# Systemy Wspomagania Decyzji
## Optymalizacja wielokryterialna

25 stycznia 2023

## Spis treści


**1 Opis projektu** **2**

**2 Implementacja metod** **2**

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

**3 Porównanie rankingów** **20**

**4 GUI** **23**

**5 Podsumowanie** **24**

**6 Podział pracy** **24**

**7 Bibliografia** **25**

# 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 syllabusa,
- 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 sylabusa 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 $\mathbb{R} \to [0,1]$. W zaimplementowanej metodzie przyjęto definicję liczby rozmytej scharakteryzowanej rzez trzy wartości: a,b oraz c :

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

Zdefiniowano także operacje przeprowadzane na liczbach rozmytych:

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

### 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) |

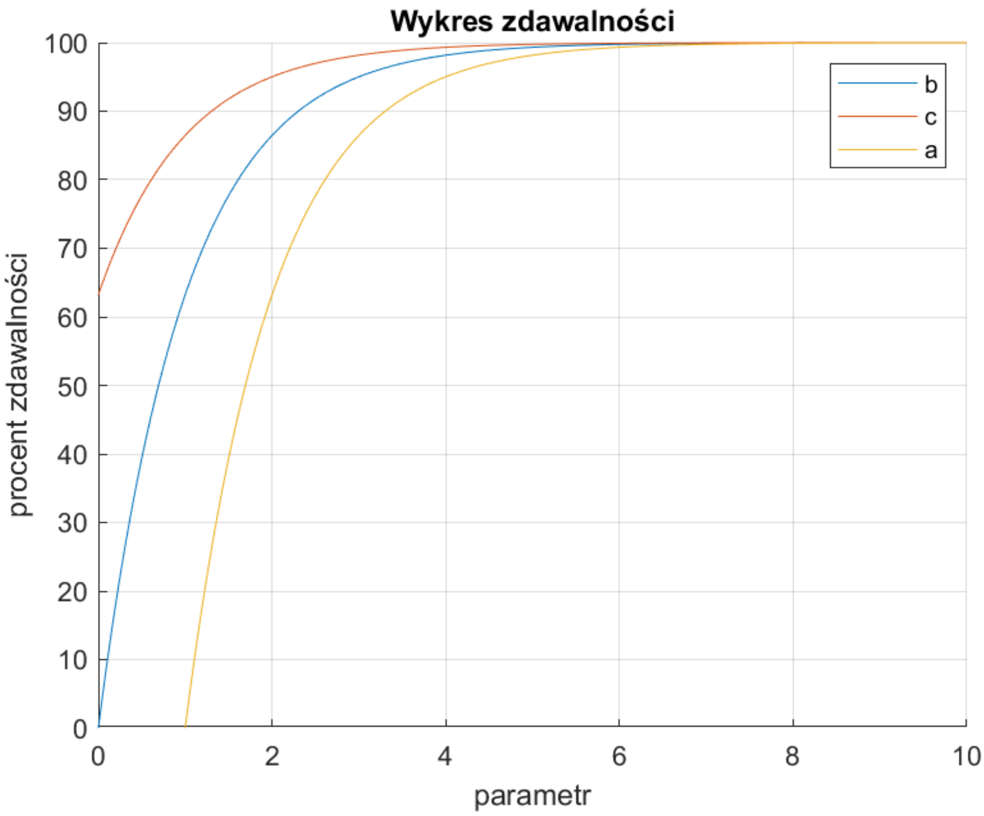
Reprezentacja kryterium *liczba semestrów* została przedstawiona w następujący sposób:

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

wybór 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:

$$\begin{cases} x = -ln(1 - \frac{prog\ zdawalnosci}{100}) \\ a = 1 - e^{-(x-1)} \\ b = 1 - e^{-(x)} \\ a = 1 - e^{-(x+1)} \end{cases}$$




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 $\widetilde{X}$:

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

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

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

- maksymalizacja:

$$\widetilde{r}_{ij} = \left(\frac{a_{ij}}{c_j^*}, \frac{b_{ij}}{c_j^*}, \frac{c_{ij}}{c_j^*}\right) \quad gdzie \quad c_j^* = \underset{i}{max}\,(c_{ij})$$

- minimalizacja:

$$\widetilde{r}_{ij} = \left(\frac{a_j^-}{c_{ij}}, \frac{a_j^-}{b_{ij}}, \frac{a_j^-}{a_{ij}}\right) \quad gdzie \quad a_j^- = \underset{i}{min}\,(a_{ij})$$

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

$$A^* = (\widetilde{v}_1^*, \widetilde{v}_2^*, ..., \widetilde{v}_n^*) \quad gdzie \quad \widetilde{v}_j^* = \underset{i}{max}\,(v_{ij3})$$

$$A^- = (\widetilde{v}_1^-, \widetilde{v}_2^-, ..., \widetilde{v}_n^-) \quad gdzie \quad \widetilde{v}_j^- = \underset{i}{min}\,(v_{ij1})$$

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

```python
import numpy as np

class FuzzyNumb:
    def __init__(self, a : float, b : float, c : float) -> None:
        self.a = a
        self.b = b
        self.c = c
        pass

    def d(self,other) -> float:
        """
        return
            distance between two trangular fuzzy numbers
        """
        return np.sqrt(1/3*((self.a-other.a)**2+(self.b-other.b)**2+(self.c-other.c)**2))

    def normalised_benefit_criteria(self, c_star):
        """
        return
            normalized over benefit criteria triangular fuzzy number
        """
        return FuzzyNumb(self.a/c_star,self.b/c_star,self.c/c_star)

    def normalised_cost_criteria(self, a_minus):
        """
        return
            normalized over cost criteria triangular fuzzy number
        """
        return FuzzyNumb(a_minus/self.c,a_minus/self.b,a_minus/self.a)

    def __mul__(self,other):
        return FuzzyNumb(self.a*other.a, self.b*other.b, self.c*other.c)

    def __gt__(self,other):
        if self.c>other.c:
            return True
        elif self.c<other.c:
            return False
        elif self.b>other.b:
            return True
        elif self.b<other.b:
            return False
        elif self.a>other.a:
            return True
```

```python
        else:
            return False

    def __lt__(self,other):
        if self.a<other.a:
            return True
        elif self.a>other.a:
            return False
        elif self.b<other.b:
            return True
        elif self.b>other.b:
            return False
        elif self.c<other.c:
            return True
        else:
            return False

    def __repr__(self):
        return f"({self.a:.3f},{self.b:.3f},{self.c:.3f})"
```

```python
from FuzzyNum import FuzzyNumb
from typing import List
import numpy as np

def normalize(fuz_num : FuzzyNumb, j, max_cols : List[float], min_cols : List[float] , cost_ : List[str]):
    if cost_[j] == "min":
        return fuz_num.normalised_cost_criteria(min_cols[j])
    else:
        return fuz_num.normalised_benefit_criteria(max_cols[j])

def cc(d_minus : np.array,d_star : np.array):
    ## step 7 and 8
    cc_i=[(i,d_minus[i]/(d_minus[i]+d_star[i])) for i in range(len(d_minus))]
    cc_i.sort(key=lambda x: x[1],reverse=True)
    return cc_i

def max_fuzz(V : List[FuzzyNumb]):
    max_fuz = FuzzyNumb(-np.inf,-np.inf,-np.inf)
    for i in range(len(V)):
        if V[i]>max_fuz:
            max_fuz = V[i]
    return max_fuz

def min_fuzz(V : List[FuzzyNumb]):
    min_fuz = FuzzyNumb(np.inf, np.inf, np.inf)
    for i in range(len(V)):
        if V[i]<min_fuz:
            min_fuz = V[i]
    return min_fuz

def fuzzy_topsis_(decision_matrix : np.ndarray, weights : List[FuzzyNumb], cost_ : List[str]):
    ## step 3
    max_cols = [max(list(decision_matrix[:,i]),key = lambda x: x.c).c for i in range(len(decision_matrix[0]))]
    min_cols = [min(list(decision_matrix[:,i]),key = lambda x: x.a).a for i in range(len(decision_matrix[0]))]
    r : List[List[FuzzyNumb]] = [[normalize(decision_matrix[i,j],j,max_cols,min_cols, cost_) for j in range(len(decision_matrix[0]))] for i in range(len(decision_matrix))]
    ## step 4
    V : List[List[FuzzyNumb]] = [[r[i][j]*weights[j] for j in range(len(r[0]))] for i in range(len(r))]
    ## step 5 compute ideal solution
    A_ideal = [max_fuzz(np.array(V)[:,i]) for i in range(len(V[0]))]
    A_antyideal = [min_fuzz(np.array(V)[:,i]) for i in range(len(V[0]))]
    ## step 6
    D_star : List[List[FuzzyNumb]] = [np.sum(np.array([V[i][j].d(A_ideal[j]) for j in range(len(V[0]))])) for i in range(len(V))]
    D_minus : List[List[FuzzyNumb]] = [np.sum(np.array([V[i][j].d(A_antyideal[j]) for j in range(len(V[0]))])) for i in range(len(V))]
    ## step 7
    return cc(np.array(D_minus),np.array(D_star))

```

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

```
import numpy as np
from FuzzyNum import *
from fuzzy_topsis import fuzzy_topsis
from typing import List
import pandas as pd

def translate_to_fuzzy_preferences(grade : int):
    if grade == 1:
        return FuzzyNumb(1,1,3)
    elif grade == 2:
        return FuzzyNumb(1,2,4)
    elif grade == 3:
        return FuzzyNumb(1,3,5)
    elif grade == 4:
        return FuzzyNumb(2,4,6)
    elif grade == 5:
        return FuzzyNumb(3,5,7)
    elif grade == 6:
        return FuzzyNumb(4,6,8)
    elif grade == 7:
        return FuzzyNumb(5,7,9)
    elif grade == 8:
        return FuzzyNumb(6,8,9)
    elif grade == 9:
        return FuzzyNumb(7,9,9)
    else:
        raise ValueError("Nieprawid owa warto  wagi")

def one_to_five_translation(grade : int):
    if grade == 1:
        return FuzzyNumb(1,1,3)
    elif grade == 2:
        return FuzzyNumb(1,3,5)
    elif grade == 3:
        return FuzzyNumb(3,5,7)
    elif grade == 4:
        return FuzzyNumb(7,9,10)
    elif grade == 5:
        return FuzzyNumb(9,10,10)
    else:
        raise ValueError(f"ocena poza skal : grade = {grade}")



def translate_value(data_frame) -> List[List[FuzzyNumb]]:
    df : pd.DataFrame = data_frame[["Procent zdawalno ci","Ocena absolwent w","W asna
    ocena sylabusa","Ilo  semestr w","Pr g rekrutacji"]]
    df = df.to_numpy()
    D : List[List[FuzzyNumb]] = []
    for j in range(df.shape[0]):
        D.append([])
        for n,i in enumerate(["Procent zdawalno ci","Ocena absolwent w","W asna ocena
    sylabusa","Ilo  semestr w","Pr g rekrutacji"]):
            if i == "Procent zdawalno ci":
                x = -np.log(1-(df[j,n]/100))
                D[j].append(FuzzyNumb(100*max(1-np.exp(-(x-1)),0),df[j,n],100*(1-np.exp
    (-(x+1)))))
            elif i == "Ocena absolwent w":
                D[j].append(one_to_five_translation(df[j,n]))
            elif i == "W asna ocena sylabusa":
                D[j].append(one_to_five_translation(df[j,n]))
            elif i == "Ilo  semestr w":
                D[j].append(FuzzyNumb(df[j,n],df[j,n],df[j,n]+2))
            elif i == "Pr g rekrutacji":
                D[j].append(FuzzyNumb(max(df[j,n]-10,0),df[j,n],min(df[j,n]+10,100)))
    return D
    pass



```

```python
def fuzzy_topsis_do_gui(weights : List[int],data_frame : pd.DataFrame,max_min : List[str]):
    """
    args:
        weights : List[int] - list that contains nombers 1-9 where 9 is Absolutely important and 1 is Equally important
        data_frame : df.DataFrame - data from database
        max_min : List[str] - list of strings for each collumn: "max" for profit , "min" for cost
    return:
        List[Tuple[int,float]] - ranking of decision first is an index of decision second is a scoring function.
    """
    weights = [translate_to_fuzzy_preferences(i) for i in weights]
    D = translate_value(data_frame)
    if len(weights) != len(max_min):
        raise ValueError(f"Nie zgadzaj  si  wymiary d ugo   wag = {len(weights)}, d ugo   data_frame = {data_frame.shape[1]}, d ugo   max_min = {len(max_min)}")
    return fuzzy_topsis(D,weights,max_min)
```

## 2.2 Metoda RSM

### 2.2.1 Opis metody

Poszukiwane jest rozwiązanie problemów optymalizacji wielokryterialnej typu:

$$[(F_1, \ldots, F_N) : U \to E] \to min(\theta)$$

gdzie U i E oznaczają odpowiednio przestrzeń decyzji i przestrzeń kryteriów (tj. zbiór, w którym wartości przyjmuje funkcja F), F = ($F_1, F_2, ....F_n$) 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:

$$v : E \to IR$$

Funkcja v jest silnie monotonicznie rosnąca, tzn:

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

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

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

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 rozwiązania 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, formułowania 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:

$$\forall x, y \subset E(v^{(x)} < v^{(y)} \Rightarrow x \leqslant_\theta y \; lub \; x \nsim y)$$

gdzie $v^{(x)}$ jest oszacowaniem użyteczności v dla x, a $x \nsim y$ 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:

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

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

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.

$$v_i = \mid A_{1i} - A_{0i} \mid$$
$$w_i = \frac{v_i}{\sum\limits_j v_j}$$

Korzystając z metryki euklidesowej funkcja skoringowa opisana jest wzorem :

$$score_i = w_{jk} + \frac{\sqrt{{B_i}^2 - {A1_k}^2}}{\sqrt{{B_i}^2 - {A1_j}^2} + \sqrt{{B_i}^2 - {A1_k}^2}}$$

gdzie B jest to macierz punktów dopuszczalnych, a A1 punktów docelowych.

### 2.2.3 Implementacja

```python
import numpy as np
from typing import Callable,List
def zdominowane(decision_matrix : np.ndarray, min_max_criterial_funct : List[Callable[[np.ndarray],float]]):
    decision_matrix_copy = decision_matrix.copy()
    decision_matrix = decision_matrix[:][:]
    lst = []
    lst2 = []
    for i in range(len(decision_matrix)):
        lst.append(list(decision_matrix[i][:]))
        lst2.append(list(decision_matrix_copy[i][:]))
    decision_matrix = np.array(lst)
    decision_matrix_copy = np.array(lst2)
    lstzd = []
    lstnzd = []
    for i in range(len(decision_matrix)):
        for j in range(len(decision_matrix)):
            if i == j:
                continue
```

```
            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]
     for k in range(len(decision_matrix[0]))])
            if temp.any():
                pass
            else:
                break
        else:
            lstnzd.append(list(decision_matrix_copy[i]))
    lst_copy=lst.copy()
    for i in range(len(lst_copy)):
            if lst_copy[i] not in lstnzd:
                lstzd.append(lst_copy[i])
    return  lstnzd,lstzd
```

```
from typing import List,Callable
from RSM.nzd_zd import zdominowane
import numpy as np
from RSM.waga_metryka import wage, metric

def RSM(df, additional_params):
    """
    RSM metoda zbior w odniesienia

    args:
        A - zbi r punkt w odniesienia
        C - zbi r punkt w dopuszczalnych

    return
        lst_skoring - punkty skoringowe zbioru B
        lst - posortowane punkty ze zbioru B wzgl dem punkt w skoringowych
    """
    min_max = [np.min if i == 'min' else np.max for i in additional_params]

    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

    B = df[df.columns[3: 8]].values

    A0, rest = zdominowane(A, min_max)
    A1, rest = zdominowane(rest, min_max)

    if len(A1) == 0:
        raise ValueError("Nieprawid owe A1")

    wages = [[wage(A0[i], A1[j]) for j in range(len(A1))] for i in range(len(A0))]

    wages = wages/np.sum(wages)
    score = []
    for i in range(len(B)):
        score.append(0)
        for j in range(len(wages)):
            for k in range(len(wages[0])):
                score[i] += wages[j,k] * metric(B[i],A1[k]) / (metric(B[i],A0[j]) +
    metric(B[i],A1[k]))

    df['RSM_score'] = score
    return df
```

```
from typing import List,Callable
import numpy as np

def wage(a0 : List[float],a1: List[float]):
    v = 1
    for n,i in enumerate(a0):
        v *= np.abs(a1[n] - i)
    return v

def metric(a : List[float],b : List[float]):
    d = np.dot(np.array(a)-np.array(b),np.array(a)-np.array(b))
    return np.sqrt(d)
```

## 2.3 Metoda SP

### 2.3.1 Opis metody

Metoda SP-CS skupia się na poszukiwaniu przy 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,

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

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$$przedstawiajzalenociponiej$ $d_{11} = \frac{u_1^1 - l_1^1}{2}$

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

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

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 quot do tego rzutowanego punktu. Tą procedurę powtarza się dla każdej pary punktu ze zbioru idealnego i status quot. Następnie dane odległości są sumowane względem wag które wyznaczane są jako hiperobiętości odpowiednich prostopadłościanów (hiperścianów) wyznaczanych przez punkty idealne i status quot które definiują naprzeciwległe wierzchołki tego hiperprostopadłościanu przez sumę wszystkich hiperobiętości w dla rozpatrywanych punktów idealnych i status quot.

### 2.3.2 Implementacja

```
from typing import List,Callable
import numpy as np
import pandas as pd
from RSM.nzd_zd import zdominowane

def cum_count_path(woron_points,metrics = None):
    if metrics == None:
        metrics = lambda x,y: np.sqrt(np.dot(y-x, y-x))
    paths = [0]
    for i in range(1,len(woron_points)):
        paths.append(paths[i-1]+metrics(woron_points[i-1],woron_points[i]))
    return np.array(paths)


def costam(w1:np.ndarray,w2: np.ndarray,w3: np.ndarray):
    w21=w2-w1
    w21_norm= w21/np.linalg.norm(w21)
    w3_p=w3-w1
    dot_p=np.dot(w21_norm,w3_p)
    new=dot_p*w21_norm
    if np.linalg.norm(w21)>np.linalg.norm(new):
```

```python
        return np.linalg.norm(new), np.linalg.norm(w3_p - new)
    return -1,-1


def zwrot_wagi(Ide,Aide):
    waga=0
    for i in range(len(Ide)):
        waga=waga+volume(Ide[i],Aide[i])
    return waga


def volume(pkt1,pkt2):
    l=[]
    for j in range(len(pkt1)):
        if pkt2[j]>pkt1[j]:
            l.append(pkt2[j]-pkt1[j])
        else:
            l.append(pkt1[j]-pkt2[j])
    V=1
    for i in range(len(l)):
        V=V*l[i]
    return V


def czy_w_obszarze(u,pkt1,pkt2):
    isTRUE=[]
    for i in range(len(u)):
        isTRUE.append(pkt1[i]<=u[i]<=pkt2[i] or pkt2[i]<=u[i]<=pkt1[i])
    for j in range(len(isTRUE)):
        if (isTRUE[j]==False):
            return -1
    return volume(pkt1,pkt2)


def woronoj(pkt_1 : np.ndarray,pkt_2 : np.ndarray):
    N = len(pkt_1)
    pkt_2 = pkt_2-pkt_1
    pkt_1_temp = pkt_1
    pkt_1 = np.zeros(N)
    woronoj_points = [np.array([0,0,0]) for i in range(2*N)]
    woronoj_points[0] = pkt_1
    woronoj_points[-1] = pkt_2
    size_of_shift = np.abs(pkt_2 - pkt_1)/2
    sign_of_shift = np.sign(pkt_2 - pkt_1)
    mask_fixed = np.zeros(N)
    dimentions = np.ones(len(pkt_1))
    for i in range(1,N):
        woronoj_points[i] = woronoj_points[0] + np.min(size_of_shift) * sign_of_shift + mask_fixed
        woronoj_points[2*N-1-i] = woronoj_points[-1] + -1*np.min(size_of_shift) * sign_of_shift - mask_fixed
        idx = np.argmin(size_of_shift)
        mask_fixed[idx] = woronoj_points[i][idx]
        size_of_shift[idx] = np.inf
        sign_of_shift[idx] = 0
        dimentions[idx] = 0
    return np.array(woronoj_points) + pkt_1_temp


def norm(A : List[List[float]], C :  List[List[float]]):
    A1=np.array(A)
    C1=np.array(C)
    normalizedA=(A1-np.min(A1))/(np.max(A1)-np.min(A1))
    normalizedC=(C1-np.min(C1))/(np.max(C1)-np.min(C1))
    return normalizedA,normalizedC


def SPCS(idealny : List[np.ndarray], antyidealny : List[np.ndarray], punkty : List[np.ndarray]):
    scoring = []
    waga = zwrot_wagi(idealny,antyidealny)
    for pkt in range(len(punkty)):
        scoring.append(0)

```

```
        for ide in range(len(idealny)):
            for anty in range(len(antyidealny)):
                minimal = np.inf
                d_path = 0
                w = czy_w_obszarze(punkty[pkt],idealny[ide],antyidealny[anty])
                if w == -1:
                    continue
                woron_point = woronoj(antyidealny[anty],idealny[ide])
                cum_path = cum_count_path(woron_point)
                for i in range(1,len(woron_point)):
                    d,metric = costam(woron_point[i-1],woron_point[i],punkty[pkt])
                    if d == -1:
                        continue
                    if minimal>metric:
                        minimal = metric
                        d_path = cum_path[i-1]+d
                ### sprawdzanie punkt w woronoja
                for i in range(1,len(woron_point)):
                    metric = np.linalg.norm(punkty[pkt]-woron_point[i])
                    if minimal>metric:
                        minimal = metric
                        d_path = cum_path[i]
                ###
                scoring[pkt] += w*d_path
        if waga == 0:
            scoring[pkt] = 0
        else:
            scoring[pkt] = scoring[pkt]/waga
    return scoring

def gui_spcs(df, additional_params):
    min_max = ['max','max','max']
    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]]
    A0, rest = zdominowane(A, min_max)
    A1, rest = zdominowane(rest, min_max)
    df_data = df[df.columns[3:6]]
    num = df_data.to_numpy()

    df['SAFETY_PRINCIPAL_score'] = SPCS(np.array(A0),np.array(A1),num)
    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 cząstkowych.

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

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 dezagregarcji-agregacji. Przyjmujemy odpowiednie oznaczenia:

- X = W1,W2,...,Wk - skończony zbiór alternatyw.
- $(X^R)$ = $(W1^*)$,$(W2^*)$,...,$(Wk^*)$ - skończony zbiór alternatyw referencyjnych dla których decydent jest w stanie wyrazić swoje preferencje.
- F = $(f_1)$,...,$(f_n)$ - 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 Wi na kryterium $f_j$
- $u_i(f_i(.))$ - cząstkowa funkcja użyteczności, czyli funkcja wyrażająca preferencje decydenta dotyczące i-tego kryterium.
- U(.) - 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,..,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,...,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) |

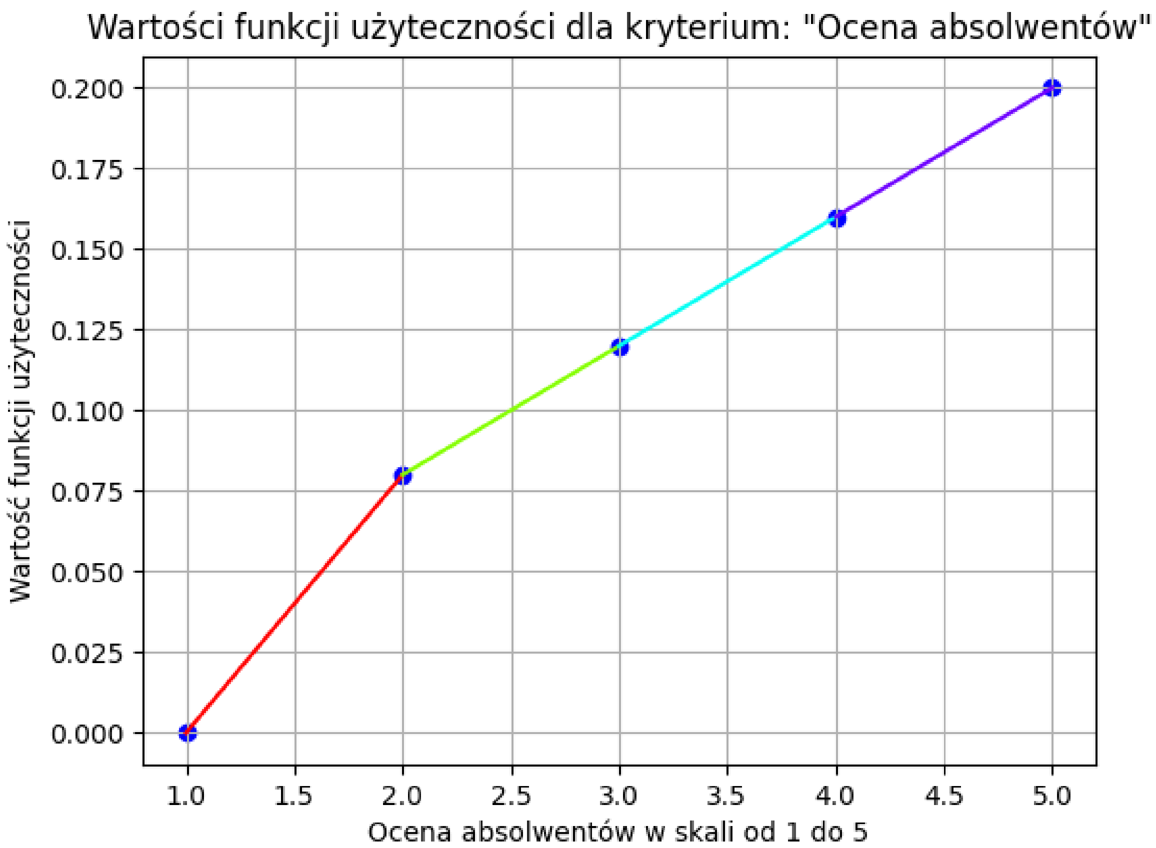
Wartości dla poszczególnych kryteriów są ściśle związane z rodzajem danych w bazie.

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

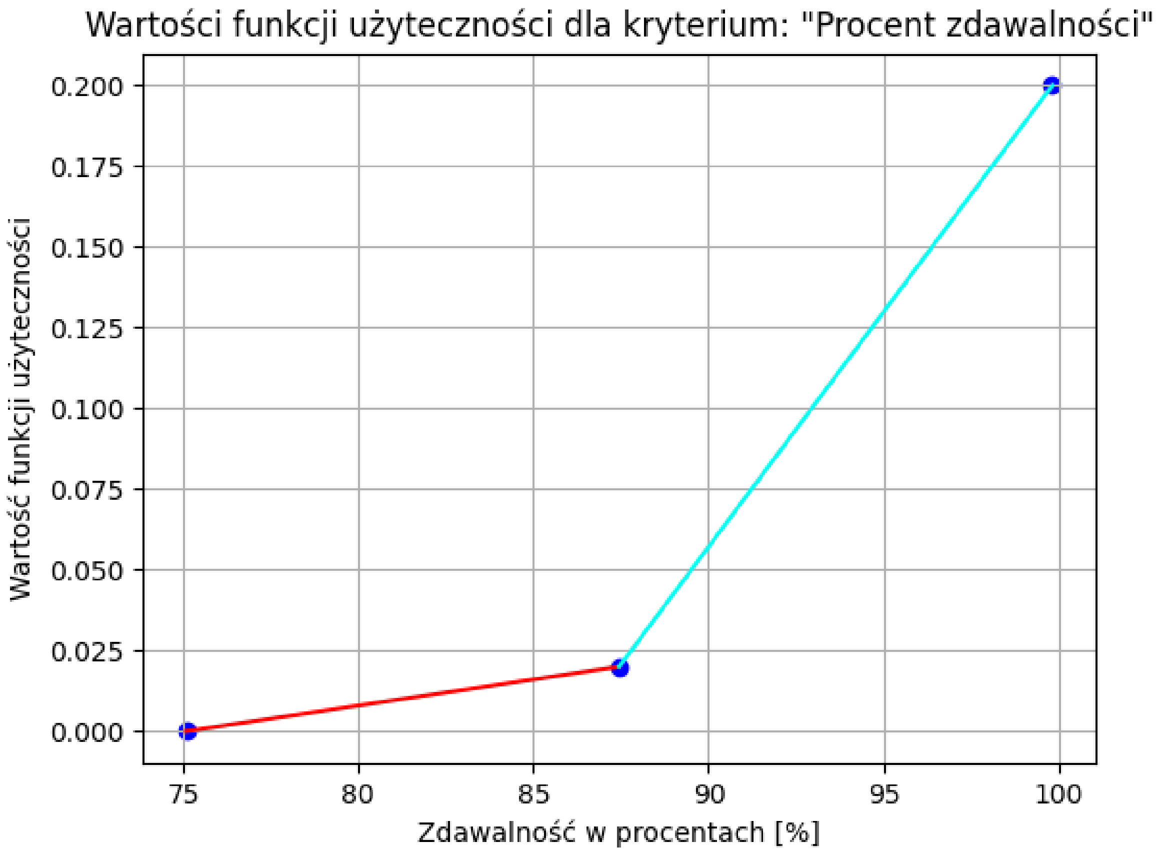
$$y(x) = a * x + b$$

  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

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

def get_min_max(inputPoints: np.ndarray):
    x, y = inputPoints.shape
    min = 100000* np.ones(y)
    max = np.zeros(y)
    for i in range(x): # iteruj po punktach
        for j in range(y): # iteruj po parametrach punktu
            a = inputPoints[i,j]
            b = min[j]
            if inputPoints[i,j] < min[j]:
                min[j] = inputPoints[i,j]
            if inputPoints[i,j] > max[j]:
                max[j] = inputPoints[i,j]

    return min,max

def split(min,max,partitions,max_or_min,func_utility = None):
    list = []
    alternatives = len(partitions) # ilo  kryteri w
    for i in range(alternatives):
        row = [(0,0)]*(partitions[i]+1)
        list.append(row)

    func_utility_max = 1/alternatives  # maksymalna warto  funkcji u ytecznosci przy danej liczbie kryteri w

    if not func_utility:
```

```
        for i in range(alternatives):
            if max_or_min[i] == 0: # je li minimalizujemy
                list[i][0] = (min[i],func_utility_max) # wpisuje maxymaln  i minimalna warto   kryterium
                list[i][-1] = (max[i],0)
                diff = max[i] - min[i]      # obliczam r  nice miedzy max a min warto ci  kryterium
                compartment = diff/partitions[i]    # wyznaczam przedzia y
                func_utility_other = func_utility_max/partitions[i] # wyznaczam warto f. u yteczno ci dla przedzia  w
                for j in range(1,partitions[i]): # wpisuje warto ci dla konkretnych przedzia  w
                    list[i][j] = (min[i]+ j*compartment,func_utility_max - func_utility_other*j)


            if max_or_min[i] == 1: # je li maksymalizujemy
                list[i][0] = (max[i],func_utility_max)
                list[i][-1] = (min[i],0)
                diff = max[i] - min[i]
                compartment = diff/partitions[i]
                func_utility_other = func_utility_max/partitions[i]
                for j in range(1,partitions[i]):
                    list[i][j] = (max[i] - j*compartment,func_utility_max - func_utility_other*j)
    else:
        for i in range(alternatives):
            if max_or_min[i] == 0: # je li minimalizujemy
                list[i][0] = (min[i],func_utility[i][0]) # wpisuje maksymaln  i minimalna warto   kryterium
                list[i][-1] = (max[i],func_utility[i][-1])
                diff = max[i] - min[i]      # obliczam r  nice miedzy max a min warto ci  kryterium
                compartment = diff/partitions[i]    # wyznaczam przedzia y
                for j in range(1,partitions[i]): # wpisuje warto ci dla konkretnych przedzia  w
                    list[i][j] = (min[i]+ j*compartment,func_utility[i][j])


            if max_or_min[i] == 1: # je li maksymalizujemy
                list[i][0] = (max[i],func_utility[i][0])
                list[i][-1] = (min[i],func_utility[i][-1])
                diff = max[i] - min[i]
                compartment = diff/partitions[i]
                for j in range(1,partitions[i]):
                    list[i][j] = (max[i] - j*compartment,func_utility[i][j])

    return list

def function_value(compartments,max_or_min):
    list = []
    idx = len(compartments) - 1
    for i in range(1,len(compartments)):
        if max_or_min == 1: # maksymalizacja
            a = (compartments[(idx-i+1)][1]-compartments[(idx-i)][1])/(compartments[(idx-i+1)][0]-compartments[(idx-i)][0])
            b = compartments[(idx-i+1)][1]-a*compartments[(idx-i+1)][0]
            list.append([a,b])

        if max_or_min == 0: # minimalizacja
            a = (compartments[i-1][1]-compartments[i][1])/(compartments[i-1][0]-compartments[i][0])
            b = compartments[(i-1)][1]-a*compartments[(i-1)][0]
            list.append([a,b])

    return list

def get_cmap(n, name='hsv'):
    return plt.cm.get_cmap(name, n)

def plot_f_utility(u,compartments,max_or_min): # u - lista wsp  czynnik w a i b dla f. u yteczno ci
    cmap = get_cmap(len(u)+1)
    idx = len(compartments)-1
```

```
    for i in range(len(u)):
        a,b = u[i]
        if max_or_min == 1: # maksymalizacja
            x = np.linspace(compartments[idx-i][0],compartments[idx-i-1][0],100)
            y = a*x+b
            if i == len(u)-1:
                plt.scatter(compartments[0][0],a*compartments[0][0]+b, c = 'blue')
            plt.scatter(compartments[idx-i][0],a*compartments[idx-i][0]+b, c = 'blue')
            plt.plot(x, y, c =cmap(i))

        if max_or_min == 0: # minimalizacja
            x = np.linspace(compartments[i][0],compartments[i+1][0],100)
            y = a*x+b
            if i == len(u)-1:
                plt.scatter(compartments[len(u)][0],a*compartments[len(u)][0]+b, c = 'blue')
            plt.scatter(compartments[i][0],a*compartments[i][0]+b, c = 'blue')
            plt.plot(x, y, c =cmap(i))

    plt.grid()
    plt.show()

def rank(utility_coef, compartments, point):
    score = 0
    for i in range(len(compartments[0])):
        for j in range(len(compartments[i])-1):

            if point[i] <= compartments[i][j][0] and point[i] >= compartments[i][j+1][0]:
                score += utility_coef[i][j][0]*point[i]+utility_coef[i][j][1]

            elif point[i] >= compartments[i][j][0] and point[i] <= compartments[i][j+1][0]:
                score += utility_coef[i][j][0]*point[i]+utility_coef[i][j][1]

    return score
```

```
from uta import *
import numpy as np
import pandas as pd
pd.options.mode.chained_assignment = None  # default='warn'
import matplotlib.pyplot as plt

def uta(df, max_or_min):

    criteria = df[["Procent zdawalno ci","Ocena absolwent w","W asna ocena sylabusa","Ilo   semestr w","Pr g rekrutacji"]]
    column_names = list(criteria.columns)
    criteria.rename(columns = {'Procent zdawalno ci':'PZ','Ocena absolwent w':'OA', 'W asna ocena sylabusa':'WOS',
                               'Ilo   semestr w':'IS','Pr g rekrutacji':'PR'}, inplace = True)
    points =[]
    for index, rows in criteria.iterrows():
        my_list =[rows.PZ, rows.OA, rows.WOS,rows.IS, rows.PR]
        points.append(my_list)

    points = np.array(points)
    min,max = get_min_max(points)
    max_or_min = np.where(max_or_min=='min', 0, 1)

    # Warto ci funkcji u yteczno ci dobrane r cznie, kod umo liwia
    # dobranie funkcji u yteczno ci proporcjonalnie, dla takiego przypadku
    # wsp  czynniki a i b wychodz  takie same dla wszystkich przedzia  w

    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],
                    [0.2,0.17,0.14,0.11,0.08,0.05,0],[0.20,0.10,0]]
    compartments = split(min,max,np.array([2,4,4,7,2]),max_or_min,func_utility)

    # Warto   wsp  czynnik w dla funkcji u yteczno ci w danych przedzia ach
    u = list()
    for i in range(len(compartments)):
        u.append(function_value(compartments[i],max_or_min[i]))

    score = []
```

```
    for point in points:
        score.append(rank(u,compartments,point))

    df['UTA_score'] = score

    # for i in range(len(func_utility)):
    #     plt.title(f'Warto ci funkcji u yteczno ci dla kryterium: {column_names[i]}')
    #     plot_f_utility(u[i],compartments[i],max_or_min[i])

    return df
```

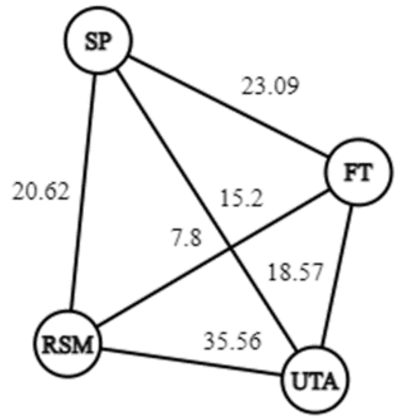
## 3 Porównanie rankingów

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:

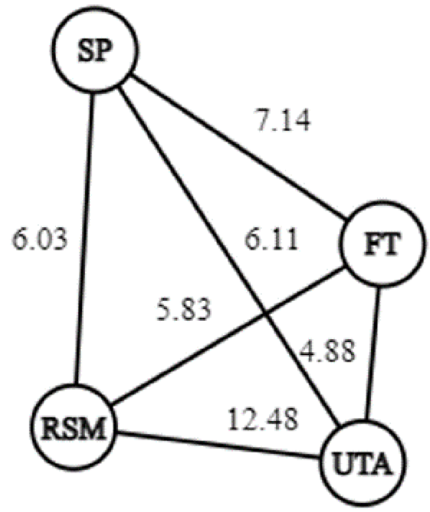
$$F(\sigma) = \sum_i w_i \left| i - \sigma(i) \right|$$

Implementacja:

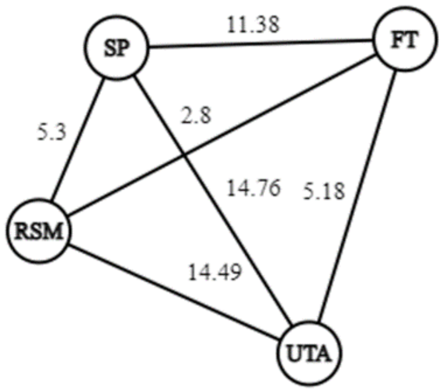
```
from typing import List, Callable
import numpy as np

def Spearman_s_Footrule(ranking1 : List[int], ranking2 : List[int], weights_function : Callable[[int],float] = None, mean = 0, deviation = 8):
    """
    argv:
        ranking1 : List[int] - pierwszy ranking do por wnania ()
        ranking2 : List[int] - drugi ranking do por wnania
        weights_function : Callable[[int],float] - funkcja wagowa
    """
    if weights_function == None:

        weights_function = lambda x: (1/np.sqrt(2*np.pi*deviation))*np.exp(-(x-mean)**2/(2*deviation))
    if type(ranking1) != list or type(ranking2) != list:
        raise ValueError("Ranking powinien by  list  !!!")

    if len(ranking1) != len(ranking2):
        raise ValueError("Rankingi nie maj  takiego samego wymiaru !!!")
    ###
    sum = 0
    for i in range(len(ranking1)):
        nr = ranking1[i]
        idx_2 = ranking2.index(nr)
        sum += ((weights_function(i)+weights_function(idx_2))/2) * abs(i-idx_2)
    return sum
```



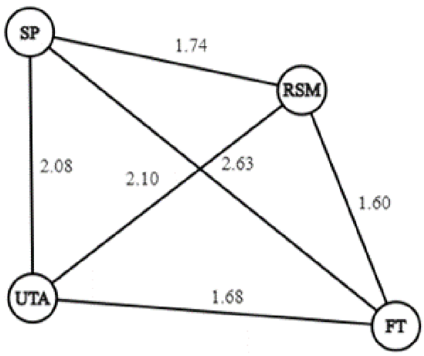

Rysunek 4: Metryki odległości poszczególnych rankingów dla każdej z metod dla wszystkich kierunków. przy założeniu maksymalizacji: procent zdawalności, ocena absolwentów, personalna ocena syllabusa oraz minimalizacji kryteriów: liczba semestrów, próg rekrutacji. Przy założeniu wag dla Fuzzy Topsis =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



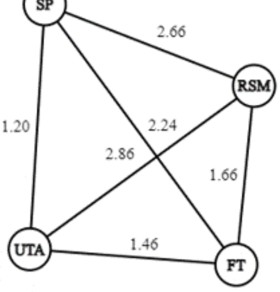

Rysunek 5: Metryki odległości poszczególnych rankingów dla każdej z metod dla kierunków technicznych. przy założeniu maksymalizacji: procent zdawalności, ocena absolwentów, personalna ocena syllabusa oraz minimalizacji kryteriów: liczba semestrów, próg rekrutacji. Przy założeniu wag dla Fuzzy Topsis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



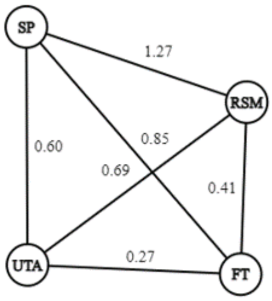

Rysunek 6: Metryki odległości poszczególnych rankingów dla każdej z metod dla kierunków humanistycznych. przy założeniu maksymalizacji: procent zdawalności, ocena absolwentów, personalna ocena syllabusa oraz minimalizacji kryteriów: liczba semestrów, próg rekrutacji.
Przy założeniu wag dla Fuzzy Topsis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



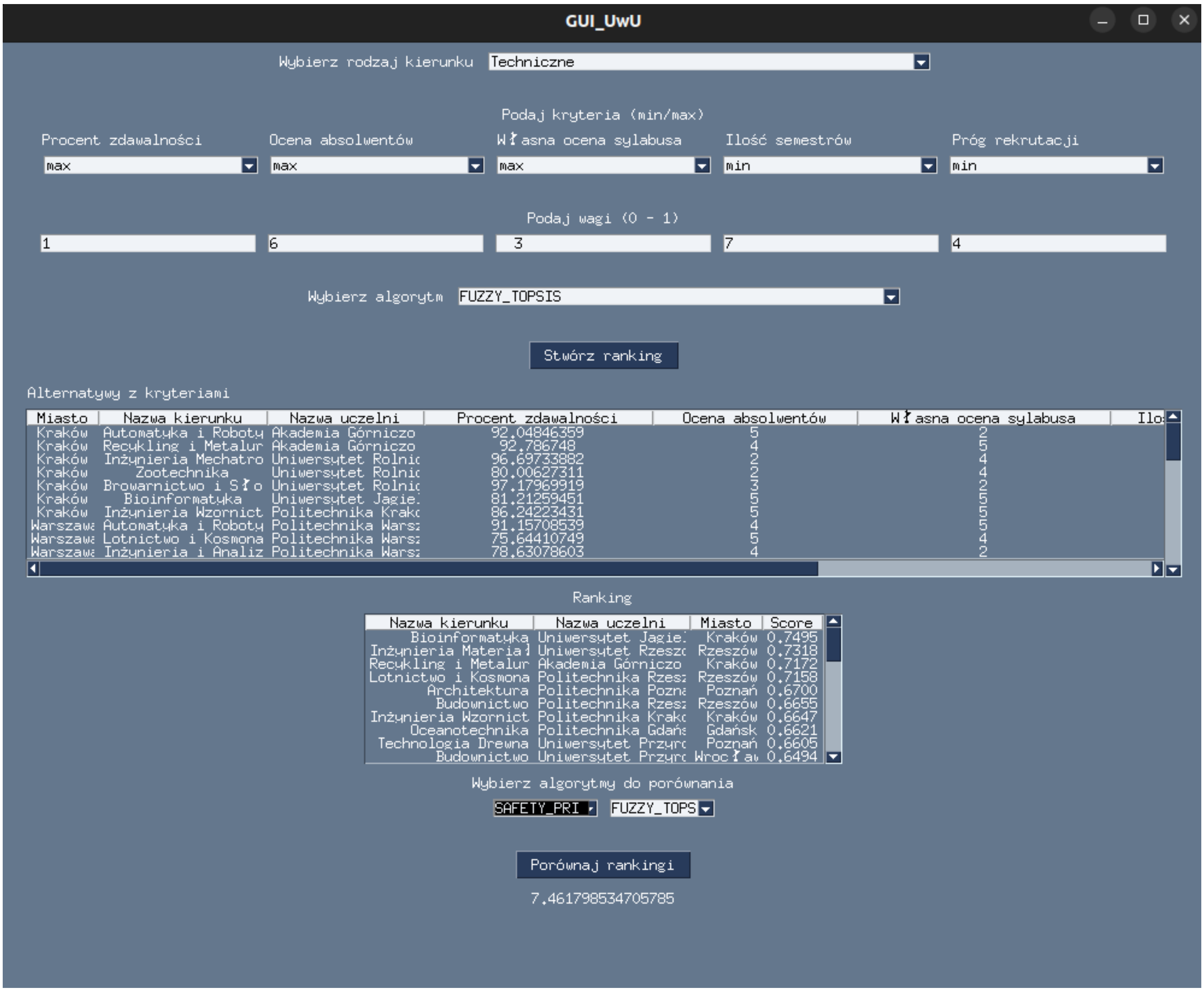

Rysunek 8: Metryki odległości poszczególnych rankingów dla każdej z metod dla kierunków lekarskich. przy założeniu maksymalizacji: procent zdawalności, ocena absolwentów, personalna ocena syllabusa oraz minimalizacji kryteriów: liczba semestrów, próg rekrutacji.
Przy założeniu wag dla Fuzzy Topsis =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




Rysunek 9: Metryki odległości poszczególnych rankingów dla każdej z metod dla kierunków scisłych. przy założeniu maksymalizacji: procent zdawalności, ocena absolwentów, personalna ocena syllabusa 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:




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 liczbowa uzyskanego wyniku, tym większa jest odległość między uzyskanymi rankingami.

# 5 Podsumowanie

Podejmowanie 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 mógłby 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:

| **Zadania:** | **Odpowiedzialne osoby:** |
|---|---|
| Wybór problemu, utworzenie bazy i GUI, redakcja wniosków | Wszyscy |
| Koordynacja 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 |
| Porównanie metod | |
| Przygotowanie prezentacji | |

Wkład własny włożony w przygotowanie raportu przedstawia poniższa tabela:

| **Rozdział** | **Odpowiedzialna osoba:** |
|---|---|
| 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 Szeląg - 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