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 th branch-and-bound algorithm. We will also see how adding a cutting plane helps in reducin computation time and effort of the algorithm. Lastly, we will explore the geometry of the bra 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 t integer programs. Before applying the branch and bound algorithm to the knapsack proble will begin by reviewing some core ideas. Furthermore, we will identify a helpful property th 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 alg Describe each.

A: A node can be fathomed when the optimal solution found is an integer, or when the opt solution found is not an integer but it is greater than an integral optimal solution found at a node.

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? is a minimization problem?

A: The LP relaxation optimal value (let's call it x) tells you that the IP optimal value is at mo a maximization problem. The LP relaxation optimal value (again, x) tells you that the IP op value is at least x for a minimizaiton problem.

Q3: Assume you have a maximization integer program with all integral coefficents in the o function. Now, suppose you are running the branch and bound algorithm and come across

with an optimal value of 44.5. The current incumbent is 44. Can you fathom this node? Wh not?

A: Yes you can fathom this node. As you add constraints, the optimal value will be at least will only go up). Since we already know one solution that is 44, we can fathom this node b will not give us any better solution

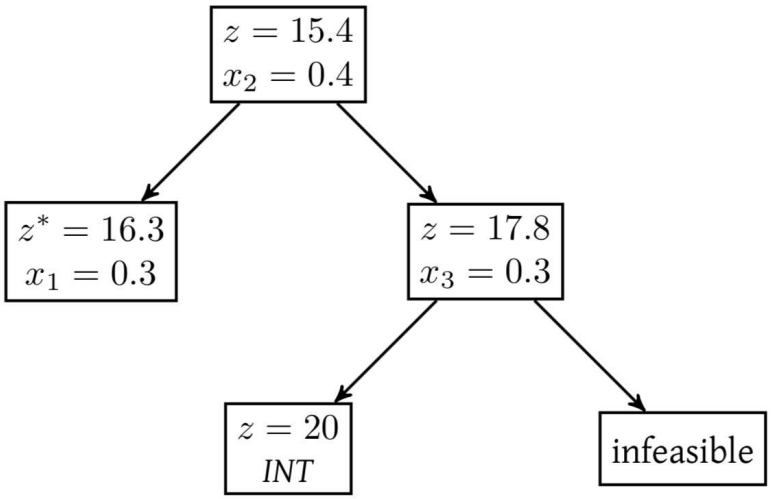
Q4: If the optimal solution to the LP relaxation of the original program is integer, then you k found an optimal solution to your integer program. Explain why this is true.

A: The IP problem has a stricter set of constraints than the LP problem. Thus, if the LP pro happens to provide an integer, there is no such way that changing any of the variables will better solution. The IP problem inherently can provide at best a solution on par with the LP problem; thus, if the LP problem already provides an integer solution, then that must be th solution.

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

A: If the LP is infeasible, then there is no solution with real numbers to it. Given that intege subset of real numbers, this means that the IP is also infeasible.

The next questions ask about the following branch and bound tree. If the solution was not the fractional x_i that was used to branch is given. If the solution was integral, it is denoted 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 minimi maximixation problem? If so, which is it?

A: It is a minimization problem because as we proceed with more nodes (and thus constra gets larger and larger. This means that we seek to minimize it.

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 co

should be imposed for each of the next two nodes?

A: It is not fathomed because the integer solution found was 20, but the solution at that no 16.3 This means that that branch could yield solutions of value 17, 18, or 19, all of which a than 20. The next constraints for the next two nodes should be x1>= 1 or x1 <= 0.

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

A: I know that the optimal value of these nodes will be at least 16.3. Since this is a minimiz problem, as we add nodes (and constraints), the optimal value will only increase. Thus the value of those nodes will be at least 16.3

Part 2: The Knapsack Problem

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

Knapsack Problem: We are given a collection of n items, where each item $i = 1, \dots, n$ h weight w_i and a value v_i. In addition, there is a given capacity W, and the aim is to selec maximum value subset of items that has a total weight at most W. Note that each item ca 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 [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 s
replace `index` with the item number that can be removed in the code below. Hint: how m
each item could we possibly take?

A: We can remove item 7 because its weight is 30 and the maximum weight that can be c
18. Thus, there is no way that item 7 can be included in the solution.

In [3]:
```
# TODO: replace index
data = data.drop(7)
data
```

Out[3]:

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

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

A: It does not change the optimal solution because the weight of 7 is 30. Since the max ca the knapsack is 18, item 7 can not be fitted into the knapsack. It cannot be in the optimal s so removing it will not change the optimal solution.

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

A: The LP relaxation's optimal value will not change. Because wi > W, including this these will not satisfy the constraint that the sum of all w must be less than or equal to W. The LP into account all constraints, so no such wi will be included in the solution to the LP relaxati such, the optimal value would not change if we take these wi out.

In **Q10-11**, you should have found that removing these items removes feasible solutions fr linear program but does not change the integer program. This is desirable as the gap betw 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 va the best feasible integer solution found thus far. Hence, it helps to have a good feasible int solution as quickly as possible (so that we stop needless work). To do this, we can first try construct a good feasible integer solution by a reasonable heuristic algorithm before startir the branch and bound procedure.

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

In [4]:
```
data['value per unit weight'] = (data['value'] / data['weight']).round(2)
data
```

Out[4]:

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 decent solution to the problem (but is not necessarily optimal).

A: A heuristic for this problem would be to maximize the count of the items with the highes per unit weight.

Q13: Run your heuristic on the data above to compute a good feasible integer solution. Yc 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: Since item 1 has the most value per weight, we should maximize the number of item 1s Evidently we can only fit 1 of them in the knapsack. Then, we would have a weight of 8 rer meaning we could fit one of items 4, 5, or 6. It would be most advantageous to fit item 4. T solutions yields a weight of 17 and a value of 64.

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

Q14: Complete the model below.

In [5]:
```
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 v
        capcity (int): An integer-capacity for the knapsack
        integer (bool): True if the variables should be integer. False ot
    """
    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 c
    # recall that we add constraints to the model using m.Add()
    m.Add(sum(w[i]*x[i] for i in ITEMS) <= capacity)

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

In [6]:
```
# 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 n and `x` represents our decision variables.

In [7]:
```
m, x = Knapsack(data, 18)
```

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

In [8]:
```
solve(m)
```

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

Out[8]:
```
{'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 inteç solution?

A: The optimal value is 70, 6 more than my solution.

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

A: This node should not be fathomed. The variable x2 should be branched on with the cor x2 <= 0 or 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 follc

In [9]:
```
m, x = Knapsack(data, 18)
m.Add(x[2] <= 0)
solve(m)
```

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

Out[9]:
```
{'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 relaxatio 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 **Q**

In [10]:
```
m, x = Knapsack(data, 18)
m.Add(x[2] >= 1)
solve(m)
```

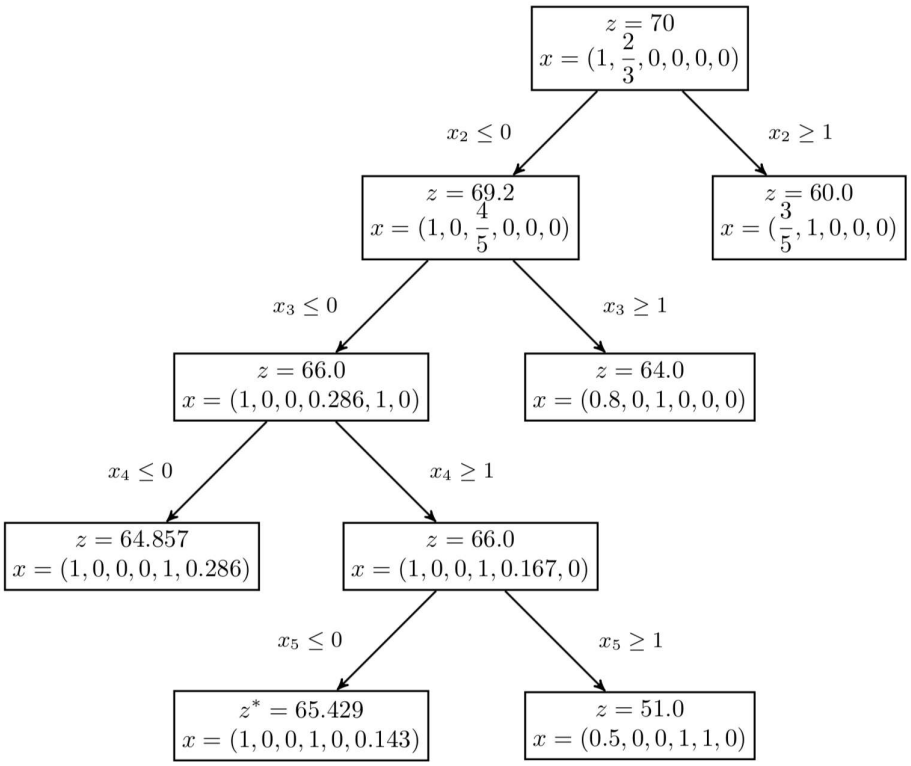
```
Objective = 60.0
iterations : 0
branch-and-bound nodes : 0
```

Out[10]:
```
{'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 f feasible integer solution with value 64.)

A: The optimal value is 60.0. This can be fathomed because we know there is a feasible ir solution of value 64. Continuing on from this node will only yield optimal values that decrea value from 60. Thus, it can be fathomed.

If we continue running the branch and bound algorithm, we will eventually reach the branc 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? think back to **Q3**)

A: We can fathom this node because we know that there exists a solution with optimal val This node can at best yield an optimal solution of 64, so continuing will not add any new information. We hope to find a node that yields an optimal value of at least 65, as this wou new optimal solution.

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

In [15]:
```
# Template
m, x = Knapsack(data, 18)
# Add constraints here
m.Add(x[5]<=0)
m.Add(x[2] <=0)
m.Add(x[3]<=0)
m.Add(x[4]>=1)
m.Add(x[6]<=0)

solve(m)
# fathomed?
```

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

Out[15]:
```
{'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 [ ]:

In [ ]:

A: We can now fathom this node because we have reached an integer solution of 64. The there is no better solution than 64.

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

A: We had to explore a total of 5 nodes.

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

Part 3: Cutting Planes

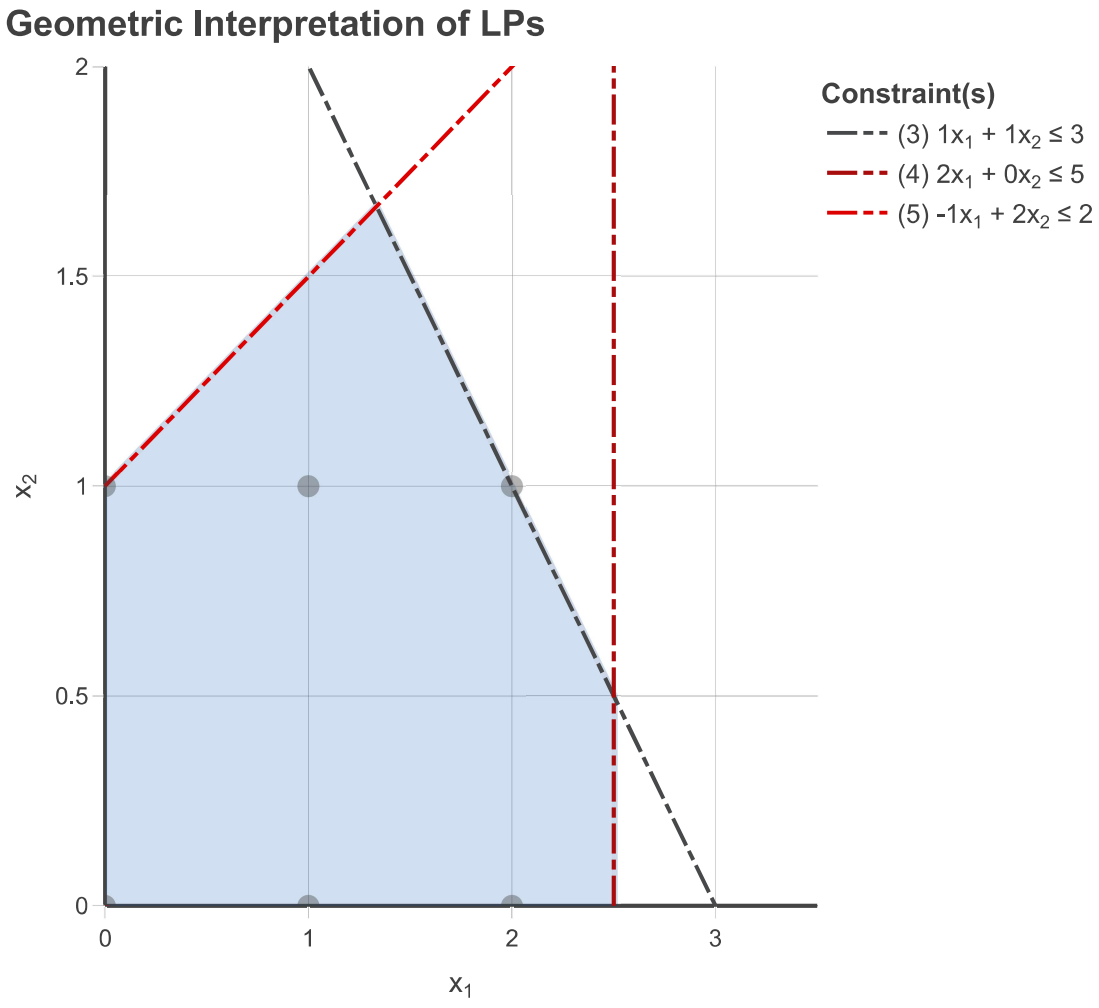
In general, a cutting plane is an additional constraint we can add to an integer program's li relaxation that removes feasible linear solutions but does not remove any integer feasible solutions. This is very useful when solving integer programs! Recall many of the problems learned in class have something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

In [18]:
```
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.

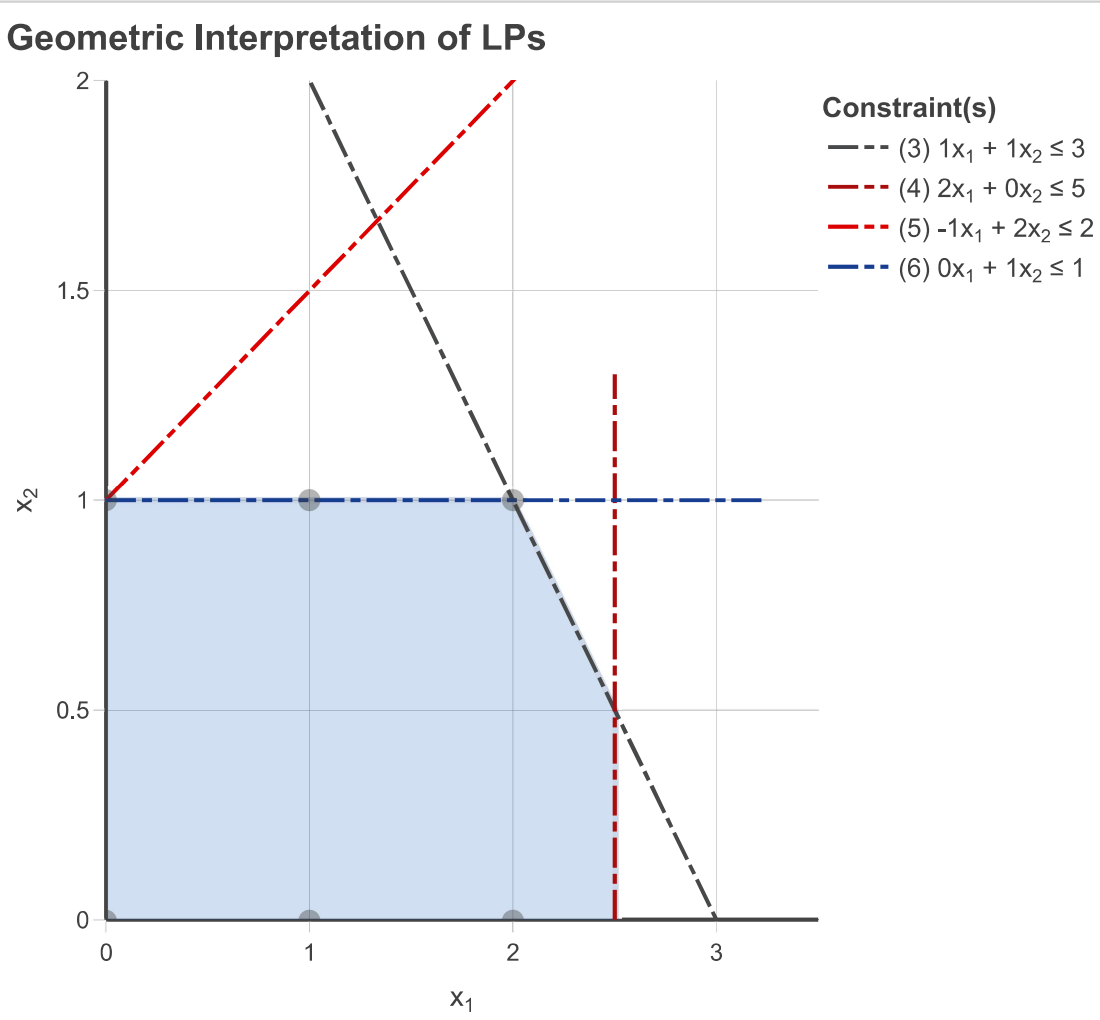
A: (0,0), (0,1), (1,0), (1,1), (2,0), (2,1)

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

A: No feasible solutions would become infeasible, so it is a cutting plane. Feasible linear p become infeasible, but we are only interested in integer points, so it is still valid to add the constraint.

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

In [19]:
```
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?

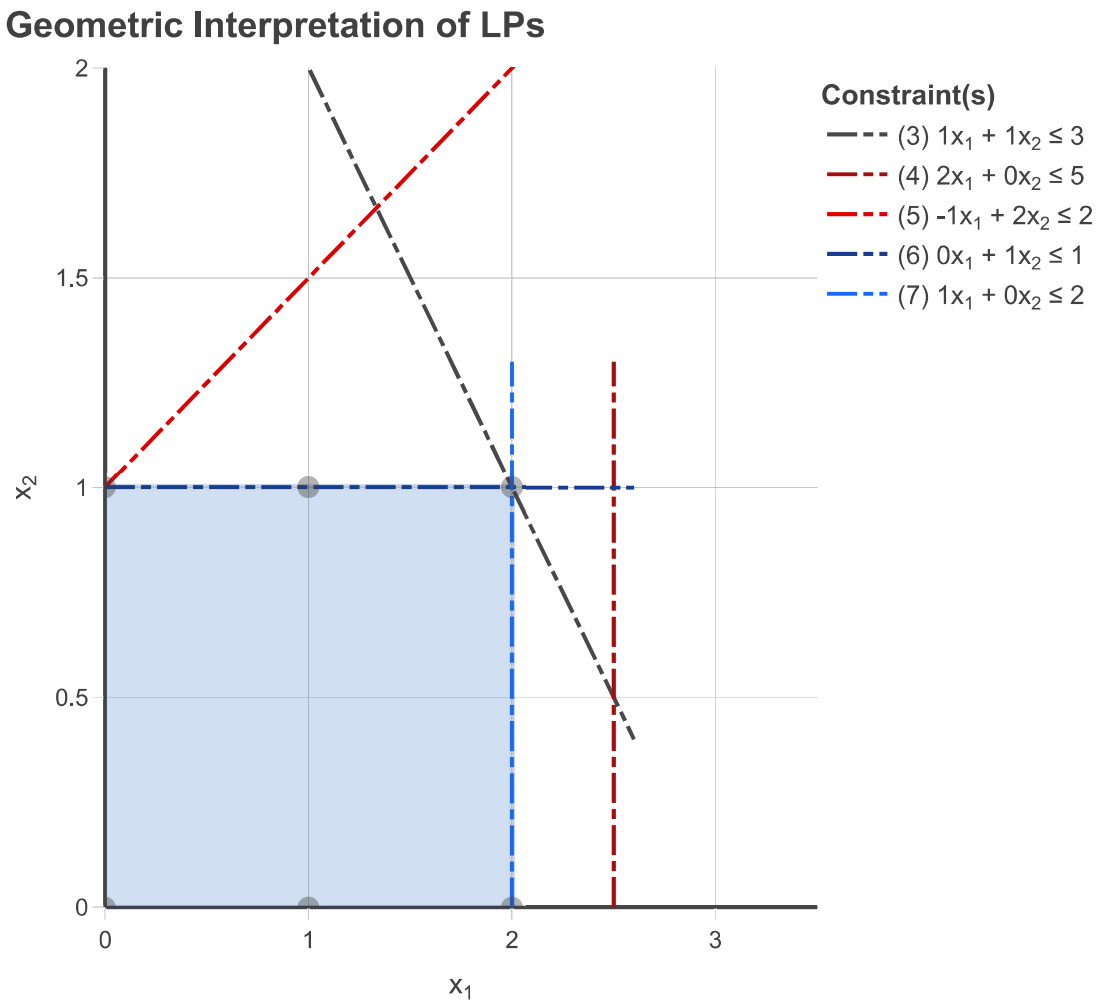
A: This is not a cutting plane. Adding this constraint does not remove any feasible linear s since it is simply a redundant constraint. A cutting plane should remove feasible linear solu but not any feasible integer solutions, and x1 <= 3 does not satisfy that definition.

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

A: Yes I can. x1 <= 2

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

In [20]:
```
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, reca
input was $W = 18$ and:

In [21]:
```
data
```

Out[21]:

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? Ca write a new constraint to capture this? If so, please provide it.

A: We can only take one of item 1, item 2, or item 3. The constraint would be x1 + x2 + x3

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

A: Yes it is a cutting plane. This constraint cut's off the feasible solution x1 = 1 and x2 = 2/ has an optimal value of 70).

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

A: At most 2 of items 4, 5, and 6 can be included. The constraint for this would be x4 + x5 2.

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.

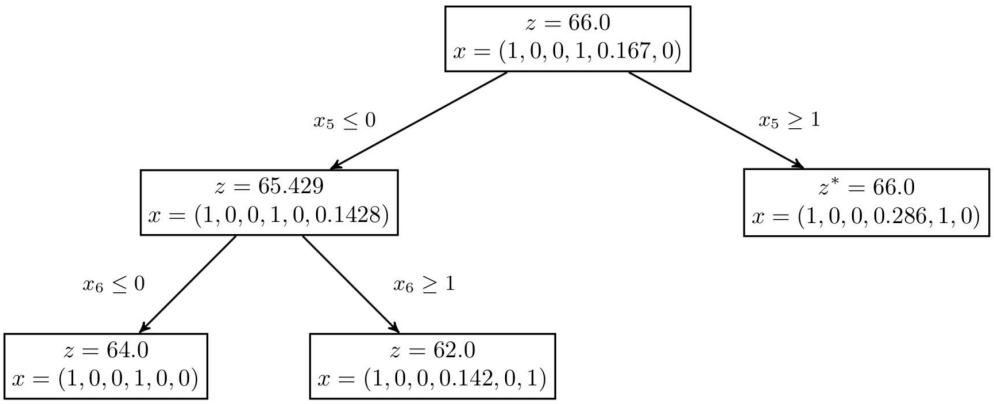
In [22]:
```
m, x = Knapsack(data, 18)
# TODO: Add cutting planes here
m.Add(x[1] + x[2] + x[3] <= 1)
m.Add(x[4] + x[5] + x[6] <= 2)
solve(m)
```

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

Out[22]:
```
{'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. Essentia added the constraint $x_7 \leq 0$. This constraint was a cutting plane! It eliminated some linea solutions but no integer ones. By adding these two new cutting planes, we can get branch bound to terminate earlier yet again! So far, we have generated cutting planes by inspectic 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 branc 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 6

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

In [23]:
```
# Template
m, x = Knapsack(data, 18)
# Add constraints here
m.Add(x[1] + x[2] + x[3] <= 1)
m.Add(x[4] + x[5] + x[6] <= 2)
m.Add(x[5] >= 1)
m.Add(x[4] <= 0)
solve(m)
# fathomed?
```

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

Out[23]:
```
{'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 [ ]:

In [ ]:

A: This node has reached an optimal solution of 64.85. Since the optimal solution will be a
64, we can fathom this node.

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

A: We explored two nodes and found that this node can be fathomed because it yields an
value of 64.85. Thus, continuing onwards would give an integer optimal value of at best 64
we have already found. It took two nodes instead of five.

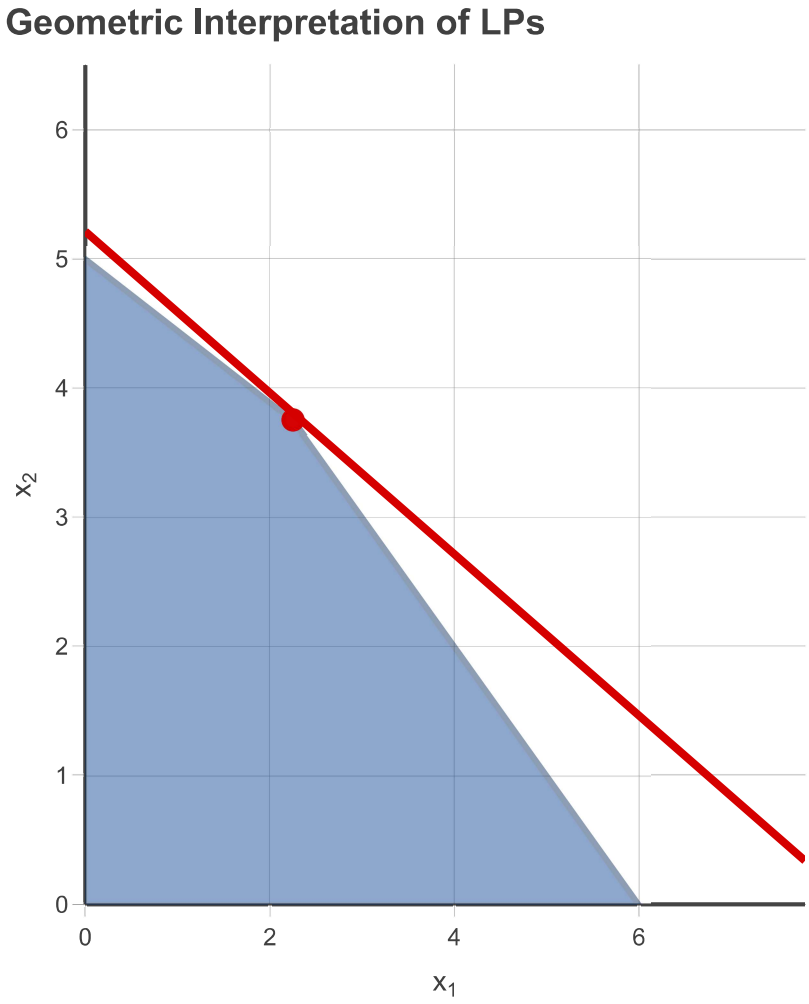
Part 4: Geometry of Branch and Bound

Previously, we used the `gilp` package to viusualize the simplex algorithm but it also has
functionality to visualize branch and bound. We will give a quick overview of the tool. Simil
`lp_visual` and `simplex_visual`, the function `bnb_visual` takes an `LP` and returns
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 bran
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 [24]:
```
nodes = gilp.bnb_visual(gilp.examples.STANDARD_2D_IP)
```

In [25]:
```
nodes[0].show()
```



Run the cells above to generate a figure for each node and view the first node. At first, you the LP relaxation on the left and the root of the branch and bound tree on the right. The sir 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 solution? (Hint: Use the objective slider and hover over extreme points).

A: THe optimal solution is (2.25,3.75) and has optimal value of 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: max 5x1 + 8x2

s.t x1 + x2 <= 6

5x1 + 9x2 <= 45

x1 <= 2

x1, x2 >= 0

max 5x1 + 8x2

s.t x1 + x2 <= 6

5x1 + 9x2 <= 45

x1 >= 3

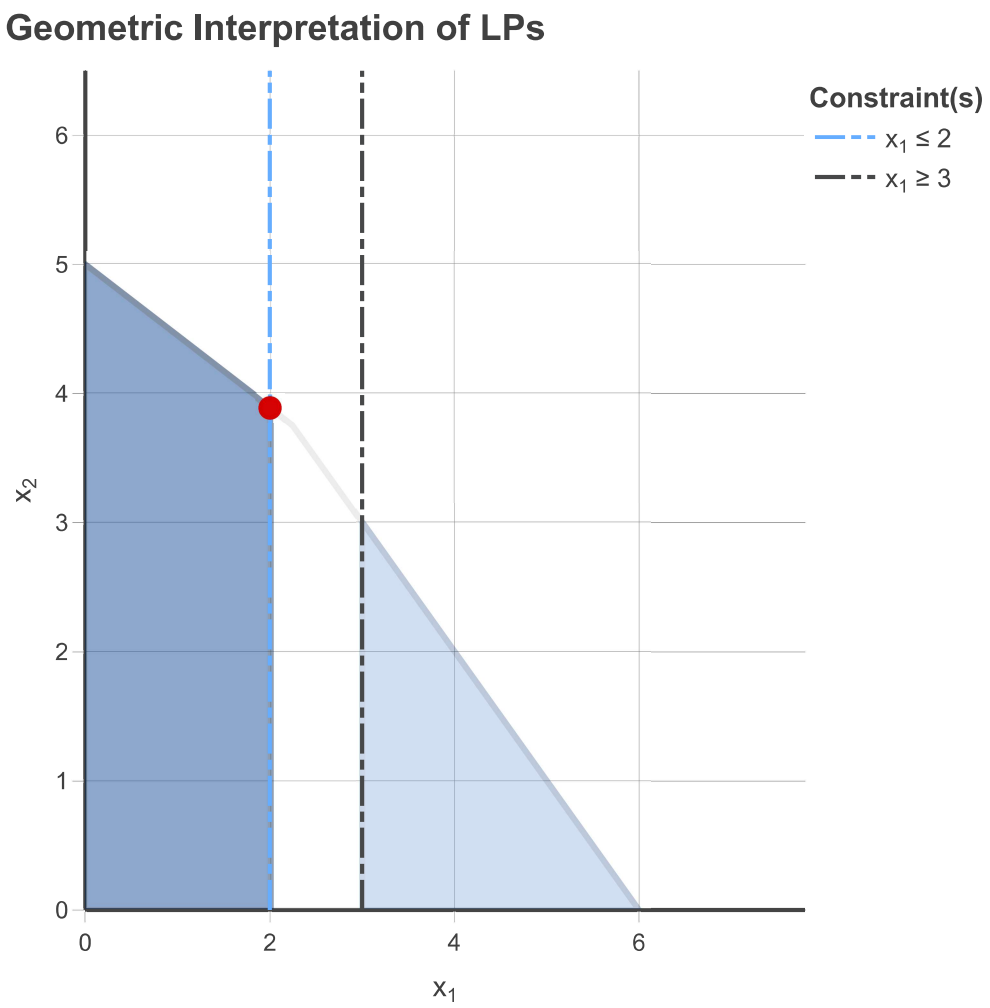
x1, x2 >= 0

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

A: See attached page.

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

In [26]:
```
nodes[1].show()
```



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

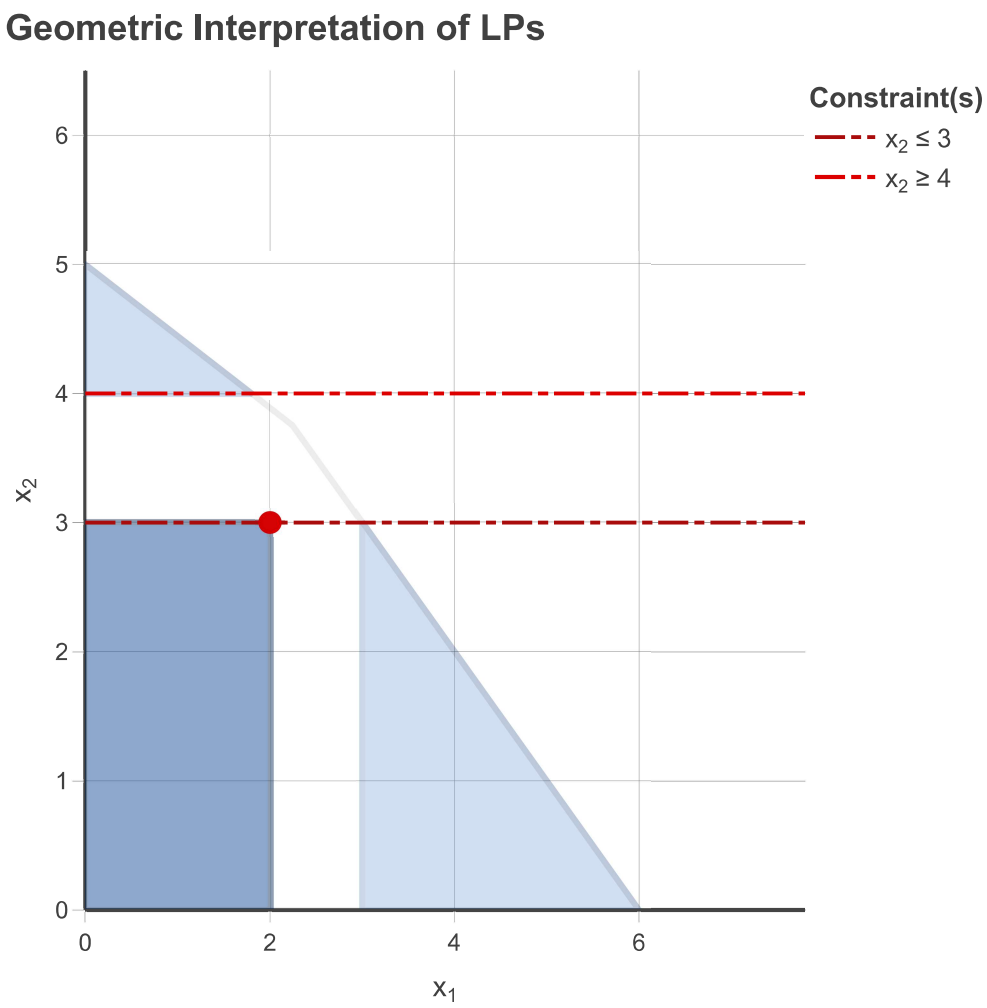
Q35: Which feasible solutions to the LP relaxation are removed by this branch?

A: It removed all feasible solutions with x1 >= 2

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

A: We will add the constraint x2 <= 3 and x2 >= 4. This will break the original LP relaxatior regions.

In [27]:
```
nodes[2].show()
```



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

A: The optimal solution at the dark node is (2,3) and has an objective value of 34. We do r to further explore this branch because we have reached an integral soution.

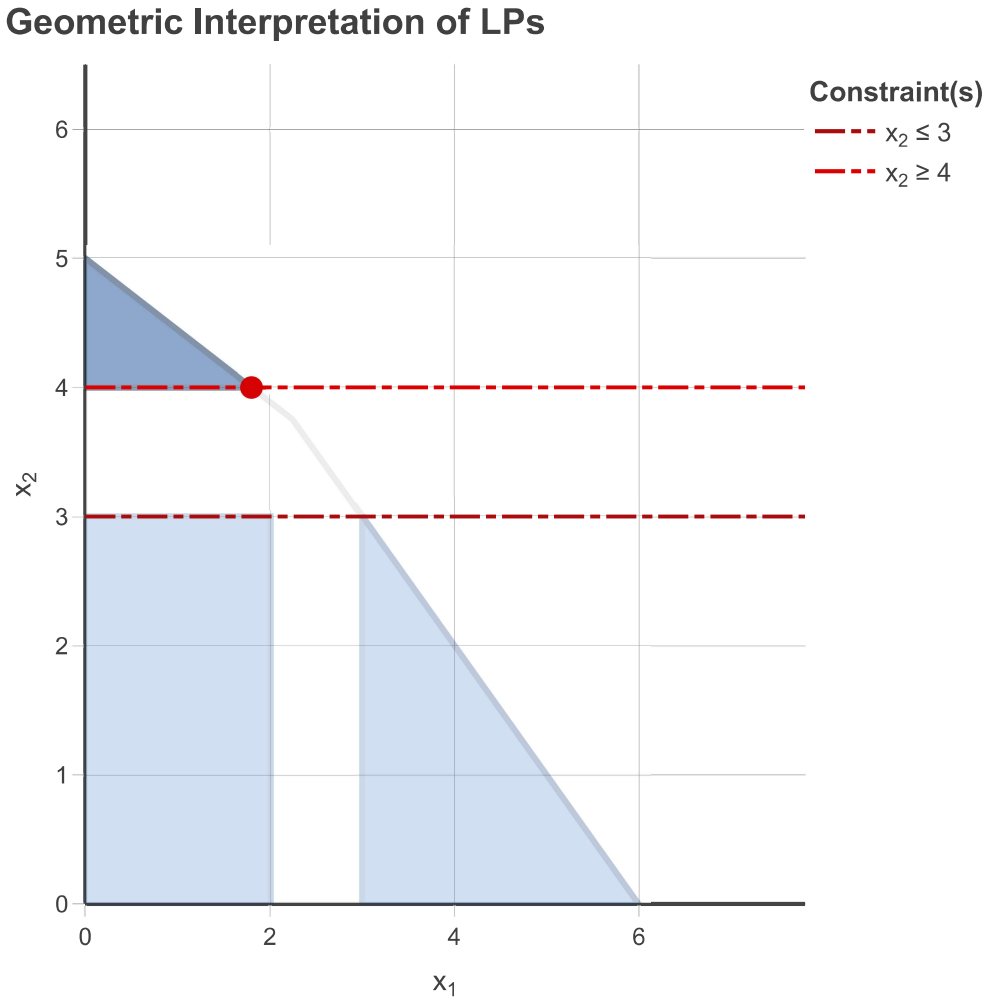
Q38: Recall shaded nodes have been explored and the node shaded darker (and feasible shaded darker) correspond to the current node and its feasible region. Nodes not shaded been explored. How many nodes have not yet been explored?

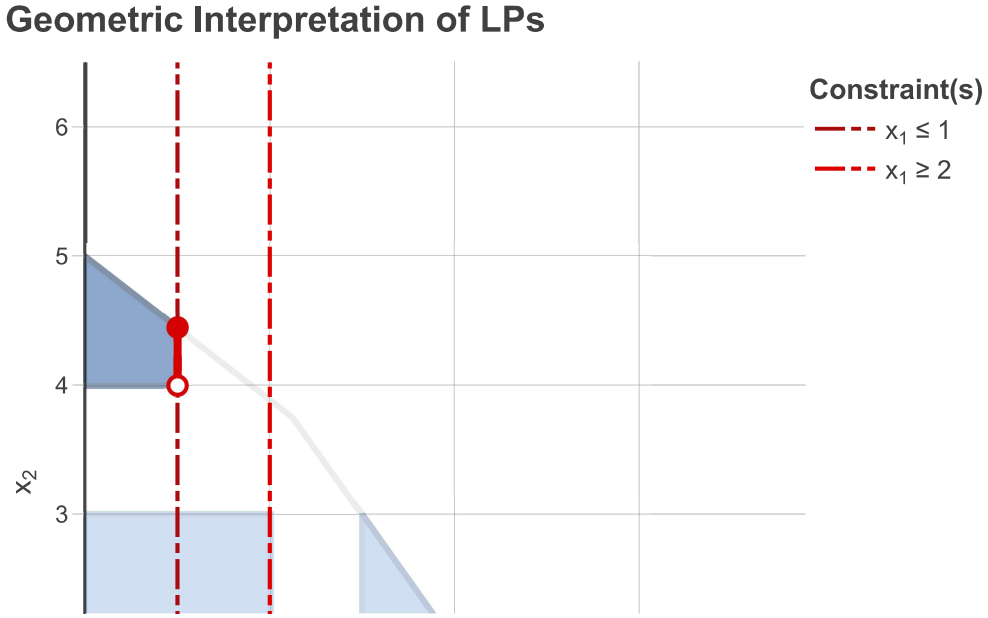
A: Two nodes have not yet been explored.

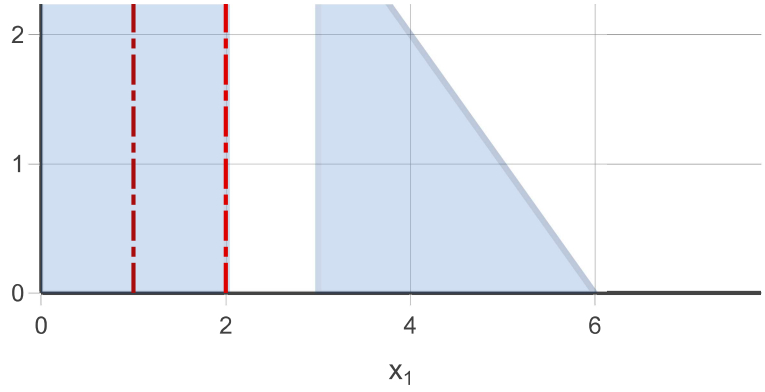
Q39: How many nodes have a degree of one in the branch and bound tree? (That is, they connected to one edge). These nodes are called leaf nodes. What is the relationship betw leaf nodes and the remaining feasible region?

A: There are three leaf nodes. The relationship is that the number of leaf nodes is equivale
portions of the feasible region.

In [28]:
```
# Show the next two iterations of the branch and bound algorithm
nodes[3].show()
nodes[4].show()
```



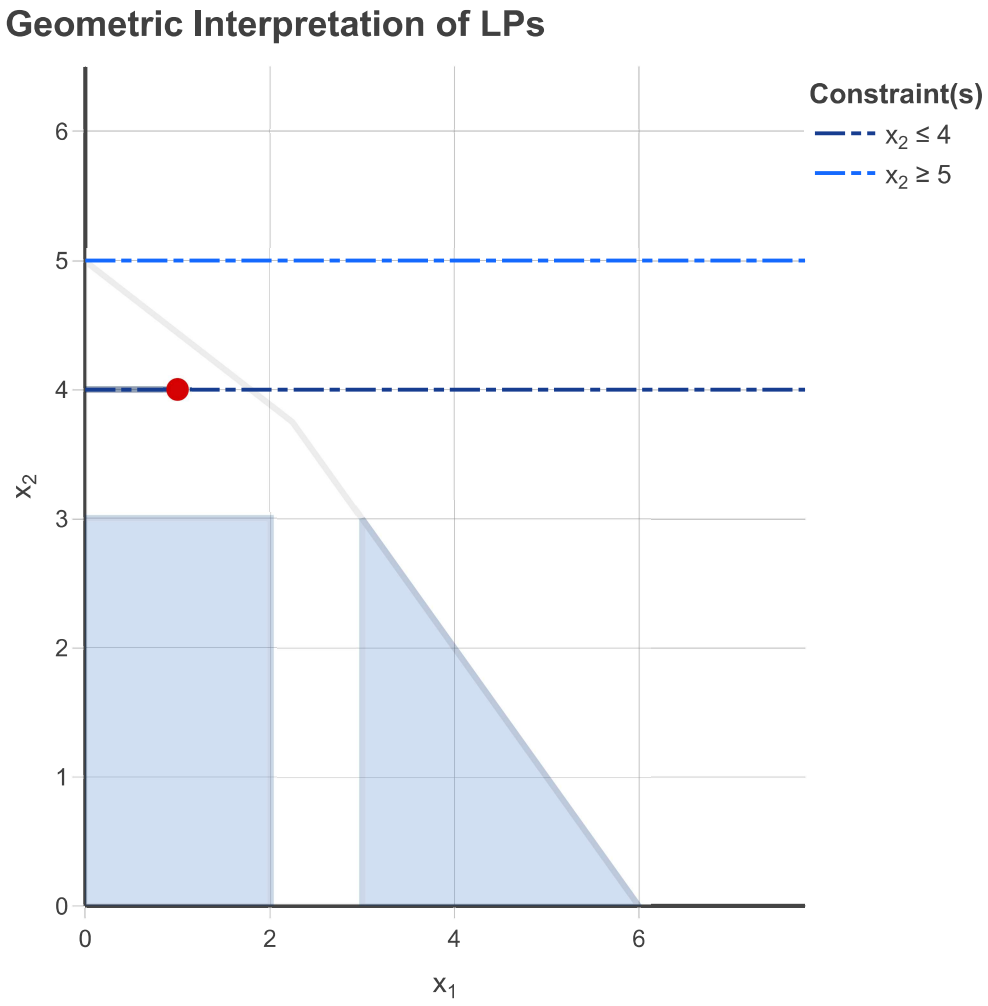


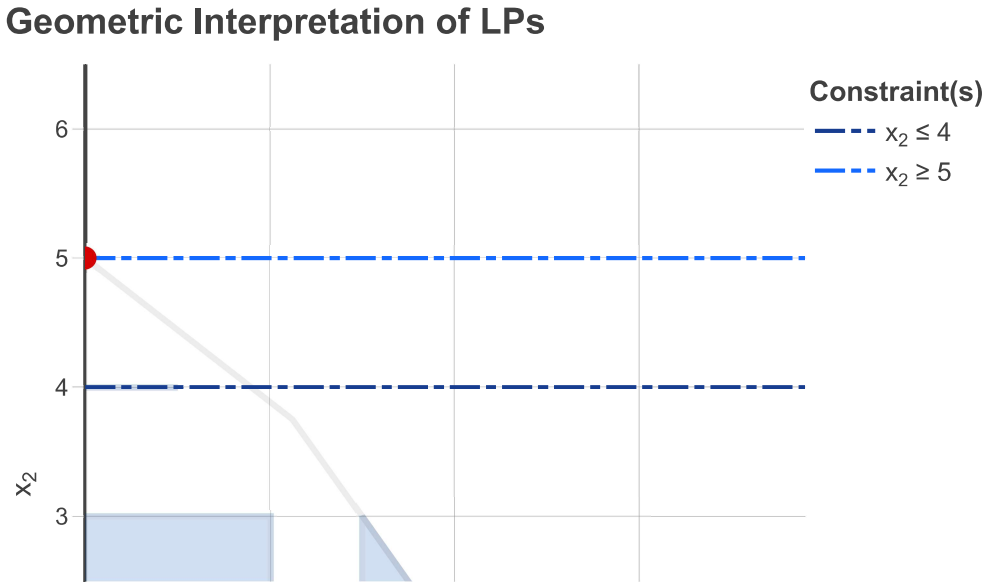


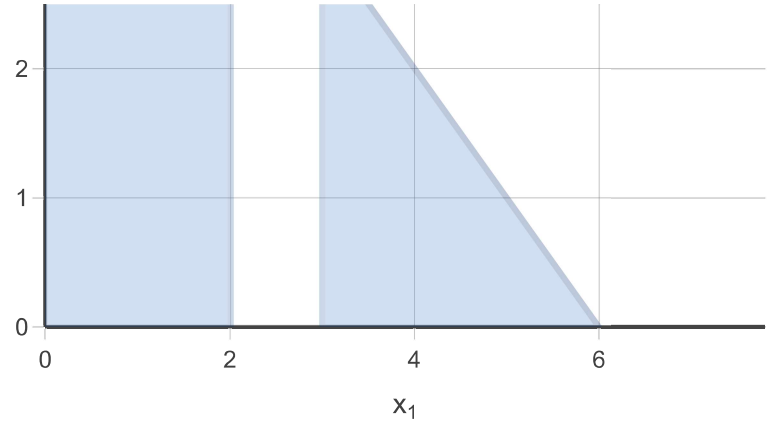
Q40: At the current (dark) node, we added the constraint $x_1 \leq 1$. Why were the fractional solutions $1 < x_1 < 2$ not eliminated for $x_2 <= 3$?

A: They were not eliminated because it was not necessary to eliminate them. The optimal to the LP when x2 <= 3 yielded an x1 value of 2. Therefore, it was not necessary to write a constraint.

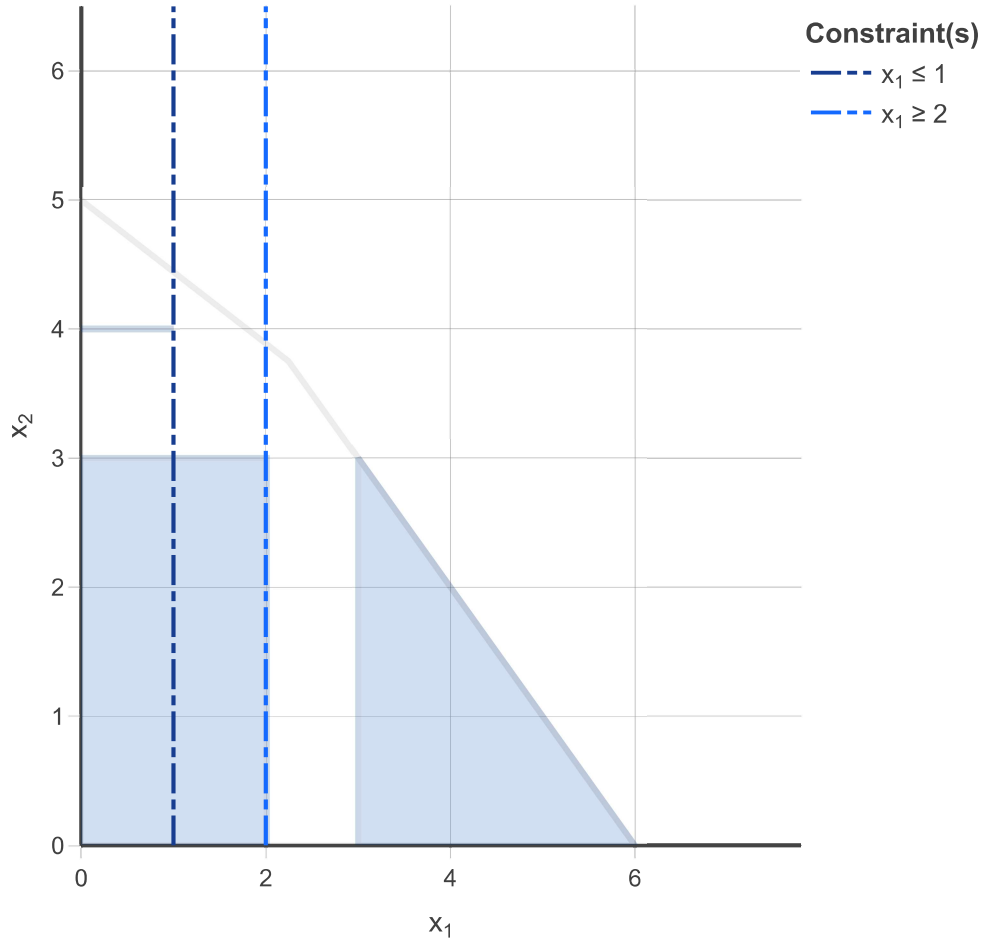
In [29]:
```
# Show the next three iterations of the branch and bound algorithm
nodes[5].show()
nodes[6].show()
nodes[7].show()
```







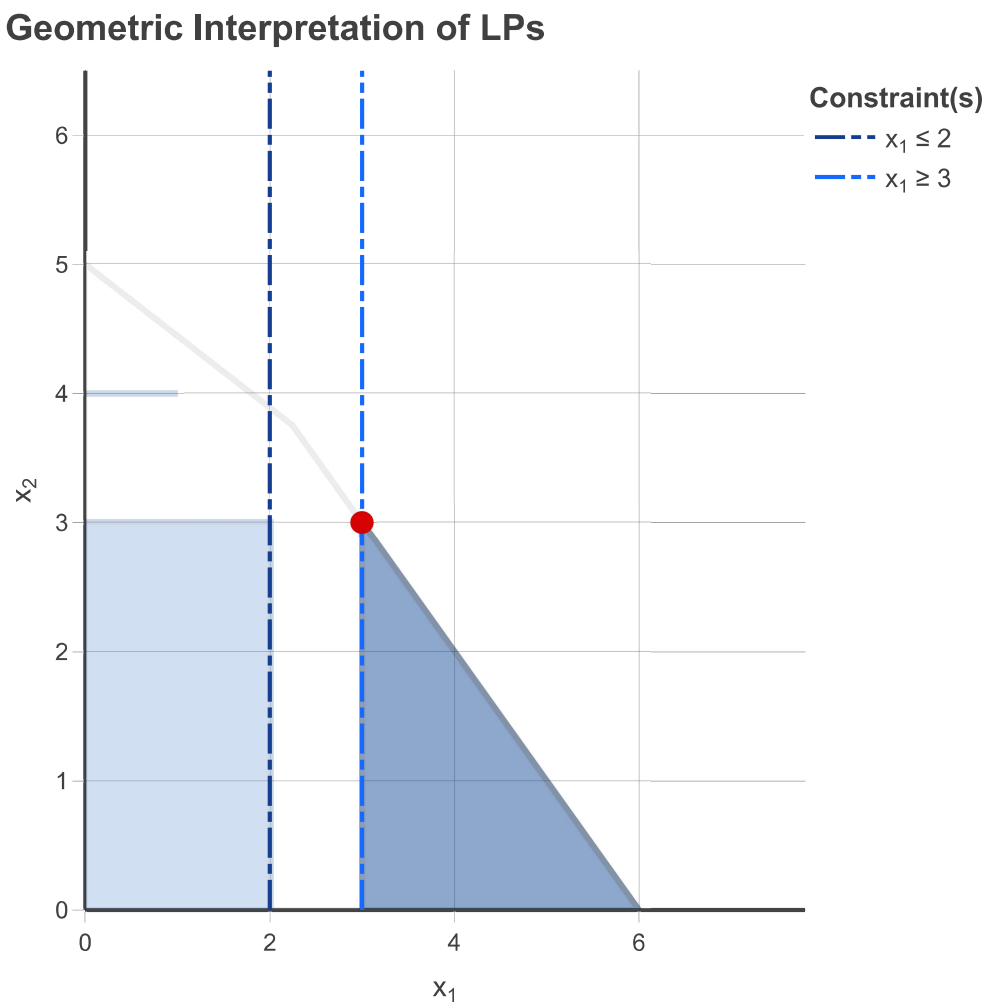
Geometric Interpretation of LPs



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

A: One constraint of the original IP was 5x1 + 9x2 <= 45. When we have the constraints x x1 >= 2, and x1 <= 4, then the minimum solution we could have (2,4) does not satisfy this constraint. Therefore, any solution from this node and onwards would be infeasible.

In [30]:
```
nodes[8].show()
```



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

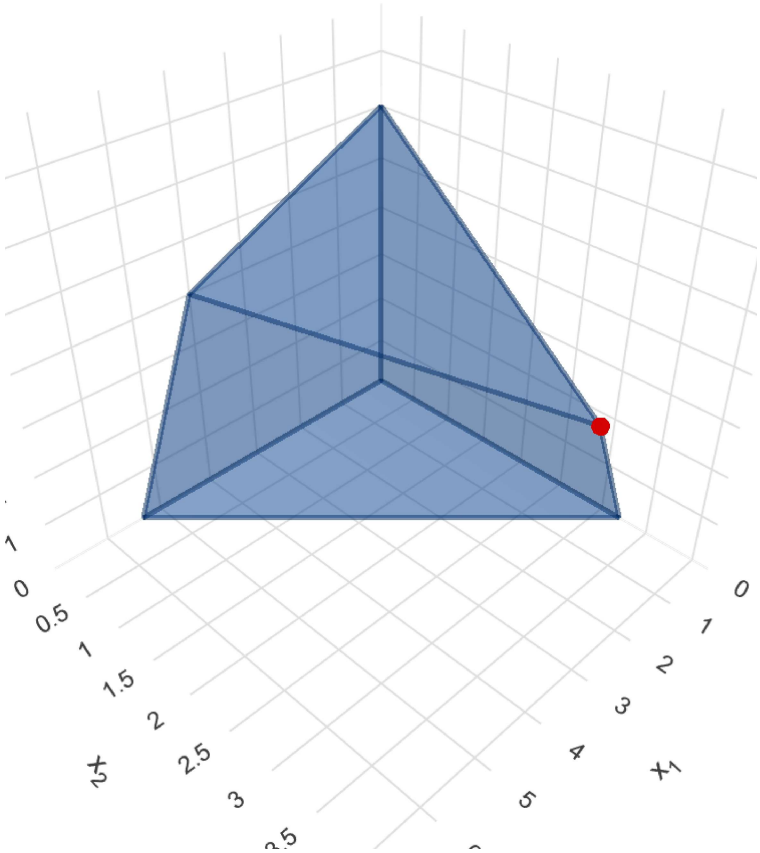
A: We are done. We have fathomed every node due to infeasible solutions or integral opti solutions. In this case, the optimal soluiton is (0,5) and has value 40.

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

In [31]:
```
nodes = gilp.bnb_visual(gilp.examples.VARIED_BRANCHING_3D_IP)
```

In [32]:
```
# Look at the first 3 iterations
nodes[0].show()
nodes[1].show()
nodes[2].show()
```

Geometric Interpretation of LPs

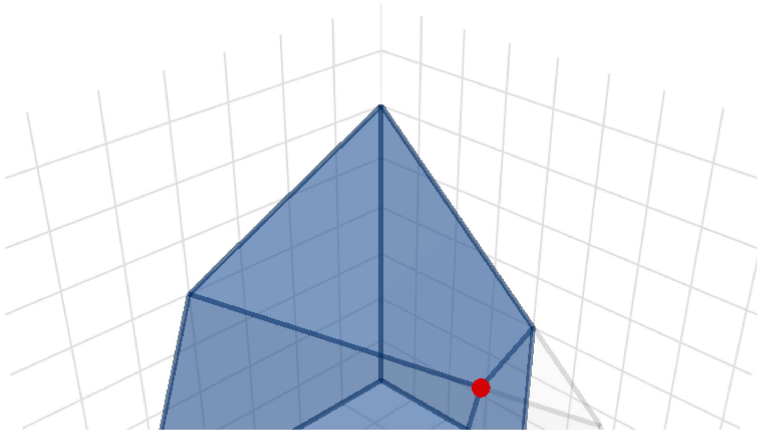


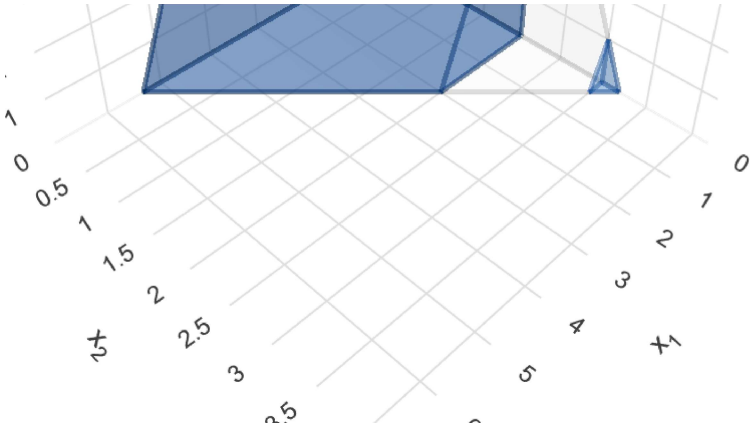
Geometric Interpretation of LPs

Constraint(s)

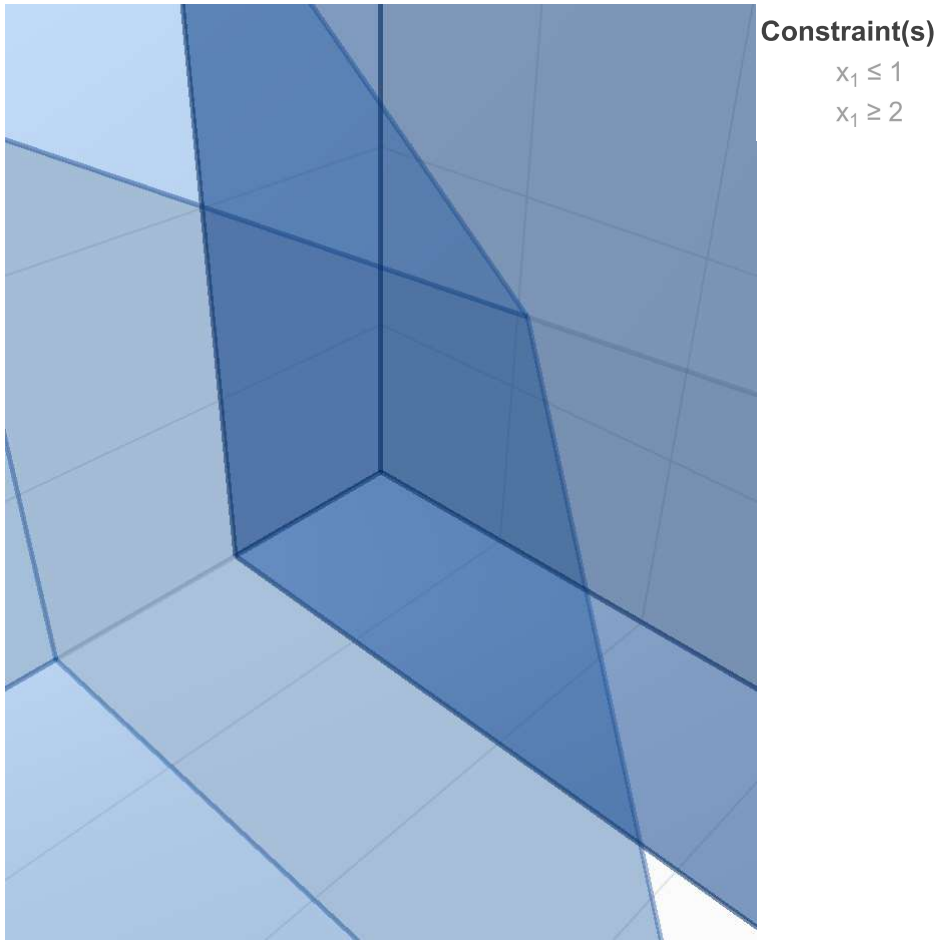
$x_2 \leq 2$

$x_2 \geq 3$





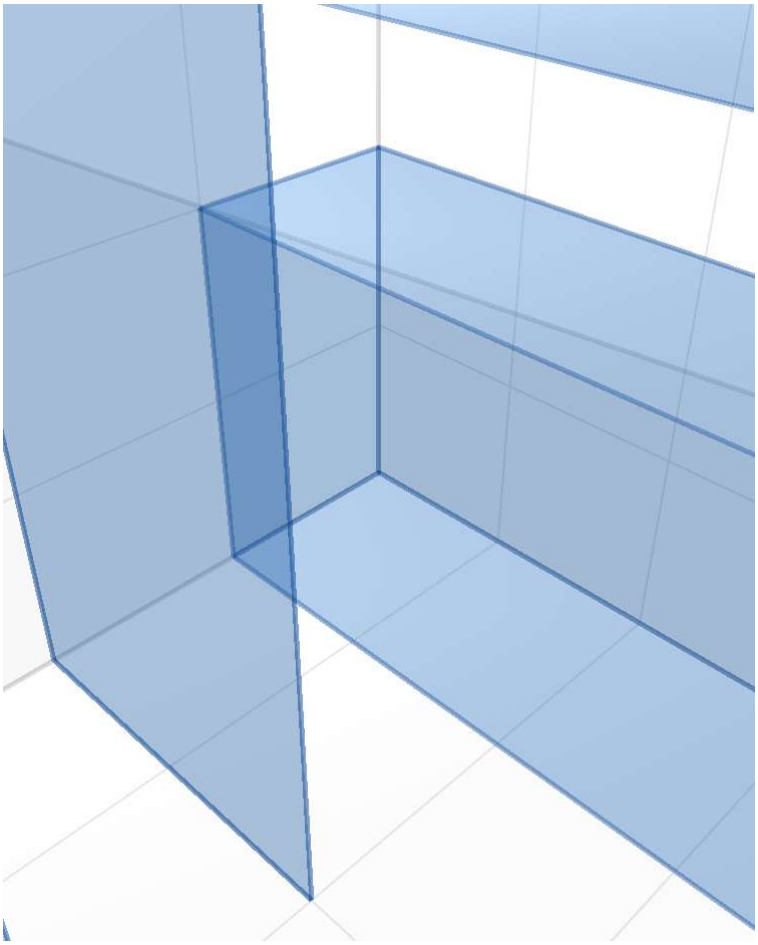
Geometric Interpretation of LPs



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

In [33]:
```
nodes[-1].show()
```

Geometric Interpretation of LPs



Q43: Consider the feasible region that looks like a rectangular box with one corner point a origin. What node does it correspond to in the tree? What is the optimal solution at that no

A: That region corresponds to the node on the left of the tree with constraints x2 <= 2, x1 and x3 <= 2. It has optimal solution (1, 2, 2) and has a value of 12.

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

A: We ended up exploring 12 nodes. If we knew the optimal solution was 13, we could hav explored one node (the original solution constrained to x2 >= 3).

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:

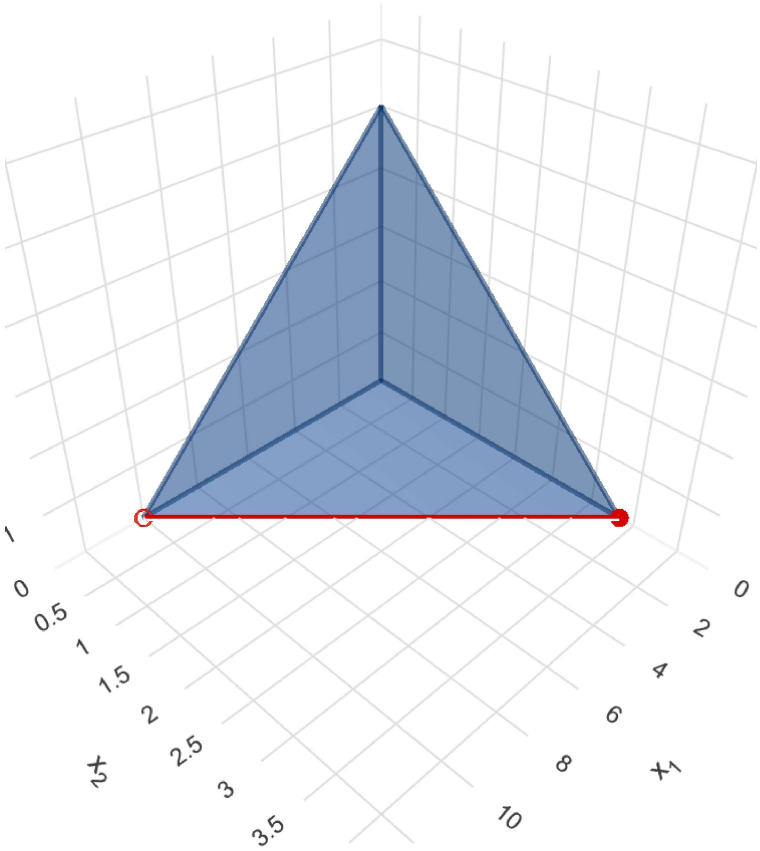
$$\begin{aligned} \max \quad & 2x_1 + 9x_2 + 6x_3 \\ \text{s.t.} \quad & 1x_1 + 3x_2 + 2x_3 \leq 10 \\ & x_1, x_2, x_3 \geq 0, \quad \text{integer} \end{aligned}$$

In gilp, we can define this lp as follows:

In [35]:
```
lp = gilp.LP([[1,3,2]],
             [10],
             [2,9,6])


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

Geometric Interpretation of LPs

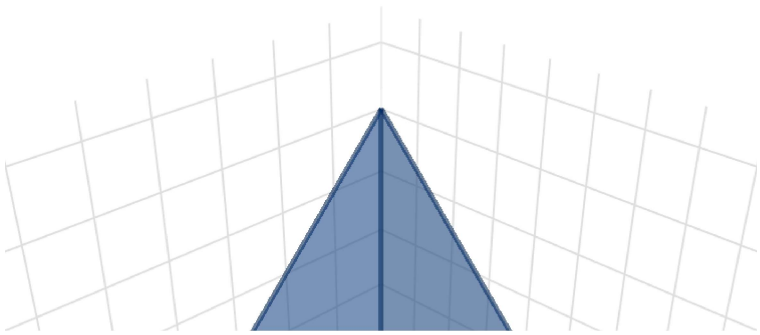


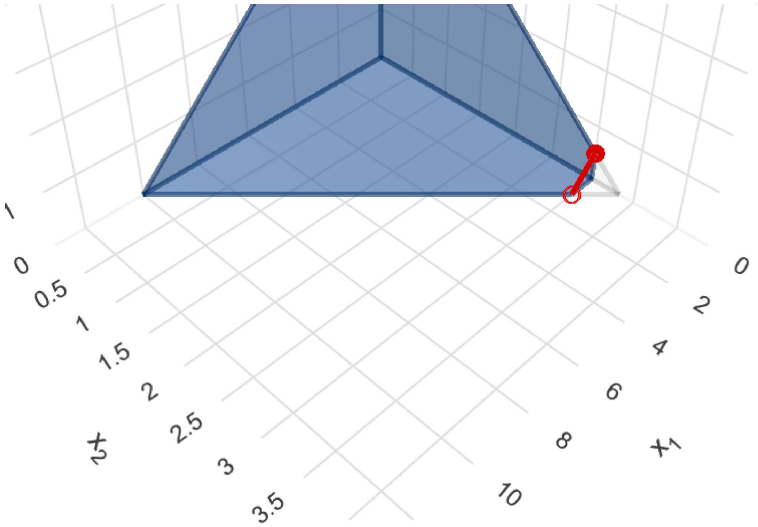
Geometric Interpretation of LPs

Constraint(s)

$x_2 \leq 3$

$x_2 \geq 4$



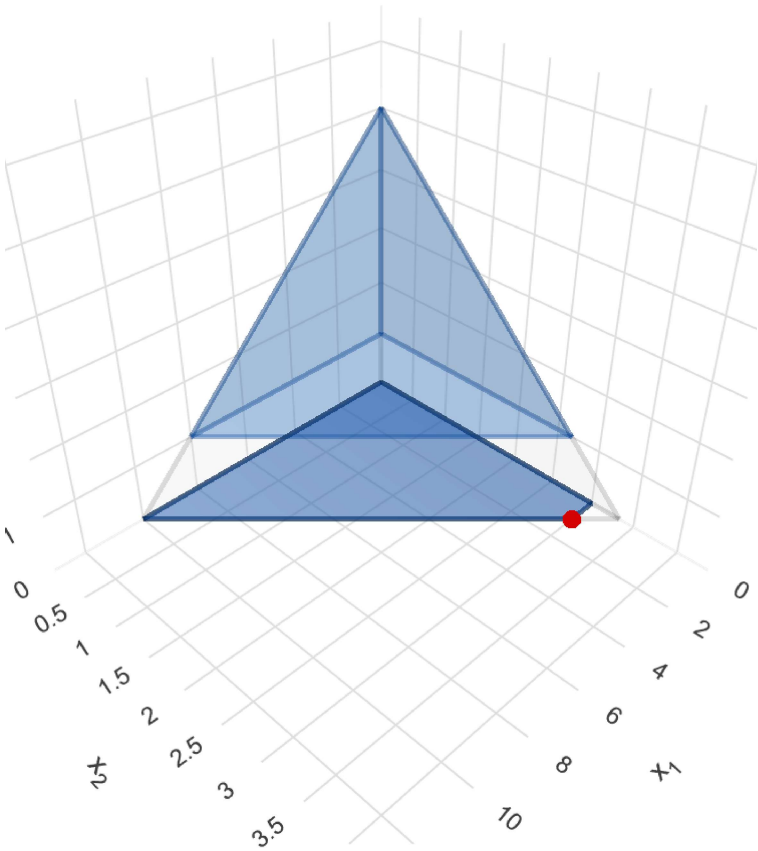


Geometric Interpretation of LPs

Constraint(s)

$x_3 \leq 0$

$x_3 \geq 1$



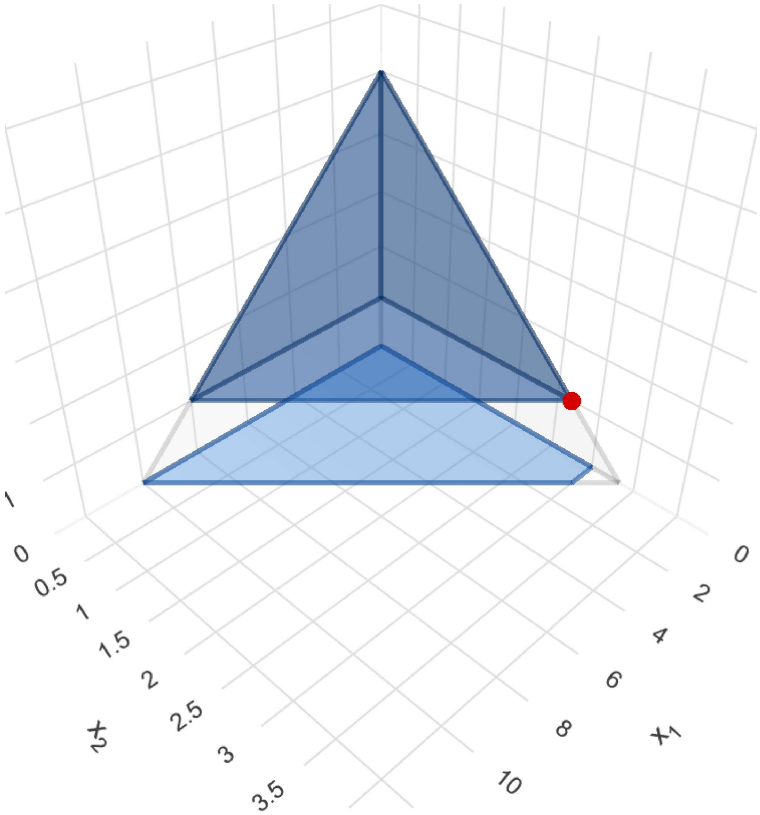
$x_3 \leq 0$
29
x* = (1, 3,

Geometric Interpretation of LPs

Constraint(s)

$x_3 \leq 0$

$x_3 \geq 1$



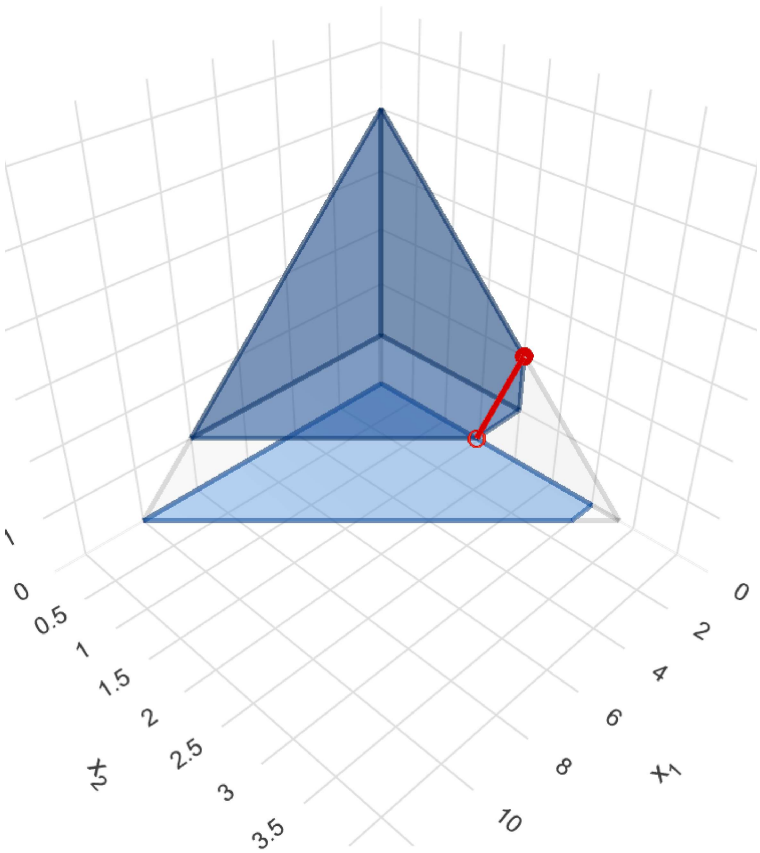
$x_3 \leq 0$
29
x* = (1, 3,

Geometric Interpretation of LPs

Constraint(s)

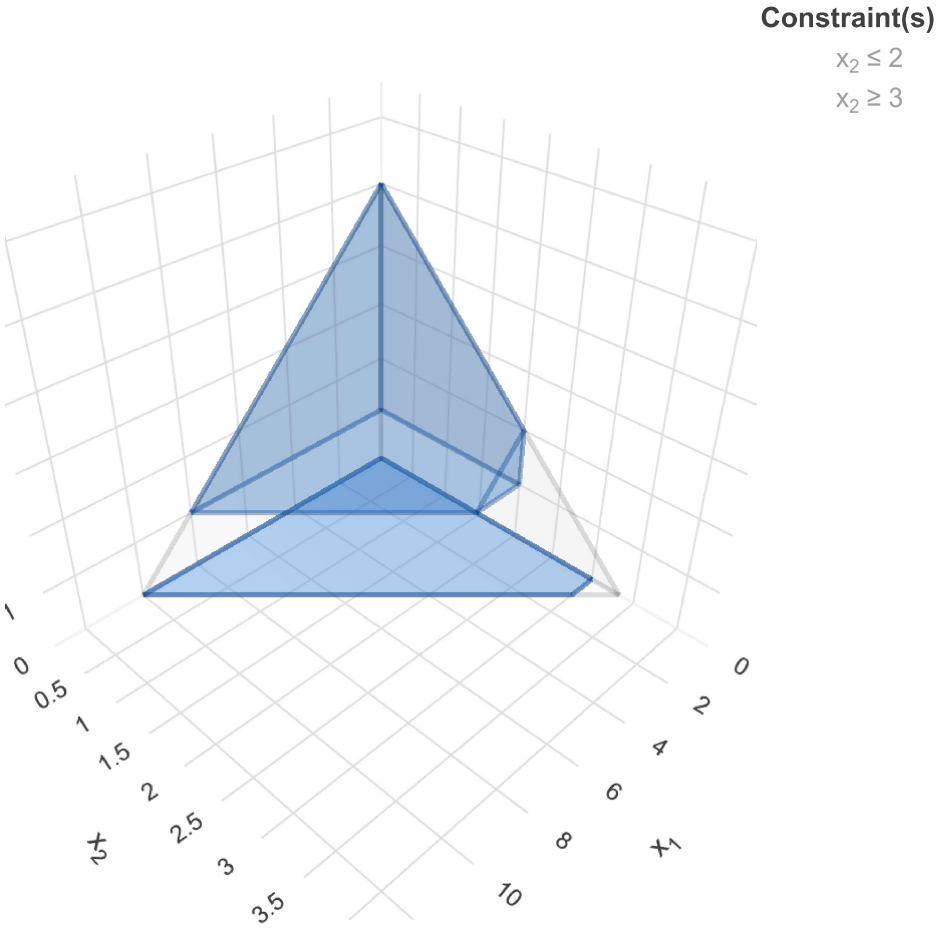
$x_2 \leq 2$

$x_2 \geq 3$

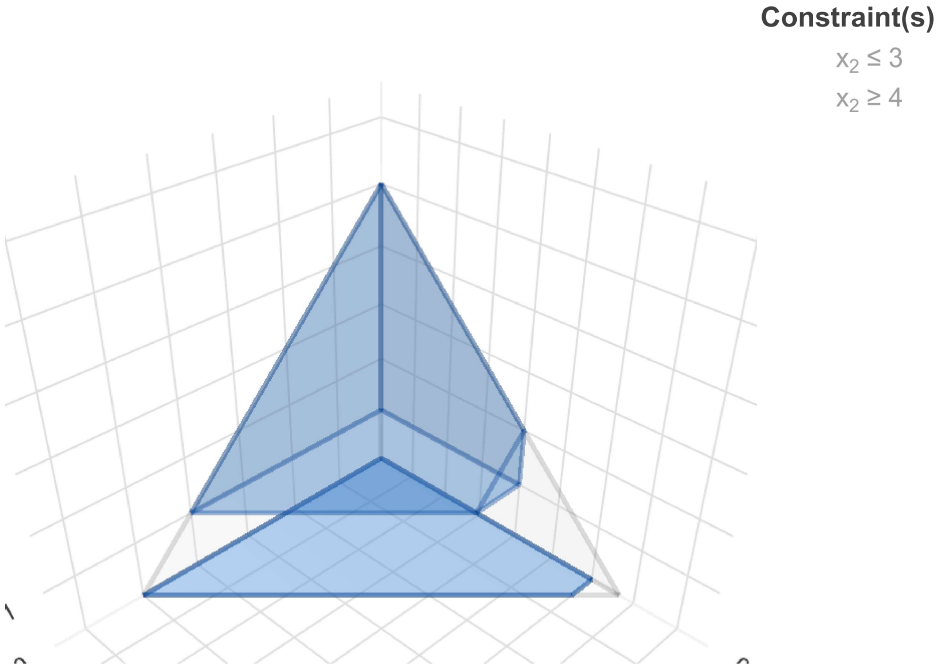


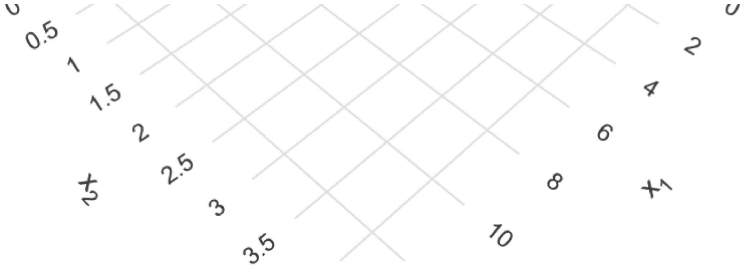
$x_3 \leq 0$
29
x* = (1, 3,

Geometric Interpretation of LPs



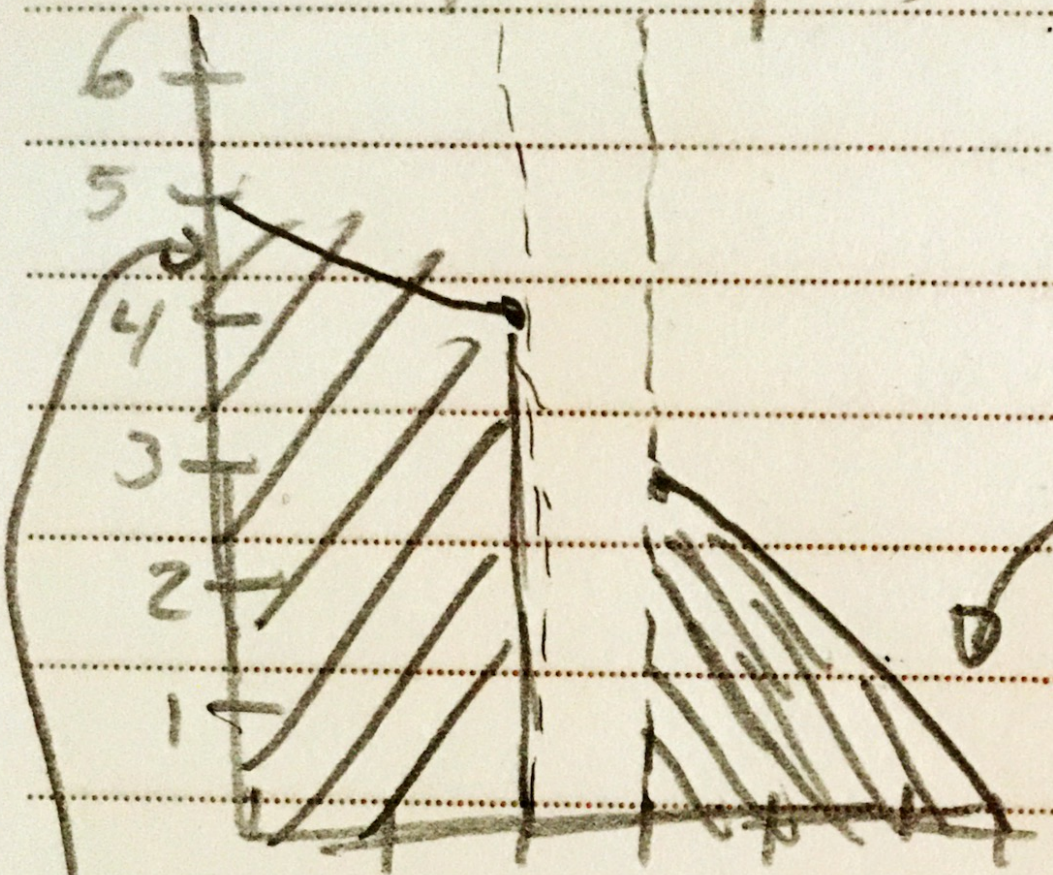
Geometric Interpretation of LPs





In [ ]:

$x_1 \geq 2$ $x_1 \geq 3$



$x_1 + x_2 \leq 6$

$5x_1 + 9x_2 \leq 45$