Topsis Algorithm

Variable description:

number\_row: Number of objects to be compared

number\_column: Number of constraints in dataset

weight[]: 1-D matrix to store the weights associated to a constraint

beneficial []: 1-D matrix to determine whether a constraint is beneficial or non-benefecial

detail\_matrix[][] : 2-D matrix to store details of the objects to be compared

normalized\_matrix[][] : 2-D matrix to store the normalized values of data in detail\_matrix

ideal\_best[]: 1-D matrix to store the best or ideal value among all the objects being compared

ideal\_worst[]: 1-D matrix to store the worst value among all the objects being compared

dist\_from\_best[]: 1-D matrix to store the distance of an object from best value in that column ( i.e. under same constraint)

dist\_from\_worst[]: 1-D matrix to store the distance of an object from worst value in that column ( i.e. under same constraint)

preference\_matrix[]: 1-D matrix to store preferability factor

rank[]: 1-D matrix to store calculated ranks corresponding to that object

Pseudocode

topsis\_algorithm()

normalization()

euceledian()

assign\_rank()

normalization()

for i 🡨 0 to number\_column do

t=0.0

for j 🡨 0 to number\_row do

t=t + (detail\_matrix[j][i] \* detail\_matrix[j][i])

end loop

price= sqrt(t)

for j 🡨 0 to number\_row do

normalized\_matrix[j][i]=(detail\_matrix[j][i]/price)\*weight[i];

end loop

end loop

end func

eucledian()

for i 🡨0 to number\_column do

if beneficial[i] == 1

ideal\_best[i]=max(i)

ideal\_worst[i]=min(i)

else if beneficial[i] == 1

ideal\_best[i]=min(i)

ideal\_best[i]=max(i)

end loop

for i 🡨0 to number\_row do

dist\_from\_best=0

dist\_from\_worst=0

for j 🡨0 to number\_column do

tem\_b=normalized\_matrix[i][j]-ideal\_best[j]

dist\_from\_best[i]=dist\_from\_best[i]+(tem\_b\*tem\_b)

tem\_w=normalized\_matrix[i][j]-ideal\_worst[j]

dist\_form\_worst[i]=dist\_from\_worst[i] + (tem\_w \* tem\_w)

end loop

dist\_from\_best[i]=sqrt(dist\_from\_best[i])

dist\_from\_worst[i]=sqrt(dist\_from\_worst[i])

end func

assign\_rank()

for j 🡨 0 to number\_row do

preference\_matrix[ j ] =(dist\_from\_worst[j])/(dist\_from\_best[j]+dist\_from\_worst[j]);

rank[j] 🡨 -1

end loop

ranking 🡨 1

for j 🡨 0 to number\_row do

max\_el\_index= max ( preference\_matrix, number\_row)

rank[max\_el\_index]=ranking;

ranking++

preference\_matrix[max\_el\_index] 🡨 -1;

end loop

// display rankings by displaying contents in rank[]

for i 🡨 0 to number\_row do

print “Object”+ (i+1)+“:”+rank[i]

end loop

end func