Computational Complexity of the Decision and Optimization Problems

- ***Decision Problem*: given a problem, determine if at least a solution exists for this problem.**
- **Example: *Feasibility Problem*: determine if at least a feasible solution exists for the considered problem.**
- ***Optimization Problem*: determine a feasible solution that maximizes (or minimizes) the objective function of the considered problem.**

Computational Complexity of the Decision and Optimization Problems (2)

- ***Size*** **of a problem** ***R*****: number of "symbols" (bit, bytes, words, …) needed to represent the input data of an instance of** ***R*** **(by neglecting the proportionality constants).**

- **Example:** ***KP-01*****: input data:** n, C, (P_j), (W_j):
  - * (P_j): n values, (W_j): n values
  - * $(2\,n + 2)$ values (symbols): size $= n$

Computational Complexity of the Decision and Optimization Problems (3)

- **Given a problem *R*: Determine the *computing time* (number of *elementary operations*), expressed as a *function of the size* of *R*, to find the solution of *R* in the *worst case*.**

- **The theory of the *Computational Complexity* of the problems has been analyzed for the *Decision Problems*, but it can be applied to the *Optimization Problems* as well.**

Polynomial Problems

- ***Polynomial Problem R*: *R* can be solved *in the worst case* through at least one *algorithm* whose *computing time* is a *polynomial function* of the *size* of *R*.**

- **Example: Given an array of *n* elements (*size = n*):**

*** *Minimum value* of *n* elements: *O(n)* time.**

*** *Sorting* of *n* elements: *O(n log n)* time.**

Classes P and *NP*

- ***CLASS P*** **contains all the *Polynomial Problems*.**

- ***CLASS NP*** **contains all the problems that can be solved in polynomial time in the best case (through a "non deterministic Turing Machine").**

Class NP

From an operational point of view, a problem *R* belongs to *Class NP* if it can be solved through a *Decisional Tree* such that:

1) **the number of "*levels*";**
2) **the number of "*descendent nodes*" of each node;**
3) **the *computing time* required to consider each node**

are *polynomial functions* of the size of *R*.

Class P* is contained in *Class NP

***Class P* = *Class NP* ?**

Example of a Problem in Class NP

Knapsack Problem in Decision Version :

Given an instance of *KP-01*, determine if there exists at least one feasible solution whose profit is not smaller than a given value *K*: *KP-01(K)*

Binary Decision Tree

*** at each level j (j = 1, 2, …, n) consider item j and fix the value of x_j to 0 or to 1:**

- **n levels;**
- **2 descendent nodes for each node;**
- **constant computing time for each node.**

KP-01(K) $\in$ ***Class NP***

Knapsack Problem in Decision Version

Binary Decision Tree

* at each level j (j = 1, 2, …, n) consider item j and fix the value of x_j to 0 or to 1.

In the worst case, the algorithm requires a computing time proportional to the number of all the nodes of the binary decision tree (exponential time with respect to the size of *KP-01*(*K*)).

Also *KP-01* $\in$ *Class NP*

Complexity of the Problems

From a practical point of view, the Feasibility and Optimization Problems can be subdiveded in three classes:

1) *Polynomial Problems* (*Class P*);

2) *NP-Hard Problems*: belong to *Class NP*, but no algorithm has been proposed for their solution in the worst case (example: *KP-01*);

3) *Surely Difficult Problems*: do not belong to *Class NP* (example: determine all the optimal solutions of *KP-01*; the number of such solutions could be exponential with respect to the size *n*).

NP-Hard Problems

A problem *R* is *NP-Hard* if:

1) $R \in$ *Class NP* ;

2) There exists an *NP-Hard Problem* *T* which is "*reducible*" to *R* (*T* *red* *R*):

for any instance of *T*, it is possible to define, in a computing time polynomial in the size of *T*, an instance of *R* such that, determined the solution of this instance of *R*, the solution of the instance of *T* can be obtained in a computing time polynomial in the size of *T*.

Partition Problem (PP)

Given: **m positive values: a_j (j = 1, …, n),**

one positive value b

determine if there exists a subset of the n values whose sum is exactly equal to the given value b.

Feasibility Problem

***Size*: m + 2 : m**

PP **is known to be *NP-Hard***

(even if $b = \Sigma_{j=1,m} \; a_j \;/\; 2$)

PP is reducible to KP-01

PP *red* *KP-01*

Given any instance of *PP*: m, (a_j), b

1) **Define (in time $O(m)$) an instance $(n, (P_j), (W_j), C)$ of *KP-01*:**

   * $n = m$

   * $C = b$

   * $P_j = a_j$ ($j = 1, \ldots, n$),

   * $W_j = a_j$ ($j = 1, \ldots, n$).

2) **Determine the optimal solution $(x_1, x_2, \ldots, x_n, z)$ of *KP-01*.**

3) **If $z = C$: PP has a feasible solution $(x_1, x_2, \ldots, x_n)$**

   If $z < C$: PP has a no feasible solution

   Computing time $O(m)$

Multiple Choice KP (MCKP)

In addition to the input data for KP01:

the set of the n items is *partitioned* into k disjoint subsets N_1, N_2,…, N_k .

- **determine a subset of the n items, with at most one item for each subset N_h (h = 1, …, k), so as to maximize the global profit, and such that the global weight is not larger than the knapsack capacity C.**
- **Size: 2 n + 3 + $k * n$ (matrix A_{hj}), with $k \le n$: $n * n$**
- **Size: 2 n + 3 + n (partition of 1, 2, …, n) : n .**
- ***Binary Decision Tree*: similar to the decision tree of *KP-01*): *n levels, 2 descendent nodes for each node*:**
- ***MCKP* $\in$ *Class NP* ;**
- ***MCKP* is a “generalization of *KP-01* : *KP-01 red MCKP***