CIS1921

Lecture 6: More Mixed-Integer Programming

Logistics

- **Homework 3:** Kidney Exchange Program
  - Will be released soon
  - Use MIP to build a model that saves lives IRL!
- Cindy's OH to be changed...

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!

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:

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

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

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:

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

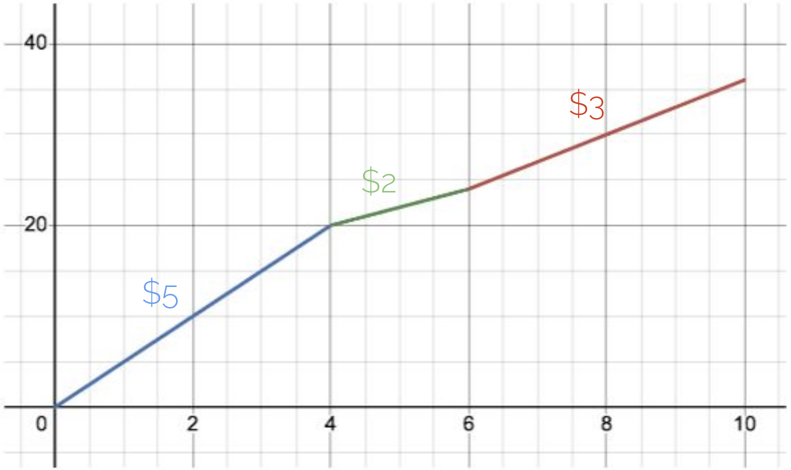
Modeling Piecewise Linear

What does the objective function look like?

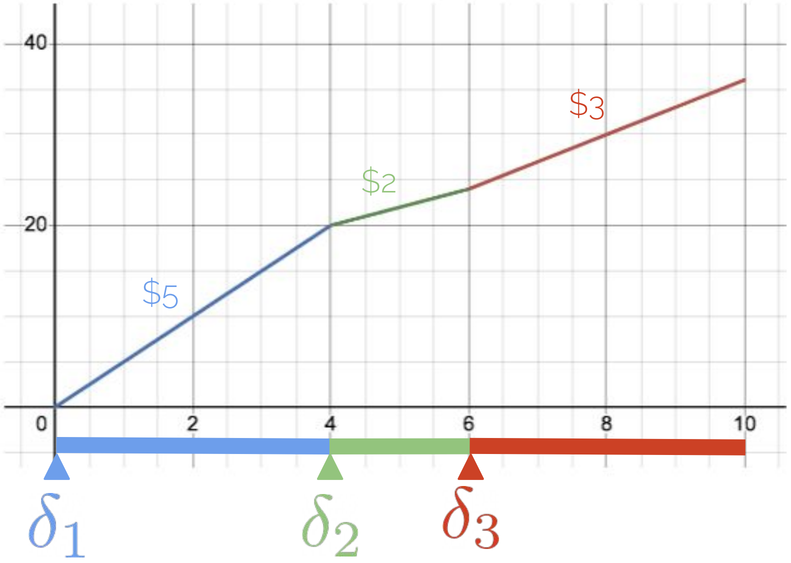
$$\begin{cases} 5 \cdot n & 0 \leq n \leq 400 \\ 5 \cdot 400 + 2 \cdot (n - 400) & 401 \leq n \leq 600 \\ 5 \cdot 400 + 2 \cdot 200 + 3 \cdot (n - 600) & 601 \leq n \leq 1000 \end{cases} = \begin{cases} 5n & 0 \leq n \leq 400 \\ 2n + 1200 & 401 \leq n \leq 600 \\ 3n + 600 & 601 \leq n \leq 1000 \end{cases}$$

Modeling Piecewise Linear

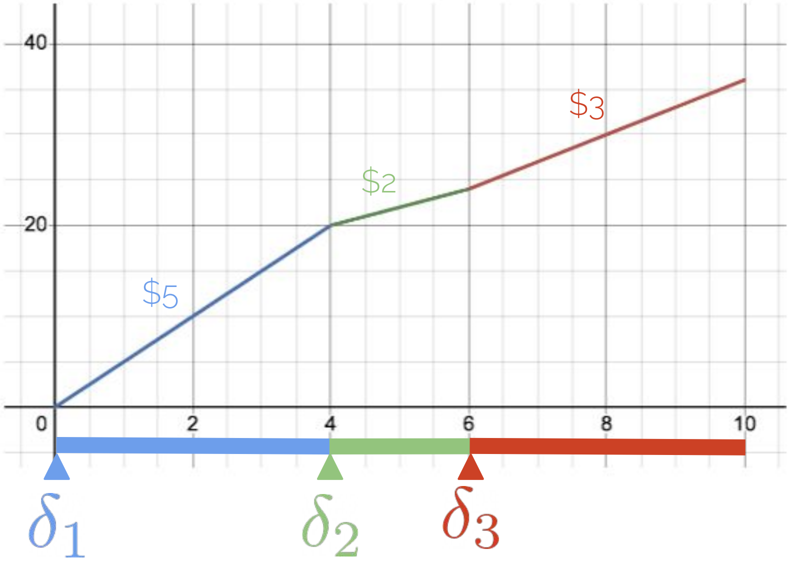
What does the objective function look like?



Modeling Piecewise Linear



Modeling Piecewise Linear



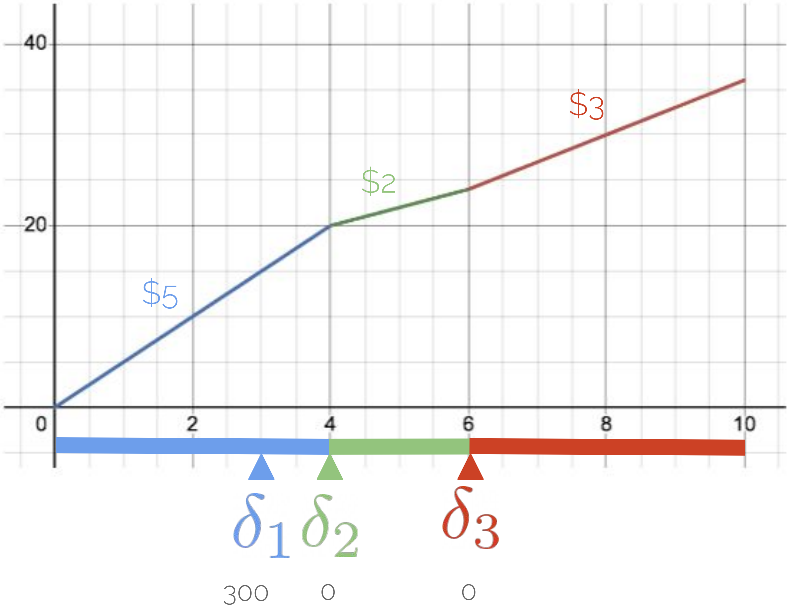
$$n = \delta_1 + \delta_2 + \delta_3$$

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

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

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

Modeling Piecewise Linear



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

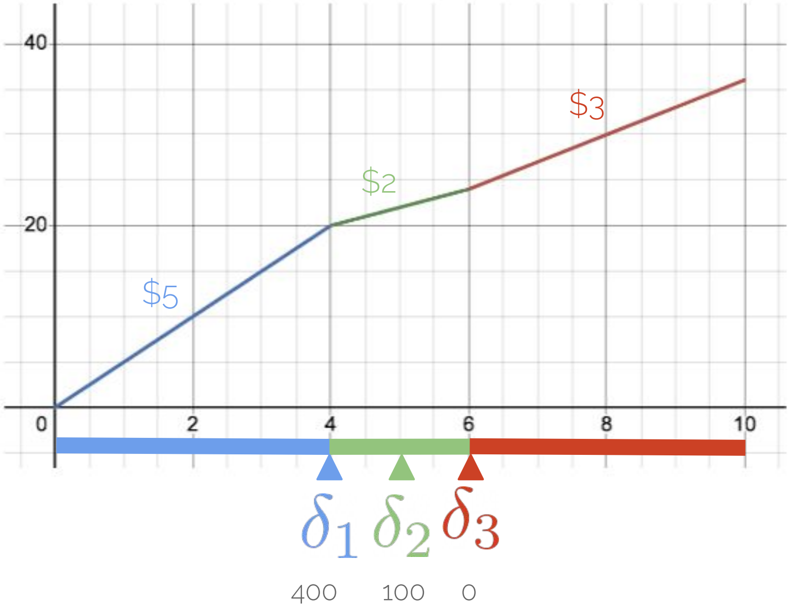
$$0 \leq \delta_1 \leq 400$$

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

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

n = 300

Modeling Piecewise Linear



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

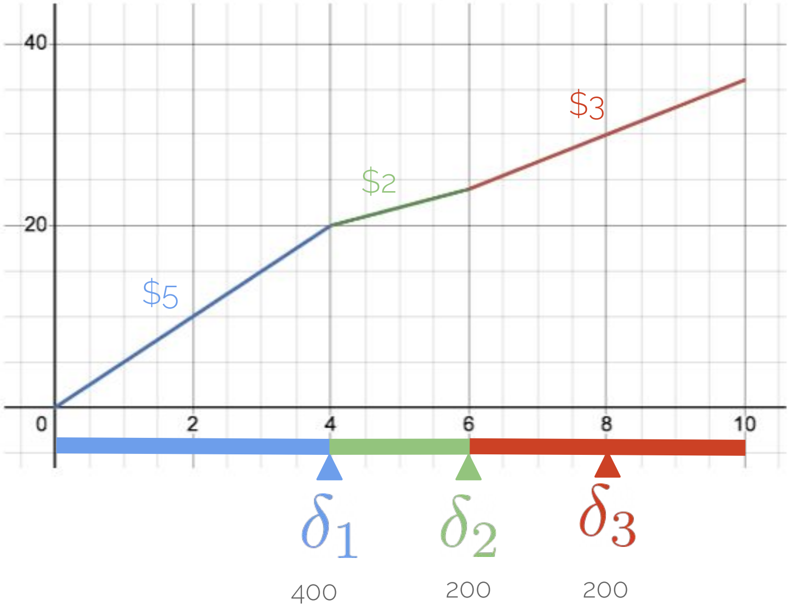
$$0 \leq \delta_1 \leq 400$$

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

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

n = 500

Modeling Piecewise Linear



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

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

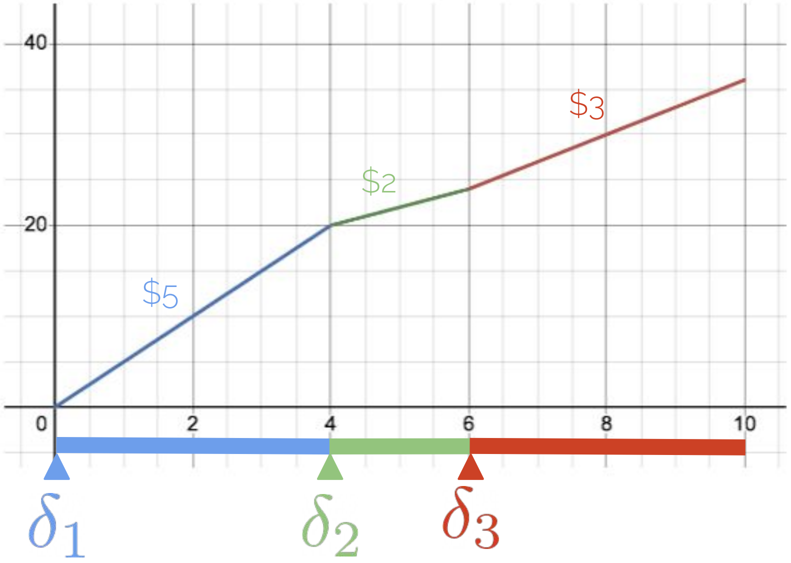
$$0 \leq \delta_2 \leq 200$$

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

n = 800

Modeling Piecewise Linear

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



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

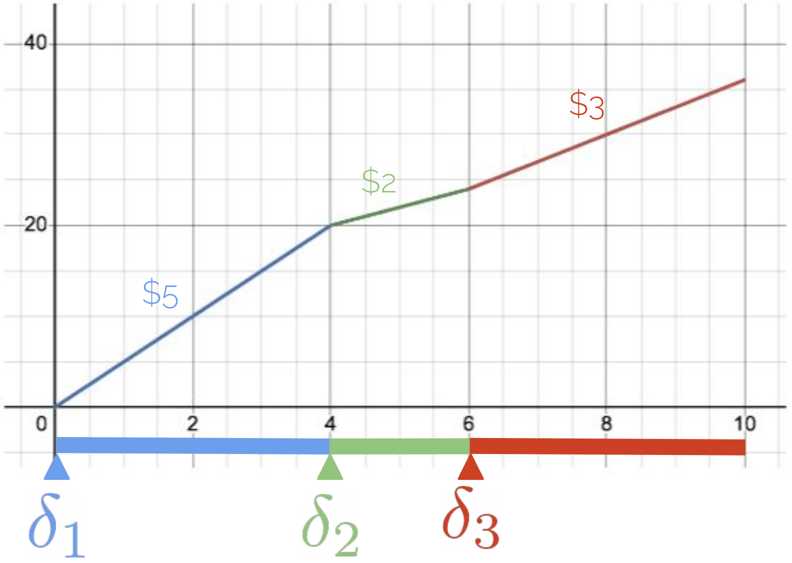
$$0 \leq \delta_1 \leq 400$$

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

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

Modeling Piecewise Linear

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



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

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

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

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

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 <= 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

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 $i_1 = 1$

Moreover, if $i_1 = 0$, then $\delta_2 = 0$

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

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

BUT WAIT! i_2 = 1 only if i_1 = 1

Moreover, if i_1 = 0, then δ_2 = 0

$$i_2 \cdot 200 \le \delta_2 \le 200 \cdot i_1$$

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

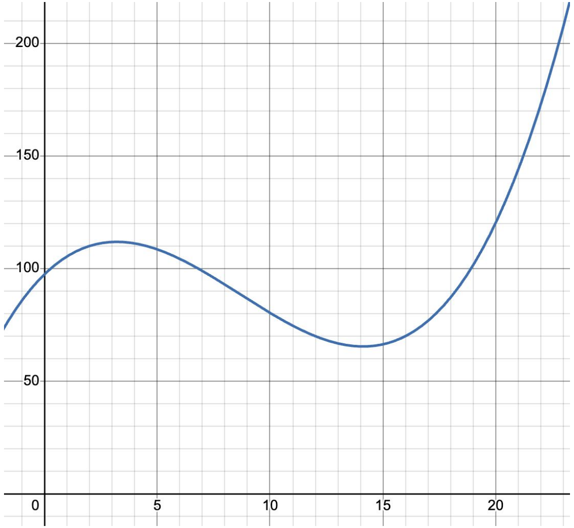
$$0 \leq \delta_3 \leq 400 \cdot i_2$$

Full MIP

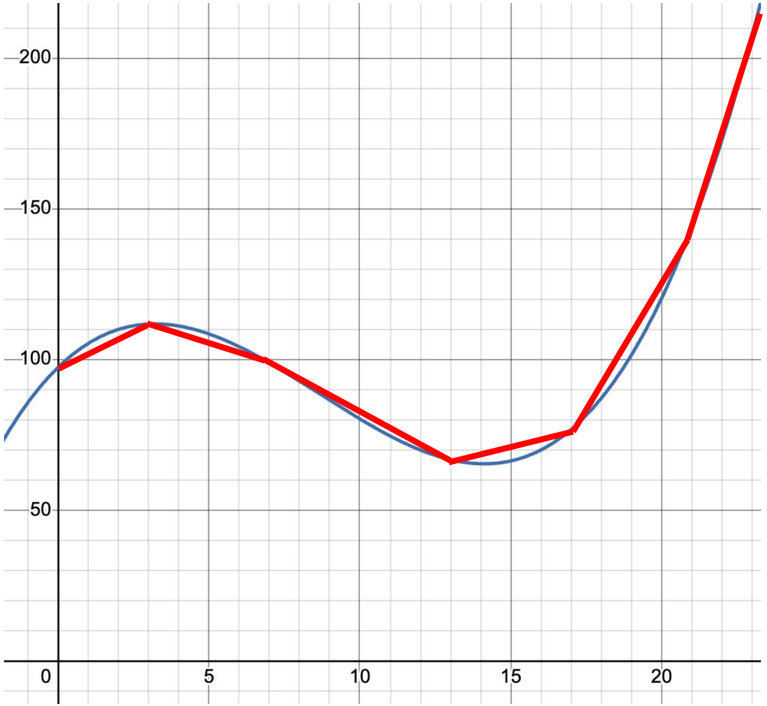
$$
\begin{array}{rl}
\text{minimize} & 5\delta_1 + 2\delta_2 + 3\delta_3 \\
\text{subject to} & i_1 \cdot 400 \leq \delta_1 \leq 400 \\
& i_2 \cdot 200 \leq \delta_2 \leq i_1 \cdot 200 \\
& 0 \leq \delta_3 \leq 400 \cdot i_2 \\
& i_1, i_2 \in \{0, 1\}
\end{array}
$$

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!

Indicators for Constraints

- More generally, can create indicator c for constraint $n \geq b$ if we have bounds $L \leq n - b \leq U$
  - Can replace n with any linear expression $a_1 n_1 + a_2 n_2 + \cdots + a_k n_k$, but it needs to be integer-valued
- To enforce $(c = 1) \Rightarrow (n \geq \mathrm{b})$, add constraint:

$$n - b \geq L(1 - c)$$

- To enforce $(c = 0) \Rightarrow (n \leq \mathrm{b} - 1)$, add constraint:

$$n - b \leq (U + 1)c - 1$$

Modeling Fixed Costs

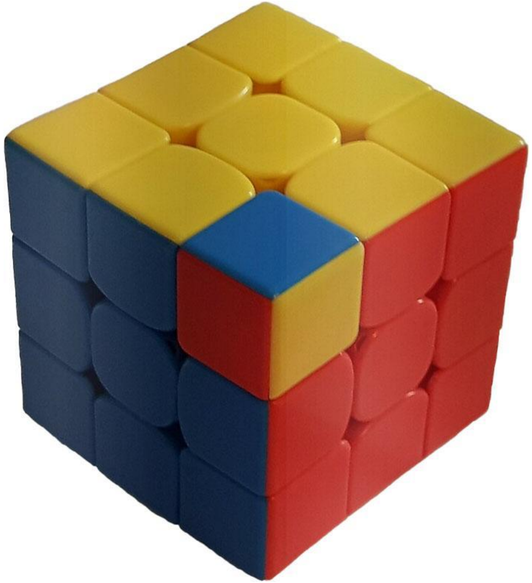
- So to make an indicator c for $n \geq 1$, add:

  $$n \leq (U+1)c$$

- If minimizing cost, don't need to enforce $(c = 1) \Rightarrow (n \geq 1)$
  - Why? Equivalent to $(n = 0) \Rightarrow (c = 0)$
  - Since cost is $250c + 10n$, solver will set $c = 0$ if possible when minimizing

Debugging Integer Programs

- Your model is `INFEASIBLE` when it shouldn't be… what to do?
- Want to find which buggy constraint(s) cannot be satisfied



Debugging Integer Programs

- Typical model has thousands, even millions of constraints
- **Insight:** bugs usually happen at the level of groups of constraints, not individual constraints

```python
# Each job gets exactly one CPU
for j in range(num_jobs):
    solver.Add(
        sum(x[c, j] for c in range(num_cpus)) == 1
    )

# Each cluster gets at most two jobs
for cluster in clusters:
    solver.Add(
        sum(x[c, j] for c in cluster for j in range(num_jobs)) <= 2
    )
```

group

group

Debugging Integer Programs

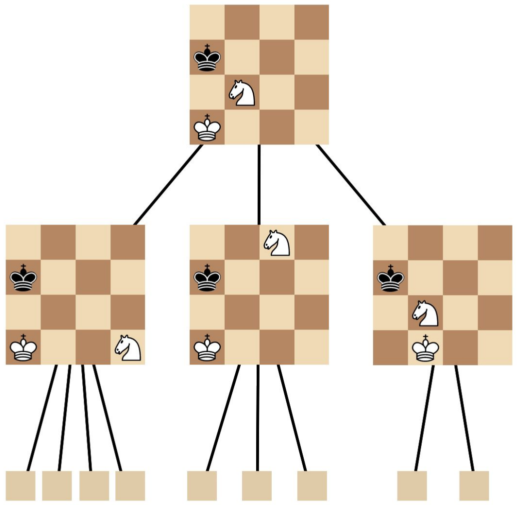
- If we get rid of all buggy constraint groups, the model should become feasible
- **Strategy:** remove groups one-by-one until model is feasible, then add them back to find minimal set of buggy groups
  - Even better: use a “binary search” strategy (remove half the constraint groups at a time)

Debugging Integer Programs

- What if the model is feasible, but the solution is wrong?
- If it's easy to see that a constraint is violated, check that one
- Otherwise, just add constraints enforcing a known "right" solution, and then model will become infeasible
    - If you don't have a known solution, enforce whatever property is violated in the wrong solution (e.g. `objective <= 300`)

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?

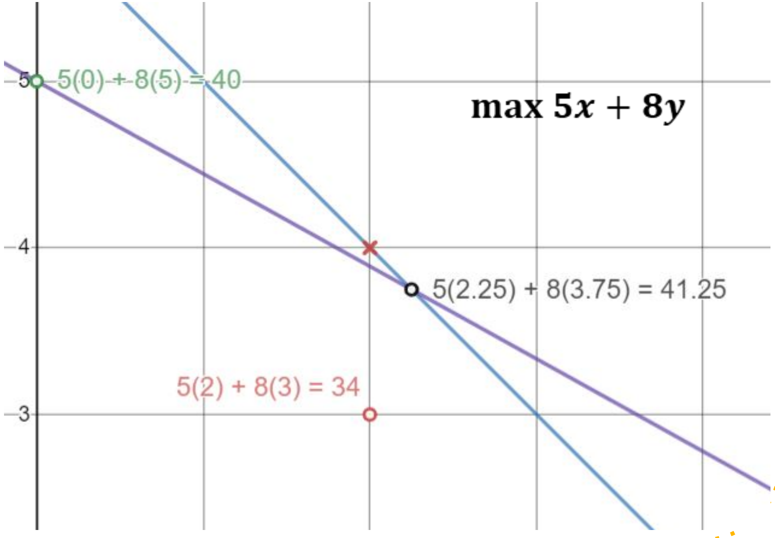
- Does naive branching even terminate?
  - Only for pure integer programs!
- Which assignments does the algorithm discard or visit?
  - Need to evaluate both branches -- visits all feasible solutions!
- Basically the same as brute force
- Runtime scales with size of search space

Recall: LP Relaxation

- For a MIP P, we get its **LP relaxation** $LP(P)$ by allowing all variables to be fractional
    - Can't just round LP solution

- **Key observation:** the LP solution is always at least as good as the MIP solution (by objective value)

- Corollary: if all integer vars take integer values in optimal solution to $LP(P)$, then it is also optimal solution to P



Adding Inference

- **Idea:** since LP is polytime-solvable, use LP solver as inference engine!
- Instead of recursing until all variables have one value, solve $LP(P)$ and check whether all integer variables have integer values
- Branch on integer variable x whose value v is fractional in $LP(P)$
  - Create subproblems $x \leq \lfloor v \rfloor$ and $x \geq \lceil v \rceil$

Pruning Fruitless Nodes

- **Idea:** discard partial solutions that will never yield a better objective value than one we've already found

- If we've seen a MIP solution with a better objective value than LP(P), discard P since any integer solution can only be worse

Branch & Bound

- First version developed by Ailsa Land and Alison Harcourt in 1960
- Combines branching of solution space with bounds-based pruning
- B&B is an **algorithm paradigm**: a “meta-algorithm” that can be used to design algorithms for many different optimization algorithms

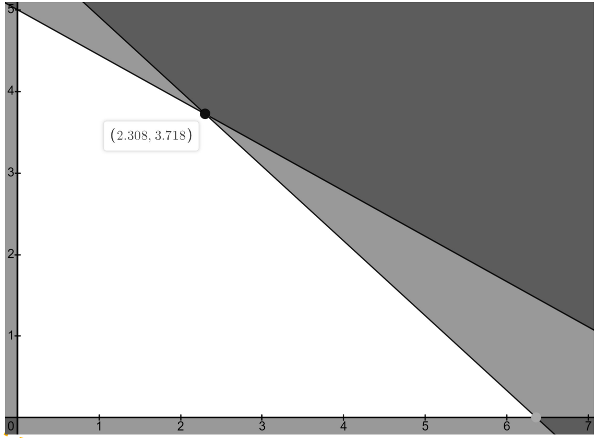
Branch & Bound (Pseudocode)

```
# find the optimal objective value for P
# best_seen is the best objective value so far
branch_and_bound(P, best_seen = -inf):
    let LP_soln = solve_LP(LP(P))
    if LP_soln = INFEASIBLE: return INFEASIBLE
    if objective_value(LP_soln) ≤ best_seen:
        return -inf
    if LP_soln satisfies integrality constraints of P:
        return objective_value(LP_soln)
    let x be an int var with fractional value v in LP_soln
    let obj1 = branch_and_bound(P|x ≤ ⌊v⌋, best_seen)
    set best_seen = max{obj1, best_seen}
    let obj2 = branch_and_bound(P|x ≥ ⌈v⌉, best_seen)
    return max{obj1, obj2}
```

Example: Branch & Bound

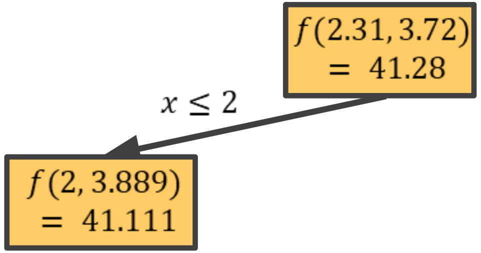
$$\begin{array}{ll} \text{max} & f(x,y) = 5x + 8y \\ \text{s.t.} & 5x + 9y \leq 45 \\ & 1.1x + 1.2y \leq 7 \\ & x, y \in [0..100] \end{array}$$

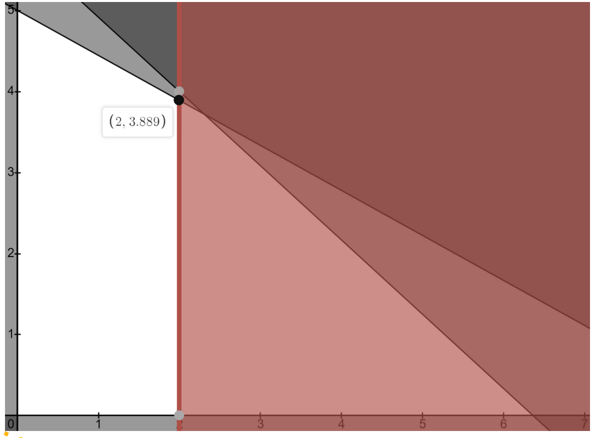
$\boxed{f(2.31, 3.72) = 41.28}$



Example: Branch & Bound

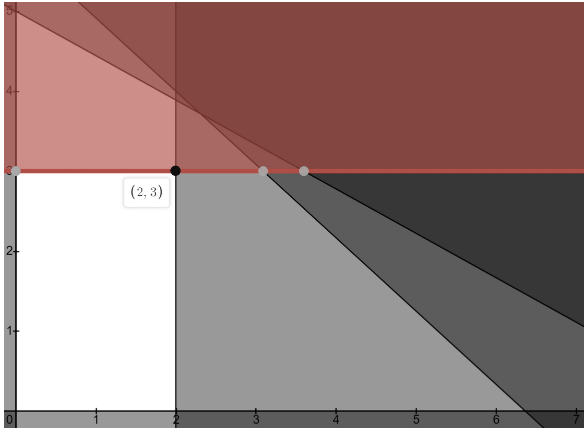
$$
\begin{array}{ll}
\text{max} & f(x,y) = 5x + 8y \\
\text{s.t.} & 5x + 9y \leq 45 \\
 & 1.1x + 1.2y \leq 7 \\
 & x, y \in [0..100]
\end{array}
$$

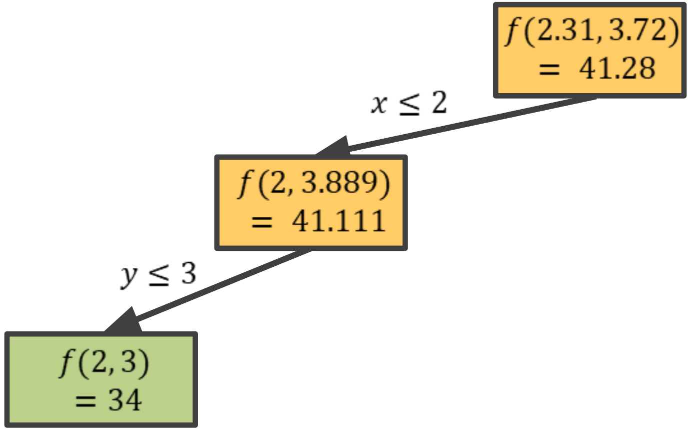




Example: Branch & Bound

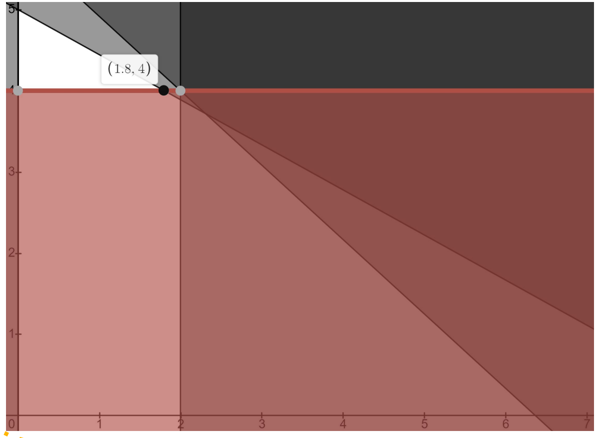
$$
\begin{array}{ll}
\max & f(x,y) = 5x + 8y \\
\text{s.t.} & 5x + 9y \leq 45 \\
 & 1.1x + 1.2y \leq 7 \\
 & x, y \in [0..100]
\end{array}
$$

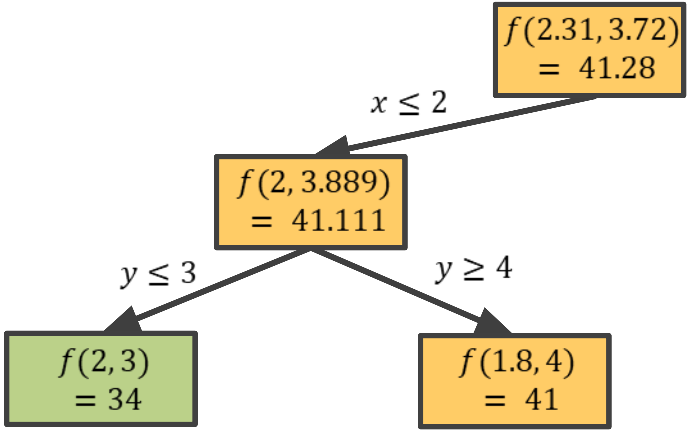




Example: Branch & Bound

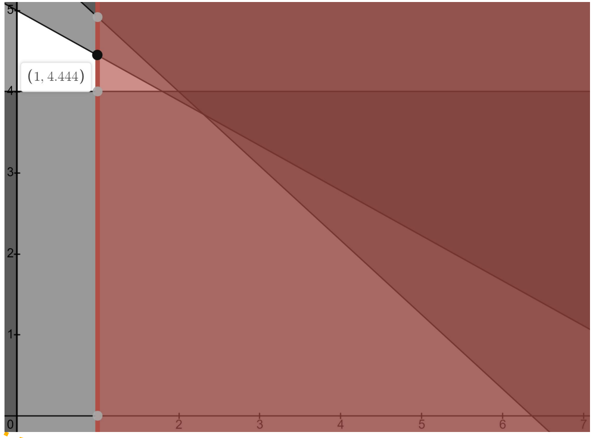
max	$f(x, y) = 5x + 8y$
s.t.	$5x + 9y \leq 45$
	$1.1x + 1.2y \leq 7$
	$x, y \in [0..100]$

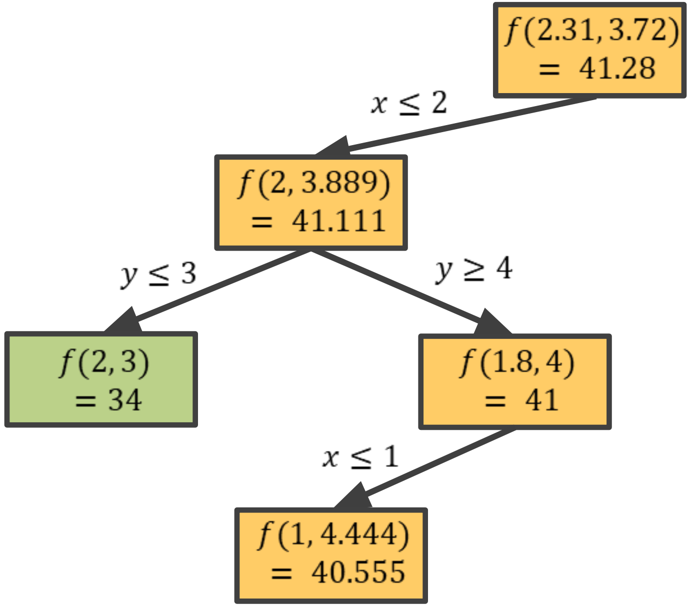




Example: Branch & Bound

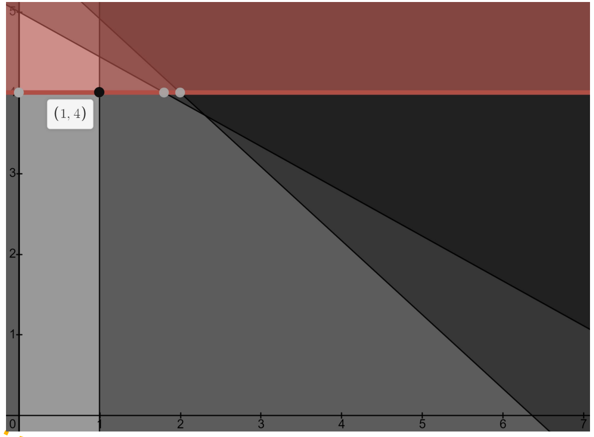
$$\begin{array}{ll} \text{max} & f(x,y) = 5x + 8y \\ \text{s.t.} & 5x + 9y \leq 45 \\ & 1.1x + 1.2y \leq 7 \\ & x, y \in [0..100] \end{array}$$

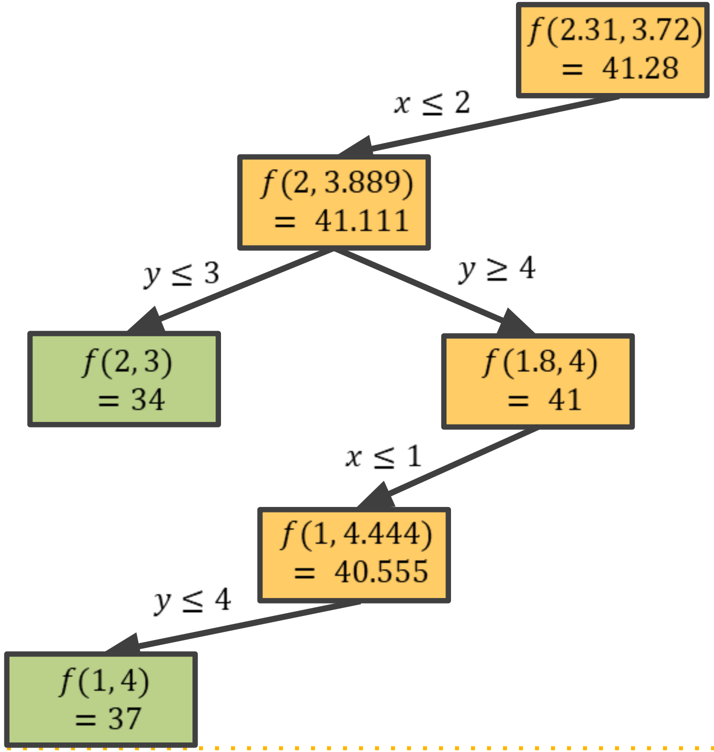




Example: Branch & Bound

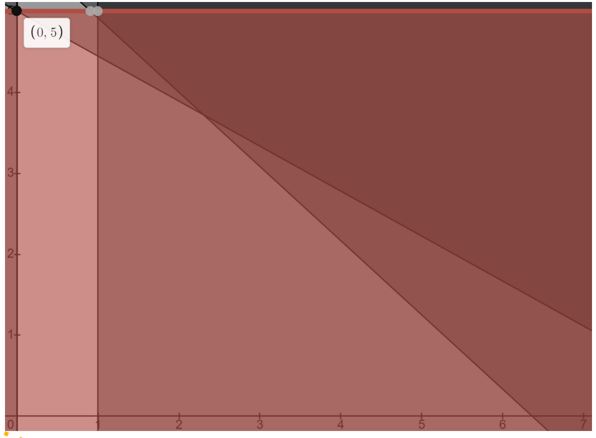
$$
\begin{array}{ll}
\text{max} & f(x,y) = 5x + 8y \\
\text{s.t.} & 5x + 9y \leq 45 \\
 & 1.1x + 1.2y \leq 7 \\
 & x, y \in [0..100]
\end{array}
$$

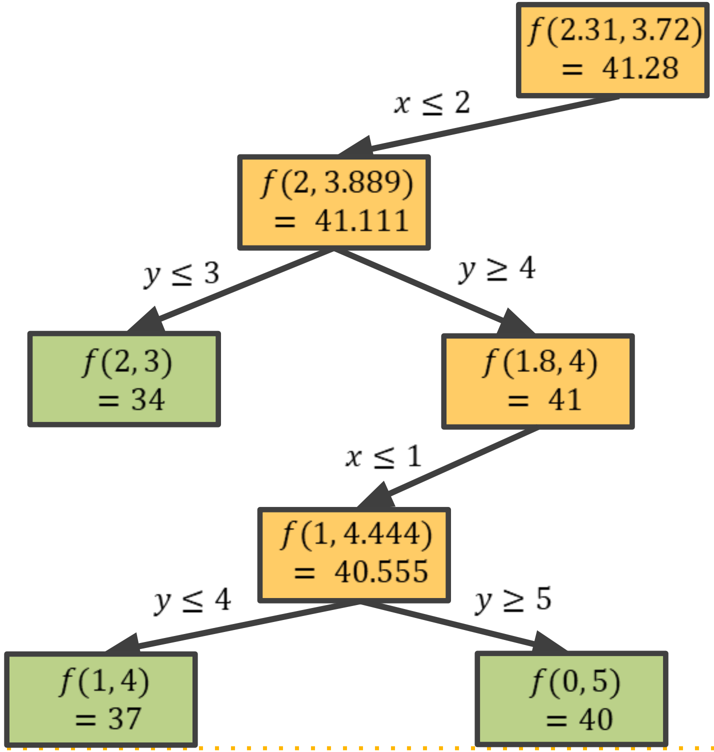




Example: Branch & Bound

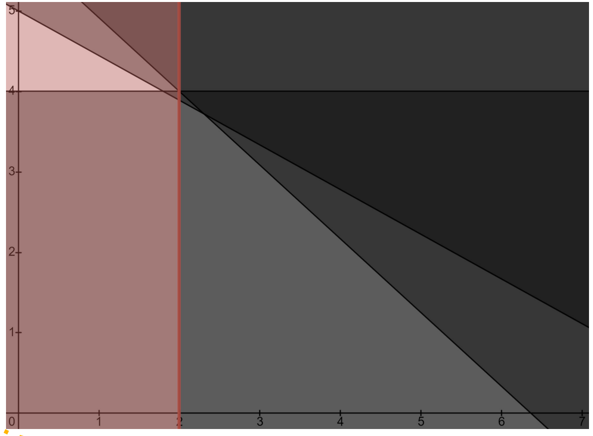
max	$f(x,y) = 5x + 8y$
s.t.	$5x + 9y \leq 45$
	$1.1x + 1.2y \leq 7$
	$x, y \in [0..100]$

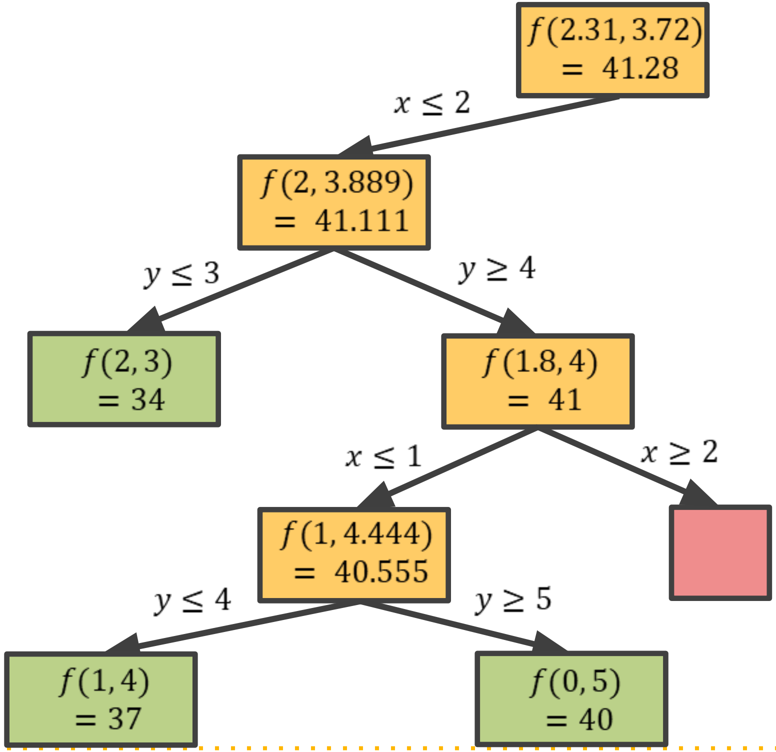




Example: Branch & Bound

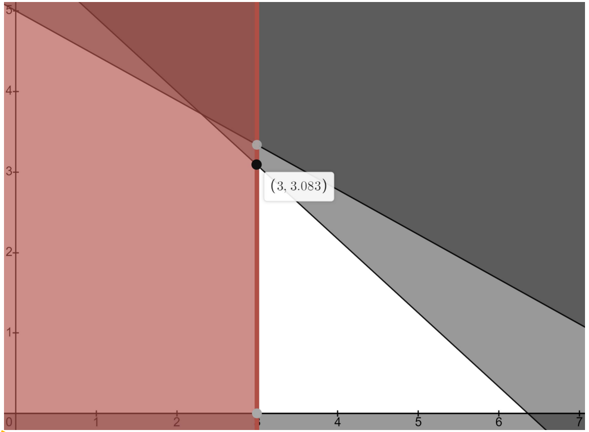
max	$f(x,y) = 5x + 8y$
s.t.	$5x + 9y \leq 45$
	$1.1x + 1.2y \leq 7$
	$x, y \in [0..100]$

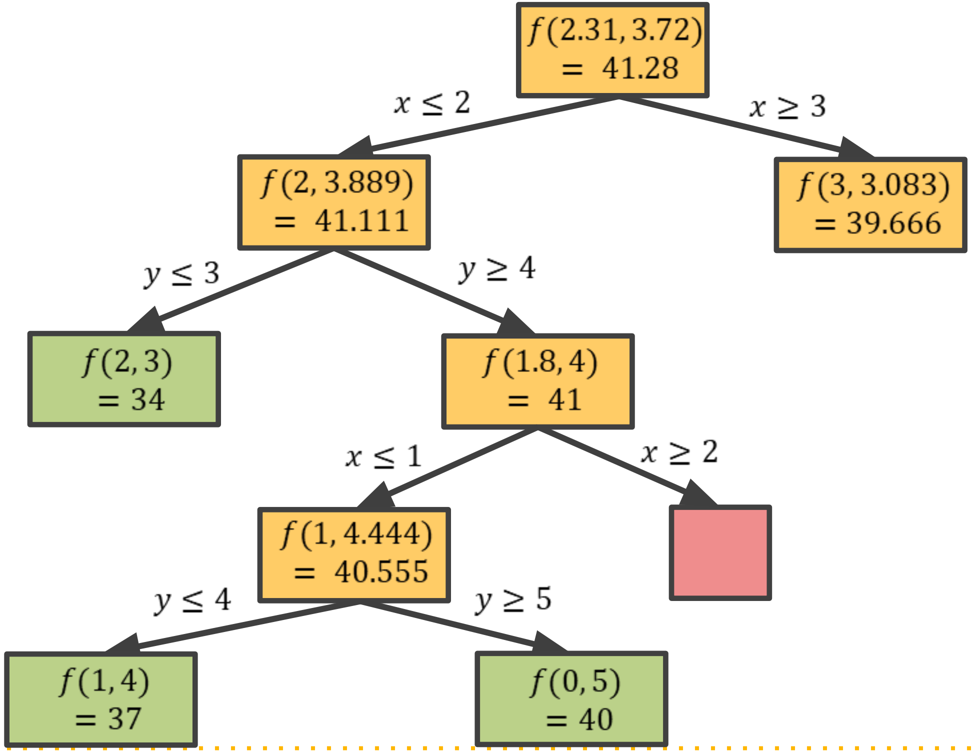




Example: Branch & Bound

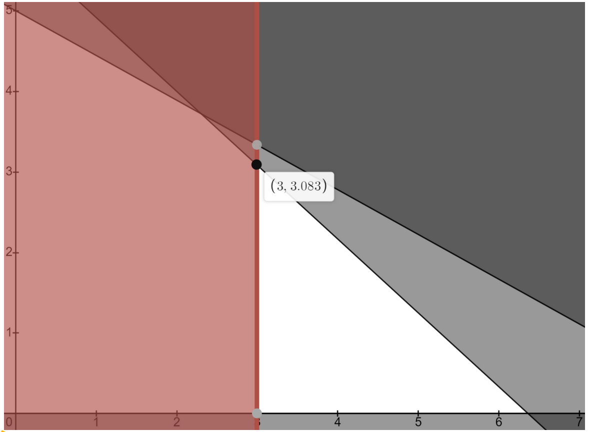
$$\begin{array}{ll} \text{max} & f(x,y) = 5x + 8y \\ \text{s.t.} & 5x + 9y \leq 45 \\ & 1.1x + 1.2y \leq 7 \\ & x, y \in [0..100] \end{array}$$

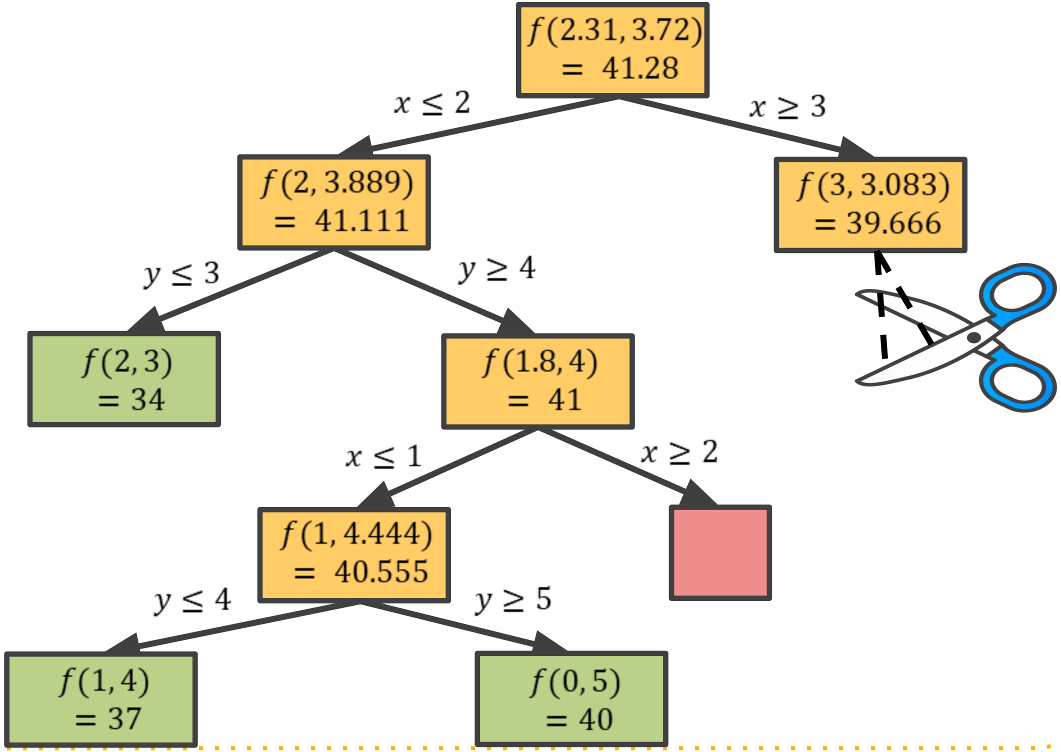




Example: Branch & Bound

$$
\begin{array}{ll}
\text{max} & f(x,y) = 5x + 8y \\
\text{s.t.} & 5x + 9y \leq 45 \\
 & 1.1x + 1.2y \leq 7 \\
 & x, y \in [0..100]
\end{array}
$$





Iterative Branch & Bound

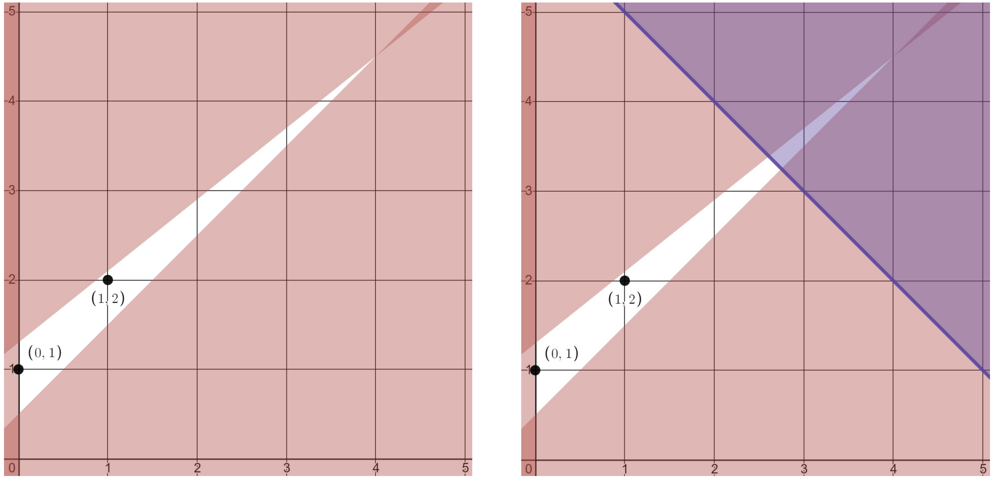
```
# find the optimal objective value for P_0
branch_and_bound(P_0):
  let best_seen = -inf
  let subproblems_to_visit = {P_0}
  while to_visit is nonempty:
    let P = subproblems_to_visit.pop()
    let LP_soln = solve_LP(LP(P))
    if LP_soln = INFEASIBLE: continue
    if objective_value(LP_soln) ≤ best_seen: continue
    if LP_soln satisfies integrality constraints for P:
      set best_seen = objective_value(LP_soln)
      continue
    let x be an int var with fractional value v in LP_soln
    subproblems_to_visit.add(branch_and_bound(P | x ≤ ⌊v⌋))
    subproblems_to_visit.add(branch_and_bound(P | x ≥ ⌈v⌉))
  return best_seen
```

Tuning Branch & Bound

- What choices can we make when implementing branch and bound?
- Which subproblem to visit next?
  - Visit first-added subproblem (BFS)
  - Visit last-added subproblem (DFS)
  - Visit subproblem with best LP objective (“best-first search”)
- Which variable to branch on?
  - Most constrained variable (smallest domain, e.g. booleans)
  - Largest/smallest coefficient in objective function
  - Closest/farthest to halfway between integers (e.g. value of 0.5)
- Most solvers allow user to tune these based on knowledge of problem

Improving B&B with Cuts

- Informally, a **cut** for a MIP P is a new constraint (inequality) that doesn't eliminate any feasible solutions for P, but does for $LP(P)$
  - Tighter LP relaxation means we converge faster to MIP solution!



Branch & Cut

- If we can find cuts of MIP, then add them and recurse on new MIP!
    - How to find cuts? Out of scope – method based on simplex algorithm
- Otherwise, branch to create subproblems as before

- Proposed by Manfred Padberg and Giovanni Rinaldi in 1989

The Knapsack Problem

- Given n items with values $v_1, \dots, v_n$ and weights $w_1, \dots w_n$, select maximum-value subset to fit into a knapsack with capacity W.

Fractional Knapsack

- What if items are subdivisible? Want to decide how much of each item to take (as a fraction from 0 to 1).
- Intuitively, do we want to prioritize... most valuable items? Lightest items? Something else?
- **Greedy algorithm:** Sort items by value-to-weight ratio. Take as much of each item as possible, in order, until knapsack is full.

0/1 Knapsack

- In the 0/1 knapsack problem, we either select an item or we don't.
- Does greedy algorithm still work?
  - No: 0/1 knapsack is NP-complete!

MIP for 0/1 Knapsack

- MIP formulation is very straightforward:

$$\text{maximize} \quad \sum_{i=1}^{n} x_i v_i$$

$$\text{subject to} \quad \sum_{i=1}^{n} x_i w_i \leq W$$

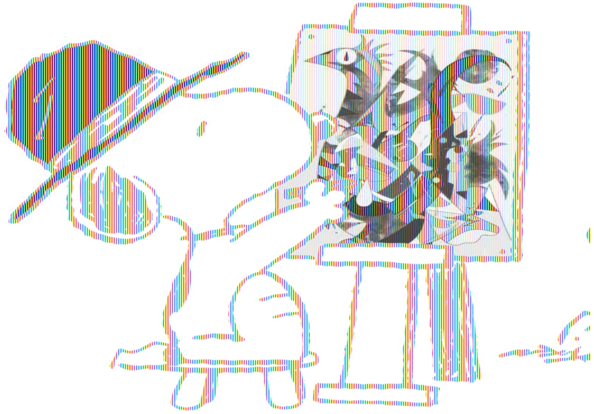
- Why use MIP instead of…
  - $O(nW)$ dynamic programming algorithm
  - $O(n \lg n)$ approximation algorithm (at least 50% of optimal)

B&B for Knapsack

- How can we use branch and bound as an **algorithm paradigm** for the 0/1 knapsack problem (without using MIP)?

```
b&b_knapsack(items, W, best_seen):
    let fractional_soln = greedy_fractional(items, W)
    if value(fractional_soln) ≤ best_seen:
        return -inf
    if fractional_soln has no fractionally-selected items:
        return value(fractional_soln)
    let x be a fractionally-selected item in fractional_soln
    let obj1 = b&b_knapsack(items - {x}, W, best_seen)}
    set best_seen = max{obj1, best_seen}
    let obj2 = v(x) + b&b_knapsack(items - {x}, W - w(x), best_seen - v(x))
    return max{obj1, obj2}
```

Stay Vigilant



"Do not set yourself on fire just to keep the others around you warm."