ALMA
ALGORITHMS

Fall 2016

Ioannis Milis

e-mail: milis@aueb.gr
phone: 210 8203537
office: Kodrigktonos 12, 3rd floor
(entrance code: 4267)

Content

THEORY AND TECHNIQUES

- Polynomial, pseudo-polynomial and exponential algorithms
- Linear Programming
- Approximation Algorithms
- Randomization

PROBLEMS

- SATisfiability problems
- Graph problems: Cuts, Paths, TSP
- Packing problems: Partition, Subset Sum, Knapsack, Bin Packing
- Covering problems: Vertex and Set covers
- Scheduling problems

Bibliography

- [DPV] S. Dasgupta, C.H. Papadimitriou, U.V. Vazirani: Algorithms, http://beust.com/algorithms.pdf
- [CLRS] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein: Introduction to Algorithms”
- [KT] J. Kleinberg, E.Tardos: Algorithm Design
- [VAZ] V. V. Vazirani: Approximation Algorithms
- [SW] D. Shmoys, D. Williamson: Design of Approximation Algorithms

Polynomial,

Pseudo-Polynomial and

Exponential

algorithms and problems

Problems

EXP(onentiation)

I: positive integers a, n

Q: calculate a^n

FIBONACCI NUMBERS

I: a positive integer n

Q: calculate n-th Fibonacci number F_n

SUBSET SUM

I: a set $S=\{a_1, a_2, ..., a_n\}$ of n positive integers and integer B

Q: is there a subset $A \subseteq S$ s. t. $\sum_{a_i \in A} a_i = B$?

SAT(isfiability)

Instance: a boolean formula φ

Question: Is φ satisfiable ?

(is there a value assignment to its variables making φ TRUE ?
= truth assignment)

Size of Instance and complexity

Consider the description of an instance,
i.e., of all the parameters and constraints

Instance —encoding→ I (encoded instance)

e.g. in decimal / binary / unary

|I| = length of encoded instance / input
= # of digits of the encoded input I

Integer n :	Decimal	Binary	Unary
# bits :	$\lfloor \log_{10} n \rfloor + 1$	$\lfloor \log_2 n \rfloor + 1$	n

Size of Instance and complexity

$|I|$ = length of encoded instance / input

= # of digits of the encoded input I

Polynomial algorithms: $O(poly(|I|)$

N(I) = the largest number in the input

Pseudo-Polynomial algorithms: $O(poly(N(I))$

- $N(I)$ is $O(exp(|I|)$
- They are $O(poly(|I|))$ if we consider I encoded in unary
- ONLY FOR PROBLEMS WITH NUMBERS

Exponential algorithms: $O(exp(|I|)$

Exponentiation

EXP(onentiation)
I: positive integers `a,n`
Q: calculate a^n

```
Algorithm exp1(a,n);
// a, n positive integers
p:= 1;
for i:=1 to n do p:=p*a;
return p;
```

Correctness: obvious

Complexity: $O(n)$

$|I| = \log n \Rightarrow n = 2^{|I|}$, $O(n)$ is $O(2^{|I|})$, $O(\exp(I))$ **NOT POLYNOMIAL !**

$N(I) = n$, $O(n)$ is $O(\text{poly}(N(I)))$ **PSEUDO-POLYNOMIAL !**

Can we do better? Is there a polynomial algorithm for EXP ?

Exponentiation

$$a^n = \begin{cases} \left(a^{\frac{n}{2}}\right)^2 & \text{if } n \text{ is even} \\ a\left(a^{\left\lfloor \frac{n}{2} \right\rfloor}\right)^2 & \text{if } n \text{ is odd} \end{cases} \qquad a^0 = 1$$

```
Algorithm exp2 (a,n);
// a, n positive integers
if n =0 then return 1;
z:= exp2 (a, ⌊n/2⌋ );
if n is even then return z²
             else return az²
```

Correctness: obvious

Complexity: $O(\log n)$, polynomial in $|I|$ (why?)

$T(n)=T(n/2) + O(1)$

Fibonacci numbers

FIBONACCI

I: Recursion $F_0 = 0$; $F_1 = 1$; $F_n = F_{n-1} + F_{n-2}$, $n \geq 2$

Q: Calculate F_n

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …

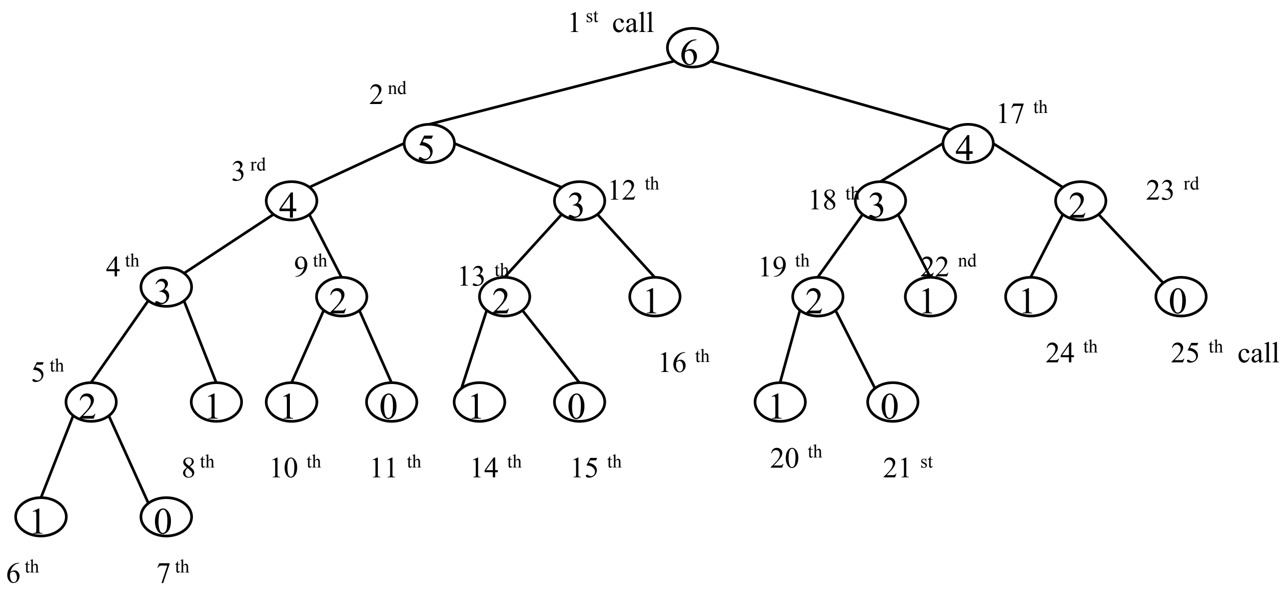
```
Algorithm fib1(n) // Direct implementation of recursion
   {if n<2 then return n
              else return fib1(n-1)+ fib1(n-2) }
```

Complexity of fib1(n): $T(0)=T(1)=1$,

$$T(n)= T(n-1) + T(n-2) + 1 \quad (= 2 F_n - 1)$$

Fibonacci numbers

`fib(1)`: Call structure of recursion for n=6



Very inefficient: T(n) is $\Omega(2^{n/2})$

Full Binary Tree at least to depth n/2 → $2^{n/2}$ nodes

Why? It calulates the same values several times

Fibonacci numbers

```
Algorithm fib2(n) // recall computed values
  {f[0]:=0; f[1]:=1;
   for i:=2 to n do f[i]:= f[i-1] + f[i-2]}
```

Complexity of fib(2): $O(n)$ NOT polynomial in $|I|=\log n$

What about space complexity?

Can we do better?

Is there an O(poly(I)) , that is an $O(\log n)$ algorithm for F_n ?

Fibonacci numbers

`fib3(n):` an O(log n) algorithm for F_n

Claim: It holds that (prove it)

$$F_n = \frac{1}{\sqrt{5}}\phi^n - \frac{1}{\sqrt{5}}\hat{\phi}^n, \text{ where } \phi = \frac{1+\sqrt{5}}{2} \text{ and } \hat{\phi} = \frac{1-\sqrt{5}}{2}$$

Exponentiation

It suffices to compute ϕ^n and $\hat{\phi}^n$

In fact, F_n is the closest integer to $F_n = \varphi^n / \sqrt{5}$, thus: $F_n = \left\lfloor \frac{\phi^n}{\sqrt{5}} + \frac{1}{2} \right\rfloor$ (why?)

Complexity of fib3(n): O(log n) (why?), **but with use of irrational numbers**

Machines use finite arithmetic, irrational numbers causes precision issues

Can we do better?

Is there a O(log n) algorithm for F_n using only integer numbers?

Fibonacci numbers

`fib4(n):` an O(log n) algorithm for F_n using only integer numbers

Claim: It holds that (prove it)

$$\begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \cdot \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

Exponentiation

It suffices to compute $\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n$

Complexity of fib4(n): O(log n) (why?)

Subset Sum

SUBSET SUM

I: a set $S=\{a_1, a_2, ..., a_n\}$ of n positive integers and integer B

Q: is there a subset $A \subseteq S$ such that $\sum_{i \in A} a_i = B$?

BRUTE FORCE

- there are 2^n possible combinations of n items
- Go through all combinations;
  stop in the first one such that $\sum_{i \in A} a_i = B$; otherwise report NO
- Complexity: $O(n2^n)$

Can we do better?

Subset Sum

RECALL COMPUTED VALUES

- Let S_i={a_1, a_2,…,a_i)
- IDEA: Compute the sums of all subsets of S_i

  using the sums of all subsets of S_{i-1}! (exclude sums > B)

- L: a list of integers
- L+x : a new list with all elements of L increased by x

  e.g. L=[1,2,3,5], L+2=[3,4,5,7]

- MERGE (L,L')
  - Returns a sorted list that is the merge of the sorted lists L and L' with no duplicate values
  - Complexity O(|L|+|L'|) (why ?)

Subset Sum

RECALL COMPUTED VALUES

L_i : list of the sums of all subsets of S_i (sums ≤ B)

```
Algorithm SubsetSum (S,B);
L0=[0];
for i=1 to n do
   Li=MERGE(Li-1, Li-1+ai);
   Remove from Li every element >B;
Check if the largest element in L equals B;
```

Example

S={1,4,5}, n=3, B=8

L_0=[0] L_0+a_1 =[1]

L_1=[0,1] L_1+a_2 =[4,5]

L_2=[0,1,4,5] L_2+a_3 =[5,6,9,10]

L_3=[0,1,4,5,6] Answer: NO

Complexity ?

Subset Sum

Complexity: $O(nB)$

At every step, the list we keep has at most B elements

$$|I| = \log a_1 + \log a_2 + \ldots + \log a_n + \log B$$
$$\leq (n+1) \log B = O(n \log B)$$

Hence, $O(nB)$ is $O(\exp(I))$ NOT POLYNOMIAL

But, $N(I) = B$, that is $O(nB)$ is $O(\mathrm{poly}(N(I))$ PSEUDO-POLYNOMIAL

Can we do better ?

Is there an O(poly) algorithm for SUBSET SUM ? **(we believe) NO !**

Boolean Formulas and SAT

Boolean variable x: T(RUE) / F(ALSE) or 1 / 0

Boolean operators: AND ($x \wedge y$), OR ($x \vee y$), NOT ($\neg x$ / $\overline{x}$)

Literal: Boolean variable (x) or its negation ($\neg x$ / $\overline{x}$)

Boolean formula: $\phi(x,y) = (\neg x \vee y) \wedge (x \vee \neg y)$

SAT

Instance: a boolean formula ϕ

Question: Is ϕ *satisfiable* ?

(is there a value assignment to its variables making ϕ TRUE ?
= Truth Assignment- TA)

Example: $\phi(x,y) = (\neg x \vee y) \wedge (x \vee \neg y)$ is satisfiable
by the assignments x=y=T
and x=y=F

CNF- SAT

<u>Clause</u> = A set of OR-ed literals, e.g. $(x \vee \neg y \vee z)$

<u>Conjunctive Normal Form (CNF)</u> of a formula ϕ:
it is the AND of a set of clauses

E.g. $\phi = (w \vee x \vee y \vee z),\ (w \vee \overline{x}),\ (x \vee \overline{y}),\ (y \vee \overline{z}),\ (z \vee \overline{w}),\ (\overline{w} \vee \overline{z})$.

Any formula ϕ can be written in CNF

<u>(CNF) SAT</u>

Instance: a CNF boolean formula ϕ

Question: Is ϕ *satisfiable* ?

SAT

Brute-force approach

- there are 2^n possible assignments for n variables
- Go through all possible assignments;
  stop in the first truth assignment or report NO
- Running time: $O(poly(n)\ 2^n)$

Backtracing:

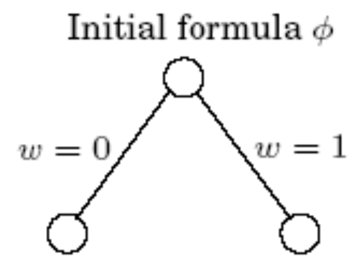
- Intelligent exhaustive search
- Consider partial assignmnets
- Prune the search space
- Example:

$$\phi = (w \vee x \vee y \vee z),\ (w \vee \overline{x}),\ (x \vee \overline{y}),\ (y \vee \overline{z}),\ (z \vee \overline{w}),\ (\overline{w} \vee \overline{z})$$

SAT

Start for the initial formula

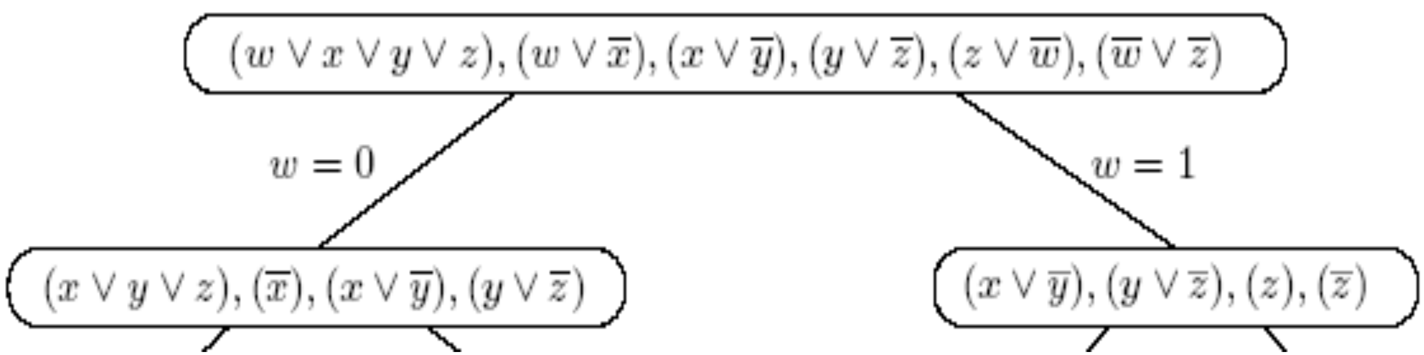
Branch on a variable, e.g. w



Plug into φ the values of w

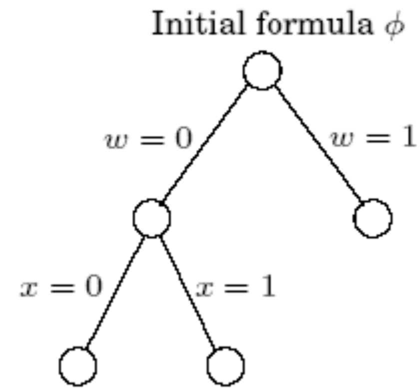
No clause is immediately violated

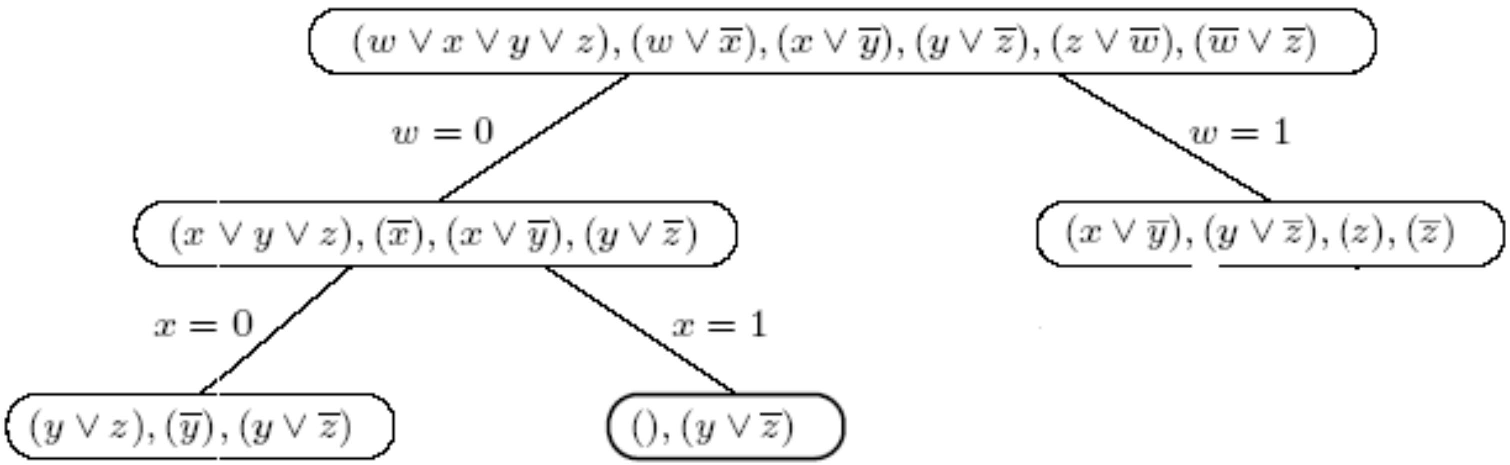
Keep active both branches



SAT

Expand an active node on a new variable, e.g. x

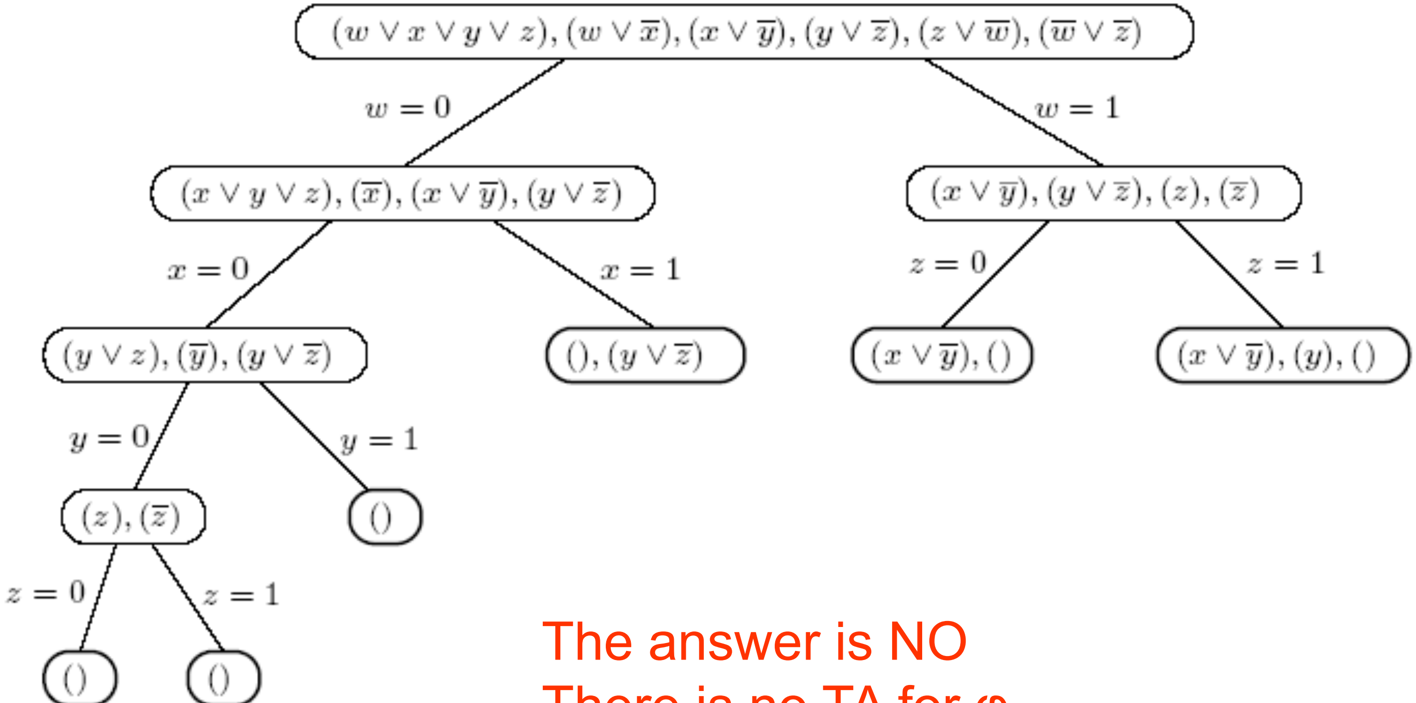




( ): FALSE clause;
Do not expand, this partial assignment can not be expanded to a TA

SAT

Finally:



The answer is NO
There is no TA for φ
Did not have to search all (2^n) assignmnets

SAT

```
Start with some problem P_0
Let S = {P_0}, the set of active subproblems
Repeat while S is nonempty:
  choose a subproblem P ∈ S and remove it from S
  expand it into smaller subproblems P_1, P_2, ..., P_k
  For each P_i:
    If test(P_i) succeeds:  halt and announce this solution
    If test(P_i) fails:  discard P_i
    Otherwise:  add P_i to S
Announce that there is no solution
```

`choose:` the smallest clause, the lowest in the tree,…

`expand:` select a variable to branch on

`Test(Pi):` success: P_i is a TA

failure: P_i does not lead to a TA

uncertainty: P_i is to be expanded

Worst case complexity; explore all $O(2^n)$ possible branchings

$O(poly(n)\ 2^n)$

3-SAT

(CNF) 3-SAT

Instance: a 3-CNF boolean formula ϕ (all φ's clauses have 3 literals)

Question: Is ϕ *satisfiable* ?

e.g. $\phi(x,y,z) = (x \vee \bar{y} \vee \bar{z}) \wedge (z \vee y \vee \bar{x}) \wedge (\bar{z} \vee y \vee x)$

n = # variables of ϕ

m = # clauses of ϕ, $m=O(n^3)$ Why?

Recursion: A 3SAT formula is either nothing

or a clause with three literals $\wedge$ a 3SAT formula

$\Phi = (x \vee y \vee z) \wedge \Phi'$

$\Phi = (x \wedge \Phi') \vee (y \wedge \Phi') \vee (z \wedge \Phi')$

$\Phi|x$: the formula obtained from Φ by assuming x=TRUE

$\Phi = \Phi'|x \vee \Phi'|y \vee \Phi'|z$

3-SAT

A naive recursive algorithm:

$\Phi = \Phi'|x \vee \Phi'|y \vee \Phi'|z$

```
3SAT (Φ)
  if Φ=∅  return TRUE
  (x ∨ y ∨ z) ∧ Φ' =Φ
  if 3SAT(Φ'|x) return TRUE
  if 3SAT(Φ'|y) return TRUE
  return 3SAT(Φ'|z)
```

Complexity: $T(n)=3T(n-1) + poly(n)$, that is $O(3^n\, poly(n))$

worst than $O(2^n\, poly(n))$!

3-SAT

A better recursive algorithm:

$\Phi = \Phi'|x \vee \Phi'|y \vee \Phi'|z$

These three recursive cases are not independent

If $\Phi'|x$ is not satisfiable, then x=FALSE in ANY TA of Φ:

recurse on $\Phi'|\bar{x}y$

If $\Phi'|\bar{x}y$ is not satisfiable, then y=false in ANY TA of Φ:

recurse on $\Phi'|\bar{x}\bar{y}z$

```
3SAT(Φ):
  if Φ = ∅
      return TRUE
  (x ∨ y ∨ z) ∧ Φ′ ← Φ
  if 3SAT(Φ|x)
      return TRUE
  if 3SAT(Φ|x̄y)
      return TRUE
  return 3SAT(Φ|x̄ȳz)
```

Complexity:

$T(n)=T(n-1)+T(n-2)+T(n-3)+poly(n)$

$T(n)=O(\lambda^n\ poly(n))$,

where λ=1,83928675521….

is the largest root of $r^3 - r^2 - r - 1 = 0$,

that is $O(1{,}839286755\mathbf{2}^n)$

3-SAT

An even better recursive algorithm:

Pure literal x: it appears in Φ, but its negation does not

it should be TRUE in ANY TA of Φ

If Φ has no pure literals then

$$\Phi = (x \vee y \vee z) \wedge (\bar{x} \vee u \vee v) \wedge \Phi'$$

and we can eliminate y and z, as well as u and v as before

```
3SAT(Φ):
  if Φ = ∅
      return TRUE
  if Φ has a pure literal x
      return 3SAT(Φ|x)
  (x ∨ y ∨ z) ∧ (x̄ ∨ u ∨ v) ∧ Φ' ← Φ
  if 3SAT(Φ|xu)
      return TRUE
  if 3SAT(Φ|xūv)
      return TRUE
  if 3SAT(Φ|x̄y)
      return TRUE
  return 3SAT(Φ|x̄ȳz)
```

Complexity:

T(n)=2T(n-2)+2T(n-3)+poly(n)

T(n)=O(μ^n poly(n)),

where μ=1,76929235424….

is the largest root of r^3 - 2r -2=0,

that is O(1,76929235425^n)

3-SAT

Can we do better ?

Yes, but still exponentially !

Best deterministic algorithm: $O(1{,}33334^n)$ [2010]

Best randomized algorithm: $O(1{,}32113^n)$ [2010]

Is there a polynomial algorithm for SAT ? **(we believe) NO**

Is there a pseudo-polynomial algorithm for SAT? **NO !**

(SAT is not a problem with numbers)

Review

Problem	Algorithms (complexity)		
	exp(|I|)	**poly(N(I)) (pseudo-poly)**	**poly(|I|)**
Exponentiation		$O(n)$	$O(\log n)$
Fibonacci numbers	$\Omega(2^{n/2})$	$O(n)$	$O(\log n)$
SUBSET SUM	$O(2^n)$	$O(nB)$	NO *
SAT	$O(1{,}33334^n)$		NO *

* Unless P=NP

Randomization for MAX k-SAT

MAX k-SAT- Randomized algorithm [CLRS 35.4]

MAX k-SAT (opt)

I: A k-CNF formula φ

Q: find an assignment satisfying the maximum number of clauses

NP-complete problem (as MAX 2-SAT or k-SAT are NP-complete)

Each clause contains c_i, $1 \le b \le c_i \le k$, distinct literals
(no clause contains a variable and its negation)

Randomized algorithm:

Independently set each variable $= \begin{cases} 1, & \text{with probality } \frac{1}{2} \\ 0, & \text{with probality } \frac{1}{2} \end{cases}$

MAX k-SAT- Randomized algorithm

$X = \#$ of true clauses (random variable)

$$X = X_1 + X_2 + \ldots + X_m = \sum_{i=1}^{m} X_i, \qquad X_i = \begin{cases} 1, & \text{if } C_i = 1 \\ 0, & \text{if } C_i = 0 \end{cases}$$

$$E[X_i] = 1 \cdot \Pr[C_i = 1] + 0 \cdot \Pr[C_i = 0] = \Pr[C_i = 1]$$

$$E[X] = E\left[\sum_{i=1}^{m} X_i\right] = \sum_{i=1}^{m} E[X_i] = \sum_{i=1}^{m} \Pr[C_i = 1]$$

$$E[X] = \sum_{i=1}^{m} \Pr[C_i = 1]$$

MAX k-SAT- Randomized algorithm [CLRS 35.4]

$$\Pr[C_i = 0] = \frac{1}{2} \cdot \frac{1}{2} \cdots \frac{1}{2} = \frac{1}{2^{c_i}}$$

$$\Pr[C_i = 1] = 1 - \frac{1}{2^{c_i}} \geq 1 - \frac{1}{2^b} \geq \frac{1}{2}$$

that is,

$$E[X] = \sum_{i=1}^{m} \Pr[C_i = 1] \geq \sum_{i=1}^{m} (1 - \frac{1}{2^b}) = (1 - \frac{1}{2^b})m \geq \frac{1}{2}m \quad (\text{for } b = 1)$$

At least $(1 - \frac{1}{2^b})$ of all the caluses are satisfied (in expectation)

OPT = maximum # of true clauses, Obviously, $OPT \leq m$

Hence, $E[X] \geq \left(1 - \frac{1}{2^b}\right) OPT \geq \frac{1}{2} OPT \quad (\text{for } b = 1)$,

that is, $E[X] \geq 0{,}5\, OPT$ (randomized) approximate solution

MAX 3-SAT- Randomized algorithm

MAX 3-SAT (opt)

I: A 3-CNF formula φ,

Q: find an assignment satisfying the maximum number of clauses

$$b = c_i = k = 3,\ \forall i:\quad E[X] \geq (1 - \frac{1}{2^3})m = \frac{7}{8}m = 0.875m$$

Fact 1: for every instance of 3-SAT, the expected # of clauses satisfied by a random assignment is at least 7/8 m

MAX 3-SAT- Randomized algorithm [KT 13.4]

Fact 1: for every instance of 3-SAT, the expected # of clauses satisfied by a random assignment is at least 7/8 m

But,
for any random variable, there is some point at which it assumes a value at least as large as its expectation
Thus,

Fact 2: for every instance of 3-SAT, there is an assignment satisfying at least 7/8 of all clauses

Fact 3: every instance of 3-SAT with at most $m \leq 7$ clauses is satisfiable !
Proof:

- By F2 there is an assignment satisfying at least $t=7/8m$ clauses
- For $m<8$ it holds that $7/8m > m-1$
- That is, all the $m<8$ clause are satisfied !

MAX 3-SAT- Randomized algorithm

Fact 2: for every instance of 3-SAT, there is an assignment satisfying at least 7/8 of all clauses

How can we find such an assignment? What is the complexity ?
How long it take until we find one by random trials ?

Waiting for the first success: Z= # of trials until success (random variable)

p= Pr [a random assignment satisfies at least 7/8m clauses]

Pr[Z=j] : probability for success in the j-th trial

Pr[Z=j] = $(1\text{-}p)^{j-1}p$

$$E[Z] = \sum_{j=1}^{\infty} j \Pr[Z = j] = \sum_{j=1}^{\infty} j(1-p)^{j-1} p = \frac{p}{1-p} \sum_{j=1}^{\infty} j(1-p)^{j}$$

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

MAX 3-SAT- Randomized algorithm

Fact 4: for every instance of 3-SAT, an assignment satisfying at least 7/8 of all clauses can be found by 1/p expected random trials.

Can we bound 1/p ?

X=# of satisfied clauses by a random assignment; Recall: **E[X]=7/8m**

p_j = Pr [a random assignment satisfies exactly j clauses]

Recall: p= Pr [a random assignment satisfies at least 7/8m clauses]

$$\sum_{j<7m/8} p_j = 1-p, \quad \sum_{j\geq 7m/8} p_j = p$$

$$E[X] = \sum_{j=1}^{m} jp_j$$

Let m'= the largest integer < 7/8m, m' < m

$$\frac{7}{8}m = E[X] = \sum_{j=1}^{m} jp_j = \sum_{j<7m/8} jp_j + \sum_{j\geq 7m/8} jp_j \leq \sum_{j<7m/8} m' p_j + \sum_{j\geq 7m/8} mp_j$$

$$= m'(1-p) + mp = m' + (m-m')p \leq m' + mp$$

MAX 3-SAT- Randomized algorithm

$$\frac{7}{8}m \leq m' + mp \Rightarrow p \geq \frac{7/8m - m'}{m}$$

$$m' = \text{largest integer} < 7/8m \Rightarrow 7/8m - m' \geq 1/8$$

$$\text{Thus, } p \geq \frac{1}{8m} \Rightarrow \frac{1}{p} \leq 8m$$

Fact 5: there is a randomized algorithm of O(m) expected complexity for finding an assignment satisfying at least 7/8 of all clauses of a 3-SAT instance

2^{nd} Randomized algorithm for MAX k-SAT

Set each variable to be TRUE, independently, with probability $p \geq 1/2$

CASE I: All c_j's with $|c_j| = 1$ consist of a positive literal

CLAIM $\Pr[c_j = 1] \geq \min(p, 1 - p^2)$

Proof:

- if $|c_j| = 1$, then $\Pr[c_j = 1] = p$
- if $|c_j| = 2$, then $\Pr[c_j = 1] = 1 - p^2$, since:

$$c_j = \overline{x_1} \vee \overline{x_2} : \Pr[c = 1] = 1 - p \cdot p = 1 - p^2$$

$$c_j = \overline{x_1} \vee x_2 : \Pr[c = 1] = 1 - p \cdot (1 - p) \geq 1 - p^2$$

$$c_j = x_1 \vee x_2 : \Pr[c = 1] = 1 - (1 - p) \cdot (1 - p) \geq 1 - p^2$$

$$p \geq \frac{1}{2} \geq 1 - p$$

2nd Randomized algorithm for MAX k-SAT

Proof (cntd)

- If $|c_j| = k$, then $\Pr[c_j = 1] = 1 - p_1 p_2 \ldots p_k \geq 1 - p^k \geq 1 - p^2$, where

$$p_i = \begin{cases} p, & \text{for positive literals} \\ 1 - p \leq p, & \text{for negative literals} \end{cases}$$

Taking into account all the case we have $\Pr[c_j = 1] \geq \min(p, 1 - p^2)$

q.e.d.

Setting $p = 1 - p^2 \Rightarrow p = \dfrac{\sqrt{5} - 1}{2} = 0{,}618$

$$\Pr[c_j = 1] \geq 0{,}618$$

$$E[X] = \sum_{i=1}^{m} \Pr[C_i = 1] \geq \boxed{0.618\, m}$$

2^{nd} Randomized algorithm for MAX k-SAT

CASE II: there are c_j's with $|c_j|=1$ consisting of a negative literal

Let $c_j = \{\bar{x}\}$

- if x does not appear in other c_j's with $|c_j| = 1$:

  Swap x and $\bar{x}$ (in all clauses)

- If x appears in other c_j's with $|c_j| = 1$

  neg = # of those c_j's

  Then, at most $m - neg$ can be satisfied

$$E[X] = \sum_{\substack{i=1 \\ C_i \neq \bar{x}}}^{m} \Pr[C_i = 1] \geq 0{,}618(n - neg)$$