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 $\Phi(p) = \{1, 2, \ldots, p-1\}$ that have exponent k.
- We already knew that $R(k) = 0$ for $k \not| (p-1)$.
- So

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

as every number has an exponent.

Size of $R(k)$

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

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

- By Lemma 61 (p. 485) 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 $R(k)$ (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 58 (p. 472).

- 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 53 (p. 457).

A Few Calculations

- Let $p = 13$.
- From p. 482 $\phi(p-1) = 4$.
- 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.

The Other Direction of Theorem 53 (p. 457)

- We show p is a prime if there is a number r 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 (continued)

- So we proceed to show $r^{(p-1)/q} = 1 \bmod p$ for some prime divisor q of $p-1$.
- $r^{\phi(p)} = 1 \bmod p$ by the Fermat-Euler theorem (p. 482).
- Because p is not a prime, $\phi(p) < p - 1$.
- Let k be the smallest integer such that $r^k = 1 \bmod p$.
- With the 1st condition, it is easy to show that $k \,|\, (p-1)$ (similar to p. 485).
- Note that $k \,|\, \phi(p)$ (p. 485).
- As $k \leq \phi(p)$, $k < p - 1$.

The Proof (concluded)

- Let q be a prime divisor of $(p-1)/k > 1$.
- Then $k \mid (p-1)/q$.
- By the definition of k,

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

- But this violates the 2nd condition.

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 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.
- 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 (recall p. 266 and p. 270).
- 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.

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 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 and Strassen (1977).
[b] "Maximal" (a local maximum) not "maximum" (a global maximum).

"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).

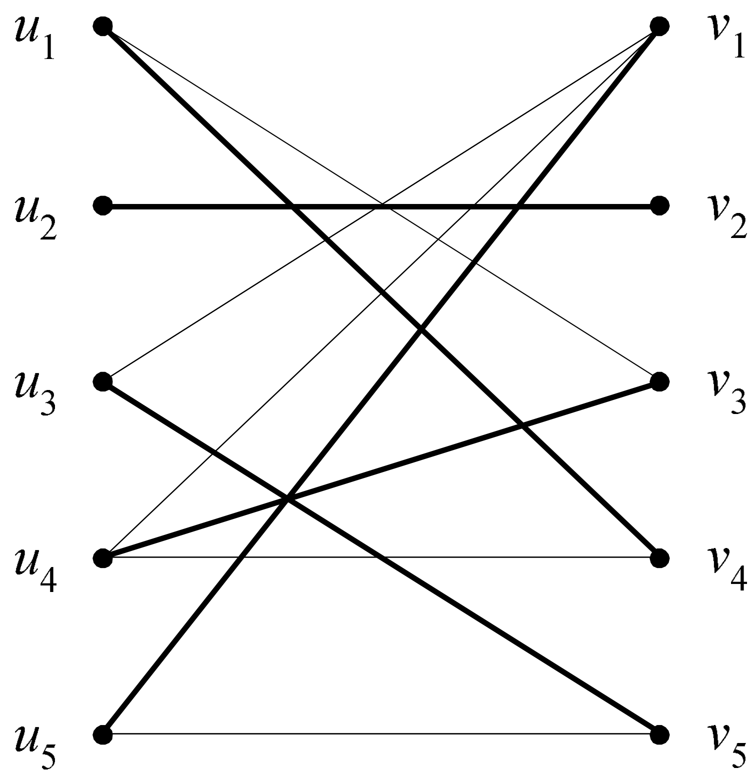
[d] Schwartz (1980); Zippel (1979).

[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 $i \in \{1, 2, \ldots, n\}$.
- 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. 508 is[a]

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

[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.
  - Equivalently, $\operatorname{sgn}(\pi) = 1$ if the number of (i, j)s such that $i < j$ and $\pi(i) > \pi(j)$ is even.[a]

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

---

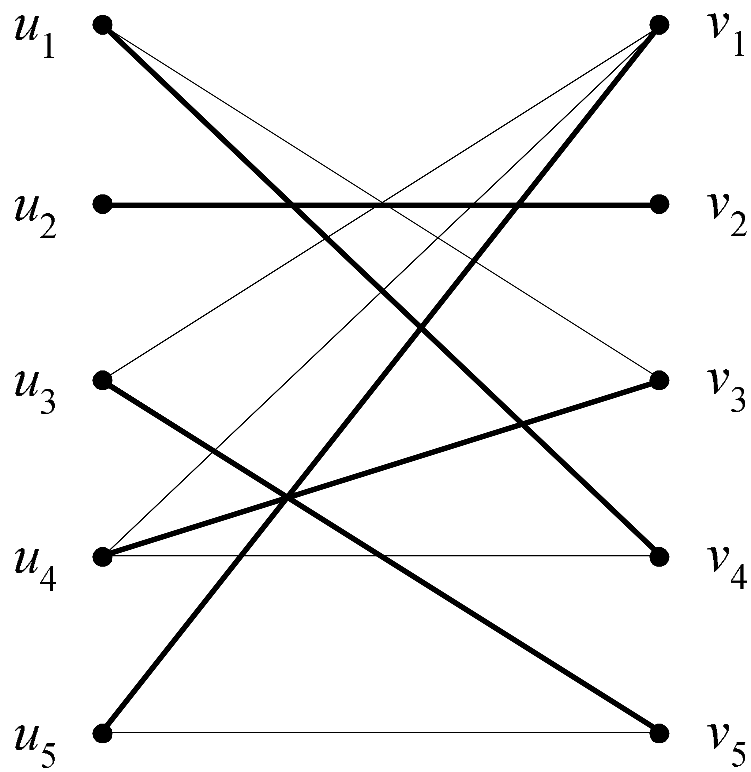
[a]Contributed by Mr. Hwan-Jeu Yu (`D95922028`) on May 1, 2008.

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

Perfect Matching and Determinant (p. 508)



Perfect Matching and Determinant (concluded)

- The matrix is (p. 510)

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

- $\det(A^G) = -x_{14}x_{22}x_{35}x_{43}x_{51} + x_{13}x_{22}x_{35}x_{44}x_{51} + x_{14}x_{22}x_{31}x_{43}x_{55} - x_{13}x_{22}x_{31}x_{44}x_{55}$.
- 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 63 (Schwartz (1980)) *Let* $p(x_1, x_2, \ldots, x_m) \not\equiv 0$ *be a polynomial in* m *variables each of degree at most* d*. Let* $M \in \mathbb{Z}^+$*. Then the number of* m*-tuples*

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

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

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

- By induction on m (consult the textbook).

Density Attack

- The density of roots in the domain is at most

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

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

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

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

Density Attack (concluded)

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

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

Analysis

- If $p(x_1, x_2, \ldots, x_m) \equiv 0$, the algorithm will always be correct as $p(i_1, i_2, \ldots, i_m) = 0$.
- Suppose $p(x_1, x_2, \ldots, x_m) \not\equiv 0$.
  - The algorithm will answer incorrectly with probability at most md/M by Eq. (9) on p. 517.
- 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. 517.
- 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!”

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

- It is at most
$$n!\left(2n^2\right)^n.$$
- Stirling's formula says $n! \sim \sqrt{2\pi n}\,(n/e)^n$.
- Hence
$$\log_2 \det(A^G(i_{11}, \ldots, i_{nn})) = O(n \log_2 n)$$
bits are sufficient for representing the determinant.
- We skip the details about how to make sure that all *intermediate* results are of polynomial sizes.

An Intriguing Question[a]

- Is there an $(i_{11}, \ldots, i_{nn})$ that will always give correct answers for the algorithm on p. 520?
- A theorem on p. 625 shows that such an $(i_{11}, \ldots, i_{nn})$ exists!
  - Whether it can be found efficiently is another matter.
- Once $(i_{11}, \ldots, i_{nn})$ is available, the algorithm can be made deterministic.

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

Randomization vs. Nondeterminism[a]

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

---

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

Monte Carlo Algorithms[a]

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

[a]Metropolis and Ulam (1949).

Monte Carlo Algorithms (continued)

- The algorithm makes a false negative with probability ≤ 0.5.[a]
  - Note this probability refers to[b]

$$\text{prob}[\,\text{algorithm answers "no"} \,|\, G \text{ has a perfect matching}\,]$$

not

$$\text{prob}[\,G \text{ has a perfect matching} \,|\, \text{algorithm answers "no"}\,].$$

---

[a] Equivalently, among the coin flip sequences, at most half of them lead to the wrong answer.

[b] In general, prob[algorithm answers "no" | input is a "yes" instance].

Monte Carlo Algorithms (concluded)

- This probability 0.5 is *not* over the space of all graphs or determinants, but *over* the algorithm's own coin flips.
  - It holds for *any* bipartite graph.
- In contrast, to calculate

  $$\text{prob}[\, G \text{ has a perfect matching} \,|\, \text{algorithm answers "no"} \,],$$

  we will need the distribution of G.
- But it is an empirical statement that is very hard to verify.

The Markov Inequality[a]

Lemma 64 *Let x be a random variable taking nonnegative integer values. Then for any $k > 0$,*

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

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

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

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

Andrei Andreyevich Markov (1856–1922)

An Application of Markov's Inequality

- Suppose 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.
  - Here, we treat C as a black box without going into its internal code.[a]
- By Markov's inequality, this new algorithm runs in time $kT(n)$ and gives the wrong answer with probability $\leq 1/k$.

---

[a] Contributed by Mr. Hsien-Chun Huang (`R03922103`) on December 2, 2014.

An Application of Markov's Inequality (concluded)

- By running this algorithm m times (the total running time is $mkT(n)$), we reduce the error probability to $\leq k^{-m}$.[a]
- 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 much worse than the previous algorithm's error probability of $\leq k^{-m}$ for the same amount of time.

[a] With the same input. Thanks to a question on December 7, 2010.

FSAT for k-SAT Formulas (p. 496)

- 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 unsatisfied 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 answer "unsatisfiable."
- The random walk algorithm needs expected exponential time for 3SAT.
  - In fact, it runs in expected $O((1.333\cdots + \epsilon)^n)$ time with $r = 3n$,[a] much better than $O(2^n)$.[b]
- We will show immediately that it works well for 2SAT.
- The state of the art as of 2006 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 65 *Suppose the random walk algorithm with $r = 2n^2$ is applied to any satisfiable* 2SAT *problem with n variables. Then a satisfying truth assignment will be discovered with probability at least 0.5.*

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

---

[a] Papadimitriou (1991).

The Proof

- Let $t(i)$ denote the expected number of repetitions of the flipping step[a] until a satisfying truth assignment is found.
- It can be shown that $t(i)$ is finite.
- $t(0) = 0$ because it means that $T = \hat{T}$ and hence $T \models \phi$.
- If $T \neq \hat{T}$ or any other satisfying truth assignment, then we need to flip the coin at least once.
- We flip a coin to pick among the 2 literals of a clause not satisfied by the present T.
- At least one of the 2 literals is true under $\hat{T}$ because $\hat{T}$ satisfies all clauses.

[a] That is, Statement 7.

The Proof (continued)

- So we have at least 0.5 chance of moving closer to $\hat{T}$.
- Thus
$$t(i) \leq \frac{t(i-1)+t(i+1)}{2}+1$$
for $0 < i < n$.
  - Inequality is used because, for example, T may differ from $\hat{T}$ in both literals.
- It must also hold that
$$t(n) \leq t(n-1)+1$$
because at $i = n$, we can only decrease i.

The Proof (continued)

- Now, put the necessary relations together:

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

- Technically, this is a one-dimensional random walk with an absorbing barrier at $i=0$ and a reflecting barrier at $i=n$ (if we replace "$\leq$" with "$=$").[a]

[a] The proof in the textbook does exactly that. But a student pointed out difficulties with this proof technique on December 8, 2004. So our proof here uses the original inequalities.

The Proof (continued)

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

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

- Simplify it to yield

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

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

The Proof (continued)

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

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

- Simplify it to yield

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

by Eq. (13) on p. 541.

The Proof (continued)

- Continuing the process, we shall obtain

$$t(i) \leq 2in - i^2.$$

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

$$t(n) \leq n^2.$$

- We conclude that

$$t(i) \leq t(n) \leq n^2$$

for $0 \leq i \leq n$.

The Proof (concluded)

- So the expected number of steps is at most n^2.
- The algorithm picks $r = 2n^2$.
  - This amounts to invoking the Markov inequality (p. 530) with $k = 2$, resulting in a probability of 0.5.[a]
- The proof does *not* yield a polynomial bound for 3SAT.[b]

---

[a] Recall p. 532.

[b] Contributed by Mr. Cheng-Yu Lee (`R95922035`) on November 8, 2006.

Boosting the Performance

- We can pick $r = 2mn^2$ to have an error probability of

$$\leq \frac{1}{2m}$$

by Markov's inequality.

- Alternatively, with the same running time, we can run the "$r = 2n^2$" algorithm m times.
- The error probability is now reduced to

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