# Linear Integer Programing

Alireza Sheikh-Zadeh, Ph.D.

*Integer programs* are almost identical to linear programs with one very important exception. Some of the decision variables in integer programs may need only integer values. Since most integer programs contain a mix of continuous variables and integer variables, they are often known as *mixed-integer programs*.

The naive way to solve an integer program is to simply assume decision variables are continuous, solve the corresponding LP (called the LP relaxation of integer program), and then round the entries of the solution to the LP relaxation. However, this solution may not be optimal and may not even be feasible; it may violate some constraints.

**Example:** The following integer linear programming (ILP or IP) problem has decision variables *x* and *y*:

*Maximize* $Z = 6x + 5y$

*s.t.*

$$2x + 3y \leq 7$$

$$2x - y \leq 2$$

$$x, y \geq 0$$

$$x, y \in integer$$

The linear constraints and bounds of the ILP determine a feasible region in two dimensions: the feasible region of the linear programming (LP) relaxation.

What is the graphical presentation?

After finding the LP solution, if the values of optimal decisions are not all integer, we use the branch-and-bound process to find the best integer solution.

## How Branch-and-Bound Works

Here are the steps to solve ILP problems:

- If the LP solution is integer, no branching is required.
- The branching variable can be any integer variable with a non-integer (fractional) value in the optimal solution to the LP relaxation.
- An infeasible subproblem can be pruned.
- A subproblem whose bound is no better than the Best-Integer solution can be pruned.
- The ILP is solved as soon as all active subproblem branches have been solved.

The Python Answer:

```
from pulp import *

# A new LP problem
prob = LpProblem("1st ILP", LpMaximize)

x = LpVariable("x1", 0, cat='Integer')

y = LpVariable("x2", 0, cat='Integer')

prob += 6*x + 5*y, "obj"

prob += 2*x - y <= 2, "c1"
prob += 2*x + 3*y <= 7,"c2"

print(prob)

prob.solve()

print("Status:", LpStatus[prob.status])

for v in prob.variables():
    print(v.name, "=", v.varValue)

print("objective=", value(prob.objective))
```

```
1st ILP:
MAXIMIZE
6*x1 + 5*x2 + 0
SUBJECT TO
c1: 2 x1 - x2 <= 2

c2: 2 x1 + 3 x2 <= 7

VARIABLES
0 <= x1 Integer
0 <= x2 Integer

Status: Optimal
x1 = 1.0
x2 = 1.0
objective= 11.0
```

**Example:** Workforce-scheduling model

Workforce scheduling is a practical yet often highly complex problem many businesses face. In the problem below, Brewer Services aims to schedule customer service workers to minimize the number of part-time workers. Here are the key information about this business:

- Brewer Services schedules customer service workers on Monday to Friday, 8 a.m. to 5 p.m.
- Staffing requirements are shown in the table.

| HOUR | MINIMUM STAFF REQUIRED |
|---|---|
| 8-9 | 5 |
| 9-10 | 12 |
| 10-11 | 15 |
| 11-NOON | 12 |
| NOON-1 | 11 |
| 1-2 | 18 |
| 2-3 | 17 |
| 3-4 | 19 |
| 4-5 | 14 |

- Five permanent employees work all day.
- In a day, there are six 4-hour shifts.

What is the minimum number of part-time employees needed to meet staffing requirements?

Model development:

$x_i$ = integer number of part-time workers that start on the $i$-th 4-hour shift ($i$ = 1 starts at 8 a.m., …, $i$ = 6 starts at 1 p.m.)

Constraints: For each hour, we need to ensure that the total number of part-time employees who work that hour is at least as large as the minimum requirements.

```
# Import PuLP modeler functions
from pulp import *

# Creates a dictionary for the hourly staff demand
staffDemand = {"8-9": 5,
            "9-10": 12,
            "10-11": 15,
            "11-noon": 12,
            "noon-1": 11,
            "1-2": 18,
            "2-3": 17,
            "3-4": 19,
            "4-5": 14}

permanentEmployee = 5

# Create the model object
```

```python
prob = LpProblem("Scheduling Problem",LpMinimize)

x1 = LpVariable("shift1", 0, cat='Integer')
x2 = LpVariable("shift2", 0, cat='Integer')
x3 = LpVariable("shift3", 0, cat='Integer')
x4 = LpVariable("shift4", 0, cat='Integer')
x5 = LpVariable("shift5", 0, cat='Integer')
x6 = LpVariable("shift6", 0, cat='Integer')


# The objective function is added to prob first
prob += x1+x2+x3+x4+x5+x6, "Sum of part time staffs"


# The capacity constraints are added to prob for each hour staff demand

prob += x1 == staffDemand["8-9"] - permanentEmployee , "Staff demand 8-9 am"
prob += x1 + x2 >= staffDemand["9-10"] - permanentEmployee, "Staff demand 9-10 am"
prob += x1 + x2 + x3 >= staffDemand["10-11"] - permanentEmployee, "Staff demand 10-11 am"
prob += x1 + x2 + x3 + x4 >= staffDemand["11-noon"] - permanentEmployee, "Staff demand 11-noon"
prob += x2 + x3 + x4 + x5 >= staffDemand["noon-1"] - permanentEmployee, "Staff demand noon-1"
prob += x3 + x4 + x5 + x6 >= staffDemand["1-2"] - permanentEmployee, "Staff demand 1-2"
prob += x4 + x5 + x6 >= staffDemand["2-3"] - permanentEmployee, "Staff demand 2-3"
prob += x5 + x6 >= staffDemand["3-4"] - permanentEmployee, "Staff demand 3-4"
prob += x6 >= staffDemand["4-5"] - permanentEmployee, "Staff demand 4-5"

print(prob)


prob.solve()


print("Status:", LpStatus[prob.status])


for v in prob.variables():
    print(v.name, "=", v.varValue)


print("objective=", value(prob.objective))
```

```
Scheduling Problem:
MINIMIZE
1*shift1 + 1*shift2 + 1*shift3 + 1*shift4 + 1*shift5 + 1*shift6 + 0
SUBJECT TO
Staff_demand_8_9_am: shift1 = 0

Staff_demand_9_10_am: shift1 + shift2 >= 7

Staff_demand_10_11_am: shift1 + shift2 + shift3 >= 10

Staff_demand_11_noon: shift1 + shift2 + shift3 + shift4 >= 7

Staff_demand_noon_1: shift2 + shift3 + shift4 + shift5 >= 6

Staff_demand_1_2: shift3 + shift4 + shift5 + shift6 >= 13

Staff_demand_2_3: shift4 + shift5 + shift6 >= 12

Staff_demand_3_4: shift5 + shift6 >= 14

Staff_demand_4_5: shift6 >= 9

VARIABLES
0 <= shift1 Integer
0 <= shift2 Integer
0 <= shift3 Integer
0 <= shift4 Integer
0 <= shift5 Integer
0 <= shift6 Integer

Status: Optimal
shift1 = 0.0
shift2 = 10.0
shift3 = 0.0
shift4 = 0.0
shift5 = 5.0
shift6 = 9.0
objective= 24.0
```