Introduction to Randomized Algorithms

Frédéric Magniez
LIAFA, Université Paris Diderot

Problem

- Input: x
- Output:

  ACCCEPT/REJECT

  F(x)

  Any y such that x R y

Algorithm

Goal

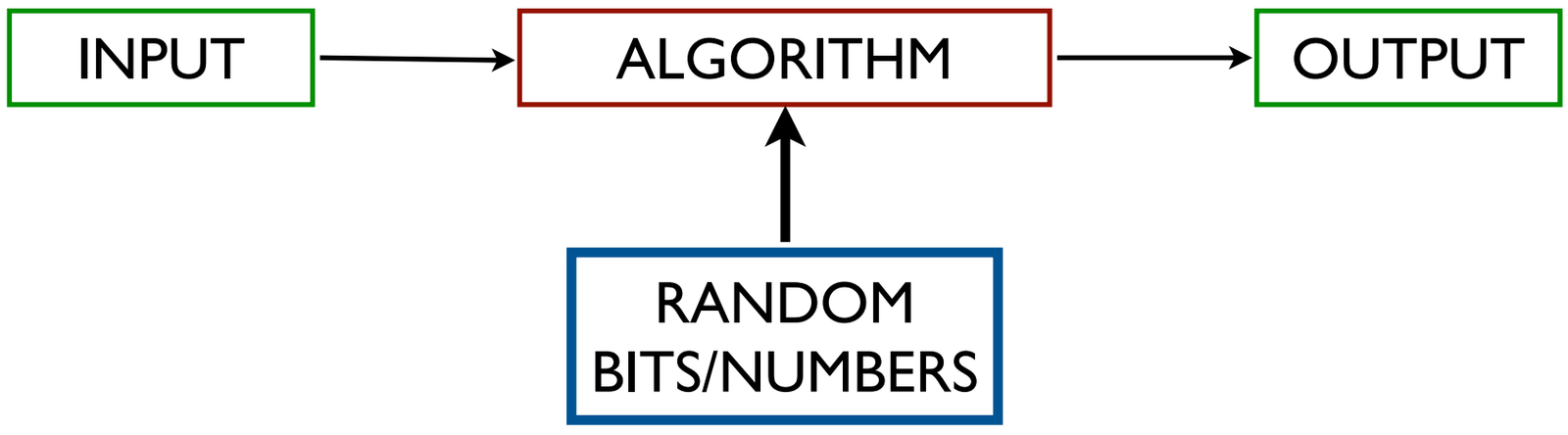
- Find an algorithm that solves the problem

  correctly on every input

  quickly

    Example: linear time, polynomial time (in the input size)

Algorithm



- Main input x
- Additional input **r**: source of random bits or numbers
- Behavior and output depend on x and **r**

Goal

- Find an algorithm such that, **on every input x**,

  Output is correct, for most of random choices r — MONTE CARLO

  Complexity is small, for all random choices r

  Output is correct, for all of random choices r

  Complexity is small, in average over random choices r — LAS VEGAS

Are they useful?

- Compare a deterministic algorithm with runtime 1h
  1. to an algorithm with running time 1min that
     a. aborts with probability 1/3, 1/2, 0.9
     b. fails with probability 1/3, 1/2, 0.9
  2. to an algorithm that is always correct but
     whose running time is random and 1min in average

What does random mean?

- Source of randomness
  bits: 0, 0, 1, 0, 1, 0 ....
  integers in some give interval [a,b]
  → Behavior can vary on a fixed input
- Always independent and uniform
- Given for free.... How to generate random bits/integers?
  Important problem .... But NOT OUR PURPOSE
  Derandomization, Pseudo-random generators, ...

≠ Probabilistic analysis of deterministic algorithms

Algorithm

- Algorithm is deterministic
- Input is sample from some known probabilistic distribution

Case 1: Correctness

- Output is correct for most inputs (w.r.t. input distribution)

Case 2: Efficiency

- Algorithm is fast in average (w.r.t. input distribution)

Duality (Yao)

- Best MONTE-CARLO Algorithm ≃ Best Case-1 Algorithm
- Best LAS-VEGAS Algorithm ≃ Best Case-2 Algorithm

Typical example

- QUICKSORT

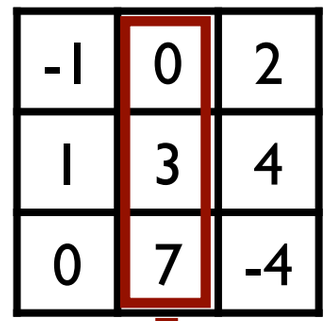
MONTE-CARLO ALGORITHMS

Problem

- Input: 3 matrices A,B,C of size n x n (over some ring)
- Output:

  ACCEPT if A x B = C

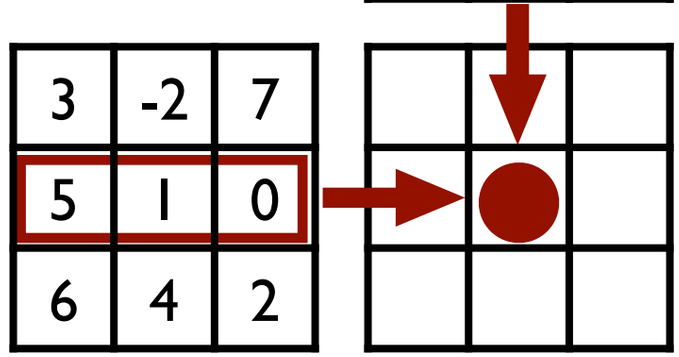
  REJECT otherwise



Trivial solution

- Compute A x B

  and compare with C



- Naive algorithm: $O(n^3)$ add/mult
- Best current algorithm: $O(n^{2.3727})$ add/mult
- Only the trivial lower bound $\Omega(n^2)$ is known

Freivalds test for Matrix Multiplication

Problem

- Input: 3 matrices A,B,C of size n x n (over some ring)
- Output:

  ACCEPT if A x B = C

  REJECT otherwise

Algorithm

- Take a random vector r in $\{0,1\}^n$
- Compute u = Br, and then v = Au
- Compute w = Cr
- ACCEPT if v=w, and REJECT otherwise

Fact

- Time complexity: $O(n^2)$ add/mult
- If AB=C, Algorithm always accepts

Lemma

- If $AB \neq C$ then $A(Br) \neq Cr$ with probability $\geq 1/2$ when r is random in $\{0,1\}^n$

Proof

- Set $D=AB-C$.
- Then $D \neq 0$. Say w.l.o.g. that $D_{11} \neq 0$

  $(Dr)_1 = D_{11} r_1 + D_{12} r_2 + \ldots + D_{1n} r_n = D_{11} r_1 + f(r_2,r_3,\ldots,r_n)$
- Therefore

  $Pr_{r1}[\ D_{11} r_1 = - f(r_2,r_3,\ldots,r_n)\ ] \leq 1/2$, for every fixed $r_2,r_3,\ldots,r_n$

Theorem

- Algorithm has time complexity $O(n^2)$

  and is ONE-SIDED ERROR

  If $AB=C$, Algorithm always accepts

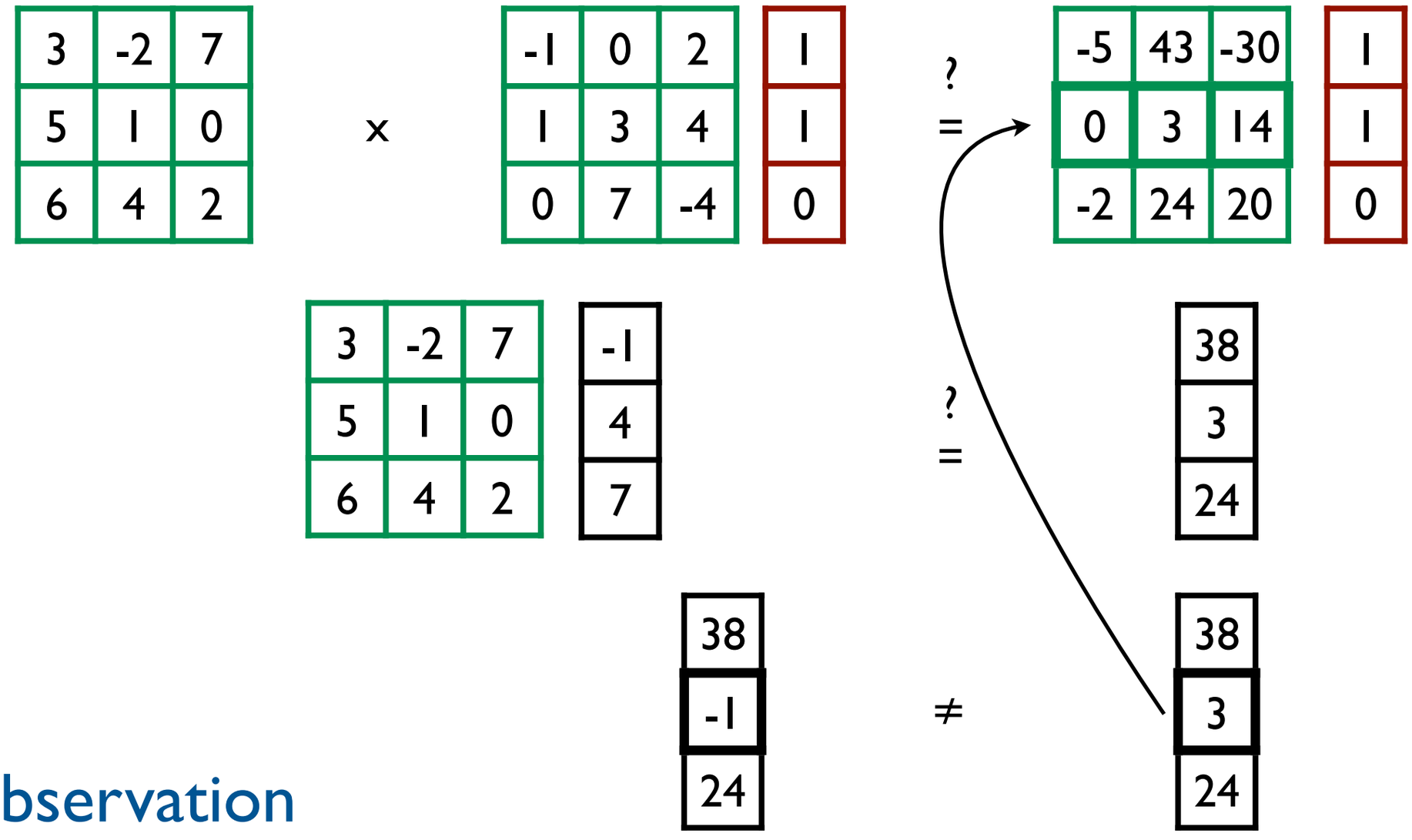
  If $AB \neq C$, Algorithm rejects with probability $\geq 1/2$

Error reduction

- Iterate 100 times: error probability $\leq 1/2^{100}$

  (age of universe $< 2^{59}$ sec)

Running the algorithm



Observation

- If Algorithm rejects

  one can find in $O(n^2)$ an error in C

Problem

- Input: access to a program **P** supposed to multiply matrices

  P is deterministic

  P is supposed to realize matrix multiplication

  P's input: 2 matrices A,B of size n x n

  P's output: 1 matrix **P**(A,B) of size n x n

- Output: Estimate the correctness of **P**

Freivalds test

- Run **P** only once
- Test the correctness of **P** on a specific input

  within negligible additional time complexity (in $O(n^2)$)

Questions

- Can we estimate the error rate of **P**?

  $err(\mathbf{P}) = Pr_{A,B}\ (\ \mathbf{P}(A,B) \neq A \times B\ )$

- What can we do if **P** is incorrect in (A,B)?

Problem

- Input: access to a program **P**

  supposed to realize matrix multiplication modulo some integer M

- Output: Estimate error rate of **P**

  $err(\mathbf{P}) = Pr_{A,B}\ (\ \mathbf{P}(A,B) \neq A \times B\ )$

Algorithm

- Do 100 times

  Take random matrices A,B

  Perform 10 Freivalds tests on A,B, and C=**P**(A,B)

  Declare that P fails if one of the 10 tests fails

- ACCEPT if P failed at most 1/10 of the time

Analysis

- Run **P** only O(1) times, with additional time complexity $O(n^2)$
- From Chernoff Bounds:

  If $err(\mathbf{P}) \leq 1/11$, then Algorithm ACCEPTS with high probability

  If $err(\mathbf{P}) \geq 1/9$, then Algorithm REJECTS with high probability

Application to Program Correcting

Problem

- Input: access to a program **P** such that err(**P**)≤1/9

  Matrices A,B
- Output: A x B

Self-reducibility

- For every matrices A,B,R,S:

  AB = (A-R + R)(B-S + S) = (A-R)(B-S) + (A-R)S + R(B-S) + RS

Algorithm

- Take random matrices R, S
- Compute and test P(A-R,B-S), P(A-R,S), P(R,B-S) and P(R,S)
- If one of the test fails, try again
- Else output the sum of the 4 computed matrices

Fact

- Output is correct with hight probability
- Run **P** only O(1) times, with an extra time complexity $O(n^2)$

Problem

- Input: 2 polynomials $Q(X_1,X_2,...,X_n)$ and $R(X_1,X_2,...,X_n)$ over integers with degree $\leq d$
- Output:

  ACCEPT if Q=R

  REJECT otherwise

Remark

- Q and R are given in such a way that they can be easily evaluated

  Example: $Q = \prod_{i<j}(X_j-X_i)$ $R = \mathrm{Det}\,(X_i^j)_{i,j}$

  arithmetic circuit
- Complexity: # of evaluations of Q and R
- Checking Q=R can be done by comparing their resp. coefficients

  Issue: Expanding Q and R can take exponential time!

Lemma

- $P(X_1,X_2,...,X_n)$: **non-zero** polynomial of degree d (over some field)
- S: subset (of the field)

  then $P(a_1,a_2,...,a_n)=0$ with probability $\leq d/|S|$

  when $a_1,a_2,...,a_n$ are uniformly random in S

Proof

- n=1: P can have only d roots
- n>1: P can be seen as a polynomial in X_1 of degree $j \leq d$

  $P(X_1,X_2,...,X_n) = S_0(X_1,X_2,...,X_{n-1}) + X_n S_1(X_1,X_2,...,X_{n-1})$
  $+ ... + X_n^j S_j(X_1,X_2,...,X_{n-1})$

  such that $S_j \neq 0$
- By induction: $S_j(a_1,a_2,...,a_{n-1})=0$ with probability $\leq (d-j)/|S|$
- Fix $a_1,a_2,...,a_{n-1}$ such that $S_j(a_1,a_2,...,a_{n-1})\neq 0$

  then $P(a_1,a_2,...,a_n)=0$ with probability $\leq j/|S|$ when a_1 is random in S
- Total: $P(a_1,a_2,...,a_n)=0$ with probability $\leq d/|S|$

Randomized algorithm for Polynomial Identity Testing

Problem

- Input: 2 polynomials $Q(X_1,X_2,...,X_n)$ and $R(X_1,X_2,...,X_n)$ over integers
  with degree $\leq d$
- Output:
  ACCEPT if Q=R
  REJECT otherwise

Algorithm

- Chose random $a_1,a_2,...,a_n$ in $\{1,2,...,2d\}$
- ACCEPT if $Q(a_1,a_2,...,a_n) = R(a_1,a_2,...,a_n)$ → **ONLY 2 EVALUATIONS**
- REJECT otherwise

Analysis

- If Q=R, Algorithm always accepts
- If Q≠R, Algorithm rejects with probability $\geq 1/2$
  ONE-SIDED ERROR

Randomized algorithm for Polynomial Identity Testing

Problem

- Input: 2 polynomials $Q(X_1,X_2,...,X_n)$ and $R(X_1,X_2,...,X_n)$ over integers
  with degree $\leq d$ and $|\text{coefficients}| < M$
- Output:
  ACCEPT if Q=R
  REJECT otherwise

Algorithm

- Find a prime p in [min(M,2d), max(2M,4d)]
- Chose random $a_1,a_2,...,a_n$ in {1,2,...,2d}
- ACCEPT if $Q(a_1,a_2,...,a_n) = R(a_1,a_2,...,a_n) \bmod p$ → 2 evaluations mod p
- REJECT otherwise

Analysis

- ONE-SIDED ERROR
  error probability $\leq 1/2$

Example

Input

- $Q_1 = X^6 - Y^6$
- $Q_2 = X^6 + Y^6$
- $R = (X^2+Y^2)(X^4-X^2Y^2+Y^4)$

Remark

- $P = Q_1$ over reals iff $P = Q_1$ mod p, with $p \geq 3$ (same for $P=Q_1$)
- Degree = 6 → Set p= 7
- One-sided error probability ≤ 6/7
- X=3, Y=2

  $Q_1(3,2) = 0$ mod 7

  $Q_2(3,2) = 2$ mod 7

  $R(3,2) = 2$ mod 7

Conclusion

- For sure $Q_1 \neq R$
- Maybe $Q_2 = R$

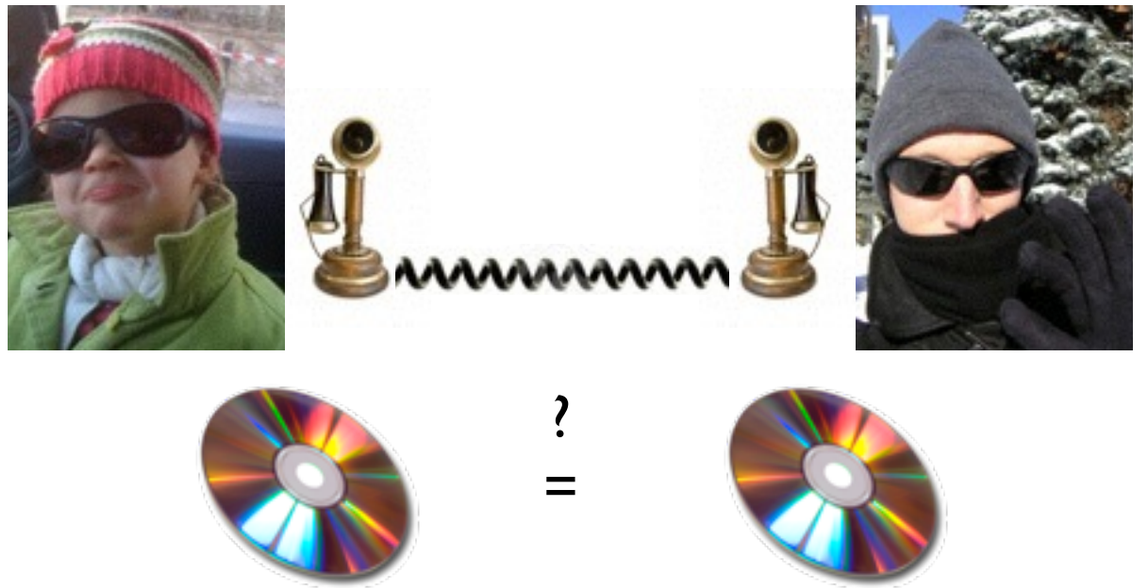
Application to Fingerprint

Problem

- Two players: Alice and Bob
- Alice's input: n-bit string u
- Bob's input: n-bit string v
- Output:

  ACCEPT if u=v

  REJECT otherwise



Remark

- Alice can send u to Bob → n bits

  Issue: u can be very huge, and the communication slow
- Constraint: Alice and Bob can only exchange O(log n) bits

Polynomial encoding

- $U(X)= u_0 + u_1X + ... + u_{n-1}X^{n-1}$ $V(X)= v_0 + v_1X + ... + v_{n-1}X^{n-1}$
- Example: u=0011010 → $U(X)= X^2+X^3+X^5$
- Then U(X)=V(X) iff u=v (this is true modulo any prime $p \geq 2$)

Application to Fingerprint

Problem

- Alice's input: n-bit string u
- Bob's input: n-bit string v
- Output:

  ACCEPT if u=v, and REJECT otherwise

Algorithm

- Alice finds and sends some prime number p in $[n^2,2n^2]$
- Alice chooses and sends at some random integer a in [1,p]
- Alice computes and sends (U(a) mod p)
- Bob computes and sends (V(a) mod p)
- They ACCEPT if U(a)=V(a) mod p, and REJECT otherwise

Analysis

- Alice and Bob exchange O(log n) bits
- Time complexity: n add/mult modulo p
- ONE SIDED ERROR: error probability $\leq$ 1/n

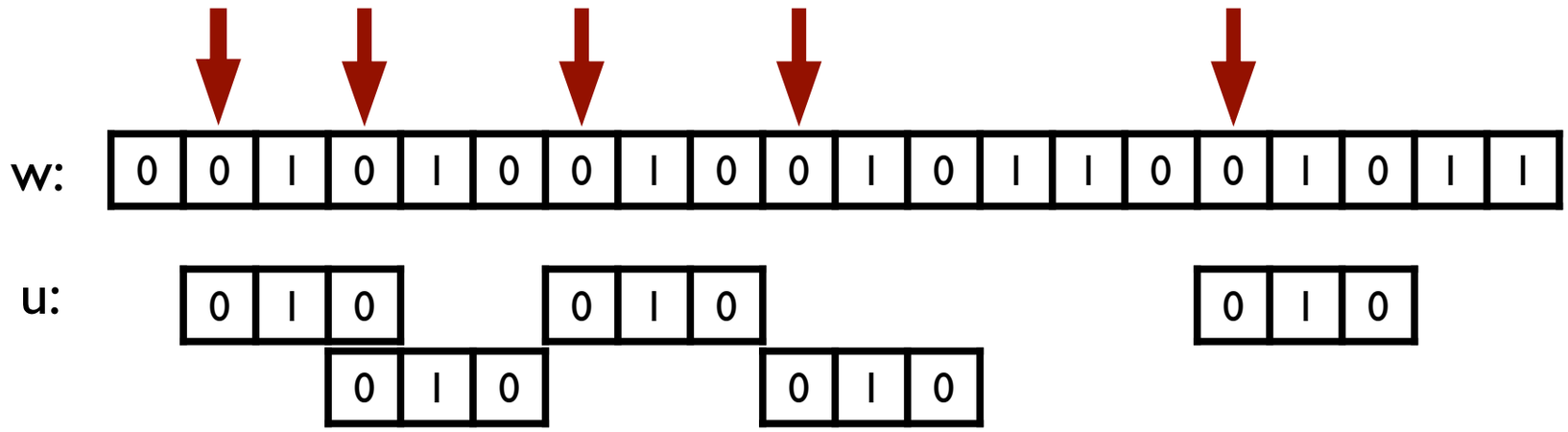
Problem

- Input:

  n-bit string w (the sequence)

  k-bit string u (the pattern)

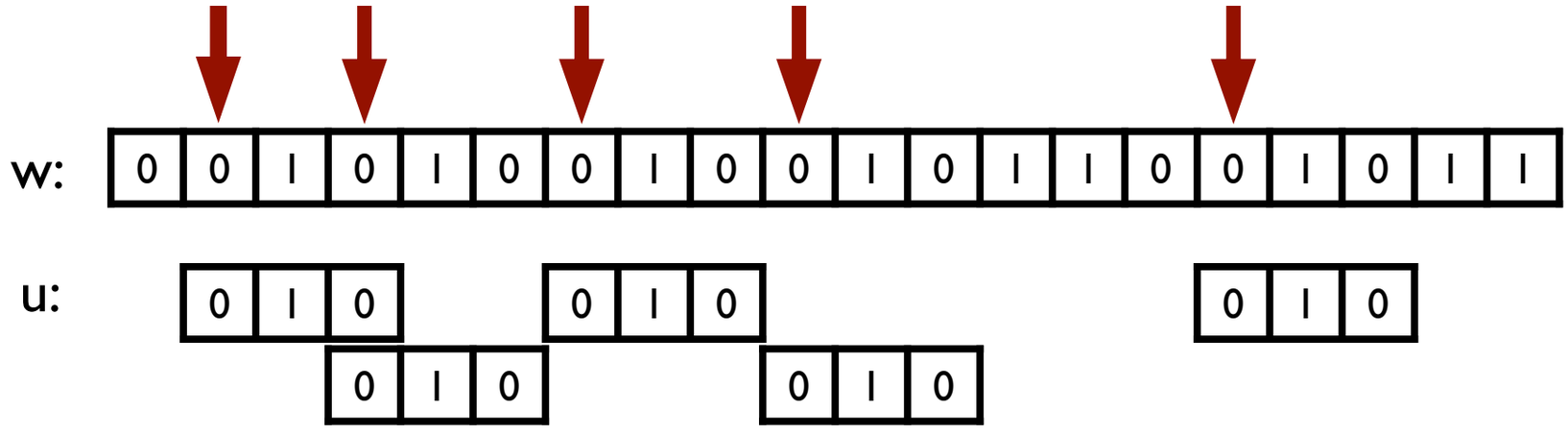
- Output: All positions i such that w[i,i+k-1] = u



Algorithm

- Apply previous algorithm on u and v=w[i,i+k-1], for all i=0,1,...,n-k

Application to Pattern Matching



Algorithm

- Find some prime number p in $[n^3,2n^3]$
- Choose at random some integer a in [1,p]
- Compute (U(a) mod p)
- For i=0 to n-k

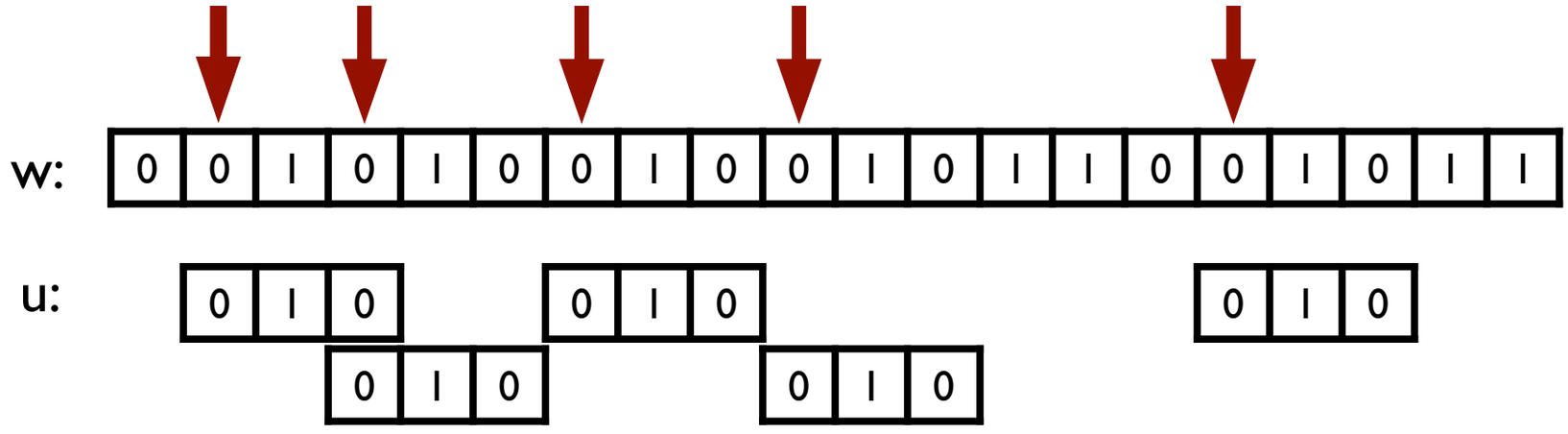
  Compute (V_i(a) mod p) (where V is the polynomial enc of w[i,i+k-1])

  Output i if U(a)=V_i(a) mod p

Analysis

- ONE SIDED ERROR: error probability $\leq n/n^2=1/n$
- Time complexity: (n **x** k) add/mult modulo p
- Observation: $V_i(a) = w[i] + a\,(V_{i+1}(a) - w[i+k]a^{k-1}) \bmod p$

Application to Pattern Matching



Algorithm

- Find some prime p in $[n^3, 2n^3]$
- Choose at random some integer a in [1,p]
- Compute (U(a) mod p) and (V_{n-k}(a) mod p)
- For i=n-k downto 0

  Update V_{i+1}(a) to V_i(a) (except for i=n-k)

  Output i if U(a)=V_i(a) mod p

Analysis

- ONE SIDED ERROR: error probability $\leq n/n^2 = 1/n$
- Time complexity: (n + k) add/mult modulo p

How to find a prime?

Problem

- Input: integer $n \geq 1$
- Output: a prime in [n,2n]

Facts

- # {primes $\leq$ n} $\approx$ n/log n
- There is a randomized algorithm for primality testing
  with complexity O(log n) add/mult modulo n
- There is a deterministic algorithm for primality testing
  with complexity $O((\log n)^5)$ add/mult modulo n

Algorithm

- Chose at random p in [n,2n]
- Run 100 times ONE-SIDED ERROR Primality Testing
- OUTPUT p if all tests accept, RESTART otherwise

Analysis

- Algorithm outputs a prime number w.h.p after O(log n) iterations

Bipartite perfect matching

Problem

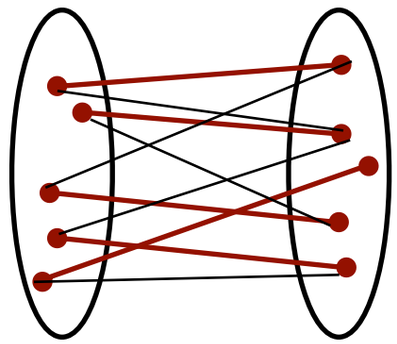
- Input:

  Bipartite graph G between vertex sets U and V of size n

- Output:

  ACCEPT if there is a perfect matching in G

  REJECT otherwise



Facts

- There is a deterministic algorithm

  with time complexity $\sqrt{n}$ x # edges

Theorem

- There is a randomized algorithm with time complexity

  the computation of O(1) determinants modulo p in [2n,4n]

  [Lovasz 1979]

Reminder

- Computing determinant reduces to matrix multiplication

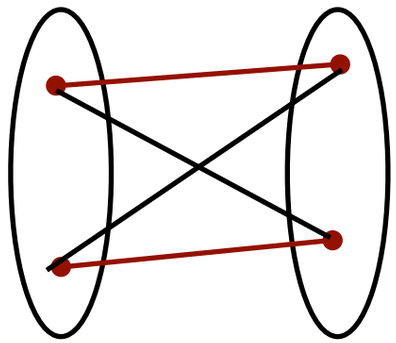
  Current best time complexity: $O(n^{2.3727})$

Adjacency matrix

- Matrix A s.t $(A)_{ij} = 1$ if (i,j) is an edge in G
  0 otherwise
- Perfect matching M ↔ permutation σ such that $M = \{ (i,\sigma(i)) : i \text{ in } [1,n] \}$

Fact

- $\text{perm}(A) = \sum_\sigma \prod_i (A)_{i\sigma(i)}$
  is the # of perfect matchings in G



$$A = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

$$\text{perm}(A) = 2$$

Theorem

- Permanent is #P-complete

Reminder

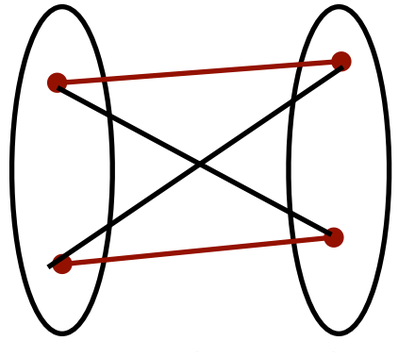
- #P={counting problems whose decision problems are in NP}
- Polynomial hierarchy is included in $P^{\#P}$

Fact

- $\det(A) = \sum_\sigma \mathrm{sgn}(\sigma) \prod_i (A)_{i\sigma(i)}$

  can be computed efficiently

  but can be 0 when # perfect matchings is even



$$A = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

$$\det(A) = 0$$

Tutte matrix

- $(T)_{ij} = X_{ij}$ if (i,j) is an edge in G

  0 otherwise

- det(T) is a polynomial in X_{ij}

  with degree $\leq$ n

  coefficients in {-1,0,1}

$$T = \begin{pmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{pmatrix}$$

$$\det(T) = X_{11}X_{22} - X_{12}X_{21}$$

Fact

- det(T) = 0 iff G has no perfect matching

Algorithm

- Evaluate det(T) in random $a_1, a_2, \ldots, a_n$ mod p

Algorithm

- Find some prime number p in $[n^2, 2n^2]$
- Choose at random integers a_{ij} in [1,p], for $1 \le i,j \le n$
- Set $R = T(a_{ij})$
- ACCEPT if $\det(R)=0 \bmod p$, and REJECT otherwise

Analysis

- Complexity: 1 determinant
- ONE-SIDED ERROR:

  error probability $\le n/n^2 \le 1/n$

Generalization

- Maximum matching reduces to matrix inversion (in general graphs)

  [Mulmuley, Vazirani, Vazirani 1987]

LAS-VEGAS ALGORITHMS

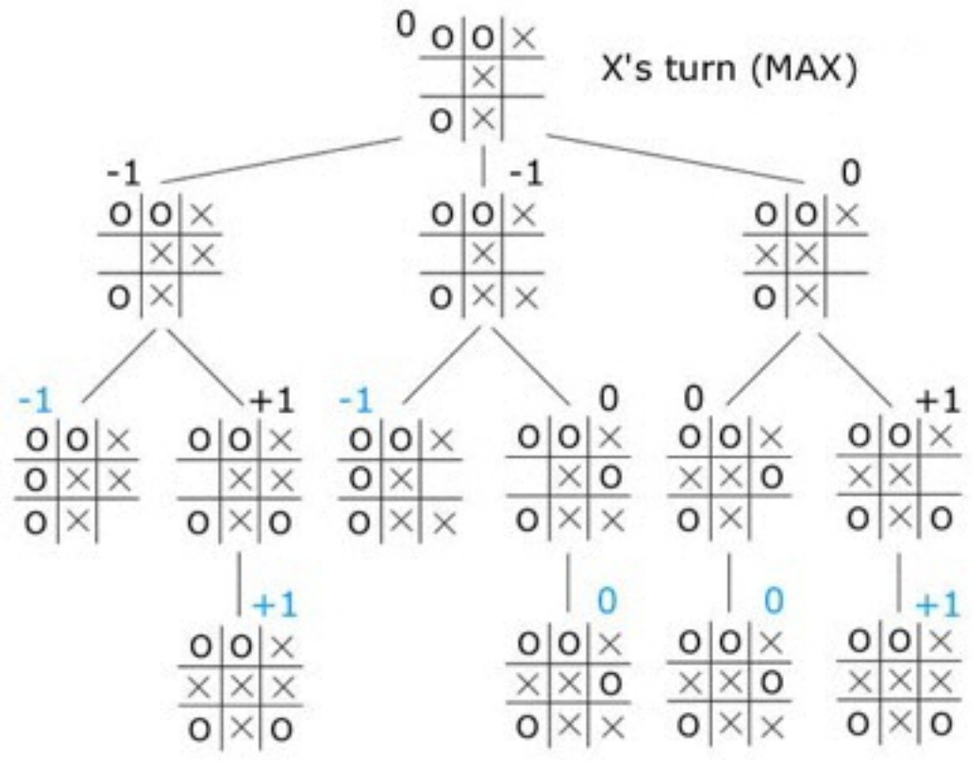
Game-tree evaluation

Game

- 2 players
- Zero sum: when one player wins (+1), the other loses (-1)
- Example: tic tac toe, checkers, othello, chess, go, ...

Tree

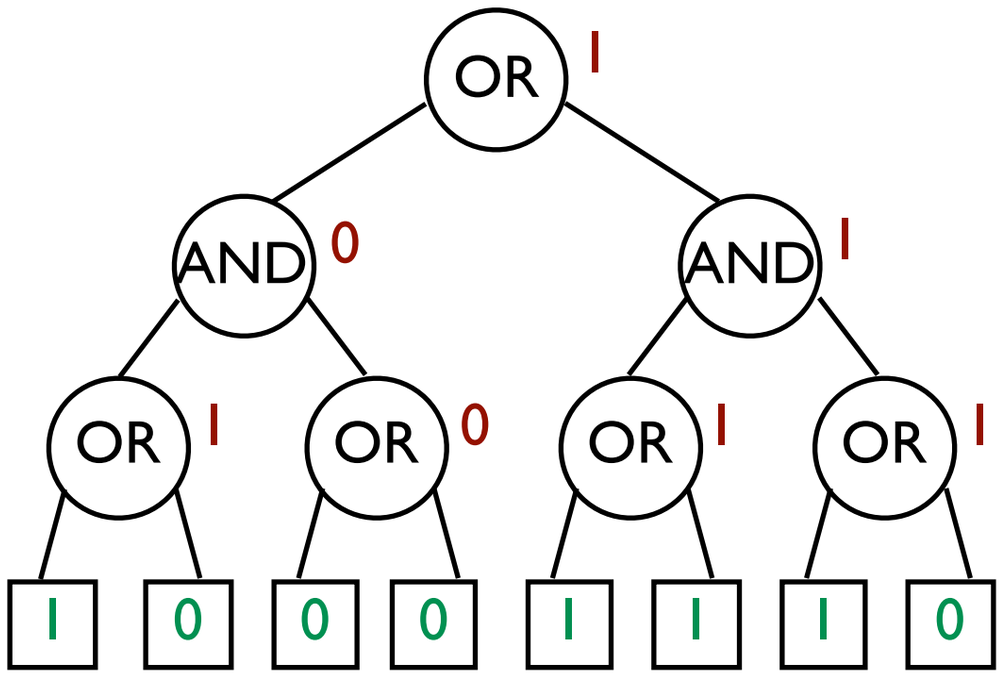
- Represents the possible evolutions of the game
- Leaves are valued according to the final score
- X tries to maximize the outcome, whereas O tries to minimize it



Tree

- Complete binary tree with height h, and $n=2^h$ leaves
- Leaves are valued by 0/1

  and encoded by a n-bit string x in $\{0,1\}^n$
- Nodes: MIN → AND MAX → OR
- Value of the root: AND-OR(x) or OR-AND(x)



OR-AND(10001110)=1

Problem

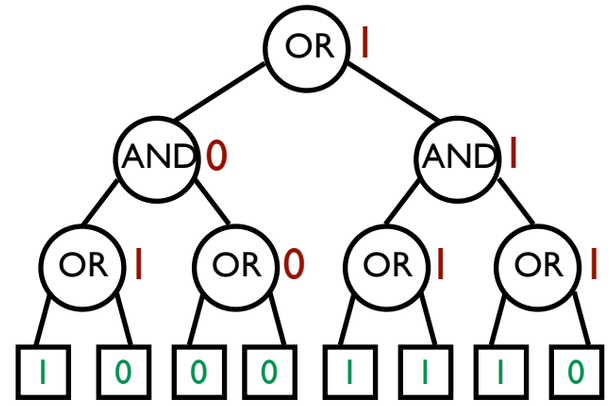
- Input:

  Access to a n-bit string x

- Output:

  Evaluate AND-OR(x)

  or OR-AND(x)



OR-AND(10001110)=1

- Complexity: # of evaluations of leaves = # of bits evaluated in x

Fact

- Deterministic algorithms must evaluate all n leaves, in the worst case

Theorem

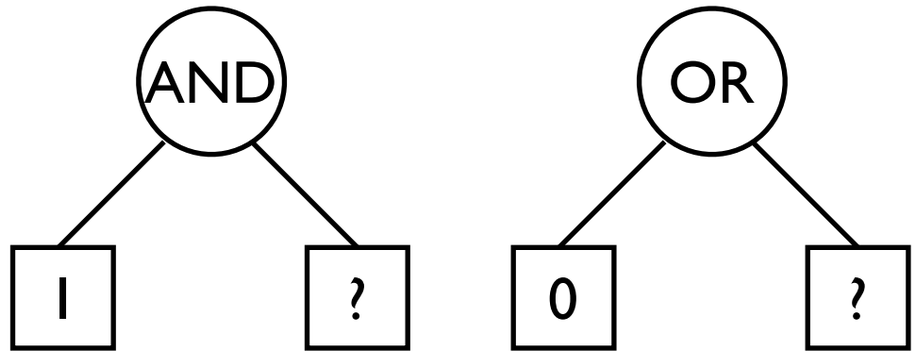
- There is a MONTE-CARLO algorithm evaluating $3^{h/2}=n^{0.792}$ leaves [Snir'85]
- This algorithm is optimal [Saks and Wigderson'86]

Algorithm

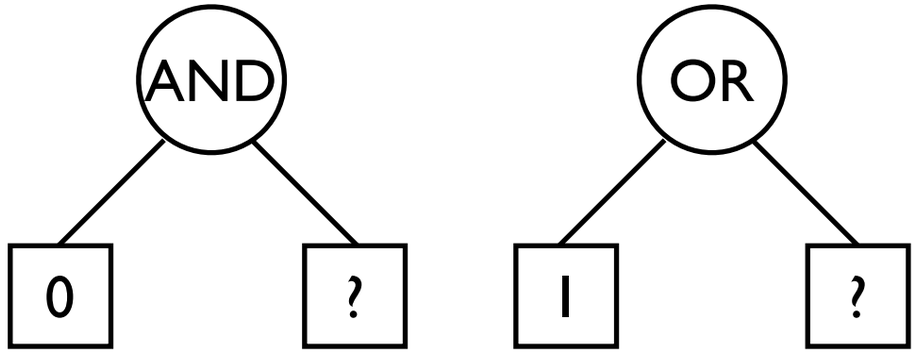
- Recursively evaluate a random child
- If necessary, evaluate the other child

Analysis

- Unlucky cases

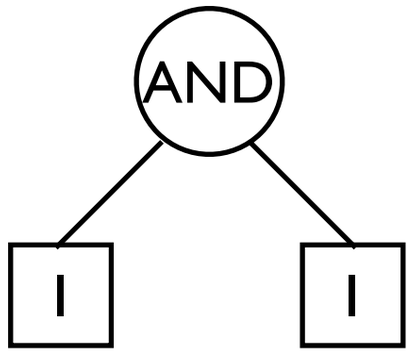


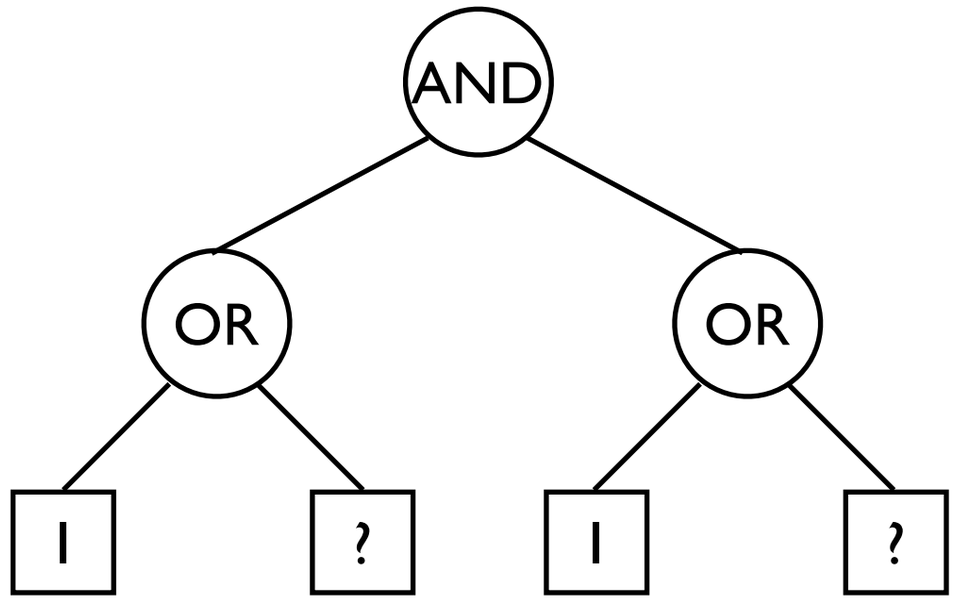
- Lucky cases



Root evaluates to 1

- Both OR-children evaluate to 1
- Each OR-children has a child that evaluate to 0



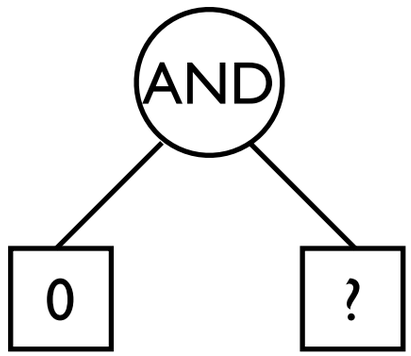


Therefore

$$T_h(\text{AND-OR}) = 2 \times (1+1/2) \times T_{h-2}(\text{AND-OR})$$

Root evaluates to 0

- 1 OR-children evaluate to 0
- Each OR-children has a child that evaluate to 0



Therefore

$$T_h(\text{AND-OR}) = (1+1/2)\, T_{h-1}(\text{OR-AND})$$

OR-subtree

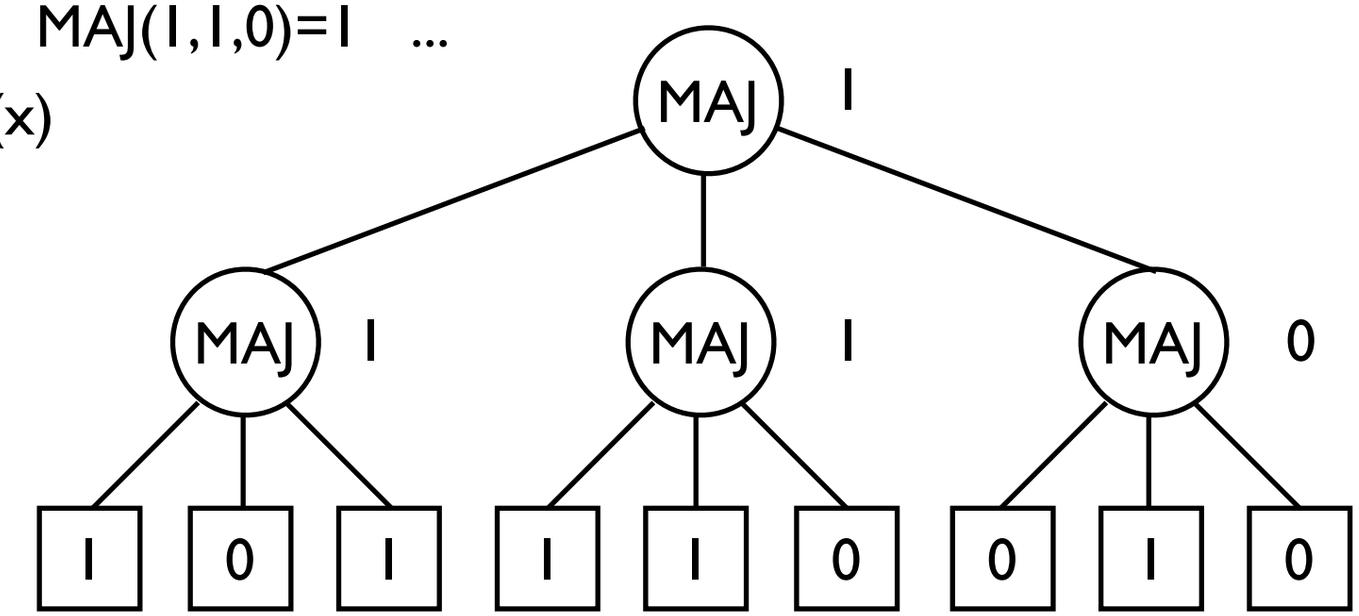
- Similar

Conclusion

- By induction, $T_h(\text{AND-OR}) = T_h(\text{AND-OR}) = 3^{h/2} = n^{0.792}$

Recursive 3-MAJ tree

- Complete ternary tree with height h, and $n=3^h$ leaves
- Leaves are valued by 0/1 and encoded by a n-bit string x in $\{0,1\}^n$
- Nodes: MAJ(0,0,1)=0 MAJ(1,1,0)=1 ...
- Value of the root: MAJ(x)



Results

- Deterministic algorithms must evaluate all n leaves, in the worst case
- There is a MONTE-CARLO algorithm that evaluating 2.64926^h
- MONTE-CARLO algorithms must evaluate 2.5^h leaves, in the worst case

RANDOM-WALK ALGORITHMS

SAT formula

- Boolean variables: $X_1, X_2, ..., X_n$
- k-Clause: disjunction of $\leq k$ variables and their negations

  $X_1 \vee \neg X_5 \vee X_3$ is a 3-clause
- k-SAT Formula: conjunction of k-clauses

  $C_1 \wedge C_2 \wedge ... \wedge C_m$

Problem k-SAT

- Input: k-SAT formula Φ on n variables
- Output:

  a in $\{0,1\}^n$ such that $\Phi(a) = 1$

  REJECT if Φ is not satisfiable: $\Phi(a) = 0$ for all a in $\{0,1\}^n$

Facts

- 2-SAT can be solved in deterministic time O(n+m)
- k-SAT is NP-complete for $k \geq 3$

Algorithm

- Start with any assignment a in $\{0,1\}^n$
- While $\Phi(a) = 0$ do at most $2n^2$ times

  Choose an unsatisfied clause

  Pick at random one of its variables $\rightarrow X_i$

  Flip the i-th bit of a
- OUTPUT a if $\Phi(a) = 1$, and REJECT otherwise

Fact

- Time complexity: $O(n^2m)$
- ONE-SIDED ERROR: Algorithm always rejects unsatisfiable formulae

Theorem

- If Φ is satisfiable,

  then Algorithm outputs some a s.t. $\Phi(a) = 1$ with probability $\geq 1/2$

Example

Formula

- $\Phi = (X_1 \vee X_2) \wedge (\neg X_2 \vee X_3) \wedge (X_1 \vee \neg X_3)$

Evolution

- Start with a = 0 0 0
- $(X_1 \vee X_2)$ is not satisfied

Flip 1st bit: a = 1 0 0 — $\Phi(a)=1$

Flip 2nd bit: a = 0 1 0 — $(\neg X_2 \vee X_3)$ is not satisfied

Flip 2nd bit: a = 0 0 0

Flip 3rd bit: a = 0 1 1 — $(X_1 \vee \neg X_3)$ is not satisfied

Flip 1st bit: a = 1 1 1 — $\Phi(a)=1$

Flip 3rd bit: a = 0 1 0

Special case

- There is a unique s in $\{0,1\}^n$ s.t. $\Phi(s) = 1$
- At iteration i, let $d_i = D_H(a,s)$ be the Hamming distance between a and s

Fact

- If $d_i=0$, then Algorithm stops at step i
- If $d_i=n$, then $d_{i+1}=n-1$
- In general, $d_{i+1} = d_i \pm 1$

a: 0 1 **0** 1 0 **0**
s: 0 1 **1** 1 0 **1**

$D_H(a,s)=2$

Lemma

- $d_{i+1} = d_i - 1$ with probability $\geq 1/2$

Proof

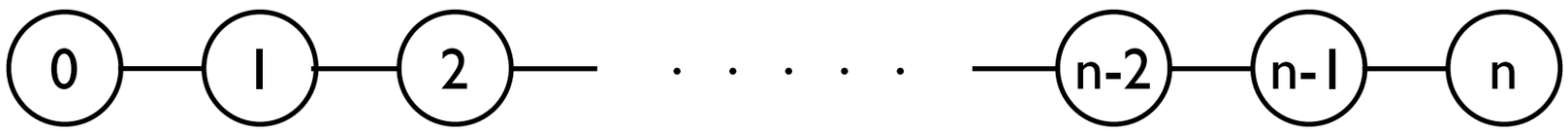
- Assume $C = \neg X_5 \vee X_3$ is the selected unsatisfied clause

  Therefore $a_5=1$ and $a_3=0$

  One of the 2 bits will be selected at random, and flipped
- s satisfies C $\rightarrow$ $s_5=0$ or $s_3=1$. Assume $s_5=0$
- a_5 is flipped to 0 with probability 1/2. If so $d_{i+1} = d_i - 1$

Random evolution of the $D_H(a,s)$

- Worst case: $d_{i+1} = d_i - 1$ with probability 1/2
- Random Walk on the line



Node ↔ $D_H(a,s)$

Transition: Select at random a node connected to current one

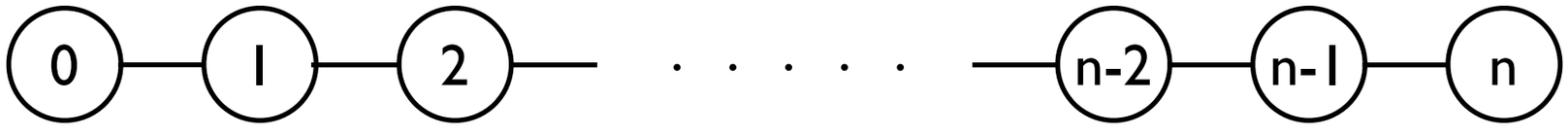
- Hitting time H(j,0): expected # of transitions to reach 0 from j

Theorem

- Consider a random walk on a connected graph with n vertices, m edges.

  Then the worst case hitting time is at most nm

End of proof



- The average number of iterations before reaching s is at most n^2
- By Markov inequality,

  $\Pr(\#\text{ of iterations} \geq 2n^2) \leq 1/(2n^2) \times E(\#\text{ of iterations}) \leq 1/2$

Case of multiple solutions

- $d_i = \text{MIN } D_H(a,s)$, where minimum is over all s in $\{0,1\}^n$ s.t. $\Phi(s) = 1$

Algorithm

- Start with any assignment a in $\{0,1\}^n$
- While $\Phi(a) = 0$ do at most ??? times

  Choose an unsatisfied clause

  Pick at random one of its variables → X_i

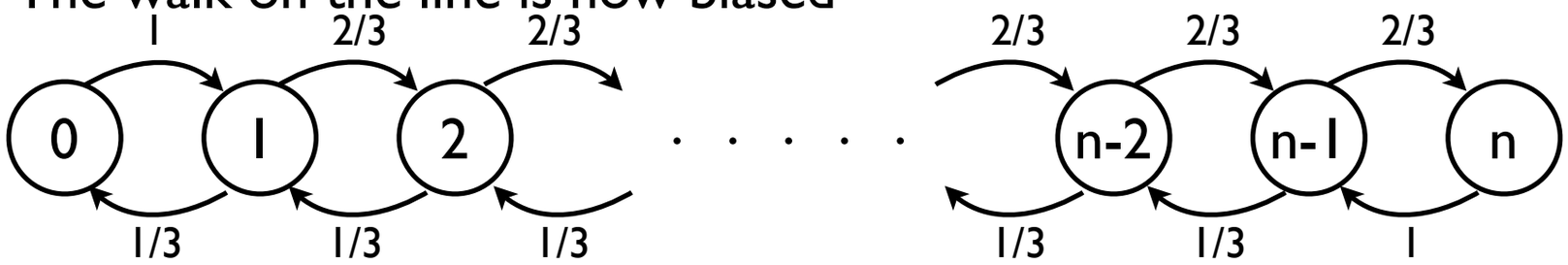
  Flip the i-th bit of a
- OUTPUT a if $\Phi(a) = 1$, and REJECT otherwise

Lemma

- $d_{i+1} = d_i - 1$ with probability at least **1/3**

Main issue

- The walk on the line is now biased



- Worst case hitting time $\approx 2^n$

Closer look

- Pr (Algorithm hits 0 from j in $\leq 3n$ steps) $\gtrsim 2^{-j}$ (up to negligible factors)
- If a is selected at random in $\{0,1\}^n$

  then Pr ($D_H(a,s)=j$) = $2^{-n} \binom{n}{j}$
- In average over the random choice of a in $\{0,1\}^n$

  Pr (Algorithm hits 0 from j in $\leq 3n$ steps)

  $\gtrsim 2^{-n} \sum_j \binom{n}{j} 2^{-j} = 2^{-n} (1+1/2)^n = (3/4)^n$

Algorithm

- Start with a random assignment a in $\{0,1\}^n$
- While $\Phi(a) = 0$ do at most 3n times

  Choose an unsatisfied clause

  Pick at random one of its variables $\rightarrow X_i$

  Flip the i-th bit of a
- OUTPUT a if $\Phi(a) = 1$, and STOP
- RESTART at most $\approx (4/3)^n$
- REJECT

Randomized algorithms

- Schoening algorithm has running time $O(1.334^n)$
- Combination of Schoening algorithm with other algorithms

  Running time $O(1.307^n)$ [Hertli 2011]

Deterministic algorithms

- Schoening algorithm has been derandomized [Moser, Scheder 2011]
- And improved to $O(1.308^n)$ [Makino, Tamaki, Yamamoto 2011]

In practice

- Many heuristics are good

  Conference SAT

Problem

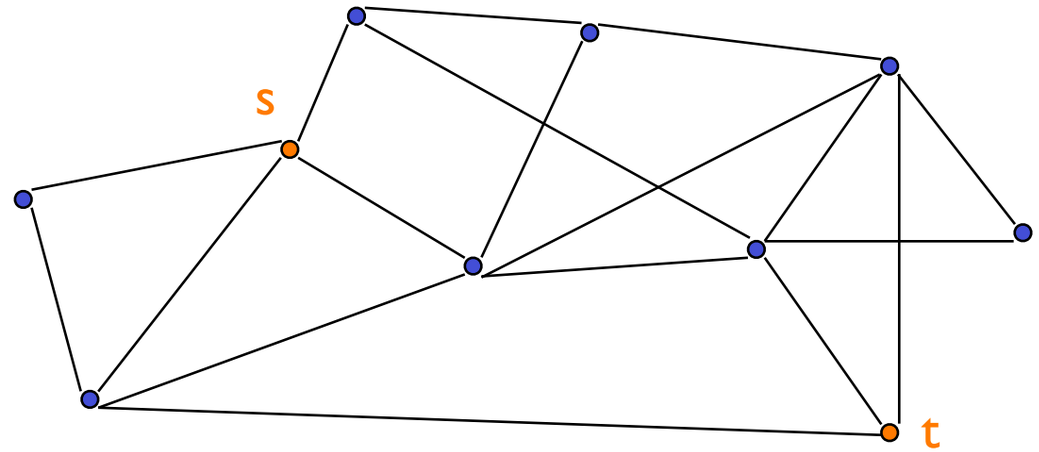
- Input: undirected graph G with n vertices and m edges

  two vertices s (source) and t (target)

- Output:

  ACCEPT if s and t are connected

  REJECT otherwise



Facts

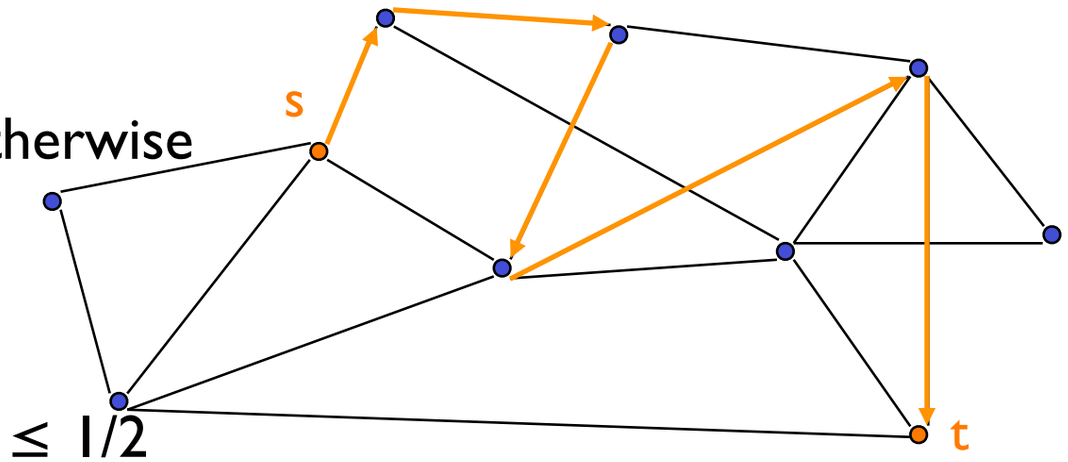
- There is a deterministic algorithm with time complexity O(m)

  But its implementation is rather complex

  And requires space Ω(m), in the worst case

Algorithm

- Set u = s (start from s)
- While u ≠ t do at most 2nm times
  - Select at random a node v connected to u
  - Set u = v (move to u)
- ACCEPT if u = t, and REJECT otherwise



Analysis

- Space O(log n) Time O(nm)
- ONE SIDE ERROR: error probability ≤ 1/2
  (Since hitting time at most nm)

Theorem

- There is a deterministic log-space algorithm for ST-Connectivity

Construction

- Derandomization of the random algorithm [Reingold 2005]
  by transforming the input graph to an expander

APPROXIMATION ALGORITHMS

Problem

- Input: SAT formula Φ on n variables and m clauses
- Output:

  a in $\{0,1\}^n$ that maximizes the number of satisfied clauses in Φ

Facts

- MAX-SAT is NP-hard (MAX-2-SAT also)
- MAX-SAT(Φ) $\leq$ m and MAX-SAT(Φ) = m when is Φ satisfiable

Relaxed problem

- c-approximation of MAX-SAT(Φ) (0<c<1)

  Find a in $\{0,1\}^n$ such that

  c MAX-SAT(Φ) $\leq$ MAX-SAT(Φ,a) ($\leq$ MAX-SAT(Φ))

Algorithm

- Output random a in $\{0,1\}^n$

Theorem

- Algorithm outputs a s.t E_a(MAX-SAT(Φ,a)) $\geq$ (1/2) MAX-SAT(Φ)

Lemma

- $E_a[\ \text{MAX-SAT}(\Phi,a)\ ] \geq m/2$

Proof

- $E_a[\ C(a)\ ] = 1\text{-}2^{-k}$ for every clause C with exactly k variables
- $\text{MAX-SAT}(\Phi,a) = \sum_j C_j(a)$
- Conclude using $k \geq 1$ and linearity of expectation

Remark: large k → better approximation

Derandomization

- Since $E_a[\ \text{MAX-SAT}(\Phi,a)\ ] \geq m/2$

then $\text{MAX-SAT}(\Phi,a) \geq m/2$ for some a

Algorithm

- For i=1 to n

Compute $E[\ \text{MAX-SAT}(\Phi,a)\ ]$ over $a_i,a_{i+1},\ldots,a_n$

Set $a_i=0$ if the expectation is $\geq m/2$, and $a_i=1$ otherwise

- Output a

Integer program

- Boolean variable X_i → integer variable Y_i in {0,1}
- Clause C_j → integer variable Z_j in {0,1}
  and linear inequality L_j
  Ex: $C_j = X_1 \vee \neg X_5 \vee X_3$ → L_j: $Y_1 + (1-Y_5) + Y_3 \geq Z_j$
- MAX-SAT(Φ) → objective function MAX $\sum_j Z_j$

LP-relaxation

- $0 \leq Y_i, Z_j \leq 1$

Fact

- Linear program can be solved in strongly polynomial time [Tardos 1985]
  Simplex method, interior-point method, ellipsoid method...

Rounding

- Let (Y=y*,Z=z*) be the optimal solution of our LP
  How to get assignment a?
  What is the quality of the approximation?

Algorithm

- Solve the previous LP and find the optimal solution (y^*,z^*)
- Set $a_i=1$ with probability y_i^*, (otherwise $a_i=0$)
- Output a

Theorem [Goemans, Williamson 1994]

- Algorithm outputs a s.t $E_a(\text{MAX-SAT}(\Phi,a)) \geq (1-1/e)\ \text{MAX-SAT}(\Phi)$
  ≈ 0.632

Lemma

- If C_j has k variables then $E_a[\ C_j(a)\ ] \geq [1 - (1-1/k)^k]\ z_j^*$

Proof

- Say $C_j = X_1 \vee \neg X_5 \vee X_3$

  then $E_a[\ C_j(a)\ ] = (1-y_1)\ y_5\ (1-y_3)$

  $\geq 1 - [\ (1-y_1 + y_5 + 1-y_3)/3\ ]^3$

  $= 1 - [\ 1 - (y_1 + 1-y_5 + y_3)/3\ ]^3$

  $\geq 1 - (\ 1 - z_j^*/3\ )^3$ from LP-constraint L_j

  $\geq [1 - (\ 1 - 1/3\ )^3]\ z_j^*$ by concavity

For every $t_1,t_2,\ldots,t_k \geq 0$
$(t_1+t_2+\ldots+t_k)/k \geq (t_1 t_2 \ldots t_k)^{1/k}$

Theorem

- Algorithm outputs a s.t $E_a(\text{MAX-SAT}(\Phi,a)) \geq (1\text{-}1/e)\ \text{MAX-SAT}(\Phi)$

Lemma

- If C_j has k variables then $E_a[\ C_j(a)\ ] \geq [1 - (1\text{-}1/k)^k]\ z_j^*$

Proof of theorem

- $(1\text{-}1/k)^k < 1/e$ for all $k \geq 1$
- Lemma → $E_a[\ \text{MAX-SAT}(\Phi,a)\ ] \geq (1\text{-}1/e) \sum_j z_j^*$
- LP-relaxation → $\text{MAX-SAT}(\Phi) \leq \sum_j z_j^*$
- $E_a[\ \text{MAX-SAT}(\Phi,a)\ ] \geq (1\text{-}1/e)\ \text{MAX-SAT}(\Phi)$

Fact: Algorithm can again be derandomized

Observation

- Previous algorithm: large k → better approximation
- This algorithm: small k → better approximation

Combination of the two algorithm

Algorithm

- Select at uniformly at random which algorithm to use

Analysis

- For one clause C_j, we have

$$E_a[\, C(a) \,] \geq (1-2^{-k})/2 + (1 - (1-1/k)^k)\, z_j^*/2$$

$$\geq [\, (1-2^{-k}) + (1 - (1-1/k)^k) \,]\, z_j^*/2$$

- $k=1$: $(1-2^{-1}) + (1 - (1-1/1)^1) = 3/2$
- $k=2$: $(1-2^{-2}) + (1 - (1-1/2)^2) = 3/2$
- $k \geq 3$: $(1-2^{-k}) + (1 - (1-1/k)^k) \geq 1-2^{-3} + 1-1/e = 1.5071\ldots \geq 3/2$
- Conclusion

$$E_a[\, C(a) \,] \geq (3/4)\, z_j^*$$

Theorem

- Combined algorithm satisfies $E_a(\text{MAX-SAT}(\Phi,a)) \geq (3/4)\, \text{MAX-SAT}(\Phi)$

Derandomization

- Run both (derandomized) algorithms and select the best output

Advantages of Randomized Algorithms

- Simplicity
- and/or Efficienty

Disadvantages

- EITHER correctness is not guaranteed (MONTE CARLO)
- OR running time can be very long (LAS VEGAS)
- NEVER both
- Analysis can be complicated

  but not necessarily the algorithm itself!
- Even correct outputs may vary for different random choices

  BELLAGIO algorithm are randomized algorithm

  whose output is almost independent on random choices

  Typical problems:

  Output $(a_1,a_2,...,a_n)$ in S such that $P(a_1,a_2,...,a_n)\neq 0$

  Output a prime in [n,2n]

Conclusion

Scoope

- Number theory (primality testing), Group theory (commutativity testing)
- Algebraic identities (polynomial, associativity testing)
- Data structure (sorting, searching), Statistics (estimation)
- Computational Geometry (?)
- Mathematical programming (linear programming, rounding)
- Graph algorithms
- Counting, Enumerating
- Parallel and distributed computing
- Proofs (interactive, zeroknowledge, PCP)
- Software testing
- **APPROXIMATION ALGORITHM**
- **COMMUNICATION COMPLEXITY**
- **STREAMING ALGORITHM**