# Spis treści



# 1 Wstęp

Po uzgodnieniu, zdecydowaliśmy zamodelować i spróbować rozwiązać problem gospodarstwa rolnego.

# 2 Model matematyczny

## 2.1 Opis problemu:

Problem polega na stworzeniu kilkuletniego planu upraw dla niewielkiego gospodarstwa rolnego w zależności od zmiennej kategorii) jakości gleby (w postaci cyfry w zakresie od 1-6) i odległości uprawy od gospodarstwa. Celem będzie maksymalizacja zysków . Zakładamy przy tym że co roku nabywamy nowy materiał siewny.

### 2.1.1 Stałe:

- N - Liczba dostępnych pól uprawnych.
- Y - liczba lat planowania upraw.
- T - stały koszt dojazdu na kilometr
- P - powierzchnia pola uprawnego w hektarach (każde pole ma identyczną powierzchnię)
- $D_i$ - Odległość i-tego pola od gospodarstwa, gdzie i = 1,...,N
- $C_x$ - koszt produkcji danej rośliny na jeden hektar (koszt materiału siewnego, koszt pracy ludzkiej, itp.), gdzie x - nazwa rośliny

- $W_x$- wpływ uprawy na glebę (zależne od uprawianej rośliny)
- $S_x$ - zsumowana ilość dopłat i wszelkich dodatków (w zależności od uprawianej rośliny)
- $G = [g_{qx}]$ - macierz zysków z pola gdzie komórka $g_{qx}$ zawiera zysk z danej rośliny w zależnie od jakości gleby q i uprawianej rośliny $x$.

#### 2.1.2 Zmienne:

- $y$ - Obecny rok, y = 1,...,Y
- $Q = [q_{yi}]_{Y \times N}$ - Macierz klas jakości gleby gdzie komórka $q_{yi}$ zawiera jakość ziemi którą na i-tym polu w roku y.

#### 2.1.3 Postać rozwiązania:

- $X = [x_{yi}]_{Y \times N}$ - macierz decyzyjna o wymiarach $Y \times N$, gdzie komórka $x_{yi}$ zawiera indeks rośliny którą siejemy na i-tym polu w roku y.

#### 2.1.4 Postać funkcji celu:

$$f(X) = \sum_{y=1}^{Y} \sum_{i=1}^{N} G_{q_{yi}x_{yi}} + S_{x_{yi}} - (C_{x_{yi}} * P + D_i * T) \tag{1}$$

$$q_{yi} = q_{(y-1)i} + W_{x_{(y-1)i}} \tag{2}$$

#### 2.1.5 Ograniczenia:

- $0 \leq q_{yi} \leq 100$ Jakość gleby może zmieniać się w zakresie od 0 do 100
- $x_{i-1} \neq x_i$, gdzie $x_k$ nie jest stanem pustym pola

# 3 Implementacja

Naszą implementację zaczeliśmy od zaimplementowania modelu matematycznego w formie funkcji pythonowej

## 3.1 Implementacja klasy jako modelu rozwiązania

```python
def __init__(self, N: int, Y: int, T: float, P: list[float], D: list[float], C: dict[str], W: dict[str],
             G: dict[str], start_quality: list[int]):
    # Inicjalizacja stałych modelu, mogą być różne dla różnych modeli ale nie zmieniają się w trakcie symulacji
    self.fieldNumber = N
    self.yearsNumber = Y
    self.transportCost = T
    self.fieldsSurfacesList = P
    self.distanceMatrix = D
    self.productionCostDict = C
    self.plantInfluenceDict = W
    self.earningsMatrix = G
    self.b = start_quality

    self.curr_year: int = 0  # Aktualny rok
    self.earnings: float = 0  # Ilość naszych pieniedzy w PLN

    # Zerowy rząd pełen początkowych jakości, reszta to zera
    self.Q: list[list[Union[int, None]]] = [self.b] + [[None] * N for _ in range(Y - 1)]
    self.decisionMatrix: list[list[str]] = []
```

Rysunek 1: init funkcji modelu

```python
"""
fieldNumber, N - Liczba dostępnych pól uprawnych.
yearsNumber, Y - Liczba lat planowania upraw.
transportCost, T - Stały koszt dojazdu na kilometr
fieldsSurfacesList, P - Lista powierzchnii poszczególnych pól uprawnych w hektarach
distanceMatrix, D - Lista odległości poszczególnych pól od gospodarstwa
productionCostDict - Słownik kosztów produkcji danej rosliny na jeden hektar (koszt materiału siewnego,
    koszt pracy ludzkiej, itp.)
plantInfluenceDict, W - Słownik wpływów poszczególnych upraw na glebe
earningsMatrix, G - Macierz zysków z danej uprawy.
Q - Macierz jakości gleby na danym polu w danym roku
decisionMatrix, X - Macierz decyzyjna zawieracjąca informacje o wybranych roślinach do uprawy na dany polu w danym
    roku
S - Słownik zamieniający nazwę roślny na przydzielony jej indeks
"""
```

Rysunek 2: oznaczenia

```python
def __reset_variables(self):  # Funkcja resetująca model do stanu początkowego
    self.curr_year = 0
    self.earnings = 0
    self.Q = [self.b] + [[None] * self.fieldNumber for _ in range(self.yearsNumber - 1)]
    self.decisionMatrix = []

LIERO +3
def display_solution(self):
    print('\nRozwiązanie dające dochód {:.2f} zł'.format(self.earnings))
    for row in self.decisionMatrix: print(row)
    print('\nMacierz jakości gleb pól na przestrzeni lat')
    for row in self.Q: print(row)
    print()
```

Rysunek 3: funkcje pomocnicze

```python
def simulate_farm(self, decision_matrix_X: list[list]):  # Funkcja celu
    self.__reset_variables()

    for y_dec in decision_matrix_X:
        self.__simulate_year_pass(y_dec)

    return self.earnings # Ma zwracać rozwiązanie
```

Rysunek 4: symulacja farmy

```python
def solve_greedy(self):  # Algorytm zachłanny - w każdym roku bierze to co da w nim największy zarobek
    self.__reset_variables()
    for y_dec in range(self.yearsNumber):

        dec = []
        for no_field in range(self.fieldNumber):
            pred_qual = self.Q[0][no_field] if y_dec == 0 else self.Q[self.curr_year - 1][no_field] - \
                                                               self.plantInfluenceDict[
                                                                   self.decisionMatrix[self.curr_year - 1][
                                                                       no_field]]
            best_plant, best_income = 'NONE', -math.inf

            for plant in PLANTS:
                if 0 <= (pred_qual - self.plantInfluenceDict[plant]) <= MQ:
                    if plant == 'EMPTY':
                        plant_inc = (self.fieldsSurfacesList[no_field] * self.earningsMatrix[plant][
                            math.ceil(pred_qual)])

                        if y_dec > 0 and plant == self.decisionMatrix[y_dec - 1][no_field]:
                            plant_inc = 0

                    elif y_dec == 0 or (y_dec > 0 and plant != self.decisionMatrix[y_dec - 1][no_field]):
                        plant_inc = (self.fieldsSurfacesList[no_field] * self.earningsMatrix[plant][
                            math.ceil(pred_qual)]) - (
                                        self.productionCostDict[plant] * self.fieldsSurfacesList[no_field] +
                                        self.distanceMatrix[no_field] * self.transportCost)

                    else:
                        plant_inc = -math.inf

                    if plant_inc > best_income:
                        best_plant, best_income = plant, plant_inc

            dec.append(best_plant)

        self.__simulate_year_pass(dec)
    return self.decisionMatrix
```

Rysunek 5:

## 3.2 symulowane wyżarzanie

Nasz problem, na podstawie sugestii pani Profesor postanowiliśmy rozwiązać algorytmem sym. wyżarzania (z ang. simulated anealling). Jest to nasz pierwszy pomysł na rozwiązanie problemu.

- Let $s = s_0$
- For $k = 0$ through $k_{\max}$ (exclusive):
  - $T \leftarrow$ temperature( 1 - $(k+1)/k_{\max}$ )
  - Pick a random neighbour, $s_{\mathrm{new}} \leftarrow \mathrm{neighbour}(s)$
  - If $P(E(s), E(s_{\mathrm{new}}), T) \geq \mathrm{random}(0, 1)$:
    - $s \leftarrow s_{\mathrm{new}}$
- Output: the final state $s$

Rysunek 6:

```
    def simulated_annealing(self, s0: list[list], k_max): # Symulowane wyżarzanie
        self.__reset_variables()

        beast_s = deepcopy(s0)  # Rozwiązanie najlepsze
        s = deepcopy(s0)  # Rozwiązanie początkowe
        for k in range(k_max):
            T = self.__annealing_temp(1 - ((k + 1) / k_max), k_max)
            s_new = self.__annealing_neig(s)
            if self.simulate_farm(s_new) > self.simulate_farm(beast_s):
                beast_s = s_new
            if self.__annealing_P(self.simulate_farm(s), self.simulate_farm(s_new), T) >= random.uniform(0, 1):
                s = deepcopy(s_new)

        return beast_s
```

Rysunek 7: główna metoda algorytmu

```
@staticmethod
def __annealing_temp(inp, k_m): # Funkcja obliczająca temperaturę
    if inp > 0:
        return inp # Najprostszy sposób
    else:
        return 1/k_m
```

Rysunek 8: temp

```python
def __annealing_neig(self, s_inp): # Funkcja wyznaczająca sąsiednie rozwiązanie

    year = random.randrange(self.yearsNumber)
    field = random.randrange(self.fieldNumber)
    curr_plant = s_inp[year][field]
    rand_plant = random.choice([plant for plant in PLANTS if plant != curr_plant])

    s_out = deepcopy(s_inp)
    s_out[year][field] = rand_plant

    # Zabezpieczenie przed wybraniem niedozwolonego rozwiązania
    # if rand_plant != 'EMPTY':
        # if (year > 0 and s_inp[year - 1][field] == rand_plant) or (year < self.years
    try:
        self.simulate_farm(s_out)
    except IndexError:
        # print('Nie spełnia ograniczenia jakości')
        return self.__annealing_neig(s_inp) # Ponowna próba
    except ValueError:
        # print('Nie spełnia ograniczenia innej rośliny w każdym roku')
        return self.__annealing_neig(s_inp) # Ponowna próba

    return s_out
```

Rysunek 9: neig

```python
LIIERO
@staticmethod
def __annealing_P(e, e_dash, temp): # Funkcja akceptująca rozwiązanie, zmodyfikowana bo maksymalizujemy
    if e_dash > e: return 1
    else: return np.exp(((-1)*(e - e_dash))/temp)
```

Rysunek 10: prob

### 3.2.1 wyniki

```python
def main():
    # Dane pozyskane z internetu:
    T = 6 / 15 * 8 * 2 * 7.546  # spalanie na godzine/predkośc*ile razy trzeba pojechać * 2 * cena paliwa

    Cwheat = (87.43 + 19.76) * 2.5 + (87.43 + 27.56) * 2 + (87.43 + 18.80) * 2 + (
            87.43 + 31.9) * 2.5 + 850 * 1.22 + 366 * 1.22 + 540 + 1458 + 1699 + 148 + 34
    Crye = (87.43 + 19.76) * 2 + (87.43 + 27.56) * 1.5 + (87.43 + 18.80) * 2 + (87.43 + 31.9) * 2.5 + 3000
    Cpotato = 2 * ((87.43 + 19.76) * 2.5 + (87.43 + 27.56) * 2 + (87.43 + 18.80) * 2 + (
            87.43 + 31.9) * 2.5) + 17519.3412754
    Ctriticale = (87.43 + 19.76) * 2.5 + (87.43 + 27.56) * 2 + (87.43 + 18.80) * 2 + (87.43 + 31.9) * 2.5 + 3573
    C = {'potato': Cpotato, 'wheat': Cwheat, 'rye': Crye, 'triticale': Ctriticale, 'EMPTY': 0}

    W = {'potato': 5, 'wheat': 8, 'rye': 3, 'triticale': 5, 'EMPTY': -5}

    G = {'potato': [], 'wheat': [], 'rye': [], 'triticale': [], 'EMPTY': []}

    for i in range(MQ):
        G['potato'].append(0 if i < 12 else (math.e ** ((i - 12) / 10) / 6002.91 * 17 + 24) * 710 + 558 + 1761.46)
        G['wheat'].append(0 if i < 35 else (math.e ** ((i - 35) / 10) / 601.845 * 10 + 3.9) * 1490 + 558)
        G['rye'].append((math.e ** (i / 10) / 19930.4 * 6 + 3) * 1150 + 558)
        G['triticale'].append(0 if i < 17 else (math.e ** ((i - 17) / 10) / 3640.95 * 9 + 3) * 1339 + 558)
        G['EMPTY'].append(40)

    # Dane dowolne:
    N = 5
    Y = 5

    P = [1.05, 4.14, 1.69, 1.81, 3.66]
    # P = [random.uniform(1, 6) for _ in range(N)]
    D = [2.08, 7.07, 8.42, 1.37, 6.60]
    # D = [random.uniform(0.1, 10) for _ in range(N)]
    b = [90, 34, 54, 5, 16]  # Poczatkowe jakości gleb na każdym polu
    # b = random.sample(range(0, MQ), N)

    # Symulacja
    f_sim = FarmSimulation(N, Y, T, P, D, C, W, G, b)
```

Rysunek 11: Inicjalizacja klasy farm

```python
# Algorytm zachłanny
print('Rozwiązanie algorytmu zachłannego')
greedy_s = f_sim.solve_greedy()
f_sim.display_solution()

# Wyżarzanie
iterations = 1000  # Maksymalna liczba iteracji

print('Wyżarzanie dla rozwiązania począkowego zachłannego')
sol = f_sim.simulated_annealing(greedy_s, iterations)
f_sim.simulate_farm(sol)
f_sim.display_solution()

# print('Wyżarzanie dla rozwiązania począkowego przykładowego')
# sol = f_sim.simulated_annealing(X, iterations)
# f_sim.simulate_farm(sol)
# f_sim.display_solution()
```

Rysunek 12:

```
Rozwiązanie dające dochód 13390.30 zł
['wheat', 'EMPTY', 'EMPTY', 'rye', 'EMPTY']
['triticale', 'rye', 'wheat', 'EMPTY', 'rye']
['wheat', 'EMPTY', 'EMPTY', 'rye', 'EMPTY']
['triticale', 'rye', 'EMPTY', 'EMPTY', 'rye']
['wheat', 'EMPTY', 'wheat', 'rye', 'EMPTY']

Macierz jakości gleb pól na przestrzeni lat
[90, 34, 54, 5, 16]
[82, 39, 59, 2, 21]
[77, 36, 51, 7, 18]
[69, 41, 56, 4, 23]
[64, 38, 61, 9, 20]

Wyżarzanie dla rozwiązania począkowego zachłannego

Rozwiązanie dające dochód 23373.18 zł
['EMPTY', 'EMPTY', 'EMPTY', 'rye', 'EMPTY']
['triticale', 'rye', 'EMPTY', 'EMPTY', 'rye']
['potato', 'EMPTY', 'EMPTY', 'rye', 'EMPTY']
['EMPTY', 'rye', 'EMPTY', 'EMPTY', 'rye']
['wheat', 'EMPTY', 'wheat', 'rye', 'EMPTY']

Macierz jakości gleb pól na przestrzeni lat
[90, 34, 54, 5, 16]
[95, 39, 59, 2, 21]
[90, 36, 64, 7, 18]
[85, 41, 69, 4, 23]
[90, 38, 74, 9, 20]
```

Rysunek 13: Rozwiązanie dla przykładowych danych

# 4 Inna wersja rozwiązania

```python
Piotr Mamos +1
def simulated_annealing(self, s0: list[list], k_max, stages): # Symulowane wyżarzanie
    self.__reset_variables()
    beast_s = deepcopy(s0) # Rozwiązanie najlepsze
    s = deepcopy(s0) # Rozwiązanie początkowe

    for k in range(k_max):
        if k_max >= 0.99**k:
            year = random.randrange(self.yearsNumber)
            field = random.randrange(self.fieldNumber)
            T = self.__annealing_temp(k, k_max)
            for _ in range(stages):
                s_new = self.__annealing_neig(s, k_max, T, year, field)
                if self.simulate_farm(s_new[0]) > self.simulate_farm(beast_s):
                    beast_s = s_new[0]
                if self.__annealing_P(self.simulate_farm(s), self.simulate_farm(s_new[0]), T) >= random.uniform(0, 1):
                    s = deepcopy(s_new[0])
                    year = s_new[1]
                    field = s_new[2]
        else:
            break

    return beast_s
```

Rysunek 14: simulated anealing

```python
def __annealing_neig(self, s_inp, k_m, T, last_year, last_field): # Funkcja wyznaczająca sąsiednie rozwiązanie
    range_year = self.__range_builder(last_year-T*self.yearsNumber/(2*k_m), last_year+T*self.yearsNumber/(2*k_m), self.yearsNumber)
    range_field = self.__range_builder(last_field-T*self.fieldNumber/(2*k_m), last_field+T*self.fieldNumber/(2*k_m), self.fieldNumber)
    year = random.randrange(range_year[0], range_year[1])
    field = random.randrange(range_field[0], range_field[1])
    curr_plant = s_inp[year][field]
    self.simulate_farm(s_inp)

    rand_plant = random.choice([plant for plant in PLANTS if plant != curr_plant or plant == "EMPTY"])
    if year > 0:
        while (self.Q[year - 1][field] - self.plantInfluenceDict[rand_plant]) < 0:
            rand_plant = random.choice([plant for plant in PLANTS if plant != curr_plant or plant == "EMPTY"])

    s_out = deepcopy(s_inp)
    s_out[year][field] = rand_plant

    # Zabezpieczenie przed wybraniem niedozwolonego rozwiązania
    # if rand_plant != 'EMPTY':
        # if (year > 0 and s_inp[year - 1][field] == rand_plant) or (year < self.yearsNumber-1 and s_inp[year + 1][field] == rand_plant):
    try:
        self.simulate_farm(s_out)
    except IndexError:
        # print('Nie spełnia ograniczenia jakości')
        return self.__annealing_neig(s_inp, k_m, T, last_year, last_field) # Ponowna próba
    except ValueError:
        # print('Nie spełnia ograniczenia innej rośliny w każdym roku')
        return self.__annealing_neig(s_inp, k_m, T, last_year, last_field) # Ponowna próba

    return s_out, year, field
```

Rysunek 15:

```python
def __range_builder(self, lower, higher, max_number):
    lower = int(lower)
    higher = int(higher)
    if lower == higher:
        lower -= 1
        higher += 1
    if lower < 0:
        lower = 0
    if higher > max_number:
        higher = max_number

    return [lower, higher]

Piotr Mamos +1
@staticmethod
def __annealing_P(e, e_dash, temp): # Funkcja akceptująca rozwiązanie, zmodyfikowana bo maksymalizujemy
    if e_dash > e:
        return 1

    else:
        return np.exp(((-1)*(e - e_dash))/temp)
```

Rysunek 16:

```
Rozwiązanie algorytmu zachłannego

Rozwiązanie dające dochód 13390.30 zł
['wheat', 'EMPTY', 'EMPTY', 'rye', 'EMPTY']
['triticale', 'rye', 'wheat', 'EMPTY', 'rye']
['wheat', 'EMPTY', 'EMPTY', 'rye', 'EMPTY']
['triticale', 'rye', 'EMPTY', 'EMPTY', 'rye']
['wheat', 'EMPTY', 'wheat', 'rye', 'EMPTY']

Macierz jakości gleb pól na przestrzeni lat
[90, 34, 54, 5, 16]
[82, 39, 59, 2, 21]
[77, 36, 51, 7, 18]
[69, 41, 56, 4, 23]
[64, 38, 61, 9, 20]

Wyżarzanie dla rozwiązania począkowego zachłannego

Rozwiązanie dające dochód 25809.20 zł
['EMPTY', 'EMPTY', 'EMPTY', 'EMPTY', 'EMPTY']
['triticale', 'rye', 'EMPTY', 'EMPTY', 'rye']
['EMPTY', 'EMPTY', 'EMPTY', 'rye', 'EMPTY']
['triticale', 'rye', 'EMPTY', 'EMPTY', 'rye']
['wheat', 'triticale', 'wheat', 'potato', 'EMPTY']

Macierz jakości gleb pól na przestrzeni lat
[90, 34, 54, 5, 16]
[95, 39, 59, 10, 21]
```

W porównaniu z naszym pierwszym rozwiązaniem

# 5 Problemy