The Complexity of Optimization Problems

Summary Lecture 1

- Complexity of algorithms and problems
- Complexity classes: P and NP
- Reducibility
  - Karp reducibility
  - Turing reducibility

Uniform and logarithmic cost

- *Uniform cost*: the overall number of instructions executed by the algorithm before halting

- *Logarithmic cost*:each instruction has a cost depending on the number of bits of the operands
  - E.g. product of two n-bit integer costs $O(n \log n)$

- Same for space measure (but we will talk only of time measure)

Example: x^y

```
begin
        r:=1;
        while y ≠ 0 do
                begin
                        r:=r*x; y:=y-1
                end;
        return r
end
```

- Uniform cost: $2+3y$
- Logarithmic cost: $ay\log y+by^2\log x(\log y+\log\log x)+c$

Worst case analysis

- Instances of the same size may result in different execution costs (e.g. sorting)

- Cost of applying the algorithm on the worst case instance of a given size

  - Gives certainty that the algorithm will perform its task within the established time bound

  - It is easier to determine

Input size

- Size of input: number of bits needed to present the specific input
- Existence of encoding scheme which is used to describe any problem instance
  - For any pair of *natural* encoding schemes and for any instance x, the resulting strings are polynomially related
    - I.e., $|e_i(x)| \leq p_{i,j}(|e_j(x)|)$ and $|e_j(x)| \leq p_{i,j}(|e_i(x)|)$
  - Avoid unary base encoding

Asymptotic Analysis

- Let $t(x)$ be the running time of algorithm A on input x. The *worst case running time* of A is given by

$$t(n)=\max(t(x) \mid x \text{ such that } |x| \leq n)$$

- *Upper bound*: A has complexity $O(f(n))$ if $t(n)$ is $O(f(n))$ (that is, we ignore constants)

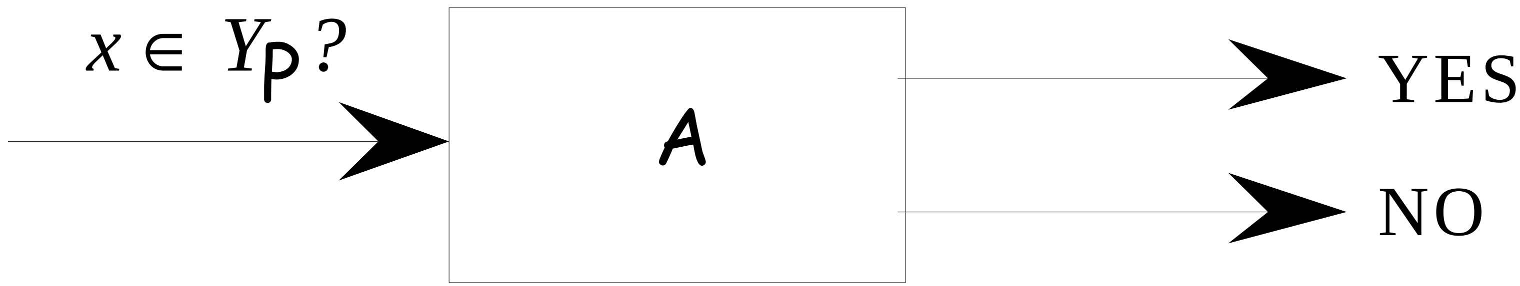
- *Lower bound*: A has complexity $\Omega(f(n))$ if $t(n)$ is $\Omega(f(n))$

Complexity of a problem

- A problem P
  - has a *complexity lower bound* $\Omega(f(n))$ if any algorithm for P has complexity $\Omega(f(n))$
  - has a *complexity upper bound* $O(f(n))$ if an algorithm for P exists with complexity $O(f(n))$

Decision problems

- Set of instances partitioned into a YES-subset and a NO-subset
  - Given an instance *x*, decide which subset *x* belongs to
- A decision problem P is solved by an algorithm A if, for every instance, A halts and returns YES if and only if the instance belongs to the YES-subset



Complexity Classes

- For any function $f(n)$, TIME($f(n)$) is the set of decision problems which can be solved with a time complexity $O(f(n))$
  - P = the union of TIME(n^k) for all k
  - EXPTIME = the union of TIME (2^{n^k}) for all k
    - P is contained in EXPTIME

- It is possible to prove (by diagonalization) that EXPTIME is not contained in P

Examples

- SATISFYING TRUTH ASSIGNMENT: given a CNF formula F and a truth assignment f, does f satisfy F?
  - SATISFYING TRUTH ASSIGNMENT is in P

- SATISFIABILITY (simply, SAT): given a CNF formula F, is F satisfiable?
  - SAT is in EXPTIME.

- Open problem: SAT is in P?

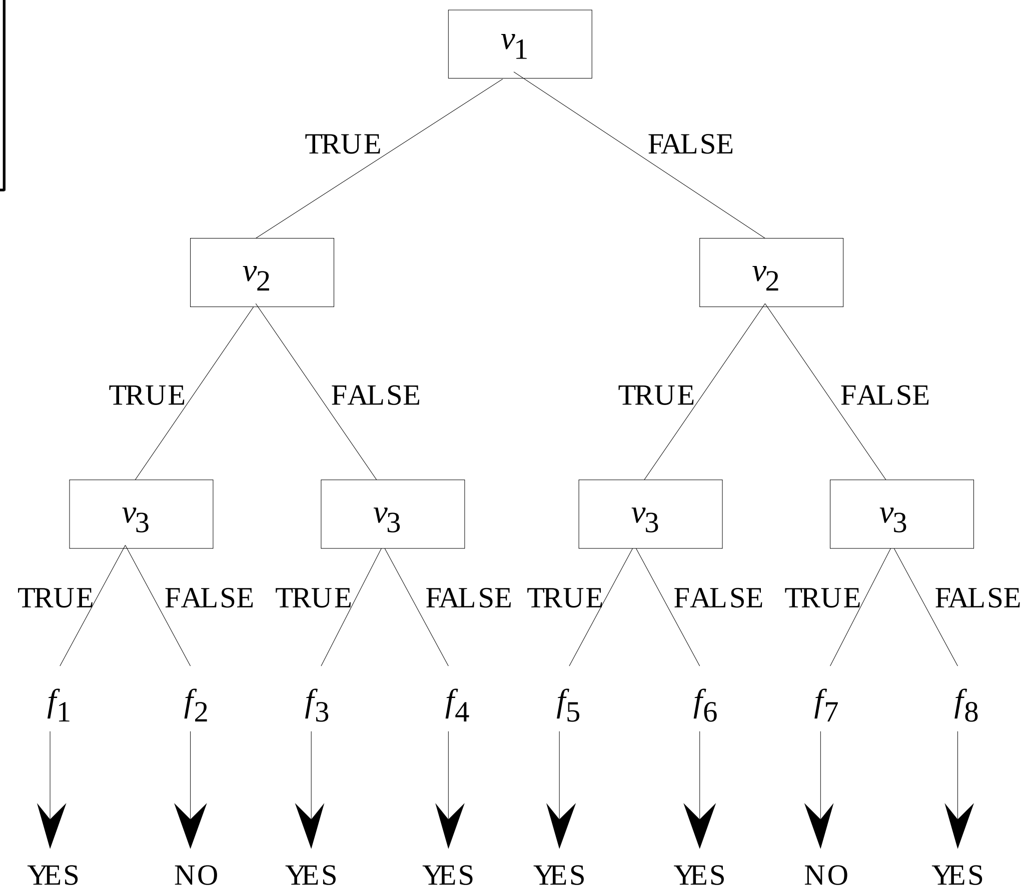
Class NP

- A problem P is in *class NP* if there exist a polynomial p and a polynomial-time algorithm A such that, for any instance x, x is a YES-instance if and only if there exists a string y with $|y| \leq p(|x|)$ such that $\mathsf{A}(x,y)$ returns YES
  - y is said to be a certificate
  - Example: SAT is in NP (the certificate is a truth assignment that satisfies the formula)
- P is contained in NP (the certificate is the computation of the polynomial-time algorithm)

Non-deterministic algorithms: SAT

begin
  for each variable v
    guess Boolean value $f(v)$;
  if f satisfies F **then return** YES
  else return NO
end.

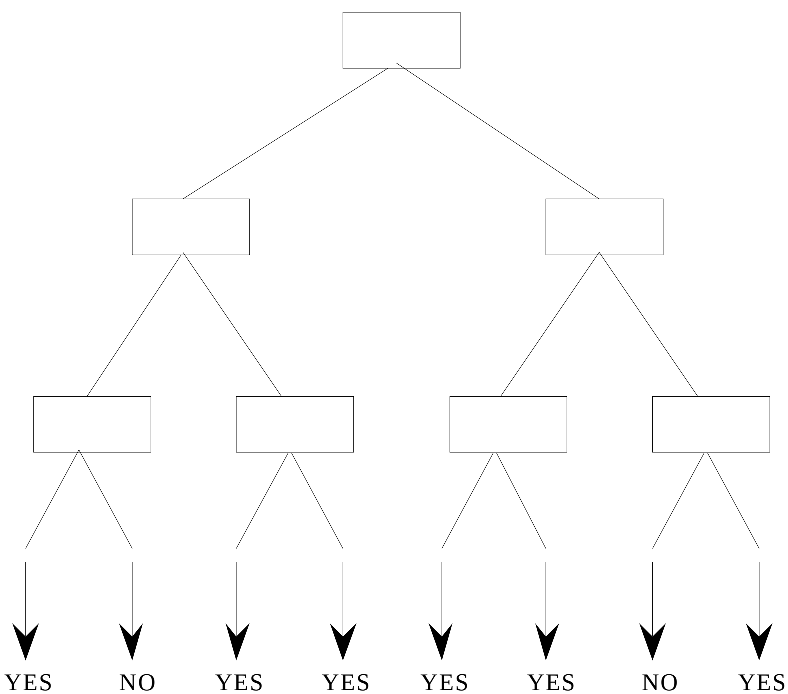
$$(v_1 \textbf{ or } v_2 \textbf{ or } (\textbf{not } v_3)) \\ \textbf{and} \\ ((\textbf{not } v_1) \textbf{ or } (\textbf{not } v_2) \textbf{ or } v_3)$$



Non-deterministic algorithms and NP

begin
    guess string y with $|y| \le p(|x|)$;
    if $A(x,y)$ returns YES **then return** YES
    else return NO
end.

Every problem in NP admits a polynomial-time non-deterministic algorithm

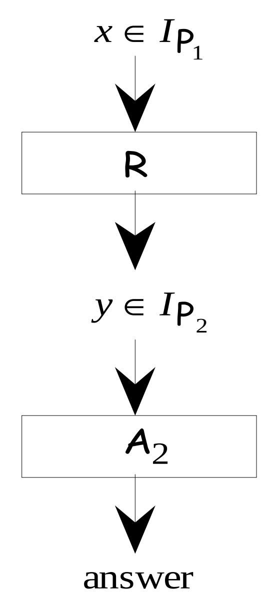


Each computation path, which returns YES, is a certificate of polynomial length that can be checked in polynomial time

Every problem that admits a polynomial-time non-deterministic algorithm is in NP

Karp reducibility

- A decision problem P_1 is *Karp reducible* to a decision problem P_2 (in short, $\mathsf{P}_1 \leq \mathsf{P}_2$) if there exists a polynomial-time computable function R such that, for any x, x is a YES-instance of P_1 if and only if $\mathsf{R}(x)$ is a YES-instance of P_2



- If $\mathsf{P}_1 \leq \mathsf{P}_2$ and P_2 is in P, then P_1 is in P

Example: {0,1}-Linear programming

- SAT $\leq$ {0,1}-LINEAR PROGRAMMING
  - For each Boolean variable v of a CNF Boolean formula F, we introduce a {0,1}-valued variable z
  - For each clause l_1 **or** l_2 **or** … **or** l_k of F, we introduce the inequality $\zeta_1 + \zeta_2 + \ldots + \zeta_k \geq 1$, where

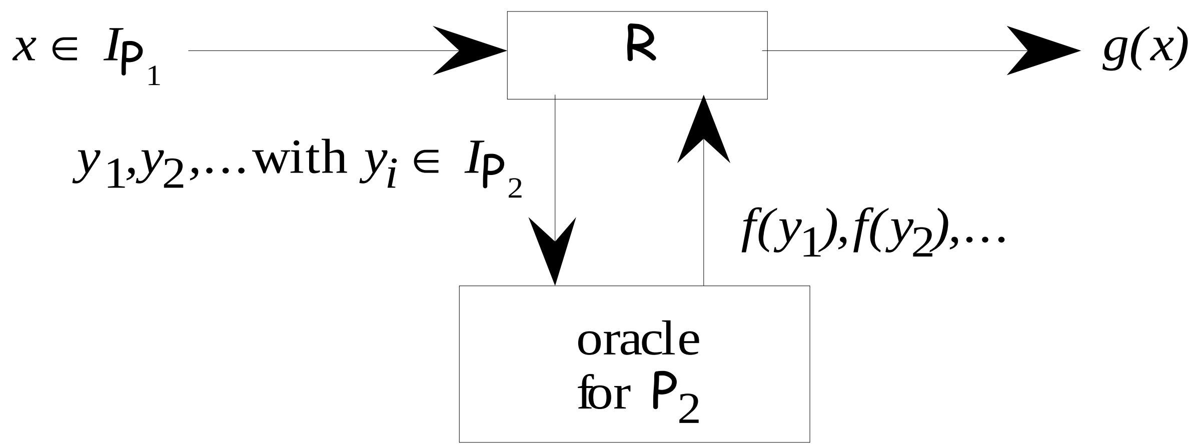
    $\zeta_i = z$ if $l_i = v$ and $\zeta_i = (1-z)$ if $l_i =$ **not** v

    - E.g. (v_1 **or** v_2 **or** (**not** v_3)) **and** ((**not** v_1) **or** (**not** v_2) **or** v_3) is transformed into the following two inequalities:

      $z_1+z_2+(1-z_3) \geq 1$ and $(1-z_1)+(1-z_2)+z_3 \geq 1$
  - If f is a truth assignment, let g be the natural corresponding {0,1}-value assignment (0=FALSE,1=TRUE)
    - f satisfies F if and only g satisfies all inequalities

Turing reducibility

- A decision problem P_1 is *Turing reducible* to a decision problem P_2 if there exists a polynomial-time algorithm R solving P_1 such that R may access to an *oracle* algorithm solving P_2



- If $P_1 \leq P_2$ then P_1 is Turing reducible to P_2

Example: Equivalent formulas

- SAT is Turing reducible to EQUIVALENT FORMULAS
  - Given a CNF Boolean formula F, query the oracle with input F and *x* **and** (**not** *x*)
  - If the oracle answers YES, then F is not satisfiable, otherwise F is satisfiable
  - It is not known whether SAT is Karp reducible to EQUIVALENT FORMULAS