CIS1921

Lecture 6:

More Mixed-Integer Programming

The Power of Indicators

Logistics

- **Homework 2:** Due on Monday!
- **Homework 3:** Kidney Exchange Program
  - Will be released soon
  - Use MIP to build a model that saves lives IRL!

Recap: LP and MIP

- **Linear programming:** maximize/minimize linear objective subject to linear (in)equalities
- **Mixed-integer programming:** same as linear programming, but some variables can take on integer values only
    - NP-complete!

Mixed-Integer Programming

- **Mixed-integer program (MIP):** some variables may be constrained to be integers, and some may not
- Objectives & constraints are still linear!
- We'll just talk about MIP, since it generalizes IP

MIP in OR-Tools

- Nearly the same as LP! Only differences:
- COIN-OR's **B**ranch-and-**Cut** solver
  - COIN-OR: Computational Optimization Infrastructure for Operations Research

```
from ortools.linear_solver.pywraplp import Solver
model = Solver('my_MIP_model', Solver.CBC_MIXED_INTEGER_PROGRAMMING)
```

- Declaring fractional or integer variables

```
x = model.NumVar(0, Solver.Infinity(), 'x')
n = model.IntVar(0, Solver.Infinity(), 'n')
b = model.BoolVar('b')
```

Capital-Budgeting Problem

- Common MIP modeling problem
- We have n possible investments, each with value v_i
- We have m resources, each with amount a_j
- Investment i costs c_{ij} units of resource j
- Want to maximize value

Capital-Budgeting Problem

- x_i is 0/1 variable indicating if we pick i^{th} investment
  - 0/1 variables are **very** common and useful in modeling

$$\begin{array}{lll} \max & \sum_i v_i x_i & \\ \text{s.t.} & \sum_i c_{ij} x_i \leq a_j & \text{for each } j \end{array}$$

"Variables are the quantities that the solver will give values to. Define your variables in a granular way so when the solver gives values to the variables, you either immediately solve your problem, or can easily derive the solution to your problem."

Capital-Budgeting Problem

- What if we need to invest in i in order to invest in j?

$$x_i \geq x_j$$

- What if i, j, k are conflicting investments?

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

Modeling Fixed Costs

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.

A First Attempt

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.

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

Fails when n = 0

A Piecewise Definition

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.

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

Some Observations

- This is NOT a MIP because Objective Function is not linear in the domain. There is a discontinuity at n=0.
- **Idea 1:** Add a constraint of n > 0
  - What is wrong with this?

- **Idea 2:** Add a constraint of n > 0, and later compare the objective value to it when we set n = 0.
  - What is not great about this?

Indicators for Constraints

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:

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

Indicators for Constraints

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:

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

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

Modeling Piecewise Linear

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:

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

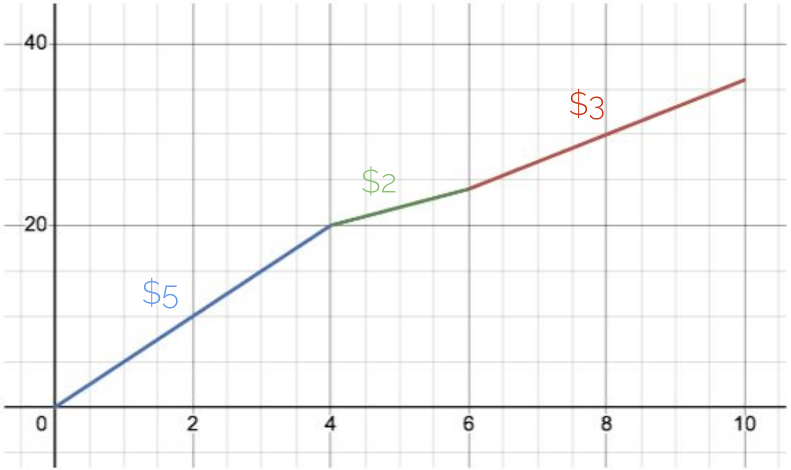
Modeling Piecewise Linear

What does the objective function look like?

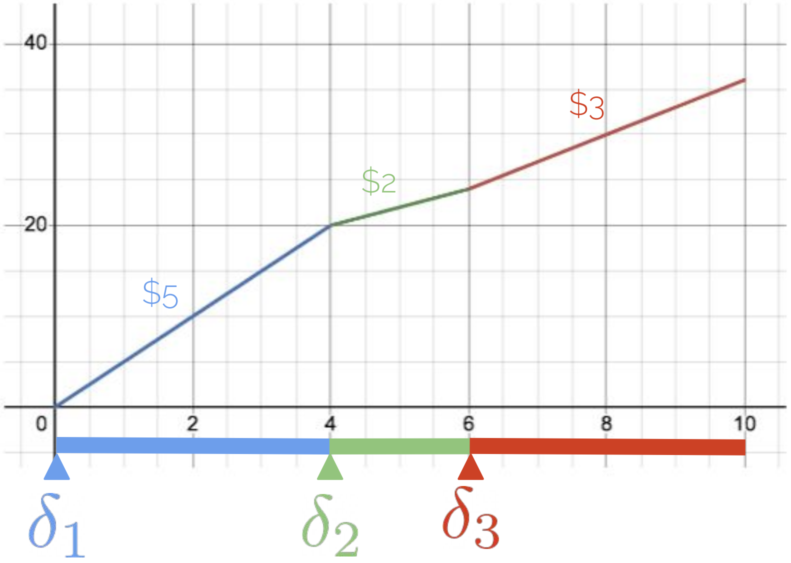
$$\begin{cases} 5 \cdot n & 0 \leq n \leq 400 \\ 5 \cdot 400 + 2 \cdot (n - 400) & 401 \leq n \leq 600 \\ 5 \cdot 400 + 2 \cdot 200 + 3 \cdot (n - 600) & 601 \leq n \leq 1000 \end{cases} = \begin{cases} 5n & 0 \leq n \leq 400 \\ 2n + 1200 & 401 \leq n \leq 600 \\ 3n + 600 & 601 \leq n \leq 1000 \end{cases}$$

Modeling Piecewise Linear

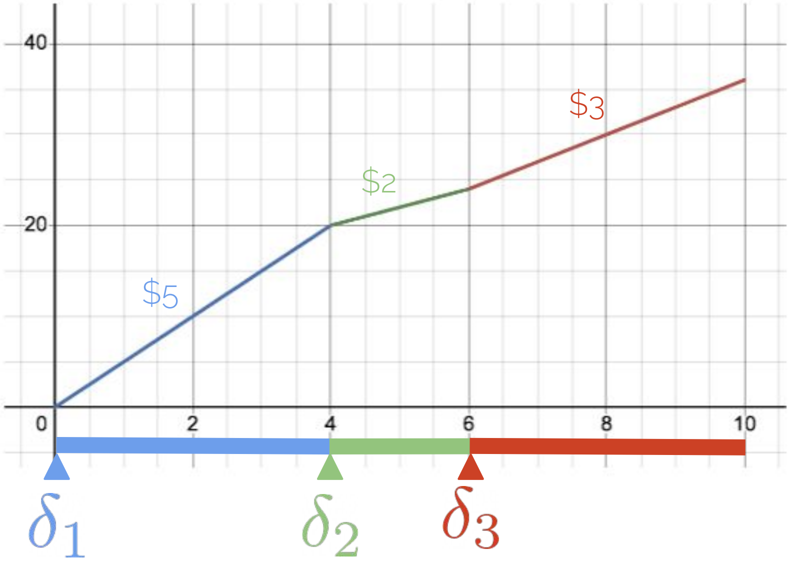
What does the objective function look like?



Modeling Piecewise Linear



Modeling Piecewise Linear



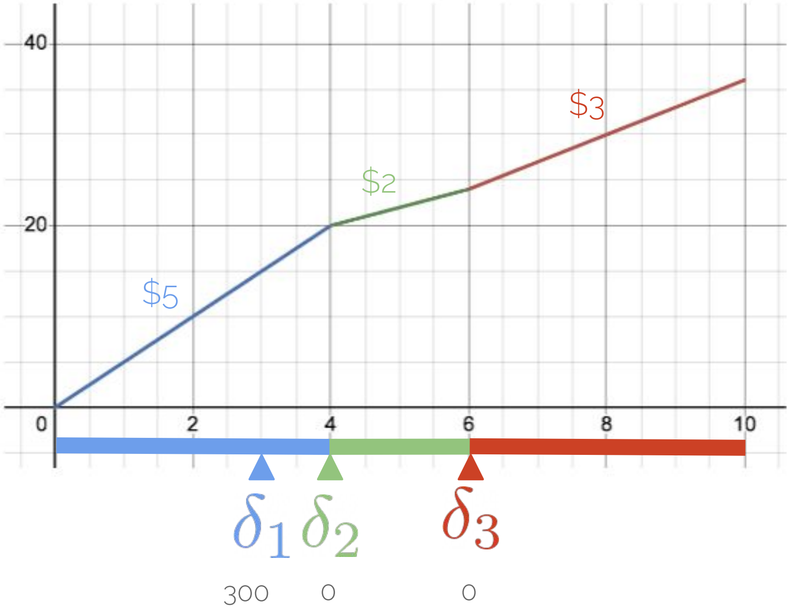
$$n = \delta_1 + \delta_2 + \delta_3$$

$$0 \leq \delta_1 \leq 400$$

$$0 \leq \delta_2 \leq 200$$

$$0 \leq \delta_3 \leq 400$$

Modeling Piecewise Linear



$$n = \delta_1 + \delta_2 + \delta_3$$

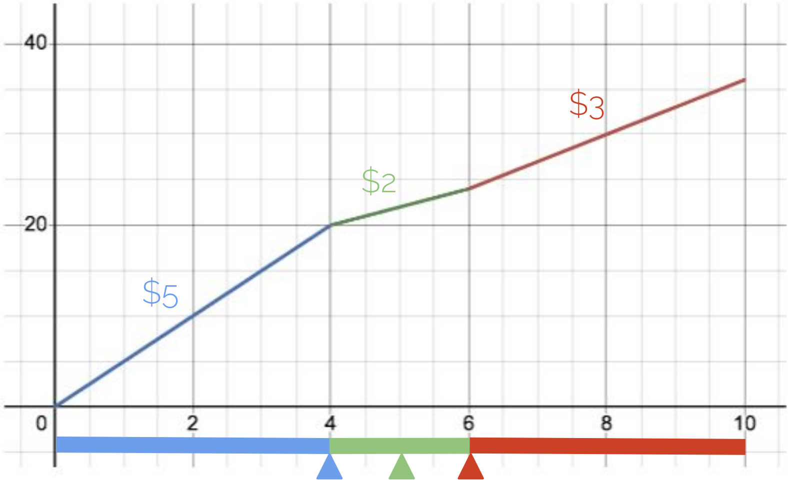
$$0 \leq \delta_1 \leq 400$$

$$0 \leq \delta_2 \leq 200$$

$$0 \leq \delta_3 \leq 400$$

n = 300

Modeling Piecewise Linear



δ_1 δ_2 δ_3

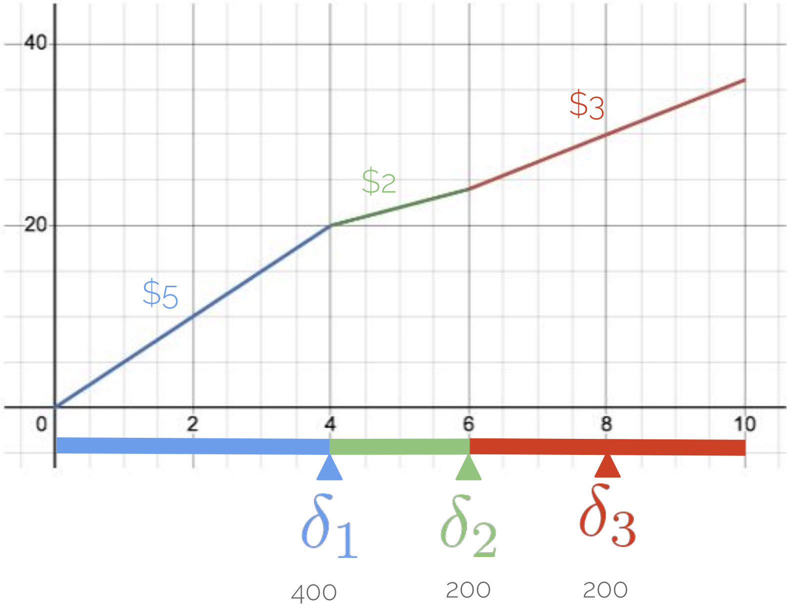
400 100 0

$$n = \delta_1 + \delta_2 + \delta_3$$

$$0 \leq \delta_1 \leq 400$$
$$0 \leq \delta_2 \leq 200$$
$$0 \leq \delta_3 \leq 400$$

n = 500

Modeling Piecewise Linear



$$n = \delta_1 + \delta_2 + \delta_3$$

$$0 \le \delta_1 \le 400$$

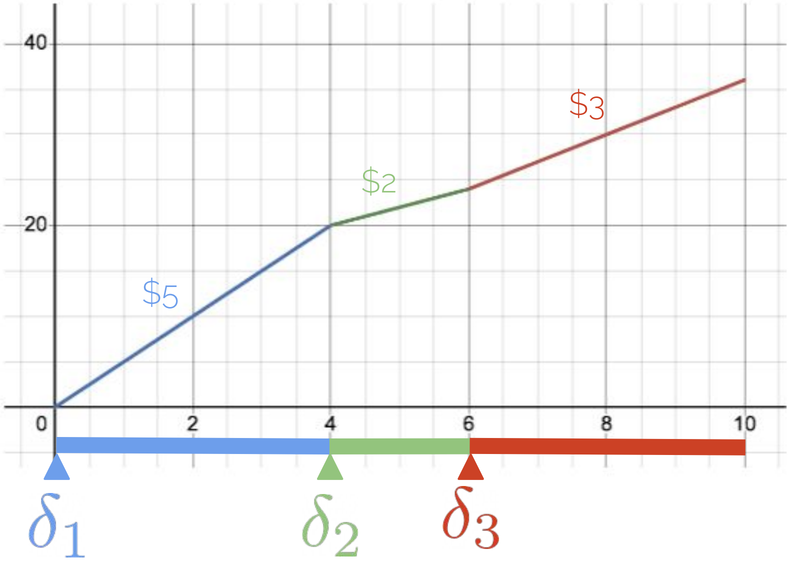
$$0 \le \delta_2 \le 200$$

$$0 \le \delta_3 \le 400$$

n = 800

Modeling Piecewise Linear

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



$$n = \delta_1 + \delta_2 + \delta_3$$

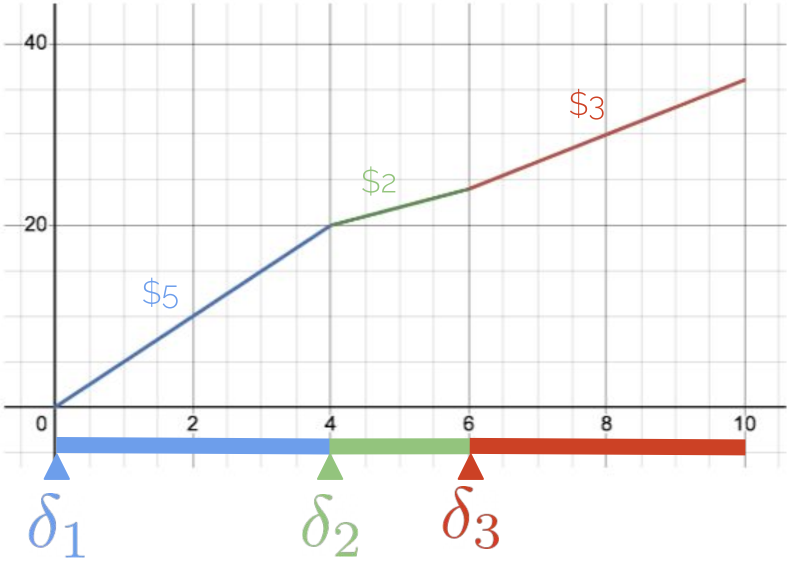
$$0 \leq \delta_1 \leq 400$$

$$0 \leq \delta_2 \leq 200$$

$$0 \leq \delta_3 \leq 400$$

Modeling Piecewise Linear

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



$$n = \delta_1 + \delta_2 + \delta_3$$

$$\begin{cases} 0 \le \delta_1 \le 400 \\ 0 \le \delta_2 \le 200 \\ 0 \le \delta_3 \le 400 \end{cases}$$

These constraints are not enough! What is missing?

Adding Constraints

- δ_2 can only be >= 0 if δ_1 is at its maximum.
- Similarly, δ_3 can only be >= 0 if δ_2 is at its maximum.

Adding Constraints

- δ_2 can only be >= 0 if δ_1 is at its maximum.
- Similarly, δ_3 can only be >= 0 if δ_2 is at its maximum.

- Introduce indicator i_1 which equals 1 if δ_1 is at its maximum
- Introduce indicator i_2 which equals 1 if δ_2 is at its maximum

Adding Constraints

- δ_2 can only be >= 0 if δ_1 is at its maximum.
- Similarly, δ_3 can only be >= 0 if δ_2 is at its maximum.

- Introduce indicator i_1 which equals 1 if δ_1 is at its maximum
- Introduce indicator i_2 which equals 1 if δ_2 is at its maximum

  - If $i_1 = 0$, then $0 <= \delta_1 <= 400$

  - If $i_1 = 1$, then $400 <= \delta_1 <= 400$

Adding Constraints

- δ_2 can only be >= 0 if δ_1 is at its maximum.
- Similarly, δ_3 can only be >= 0 if δ_2 is at its maximum.

- Introduce indicator i_1 which equals 1 if δ_1 is at its maximum
- Introduce indicator i_2 which equals 1 if δ_2 is at its maximum

- If $i_1 = 0$, then $0 <= \delta_1 <= 400$
- If $i_1 = 1$, then $400 <= \delta_1 <= 400$

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

Adding Constraints

- δ_2 can only be >= 0 if δ_1 is at its maximum.
- Similarly, δ_3 can only be >= 0 if δ_2 is at its maximum.

- Introduce indicator i_1 which equals 1 if δ_1 is at its maximum
- Introduce indicator i_2 which equals 1 if δ_2 is at its maximum

  - If $i_2 = 0$, then $0 <= \delta_2 <= 200$

  - If $i_2 = 1$, then $200 <= \delta_1 <= 200$

Adding Constraints

- δ_2 can only be >= 0 if δ_1 is at its maximum.
- Similarly, δ_3 can only be >= 0 if δ_2 is at its maximum.

- Introduce indicator i_1 which equals 1 if δ_1 is at its maximum
- Introduce indicator i_2 which equals 1 if δ_2 is at its maximum

  - If i_2 = 0, then 0 <= δ_2 <= 200
  - If i_2 = 1, then 200 <= δ_1 <= 200

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

Adding Constraints

- δ_2 can only be >= 0 if δ_1 is at its maximum.
- Similarly, δ_3 can only be >= 0 if δ_2 is at its maximum.

- Introduce indicator i_1 which equals 1 if δ_1 is at its maximum
- Introduce indicator i_2 which equals 1 if δ_2 is at its maximum

- If $i_2 = 0$, then $0 <= \delta_2 <= 200$
- If $i_2 = 1$, then $200 <= \delta_1 <= 200$

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

BUT WAIT! $i_2 = 1$ only if $i_1 = 1$

Adding Constraints

- δ_2 can only be >= 0 if δ_1 is at its maximum.
- Similarly, δ_3 can only be >= 0 if δ_2 is at its maximum.

- Introduce indicator i_1 which equals 1 if δ_1 is at its maximum
- Introduce indicator i_2 which equals 1 if δ_2 is at its maximum

- If i_2 = 0, then 0 <= δ_2 <= 200
- If i_2 = 1, then 200 <= δ_1 <= 200

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

BUT WAIT! i_2 = 1 only if i_1 = 1

Moreover, if i_1 = 0, then δ_2 = 0

Adding Constraints

- δ_2 can only be >= 0 if δ_1 is at its maximum.
- Similarly, δ_3 can only be >= 0 if δ_2 is at its maximum.

- Introduce indicator i_1 which equals 1 if δ_1 is at its maximum
- Introduce indicator i_2 which equals 1 if δ_2 is at its maximum

- If i_2 = 0, then 0 <= δ_2 <= 200
- If i_2 = 1, then 200 <= δ_1 <= 200

$$\cancel{i_2 \cdot 200 \leq \delta_2 \leq 200}$$

BUT WAIT! i_2 = 1 only if i_1 = 1

Moreover, if i_1 = 0, then δ_2 = 0

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

Adding Constraints

- δ_2 can only be >= 0 if δ_1 is at its maximum.
- Similarly, δ_3 can only be >= 0 if δ_2 is at its maximum.

- Introduce indicator i_1 which equals 1 if δ_1 is at its maximum
- Introduce indicator i_2 which equals 1 if δ_2 is at its maximum

- δ_3 must be 0 if i_2 = 0. Otherwise, it can be any value in its range

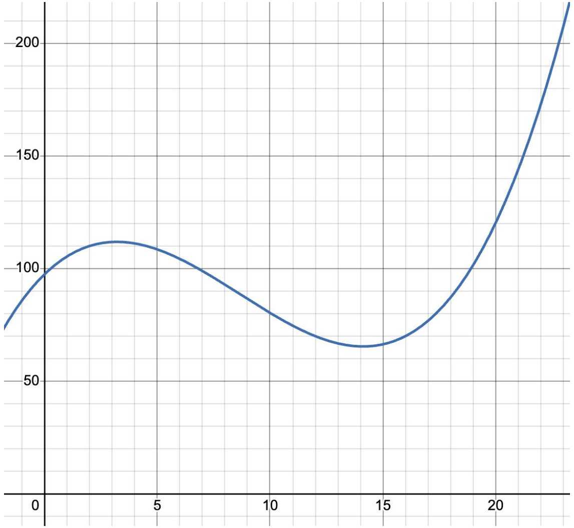
$$0 \leq \delta_3 \leq 400 \cdot i_2$$

Full MIP

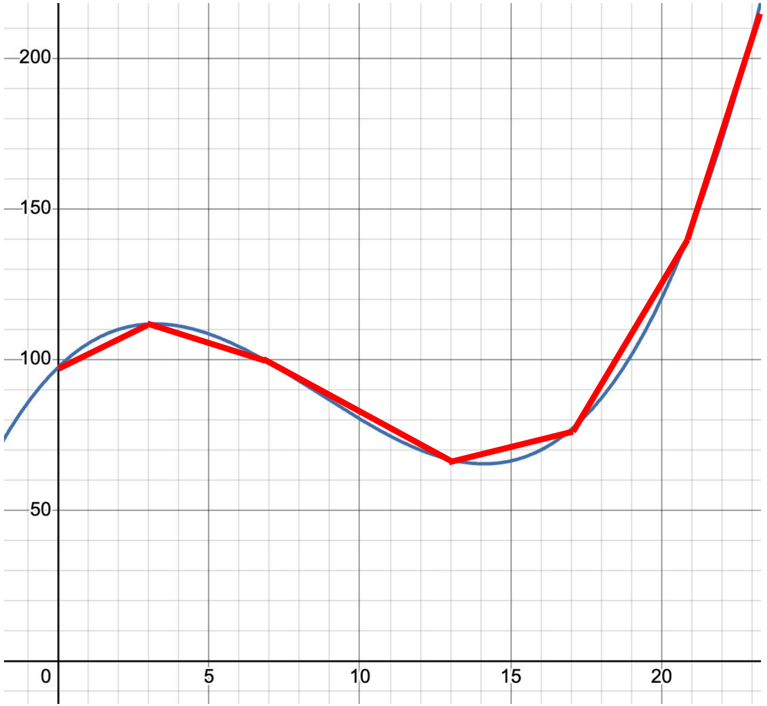
$$
\begin{aligned}
\text{minimize} \quad & 5\delta_1 + 2\delta_2 + 3\delta_3 \\
\text{subject to} \quad & i_1 \cdot 400 \leq \delta_1 \leq 400 \\
& i_2 \cdot 200 \leq \delta_2 \leq i_1 \cdot 200 \\
& 0 \leq \delta_3 \leq 400 \cdot i_2 \\
& i_1, i_2 \in \{0, 1\}
\end{aligned}
$$

Towards Continuity

- What if we choose to move away from a linear objective function altogether?
    - What do we do if our objective function is a curve?



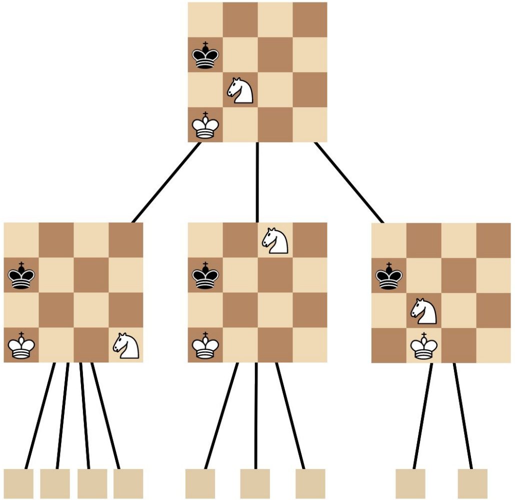
Towards Continuity



Approximate it as a piecewise linear function!

How do MIP solvers work?

- Most fundamental technique: **branch and bound**
  - Chess engines work using branch and bound too (“alpha-beta pruning”)
- For simplicity, let’s assume that all integer variables have lower and upper bounds
  - $\mathrm{lb}(x) \leq x \leq \mathrm{ub}(x)$



Naive Branching

- Want to solve MIP P where integer variables are bounded
- What's a first step for tree traversal of the search space?
- **Idea:** split the domain of a variable in half
  - Generates subproblems which can be solved recursively
- Pick whichever subproblem has the higher objective value, and discard infeasible solutions

Naive Branching (Pseudocode)

```
# find the optimal objective value for P

naive(P):
    if lb = ub for all vars:
        if P violates a constraint:
            return INFEASIBLE (-inf)
        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)}
```

How bad is Naive Branching?

- Does naive branching even terminate?
  - Only for pure integer programs!
- Which assignments does the algorithm discard or visit?
  - Need to evaluate both branches -- visits all feasible solutions!
- Basically the same as brute force
- Runtime scales with size of search space