CIS1921

Lecture 7:

Concluding Mixed-Integer Programming

Reminders

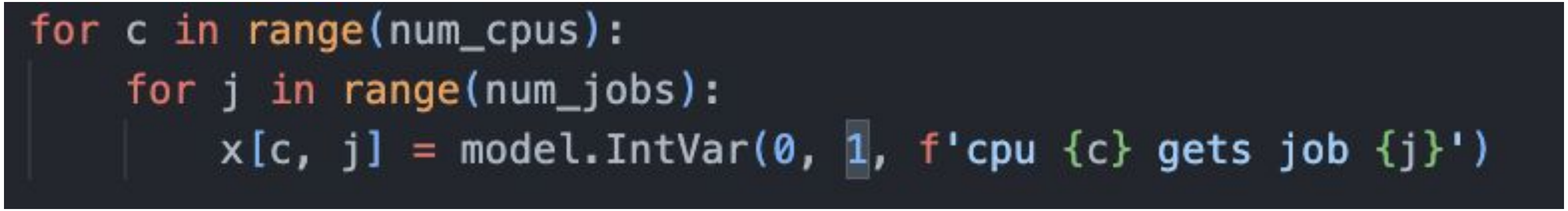
- HW3 release tonight or tomorrow depending on how I'm feeling
- HW3 due March 18
- Next class
- Following week spring break
- Week after that – super special lecture.

CPU Job Assignment Problem

- There are *n* jobs that must be completed
- There are *m* CPUs available to do the jobs
    - Each CPU can do at most one job, hence m >= n
- There is a cost associated with running a particular job on a particular CPU
- How to assign jobs to CPUs, minimizing total cost?

Variables

$$x_{c,j} = \begin{cases} 1 & \text{if CPU } c \text{ gets job } j \\ 0 & \text{else} \end{cases}$$

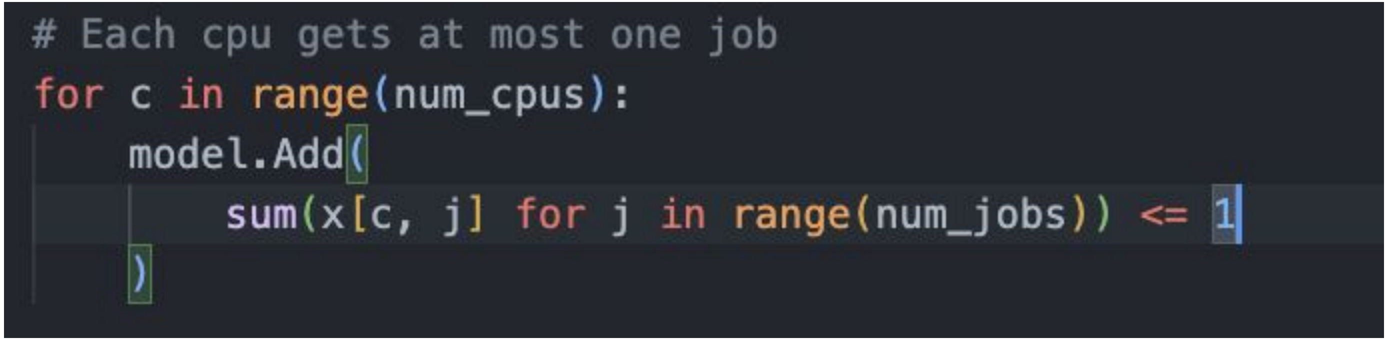


```
for c in range(num_cpus):
    for j in range(num_jobs):
        x[c, j] = model.IntVar(0, 1, f'cpu {c} gets job {j}')
```

Constraint

- Each CPU gets at most 1 job

$$\text{For each CPU } c, \quad \sum_{j \in \text{jobs}} x_{c,j} \leq 1$$

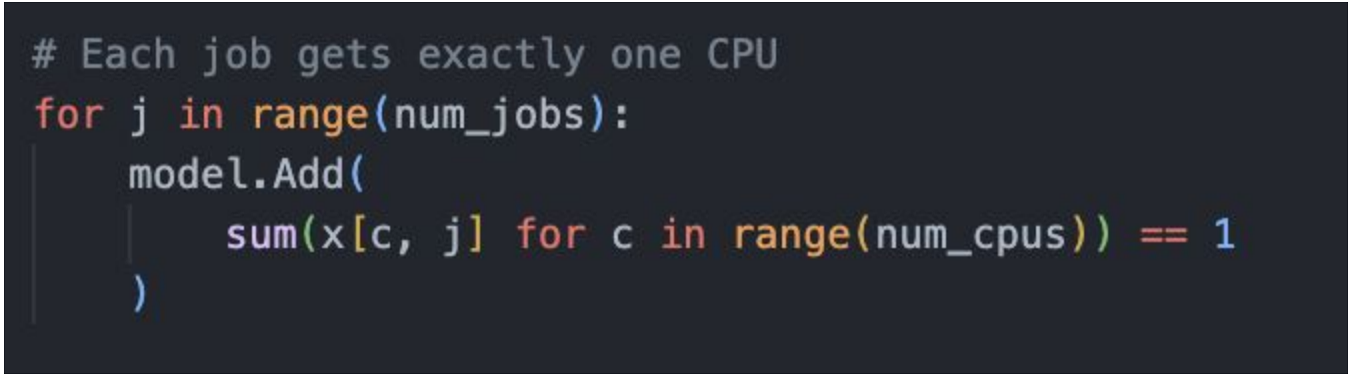


```
# Each cpu gets at most one job
for c in range(num_cpus):
    model.Add(
        sum(x[c, j] for j in range(num_jobs)) <= 1
    )
```

Constraint

- Each Job gets exactly one CPU

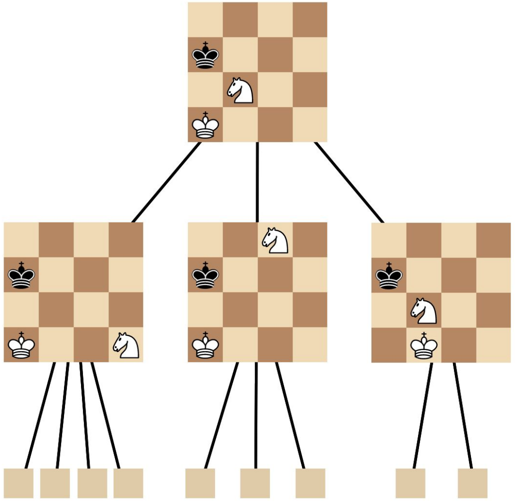
For a job $j^\star$, over all CPUs, exactly one of $x_{c_1,j^\star}, x_{c_2,j^\star}, \ldots, x_{c_n,j^\star}$ equals 1



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

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)

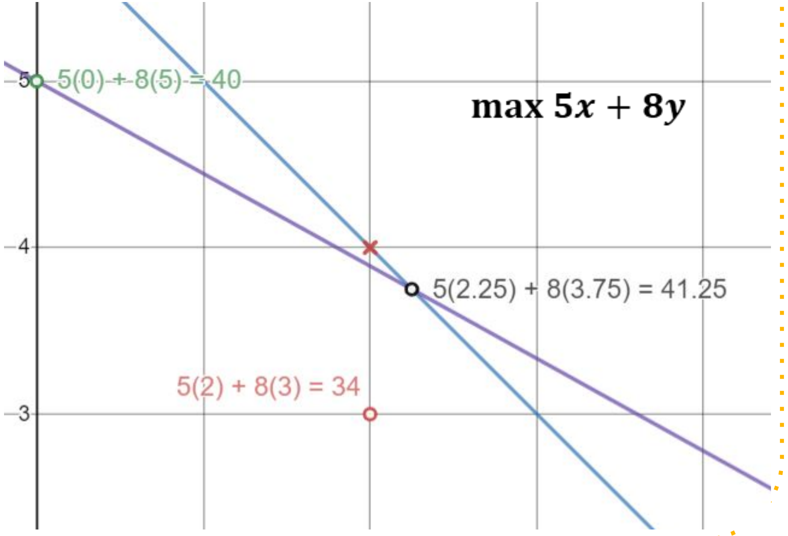
```
# find the optimal objective value for $P$

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 = $\lfloor$(lb($x$) + ub($x$)) / 2$\rfloor$
    return max{naive($P|x \le$ m), naive($P|x \ge$ 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

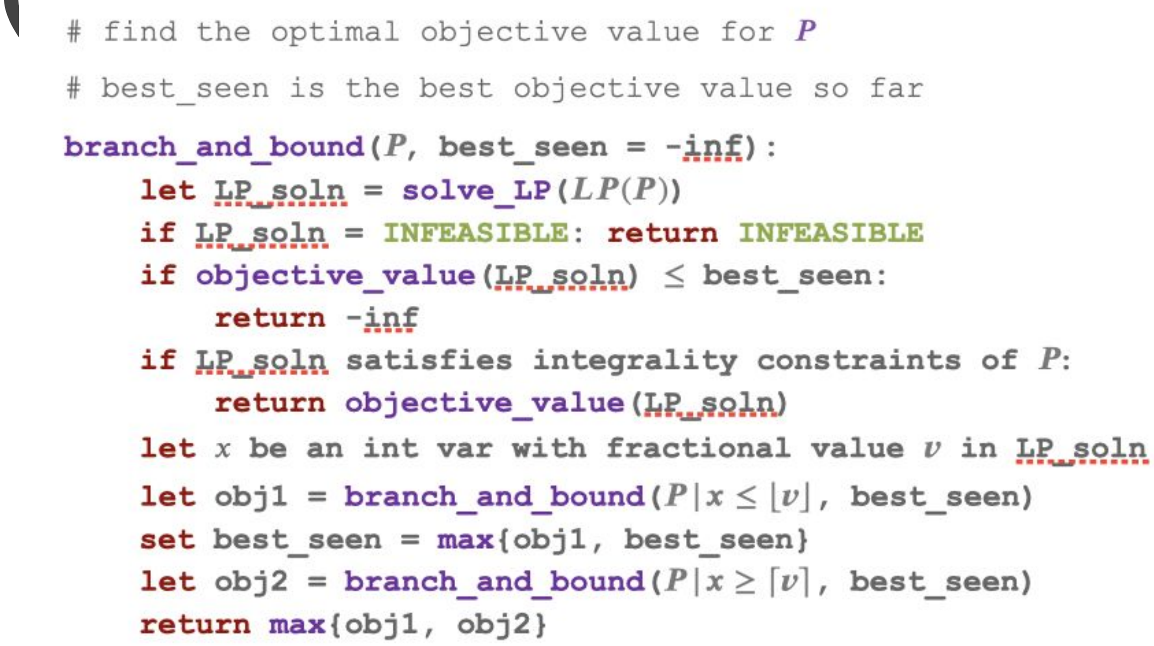
- **Idea:** discard partial solutions that will never yield a better objective value than one we've already found

- 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)

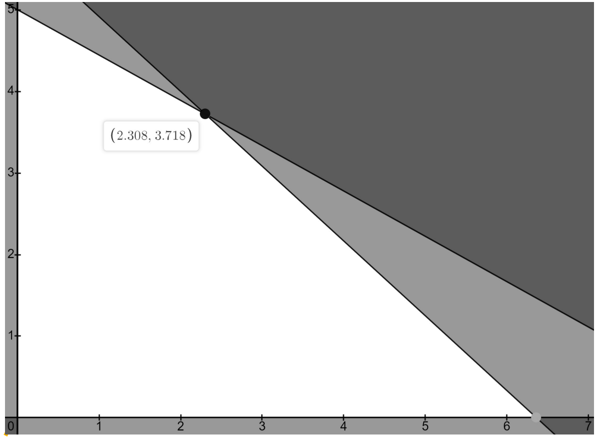


```
# find the optimal objective value for $P$
# best_seen is the best objective value so far
branch_and_bound($P$, best_seen = -inf):
    let LP_soln = solve_LP($LP(P)$)
    if LP_soln = INFEASIBLE: return INFEASIBLE
    if objective_value(LP_soln) $\leq$ best_seen:
        return -inf
    if LP_soln 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($P|x \leq \lfloor v \rfloor$, best_seen)
    set best_seen = max{obj1, best_seen}
    let obj2 = branch_and_bound($P|x \geq \lceil v \rceil$, best_seen)
    return max{obj1, obj2}
```

Example: Branch & Bound

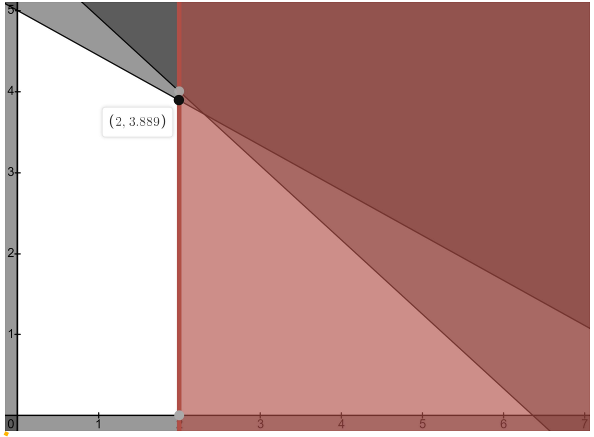
$$
\begin{array}{ll}
\text{max} & f(x,y) = 5x + 8y \\
\text{s.t.} & 5x + 9y \leq 45 \\
 & 1.1x + 1.2y \leq 7 \\
 & x, y \in [0..100]
\end{array}
$$

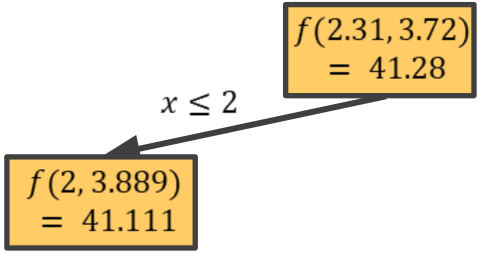
$$\boxed{\begin{aligned} f(2.31, 3.72) \\ = 41.28 \end{aligned}}$$



Example: Branch & Bound

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





$f(2.31, 3.72) = 41.28$

$x \leq 2$

$f(2, 3.889) = 41.111$

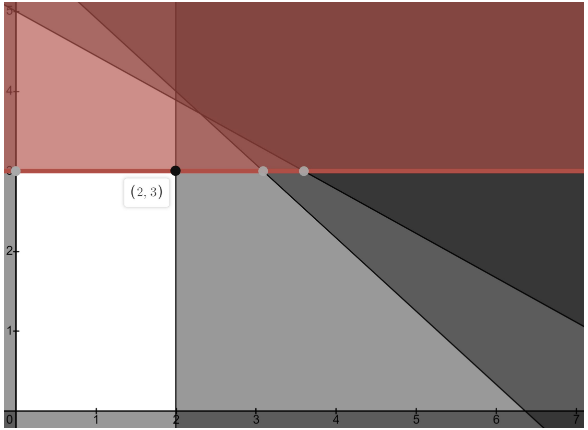
Example: Branch & Bound

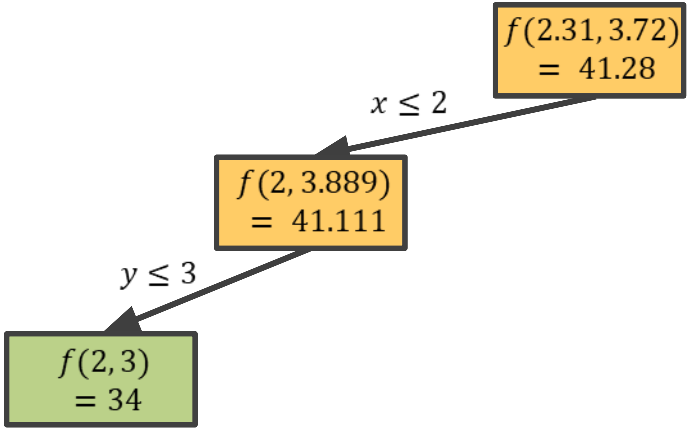
max $f(x, y) = 5x + 8y$

s.t. $5x + 9y \leq 45$

$1.1x + 1.2y \leq 7$

$x, y \in [0..100]$





$f(2.31, 3.72) = 41.28$

$x \leq 2$

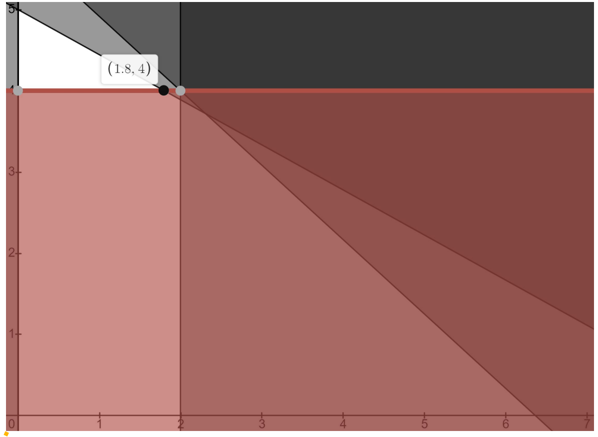
$f(2, 3.889) = 41.111$

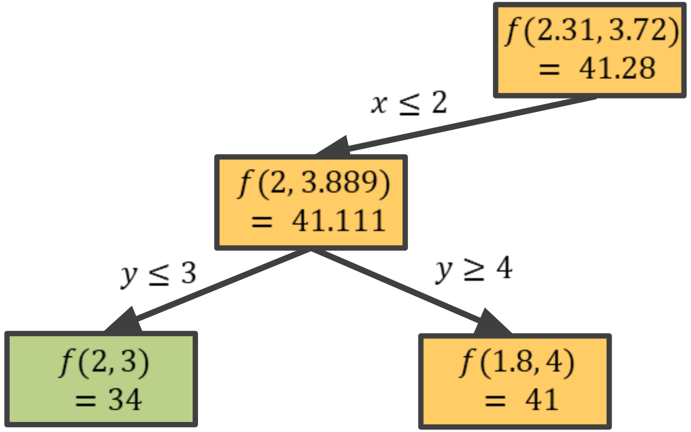
$y \leq 3$

$f(2, 3) = 34$

Example: Branch & Bound

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




$f(2.31, 3.72) = 41.28$

$x \leq 2$

$f(2, 3.889) = 41.111$

$y \leq 3$

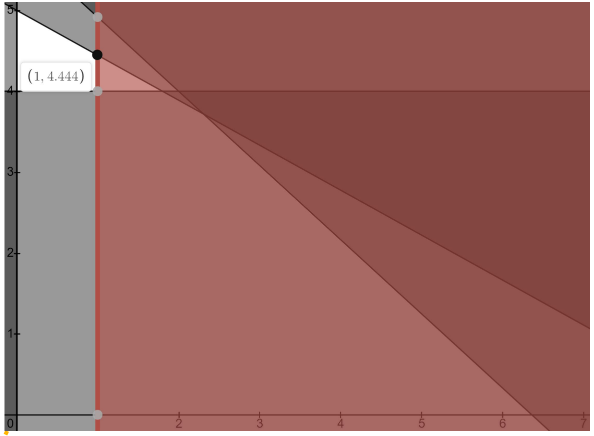
$y \geq 4$

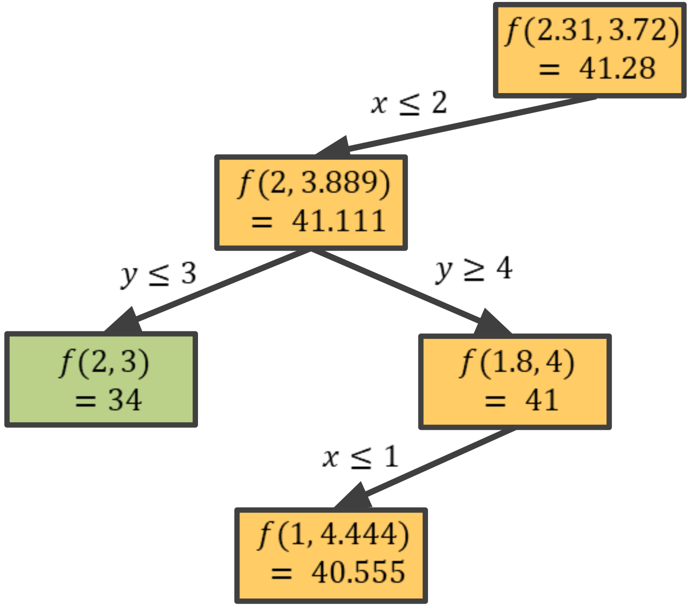
$f(2, 3) = 34$

$f(1.8, 4) = 41$

Example: Branch & Bound

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





$f(2.31, 3.72) = 41.28$

$x \leq 2$

$f(2, 3.889) = 41.111$

$y \leq 3$

$f(2,3) = 34$

$y \geq 4$

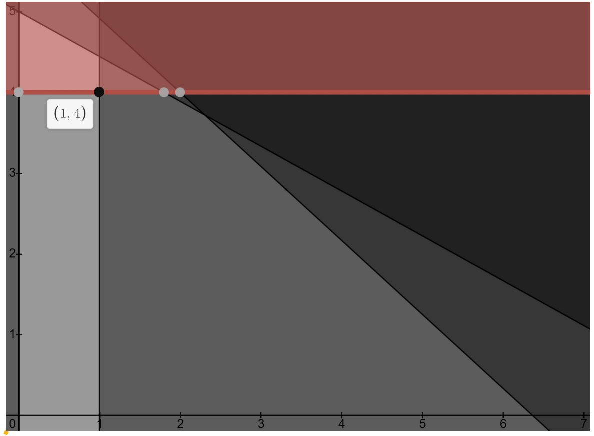
$f(1.8, 4) = 41$

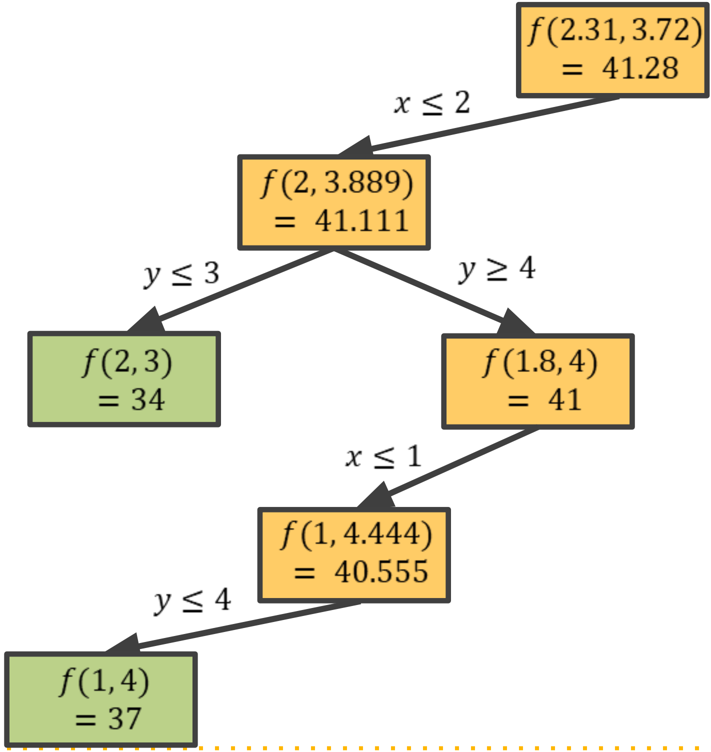
$x \leq 1$

$f(1, 4.444) = 40.555$

Example: Branch & Bound

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




$f(2.31, 3.72) = 41.28$

$x \leq 2$: $f(2, 3.889) = 41.111$

$y \leq 3$: $f(2, 3) = 34$

$y \geq 4$: $f(1.8, 4) = 41$

$x \leq 1$: $f(1, 4.444) = 40.555$

$y \leq 4$: $f(1, 4) = 37$

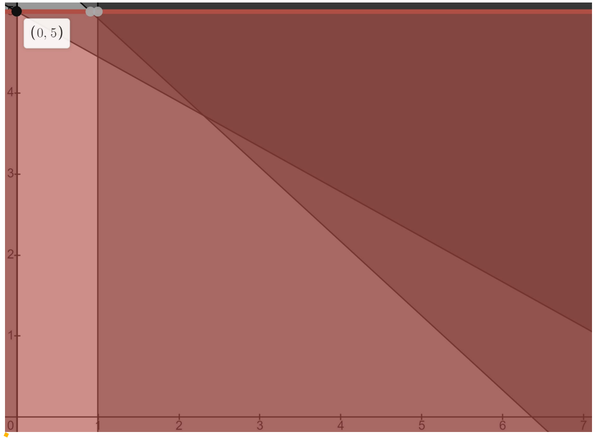
Example: Branch & Bound

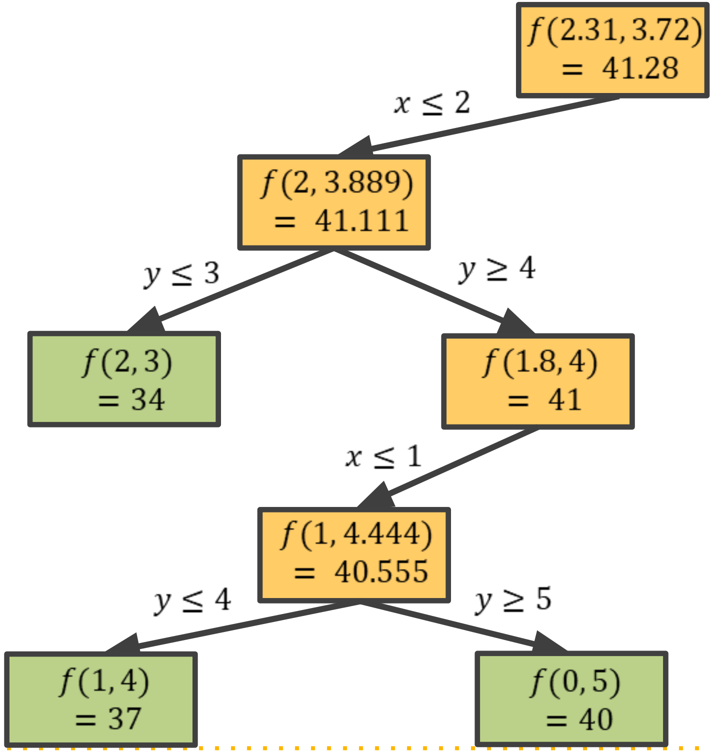
max $\quad f(x,y) = 5x + 8y$

s.t. $\quad 5x + 9y \leq 45$

$1.1x + 1.2y \leq 7$

$x, y \in [0..100]$





$f(2.31, 3.72) = 41.28$

$x \leq 2$

$f(2, 3.889) = 41.111$

$y \leq 3$

$f(2,3) = 34$

$y \geq 4$

$f(1.8, 4) = 41$

$x \leq 1$

$f(1, 4.444) = 40.555$

$y \leq 4$

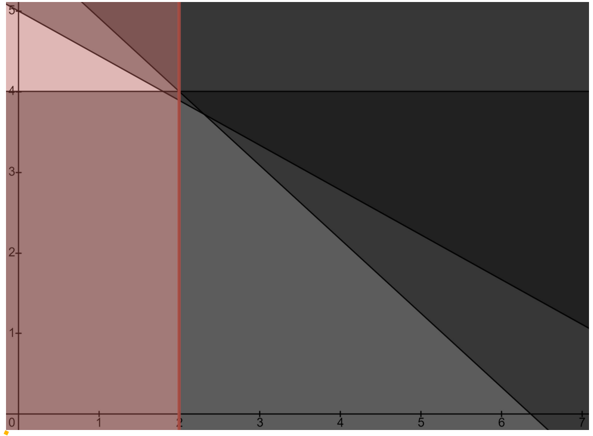
$f(1,4) = 37$

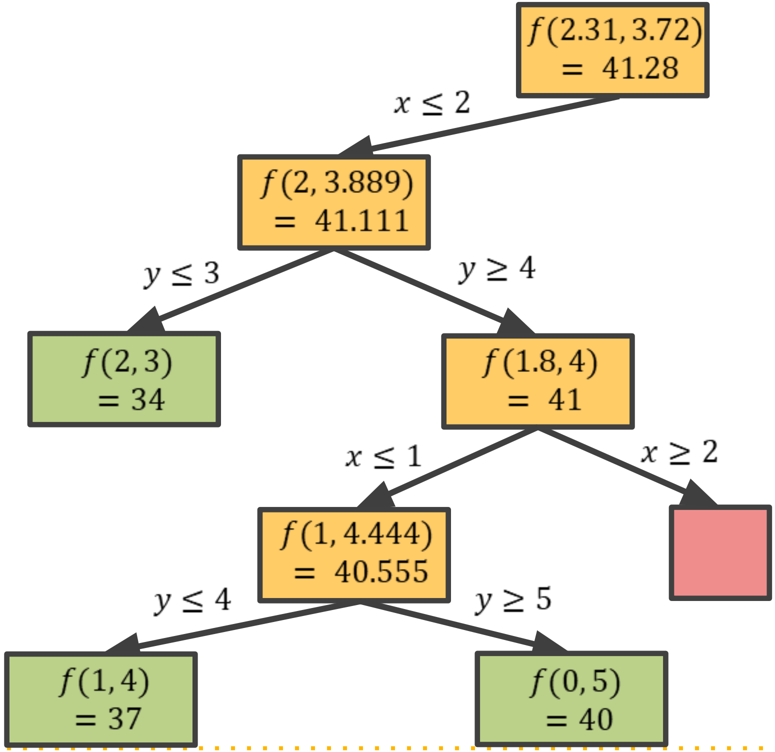
$y \geq 5$

$f(0,5) = 40$

Example: Branch & Bound

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





$f(2.31, 3.72) = 41.28$

$x \leq 2$

$f(2, 3.889) = 41.111$

$y \leq 3$

$y \geq 4$

$f(2,3) = 34$

$f(1.8, 4) = 41$

$x \leq 1$

$x \geq 2$

$f(1, 4.444) = 40.555$

$y \leq 4$

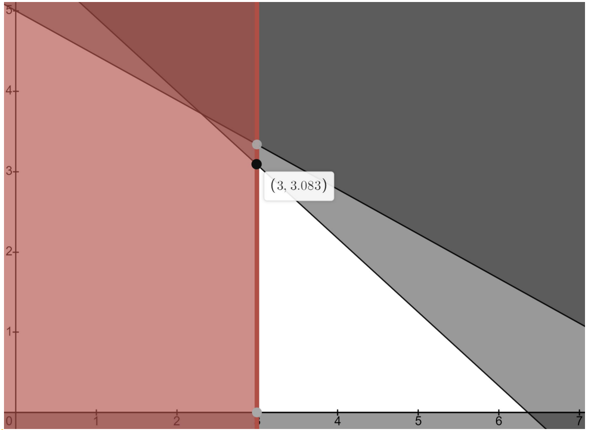
$y \geq 5$

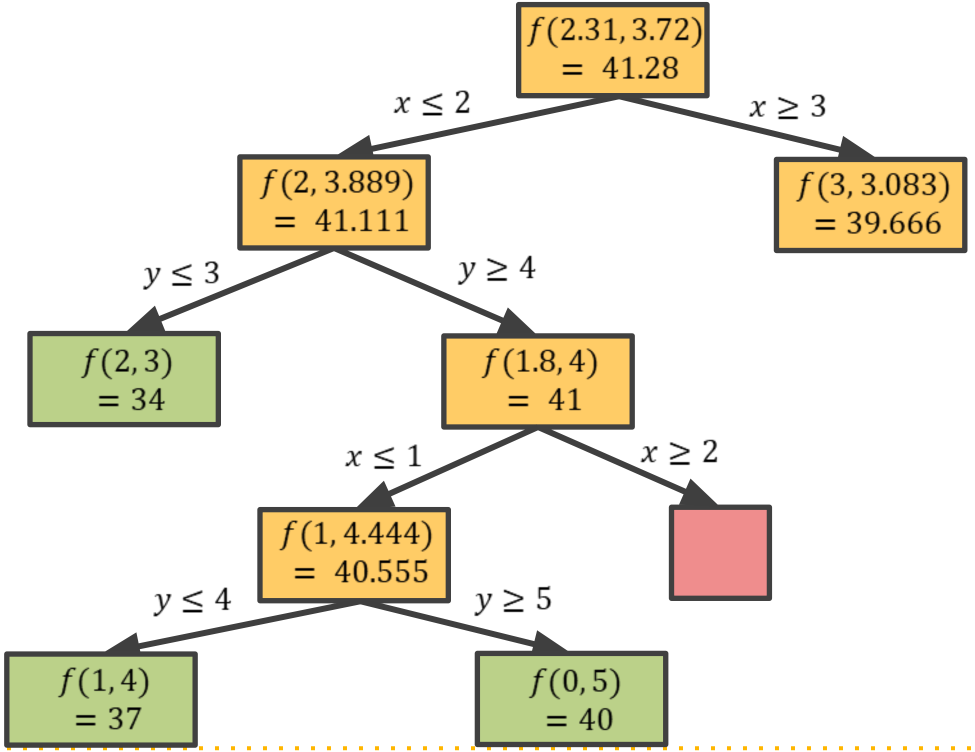
$f(1,4) = 37$

$f(0,5) = 40$

Example: Branch & Bound

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




$f(2.31, 3.72) = 41.28$

$x \leq 2$

$x \geq 3$

$f(2, 3.889) = 41.111$

$f(3, 3.083) = 39.666$

$y \leq 3$

$y \geq 4$

$f(2,3) = 34$

$f(1.8, 4) = 41$

$x \leq 1$

$x \geq 2$

$f(1, 4.444) = 40.555$

$y \leq 4$

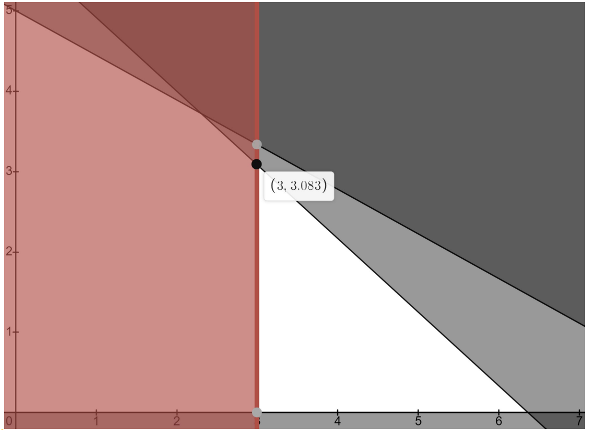
$y \geq 5$

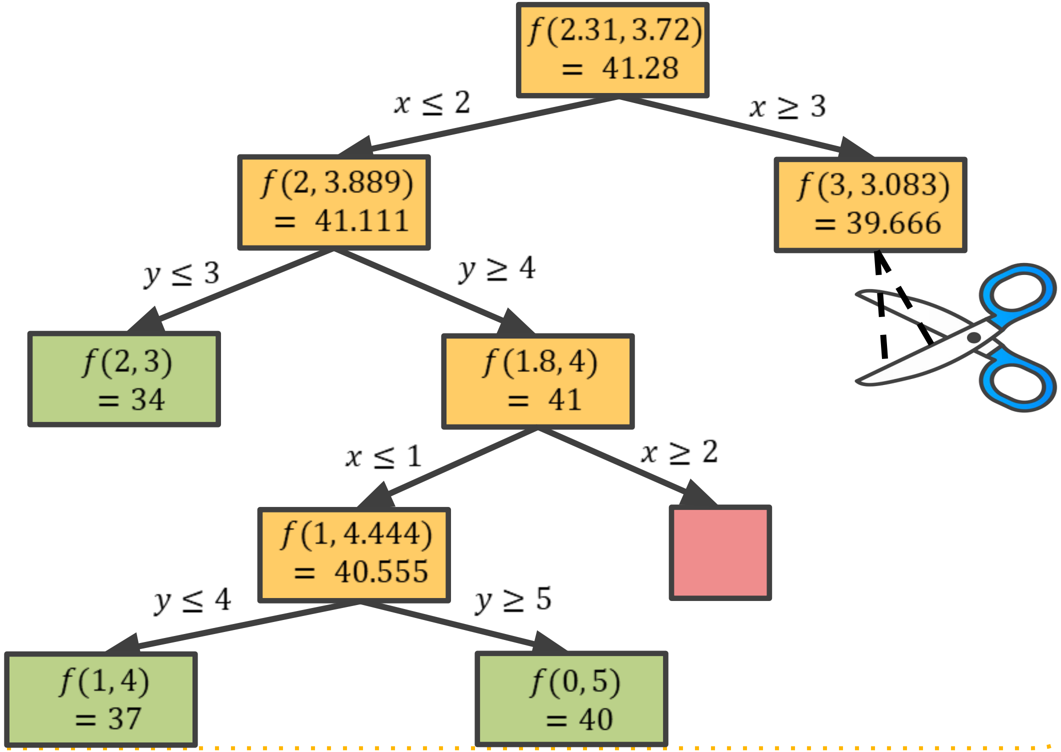
$f(1,4) = 37$

$f(0,5) = 40$

Example: Branch & Bound

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





$f(2.31, 3.72) = 41.28$

$x \leq 2$

$x \geq 3$

$f(2, 3.889) = 41.111$

$f(3, 3.083) = 39.666$

$y \leq 3$

$y \geq 4$

$f(2, 3) = 34$

$f(1.8, 4) = 41$

$x \leq 1$

$x \geq 2$

$f(1, 4.444) = 40.555$

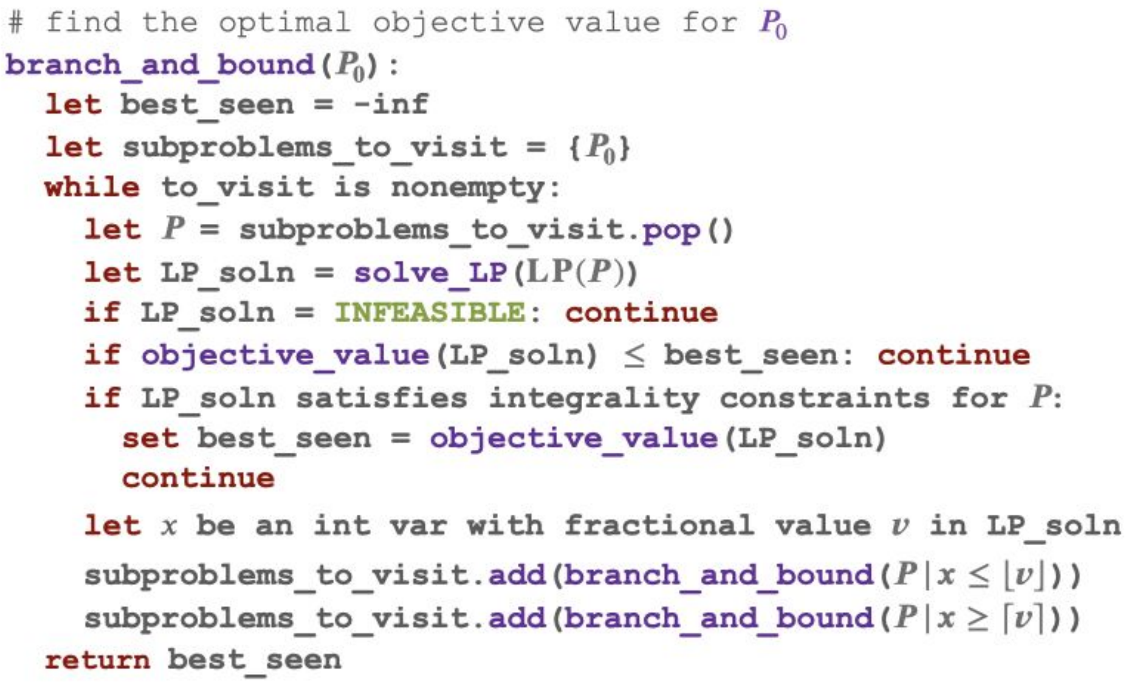
$y \leq 4$

$y \geq 5$

$f(1, 4) = 37$

$f(0, 5) = 40$

Iterative Branch & Bound



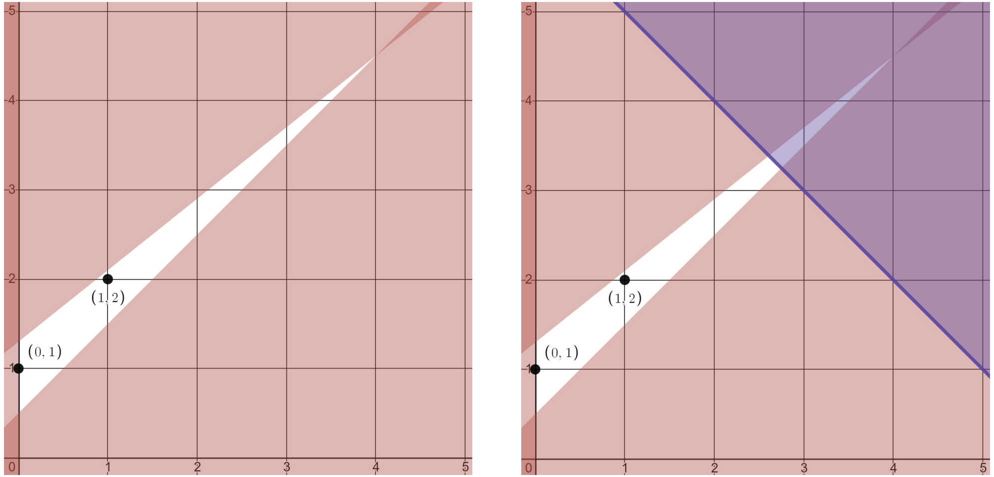
```
# find the optimal objective value for $P_0$
branch_and_bound($P_0$):
  let best_seen = -inf
  let subproblems_to_visit = {$P_0$}
  while to_visit is nonempty:
    let $P$ = subproblems_to_visit.pop()
    let LP_soln = solve_LP($\mathrm{LP}(P)$)
    if LP_soln = INFEASIBLE: continue
    if objective_value(LP_soln) $\leq$ 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($P \mid x \leq \lfloor v \rfloor$))
    subproblems_to_visit.add(branch_and_bound($P \mid x \geq \lceil v \rceil$))
  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.

0.5 oz., $500

Max Weight: 400 oz.

100 oz., $2,000

300 oz., $4,000

1 oz., $5,000

200 oz., $5,000

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!

Max Weight: 300 oz.

100 oz., $2,000

300 oz., $4,000

MIP for 0/1 Knapsack

- MIP formulation is very straightforward:

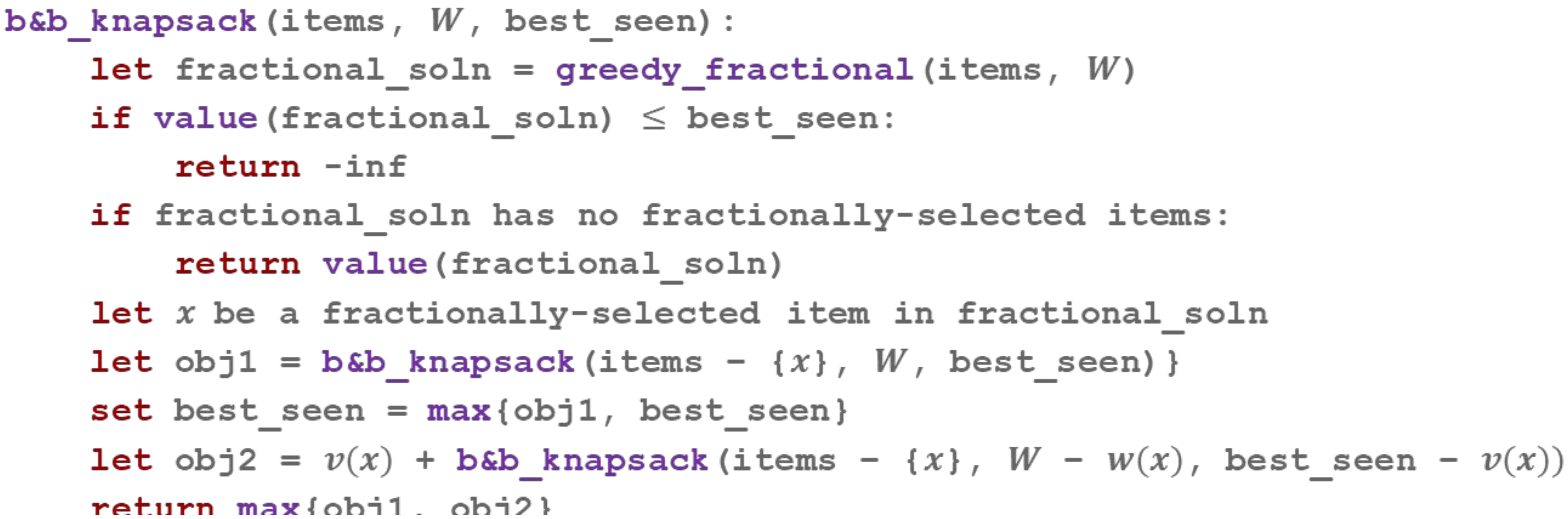
$$\text{maximize} \quad \sum_{i=1}^{n} x_i v_i$$

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

- 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) $\le$ 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}
```