Branch & Bound and Knapsack Lab

Objectives

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

Brief description: In this lab, we will try solving an example of a knapsack problem with the branch-and-bound algorithm. We will also see how adding a cutting plane helps in reducing the computation time and effort of the algorithm. Lastly, we will explore the geometry of the branch and bound algorithm.

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

Part 1: Branch and Bound Algorithm

Recall that the branch and bound algorithm (in addition to the simplex method) allows us to solve integer programs. Before applying the branch and bound algorithm to the knapsack problem, we will begin by reviewing some core ideas. Furthermore, we will identify a helpful property that will make branch and bound terminate quicker later in the lab!

Q1: What are the different ways a node can be fathomed during the branch and bound algorithm? Describe each.

A: A node can be fathomed when we don't need to explore it any further. This can happen if a solution is less optimal, if you find an integer solution, and if for the program with extra constraint, there is no solution.

Q2: Suppose you have a maximization integer program and you solve its linear program relaxation. What does the LP-relaxation optimal value tell you about the IP optimal value? What if it is a minimization problem?

A: The LP relaxation optimal value is the upper bound for the maximization IP, and tells that the IP optimal value is the lower bound. If this were a minimization problem, then the IP optimal value would be the upper bound, and the LP optimal value would be the lower bound.

Q3: Assume you have a maximization integer program with all integral coefficents in the objective function. Now, suppose you are running the branch and bound algorithm and come across a node with an optimal value of 44.5. The current incumbent is 44. Can you fathom this node? Why or why not?

A: You can't fathom this node because you can explore it further. Because this is a maximization problem, 44.5 could be 45, so we would need to explore that.

Q4: If the optimal solution to the LP relaxation of the original program is integer, then you have found an optimal solution to your integer program. Explain why this is true.

A: Because we're trying to find an optimal integer solution, and the LP relaxation solution is optimal, if that solution is integer, then that would also be the optimal soltuion to the IP. The reason that we can explore a node is because the solution is fractional, and we can split it up into different nodes. But if the solution is integer we cannot split it up further.

Q5: If the LP is infeasible, then the IP is infeasible. Explain why this is true.

A: When we go through each branch, we solve an LP, so if the LP is infeasible, we cannot possible find an IP solution, so the IP would also be infeasible.

The next questions ask about the following branch and bound tree. If the solution was not integral, the fractional x_i that was used to branch is given. If the solution was integral, it is denoted *INT*. In the current iteration of branch and bound, you are looking at the node with the *****.

Q6: Can you determine if the integer program this branch and bound tree is for is a minimization or maximixation problem? If so, which is it?

A: This is a minimization problem because as additional constraints are placed, the z value increased.

Hint: For **Q7-8**, you can assume integral coefficents in the objective function.

Q7: Is the current node (marked z^*) fathomed? Why or why not? If not, what additional constraints should be imposed for each of the next two nodes?

A: It would not be fathomed because 16.3 is better than 17.8 and so we can explore z* to find a better solution than the one found from z=17.8. The additional constraints imposed would be x≥1 and x≤0.

Q8: Consider the nodes under the current node (where $z = 16.3$). What do you know about the optimal value of these nodes? Why?

A: The optimal value of these nodes would either be the same as 16.3 or worse because addition constraints are being added.

Part 2: The Knapsack Problem

In this lab, you will solve an integer program by branch and bound. The integer program to be solved will be a knapsack problem.

Knapsack Problem: We are given a collection of n items, where each item $i = 1, \ldots, n$ has a weight w_i and a value v_i. In addition, there is a given capacity W, and the aim is to select a maximum value subset of items that has a total weight at most W. Note that each item can be brought at most once.

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

Consider the following data which we import from a CSV file:

In [2]:
```
data = pd.read_csv('knapsack_data_1.csv', index_col=0)
data
```

Out[2]:

item	value	weight
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 input to simplify this problem? Why? If so, replace `index` with the item number that can be removed in the code below. Hint: how many of each item could we possibly take?

A: We can remove item 7 because the weught of that item has a bigger value than the weight constraint of 18.

In [8]:
```
#TODO: replace 7
data = data.drop(7)
```

Q10: If we remove item 7 from the knapsack, it does not change the optimal solution to the integer program. Explain why.

A: It would not change the optimal solution to the IP because it would never be included in the optimal solution because it does not match the weight constraint.

Q11: Consider removing items i such that $w_i > W$ from a knapsack input. How does the LP relaxation's optimal value change?

A: It doesn't change because those items would not have been included in the first place.

In **Q10-11**, you should have found that removing these items removes feasible solutions from the linear program but does not change the integer program. This is desirable as the gap between the optimal IP and LP values can become smaller. By adding this step, branch and bound may terminate sooner.

Recall that a branch and bound node can be fathomed if its bound is no better than the value of the best feasible integer solution found thus far. Hence, it helps to have a good feasible integer solution as quickly as possible (so that we stop needless work). To do this, we can first try to construct a good feasible integer solution by a reasonable heuristic algorithm before starting to run the branch and bound procedure.

In designing a heuristic for the knapsack problem, it is helpful to think about the value per unit weight for each item. We compute this value in the table below.

In [9]:
```
data['value per unit weight'] = (data['value'] / data['weight']).round(2)
data
```

Out[9]:

item	value	weight	value per unit weight
1	50	10	5.00
2	30	12	2.50
3	24	10	2.40
4	14	7	2.00
5	12	6	2.00
6	10	7	1.43

Q12: Design a reasonable heuristic for the knapsack problem. Note a heuristic aims to find a decent solution to the problem (but is not necessarily optimal).

A: Take item with maximum value and take the weight value of that item as the starting point, and add items with the next largest value that have a weight value that when added to the first stil fits within the weight constraint.

Q13: Run your heuristic on the data above to compute a good feasible integer solution. Your heuristic should generate a feasible solution with a value of 64 or better. If it does not, try a different heuristic (or talk to your TA!)

A: 50+14 = 64

We will now use the branch and bound algorithm to solve this knapsack problem! First, let us define a mathematical model for the linear relaxation of the knapsack problem.

Q14: Complete the model below.

In [21]:
```
def Knapsack(table, capacity, integer = False):
    """Model for solving the Knapsack problem.

    Args:
        table (pd.DataFrame): A table indexd by items with a column for value an
        capcity (int): An integer-capacity for the knapsack
        integer (bool): True if the variables should be integer. False otherwise
    """
```

```
    ITEMS = list(table.index)          # set of items
    v = table.to_dict()['value']       # value for each item
    w = table.to_dict()['weight']      # weight for each item
    W = capacity                       # capacity of the knapsack

    # define model
    m = OR.Solver('knapsack', OR.Solver.CBC_MIXED_INTEGER_PROGRAMMING)

    # 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))

    # TODO: Add a constraint that enforces that weight must not exceed capacity
    # recall that we add constraints to the model using m.Add()
    m.Add(sum(w[i]*x[i] for i in ITEMS) <= W)

    return (m, x)  # return the model and the decision variables
```

In [22]:

```
# You do not need to do anything with this cell but make sure you run it!
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 in m.variables()})
```

We can now create a linear relaxation of our knapsack problem. Now, `m` represents our model and `x` represents our decision variables.

In [23]:

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

We can use the next line to solve the model and output the solution

In [24]:

```
solve(m)
```

```
Objective = 70.0
iterations : 0
branch-and-bound nodes : 0
```

Out[24]:

```
{'x_1': 1.0,
 'x_2': 0.6666666666666667,
 'x_3': 0.0,
 'x_4': 0.0,
 'x_5': 0.0,
 'x_6': 0.0}
```

Q15: How does this optimal value compare to the value you found using the heuristic integer solution?

A: This optimal value is better than the value I found using the heurisitc integer solution.

Q16: Should this node be fathomed? If not, what variable should be branched on and what additional constraints should be imposed for each of the next two nodes?

A: No this node should not be fathomed. The variable that should be branched on is x2 because it does not have an integer value. The next two nodes would have the constraints x2≤0 and x2≥1.

After constructing the linear relaxation model using `Knapsack(data1, 18)` we can add additional constraints. For example, we can add the constraint $x_2 \leq 0$ and solve it as follows:

In [26]:

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

```
Objective = 69.2
iterations : 0
branch-and-bound nodes : 0
```

Out[26]:

```
{'x_1': 1.0, 'x_2': 0.0, 'x_3': 0.8, 'x_4': 0.0, 'x_5': 0.0, 'x_6': 0.0}
```

NOTE: The line `m, x = Knapsack(data1, 18)` resets the model `m` to the LP relaxation. All constraints from branching have to be added each time.

Q17: Use the following cell to compute the optimal value for the other node you found in **Q16**.

In [27]:

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

```
Objective = 60.0
iterations : 0
branch-and-bound nodes : 0
```

Out[27]:

```
{'x_1': 0.6000000000000001,
 'x_2': 1.0,
 'x_3': 0.0,
 'x_4': 0.0,
 'x_5': 0.0,
 'x_6': 0.0}
```

Q18: What was the optimal value? Can this node be fathomed? Why? (Hint: In **Q13**, you found a feasible integer solution with value 64.)

A: The optimal value for this node is 60 and it can be fathomed because it has a value of less than 64, and 64 was the minimum value that the optimal solution should have.

If we continue running the branch and bound algorithm, we will eventually reach the branch and bound tree below where the z^* indictes the current node we are looking at.

Q19: The node with $z = 64.857$ was fathomed. Why are we allowed to fathom this node? (Hint: think back to **Q3**)

A: Because exploring this node further would give a value of less than 64.857 and since out starting value was 64, exploring this node would give a worse optimal value.

Q20: Finish running branch and bound to find the optimal integer solution. Use a separate cell for each node you solve and indicate if the node was fathomed with a comment. (Hint: Don't forget to include the constraints further up in the branch and bound tree.)

In [31]:
```
# Template
m, x = Knapsack(data, 18)
m.Add(x[2] <= 0)
m.Add(x[3] <= 0)
m.Add(x[4] >= 1)
m.Add(x[5] <= 0)
m.Add(x[6] <= 0)

solve(m)
# fathomed?
```

```
Objective = 64.0
iterations : 0
branch-and-bound nodes : 0
```

Out[31]:
```
{'x_1': 1.0, 'x_2': 0.0, 'x_3': 0.0, 'x_4': 1.0, 'x_5': 0.0, 'x_6': 0.0}
```

In [32]:
```
# Template
m, x = Knapsack(data, 18)
m.Add(x[2] <= 0)
m.Add(x[3] <= 0)
m.Add(x[4] >= 1)
m.Add(x[5] <= 0)
m.Add(x[6] >= 1)

solve(m)
# fathomed?
```

```
Objective = 44.0
iterations : 0
branch-and-bound nodes : 0
```

Out[32]:
```
{'x_1': 0.4, 'x_2': 0.0, 'x_3': 0.0, 'x_4': 1.0, 'x_5': 0.0, 'x_6': 1.0}
```

In [ ]:
```
```

A:

Q21: How many nodes did you have to explore while running the branch and bound algorithm?

A: I explored 5 nodes while running this algorithm.

In the next section, we will think about additional constraints we can add to make running branch and bound quicker.

Part 3: Cutting Planes

In general, a cutting plane is an additional constraint we can add to an integer program's linear relaxation that removes feasible linear solutions but does not remove any integer feasible solutions. This is very useful when solving integer programs! Recall many of the problems we have learned in class have something we call the "integrality property". This is useful because it allows us to ignore the integrality constraint since we are garunteed to reach an integral solution. By cleverly adding cutting planes, we strive to remove feasible linear solutions (without removing any integer feasible solutions) such that the optimal solution to the linear relaxation is integral!

Conisder an integer program whose linear program releaxation is

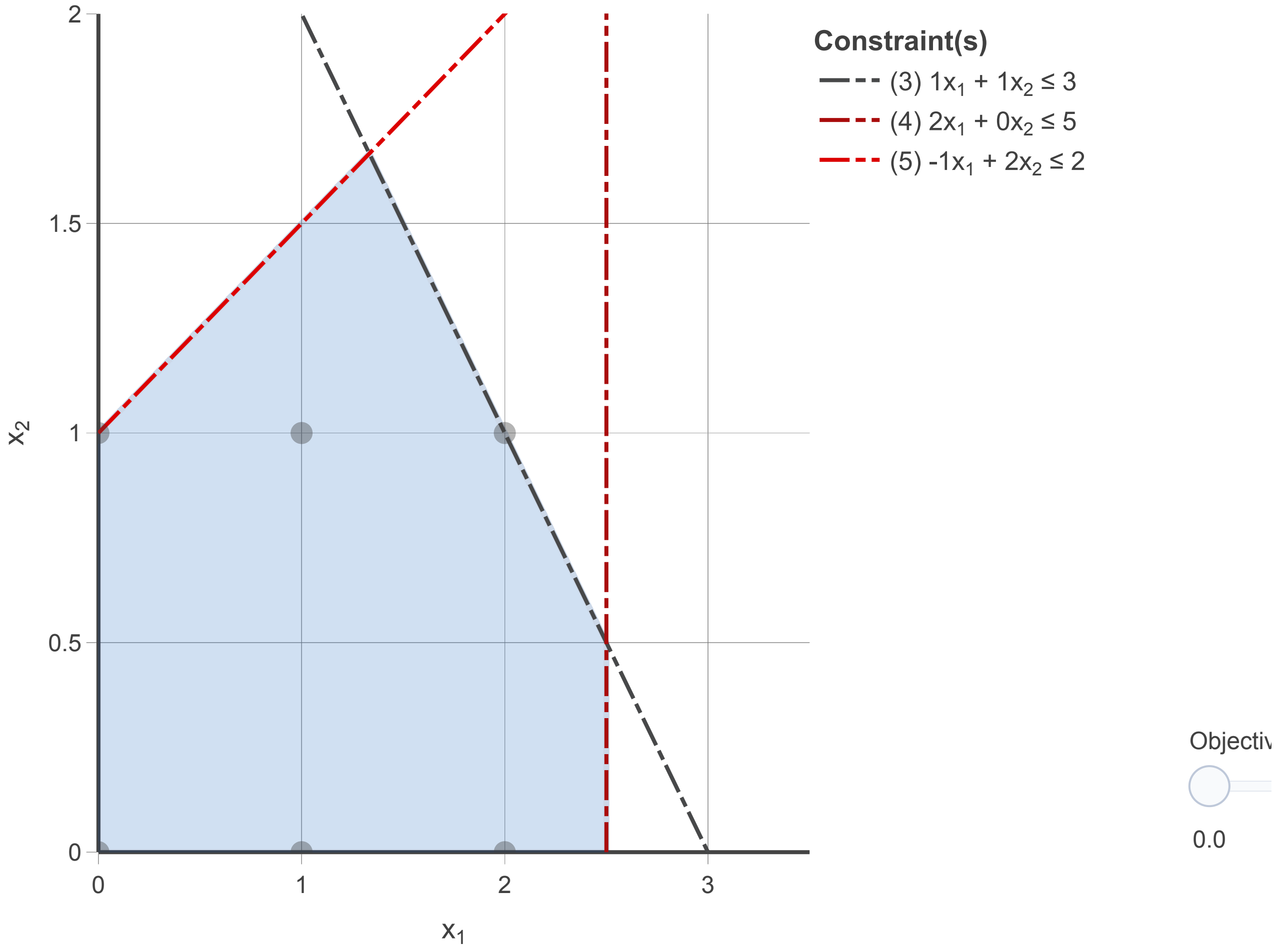
$$\begin{aligned} \max \quad & 2x_1 + x_2 \\ \text{s.t.} \quad & x_1 + x_2 \leq 3 \\ & 2x_1 \leq 5 \\ & -x_1 + 2x_2 \leq 2 \\ & x_1, x_2 \geq 0 \end{aligned}$$

We can define this linear program and then visualize its feasible region. The integer points have been highlighted.

In [33]:

```
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
```

Geometric Interpretation of LPs



Q22: List every feasible solution to the integer program.

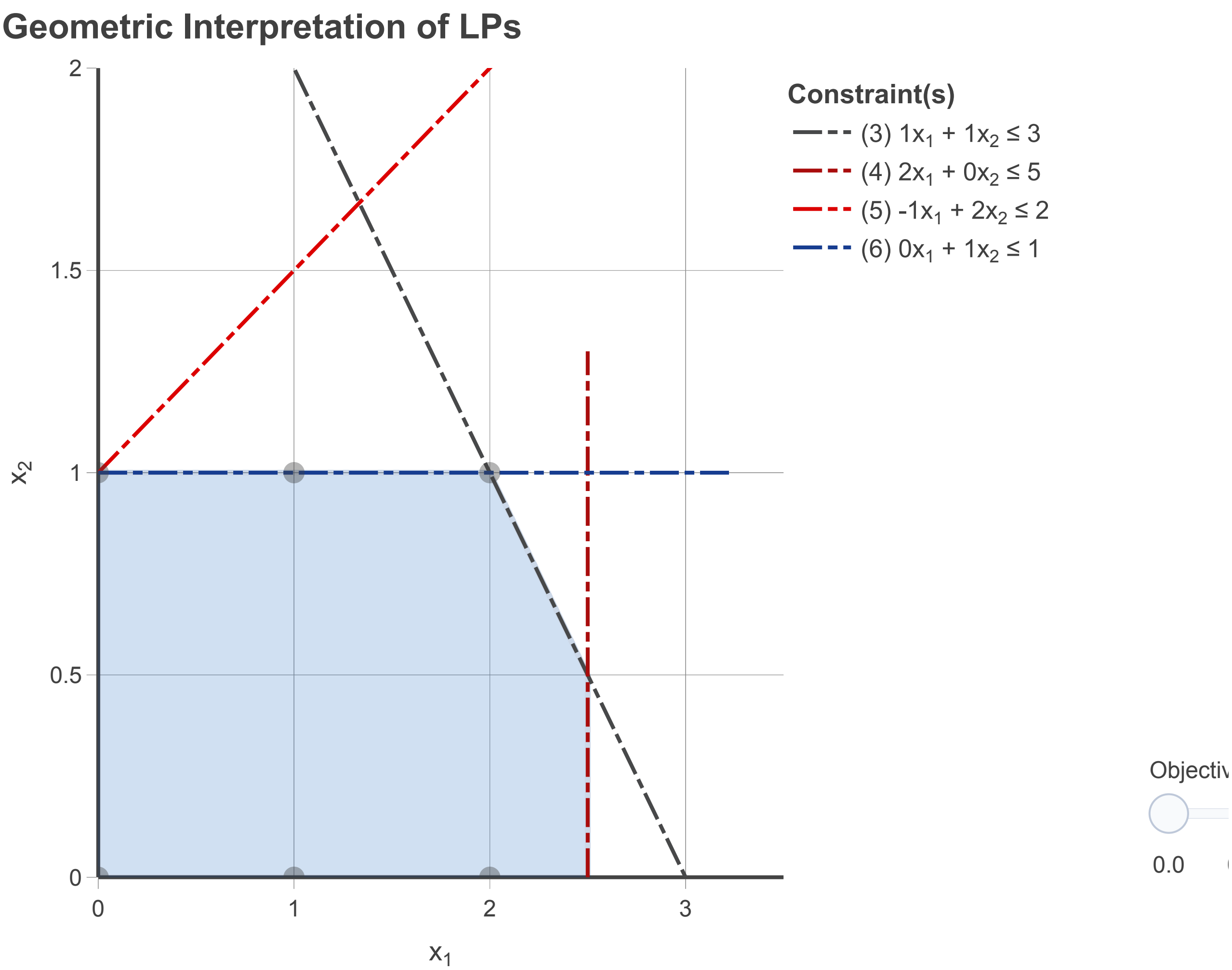
A: (0,0), (1,0), (2,0), (0,1), (1,1), (2,1)

Q23: Is the constraint $x_2 \leq 1$ a cutting plane? Why? (Hint: Would any feasible integer points become infeasible? What about feasible linear points?)

A: This constraint is a cutting plane because the optimal LP solution is now infeasible, but none of the integer points are infeasible.

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

In [34]:
```
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: It is not a cutting plane because it is not eliminating any feasible LP solutions.

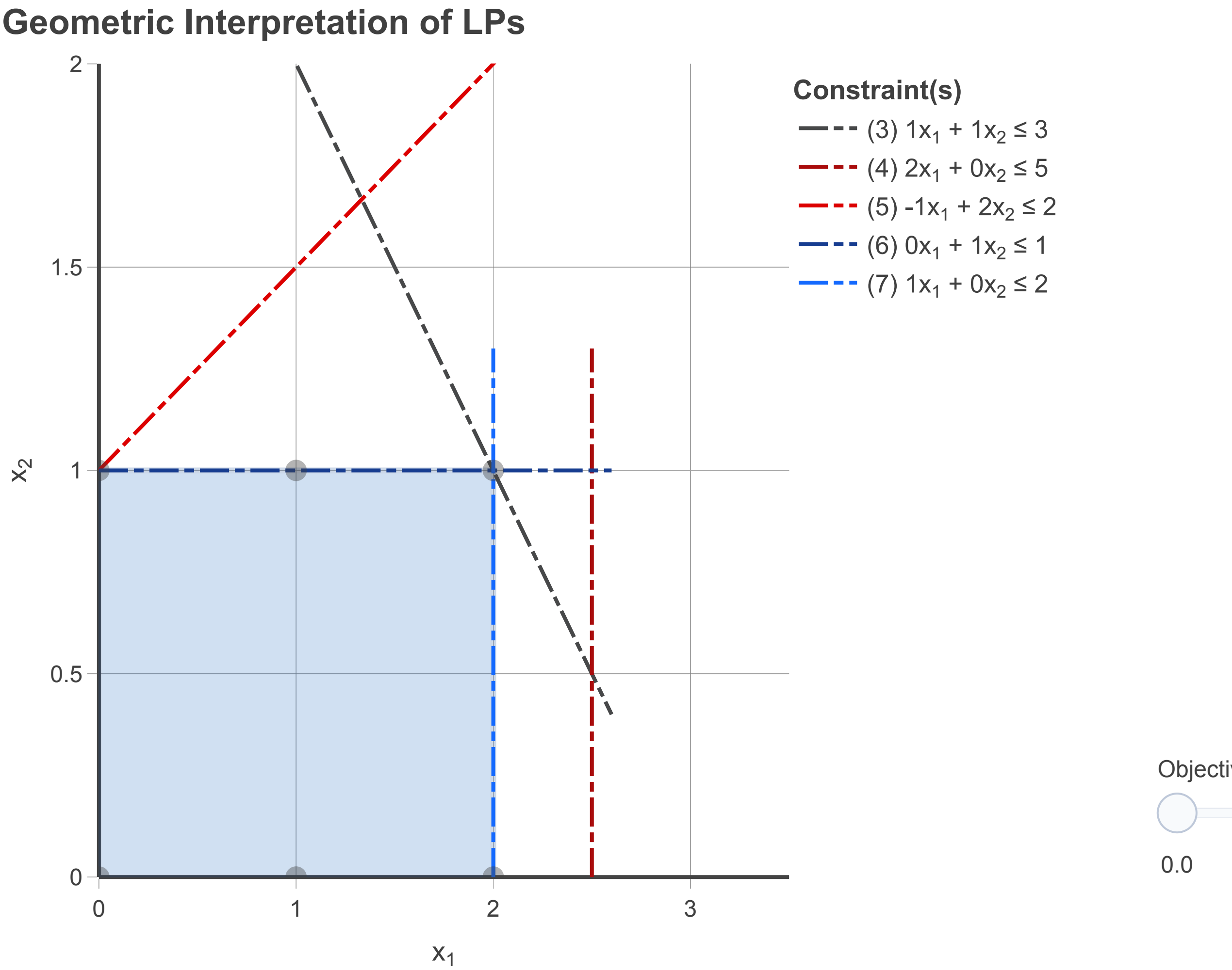
Q25: Can you provide another cutting plane? If so, what is it?

A: x1≤2

Let's look at the feasible region after adding the cutting plane from **Q23** and one of the possible answers from **Q25**. Notice the optimal solution to the LP relaxation is now integral!

In [35]:
```
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 to the knapsack problem! Again, recall our input was $W = 18$ and:

In [36]:
```
data
```

Out[36]:

	value	weight	value per unit weight
item			
1	50	10	5.00
2	30	12	2.50
3	24	10	2.40
4	14	7	2.00
5	12	6	2.00
6	10	7	1.43

Q26: Look at items 1, 2, and 3. How many of these items can we take simultaneously? Can you write a new constraint to capture this? If so, please provide it.

A: You can only take one, because taking two would not satisfy the weight constraint. x1+x2+x3≤1

Q27: Is the constraint you found in **Q26** a cutting plane? If so, provide a feasible solution to the linear program relaxation that is no longer feasible (i.e. a point the constraint *cuts off*).

A: It is a cutting plane because it cuts off linear solutions.

Q28: Provide another cutting plane involving items 4,5 and 6 for this integer program. Explain how you derived it.

A: x4+x5+x6≤2 this is because only two items can be chosen due to the weight constraint.

Q29: Add the cutting planes from **Q26** and **Q28** to the model and solve it. You should get a solution in which we take items 1 and 4 and $\frac{1}{6}$ of item 5 with an objective value of 66.

In [38]:
```
m, x = Knapsack(data, 18)
m.Add(x[1] + x[2] + x[3] <= 1)
m.Add(x[4] + x[5] + x[6] <= 2)
solve(m)
```

```
Objective = 66.0
iterations : 0
branch-and-bound nodes : 0
```

Out[38]:
```
{'x_1': 1.0,
 'x_2': 0.0,
 'x_3': 0.0,
 'x_4': 1.0,
 'x_5': 0.16666666666666696,
 'x_6': 0.0}
```

Let's take a moment to pause and reflect on what we are doing. Recall from **Q9-11** that we dropped item 7 becuase its weight was greater than the capcity of the knapsack. Essentially we added the constraint $x_7 \leq 0$. This constraint was a cutting plane! It eliminated some linear feasible solutions but no integer ones. By adding these two new cutting planes, we can get branch and bound to terminate earlier yet again! So far, we have generated cutting planes by inspection. However, there are more algorithmic ways to identify them (which we will ignore for now).

If we continue running the branch and bound algorithm, we will eventually reach the branch and bound tree below where the z^* indictes the current node we are looking at.

NOTE: Do not forget about the feasible integer solution our heuristic gave us with value 64.

Q30 Finish running branch and bound to find the optimal integer solution. Use a separate cell for each node you solve and indicate if the node was fathomed with a comment. Hint: Don't forget the cutting plane constraints should be included in every node of the branch and bound tree.

In [40]:
```
# Template
m, x = Knapsack(data, 18)
m.Add(x[1] + x[2] + x[3] <= 1)
m.Add(x[4] + x[5] + x[6] <= 2)
```

```
m.Add(x[5] >= 1)

solve(m)
# fathomed? no
```

```
Objective = 66.0
iterations : 0
branch-and-bound nodes : 0
```

Out[40]:
```
{'x_1': 1.0,
 'x_2': 0.0,
 'x_3': 0.0,
 'x_4': 0.28571428571428586,
 'x_5': 1.0,
 'x_6': 0.0}
```

In [41]:
```
# Template
m, x = Knapsack(data, 18)
m.Add(x[1] + x[2] + x[3] <= 1)
m.Add(x[4] + x[5] + x[6] <= 2)
m.Add(x[4] >= 1)

solve(m)
# fathomed? no
```

```
Objective = 66.0
iterations : 0
branch-and-bound nodes : 0
```

Out[41]:
```
{'x_1': 1.0,
 'x_2': 0.0,
 'x_3': 0.0,
 'x_4': 1.0,
 'x_5': 0.16666666666666674,
 'x_6': 0.0}
```

In [42]:
```
# Template
m, x = Knapsack(data, 18)
m.Add(x[1] + x[2] + x[3] <= 1)
m.Add(x[4] + x[5] + x[6] <= 2)
m.Add(x[4] <= 0)

solve(m)
# fathomed? no
```

```
Objective = 64.85714285714286
iterations : 0
branch-and-bound nodes : 0
```

Out[42]:
```
{'x_1': 1.0,
 'x_2': 0.0,
 'x_3': 0.0,
 'x_4': 0.0,
 'x_5': 1.0,
 'x_6': 0.28571428571428586}
```

A: Optimal integer solution is 64.

Q31: Did you find the same optimal solution? How many nodes did you explore? How did this compare to the number you explored previously?

A: Yes we found the same optimal solution. We explored 3 nodes.

Part 4: Geometry of Branch and Bound

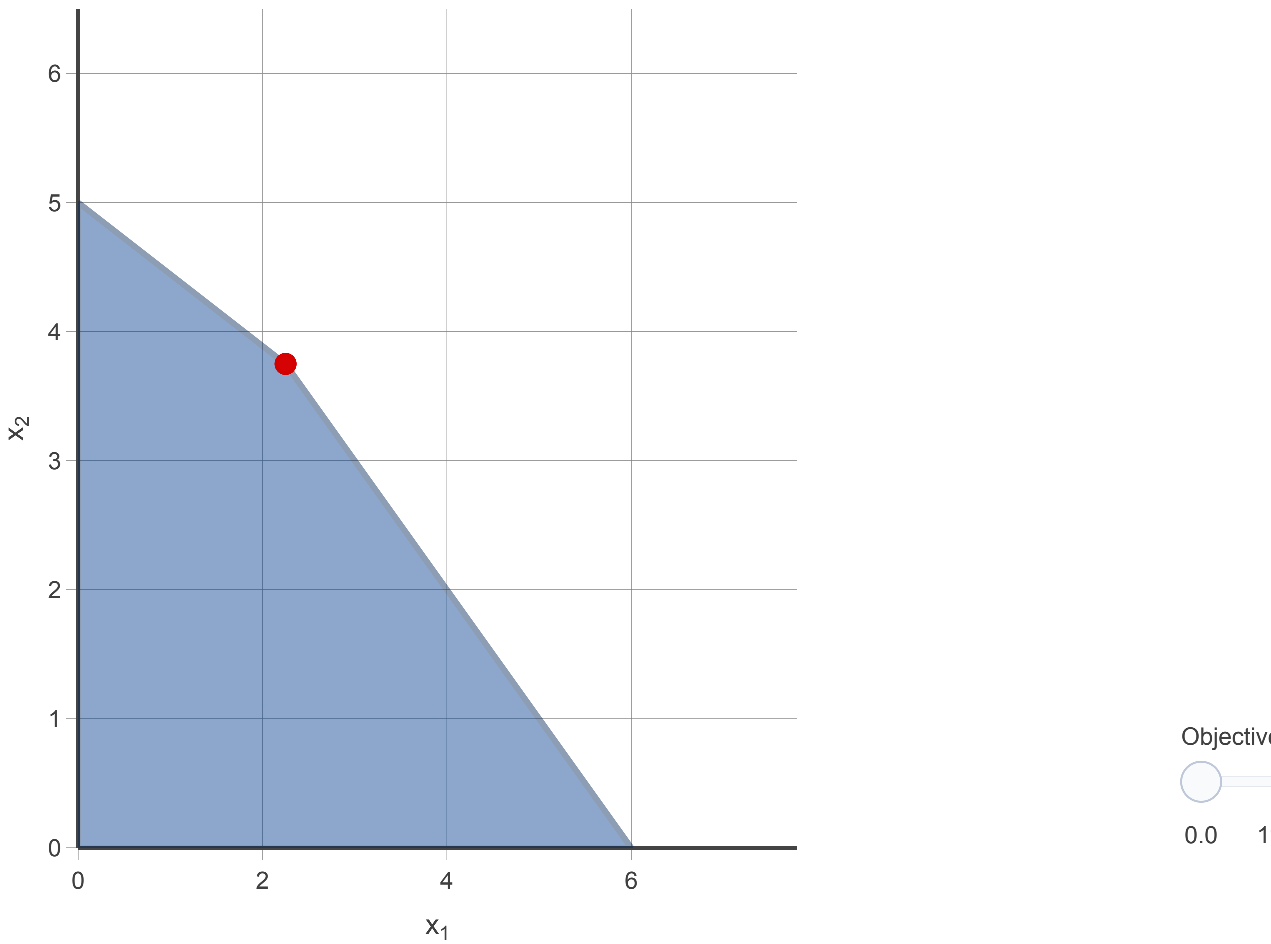
Previously, we used the `gilp` package to viusualize the simplex algorithm but it also has the functionality to visualize branch and bound. We will give a quick overview of the tool. Similar to `lp_visual` and `simplex_visual`, the function `bnb_visual` takes an `LP` and returns a visualization. It is assumed that every decision variable is constrained to be integer. Unlike previous visualizations, `bnb_visual` returns a series of figures for each node of the branch and bound tree. Let's look at a small 2D example:

$$\begin{aligned} \max \quad & 5x_1 + 8x_2 \\ \text{s.t.} \quad & x_1 + x_2 \leq 6 \\ & 5x_1 + 9x_2 \leq 45 \\ & x_1, x_2 \geq 0, \quad \text{integral} \end{aligned}$$

In [43]:
```
nodes = gilp.bnb_visual(gilp.examples.STANDARD_2D_IP)
```

In [44]:
```
nodes[0].show()
```

Geometric Interpretation of LPs



Run the cells above to generate a figure for each node and view the first node. At first, you will see the LP relaxation on the left and the root of the branch and bound tree on the right. The simplex path and isoprofit slider are also present.

Q32: Recall the root of a branch and bound tree is the unaltered LP relaxation. What is the optimal solution? (Hint: Use the objective slider and hover over extreme points).

A: The optimal solution is 41.25.

Q33: Assume that we always choose the variable with the minimum index to branch on if there are multiple options. Write down (in full) each of the LPs we get after branching off the root node.

A: We branch on both x1 and x2. x1≤2 x1≥3 x2≤3 and x2≥4

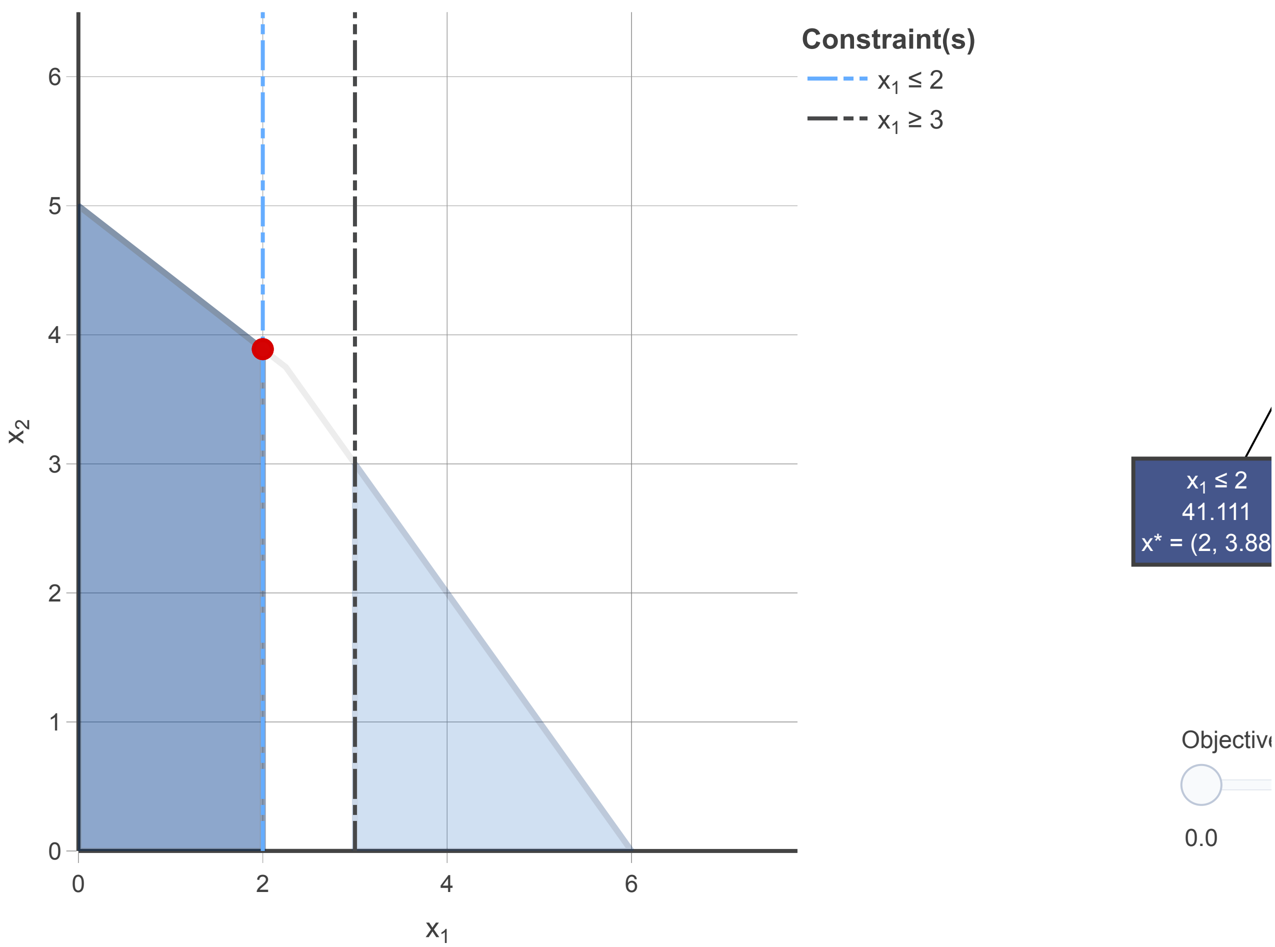
Q34: Draw the feasible region to each of the LPs from **Q33** on the same picture.

A: Drew the region on paper

Run the following cell to see if the picture you drew in **Q34** was correct.

In [45]:
```
nodes[1].show()
```

Geometric Interpretation of LPs



The outline of the original LP relaxation is still shown on the left. Now that we have eliminated some of the fractional feasible solutions, we now have 2 feasible regions to consider. The darker one is the feasible region associated with the current node which is also shaded darker in the branch and bound tree. The unexplored nodes in the branch and bound tree are not shaded in.

Q35: Which feasible solutions to the LP relaxation are removed by this branch?

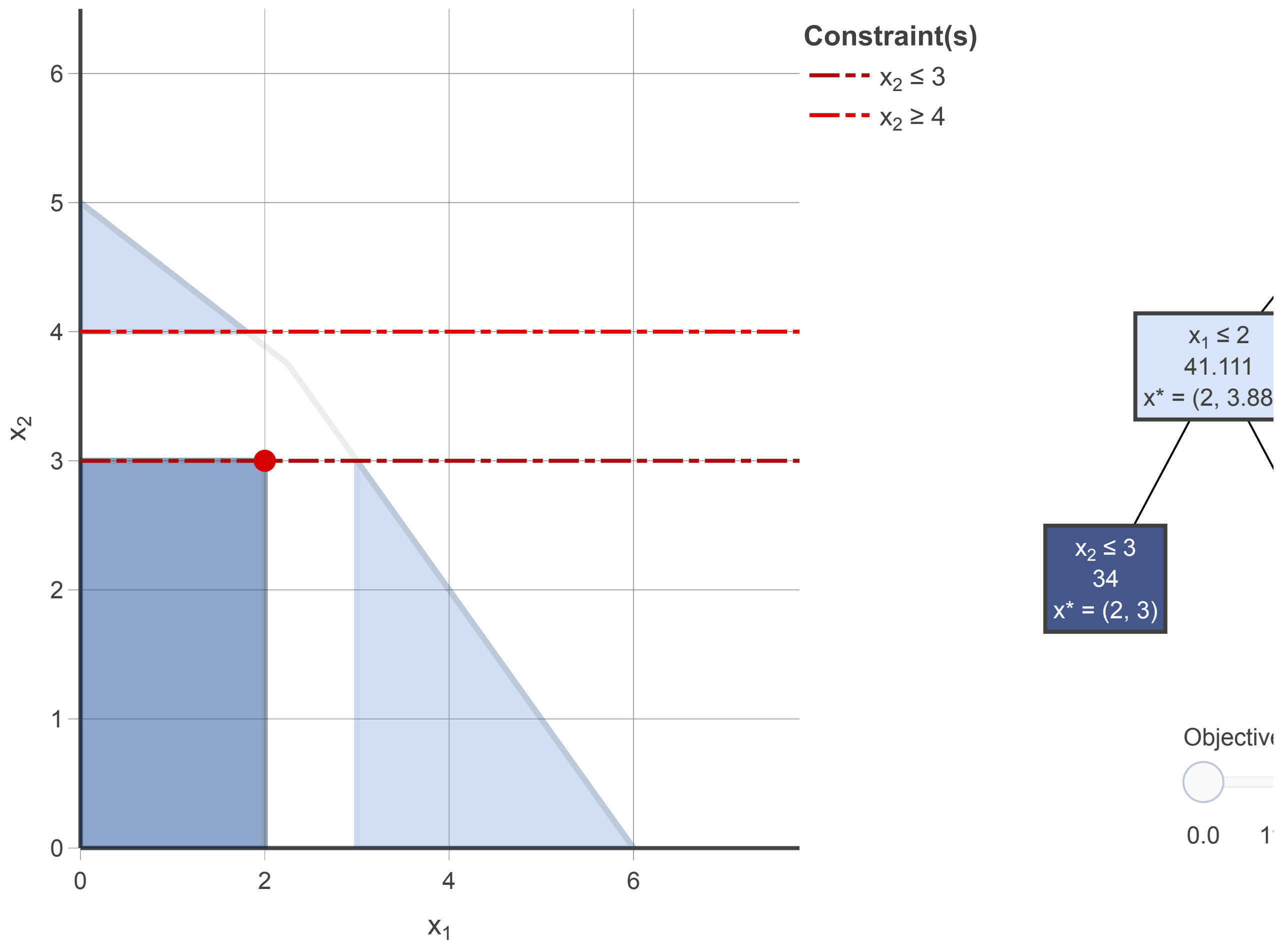
A: The feasible solutions that are removed are when 2≤x1≤3

Q36: At the current (dark) node, what constraints will we add? How many feasible regions will the original LP relaxation be broken into?

A: x2≤3 and x2≥4

In [46]:
```
nodes[2].show()
```

Geometric Interpretation of LPs



Q37: What is the optimal solution at the current (dark) node? Do we have to further explore this branch? Explain.

A: The current optimal solution is 34. We do not have to explore this branch further because it is an integer solution.

Q38: Recall shaded nodes have been explored and the node shaded darker (and feasible region shaded darker) correspond to the current node and its feasible region. Nodes not shaded have not been explored. How many nodes have not yet been explored?

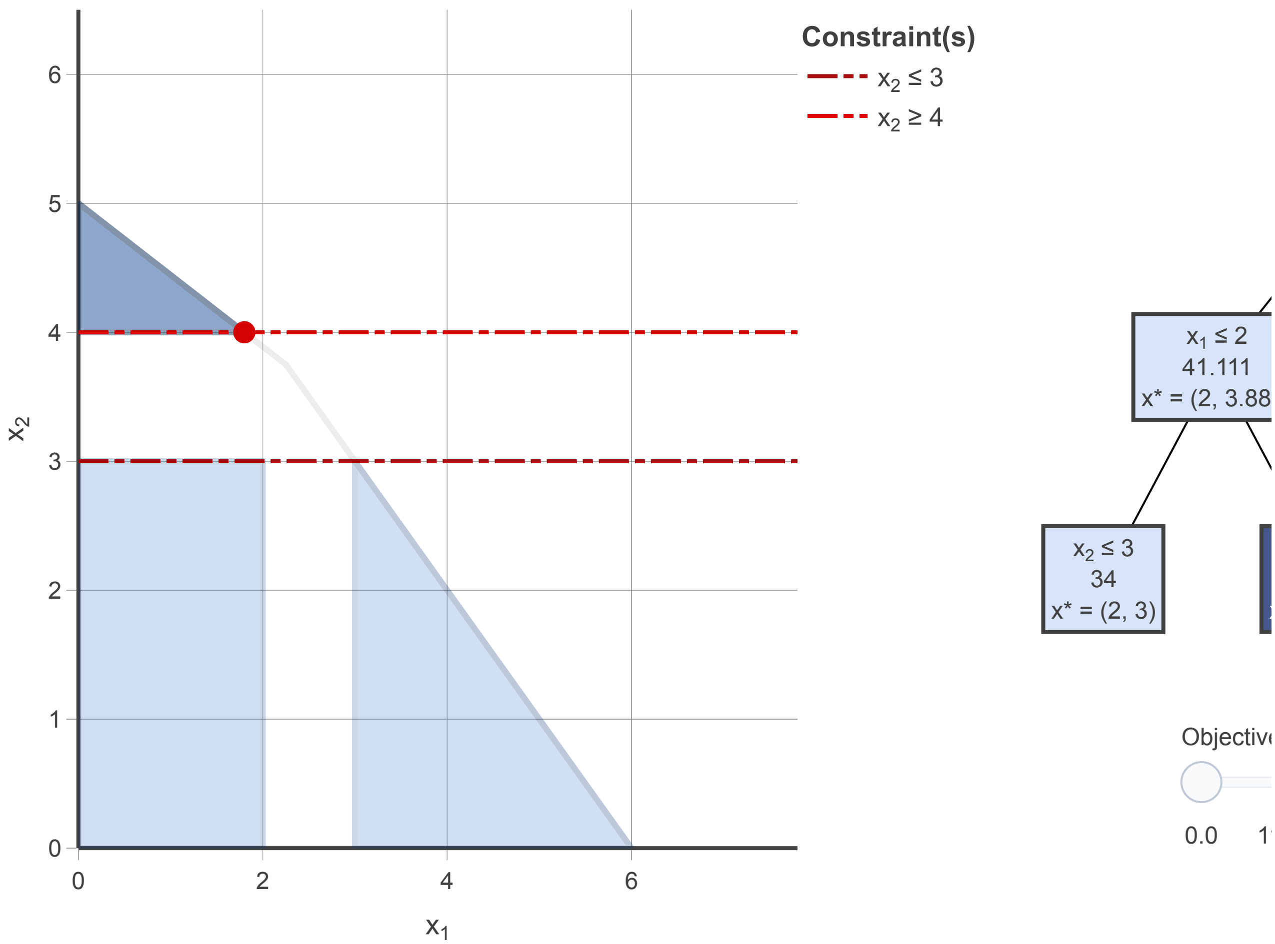
A: 2

Q39: How many nodes have a degree of one in the branch and bound tree? (That is, they are only connected to one edge). These nodes are called leaf nodes. What is the relationship between the leaf nodes and the remaining feasible region?

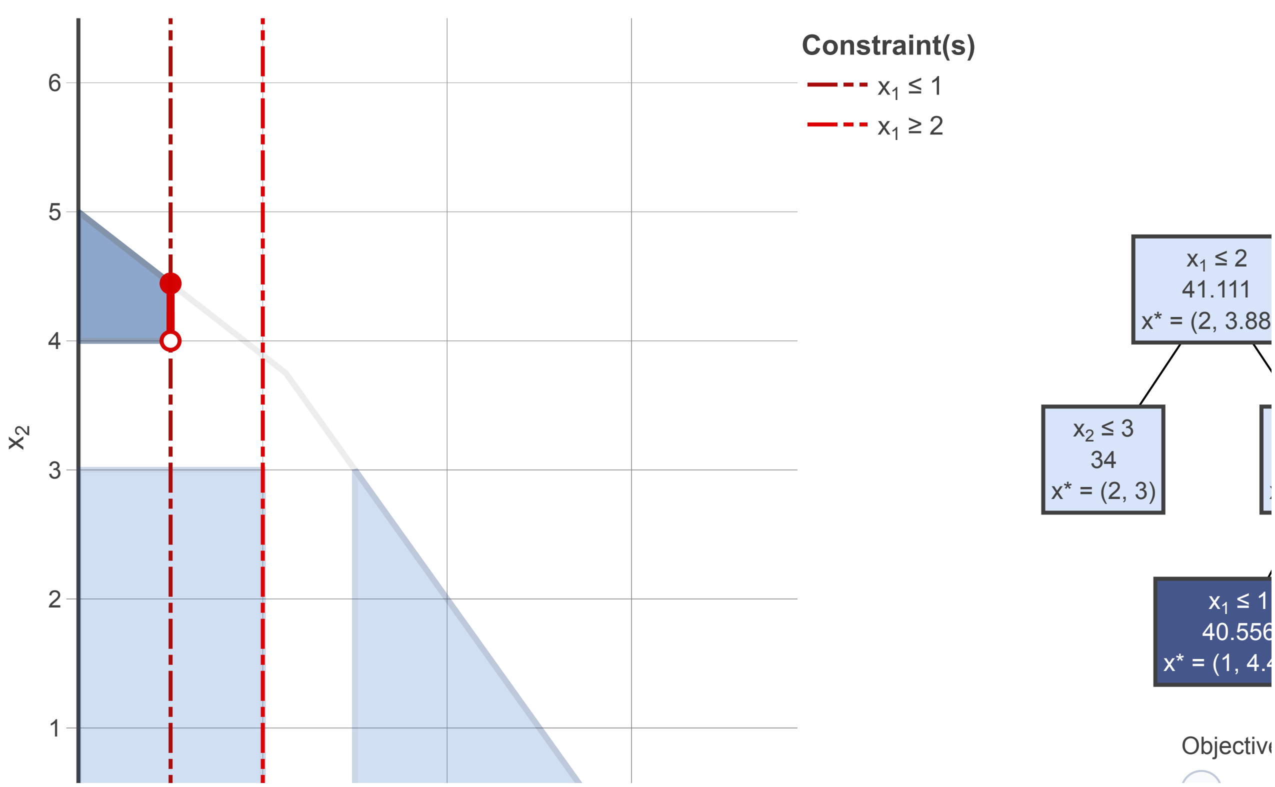
A: 1 node that is shaded darker than the others. This tells us that leaf nodes represent optimal solution.

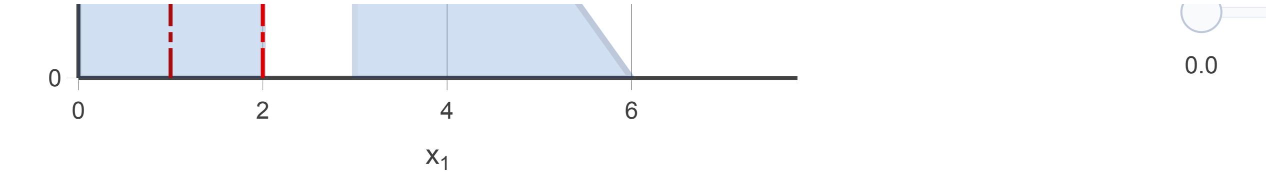
In [47]:
```
# Show the next two iterations of the branch and bound algorithm
nodes[3].show()
nodes[4].show()
```

Geometric Interpretation of LPs



Geometric Interpretation of LPs



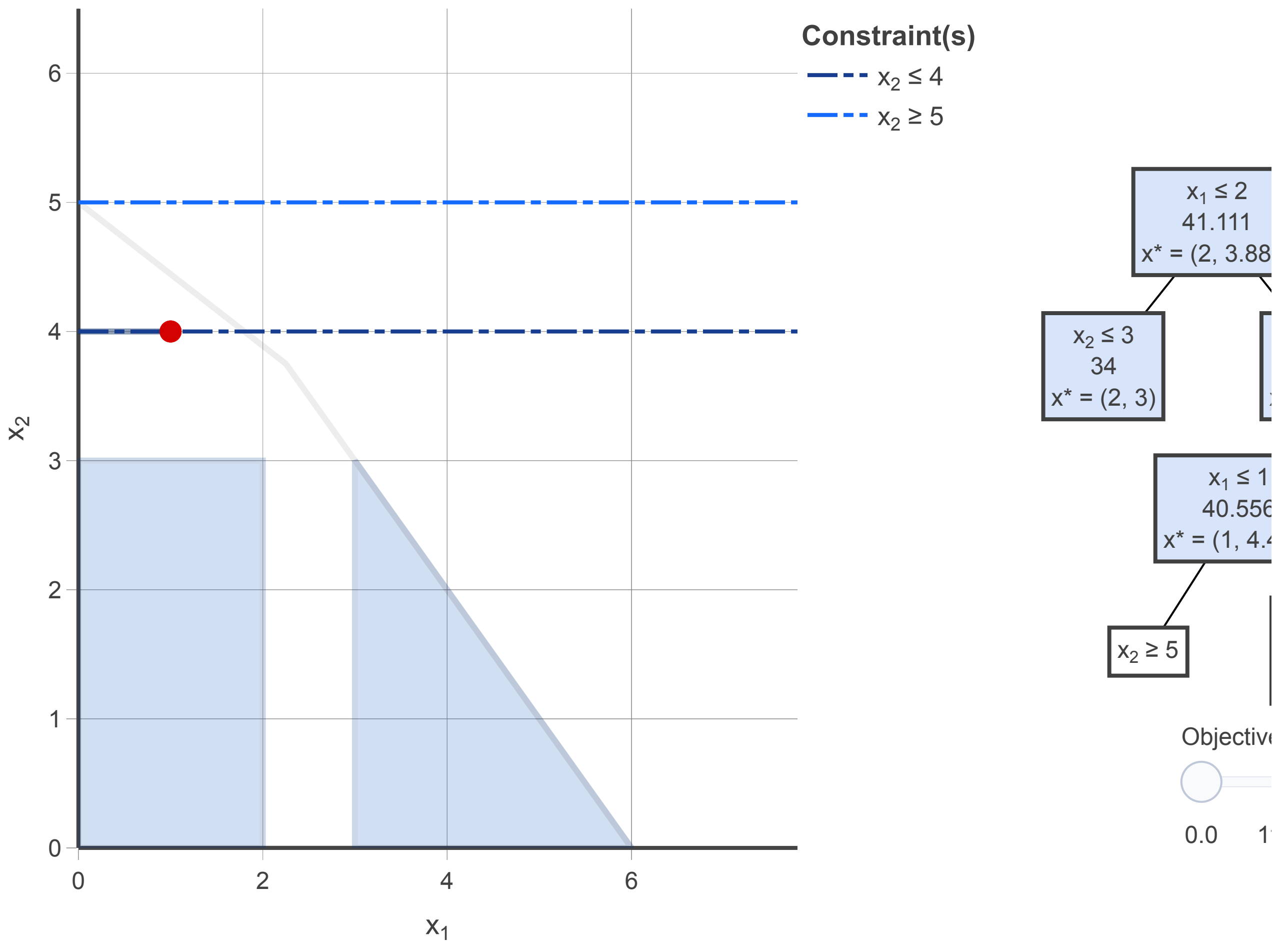


Q40: At the current (dark) node, we added the constraint $x_1 \leq 1$. Why were the fractional solutions $1 < x_1 < 2$ not eliminated for $x_2 <= 3$?

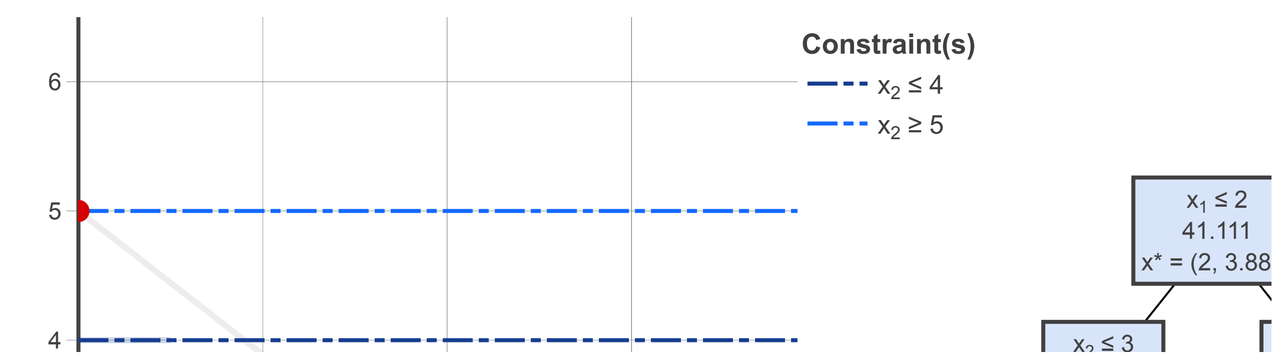
A: Because it produced an integer solution.

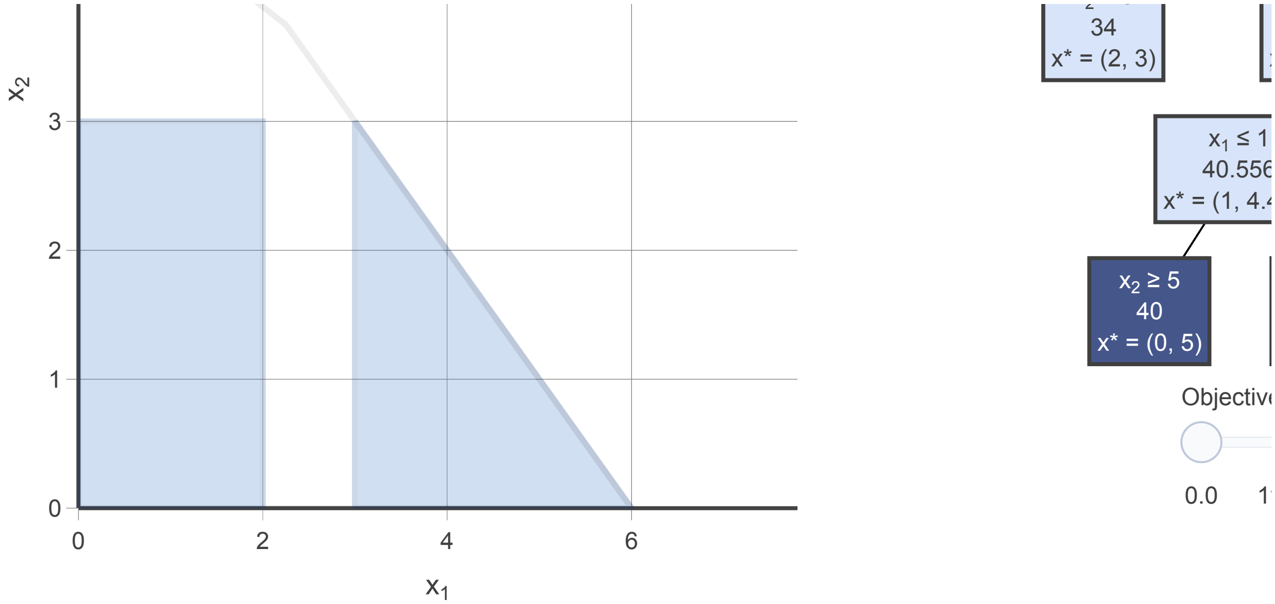
In [48]:
```
# Show the next three iterations of the branch and bound algorithm
nodes[5].show()
nodes[6].show()
nodes[7].show()
```

Geometric Interpretation of LPs

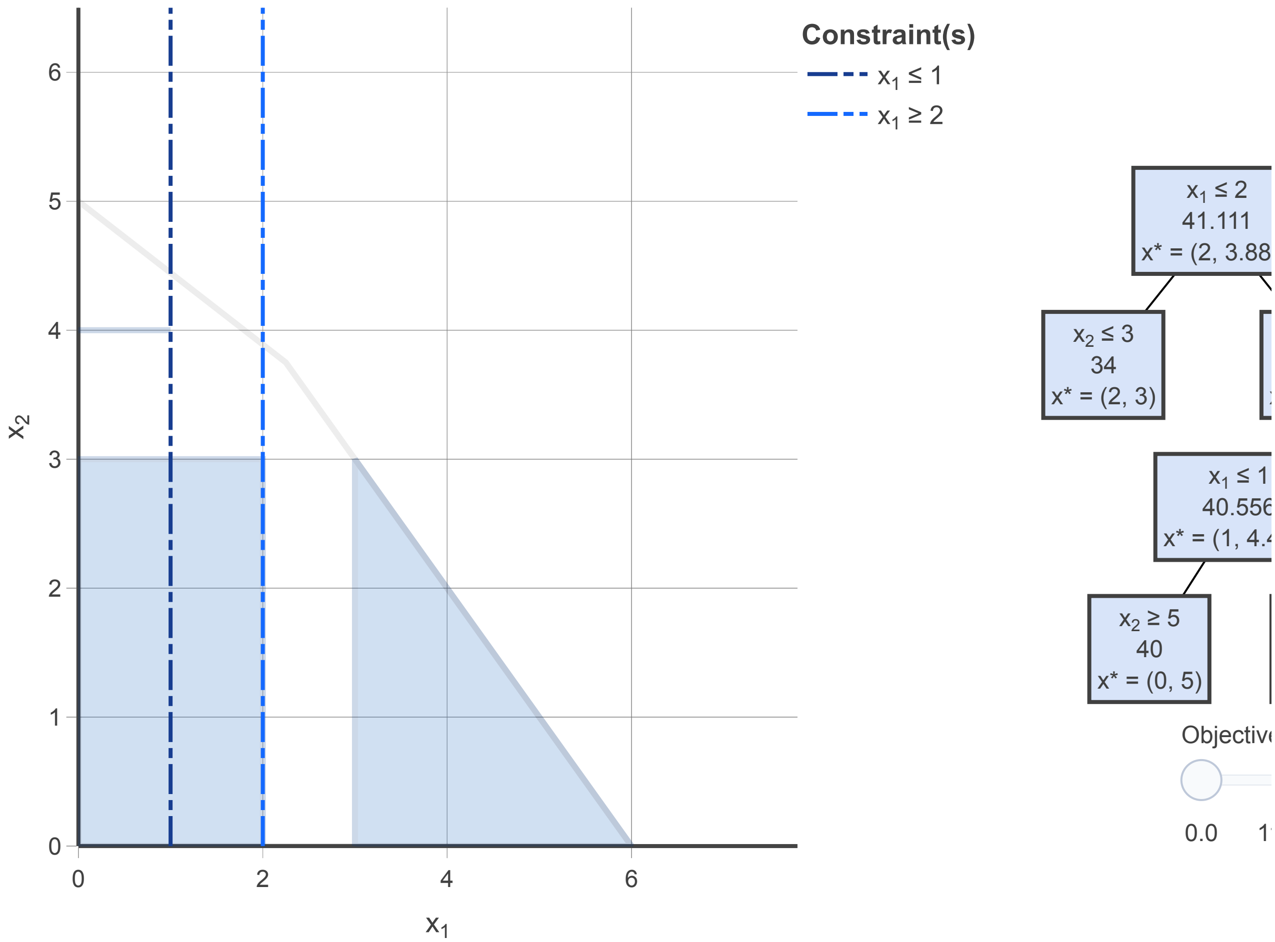


Geometric Interpretation of LPs





Geometric Interpretation of LPs

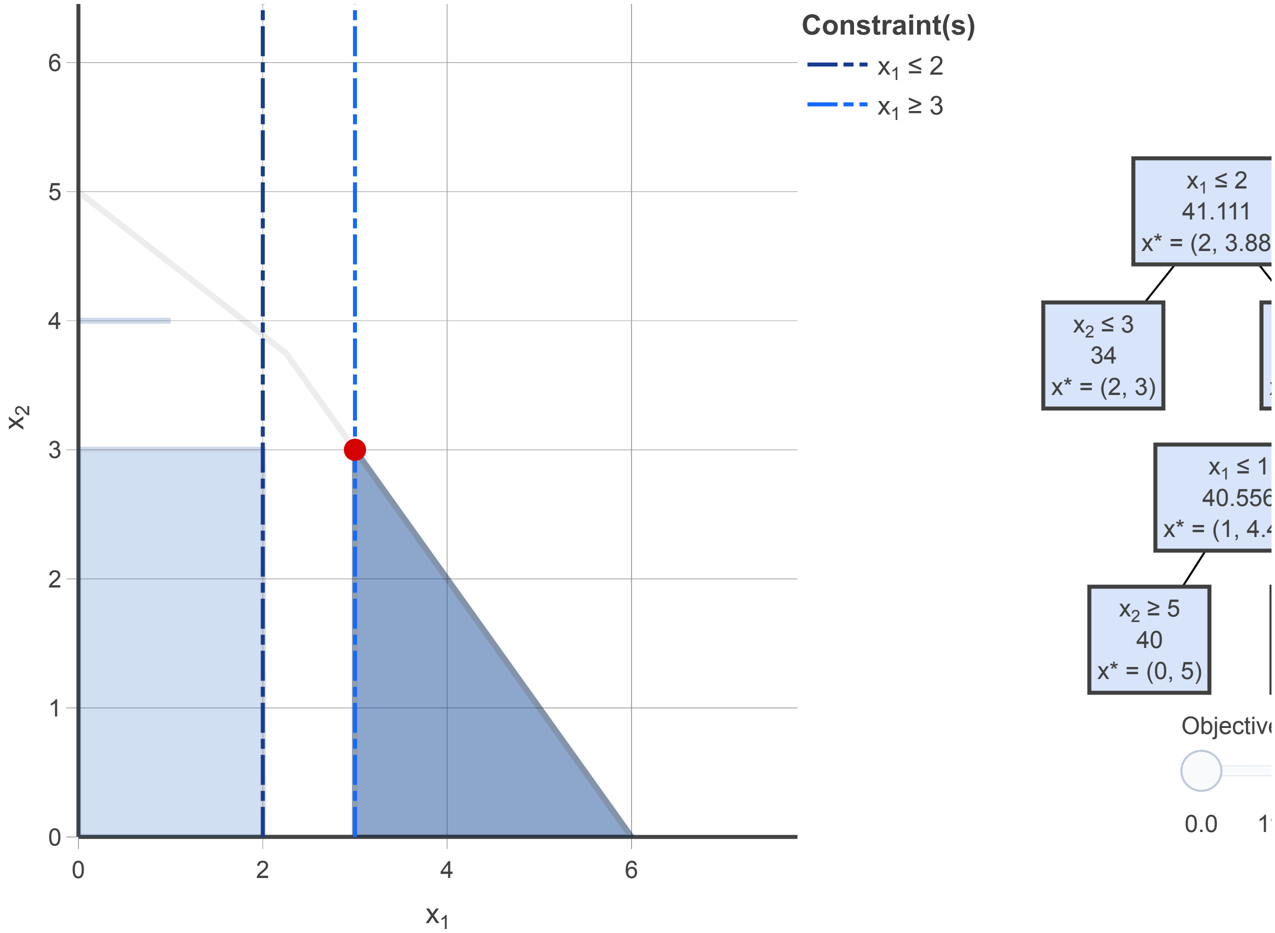


Q41: What constraints are enforced at the current (dark) node? Why are there no feasible solutions at this node?

A: There is no solution where x1=2 and all the other constraints also hold.

In [49]:
```
nodes[8].show()
```

Geometric Interpretation of LPs



Q42: Are we done? If so, what nodes are fathomed and what is the optimal solution? Explain.

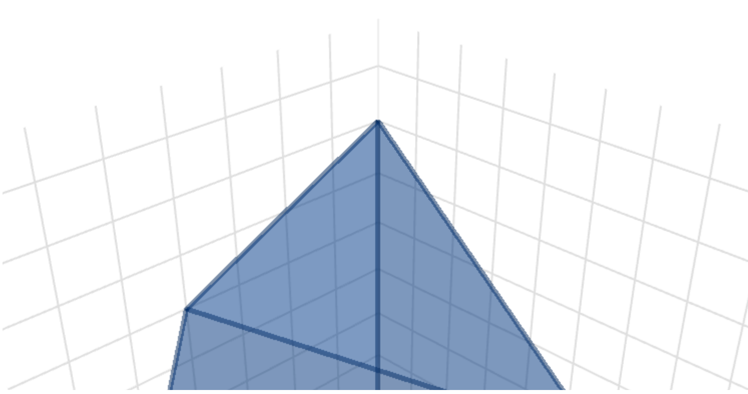
A: We are done because we have all integer solutions. The darkest node is fathomed. The optimal solution is 40.

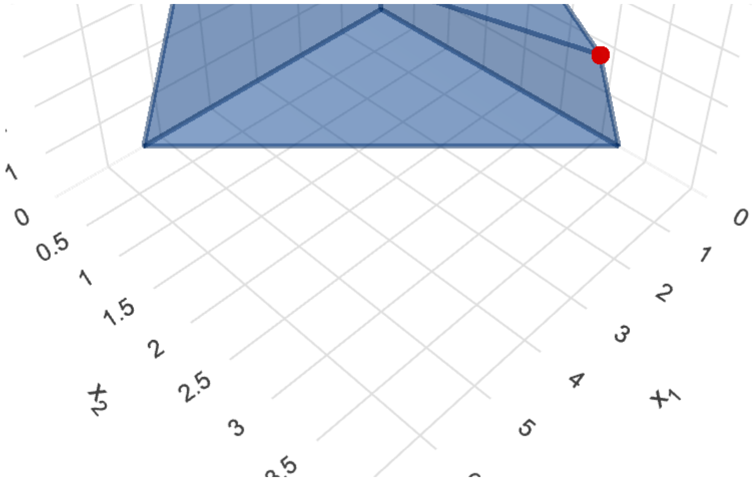
Let's look at branch and bound visualization for an integer program with 3 decision variables!

In [50]:
```
nodes = gilp.bnb_visual(gilp.examples.VARIED_BRANCHING_3D_IP)
```

In [51]:
```
# Look at the first 3 iterations
nodes[0].show()
nodes[1].show()
nodes[2].show()
```

Geometric Interpretation of LPs





Objective V

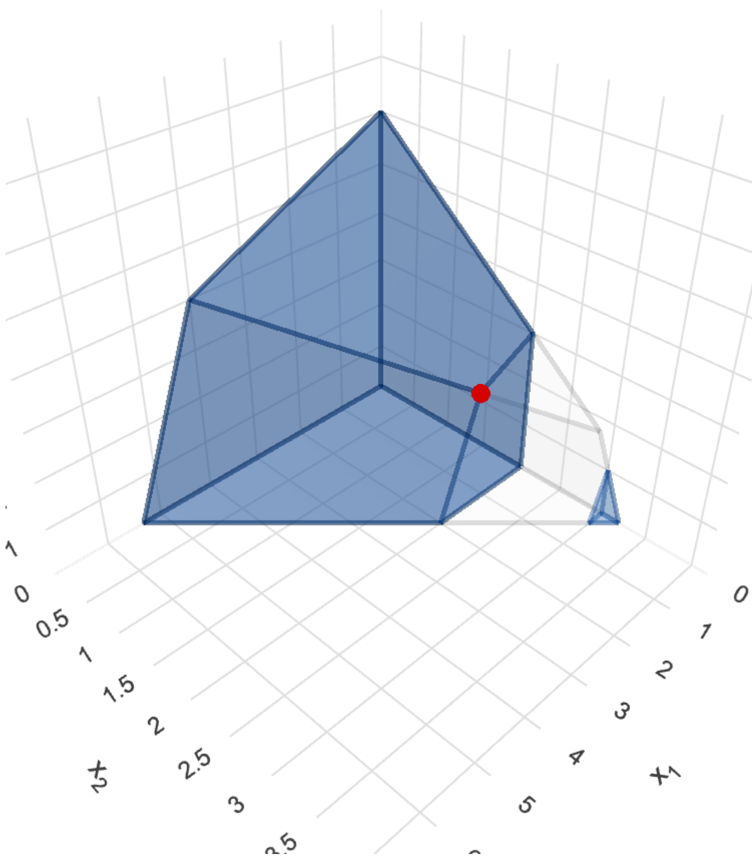
0.0

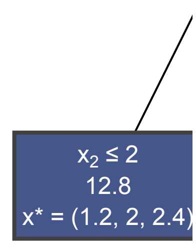
Geometric Interpretation of LPs

Constraint(s)

$x_2 \leq 2$

$x_2 \geq 3$





Objective V

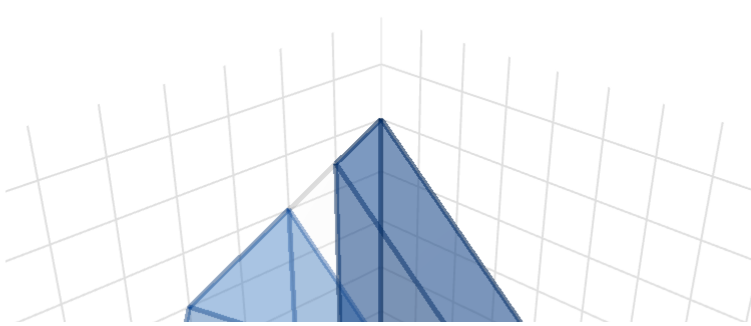
0.0 4.7

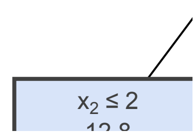
Geometric Interpretation of LPs

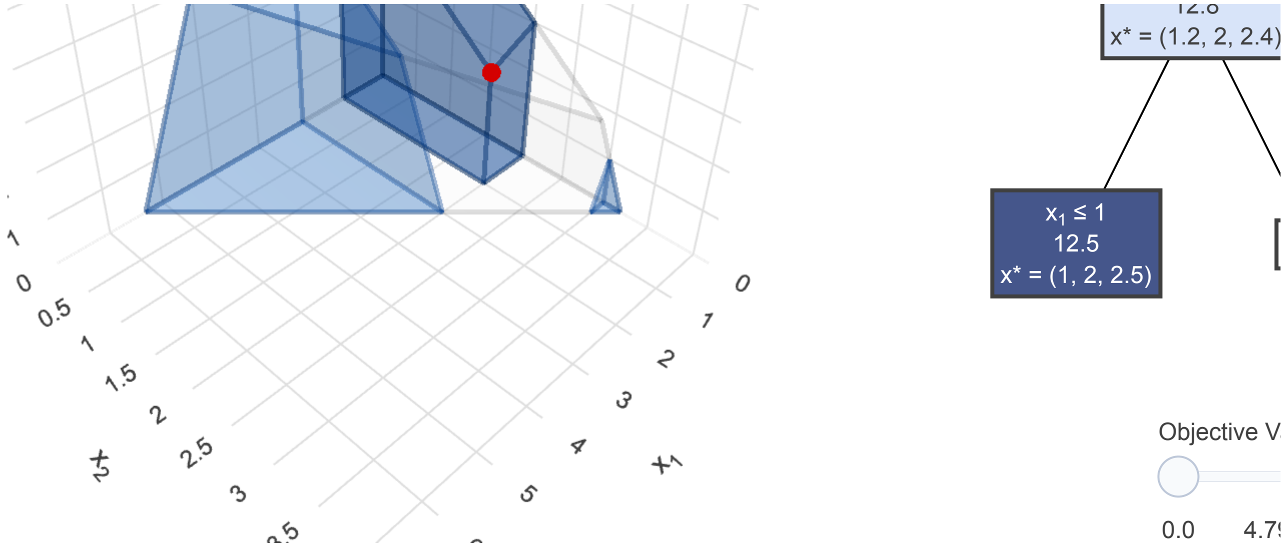
Constraint(s)

$x_1 \leq 1$

$x_1 \geq 2$



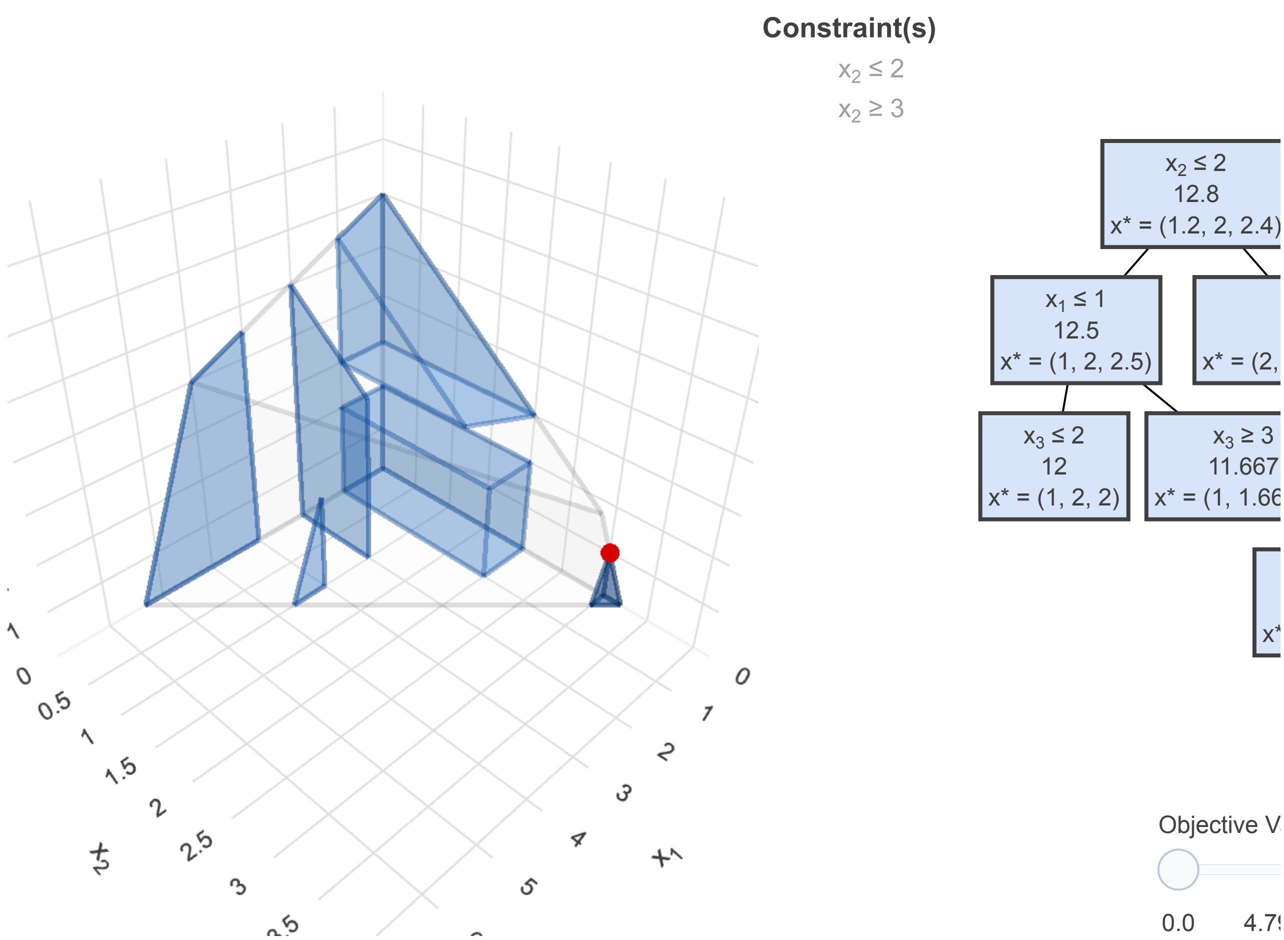




Let's fast-forward to the final iteration of the branch and bound algorithm.

In [52]:
```
nodes[-1].show()
```

Geometric Interpretation of LPs



Q43: Consider the feasible region that looks like a rectangular box with one corner point at the origin. What node does it correspond to in the tree? What is the optimal solution at that node?

A: that corresponds to x1≤1 and x2≤2 and x3≤2 and the optimal solution there is 12.

Q44: How many branch and bound nodes did we explore? What was the optimal solution? How many branch and bound nodes would we have explored if we knew the value of the optimal solution before starting branch and bound?

A: We explored 6 nodes. The optimal solution is 13. We would have only explored the root node if we had known the value of the optimal solution.

Bonus: Branch and Bound for Knapsack

Consider the following example:

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

The linear program formulation will be:

$$\begin{aligned} \max \quad & 2x_1 + 9x_2 + 6x_3 \\ \text{s.t.} \quad & 1x_1 + 3x_2 + 2x_3 \leq 10 \\ & x_1, x_2, x_3 \geq 0, \quad \text{integer} \end{aligned}$$

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()
```