TSEAT Algorithm

Parameter Weight matrix, P[m][m]

1. Algorithm
   1. User answer the pairwise comparisons between two parameters (m)
   2. The options are: equally, moderately, greatly, extremely,
      1. respect to weights are: 1.0, 1.2,1.5,2.0
   3. Build up a consistent matrix. Transform matrix using the inverse function
   4. Normalize elements in P dividing by the summation of column

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | f(a) | f(a)f(b) | f(a)f(b)f(c) | f(a)f(b)f(c)f(d) |
|  | 1 | f(b) | f(b)f(c) | f(b)f(c)f(d) |
|  |  | 1 | f(c) | f(c)(d) |
|  |  |  | 1 | f(d) |
|  |  |  |  | 1 |

1. Pseudo code
   1. For i in range (m): comparisons[i] = a, b, …, m-1
   2. For i in range (m):
      1. If comparisons [i] == equally

P[i][i]=1.0; P[i][i+1]=1.0; P[i+1][i]=1.0/P[i][i+1];

* + 1. If comparisons [i] == moderately

P[i][i]=1.0; P[i][i+1]=1.2; P[i+1][i]=1.0/P[i][i+1];

* + 1. If comparisons [i] == greatly

P[i][i]=1.0; P[i][i+1]=1.5; P[i+1][i]=1.0/P[i][i+1];

* + 1. If comparisons [i] == extremely

P[i][i]=1.0; P[i][i+1]=2.0; P[i+1][i]=1.0/P[i][i+1];

* 1. for(int i=0; i<m-2; i++)

for(int j=i+2; j<m; j++)

P[i][j]=P[i][j-1]\*P[j-1][j];

P[j][i]=1.0/P[i][j];

* 1. For i in range (m):

sum1stColP+= P[i][0]

For i in range (m):

parameterWeights[i] = W[i] = P[i][0] / sum1stColP

Option Weight Matrices respect parameter k, O[n][n][m]

1. Algorithm
   1. User input raw data, matrix R[m][n], options (n) respect parameters (m)
   2. Calculate the options matrix O respect parameter k
      1. If preference is higher
      2. If preference is lower
2. Pseudo code
   1. For i in range (m):

For j in range (n):

R[i][j] = r[i][j]

* 1. For k in range (m):

For i in range (n):

For j in range (n):

If Preference == 1:

If R[k][i] >= R[k][j]:

O[i][j][k] = 1 +

If R[k][i] < R[k][j]:

O[i][j][k] =

If Preference == 0:

If R[k][i] <= R[k][j]:

O[i][j][k] = 1 +

If R[k][i] > R[k][j]:

O[i][j][k] =

Utility Score, S[n]

1. Algorithm
   1. Normalize elements in O dividing by the summation of column respect each parameter k
   2. Divided each elements in column by the maximum elements in that column get optionWeight[n][m]
   3. Multiply optionWeight by parameterWeight to get utility score S, and store the max score and the index(the option)
   4. Normalize U by diving the sum of the elements.
      1. The first entry will correspond to the first option and will continue in the order of options.
2. Pseudo code
   1. For j in range (m):

For i in range (n):

sum1stColO [j] += O[i][0][j]

For j in range (m):

For i in range (n):

optionWeights[i][j] = O[i][0][j] / sum1stColO [j]

if optionWeights[i][j] > max[j]:

max[j] = optionWeights[i][j]

* 1. For j in range (m):

For i in range (n):

optionWeights[i][j] /= max[j]

* 1. For i in range (n):

For j in range (m):

S[i] = optionWeights[i][j] \* parameterWeights[j]

if(S[i]>optimalScore)

optimalScore = S[i];

optimalOption = i;

sumS += S[i]

* 1. For i in range (n):

S[i] /= sumS

Confidence Scores, SDR[m][n-1]

1. Algorithm
   1. Normalize elements in O dividing by the summation of the optimal option column (z)respect each parameter k
   2. After changing in the Option Weight Matrices, E, the new utility score of option will pass the new utility score of optimal option. How many stander deviation, SDR[m][n-1], change in the raw data depends on the range of original raw date
2. Pseudo code
   1. For j in range (m):

For i in range (n-1):

sumC[j] += O[i][z][j]

* 1. For i in range (m):

For j in range (n):

If (j != z):

E =

If (O[j][z][i] >= 1):

SDR[i][j] = E

Else:

If (O[j][z][i] + E <= 1):

SDR[i][j] =

Else:

SDR[i][j] =