

# TODIM Ranking

July 1, 2021

## 1 TODIM Ranking

```
[1]: import math                # for sqrt and other functions
import numpy as np             # for linear algebra
import pandas as pd            # for tabular output
from scipy.stats import rankdata # for ranking the candidates
```

## 2 Step 0 - Obtaining and preprocessing data

```
[2]: attributes_data = pd.read_csv('../data/criteria.csv')
attributes_data
```

```
[2]:
```

	Criteria	Name	Rank	Ideally
0	Price	Price	3	Lower
1	Quality	Quality	5	Higher
2	EC	Energy Consumption	2	Lower
3	GD	Green Design	1	Higher
4	DS	Delivery Speed	6	Higher
5	CSR	Corporate Social Responsibility	4	Higher
6	EE	Employee Education	7	Higher

```
[3]: benefit_attributes = set()
attributes = []
rankings = []
n = 0

for i, row in attributes_data.iterrows():
    attributes.append(row['Criteria'])
    rankings.append(row['Rank'])
    n += 1

    if row['Ideally'] == 'Higher':
        benefit_attributes.add(i)
```

```
[4]: pd.DataFrame(attributes, columns=['Criteria Name'])
```

```
[4]: Criteria Name
0      Price
1      Quality
2      EC
3      GD
4      DS
5      CSR
6      EE
```

```
[5]: rankings = np.array(rankings)
weights = 2 * (n + 1 - rankings) / (n * (n + 1))

pd.DataFrame(weights, columns=['Weights'])
```

```
[5]: Weights
0  0.178571
1  0.107143
2  0.214286
3  0.250000
4  0.071429
5  0.142857
6  0.035714
```

```
[6]: original_dataframe = pd.read_csv('../data/alternatives.csv')
candidates = original_dataframe['Name'].to_numpy()
raw_data = pd.DataFrame(original_dataframe, columns=attributes).to_numpy()

dimensions = raw_data.shape
m = dimensions[0]
n = dimensions[1]

pd.DataFrame(data=raw_data, index=candidates, columns=attributes)
```

```
[6]: Price  Quality  EC  GD  DS  CSR  EE
S1   10.0     4.0  8.0 10.0  2.0  0.70  8.0
S2    4.0     2.0  6.0  8.0  2.0  0.75  6.0
S3    1.0     1.0  8.0  6.0  2.0  0.65  6.0
S4   10.0    10.0  8.0 10.0  8.0  0.85  8.0
S5    2.0     4.0  6.0  6.0  2.0  0.75  6.0
S6   10.0     6.0  8.0  8.0  8.0  0.85  8.0
```

### 3 Step 1 - Normalizing the ratings and weights

```
[7]: for j in range(n):
      column = raw_data[:,j]
      if j in benefit_attributes:
          raw_data[:,j] /= sum(column)
```

```

else:
    column = 1 / column
    raw_data[:,j] = column / sum(column)

pd.DataFrame(data=raw_data, index=candidates, columns=attributes)

```

```

[7]:
      Price  Quality  EC      GD      DS      CSR      EE
S1  0.048780  0.148148  0.15  0.208333  0.083333  0.153846  0.190476
S2  0.121951  0.074074  0.20  0.166667  0.083333  0.164835  0.142857
S3  0.487805  0.037037  0.15  0.125000  0.083333  0.142857  0.142857
S4  0.048780  0.370370  0.15  0.208333  0.333333  0.186813  0.190476
S5  0.243902  0.148148  0.20  0.125000  0.083333  0.164835  0.142857
S6  0.048780  0.222222  0.15  0.166667  0.333333  0.186813  0.190476

```

```

[8]: max_weight = max(weights)
      weights /= max_weight

      pd.DataFrame(data=weights, index=attributes, columns=['Weight'])

```

```

[8]:
      Price  Weight
Quality  0.428571
EC       0.857143
GD       1.000000
DS       0.285714
CSR      0.571429
EE       0.142857

```

## 4 Step 2 - Calculating Dominance Degrees

```

[9]: # The loss attenuation factor
      theta = 2.5

```

```

[10]: phi = np.zeros((n, m, m))

      weight_sum = sum(weights)

      for c in range(n):
          for i in range(m):
              for j in range(m):
                  pic = raw_data[i,c]
                  pjc = raw_data[j,c]
                  val = 0
                  if pic > pjc:
                      val = math.sqrt((pic - pjc) * weights[c] / weight_sum)
                  if pic < pjc:

```

```

        val = -1.0 / theta * math.sqrt(weight_sum * (pjc - pic) /
↪weights[c])
        phi[c, i, j] = val

```

```

[11]: delta = np.zeros((m, m))
      for i in range(m):
          for j in range(m):
              delta[i,j] = sum(phi[:,i,j])

      pd.DataFrame(data=delta, index=candidates, columns=candidates)

```

```

[11]:
      S1      S2      S3      S4      S5      S6
S1  0.000000 -0.327819 -0.292882 -1.516551 -0.536707 -1.171015
S2 -0.700333  0.000000 -0.247943 -1.977770 -0.561087 -1.619643
S3 -0.931105 -0.492989  0.000000 -2.088571 -0.548755 -1.841270
S4  0.356560  0.061872 -0.039756  0.000000 -0.081800  0.228050
S5 -0.363026  0.073358 -0.198827 -1.883937  0.000000 -1.572823
S6  0.128045 -0.092376 -0.130155 -0.633655 -0.189292  0.000000

```

## 5 Step 3 - Calculate ratings from the normalised dominance degree values

```

[12]: delta_sums = np.zeros(m)
      for i in range(m):
          delta_sums[i] = sum(delta[i,:])
      pd.DataFrame(data=delta_sums, index=candidates, columns=['Sum'])

```

```

[12]:
      Sum
S1 -3.844974
S2 -5.106776
S3 -5.902690
S4  0.524926
S5 -3.945254
S6 -0.917433

```

```

[13]: delta_min = min(delta_sums)
      delta_max = max(delta_sums)
      pd.DataFrame(data=[delta_min, delta_max], columns=['Value'], index=['Minimum',
↪'Maximum'])

```

```

[13]:
      Value
Minimum -5.902690
Maximum  0.524926

```

```

[14]: ratings = (delta_sums - delta_min) / (delta_max - delta_min)
      pd.DataFrame(data=ratings, index=candidates, columns=['Rating'])

```

```
[14]:      Rating
      S1  0.320137
      S2  0.123827
      S3  0.000000
      S4  1.000000
      S5  0.304535
      S6  0.775600
```

## 6 Step 4 - Create rankings based on calculated $\xi_i$ values

```
[15]: def rank_according_to(data):
      ranks = (rankdata(data) - 1).astype(int)
      storage = np.zeros_like(candidates)
      storage[ranks] = candidates
      return storage[:, -1]
```

```
[16]: result = rank_according_to(ratings)
      pd.DataFrame(data=result, index=range(1, m + 1), columns=['Name'])
```

```
[16]:      Name
      1   S4
      2   S6
      3   S1
      4   S5
      5   S2
      6   S3
```

```
[17]: print("The best candidate/alternative according to C* is " + str(result[0]))
      print("The preferences in descending order are " + ", ".join(str(r) for r in
      ↪ result) + ".")
```

The best candidate/alternative according to C\* is S4

The preferences in descending order are S4, S6, S1, S5, S2, S3.