```
import gurobipy as gp
from gurobipy import GRB
import re

def readinfiles(mpsfilename, constrfilename):
    """
    This function takes the mps file to extract the information about the problem
    and uses the file containing information about hard constraints to slice the
constraints
    :param mpsfilename: the name of the mps file
    :param constrfilename: the file about the hard constraints
    :return: coeffmatrix, rhside: all constraints and rhs values of the mps
             A1, b1: the hard constraints and rhs values
             A2, b2: the soft constraints and rhs values
             objcoeff: the coefficient for variables from the objective function
             lpr.objVal: objective value from Linear relaxed program
             -1*mipexact.ObjVal: objective value by solving exactly
             directopt: objective value by solving the mps file exactly
    """
    m = gp.read(mpsfilename)
    m.Params.LogToConsole = 0
    m.optimize()
    directopt = m.objVal

    # convert the constraint information from mps file into matrix type
    num_rows = m.numconstrs
    num_col = m.numvars
    coeffmatrix = [None] * num_rows

    for k in range(num_rows):
        constrslist = [None] * num_col
        for i in range(num_col):
            constrslist[i] = m.getCoeff(m.getConstrs()[k], m.getVars()[i])
        coeffmatrix[k] = constrslist

    rhside = m.rhs  # right hand side values of the constraints
    objcoeff = m.obj  # coefficient in the objective function

    # print(objcoeff)
    # print(coeffmatrix[0])
    # print(rhside)

    # Now solve the Mixed-integer Program exactly, and check if the output is the
same as directly running the mps file
    mipexact = gp.Model()
    mipexact.Params.LogToConsole = 0
    x = []
    for i in range(len(objcoeff)):
        x.append(mipexact.addVar(vtype=GRB.BINARY))

    for j in range(len(rhside)):
        mipexact.addConstr(sum(coeffmatrix[j][i] * x[i] for i in range(len(x))) <=
rhside[j])

    mipexact.setObjective(-1*sum(objcoeff[m] * x[m] for m in range(len(x))),
GRB.MAXIMIZE)
    mipexact.optimize()

    print("Successfully draw information from the mps file!")
```

```
    #print("For mixed-integer program, optimal solution is: ", mipexact.X)
    # Next solve the relaxed version
    lpr = gp.Model()
    lpr.Params.LogToConsole = 0
    x = []
    for i in range(len(objcoeff)):
        x.append(lpr.addVar(vtype=GRB.CONTINUOUS))

    for j in range(len(rhside)):
        lpr.addConstr(sum(coeffmatrix[j][i] * x[i] for i in range(len(x))) <=
rhside[j])

    lpr.setObjective(sum(objcoeff[m] * x[m] for m in range(len(x))))
    lpr.optimize()

    #print("For LP relaxation, optimal solution is: ", lpr.X)

    # Suppose we have a file that indicates which constraints are hard
    hardconstridx = []
    with open(constrfilename, 'r') as file:
        # Loop over each line in the file
        for line in file:
            # Use regular expressions to find the numbers in the line
            numbers = re.findall(r'\d+', line)
            hardconstridx.append(int(numbers[0]))

    # separate the constraints into hard ones and soft ones according to the
information above
    A1 = coeffmatrix[hardconstridx[0]:hardconstridx[-1] + 1]
    b1 = rhside[hardconstridx[0]:hardconstridx[-1] + 1]
    A2 = coeffmatrix[hardconstridx[-1] + 1:]
    b2 = rhside[hardconstridx[-1] + 1:]

    return coeffmatrix, rhside, A1, b1, A2, b2, objcoeff, lpr.objVal ,
mipexact.ObjVal, directopt

# Extract information from the mps file
[cc, bb, coef1, rhs1, coef2, rhs2, cfobj1, lpz, iz1, iz2] = readinfiles('lseu.mps',
'hardconstraints.txt')

cfobj = []
for value in cfobj1:
    cfobj.append(-value)

# Phase I to find initial feasible solution

phi = gp.Model()
phi.Params.LogToConsole = 0
x = []
for i in range(len(coef1[0])):
    x.append(phi.addVar(vtype=GRB.CONTINUOUS))

y = []
for j in range(len(cc)):
    y.append(phi.addVar(vtype=GRB.CONTINUOUS))

iddmat = []

for i in range(len(cc)):
```

```
    row = [0] * len(cc)  # create a row of zeros
    row[i] = 1  # set the diagonal element to 1
    iddmat.append(row)  # append the row to the result list

for j in range(len(bb)):
    phi.addConstr(sum(cc[j][i] * x[i] for i in range(len(x))) + sum(iddmat[j][l] *
y[l] for l in range(len(y))) <= bb[j])

phi.setObjective(sum(99999999 * y[m] for m in range(len(y))), GRB.MINIMIZE)
phi.optimize()

#print(phi.X)
initial_v = phi.X[:len(x)]
# print(initial_v) # first extreme points we found

# first RMP we solve

extrpoint = []
extrpoint.append(initial_v)
extrrays = []

def RMPsolve(extremepoints, extremerays):
    cv = []
    for m in range(len(extremepoints)):
        cv.append(sum(cfobj[k] * extremepoints[m][k] for k in range(len(cc))))
    print("cv is ", cv)

    rmp = gp.Model()
    rmp.Params.LogToConsole = 0
    lbd = []
    for i in range(len(extremepoints)):
        lbd.append(rmp.addVar(vtype=GRB.CONTINUOUS))
    # add extreme points
    av = []
    for point in extremepoints:
        dot_product_1 = sum([a * b for a, b in zip(point, coef1[0])])
        dot_product_2 = sum([a * b for a, b in zip(point, coef1[1])])
        av.append([dot_product_1, dot_product_2])
    print("av is ", av)

    ch = []
    for m in range(len(extremerays)):
        ch.append(sum(cfobj[k] * extremerays[m][k] for k in range(len(cc))))
    print("ch is ", ch)

    miu = []
    for i in range(len(extremerays)):
        miu.append(rmp.addVar(vtype=GRB.CONTINUOUS))
    # add extreme rays
    ar = []
    for point in extremerays:
        dot_product_1 = sum([a * b for a, b in zip(point, coef1[0])])
        dot_product_2 = sum([a * b for a, b in zip(point, coef1[1])])
        ar.append([dot_product_1, dot_product_2])
    print("ar is ", ar)

    for h in range(len(rhs1)):
        rmp.addConstr(sum(av[p][h] * lbd[p] for p in range(len(av))) + sum(ar[p][h]
* miu[p] for p in range(len(ar))) <= rhs1[h])
```

```
    rmp.addConstr(sum(1 * lbd[j] for j in range(len(extremepoints))) == 1)

    rmp.setObjective(sum(cv[m] * lbd[m] for m in range(len(lbd)))+sum(ch[m] *
miu[m] for m in range(len(miu))), GRB.MAXIMIZE)
    rmp.optimize()
    [pi1, pi2, sigma] = rmp.pi  # get pi and sigma for its dual
    print("pi is ", pi1)
    print("another pi is ", pi2)
    print("Current sigma is ", sigma)
    return cv, ch, av, ar, pi1, pi2, sigma, rmp.X, rmp.objVal

[cv, ch, av, ar, pi1, pi2, sigma, rmplbd, rmpobjval] = RMPsolve(extrpoint,extrrays)


def pricingsolver(pione, pitwo, sigma1):
    pia = []

    for i in range(len(cfobj)):
        pia.append(pione * coef1[0][i] + pitwo * coef1[1][i])

    pricingcoeff = []
    for k in range(len(cfobj)):
        pricingcoeff.append(cfobj[k] - pia[k])

    pric = gp.Model()
    pric.Params.LogToConsole = 0
    x = []
    for i in range(len(pricingcoeff)):
        x.append(pric.addVar(vtype=GRB.BINARY))

    for j in range(len(rhs2)):
        pric.addConstr(sum(coef2[j][i] * x[i] for i in range(len(x))) <= rhs2[j])

    pric.setObjective(sum(pricingcoeff[m] * x[m] for m in range(len(x))),
GRB.MAXIMIZE)
    pric.optimize()

    reducedcost = pric.objVal - sigma1

    return reducedcost, pric.X

[redcost, newv] = pricingsolver(pi1, pi2, sigma)
print("New reduced cost is ", redcost)

iter = 0
while iter < 1000:
    if redcost <= 0:
        print("Find optimal solution!")
        print("Optimal lambda and miu are ", rmplbd)
        print("Optimal objval is: ", rmpobjval)
        print("Convert it back to minimization problem, objval is: ", -rmpobjval)
        break
    elif redcost > 0:
        print("Need to resolve with new lambda!")
        print("Current objval is: ", rmpobjval)
        print("New extreme point is ", newv)
```

```
        extrpoint.append(newv)
        [cv, ch, av, ar, pi1, pi2, sigma, rmplbd, rmpobjval] = RMPsolve(extrpoint, 
extrrays)
        [redcost, newv] = pricingsolver(pi1, pi2, sigma)
        print("New reduced cost is ", redcost)

    iter += 1

print("The final optimal solution is ", -rmpobjval)
print("Recall that zLP = 662.974,  Z_MIP = 1120, zLD = 1014, which is within some 
levels of tolerance.")
```