Branch & Bound and Knap

Objectives

- Preform the branch and bound algorithm
- Apply branch and bound to the knapsack probler
- Understand the geometry of the branch and bou

Brief description: In this lab, we will try solving an
branch-and-bound algorithm. We will also see how ac
computation time and effort of the algorithm. Lastly,
and bound algorithm.

In [1]:
```
# imports -- don't forget to run this cell
import pandas as pd
import gilp
from ortools.linear_solver import pywraplp as OR
from gilp.visualize import feasible_integer_pts
```

Part 1: Branch and Bound Algo

Recall that the branch and bound algorithm (in addit
integer programs. Before applying the branch and bc
will begin by reviewing some core ideas. Furthermore
make branch and bound terminate quicker later in th

Q1: What are the different ways a node can be fatho
algorithm? Describe each.

A: A node can be fathomed if there is no better integ
integer program solution

Q2: Suppose you have a maximization integer progr
relaxation. What does the LP-relaxation optimal valu
it is a minimization problem?

A: It is an upper bound in a maximization problem ar

Q3: Assume you have a maximization integer progra
objective function. Now, suppose you are running the

across a node with an optimal value of 44.5. The curr
node? Why or why not?

A: Yes because there is no other integer solution pos

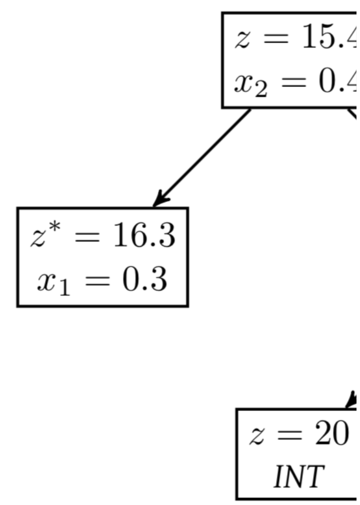
Q4: If the optimal solution to the LP relaxation of the
found an optimal solution to your integer program. E

A: This is because the bound is an integer, thus the l

Q5: If the LP is infeasible, then the IP is infeasible. Ex

A: Because the IP is a feasible solution to the LP

The next questions ask about the following branch ar
integral, the fractional x_i that was used to branch is
denoted *INT*. In the current iteration of branch and bo
*****.



Q6: Can you determine if the integer program this br
or maximixation problem? If so, which is it?

A: Yes, minimization

Hint: For **Q7-8**, you can assume integral coefficents i

Q7: Is the current node (marked z^*) fathomed? Why
constraints should be imposed for each of the next tw

A: It is fathomed

Q8: Consider the nodes under the current node (whe
optimal value of these nodes? Why?

A: They are lower bounds for the IP

Part 2: The Knapsack Problem

In this lab, you will solve an integer program by bran
solved will be a knapsack problem.

Knapsack Problem: We are given a collection of n
weight w_i and a value v_i. In addition, there is a give
maximum value subset of items that has a total weig
brought at most once.

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} v_i x_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} w_i x_i \leq W \\ & 0 \leq x_i \leq 1, \text{inte} \end{aligned}$$

Consider the following data which we import from a (

In [4]:

```
data = pd.read_csv('knapsack_data_1.csv', index_col=0)
data
```

	value	weight
item		
1	50	10
2	30	12
3	24	10
4	14	7
5	12	6
6	10	7
7	40	30

and $W = 18$.

Q9: Are there any items we can remove from our inp
replace `index` with the item number that can be rem
each item could we possibly take?

A: 7 can be dropped because it alone is more than W

In [ ]:

```
# TODO: replace index
data = data.drop(7)
```

Q10: If we remove item 7 from the knapsack, it does
integer program. Explain why.

A: It is bigger than W on its own so it cannot be carri

Q11: Consider removing items i such that $w_i > W$
relaxation's optimal value change?

A: It does not

In **Q10-11**, you should have found that removing the
linear program but does not change the integer prog
the optimal IP and LP values can become smaller. By
terminate sooner.

Recall that a branch and bound node can be fathome

the best feasible integer solution found thus far. Hen
solution as quickly as possible (so that we stop need
construct a good feasible integer solution by a reaso
run the branch and bound procedure.

In designing a heuristic for the knapsack problem, it
weight for each item. We compute this value in the t

In [ ]:
```
data['value per unit weight'] = (data['value'] / data[
data
```

Q12: Design a reasonable heuristic for the knapsack
decent solution to the problem (but is not necessaril

A: Pick the biggest value per weight and then pick th

Q13: Run your heuristic on the data above to compu
heuristic should generate a feasible solution with a v
different heuristic (or talk to your TA!)

A: Got 64

We will now use the branch and bound algorithm to s
define a mathematical model for the linear relaxatio

Q14: Complete the model below.

In [ ]:
```
def Knapsack(table, capacity, integer = False):
    """Model for solving the Knapsack problem.

    Args:
        table (pd.DataFrame): A table indexd by items
        capcity (int): An integer-capacity for the kna
        integer (bool): True if the variables should b
    """
    ITEMS = list(table.index)        # set of items
    v = table.to_dict()['value']     # value for each
    w = table.to_dict()['weight']    # weight for each
    W = capacity                     # capacity of the

    # define model
    m = OR.Solver('knapsack', OR.Solver.CBC_MIXED_INTE
```

```
    # decision variables
    x = {}
    for i in ITEMS:
        if integer:
            x[i] = m.IntVar(0, 1, 'x_%d' % (i))
        else:
            x[i] = m.NumVar(0, 1, 'x_%d' % (i))

    # define objective function here
    m.Maximize(sum(v[i]*x[i] for i in ITEMS))

    # recall that we add constraints to the model usi
    m.Add(W >= sum(w[i] for i in ITEMS))

    return (m, x)  # return the model and the decisio
```

In [ ]:

```
# You do not need to do anything with this cell but ma
def solve(m):
    """Used to solve a model m."""
    m.Solve()

    print('Objective =', m.Objective().Value())
    print('iterations :', m.iterations())
    print('branch-and-bound nodes :',m.nodes())

    return ({var.name() : var.solution_value() for var
```

We can now create a linear relaxation of our knapsac
`x` represents our decision variables.

In [ ]:

```
m, x = Knapsack(data, 18)
```

We can use the next line to solve the model and outp

In [ ]:

```
solve(m)
```

Q15: How does this optimal value compare to the
integer solution?

A: It is larger

Q16: Should this node be fathomed? If not, what var
additional constraints should be imposed for each of

A: It should not be fathomed. x_3 should be branche

After constructing the linear relaxation model using r
constraints. For example, we can add the constraint

In [ ]:
```
m, x = Knapsack(data, 18)
m.Add(x[2] <= 0)
solve(m)
```

NOTE: The line `m, x = Knapsack(data1, 18)` resets th
constraints from branching have to be added each ti

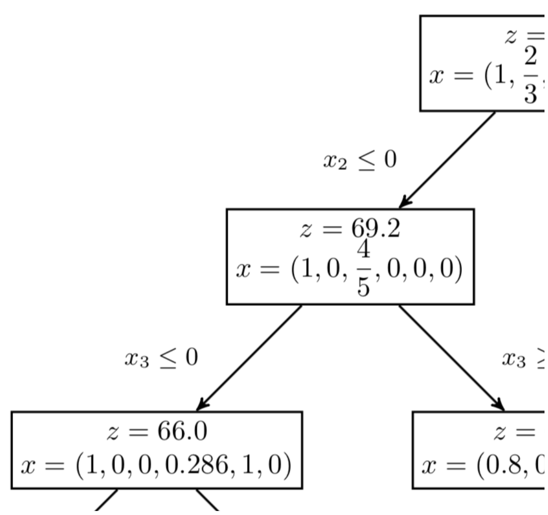
Q17: Use the following cell to compute the optimal v

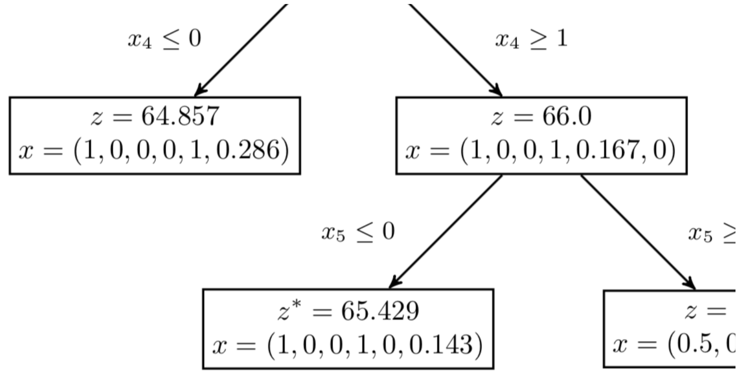
In [ ]:
```
m, x = Knapsack(data, 18)
m.Add(x[2] <= 0)
m.Add(x[2] >= 1)
solve(m)
```

Q18: What was the optimal value? Can this node be
feasible integer solution with value 64.)

A: It was 60 and it can be fathomed because 60 < 64

If we continue running the branch and bound algorith
bound tree below where the z^* indictes the current r





Q19: The node with $z = 64.857$ was fathomed. Why think back to **Q3**)

A: There's no integer between it and 64.

Q20: Finish running branch and bound to find the op each node you solve and indicate if the node was fat to include the constraints further up in the branch ar

In [ ]:
```
# Template
m, x = Knapsack(data, 18)
m.Add(x[2] <= 0)
m.Add(x[2] >= 1)
m.Add(x[3] >= 1)
m.Add(x[3] <= 0)
solve(m)
# fathomed: x_3 <= 0
```

In [ ]:
```
# Template
m, x = Knapsack(data, 18)
m.Add(x[2] <= 0)
m.Add(x[2] >= 1)
m.Add(x[3] >= 1)
m.Add(x[3] <= 0)
m.Add(x[4] >= 1)
m.Add(x[4] <= 0)
solve(m)
# fathomed: x_3 <= 0, x_4 <= 0
```

In [ ]:
```
# Template
m, x = Knapsack(data, 18)
```

```
m.Add(x[2] <= 0)
m.Add(x[2] >= 1)

m.Add(x[3] >= 1)
m.Add(x[3] <= 0)
m.Add(x[4] >= 1)
m.Add(x[4] <= 0)
m.Add(x[5] >= 1)
m.Add(x[5] <= 0)
solve(m)
# fathomed: x 3 <= 0, x 4 <= 0
```

In [ ]:

```
# Template
m, x = Knapsack(data, 18)
m.Add(x[2] <= 0)
m.Add(x[2] >= 1)
m.Add(x[3] >= 1)
m.Add(x[3] <= 0)
m.Add(x[4] >= 1)
m.Add(x[4] <= 0)
m.Add(x[5] >= 1)
m.Add(x[5] <= 0)
m.Add(x[6] >= 1)
m.Add(x[6] <= 0)
solve(m)
```

A: 64

Q21: How many nodes did you have to explore while

A: 10

In the next section, we will think about additional co and bound quicker.

Part 3: Cutting Planes

In general, a cutting plane is an additional constraint relaxation that removes feasible linear solutions but solutions. This is very useful when solving integer pr

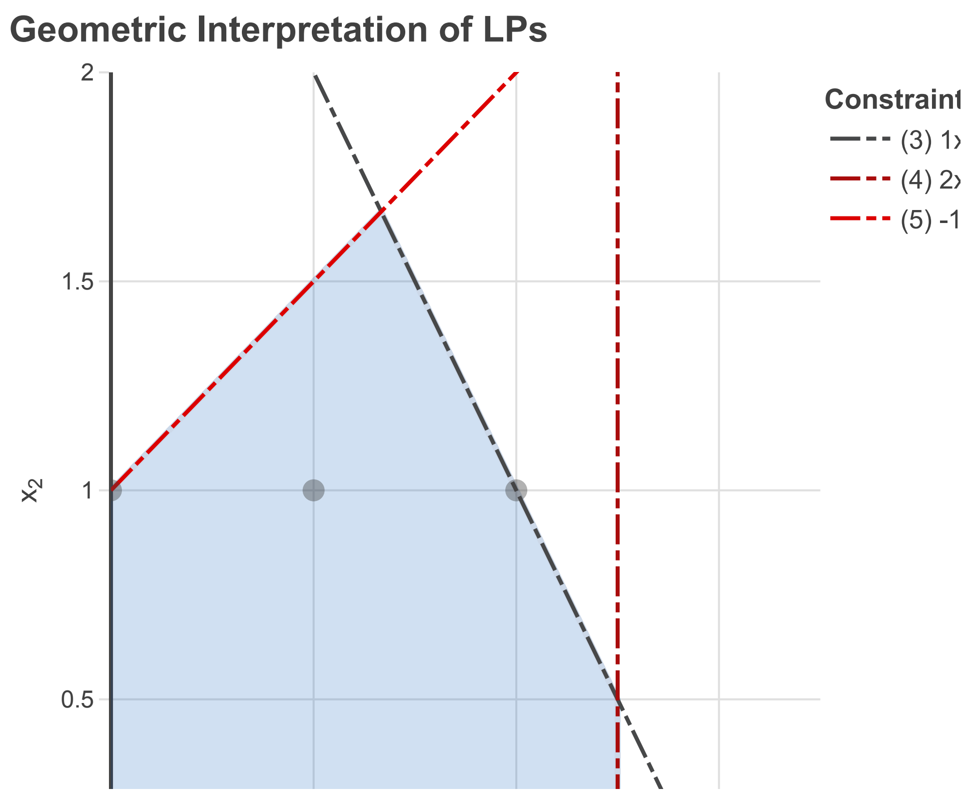
have learned in class have something we call the "in
allows us to ignore the integrality constraint since we
By cleverly adding cutting planes, we strive to remov

Conisder an integer program whose linear program r

$$\begin{aligned} \max \quad & 2x_1 + x \\ \text{s.t.} \quad & x_1 + x_2 \\ & 2x_1 \leq 5 \\ & -x_1 + \\ & x_1, x_2 \geq \end{aligned}$$

We can define this linear program and then visualize
been highlighted.

In [2]:
```
lp = gilp.LP([[1,1],[2,0],[-1,2]],
             [3,5,2],
             [2,1])
fig = gilp.lp_visual(lp)
fig.set_axis_limits([3.5,2])
fig.add_trace(feasible_integer_pts(lp, fig))
fig
```



Q22: List every feasible solution to the integer progr

A: 0,1,2,3,4

Q23: Is the constraint $x_2 \leq 1$ a cutting plane? Why?
become infeasible? What about feasible linear points

A: It is a cutting plane. It removes feasible solutions

Let's add this cutting plane to the LP relaxation!

In [ ]:
```
lp = gilp.LP([[1,1],[2,0],[-1,2],[0,1]],
             [3,5,2,1],
             [2,1])
fig = gilp.lp_visual(lp)
fig.set_axis_limits([3.5,2])
fig.add_trace(feasible_integer_pts(lp, fig))
fig
```

Q24: Is the constraint $x_1 \leq 3$ a cutting plane? Why?

A: no it removes no feasible solutions

```
**Q25:** Can you provide another cutting plane? If s


**A:** x 1 <= 4
```

Let's look at the feasible region after adding the cutt
answers from **Q25**. Notice the optimal solution to the

In [ ]:
```
lp = gilp.LP([[1,1],[2,0],[-1,2],[0,1],[1,0]],
             [3,5,2,1,2],
             [2,1])
fig = gilp.lp_visual(lp)
fig.set_axis_limits([3.5,2])
```

```
fig.add_trace(feasible_integer_pts(lp, fig))
fig
```

Let's try applying what we know about cutting planes
input was $W = 18$ and:

In [5]:
```
data
```

	value	weight
item		
1	50	10
2	30	12
3	24	10
4	14	7
5	12	6
6	10	7
7	40	30

Q26: Look at items 1, 2, and 3. How many of these i
write a new constraint to capture this? If so, please p

A: x_1 + x_2 <= 18, x_2 + x_3 <= 18, x_3+x_1 <=1

Q27: Is the constraint you found in **Q26** a cutting pla
linear program relaxation that is no longer feasible (i

A: yes it cuts off fractional values of each item

Q28: Provide another cutting plane involving items 4
how you derived it.

A:

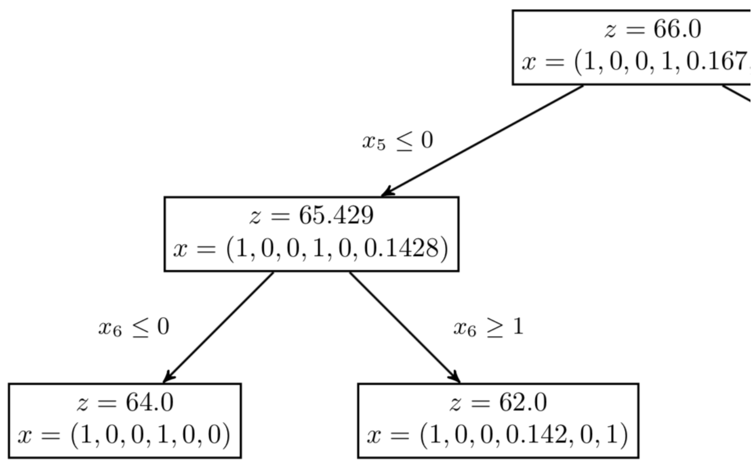
Q29: Add the cutting planes from **Q26** and **Q28** to t
solution in which we take items 1 and 4 and $\frac{1}{6}$ of ite

In [ ]:
```
m, x = Knapsack(data, 18)
# TODO: Add cutting planes here

solve(m)
```

Let's take a moment to pause and reflect on what we
dropped item 7 becuase its weight was greater than
added the constraint $x_7 \leq 0$. This constraint was a c
feasible solutions but no integer ones. By adding the
branch and bound to terminate earlier yet again! So
inspection. However, there are more algorithmic way
now).

If we continue running the branch and bound algorith
bound tree below where the z^* indictes the current r



NOTE: Do not forget about the feasible integer solut

Q30 Finish running branch and bound to find the opt
each node you solve and indicate if the node was fat
the cutting plane constraints should be included in e

In [ ]:
```
# Template
m, x = Knapsack(data, 18)
# Add constraints here


solve(m)
# fathomed?
```

In [ ]:

In [ ]:

A:

Q31: Did you find the same optimal solution? How m
compare to the number you explored previously?

A:

Part 4: Geometry of Branch an

Previously, we used the `gilp` package to viusualize t
functionality to visualize branch and bound. We will g
`lp_visual` and `simplex_visual`, the function `bnb_visua`
is assumed that every decision variable is constraine
visualizations, `bnb_visual` returns a series of figures
Let's look at a small 2D example:

$$\begin{aligned} \max \quad & 5x_1 + 8x_2 \\ \text{s.t.} \quad & x_1 + x_2 \leq \\ & 5x_1 + 9x_2 \\ & x_1, x_2 \geq 0. \end{aligned}$$

In [ ]:
```
nodes = gilp.bnb_visual(gilp.examples.STANDARD_2D_IP)
```

In [ ]:
```
nodes[0].show()
```

Run the cells above to generate a figure for each noc
see the LP relaxation on the left and the root of the b
simplex path and isoprofit slider are also present.

Q32: Recall the root of a branch and bound tree is th
optimal solution? (Hint: Use the objective slider and l

A:

Q33: Assume that we always choose the variable wi
are multiple options. Write down (in full) each of the

A:

Q34: Draw the feasible region to each of the LPs fror

A:

Run the following cell to see if the picture you drew i

In [ ]:
```
nodes[1].show()
```

The outline of the original LP relaxation is still shown
some of the fractional feasible solutions, we now hav
one is the feasible region associated with the current
branch and bound tree. The unexplored nodes in the

Q35: Which feasible solutions to the LP relaxation ar

A:

Q36: At the current (dark) node, what constraints wi
original LP relaxation be broken into?

A:

In [ ]:
```
nodes[2].show()
```

Q37: What is the optimal solution at the current (dar
branch? Explain.

A:

Q38: Recall shaded nodes have been explored and t
shaded darker) correspond to the current node and it
been explored. How many nodes have not yet been e

A:

Q39: How many nodes have a degree of one in the b
only connected to one edge). These nodes are called
the leaf nodes and the remaining feasible region?

A:

In [ ]:
```
# Show the next two iterations of the branch and bounc
nodes[3].show()
nodes[4].show()
```

```
**Q40:** At the current (dark) node, we added the co
fractional solutions $1 < x_1 < 2$ not eliminated fo

**A:**
```

In [ ]:
```
# Show the next three iterations of the branch and bou
nodes[5].show()
nodes[6].show()
nodes[7].show()
```

Q41: What constraints are enforced at the current (c
solutions at this node?

A:

In [ ]:
```
nodes[8].show()
```

Q42: Are we done? If so, what nodes are fathomed a

A:

Let's look at branch and bound visualization for an in

In [ ]:
```
nodes = gilp.bnb_visual(gilp.examples.VARIED_BRANCHING
```

In [ ]:
```
# Look at the first 3 iterations
nodes[0].show()
nodes[1].show()
nodes[2].show()
```

Let's fast-forward to the final iteration of the branch

In [ ]:
```
nodes[-1].show()
```

Q43: Consider the feasible region that looks like a re
origin. What node does it correspond to in the tree? \

A:

Q44: How many branch and bound nodes did we exp
many branch and bound nodes would we have explo
solution before starting branch and bound?

A:

Bonus: Branch and Bound for K

Consider the following example:

item	value	weight
1	2	1
2	9	3
3	6	2

The linear program formulation will be:

$$\begin{aligned} \max \quad & 2x_1 + 9x_2 + \\ \text{s.t.} \quad & 1x_1 + 3x_2 + \\ & x_1, x_2, x_3 \geq \end{aligned}$$

In gilp, we can define this lp as follows:

In [ ]:

```
lp = gilp.LP([[1,3,2]],
             [10],
             [2,9,6])

for fig in gilp.bnb_visual(lp):
    fig.show()
```