### CaseStudyCplex.py

## Program running the bi-objective model for

"Designing Spatially Cohesive Nature Reserves With Backup Coverage (IJGIS)

## Authors: Delmelle, Desjardins, Deng

## Inputs: Study area, distance between parcels/patches

weareGrass_Area.txt: information on each patch, including ID, location and area

weareGrass_weight.txt: information of biodiversity for each patch

weareGrass_Dist.txt: distance between each patch, using vertices

## Outputs: LP file

## Contact: delmelle@gmail.com

---

```python
# import modules
import random, string, os, math, time

# define distance function
def distance(x1, y1, x2, y2):
dx = x2 - x1
dy = y2 - y1
d = math.sqrt(dx**2 + dy**2)
return d

#### INPUTS ####

# extent of study area; for predefined grid
left = 947994
right = 991752
bottom = 187782
top = 237660

alpha = .95 # weighting scheme; equation 1
K=3 # number of clusters (K); equation 6
interval=20000 # discretization level
S = 10000 # separation distance; equation 7
M=50000000 # a very large number
A=.5 # percentage (ratio) of total area to set aside (A); equation 9
F=.16 # minimum percentage (ratio) of total area to set aside (F - note that F * K cannot exceed A); equation 8

#### OUTPUT ####

outputFileName = "C:\temp\casestudy_weareAlpha" + str(alpha) + ".lp" # naming convention: concatenate alpha value.
outputFile = open(outputFileName,"w")

#### INFORMATION ON PARCELS (or PATCHES) ####

# creating empty vectors
areaList = []
parcelID = []
parcelAREA = []
parcelX = []
parcelY = []

# Read and store information on area, coordinates and ID of parcels
areaFile = open(\"weareGrass_Area.txt\",\'r\")
for line in areaFile:
    arealist.append(line.strip(\"\n\")\split(\"\t\")

for row in arealist:
```
import pandas as pd

# Read and store information on biodiversity within each parcel
# Normalize biodiversity
# Reading distances between parcels (distance must be pre-calculated)

# INFORMATION ON RESERVES (or CLUSTERS) 

## Reading distances between parcels (distance must be pre-calculated)
### DISTANCE COMPUTATION ###

#### Computing euclidean distances between parcels/patches and centers of cluster/reserve

```python
Dist =[]
for i in range(0,len(parcelX), 1):
    j=0
    while j<len(centerX):
        Dist.append(float(distance(parcelX[i], parcelY[i], centerX[j], centerY[j])))
        j=j+1
    i=i+1

##Normalize Distances
print "minDist=" + str(min(Dist))
print "maxDist=" + str(max(Dist))
NormDist=[]
for i in range(0,len(Dist),1):
    NormVal=(float(NormDist[(i*len(centerID))+r]),(1-alpha))
    if b<0.000001:
        b=0.000001
    outputFile.write(" +" + str(b)+" X")
    outputFile.write(str(int(parcelID[i]))+"_"+str(int(centerID[r])))

# Objective function (equations 1, 2 and 3)
outputFile.write("Minimize
")
for i in range(0,len(parcelX), 1):
    for r in range(0,len(centerX), 1):
        b = math.pow(float(NormDist[(i*len(centerID))+r]), 1) #change power here
        outputFile.write(" +" + str(b)+" X")
```

```python
outputFile.write(str(int(parcelID[i]))+"_"+str(int(centerID[r])))
```

```python
outputFile.write("-f1 = 0\n")
```

```python
# Objective function done
```

```python
outputFile.write("\nSubject to")
```
if b<0.000001:
    b=0.0001
outputFile.write(" + str(b) + 'B"
)  
outputFile.write(" - f2 = 0\n")  
print "tracking f1 and f2 done"

outputFile.write(str(int(parcelID[i])))

outputFile.write("f1 > 0 \n")
outputFile.write("f2 > 0 \n")
print "f1 and f2 must be greater than zero"

# definition of state variables
outputFile.write(" define state variable Z \n")
r=0 #cluster r
while r < len(centerID):
    s = 0 #cluster s, s=r+1
    while s<len(centerID):
        i=0 #parcel i
        while i<len(parcelID):
            j=i #parcel j
            while j<len(parcelID):
                if (r != s) and (j != i):
                    outputFile.write("Z" + str(int(parcelID[i]))+"_"+str(int(centerID[r ]))) + "_" + str(int(parcelID[j]))+"_"+str(int(centerID[s])))
                    outputFile.write(" + X" + str(int(parcelID[i]))+"_"+str(int(centerID[j]))) + "_"+str(int(centerID[s])))
            outputFile.write("=2\n")
            j+=1
        i+=1
        s+=1
    r+=1

# Equation 4: when a patch i is selected, it can belong to at most one reserve
i=0
while i < len(parcelID):
    r=0
    while r < len(centerID):
        outputFile.write("+")
        outputFile.write("X")
        outputFile.write(str(int(parcelID[i]))+"_"+str(r))
        r+=1
        outputFile.write("-B")
        outputFile.write(str(int(parcelID[i])))
        outputFile.write("=0\n")
        i+=1
    outputFile.write("\n")

# Equation 5: Each parcel must be assigned to only one cluster
outputFile.write("\n")
outputFile.write(" parcel be assigned to a cluster iff cluster exist \n")
i=0
while i < len(parcelID):
    r=0
    while r < len(centerID):
        outputFile.write("X")
        outputFile.write(str(int(parcelID[i]))+"_"+str(int(centerID[r]) + ' - '))
        outputFile.write("Y"+str(int(centerID[r])) +"<0\n")
r=r+1
i=i+1
print "constraint that each parcel must be assigned to only one cluster/reserve... done"

# equation 6: Number of clusters (reserves) should equal to K (p in the formulation)
outputFile.write("\n")
outputFile.write(\"total number of clusters \n\")
r=0
while r < len(centerID):
    outputFile.write("+\"
    outputFile.write("Y"+str(int(centerID[r])))
    r=r+1
outputFile.write("=\" + str(K))
print "constraint stipulating the number of clusters/reserves... done"

# equation 7: If a parcel i belongs to cluster r and parcel i' belongs to cluster r', then
# i and i' must be separated by a minimum distance S. d(i,i',r,r') >= S-(2-Xi-Xi') * M
outputFile.write("\n")
outputFile.write("\nminimum distance between parcels\n")
r=0  #cluster r
while r < len(centerID):
    s = 0  #cluster s, s=r+1
    while s<len(centerID):
        i=0  #parcel i
        while i<len(parcelID):
            j=i  #parcel j
            while j<len(parcelID):
                if (r != s) and (j != i):
                    b = round(float(minDist[i][j]), 2)
                    outputFile.write(str(M) + " Z" + str(int(parcelID[i]))+"._"+str(int(centerID[r])) + ":_" + str(int(parcelID[j]))+"._"+str(int(centerID[s])))
                    outputFile.write(">=\" + str(S-b) + "\n")
            j+=1
        i+=1
    s+=1
    r+=1

# equation 8: cluster (reserve) should contain at least a certain percentage (F) of the total area
outputFile.write("\n\a cluster/reserve should contain at least a percentage (F) of the total area"
outputFile.write("\n")
r=0
while r < len(centerID):
    i=0
    while i<len(parcelID):
        outputFile.write("+\" + str(parcelAREA[i])
        outputFile.write(" X")
        outputFile.write(str(int(parcelID[i]))+"."+str(int(centerID[r])))
        i+=1
    outputFile.write("-\" + str(float(F) * sum(parcelAREA)) + " Y" + str(int(centerID[r])) + "}\n")
    r+=1
outputFile.write("\n")
print "cluster should contain at least 8 parcels"

# equation 9: total area of parcels which are kept should be equal a percentage (A) of the total area
outputFile.write("\n total number of area set aside should be equal a percentage (A) of the total area")
outputFile.write("\n")
i=0
while i < len(parcelID):
    outputFile.write("+ " + str(parcelAREA[i]))
    outputFile.write("B")
    outputFile.write(str(int(parcelID[i])))
i+=1
outputFile.write(">=" + str(float(A) * sum(parcelAREA)) + "\n")
print "total area greater than A"

# equation 10: General constraints
outputFile.write("\n\nGENERAL\n")
r=0
while r < len(centerID):
    s = 0
    while s < len(centerID):
        i=0
        while i < len(parcelID):
            j=i
            while j < len(parcelID):
                if (r != s) and (j != i):
                    outputFile.write("Z" + str(int(parcelID[i])) + "_" + str(int(centerID[r])) + "_" + str(int(parcelID[j])) + "_" + str(int(centerID[s])))
            outputFile.write("\n")
        j+=1
    i+=1
    s+=1
    r+=1
print "constraint: integer constraint...done"
outputFile.write("\n\n")

# equation 10: Binary constraints
outputFile.write("BINARY\n")
i=0
while i < len(parcelID):
    r=0
    while r < len(centerID):
        outputFile.write("X")
        outputFile.write(str(int(parcelID[i])) + "_" + str(int(centerID[r])))
        outputFile.write("\n")
    r +=1
    i=i+1
print "constraint: binary constraint...done"
```python
350.    outputFile.write("B"+str(int(parcelID[i]))+"\n")
351.    outputFile.write("\n")
352.    i=i+1
353.    print "constraint: binary constraint...done"
354.
355.
356.    r=0
357.    while r < len(centerID):
358.        outputFile.write("Y")
359.        outputFile.write(str(int(centerID[r])))
360.        outputFile.write("\n")
361.        r +=1
362.
363.    outputFile.write("END\n")
364.    outputFile.close()
```