LECTURE 6

Ishaan Lal

February 21, 2025

1 Introduction

Last lecture, we introduced Linear Programming (LP), Integer Linear Programming (ILP) and Mixed-Integer Programming (MIP). Today, we will dive deeper into MIP.

Recall:

- **LP** involved maximizing/minimizing linear objective functions subject to linear inequalities
- **MIP** is identical to LP, with the extra constraint that some variables must be integers.

LP is poly-time solvable. MIP is NP-Complete.

2 Mixed Integer Program

Often, we will generalize ILP to **MIP (Mixed Integer Programming)**. With MIP, some variables may be constrained to integers, and some may not. Still, all objective functions and constraints are linear. We'll just talk about MIP since it encapsulates ILP.

2.1 Capital Budgeting Problem

To warm up our skills for MIP, let's consider the following problem:

- There are n possible investments, each with some associated utility/value v_i
- There are m resources, each with amount a_j
- Investment i costs c_{ij} units of resource j
- We want to make investments in a way so as to **maximize value**.

As we did with LP problems, we first need to choose our variables. Again, we ask ourselves: *What has the ability to change?*

The answer to this question may not be immediate, but it is: whether or not we choose each of the investments. Formally, our variables are x_i where:

$$x_i = \begin{cases} 1 & \text{we pick the } i^{th} \text{ investment} \\ 0 & \text{else} \end{cases} \qquad \forall i \in [1..n]$$

This idea of using "indicator" variables is **very** common and very useful in MIP and modeling in general.

Our objective function is to maximize value. This is simply the sum of all investments chosen:

$$\text{maximize} \quad \sum_i v_i x_i$$

Note that the investments that we do not choose do not contribute to the sum, because they would have $x_i = 0$

Lastly, we consider the only constraint: for each resource, the amount of it that we use does not exceed the amount available:

$$\sum_i c_{ij} \cdot x_i \leq a_j \quad \text{for each resource } j$$

2.2 Some Additional Constraints

Suppose we added an additional constraint: that we *need* to invest in investment i in order for us to be able to invest in investment j. How can we encode this as a constraint?

Let's consider if $x_i = 0$ – we do not invest in i. Then, we MUST have $x_j = 0$.
However, if $x_i = 1$, then we can have $x_j = 0$ or we could have $x_j = 1$.
The simplest way of expressing this situation is:

$$x_i \geq x_j$$

Suppose we had a constraint that investments i, j, k are conflicting. That is, we can only have at most one of these three investments. Then naturally, we would add the constraint:

$$x_i + x_j + x_k \leq 1$$

3 A Step Towards Non-Linearity

3.1 Pointwise Discontinuity

We seem to have a good grasp of how to solve LP-related problems when the objective function is indeed Linear. But we may choose to ask: *What do we do when the objective is discontinuous and linear?* or perhaps *What do we do when the objective is piecewise linear?* These are both valid questions, and may arise in many real world applications.

Consider the following setting:

Problem Setting

You are the proud owner of your business called *Quackulus*, where you specialize in creating novel rubber ducks. Suppose it costs \$10 to produce a single duck. There is also a fixed setup cost of \$250 if you choose to produce any units. Additionally, you can only create a maximum of 1000 ducks.

You are aiming to minimize your cost of production subject to some unknown linear constraints.

Given the setting, naturally, our objective function would be formalized as:

$$\text{minimize} \quad 250 + 10n$$

where $n \in \mathbb{N}$ represents the number of ducks we produce. However, when $n = 0$, our objective function evaluates to 250, when it should be 0. Thus, our correct function is:

$$\begin{cases} 0, & n = 0 \\ 250 + 10n & n > 0 \end{cases}$$

This is **NOT** a MIP because our objective function is not linear in the domain. There is a discontinuity at $n = 0$. MIP requires our objective function to be linear. How do we fix this?

- **Idea 1:** Add a constraint of $n > 0$. This doesn't work, because we definitely would want to consider the option of producing $n = 0$ ducks.
- **Idea 2:** Still add the constraint of $n > 0$, then solve the MIP, which would indicate the "best solution given that we produce a duck", and later compare that with producing 0 ducks, and choose the better option.

  This can work, but think about what happens if we have a more complex discontinuous linear function. Things could become impractical. Also consider what would happen if the discontinuity was somewhere in the middle of the domain (instead of at one of the bounds).

Solution

Notice that the number of ducks we can produce is at most 1000. So if we choose to produce ducks, then $n \leq 1000$, otherwise, $n = 0$. To formalize this, we will introduce an indicator variable z whereby:

$$z = \begin{cases} 1 & \text{we make ducks} \\ 0 & \text{we do not make ducks} \end{cases}$$

We can now reformulate our objective as a linear function:

$$\begin{aligned} \text{minimize} \quad & 250 \cdot z + 10 \cdot n \\ \text{subject to} \quad & n \leq 1000 \cdot z \\ & n \geq 0 \\ & z \in \{0, 1\} \\ & \text{other constraints} \end{aligned}$$

3.2 Piecewise Linear Functions

We now consider a different scenario, where our objective function is best described as a piecewise function.

Problem Setting

Quackulus has undergone some improvements where the cost of production has changed. Now, there is no fixed set-up cost. However, the cost per unit depends on the number of units produced.

The first 400 ducks you produce will cost \$5 each to produce. The next 200 ducks will cost only \$2 each. And the next 400 ducks will cost only \$3 each.

For example, if you choose to create 500 ducks, it will cost you:

$$400 \cdot \$5 + 100 \cdot \$2 = \$2200$$

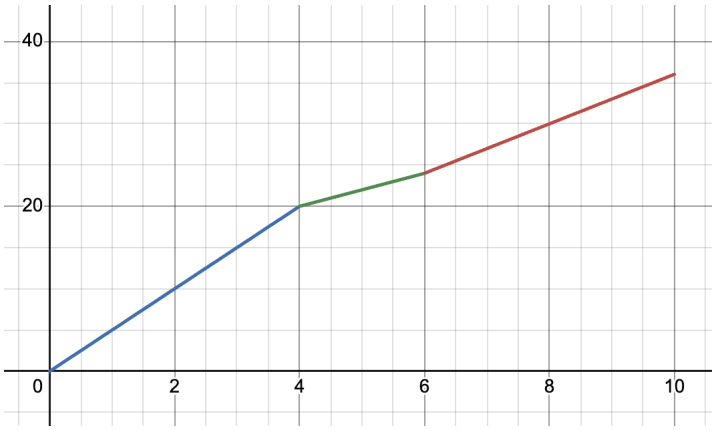
And if you choose to create 900 ducks, it will cost you:

$$400 \cdot \$5 + 200 \cdot \$2 + 300 \cdot \$3 = \$3300$$

Under this setting, we can write the objective function as a piecewise linear function:

$$\begin{cases} 5 \cdot n & 0 \leq n \leq 400 \\ 5 \cdot 400 + 2 \cdot (n - 400) & 401 \leq n \leq 600 \\ 5 \cdot 400 + 2 \cdot 200 + 3 \cdot (n - 600) & 601 \leq n \leq 1000 \end{cases} = \begin{cases} 5n & 0 \leq n \leq 400 \\ 2n + 1200 & 401 \leq n \leq 600 \\ 3n + 600 & 601 \leq n \leq 1000 \end{cases}$$

Graphically, our objective function looks as follows, where the x-axis represents the number of hundreds of ducks created, and the y-axis represents the cost in hundreds of dollars:



Similar to how we handled discontinuity, we'll leverage the property of indicators. Here, indicators would be helpful to indicate which segment of the curve we are on.

Consider variables $\delta_1, \delta_2, \delta_3$ where $n = \delta_1 + \delta_2 + \delta_3$ and where $0 \leq \delta_1 \leq 400$, $0 \leq \delta_2 \leq 200$, $0 \leq \delta_3 \leq 400$.

δ_i represents how far into the ith piece (segment) of the piecewise function we are. For example if $n = 500$ ducks, then $\delta_1 = 400$, $\delta_2 = 100$, $\delta_3 = 0$, indicating that we have passed the first segment (since δ_1 is at its max) and are 100 units in to the second segment. With this formulation, it is easy to see that:

$$\text{Cost} = \$5 \cdot \delta_1 + \$2 \cdot \delta_2 + \$3 \cdot \delta_3$$

Our objective function is now linear, but we must codify this idea of the definition of δ_i into our constraints. We do this as follows: Note that δ_2 is contingent on δ_1, as $\delta_2 > 0$ only if $\delta_1 = 400$ (at its max). Similar for δ_3 being contingent on δ_2.

Let i_1 be an indicator that is 1 if δ_1 has reached its upper limit. Let i_2 be an indicator that is 1 if δ_2 has reached its upper limit.

Then, if $i_1 = 0$ we must have $0 \leq \delta_1 \leq 400$, but if $i_1 = 1$, it must be that $\delta_1 = 400$, or equivalently, $400 \leq \delta_1 \leq 400$. We can combine this into a single constraint as follows:

$$i_1 \cdot 400 \leq \delta_1 \leq 400$$

Similarly, if $i_2 = 0$, then $0 \leq \delta_2 \leq 200$, and if $i_2 = 1$, then $200 \leq \delta_2 \leq 200$. As a single constraint:

$$i_2 \cdot 200 \leq \delta_2 \leq 200$$

But recall that $i_2 = 1$ only if $i_1 = 1$. How do we incorporate this? One small change is needed:

$$i_2 \cdot 200 \leq \delta_2 \leq 200 \cdot i_1$$

As an exercise, take some time to convince yourself why this constraint enforces the implication we had.

Following the pattern from above, we can only have $\delta_3 \geq 0$ if $i_2 = 1$. Then, our last constraint is:

$$0 \leq \delta_3 \leq 400 \cdot i_2$$

Our fully formalized MIP is:

$$\begin{aligned} \text{minimize} \quad & 5\delta_1 + 2\delta_2 + 3\delta_3 \\ \text{subject to} \quad & i_1 \cdot 400 \leq \delta_1 \leq 400 \\ & i_2 \cdot 200 \leq \delta_2 \leq i_1 \cdot 200 \\ & 0 \leq \delta_3 \leq 400 \cdot i_2 \\ & i_1, i_2 \in \{0, 1\} \end{aligned}$$

3.3 Towards Continuity

But of course, the most fascinating question is what happens when our objective function is simply continuous and non-linear? While we will not go very in-depth into this question, a simple idea is as follows: We can approach the optimal solution by approximating the continuous curve of the objective function as a piecewise linear function (with many, but finite, number of segments). From here, use the approach outlined in the previous section to get an approximation of the solution.

4 How MIP Solvers Work

We'll now take some time to try to develop an algorithm to solve MIP. Remember, MIP is an NP-complete problem. MIP solvers use a technique called "Branch and Bound", which we will get to shortly. First, we need to build our way up.

For simplicity, we will assume that the variables involved in our MIP have an upper and lower bound:

$$lb(x) \leq x \leq ub(x)$$

Keep in mind, in most applications of MIP, some bound would be present (for example, it is almost certain that I will create less than 10^{15} ducks).

Additionally, we will assume that we are solving a "maximization" problem. "Minimization" problems work similarly, just some things will be mirrored.

As per usual, we can treat our search space as a tree, where from one node, we traverse down a branch to a lower level when we assign a new variable. At the leaves, all variables have been assigned, and each leaf corresponds to a different assignment.

4.1 Naive Branching

Let P represent the objective function of our MIP. A first approach to solving MIP would be as follows:

- Choose a variable and split its viable domain in half
- Generate subproblems on each "half"
- Solve the subproblems recursively
- Choose whichever subproblem has the higher (or lower, if minimizing) objective value, and discard infeasible solutions.

In terms of code, our initial algorithm is given as the following. Note that "lb" represents "lower bound" and "ub" represents "upper bound".

```
naive(P):
    if lb = ub for all vars:
        if P violates a constraint:
            return INFEASIBLE
        return objective_value(P)
    let x be a variable with lb(x) < ub(x)
    let m = ⌊(lb(x) + ub(x))/2⌋
    return max{naive(P | x ≤ m), naive(P | x ≥ m)}
```

With any algorithm, we tend to be curious about its runtime. What is the runtime of the above code? It may be a bit difficult to compute. But a critical observation is that **this algorithm will only terminate for pure integer programs**.

But here is an even more pressing issue: we essentially have to explore our entire search tree. That is, in our recursion statement, we have to check both branches. This is noticeably different from our naive backtracking algorithm for DPLL, because with that algorithm, we could stop immediately when we found a solution (since it was a decision problem). Here, we explore everything. In the next lecture, we will explore some some ways to speed up our algorithm, akin to how the aspects of DPLL (unit propagation, etc.) were an improved version of naive backtracking.