Final Project

Deterministic Models and Optimisation

Consider the classical 0–1 Knapsack Problem and its Integer Programming formulation:

$$\max \quad \sum_{i=1}^{n} p_i x_i \tag{1}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} w_i x_i \leq c \tag{2}$$

$$x_i \in \{0,1\} \quad \forall i \in \{1,\dots,n\}, \tag{3}$$

where we must maximise profits p_i collected packing n items of weights w_i in a knapsack of capacity c. For simplicity, we can assume that we deal with non-negative real values ($p_i, w_i, c \in \mathbb{R}_0^+$ for all $i \in \{1,\dots,n\}$) and thus we are not limited to integer values. We can also assume that the problem is not trivial, i.e., that $\sum_{i=1}^{n} w_i > c$.

Task 1 (2pt). Devise a simple and fast *primal* heuristic for the 0–1 Knapsack Problem. You have complete freedom on how to implement it, but note that it should run almost instantaneously on instances with up to 500 customers. You should not focus on finding the optimal solution within a long runtime, but rather in finding a good solution in a short time. You can assume that you receive the items in the instance file already sorted by density (p_i/w_i).

Denote with N the set of item indices ($N = \{1,\dots,n\}$). A subset $C \subseteq N$ is called a *cover* if $\sum_{i \in C} w_i > c$, that is, the total weight of the items in the cover exceeds the capacity and, therefore, not all $|C|$ items in the cover can be packed. It follows that, given a cover C, inequality

$$\sum_{i \in C} x_i \leq |C| - 1 \tag{4}$$

is valid. Such an inequality is called a "cover inequality".

This inequality is not required for a correct model: (1)–(3) is complete and correct even without these inequalities. However, inequalities (4) can strengthen the linear relaxation of the model.

Task 2 (1pt). Devise a small instance in which the optimal solution of the linear relaxation of model (1)–(3) violates an inequality of type (4).

Next we analyse a separation method for inequalities (4). Given a solution $x_1^*, \ldots, x_n^*$ of the linear relaxation of model (1)–(3), we want to find at least one cover C which yields a violated inequality of type (4), or prove that no such cover exists. Consider the following optimisation problem:

$$\min \quad \sum_{i=1}^{n} (1 - x_i^*) y_i \tag{5}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} w_i y_i > c \tag{6}$$

$$y_i \in \{0, 1\} \quad \forall i \in \{1, \ldots, n\}, \tag{7}$$

which uses binary decision variables $y_i \in \{0, 1\}$ for each $i \in \{1, \ldots, n\}$. Note that x_i^* are *not* decision variables in this problem.

Question 3 (1pt). Is this a valid Integer Programme? Motivate your answer.

The problem defined by (5)–(7) is instrumental in devising a separation routine. In particular, if the objective value of the optimal solution to this problem is < 1, then a cover inequality is violated by solution $x_1^*, \ldots, x_n^*$. Otherwise, the solution satisfies all cover inequalities.

Task 4 (2p). Let $y_1^*, \ldots, y_n^*$ be the optimal solution of (5)–(7). Prove that, if $\sum_{i=1}^{n} (1 - x_i^*) y_i^* < 1$, then there is a violated cover inequality. Explain how to build the corresponding cover C.

To practically solve separation problem (5)–(7), we need one last observation.

Task 5 (1p). Prove that solving problem (5)–(7) is equivalent to solving a 0–1 Knapsack Problem using a strict $<$ capacity inequality (let's call it 01KP-Strict). In particular, given the original problem data and a solution $x_1^*, \ldots, x_n^*$, explain how to build the corresponding instance of 01KP-Strict which solves problem (5)–(7). Show how, in practice, an instance of 01KP-Strict can be turned into an instance of the classical 0–1 Knapsack Problem (the one with the $\leq$ capacity inequality) using a small numerical trick.

How interesting! We have a family of valid inequalities (the cover inequalities) for the 0–1 Knapsack Problem and, to separate them, we can solve... another knapsack problem!

Note that any solution $y_1^*, \ldots, y_n^*$ which is feasible for (5)–(7) and whose objective value is strictly smaller than 1 gives a cover inequality. In other words, we do not need to solve (5)–(7) to optimiality and, therefore, we do not need to solve the corresponding equivalent knapsack problem to optimality. Therefore, now is the time for your heuristic algorithm from *Task 1* to shine.

Task 6 (3pt). Devise and implement a branch-and-cut algorithm for the 0–1 Knapsack Problem. The algorithm should start from the initial formulation (1)–(3) and, every time the solver produces a fractional solution, it should attempt to separate violated cover inequalities. To do so, it will use your knapsack heuristic to solve the 0–1 Knapsack Problem associated to (5)–(7). There are two possible outcomes when you use your heuristic:

1. If it finds a solution corresponding to a violated cover inequality, you will add such cut.
2. Otherwise, the ball is back in the solver's court. Because you have a heuristic solver, you cannot prove that there is no violated cover inequality. But, because cover inequalities are valid but *not* required, this is not an issue. We can just tell the solver to keep exploring the branch-and-bound tree, and you will attempt to separate another cover inequality when the solver finds a new fractional solution.

Hint: Gurobi's preprocessing is quite aggressive and might close many instances before optimisation even starts. To give your B&C model a chance to prove its worth, disable Gurobi's preprocessing as follows:

```
model.Params.Cuts = 0
model.Params.Heuristics = 0
model.Params.Presolve = 0
```