Lesson 16. Introduction to Algorithm Design

Algorithms

- 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 optimization algorithms:
  - **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?

The partition problem

- Given a collection of numbers, partition them into 2 groups such that the difference in the sums is as small as possible
- Feasible solution $\Leftrightarrow$ partition into 2 groups
- Objective: minimize the difference of the sums of the 2 groups
- Example: 4, 5, 7, 9, 12, 13, 14, 35, 47
- One feasible solution: $(\{4, 5, 35, 47\}, \{7, 9, 12, 13, 14\})$
  - first sum is 91, second sum is 55, difference is 36
- Can you do better?

- Naïve approach: enumerate and try every feasible solution!
- How many different feasible solutions are there?
  - For 6 objects, there are _____ partitions
  - For n objects, there are _____ partitions

- The number of feasible solutions grows very, very fast:

n	5	10	15	20	25	50
2^{n-1}	16	512	16,384	524,288	16,777,216	562,949,953,421,312

- Even if you could try $2^{30} \approx 1$ billion feasible solutions per second, evaluating all feasible solutions when $n = 50$ would take more than 6 days
- This enumeration approach is impractical for even relatively small problems
- LPs can have infinitely many feasible solutions – naïve enumeration doesn't even make sense!

What to 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 my 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?

Preview: local search / improving search algorithms

- Idea:
    - Start at a feasible solution
    - Repeatedly move to a "close" feasible solution with better objective function value
- The **neighborhood** of a feasible solution is the set of all feasible solutions close to it, where distance is measured by some predefined metric
- By changing our definition of distance, we change the neighborhood of the feasible solution

Example 1. In the partition problem, we can define the neighborhood of a feasible solution as the set of all feasible solutions obtained by putting one of the numbers into the other group. Find the neighborhood of $(\{4, 5, 35, 47\}, \{7, 9, 12, 13, 14\})$. Is there a feasible solution in the neighborhood of this solution that has a better objective function value?