Complexity Theory

Massimo Paolucci

massimo.paolucci@unige.it

DIBRIS – University of Genova

Problems and their difficulty

- The subject of the complexity theory is to establish if a problem is «easy» or «hard»
- The difficulty of a problem is a general feature that is not associated with a specific instances of the problem

Definition

A <u>problem</u> is the demand to answer a question about the properties of a mathematical structure as a function of the values of constant parameters and the undetermined values of variables

Example

Problem: **HamiltonianCircuit(G(V,E), K)**

Given a graph $G(V,E)$, is there an Hamiltonian circuit of length not greater than K?

Problem structure $\Rightarrow$ graph $G(V,E)$

Solution structure $\Rightarrow$ hamiltonian circuit

Parameters $\Rightarrow$ the specific nodes, edges and weights of G and value K

Definitions

Problem instance

A specific problem case for which all the values for the parameters are fixed

A problem is the set of its infinite instances

Decision problem

A problem whose answer is Yes or No

Example: is there an Hamiltonian circuit of length <K?

Search problem

The problem of finding a solution proving that the answer to the corresponding decision problem is Yes

Example: find an Hamiltonian circuit of length <K

Enumeration problem

Find all the possible solutions to a problem

Example: find all the possible Hamiltonian circuit of length <K

Definitions

Algorithm

A methods for solving a problem composed by a succession of steps

Difficulty of problems

- It does not refer to the possibility of defining an algorithm able to solve a problem (we only consider computable problems)
- We know that all combinatorial problems can be solved by brute force algorithm, but we also know that this is not a «good» algorithm
- The difficulty of a problem refers to the characteristics of the best algorithm available for solving it
- A problem is easy if there exists an efficient algorithm to solve it
- A problem is hard (difficult) if there exist no efficient algorithm to solve it
- To establish the difficulty of a problem we need to measure the efficiency of the (best) algorithm available to solve it

Algorithm efficiency – coding instances

- Measuring the efficiency of an algorithm means to determine its computational complexity
- The efficiency of an algorithm is measured according to two dimensions:
  - The amount of time needed to find the solution (*Time*)
  - The amount of memory needed to store the problem data (*Space*)
- Such two quantities are function of the dimension of the instances of the problems: the number of variables, constraints and the magnitude of the constant data
- The dimension of an instance then is given by the quantity of pieces of information needed to encode the instance

Example of encoding an instance

A LP problem: data to encode are $n+m\times n+m$ coefficients

$c_i,\ i=1,...n;\ a_{ij},\ i=1,...,n,\ j=1,...,m;\ b_j,\ j=1,...m$

Algorithm efficiency – Encoding instances

- Using the binary encoding, for a number x we need k+1 bits, $k \leq \log_2 x \leq k+1$, plus one bit for the sign
- All the numbers to encode have a finite precision: integer or rational numbers

Reasonable encoding

An encoding using at least 2 symbols and that it neither introduces irrelevant data nor it requires an exponential generation of data

Example: Decimal encoding (reasonable)

1 digit (0-9) encodes 10 symbols

10 (2 digits) encodes $100=10^2$ symbols

10x10=100 (3 digits) encodes $1000=10^3$ symbols

100x10=1000 (4 digits) encodes $10000=10^4$ symbols

The encoded symbols grow by factor 10, the number of digits grows as $\log_{10}$

Algorithm efficiency – Encoding instances

Example: Unary encoding (not reasonable)

1 digit (1) encodes 1 symbol

10 decimal (10 digits) encodes 10 symbols

100 decimal (100 digits) encodes 100 symbols

1000 decimal (1000 digits) encodes 1000 symbols

The encoded symbols grow by factor 10, the number of digits grows linearly

- Comparing the two types of encoding for representing a number $X=a^k$
  - Reasonable: $X=a^k$ needs $\log_a X=k$ digits
  - Not reasonable : $X=a^k$ needs $X=a^k$ digits

Algorithm efficiency – Encoding instances

Example of exponential generation of data

A difficult decision problem: $\mathrm{HamiltonianCircuit}(G(V,E), K)$

An easy decision problem: $\mathrm{Sort}(n,K)$

Given n numbers, find if there is a number <K

With a non reasonable encoding (encode the graph and all its Hamiltonian cycles) the difficult HamiltonianCircuit can be transformed into the easy Sort problem

Such encoding is not reasonable since it contains the generation of all the solutions of the problem

Algorithm efficiency – definition

- We focus on problems whose required amount of memory (*Space* dimension) does not grow exponentially
- We consider the *Time* dimension only

Definition

The efficiency of an algorithm is measured as the time needed to solve an instance of the problem that has been reasonably encoded

- The computation time considered is independent from the characteristics of the particular computer used

Solution time

the number of elementary operations needed by the algorithm to solve an instance of a given dimension

- Elementary operations: sums, multiplications, comparisons
- Such measure is given by a *time complexity function*

Algorithm efficiency – definition

Definition

the **time complexity function (tcf)**, $f(k)$, where k is the instance dimension, is a function that gives the maximum number of elementary operations needed by an algorithm to solve the instances of dimension k of a problem

- Tcf is determined by the *worst case analysis*: it evaluates the algorithm behaviour assuming that the algorithm works in the worst situation
- Alternatively, complexity can be evaluated by a statistical analysis (much more complicated as it needs to know how the problem instances are distributed)
- Only the asymptotic behaviour ($k \to \infty$) of tcf is needed, i.e., the order of $f(k)$

Definition

A function $f(k)$ is of order $g(k)$, denoted as $O(g(k))$, if there exists k' and a constant c such that $f(k) \leq c \cdot g(k)$ per $k \geq k'$

Example: a polynomial $p(k) = \sum_{i=0}^{n} c_i k^i$ is of order $O(k^n)$

Algorithm efficiency – definition

Definition

An algorithm is of polynomial type if it has a tcf that is $O(k^p)$ where k is the instance dimension and p is a fixed constant

Definition

An algorithm is of exponential type if it has not a tcf of polynomial order

- An algorithm is efficient if it is of polynomial type
- The increase of the computational time for polynomial type algorithms with the dimension of instances may be acceptable
- Practically this is true if the exponent p is small (good algorithms should have $p < 4$)

Complexity classes

Class P

Class P includes all the decision problems that can be solved by polynomial type algorithms

- <u>Complexity classes are defined for decision problems</u>
- Problems in P as said well-solved since an efficient algorithm is available for solving them
- Unfortunately most of the real problems do not belong to class P

Example: 3 different algorithms running on a computer that executes an operation in 10^{-6} sec

			k		
		10	30	60	
$O(f(k))$	k^2	$10^{-4}s$	$9 \cdot 10^{-4}s$	$3.6 \cdot 10^{-3}s$	Polynomial complexity
	k^5	$0.1s$	$24.3s$	$13m$	
	2^k	$10^{-3}s$	$17.9m$	$366c$	Exponential complexity

Legend: s=seconds, m=minutes, c=centuries (!)

Complexity classes

Example of problem in P:

- The decision problem associated with the Minimum Spanning Tree (MST)

An optimization problem seeks for a solution that minimizes (maximizes) the objective function

The decision problem for MST:

- – Given a graph determine if there exists a spanning tree of length <K

Such problem is in P since there are polynomial algorithms for solving it :

- – Kruskal is $O(|E|^2)$
- – Prim is $O(|E||V|)$

The NP class of problems

Definition

The NP class of problems includes all the decision problems for which, given a solution associated with the answer Yes, it is possible to verify its correctness in a polynomial time (*Non deterministic Polynomial time algorithm*)

- The solution to be verifies must be encoded
- Solution encode = feasibility certificate

Example

Problem: HamiltonianCircuit($G(V,E)$, K)

Given a graph G is there an Hamiltonian circuit of length < K?

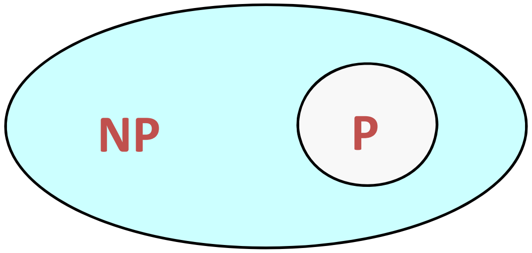
Answer: Yes

Feasibility certificate: a sequence of node of G

Proof: follow the sequence of nodes verifying that it corresponds to an Hamiltonian circuit of length <K (polynomial procedure)

The NP class of problems

- NP problems can be solved by a **non deterministic polynomial time algorithm**
- An NP algorithm is composed by two phases:
  - Guessing stage: An oracole “guesses” the solution (non deterministic phase)
  - Checking stage: The solution is verified in polynomial time
- NP algorithms are only an abstraction and apparently they do not really exist
- A representation of the classes P and NP as sets



- Problems in P can clearly be verified in polynomial time then P$\subseteq$NP
- The conjecture P$\neq$NP has never been demostrated so far

The NP class of problems

- The conjecture P≠NP is one of the most important open problems in mathematics (if anyone was able to solve it she/he would win a 1.000.000$ award from Clay Mathematics Institute - www.claymath.org/millennium/)

  Example: decision problem for MST, Hamiltonian circuit and LP (as a decision problem) are all NP problems, but
  - MST and LP are in P (for LP the first published polynomial algorithm was the ellipsoid algorithm in 1979)
  - Hamiltonian circuit in not in P

Decision and optimization problems

Definitions of Complexity Theory are given for decision problems but they can be extended to optimization problems

Example:

- Decision problem ($HamiltonianCircuit(G(V,E), K)$ = $HC(G,K)$)

  Given a graph G is there an Hamiltonian circuit of length < K?

- The correspondent optimization problem: Traveling Salesman Problem (TSP):

  Find the Hamiltonian circuit in G of minimal length

- Having an algorithm S for solving $HC(G,K)$ it can be used to define an algorithm that solves TSP

Decision and optimization problems

Example (cont.):

The algorithm for TSP is based on a **dichotomic procedure**:

1. Let A and B respectively the minimum and maximum length of the edges in G, m the number of nodes of G and L the length of the Hamiltonian circuit

   Then $mA \leq L \leq mB$

   Set $k = m(B - A)/2$
2. Use algorithm S to solve $HC(G,mA+k)$
3. If the answer is Yes, update $k=k/2$ otherwise $k=k(3/2)$ and go to 2

Since the coefficients are integer at most $(log_2(m(B\text{-}A))+1$ iterations are needed

Complement problems

- Complement problems (co-problems) are decision problems for which the answer is reversed from Yes to No

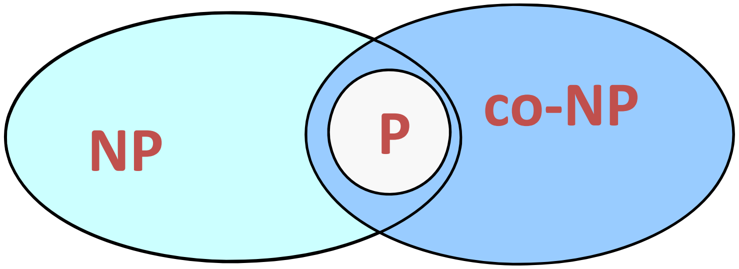
  Example:

  Co-problem of $HC(G,K)$:

  Given graph G, is it true that there is no Hamiltonian circuit of length $<K$?

- Complement problems of problems in P are still in P

  Set representation



  P=co-P

  Conjecture NP≠co-NP ⇒ P⊆NP∩co-NP

- Some complement problems are not in NP

  Example: the feasibility certificate for co-HC(G,K) is the list of all possible Hamiltonian circuits (for a complete graph is |E|! ⇒ not verifiable in polynomial time)

Transformations and NP-complete problems

Definition

Complete problems are defined as the most difficult problems among the ones belonging to a given complexity class, i.e., such problems are at least as difficult as any other problem in the class

Polynomial transformation

An algorithm that, given an instance I of a problem π, generates in polynomial time an instance I' of a problem π' such that, if the answer is Yes for I then the answer is Yes also for I'

- Polynomial transformation of π to π' is denoted as

  $\pi \propto \pi'$ (π is reduced to π')

- Polynomial transformation are transitive
- With such transformations an algorithm for π' can be used as subroutine in an algorithm for π without changing the computational complexity (if π' is polynomial the resulting algorithm is polynomial too)

Transformations and NP-complete problems

The NP-Complete (NP-C) class

NP-C class includes all the decision problems (said NP-Complete) in NP that are at least difficult as any other problem in NP

Definition

A problem $\pi \in$ NP is $\pi \in$ NP-C if $\forall\ \pi' \in$ NP it is possible $\pi' \propto \pi$

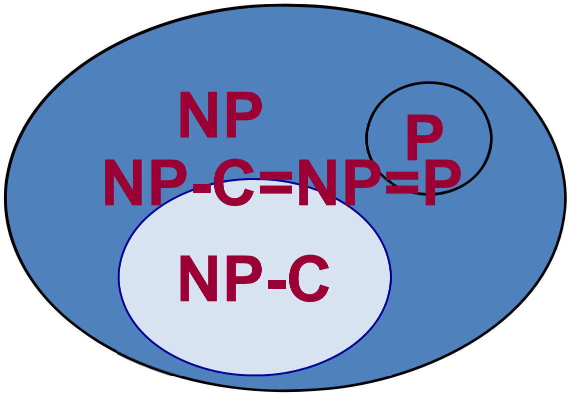
- Namely, NP-C are those problems *to which any other problem in NP can be reduced by means of a polynomial transformation*
- NP-C problems are equivalent to any other problem in NP since an algorithm for solving one NP-C problem can be used to solve any other problem in NP
- NP-C problems are equivalent to each other
- NP-C $\subset$NP

Transformations and NP-complete problems

Consequences of the definition of NP-C:

- If solving a problem $\pi \in$ NP-C with an efficient algorithm was possible then all the NP problem would be solved efficiently

- Therefore, to demonstrate that P=NP it is sufficient to prove that only one NP-C problem is in P



The Satisfiability problem

It is the first NP problem that has been demonstrated to belong to the class NP-C

Definition

SAT Problem

Given a boolean expression in conjunctive normal form with n variables $x_1,\ldots,x_n$, and their complements, the problem of establish if the expression is satisfiable

- Conjunctive normal form = conjunction (AND) of clauses composed by disjunction (OR) of literals (variables or their complements):

$$\bigwedge_{i=1}^{K}\left(y_{1_i} \vee \underset{\uparrow}{\ldots} \vee y_{h_i}\right) \quad \text{where } \; y_{j_i} \in \{x_1,\ldots,x_n,\bar{x}_1,\ldots,\bar{x}_n\}$$

clauses of cardinality *h*

The Satisfiability problem

Example

$$E = (x_1 \vee x_2 \vee x_4) \wedge (\bar{x}_1 \vee \bar{x}_2) \wedge (\bar{x}_4) \wedge (x_1 \vee x_3 \vee \bar{x}_4) \wedge (x_2 \vee \bar{x}_3 \vee x_4)$$

the problem = is there a True/False assignment to the variables such that E=True ?

- Cook (1971) demontrated that SAT$\in$ NP-C if $h\geq 3$ (Cook’s Theorem)
- Since then to prove that a problem $\pi\in$ NP is $\pi\in$ NP-C becomes simpler:
  - It is sufficient to consider another $\pi'\in$ NP-C and prove that $\pi' \propto \pi$
  - For transitivity of the polynomial transformation

$$\forall\ \pi'' \in \text{NP} \quad \pi'' \propto \pi' \propto \pi \ \Rightarrow\ \pi'' \propto \pi$$

NP-hard problems

Definition

A problem is NP-hard if there exists a NP-C decision problem that can be reduced to it with a polynomial transformation

- The considered problem must not be a decision problem, hence optimization problems that can be reduced to NP-C decision problems are NP-hard
- NP-hard problems are at least as difficult as any other NP-C problem Example: HC can be reduced to TSP
- If $\pi \in$ NP-C the problem instance of large enough dimension cannot be optimally solved within an acceptable computation time by exact algorithms

NP-hard problems

- In general it is only possible to determine acceptable (sub-optimal) solutions for $\pi \in$ NP-C by means of
  - **Approximated algorithms**: polynomial time algorithms that produce solutions guaranteeing a maximum (absolute or relative) distance from the optimum (e.g., la for and instance I algorithm A is $A(I) \leq 3 \cdot Opt(I)$)
  - **Heuristic algorithms**: polynomial time algorithms producing solutions that have been empirically verified to be good or acceptable, without providing any theoretical guarantee for the solution performance

- Types of heuristic algorithms:
  - Constructive heuristics (e.g., Nearest Neighborhood for TSP)
  - Improvement heuristics
  - Metaheuristics
  - Matheuristics

- Some NP-hard problems
  - Knapsack problem (weak NP-hard)
  - Set partitioning, packing, covering
  - Integer programming
  - TSP