# Discussion 12

**1.** The *bin packing* problem is as follows. You have an infinite supply of bins, each of which can hold $M$ maximum weight. You also have $n$ objects, each of which has a (possibly distinct) weight $w_i$ (any given $w_i$ is at most $M$). Our goal is to partition the objects into bins, such that no bin holds more than $M$ total weight, and that we use as few bins as possible. This problem in general is **NP**-hard.

Give a 2-approximation to the *bin packing* problem. That is, give an algorithm that will compute a valid partitioning of objects into bins, such that no bin holds more than $M$ weight, and our algorithm uses at most twice as many bins as the optimal solution. Prove that the approximation ratio of your algorithm is two.

Solution:
Place objects in the bin in the order given starting with bin #1. When there is no more room in bin #1 start placing objects in bin #2, and so on.

Claim: This will give us a 2-approximation
Proof: the total weight of objects in bins 1 and 2 in our solution is greater than M, since the reason why we started placing objects in bin #2 was that we ran out of space in bin #1. So, since the total weight is greater than M, the optimal solution needs at least 1 bin to hold these objects (i.e. 1 bin is a bound on how good the optimal solution can be).
This can be said about every pair of bins in our solution. So if our solution ends up with 2k bins, the optimal solution will need at least k bins. Therefore we have 2-approximation.

**2.** Suppose you are given a set of positive integers $A$: $a_1\ a_2\ \ldots\ a_n$ and a positive integer $B$. A subset $S \subseteq A$ is called *feasible* if the sum of the numbers in $S$ does not exceed $B$.

The sum of the numbers in $S$ will be called the *total sum* of $S$. You would like to select a feasible subset $S$ of $A$ whose total sum is as large as possible.

**Example**: If A = {8, 2, 4} and B = 11 then the optimal solution is the subset S = {8, 2}.

Give a linear-time algorithm for this problem that finds a feasible set $S \subseteq A$ whose total sum is at least half as large as the maximum total sum of any feasible set $S' \subseteq A$. Prove that your algorithm achieves this guarantee.
You may assume that each $a_i \leq B$.

Solution:

Starting from a1, place integers into the set S for as long as their total does not exceed B until we reach B/2. If at any time the total weight of the set goes above B/2 we have achieved a .5 approximation (since the optimal solution cannot have a value greater than B). But if at some point in this process, the total has not reach B/2 and the next item in the set (ai) causes the total to exceed B, then we will remove all elements in the set and replace it with ai. ai itself will give us a .5 approximation since it must be greater than B/2.

Above process can of course be carried out in linear time.

**3.** A cargo plane can carry a maximum weight of 100 tons and a maximum volume of 60 cubic meters. There are three materials to be transported, and the cargo company may choose to carry any amount of each, up to the maximum available limits given below.

- Material 1 has density 2 tons/cubic meter, maximum available amount 40 cubic meters, and revenue $1,000 per cubic meter.
- Material 2 has density 1 ton/cubic meter, maximum available amount 30 cubic meters, and revenue $1,200 per cubic meter.
- Material 3 has density 3 tons/cubic meter, maximum available amount 20 cubic meters, and revenue $12,000 per cubic meter.

Write a linear program that optimizes revenue within the constraints. You do not need to solve the linear program.

Let M1, M2, and M3 denote the cubic meters of the three materials we're going to transport.

We want to maximize the profit. So the objective function will be:
Maximize 1000 * M1 + 1200 * M2 + 12000 * M3

Subject to these constraints:

```
M1 + M2 + M3 <= 60              // 60 cubic meters space available
2 * M1 + M2 + 3 * M3 <= 100     // 100 tons weight capacity
0 <= M1 <= 40
0 <= M2 <= 30                   // maximum available for these three
0 <= M3 <= 20
```

**4.** Recall the 0/1 knapsack problem:
Input: n items, where item i has profit $p_i$ and weight $w_i$, and knapsack size is W.
Output: A subset of the items that maximizes profit such that the total weight of the set ≤ W.
You may also assume that all $p_i \geq 0$, and $0 < w_i \leq W$.

We have created the following greedy approximation algorithm for 0/1 knapsack:

Sort items according to decreasing $p_i/w_i$ (breaking ties arbitrarily).
While we can add more items to the knapsack, pick items in this order.

Show that this approximation algorithm has no nonzero constant approximation ratio.
In other words, if the value of the optimal solution is P*, prove that there is no constant $\rho$ ($1 > \rho > 0$), where we can guarantee our greedy algorithm to achieve an approximate solution with total profit $\rho$ P*.

Solution:
Consider an item with size 1 and profit 2, and another with size W and profit W. Our greedy algorithm above will take the first item, while the optimal solution is taking the second item (for W>1). So this algorithm has approximation ratio 2/W which can go arbitrarily close to zero as we increase W.

**5.** There are *n* people and *n* jobs. You are given a cost matrix, C, where $C_{ij}$ represents the cost of assigning person i to do job j. Each applicant *i* has a subset of jobs $S_i$ that he/she is interested in. Each job can only accept one applicant and an applicant can be appointed to only one job. Write an integer linear program to minimize the total cost of assigning all applicants. You can assume that a feasible assignment of people to jobs exists.

Solution:
We are looking for a perfect matching on a bipartite graph with the minimal cost.
Our discrete variables will have the following meanings:

$X_{ij}$ = 1 if person *i* is matched with job *j*
$X_{ij}$ = 0 otherwise

ILP formulation:
Minimize $\sum_{i,j} C_{ij} X_{ij}$ for i=1 to *n*, j=1 to *n*

Subject to:
$X_{ij} \in \{0,1\}$
$\sum_j X_{ij} = 1$ for *i*=1 to *n*
$\sum_i X_{ij} = 1$ for *j*=1 to *n*
$X_{ij} < 1$ if person *i* is not interested in job *j* (i.e. for person *i*, $j \notin S_i$)