for $\delta_t[\, i \,] \leq 0$.

Proof of Lemma 102 (continued)

- There is a path $[\, i_0 = i^+, i_1, i_2, \ldots, i_{2m} = i^- \,]$ with an even number of edges between i^+ and i^-.
  - Add self-loops to make it true.
- The positive deviation $\delta_t[\, i^+ \,]$ from i^+ will travel along this path for m steps, always subdivided by the degree of the current node.
- Similarly for the negative deviation $\delta_t[\, i^- \,]$ from i^-.

Proof of Lemma 102 (continued)

- At least a positive deviation equal to $\frac{1}{|V|^m}$ of the original amount will arrive at the middle node i_m.
- Similarly for a negative deviation from the opposite direction.
- So after $m \leq n$ steps, a positive deviation of at least $\frac{\Delta_t}{2 \cdot |V|^n}$ will cancel an equal amount of negative deviation.
- We do not need to care about cases where numbers of the same sign meet at a node; they will not change Δ_t.

Proof of Lemma 102 (concluded)

- So in n steps the total absolute deviation decreases from Δ_t to at most $\Delta_t(1 - \frac{1}{|V|^n})$.
- But we already knew that Δ_t will never increase.[a]
- So in the limit, $\Delta_t \to 0$ (but exponentially slow).

[a]Contributed by Mr. Chih-Duo Hong (R95922079) on January 11, 2007.

First Return Times

- Lemma 102 (p. 783) and theory of Markov chains[a] imply that the walk returns to i every $2 \cdot |\, E \,|/d_i$ steps, *asymptotically and on the average.*
- Equivalently, if $v_t = i$, then the expected time until the walk comes back to i for the first time after t is $2 \cdot |\, E \,|/d_i$, asymptotically.
  - This is called the **mean recurrence time**.

---

[a] Particularly, theory of homogeneous Markov chains on first passage time.

First Return Times (concluded)

- Although the above is an asymptotic statement, the said expected return time is *the same* for any t—including the beginning $t = 0$.
- So from the beginning and onwards, the expected time between two successive visits to node i is exactly $2 \cdot |\, E \,| / d_i$.

Average Time To Reach Target Node n

- Assume there is a path $[1, i_1, \ldots, i_m = n]$ from 1 to n.
  - If there is none, we are done because the algorithm then returns no false positives.
- Starting from 1, we will return to 1 every expected $2 \cdot |E|/d_1$ steps.
- Every cycle of leaving and returning uses at least *two* edges of 1.
  - They may be identical.

Average Time To Reach Target Node n (continued)

- So after an expected $\frac{d_1}{2}$ of such returns, the walk will head to i_1.
  - There are d_1^2 pairs of edges incident on node 1 used for the cycles.
  - Among them, d_1 of them leave node 1 by way of i_1 and d_1 of them return by way of i_1.
- The expected number of steps is

$$\frac{d_1}{2} \frac{2 \cdot |\, E\,|}{d_1} = |\, E\,|.$$

Average Time To Reach Target Node n (concluded)

- Repeat the above argument from $i_1, i_2, \ldots$
- After an expected number of $\leq n \cdot |E|$ steps, we will have arrived at node n.
- Markov's inequality (p. 410) suggests that we run the algorithm for $2n \cdot |\, E \,|$ steps to obtain the desired probability of success, 0.5.

Probability To Visit All Nodes

Corollary 103 *With probability at least 0.5, the random walk algorithm visits all nodes in $2n \cdot | E |$ steps.*

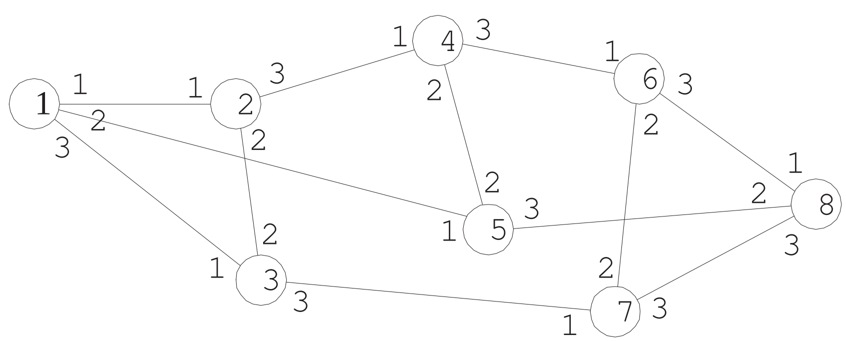
- Repeat the above arguments for this particular path: $[1, 2, \ldots, n]$.

The Complete Algorithm

$x := 1;$
$c := 0;$
while $x \neq n$ and $c < 2n \cdot |\, E \,|$ **do**
  Pick y uniformly from x's neighbors (including x);
  $x := y;$
  $c := c + 1;$
end while
if $x = n$ **then**
  "yes";
else
  "no";
end if

Some Graph-Theoretic Notions

- A d-regular (undirected) graph has degree d for each node.
- Let G be d-regular.
- Each node's incident edge is labeled from 1 to d.
  - An edge is labeled at both ends.



Universal Sequences[a]

- A sequence of numbers between 1 and d results in a walk on the graph if given the starting node.
  - E.g., $(1, 3, 2, 2, 1, 3)$ from node 1.
- A sequence of numbers between 1 and d is called **universal** for d-regular graphs with n nodes if:
  - For *any* labeling of *any* n-node d-regular graph G, and for *any* starting node, all nodes of G are visited.
  - A node may be visited more than once.
- Useful for museum visitors, security guards, etc.

[a]Attributed to Cook.

Existence of Universal Sequences

Theorem 104 *For any n, a universal sequence exists for the set of d-regular connected undirected n-node graphs.*

- Enumerate all the different labelings of d-regular n-node connected graphs and all starting nodes.
- Call them $(G_1, v_1), (G_2, v_2), \ldots$ (finitely many).
- S_1 is a sequence that traverses G_1, starting from v_1.
  - A spanning tree will accomplish this.
- S_2 is a sequence that traverses G_2, starting from the node at which S_1 ends *when applied to* (G_2, v_2).

The Proof (concluded)

- S_3 is a sequence that traverses G_3, starting from the node at which $S_1 S_2$ ends when applied to (G_3, v_3), etc.
- The sequence $S \equiv S_1 S_2 S_3 \cdots$ is universal.
  - Suppose S starts from node v of a labeled d-regular n-node graph G'.
  - Let $(G', v) = (G_k, n_k)$, the kth enumerated pair.
  - By construction, S_k will traverse G' (if not earlier).

A $O(n^3 \log n)$ Bound on Universal Sequences

Theorem 105 *For any n and d, a universal sequence of length $O(n^3 \log n)$ for d-regular n-node connected graphs exists.*

- Fix a d-regular labeled n-node graph G.
- A random walk of length $2n \cdot |\, E\,| = n^2 d = O(n^2)$ fails to traverse G with probability at most $1/2$.
  - By Corollary 103 (p. 797).
  - This holds wherever the walk starts.
- The failure probability for G drops to $2^{-\Theta(n \log n)}$ if the random walk has length $\Theta(n^3 \log n)$.

The Proof (continued)

- There are $2^{O(n \log n)}$ d-regular labeled n-node graphs.
  - Each node has $\leq n^d$ choices of neighbors.
  - So there are $\leq n^{d+1}$ d-regular graphs on nodes $\{1, 2, \ldots, n\}$.
  - Each node's d edges are labeled with unique integers between 1 and d.
  - Hence the count is

$$\leq n^{d+1}(d!)^n = n^{O(n)} = 2^{O(n \log n)}.$$

The Proof (concluded)

- The probability that there exists a d-regular labeled n-node graph that the random walk fails to traverse can be made at most $1/2$.
  - Lengthen the length of the walk suitably.
- Because the probability is less than one, there *exists* a walk that traverses all labeled d-regular graphs.

Finis