The Knapsack Problem

The input is n objects where object i is described by the pair (w_i, u_i). w_i is the weight of the object and u_i is the utility; both are positive integers.

The decision problem is to decide if it is possible to make a choice of objects with total weight at most W and such that the utility sum is at least U.

This problem is NP-Complete.

The corresponding optimization problem can be stated

$$\max \sum_i x_i u_i$$

$$\text{such that } \sum_i x_i w_i \leq W$$

$$\text{and } x_i \in \{0, 1\} \; \forall i$$

A Dynamic Programming solution to the knapsack problem

Let $V_{i,j}$ be the smallest total weight that gives the utility i when we just use the j first objects. We have $1 \le j \le n$ and $0 \le i \le n \max u_i$ since the maximal utility is at most $n \max u_i$.

We get

$$
\begin{aligned}
&V_{0,j} = 0, \quad j = 1, \ldots, n \\
&V_{u_1,1} = w_1 \\
&V_{i,1} = \infty, \quad i \neq u_1 \\
&V_{i,j} = \min\left\{V_{i,j-1}, V_{i-u_j,j-1} + w_j\right\}
\end{aligned}
$$

The time it takes to compute the array is $O(n^2 \max u_i)$.

The optimal value is the largest i such that $V_{i,n} \le W$.

Pseudo-polynomial algorithms

An algorithm A is *pseudo-polynomial* if the time complexity is polynomial in n and M, the greatest number in the input.

An algorithm that is pseudo-polynomial can be exponential in n. I input is an n-digit binary number we have $M \sim 2^n$.

If we assume that $P \neq NP$ we say that a *strongly NP-Complete* problem is an NP-Complete problem which cannot be solved by a pseudo-polynomial algorithm.

An approximation algorithm for Knapsack

The exact Dyn P - algorithm works well if the coefficients u_i are small. Therefore, we ''scale'' them with a factor δ and then run the exact algorithm:

ApproxKnapsack(U, W)
(1) $\delta \leftarrow \frac{2n}{\epsilon u_{max}}$
(2) **for** $i = 1$ **to** n
(3) $u'_i \leftarrow \lfloor \delta u_i \rfloor$
(4) **return** ExactKnapsack(U', W)

(We let $u_{max} = \max u_i$ and $w_{max} = \max w_i$.)

Questions:

- What is the approximation quotient?
- What is the time complexity?

Analysis of the algorithm

We assume that

1. $w_{max} \leq W$; $w_i > W$ can be removed.

2. $W \leq nw_{max}$; otherwise we could choose all objects.

Let opt be the value of the optimal solution to the (U, W)-instance and $approx$ be the value returned by our algorithm.

Let $I \subseteq [n]$ be the indices of the objects that form the solution to the (U, W)-instance and I' be the solution to the (U', W)-instansen. Then

$$approx = \sum_{i \in I'} u_i \geq \frac{1}{\delta} \sum_{i \in I'} \lfloor \delta u_i \rfloor \geq \frac{1}{\delta} \sum_{i \in I} \lfloor \delta u_i \rfloor$$

$$\geq \frac{1}{\delta}\Big(\delta \sum_{i \in I} u_i - n\Big) = \frac{1}{\delta}(\delta opt - n) =$$

$$= opt - \frac{n}{\delta} \geq opt(1 - \epsilon/2)$$

Analys, cont.

The approximation quotient is

$$\frac{opt}{approx} \leq \frac{1}{1-\epsilon/2} \leq 1+\epsilon$$

The running time is $O(n^3/\epsilon)$.

It means that we can get an approximation quotient arbitrarily close to 1 and still run in polynomial time.

This is an example of an **P**olynomial **T**ime **A**pproximation **S**cheme, PTAS).

There are several other PTAS:s for other NP-Complete problems based on similar types of scalings.

Terminology

NPO

The class of all optimization problems corresponding to decision problems in NP.

APX

The problems that can be approximated within some constant.

PTAS

The problems that can be approximated within any constant $1 + \epsilon$.

There are problems, like Vertex Cover, that belongs to **APX** but not to **PTAS** (Assuming $P \neq NP$).

Probabilistic approximation of MAX 3-CNF SAT

We have a 3-CNF-formula $\Phi = c_1 \wedge \cdots \wedge c_m$ where each clause c_i contains exactly 3 distinct literals. We want to choice values for the boolean variables $x_1, \ldots, x_n$ such that the number of satisfied clauses is maximal.

RandomMax3CNFSAT$(n, \{c_i\}_{i=1}^{m})$
(1) **for** $i = 1$ **to** n
(2) $x_i \leftarrow Random(0, 1)$
(3) **return** $\{x_i\}_{i=1}^{n}$

Look at the clause c_i:

$$[c_i \text{ is not satisfied}] = \frac{1}{8}$$

Let Y be the number of satisfied clauses. Then

$$[Y] = \sum_i [c_i \text{ is satisfied}] = \frac{7m}{8} \geq \frac{7}{8} opt$$

The algorithm has an approximation quotient $\frac{8}{7}$ *in the mean*.

Bin Packing

Given numbers $S = \{s_1, s_2, ..., s_n\}$ $0 < s_i \leq 1$ we want to pack the numbers in bins that contains exactly the sum 1. We want to use as few bins as possible.

Decision problem: Given S and K, is it possible to pack the numbers into K bins?

The problem is NP-Complete.

Proof: We show that PARTITION $\leq$ BIN PACKING.

Given a list P of numbers we compute $W = \sum_i p_i$. We can assume that $p_i \leq \frac{W}{2}$ for all i (otherwise, the answer to the problem would be no trivially.) Now rescale:

$s_i = \frac{2p_i}{W}$

Let $K = 2$. This gives us an instance of BIN PACKING.

Approximation of Bin Packing

We use a method called First Fit Decreasing (FFD).

Sort S such that $s_1 \geq s_2 \geq ... \geq s_n$. We can assume that we have a set of empty bins ready. We place s_1 in the first bin. Then we place each number in the first possible bin.

Ex: $S = \{0,8\,,\,0,5\,,\,0,4\,,\,0,4\,,\,0,3\,,\,0,2\,,\,0,2\,,\,0,2\}$.

FFD will place the numbers as:

L_1	0,8 , 0,2
L_2	0,5 , 0,4
L_3	0,4 , 0,3 , 0,2
L_4	0,2

This placing is not optimal since it is possible to place the numbers in 3 bins. (How?)

The complexity is $O(n^2)$. We now try to find an approximation quotient.

Claim: Let OPT be the minimal number of bins needed in the optimal solution. All numbers that are placed in ''superfluous'' bins by our algorithm are $\leq \frac{1}{2}$.

Why?: Numbers $> \frac{1}{2}$ must be placed in different bins. There is no choice for these numbers. The only numbers that can be placed in a wrong bin are the ones $\leq \frac{1}{2}$.

Claim: Less than $OPT - 1$ numbers are placed in 'superfluous' bins.

Why: We know that all numbers can be placed in OPT bins. This means that $\sum_i s_i \leq OPT$.

Assume OPT numbers $t_1, t_2, ..., t_{OPT}$ are placed in superfluous bins. Let b_i be the sum of the numbers in bin i for $1 \le i \le OPT$ after we have run our algorithm. Then $t_i + b_i > 1$.

$\sum_i s_i \ge \sum_{i=1}^{OPT} b_i + \sum_{i=1}^{OPT} t_i = \sum_{i=1}^{OPT} (b_i + t_i) > OPT$. That gives us a contradiction.

If $OPT = m$ we now know that at most $m-1$ numbers of size $\frac{1}{2}$ ate placed in superfluous bins.

$APP \le m + \lceil \frac{m-1}{2} \rceil$

$B \le 1 + \frac{\lceil \frac{m-1}{2} \rceil}{m} \le 1 + \frac{\frac{m}{2}}{m} = \frac{3}{2}$.

This means that the algorithm approximates within factor $\frac{3}{2}$.