coNP and Function Problems

coNP

- By definition, coNP is the class of problems whose complement is in NP.
  - $L \in \text{coNP}$ if and only if $\bar{L} \in \text{NP}$.
- NP problems have succinct certificates.[a]
- coNP is therefore the class of problems that have succinct **disqualifications**:[b]
  - A "no" instance possesses a short proof of its being a "no" instance.
  - Only "no" instances have such proofs.

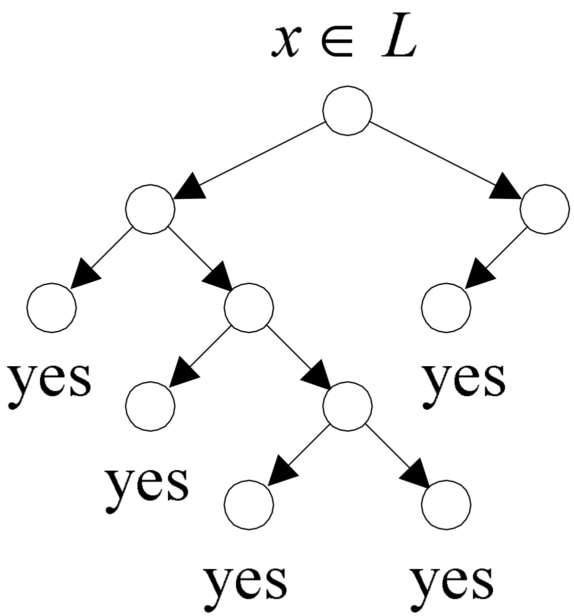
---

[a]Recall Proposition 41 (p. 331).

[b]To be proved in Proposition 54 (p. 459).

coNP (continued)

- Suppose L is a coNP problem.
- There exists a nondeterministic polynomial-time 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.
- If we swap "yes" and "no" in M, the new algorithm decides $\bar{L} \in$ NP in the classic sense (p. 108).



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

coNP (continued)

- So there are 3 major approaches to proving $L \in$ coNP.
  1. Prove $\bar{L} \in$ NP.
     - Especially when you already knew $\bar{L} \in$ NP.
  2. Prove that only "no" instances possess short proofs (for their not being in L).[a]
  3. Write an algorithm for it directly.

---

[a]Recall Proposition 41 (p. 331).

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 p. 156).

Some coNP Problems

- SAT COMPLEMENT $\in$ coNP.
  - SAT COMPLEMENT is the complement of SAT.
  - Or, the disqualification is a truth assignment that *satisfies* it.
- HAMILTONIAN PATH COMPLEMENT $\in$ coNP.
  - HAMILTONIAN PATH COMPLEMENT is the complement of HAMILTONIAN PATH.
  - Or, the disqualification is a Hamiltonian path.

Some coNP Problems (concluded)

- VALIDITY $\in$ coNP.
  - If ϕ is not valid, it can be disqualified very succinctly: a truth assignment that does *not* satisfy it.
- OPTIMAL TSP (D) $\in$ coNP.
  - OPTIMAL TSP (D) asks if the optimal tour has a total distance of B, where B is an input.[a]
  - The disqualification is a tour with a length $\geq B$ plus a tour with a length $< B$.

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

A Nondeterministic Algorithm for SAT COMPLEMENT (See also p. 119)

ϕ is a boolean formula with n variables.

for $i = 1, 2, \ldots, n$ **do**
    Guess $x_i \in \{0, 1\}$; {Nondeterministic choice.}
end for
{Verification:}
if $\phi(x_1, x_2, \ldots, x_n) = 1$ **then**
    "no";
else
    "yes";
end if

Analysis

- The algorithm decides language $\{ \phi : \phi \text{ is unsatisfiable} \}$.
  - The computation tree is a complete binary tree of depth n.
  - Every computation path corresponds to a particular truth assignment out of 2^n.
  - ϕ is unsatisfiable if and only if every truth assignment falsifies ϕ.
  - But every truth assignment falsifies ϕ if and only if every computation path results in "yes."

An Alternative Characterization of coNP

Proposition 54 *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. 330, we assume $|\, y \,| \leq |\, x \,|^k$ for some k.)

- $\bar{L} = \{ x : \exists y \, (x, y) \in \neg R \}$.
- Because $\neg R$ remains polynomially balanced, $\bar{L} \in \text{NP}$ by Proposition 41 (p. 331).
- Hence $L \in \text{coNP}$ by definition.

coNP-Completeness

Proposition 55 *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 $\overline{L'}$ 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 $R(x) \in L$.
- By the law of transposition, $x \notin L'$ if and only if $R(x) \notin L$.

coNP Completeness (concluded)

- So $x \in \overline{L'}$ if and only if $R(x) \in \bar{L}$.
- The *same* R is a reduction from $\overline{L'}$ to $\bar{L}$.
- This shows $\bar{L}$ is coNP-hard.
- But $\bar{L} \in$ coNP.
- This shows $\bar{L}$ is coNP-complete.

Some coNP-Complete Problems

- SAT COMPLEMENT is coNP-complete.
- HAMILTONIAN PATH COMPLEMENT is coNP-complete.
- VALIDITY is coNP-complete.
  - ϕ is valid if and only if $\neg\phi$ is not satisfiable.
  - $\phi \in$ VALIDITY if and only if $\neg\phi \in$ SAT COMPLEMENT.
  - The reduction from SAT COMPLEMENT to VALIDITY is hence easy: $R(\phi) = \neg\phi$.

Possible Relations between P, NP, coNP

1. $\text{P} = \text{NP} = \text{coNP}$.

2. $\text{NP} = \text{coNP}$ but $\text{P} \neq \text{NP}$.

3. $\text{NP} \neq \text{coNP}$ and $\text{P} \neq \text{NP}$.

    - This is the current "consensus."[a]

[a] Carl Gauss (1777–1855), "I could easily lay down a multitude of such propositions, which one could neither prove nor dispose of."

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 $\sqrt{N} = 2^{0.5 \log N}$.
  - It is an exponential-time algorithm.
- A polynomial-time algorithm for PRIMES was not found until 2002 by Agrawal, Kayal, and Saxena!
- The running time is $\tilde{O}(\log^{7.5} N)$.

```
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.}
```

The Primality Problem (concluded)

- Later, we will focus on efficient "randomized" algorithms for PRIMES (used in *Mathematica*, e.g.).
- NP $\cap$ 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.
- We will see that PRIMES $\in$ NP $\cap$ coNP.
  - In fact, PRIMES $\in$ P as mentioned earlier.

Basic Modular Arithmetics[a]

- Let $m, n \in \mathbb{Z}^+$.
- $m \mid n$ means m divides n; 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 56 (p. 469) is a primitive root of p.

---

[a] Carl Friedrich Gauss.

Basic Modular Arithmetics (concluded)

- We use

$$a \equiv b \mod n$$

if $n \,|\, (a - b)$.

  – So $25 \equiv 38 \bmod 13$.

- We use

$$a = b \bmod n$$

if b is the remainder of a divided by n.

  – So $25 = 12 \bmod 13$.

Primitive Roots in Finite Fields

Theorem 56 (Lucas & Lehmer, 1927) [a] *A number $p > 1$ is a prime if and only if there is a number $1 < r < p$ 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 $p - 1$.*

- This r is called the **primitive root** or **generator**.
- We will prove one direction of the theorem later.[b]

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

[b] See pp. 480ff.

Derrick Lehmer[a] (1905–1991)

[a]Inventor of the linear congruential generator in 1951.

Pratt's Theorem

Theorem 57 (Pratt, 1975) PRIMES $\in NP \cap coNP$.

- PRIMES $\in$ coNP because a succinct disqualification is a proper divisor.
  - A proper divisor of a number means it is *not* a prime.
- Now suppose p is a prime.
- p's certificate includes the r in Theorem 56 (p. 469).
  - There may be multiple choices for r.

The Proof (continued)

- Use recursive doubling to check if $r^{p-1} = 1 \bmod p$ in time polynomial in the length of the input, $\log_2 p$.
  - $r, r^2, r^4, \ldots \bmod p$, a total of $\sim \log_2 p$ steps.
- We also need all *prime* divisors of $p - 1$: $q_1, q_2, \ldots, q_k$.
  - Whether $r, q_1, \ldots, q_k$ are easy to find is irrelevant.
- Checking $r^{(p-1)/q_i} \neq 1 \bmod p$ is also easy.
- Checking $q_1, q_2, \ldots, q_k$ are all the divisors of $p - 1$ is easy.

The Proof (concluded)

- We still need certificates for the primality of the q_i's.
- The complete certificate is recursive and tree-like:

$$C(p) = (r; q_1, C(q_1), q_2, C(q_2), \ldots, q_k, C(q_k)). \tag{5}$$

- We next prove that $C(p)$ is succinct.
- As a result, $C(p)$ can be checked in polynomial time.

A Certificate for 23[a]

- Note that 5 is a primitive root modulo 23 and $23 - 1 = 22 = 2 \times 11$.[b]
- So
$$C(23) = (5; 2, C(2), 11, C(11)).$$
- Note that 2 is a primitive root modulo 11 and $11 - 1 = 10 = 2 \times 5$.
- So
$$C(11) = (2; 2, C(2), 5, C(5)).$$

[a] Thanks to a lively discussion on April 24, 2008.
[b] Other primitive roots are $7, 10, 11, 14, 15, 17, 19, 20, 21$.

A Certificate for 23 (concluded)

- Note that 2 is a primitive root modulo 5 and $5 - 1 = 4 = 2^2$.
- So
$$C(5) = (2; 2, C(2)).$$
- In summary,
$$C(23) = (5; 2, C(2), 11, (2; 2, C(2), 5, (2; 2, C(2)))).$$
  - In *Mathematica*, `PrimeQCertificate[23]` yields
$$\{ 23, 5, \{ 2, \{ 11, 2, \{ 2, \{ 5, 2, \{ 2 \}\}\}\}\}\}$$

The Succinctness of the Certificate

Lemma 58 *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 $k \leq \log_2 p$ prime divisors $q_1 = 2, q_2, \ldots, q_k$.
  - Reason:
$$2^k \leq \prod_{i=1}^{k} q_i \leq p - 1.$$
- Note also that, as $q_1 = 2$,
$$\prod_{i=2}^{k} q_i \leq \frac{p-1}{2}. \tag{6}$$

The Proof (continued)

- $C(p)$ requires:
  - 2 parentheses;
  - $2k < 2\log_2 p$ separators (at most $2\log_2 p$ bits);
  - r (at most $\log_2 p$ bits);
  - $q_1 = 2$ and its certificate 1 (at most 5 bits);
  - $q_2, \ldots, q_k$ (at most $2\log_2 p$ bits);[a]
  - $C(q_2), \ldots, C(q_k)$.

[a] Why?

The Proof (concluded)

- $C(p)$ is succinct because, by induction,

$$
\begin{aligned}
|\,C(p)\,| &\leq 5\log_2 p + 5 + 5\sum_{i=2}^{k} \log_2^2 q_i \\
&\leq 5\log_2 p + 5 + 5\left(\sum_{i=2}^{k} \log_2 q_i\right)^2 \\
&\leq 5\log_2 p + 5 + 5\log_2^2 \frac{p-1}{2} \quad \text{by inequality (6)} \\
&< 5\log_2 p + 5 + 5[\,(\log_2 p) - 1\,]^2 \\
&= 5\log_2^2 p + 10 - 5\log_2 p \leq 5\log_2^2 p
\end{aligned}
$$

for $p \geq 4$.

Turning the Proof into an Algorithm[a]

- How to turn the proof into a nondeterministic polynomial-time algorithm?
- First, guess a $\log_2 p$-bit number r.
- Then guess up to $\log_2 p$ numbers $q_1, q_2, \ldots, q_k$ each containing at most $\log_2 p$ bits.
- Then recursively do the same thing for each of the q_i to form a certificate (5) on p. 473.
- Finally check if the two conditions of Theorem 56 (p. 469) hold throughout the tree.

[a]Contributed by Mr. Kai-Yuan Hou (B99201038, R03922014) on November 24, 2015.

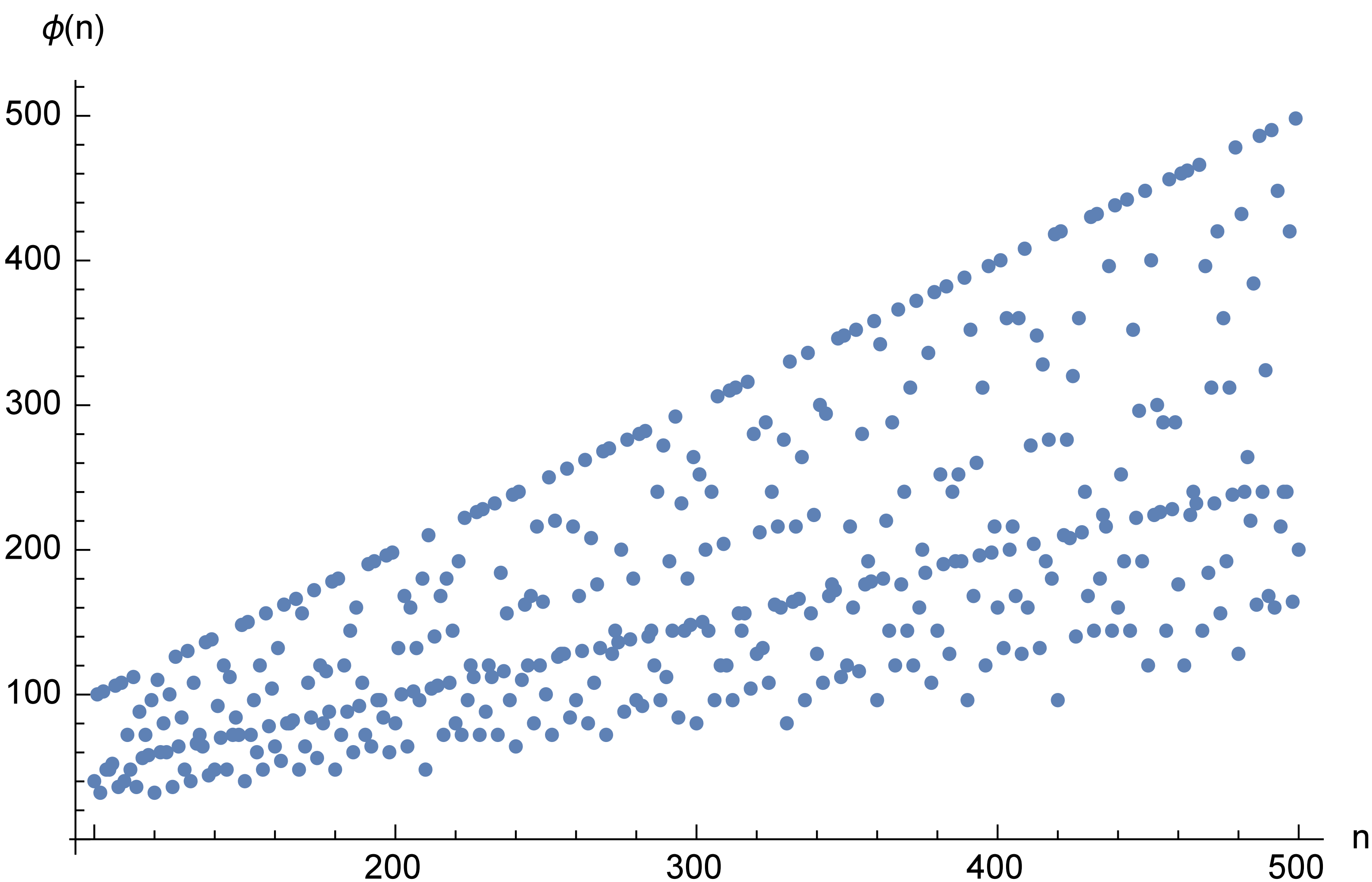
Euler's[a] Totient or Phi Function

- 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.[b]
  - $\Phi(12) = \{\, 1, 5, 7, 11 \,\}$.
- 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).

[b] Z_n^* is an alternative notation.



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

Leonhard Euler (1707–1783)

Three Properties of Euler's Function[a]

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

Lemma 59 *If* $n = p_1^{e_1} p_2^{e_2} \cdots p_\ell^{e_\ell}$ *is the prime factorization of* n*, then*

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

- For example, if $n = pq$, where p and q are distinct primes, then

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

---

[a] See p. 224 of the textbook.

[b] Consult any textbooks on discrete mathematics.

Three Properties of Euler's Function (concluded)

Corollary 60 $\phi(mn) = \phi(m)\,\phi(n)$ *if* $\gcd(m, n) = 1$.

Lemma 61 (Gauss) $\sum_{m|n} \phi(m) = n$.

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

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

[a] Pierre de Fermat (1601–1665).

The Proof (concluded)

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

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

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

The Fermat-Euler Theorem[a]

Corollary 63 *For all $a \in \Phi(n)$, $a^{\phi(n)} = 1 \bmod n$.*

- The proof is similar to that of Lemma 62 (p. 486).
- Consider $a\Phi(n) = \{ am \bmod n : m \in \Phi(n) \}$.
- $a\Phi(n) = \Phi(n)$.
  - $a\Phi(n) \subseteq \Phi(n)$ as a remainder must be between 0 and $n - 1$ and relatively prime to n.
  - Suppose $am \equiv am' \bmod n$ for $m' < m < n$, where $m, m' \in \Phi(n)$.
  - That means $a(m - m') = 0 \bmod n$, and n divides a or $m - m'$, which is impossible.

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

The Proof (concluded)[a]

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

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

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

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

An Example

- As $12 = 2^2 \times 3$,
$$\phi(12) = 12 \times \left(1 - \frac{1}{2}\right)\left(1 - \frac{1}{3}\right) = 4.$$
- In fact, $\Phi(12) = \{ 1, 5, 7, 11 \}$.
- For example,
$$5^4 = 625 = 1 \bmod 12.$$

Exponents

- The **exponent** of $m \in \Phi(p)$ is the least $k \in \mathbb{Z}^+$ such that

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

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

Lemma 64 *Any nonzero polynomial of degree k has at most k distinct roots modulo p.*

Exponents and Primitive Roots

- From Fermat's "little" theorem (p. 486), all exponents divide $p-1$.
- A primitive root of p is thus a number with exponent $p-1$.
- Let $R(k)$ denote the total number of residues in $\Phi(p) = \{1, 2, \ldots, p-1\}$ that have exponent k.
- We already knew that $R(k) = 0$ for $k \not| (p-1)$.
- As every number has an exponent,

$$\sum_{k \mid (p-1)} R(k) = p - 1.$$

Size of $R(k)$

- Any $a \in \Phi(p)$ of exponent k satisfies $x^k = 1 \bmod p$.
- By Lemma 64 (p. 491) there are at most k residues of exponent k, i.e., $R(k) \leq k$.
- Let s be a residue of exponent k.
- $1, s, s^2, \ldots, s^{k-1}$ are distinct modulo p.
  - Otherwise, $s^i \equiv s^j \bmod p$ with $i < j$.
  - Then $s^{j-i} = 1 \bmod p$ with $j - i < k$, a contradiction.
- As all these k distinct numbers satisfy $x^k = 1 \bmod p$, they comprise *all* the solutions of $x^k = 1 \bmod p$.

Size of $R(k)$ (continued)

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

Size of $R(k)$ (continued)

- 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 61 (p. 484).

- Hence

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

Size of $R(k)$ (concluded)

- Incidentally, we have shown that

$$g^{\ell}, \quad \text{where } \ell \in \Phi(k),$$

  are all the numbers with exponent k if g has exponent k.

- As $R(p-1) = \phi(p-1) > 0$, p has primitive roots.

- This proves one direction of Theorem 56 (p. 469).

A Few Calculations

- Let $p = 13$.
- From p. 488 $\phi(p-1) = 4$.
- Hence $R(12) = 4$.
- Indeed, there are 4 primitive roots of p.
- As
$$\Phi(p-1) = \{ 1, 5, 7, 11 \},$$
the primitive roots are
$$g^1, g^5, g^7, g^{11},$$
where g is *any* primitive root.

Function Problems

- Decision problems are yes/no problems (SAT, TSP (D), etc.).
- **Function problems** require a solution (a satisfying truth assignment, a best TSP tour, etc.).
- Optimization problems are clearly function problems.
- What is the relation between function and decision problems?
- Which one is harder?

Function Problems Cannot Be Easier than Decision Problems

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

FSAT

- FSAT is this function problem:
  - Let $\phi(x_1, x_2, \ldots, x_n)$ be a boolean expression.
  - If 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.
- SAT is a subroutine (black box) that returns "yes" or "no" on the satisfiability of the input.

An Algorithm for FSAT Using SAT

```
t := ε; {Truth assignment.}
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

- If SAT can be solved in polynomial time, so can FSAT.
  - There are $\leq n+1$ calls to the algorithm for SAT.[a]
  - Boolean expressions shorter than ϕ are used in each call to the algorithm for SAT.
- Hence SAT and FSAT are equally hard (or easy).
- Note that this reduction from FSAT to SAT is not a Karp reduction.[b]
- Instead, it calls SAT multiple times as a subroutine, and its answers guide the search on the computation tree.

---

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

[b]Recall p. 262 and p. 266.

TSP and TSP (D) Revisited

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

An Algorithm for TSP Using TSP (D)

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

Analysis

- An edge which is not on *any* remaining optimal tours will be eliminated, with its d_{ij} set to $C+1$.
- So the algorithm ends with n edges which are not eliminated (why?).
- This is true even if there are multiple optimal tours![a]

[a] Thanks to a lively class discussion on November 12, 2013.

Analysis (concluded)

- There are $O(|x| + n^2)$ calls to the algorithm for TSP (D).
- Each call has an input length of $O(|x|)$.
- So if TSP (D) can be solved in polynomial time, so can TSP.
- Hence TSP (D) and TSP are equally hard (or easy).

Randomized Computation

I know that half my advertising works,
I just don't know which half.
— John Wanamaker

I know that half my advertising is
a waste of money,
I just don't know which half!
— McGraw-Hill ad.

Randomized Algorithms[a]

- Randomized algorithms flip unbiased coins.
- There are important problems for which there are no known efficient *deterministic* algorithms but for which very efficient randomized algorithms exist.
  - Extraction of square roots, for instance.
- There are problems where randomization is *necessary*.
  - Secure protocols.
- Randomized version can be more efficient.
  - Parallel algorithms for maximal independent set.[b]

[a] Rabin (1976); Solovay & Strassen (1977).
[b] "Maximal" (a local maximum) not "maximum" (a global maximum).

Randomized Algorithms (concluded)

- Are randomized algorithms algorithms?[a]
- Coin flips are occasionally used in politics.[b]

---

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

[b]In the 2016 Iowa Democratic caucuses, e.g. (see `http://edition.cnn.com/2016/02/02/politics/hillary-clinton-coin-flip-iowa-bernie-sanders/index.html`).

“Four Most Important Randomized Algorithms”[a]

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

---

[a] Trevisan (2006).

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

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

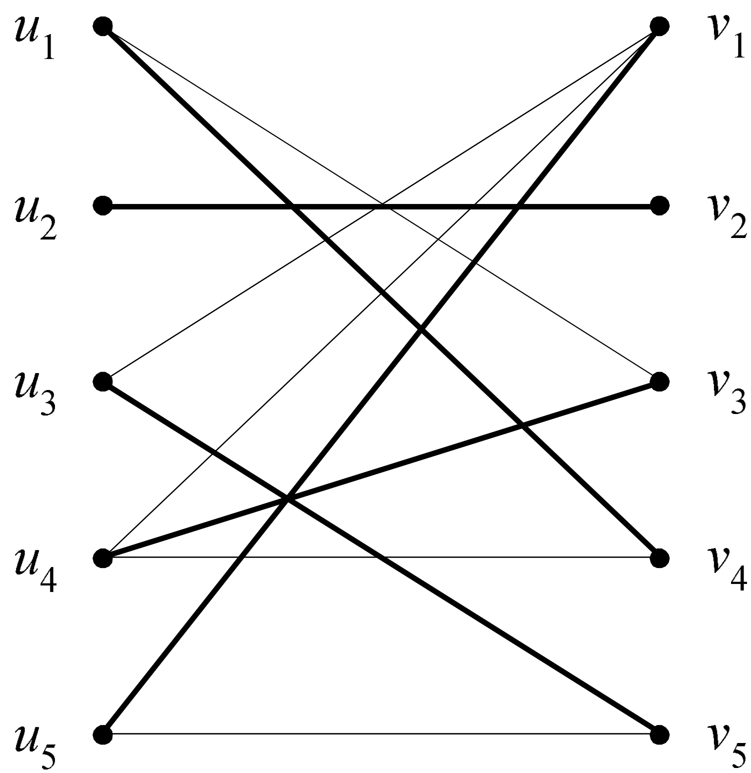
[d] Schwartz (1980); Zippel (1979).

[e] Sinclair & Jerrum (1989).

Bipartite Perfect Matching

- We are given a **bipartite graph** $G = (U, V, E)$.
  - $U = \{ u_1, u_2, \ldots, u_n \}$.
  - $V = \{ v_1, v_2, \ldots, v_n \}$.
  - $E \subseteq U \times V$.
- We are asked if there is a **perfect matching**.
  - A permutation π of $\{ 1, 2, \ldots, n \}$ such that
    $$(u_i, v_{\pi(i)}) \in E$$
    for all $i \in \{ 1, 2, \ldots, n \}$.
- A perfect matching contains n edges.

A Perfect Matching in a Bipartite Graph



Symbolic Determinants

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

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

Symbolic Determinants (continued)

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

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

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

Symbolic Determinants (concluded)

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

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

  - 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.[a]

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

---

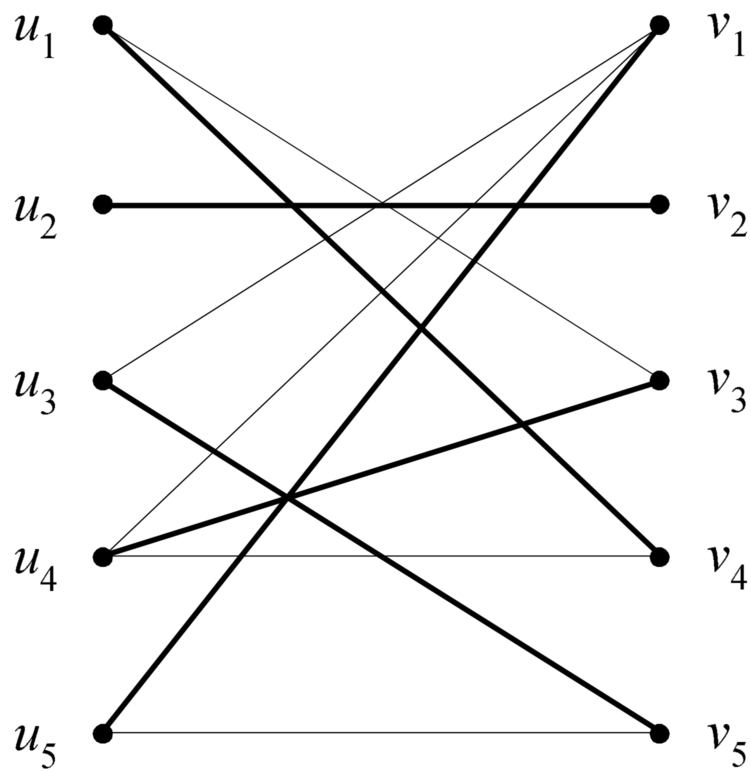
[a]Equivalently, $\operatorname{sgn}(\pi) = 1$ if the number of (i, j)s such that $i < j$ and $\pi(i) > \pi(j)$ is even. Contributed by Mr. Hwan-Jeu Yu (`D95922028`) on May 1, 2008.

Determinant and Bipartite Perfect Matching

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

Proposition 65 (Edmonds, 1967) *G has a perfect matching if and only if* $\det(A^G)$ *is not identically zero.*

Perfect Matching and Determinant (p. 513)



Perfect Matching and Determinant (concluded)

- The matrix is (p. 515)

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

- $\det(A^G) = -x_{14}x_{22}x_{35}x_{43}x_{51} + x_{13}x_{22}x_{35}x_{44}x_{51} + x_{14}x_{22}x_{31}x_{43}x_{55} - x_{13}x_{22}x_{31}x_{44}x_{55}$.
- Each nonzero term denotes a perfect matching, and vice versa.

How To Test If a Polynomial Is Identically Zero?

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

Number of Roots of a Polynomial

Lemma 66 (Schwartz, 1980) *Let* $p(x_1, x_2, \ldots, x_m) \not\equiv 0$ *be a polynomial in* m *variables each of degree at most* d. *Let* $M \in \mathbb{Z}^+$. *Then the number of* m*-tuples*

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

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

$$\leq mdM^{m-1}.$$

- By induction on m (consult the textbook).

Density Attack

- The density of roots in the domain is at most

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

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

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

  has a probability of $\leq md/M$ of being a root of p.
- Note that M is under our control!
  - One can raise M to lower the error probability, e.g.

Density Attack (concluded)

Here is a sampling algorithm to test if $p(x_1, x_2, \ldots, x_m) \not\equiv 0$.

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

Analysis

- If $p(x_1, x_2, \ldots, x_m) \equiv 0$, the algorithm will always be correct as $p(i_1, i_2, \ldots, i_m) = 0$.
- Suppose $p(x_1, x_2, \ldots, x_m) \not\equiv 0$.
  - The algorithm will answer incorrectly with probability at most md/M by Eq. (9) on p. 522.
- We next return to the original problem of bipartite perfect matching.

A Randomized Bipartite Perfect Matching Algorithm[a]

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

[a]Lovász (1979). According to Paul Erdős, Lovász wrote his first significant paper "at the ripe old age of 17."

Analysis

- If G has no perfect matchings, the algorithm will always be correct as $\det(A^G(i_{11}, \ldots, i_{nn})) = 0$.
- Suppose G has a perfect matching.
  - The algorithm will answer incorrectly with probability at most $md/M = 0.5$ with $m = n^2$, $d = 1$ and $M = 2n^2$ in Eq. (9) on p. 522.
- Run the algorithm *independently* k times.
- Output "G has no perfect matchings" if and only if *all* say "(probably) no perfect matchings."
- The error probability is now reduced to at most 2^{-k}.

Lószló Lovász (1948–)

Remarks[a]

- Note that we are calculating

$$\text{prob}[\,\text{algorithm answers "no"} \mid G \text{ has no perfect matchings}\,],$$

$$\text{prob}[\,\text{algorithm answers "yes"} \mid G \text{ has a perfect matching}\,].$$

- We are *not* calculating[b]

$$\text{prob}[\,G \text{ has no perfect matchings} \mid \text{algorithm answers "no"}\,],$$

$$\text{prob}[\,G \text{ has a perfect matching} \mid \text{algorithm answers "yes"}\,].$$

---

[a] Thanks to a lively class discussion on May 1, 2008.

[b] *Numerical Recipes in C* (1988), "statistics is *not* a branch of mathematics!" Similar issues arise in MAP (maximum a posteriori) estimates and ML (maximum likelihood) estimates.

But How Large Can $\det(A^G(i_{11}, \ldots, i_{nn}))$ Be?

- It is at most[a]

$$n!\,\left(2n^2\right)^n.$$

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

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

bits are sufficient for representing the determinant.

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

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

An Intriguing Question[a]

- Is there an $(i_{11}, \ldots, i_{nn})$ that will always give correct answers for the algorithm on p. 525?
- A theorem on p. 620 shows that such an $(i_{11}, \ldots, i_{nn})$ exists!
  - Whether it can be found efficiently is another matter.
- Once $(i_{11}, \ldots, i_{nn})$ is available, the algorithm can be made deterministic.

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