# 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 [2]:
```
# 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 the original branch of the branch and bound has value beta and a child of that branch produces a value that is the rounded up integer value of that beta. In this case, no other child can get a better value than the value just produced, so all other searches can be fathoms. It can also be fathomed when a linear program finds a value greater than the found integer value (no better integer 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 for a maximization serves as an upper bound for the ip. In a minimization problem, it acts as the lower bound for the IP (IP can be equal to or higher than that value. This also goes for optimal solutions.

**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 fathom this solution if and only if there was already a integer program with solution 44 found. In this case, 44 would already be accounted for because we run the lower integer program.

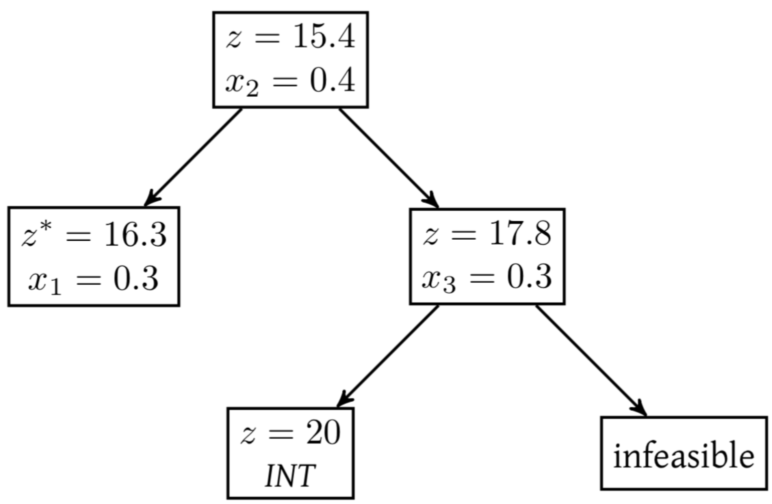
**Q4:** If the optimal solution to the LP relaxation of the original program is integer, then you have found an optimal solution to your integer program. Explain why this is true.

**A:** The lp's optimal solution is a feasible solution for the ip (lp>=ip) and the ip's optimal solution can be a solution to the lp because of integrality constraints being additional (ip>=lp). so Lp opt = ip opt.

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

**A:** The LP is a less constrained version on the IP and the only difference is the integrality constraints, so if the more relaxed problem is infeasible, it is clear that the less relaxed problem must 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 a minimization or maximixation problem? If so, which is it?

**A:** Minimization, constraints create a higher ip. lp lower than ip

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:** Not fathomed because there is still a possibility of finding a ip of 17, 18, or 19, which is lower than the ip currently found through the manipulation of x1 constraints. x1>=1 and x1=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:** These nodes must be greater than or equal to 16.3. Greater than or equal to 17 if an ip is found.

# 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 $i = 1, \dots, n$ 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.

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

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

In [6]:
```
data = pd.read_csv('knapsack_data_1.csv', index_col=0)
data
```

Out[6]:

| 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:** Get rid of 7, weight 30. greater than 18

In [7]:
```
# TODO: replace index
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:** Item 7 could never be used in the first place due to the weight constraint of 18. If it's not used in the lp, it obviuosly is not used in the ip because of more constraints.

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

**A:** 7. This acts as a cutting plane and creates a less feasible range and the optimal value will be less than or equal to the original value.

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 [8]:
```
data['value per unit weight'] = (data['value'] / data['weight']).round(2)
data
```

Out[8]:

| 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:** Select the highest value per unit weight as long as you are still under the W value. Rank in decreasing order and add only if thiis constraint is kept in tact. If value per unit weight is equal try to utilize the higher value one if possible.

**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:** Item 1 and item 4. Value 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 [22]:
```
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 and
        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
    m.Add(sum(w[i]*x[i] for i in ITEMS)<=W)

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

In [23]:
```
# 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 [24]:
```
m, x = Knapsack(data, 18)
```

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

In [25]:
```
solve(m)
```

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

Out[25]:
```
{'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:** Well, the value is higher because this is a linear program. The heuristic solution was constrained to be integer so the lp acted as an upper bound.

**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:** This node shouldn't be fathomed because we haven't found a better integer solution yet, and it still might exist since the one we found is 64. x2 var should be branched on with constraints x_2=0 and x_2>=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 [28]:
```
# TODO: Answer Q17
m, x = Knapsack(data, 18)
m.Add(x[2] >= 1)
solve(m)
```

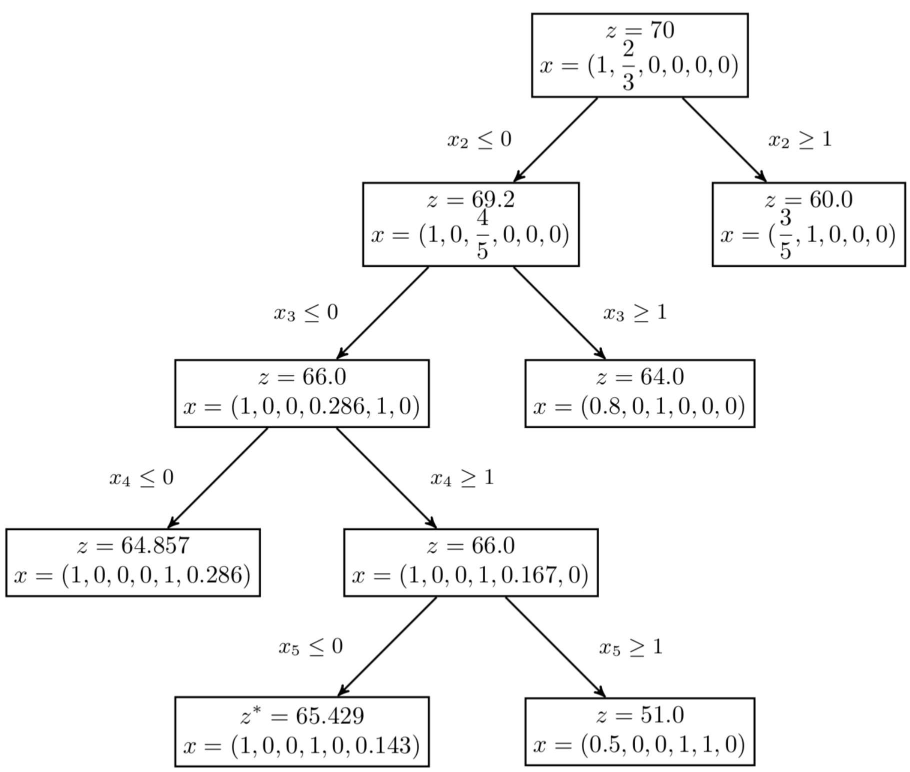
```
Objective = 60.0
iterations : 0
branch-and-bound nodes : 0
```

Out[28]:
```
{'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:** 60. This node can be fathomed because it is lower than the ip found and constraints will only limit it further. Fathomed

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:** We know that 64 is a feasible ip solution and this node at best can creat a ip of 64 so it's not making any progress. This is found by the lower integer value of [64.857].

**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 [29]:
```
# 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- yes
```

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

Out[29]:
```
{'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 [30]:
```
# 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 - nah (it's the end so technically yes, but answer is solved)
```

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

Out[30]:
```
{'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 [ ]:
```

```

**A:** 64 (ITEMS 1 and 4) matching heuristsics

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

**A:** 2 additional. 11 in total

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

# Part 3: Cutting Planes

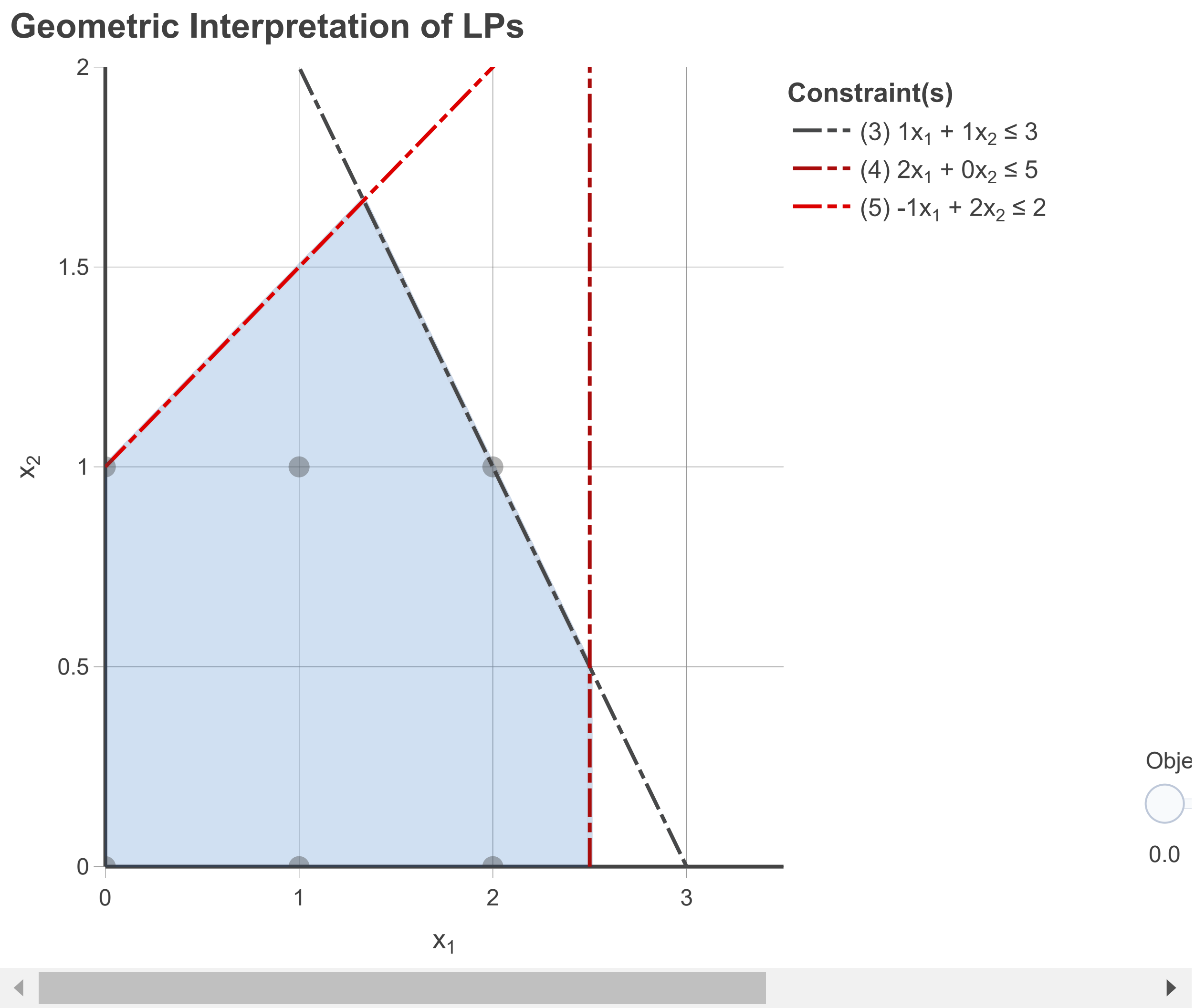
In general, a cutting plane is an additional constraint we can add to an integer program's linear relaxation that removes feasible linear solutions but does not remove any integer feasible solutions. This is very useful when solving integer programs! Recall many of the problems we have learned in class 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 [32]:
```
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 program.

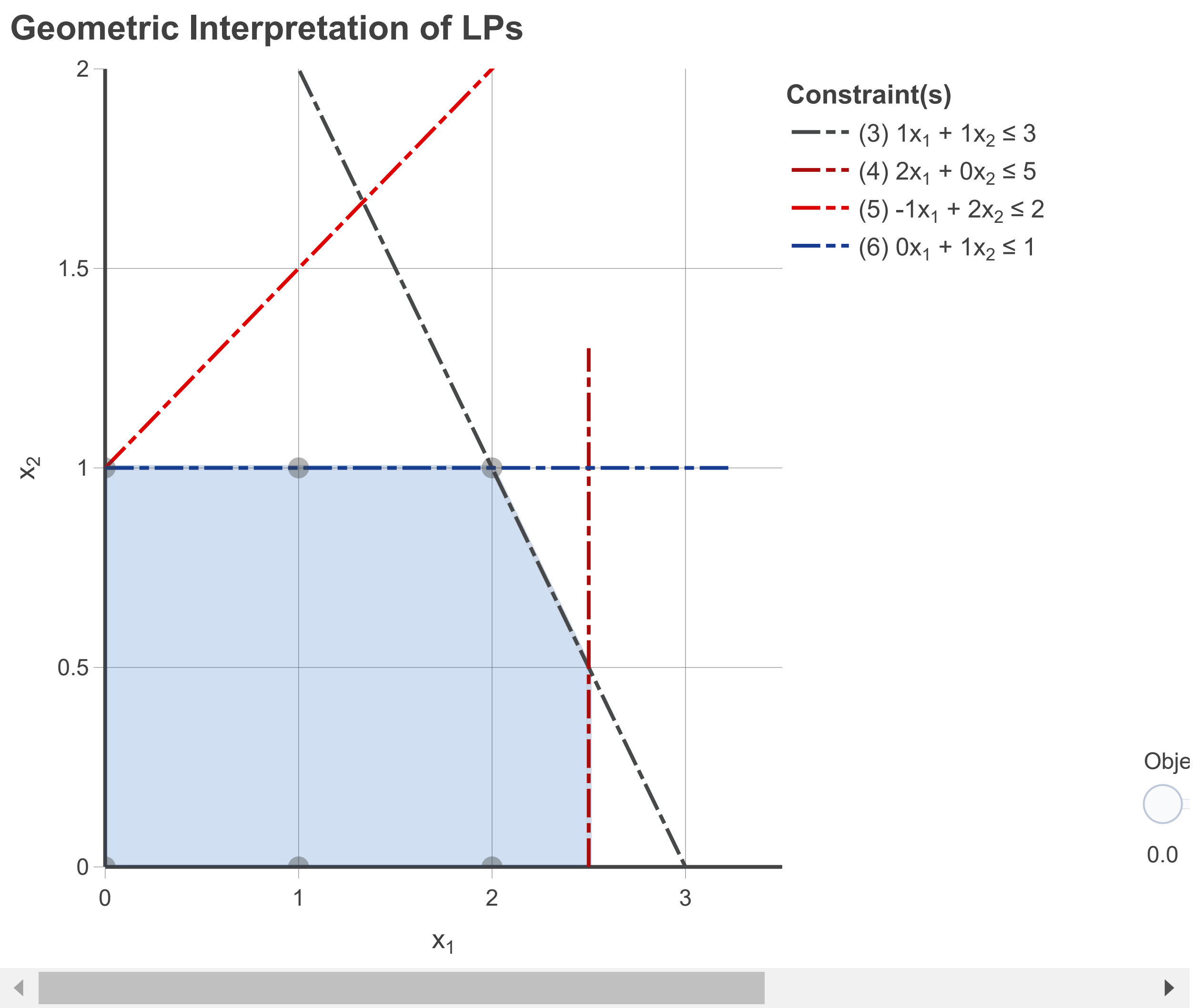
**A:** (0,0) 0, (0,1) 1, (1,0) 2, (1,1) 3, (2,0) 4, (2,1) 5

**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:** Yes, because it cuts the feasible region without getting rid of an integer pint that's feasible. Feasible linear points cut out would become infeasible

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

In [33]:
```
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 $x_1 \le 3$ a cutting plane? Why?

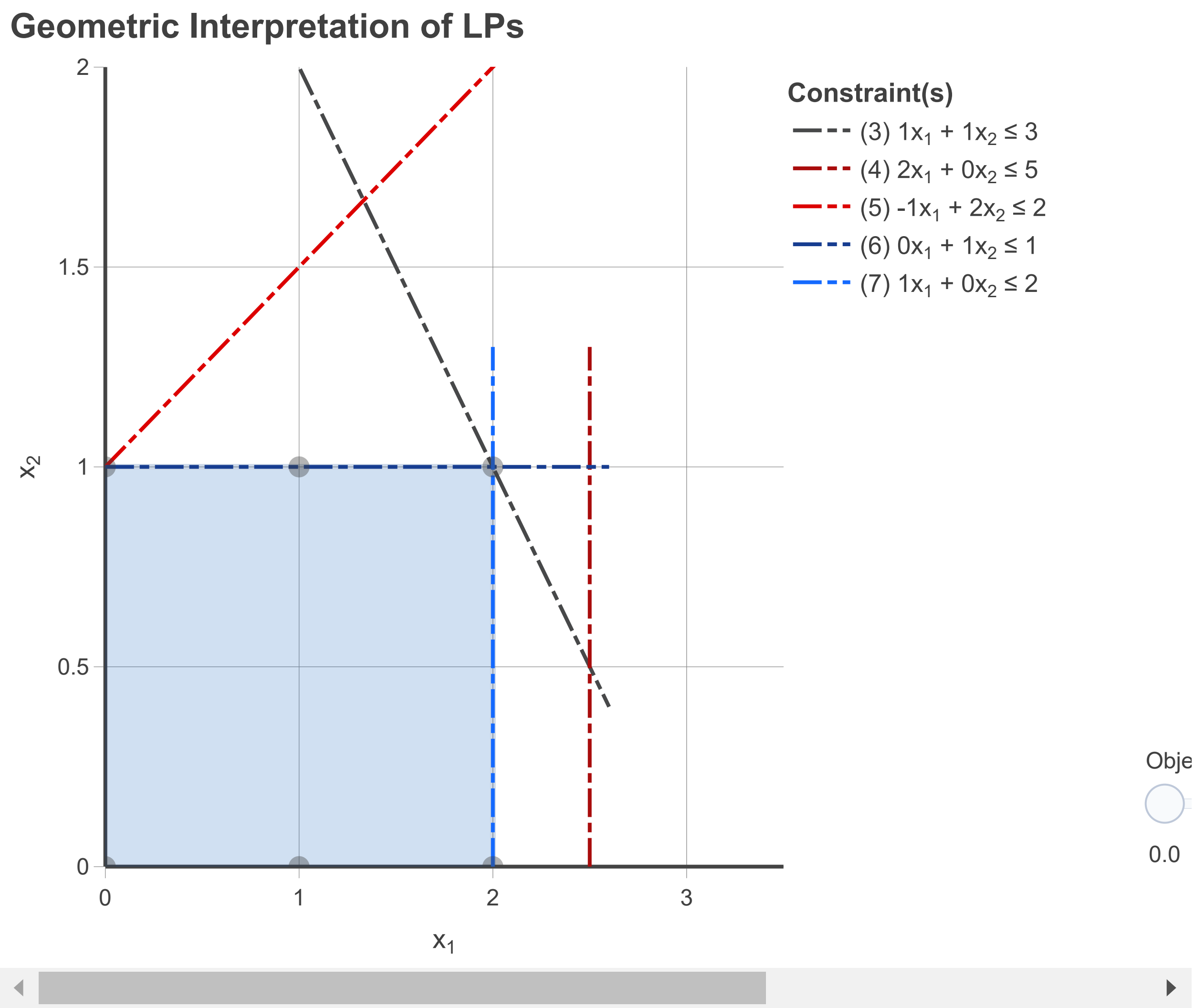
**A:** No, beacuse even though it keeps all the integer points in tact, it doesn't cut the lp optimum. It is unaffected.

**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]:

| 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 |

**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:** 1 in each case. x(1) + x(2) + x(3) <=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:** The constraint we found is a cutting plane. item 1 and 0.66667 item 2. x(1) = 1 x(2)=2/3

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

**A:** x(4) + x(5) + x(6) <=2. You can't have more than 2 items while living under W=18

**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.

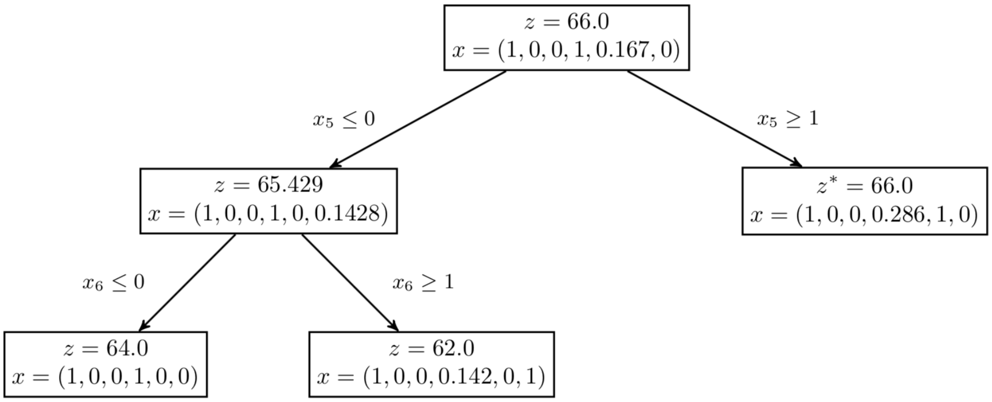
In [37]:
```
m, x = Knapsack(data, 18)
m.Add(x[1]+x[2]+x[3]<=1)
m.Add(x[4]+x[5]+x[6]<=2)

solve(m)
```

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

Out[37]:
```
{'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 [39]:
```
# Template
m, x = Knapsack(data, 18)
m.Add(x[5]>=1)
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 - yes
```

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

Out[39]:
```
{'x_1': 1.0,
 'x_2': 0.0,
 'x_3': 0.0,
 'x_4': 0.0,
 'x_5': 1.0,
 'x_6': 0.28571428571428586}
```

In [41]:
```
# Template
m, x = Knapsack(data, 18)
m.Add(x[5]>=1)
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 - yes
```

```
Objective = 51.0
iterations : 0
branch-and-bound nodes : 0
```

Out[41]:
```
{'x_1': 0.5, 'x_2': 0.0, 'x_3': 0.0, 'x_4': 1.0, 'x_5': 1.0, 'x_6': 0.0}
```

In [ ]:

**A:** Both were fathomed, the opt sol to the ip still remains 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. 7 nodes explored. Lower by 4 nodes.

## Part 4: Geometry of Branch and Bound

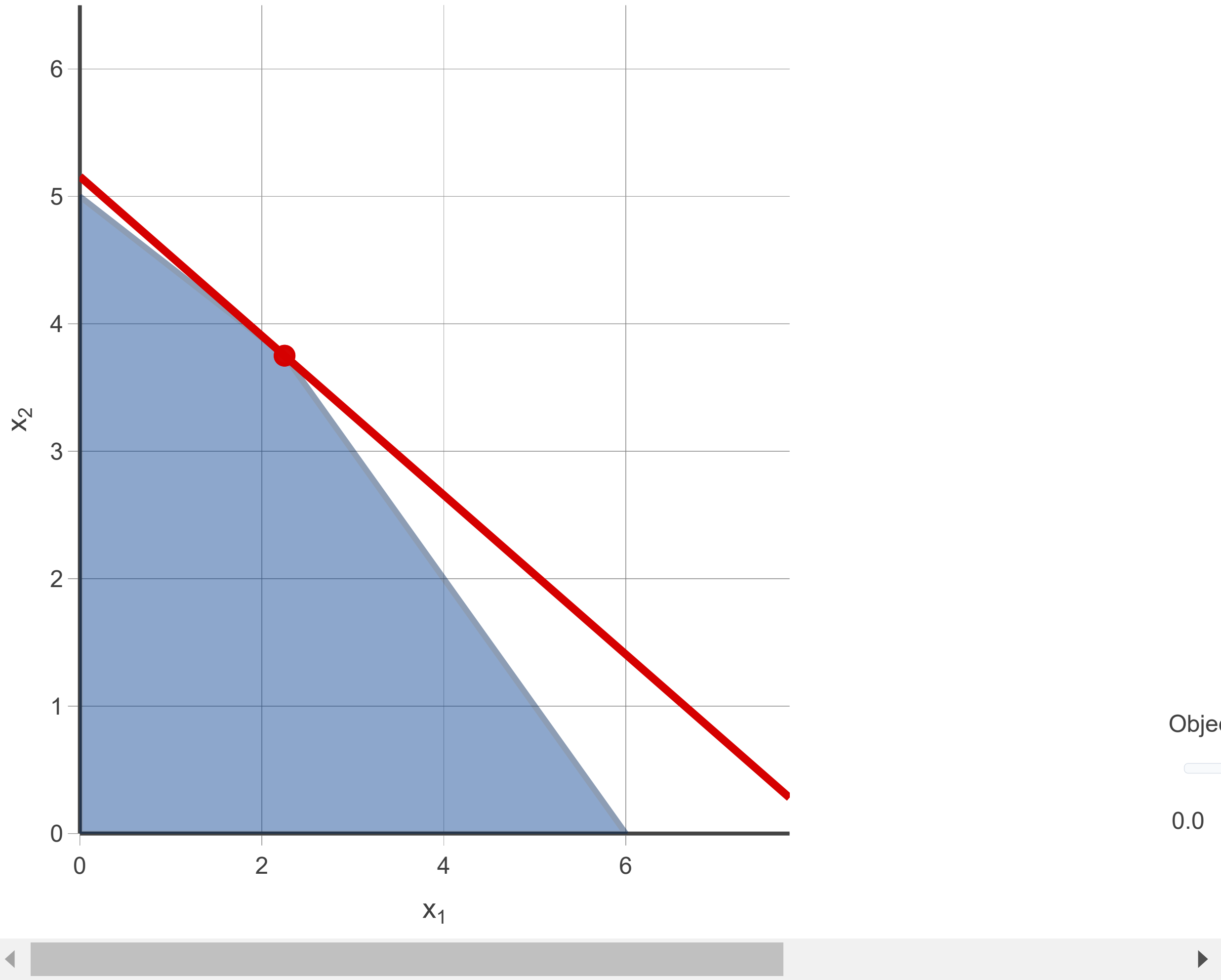
Previously, we used the `gilp` package to viusualize the simplex algorithm but it also has the functionality to visualize branch and bound. We will give a quick overview of the tool. Similar to `lp_visual` and `simplex_visual`, the function `bnb_visual` takes an `LP` and returns a visualization. It is 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:

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

In [42]:
```
nodes = gilp.bnb_visual(gilp.examples.STANDARD_2D_IP)
```

In [43]:
```
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 for the lp 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:** The two constraints are x1<=2 and x1>=3

$$\begin{array}{rl} \max & 5x_1 + 8x_2 \\ \text{s.t.} & x_1 + x_2 \leq 6 \\ & x_1 \leq 2 \\ & 5x_1 + 9x_2 \leq 45 \\ & x_1, x_2 \geq 0, \quad \text{integral} \end{array}$$

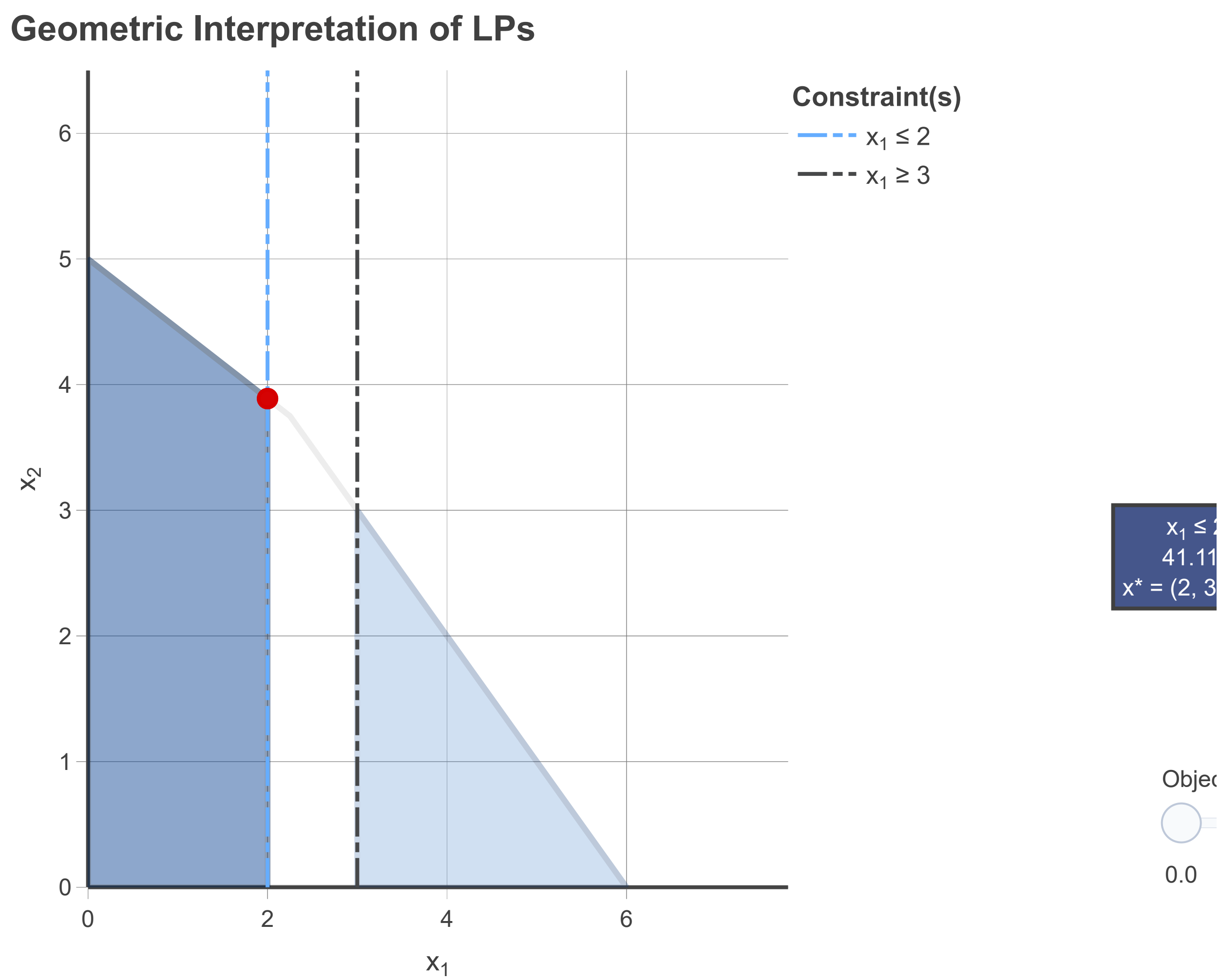
$$\begin{array}{rl} \max & 5x_1 + 8x_2 \\ \text{s.t.} & x_1 + x_2 \leq 6 \\ & x_1 \geq 3 \\ & 5x_1 + 9x_2 \leq 45 \\ & x_1, x_2 \geq 0, \quad \text{integral} \end{array}$$

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

**A:** Cut at x1 =2 and x1 =3 not including middle ground between constraints

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

In [44]:
```
nodes[1].show()
```




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.

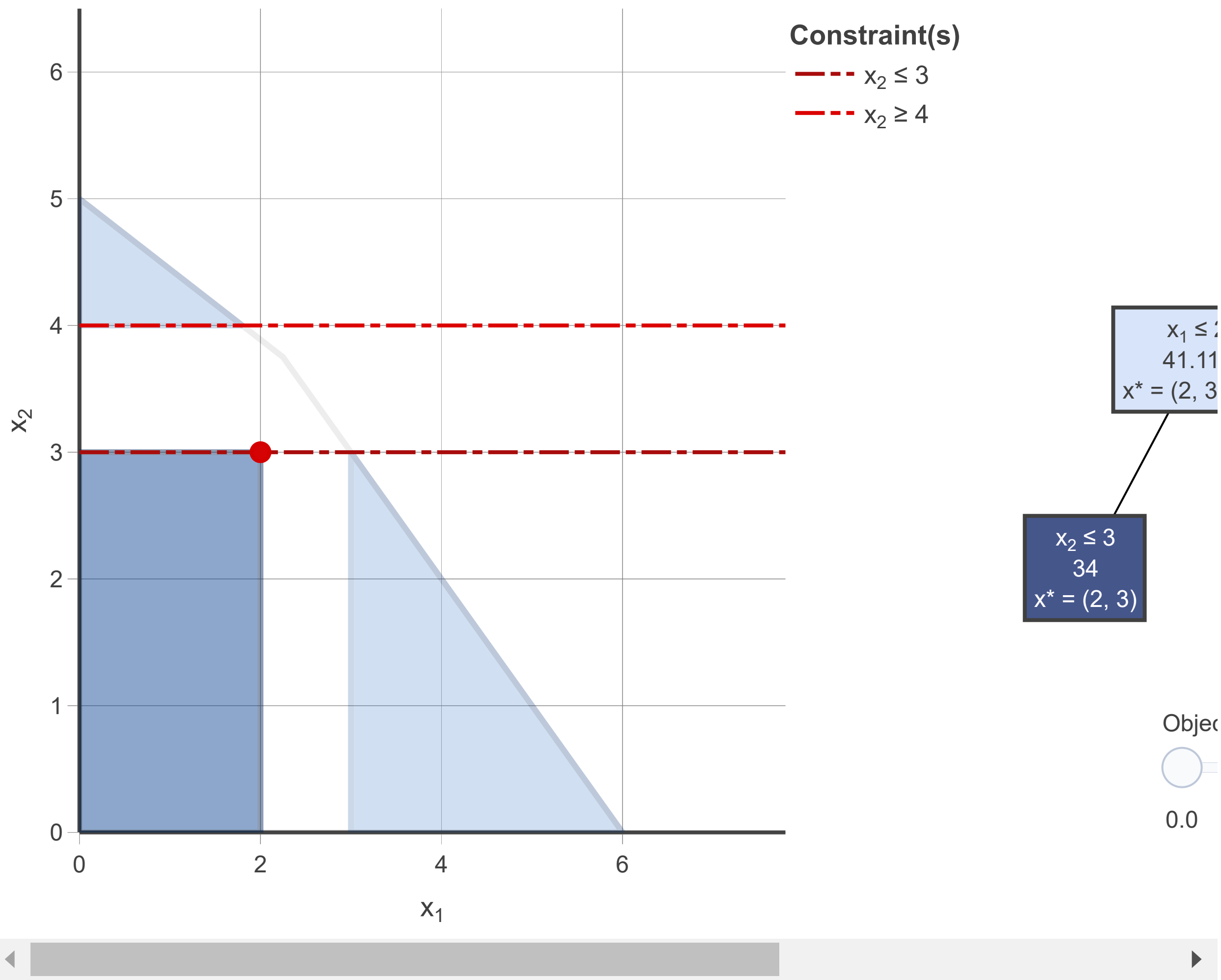
**Q35:** Which feasible solutions to the LP relaxation are removed by this branch?

**A:** removes all feasible solutions with x1>2

```
**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. 3 feasible regions
```

In [45]:
```
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:** 34, no an integer solution is found. fathom

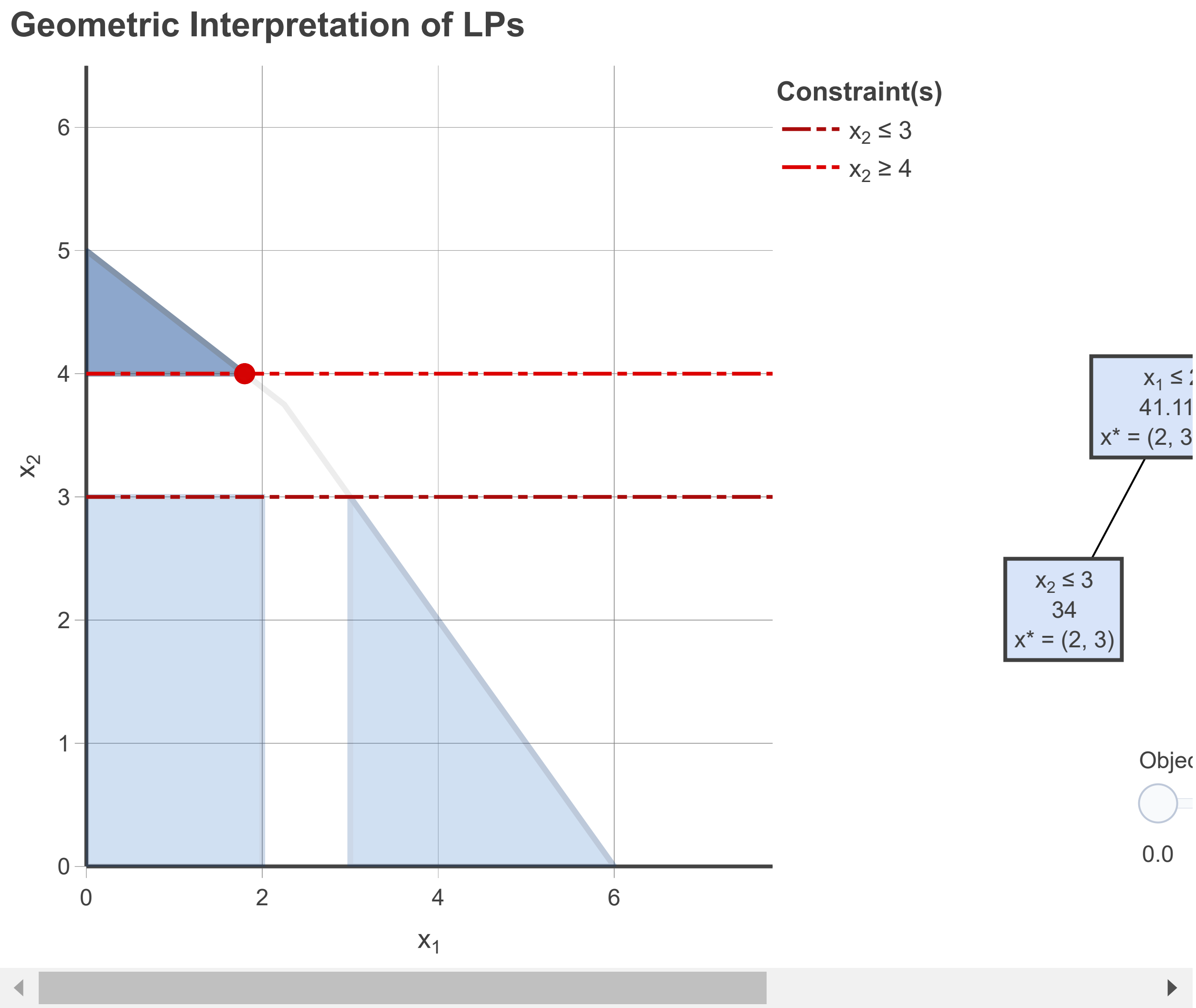
**Q38:** Recall shaded nodes have been explored and the node shaded darker (and feasible region shaded darker) correspond to the current node and its feasible region. Nodes not shaded have not been explored. How many nodes have not yet been explored?

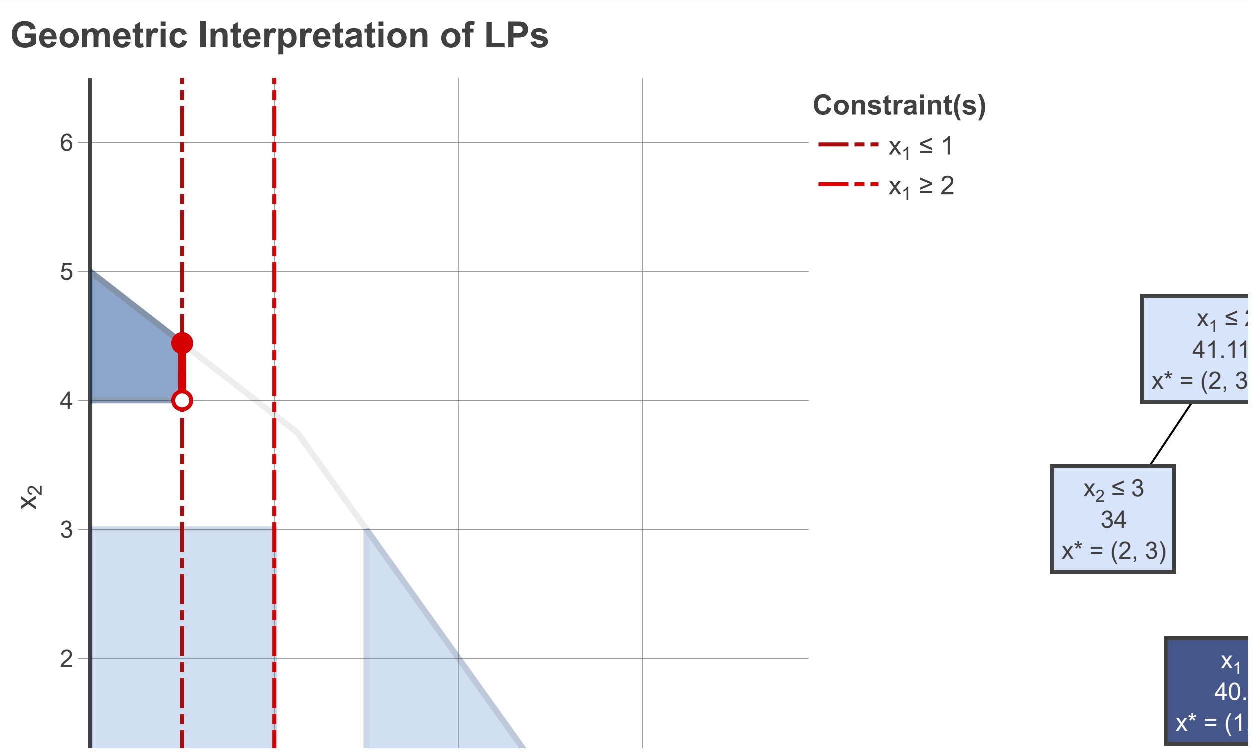
**A:** 2

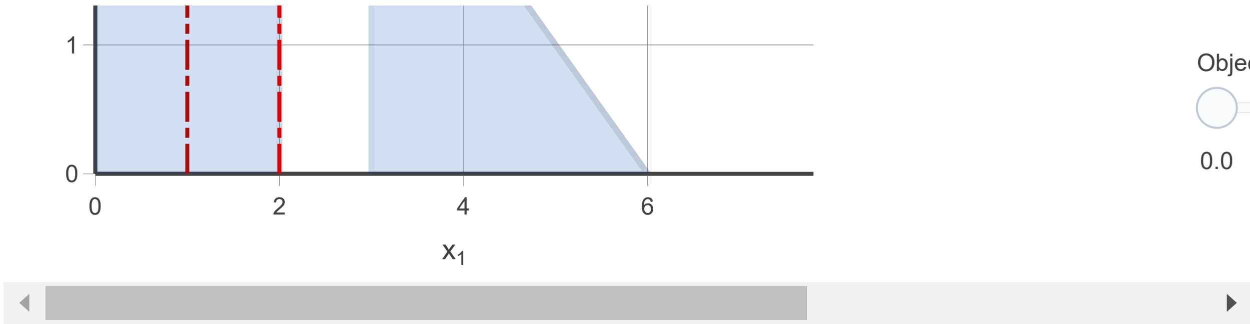
**Q39:** How many nodes have a degree of one in the branch and bound tree? (That is, they are only connected to one edge). These nodes are called leaf nodes. What is the relationship between the leaf nodes and the remaining feasible region?

**A:** 3 nodes. They are the feasible regions that the og lp was cut into

In [46]:
```
# Show the next two iterations of the branch and bound algorithm
nodes[3].show()
nodes[4].show()
```



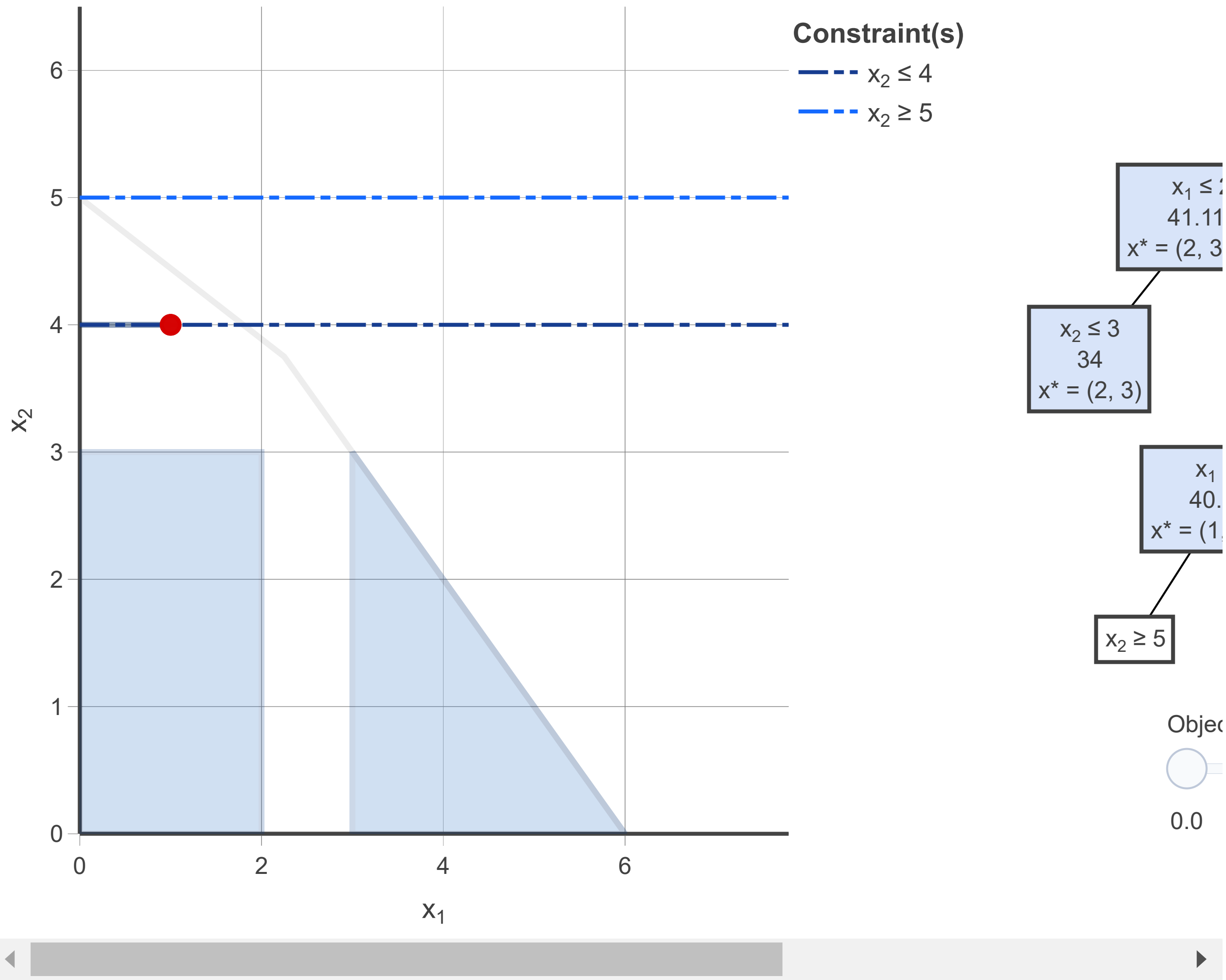



**Q40:** At the current (dark) node, we added the constraint $x_1 \leq 1$. Why were the fractional solutions $1 < x_1 < 2$ not eliminated for $x_2 <= 3$?

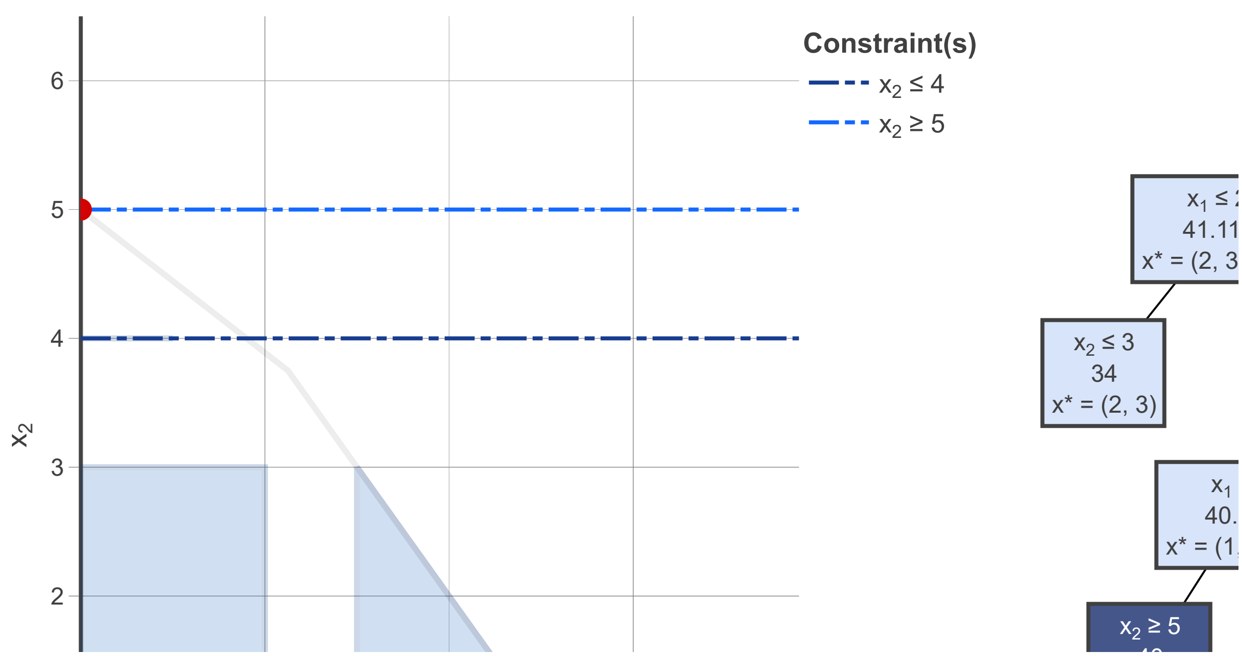
**A:** Because we found an integer solution and this was techically fathomed because an integer sol was found

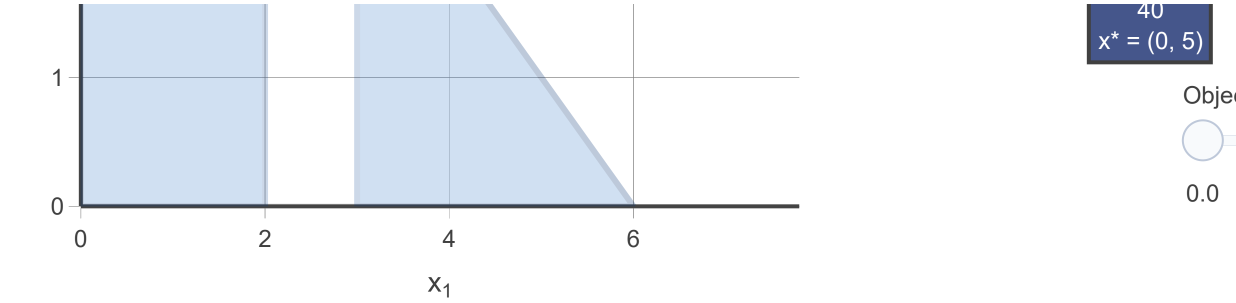
In [47]:
```
# Show the next three iterations of the branch and bound algorithm
nodes[5].show()
nodes[6].show()
nodes[7].show()
```

## Geometric Interpretation of LPs

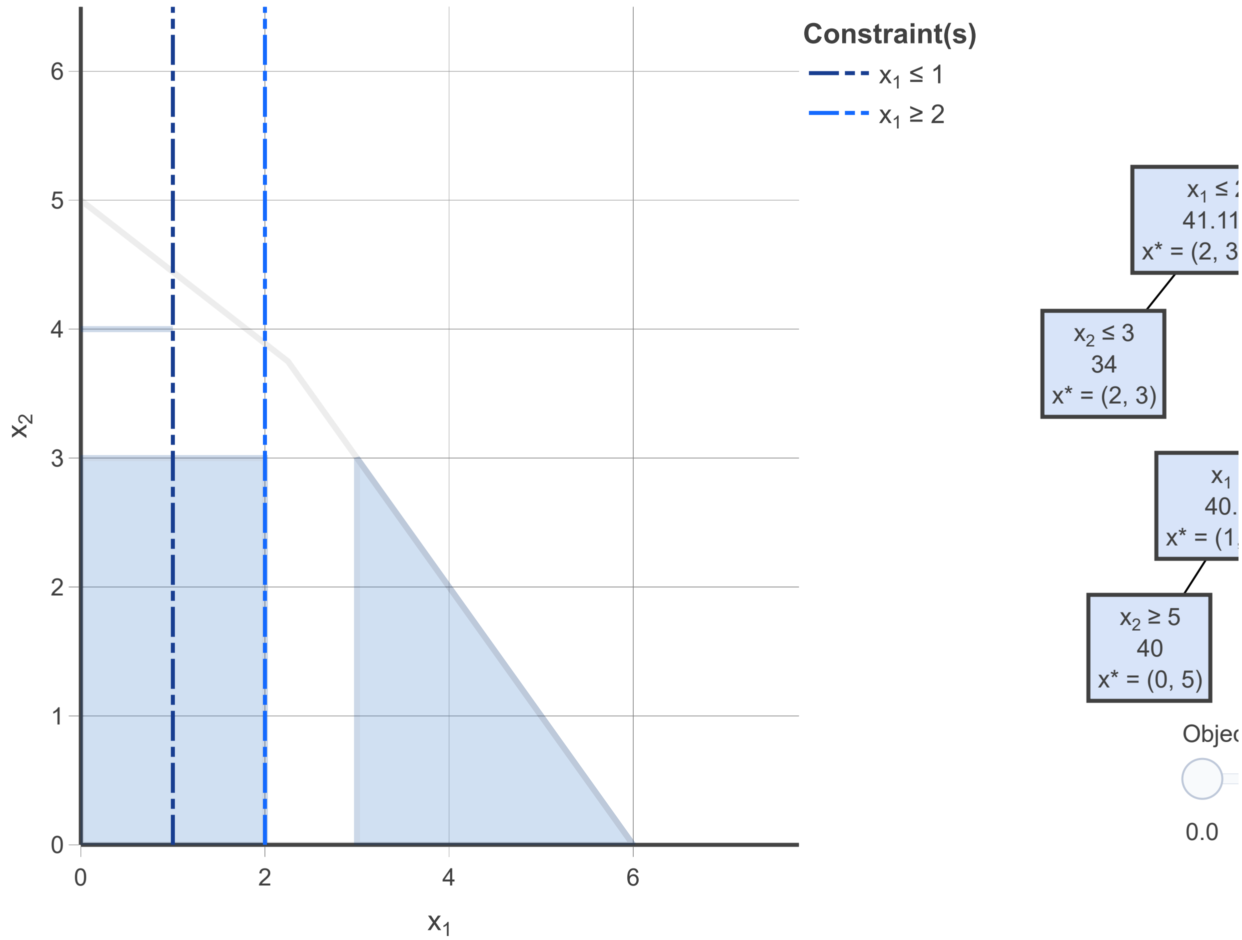



## Geometric Interpretation of LPs





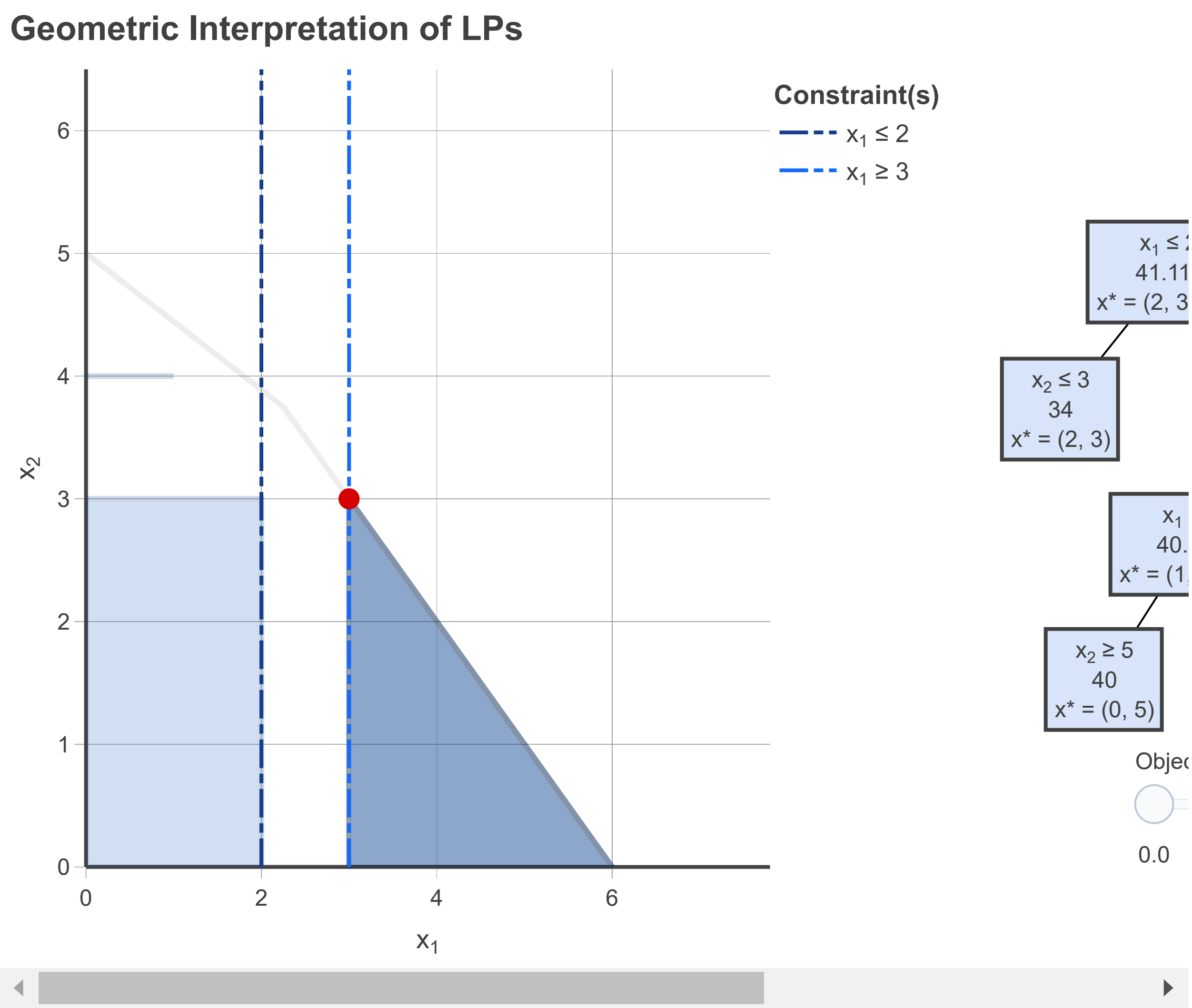

## Geometric Interpretation of LPs




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

**A:** x1<=2, x2>=4, x1>=2. both x1 constarints contradict themseves. a line is left, not a shape. x1=2 already explored

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




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

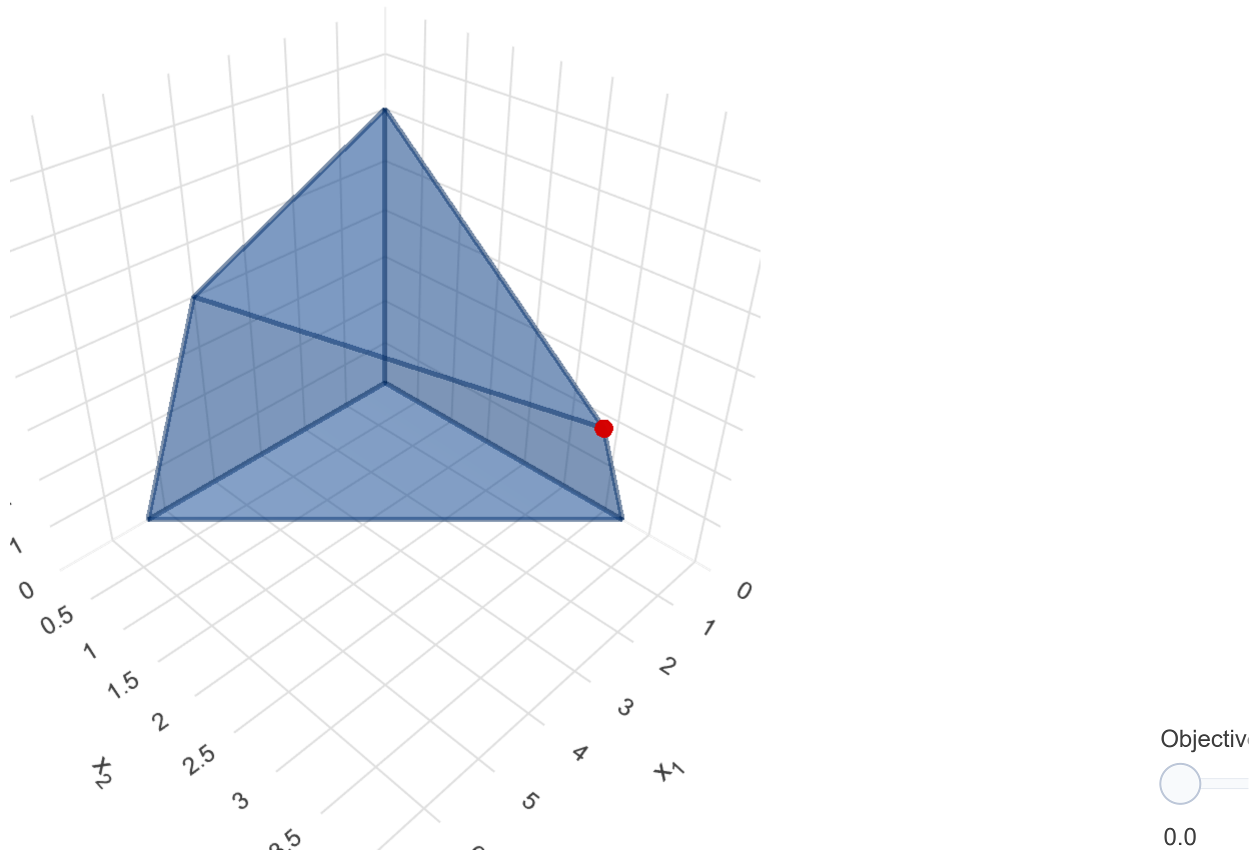
**A:** Yes. Because all leaves are integer of infeasible. Every leaf node is fathomed except for the one with objective value 40. Technically that one is fathomed as well.

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

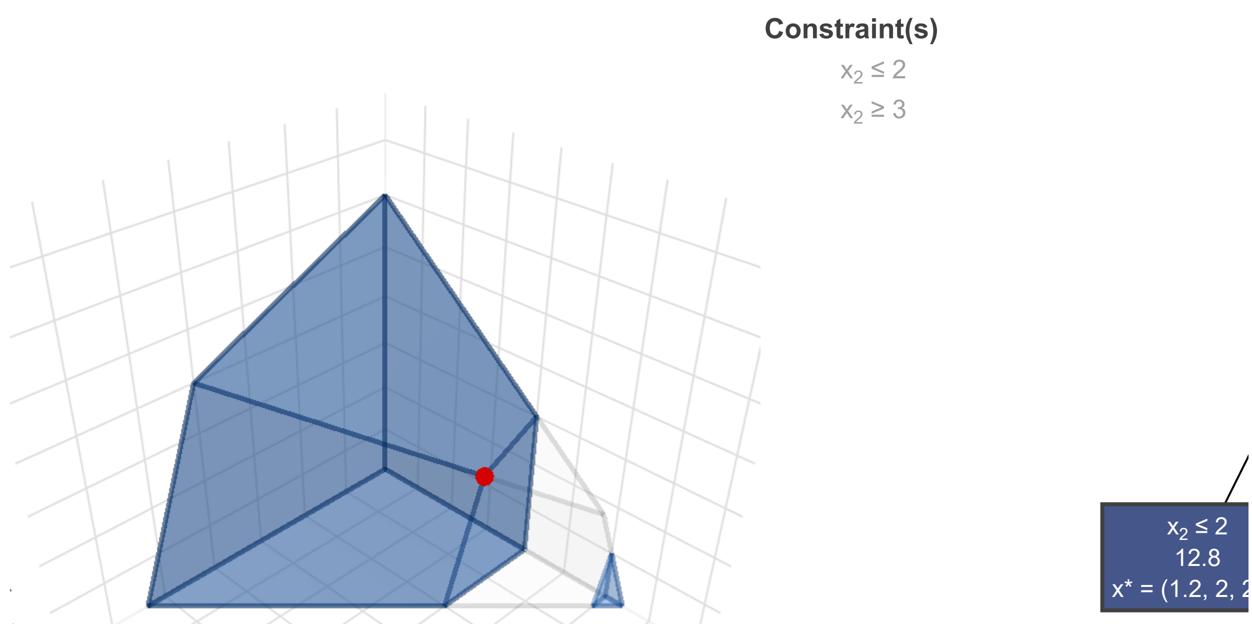
In [50]:
```
nodes = gilp.bnb_visual(gilp.examples.VARIED_BRANCHING_3D_IP)
```

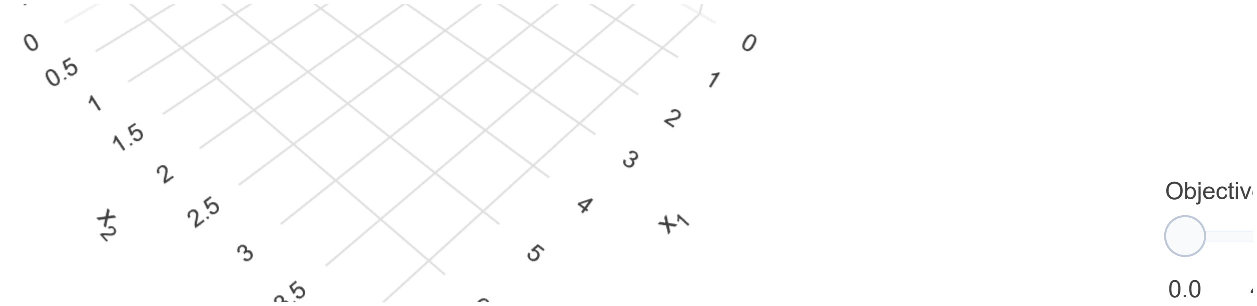
In [51]:
```
# Look at the first 3 iterations
nodes[0].show()
nodes[1].show()
nodes[2].show()
```

### Geometric Interpretation of LPs

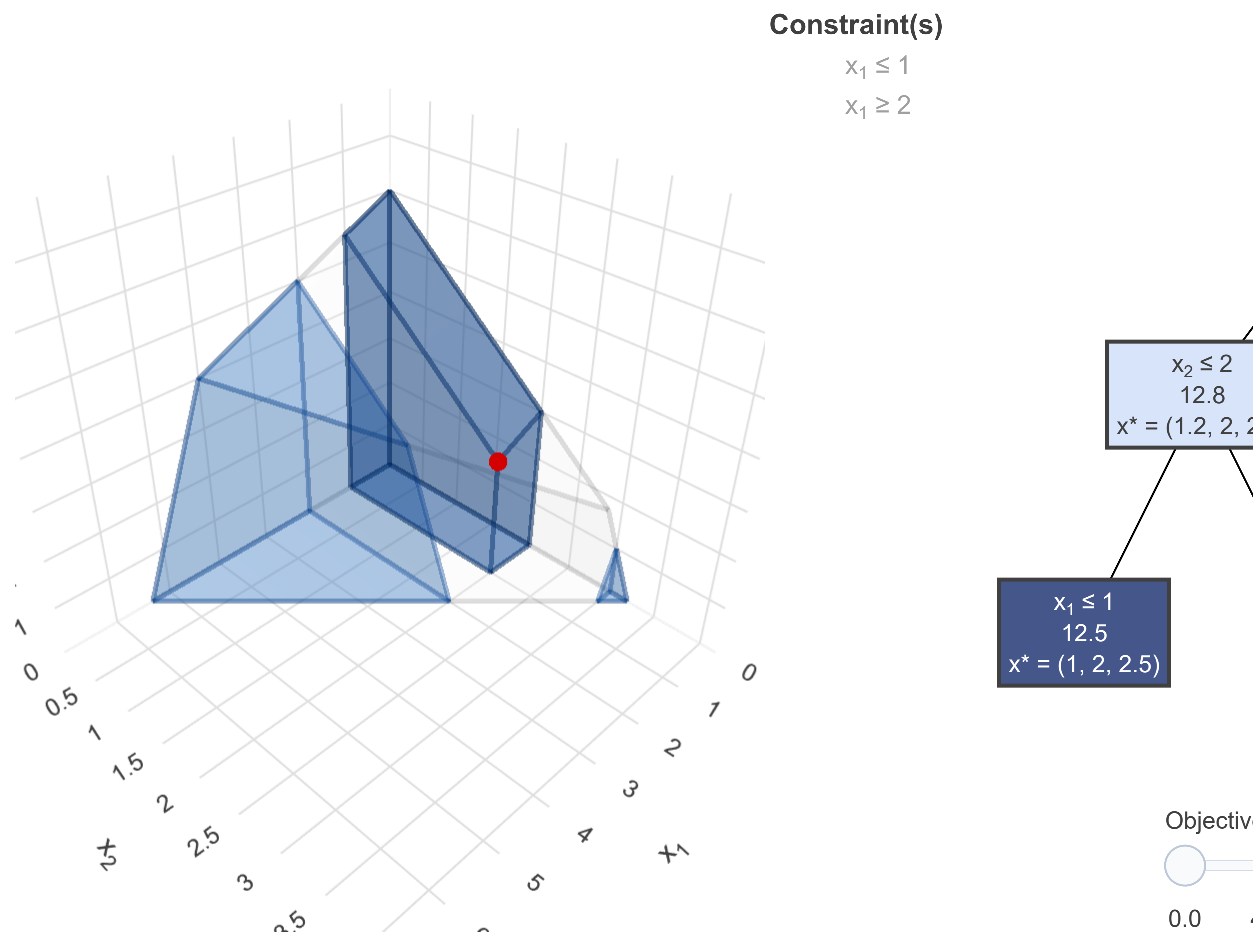



### Geometric Interpretation of LPs



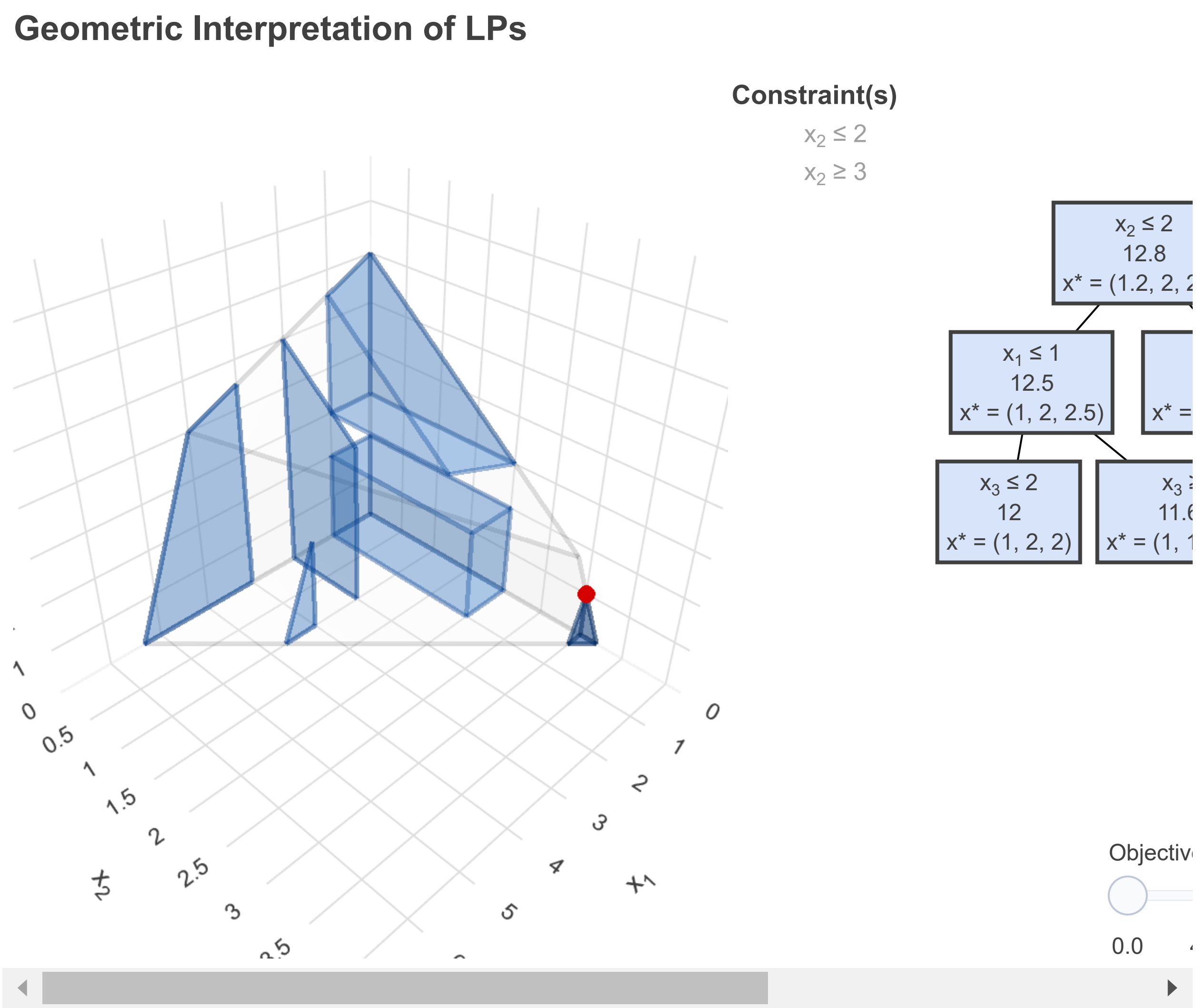



## Geometric Interpretation of LPs




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

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




**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:** leaf node all the way on left with value, optimal solution, 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:** 13 nodes. optimal solution was 13. At best 2 nodes would have to be explored to find that value 13 solution and prove optimality

# Bonus: Branch and Bound for Knapsack

Consider the following example:

| item | value | weight |
|---|---|---|
| 1 | 2 | 1 |

| item | value | weight |
|---|---|---|
| 2 | 9 | 3 |
| 3 | 6 | 2 |

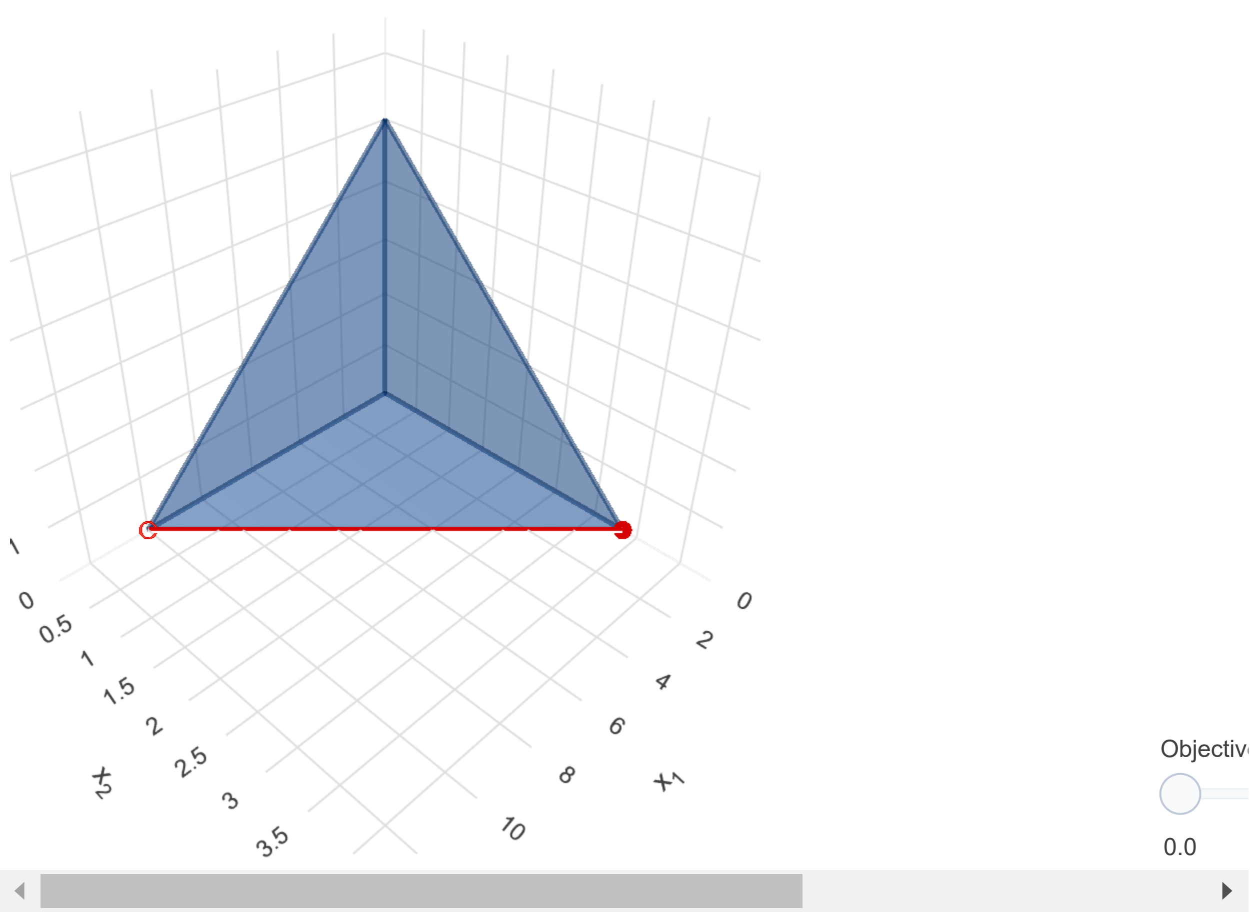
The linear program formulation will be:

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

In gilp, we can define this lp as follows:

In [53]:
```
lp = gilp.LP([[1,3,2]],
             [10],
             [2,9,6])

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

## Geometric Interpretation of LPs

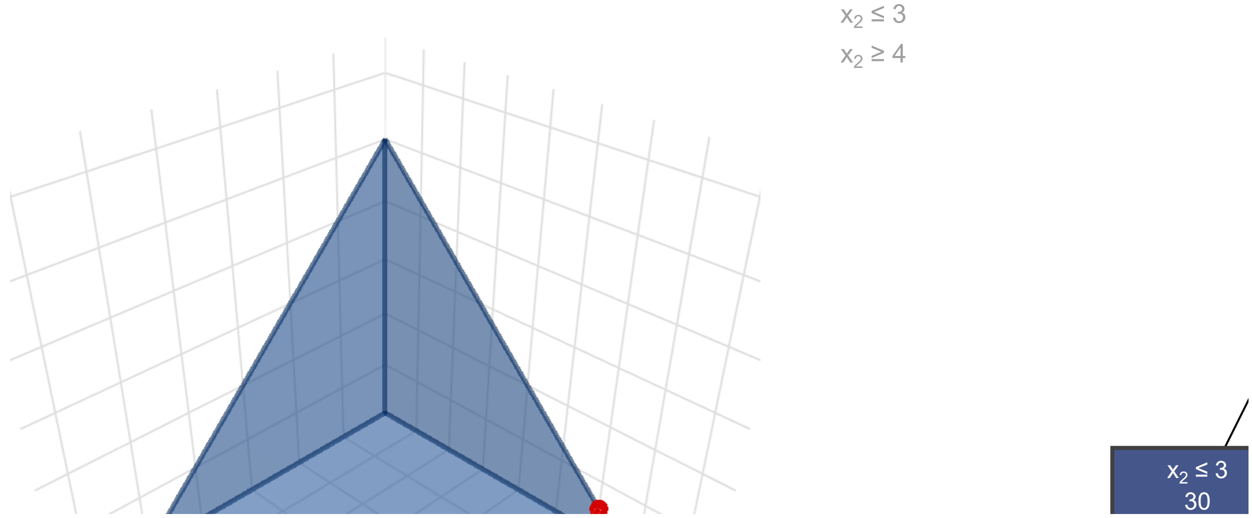


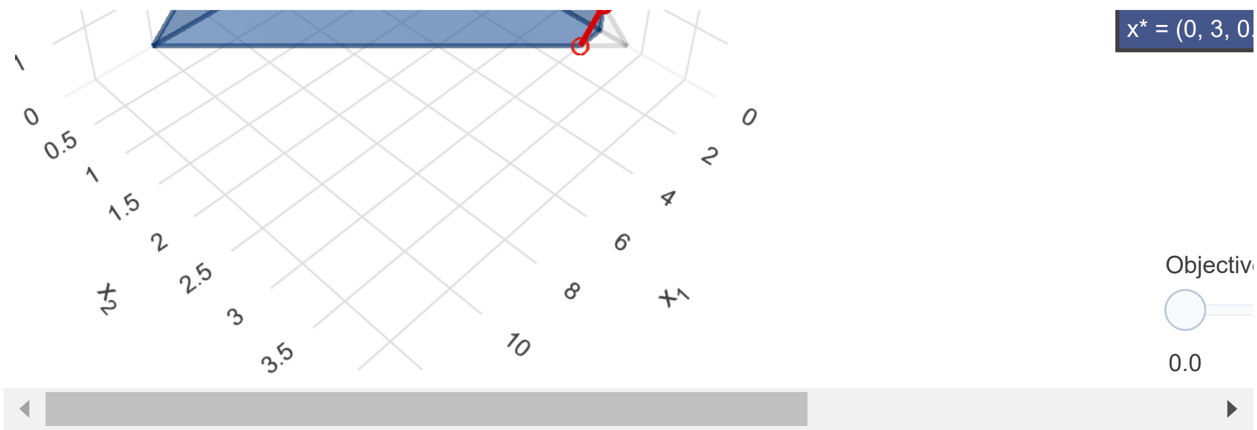

## Geometric Interpretation of LPs

**Constraint(s)**

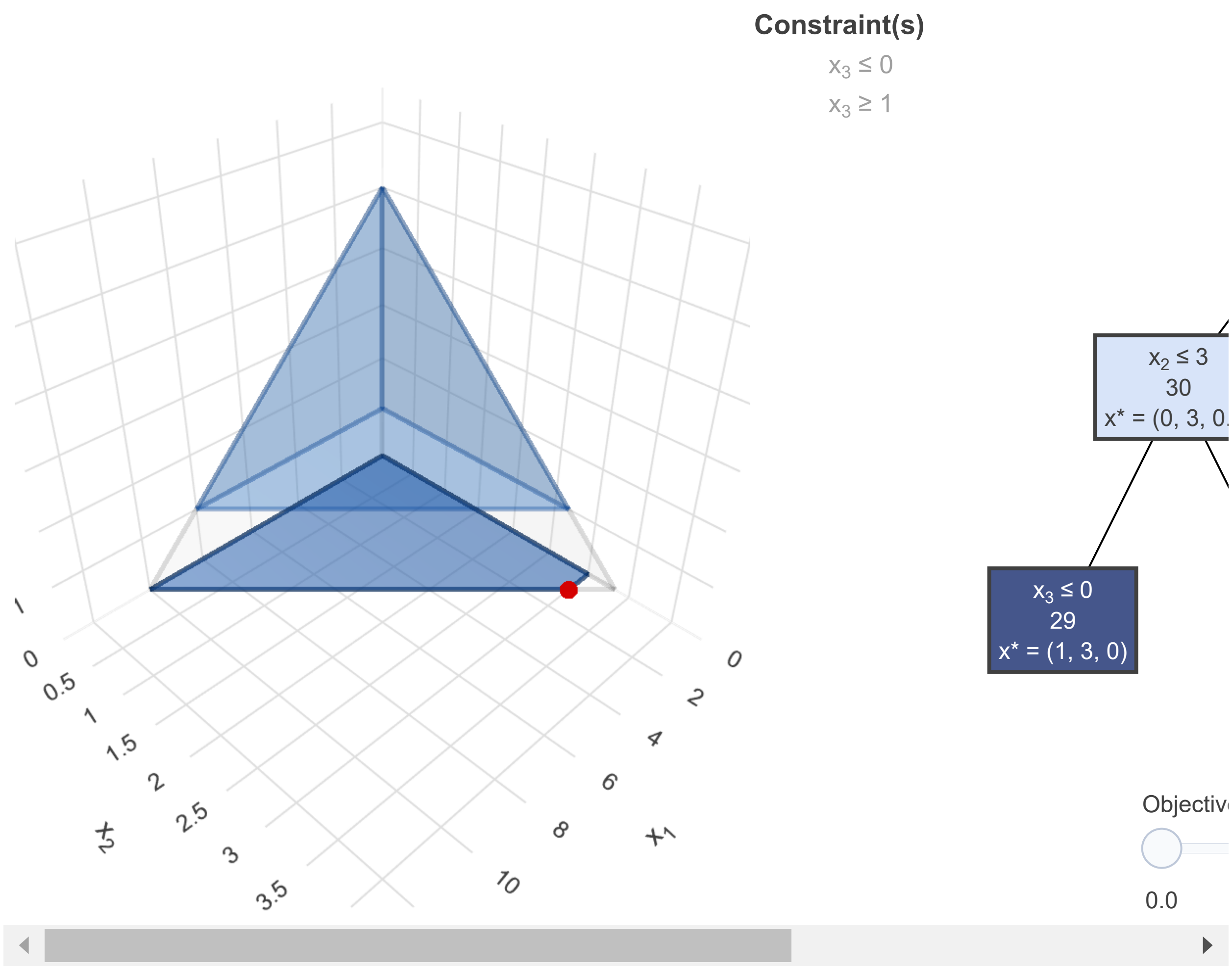
$x_2 \leq 3$

$x_2 \geq 4$

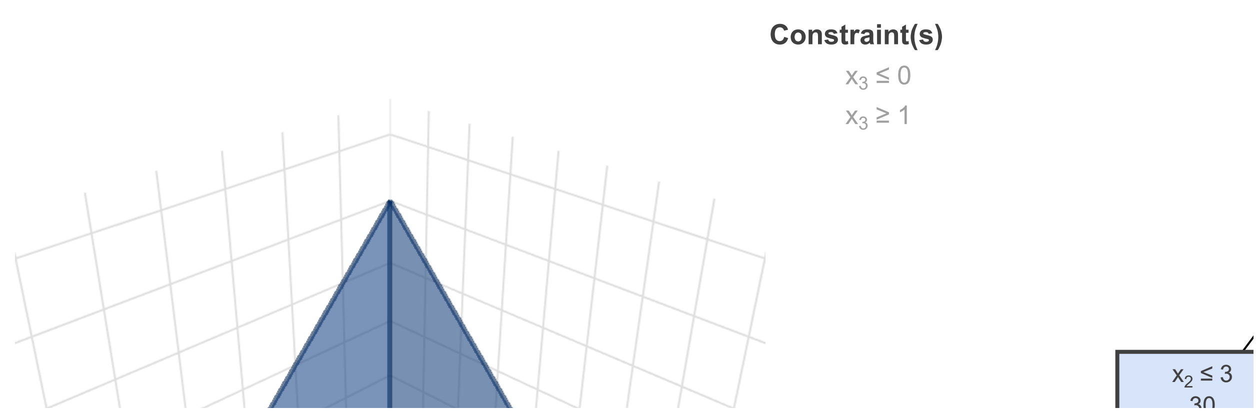


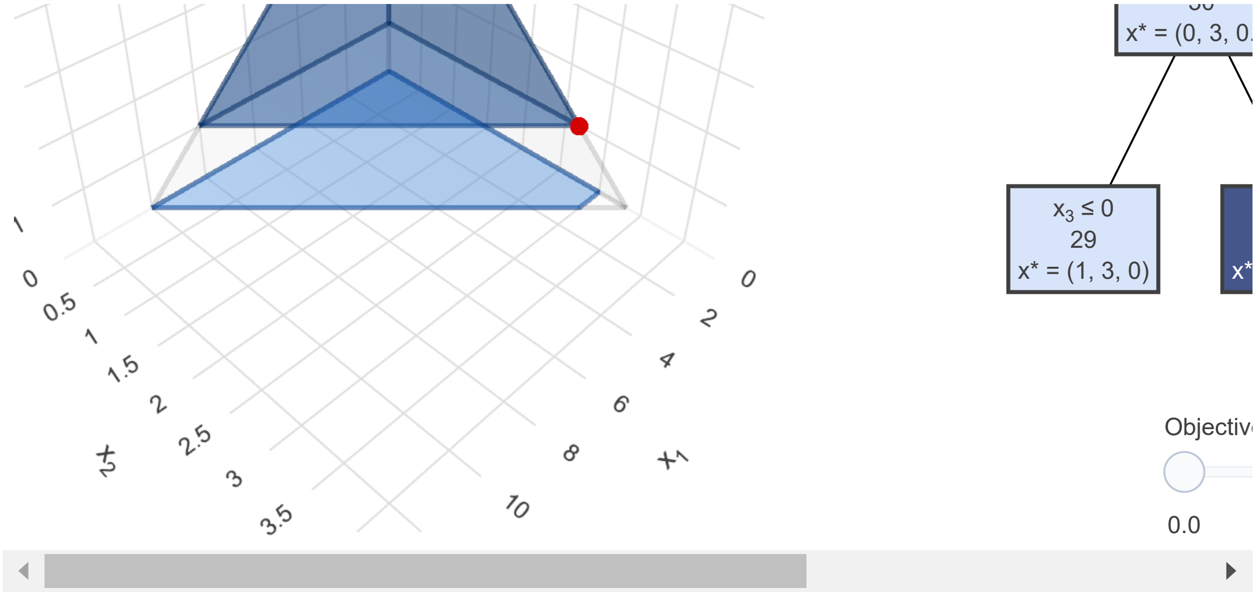



## Geometric Interpretation of LPs

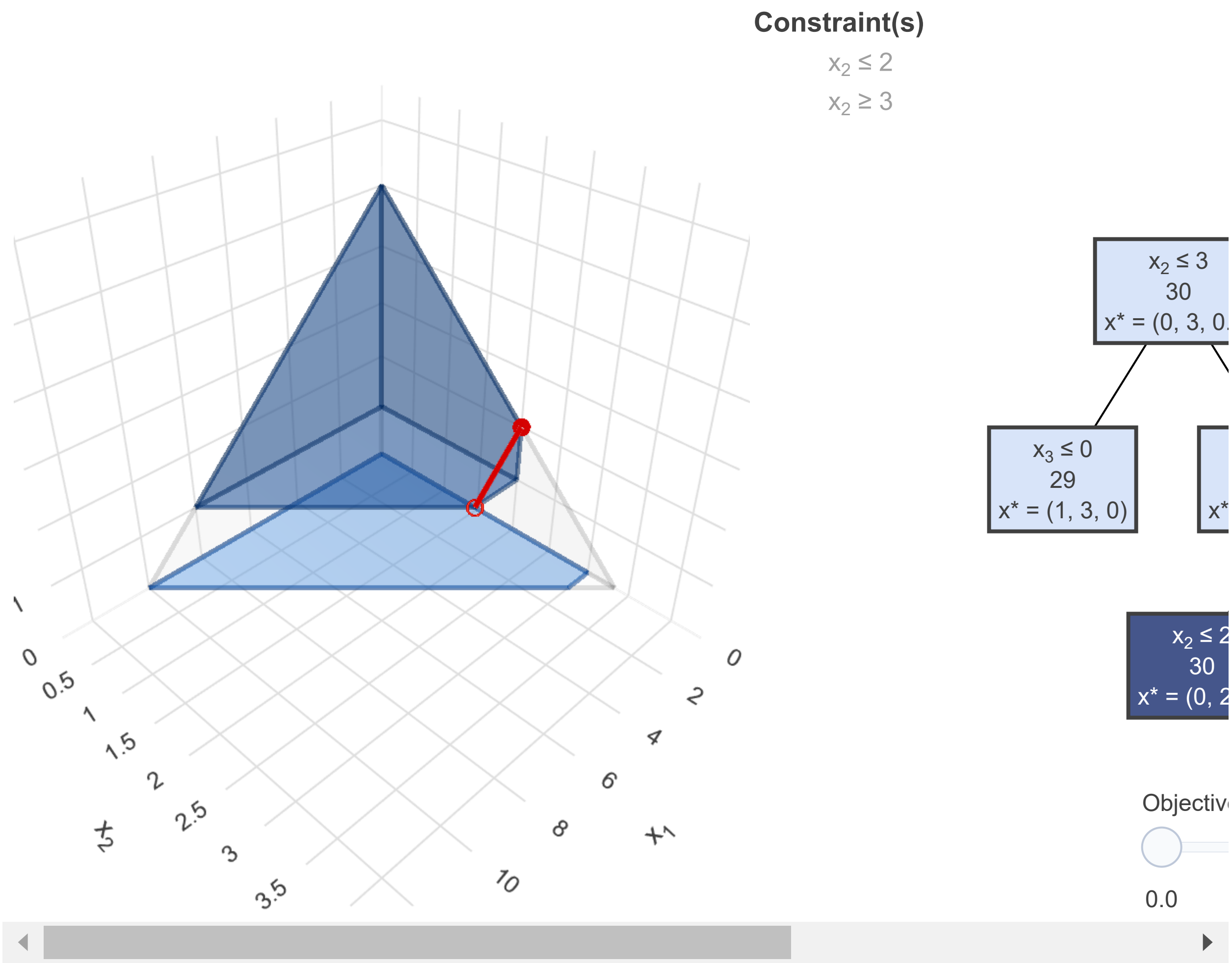



## Geometric Interpretation of LPs

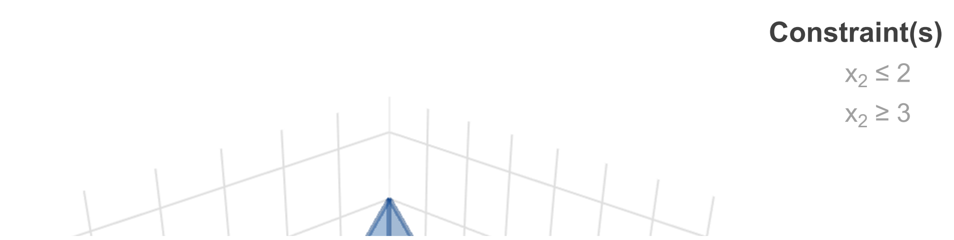


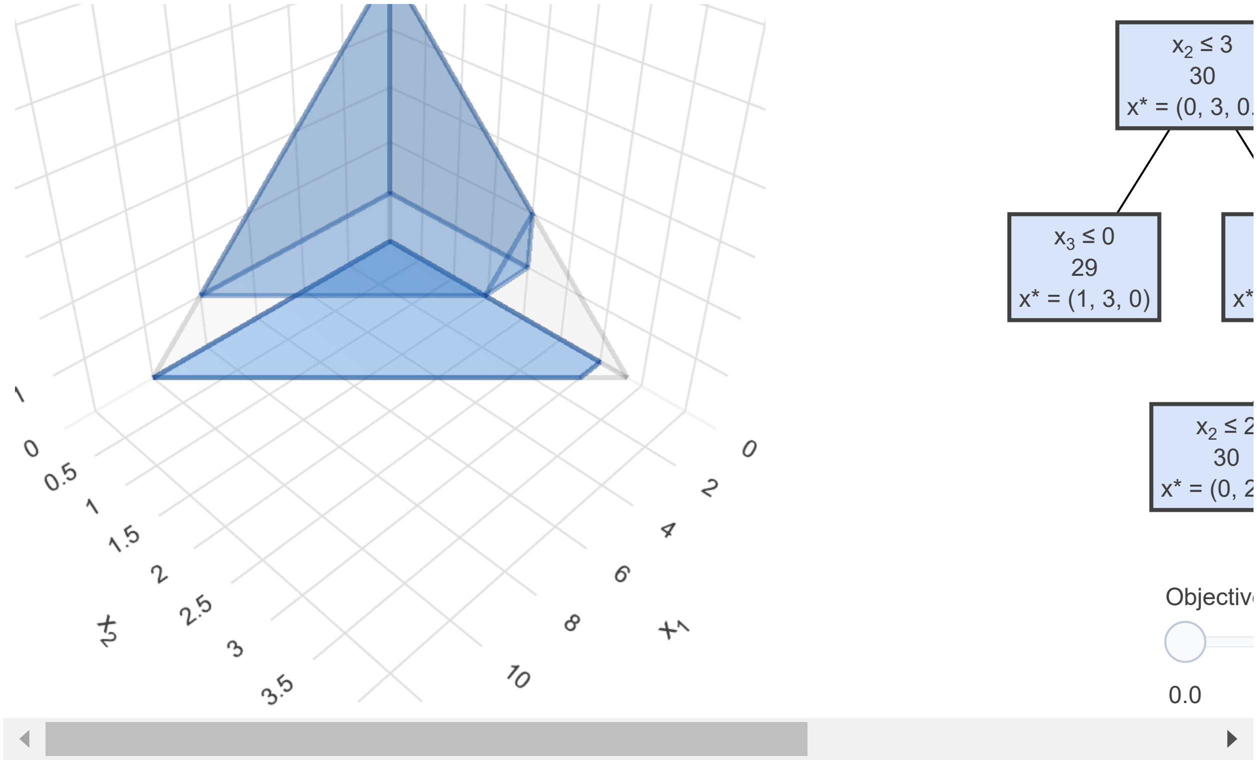



## Geometric Interpretation of LPs

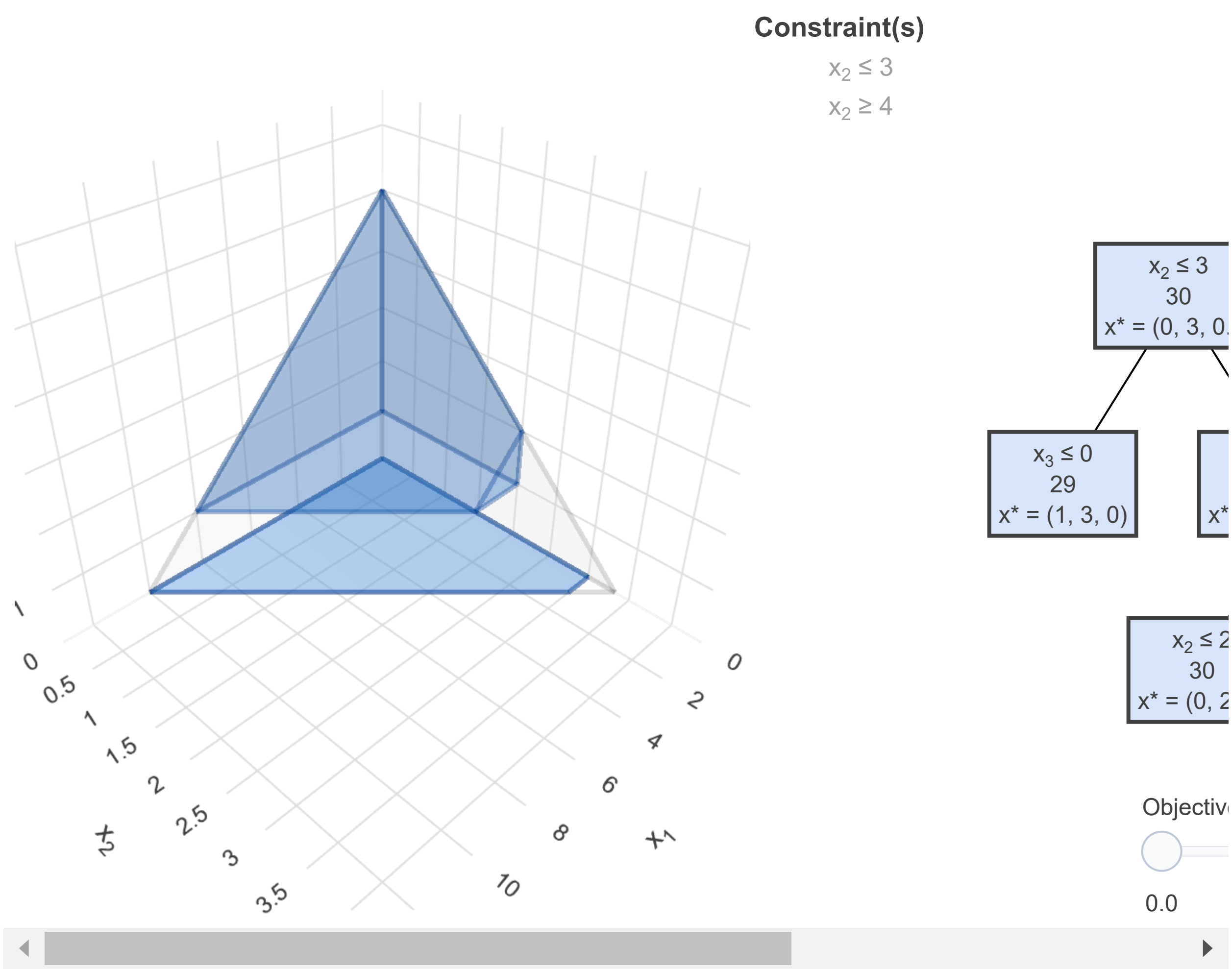



## Geometric Interpretation of LPs






## Geometric Interpretation of LPs




In [ ]: