# Web Algorithms

Eng. Fabio Persia, PhD

# Overview

## Approximation Algorithms

- Computational complexity
- Optimization problems
- Approximation
- Algorithmic techniques:
  - greedy
  - local search
  - linear programming
    - rounding
    - primal-dual
  - dynamic programming
- Approximation schemes
- Alternative approaches

## Web Search

- Social networks and bibliometry
- Centrality measures
- Spectral analysis and prestige index
- Link Analysis
- Web structure

## Sponsored search

- Search and advertising
- Matching markets
- Auctions
- VCG mechanism
- GSP mechanism

# References (Approximation)

G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela,
M. Protasi:

*Complexity and Approximation*

Springer 1999, ISBN: 3-540-65431-3

Vijay V. Vazirani:

*Approximation algorithms*

Springer 2001, ISBN: 3-540-65367-8

# References (Web Search)

David Easley and Jon Kleinberg:

*Networks, Crowds, and Markets*

Cambridge University Press, 2010, ISBN: 9780521195331

Jure Leskovec, Anand Rajaraman and Jeff Ullman:

*Mining of Massive Datasets (free version online)*

Stanford University, 2011, ISBN: 9781107015357

Soumen Chakrabarti:

*Mining the Web – Discovering Knowledge from Hypertex Data*

Morgan Kaufmann, 2003, ISBN: 9781558607545

# Computational Complexity

# Problems in Computer Science

A problem $\pi$ is a relation:

$$\pi \subseteq I_\pi \ x \ S_\pi$$

where

$I_\pi$ = set of the input instances of the problem

$S_\pi$ = set of the solutions of the problem

# Problem types

- **decision**:
  - They check if a given property holds for a certain input
  - $S_\pi$=*{true,false}* or simply $S_\pi=\{0,1\}$ and the relation $\pi \subseteq I_\pi$ x $S_\pi$ corresponds to a function $f: I_\pi \rightarrow \{0,1\}$
  - Ex: satisfiablity, test of graph connectivity, etc…

- **search**:
  - Given an instance $x \in I_\pi$ they ask for the determination of a solution $y \in S_\pi$ such that the pair $(x,y) \in \pi$ belongs to the relation defining the problem
  - Ex: satisfiablity, clique and vertex cover, in which we ask in output a satisfying truth assignment, a clique and a vertex cover, respectively, instead of simply yes or not

- **optimization**:
  - Given an instance $x \in I_{\pi}$, they ask for the determination of a solution $y \in S_{\pi}$ optimizing a given measure of cost function;
  - Es: min spanning tree, max SAT, max clique, min vertex cover, min TSP, etc…

# Complexity of algorithms and problems

Expressed as a function of the input size (denoted as $|x|$ $\forall x \in I_\pi$).

Size of instance *x*

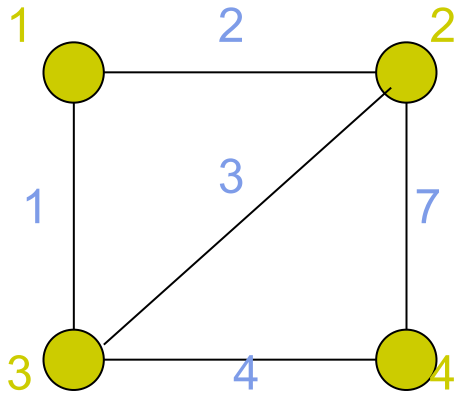
- Amount of memory necessary to store *x* in a computer
- Length $|x|_c$ of the string encoding *x* in a particular natural code $c: I_\pi \rightarrow \Sigma$, where $\Sigma$ is the alphabet of code *c*

Natural code:

- concise: strings encoding instances must not be redundant or unnecessarily lengthened
- numbers expressed in base ≥ 2

# Example:

- Instance: Graph *G*:



- Code for *G*:

$\sum=\{\ \{,\},,,0,1,2,3,4,5,6,7,8,9\}$ (Symbols)

$$c(G)=\{\underbrace{1,2,3,4}_{\text{Nodes}},\underbrace{\{1,2\},\{1,3\},\{2,3\},\{2,4\},\{3,4\}}_{\text{Edges}},\underbrace{2,1,3,7,4}_{\text{Weights}}\}$$

$|G|_c=49$

- Def: Let $t_A(x)$ be the running time of algorithm *A* for input *x*; then the worst case running time of A is :

$$T_A(n)=max\{t_A(x) \mid |x|\leq n\} \ (\forall n>0)$$

- Def: Algorithm *A* has (time) complexity:
  - $O(g(n))$ if $T_A(n)=O(g(n))$ (i.e. $\lim_{n\to\infty}\frac{T_A(n)}{g(n)}\leq c$ for a constant $c$>0)
  - $\Omega(g(n))$ if $T_A(n)=\Omega(g(n))$ (i.e. $\lim_{n\to\infty}\frac{T_A(n)}{g(n)}\geq c$ for a constant $c>0$)
  - $\Theta(g(n))$ if $T_A(n)=\Theta(g(n))$ (i.e $T_A(n)=\Omega(g(n))$ and $T(n)=O(g(n))$)

- Def: A problem has complexity:
  - $O(g(n))$ if **there exists** an algorithm *A* solving it having complexity $O(g(n))$;
  - $\Omega(g(n))$ if **every algorithm** *A* solving it has complexity $\Omega(g(n))$;
  - $\Theta(g(n))$ if it has complexity $O(g(n))$ and $\Omega(g(n))$.

# Decision problems and complexity classes

- Decision problems are usually described by an input instance or simply INPUT and a QUESTION about the input.

- Examples:
  - *Satisfiability:*
    - INPUT: CNF formula defined on a set of variables *V*.
    - QUESTION: ∃ a truth assignment $\tau : V \rightarrow \{0,1\}$ satisfying the formula?

# A possible instance of the SATISFIABILITY problem

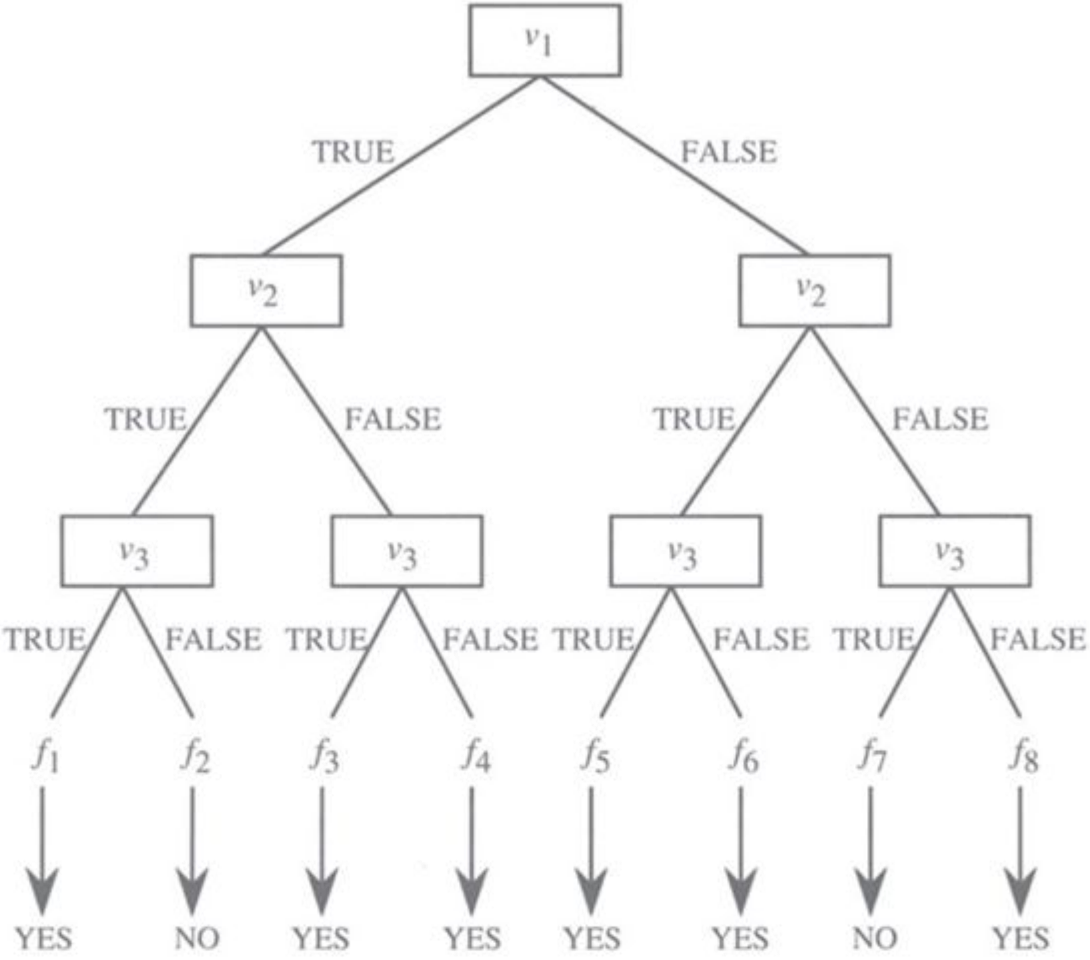
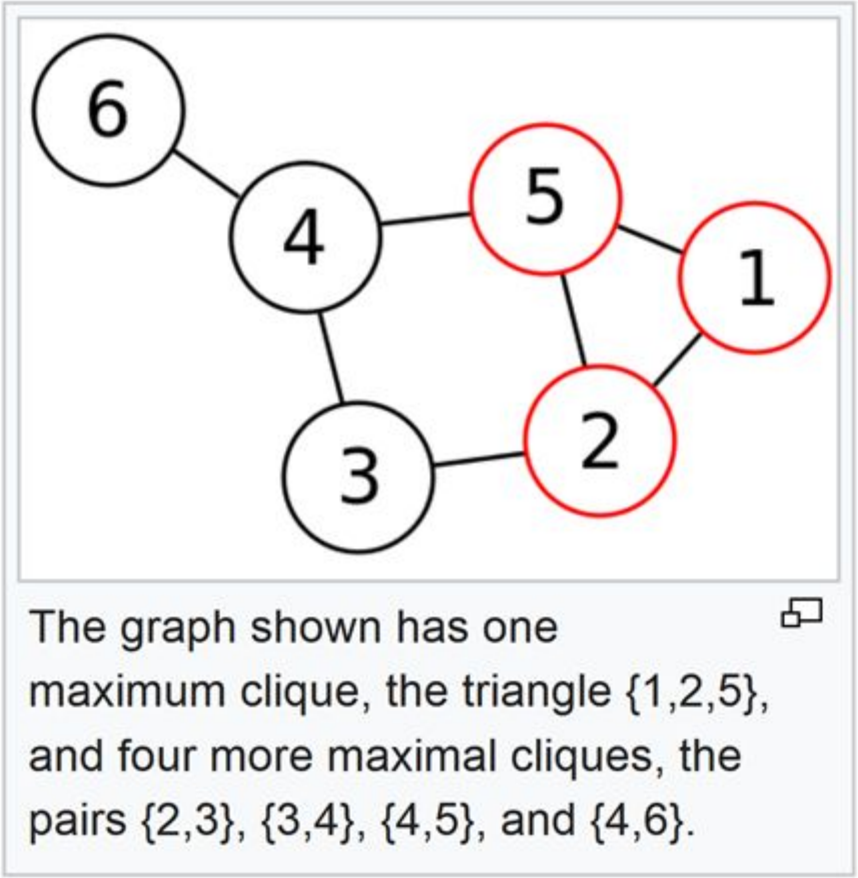


Figure 1.2
A nondeterministic algorithm for SATISFIABILITY with input $v_1 \vee v_2 \vee \bar{v}_3, \bar{v}_1 \vee \bar{v}_2 \vee v_3$

- *Clique:*
  - INPUT: non oriented graph $G=(V,E)$ of $n$ nodes and an integer $k>0$.
  - QUESTION: ∃ in $G$ a clique of **at least** $k$ nodes, that is a subset $U \subseteq V$ s.t. $|U| \geq k$ and $\{u,v\} \in E \ \forall u,v \in U$?

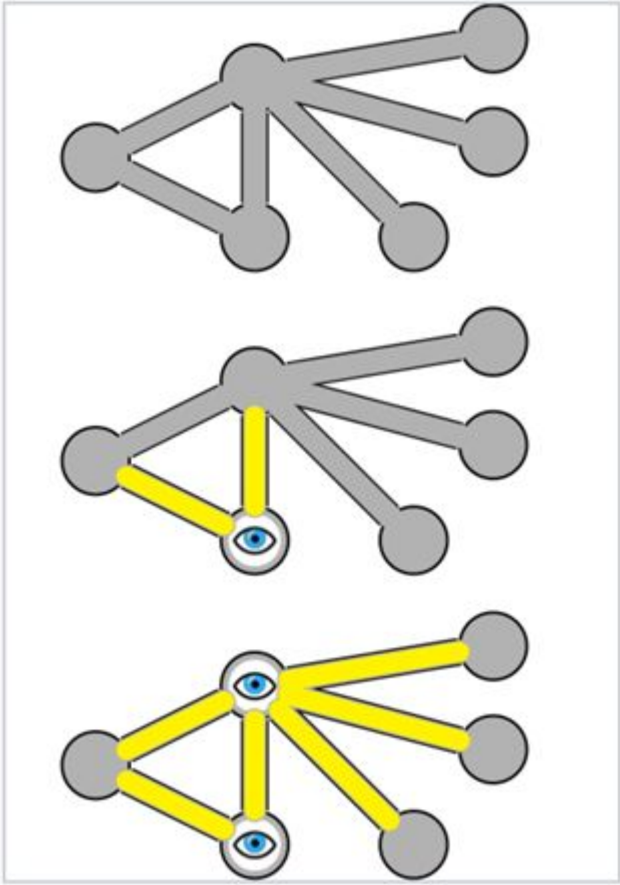
- *Vertex cover:*
  - INPUT: non oriented graph $G=(V,E)$ of $n$ nodes and an integer $k>0$.
  - QUESTION: ∃ in $G$ a vertex cover of **at most** $k$ nodes, that is a subset $U \subseteq V$ s.t. $|U| \leq k$ and $u \in U$ or $v \in U \ \ \forall \{u,v\} \in E$?

# A possible instance of the CLIQUE problem



The graph shown has one maximum clique, the triangle {1,2,5}, and four more maximal cliques, the pairs {2,3}, {3,4}, {4,5}, and {4,6}.

# A possible instance of the VERTEX COVER



Example graph that has a vertex cover comprising 2 vertices (bottom), but none with fewer.

In decision problems $I_\pi = Y_\pi \cup N_\pi$

where

$Y_\pi$= Set of positive instances, that is with solution 1,
$N_\pi$= Set of negative instances, that is with solution *0*.

- Def: An algorithm *A solves π* if and only if ∀ input $x \in I_\pi$, *A* answers *1* if and only if $x \in Y_\pi$.

- Def: *TIME(g(n))* = class of decision problems with complexity *O(g(n))*.

# Non-deterministic algorithms for decision problems

They consist of 2 phases:

- Phase 1: non-deterministically generate a "*certificate*" $y$;
- Phase 2: starting from the input $x$ and the certificate $y$, check if $x$ is a positive instance.

**Def:** A non-deterministic algorithm $A$ solves $\pi$ if it stops for every possible certificate $y$ and there exists a certificate $y$ for which $A$ answers $1$ (TRUE) if and only if $x \in Y_{\pi}$.

Complexity:

- cost phase 2;
- expressed as a function of $|x|$.

*NTIME(g(n))* = class of decision problems with non-deterministic complexity $O(g(n))$

# Example: non-deterministic algorithm for Clique

- Phase 1: Given the input graph $G=(V,E)$, non-deterministically generate a subset $U \subseteq V$ of $k$ nodes.

- Phase 2: Check if $U$ is a clique, that is if $\{u,v\} \in E\ \forall u,v \in U$, and in such a case answer *1*, otherwise answer *0*.

- Clearly the algorithm solves Clique, as it stops for any possible subset $U$ and there exists a subset $U$ for which it answers *1* if and only if there exists a clique of $k$ nodes in $G$, that is if and only if $(G,k) \in Y_{Clique}$.

- Complexity: $O(n^2)$, since $|U| <= |V| = n$.

# Remarks

1. A deterministic algorithm is less powerful than a non-deterministic one as it cannot execute Phase 1.

1. If there is a deterministic algorithm A solving *π*, then there exists also a non-deterministic algorithm *A'* solving *π* with the same complexity as follows:
   - it executes Phase 1 and coincides with *A* in Phase 2, ignoring certificate *y*.

Corollary remark 2:

$$TIME(g(n)) \subseteq NTIME(g(n))$$

Class of problems deterministically solvable in time *O(g(n))*

Class of problems non-deterministically solvable in time *O(g(n))*

# Efficiency and tractability

- A problem is tractable if it can be solved efficiently (deterministically).

- Are considered tractable or efficiently solvable all the problems having complexity bounded by a polynomial of the input size.

TRACTABILITY≡ EFFICIENCY ≡POLINOMIALITY

# Reason 1:

1. Growth of polynomial functions with respect to exponential ones.

Example: if $\pi_1$ has complexity $n$ and $\pi_2$ has complexity $2^n$

| Running time of a function of the input size | | |
|---|---|---|
| Size n | Time $\pi_1$ | Time $\pi_2$ |
| 1 | 1 | 2 |
| 2 | 2 | 4 |
| 3 | 3 | 8 |
| 4 | 4 | 16 |
| 5 | 5 | 32 |
| … | … | … |
| 10 | 10 | 1024 |
| … | … | … |
| 100 | 100 | $2^{100}$ |

| Size of instances solvable within a certain time as a function of the computer performance | | |
|---|---|---|
| Speed | Size $\pi_1$ | Size $\pi_2$ |
| 2 | 2 | 1 |
| 4 | 4 | 2 |
| 8 | 8 | 3 |
| … | … | … |
| 256 | 256 | 8 |
| … | … | … |
| 1024 | 1024 | 10 |
| 2048 | 2048 | 11 |
| 4096 | 4096 | 12 |

# Reason 2:

2. Robustness of the concept of polynomial time solvability

The composition of polynomials is a polynomial and thus the polynomial time solvability of a problem is independent of:

- the used natural code, as all the natural codes are polynomially related
- the adopted computational model if reasonable (that is constructible in practice or in better words able to perform a constant bounded work per step), as such models are polynomially related, that is they can simulate each other in polynomial time

*Remark:* the non-deterministic Turing machine is not a reasonable computational model, as the amount of work done at each step (each level of the tree of the computations) grows exponentially

# Polynomially related codes

- Def: Two codes $c_1$ and $c_2$ for a problem $\pi$ are polynomially related if there exist two polynomials $p_1$ and $p_2$ s.t. $\forall x \in I_\pi$ :
  1. $|x|_{c1} \leq p_1(|x|_{c2})$
  2. $|x|_{c2} \leq p_2(|x|_{c1})$

- If the complexity with respect to $c_1$ is $O(q_1(|x|_{c1}))$ for a given polynomial $q_1$, then with respect to $c_2$ it is $O(q_1(p_1(|x|_{c2})))=O(q_2(|x|_{c2}))$, where $q_2$ is the polynomial such that $\forall\lambda\ q_2(\lambda) = q_1(p_1(\lambda))$.

- All the natural codes are polynomially related, that is polynomial solvability does not depend on the particular used code.

- Input size: any quantity polynomially related to a natural code (and thus to any possible natural code, given that all natural codes are polynomially related and the composition of polynomials is a polynomial).

# Example

- Assume that for any graph $G$ of $n$ nodes
  - $|G|_{c1}=10n^2$
  - $|G|_{c2}=n^3$

If $p_1(\lambda)=10\lambda$ and $p_2(\lambda)=\lambda^2$ we have that

- $|G|_{c1}= \quad 10n^2 \quad \leq \quad 10n^3 \quad = p_1(|G|_{c2})$
- $|G|_{c2}= \quad n^3 \quad \leq \quad 100n^4 \quad = p_2(|G|_{c1})$

Thus the two codes are polynomially related.

Rule of thumb: two quantities are polynomially related if they are polynomials on the same variables

# Example: non natural encoding

*Primality*

- INPUT: integer $n>0$.
- QUESTION: is $n$ a prime number?

ALGORITHM (trivial): scan all the numbers from $2$ to $n-1$ and answer $1$ if none of them divides $n$.

Complexity $O(n)$: polynomial?

- Code $c_1$ (natural): $n$ expressed in base $2$, i.e. $|n|_{c1}=log_2n$
- Code $c_2$ (non natural): $n$ expressed in base $1$, i.e. $|n|_{c2}=n$

Thus the complexity of the algorithm is:

- $O(2^{|n|_{c1}})$ with respect to $c_1$, that is exponential
- $O(|n|_{c2})$ with respect to $c_2$, that is polynomial!!!

Input size (polynomially related to natural codes): $|n|_{c1}=log_2n$

# Polynomially simulable computational models

- Def: Two computational models $M_1$ and $M_2$ are mutually polynomially simulable if there exist two polynomials $p_1$ and $p_2$ such that

  1. Every algorithm $A$ for $M_1$ with complexity $T_A(n)$ can be simulated on $M_2$ in time $p_1(T_A(n))$
  2. Every algorithm $A$ for $M_2$ with complexity $T_A(n)$ can be simulated on $M_1$ in time $p_2(T_A(n))$

  Thus if $A$ is polynomial in $M_1$ then it is polynomial also in $M_2$ and vice versa.

All the reasonable computational models are mutually polynomially simulable, that is polynomial solvability does not depend on the particular used model.

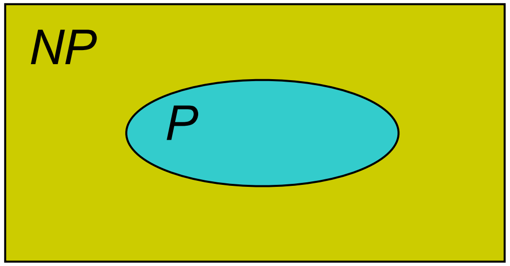
# Classes P and NP

- ***P*** = class of all problems solvable deterministically in polynomial time, that is:

$$P = \bigcup_{k=0}^{\infty} DTIME(n^k)$$

- ***NP*** = class of all problems solvable non-deterministically in polynomial time, that is

$$NP = \bigcup_{k=0}^{\infty} NTIME(n^k)$$

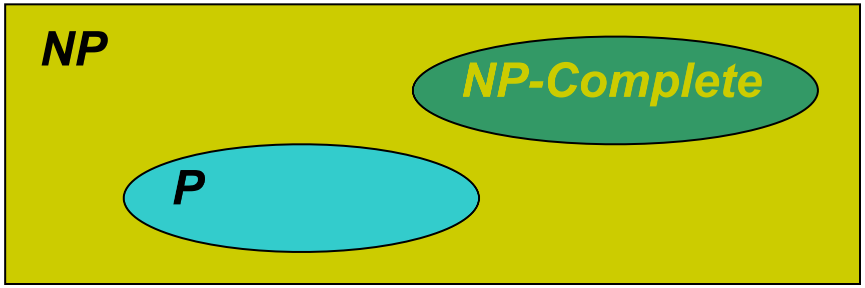


***P=NP***?

No one could prove it!

# NP-Complete Problems

- *NP-Complete* problems: the most difficult problems of *NP* and such that if *P≠NP* they do not belong to *P*; vice versa, if one of them belongs to *P* then *P=NP*.



- So far no one succeeded to find a (deterministic) polynomial time algorithm for any *NP-Complete* problem

↓

Conjecture: *P≠NP*