Computational complexity theory

Introduction to computational complexity theory

- **Complexity (computability) theory deals with two aspects:**
  - **Algorithm's complexity.**
  - **Problem's complexity.**

- **References**
  - **S. Cook, « The complexity of Theorem Proving Procedures », 1971.**
  - **Garey and Johnson, « Computers and Intractability, A guide to the theory of NP-completeness », 1979.**
  - **J. Carlier et Ph. Chrétienne « Problèmes d'ordonnancements : algorithmes et complexité », 1988.**

Basic Notions

- Some problem is a “question” characterized by parameters and needs an answer.
  - Parameters description;
  - Properties that a solutions must satisfy;
  - An instance is obtained when the parameters are fixed to some values.
- An algorithm: a set of instructions describing how some task can be achieved or a problem can be solved.
- A program : the computational implementation of an algorithm.

Algorithm’s complexity (I)

- There may exists several algorithms for the same problem
- Raised questions:
  - Which one to choose ?
  - How they are compared ?
  - How measuring the efficiency ?
  - What are the most appropriate measures, running time, memory space ?

Algorithm's complexity (II)

- Running time depends on:
  - The data of the problem,
  - Quality of program...,
  - Computer type,
  - Algorithm's efficiency,
  - etc.
- Proceed by analyzing the algorithm:
  - Search for some *n* characterizing the data.
  - Compute the running time in terms of *n*.
  - Evaluating the number of elementary operations, *(elementary operation = simple instruction of a programming language).*

Algorithm's evaluation (I)

- Any algorithm is composed of two main stages: initialization and computing one
- The complexity parameter is the size data n (binary coding).

Definition:

Let be $n>0$ and $T(n)$ the running time of an algorithm expressed in terms of the size data n, $T(n)$ is of $O(f(n))$ iff $\exists n_0$ and some constant c such that:

$\forall n \geq n_0$, we have $T(n) \leq c\, f(n)$.

If f(n) is a polynomial then the algorithm is of polynomial complexity.

- Study the complexity in the worst case.
- Study the complexity in average : *tm(n)* is the mean value of execution time when the data and the associated distribution law are given.

Algorithm's evaluation (II) example

Given N numbers a_1, a_2, .., a_n in {1,…, K}.

The algorithm MIN finds the minimum value.

Algorithm MIN

```
Begin
  for i=1 to n read a_i;
  B:=a_1, j:=1;
  for i=2 to n do :
        if B=1 then j:=n;
        elseif a_i < B then
              begin
                    j:=i;
                    B:=a_i;
              end;
  write : the min value is a_j;
End.
```

Algorithm's evaluation (III)

- Study the complexity of algorithm MIN.

  Proposition 1. The complexity of Algorithm MIN is in O(n).

  Proposition 2. if numbers a_i are independent random values and if any a_i has some probability 1/K to be equal to 1, the complexity in average of algorithm MIN, apart the reading data phase, is in O(1).

Importance of polynomial algorithms (I)

f(n)	n=10	n=20	n=30	n=40	n=50
n	0.00001 sec	0.00002sec	0.00003 sec	0.00004 sec	0.00005 sec
n^2	0.0001 sec	0.0004 sec	0.0009 sec	0.0016 sec	0.0025 sec
n^3	0.001 sec	0.008 sec	0.027 sec	0.064 sec	0.125 sec
2^n	0.001 sec	1 sec	17.9 min	12.7 days	35.7 years
3^n	0.059 sec	58 min	6.5 years	3.855 centuries	$2*10^8$ centuries

An elementary operation is run in one microsecond.

Importance of polynomial algorithms (II)

f(n)	Todays computers	100 times faster	1000 times faster
n	N1	100 N1	1000 N1
n^2	N2	10 N2	31.6 N2
n^3	N3	4.64 N3	10N3
2^n	N4	N4 +6.64	N4 + 9.97
3^n	N5	N5 + 4.19	N5 + 6.29

Problem's sizes solved in one hour run time

Computational complexity theory

- The decision problem is some mathematical question requiring some answer yes or no.
- Computational Complexity Theory is concerned with the question: for which decision problems do efficient algorithms exist ?

Decision problems: SAT

- Satisfiability is the problem of determining if the variables of a given boolean formula can be assigned in such a way as to make the formula evaluate to TRUE.
- In complexity theory, the Boolean satisfiability problem (SAT) is a decision problem, whose instance is a Boolean expression written using only AND, OR, NOT, variables, and parentheses.
- The question is: given the expression, is there some assignment of *TRUE* and *FALSE* values to the variables that will make the entire expression true?
- A formula of propositional logic is said to be *satisfiable* if logical values can be assigned to its variables in a way that makes the formula true.

Travelling salesman problem

- Given a weighted graph G=(X,E,v)

  X = Vertices (= Cities)
  E = Edges (pair of cities)
  v = Distances between cities

- *Question: is-there a tour that visits all cities exactly once and its length(weight) is less than a given number B ?*

Partition problems

- Partition problem
  - Data: given a set $A=\{a_i \mid i \in I\}$ of n integer numbers.
  - Question : is-there some partition of A in two subsets A1 and A2 of equal weight ?
- Tripartition problem
  - Data: given a set $A=\{a_i \mid i \in I\}$ of n=3q integer numbers such that $\sum_{i \in I} a_i = qB$ and $B/4 < a_i < B/2$ and B some positive integer.
  - Question : Is-there some partition of A in q subsets of cardinal 3 and weight B?

Some equivalent problems

- Vertex covering.
  - Data : a graph $G=(V,E)$ with V a set of vertex, E a set of edges and some positive integer $B \leq |V|$.
  - Question : Is-there some subset $V' \subseteq V$ such that $|V'| \leq B$, and for each edge $(i,j) \in E$, $i \in V'$ or $j \in V'$?
- Independent set
  - Data : a graph $G=(V,E)$ with the set of vertex *V and E* the set of edges and some positive integer $B \leq |V|$.
  - Question : Is-there some $V' \subseteq V$ $|V'| \geq B$ such that for any $(i,j) \in E$, $i \notin V'$ or $j \notin V'$?
- Maximal clique.
  - Data : a graph G = (V,E) where V is the vertex set and E the set of edges, and a positive integer B.
  - Question : Is-there some $V' \subseteq V$ such that the corresponding sub-graph is complete and of size greater or equal to B?

Exact cover

- Exact cover by 3-sets (*X3C*) *Problem*.
  - Data : Given a finite X with $|X|=3q$ and a collection C of 3-elements subsets of X,
  - Question : is-there some exact cover of X, that means is-there a sub-collection $C' \subseteq C$ such that $\forall x \in X$, there is a only one $c \in C'$ with $x \in c$?

Scheduling problems

- One machine problem
  - Data : a set I of n independent and indivisible tasks; for each task $i \in I$ we have its duration p_i, availability date r_i and its deadline d_i.
  - Question : is-there some scheduling ϕ of these n tasks in a single machine that satisfies the availability and deadline dates?
- Two processors problem
  - Data : a set I of n independent and indivisible tasks , durations p_i and a positive integer B.
  - Question : Is-there a scheduling ϕ of these n tasks on two processors of duration less or equal to B ?
- NTRM
  - Data : a set I={1,2,.. n} of tasks of durations 1 and deadlines d_1, d_2, .., d_n, a partial order < on I, and a positive integer B.
  - Question : Is-there a scheduling ϕ of these tasks on one machine satisfying the partial order and such that the number of delayed tasks is $\leq$ B.

Complexity theory: basic notions

- Why using the decision problems?
  - To introduce a simple formalism and making possible and easier the comparison between problems.
- *The complexity theory relies on Turing Machine…*
  - **….**

The class NP

Alternatively:

We distinguish the following complexity classes:

- Class P : some problem is in P if it can be solved in polynomial time to the size of data by a determinist algorithm.
- A problem is said to be in **NP** if and only if for a guessed solution there exists a polynomial time algorithm verifying the solution.
- *Class NP : it groups all decision problems such that an answer yes can be decided by a non-determinist algorithm in polynomial time to the size of data.*
  - **Polynomial time checking**

NP-completeness

Paper of Stephen Cook, «The complexity of Theorem Proving Procedures», 1971.

- Defines the polynomial reduction
- Defines the decision problems and the class NP.
- Shows that the problem SAT is at least as difficult as all the others in NP → NP-complete

Complexity theory polynomial reduction

- Polynomial reduction allows to compare NP problems in terms of computational complexity
- *Definition: P_1 is polynomially reduced to P_2 ($P_1 \propto P_2$) if P_1 is polynomial or there exists a polynomial algorithm A that builds for any d_1 of P_1 some data d_2 of P_2 such that d_1 has answer YES iff $d_2=A(d_1)$ has answer YES.*
- Some problem is said NP-Complete if it is in NP, and any NP-Complete problem can be polynomially reduced to this problem.
- The polynomial reduction defines a pre-order relation on NP.
- Cook'sTheorem : SAT is NP-Complete.

NP-completeness (I)

- The general method:
  - 1) show first that $\Pi \in$ NP
  - 2) show that there exists P'$\in$NP-complete such that P'$\propto \Pi$.
- The following decision problems are NP-complete.
  - TSP,
  - Partition problems,
  - Exact cover
  - Clique

NP-completeness (II)

- Three main techniques are used to show the NP-completeness of combinatorial problems.
  - Restriction
    - examples : minimal covering, knapsack, etc.
  - Local replacement
    - examples : 3SAT, X4C…
  - Component design
    - example : NTRM…

NP-completeness (III)

Show the following results :

- Proposition 1. Two processors problem is NP-complete.
  - hint: ***partition*** $\propto$ ***two processors***.
- Proposition 2. One machine problem is NP-complete.
  - hint: ***partition*** $\propto$ ***one machine problem***.

Exercises

1) Show that the knapsack problem is NP-complet.
   - Data: a finite set X, for all $x_i \in X$, there is a weight $s(x_i)$, a value $v(x_i)$. There are also a number $B \in Z^+$ and a number $K \in Z^+$.
   - Question : Is-there any $X' \subseteq X$ such that $\sum_{xi \in X'} s(x_i) \leq B$ and $\sum_{xi \in X'} v(x_i) \geq K$.

   hint: use partition.

2) Show that the problem X4C, is NP-complete. (*hint: X3C)*.

3) Show that the minimum sum oif squares problem is NP-complete.
   - data: a finite set A of positive integer numbers ($a \in Z+$) and two positive integers K and J.
   - Question : is-there any partition of A into K disjoint sets A_1, A_2, .. A_K such that $\sum_{i \leq K} (\sum_{a \in Ai} a)^2 \leq J$.

     hint: bipartition, tripartition...

NP-completeness : conjecture

- Fundamental Conjecture: P≠NP.

You win 1000000 USD if You show that P=NP or P≠NP.

Pseudo-polynomial algorithms (I)

- Dynamic programming
  - Idea : breaking down the initial problem in a sequence of simpler problems, solving the n-*th* problem can be done by recurrence on this of (n-1)-*th* one.
  - WESS problem (weight of a subset)
    - Data : a finite set A composed of n elements $a_i \in Z+$ and a positive integer K.
    - Question : is-there a subset of A of weight K ?

Pseudo-polynomial algorithms (II)

Algorithm WESS
Begin
for k=0 to $\sum_{ai \in A} a_i$
for j=1 to n do :
WEIGHT(k,j):=false;
end for;
end for;
set WEIGHT(0,0) := true;
for j=1 to n
for k=0 to $\sum_{ai \in A} a_i$ do :
if ($k \geq a_j$ and WEIGHT(k-a_j, j-1)=true) or WEIGHT(k, j-1)=true
then WEIGHT(k, j)=true;
end for;
end for;
end.

Proposition : Algorithm WESS is of complexity $O(n\sum_{ai \in A} a_j)$ and assigns true to WEIGHT(K, n) if there exists some subsets of n numbers $a_1, a_2, .., a_n$ of weight K.

Exercise: AN INVESTMENT PROBLEM

An example. 6 million is at our disposal, to invest in 3 regions. The following table shows the benefits given by the invested sums.

	Region I	Region II	Region III
1 million	0.2	0.1	0.4
2 million	0.5	0.2	0.5
3 million	0.9	0.8	0.6
4 million	1	1	0.7

1) Determine the optimal investment policy for the three regions using a "dynamic programming" method. The idea is to associate a graph with levels to the data. Level 0 contains only the vertex (0,0), (because no money has been invested yet). Level 1 contains the vertices (1,0) (1,1) (1,2) (1,3) (1,4), which correspond to the cumulated amounts invested in region 1. Level i contains the vertices (i, 0), (i, 1), (i, 2), (i, 3), (i, 4), (i, 5), (i, 6), which correspond to the sums invested in the regions 1 .. i (i = 2, 3). The arcs are placed between the levels i and i +1, valuated by the sums invested in the region i +1. The last vertex is (3,6). The goal is to seek a maximum value path in this graph.
2) The general case. More generally, we have B million to invest in n regions. We shall set fi(y), the optimal profit for a cumulative investment of a sum y in the regions 1, 2, .., i. We have f0(0) = 0. Determine a recurrence formula connecting fi to fi-1 for i from 1 to n.
3) What is the complexity of the dynamic programming method in function of n and B, and complexity of enumerating all possible solutions.

Strong NP-completeness (I)

- *Binary code* is the system of representing text or computer processor instructions by the use of the *Binary* number system's two-*binary* digits "0" and "1".
- Binary code of a → $\log_2(a)$ places
  - example : the size of 2^{31} is $\log_2(2^{31}) + 1 = 32$, thus we need 32 bits to code it in machine;
- In computational complexity theory, a numeric algorithm runs in pseudo-polynomial time if its running time is polynomial in the *numeric value* (unary code) of the input.

Strong NP-completeness (II)

- An NP-complete problem with known pseudo-polynomial time algorithms is called weakly NP-complete.
- An NP-complete problem is called strongly NP-complete if any pseudo-polynomial time algorithm solving it implies the existence of a polynomial time algorithm solving. it
  - Examples : TSP, tripartition, clique, sat.
  - What to say about bipartition, two-processeurs ? And about one machine ?

Combinatorial optimization problems

- *Research problems*
  - A research problem consists in a set of instances D_M and of solutions $S_M(D)$. Solving it for instance $D \in D_M$ comes to show that $S_M(D)$ is empty or provide a solution from $S_M(D)$.
  - In an optimization problem, $S_M(D)$ is the set of « optimal solutions ».
  - A decision problem can be seen as a special case of a research problem: the set of solution is empty if the answer is NON and otherwise $S_M(D)=\{YES\}$.

Turing reduction

- Turing Reduction : a research problem P1 is polynomialy reduced to a research problem P2 by Turing reduction *(P1 $\propto_T$ P2)* , if it exists algorithm A1 using as a subroutine an algorithm A2 solving P2 such that A1 is polynomial-time if any call to A2 is taken to be of O(1).
  - *A1 is polynomial-time if A2 is polynomial-time.*
  - *Turing reduction defines a preorder relation on the research problems.*
  - *Turing reduction generalizes the polynomial reduction to research problems.*
- Definition : some problem P2 is said NP-hard if it exists P1 NP-complete such that *(P1 $\propto_T$ P2).*

END