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

start = time.time()

coeffmatrix = [[-1,2],[5,1],[-2,-2],[-1,0],[0,1]]
rhside = [4,20,-7,-2,4]
objcoeff = [7,2]

# Suppose we have a file that indicates which constraints are hard
hardconstridx = [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:]
objc = objcoeff

# Next solve the Lagrangian dual problem
ldp = gp.Model()
ldp.Params.LogToConsole = 0
x = []
for i in range(len(objc)):
    x.append(ldp.addVar(vtype=GRB.INTEGER))

# Add the soft constraints
for j in range(len(b2)):
    ldp.addConstr(sum(A2[j][i] * x[i] for i in range(len(x))) <= b2[j])

# # Initialize the penalty/s
# s = [ldp.addVar() for i in range(len(bb1))]

# # Express hard constraints as penalty
# for m in range(len(bb1)):
#     ldp.addConstr(bb1[m] - sum(Aa1[m][i] * x[i] for i in range(len(x))) == s[m])

# Initialize the multiplier
currlmbd = [0.5] * len(b1)

ldp.setObjective((sum(objc[m] * x[m] for m in range(len(x)))) +
                 sum(currlmbd[i] * b1[i] for i in range(len(b1))) -
                 sum(currlmbd[i] * A1[i][j] * x[j] for j in range(len(x)) for i in
range(len(currlmbd))), GRB.MAXIMIZE)

# ldp.setObjective((sum(objc[m] * x[m] for m in range(len(x)))) -
#                  sum(currlmbd[n] * s[n] for n in range(len(currlmbd))))

ldp.optimize()
iter = 1
print("Iteration ", iter)
print("Obj value is: ", ldp.objVal)
currg = ldp.objVal

for ss in range(50000):
    tmp_x = ldp.X
    print("Now optimal x is: ",tmp_x)
    s_star = [None] * len(b1)
    for k in range(len(s_star)):
        s_star[k] = b1[k] - sum(A1[k][i] * tmp_x[i] for i in range(len(tmp_x)))

    print("lambda is: ", currlmbd)
    #print((s[i].x for i in range(len(s))))
    print("s_star is: ", s_star)

    if all(elem == 0 for elem in s_star):
        print("Optimal lambda is: ", currlmbd)
        break

    else:
        if iter != 1:
            a = 1 / (iter*math.log(iter))
        elif iter == 1:
            a = 1

        nextlmbd = [currlmbd[p] - a*s_star[p] for p in range(len(currlmbd))]

        ldp.setObjective((sum(objc[m] * x[m] for m in range(len(x)))) +
                         sum(nextlmbd[i] * b1[i] for i in range(len(b1))) -
                         sum(nextlmbd[i] * A1[i][j] * x[j] for j in range(len(x))
for i in range(len(nextlmbd))))
        # ldp.setObjective((sum(objc[m] * x[m] for m in range(len(x)))) -
        #                  sum(nextlmbd[n] * s[n] for n in range(len(nextlmbd))))

        ldp.optimize()
        newg = ldp.objVal

        if newg <= currg:
            currlmbd = nextlmbd
            currg = newg
        elif newg > currg:
            currg = currg
            currlmbd = currlmbd


        iter = iter + 1

        print("Iteration ", iter)
        print("Obj value is: ", currg)



print("For Lagrangian Relaxation:")
print("Optimal x is: ", ldp.X)
print("Optimal s is: ", s_star)
print("Optimal value is: ", currg)
print("Optimal lambda is: ", currlmbd)

mipexact = gp.Model()
mipexact.Params.LogToConsole = 0
x = []
for i in range(len(objcoeff)):
    x.append(mipexact.addVar(vtype=GRB.INTEGER))
```

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

mipexact.setObjective(sum(objcoeff[m] * x[m] for m in range(len(x))),GRB.MAXIMIZE)
mipexact.optimize()
print("For MIP, optimal solution is: ", mipexact.X)
print("For MIP, optimal objval is: ", mipexact.ObjVal)


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))),GRB.MAXIMIZE)
lpr.optimize()
print("For LP relaxation, optimal solution is: ", lpr.X)
print("For LP relaxation, optimal objval is: ", lpr.ObjVal)

print("We can observe that zI <= zLD <= zLP")

end = time.time()
print("running time is", end - start)
```