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.
```

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 ?

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?

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,
  - Clique
  - …

NP-completeness : conjecture

- Fundamental Conjecture: P≠NP.

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

Methods for dealing with NP-Complet problems

- Exact methods
  - Dynamic programming
  - Tree methods
- Heuristic methods

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 ∑_{ai∈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 ∑_{ai∈A} a_i do :
       if (k≥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.

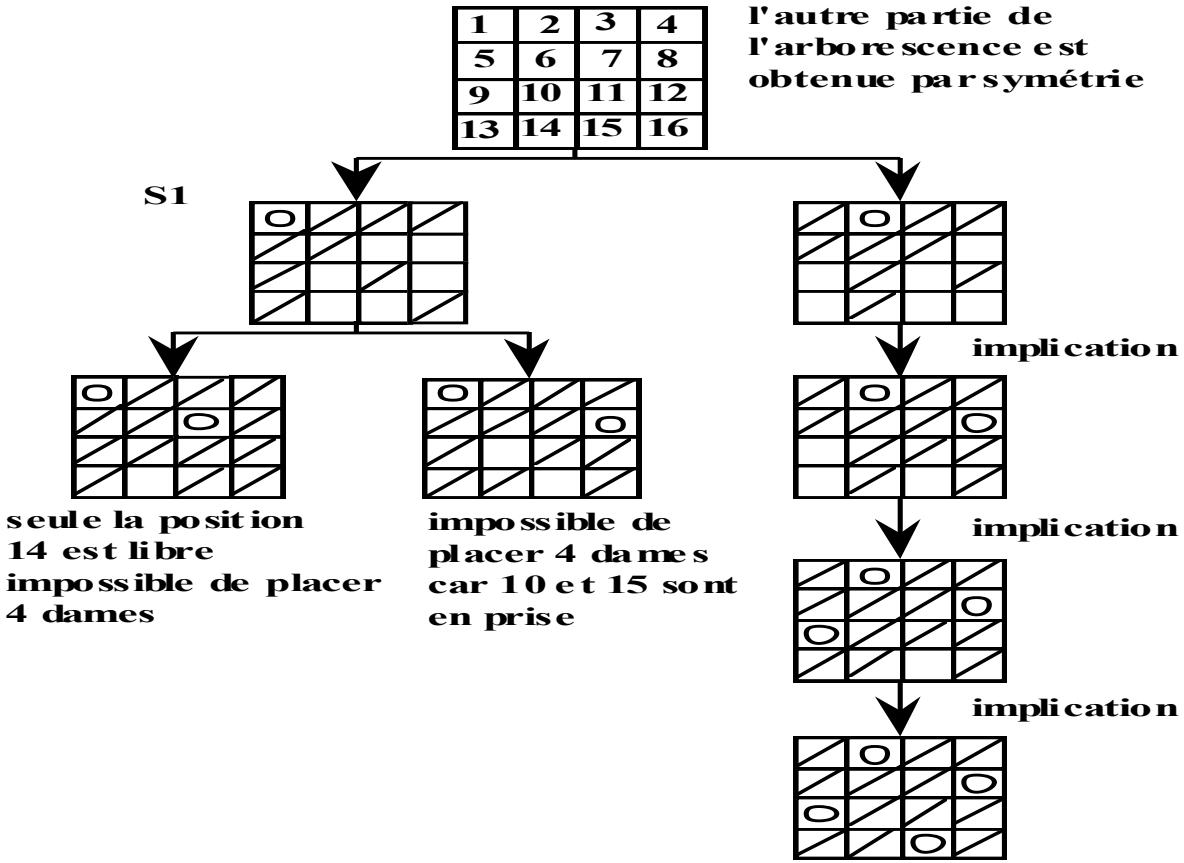
	Region I	Region II	Region III
1 million	0.2	0.1	0.4
2 million	0.5	0.2	0.5
3 million	0.9	0.8	0.6
4 million	1	1	0.7

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.

Tree methods

Tree methods: a family of algorithms that implicitly list all the solutions to a problem.

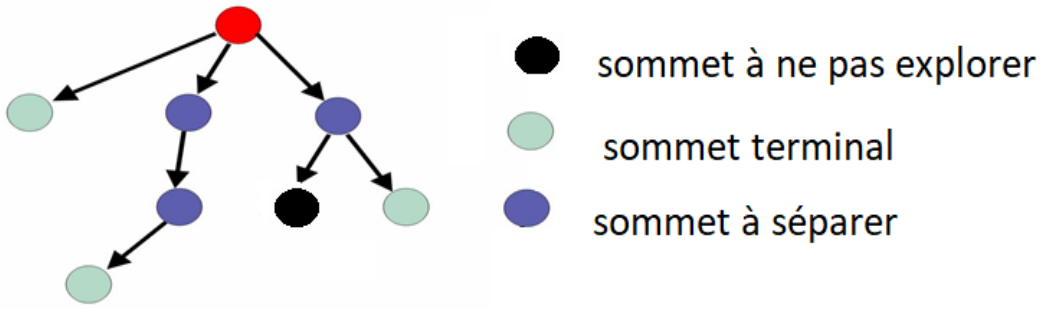
Gauss's ladies' tree



Tree methods

Tree methods use separation and evaluation algorithms built on the following elements:

- Separation consists of breaking down (recursively) the universe of solutions into several generally disjointed subsets whose union covers all possible solutions.
- evaluation consists of associating a B-marker with the problem studied that is calculated for each branch explored. A lower terminal (respectively an upper terminal) is referred to in the case of a minimisation (in the case of maximization, respectively). This terminal allows us to estimate the performance of the sector evaluated at best.
- exploration consists of determining the order of visit of the different branches by choosing to explore the most promising branches. During the exploration we can distinguish the following cases: a summit not to be explored, a summit to separate, a terminal summit.



Tree method – an exercice

Five people, with different nationalities, live in the first five houses of a street, all on the same side of the street (numbered from 1 to 5).

These people have each a different job as well as a favourite drink and animal (each different for each person).The five houses are each of a different colour.

1 : The englishman lives in the red house.

2 : The spaniard has a dog.

3 : The japanese man is a painter.

4 : The italian drinks tea.

5 : The nowegian lives in the first house on the left.

6 : The owner of the green houses drinks coffee.

7 : This green house is immediately on the right of the white house.

8 : The sculptor raises snails.

9 : The diplomat lives in the yellow house.

10 : The person who lives in the middle house drinks milk.

11 : The norwegian lives next to the blue house.

12 : The violinist drinks fruit juice.

13 : The fox is in a house next to the doctor's house.

14 : The horse is next to the diplomat's house.

Who has a zebra and drinks water ? *(Attributed to Lewis Carrol)*

END