Assignment Problem:

Suppose we just started the e-commerce business and company started to grow. We need to urgently fill 8 different positions in 8 different departments. We have a total of 8 positions that needs to be filled by 8 new people. We have 8 different candidates and their overall experience in related field.

Candidate Name	Engineering	Human Resources	Data Science	Finance	Operations	Trans	Sales	Marketing
Sachin Khan	2	2	3	4	2	2	0	1
Louis Edwin	0	0	0	0	1	4	2	2
Teddy Bear	1	0	4	1	2	5	3	1
Connor Joker	1	1	1	1	1	5	5	0
taby taboo	0	1	0	7	2	4	6	0
Amanda Panda	1	2	3	4	5	3	7	3
Bryanna Bre	5	4	3	2	1	2	6	4
Allsion K	4	2	4	2	1	3	1	3

Based on their experience, assign each new candidate to new openings to give greatest total years of experience for all jobs.

Solution:

Sets and Indices:

$i \in I$ Index on each candidate

$j \in J$ Index on each department

$(i,j) \in A$ Set containing candidate and department

Data:

$Experience_{ij}$ experience of each candidate on each department

Decision Variables:

X_{ij} Indicates if i assigned to j

Constraints:

$\sum_j X_{ij} = 1, \forall i \in I$ states that each candidate must be assigned to only one open job

$\sum_i Xij = 1, \forall j \in J$ states that each job must be assigned to a candidate only once

$X_{ij} \geq 0$, for all i and j

$X_{ij} \in Binary$

Objective:

$\max \sum_{(i,j) \in A} Experience_{ij} X_{ij}$ states to maximize the number of years of experience for each candidate and job

In [1]:
```
#Importing required libraries
from pyomo.environ import *
from pyomo.opt import *
from pyomo.core import *
import cplex
```

In [7]:
```
def objective_rule(model):
    # Create objective function
    return sum(model.experience[i,j] * model.SELECT[i,j] for (i,j) in model.arcs)

def person_rule(model,k):
    # Constraint requiring each person to have only one job
    return sum(model.SELECT[i,j] for (i,j) in model.arcs if i ==k) == 1

def job_rule(model,k):
    # Constraint requiring each job to be assigned exactly once
    return sum(model.SELECT[i,j] for (i,j) in model.arcs if j ==k) == 1

def SolveUsingPyomo():
    ''' Create and solve a concrete Pyomo model '''

    # Initialize data structures with hard-coded data from example
    numPeople = 8 # Number of candidates
    numJobs = 8 # Number of jobs
    numNodes = numPeople + numJobs
    experience = {
              (1,9):2, (1,10):2, (1,11):3, (1,12):4, (1,13):2, (1,14):2, (1,15):0, (1,16):1,
              (2,9):0, (2,10):0, (2,11):0, (2,12):0, (2,13):1, (2,14):4, (2,15):2, (2,16):2,
              (3,9):1, (3,10):0, (3,11):4, (3,12):1, (3,13):2, (3,14):5, (3,15):3, (3,16):1,
              (4,9):1, (4,10):1, (4,11):1, (4,12):1, (4,13):1, (4,14):5, (4,15):5, (4,16):0,
              (5,9):5, (5,10):1, (5,11):0, (5,12):7, (5,13):2, (5,14):5, (5,15):6, (5,16):0,
              (6,9):1, (6,10):2, (6,11):3, (6,12):4, (6,13):5, (6,14):3, (6,15):7, (6,16):3,
              (7,9):5, (7,10):4, (7,11):3, (7,12):2, (7,13):1, (7,14):2, (7,15):6, (7,16):4,
              (8,9):4, (8,10):2, (8,11):4, (8,12):2, (8,13):1, (8,14):3, (8,15):1, (8,16):3
                  }

    arcs = [
              (1,9), (1,10), (1,11), (1,12), (1,13), (1,14), (1,15),(1,16),
              (2,9), (2,10), (2,11), (2,12), (2,13), (2,14), (2,15),(2,16),
              (3,9), (3,10), (3,11), (3,12), (3,13), (3,14), (3,15),(3,16),
              (4,9), (4,10), (4,11), (4,12), (4,13), (4,14), (4,15),(4,16),
              (5,9), (5,10), (5,11), (5,12), (5,13), (5,14), (5,15),(5,16),
              (6,9), (6,10), (6,11), (6,12), (6,13),  (6,14), (6,15),(6,16),
              (7,9), (7,10), (7,11), (7,12), (7,13), (7,14), (7,15),(7,16),
              (8,9), (8,10), (8,11), (8,12), (8,13), (8,14), (8,15),(8,16)
             ]

    # Create a concrete Pyomo model
    print("Building Pyomo model...")
    model = ConcreteModel()

    # Define indices and sets
    print("Creating indices and sets...")
    model.i = Set(initialize=[i for i in range(1, numNodes+1)], ordered=True) # Index on each node
    model.peopleNodes = Set(within=model.i, initialize = [*range(1,numPeople+1)]) # Index on each perso
n node
    model.jobNodes = Set(within=model.i, initialize = [*range(numPeople+1,numNodes+1)]) # Index on each
 job node
    model.arcs = Set(within=model.i * model.i, initialize = arcs) # Create the set of arcs

    # Define variables
    print("Creating variables...")
    model.SELECT = Var(model.arcs, domain=Binary, initialize = 0) # Flow variable

    # Create parameters (i.e., data)
    print("Creating parameters...")
    model.experience = Param(model.arcs, initialize = experience)

     # Create objective function
    print("Creating objective function...")
    model.objective = Objective(rule=objective_rule, sense = maximize)

    print("Creating assignment constraints ...")
    model.personConstraint = Constraint(model.peopleNodes, rule=person_rule)
    model.jobConstraint = Constraint(model.jobNodes, rule=job_rule)

    print("Done.")

    print("Running solver...")
    opt = SolverFactory("cplex") #, solver_io="direct") #("_cplex_direct") #, solver_io="python") # Opt
ions: cplex, gurobi, gipk  SolverFactory("gurobi", solver_io='python')
    # model.dual = Suffix(direction=Suffix.IMPORT) # Import dual values from model
    # model.rc = Suffix(direction=Suffix.IMPORT) # Import reduced costs from model
    model.write('testLPfile.lp', io_options={'symbolic_solver_labels':True} ) # Use this if you want to
 write the .lp file
    results = opt.solve(model, tee=True) # This runs the solver
    print("Done.")

    # Write full optimizer results
    # results.write()

    # Print results (this is hard-coded to be specific to this problem)
    print("The objective value is: " + str(model.objective.expr()))
    for i,j in model.arcs:
        if(model.SELECT[i,j].value == 1): # If there is flow (an assignment) on this arc
            print("Person " + str(i) + " is assigned to job " + str(j) + ", with " +str(model.experienc
e[i,j]) + " years of experience.")

SolveUsingPyomo() # Run above code
```

```
Building Pyomo model...
Creating indices and sets...
Creating variables...
Creating parameters...
Creating objective function...
Creating assignment constraints ...
Done.
Running solver...

Welcome to IBM(R) ILOG(R) CPLEX(R) Interactive Optimizer 20.1.0.0
  with Simplex, Mixed Integer & Barrier Optimizers
5725-A06 5725-A29 5724-Y48 5724-Y49 5724-Y54 5724-Y55 5655-Y21
Copyright IBM Corp. 1988, 2020.  All Rights Reserved.

Type 'help' for a list of available commands.
Type 'help' followed by a command name for more
information on commands.

CPLEX> Logfile 'cplex.log' closed.
Logfile 'C:\Users\kkhar\AppData\Local\Temp\tmpjvbyfize.cplex.log' open.
CPLEX> Problem 'C:\Users\kkhar\AppData\Local\Temp\tmpw145iqgy.pyomo.lp' read.
Read time = 0.00 sec. (0.00 ticks)
CPLEX> Problem name         : C:\Users\kkhar\AppData\Local\Temp\tmpw145iqgy.pyomo.lp
Objective sense      : Maximize
Variables            :      65  [Nneg: 1,  Binary: 64]
Objective nonzeros   :      54
Linear constraints   :      17  [Equal: 17]
  Nonzeros           :     129
  RHS nonzeros       :      17

Variables            : Min LB: 0.000000         Max UB: 1.000000
Objective nonzeros   : Min   : 1.000000         Max   : 7.000000
Linear constraints   :
  Nonzeros           : Min   : 1.000000         Max   : 1.000000
  RHS nonzeros       : Min   : 1.000000         Max   : 1.000000
CPLEX> Version identifier: 20.1.0.0 | 2020-11-10 | 9bedb6d68
Found incumbent of value 27.000000 after 0.00 sec. (0.01 ticks)
Tried aggregator 1 time.
MIP Presolve eliminated 1 rows and 1 columns.
Reduced MIP has 16 rows, 64 columns, and 128 nonzeros.
Reduced MIP has 64 binaries, 0 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (0.07 ticks)
Probing time = 0.00 sec. (0.08 ticks)
Tried aggregator 1 time.
Detecting symmetries...
Reduced MIP has 16 rows, 64 columns, and 128 nonzeros.
Reduced MIP has 64 binaries, 0 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (0.08 ticks)
Probing time = 0.00 sec. (0.08 ticks)
Clique table members: 16.
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: deterministic, using up to 8 threads.
Root relaxation solution time = 0.00 sec. (0.04 ticks)

        Nodes                                         Cuts/
   Node  Left     Objective  IInf  Best Integer    Best Bound    ItCnt     Gap

*     0+    0                           27.0000      152.0000           462.96%
*     0+    0                           28.0000      152.0000           442.86%
*     0     0      integral     0       35.0000       35.0000        5    0.00%
Elapsed time = 0.00 sec. (0.44 ticks, tree = 0.00 MB, solutions = 3)

Root node processing (before b&c):
  Real time             =    0.00 sec. (0.45 ticks)
Parallel b&c, 8 threads:
  Real time             =    0.00 sec. (0.00 ticks)
  Sync time (average)   =    0.00 sec.
  Wait time (average)   =    0.00 sec.
                          ------------
Total (root+branch&cut) =    0.00 sec. (0.45 ticks)

Solution pool: 3 solutions saved.

MIP - Integer optimal solution:  Objective =  3.5000000000e+01
Solution time =    0.00 sec.  Iterations = 5  Nodes = 0
Deterministic time = 0.45 ticks  (447.25 ticks/sec)

CPLEX> Incumbent solution written to file 'C:\Users\kkhar\AppData\Local\Temp\tmpqheyw8r6.cplex.sol'.
CPLEX> Done.
The objective value is: 35.0
Person 1 is assigned to job 10, with 2 years of experience.
Person 2 is assigned to job 14, with 4 years of experience.
Person 3 is assigned to job 11, with 4 years of experience.
Person 4 is assigned to job 15, with 5 years of experience.
Person 5 is assigned to job 12, with 7 years of experience.
Person 6 is assigned to job 13, with 5 years of experience.
Person 7 is assigned to job 16, with 4 years of experience.
Person 8 is assigned to job 9, with 4 years of experience.
```