

Knapsack Problem (KP01)

Given:

n items,

P_j “profit” of item j , $j = 1, \dots, n$ ($P_j > 0$),

W_j “weight” of item j , $j = 1, \dots, n$ ($W_j > 0$),

one container (“knapsack”) with “capacity” C :

determine a subset of the n items so as to **maximize** the global profit, and such that the global weight is **not larger** than the knapsack capacity C .

***KP01* is NP-Hard**

Knapsack Problem (KP01)

Determine a subset of items so as to maximize the global profit, and such that the global weight is not larger than the knapsack capacity C .

We assume:

$$n \geq 2$$

$$P_j > 0, \quad j = 1, \dots, n$$

$$W_j > 0, \quad W_j \leq C, \quad j = 1, \dots, n$$

$$\sum_{j=1, n} W_j > C$$

Mathematical Model KP01

$$x_j = \begin{cases} 1 & \text{if item } j \text{ is inserted in the knapsack} \\ 0 & \text{otherwise} \end{cases} \quad (j = 1, \dots, n)$$

$$\max \quad \sum_{j=1, n} P_j x_j$$

$$\sum_{j=1, n} W_j x_j \leq C$$

$$x_j \in \{0, 1\} \quad (j = 1, \dots, n)$$

or

$$0 \leq x_j \leq 1 \quad \text{integer} \quad (j = 1, \dots, n)$$

ILP Model (Binary Linear Programming Model)

There exist items j with $P_j < 0$ and $W_j < 0$

Let $N = \{j: P_j > 0 \text{ and } W_j > 0, j = 1, \dots, n\}$;

Let $R = \{j: P_j < 0 \text{ and } W_j < 0, j = 1, \dots, n\}$.

Set $y_j = x_j, P'_j = P_j, W'_j = W_j \quad j \in N$

Set $y_j = 1 - x_j, P'_j = -P_j, W'_j = -W_j \quad (x_j = 1 - y_j) \quad j \in R$

$$Z = \sum_{j=1, n} P_j x_j = \sum_{j \in N} P_j y_j + \sum_{j \in R} P_j (1 - y_j) =$$

$$\sum_{j \in N} P'_j y_j + \sum_{j \in R} P'_j y_j + \sum_{j \in R} P_j = \sum_{j=1, n} P'_j y_j + a$$

where $a = \sum_{j \in R} P_j$

There exist items j with $P_j < 0$ and $W_j < 0$

Let $N = \{j: P_j > 0 \text{ and } W_j > 0, j = 1, \dots, n\}$;

Let $R = \{j: P_j < 0 \text{ and } W_j < 0, j = 1, \dots, n\}$.

Set $y_j = x_j, P'_j = P_j, W'_j = W_j \quad j \in N$

Set $y_j = 1 - x_j, P'_j = -P_j, W'_j = -W_j \quad (x_j = 1 - y_j) \quad j \in R$

$$Z = \sum_{j=1, n} P_j x_j = \sum_{j=1, n} P'_j y_j + a \quad \text{where } a = \sum_{j \in R} P_j$$

$$\sum_{j=1, n} W_j x_j = \sum_{j=1, n} W'_j y_j + b \quad \text{where } b = \sum_{j \in R} W_j$$

There exist items j with $P_j < 0$ and $W_j < 0$

$$\max \quad \sum_{j=1, n} P'_j y_j + a$$

$$\sum_{j=1, n} W'_j y_j \leq C' \quad (\text{where } C' = C - b)$$

$$y_j \in \{0, 1\} \quad (j = 1, \dots, n)$$

$$\text{with } P'_j > 0, \quad W'_j > 0 \quad (j = 1, \dots, n)$$

$$x_j = y_j \quad j \in N; \quad x_j = 1 - y_j \quad j \in R$$

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
 - * $(2n + 2)$ 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

A **Problem R** is **Polynomial** if R can be solved through at least one algorithm whose computing time in the worst case grows according to a polynomial function of the size of R .

- **Examples:**

given an array A_j of n elements ($j = 1, \dots, n$):

- Determine the **minimum value** of the n elements:

effective algorithm: $O(n)$ time (in the worst case the computing time is proportional to n : “linear” in n).

- **Sort** the n elements according to non-decreasing (or non-increasing) values:

effective algorithm: $O(n \log n)$ 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 ($j = 1, 2, \dots, n$) 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 for $KP-01(K)$

* at each level j ($j = 1, 2, \dots, n$) 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 global number of 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 subdivided in three main classes:

- 1) *Polynomial Problems (Class P)*;
- 2) *NP-Hard Problems (also called NP-Complete Problems)*: belong to *Class NP*, but no polynomial 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) $R \in \text{Class } NP$;

2) There exists an *NP-Hard Problem* T which is “*reducible*” to R ($T \propto R$):

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

Partition Problem (PP)

Given: m positive values: a_j ($j = 1, \dots, n$),
one positive value b

determine if there exists a **subset** of the m 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 $b = \sum_{j=1,m} a_j / 2$)

KP-01 is NP-Hard

1) *KP-01* \in *Class NP*

2) *PP* \propto *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*:

* $n = m$

* $C = b$

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

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

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

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

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

Computing time $O(m)$

Particular Cases of KP01

a) *Constant Profits: the Problem is Polynomial*

$$P_j = K \quad (j = 1, \dots, n)$$

1) *Sort* the n items according to **non-decreasing values of the weights W_j** : ($O(n \log n)$ time);

2) Insert the items until the first item s is found such that:

$$\sum_{j=1, s} W_j > C \quad (\text{items } s, s+1, \dots, n \text{ are not inserted}):$$

$O(n)$ time.

Global computing time: $O(n \log n) + O(n): O(n \log n)$

Particular Cases of KP01 (2)

b) *Constant Weights: the Problem is Polynomial*

$$W_j = K \quad (j = 1, \dots, n)$$

1) *Sort* the n items according to **non-increasing values of the profits P_j** : ($O(n \log n)$ time);

2) Insert the items until the first item s is found such that:

$$\sum_{j=1, s} W_j > C \quad (\text{items } s, s+1, \dots, n \text{ are not inserted}):$$

$O(n)$ time.

Global computing time: $O(n \log n) + O(n): O(n \log n)$