Group 8

Thesis Defense Committee Optimization Project

Fundamentals of optimization - 20211

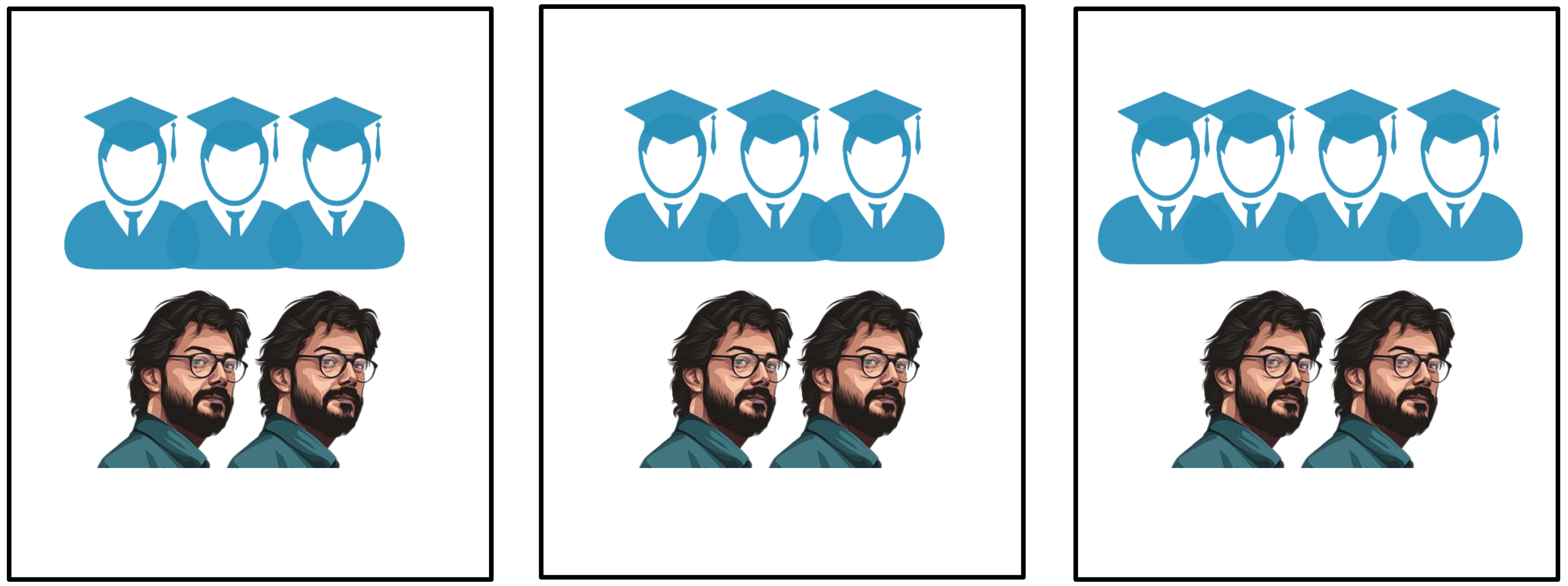
About our problem

HUST is holding thesis defense for this semester

There are N students preparing for their defense

There are M professors going to review the theses

They are going to be assigned to K committees, for example



Here’s the case when N = 10, M = 6, K = 3

To be continued

About our problem

Between students, there are theses about the same problem or sometimes they are quite similar, then the theses should be in the same committee

The similarity between theses are described by a matrix s[NxN], for example, for 6 students:

$$\begin{pmatrix} 0 & 8 & 8 & 1 & 1 & 7 \\ 8 & 0 & 8 & 1 & 1 & 9 \\ 8 & 8 & 0 & 1 & 1 & 7 \\ 1 & 1 & 1 & 0 & 8 & 1 \\ 1 & 1 & 1 & 8 & 0 & 1 \\ 7 & 9 & 7 & 1 & 1 & 0 \end{pmatrix}$$

To be continued

About our problem

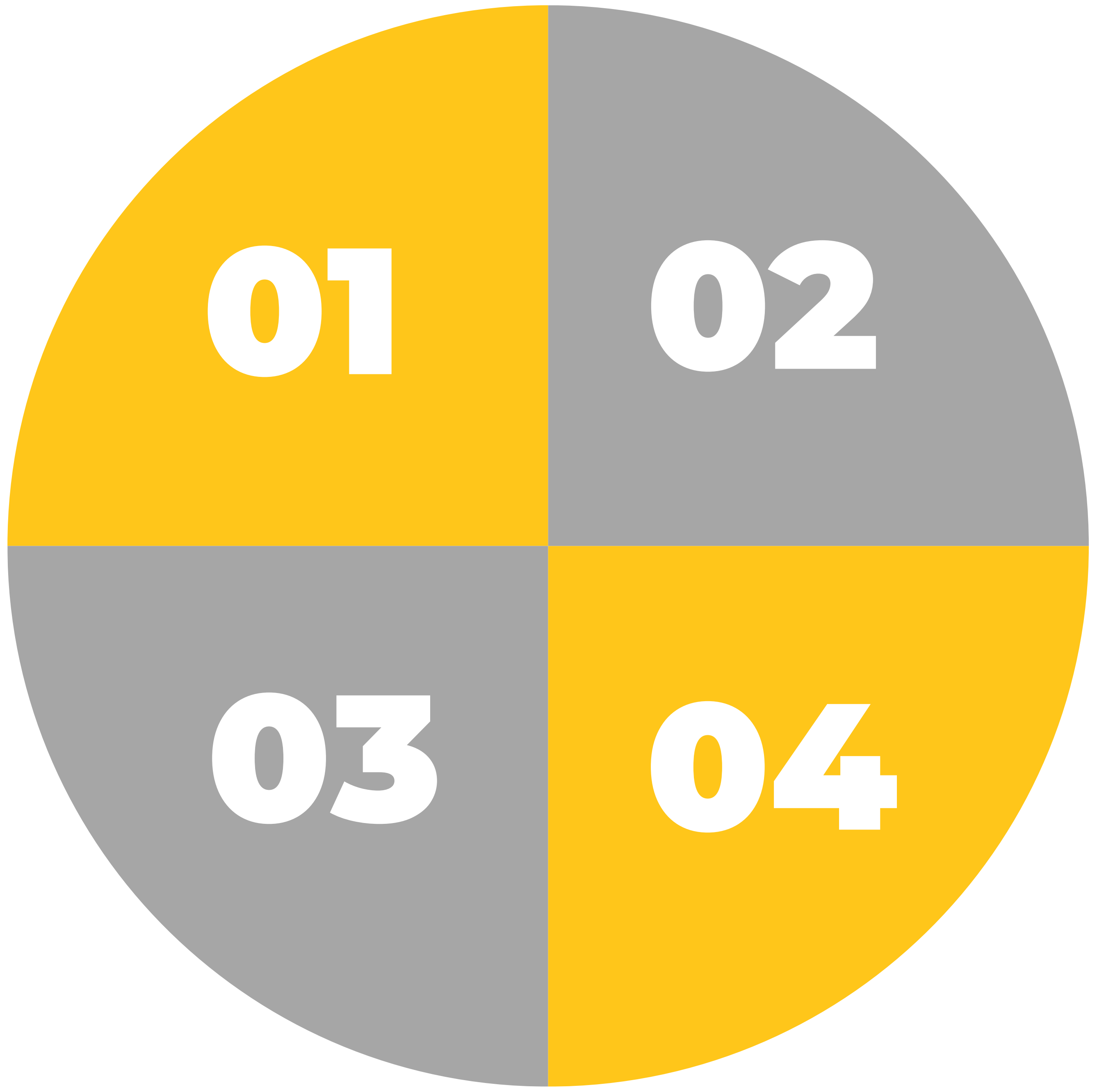
A professor has his/her own speciality and research field, and some of the theses may match his/her speciality. Then he/she should be a committee member to review those thesis

The similarity between professors and theses are described by a matrix g[MxN], for example, for 3 professors and 6 students

$$\begin{pmatrix} 3 & 8 & 4 & 9 & 1 & 7 \\ 5 & 1 & 8 & 1 & 6 & 9 \\ 6 & 6 & 4 & 1 & 2 & 7 \end{pmatrix}$$

However, the professor who instructed student A can NOT be in the same committee as student A

What to optimize?



Maximize e

The variable e is the smallest similarity index between students in the same committee

Maximize f

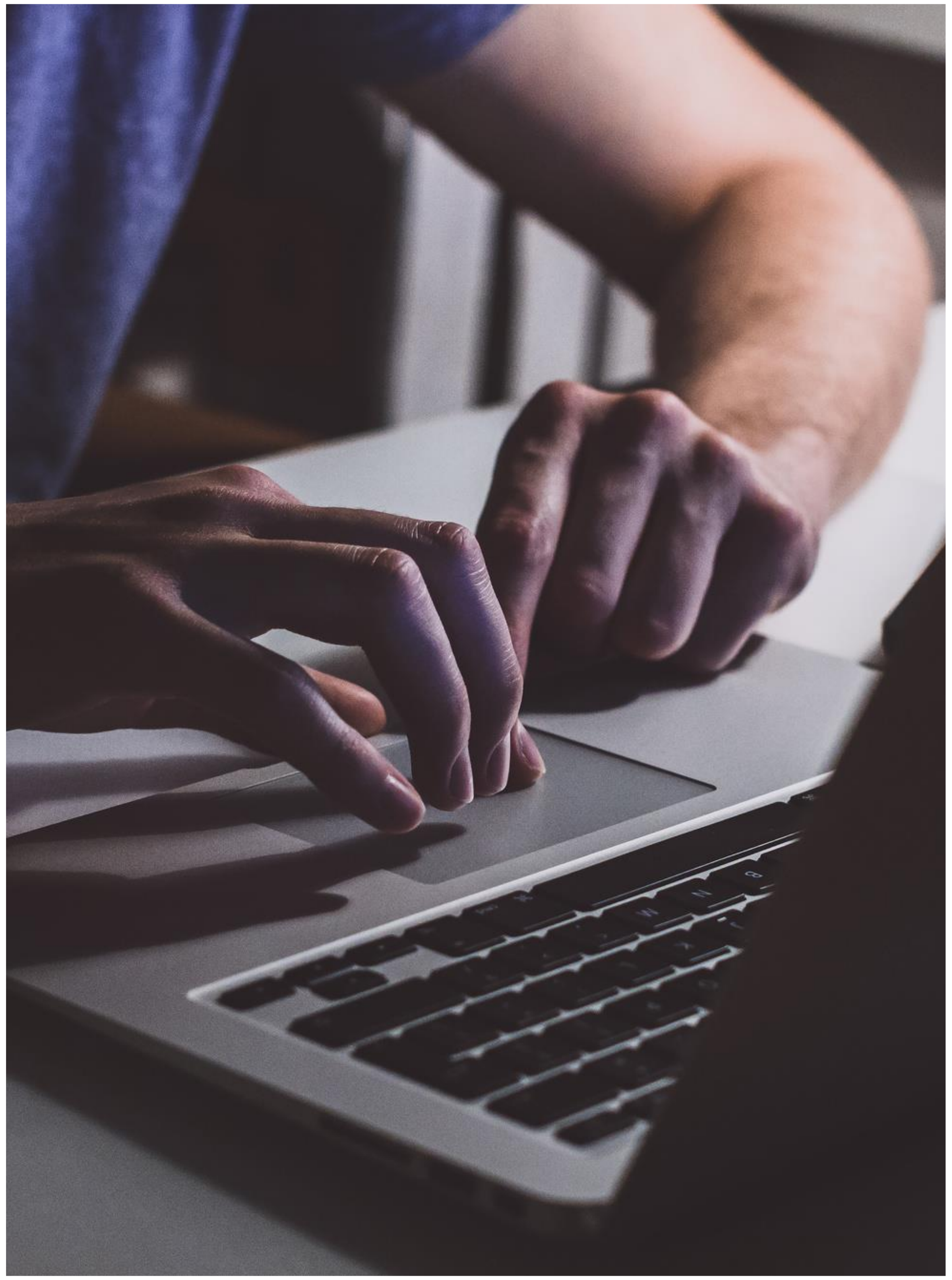
The variable f is the smallest similarity index between students and professors in the same committee

Maximize total of similarity index

So that every committee has an acceptable value of similarity index

Mixed

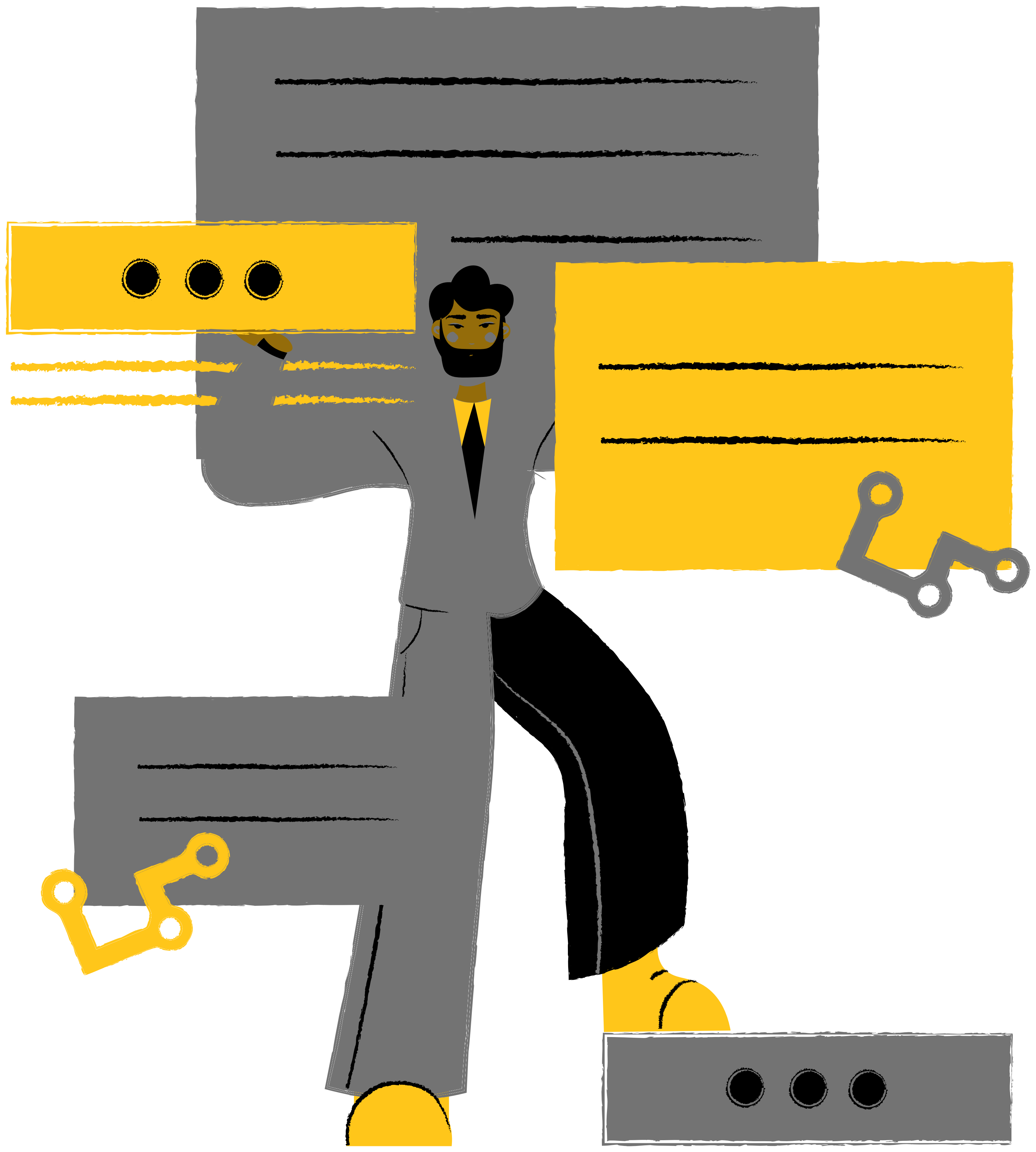
Maximize
$\alpha * e + \beta * f + \gamma * Total$
α, β, γ depend on the priority order we set



CONTENT OF INPUT FILE

- First row: N, M and K
- Second row: a , b – Minimum and maximum number of students/ committee
  c, d – Minimum and maximum number of professors/ committee
  e – Minimum similarity index between students in the same committee
  f – Minimum similarity index between professors and students in the same committee
- Next N rows: matrix s
- Next M rows: matrix g
- Last row, t[Size N]: element [i] means which professor instruct student i+1

CONTENTS



01

CONSTRAINT PROGRAMMING

```
from ortools.sat.python import cp_model
```

CONSTRAINT PROGRAMMING – ENUMERATE ALL SOLUTIONS

Declare variables

$x[i,j]=1$, means that thesis i is presented in committee j ($z[i,j]=0$ otherwise) $\forall i=1 \rightarrow N \ and \ \forall j=1 \rightarrow K$

$y[i,j]=1$, means that professor i is assign to committee j ($y[i,j]=0$ otherwise)$\forall i=1 \rightarrow M \ and \ \forall j=1 \rightarrow K$

$x_{int}[i]$ is the committee for thesis i

$y_{int}[i]$ is the committee for professor i

Code

```
x_bin = [[model.NewIntVar(0,1, 'x_bin['+str(i)+']['+str(j)+']') for j in range(K)] for i in range(N)]
x_int = [model.NewIntVar(0,K-1, 'x_int['+str(u)+']') for u in range(N)]
y_bin = [[model.NewIntVar(0,1, 'y_bin['+str(m)+']['+str(n)+']') for n in range(K)] for m in range(M)]
y_int = [model.NewIntVar(0,K-1, 'y_int['+str(v)+']') for v in range(M)]
```

CONSTRAINT PROGRAMMING – ENUMERATE ALL SOLUTIONS

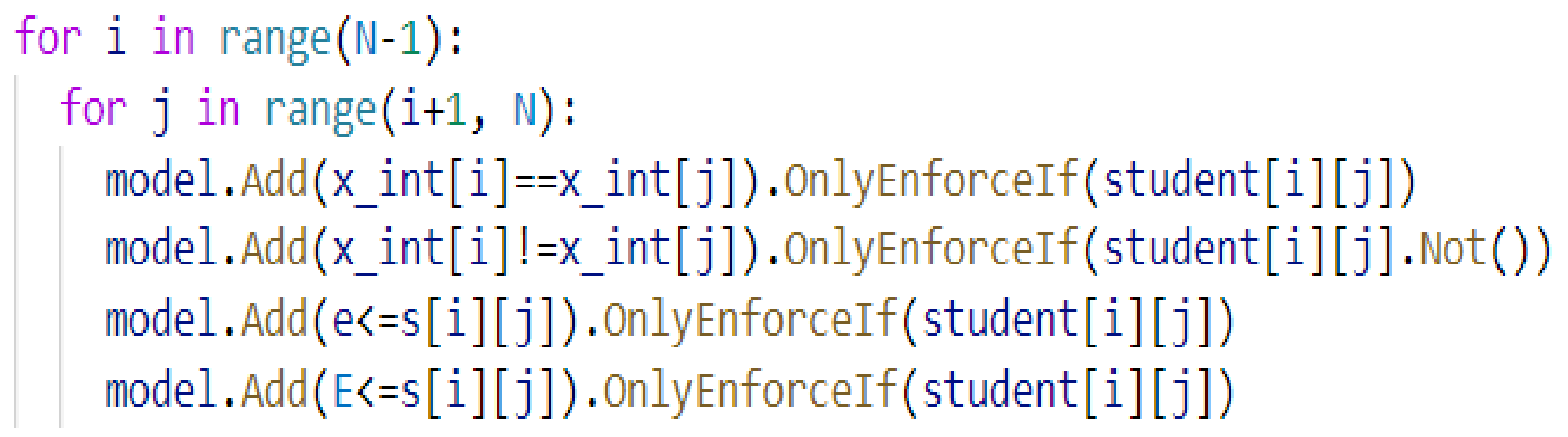
Variable constraint

$$x_{int}[i] = j \rightarrow x[i,j] = 1 \; (\forall i = 1 \rightarrow N \text{ and } \forall j = 1 \rightarrow K)$$

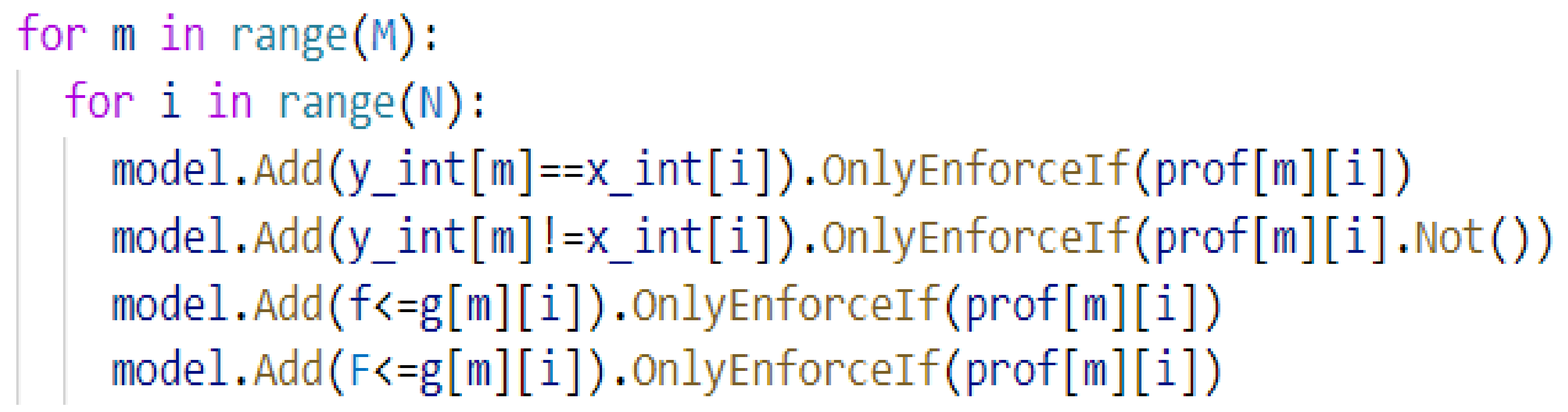
$$x_{int}[i] \neq j \rightarrow x[i,j] = 1 \; (\forall i = 1 \rightarrow N \text{ and } \forall j = 1 \rightarrow K)$$

$$y_{int}[i] = j \rightarrow y[i,j] = 1 \; (\forall i = 1 \rightarrow M \text{ and } \forall j = 1 \rightarrow K)$$

$$y_{int}[i] \neq j \rightarrow y[i,j] = 0 \; (\forall i = 1 \rightarrow M \text{ and } \forall j = 1 \rightarrow K)$$



```
for i in range(N-1):
    for j in range(i+1, N):
        model.Add(x_int[i]==x_int[j]).OnlyEnforceIf(student[i][j])
        model.Add(x_int[i]!=x_int[j]).OnlyEnforceIf(student[i][j].Not())
        model.Add(e<=s[i][j]).OnlyEnforceIf(student[i][j])
        model.Add(E<=s[i][j]).OnlyEnforceIf(student[i][j])
```



```
for m in range(M):
    for i in range(N):
        model.Add(y_int[m]==x_int[i]).OnlyEnforceIf(prof[m][i])
        model.Add(y_int[m]!=x_int[i]).OnlyEnforceIf(prof[m][i].Not())
        model.Add(f<=g[m][i]).OnlyEnforceIf(prof[m][i])
        model.Add(F<=g[m][i]).OnlyEnforceIf(prof[m][i])
```

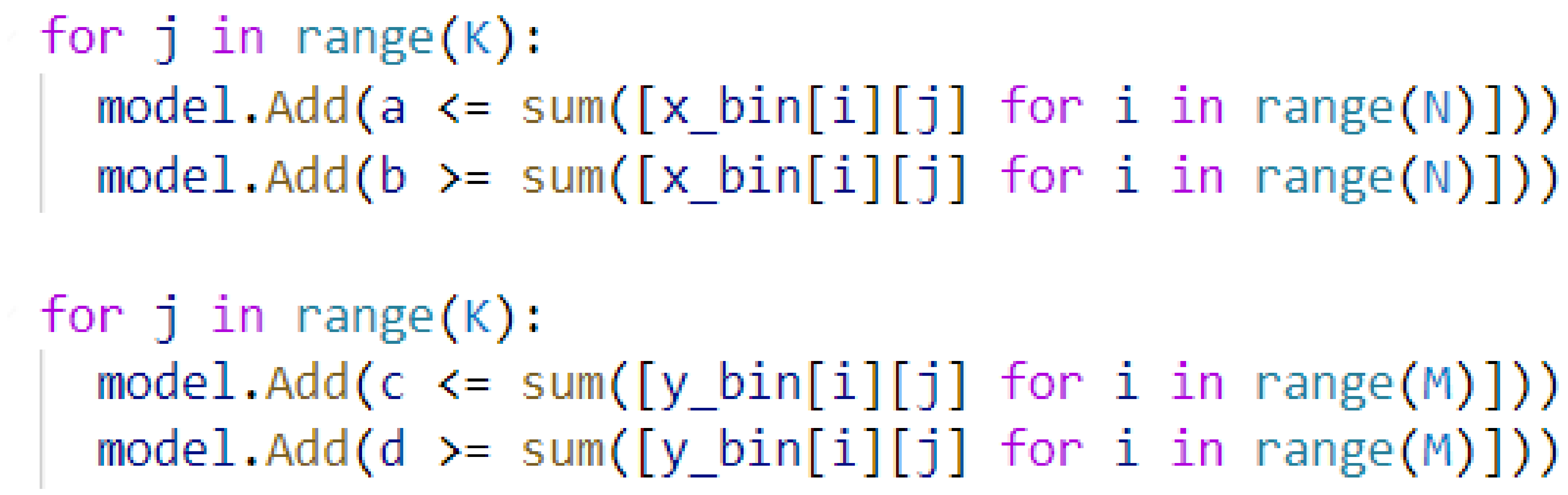
CONSTRAINT PROGRAMMING – ENUMERATE ALL SOLUTIONS

a , b – Minimum and maximum number of students/ committee

c, d – Minimum and maximum number of professors/ committee

$$a \le \sum_{i=1}^{N} x[i,j] \le b, \forall j = 1 \to K$$

$$c \le \sum_{i=1}^{M} y[i,j] \le d, \forall j = 1 \to K$$

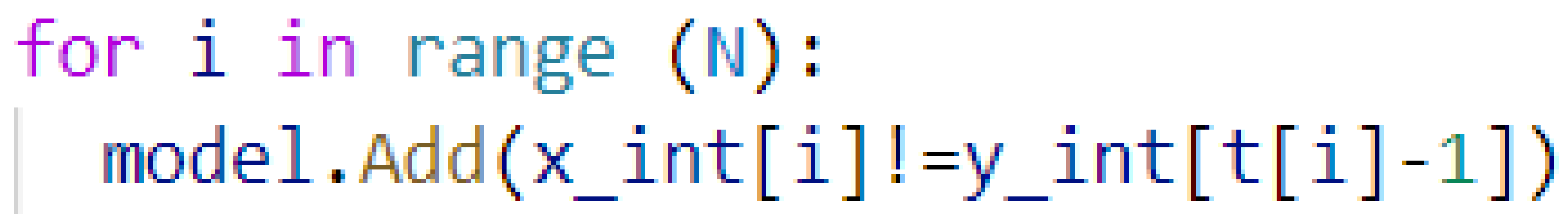


```
for j in range(K):
  model.Add(a <= sum([x_bin[i][j] for i in range(N)]))
  model.Add(b >= sum([x_bin[i][j] for i in range(N)]))

for j in range(K):
  model.Add(c <= sum([y_bin[i][j] for i in range(M)]))
  model.Add(d >= sum([y_bin[i][j] for i in range(M)]))
```

CONSTRAINT PROGRAMMING – ENUMERATE ALL SOLUTIONS

Last row, t[Size N]: element [i] = j means which professor j instructed student i+1, and thus they won’t be in the same committee

Code



```
for i in range (N):
  model.Add(x_int[i]!=y_int[t[i]-1])
```

This line means student i+1 is in a committee different from his/her instructor

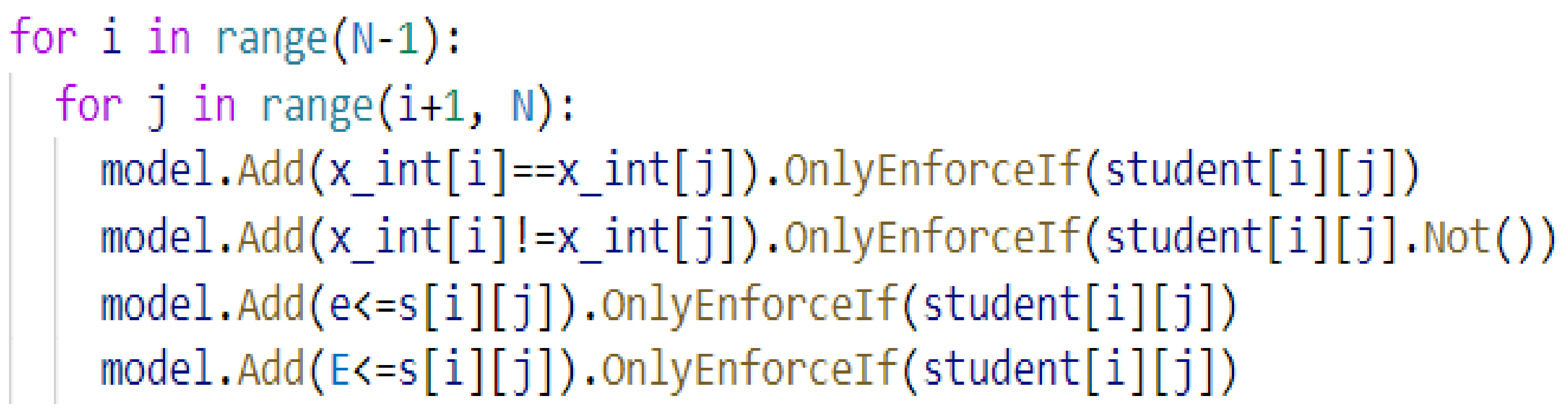
CONSTRAINT PROGRAMMING – ENUMERATE ALL SOLUTIONS

e – Minimum similarity index between students in the same committee

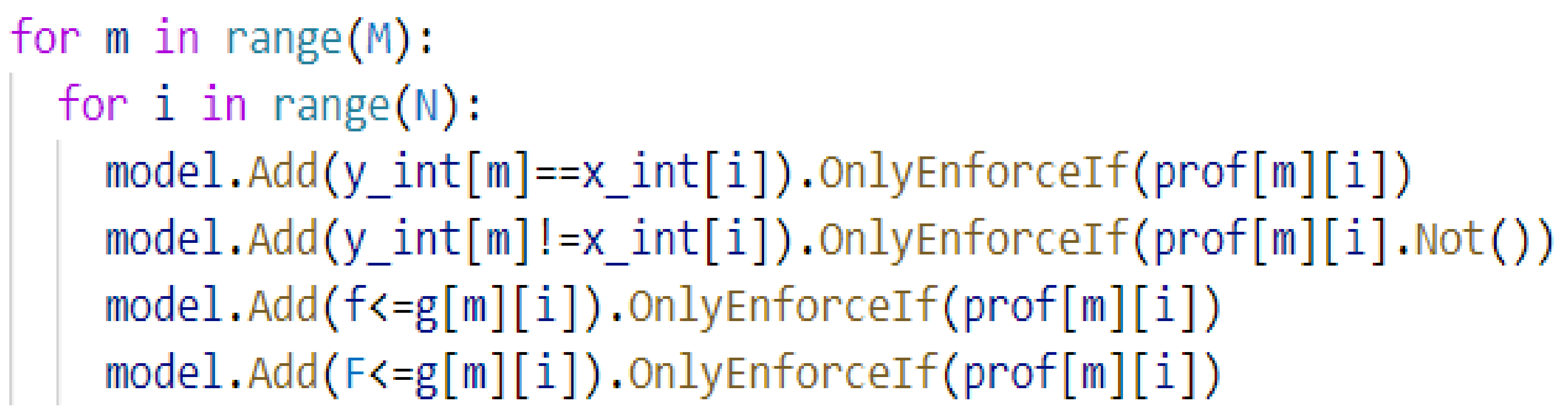
$$\forall i = 1 \rightarrow N-1, \forall j = i+1 \rightarrow N : x_{int}[i] = x_{int}[j] \rightarrow e \leq s[i][j]$$

f – Minimum similarity index between professors and students in the same committee

$$\forall i = 1 \rightarrow N, \forall j = 1 \rightarrow M : x_{int}[i] = y_{int}[j] \rightarrow f \leq g[j][i]$$



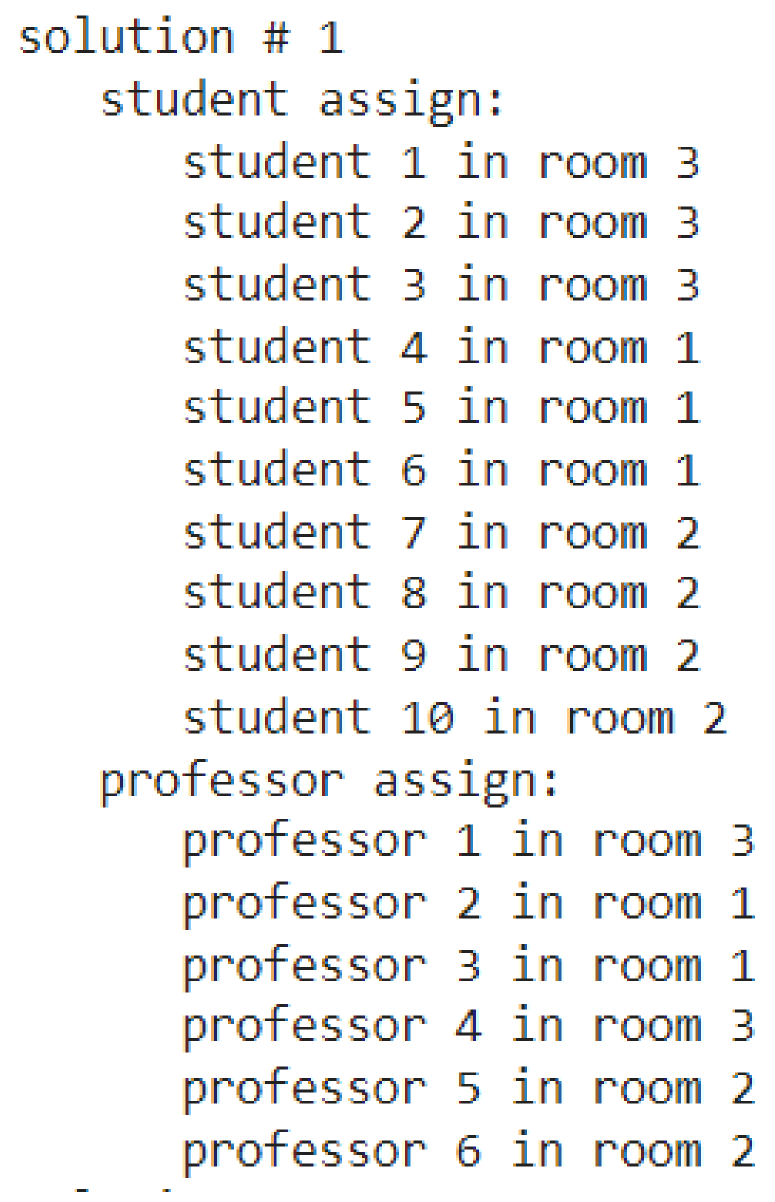
```
for i in range(N-1):
  for j in range(i+1, N):
    model.Add(x_int[i]==x_int[j]).OnlyEnforceIf(student[i][j])
    model.Add(x_int[i]!=x_int[j]).OnlyEnforceIf(student[i][j].Not())
    model.Add(e<=s[i][j]).OnlyEnforceIf(student[i][j])
    model.Add(E<=s[i][j]).OnlyEnforceIf(student[i][j])
```



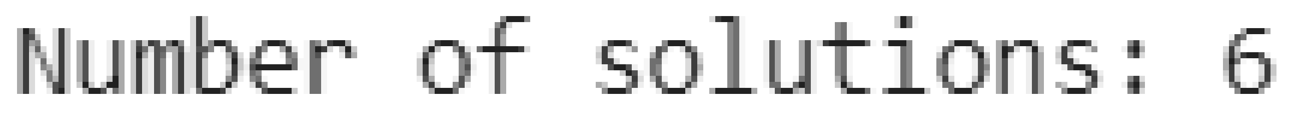
```
for m in range(M):
  for i in range(N):
    model.Add(y_int[m]==x_int[i]).OnlyEnforceIf(prof[m][i])
    model.Add(y_int[m]!=x_int[i]).OnlyEnforceIf(prof[m][i].Not())
    model.Add(f<=g[m][i]).OnlyEnforceIf(prof[m][i])
    model.Add(F<=g[m][i]).OnlyEnforceIf(prof[m][i])
```

CONSTRAINT PROGRAMMING – ENUMERATE ALL SOLUTIONS

Result (example testcase)



```
solution # 1
   student assign:
      student 1 in room 3
      student 2 in room 3
      student 3 in room 3
      student 4 in room 1
      student 5 in room 1
      student 6 in room 1
      student 7 in room 2
      student 8 in room 2
      student 9 in room 2
      student 10 in room 2
   professor assign:
      professor 1 in room 3
      professor 2 in room 1
      professor 3 in room 1
      professor 4 in room 3
      professor 5 in room 2
      professor 6 in room 2
```



```
Number of solutions: 6
```

We can find optimal value from these solutions, but it is not possible when number of solutions increases

CONSTRAINT PROGRAMMING – OPTIMIZING

$$f(Total, E, F) = \alpha * E + \beta * F + \gamma * Total \rightarrow \text{MAXIMUM}$$

CONSTRAINT PROGRAMMING – OPTIMIZING

Problems with enumeration:

- K committees -> at least K! solutions
- No variables to link: If student A and B are in the same committee?
  If student X and professor Y are in the same committee?

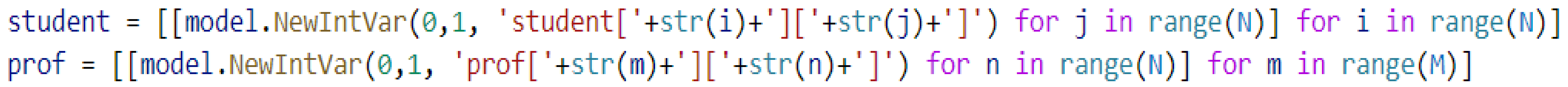
ix: declare new variables, add their constraints

$student[i,j] = 1$ means that student i, student j will present in the same committee ($student[i,j] = 0$ otherwise)$\forall i = 1 \rightarrow N \; and \; \forall j = 1 \rightarrow N$

$prof[i,j] = 1$ means that thesis j and professor i are in the same committee ($prof[i,j] = 0$ otherwise)$\forall i = 1 \rightarrow M \; and \; \forall j = 1 \rightarrow N$

CONSTRAINT PROGRAMMING – OPTIMIZING

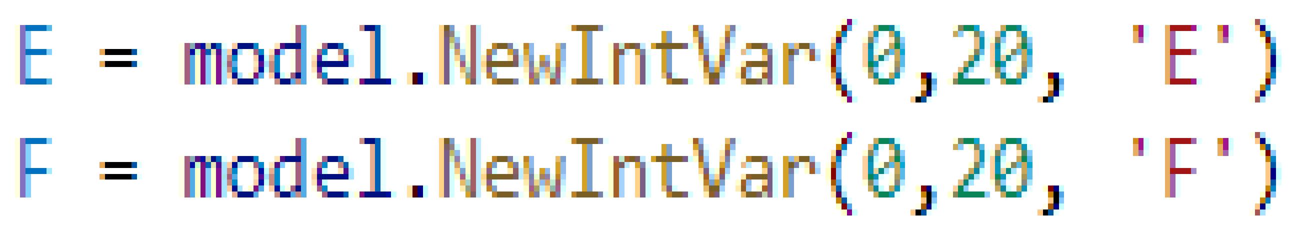
Codes



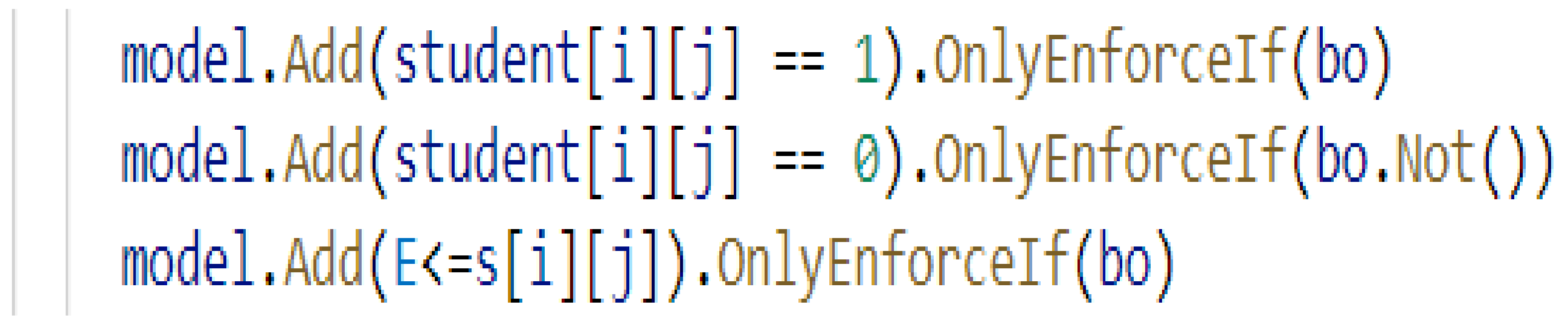
```
student = [[model.NewIntVar(0,1, 'student['+str(i)+']['+str(j)+']') for j in range(N)] for i in range(N)]
prof = [[model.NewIntVar(0,1, 'prof['+str(m)+']['+str(n)+']') for n in range(N)] for m in range(M)]
```

CONSTRAINT PROGRAMMING – OPTIMIZING

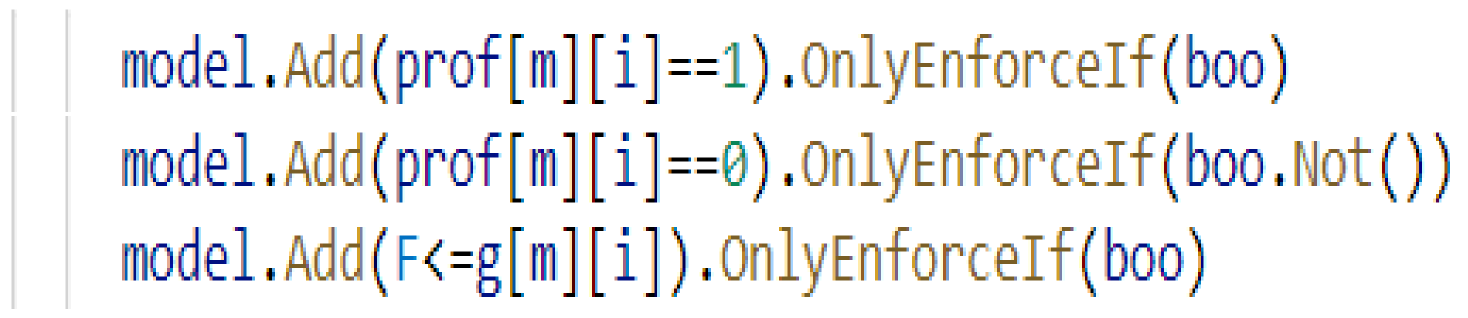
Declare E, F, Total



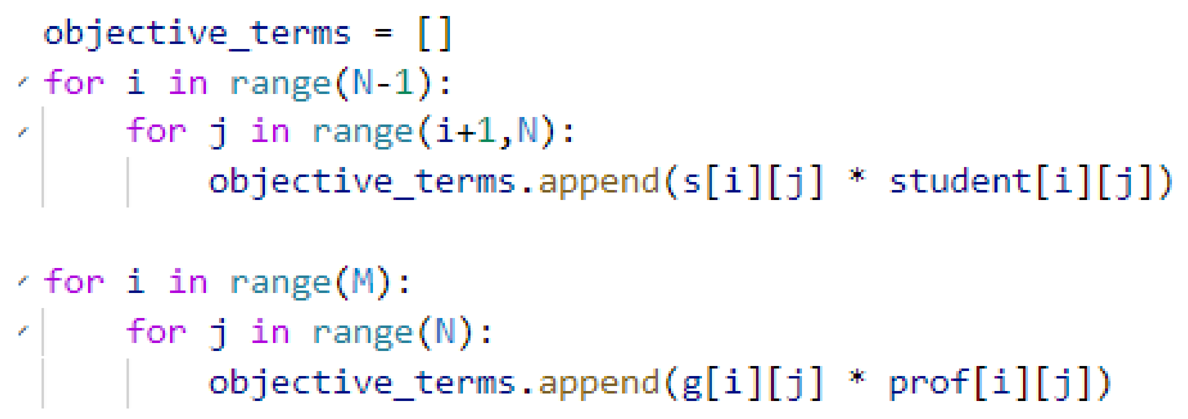
```
E = model.NewIntVar(0,20, 'E')
F = model.NewIntVar(0,20, 'F')
```



```
model.Add(student[i][j] == 1).OnlyEnforceIf(bo)
model.Add(student[i][j] == 0).OnlyEnforceIf(bo.Not())
model.Add(E<=s[i][j]).OnlyEnforceIf(bo)
```



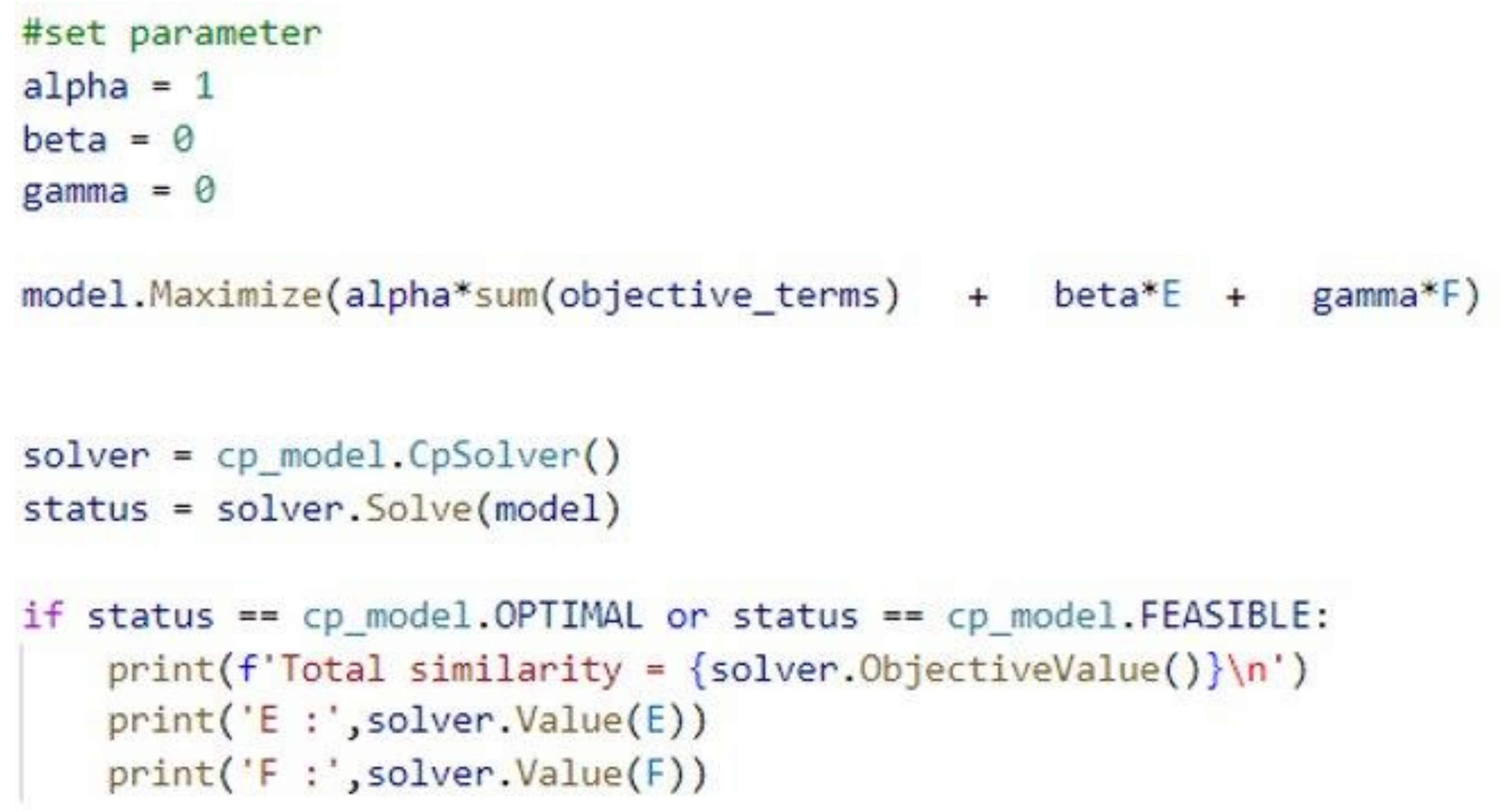
```
model.Add(prof[m][i]==1).OnlyEnforceIf(boo)
model.Add(prof[m][i]==0).OnlyEnforceIf(boo.Not())
model.Add(F<=g[m][i]).OnlyEnforceIf(boo)
```



```
objective_terms = []
for i in range(N-1):
    for j in range(i+1,N):
        objective_terms.append(s[i][j] * student[i][j])

for i in range(M):
    for j in range(N):
        objective_terms.append(g[i][j] * prof[i][j])
```

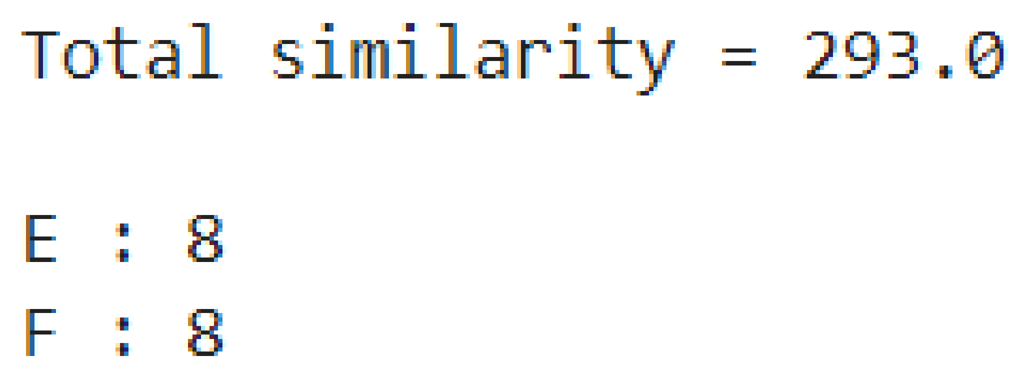
CONSTRAINT PROGRAMMING – OPTIMIZING

Objective function



```
#set parameter
alpha = 1
beta = 0
gamma = 0

model.Maximize(alpha*sum(objective_terms)   +   beta*E  +   gamma*F)


solver = cp_model.CpSolver()
status = solver.Solve(model)

if status == cp_model.OPTIMAL or status == cp_model.FEASIBLE:
    print(f'Total similarity = {solver.ObjectiveValue()}\n')
    print('E :',solver.Value(E))
    print('F :',solver.Value(F))
```

Result (example testcase)

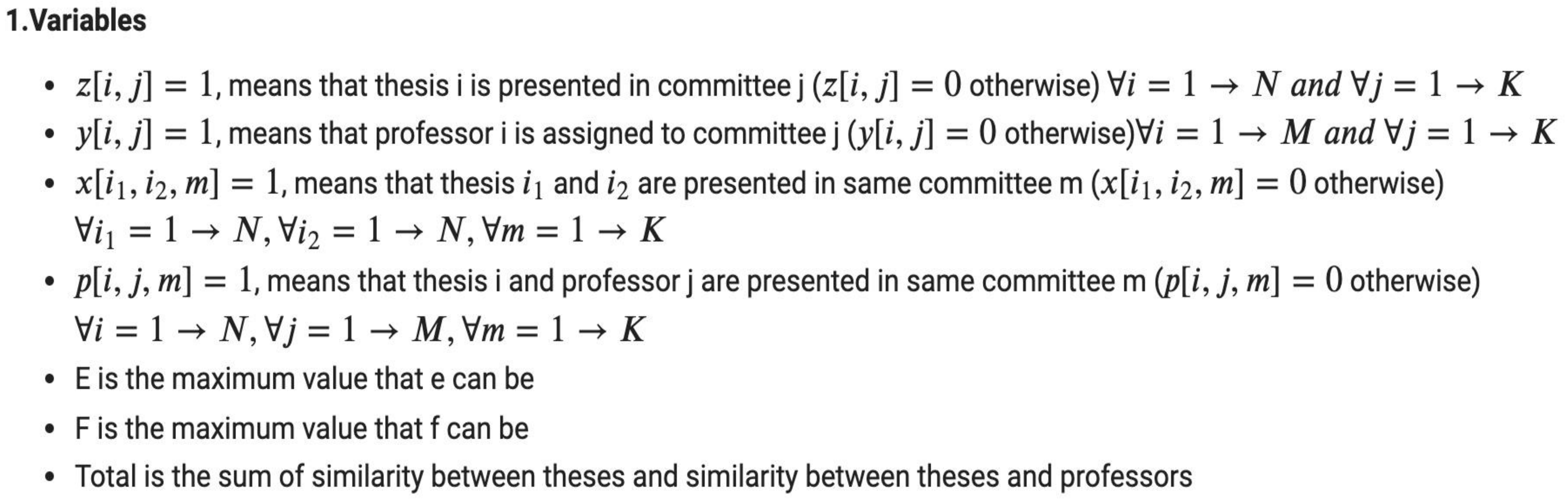


```
Total similarity = 293.0

E : 8
F : 8
```

02

MIXED INTEGER PROGRAMMING

Declaring variables



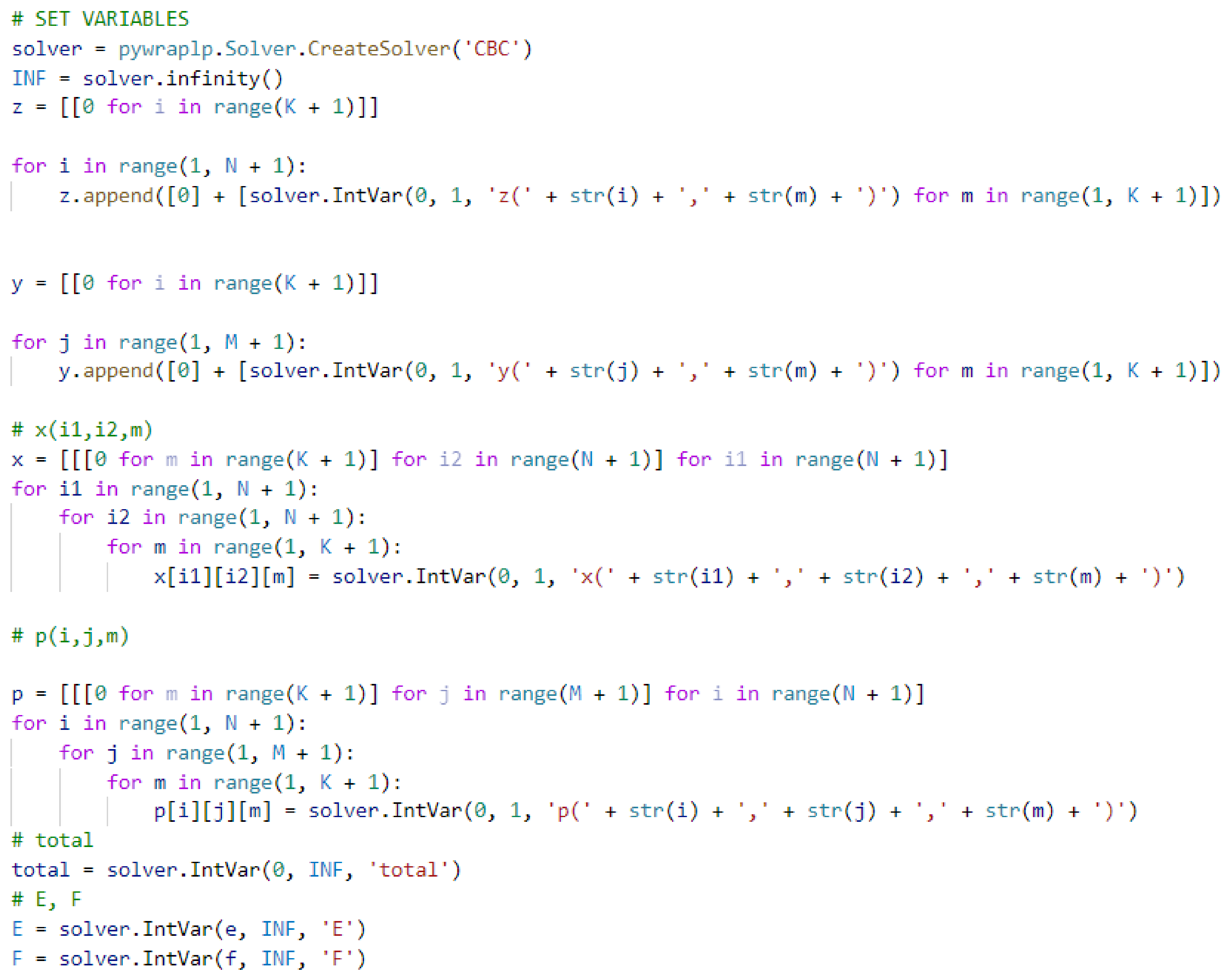
1.Variables

- $z[i,j]=1$, means that thesis i is presented in committee j ($z[i,j]=0$ otherwise) $\forall i=1\rightarrow N\ and\ \forall j=1\rightarrow K$
- $y[i,j]=1$, means that professor i is assigned to committee j ($y[i,j]=0$ otherwise)$\forall i=1\rightarrow M\ and\ \forall j=1\rightarrow K$
- $x[i_1,i_2,m]=1$, means that thesis i_1 and i_2 are presented in same committee m ($x[i_1,i_2,m]=0$ otherwise) $\forall i_1=1\rightarrow N, \forall i_2=1\rightarrow N, \forall m=1\rightarrow K$
- $p[i,j,m]=1$, means that thesis i and professor j are presented in same committee m ($p[i,j,m]=0$ otherwise) $\forall i=1\rightarrow N, \forall j=1\rightarrow M, \forall m=1\rightarrow K$
- E is the maximum value that e can be
- F is the maximum value that f can be
- Total is the sum of similarity between theses and similarity between theses and professors

02

MIXED INTEGER PROGRAMMING

Code for declaring variables



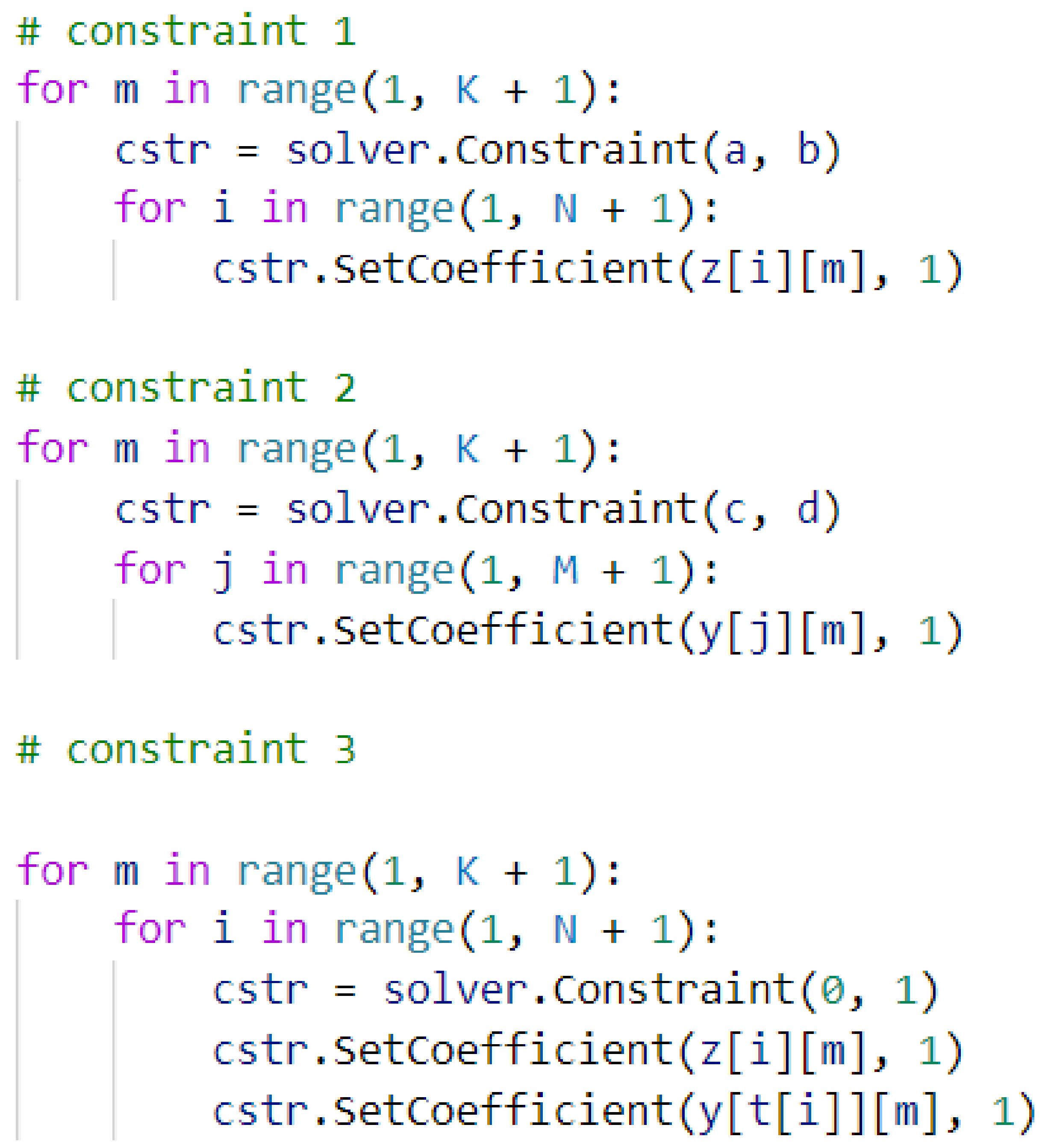
```python
# SET VARIABLES
solver = pywraplp.Solver.CreateSolver('CBC')
INF = solver.infinity()
z = [[0 for i in range(K + 1)]]

for i in range(1, N + 1):
    z.append([0] + [solver.IntVar(0, 1, 'z(' + str(i) + ',' + str(m) + ')') for m in range(1, K + 1)])


y = [[0 for i in range(K + 1)]]

for j in range(1, M + 1):
    y.append([0] + [solver.IntVar(0, 1, 'y(' + str(j) + ',' + str(m) + ')') for m in range(1, K + 1)])

# x(i1,i2,m)
x = [[[0 for m in range(K + 1)] for i2 in range(N + 1)] for i1 in range(N + 1)]
for i1 in range(1, N + 1):
    for i2 in range(1, N + 1):
        for m in range(1, K + 1):
            x[i1][i2][m] = solver.IntVar(0, 1, 'x(' + str(i1) + ',' + str(i2) + ',' + str(m) + ')')

# p(i,j,m)

p = [[[0 for m in range(K + 1)] for j in range(M + 1)] for i in range(N + 1)]
for i in range(1, N + 1):
    for j in range(1, M + 1):
        for m in range(1, K + 1):
            p[i][j][m] = solver.IntVar(0, 1, 'p(' + str(i) + ',' + str(j) + ',' + str(m) + ')')
# total
total = solver.IntVar(0, INF, 'total')
# E, F
E = solver.IntVar(e, INF, 'E')
F = solver.IntVar(f, INF, 'F')
```

MIXED INTERGER PROGRAMMING

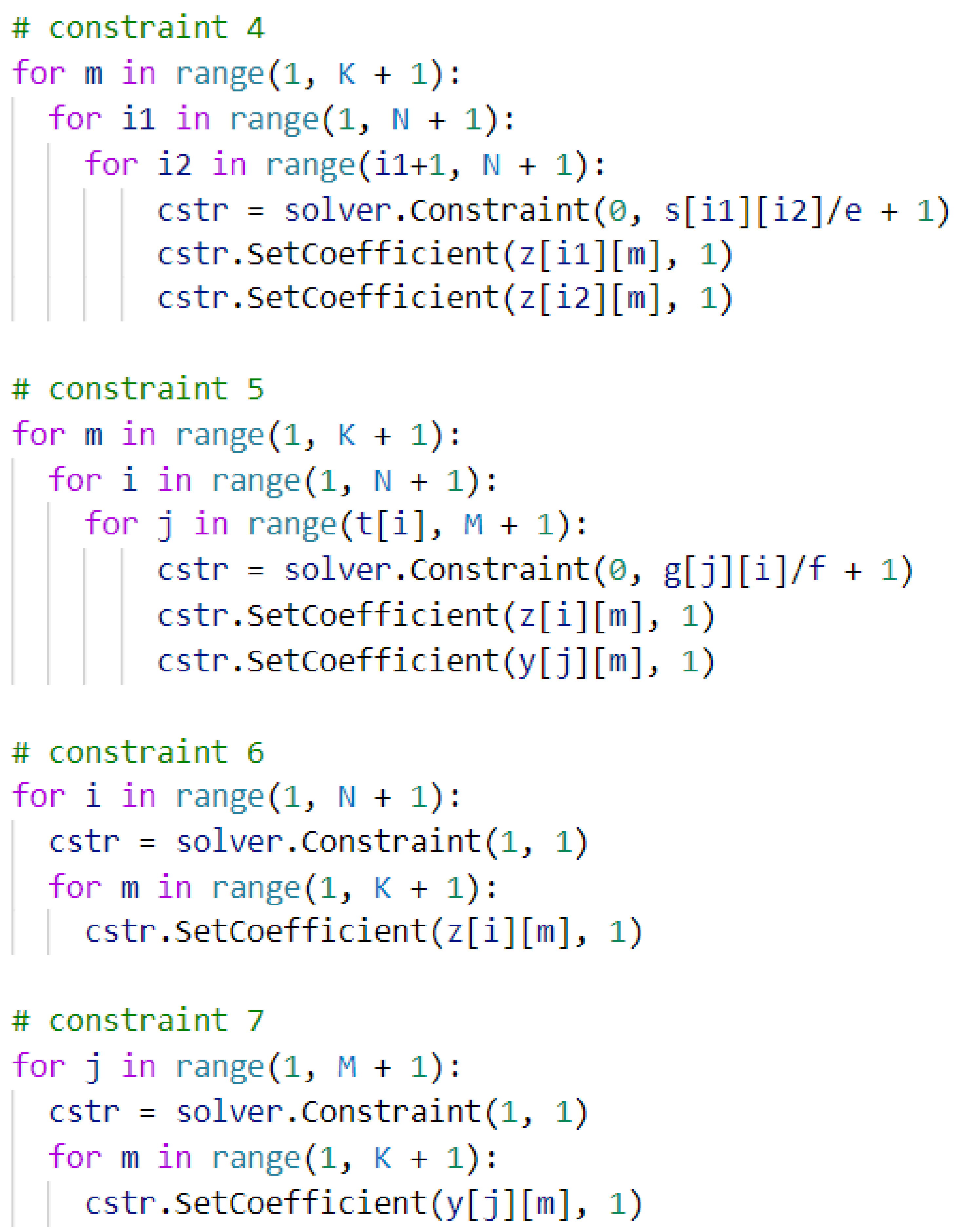
2.Constraint

- $a \leq \sum_{i=1}^{N} z[i,j] \leq b, \forall j = 1 \rightarrow K$
- $c \leq \sum_{i=1}^{M} y[i,j] \leq d, \forall j = 1 \rightarrow K$
- $\sum_{j=1}^{K} y[i,j] = 1, \forall i = 1 \rightarrow M$
- $\sum_{j=1}^{K} z[i,j] = 1, \forall i = 1 \rightarrow N$
- $z[i,j] + y[t(i),j] = 1\ , \forall i = 1 \rightarrow N\ , \forall j = 1 \rightarrow K$
- $\forall i_1 = 1 \rightarrow N, \forall i_2 = i_1 + 1 \rightarrow N, \forall m = 1 \rightarrow K :\ \ z[i_1,m] + z[i_2,m] \leq s[i_1,i_2]/e + 1$
- $\forall m = 1 \rightarrow K, \forall i = 1 \rightarrow N, \forall j = t(i) \rightarrow M :\ z[i,m] + y[j,m] \leq g[i,j]/f + 1$

MIXED INTERGER PROGRAMMING



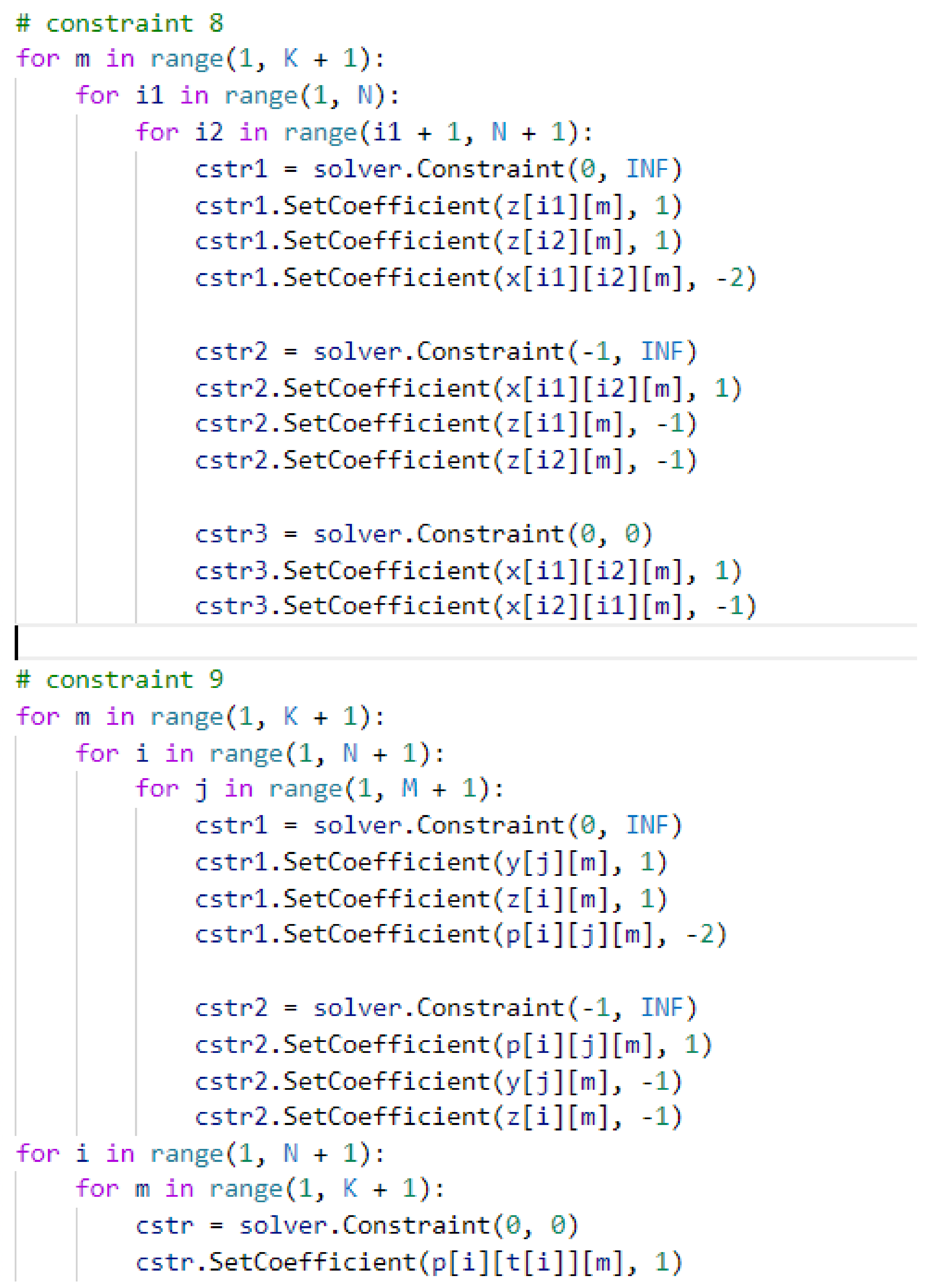
```python
# constraint 1
for m in range(1, K + 1):
    cstr = solver.Constraint(a, b)
    for i in range(1, N + 1):
        cstr.SetCoefficient(z[i][m], 1)

# constraint 2
for m in range(1, K + 1):
    cstr = solver.Constraint(c, d)
    for j in range(1, M + 1):
        cstr.SetCoefficient(y[j][m], 1)

# constraint 3

for m in range(1, K + 1):
    for i in range(1, N + 1):
        cstr = solver.Constraint(0, 1)
        cstr.SetCoefficient(z[i][m], 1)
        cstr.SetCoefficient(y[t[i]][m], 1)
```



```python
# constraint 4
for m in range(1, K + 1):
  for i1 in range(1, N + 1):
    for i2 in range(i1+1, N + 1):
      cstr = solver.Constraint(0, s[i1][i2]/e + 1)
      cstr.SetCoefficient(z[i1][m], 1)
      cstr.SetCoefficient(z[i2][m], 1)

# constraint 5
for m in range(1, K + 1):
  for i in range(1, N + 1):
    for j in range(t[i], M + 1):
      cstr = solver.Constraint(0, g[j][i]/f + 1)
      cstr.SetCoefficient(z[i][m], 1)
      cstr.SetCoefficient(y[j][m], 1)

# constraint 6
for i in range(1, N + 1):
  cstr = solver.Constraint(1, 1)
  for m in range(1, K + 1):
    cstr.SetCoefficient(z[i][m], 1)

# constraint 7
for j in range(1, M + 1):
  cstr = solver.Constraint(1, 1)
  for m in range(1, K + 1):
    cstr.SetCoefficient(y[j][m], 1)
```

MIXED INTEGER PROGRAMMING

- $\forall i_1 = 1 \to N, \forall i_2 = 1 \to N, \forall m = 1 \to K$:
  $z(i_1, m) + z(i_2, m) \geq 2 * x(i_1, i_2, m)$
  $x(i_1, i_2, m) \geq z(i_1, m) + z(i_2, m) - 1$
  $x(i_1, i_2, m) = x(i_2, i_1, m)$
- $\forall i = 1 \to N, \forall j = 1 \to M, \forall m = 1 \to K$:
  $y(i, m) + z(i, m) \geq 2p(i, j, m)$
  $p(i, j, m) \geq y(i, m) + z(i, m) - 1$
- $\forall i = 1 \to N, \forall m = 1 \to K : p(i, t(i), m) = 0$



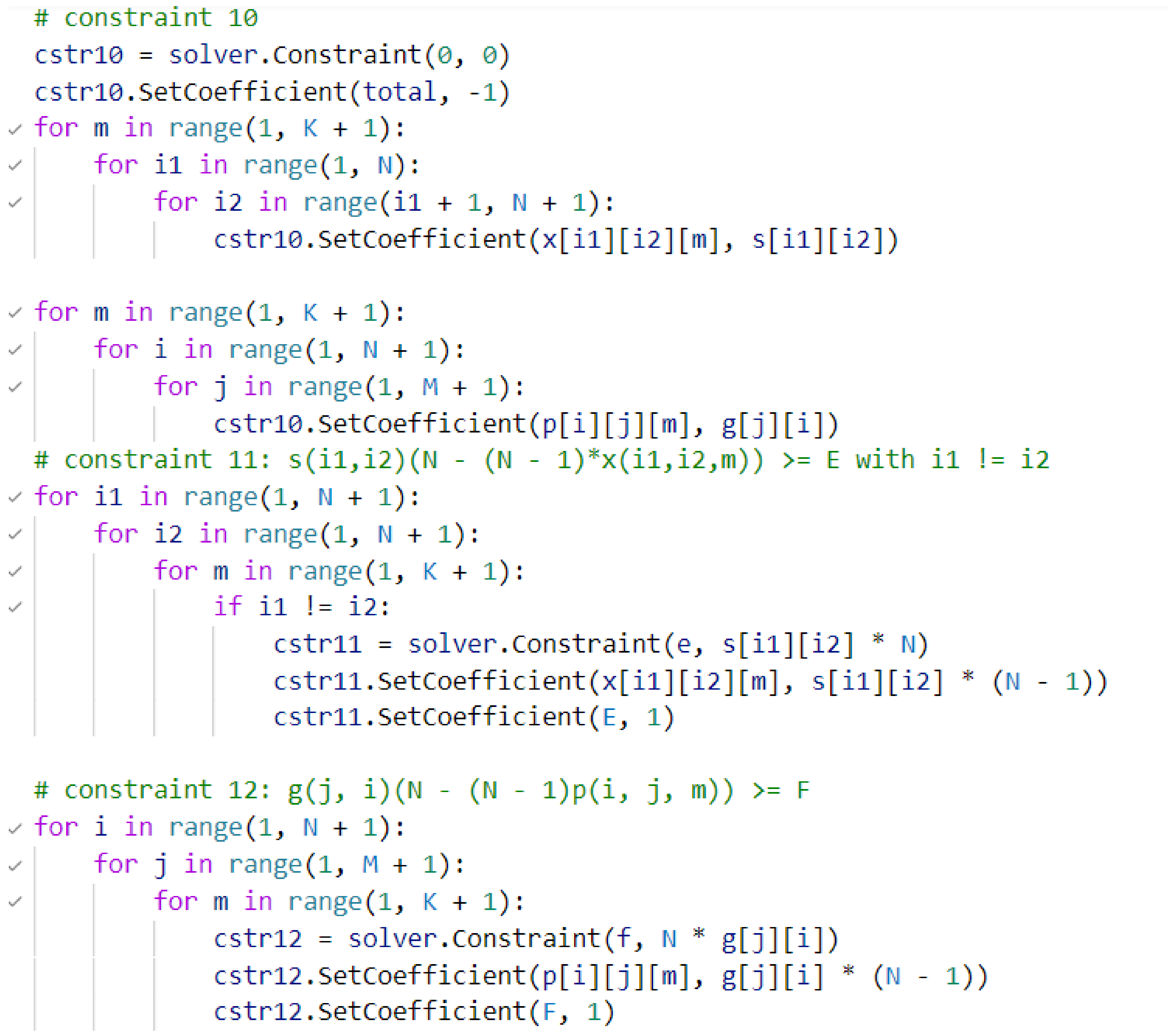
```
# constraint 8
for m in range(1, K + 1):
    for i1 in range(1, N):
        for i2 in range(i1 + 1, N + 1):
            cstr1 = solver.Constraint(0, INF)
            cstr1.SetCoefficient(z[i1][m], 1)
            cstr1.SetCoefficient(z[i2][m], 1)
            cstr1.SetCoefficient(x[i1][i2][m], -2)

            cstr2 = solver.Constraint(-1, INF)
            cstr2.SetCoefficient(x[i1][i2][m], 1)
            cstr2.SetCoefficient(z[i1][m], -1)
            cstr2.SetCoefficient(z[i2][m], -1)

            cstr3 = solver.Constraint(0, 0)
            cstr3.SetCoefficient(x[i1][i2][m], 1)
            cstr3.SetCoefficient(x[i2][i1][m], -1)

# constraint 9
for m in range(1, K + 1):
    for i in range(1, N + 1):
        for j in range(1, M + 1):
            cstr1 = solver.Constraint(0, INF)
            cstr1.SetCoefficient(y[j][m], 1)
            cstr1.SetCoefficient(z[i][m], 1)
            cstr1.SetCoefficient(p[i][j][m], -2)

            cstr2 = solver.Constraint(-1, INF)
            cstr2.SetCoefficient(p[i][j][m], 1)
            cstr2.SetCoefficient(y[j][m], -1)
            cstr2.SetCoefficient(z[i][m], -1)
for i in range(1, N + 1):
    for m in range(1, K + 1):
        cstr = solver.Constraint(0, 0)
        cstr.SetCoefficient(p[i][t[i]][m], 1)
```

MIXED INTEGER PROGRAMMING

- $Total = \sum_{m=1}^{M} [\sum_{i_1=1,i_2=1}^{N} x(i1, i2, m) * s(i1, i2) + \sum_{i=1,\, j=1}^{i=N,\, j=M} p(i, j, m) * g(j, i)]$
- $\forall i_1 = 1 \rightarrow N, \forall i_2 = i_1 + 1 \rightarrow N, \forall m = 1 \rightarrow K : E \leq s(i_1, i_2) * [(N - (N - 1) * x(i_1, i_2, m)]$
- $\forall i = 1 \rightarrow N, \forall j = 1 \rightarrow M, \forall m = 1 \rightarrow K : F \leq g(j, i) * [N - (N - 1) * p(i, j, m)]$

lectures, speeches, reports

MIXED INTEGER PROGRAMMING

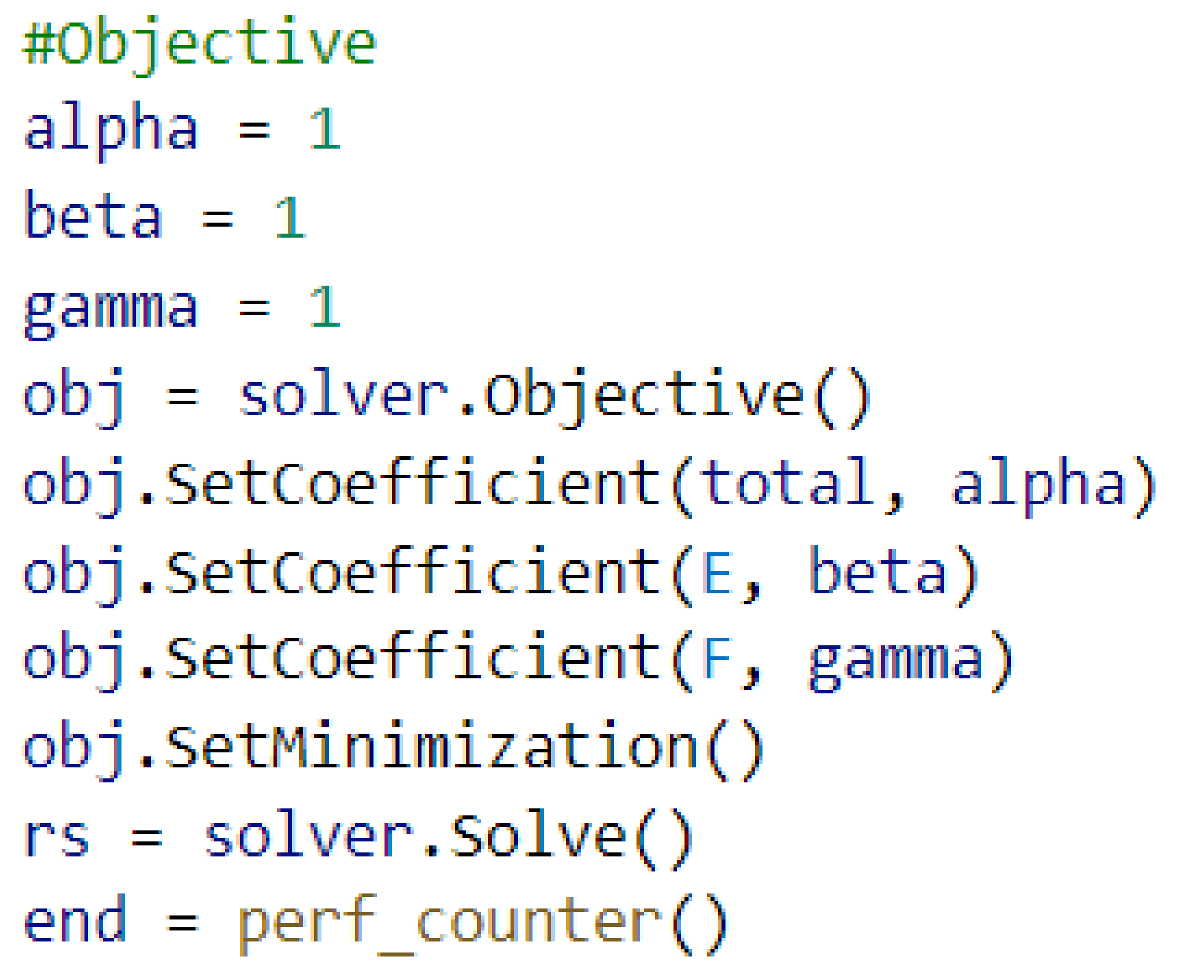


```python
# constraint 10
cstr10 = solver.Constraint(0, 0)
cstr10.SetCoefficient(total, -1)
for m in range(1, K + 1):
    for i1 in range(1, N):
        for i2 in range(i1 + 1, N + 1):
            cstr10.SetCoefficient(x[i1][i2][m], s[i1][i2])

for m in range(1, K + 1):
    for i in range(1, N + 1):
        for j in range(1, M + 1):
            cstr10.SetCoefficient(p[i][j][m], g[j][i])
# constraint 11: s(i1,i2)(N - (N - 1)*x(i1,i2,m)) >= E with i1 != i2
for i1 in range(1, N + 1):
    for i2 in range(1, N + 1):
        for m in range(1, K + 1):
            if i1 != i2:
                cstr11 = solver.Constraint(e, s[i1][i2] * N)
                cstr11.SetCoefficient(x[i1][i2][m], s[i1][i2] * (N - 1))
                cstr11.SetCoefficient(E, 1)

# constraint 12: g(j, i)(N - (N - 1)p(i, j, m)) >= F
for i in range(1, N + 1):
    for j in range(1, M + 1):
        for m in range(1, K + 1):
            cstr12 = solver.Constraint(f, N * g[j][i])
            cstr12.SetCoefficient(p[i][j][m], g[j][i] * (N - 1))
            cstr12.SetCoefficient(F, 1)
```

MIXED INTEGER PROGRAMMING

3.Objective

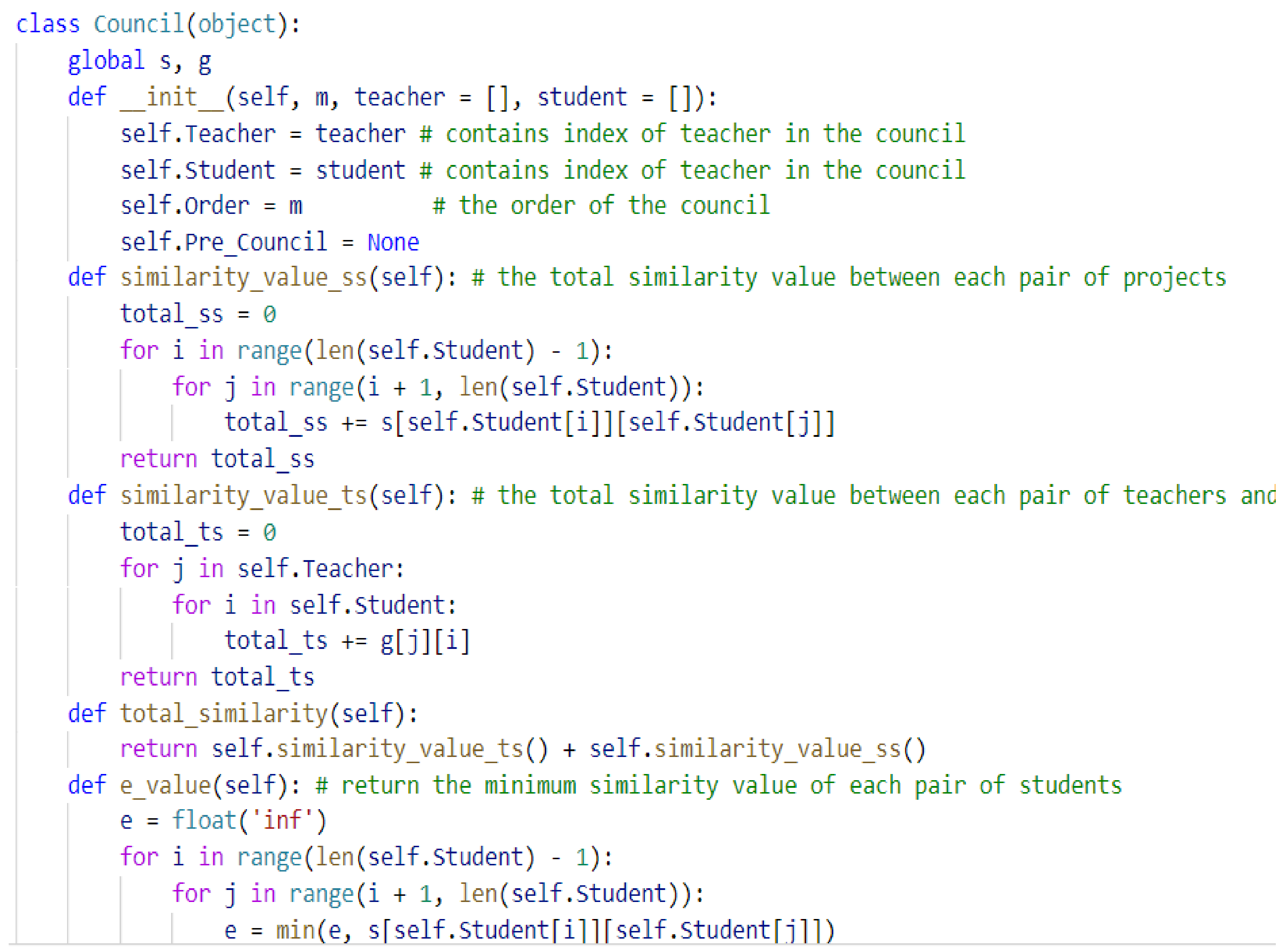
- $f(Total, E, F) = \alpha * E + \beta * F + \gamma * Total \rightarrow$ MAXIMUM



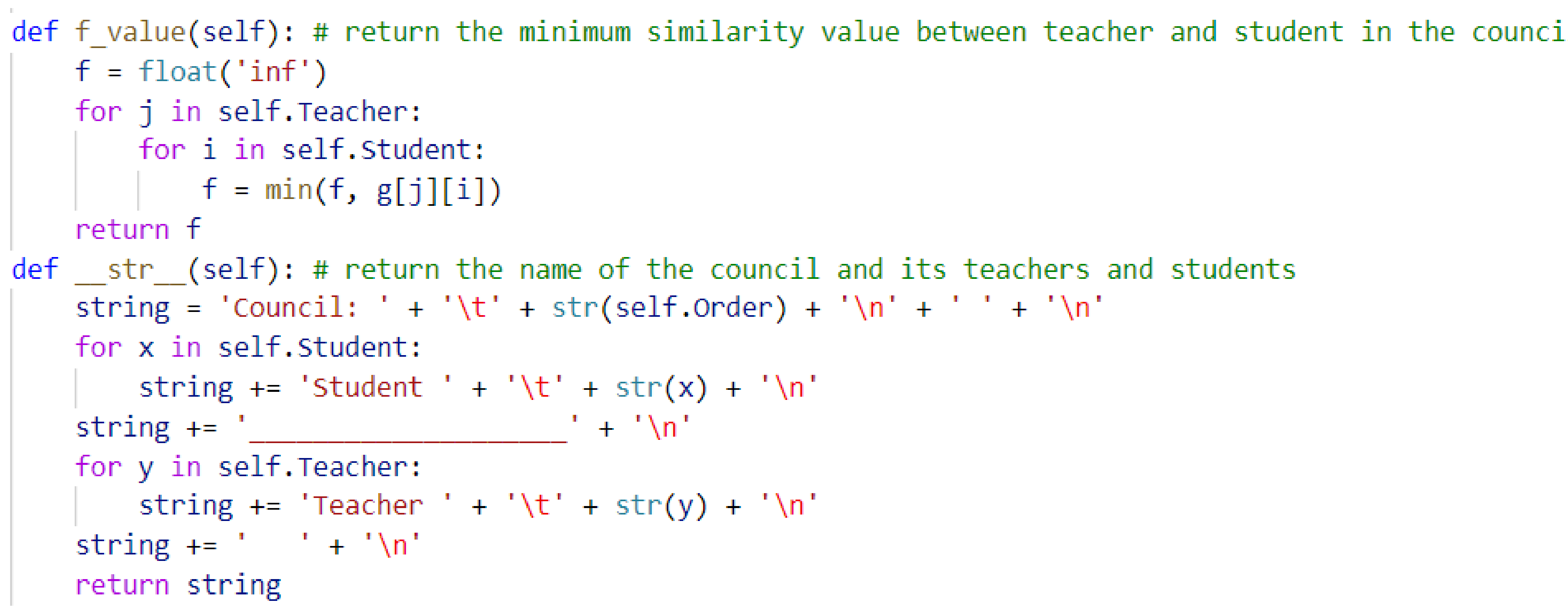
```
#Objective
alpha = 1
beta = 1
gamma = 1
obj = solver.Objective()
obj.SetCoefficient(total, alpha)
obj.SetCoefficient(E, beta)
obj.SetCoefficient(F, gamma)
obj.SetMinimization()
rs = solver.Solve()
end = perf_counter()
```

03

BRUTE FORCE SOLUTION



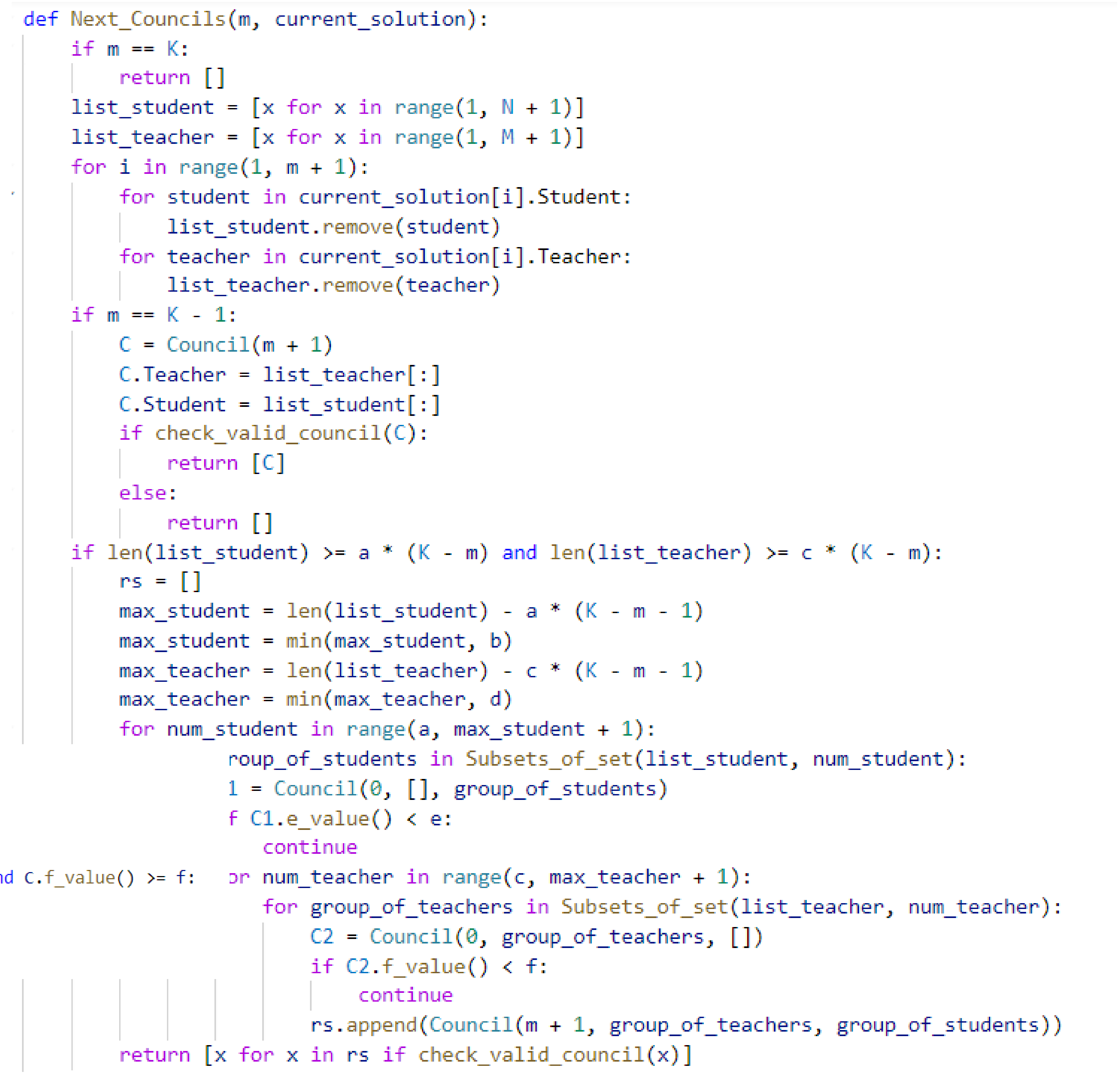
```python
class Council(object):
    global s, g
    def __init__(self, m, teacher = [], student = []):
        self.Teacher = teacher # contains index of teacher in the council
        self.Student = student # contains index of teacher in the council
        self.Order = m         # the order of the council
        self.Pre_Council = None
    def similarity_value_ss(self): # the total similarity value between each pair of projects
        total_ss = 0
        for i in range(len(self.Student) - 1):
            for j in range(i + 1, len(self.Student)):
                total_ss += s[self.Student[i]][self.Student[j]]
        return total_ss
    def similarity_value_ts(self): # the total similarity value between each pair of teachers and
        total_ts = 0
        for j in self.Teacher:
            for i in self.Student:
                total_ts += g[j][i]
        return total_ts
    def total_similarity(self):
        return self.similarity_value_ts() + self.similarity_value_ss()
    def e_value(self): # return the minimum similarity value of each pair of students
        e = float('inf')
        for i in range(len(self.Student) - 1):
            for j in range(i + 1, len(self.Student)):
                e = min(e, s[self.Student[i]][self.Student[j]])
```



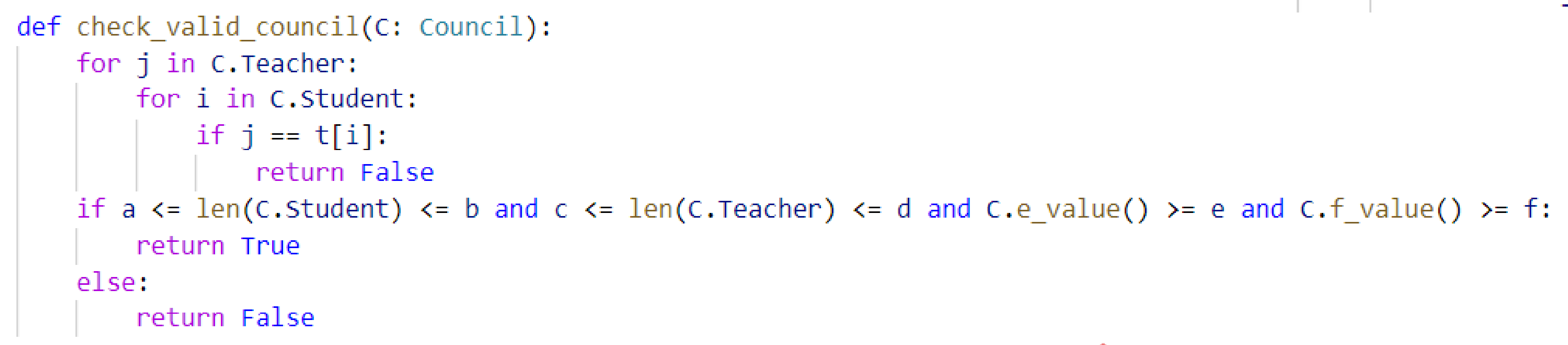
```python
def f_value(self): # return the minimum similarity value between teacher and student in the counci
    f = float('inf')
    for j in self.Teacher:
        for i in self.Student:
            f = min(f, g[j][i])
    return f
def __str__(self): # return the name of the council and its teachers and students
    string = 'Council: ' + '\t' + str(self.Order) + '\n' + ' ' + '\n'
    for x in self.Student:
        string += 'Student ' + '\t' + str(x) + '\n'
    string += '______________________' + '\n'
    for y in self.Teacher:
        string += 'Teacher ' + '\t' + str(y) + '\n'
    string += '    ' + '\n'
    return string
```

03

BRUTE FORCE SOLUTION

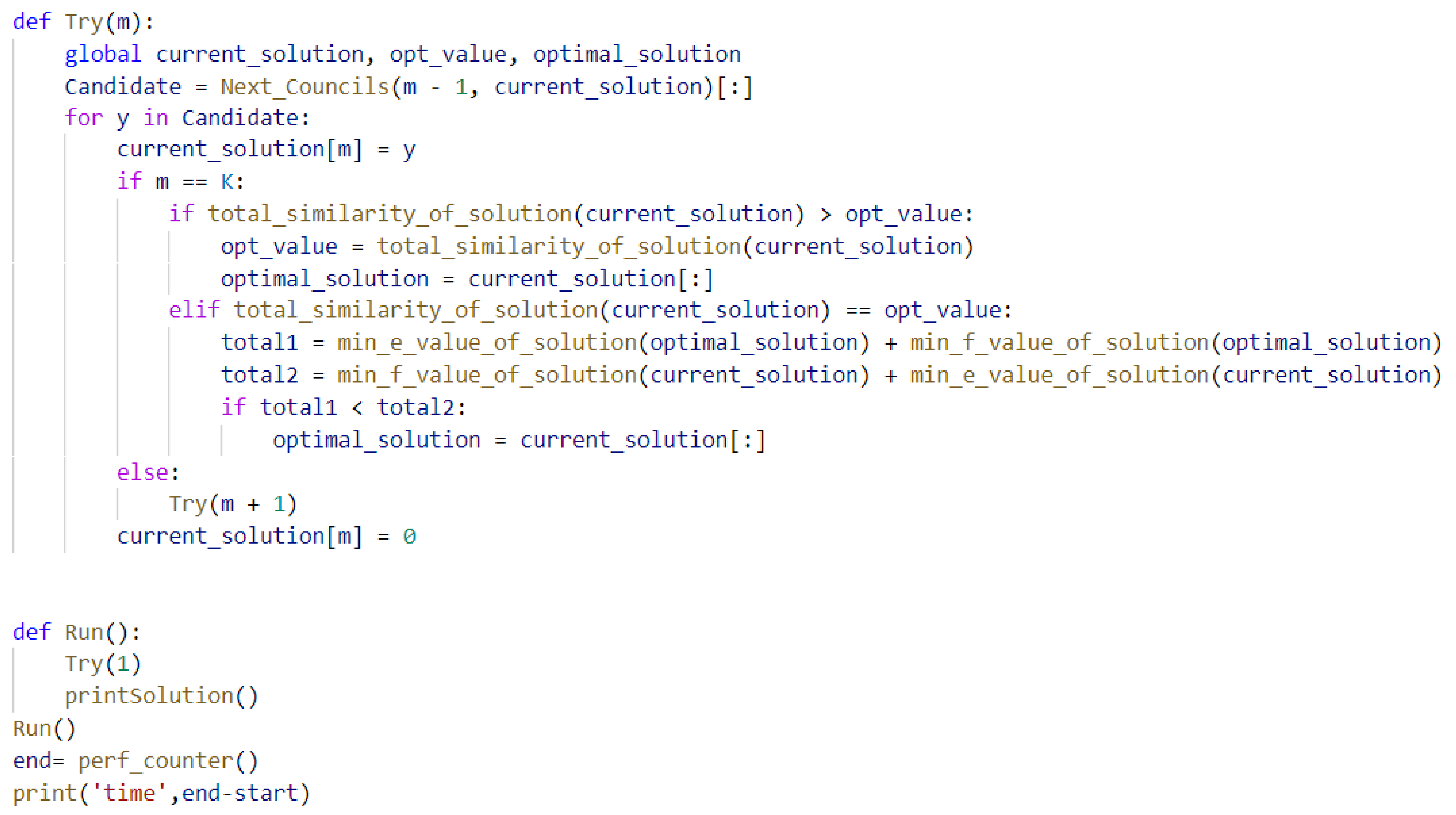


```
def Next_Councils(m, current_solution):
    if m == K:
        return []
    list_student = [x for x in range(1, N + 1)]
    list_teacher = [x for x in range(1, M + 1)]
    for i in range(1, m + 1):
        for student in current_solution[i].Student:
            list_student.remove(student)
        for teacher in current_solution[i].Teacher:
            list_teacher.remove(teacher)
    if m == K - 1:
        C = Council(m + 1)
        C.Teacher = list_teacher[:]
        C.Student = list_student[:]
        if check_valid_council(C):
            return [C]
        else:
            return []
    if len(list_student) >= a * (K - m) and len(list_teacher) >= c * (K - m):
        rs = []
        max_student = len(list_student) - a * (K - m - 1)
        max_student = min(max_student, b)
        max_teacher = len(list_teacher) - c * (K - m - 1)
        max_teacher = min(max_teacher, d)
        for num_student in range(a, max_student + 1):
                 roup_of_students in Subsets_of_set(list_student, num_student):
                 1 = Council(0, [], group_of_students)
                 f C1.e_value() < e:
                    continue
                 or num_teacher in range(c, max_teacher + 1):
                    for group_of_teachers in Subsets_of_set(list_teacher, num_teacher):
                        C2 = Council(0, group_of_teachers, [])
                        if C2.f_value() < f:
                            continue
                        rs.append(Council(m + 1, group_of_teachers, group_of_students))
        return [x for x in rs if check_valid_council(x)]
```



```
def check_valid_council(C: Council):
    for j in C.Teacher:
        for i in C.Student:
            if j == t[i]:
                return False
    if a <= len(C.Student) <= b and c <= len(C.Teacher) <= d and C.e_value() >= e and C.f_value() >= f:
        return True
    else:
        return False
```

BRUTE FORCE SOLUTION

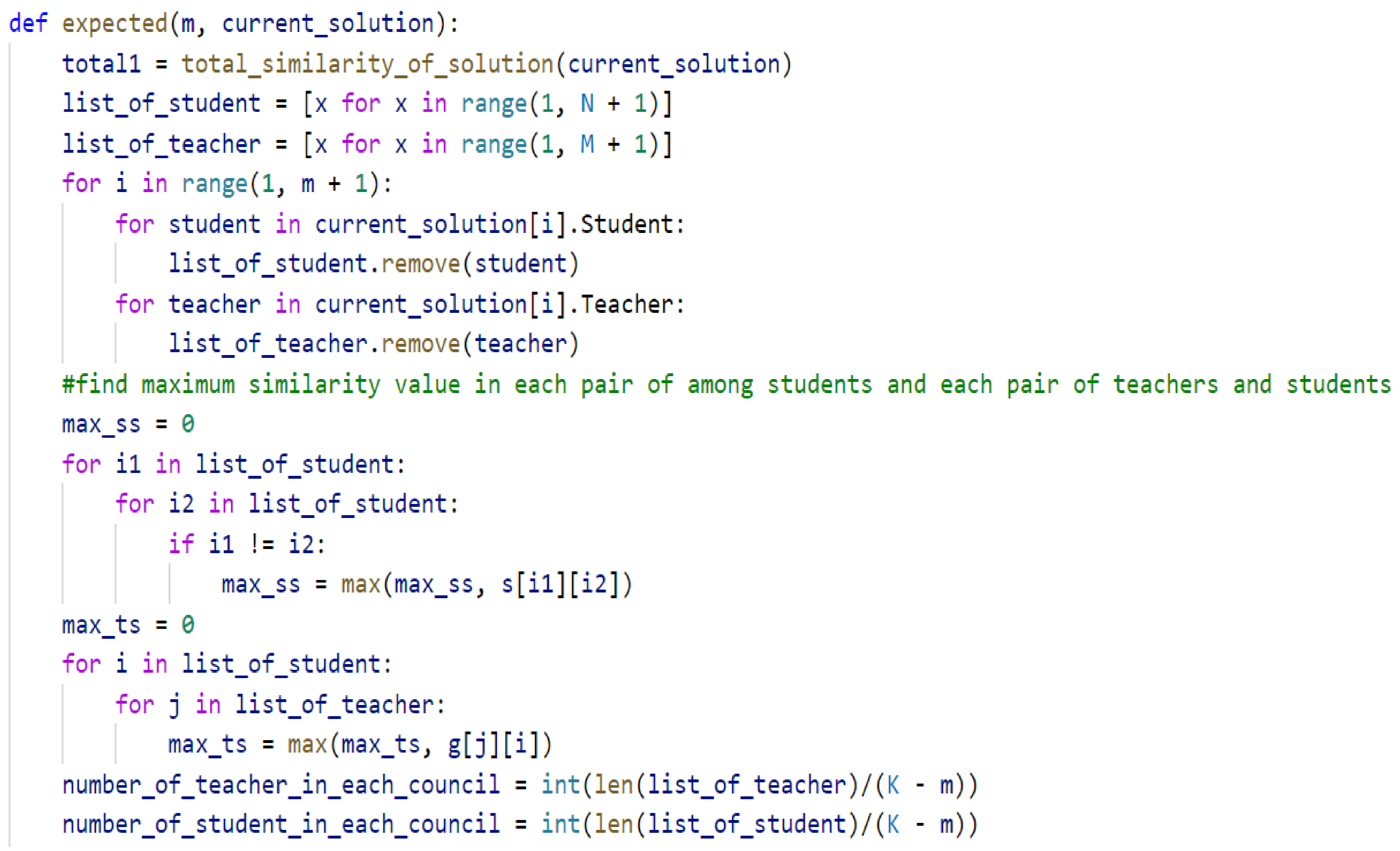


```python
def Try(m):
    global current_solution, opt_value, optimal_solution
    Candidate = Next_Councils(m - 1, current_solution)[:]
    for y in Candidate:
        current_solution[m] = y
        if m == K:
            if total_similarity_of_solution(current_solution) > opt_value:
                opt_value = total_similarity_of_solution(current_solution)
                optimal_solution = current_solution[:]
            elif total_similarity_of_solution(current_solution) == opt_value:
                total1 = min_e_value_of_solution(optimal_solution) + min_f_value_of_solution(optimal_solution)
                total2 = min_f_value_of_solution(current_solution) + min_e_value_of_solution(current_solution)
                if total1 < total2:
                    optimal_solution = current_solution[:]
        else:
            Try(m + 1)
        current_solution[m] = 0


def Run():
    Try(1)
    printSolution()
Run()
end= perf_counter()
print('time',end-start)
```

04

BRANCH AND BOUND

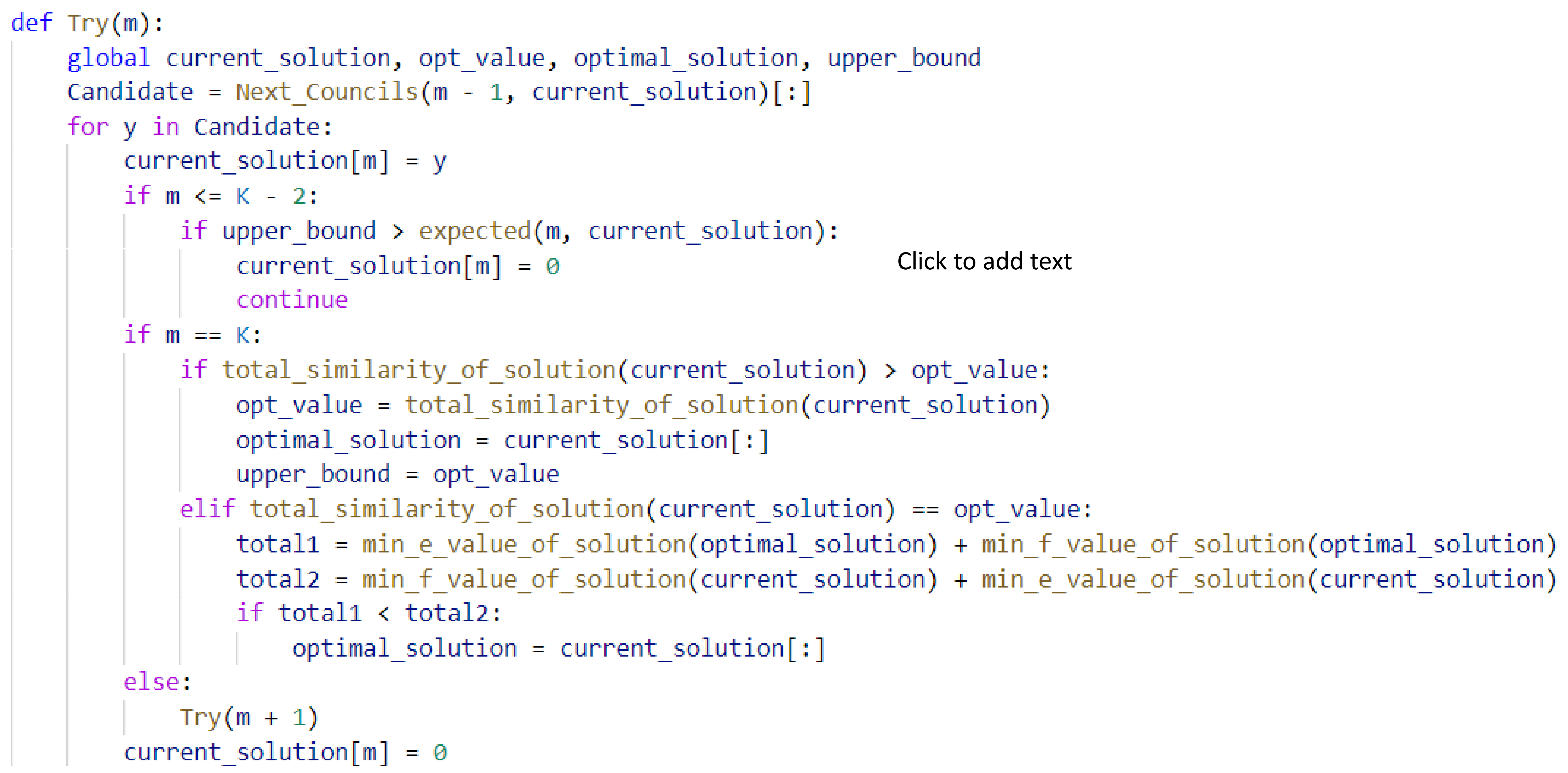
Define: The algorithm build the solution part-by-part and find the optimal solution, we use the heuristic function to estimate the upper bound of the total value of similarity in the current solution which we are building, and remove it if the value is smaller than the value of the best solution we can get until that time.



```python
def expected(m, current_solution):
    total1 = total_similarity_of_solution(current_solution)
    list_of_student = [x for x in range(1, N + 1)]
    list_of_teacher = [x for x in range(1, M + 1)]
    for i in range(1, m + 1):
        for student in current_solution[i].Student:
            list_of_student.remove(student)
        for teacher in current_solution[i].Teacher:
            list_of_teacher.remove(teacher)
    #find maximum similarity value in each pair of among students and each pair of teachers and students
    max_ss = 0
    for i1 in list_of_student:
        for i2 in list_of_student:
            if i1 != i2:
                max_ss = max(max_ss, s[i1][i2])
    max_ts = 0
    for i in list_of_student:
        for j in list_of_teacher:
            max_ts = max(max_ts, g[j][i])
    number_of_teacher_in_each_council = int(len(list_of_teacher)/(K - m))
    number_of_student_in_each_council = int(len(list_of_student)/(K - m))
```

04

BRANCH AND BOUND



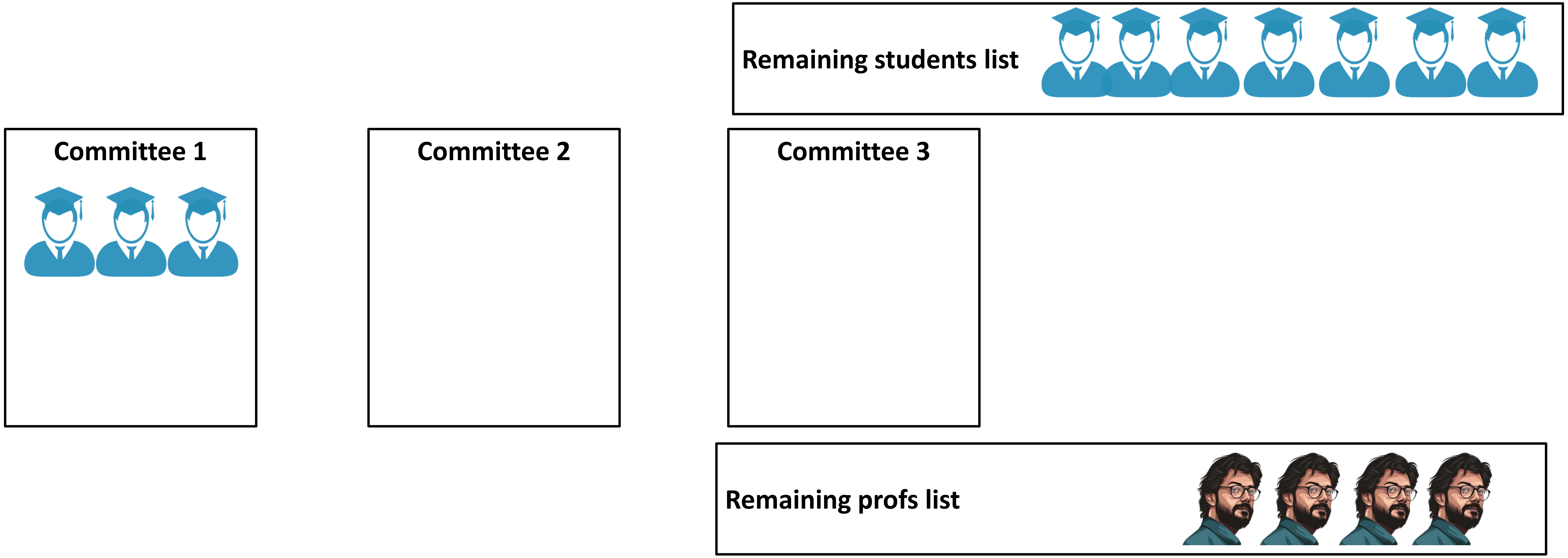
```python
def Try(m):
    global current_solution, opt_value, optimal_solution, upper_bound
    Candidate = Next_Councils(m - 1, current_solution)[:]
    for y in Candidate:
        current_solution[m] = y
        if m <= K - 2:
            if upper_bound > expected(m, current_solution):
                current_solution[m] = 0
                continue
        if m == K:
            if total_similarity_of_solution(current_solution) > opt_value:
                opt_value = total_similarity_of_solution(current_solution)
                optimal_solution = current_solution[:]
                upper_bound = opt_value
            elif total_similarity_of_solution(current_solution) == opt_value:
                total1 = min_e_value_of_solution(optimal_solution) + min_f_value_of_solution(optimal_solution)
                total2 = min_f_value_of_solution(current_solution) + min_e_value_of_solution(current_solution)
                if total1 < total2:
                    optimal_solution = current_solution[:]
        else:
            Try(m + 1)
        current_solution[m] = 0
```

Click to add text

05

GREEDY METHODS

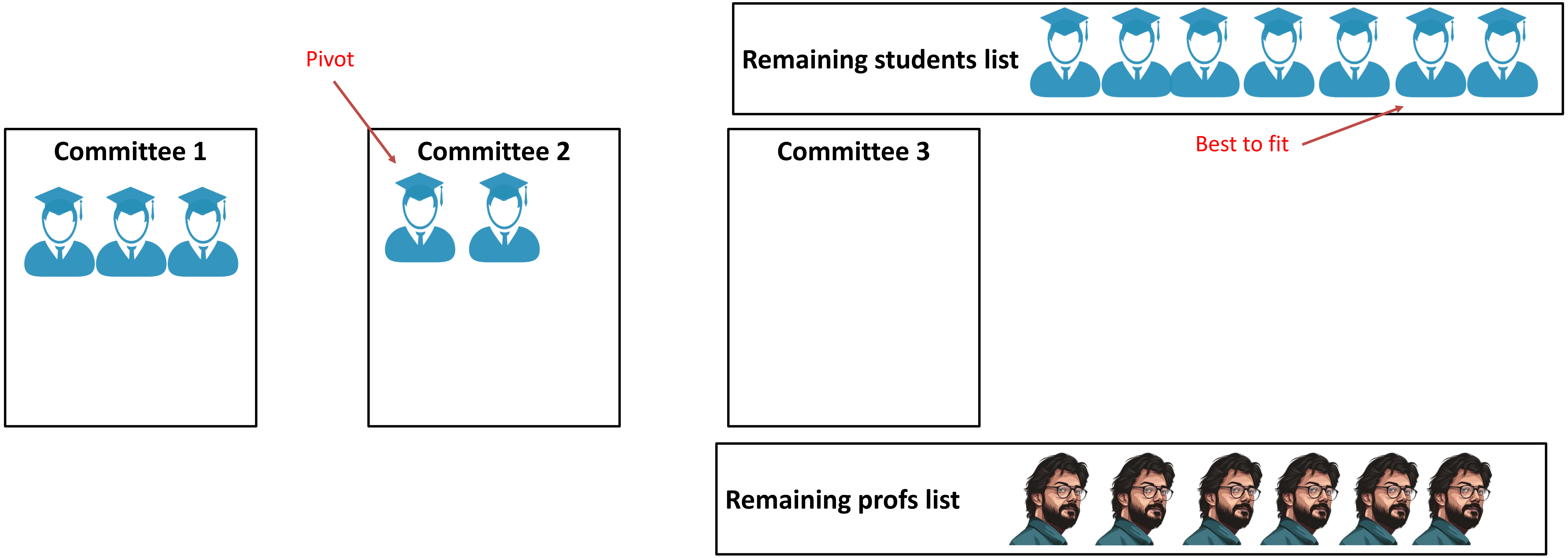
Our idea: If committee k already has enough students (a), take out the first student from the student list to committee k+1



05

GREEDY METHODS

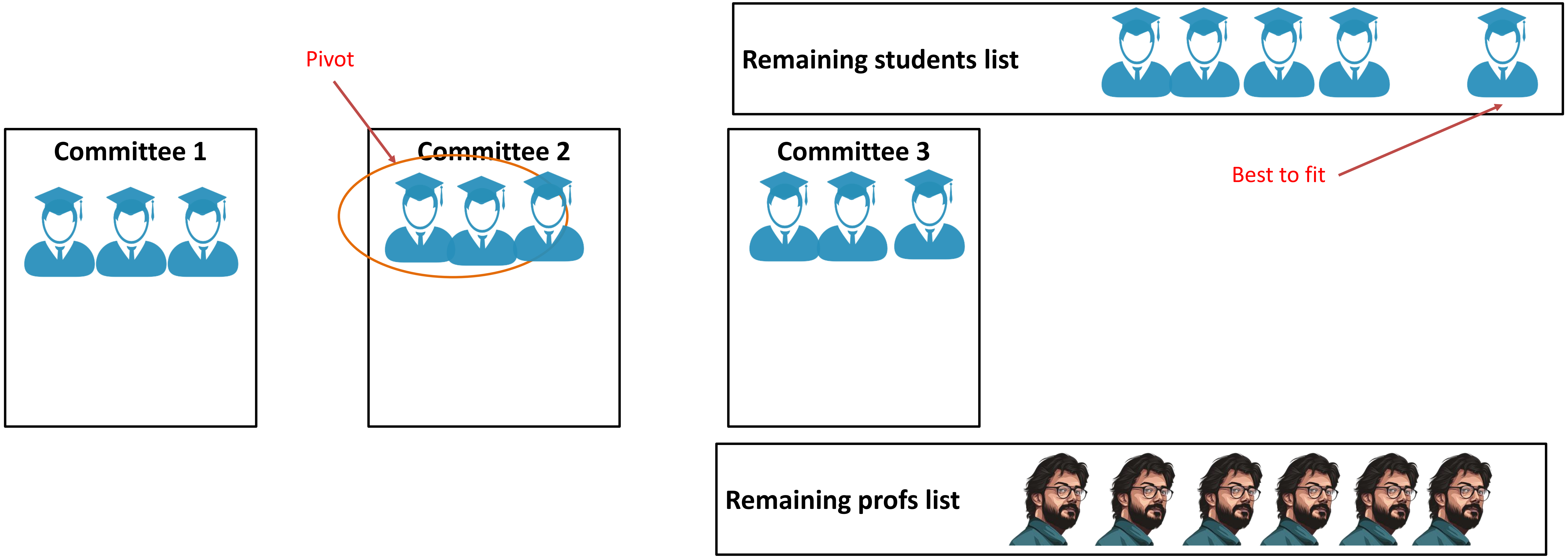
Our idea: He then will be pivot, the student whose has highest similarity index with him will join the same committee



05

GREEDY METHODS

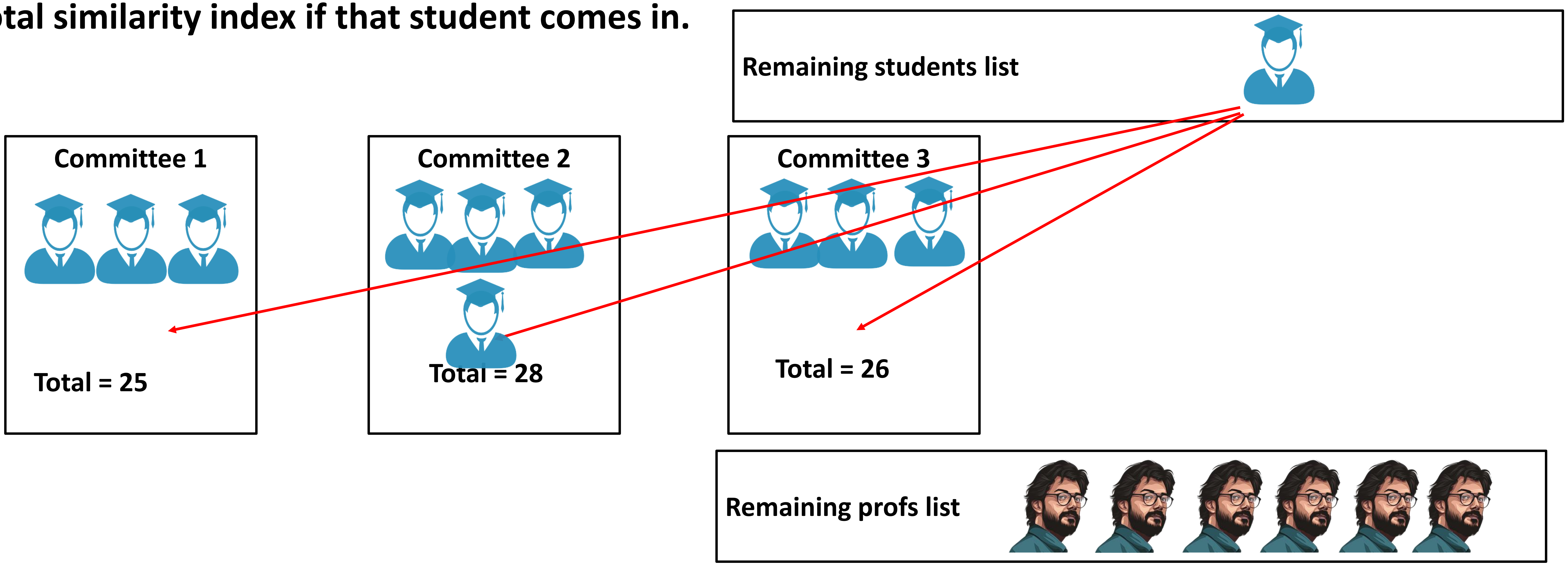
Our idea: Then both will be pivot (which means we will calculate the sum of similarity indices), and the process will continue until every committee has "enough" student



05

GREEDY METHODS

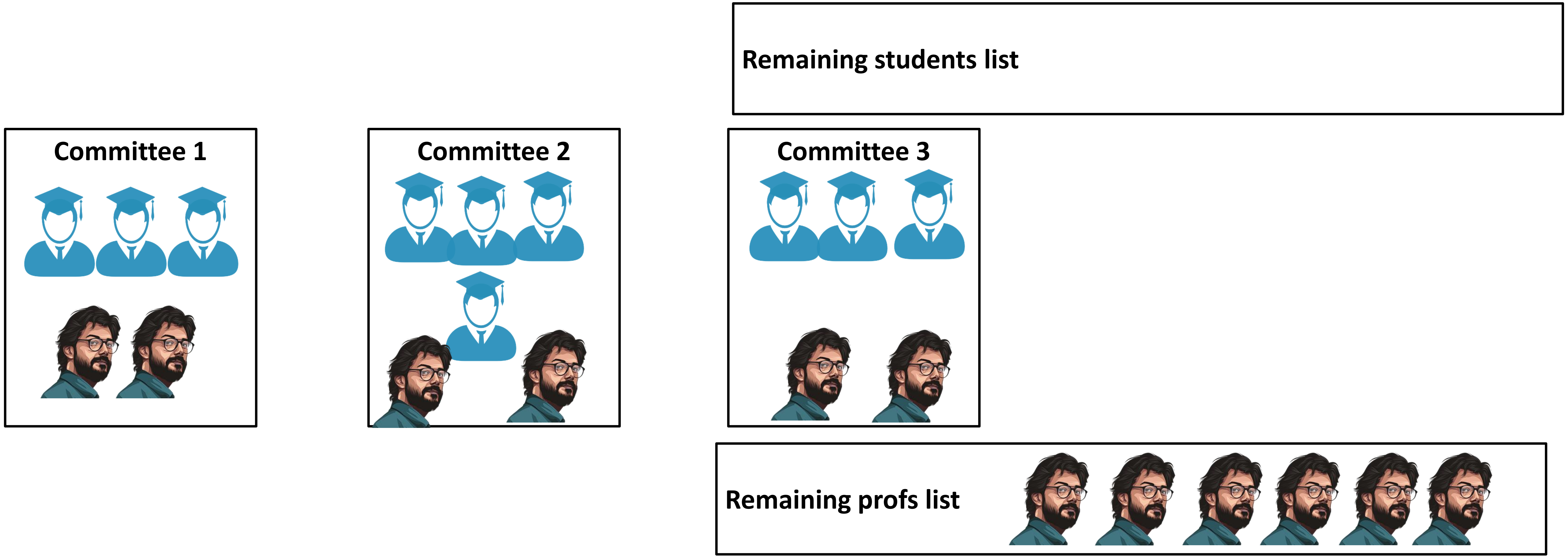
Our idea: In the end, there may be some students to remain because each committee has between a and b students, but we only drafted (a) students to each committee. So here we would calculate total similarity index if that student comes in.



05

GREEDY METHODS

Our idea: Same applies to professors

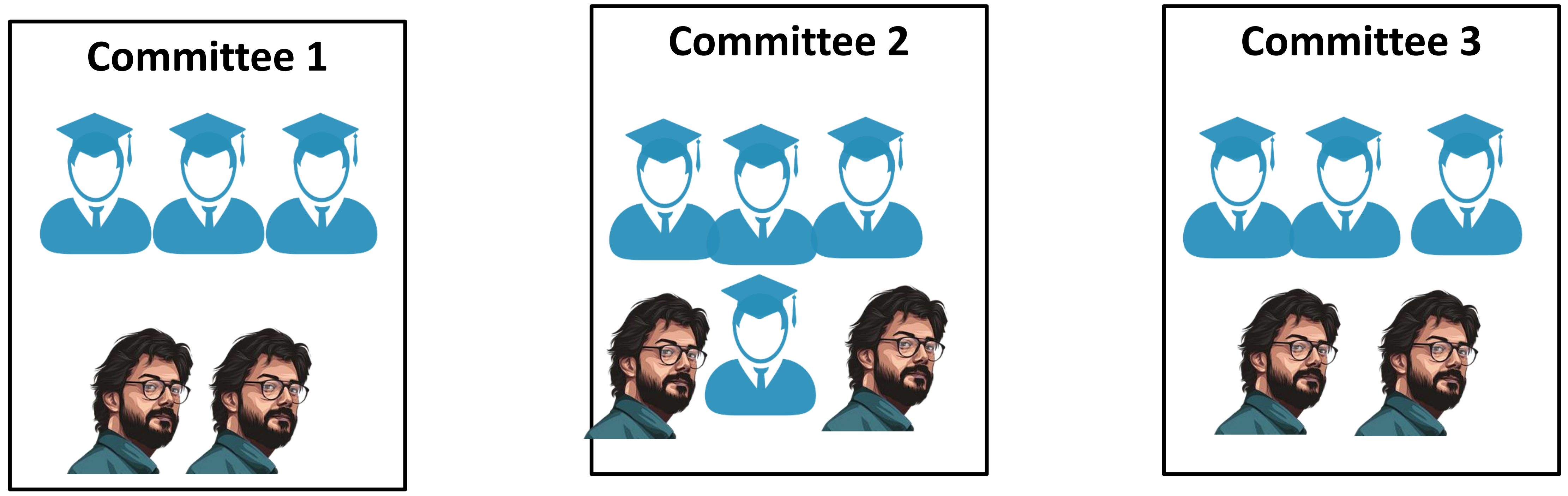


05

GREEDY METHODS

Some drawbacks of greedy methods:

- Non-optimal solutions: Hard to avoid, for example, student 1 as a pivot matches student 2 better than student 3. However, when finish arranging, student 3's thesis actually fits other students in that committee more than student 2. (but it's too late)



- Constraint violation (instructor-student constraint): Can be fixed by setting g[instructor][student] to –INF or by choosing professors before choosing remaining students

6.ANALYSIS

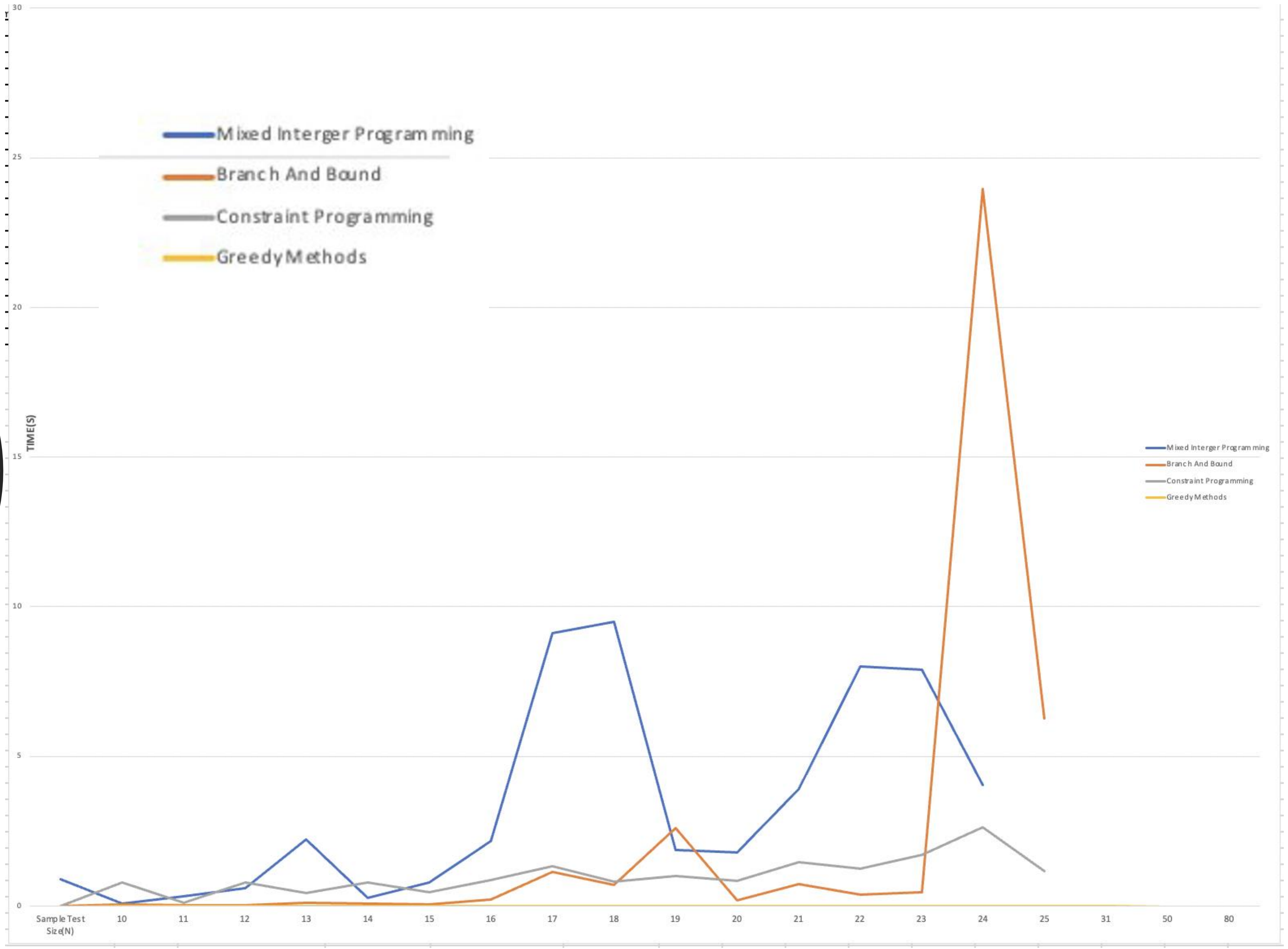
TABLE OF OPTIMAL VALUE

Sample Test	Sample Test Size	Brute Force	MIP	Branch And Bound	Constraint Programming	Greedy Method
D10(N=10,M=5,K=3)	10	TOTAL =205,E=7,F=4	TOTAL =205,E=7,F=4	TOTAL =205,E=7,F=4	TOTAL =205,E=7,F=4	TOTAL = 198
D11(N=11,M=5,K=2)	11	TOTAL =388,E=7,F=4	TOTAL =388,E=7,F=4	TOTAL =388,E=7,F=4	TOTAL =388,E=7,F=4	TOTAL =332
D12(N=12,M=7,K=4)	12	TOTAL =220,E=7,F=4	TOTAL =220,E=7,F=4	TOTAL =220,E=7,F=4	TOTAL =220,E=7,F=4	TOTAL =200
D13(N=13,M=8,K=3)	13	TOTAL = 271,E=7,F=4	TOTAL = 271,E=7,F=4	TOTAL = 271,E=7,F=4	TOTAL = 271,E=7,F=4	TOTAL =252
D14(N=14,M=8,K=4)	14	TOTAL =405,E=7,F=7	TOTAL =405,E=7,F=7	TOTAL =405,E=7,F=7	TOTAL =405,E=7,F=7	TOTAL =336
D15(N=15,M=8,K=3)	15	TOTAL = 485,E=7,F=7	TOTAL = 485,E=7,F=7	TOTAL = 485,E=7,F=7	TOTAL = 485,E=7,F=7	TOTAL =485
D16(N=16,M=10,K=4)	16	TOTAL = 471,E=7,F=4	TOTAL = 471,E=7,F=4	TOTAL = 471,E=7,F=4	TOTAL = 471,E=7,F=4	TOTAL =397
D17(N=17,M=9,K=4)	17	TOTAL = 467,E=7,F=4	TOTAL = 467,E=7,F=4	TOTAL = 467,E=7,F=4	TOTAL = 467,E=7,F=4	TOTAL =474
D18(N=18,M=9,K=4)	18	TOTAL =502,E=7,F=4	TOTAL =502,E=7,F=4	TOTAL =502,E=7,F=4	TOTAL =502,E=7,F=4	TOTAL =416
D19(N=19,M=12,K=4)	19	TOTAL =639,E=7,F=4	TOTAL =639,E=7,F=4	TOTAL =639,E=7,F=4	TOTAL =639,E=7,F=4	TOTAL =576
D20(N=20,M=12,K=5)	20	TOTAL =620,E=7,F=7	TOTAL =620,E=7,F=7	TOTAL =620,E=7,F=7	TOTAL =620,E=7,F=7	TOTAL =620
D21(N=21,M=10,K=5)	21	TOTAL = 607,E=7,F=7	TOTAL = 607,E=7,F=7	TOTAL = 607,E=7,F=7	TOTAL = 607,E=7,F=7	TOTAL =607
D22(N=22,M=12,K=5)	22	TOTAL =724,E=7,F=7	TOTAL =724,E=7,F=7	TOTAL =724,E=7,F=7	TOTAL =724,E=7,F=7	TOTAL = 724
D23(N=23,M=9,K=5)	23		TOTAL = 658,E=7,F=7	TOTAL = 658,E=7,F=7	TOTAL = 658,E=7,F=7	TOTAL =488
D24(N=24,M=9,K=5)	24		TOTAL =814,E=7,F=7	TOTAL =814,E=7,F=7	TOTAL =814,E=7,F=7	TOTAL =689
D25(N=25,M=14,K=5)	25		TOTAL = 967,E=7,F =7	TOTAL = 967,E=7,F =7	TOTAL = 967,E=7,F =7	TOTAL =967
D31(N=31,M=12,K=4)	31					TOTAL =1494
D50(N=50,M=18,K=5)	50					TOTAL =2963
D80(N=80,M=30.K=7)	80					TOTAL =5733
D100(N=100,M=35,K=11)	100					TOTAL = 5781

RUNTIME REPORT OF SAMPLE TESTCASES

Sample Test	Sample Test Size(N)	Brute Force	Mixed Interger Programming	Branch And Bound	Constraint Programming	Greedy Method
D10(N=10,M=5,K =3)	10	0,195	0,897704	0,0505734	0,78587	0,0008
D11(N=11,M=5,K=2)	11	0,0294	0,069161	0,02854309	0,09922	0,0004
D12(N=12,M=7,K=4)	12	0,1211	0,310710125	0,033995	0,77321	0,0004
D13(N=13,M=8,K=3)	13	1,4716	0,583161667	0,0925165	0,42903	0,00042
D14(N=14,M=8,K=4)	14	1,0604	2,219439875	0,0769189	0,77026	0,00045
D15(N=15,M=8,K=3)	15	3,1822	0,273431583	0,0401008	0,46396	0,00051
D16(N=16,M=10,K=4)	16	59,1363	0,771139458	0,2005659	0,86417	0,00057
D17(N=17,M=9,K=4)	17	206,723	2,171338875	1,1430482	1,3299	0,00046
D18(N=18,M=10,K=4)	18	76,0089	9,121349167	0,700819	0,81662	0,0006
D19(N=19,M=12,K=4)	19	25,9109	9,496691166	2,6023697	0,99325	0,00042
D20(N=20,M=12,K=5)	20	18,0486	1,856543875	0,174322	0,82775	0,0006
D21(N=21,M=10,K=5)	21	966,0521	1,786952083	0,7240319	1,46979	0,00038
D22(N=22,M=12,K=5)	22		3,896060875	0,3806014	1,23661	0,00051
D23(N=23,M=9,K=5)	23		7,994489542	0,4604143	1,69188	0,00042
D24(N=24,M=9,K=5)	24		7,878824584	23,9466507	2,63832	0,00036
D25(N=25,M=14,K=5)	25		4,034628375	6,260473	1,16723	0,00064
D31(N=31,M=12,K=4)	31					0,0005
D50(N=50,M=18,K=5)	50					0.0084
D80(N=80,M=30,K=7)	80					0.0134
D100(N=100,M=35,K=11)	100					0,0164

RUNTIME COMPARISON



CONCLUSION

- CP is better than MIP for this problem

- Branch and bound can be the best algorithm for several instances, but it is unstable (for example D24)

- With large input size, Greedy Method is the most suitable algorithm, however there are some drawbacks

Thanks For Your Attention !