

# Systemy Wspomagania Decyzji

## Optymalizacja wielokryterialna

25 stycznia 2023

### Spis treści

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Opis projektu</b>  | <b>2</b>  |
| <b>2</b> | <b>Implementacja metod</b>  | <b>2</b>  |
| 2.1      | Metoda Fuzzy Topsis . . . . .   | 2         |
| 2.1.1    | Podstawowe definicje . . . . .  | 2         |
| 2.1.2    | Reprezentacja danych z alternatyw w postaci liczb rozmytych . . . . . | 3         |
| 2.1.3    | Opis metody . . . . .   | 4         |
| 2.1.4    | Implementacja . . . . .   | 5         |
| 2.2      | Metoda RSM . . . . .  | 8         |
| 2.2.1    | Opis metody . . . . .   | 8         |
| 2.2.2    | Opis działania algorytmu . . . . .                                    | 10        |
| 2.2.3    | Implementacja . . . . .   | 10        |
| 2.3      | Metoda SP . . . . .   | 12        |
| 2.3.1    | Opis metody . . . . .   | 12        |
| 2.3.2    | Implementacja . . . . .   | 12        |
| 2.4      | Metoda UTA . . . . .  | 14        |
| 2.4.1    | Podstawowe definicje . . . . .  | 14        |
| 2.4.2    | Opis metody . . . . .   | 14        |
| 2.4.3    | Implementacja . . . . .   | 17        |
| <b>3</b> | <b>Porównanie rankingów</b>   | <b>20</b> |
| <b>4</b> | <b>GUI</b>  | <b>23</b> |
| <b>5</b> | <b>Podsumowanie</b>   | <b>23</b> |
| <b>6</b> | <b>Podział pracy</b>  | <b>24</b> |
| <b>7</b> | <b>Bibliografia</b>   | <b>24</b> |

# 1 Opis projektu

Problem zrealizowany w ramach projektu polegał na wyborze kierunku studiów i uczelni na podstawie kryteriów takich jak:

- procent zdawalności,
- ocena absolwentów,
- personalna ocena sylabusu,
- liczba semestrów,
- próg rekrutacji.

W Excelu została stworzona przykładowa baza danych, która umożliwiła zaprezentowanie działania algorytmów jak również aplikacji. Baza zawiera następujące informacje:

- Miasto
- Nazwa kierunku
- Nazwa uczelni
- Procent zdawalności w zakresie 0-100%
- Ocena absolwentów w zakresie od 1-5
- Własna ocena sylabusu w zakresie 1-5
- Ilość semestrów
- Próg rekrutacji w poprzednim roku w zakresie 0-100%
- Rodzaj kierunku (np. tech, hum, ekon etc.)

a następnie wyeksportowana do plików csv. Problem przeanalizowano pod kątem różnych metod optymalizacji wielokryterialnej i utworzono rankingi porządkujące kierunki na podstawie powyższych kryteriów i odpowiadających im wag, które zostały przypisane przez użytkownika aplikacji. Następnie stworzono proste GUI umożliwiające wybór odpowiadających kryteriów i wag za pomocą list rozwijanych oraz pól do wpisania zmiennych liczbowych, pokazuje ono zbiór alternatyw, a także wygenerowany przez wybrany algorytm ranking oraz porównanie rankingów.

## 2 Implementacja metod

### 2.1 Metoda Fuzzy Topsis

#### 2.1.1 Podstawowe definicje

Liczba rozmyta to funkcja mapująca  $\mathbb{R} \rightarrow [0, 1]$ . W zaimplementowanej metodzie przyjęto definicję liczby rozmytej scharakteryzowanej przez trzy wartości: a, b oraz c :

$$x(t) = \begin{cases} 0 & \text{if } t < a \\ \frac{t-a}{b-a} & \text{if } a \leq t < b \\ \frac{c-t}{c-b} & \text{if } b \leq t < c \\ 0 & \text{if } t > c \end{cases}$$

Zdefiniowano także operacje przeprowadzane na liczbach rozmytych:

- $\tilde{x} + \tilde{y} = (a_x + a_y, b_x + b_y, c_x + c_y)$
- $\tilde{x} - \tilde{y} = (a_x - a_y, b_x - b_y, c_x - c_y)$
- $\alpha \tilde{x} = (\alpha a_x, \alpha b_x, \alpha c_x)$
- $\tilde{x} \times \tilde{y} = (a_x a_y, b_x b_y, c_x c_y)$
- $d(\tilde{x}, \tilde{y}) = \sqrt{\frac{1}{3} [(a_x - a_y)^2 + (b_x - b_y)^2 + (c_x - c_y)^2]}$

### 2.1.2 Reprezentacja danych z alternatyw w postaci liczb rozmytych

Algorytm wykorzystuje liczby rozmyte, dlatego na początku należało utworzyć rozmytą reprezentację naszych danych z bazy. Pod uwagę wzięliśmy 5 kryteriów :

- procent zdawalności
- ocena absolwentów
- własna ocena syllabusa
- liczba semestrów
- próg rekrutacji

Do rozmycia ocen zawierających się w zbiorze 1-5 zastosowano skalę lingwistyczną:

|               |           |
|---------------|-----------|
| Very good (5) | (9,10,10) |
| Good (4)      | (7,9,10)  |
| Medium (3)    | (3,5,7)   |
| Poor (2)      | (1,3,5)   |
| Very poor (1) | (1,1,3)   |

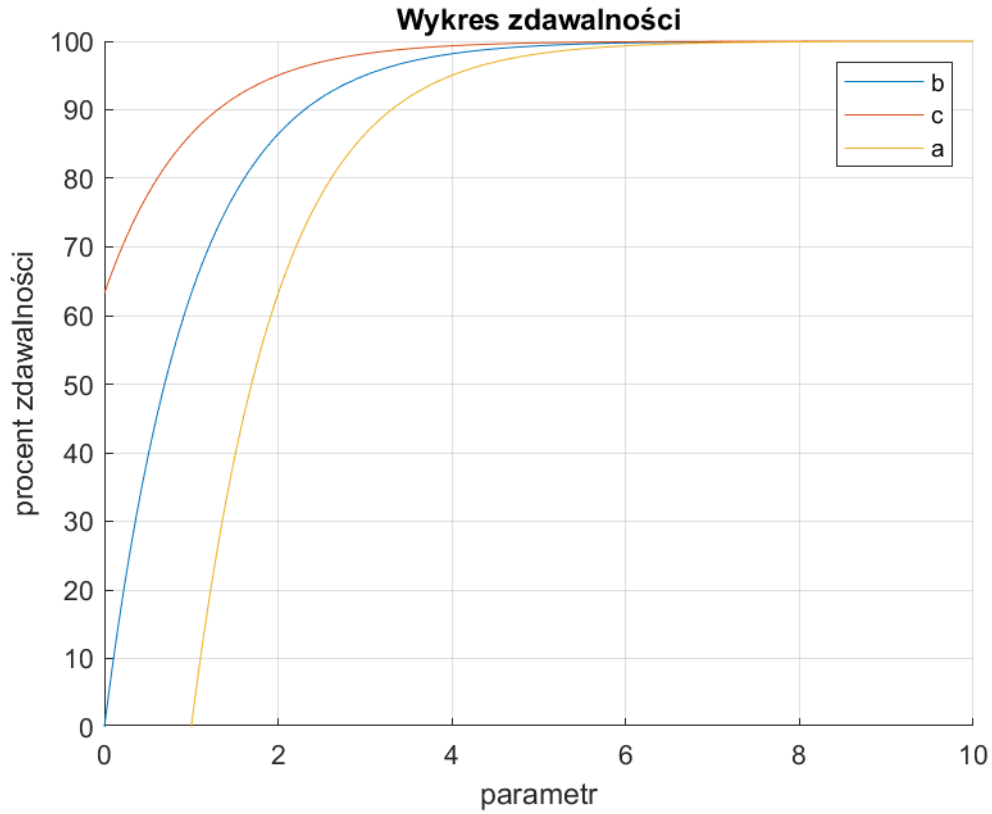
Reprezentacja kryterium *liczba semestrów* została przedstawiona w następujący sposób:

(ilość semestrów, ilość semestrów, ilość semestrów + 2),

wyбір ten wynika z faktu, że dla studentów, którzy nie uzyskali zaliczenia danego semestru czas studiowania wydłuża się o jeden rok z uwagi na warunek. Kryterium *próg rekrutacji* jest reprezentowane jako próg rekrutacji (b), przy czym dolna (a) i górna wartość (c) różni się o 10 punktów procentowych z uwzględnieniem, że wartość maksymalna nie może przekroczyć 100 i wartość minimalna nie może być mniejsza niż 0.

Reprezentację kryterium *procent zdawalności* przedstawiliśmy jako nieliniową funkcję:

$$\begin{cases} x = -\ln\left(1 - \frac{\text{prog zdawalności}}{100}\right) \\ a = 1 - e^{-(x-1)} \\ b = 1 - e^{-(x)} \\ c = 1 - e^{-(x+1)} \end{cases}$$



Rysunek 1: Rozmycie kryterium *procent zdawalności* w zależności od aktualnego procentu zdawalności (b)

Dla wag kryteriów zastosowano skalę 1-9 gdzie poszczególne wagi w odniesieniu do liczb rozmytych można przedstawić w następujący sposób:

|                                     |         |
|-------------------------------------|---------|
| Absolutely important (9)            | (7,9,9) |
| Very strongly extreme important (8) | (6,8,9) |
| Very strongly important (7)         | (5,7,9) |
| Strongly important (6)              | (4,6,8) |
| Moderately strong important (5)     | (3,5,7) |
| Moderate important (4)              | (2,4,6) |
| Weakly important (3)                | (1,3,5) |
| Equally moderate important (2)      | (1,2,4) |
| Equally important (1)               | (1,1,3) |

### 2.1.3 Opis metody

Po konwersji danych do liczb rozmytych otrzymujemy macierz alternatyw  $\tilde{X}$ :

$$\tilde{X} = \begin{bmatrix} \tilde{x}_{11} & \tilde{x}_{12} & \dots & \tilde{x}_{1n} \\ \tilde{x}_{21} & \tilde{x}_{22} & \dots & \tilde{x}_{2n} \\ \dots & \dots & \dots & \dots \\ \tilde{x}_{m1} & \tilde{x}_{m2} & \dots & \tilde{x}_{mn} \end{bmatrix}$$

W rozważanym problemie uwzględniono udział jednego eksperta, który przypisuje wagi odpowiednim kryteriom podczas wywoływania metody (wagi oznaczane są jako  $\tilde{w}_j$  i są reprezentowane poprzez liczbę rozmytą dla j-tego kryterium).

Następnym etapem była normalizacja macierzy  $\tilde{X}$  oraz  $\tilde{R} = [\tilde{r}_{ij}]$  - osiągnięto ją normalizując odpowiednie kolumny macierzy  $\tilde{X}$  w zależności od występowania maksymalizacji lub minimalizacji kryterium:

- maksymalizacja:

$$\tilde{r}_{ij} = \left( \frac{a_{ij}}{c_j^*}, \frac{b_{ij}}{c_j^*}, \frac{c_{ij}}{c_j^*} \right) \quad \text{gdzie} \quad c_j^* = \max_i (c_{ij})$$

- minimalizacja:

$$\tilde{r}_{ij} = \left( \frac{a_j^-}{c_{ij}}, \frac{a_j^-}{b_{ij}}, \frac{a_j^-}{a_{ij}} \right) \quad \text{gdzie} \quad a_j^- = \min_i (a_{ij})$$

W kolejnym kroku należało policzyć ważoną znormalizowaną macierz  $\tilde{V} = [\tilde{r}_{ij} \times \tilde{w}_j]$ , a następnie wyznaczyć punkty: *idealny* i *antyidealny* zgodnie ze wzorem:

$$A^* = (\tilde{v}_1^*, \tilde{v}_2^*, \dots, \tilde{v}_n^*) \quad \text{gdzie} \quad \tilde{v}_j^* = \max_i (v_{ij3})$$

$$A^- = (\tilde{v}_1^-, \tilde{v}_2^-, \dots, \tilde{v}_n^-) \quad \text{gdzie} \quad \tilde{v}_j^- = \min_i (v_{ij1})$$

Ostatni etap polegał na obliczeniu dla każdej alternatywy  $A_i$  (stosując wyznaczone wcześniej odległości) współczynnika  $CC_i$  według wzoru:

$$CC_i = \frac{d_i^-}{d_i^- + d_i^*},$$

a także posortowano je tworząc ranking rozważanych alternatyw.

### 2.1.4 Implementacja

```

1 import numpy as np
2
3 class FuzzyNumb:
4     def __init__(self, a : float, b : float, c : float) -> None:
5         self.a = a
6         self.b = b
7         self.c = c
8         pass
9
10    def d(self, other) -> float:
11        """
12        return
13        distance between two triangular fuzzy numbers
14        """
15        return np.sqrt(1/3*((self.a-other.a)**2+(self.b-other.b)**2+(self.c-other.c)**2))
16
17    def normalised_benefit_criteria(self, c_star):
18        """
19        return
20        normalized over benefit criteria triangular fuzzy number
21        """
22        return FuzzyNumb(self.a/c_star, self.b/c_star, self.c/c_star)
23
24    def normalised_cost_criteria(self, a_minus):
25        """
26        return
27        normalized over cost criteria triangular fuzzy number
28        """
29        return FuzzyNumb(a_minus/self.c, a_minus/self.b, a_minus/self.a)
30
31    def __mul__(self, other):
32        return FuzzyNumb(self.a*other.a, self.b*other.b, self.c*other.c)
33
34    def __gt__(self, other):
35        if self.c > other.c:
36            return True
37        elif self.c < other.c:
38            return False
39        elif self.b > other.b:
40            return True
41        elif self.b < other.b:
42            return False
43        elif self.a > other.a:
44            return True

```

```

45         else:
46             return False
47
48     def __lt__(self, other):
49         if self.a < other.a:
50             return True
51         elif self.a > other.a:
52             return False
53         elif self.b < other.b:
54             return True
55         elif self.b > other.b:
56             return False
57         elif self.c < other.c:
58             return True
59         else:
60             return False
61
62     def __repr__(self):
63         return f"({self.a:.3f},{self.b:.3f},{self.c:.3f})"

```

```

1 from FuzzyNum import FuzzyNumb
2 from typing import List
3 import numpy as np
4
5 def normalize(fuz_num : FuzzyNumb, j, max_cols : List[float], min_cols : List[float] ,
6 cost_ : List[str]):
7     if cost_[j] == "min":
8         return fuz_num.normalised_cost_criteria(min_cols[j])
9     else:
10         return fuz_num.normalised_benefit_criteria(max_cols[j])
11
12 def cc(d_minus : np.array, d_star : np.array):
13     ## step 7 and 8
14     cc_i = [(i, d_minus[i]/(d_minus[i]+d_star[i])) for i in range(len(d_minus))]
15     cc_i.sort(key=lambda x: x[1], reverse=True)
16     return cc_i
17
18 def max_fuzz(V : List[FuzzyNumb]):
19     max_fuz = FuzzyNumb(-np.inf, -np.inf, -np.inf)
20     for i in range(len(V)):
21         if V[i] > max_fuz:
22             max_fuz = V[i]
23     return max_fuz
24
25 def min_fuzz(V : List[FuzzyNumb]):
26     min_fuz = FuzzyNumb(np.inf, np.inf, np.inf)
27     for i in range(len(V)):
28         if V[i] < min_fuz:
29             min_fuz = V[i]
30     return min_fuz
31
32 def fuzzy_topsis(decision_matrix : np.ndarray, weights : List[FuzzyNumb], cost_ : List[
33 str]):
34     ## step 3
35     max_cols = [max(list(decision_matrix[:, i]), key = lambda x: x.c) for i in range(len(
36 decision_matrix[0]))]
37     min_cols = [min(list(decision_matrix[:, i]), key = lambda x: x.a) for i in range(len(
38 decision_matrix[0]))]
39     r : List[List[FuzzyNumb]] = [[normalize(decision_matrix[i, j], j, max_cols, min_cols,
40 cost_) for j in range(len(decision_matrix[0]))] for i in range(len(decision_matrix))]
41     ## step 4
42     V : List[List[FuzzyNumb]] = [[r[i][j]*weights[j] for j in range(len(r[0]))] for i in
43 range(len(r))]
44     ## step 5 compute ideal solution
45     A_ideal = [max_fuzz(np.array(V[:, i])) for i in range(len(V[0]))]
46     A_antiideal = [min_fuzz(np.array(V[:, i])) for i in range(len(V[0]))]
47     ## step 6
48     D_star : List[List[FuzzyNumb]] = [np.sum(np.array([V[i][j].d(A_ideal[j]) for j in
49 range(len(V[0]))])) for i in range(len(V))]
50     D_minus : List[List[FuzzyNumb]] = [np.sum(np.array([V[i][j].d(A_antiideal[j]) for j
51 in range(len(V[0]))])) for i in range(len(V))]
52     ## step 7
53     return cc(np.array(D_minus), np.array(D_star))

```

```

47 def fuzzy_topsis(decision_matrix : np.ndarray, weights : List[FuzzyNumb], cost_ : List[
48     str]):
49     return fuzzy_topsis_(np.array(decision_matrix), weights, cost_)

1 import numpy as np
2 from FuzzyNum import *
3 from fuzzy_topsis import fuzzy_topsis
4 from typing import List
5 import pandas as pd
6
7 def translate_to_fuzzy_preferences(grade : int):
8     if grade == 1:
9         return FuzzyNumb(1,1,3)
10    elif grade == 2:
11        return FuzzyNumb(1,2,4)
12    elif grade == 3:
13        return FuzzyNumb(1,3,5)
14    elif grade == 4:
15        return FuzzyNumb(2,4,6)
16    elif grade == 5:
17        return FuzzyNumb(3,5,7)
18    elif grade == 6:
19        return FuzzyNumb(4,6,8)
20    elif grade == 7:
21        return FuzzyNumb(5,7,9)
22    elif grade == 8:
23        return FuzzyNumb(6,8,9)
24    elif grade == 9:
25        return FuzzyNumb(7,9,9)
26    else:
27        raise ValueError("Nieprawid owa warto wagi")
28
29 def one_to_five_translation(grade : int):
30     if grade == 1:
31         return FuzzyNumb(1,1,3)
32     elif grade == 2:
33         return FuzzyNumb(1,3,5)
34     elif grade == 3:
35         return FuzzyNumb(3,5,7)
36     elif grade == 4:
37         return FuzzyNumb(7,9,10)
38     elif grade == 5:
39         return FuzzyNumb(9,10,10)
40     else:
41         raise ValueError(f"ocena poza skal : grade = {grade}")
42
43
44
45 def translate_value(data_frame) -> List[List[FuzzyNumb]]:
46     df : pd.DataFrame = data_frame[["Procent zdawalno ci","Ocena absolwent w","W asna
47     ocena sylabusa","Ilo semestr w","Pr g rekrutacji"]]
48     df = df.to_numpy()
49     D : List[List[FuzzyNumb]] = []
50     for j in range(df.shape[0]):
51         D.append([])
52         for n,i in enumerate(["Procent zdawalno ci","Ocena absolwent w","W asna ocena
53         sylabusa","Ilo semestr w","Pr g rekrutacji"]):
54             if i == "Procent zdawalno ci":
55                 x = -np.log(1-(df[j,n]/100))
56                 D[j].append(FuzzyNumb(100*max(1-np.exp(-(x-1)),0),df[j,n],100*(1-np.exp
57                 (-(x+1)))))
58             elif i == "Ocena absolwent w":
59                 D[j].append(one_to_five_translation(df[j,n]))
60             elif i == "W asna ocena sylabusa":
61                 D[j].append(one_to_five_translation(df[j,n]))
62             elif i == "Ilo semestr w":
63                 D[j].append(FuzzyNumb(df[j,n],df[j,n],df[j,n]+2))
64             elif i == "Pr g rekrutacji":
65                 D[j].append(FuzzyNumb(max(df[j,n]-10,0),df[j,n],min(df[j,n]+10,100)))
66
67     return D
68 pass

```

```

68 def fuzzy_topsis_do_gui(weights : List[int], data_frame : pd.DataFrame, max_min : List[str
69 ]):
70     """
71     args:
72         weights : List[int] - list that contains numbers 1-9 where 9 is Absolutely
73         important and 1 is Equally important
74         data_frame : df.DataFrame - data from database
75         max_min : List[str] - list of strings for each column: "max" for profit , "min"
76         for cost
77     return:
78         List[Tuple[int,float]] - ranking of decision first is an index of decision second
79         is a scoring function.
80     """
81     weights = [translate_to_fuzzy_preferences(i) for i in weights]
82     D = translate_value(data_frame)
83     if len(weights) != len(max_min):
84         raise ValueError(f"Nie zgadzają się wymiary d ugo wag = {len(weights)},
85         d ugo data_frame = {data_frame.shape[1]}, d ugo max_min = {len(max_min)}")
86     return fuzzy_topsis(D, weights, max_min)

```

## 2.2 Metoda RSM

### 2.2.1 Opis metody

Poszukiwane jest rozwiązanie problemów optymalizacji wielokryterialnej typu:

$$[(F_1, \dots, F_N) : U \rightarrow E] \rightarrow \min(\theta)$$

gdzie  $U$  i  $E$  oznaczają odpowiednio przestrzeń decyzji i przestrzeń kryteriów (tj. zbiór, w którym wartości przyjmuje funkcja  $F$ ),  $F = (F_1, F_2, \dots, F_n)$  jest wektorową funkcją celu, a  $\theta$  jest domkniętym i wypukłym stożkiem, wprowadzającym częściowy porządek w  $E$ . W najbardziej powszechnym przypadku:

$$\theta = IR_+^N, E = IR^N$$

Punkt odniesienia definiujemy jako element przestrzeni kryteriów reprezentujący wartości kryteriów o szczególnym znaczeniu dla decydenta.

#### Funkcja użyteczności

Definiujemy:

$$v : E \rightarrow IR$$

Funkcja  $v$  jest silnie monotonicznie rosnąca, tzn:

$$\forall x, y \in E \quad (x \leq_\theta y, xy \Rightarrow v(x) < v(y))$$

W konsekwencji minimum funkcji  $v$  może być osiągnięte tylko na zbiorze niezdominowanych wartości  $F$ :  $FP(U, \theta)$  i określa najlepsze kompromisowe rozwiązanie problemu. Problem wielokryterialnego podejmowania decyzji sprowadza się do znalezienia lub oszacowania  $v$  i rozwiązania problemu minimalizacji

$$v : (F(U) \rightarrow IR) \rightarrow \min$$

Z silnej monotoniczności  $v$  wynika, że:

$$\arg \min\{v(x) : x \in F(U)\} \subset FP(U)$$

**Każdy punkt odniesienia można scharakteryzować za pomocą dwóch typów informacji:**

- znaczenie dla decydenta, określane na ogół a priori przez ekspertów zaangażowanych we wspomaganie decyzji, zwykle bez brania pod uwagę ograniczeń problemu optymalizacji wektorowej
- relacja do zbioru osiągalnych wartości kryteriów w problemie optymalizacji wektorowej

**Schemat rozwiązywania z zastosowaniem zbiorów odniesienia**

1. Sformułowanie wielokryterialnego problemu optymalizacji
2. Wielokryterialne podejmowanie decyzji
3. Wprowadzenie dodatkowej informacji o preferencjach



- ograniczenia na współczynniki substytucji,
- punkty odniesienia,
- ograniczenia w przestrzeni celów.

4. Oszacowanie funkcji użyteczności, wygenerowanie propozycji rozwiązania kompromisowego
5. Ocena decydenta: prośba o aktualizację rozwiązania, aktualizacja dodatkowej informacji, formułowanie problemu, zatwierdzenie rozwiązania.

### Klasyfikacja punktów odniesienia

Klasyfikacja oparta o informację przekazaną decydentowi z zewnątrz przez ekspertów:

$A_0$  - granice optymalności - punkty odniesienia, które określają dolną granicę obszaru  $Q$ , gdzie optymalizacja kryteriów ma sens. Szacowana użyteczność: taka sama jak dla punktów docelowych tj.:  $v(A_0) = a_1 > 0$ .

$A_1$  - punkty docelowe (poziomy aspiracji, punkty idealne) - elementy  $E$ , które modelują idealne rozwiązanie pożądane przez decydenta. Szacowana użyteczność:  $v(A_1) = a_1 > 0$ .

$A_2$  - Rozwiązania status quo (poziomy zastrzeżone, wartości pożądane) - wartości kryteriów, które muszą być przekroczone podczas procesu decyzyjnego. Szacowana użyteczność:  $v(A_2) = a_2$  gdzie  $a_2 < a_1$

$A_3$  - Antyidealne punkty odniesienia (poziomy porażki) - elementy przestrzeni kryteriów, które odpowiadają rozwiązaniom niekorzystnym. Szacowana użyteczność:  $v(A_3) = a_3$ , gdzie  $a_3 < a_2 < a_1$

### Niesprzeczność punktów odniesienia

**Racjonalność procesu decyzyjnego** Proces decyzyjny będzie nazwany racjonalnym wtedy i tylko wtedy, gdy prowadzi do niezdominowanego rozwiązania problemu optymalizacji wielokryterialnej.

#### Niesprzeczność procesu decyzyjnego

Proces podejmowania decyzji w oparciu o oszacowane użyteczności jest niesprzeczny wtedy i tylko wtedy, gdy:

$$\forall x, y \in E (v(x) < v(y) \Rightarrow x \leq_{\theta} y \text{ lub } x \approx y)$$

gdzie  $v(x)$  jest oszacowaniem użyteczności  $v$  dla  $x$ , a  $x \approx y$  oznacza relację nieporównywalności  $x$  i  $y$ .

Uwaga: Wartości oszacowane muszą być też zgodne z zasadą użyteczności w skalaryzacji przez odległości z powszechnym rozumieniem pojęcia punktów odniesienia

### Wewnętrzna i wzajemna niesprzeczność punktów odniesienia

Zbiór punktów referencyjnych jest wewnętrznie niesprzeczny wtedy i tylko wtedy gdy:  $\forall q_1, q_2 \in A_i$ , a  $q_1$  oraz  $q_2$  są nieporównywalne.

Klasy  $A_j$  oraz  $A_{j+1}$  są wzajemnie niesprzeczne jeśli:

$$\forall x \in A_j \exists y \in A_{j+1} : x \leq_{\theta} y$$

$$\forall y \in A_{j+1} \exists x \in A_j : x \leq_{\theta} y$$

**Twierdzenie 1.** Jeżeli wszystkie klasy punktów odniesienia  $A_i$  dla powyższych problemów są zarówno wewnętrznie, jak i wzajemnie niesprzeczne, wówczas proces rozwiązania jest niesprzeczny.

Sytuacja w której początkowe oceny decydenta są zgodne z sytuacją rzeczywistą osiągalnych wartości jest przedstawiona poniżej jako warunki 1-4:

**Warunek 1.** Docelowe punkty odniesienia powinny mieć niepustą część wspólną ze zbiorem nieosiągalnych, ściśle dominujących punktów.

**Warunek 2.** Rozwiązania status quo powinny być osiągalne.

**Warunek 3.** Anty-idealne punkty odniesienia powinny być zdominowane przez co najmniej jeden punkt osiągalny lub powinny być nieporównywalne z FP(U).

**Warunek 4.** Dolne granice optymalności powinny być częściowo dominujące lub nieporównywalne.

### Proces szacowania użyteczności

Proces szacowania użyteczności obejmuje trzy etapy:

- przybliżone obliczenie poziomu użyteczności  $v$  dla zbiorów odniesienia  $A_j$ ,
- określenie dziedziny  $E$ , gdzie zdefiniowane jest oszacowanie funkcji użyteczności  $v^w$ ,
- interpolacja  $v^w$  obszarach ograniczonych przez zbiory poziomów.

### 2.2.2 Opis działania algorytmu

Nasz algorytm przyjmuje tabelę odniesienia zawierającą wartości z naszej bazy danych oraz dodatkowe parametry, które w tym przypadku są kryteriami (min lub max) wybranymi przez użytkownika (decydenta) wprowadzonymi w GUI. Podana też jest macierz  $A$ , która jest zbiorem punktów odniesienia. Z tabeli wydzielamy te kolumny, które odpowiadają kryteriom potrzebnym do stworzenia rankingu.

Następnie dzięki metodzie wydzielającej punkty niezdominowane i zdominowane znajdujemy nasze macierze  $A_0$  (granice optymalności) i  $A_1$  (punkty docelowe), dzięki użyciu dwukrotnie przywołanej w tym zdaniu funkcji.

Uwaga: Macierz  $A_1$  nie może być pusta.

Do policzenia funkcji skoringowej najpierw musimy policzyć wagi każdego z kryterium a także je znormalizować.

$$v_i = |A_{1i} - A_{0i}|$$
$$w_i = \frac{v_i}{\sum_j v_j}$$

Korzystając z metryki euklidesowej funkcja skoringowa opisana jest wzorem :

$$score_i = w_{jk} * \frac{\sqrt{\sum_i (B_i - A_{0k})^2}}{\sqrt{\sum_i (B_i - A_{1j})^2} + \sqrt{\sum_i (B_i - A_{0k})^2}}$$

gdzie  $B$  jest to macierz punktów dopuszczalnych,  $A_1$  punktów docelowych, a  $A_0$  punktów status-quo.

### 2.2.3 Implementacja

```
1 import numpy as np
2 from typing import Callable, List
3 def zdominowane(decision_matrix : np.ndarray, min_max_criterial_funct : List[Callable[[np
4     .ndarray], float]]):
5     decision_matrix_copy = decision_matrix.copy()
6     decision_matrix = decision_matrix[:, :]
7     lst = []
8     lst2 = []
9     for i in range(len(decision_matrix)):
10         lst.append(list(decision_matrix[i][:]))
11         lst2.append(list(decision_matrix_copy[i][:]))
12     decision_matrix = np.array(lst)
13     decision_matrix_copy = np.array(lst2)
14     lstzd = []
15     lstnzd = []
16     for i in range(len(decision_matrix)):
17         for j in range(len(decision_matrix)):
```

```

17         if i == j:
18             continue
19         temp = np.array([decision_matrix[i,k]<decision_matrix[j][k] if
min_max_criterial_funct[k] == np.min else decision_matrix[i,k]>decision_matrix[j][k]
20         for k in range(len(decision_matrix[0]))])
21         if temp.any():
22             pass
23         else:
24             break
25     else:
26         lstnzd.append(list(decision_matrix_copy[i]))
27     lst_copy=lst.copy()
28     for i in range(len(lst_copy)):
29         if lst_copy[i] not in lstnzd:
30             lstzdz.append(lst_copy[i])
31     return lstnzd,lstzdz

```

```

1 from typing import List,Callable
2 from RSM.nzd_zd import zdominowane
3 import numpy as np
4 from RSM.waga_metryka import wage, metric
5
6 def RSM(df, additional_params):
7     """
8     RSM metoda zbior w odniesienia
9
10    args:
11        A - zbiór punkt w odniesienia
12        C - zbiór punkt w dopuszczalnych
13
14    return
15        lst_skoring - punkty skoringowe zbioru B
16        lst - posortowane punkty ze zbioru B wzgl dem punkt w skoringowych
17    """
18    min_max = [np.min if i == 'min' else np.max for i in additional_params]
19
20    A =
[[120,6,6,3,1],[10,1,1,12,100],[1,1,7,3,100],[120,1,1,13,10],[120,1,0,4,10],[1,1,1,2,10],[130,7,1,1
21
22    B = df[df.columns[3: 8]].values
23
24    A0, rest = zdominowane(A, min_max)
25    A1, rest = zdominowane(rest, min_max)
26
27    if len(A1) == 0:
28        raise ValueError("Nieprawid owe A1")
29
30    wages = [[wage(A0[i], A1[j]) for j in range(len(A1))] for i in range(len(A0))]
31
32    wages = wages/np.sum(wages)
33    score = []
34    for i in range(len(B)):
35        score.append(0)
36        for j in range(len(wages)):
37            for k in range(len(wages[0])):
38                score[i] += wages[j,k] * metric(B[i],A1[k]) / (metric(B[i],A0[j]) +
39                metric(B[i],A1[k]))
40
41    df['RSM_score'] = score
42    return df

```

```

1 from typing import List,Callable
2 import numpy as np
3
4 def wage(a0 : List[float],a1: List[float]):
5     v = 1
6     for n,i in enumerate(a0):
7         v *= np.abs(a1[n] - i)
8     return v
9
10 def metric(a : List[float],b : List[float]):
11     d = np.dot(np.array(a)-np.array(b),np.array(a)-np.array(b))
12     return np.sqrt(d)

```

## 2.3 Metoda SP

### 2.3.1 Opis metody

Metoda SP-CS skupia się na poszukiwaniu takich rozwiązań niezdominowanych, które przy użyciu łamanej szkieletowej łączą punkt status-quo z punktem docelowym. Dla problemu z wieloma punktami odniesienia i docelowymi tworzy się wszystkie możliwe łamane szkieletowej. Wartość funkcji skoringowej każdego punktu oblicza się sumując odległość punktu od łamanej i jej parametr.

Wyznaczanie punktów załamania zaczyna się od wyznaczenia zbioru  $d$ ,

$$d_{j-1i} = \frac{u_i^{j-1} - l_i^{j-1}}{2} \quad 1 \leq i \leq k;$$

k-liczba współrzędnych

Tworzenie zbioru  $d_0$  przedstawiają zależności poniżej

$$d_{01} = \frac{u_1^0 - l_1^0}{2}$$

$$d_{02} = \frac{u_2^0 - l_2^0}{2}$$

$$d_{03} = \frac{u_3^0 - l_3^0}{2}$$

Tworzenie zbioru  $d_1$  przedstawiają zależności poniżej  $d_{11} = \frac{u_1^1 - l_1^1}{2}$

$$d_{12} = \frac{u_2^1 - l_2^1}{2}$$

$$d_{13} = \frac{u_3^1 - l_3^1}{2}$$

Idea zaimplementowanego algorytmu polega na rzutowaniu punktu (alternatywy) na krzywą w miejsce gdzie względem metryki, która definiowana jest najbliższą odległością względem metryki euklidesowej do krzywej woronoja, punkt z krzywej jest najbliżej punktu ocenianego. następnie liczy się długość kawałka krzywej woronoja od punktu status quo do tego rzutowanego punktu. Tę procedurę powtarza się dla każdej pary punktu ze zbioru idealnego i status quo. Następnie dane odległości są sumowane względem wag które wyznaczane są jako hiperobiętości odpowiednich prostopadłościów (hiperścianów) wyznaczanych przez punkty idealne i status quo które definiują naprzeciwległe wierzchołki tego hiperprostopadłościanu przez sumę wszystkich hiperobiętości w dla rozpatrywanych punktów idealnych i status quo.

### 2.3.2 Implementacja

```
1 from typing import List, Callable
2 import numpy as np
3 import pandas as pd
4 from RSM.nzd_zd import zdominowane
5
6 def cum_count_path(woron_points, metrics = None):
7     if metrics == None:
8         metrics = lambda x, y: np.sqrt(np.dot(y-x, y-x))
9     paths = [0]
10    for i in range(1, len(woron_points)):
11        paths.append(paths[i-1]+metrics(woron_points[i-1], woron_points[i]))
12    return np.array(paths)
13
14
15 def costam(w1: np.ndarray, w2: np.ndarray, w3: np.ndarray):
16     w21=w2-w1
17     w21_norm= w21/np.linalg.norm(w21)
18     w3_p=w3-w1
19     dot_p=np.dot(w21_norm, w3_p)
20     new=dot_p*w21_norm
21     if np.linalg.norm(w21)>np.linalg.norm(new):
```

```

22         return np.linalg.norm(new), np.linalg.norm(w3_p - new)
23     return -1,-1
24
25
26 def zwrot_wagi(Ide,Aide):
27     waga=0
28     for i in range(len(Ide)):
29         waga=waga+volume(Ide[i],Aide[i])
30     return waga
31
32
33 def volume(pkt1,pkt2):
34     l=[]
35     for j in range(len(pkt1)):
36         if pkt2[j]>pkt1[j]:
37             l.append(pkt2[j]-pkt1[j])
38         else:
39             l.append(pkt1[j]-pkt2[j])
40     V=1
41     for i in range(len(l)):
42         V=V*l[i]
43     return V
44
45
46 def czy_w_obszarze(u,pkt1,pkt2):
47     isTRUE=[]
48     for i in range(len(u)):
49         isTRUE.append(pkt1[i]<=u[i]<=pkt2[i] or pkt2[i]<=u[i]<=pkt1[i])
50     for j in range(len(isTRUE)):
51         if (isTRUE[j]==False):
52             return -1
53     return volume(pkt1,pkt2)
54
55
56 def woronoi(pkt_1 : np.ndarray,pkt_2 : np.ndarray):
57     N = len(pkt_1)
58     pkt_2 = pkt_2-pkt_1
59     pkt_1_temp = pkt_1
60     pkt_1 = np.zeros(N)
61     woronoi_points = [np.array([0,0,0]) for i in range(2*N)]
62     woronoi_points[0] = pkt_1
63     woronoi_points[-1] = pkt_2
64     size_of_shift = np.abs(pkt_2 - pkt_1)/2
65     sign_of_shift = np.sign(pkt_2 - pkt_1)
66     mask_fixed = np.zeros(N)
67     dimentions = np.ones(len(pkt_1))
68     for i in range(1,N):
69         woronoi_points[i] = woronoi_points[0] + np.min(size_of_shift) * sign_of_shift +
        mask_fixed
70         woronoi_points[2*N-1-i] = woronoi_points[-1] + -1*np.min(size_of_shift) *
        sign_of_shift - mask_fixed
71         idx = np.argmin(size_of_shift)
72         mask_fixed[idx] = woronoi_points[i][idx]
73         size_of_shift[idx] = np.inf
74         sign_of_shift[idx] = 0
75         dimentions[idx] = 0
76     return np.array(woronoi_points) + pkt_1_temp
77
78
79 def norm(A : List[List[float]], C : List[List[float]]):
80     A1=np.array(A)
81     C1=np.array(C)
82     normalizedA=(A1-np.min(A1))/(np.max(A1)-np.min(A1))
83     normalizedC=(C1-np.min(C1))/(np.max(C1)-np.min(C1))
84     return normalizedA,normalizedC
85
86
87 def SPCS(idealny : List[np.ndarray], antyidealny : List[np.ndarray], punkty : List[np.
    ndarray]):
88     scoring = []
89     waga = zwrot_wagi(idealny,antyidealny)
90     for pkt in range(len(punkty)):
91         scoring.append(0)
92

```

```

93     for ide in range(len(idealny)):
94         for anty in range(len(antyeidealny)):
95             minimal = np.inf
96             d_path = 0
97             w = czy_w_obszarze(punkty[pkt], idealny[ide], antyeidealny[anty])
98             if w == -1:
99                 continue
100             woron_point = voronoi(antyeidealny[anty], idealny[ide])
101             cum_path = cum_count_path(woron_point)
102             for i in range(1, len(woron_point)):
103                 d, metric = costam(woron_point[i-1], woron_point[i], punkty[pkt])
104                 if d == -1:
105                     continue
106                 if minimal > metric:
107                     minimal = metric
108                     d_path = cum_path[i-1] + d
109             ### sprawdzanie punktu w voronoja
110             for i in range(1, len(woron_point)):
111                 metric = np.linalg.norm(punkty[pkt] - woron_point[i])
112                 if minimal > metric:
113                     minimal = metric
114                     d_path = cum_path[i]
115             ###
116             scoring[pkt] += w * d_path
117             if waga == 0:
118                 scoring[pkt] = 0
119             else:
120                 scoring[pkt] = scoring[pkt] / waga
121     return scoring
122
123 def gui_spcs(df, additional_params):
124     min_max = ['max', 'max', 'max']
125     A = [[120, 6, 6], [10, 1, 1], [1, 1, 1], [120, 1, 1], [120, 1, 0], [1, 1, 1], [130, 5, 8], [5, 2, 1]]
126     A0, rest = zdominowane(A, min_max)
127     A1, rest = zdominowane(rest, min_max)
128     df_data = df[df.columns[3:6]]
129     num = df_data.to_numpy()
130
131     df['SAFETY_PRINCIPAL_score'] = SPCS(np.array(A0), np.array(A1), num)
132     return df

```

## 2.4 Metoda UTA

### 2.4.1 Podstawowe definicje

Metoda UTA służy porządkowaniu wariantów decyzyjnych z rozważanego zbioru alternatyw, to znaczy, że rozważana jest problematyka stworzenia rankingu. Model zawierający preferencje decydenta jest reprezentowany przez funkcję użyteczności. Globalna użyteczność dla danej alternatywy jest sumą użyteczności częściowych.

$$U(x) = \sum_i u_i(x_i)$$

Funkcje ( $u_i$ ) wyznaczane są niezależnie dla poszczególnych kryteriów, w oparciu o przyjmowane założenie o niezależności kryteriów w sensie preferencji.

Ranking jest zmodyfikowany w taki sposób, aby wartości zawierały się w przedziale  $[0,1]$ .

### 2.4.2 Opis metody

Metoda UTA jest metodą wielokryterialnego podejmowania decyzji. Opiera się na paradygmacie dezagregacji-agregacji. Przyjmujemy odpowiednie oznaczenia:

- $X = W_1, W_2, \dots, W_k$  - skończony zbiór alternatyw.
- $(X^R) = (W_1^*), (W_2^*), \dots, (W_k^*)$  - skończony zbiór alternatyw referencyjnych dla których decydent jest w stanie wyrazić swoje preferencje.
- $F = (f_1), \dots, (f_n)$  - rodzina  $n$  kryteriów oceny.

- $(X_j) = f_j(W_i)$ ,  $W_i \in X$  - zbiór różnych ocen alternatyw na j-tym kryterium. Zakłada się rosnący kierunek preferencji wszystkich kryteriów, tzn. im większa ocena  $(f_j)(W_i)$ , tym lepszy wariant  $W_i$  na kryterium  $f_j$
- $u_i(f_i(\cdot))$  - cząstkowa funkcja użyteczności, czyli funkcja wyrażająca preferencje decydenta dotyczące i-tego kryterium.
- $U(\cdot)$  - funkcja użyteczności globalnej, czyli całkowitej.

Upraszczając opis metody UTA, składa się ona z następujących etapów:

**Etap 1.** Zdefiniowanie skończonego zbioru wariantów  $X$ , określenie spójnej rodziny kryteriów  $F$  oraz zbiorów ocen  $(X_j)$  ( $j=1,2,\dots,n$ ) kryteriów.

**Etap 2.** Wybór zbioru  $X^R$  wariantów referencyjnych, gdzie  $X^R \subseteq X$ .

**Etap 3.** Utworzenie rankingu (uporządkowanie) wariantów ze zbioru referencyjnego.

**Etap 4.** Wyznaczenie zbioru cząstkowych funkcji użyteczności  $u_i(f_j)$  ( $i=1,2,\dots,n$ ), czyli funkcji wyrażających preferencje decydenta dotyczące i-tego kryterium, zgodnych z podanym rankingiem (rozwiązanie problemu regresji porządkowej – problem PM typu liniowego).

**Etap 5.** Konstrukcja funkcji użyteczności  $U$  oraz wyznaczenie ocen końcowych wariantów decyzyjnych oraz ich uporządkowanie. Funkcja użyteczności globalnej (całkowitej) na postać:

$$U(x) = \sum_i u_i(x_i)$$

**Etap 6.** Uporządkowanie wariantów decyzyjnych rosnąco według wartości funkcji użyteczności. Najlepszym wariantem jest ten, dla którego ocena końcowa wynosi 1, najgorszym wariant o ocenie 0.

Dostosowując wyżej wymienione kroki do naszego problemu, algorytm UTA przebiegał następująco:

- Wyznaczyliśmy punkty idealne i antyidealne dla każdego z kryterium.
- Dobraliśmy odpowiednie przedziały dla wartości ogólnej funkcji użyteczności w każdym kryterium, tak aby sumowały się do jedynki, dla punktu idealnego.
- W naszym zadaniu przyjęliśmy 5 kryteriów, którym przyporządkowaliśmy odpowiednie wagi w przedziałach aby wyznaczyć funkcje użyteczności. Z racji na rodzaj zakresów danych w bazie prezentują się one następująco:

| Kryterium             | Zakres danych | Dobrana funkcja użyteczności     |
|-----------------------|---------------|----------------------------------|
| Procent zdawalności   | (0-100)       | (0.2, 0.02, 0)                   |
| Ocena absolwentów     | (1-5)         | (0.2, 0.16, 0.12, 0.08, 0)       |
| Własna ocena sylabusa | (1-5)         | (0.2, 0.16, 0.12, 0.08, 0)       |
| Ilość semestrów       | (6-12)        | (0.2, 0.16, 0.12, 0.08, 0.04, 0) |
| Próg rekrutacji       | (0-100)       | (0.2, 0.10, 0)                   |

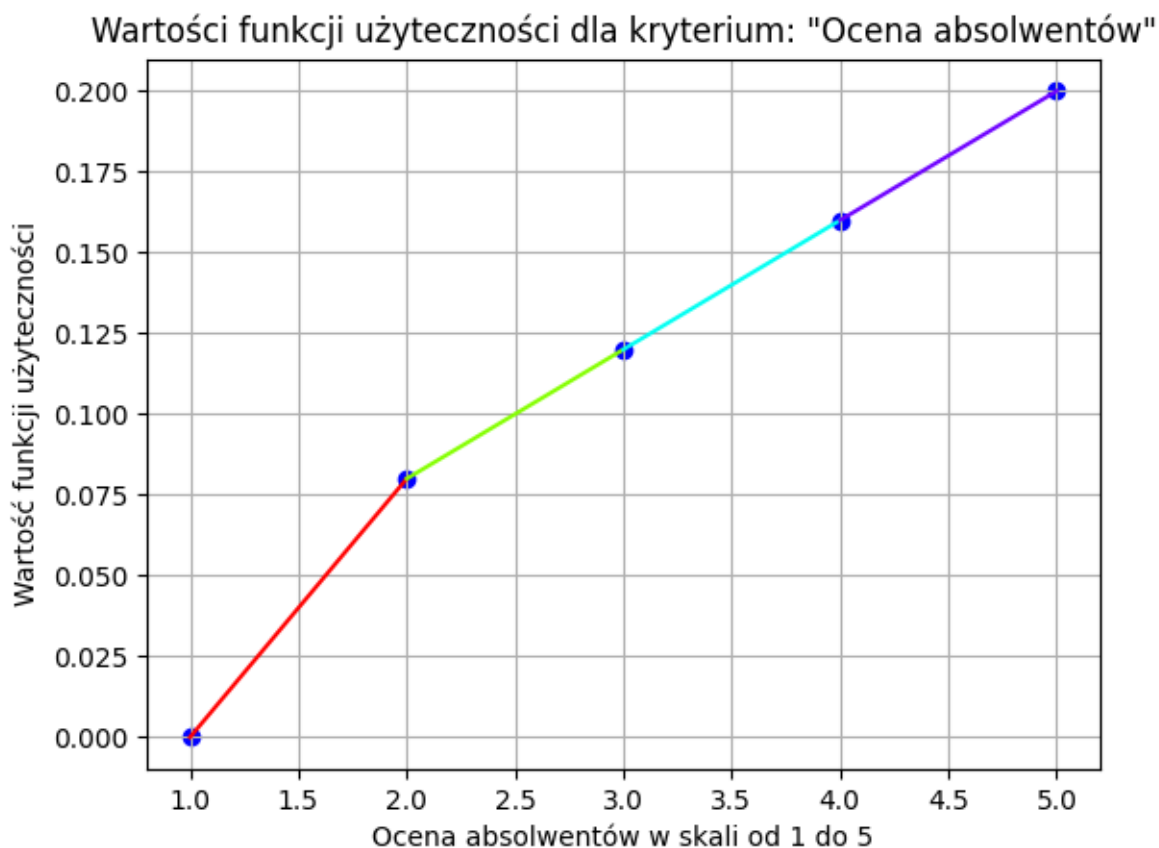
Wartości dla poszczególnych kryteriów są ściśle związane z rodzajem danych w bazie.

- Wyliczyliśmy wartości współczynników funkcji użyteczności dla kryteriów. Sama funkcja użyteczności zawiera się wzorem:

$$y(x) = a * x + b$$

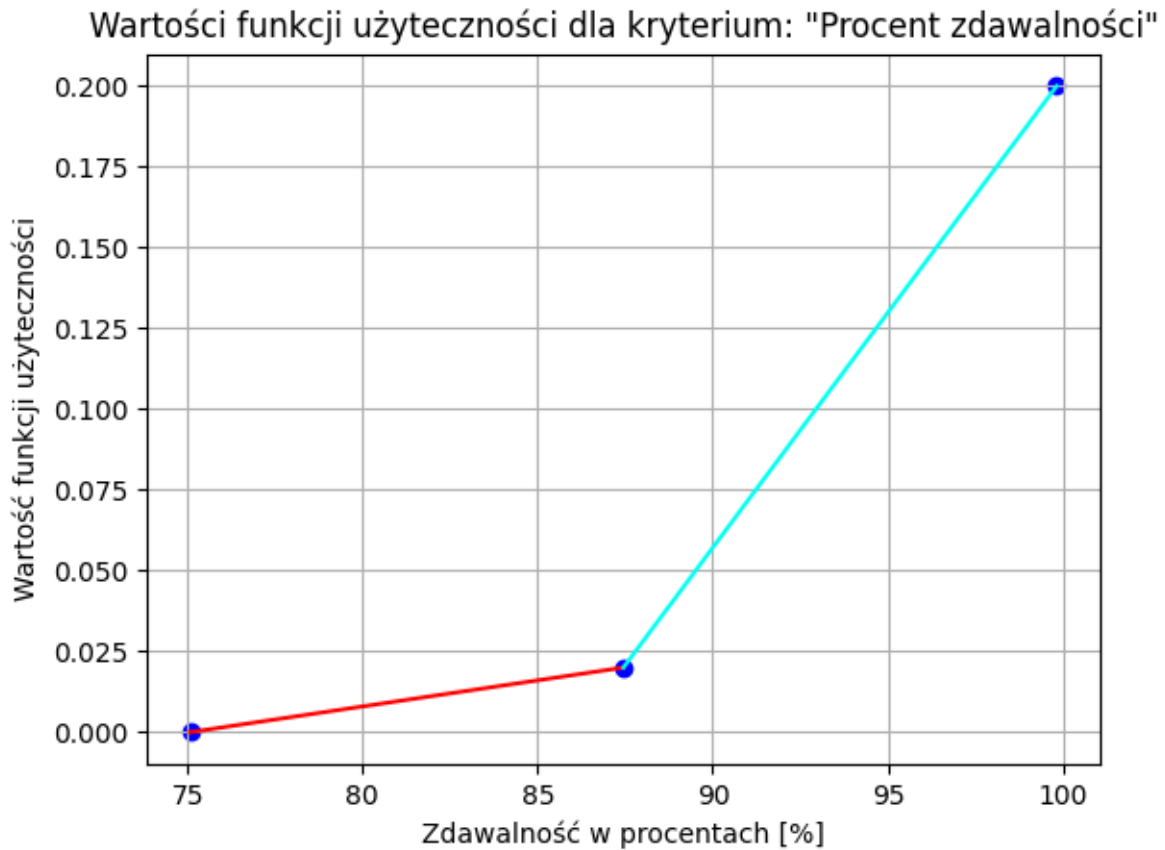
Znak współczynnika kierunkowego  $b$  zależał od wyboru maksymalizacji lub minimalizacji kryterium.

Wykresy poniżej prezentują wartość funkcji użyteczności dla dwóch wybranych kryteriów. Wartość współczynnika kierunkowego determinuje szybkość narastania/opadania funkcji co z kolei jest ściśle powiązane z dobranymi wagami ogólnej funkcji użyteczności.



Rysunek 2: Wykres funkcji użyteczności dla kryterium **ocena absolwentów**.





Rysunek 3: Wykres funkcji użyteczności dla kryterium **procent zdawalności**.

### 2.4.3 Implementacja

```

1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4
5 def get_min_max(inputPoints: np.ndarray):
6     x, y = inputPoints.shape
7     min = 100000 * np.ones(y)
8     max = np.zeros(y)
9     for i in range(x): # iteruj po punktach
10         for j in range(y): # iteruj po parametrach punktu
11             a = inputPoints[i,j]
12             b = min[j]
13             if inputPoints[i,j] < min[j]:
14                 min[j] = inputPoints[i,j]
15             if inputPoints[i,j] > max[j]:
16                 max[j] = inputPoints[i,j]
17
18     return min,max
19
20 def split(min,max,partitions,max_or_min,func_utility = None):
21     list = []
22     alternatives = len(partitions) # ilo kryteri w
23     for i in range(alternatives):
24         row = [(0,0)]*(partitions[i]+1)
25         list.append(row)
26
27     func_utility_max = 1/alternatives # maksymalna warto funkcji u ytecznosci przy
28     danej liczbie kryteri w
29     if not func_utility:

```

```

30     for i in range(alternatives):
31         if max_or_min[i] == 0: # je li minimalizujemy
32             list[i][0] = (min[i],func_utility_max) # wpisuje maksymaln i minimalna
warto kryterium
33             list[i][-1] = (max[i],0)
34             diff = max[i] - min[i] # obliczam r nice miedzy max a min
warto ci kryterium
35             compartment = diff/partitions[i] # wyznaczam przedzia y
36             func_utility_other = func_utility_max/partitions[i] # wyznaczam warto
f. u yteczno ci dla przedzia w
37             for j in range(1,partitions[i]): # wpisuje warto ci dla konkretnych
przedzia w
38                 list[i][j] = (min[i]+ j*compartment,func_utility_max -
func_utility_other*j)
39
40
41         if max_or_min[i] == 1: # je li maksymalizujemy
42             list[i][0] = (max[i],func_utility_max)
43             list[i][-1] = (min[i],0)
44             diff = max[i] - min[i]
45             compartment = diff/partitions[i]
46             func_utility_other = func_utility_max/partitions[i]
47             for j in range(1,partitions[i]):
48                 list[i][j] = (max[i] - j*compartment,func_utility_max -
func_utility_other*j)
49     else:
50         for i in range(alternatives):
51             if max_or_min[i] == 0: # je li minimalizujemy
52                 list[i][0] = (min[i],func_utility[i][0]) # wpisuje maksymaln i
minimalna warto kryterium
53                 list[i][-1] = (max[i],func_utility[i][-1])
54                 diff = max[i] - min[i] # obliczam r nice miedzy max a min
warto ci kryterium
55                 compartment = diff/partitions[i] # wyznaczam przedzia y
56                 for j in range(1,partitions[i]): # wpisuje warto ci dla konkretnych
przedzia w
57                     list[i][j] = (min[i]+ j*compartment,func_utility[i][j])
58
59
60             if max_or_min[i] == 1: # je li maksymalizujemy
61                 list[i][0] = (max[i],func_utility[i][0])
62                 list[i][-1] = (min[i],func_utility[i][-1])
63                 diff = max[i] - min[i]
64                 compartment = diff/partitions[i]
65                 for j in range(1,partitions[i]):
66                     list[i][j] = (max[i] - j*compartment,func_utility[i][j])
67
68     return list
69
70 def function_value(compartments,max_or_min):
71     list = []
72     idx = len(compartments) - 1
73     for i in range(1,len(compartments)):
74         if max_or_min == 1: # maksymalizacja
75             a = (compartments[(idx-i+1)][1]-compartments[(idx-i)][1])/(compartments[(idx-
i+1)][0]-compartments[(idx-i)][0])
76             b = compartments[(idx-i+1)][1]-a*compartments[(idx-i+1)][0]
77             list.append([a,b])
78
79         if max_or_min == 0: # minimalizacja
80             a = (compartments[i-1][1]-compartments[i][1])/(compartments[i-1][0]-
compartments[i][0])
81             b = compartments[(i-1)][1]-a*compartments[(i-1)][0]
82             list.append([a,b])
83
84     return list
85
86 def get_cmap(n, name='hsv'):
87     return plt.cm.get_cmap(name, n)
88
89 def plot_f_utility(u,compartments,max_or_min): # u - lista wsp czynnik w a i b dla f.
u yteczno ci
90     cmap = get_cmap(len(u)+1)
91     idx = len(compartments)-1

```

```

92     for i in range(len(u)):
93         a,b = u[i]
94         if max_or_min == 1: # maksymalizacja
95             x = np.linspace(compartments[idx-i][0],compartments[idx-i-1][0],100)
96             y = a*x+b
97             if i == len(u)-1:
98                 plt.scatter(compartments[0][0],a*compartments[0][0]+b, c = 'blue')
99                 plt.scatter(compartments[idx-i][0],a*compartments[idx-i][0]+b, c = 'blue')
100                 plt.plot(x, y, c =cmap(i))
101
102         if max_or_min == 0: # minimalizacja
103             x = np.linspace(compartments[i][0],compartments[i+1][0],100)
104             y = a*x+b
105             if i == len(u)-1:
106                 plt.scatter(compartments[len(u)][0],a*compartments[len(u)][0]+b, c = '
107                 blue')
108                 plt.scatter(compartments[i][0],a*compartments[i][0]+b, c = 'blue')
109                 plt.plot(x, y, c =cmap(i))
110
111     plt.grid()
112     plt.show()
113
114 def rank(utility_coef, compartments, point):
115     score = 0
116     for i in range(len(compartments[0])):
117         for j in range(len(compartments[i])-1):
118             if point[i] <= compartments[i][j][0] and point[i] >= compartments[i][j+1][0]:
119                 score += utility_coef[i][j][0]*point[i]+utility_coef[i][j][1]
120
121             elif point[i] >= compartments[i][j][0] and point[i] <= compartments[i][j
122             +1][0]:
123                 score += utility_coef[i][j][0]*point[i]+utility_coef[i][j][1]
124
125     return score

```

```

1  from uta import *
2  import numpy as np
3  import pandas as pd
4  pd.options.mode.chained_assignment = None # default='warn'
5  import matplotlib.pyplot as plt
6
7  def uta(df, max_or_min):
8
9      criteria = df[["Procent zdawalno ci","Ocena absolwent w","W asna ocena sylabusa","
10      Ilo      semestr w","Pr g rekrutacji"]]
11      column_names = list(criteria.columns)
12      criteria.rename(columns = {'Procent zdawalno ci':'PZ','Ocena absolwent w':'OA', '
13      W asna ocena sylabusa':'WOS',
14      'Ilo      semestr w':'IS','Pr g rekrutacji':'PR'}, inplace
15      = True)
16      points =[]
17      for index, rows in criteria.iterrows():
18          my_list =[rows.PZ, rows.OA, rows.WOS,rows.IS, rows.PR]
19          points.append(my_list)
20
21      points = np.array(points)
22      min,max = get_min_max(points)
23      max_or_min = np.where(max_or_min=='min', 0, 1)
24
25      # Warto ci funkcji u yteczno ci dobrane r cznie, kod umo liwia
26      # dobranie funkcji u yteczno ci proporcjonalnie, dla takiego przypadku
27      # wsp czynniki a i b wychodz takie same dla wszystkich przedzia w
28
29      func_utility = [[0.2,0.02,0],[0.2,0.16,0.12,0.08,0],[0.2,0.16,0.12,0.08,0],
30      [0.2,0.17,0.14,0.11,0.08,0.05,0],[0.20,0.10,0]]
31      compartments = split(min,max,np.array([2,4,4,7,2]),max_or_min,func_utility)
32
33      # Warto      wsp      czynnik w dla funkcji u yteczno ci w danych przedzia ach
34      u = list()
35      for i in range(len(compartments)):
36          u.append(function_value(compartments[i],max_or_min[i]))
37
38      score = []

```

```

36     for point in points:
37         score.append(rank(u, compartments, point))
38
39     df['UTA_score'] = score
40
41     # for i in range(len(func_utility)):
42     #     plt.title(f'Warto ci funkcji u yteczno ci dla kryterium: {column_names[i]}')
43     #     plot_f_utility(u[i], compartments[i], max_or_min[i])
44
45     return df

```

### 3 Porównanie rankingów

W porównywaniu rankingów zastosowaliśmy metrykę zwaną "Spearman's Footrule" z dodatkowymi wagami wynikającymi z położenia elementów w obu rankingach. Algorytm zakłada przekazanie własnej funkcji wyliczającej wagi lecz preferowanym rozwiązaniem jest zastosowanie funkcji domyślnej. Domyślne wagi są obliczane na podstawie funkcji rozkładu normalnego, gdzie w funkcji można wyspecyfikować podstawowe jego parametry (domyślnie wariancja = 8, średnia = 0). Wagi ( $w_i$ ) dla poszczególnych elementów są wyznaczane jako średnia arytmetyczna wag na pozycjach w obydwóch rankingach. Odległość między rankingami można zapisać jako:

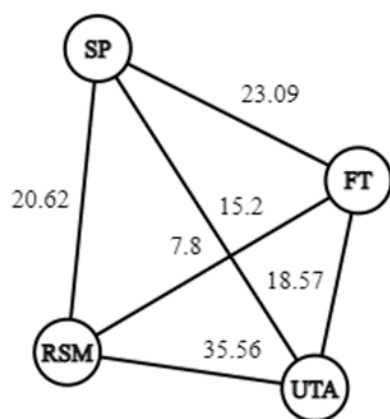
$$F(\sigma) = \sum_i w_i |i - \sigma(i)|$$

Implementacja:

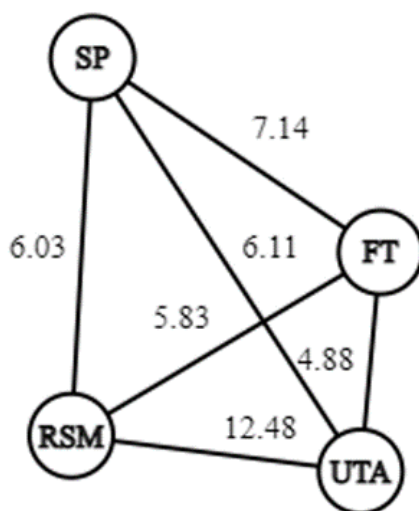
```

1  from typing import List, Callable
2  import numpy as np
3
4  def Spearman_s_Footrule(ranking1 : List[int], ranking2 : List[int], weights_function :
5      Callable[[int],float] = None, mean = 0, deviation = 8):
6      """
7      argv:
8          ranking1 : List[int] - pierwszy ranking do porównania ()
9          ranking2 : List[int] - drugi ranking do porównania
10         weights_function : Callable[[int],float] - funkcja wagowa
11
12         """
13         if weights_function == None:
14
15             weights_function = lambda x: (1/np.sqrt(2*np.pi*deviation))*np.exp(-(x-mean)
16             **2/(2*deviation))
17         if type(ranking1) != list or type(ranking2) != list:
18             raise ValueError("Ranking powinien by list !!!")
19
20         if len(ranking1) != len(ranking2):
21             raise ValueError("Rankingi nie maj takiego samego wymiaru !!!")
22         ###
23         sum = 0
24         for i in range(len(ranking1)):
25             nr = ranking1[i]
26             idx_2 = ranking2.index(nr)
27             sum += ((weights_function(i)+weights_function(idx_2))/2) * abs(i-idx_2)
28         return sum

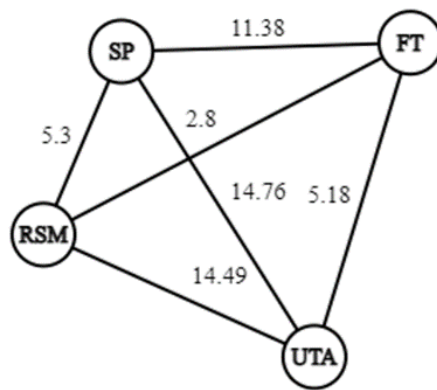
```



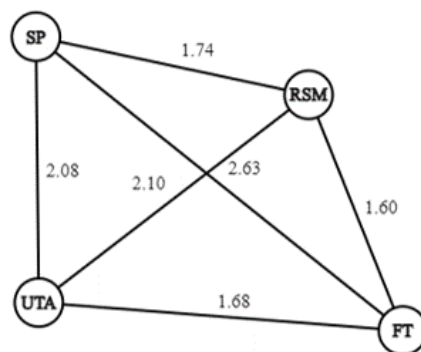
Rysunek 4: Metryki odległości poszczególnych rankingów dla każdej z metod dla wszystkich kierunków. przy założeniu maksymalizacji: procent zdawalności, ocena absolwentów, personalna ocena sylabusu oraz minimalizacji kryteriów: liczba semestrów, próg rekrutacji. Przy założeniu wag dla Fuzzy Topsis =  $[1,1,1,1,1]$ .



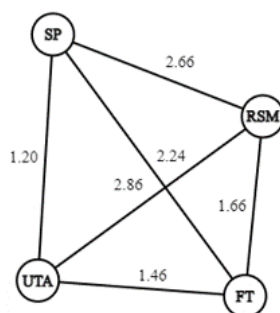
Rysunek 5: Metryki odległości poszczególnych rankingów dla każdej z metod dla kierunków technicznych. przy założeniu maksymalizacji: procent zdawalności, ocena absolwentów, personalna ocena sylabusu oraz minimalizacji kryteriów: liczba semestrów, próg rekrutacji. Przy założeniu wag dla Fuzzy Topsis =  $[1,1,1,1,1]$ .



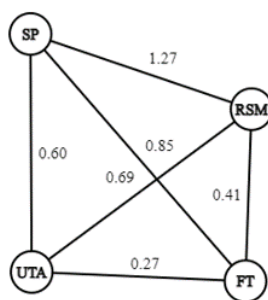
Rysunek 6: Metryki odległości poszczególnych rankingów dla każdej z metod dla kierunków humanistycznych. przy założeniu maksymalizacji: procent zdawalności, ocena absolwentów, personalna ocena sylabusa oraz minimalizacji kryteriów: liczba semestrów, próg rekrutacji. Przy założeniu wag dla Fuzzy Topsis = [1,1,1,1,1].



Rysunek 7: metryki odległości poszczególnych rankingów dla każdej z metod dla kierunków ekonomicznych. przy założeniu maksymalizacji: procent zdawalności, ocena absolwentów, personalna ocena sylabusa oraz minimalizacji kryteriów: liczba semestrów, próg rekrutacji. Przy założeniu wag dla Fuzzy Topsis = [1,1,1,1,1]



Rysunek 8: Metryki odległości poszczególnych rankingów dla każdej z metod dla kierunków lekarskich. przy założeniu maksymalizacji: procent zdawalności, ocena absolwentów, personalna ocena sylabusa oraz minimalizacji kryteriów: liczba semestrów, próg rekrutacji. Przy założeniu wag dla Fuzzy Topsis = [1,1,1,1,1].



Rysunek 9: Metryki odległości poszczególnych rankingów dla każdej z metod dla kierunków ścisłych. przy założeniu maksymalizacji: procent zdawalności, ocena absolwentów, personalna ocena sylabusa oraz minimalizacji kryteriów: liczba semestrów, próg rekrutacji. Przy założeniu wag dla Fuzzy Topsis = [1,1,1,1,1].

## 4 GUI

Widok GUI przedstawia poniższy rysunek:

Wybierz rodzaj kierunku: Wszystkie

Podaj kryteria (min/max)

| Kryterium             | Wartość |
|-----------------------|---------|
| Procent zdawalności   | min     |
| Ocena absolwentów     | min     |
| Własna ocena sylabusa | min     |
| Ilość semestrów       | min     |

Podaj wagi (1 - 9)

|   |   |   |   |   |
|---|---|---|---|---|
| 1 | 4 | 7 | 1 | 1 |
|---|---|---|---|---|

Wybierz algorytm: FUZZY\_TOPSIS

Stwórz ranking

Alternatywy z kryteriami

| Miasto | Nazwa kierunku                  | Nazwa uczelni        | Procent zdawalności | Ocena |
|--------|---------------------------------|----------------------|---------------------|-------|
| Kraków | Automatyka i Robotyka           | Akademia Górniczo Hu | 92.04846359         |       |
| Kraków | Kulturoznawstwo                 | AKADEMIA IGNATIANI   | 84.81862478         |       |
| Kraków | Recykling i Metalurgia          | Akademia Górniczo Hu | 92.786748           |       |
| Kraków | Inżynieria Mechatroniczna       | Uniwersytet Rolniczy | 96.69733882         |       |
| Kraków | Etologia i Psychologia Zwierząt | Uniwersytet Rolniczy | 95.27443724         |       |
| Kraków | Zootechnika                     | Uniwersytet Rolniczy | 80.00627311         |       |

Ranking

| Nazwa kierunku                   | Nazwa uczelni                   | Miasto   | Score   |
|----------------------------------|---------------------------------|----------|---------|
| Automatyka i Robotyka            | Politechnika Rzeszowska         | Rzeszów  | 0.82737 |
| Socjologia                       | Uniwersytet Śląski              | Katowice | 0.79159 |
| Region karpacki: etnolingwistyka | Uniwersytet Jagielloński        | Kraków   | 0.74340 |
| Iberystyka                       | Uniwersytet Gdański             | Gdańsk   | 0.74323 |
| Fizjoterapia                     | Warszawski Uniwersytet Medyczny | Warszawa | 0.68785 |
| Logistyka                        | Politechnika Śląska             | Katowice | 0.67388 |

Wybierz algorytm do porównania

FUZZY\_TOPSIS RSM

Porównaj rankingi

29\_504994947261284

Rysunek 10: GUI

Użytkownik aplikacji wybiera rodzaj kierunku i określa, czy następuje minimalizacja czy maksymalizacja rozważanych kryteriów. Następnie każdemu z nich przypisuje wagi zgodnie z indywidualnymi preferencjami i wybiera algorytm, na podstawie którego tworzony jest ranking. Decydent ma również możliwość przeglądania dostępnych w bazie alternatyw oraz może porównać ze sobą wyniki uzyskane za pomocą dwóch wybranych metod, przy czym należy zwrócić uwagę na fakt, że im większa wartość liczbowo uzyskanego wyniku, tym większa jest odległość między uzyskanymi rankingami.

## 5 Podsumowanie

Podjęcie decyzji dotyczących kierunku jest problemem dość trudnym bez wcześniejszej wiedzy na temat algorytmów wspomagania decyzji. Pomocna może okazać się nasza aplikacja pozwalająca ułatwić wybór kierunku studiów na podstawie jednej z dogodnych dla nas metod oceny. Co ważne jest możliwość uściślenia kryteriów do tych które są dla nas istotne.

Aplikacja posiada potencjał na dalszy rozwój np poprzez podpięcie API z takich stron jak perspektywy.pl w celu aktualizowania bazy danych na bieżąco. Dodatkowe metody oceny mogłyby być również przydatne dla ewentualnych przyszłych użytkowników, a z implementacji niektórych metod można by napisać niejedną pracę naukową.

Podczas projektu nie dało się uniknąć trudności związanych z tematyką problemu. Ustalenia co do istotności parametrów na podstawie których podejmowano decyzje sprawiły małe problemy. Podstawową kwestią w projekcie było uzyskanie danych co dla np danych oceny musiało się odbywać na wartościach przybliżonych podobnie jak z procentem zdawalności co może wprowadzać pewne nieścisłości w finalnych wynikach. Łączenie wyników i interfejsów mogło sprawiać trudności jednak dzięki optymalnemu projektowaniu aplikacji uniknęliśmy tego typu problemów.

## 6 Podział pracy

Podział pracy ze względu na implementowane metody:

| <b>Zadania:</b>  | <b>Odpowiedzialne osoby:</b> |
|--|------------------------------|
| Wybór problemu, utworzenie bazy i GUI, redakcja wniosków   | Wszyscy                      |
| Koordinacja pracy i edycja sprawozdania  | S2,S3                        |
| Implementacja metody Fuzzy Topsis wraz z utworzeniem rankingu i wyborem rozwiązania kompromisowego | S3,S2                        |
| Implementacja metody RSM wraz z utworzeniem rankingu i wyborem rozwiązania kompromisowego          | S4,S5                        |
| Implementacja metody SP wraz z utworzeniem rankingu i wyborem rozwiązania kompromisowego           | S1,S2,S3,S4                  |
| Implementacja metody UTA wraz z utworzeniem rankingu i wyborem rozwiązania kompromisowego          | S6,S7,S8,S1                  |
| Implementacja metody porównywania rankingów  | S2,S3,S4                     |
| Przygotowanie prezentacji  | wszyscy                      |

Wkład własny włożony w przygotowanie raportu przedstawia poniższa tabela:

| <b>Rozdział</b>      | <b>Odpowiedzialna osoba:</b> |
|----------------------|------------------------------|
| Opis projektu        | S2,S3,S1,S4                  |
| Metoda Fuzzy Topsis  | S2,S3                        |
| Metoda RSM           | S4,S5                        |
| Metoda SP            | S1,S3                        |
| Metoda UTA           | S6,S7                        |
| Porównanie rankingów | S3,S2                        |
| GUI                  | S1,S7                        |
| Podsumowanie         | S8                           |

Oznaczenia symboli:

S1-Michał Pajor  
S2-Sylwia Michalska  
S3-Michał Michniak  
S4-Jakub Pacoń  
S5-Marcin Ryznar  
S6-Adam Pękala  
S7-Rafał Maciasz  
S8-Paweł Radzik

## 7 Bibliografia

Materiały wykorzystane podczas realizacji zadania:

- Andrzej M.J. Skulimowski - Decision support systems based on reference sets



- Paweł KONOPKA, Ewa ROSZKOWSKA - ZASTOSOWANIE METODY UTA DO WSPOMAGANIA PODEJMOWANIA DECYZJI O FINANSOWANIU STARTUPÓW DZIAŁALNOŚCI GOSPODARCZEJ
- Marcin Szelaąg - opracowanie Metody UTA
- <https://theory.stanford.edu/~sergei/slides/www10-metrics.pdf> - Spearman's Footrule, weighted rankings
- <https://www.sciencedirect.com/science/article/pii/S187705091631273X> - Fuzzy topsis