Integer Programming

Integer programs are almost identical to linear programs with one very important exception. Some of the decision variables in integer programs may need to have only integer values. The variables are known as integer variables. 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 ILP is to simply remove the constraint that x is an integer, solve the corresponding LP (called the LP relaxation of the ILP), and then round the entries of the solution to the LP relaxation. But, not only may this solution not be optimal, it may not even be feasible; that is, it may violate some constraint.

Example: The following integer linear programming (ILP) problem has integer decision variables x_1, x_2.

Maximize

$3x_1 + 5x_2$

s.t.

$2x_1 + 4x_2 \leq 25$

$x_1 \leq 8$

$2x_2 \leq 10$

$x_1, x_2 \geq 0$

$x_1, x_2 \in$ 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?

Integer programs can be very difficult problems to solve and there is a lot of current research finding "good" ways to solve integer programs. Integer programs can be solved using the branch-and-bound process.

How Branch-and-Bound Works

- If the LP relaxation of a subproblem has an optimal solution that is feasible to the ILP, no branching is required.
- The branching variable can be an integer variable that has a fractional value in the optimal solution to the LP relaxation.
- An infeasible subproblem can be pruned.
- A subproblem whose bound is no better than BestInteger (the objective value of a known feasible solution to the MILP) can be pruned (or fathomed).
- A subproblem whose bound is no better than BestInteger (the objective value of a known feasible solution to the MILP) can be pruned (or fathomed).
- The ILP is solved as soon as all active subproblems (or active nodes of the branch-and-bound tree) have been solved.

The Python Answer:

```
from pulp import *

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

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

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

prob += 3*x1 + 5*x2, "obj"

prob += 2*x1 + 4*x2 <= 25
prob += x1 <= 8
prob += 2*x2 <= 10

print(prob)

## 1st ILP:
## MAXIMIZE
## 3*x1 + 5*x2 + 0
## SUBJECT TO
## _C1: 2 x1 + 4 x2 <= 25
##
## _C2: x1 <= 8
##
## _C3: 2 x2 <= 10
##
## VARIABLES
## 0 <= x1 Integer
## 0 <= x2 Integer

prob.solve()

## 1

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

## Status: Optimal

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

## x1 = 8.0
## x2 = 2.0

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

## objective= 34.0
```

Example: Workforce-scheduling model

Workforce scheduling is a practical, yet a highly complex, problem that many businesses face.

- Brewer Services schedules customer service workers from 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

- 5 permanent employees work all day.
- Part-time employees work 4-hour shifts.

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

Model development:

Xi = integer number of part-time workers that start on the ith 4-hour shift (i = 1 for 8 a.m., ., i = 6 for 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 modeller 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

# Creates the prob variable to contain the problem data
prob = LpProblem("Transportation 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 == 0
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)
```

```
## Transportation Problem:
## MINIMIZE
## 1*shift1 + 1*shift2 + 1*shift3 + 1*shift4 + 1*shift5 + 1*shift6 + 0
## SUBJECT TO
## _C1: 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

prob.solve()

## 1

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

## Status: Optimal

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

## objective= 24.0

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

## shift1 = 0.0
## shift2 = 10.0
## shift3 = 0.0
## shift4 = 0.0
## shift5 = 5.0
## shift6 = 9.0
```

Practice: Project selection model (Binary Integer Programming (BIP)

- Five potential projects are being considered. Each project is expected to generate a return (given by the net present value) but requires a fixed amount of cash and personnel.

	Project 1	Project 2	Project 3	Project 4	Project 5	Available Resources
Expected return (NPV)	$180,000	$220,000	$150,000	$140,000	$200,000	
Cash requirements	$55,000	$83,000	$24,000	$49,000	$61,000	$150,000
Personnel requirements	5	3	2	5	3	12

- The available budget and human resources do not allow selection of all 5 projects.
- The objective is to maximize expected return.