Lesson 12. Introduction to Algorithm Design

1 What is an algorithm?

- An **algorithm** is a sequence of computational steps that takes a set of values as **input** and produces a set of values as **output**
- For example:
  - input = a linear program
  - output = an optimal solution to the LP, or a statement that LP is infeasible or unbounded
- Types of algorithms for optimization models:
  - **Exact algorithms** find an optimal solution to the problem, no matter how long it takes
  - **Heuristic algorithms** attempt to find a near-optimal solution quickly
- Why is algorithm design important?

2 The knapsack problem

- You are a thief deciding which precious metals to steal from a vault:

	Metal	Weight (kg)	Total Value
1	Gold	10	100
2	Silver	20	120
3	Bronze	25	200
4	Platinum	5	75

- You have a knapsack that can hold at most 30 kg
- Assume you can take some or all of each metal
- Which items should you take to maximize the value of your theft?
- A linear program:

$$x_i = \underline{\text{fraction}} \text{ of metal } i \text{ taken} \quad \text{for } i \in \{1, 2, 3, 4\}$$

$$\begin{aligned} \max \quad & 100x_1 + 120x_2 + 200x_3 + 75x_4 \\ \text{s.t.} \quad & 10x_1 + 20x_2 + 25x_3 + 5x_4 \leq 30 \\ & 0 \leq x_i \leq 1 \quad \text{for } i \in \{1, 2, 3, 4\} \end{aligned}$$

- Try to come up with the best possible feasible solution you can
- What was your methodology?

3 Some possible algorithms for the knapsack problem

3.1 Enumeration

- Naïve idea: just list all the possible solutions, pick the best one
- One problem: since the decision variables are continuous, there are an infinite number of feasible solutions!
- Suppose we restrict our attention to feasible solutions where $x_i \in \{0,1\}$ for $i \in \{1,2,3,4\}$
- How many different possible feasible solutions are there?

$$\frac{0/1}{x_1} \quad \frac{0/1}{x_2} \quad \frac{0/1}{x_3} \quad \frac{0/1}{x_4}$$

  - For 4 variables, there are at most $\boxed{2^4 = 16}$ 0-1 feasible solutions
  - For n variables, there are at most $\boxed{2^n}$ 0-1 feasible solutions

- The number of possible 0-1 solutions grows very, very fast:

n	5	10	15	20	25	50
2^n	32	1024	32,768	1,048,576	33,554,432	1,125,899,906,842,624

- Even if you could evaluate $2^{30} \approx 1$ billion solutions per second (check feasibility and compute objective value), evaluating all solutions when $n = 50$ would take more than 12 days!
- This enumeration approach is impractical for even relatively small problems

3.2 Best bang for the buck

- Idea: Be greedy and take the metals with the best "bang for the buck": best value-to-weight ratio
- For this particular instance of the knapsack problem:

	Metal	Weight (kg)	Total Value	Value-to-weight ratio
1	Gold	10	100	10
2	Silver	20	120	6
3	Bronze	25	200	8
4	Platinum	5	75	15

- Optimal solution and value:

$x_4 = 1 \Rightarrow 25$ kg left
$x_1 = 1 \Rightarrow 15$ kg left
$x_3 = \frac{3}{5} \Rightarrow 0$ kg left

Optimal solution:
$x_1 = 1, \; x_2 = 0, \; x_3 = \frac{3}{5}, \; x_4 = 1$

Optimal value:
$100(1) + 120(0) + 200(\frac{3}{5}) + 75(1) = 295$

- This "greedy algorithm" turns out to be an exact algorithm for the knapsack problem
- Some issues:
    - How do we know this algorithm always finds an optimal solution?
    - Can this be extended to LPs with more constraints?

4 What should we ask when designing algorithms?

1. Is there an optimal solution? Is there even a feasible solution?
    - e.g. an LP can be unbounded or infeasible – can we detect this quickly?
2. If there is an optimal solution, how do we know if the current solution is one? Can we characterize mathematically what an optimal solution looks like, i.e., can we identify **optimality conditions**?
3. If we are not at an optimal solution, how can we get to a feasible solution better than our current one?
    - This is the fundamental question in algorithm design, and often tied to the characteristics of an optimal solution
4. How do we start an algorithm? At what solution should we begin?
    - Starting at a feasible solution usually makes sense – can we even find one quickly?