Supplement of Geosci. Model Dev., 17, 4031–4052, 2024
https://doi.org/10.5194/gmd-17-4031-2024-supplement

Geoscientific Model Development
Open Access

Supplement of

Decision-making strategies implemented in SolFinder 1.0 to identify eco-efficient aircraft trajectories: application study in AirTraf 3.0

Federica Castino et al.

Correspondence to: Federica Castino (f.castino@tudelft.nl)

SolFinder

Release 1.0.4

Federica Castino

Jun 7, 2023

CONTENTS:

CHAPTER

ONE

INTRODUCTION

1.1 Background information

SolFinder library to find a solution among a set of Pareto optimal solutions, according to the preferences of the decision-maker. This library has been developed with the aim of identifying eco-efficient aircraft trajectories. The following options are available:

- option selecting a solution closest to a target change in one of the objectives
- Gray Relational Analysis (GRA, [1])
- Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS, [1, 2])
- Viekriterijumsko Kompromisno Rangiranje (VIKOR, [1, 3]) method.

1.2 Installation

```
pip install solfinder
```

1.3 Get started

```
import solfinder.MCDM as MCDM
import numpy as np

# Upload example dataset
with open(r'tests/Data_example/POBJ_20180101000005_0_51_28_41.dat', 'r') as f:
    data = np.loadtxt(f, unpack=True)

# Values of objective functions
soc = data[0] # Simple Operating Costs
atr = data[1] # ATR20 total

# Find Pareto optimal solutions using available options
```

```
index_target_05 = MCDM.Target.solution_found_with_target(MCDM.Target(),0.5,soc)
index_gra       = MCDM.GRA.solution_found_by_gra(MCDM.GRA(), data)
index_topsis    = MCDM.TOPSIS.solution_found_by_topsis(MCDM.TOPSIS(), data, [0.5,0.5])
set_indices_vikor, index_vikor = MCDM.VIKOR.solution_found_by_vikor(MCDM.VIKOR(),
                                  data, 0.5, [0.5, 0.5])
```

CHAPTER

TWO

CODE DOCUMENTATION

2.1 Description of solfinder.MCDM module

class solfinder.MCDM.**GRA**

Bases: `object`

A class used to identify the Pareto optimal solution using the Gray Relational Analysis (GRA).

static dist_gra(*v*)

Calculate distance as defined in Gray Relational Analysis (GRA)

Parameters
v (*numpy.ndarray*) – values of i^th objective

Returns
distances from maximum value of v

Return type
numpy.ndarray

static grc(*distances*, *n_sol*)

Calculate Gray Relational Coefficient (GRC)

Parameters

- **distances** (*numpy.ndarray*) – distances from maximum value of v
- **n_sol** (*int*) – number of Pareto optimal solutions

Returns
GRC for each solution

Return type
numpy.ndarray

static norm_gra_min(*v*)

Normalize values of objective function to be minimized

Parameters
v (*numpy.ndarray*) – values of objective function

Returns
normalized v

Return type
numpy.ndarray

static pref_gra(*grc*)

Identify solution with the largest Gray Relational Coefficient (GRC)

Parameters
grc (*numpy.ndarray*) – values of Gray Relational Coefficient for each solution

Returns
index of solution maximizing GRC

Return type
int

solution_found_by_gra(*pobj*)

Select a single solution among the Pareto optimal solutions using Gray Relational Analysis (GRA)

Parameters
pobj (*numpy.ndarray*) – values of objective functions

Returns
index of identified solution

Return type
int

class solfinder.MCDM.**TOPSIS**

Bases: object

A class used to identify the Pareto optimal solution using the Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS).

static closeness(*dist_nis*, *dist_pis*)

Calculate Closeness parameter

Parameters

- **dist_nis** (*numpy.ndarray*) – distances form negative ideal solution
- **dist_pis** (*numpy.ndarray*) – distances from positive ideal solution

Returns
closeness parameters for each solution

Return type
numpy.ndarray

static dist_nis_topsis(*v*)

Calcluate distance of each solution to the Negative Ideal Solution (NIS)

Parameters
v (*numpy.ndarray*) – values of objective function

Returns
distances form NIS

Return type
numpy.ndarray

static dist_pis_topsis(*v*)

Calcluate distance of each solution to the Positive Ideal Solution (PIS)

Parameters
v (*numpy.ndarray*) – values of objective function

Returns
distances form PIS

Return type
numpy.ndarray

`static norm_topsis(`*w*, *v*`)`

Normalize and weight values of objective function

Parameters

- **w** (*float*) – relative weight assigned to objective v
- **v** (*numpy.ndarray*) – values of objective function

Returns
normalized and weighted values of v

Return type
numpy.ndarray

`static pref_topsis(`*c*`)`

Identify solution with largest value of the closeness parameter

Parameters
c (*numpy.ndarray*) – closeness parameter

Returns
index of identified solution

Return type
int

`solution_found_by_topsis(`*pobj*, *weights*`)`

Select a single solution among the Pareto optimal solutions using Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS)

Parameters

- **pobj** (*numpy.ndarray*) – values of optimization objective functions
- **weights** (*list*) – relative weights of objective functions

Returns
index of solution found using TOPSIS

Return type
int

`class solfinder.MCDM.Target`

Bases: `object`

A class used to identify the Pareto optimal solution closest to a target percentage change in one of the objective functions.

`static rel_change(`*v*`)`

Calculate relative change w.r.t. minimum value of objective

Parameters
v (*numpy.ndarray*) – values of objective function at each pareto opt. solution

Returns
relative change w.r.t. minimum value of v

Return type
numpy.ndarray

`solution_found_with_target`(*x*, *v*)

Identify solution closest to x% increase in objective v

Parameters

- **x** (*float*) – target percentage change
- **v** (*numpy.ndarray*) – values of objective function at each pareto opt. solution

Returns
index of identified solution

Return type
numpy.int64

`class solfinder.MCDM.VIKOR`

Bases: `object`

A class used to identify the Pareto optimal solution using the Viekriterijumsko Kompromisno Rangiranje (VIKOR) method.

`static dist_r_vikor`(*w*, *v*)

Calculate regret measure (R)

Parameters

- **w** (*list*) – relative weights of optimization objectives
- **v** (*numpy.ndarray*) – values of objective functions

Returns
values of the parameter R

Return type
numpy.ndarray

`static dist_s_vikor`(*w*, *v*)

Calculate utility measure (S)

Parameters

- **w** (*list*) – relative weights of optimization objectives
- **v** (*numpy.ndarray*) – values of objective functions

Returns
values of the parameter S

Return type
numpy.ndarray

`static pref_vikor`(*q*, *dist_s*, *dist_r*)

Rank solutions and use conditions of acceptable advantage and stability

Parameters

- **q** (*numpy.ndarray*) – values of the parameter Q
- **dist_s** (*numpy.ndarray*) – values of the parameter Q
- **dist_r** (*numpy.ndarray*) – values of the parameter Q

Returns
index/indices of recommended solution(s)

Return type
list

static q_vikor(*gamma*, *s*, *r*)

Calculate parameter Q, combining S and R

Parameters

- **gamma** (*float*) – relative importance of group utility
- **s** (*numpy.ndarray*) – values of the parameter S
- **r** (*numpy.ndarray*) – values of the parameter R

Returns
values of the parameter Q

Return type
numpy.ndarray

static single_vikor(*pobj*, *weights*, *set_sol_selected_by_vikor*)

Select a single solution among the set of solutions identified with the VIKOR method

Parameters

- **pobj** (*numpy.ndarray*) – values of objective function
- **weights** (*list*) – relative importance of optimization objectives
- **set_sol_selected_by_vikor** (*list*) – indices of solutions selected by VIKOR

Returns
index of selected solution

Return type
int

solution_found_by_vikor(*pobj*, *gamma*, *weights*)

Select a set of solutions among the Pareto optimal solutions using the Viekriterijumsko Kompromisno Rangiranje (VIKOR) method

Parameters

- **pobj** (*numpy.ndarray*) – values of objective function
- **gamma** (*float*) – relative importance of group utility
- **weights** (*list*) – relative importance of optimization objectives

Returns
set of solutions, single solution selected by VIKOR

Return type
list, int

class solfinder.MCDM.**VikorTarget**

Bases: `object`

A class used to identify the Pareto optimal solution using the Viekriterijumsko Kompromisno Rangiranje (VIKOR) method, while constraining the increase in one of the objectives

static solution_found_by_vikor_target(*pobj*, *gamma*, *weights*, *index_limited_obj*, *x*)

Select a solution with VIKOR. If the resulting change in one of the objective is larger than a threshold, then pick the solution closest to such threshold instead.

Parameters

- **pobj** (*numpy.ndarray*) – values of objective function
- **gamma** (*float*) – relative importance of group utility
- **weights** (*list*) – relative importance of optimization objectives
- **index_limited_obj** (*int*) – index of the objective to be constrained
- **x** (*float*) – target/threshold relative change of objective

Returns
index of selected solution

Return type
int

2.2 Example application

```
import solfinder.MCDM as MCDM
import numpy as np
import matplotlib.pyplot as plt

# Upload example dataset
with open(r'tests/Data_example/POBJ_20180101000005_0_51_28_41.dat', 'r') as f:
    data = np.loadtxt(f, unpack=True)

# Values of objective functions
soc = data[0]
atr = data[1]

# Number of Pareto optimal solutions
n_sol = len(soc)

# Find Pareto optimal solutions using available options
index_target_05 = MCDM.Target.solution_found_with_target(MCDM.Target(),0.5,soc)
index_gra       = MCDM.GRA.solution_found_by_gra(MCDM.GRA(), data)
index_topsis    = MCDM.TOPSIS.solution_found_by_topsis(MCDM.TOPSIS(), data, [0.5,0.5])
set_indices_vikor, index_vikor = MCDM.VIKOR.solution_found_by_vikor(MCDM.VIKOR(),
                                 data, 0.5, [0.5, 0.5])

# Plot of Pareto front and selected solutions
plt.scatter(100 * (atr - max(atr)) / max(atr),
            MCDM.Target.rel_change(soc), s=20, c='grey')
plt.scatter(100 * (atr[index_target_05] - max(atr)) / max(atr),
            100 * (soc[index_target_05] - min(soc)) / min(soc),
            s=40, c='red', label='Target +0.5% SOC')
plt.scatter(100 * (atr[index_gra] - max(atr)) / max(atr),
            100 * (soc[index_gra] - min(soc)) / min(soc),
            s=40, c='blue', label='GRA')
```

```
plt.scatter(100 * (atr[index_topsis] - max(atr)) / max(atr),
            100 * (soc[index_topsis] - min(soc)) / min(soc),
            s=40, c='orange', label='TOPSIS')
plt.scatter(100 * (atr[index_vikor] - max(atr)) / max(atr),
            100 * (soc[index_vikor] - min(soc)) / min(soc),
            s=40, c='green', label='VIKOR')
plt.xlabel(r'Change in ATR20 [%]', fontsize=16, labelpad=15)
plt.ylabel(r'Change in SOC [%]', fontsize=16, labelpad=15)
plt.title('Number of solutions: {}'.format(n_sol))
plt.legend()
plt.grid(True)
plt.show()
```

Output:

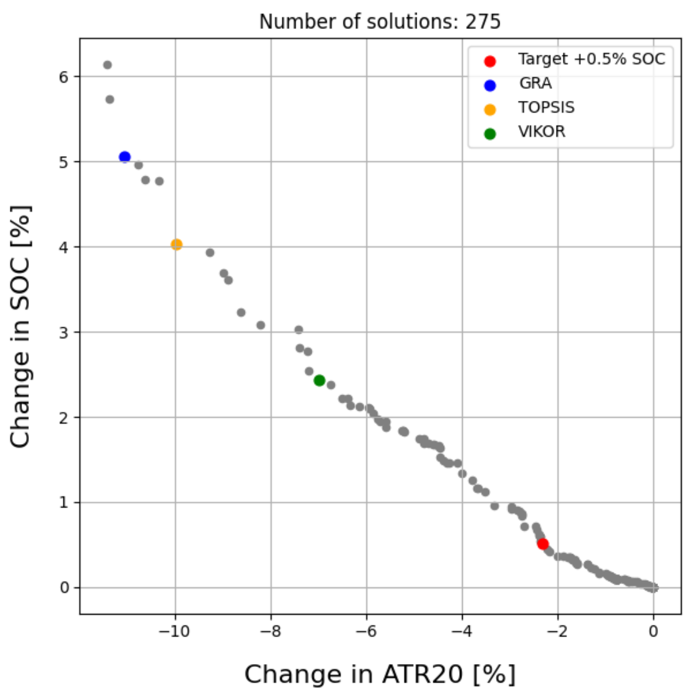

Figure S1: Output of example application code.

BIBLIOGRAPHY

[1] Zhiyuan Wang and Gade Pandu Rangaiah. Application and Analysis of Methods for Selecting an Optimal Solution from the Pareto-Optimal Front obtained by Multiobjective Optimization. *Industrial & Engineering Chemistry Research*, 56(2):560–574, 1 2017.

[2] Shu-Jen Chen and Ching-Lai Hwang. *Fuzzy Multiple Attribute Decision Making*, volume 375. Springer Berlin Heidelberg, Berlin, Heidelberg, 1992.

[3] Serafim Opricovic and Gwo-Hshiung Tzeng. Compromise solution by MCDM methods: A comparative analysis of VIKOR and TOPSIS. *European Journal of Operational Research*, 156(2):445–455, 7 2004.

INDEX