Part 0:

```
Optimize a model with 248 rows, 60 columns and 772 nonzeros
Model fingerprint: 0xd6346155
Variable types: 0 continuous, 60 integer (52 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+07]
  Objective range  [1e+00, 4e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 2e+07]
Found heuristic solution: objective 10.0000000
Presolve removed 86 rows and 21 columns
Presolve time: 0.00s
Presolved: 162 rows, 39 columns, 503 nonzeros
Variable types: 0 continuous, 39 integer (34 binary)

Root relaxation: objective 1.000000e+00, 33 iterations, 0.00 seconds (0.00 work units)

      sh_0_2           1
      ah_0_2           1
      sh_0_3           1
      ah_0_3           1
      ah_1_2           1
      ah_1_3           1
      sh_2_0           1
      ah_2_1           1
      sh_2_3           1
      ah_2_3           1

Root relaxation: objective 2.333333e+02, 2 iterations, 0.00 seconds (0.00 work units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0  233.33333    0    2  155.00000  233.33333  50.5%     -    0s
H    0     0                    230.0000000  233.33333  1.45%     -    0s
     0     0  233.33333    0    2  230.00000  233.33333  1.45%     -    0s

Explored 1 nodes (2 simplex iterations) in 0.01 seconds (0.00 work units)
Thread count was 16 (of 16 available processors)

Solution count 3: 230 155 150

Optimal solution found (tolerance 1.00e-04)
Best objective 2.300000000000e+02, best bound 2.300000000000e+02, gap 0.0000%
x1 6
x1 6
x2 7
x2 7
Obj: 230
PS C:\Users\laure\Desktop\Masters\Spring 2023\503\HW\3>
```

This part is fairly straightforward as I was just running the python file that was in the given homework assignment.

Part 1:

```
# linearized scheduling constraint 1
for i in range(num_tasks):
    for j in range(num_tasks):
        if i!=j:
            # complete this part
            m.addConstr(s_var[i] + t_var[i] <= s_var[j] + (1 - 
sh_var[i][j])*my_infty + (1 - ah_var[i][j])*my_infty, 
name="schedulingConstraint1"+str(i)+"_"+str(j))

# linearized scheduling constraint 2
for i in range(num_tasks):
    for j in range(num_tasks):
        if i!=j:
```

```
            # complete this part
            m.addConstr(s_var[j] + t_var[j] <= s_var[i] + (1 -
sh_var[i][j])*my_infty + ah_var[i][j]*my_infty,
name="schedulingConstraint2"+str(i)+"_"+str(j))
```

This part was fairly straightforward as I simply applied these two scheduling constraints to the pre-existing skeleton code as given in the hw document and the slides we went over in class that cover linear programming optimization.

- For all v_i , $v_j \in V$, s.t. $i \neq j$
  - $s_i + t_i \leq s_j + (1 - sh_{i,j})*\infty + (1 - ah_{i,j})*\infty$
  - $s_j + t_j \leq s_i + (1 - sh_{i,j})*\infty + ah_{i,j}*\infty$

The screenshots below show the optimization results for each latency value tested.

Latency:2

```
Explored 0 nodes (0 simplex iterations) in 0.00 seconds (0.00 work units)
Thread count was 1 (of 16 available processors)

Solution count 0

Model is infeasible
Best objective -, best bound -, gap -
=============================================
Optimization Results:

    Variable            X
-------------------------
Traceback (most recent call last):
  File "C:\Users\laure\Desktop\Masters\Spring 2023\503\HW\3\map_skel.py", line 155, in <module>
    m.printAttr('X')
  File "src\gurobipy\model.pxi", line 2524, in gurobipy.Model.printAttr
  File "src\gurobipy\model.pxi", line 1938, in gurobipy.Model.getAttr
  File "src\gurobipy\attrutil.pxi", line 148, in gurobipy.__gettypedattrlist
gurobipy.GurobiError: Unable to retrieve attribute 'X'
```

Latency: 3

```
Solution count 1: 10

Optimal solution found (tolerance 1.00e-04)
Best objective 1.000000000000e+01, best bound 1.000000000000e+01, gap 0.0000%
=============================================
Optimization Results:

    Variable            X
-------------------------
       m_0_0            1
       m_1_1            1
       m_2_2            1
       m_3_3            1
         s_1            1
         s_2            1
         s_3            2
         t_0            1
         t_1            1
         t_2            1
         t_3            1
      ah_0_1            1
      ah_0_3            1
      ah_1_3            1
      ah_2_3            1
      used_0            1
      used_1            1
      used_2            1
      used_3            1
```

Latency: 4

```
Solution count 2: 6 10

Optimal solution found (tolerance 1.00e-04)
Best objective 6.000000000000e+00, best bound 6.000000000000e+00, gap 0.0000%
=============================================
Optimization Results:

    Variable            X
-------------------------
       m_0_0            1
       m_1_1            1
       m_2_2            1
       m_3_0            1
         s_1            1
         s_2            1
         s_3            2
         t_0            1
         t_1            1
         t_2            1
         t_3            2
      ah_0_1            1
      ah_0_2            1
      sh_0_3            1
      ah_0_3            1
      ah_1_2            1
      ah_2_1            1
      sh_3_0            1
      used_0            1
      used_1            1
      used_2            1
=============================================
```

Latency: 5

```
Solution count 4: 3 7 8 10

Optimal solution found (tolerance 1.00e-04)
Best objective 3.000000000000e+00, best bound 3.000000000000e+00, gap 0.0000%
=============================================
Optimization Results:

    Variable           X
-------------------------
       m_0_0           1
       m_1_1           1
       m_2_0           1
       m_3_0           1
         s_1           1
         s_2           1
         s_3           3
         t_0           1
         t_1           1
         t_2           2
         t_3           2
      ah_0_1           1
      sh_0_2           1
      ah_0_2           1
      sh_0_3           1
      ah_0_3           1
      ah_1_2           1
      ah_1_3           1
      sh_2_0           1
      ah_2_1           1
      sh_2_3           1
      ah_2_3           1
      sh_3_0           1
      sh_3_2           1
      used_0           1
      used_1           1
=============================================
```

Part 2:

New intialized values:

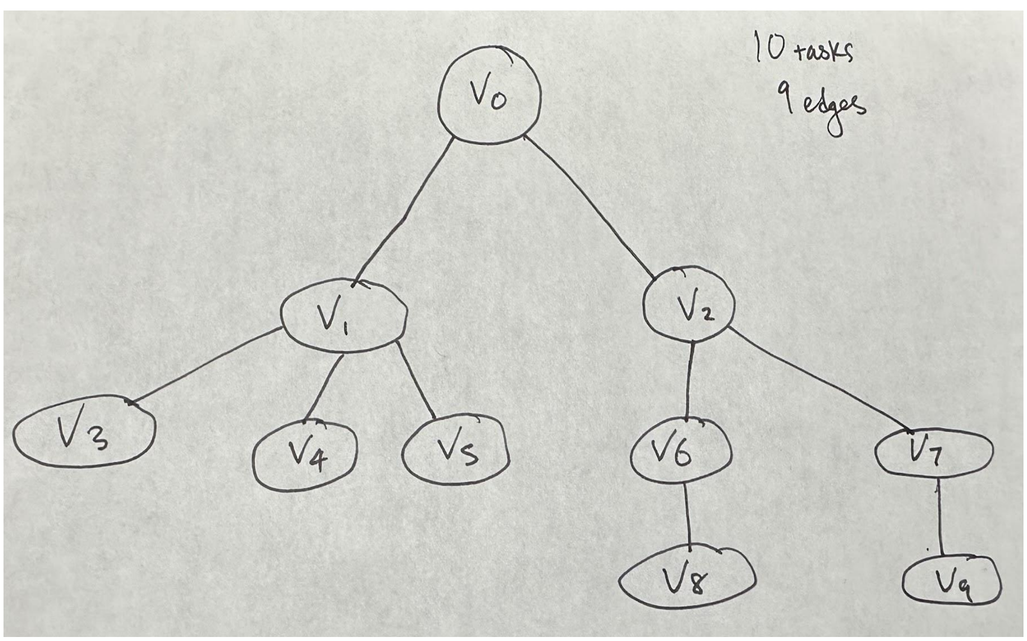
```
# num tasks: 10
num_tasks = 10
# num procs: 4
num_procs = 4
# latency constraint
latency_const = 18

# exec. info
exec_time = [
[1,2,3,4],
[2,1,3,4],
[2,3,1,4],
```

```
[2,3,4,1],
[1,2,3,4],
[2,1,3,4],
[2,3,1,4],
[2,3,4,1],
[1,2,3,4],
[2,1,3,4]
]

# edge info
edge = [
[0,1,1,0,0,0,0,0,1,0], #v0->v1,v2
[0,0,0,1,1,1,0,0,0,0], #v1->v3,v4,v5
[0,0,0,0,0,0,1,1,0,0], #v2->v6,v7
[0,0,0,0,0,0,0,0,0,0], #v3->
[0,0,0,0,0,0,0,0,0,0], #v4->
[0,0,0,0,0,0,0,0,0,0], #v5->
[0,0,0,0,0,0,0,0,1,0], #v6->v8
[0,0,0,0,0,0,0,0,0,1], #v7->v9
[0,0,0,0,0,0,0,0,0,0], #v8->
[0,0,0,0,0,0,0,0,0,0]  #v9->
]
```

Picture visualizing the new graph with 10 nodes and 9 edges



M. optmizations

```
# objective
lat = m.addVar(vtype=GRB.INTEGER, name="lat")

# update latency constraints and redefine them as completion time
constraints
for i in range(num_tasks):
    m.addConstr(s_var[i] + t_var[i] <= lat,
name="completion_time_const_"+str(i))

# objective - minimize latency
m.setObjective(lat, GRB.MINIMIZE)

m.write("formulation.lp")
m.optimize()
```

The main portion of this code that I changed areas shown above. In part 2, we are trying to minimize the latency with respect to the cost constraint. The key points that i show in the code above is that we initialized 10 tasks with 9 edges. And set the exec_time and cost tables that reflect this. This is initialized in such a way that many multiple feasible solutions exist.

The screenshot below shows the run of the code with the input values as num_tasks = 10, num_procs = 4, and latency_const = 18.

```
Explored 1 nodes (234 simplex iterations) in 0.03 seconds (0.04 work units)
Thread count was 16 (of 16 available processors)

Solution count 2: 4 18

Optimal solution found (tolerance 1.00e-04)
Best objective 4.000000000000e+00, best bound 4.000000000000e+00, gap 0.0000%
=============================================
Optimization Results:

    Variable           X
      ah_0_4           1
      sh_0_8           1
      ah_0_8           1
      ah_1_3           1
      ah_1_4           1
      sh_1_5           1
      ah_1_5           1
      sh_1_9           1
      ah_1_9           1
      sh_2_6           1
      ah_2_6           1
      ah_2_7           1
      sh_3_7           1
      sh_4_0           1
      sh_4_8           1
      ah_4_8           1
      sh_5_1           1
      sh_5_9           1
      ah_5_9           1
      sh_6_2           1
      ah_6_8           1
      sh_7_3           1
      ah_7_3           1
      ah_7_9           1
      sh_8_0           1
      sh_8_4           1
      sh_9_1           1
      sh_9_5           1
      used_0           1
      used_1           1
      used_2           1
      used_3           1
         lat           4
=============================================
```

Part 3:

```
# Extending edge constraints using sh_var
for i in range(num_tasks):
    for j in range(num_tasks):
        if edge[i][j] == 1:
```

```
            m.addConstr(s_var[i] + t_var[i] + 1 <= s_var[j] + my_infty * (1 - sh_var[i][j]), name="edge_const_extended_"+str(i)+"_"+str(j))
```

As shown in the code above we are extending a communication delay extending from the basic formulation which we created in part 1. By assuming that each dependency results in communication delay of 1 time unit if the source and destination tasks are mapped do different processors. The communication delay which we created is conditional that checks for all $e_{i,j} \in E$, but applies only for existing k such that $m_{i,k} = m_{j,k} = 1$. We have a predefined constraint to find the sh value in shvars which is a list of binary variables. So if sh_{ij} is 1, then we define the edge constraint as $s_i + t_i \leq s_j$. If it is 0, then we set it as $s_i + t_i + 1 \leq s_j$.
The below is the optimization screenshot of a latency constraint of 5 based on the skeleton code.

```
Solution count 3: 3 6 10

Optimal solution found (tolerance 1.00e-04)
Best objective 3.000000000000e+00, best bound 3.000000000000e+00, gap 0.0000%
=============================================
Optimization Results:

    Variable           X
-------------------------
       m_0_0           1
       m_1_1           1
       m_2_0           1
       m_3_1           1
         s_2           2
         s_3           2
         t_0           1
         t_1           1
         t_2           2
         t_3           3
      ah_0_1           1
      sh_0_2           1
      ah_0_2           1
      ah_0_3           1
      ah_1_2           1
      sh_1_3           1
      ah_1_3           1
      sh_2_0           1
      ah_2_3           1
      sh_3_1           1
      used_0           1
      used_1           1
=============================================
```

Code Dictionary
product.py:
```
#!/usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC

# This example formulates and solves the following simple MIP model:
#  maximize
#        x +   y + 2 z
#  subject to
#        x + 2 y + 3 z <= 4
#        x +   y       >= 1
#        x, y, z binary

import gurobipy as gp
from gurobipy import GRB

try:

    # Create a new model
    m = gp.Model("prod")

    # Create variables
    x1 = m.addVar(vtype=GRB.INTEGER, name="x1")
    x2 = m.addVar(vtype=GRB.INTEGER, name="x2")

    # Set objective
    m.setObjective(15 * x1 + 20 * x2, GRB.MAXIMIZE)

    # Add constraint c2: x1 + 2 * x2 <= 20
    m.addConstr(x1  + 2 * x2 <= 20, "c2")

    # Add constraint c2: 2* x1 + x2 <= 20
    m.addConstr(2 * x1  +  x2 <= 20, "c3")

    # Add constraint x1>=0, x2>=0
    m.addConstr(x1 >= 0, "c4_1")
    m.addConstr(x2 >= 0, "c4_2")
```

```
    # Optimize model
    m.optimize()

    for v in m.getVars():
        print('%s %g' % (v.varName, v.x))
        print('%s %g' % (v.varName, v.x))

    print('Obj: %g' % m.objVal)

except gp.GurobiError as e:
    print('Error code ' + str(e.errno) + ': ' + str(e))

except AttributeError:
    print('Encountered an attribute error')
```

Map_skel.py

```
###########################
# mapping optimization
###########################

import gurobipy as gp
from gurobipy import GRB

print("=============================================");
print("mapping ILP optimizer");
print("=============================================");

# benchmark information

my_infty = 10000000
# num tasks: 4
num_tasks = 4
# num procs: 4
num_procs = 4
# latency constraint
latency_const = 5

# exec. info
exec_time = [
[1,2,3,4],
```

```
[2,1,3,4],
[2,3,1,4],
[2,3,3,1],
]

# edge info
edge = [
[0,1,1,0],
[0,0,0,1],
[0,0,0,1],
[0,0,0,0],
]

# cost info
cost = [1, 2, 3, 4]

# model generation
m = gp.Model("mapping")

# mapping variables m
m_var = [] # m variables
for i in range(num_tasks):
    temp_m_var = []
    mylhs = gp.LinExpr(0.0)
    for j in range(num_procs):
        temp_m_var.append(m.addVar(vtype=GRB.BINARY, name="m_" +str(i) +
"_" + str(j))) # binary variable m_i_j
        mylhs.add(temp_m_var[j])
    m.addConstr(lhs=mylhs, sense=GRB.EQUAL, rhs=1,
name="mapping_const_"+str(i))
    m_var.append(temp_m_var)

# starting time variables s
s_var = [] # s variables
for i in range(num_tasks):
    s_var.append(m.addVar(vtype=GRB.INTEGER, name="s_" + str(i))) #
integer variable s_i

# execution time variables t
t_var = [] # t variables
```

```
for i in range(num_tasks):
    t_var.append(m.addVar(vtype=GRB.INTEGER, name="t_" + str(i))) #
integer variable t_i
    myrhs = gp.LinExpr(0.0)
    for j in range(num_procs):
        myrhs.add(exec_time[i][j]* m_var[i][j])
    m.addConstr(lhs=t_var[i], sense=GRB.EQUAL, rhs=myrhs,
name="exec_time_const_"+str(i))

# latency constraints
for i in range(num_tasks):
    mylhs = gp.LinExpr(0.0)
    mylhs.add(s_var[i])
    mylhs.add(t_var[i])
    m.addConstr(lhs=mylhs, sense=GRB.LESS_EQUAL, rhs=latency_const,
name="latency_const_"+str(i))


# edge constraints
for i in range(num_tasks):
    for j in range(num_tasks):
        if edge[i][j] == 1:
            m.addConstr(s_var[i] + t_var[i] <= s_var[j],
name="edge_const_"+str(i)+"_"+str(j))

# meta variables sh, a
sh_var = []
ah_var = []
for i in range(num_tasks):
    temp_sh_var = []
    temp_ah_var = []
    for j in range(num_tasks):
        temp_sh_var.append(m.addVar(vtype=GRB.BINARY, name="sh_" + str(i)
+ "_" + str(j))) # binary variable sh_i_j
        temp_ah_var.append(m.addVar(vtype=GRB.BINARY, name="ah_" + str(i)
+ "_" + str(j))) # binary variable ah_i_j
    sh_var.append(temp_sh_var)
    ah_var.append(temp_ah_var)

# linearized scheduling constraint 1
```

```
for i in range(num_tasks):
    for j in range(num_tasks):
        if i!=j:
            # complete this part
            m.addConstr(s_var[i] + t_var[i] <= s_var[j] + (1 -
sh_var[i][j])*my_infty + (1 - ah_var[i][j])*my_infty,
name="schedulingConstraint1"+str(i)+"_"+str(j))

# linearized scheduling constraint 2
for i in range(num_tasks):
    for j in range(num_tasks):
        if i!=j:
            # complete this part
            m.addConstr(s_var[j] + t_var[j] <= s_var[i] + (1 -
sh_var[i][j])*my_infty + ah_var[i][j]*my_infty,
name="schedulingConstraint2"+str(i)+"_"+str(j))


# binary variables used
used_vars = []
for i in range(num_procs):
    used_vars.append(m.addVar(vtype=GRB.BINARY, name="used_"+str(i)))
    myrhs = gp.LinExpr(0.0)
    for j in range(num_tasks):
        m.addConstr(lhs=used_vars[i], sense=GRB.GREATER_EQUAL,
rhs=m_var[j][i], name="used_const_"+str(i)+"_"+str(j))


# sh constraints 1
for k in range(num_procs):
        for i in range(num_tasks):
                for j in range(num_tasks):
                        if i!=j:
                                mylhs = gp.LinExpr(0.0)
                                myrhs = gp.LinExpr(0.0)
                                m.addConstr(sh_var[i][j] >= m_var[i][k] +
m_var[j][k] -1, name="sh_const_"+str(k)+"_"+str(i)+"_"+str(j))

# sh constraints 2
for k in range(num_procs):
```

```
        for l in range(num_procs):
                for i in range(num_tasks):
                        for j in range(num_tasks):
                                if i!=j and k!=l:
                                        mylhs = gp.LinExpr(0.0)
                                        myrhs = gp.LinExpr(0.0)
                                        m.addConstr(sh_var[i][j] <=
(1-m_var[i][k])*my_infty + (1-m_var[j][l])*my_infty ,
name="sh_const_sec_"+str(k)+"_"+str(l)+"_"+str(i)+"_"+str(j))



# objective
myobj = gp.LinExpr(0.0)
for i in range(num_procs):
    myobj.add(used_vars[i] * cost[i])
m.setObjective(myobj, GRB.MINIMIZE)

m.write("formulation.lp")
m.optimize()

print("=============================================")
print("Optimization Results:")
m.printAttr('X')
print("=============================================")
```

P2.py

```
##########################
# mapping optimization
##########################

import gurobipy as gp
from gurobipy import GRB

print("=============================================");
print("mapping ILP optimizer");
print("=============================================");

# benchmark information
my_infty = 10000000
```

```
# num tasks: 10
num_tasks = 10
# num procs: 4
num_procs = 4
# latency constraint
latency_const = 18

# exec. info
exec_time = [
[1,2,3,4],
[2,1,3,4],
[2,3,1,4],
[2,3,4,1],
[1,2,3,4],
[2,1,3,4],
[2,3,1,4],
[2,3,4,1],
[1,2,3,4],
[2,1,3,4]
]

# edge info
edge = [
[0,1,1,0,0,0,0,0,1,0], #v0->v1,v2
[0,0,0,1,1,1,0,0,0,0], #v1->v3,v4,v5
[0,0,0,0,0,0,1,1,0,0], #v2->v6,v7
[0,0,0,0,0,0,0,0,0,0], #v3->
[0,0,0,0,0,0,0,0,0,0], #v4->
[0,0,0,0,0,0,0,0,0,0], #v5->
[0,0,0,0,0,0,0,0,1,0], #v6->v8
[0,0,0,0,0,0,0,0,0,1], #v7->v9
[0,0,0,0,0,0,0,0,0,0], #v8->
[0,0,0,0,0,0,0,0,0,0]  #v9->
]

# cost info
cost = [1, 2, 3, 4]



# model generation
```

```
m = gp.Model("mapping")

# mapping variables m
m_var = [] # m variables
for i in range(num_tasks):
    temp_m_var = []
    mylhs = gp.LinExpr(0.0)
    for j in range(num_procs):
        temp_m_var.append(m.addVar(vtype=GRB.BINARY, name="m_" +str(i) +
"_" + str(j))) # binary variable m_i_j
        mylhs.add(temp_m_var[j])
    m.addConstr(lhs=mylhs, sense=GRB.EQUAL, rhs=1,
name="mapping_const_"+str(i))
    m_var.append(temp_m_var)

# starting time variables s
s_var = [] # s variables
for i in range(num_tasks):
    s_var.append(m.addVar(vtype=GRB.INTEGER, name="s_" + str(i))) #
integer variable s_i

# execution time variables t
t_var = [] # t variables
for i in range(num_tasks):
    t_var.append(m.addVar(vtype=GRB.INTEGER, name="t_" + str(i))) #
integer variable t_i
    myrhs = gp.LinExpr(0.0)
    for j in range(num_procs):
        myrhs.add(exec_time[i][j]* m_var[i][j])
    m.addConstr(lhs=t_var[i], sense=GRB.EQUAL, rhs=myrhs,
name="exec_time_const_"+str(i))

# latency constraints
for i in range(num_tasks):
    mylhs = gp.LinExpr(0.0)
    mylhs.add(s_var[i])
    mylhs.add(t_var[i])
    m.addConstr(lhs=mylhs, sense=GRB.LESS_EQUAL, rhs=latency_const,
name="latency_const_"+str(i))
```

```
# edge constraints
for i in range(num_tasks):
    for j in range(num_tasks):
        if edge[i][j] == 1:
            m.addConstr(s_var[i] + t_var[i] <= s_var[j],
name="edge_const_"+str(i)+"_"+str(j))

# meta variables sh, a
sh_var = []
ah_var = []
for i in range(num_tasks):
    temp_sh_var = []
    temp_ah_var = []
    for j in range(num_tasks):
        temp_sh_var.append(m.addVar(vtype=GRB.BINARY, name="sh_" + str(i)
+ "_" + str(j))) # binary variable sh_i_j
        temp_ah_var.append(m.addVar(vtype=GRB.BINARY, name="ah_" + str(i)
+ "_" + str(j))) # binary variable ah_i_j
    sh_var.append(temp_sh_var)
    ah_var.append(temp_ah_var)

# linearized scheduling constraint 1
for i in range(num_tasks):
    for j in range(num_tasks):
        if i!=j:
            # complete this part
            m.addConstr(s_var[i] + t_var[i] <= s_var[j] + (1 -
sh_var[i][j])*my_infty + (1 - ah_var[i][j])*my_infty,
name="schedulingConstraint1"+str(i)+"_"+str(j))

# linearized scheduling constraint 2
for i in range(num_tasks):
    for j in range(num_tasks):
        if i!=j:
            # complete this part
            m.addConstr(s_var[j] + t_var[j] <= s_var[i] + (1 -
sh_var[i][j])*my_infty + ah_var[i][j]*my_infty,
name="schedulingConstraint2"+str(i)+"_"+str(j))
```

```
# binary variables used
used_vars = []
for i in range(num_procs):
    used_vars.append(m.addVar(vtype=GRB.BINARY, name="used_"+str(i)))
    myrhs = gp.LinExpr(0.0)
    for j in range(num_tasks):
        m.addConstr(lhs=used_vars[i], sense=GRB.GREATER_EQUAL,
rhs=m_var[j][i], name="used_const_"+str(i)+"_"+str(j))


# sh constraints 1
for k in range(num_procs):
        for i in range(num_tasks):
                for j in range(num_tasks):
                        if i!=j:
                                mylhs = gp.LinExpr(0.0)
                                myrhs = gp.LinExpr(0.0)
                                m.addConstr(sh_var[i][j] >= m_var[i][k] +
m_var[j][k] -1, name="sh_const_"+str(k)+"_"+str(i)+"_"+str(j))

# sh constraints 2
for k in range(num_procs):
        for l in range(num_procs):
                for i in range(num_tasks):
                        for j in range(num_tasks):
                                if i!=j and k!=l:
                                        mylhs = gp.LinExpr(0.0)
                                        myrhs = gp.LinExpr(0.0)
                                        m.addConstr(sh_var[i][j] <=
(1-m_var[i][k])*my_infty + (1-m_var[j][l])*my_infty ,
name="sh_const_sec_"+str(k)+"_"+str(l)+"_"+str(i)+"_"+str(j))



# objective
lat = m.addVar(vtype=GRB.INTEGER, name="lat")

# update latency constraints and redefine them as completion time
constraints
```

```
for i in range(num_tasks):
    m.addConstr(s_var[i] + t_var[i] <= lat,
name="completion_time_const_"+str(i))

# objective - minimize latency
m.setObjective(lat, GRB.MINIMIZE)

m.write("formulation.lp")
m.optimize()

print("=============================================")
print("Optimization Results:")
m.printAttr('X')
print("=============================================")
```

P3.py

```
###########################
# mapping optimization
###########################

import gurobipy as gp
from gurobipy import GRB

print("=============================================");
print("mapping ILP optimizer");
print("=============================================");

# benchmark information

my_infty = 10000000
# num tasks: 4
num_tasks = 4
# num procs: 4
num_procs = 4
# latency constraint
latency_const = 5

# exec. info
exec_time = [
[1,2,3,4],
```

```
[2,1,3,4],
[2,3,1,4],
[2,3,3,1],
]

# edge info
edge = [
[0,1,1,0],
[0,0,0,1],
[0,0,0,1],
[0,0,0,0],
]

# cost info
cost = [1, 2, 3, 4]

# model generation
m = gp.Model("mapping")

# mapping variables m
m_var = [] # m variables
for i in range(num_tasks):
    temp_m_var = []
    mylhs = gp.LinExpr(0.0)
    for j in range(num_procs):
        temp_m_var.append(m.addVar(vtype=GRB.BINARY, name="m_" +str(i) +
"_" + str(j))) # binary variable m_i_j
        mylhs.add(temp_m_var[j])
    m.addConstr(lhs=mylhs, sense=GRB.EQUAL, rhs=1,
name="mapping_const_"+str(i))
    m_var.append(temp_m_var)

# starting time variables s
s_var = [] # s variables
for i in range(num_tasks):
    s_var.append(m.addVar(vtype=GRB.INTEGER, name="s_" + str(i))) #
integer variable s_i

# execution time variables t
t_var = [] # t variables
```

```
for i in range(num_tasks):
    t_var.append(m.addVar(vtype=GRB.INTEGER, name="t_" + str(i))) #
integer variable t_i
    myrhs = gp.LinExpr(0.0)
    for j in range(num_procs):
        myrhs.add(exec_time[i][j]* m_var[i][j])
    m.addConstr(lhs=t_var[i], sense=GRB.EQUAL, rhs=myrhs,
name="exec_time_const_"+str(i))

# latency constraints
for i in range(num_tasks):
    mylhs = gp.LinExpr(0.0)
    mylhs.add(s_var[i])
    mylhs.add(t_var[i])
    m.addConstr(lhs=mylhs, sense=GRB.LESS_EQUAL, rhs=latency_const,
name="latency_const_"+str(i))


# # edge constraints
# for i in range(num_tasks):
#   for j in range(num_tasks):
#       if edge[i][j] == 1:
#           m.addConstr(s_var[i] + t_var[i] <= s_var[j],
name="edge_const_"+str(i)+"_"+str(j))

# meta variables sh, a
sh_var = []
ah_var = []
for i in range(num_tasks):
    temp_sh_var = []
    temp_ah_var = []
    for j in range(num_tasks):
        temp_sh_var.append(m.addVar(vtype=GRB.BINARY, name="sh_" + str(i)
+ "_" + str(j))) # binary variable sh_i_j
        temp_ah_var.append(m.addVar(vtype=GRB.BINARY, name="ah_" + str(i)
+ "_" + str(j))) # binary variable ah_i_j
    sh_var.append(temp_sh_var)
    ah_var.append(temp_ah_var)

# Extending edge constraints using sh_var
```

```
for i in range(num_tasks):
    for j in range(num_tasks):
        if edge[i][j] == 1:
            m.addConstr(s_var[i] + t_var[i] + 1 <= s_var[j] + my_infty *
(1 - sh_var[i][j]), name="edge_const_extended_"+str(i)+"_"+str(j))

# linearized scheduling constraint 1
for i in range(num_tasks):
    for j in range(num_tasks):
        if i!=j:
            # complete this part
            m.addConstr(s_var[i] + t_var[i] <= s_var[j] + (1 -
sh_var[i][j])*my_infty + (1 - ah_var[i][j])*my_infty,
name="schedulingConstraint1"+str(i)+"_"+str(j))

# linearized scheduling constraint 2
for i in range(num_tasks):
    for j in range(num_tasks):
        if i!=j:
            # complete this part
            m.addConstr(s_var[j] + t_var[j] <= s_var[i] + (1 -
sh_var[i][j])*my_infty + ah_var[i][j]*my_infty,
name="schedulingConstraint2"+str(i)+"_"+str(j))


# binary variables used
used_vars = []
for i in range(num_procs):
    used_vars.append(m.addVar(vtype=GRB.BINARY, name="used_"+str(i)))
    myrhs = gp.LinExpr(0.0)
    for j in range(num_tasks):
        m.addConstr(lhs=used_vars[i], sense=GRB.GREATER_EQUAL,
rhs=m_var[j][i], name="used_const_"+str(i)+"_"+str(j))


# sh constraints 1
for k in range(num_procs):
        for i in range(num_tasks):
                for j in range(num_tasks):
                        if i!=j:
```

```
                                mylhs = gp.LinExpr(0.0)
                                myrhs = gp.LinExpr(0.0)
                                m.addConstr(sh_var[i][j] >= m_var[i][k] +
m_var[j][k] -1, name="sh_const_"+str(k)+"_"+str(i)+"_"+str(j))

# sh constraints 2
for k in range(num_procs):
        for l in range(num_procs):
                for i in range(num_tasks):
                        for j in range(num_tasks):
                                if i!=j and k!=l:
                                        mylhs = gp.LinExpr(0.0)
                                        myrhs = gp.LinExpr(0.0)
                                        m.addConstr(sh_var[i][j] <=
(1-m_var[i][k])*my_infty + (1-m_var[j][l])*my_infty ,
name="sh_const_sec_"+str(k)+"_"+str(l)+"_"+str(i)+"_"+str(j))



# objective
myobj = gp.LinExpr(0.0)
for i in range(num_procs):
    myobj.add(used_vars[i] * cost[i])
m.setObjective(myobj, GRB.MINIMIZE)

m.write("formulation.lp")
m.optimize()

print("=============================================")
print("Optimization Results:")
m.printAttr('X')
print("=============================================")
```