AGH WEAIiIB	AiR rok III semestr 5	04.01.2022
Badania operacyjne II		
Skład grupy: *Barbara Pobiedzińska* *Marcin Biela* *Tomasz Brania*		grupa: środa 8:15 - 9:45 (1a)

Implementacja algorytmów Tabu Search oraz Genetycznego

1. Wstęp

Celem projektu jest zapoznanie się z wybranymi algorytmami. Należało wybrać problem do optymalizacji, a następnie wdrożyć 2 algorytmy w naszym przypadku było to Tabu Search i Algorytm genetyczny, aby końcowo otrzymać rozwiązanie suboptymalne według wybranych kryteriów i ograniczeń.

2. Opis zagadnienia

Sformułowanie problemu:

Proponowanym problemem jest ustalenie grafiku na określony przez użytkownika czas, według którego należy wybierać się do sklepu po zakupy.

Każde wyjście wiąże się z kosztami bezpośrednimi (zapłata za zakupy) jak i pośrednimi (np. ciężar produktów które trzeba unieść, czas poświęcony na zakupy). Są to wartości, które możemy optymalizować – w tym przypadku wybieramy czas poświęcony na zakupy, wyliczany przez ilość wyjść do sklepu.

Optymalizacji podlega ilość wyjść do sklepu. Jest to optymalizacja jednokryterialna.

Model matematyczny:

Istotne uwarunkowania i zależności

- Pojemność torby (jako maksymalny ciężar do uniesienia)
- Dni w które można robić zakupy (uwzględnienie niedziel niehandlowych)
- Uwzględnienie jak szybko zużywają się „zapasy” oraz to, że nie może być ich ujemna ilość
- Robienie maksymalnie 1 zakupów na dzień – aby nie okazało się że np. trzeba zrobić zakupy kilka razy w tym samym dniu (fizycznie niemożliwe lub bardzo ciężkie do zrealizowania)
- Bilans kaloryczny powinien być zaspokojony

● Lodówka ma ograniczoną pojemność

Zastosowane uproszczenia i uzasadnienie

● Nieuwzględnienie daty ważności produktów (upraszcza rachunki)
● Założenie, że każde zakupy zajmują tyle samo czasu – niezależnie od wielkości zakupów, dnia tygodnia, czy pory dnia (upraszcza rachunki)

Jakie informacje (dane) konieczne są do rozwiązania problemu

● Zależność wagi od kaloryczności produktu np. forma tabeli

Informacje cd.

● Pojemność torby (stała pojemność)
● Zależność czasu od ilości zabranych produktów (w przypadku uwzględnienia różnego czasu trwania zakupu)
● Stan początkowy $x_0 = 0$ → zerowe zapasy lodówki
● Ograniczenia dolne - konieczność podjęcia decyzji pójścia na zakupy domyślnie 0 ale można zwiększyć

Model matematyczny - struktury danych

● x_n - stan "lodówki" (na początku $x_0 = 0$)
● y_n - decyzja czy w danym dniu idziemy na zakupy (tożsame z wyjściem)
● n - n-ty dzień
● a_n - ograniczenie pojemności plecaka w danym dniu
● b - prędkość zużywania zapasów (zapotrzebowanie kaloryczne)
● c_i - tabela wagi kaloryczności
● N - ilość dni do ustalenia terminarza

Model matematyczny - zależności

● $x_n = x_{n-1} - b + \sum_{i=0}^{m} ci$ (kaloryczność) $*$ ci (decyzja) → zapas
● $x_n \geq 0$
● $y_n \in \{0, 1\}$
● c_i (decyzja) $\in \{0, 1\}$
● $\sum_{i=0}^{m}$ ci (waga) $*$ ci (decyzja) ≤ an

Funkcja celu

$\sum_{n=1}^{N} yn \rightarrow min$ (minimalizujemy ilość wyjść w danym okresie czasu)

Forma przykładowego rozwiązania

$$A_{4x7} \in \{0, 1\}$$

tydzien/dzien tygodnia	PON	WT	ŚR	CZW	PT	SOB	NDZ
1	1	0	0	0	1	0	0
2	1	0	1	0	0	1	0
3	1	1	0	0	0	1	0
4	0	0	1	1	0	0	0

3. Opis algorytmów

3.1 Algorytm Tabu Search

Schemat algorytmu podstawowego / pseudokodu:

Wybieramy losowe rozwiązanie startowe
s = best = RANDOM SOLUTION()
Inicjujemy listę tabu
t = []
Wykonujemy działania w pętli dopóki nie zostaną spełnione określone warunki końcowe
WHILE NOT(TERMINATION CONDITION)
{
Wykonujemy krok
s = SELECT(NEIGHBORS(s), t))
Aktualizujemy listę tabu
t = UPDATE TABU(s ,t)
Zapamiętujemy najlepsze rozwiązanie
if (f(best) < f(s))
{
best = s
}
}
Końcowo zwracamy najlepsze otrzymane rozwiązanie
return best

Adaptacja:

Realizację zaczynamy od stworzenia potrzebnych danych.
Definiujemy i określamy nasze ograniczenia w klasie Ograniczenia:

```python
class Ograniczenia:
    poczatkowy_stan_lodowki = 0  # poczatkowy stan lodowki

    maksymalna_poj_lodowki = 20  # maksymalna ilosc produktow w lodowce
    maks_liczba = 1  # maksymalna ilosc tego samego produktu
    N = 365  # tbd
    max_poj_plecaka = 7  # kg maksymalna pojemnosc_plecaka
    zapotrz_kal = 3000  # Zapotrzebowanie kaloryczne w danym dniu - w każdym dniu tyle samo
    kryterium_stopu = 100  # maksymalna ilosc iteracji
```

Utworzyliśmy także klasę Solution, która odpowiada za rozwiązanie końcowe. Znajduje się tutaj końcowa reprezentacja wyniku wraz z poprawnym wyświetlaniem liczbowym: (Nie używane w późniejszej implementacji - zmiana koncepcji)

```python
class Solution:
    def __init__(self, decyzja: int, lista_produktow: List[int], bilans: float):
        self.decyzja = decyzja
        self.lista_produktow = lista_produktow
        self.bilans = bilans

    def __str__(self):
        return f"d:{self.decyzja} lst: {self.lista_produktow} b: {self.bilans}"

    def __repr__(self):
        return self.__str__()

    def __len__(self):
        return len(self.lista_produktow)
```

Kolejno tworzymy listę produktów, które będziemy kupować oraz generujemy losowo

- wartości wagi
- kaloryczności (rozkład normalny o mean 1000 i d = 300)
- przydatności (końcowo tabela przydatności nie jest używana w implementacji)

dla każdego produktu i budujemy z nich tabele:

```python
def generuj_liste_produktow():
    # dane
    n_records = 10

    tabela_produktow = [f"Produkt{i + 1}" for i in range(n_records)]

    # tabele zczytywac z excela c(wartociowosc od wagi od czegos dodatkowego np kalorie)
    # ograniczenie plecaka tez (czy robic funkcje ktora wylicza ktore dni sa niehandlowe (niedziele i swieta))

    tabela_wartosciowosc_kalorie = np.round(np.random.normal(1000, 300, size=n_records))
    tabela_wartosciowosc_przydatnosc = np.random.randint(low=0, high=2, size=n_records)
    tabela_wag = np.round(np.random.random(n_records) * 0.7, 2)
    tabela_merge = {}
    for i in range(len(tabela_produktow)):
        tabela_merge[tabela_produktow[i]] = [tabela_wag[i], tabela_wartosciowosc_kalorie[i],
                                             tabela_wartosciowosc_przydatnosc[i]]

    df = pd.DataFrame(tabela_merge).transpose()
    df.rename(columns={0: "waga", 1: "kaloryczność", 2: "przydatnosc"})

    return df
```

Poniżej znajduje się funkcja zwracająca kalendarz, który ma za zadanie być reprezentacją czasową uwzględnianą przy badaniach. Kalendarz uwzględniałby święta oraz pojemność plecaka, która miala być zmienna na weekendach - wartość 0 oznacza, że w dany dzień nie można zrobić zakupów (w implementacji zakładamy, że w każdym dniu można zrobić zakupy) :

```python
def return_calendar(first_day, first_month, first_year, last_day, last_month, last_year):
    # doesnt include the last day -> update po dodaniu days = 1 tak
    # lista dni iteruje od 0

    # wielkanoc_mth = input("Kiedy Wielkanoc: numer miesiąca")
    # wielkanoc_day = input("Kiedy Wielkanoc: numer dnia")
    # boze_cialo_mth = input("Kiedy Boże Ciało: numer miesiąca")
    # boze_cialo_day = input("Kiedy Boże Ciało: numer dnia")

    # roboczo zeby nie wpisywac
    wielkanoc_mth = 4
    wielkanoc_day = 17
    boze_cialo_mth = 6
    boze_cialo_day = 16

    # lista_swiat_here = lista_swiat.copy()
    # lista_swiat_here.append((int(wielkanoc_mth), int(wielkanoc_day)))
    # lista_swiat_here.append((int(boze_cialo_mth), int(boze_cialo_day)))
    # # print(lista_swiat_here)

    # lista_swiat_here.sort()
    # # print(lista_swiat_here)
```

```python
vector_days = []

d0 = datetime.date(first_year, first_month, first_day)
d1 = datetime.date(last_year, last_month, last_day)

start_date = d0
end_date = d1
delta = datetime.timedelta(days=1)

while start_date <= end_date:
    weight = Ograniczenia.max_poj_plecaka
    # if start_date.weekday() == 6:
    #   weight = 0

    if start_date in lista_swiat:
        weight = 0  # poj plecaka

    vector_days.append((start_date, start_date.weekday(), weight))

    start_date += delta

return vector_days
```

Mając już stworzone odpowiednie struktury możemy przejść do implementacji właściwego algorytmu Tabu Search.
Całość zawarta będzie w klasie lodowka:

Atrybuty klasowe przedstawiają wszystkie ograniczenia i parametry

```
class lodowka():
    ograniczenia = ds.Ograniczenia

    poczatkowy_stan_lodowki = ograniczenia.poczatkowy_stan_lodowki  # początkowy stan lodowki

    maksymalna_poj_lodowki = ograniczenia.maksymalna_poj_lodowki  # maksymalna ilosc produktow w lodowce
    maks_liczba = ograniczenia.maks_liczba  # maksymalna ilosc tego samego produktu
    N = ograniczenia.N  # tbd
    max_poj_plecaka = ograniczenia.max_poj_plecaka  # kg maksymalna pojemnosc_plecaka
    zapotrz_kal = ograniczenia.zapotrz_kal  # Zapotrzebowanie kaloryczne w danym dniu - w każdym dniu tyle samo
    kryterium_stopu = ograniczenia.kryterium_stopu  # maksymalna ilosc iteracji
```

Na początku tworzony jest terminarz określonego okresu dni, lista produktów oraz generowane jest początkowe rozwiązanie, które w tej chwili jest przyjmowane jako najlepsze. Tworzymy także naszą listę tabu, która zawierać będzie niedozwolone rozwiązania lub rozwiązania, które wcześniej się pojawiły.

```
def __init__(self, terminarz, lista_produktow):
    """
    parameters:
    terminarz - Przechowuje informacje w postaci (data, dzien_tygodnia, maksymalna_dopuszczalna_waga)
    jeżeli maksymalna_dopuszczalna_waga = 0 wtedy w danym dniu nie idziemy do sklepu w przeciwnym wypadku jest równe max_poj_plecaka

    lista produktow - przechowuje informacja w postaci np.ndarray, gdzie pierwsze

    initial_solution przyjmuje to co zwraca metoda generate_initial_solution()
    """
    self.terminarz = terminarz
    self.lista_produktow = lista_produktow
    self.initial_solution = self.generete_initial_solution()
    self.tabu_list = []  # Lista tabu
    self.best_solution = self.initial_solution
    self.best_sol = self.zwroc_najlepsze_rozwiazanie(self.initial_solution)
```

Poniżej znajduje się funkcja odpowiedzialna za wylosowanie rozwiązania początkowego. Ważne jest, aby rozwiązanie to spełniało ograniczenia i warunki tj. ograniczenie pojemności lodówki, wagę produktów w plecaku oraz bilans kaloryczny, który nie powinien być ujemny. Bilans równy 0 oznacza, że dzienne zapotrzebowanie zostało zaspokojone, natomiast bilans ujemny świadczy o niedoborze kalorycznym. Ponadto w rozwiązaniu zobaczyć możemy decyzje o wyjściu do sklepu (0 - nie idziemy, 1 - idziemy) oraz listę produktów, które kupiliśmy w danym dniu. Przykładowo [1, 0, 0, 1, 0] oznacza, że kupiliśmy produkt nr 1 i nr 4.

```
def generete_initial_solution(self) -> np.ndarray:
    """
    Returns:
    initial_solution (list): Rozwiązanie początkowe, baza do kolejnych kroków.
    Kolejne elementy initial_solution oznaczają kolejne dni terminarza.
    Przykładowe elementy listy:
    [1, [0, 0, 1, 0, 0, 1, 0, 1, 1, 1], 0]
    [0, [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -19.0]
    gdzie:  initial_solution[i][0]: decyzja o pójściu na zakupy
            initial_solution[i][1]: lista wziętych produktów
            initial_solution[i][2]: bilans kalorii

    """
    initial_solution = []  # Rozwiązanie początkowe
    self.lista_produktow = np.array(self.lista_produktow)
    aktualny_stan_lodowki = self.poczatkowy_stan_lodowki
    zawartosc_lodowki = []
    for i in range(len(self.terminarz)):
        aktualny_stan_plecaka = 0
        if self.terminarz[i][2] != 0:
            teoretyczna_lista_zakupow = random.sample(range(10), 10)
            lista_1 = [0] * len(self.lista_produktow)
            count = 0
            while True and count < len(self.lista_produktow):
                indeks_produktu_ktory_bierzemy = teoretyczna_lista_zakupow[count]
                waga_produktu = self.lista_produktow[indeks_produktu_ktory_bierzemy][0]
```

```
            # sprawdzenie czy produkt który zamierzamy wziąć spełnia ograniczenia lodówki i plecaka:
            if aktualny_stan_lodowki + 1 <= self.maksymalna_poj_lodowki and aktualny_stan_plecaka + waga_produktu <= self.max_poj_plecaka:
                lista_1[indeks_produktu_ktory_bierzemy] = 1
                aktualny_stan_lodowki += 1
                aktualny_stan_plecaka += waga_produktu
                count += 1
            else:
                break
        initial_solution.append([1, lista_1, 0])
        zawartosc_lodowki.append(lista_1)
    else:
        initial_solution.append([0, [0] * len(self.lista_produktow), 0])
```

```
    wiersz_copy = copy.deepcopy(initial_solution[-1])
    # aktualizacja zużycia produktow (sprawdzenie w których dniach będzie niedobór kaloryczny (poprawiane następnie w check_kalorie())):
    aktualne_zuzycie = 0
    while aktualne_zuzycie < self.zapotrz_kal:
        if (len(np.nonzero(zawartosc_lodowki)[0])) > 0:
            id_row = np.nonzero(zawartosc_lodowki)[0][0]
            id_col = np.nonzero(zawartosc_lodowki)[1][0]
            jedzony_produkt = self.lista_produktow[id_col]
            zawartosc_lodowki[id_row][id_col] = 0
            kalorycznosc = jedzony_produkt[1]
            aktualne_zuzycie += kalorycznosc
            aktualny_stan_lodowki -= 1
        else:
            wiersz_copy[
                2] += aktualne_zuzycie - self.zapotrz_kal  # aktualizacja bilansu kalorii (zaznaczenie niedoboru)
            break

    initial_solution[-1] = wiersz_copy
return initial_solution
```

Szczególnie potrzebna była nam funkcja do sprawdzania, czy będziemy w stanie dostarczyć odpowiednio dużą ilość kalorii dla danego dnia. Przyjmujemy zmienną niedobór jako różnice kaloryczności naszych wybranych produktów i zapotrzebowania dobowego. Niedobór u nas powinien być dodatni, co końcowo oznaczymy jako 0.

```
def check_kalorie(self):
    """
    Metoda sprawdza czy produkty, które w danym dniu wybraliśmy spełniają nasze
    ograniczenie dotyczące zapotrzebowania dziennego na kalorie.

    Jeśli niedobor w danym dniu czyli (suma kaloryczności poszczególnych
    produktów) - (zapotrzebowanie dzienne) <  0 to wtedy wybieramy produkty o
    najmniejszej możliwej wadze dopóki niedobór będzie większy od 0.
    note: w tym momencie nie sprawdzamy czy po dadaniu produktu przekroczyliśmy
    maksymalna dopuszczalna pojemnosc plecaka max_poj_plecaka
    """
    for i in range(len(self.initial_solution)):
        if self.initial_solution[i][2] < 0:  # jeśli niedobor jest mniejszy od 0
            self.initial_solution[i][0] = 1  # trzeba bedzie pojsc na zakupy
            lista_zakupow = self.initial_solution[i][1]
            stan_plecaka = 0
            produkty_do_wziecia = copy.deepcopy(self.lista_produktow)
            for j in range(len(lista_zakupow)):
                stan_plecaka += lista_zakupow[j] * self.lista_produktow[j][0]
                if lista_zakupow[j] == 1:  # Bierzemy dany produkt jeśli 1
                    produkty_do_wziecia[j][0] = self.max_poj_plecaka + 1  # Zabronione przejscie
```

```
            # tym kroku spradzamy ponownie czy po wybraniu w pierwszej turze
            # x produktów i ich sumaryczna kaloryczność < zapotrzebowanie dzienne
            # to wtedy bierzemy n produktów o najmniejszej możliwej dopuszczalnej
            # masie tak aby sumaryczna_kaloryczność >= zapotrzebowanie_dzienne
            niedobor = self.initial_solution[i][2]
            while niedobor < 0:
                min_waga = np.min(produkty_do_wziecia[:, 0])
                itemindex = np.where(produkty_do_wziecia[:, 0] == min_waga)[0][0]
                self.initial_solution[i][1][itemindex] = 1
                niedobor += produkty_do_wziecia[itemindex][1]
                produkty_do_wziecia[itemindex][0] = self.max_poj_plecaka + 1
            self.initial_solution[i][2] = 0
        else:
            continue
```

Kolejnym krokiem jest wygenerowanie sąsiednich rozwiązań w funkcji step1. Odbywa się to w pętli for w ilości zależnej od ilości dni. Empirycznie wyznaczyliśmy, że dla okresu 7 dni optymalne będzie 30 wykonań. Zamiany realizowane są w oparciu o sąsiednie przesuwanie produktów w terminarzu. Przyjęliśmy 4 możliwe kierunki przesunięcia (odpowiednio: 1: do góry, 2: w dół, 3: w lewo, 4: w prawo) oraz uwzględniliśmy ograniczeniu ruchu przy “fizycznych” brzegach macierzy (przykładowo element o indeksach [0,0] nie może być przesunięty w górę oraz w lewo). Po wykonaniu wszystkich zamian zwracamy zmienioną macierz - sąsiednie rozwiązanie.

```
def step1(self, macierz_pom_produktow):
    """
    Generowanie sąsiednich rozwiązan
    """

    ile_zamian = int(macierz_pom_produktow.shape[0]/7*30)
    for i in range(ile_zamian):
        kierunek_przesuniecia = [1, 2, 3, 4]  # 1 - gora, 2 - dol, 3-lewo, 4-
        x_idx = np.random.randint(0, macierz_pom_produktow.shape[1])
        y_idx = np.random.randint(0, macierz_pom_produktow.shape[0])
        if x_idx == 0:
            kierunek_przesuniecia.remove(4)
        elif x_idx == macierz_pom_produktow.shape[1] - 1:
            kierunek_przesuniecia.remove(2)

        if y_idx == 0:
            kierunek_przesuniecia.remove(1)

        elif y_idx == macierz_pom_produktow.shape[0] - 1:
            kierunek_przesuniecia.remove(3)
```

```
        kierunek_przesuniecia_wybor = random.choice(kierunek_przesuniecia)
        if kierunek_przesuniecia_wybor == 1:  # gora
            macierz_pom_produktow[y_idx, x_idx], macierz_pom_produktow[y_idx - 1, x_idx] = macierz_pom_produktow[
                                                                                               y_idx - 1, x_idx], \
                                                                                           macierz_pom_produktow[
                                                                                               y_idx, x_idx]
        elif kierunek_przesuniecia_wybor == 3:  # dol
            macierz_pom_produktow[y_idx, x_idx], macierz_pom_produktow[y_idx + 1, x_idx] = macierz_pom_produktow[
                                                                                               y_idx + 1, x_idx], \
                                                                                           macierz_pom_produktow[
                                                                                               y_idx, x_idx]

        elif kierunek_przesuniecia_wybor == 2:  # prawo
            macierz_pom_produktow[y_idx, x_idx], macierz_pom_produktow[y_idx, x_idx + 1] = macierz_pom_produktow[
                                                                                               y_idx, x_idx + 1], \
                                                                                           macierz_pom_produktow[
                                                                                               y_idx, x_idx]
        else:  # lewo
            macierz_pom_produktow[y_idx, x_idx], macierz_pom_produktow[y_idx, x_idx - 1] = macierz_pom_produktow[
                                                                                               y_idx, x_idx - 1], \
                                                                                           macierz_pom_produktow[
                                                                                               y_idx, x_idx]

    return macierz_pom_produktow
```

Stworzyliśmy także funkcję, która ma za zadanie sprawdzić, czy pojemność naszej lodówki nie będzie przekroczona. Według naszych założeń wartość ta ma być z przedziału 0 - 20. Funkcja może otrzymywać macierz zawierającą listę zakupionych produktów w danym okresie, lecz jeśli nie podamy tego parametru to ta macierz zostanie wygenerowana. Bazujemy na macierzy, z której możemy prosto zsumować liczbę zakupionych produktów i sprawdzić czy zmieszczą się one do lodówki. Szczególnie ważne jest, aby rozwiązanie jednocześnie spełniało wszystkie postawione warunki, więc musimy także uwzględnić spełnienie warunku kalorycznego.

```
def check_capacity(self, macierz_pom_produktow2=None):
    """
    jako parametr przyjmuje macierz w której wiersze reprezentują kolejne dni, natomiast
    kolumny listę produktów
    Zwraca List informujaca czy w danym dniu zakres lodowki został przekroczony
    """
    if macierz_pom_produktow2 is None:
        macierz_pom_produktow2 = np.empty((len(self.initial_solution), 10))
        for i in range(0, len(self.initial_solution)):
            macierz_pom_produktow2[i] = self.initial_solution[i][1]

    ponad_stan_lst_lodowka = []
    bilans_kalorie = []
    aktualny_stan_lodowki = self.poczatkowy_stan_lodowki
    zawartosc_lodowki = []
    for row in range(len(macierz_pom_produktow2)):

        if not all([v == 0 for v in macierz_pom_produktow2[row]]):
            zawartosc_lodowki.append(macierz_pom_produktow2[row])
            aktualny_stan_lodowki += sum(macierz_pom_produktow2[row])
            ponad_stan_lst_lodowka.append(aktualny_stan_lodowki)
```

```
        aktualne_zuzycie = 0
        bilans_kalorie.append(0)
        while aktualne_zuzycie < self.zapotrz_kal:
            if (len(np.nonzero(zawartosc_lodowki)[0])) > 0:
                id_row = np.nonzero(zawartosc_lodowki)[0][0]
                id_col = np.nonzero(zawartosc_lodowki)[1][0]
                jedzony_produkt = self.lista_produktow[id_col]

                zawartosc_lodowki[id_row][id_col] = 0
                kalorycznosc = jedzony_produkt[1]
                aktualne_zuzycie += kalorycznosc
                aktualny_stan_lodowki -= 1

            else:
                bilans_kalorie.append(aktualne_zuzycie - self.zapotrz_kal)
                break

    return ponad_stan_lst_lodowka, bilans_kalorie
```

Utworzyliśmy także funkcję pomocniczą. która ma za zadanie sprawdzić, czy otrzymane rozwiązanie jest już w liście tabu. W tym celu przechodzimy po liście tabu i sprawdzamy czy elementy z naszego rozwiązania są identyczne. Jeśli nie to zwracamy fałsz, jeśli wszystko się pokrywa zwracamy prawdę.

```
def check_current_sol_in_tabu_list(self, list_of_sol: List[np.ndarray], current_solution: np.ndarray) -> bool:
    """
    Sprawdza czy obecne rozwiązanie jest w tabu list zwraca True jeśli tak
    """
    return next((True for elem in self.tabu_list if elem is current_solution), False)
```

Poniższa funkcja ma za zadanie wyłonic nam najlepsze otrzymane rozwiązanie, czyli najmniejszą ilość wyjść do sklepu. Przekazujemy do funkcji dane rozwiązanie, a ona wylicza sumuje nam ilość decyzji pozytywnych - reprezentowanych jako 1.

```
def zwroc_najlepsze_rozwiazanie(self, initial_solution):
    min = 0
    for i in range(len(initial_solution)):
        min += initial_solution[i][0]
    return min
```

Potrzebowaliśmy dodatkowo funkcj do wyciągania listy samych produktów z reprezentacji całego rozwiązania (bez decyzji i kaloryczności). Jeśli przekażemy całe rozwiązanie to lista produktów zostanie przepisana i zwrócona jako wynik, natomiast jeśli nie podamy żadnego parametru to lista produktów będzie przepisana z początkowego rozwiązania.

```
def zwroc_liste_produktow(self, macierz_pom_produktow=None):

    if macierz_pom_produktow is not None:
        macierz_pom_produktow2 = np.empty((len(macierz_pom_produktow), 10))
        for i in range(0, len(macierz_pom_produktow)):
            macierz_pom_produktow2[i] = macierz_pom_produktow[i][1]
        return macierz_pom_produktow2

    else:
        macierz_pom_produktow = np.empty((len(self.initial_solution), 10))
        for i in range(0, len(self.initial_solution)):
            macierz_pom_produktow[i] = self.initial_solution[i][1]
        return macierz_pom_produktow
```

Utworzona została funkcja, która tworzy nam ujednolicony, reprezentowalny twór rozwiązania. Tak jak wspomniane było już wcześniej, w fazie implementacji bazowaliśmy na rozwiązaniu w postaci macierzy zawierającej w każdym wierszu decyzję wyjścia (0 lub 1), listę produktów oraz bilans kaloryczny dla każdego dnia.

```
def postac_do_rozwiazania(self, lista_produktow):
    lista = copy.deepcopy(lista_produktow)
    rozwiazanie = []
    bilans = self.check_capacity(lista)[1]
    for i in range(len(lista_produktow)):
        decyzja = 1
        if all([v == 0 for v in lista_produktow[i]]):
            decyzja = 0
        rozwiazanie.append([decyzja, lista_produktow[i], bilans[i]])
    return rozwiazanie
```

Przydała nam się również prosta funkcja do wypisywania tegoż rozwiązania.

```
def print_solution(self, solution):
    S = ''
    for elem in solution:
        S += f"d: {elem[0]}  lst: {elem[1]}\n"
    print(S)
```

Poniżej pokazana jest główna funkcja realizująca algorytm Tabu Search. Zaczynamy od utworzenia listy produktów i dodaniu jej do listy tabu. Na początku przyjmujemy, że początkowe rozwiązanie jest naszym najlepszy i wypisujemy wynik początkowy. Następnie w pętli wykonujemy kod aż nie zostanie osiągnięte kryterium stopu. Zwiększamy iteracje i generujemy nową macierz wykonując mieszanie funkcją step1(). Wyliczamy ograniczenia oraz sprawdzamy czy otrzymane rozwiązanie nie jest już w liście tabu. Jeśli jest to robimy kolejną iterację pętli. W przeciwnym przypadku jeśli ograniczenia nie są spełnione to rozwiązanie to trafia do listy tabu. Spełnienie ograniczeń oznacza, że tworzymy postać rozwiązania końcowego i porównujemy czy to właśnie otrzymane jest lepsze od poprzedniego. Jeśli mamy lepszy wynik uaktualniamy najlepsze rozwiązanie i wyświetlamy je w konsoli.

```
def tabu_solution(self):
    it = 0
    lista_produktow_poczatkowa = self.zwroc_liste_produktow()
    self.tabu_list.append(lista_produktow_poczatkowa)
    # print("Poczatkowe najlepsze rozwiazanie",
    #       self.zwroc_najlepsze_rozwiazanie(self.postac_do_rozwiazania(lista_produktow_poczatkowa)))
    # self.print_solution(self.postac_do_rozwiazania(lista_produktow_poczatkowa))
    poprzednie_rozwiazanie = self.zwroc_liste_produktow()
    print("Początkowe rozwiazanie: ", self.best_sol)
```

```
    while it < self.kryterium_stopu:
        it += 1
        copy_for_step = copy.deepcopy(poprzednie_rozwiazanie)
        # sąsiednie rozwiązanie wygenerowane z poprzedniego
        sasiednie_rozwiazanie = self.step1(copy_for_step)
        # self.check_kalorie()
        sasiednie_rozwiazanie = self.zwroc_liste_produktow(sasiednie_rozwiazanie)
        for_check = copy.deepcopy(sasiednie_rozwiazanie)
        sasiednie_rozwiazanie_lodowka_ponad_stan, kalorie_sasiednie_rozwiazanie = self.check_capacity(
            macierz_pom_produktow2=for_check)

        if self.check_current_sol_in_tabu_list(self.tabu_list, for_check):
            continue

        elif any([v > self.maksymalna_poj_lodowki for v in sasiednie_rozwiazanie_lodowka_ponad_stan]) or any(
                [v < 0 for v in kalorie_sasiednie_rozwiazanie]):

            self.tabu_list.append(for_check)
```

```
        else:
            rozwiazanie = self.postac_do_rozwiazania(sasiednie_rozwiazanie)
            best_current_sol = self.zwroc_najlepsze_rozwiazanie(rozwiazanie)
            poprzednie_rozwiazanie = self.zwroc_liste_produktow(rozwiazanie)

            if self.zwroc_najlepsze_rozwiazanie(self.best_solution) > best_current_sol:
                print(f"Najlepsze rozwiazanie\t: {best_current_sol} , it: {it}")
                # self.print_solution(rozwiazanie)
                self.best_sol = best_current_sol
                self.best_solution = rozwiazanie
            else:
                continue
```

W pliku main.py stworzyliśmy funkcję do wyznaczania rozwiązania dla postawionego problemu za pomocą algorytmu Tabu Search. Wewnątrz tworzymy kalendarz dla badanego okresu, listę produktów możliwych do kupienia, inicjujemy struktury danych oraz rozwiązujemy problem i końcowo zwracamy najlepsze rozwiązanie. W dalszej implementacji algorytmu zrezygnowaliśmy z podawania dni w które nie możemy robić zakupów.

```
def test_tabu_solution():
    terminarz = ds.return_calendar(1, 1, 2022, 30, 1, 2022)
    lista_produktow = ds.generuj_liste_produktow()
    lod1 = tsol.lodowka(terminarz, lista_produktow)
    lod1.tabu_solution()
    lod1.print_solution(lod1.best_solution)
    print(lod1.zwroc_najlepsze_rozwiazanie(lod1.best_solution))
```

Dodatkowo stworzone, lecz niewykorzystane:

Utworzona została lista świąt w roku 2022:

```
# lista_swiat = [datetime.date(2022, 1, 1),
#                datetime.date(2022, 1, 6),
#                datetime.date(2022, 4, 17),
#                datetime.date(2022, 4, 18),
#                datetime.date(2022, 5, 1),
#                datetime.date(2022, 5, 3),
#                datetime.date(2022, 6, 5),
#                datetime.date(2022, 6, 16),
#                datetime.date(2022, 7, 15),
#                datetime.date(2022, 11, 1),
#                datetime.date(2022, 11, 11),
#                datetime.date(2022, 12, 25),
#                datetime.date(2022, 12, 26)]
lista_swiat = []
```

Zaimplementowana została funkcja sprawdzająca wagę zakupionych produktów w stosunku do maksymalnego udźwigu plecaka.

```
# def check_weight(self, macierz_pom_produktow: np.ndarray) -> List[bool]:
#     """
#     jako parametr przyjmuje macierz w której wiersze reprezentują kolejne dni, natomiast
#     kolumny listę produktów
#     Zwraca List[bool] informujaca czy w danym dniu zakres plecaka został przekroczony
#     """
#     ponad_stan_lst = []
#     for row in range(len(macierz_pom_produktow)):
#         waga = 0
#         for col in range(len(macierz_pom_produktow[row])):
#             waga += macierz_pom_produktow[row][col] * self.lista_produktow[col][0]
#
#         if waga > self.max_poj_plecaka:
#             ponad_stan_lst.append(True)
#
#         else:
#             ponad_stan_lst.append(False)
#
#     return ponad_stan_lst
```

3.2 Algorytm genetyczny

Schemat algorytmu podstawowego / pseudokodu:

1. Utwórz początkową populację chromosomów P.
2. Oceń dopasowanie każdego chromosomu.
3. Wybierz rodziców P/2 z obecnej populacji poprzez selekcję proporcjonalną.
4. Losowo wybierz dwóch rodziców, aby stworzyć potomstwo za pomocą operatora krzyżowania.
5. Zastosuj operatory mutacji dla drobnych zmian w wynikach.
6. Powtarzaj kroki 4 i 5, aż wszyscy rodzice zostaną wybrani i połączeni.
7. Wymień starą populację chromosomów na nową.

8. Oceń dopasowanie każdego chromosomu w nowej populacji.
9. Zakończ, jeśli liczba pokoleń osiągnie pewną górną granicę; w przeciwnym razie przejdź do kroku 3.

Adaptacja:

Potrzebne struktury danych mamy już stworzone i są one identyczne, jak dla poprzedniego algorytmu.

Przydała nam się również prosta funkcja do wypisywania rozwiązania.

```
def print_solution(solution):
    S = ''
    for elem in solution:
        S += f"d: {elem[0]}  lst: {elem[1]}  b: {elem[2]}\n"
    print(S)
```

Dodaliśmy też drugą funkcję wyświetlającą. Ma ona inną strukturę do pokazania oraz dodaje wybrany tytuł.

```
def print_solution2(data: List[List[int]], title: str = '...'):
    print(title, sep='\n')
    for i in range(len(data)):
        tmp = np.array(data[i])
        print(tmp)
        print('\n')
```

Przechodzimy do pisania kolejnych kroków algorytmu genetycznego. Zaczynamy od utworzenia początkowej populacji chromosomów P. Tworzymy funkcję inicjującą, do której możemy przekazać ilość osobników do otrzymania, kalendarz oraz produkty wraz z ich parametrami. W początkowej populacji znajdują się rozwiązania początkowe.

```
# inicjalizacja populacji p osobników
def init_population(n, terminarz, lista_produktow):
    """
    Parameters:
    n - ilość osobników do otrzymania
    terminarz - lista dni na jakie chcemy zrobić rozpiskę
    lista_produktow - stała lista produktów z ich wagą i kalorycznością
    Returns:
    init_population (list): lista rozwiązań początkowych
    """
    init_population = []
    for i in range(n):
        lodowka_ = lodowka(terminarz, lista_produktow).initial_solution
        init_population.append(lodowka_)
    return init_population
```

W drugim kroku wykonujemy ocenę każdego chromosomu. Miarą oceny jest liczba decyzji o wyjściu do sklepu. W naszym przypadku to suma decyzji będących 0 lub 1.

```python
# wyliczenie wartości funkcji celu osobnika
def evaluate_chromosome(solution):
    """
    Parameters:
    solution - osobnik
    Returns:
    fitness: wartość funkcji celu dla danego osobnika
    """
    fitness = 0
    for i in range(len(solution)):
        fitness += solution[i][0]
    return fitness
```

Wykonaliśmy także funkcję, która korzystając z kodu powyżej, wylicza i przydziela (tworząc listę) każdemu osobnikowi jego wartość funkcji celu (ilość wyjść w danym okresie).

```python
# wyliczenie wartosci funkcji celu dla populacji
def eval_init_population(init_population):
    evaluated_init_population = []
    for i in range(len(init_population)):
        evaluated_init_population.append((init_population[i], evaluate_chromosome(init_population[i])))
    return evaluated_init_population
```

Kolejnym etapem jest wybranie rodziców P/2 z obecnej populacji.
Pierwotnie wykorzystywaliśmy metodę koła ruletki.
Tworzymy wirtualne koło, którego wycinki są proporcjonalną częścią do wartości oceny każdego osobnika. Im większy wycinek koła, tym większe prawdopodobieństwo wylosowania danego osobnika.
Ponieważ powodowało to częste wykluczanie osobników o poprawionej wartości funkcji celu, zdecydowaliśmy się na wybieranie P//2 najlepszych osobników.
Do funkcji przekazujemy otrzymaną listę z poprzedniego kroku oraz wielkość populacji. Aby móc podzielić rodziców musimy mieć parzystą ich ilość, więc jeśli jest ona nieparzysta to zwiększamy ją o 1. W pętli while następuje podział rodziców, a później funkcja zwraca listę z P/2 rodzicami.
W kolejnych iteracjach algorytmu wybieramy tę pierwotną ilość rodziców aby populacja się nie zmniejszała.

```python
# wybranie osobnikow do krzyżowania - wybierane jest n najlepszych
# ponieważ po krzyzowaniu niektóre osobniki potomne nie spełniają ograniczeń
# i nie sa dalej przekazywane, n jest stałą liczbą równą połowie długości początkowej populacji
def choose_parents(evaluated_init_population, len_init_population):
    """
    Parameters:
    evaluated_init_population - lista osobników wraz z ich wartościami funkcji celu
    len_init_population - długość początkowej populacji
    Returns:
    parents_list - lista rodziców wybranych do krzyżowania
    """

    parents_list = []
    ite = 0
    list_eval = copy.deepcopy(evaluated_init_population)
```

```
    how_many = len_init_population // 2
    if how_many % 2 != 0:
        how_many += 1

    lista_pomocnicza2 = []
    while ite < how_many:
        ite += 1
        elem = list_eval.pop(0)
        parents_list.append(elem[0])
        lista_pomocnicza2.append(elem[1])

    return parents_list
```

Teraz musimy wybrać rodziców do krzyżowania. Dokonuje tego funkcja return_pairs(). Przechodzimy po otrzymanej liście rodziców z krokiem 2 i wybieramy tych z kolejnymi indeksami. Funkcja zwraca gotowe pary.

```
# podział rodziców w pary: 1-2, 3-4 etc
def return_pairs(parents_list):
    pairs_list = []
    for i in range(0, len(parents_list) - 1, 2):
        pairs_list.append((parents_list[i], parents_list[i + 1]))
    return pairs_list
```

Przyszedł czas na krzyżowanie rodziców. Wybieramy losowo punkt, w którym nastąpi podział rodziców. Odcięte części rodziców zamieniamy ze sobą tak, aby wykonać krzyżowanie i długość powstałego w ten sposób potomstwa była stała i równa rodzicom. Otrzymane dzieci umieszczamy w liście, która jest zwracana jako wynik krzyżowania.

```
# crossover
def crossover(pairs_list):
    offsprings_list = []
    for i in range(len(pairs_list)):
        cutpoint = random.randint(1, len(pairs_list[i][0]) - 1)
        offspring1 = []
        offspring2 = []
        for j in range(len(pairs_list[i][0])):
            if j < cutpoint:
                offspring1.append(pairs_list[i][0][j][1])
                offspring2.append(pairs_list[i][1][j][1])
            else:
                offspring1.append(pairs_list[i][1][j][1])
                offspring2.append(pairs_list[i][0][j][1])
        offsprings_list.append(offspring1)
        offsprings_list.append(offspring2)
    return offsprings_list
```

Kolejnym krokiem jest wykonanie mutacji rozwiązań. Pojedyncza mutacja jest zawarta w poniżej i jest ona zaczerpnięta z funkcji step1() z poprzedniego algorytmu. Całość odbywa się w pętli *for* w ilości zależnej od ilości dni. Empirycznie wybraliśmy wartość 30 zamian na każde 7 dni terminarza. Zamiany realizowane są w oparciu o sąsiednie przesuwanie produktów w terminarzu. Przyjęliśmy 4 kierunki przesuwania (odpowiednio: 1: do góry, 2: w dół, 3: w lewo, 4: w prawo) oraz uwzględniliśmy ograniczeniu ruchu przy “fizycznych”

brzegach macierzy (przykładowo element o indeksach [0,0] nie może być przesunięty w górę oraz w lewo).

```
# mutacja pojedynczego osobnika
def mutation_singular(macierz_pom_produktow, ile_zamian=None):
    macierz_pom_produktow2 = np.array(macierz_pom_produktow)

    if ile_zamian == None:
        ile_zamian = int(macierz_pom_produktow2.shape[0] / 7 * 30)

    for i in range(ile_zamian):
        kierunek_przesuniecia = [1, 2, 3, 4]  # 1 - gora, 2 - dol, 3-lewo, 4-
        x_idx = np.random.randint(0, macierz_pom_produktow2.shape[1])
        y_idx = np.random.randint(0, macierz_pom_produktow2.shape[0])
        if x_idx == 0:
            kierunek_przesuniecia.remove(4)
        elif x_idx == macierz_pom_produktow2.shape[1] - 1:
            kierunek_przesuniecia.remove(2)

        if y_idx == 0:
            kierunek_przesuniecia.remove(1)
        elif y_idx == macierz_pom_produktow2.shape[0] - 1:
            kierunek_przesuniecia.remove(3)
```

```
        kierunek_przesuniecia_wybor = random.choice(kierunek_przesuniecia)
        if kierunek_przesuniecia_wybor == 1:  # gora
            macierz_pom_produktow[y_idx][x_idx], macierz_pom_produktow[y_idx - 1][x_idx] = macierz_pom_produktow[
                                                                                               y_idx - 1][x_idx], \
                                                                                           macierz_pom_produktow[
                                                                                               y_idx][x_idx]
        elif kierunek_przesuniecia_wybor == 3:  # dol
            macierz_pom_produktow[y_idx][x_idx], macierz_pom_produktow[y_idx + 1][x_idx] = macierz_pom_produktow[
                                                                                               y_idx + 1][x_idx], \
                                                                                           macierz_pom_produktow[
                                                                                               y_idx][x_idx]

        elif kierunek_przesuniecia_wybor == 2:  # prawo
            macierz_pom_produktow[y_idx][x_idx], macierz_pom_produktow[y_idx][x_idx + 1] = macierz_pom_produktow[
                                                                                               y_idx][x_idx + 1], \
                                                                                           macierz_pom_produktow[
                                                                                               y_idx][x_idx]
        else:  # lewo
            macierz_pom_produktow[y_idx][x_idx], macierz_pom_produktow[y_idx][x_idx - 1] = macierz_pom_produktow[
                                                                                               y_idx][x_idx - 1], \
                                                                                           macierz_pom_produktow[
                                                                                               y_idx][x_idx]

    return macierz_pom_produktow
```

Funkcja mutation() odpowiada za przeprowadzenie mutacji w oparciu o prawdopodobieństwo mutacji. Do funkcji trafia lista z potomstwem, prawdopodobieństwo zmiany oraz flaga zamiany. Mutacja odbywa się tylko, gdy wylosowana liczba jest mniejsza od zadanego prawdopodobieństwa. Potomek trafia wtedy do funkcji odpowiedzialnej za zamiany. Na koniec każde dziecko (zmutowane lub oryginalne) jest wpisywane do listy i zwracane.

```python
def mutation(offsprings_list, prob_od_mut=0.1, changes_no=None):
    offsprings_list_mutated = []
    for i in range(len(offsprings_list)):
        mutation = random.random()
        if mutation <= prob_od_mut:
            mutated = mutation_singular((offsprings_list[i]), changes_no)
            offsprings_list_mutated.append(mutated)
        else:
            offsprings_list_mutated.append((offsprings_list[i]))
    return offsprings_list_mutated
```

W przypadku gdy po krzyżowaniu i mutacji dany osobnik nie spełnia założeń, nie powinien być brany pod uwagę. Zdecydowalismy się na odrzucanie takiego osobnika i nie przekazywanie go do dalszej populacji.
Do dalszej populacji przekazujemy rodziców i potomków, następnie wybierając połowę z nich. Przez to istnieje prawdopodobieństwo, że któryś z potomków będzie odrzucony i przez to otrzymana populacja będzie mniejsza - stąd decyzja, żeby w funkcji choose parents wybierana liczba osobników była stała i wynosiła połowę z pierwotnej liczby osobników.

```python
# !! napisac w dokumentacji o dostosowaniu plecaka do wag

# po mutacji mozemy  sprawdzac poprawnosc otrzymanego osobnika (czy spelnia zalozenia)
# jezeli nie to mozemy
# a) odrzucic go i do wynikowej listy osobnikow (nowej populacji) go nie wpisywac
#   -> zeby populacja sie nie zmniejszala mozemy ze starej populacji podmmieniac te osobniki o najgorszej wartosci f celu
# b) mozemy go probowac poprawiac (jezeli brakuje kalorii do dodawac produkty) -> zeby nie bylo problemu to mozemy zwiekszyc ktores z ograniczen
# z kotrego bysmy nie skorzystali rozwiazania trzeba bedzie dokladnie opisac w dokumenacji, bo to nie jest zbyt "standardowe" rozwiazanie
```

Poniższa funkcja ma za zadanie sprawdzić założenia dla danego osobnika. W pętli brany jest każdy wiersz, który zawiera zakupione produkty, a następnie sprawdzamy czy produkty zmieszczą się w lodówce i spełniony będzie bilans kaloryczny. Jeśli dla każdego dnia ograniczenia są nienaruszone to zwracamy 1, jako wartość logiczną. W przeciwnym przypadku funkcja daje 0.

```python
def check_offspring_singular(offspring):
    """
    jako parametr przyjmuje macierz w której wiersze reprezentują kolejne dni, natomiast
    kolumny listę produktów
    Zwraca bool - czy osobnik spelnia zalozenia, czy nie
    """

    ponad_stan_lst_lodowka = []
    bilans_kalorie = []
    aktualny_stan_lodowki = ograniczenia.poczatkowy_stan_lodowki
    zawartosc_lodowki = []
    for row in range(len(offspring)):

        if not all([v == 0 for v in offspring[row]]):
            zawartosc_lodowki.append(offspring[row])
            aktualny_stan_lodowki += sum(offspring[row])
            ponad_stan_lst_lodowka.append(aktualny_stan_lodowki)
```

```
        aktualne_zuzycie = 0
        bilans_kalorie.append(0)
        while aktualne_zuzycie < ograniczenia.zapotrz_kal:
            if (len(np.nonzero(zawartosc_lodowki)[0])) > 0:
                id_row = np.nonzero(zawartosc_lodowki)[0][0]
                id_col = np.nonzero(zawartosc_lodowki)[1][0]
                jedzony_produkt = lista_produktow[id_col]

                zawartosc_lodowki[id_row][id_col] = 0
                kalorycznosc = jedzony_produkt[1]
                aktualne_zuzycie += kalorycznosc
                aktualny_stan_lodowki -= 1

            else:
                bilans_kalorie.append(aktualne_zuzycie - ograniczenia.zapotrz_kal)
                break
    if all([elem <= ograniczenia.maksymalna_poj_lodowki for elem in
            ponad_stan_lst_lodowka]) and all([elem == 0 for elem in bilans_kalorie]):
        return 1
    return 0
```

Check_offspring() jest główną funkcja sprawdzającą, czy dany potomek spełnia założenia. Jeśli założenia się zgadzają to element trafia na listę, która jest zwracana jako wynik.

```
def check_offspring(offsprings_list_mutated):
    offsprings_checked = []
    for i in range(len(offsprings_list_mutated)):
        offs_copy = copy.deepcopy(offsprings_list_mutated[i])
        offspring_ok = check_offspring_singular(offs_copy)

        if offspring_ok:
            offsprings_checked.append(offsprings_list_mutated[i])
        else:
            continue
    return offsprings_checked
```

Funkcja sprawdza, czy w danym wierszu jest element niezerowy. Jeśli jest zwracana jest 1.

```
def if_row_has_zero(row: List[int]) -> int:
    if any(row):
        return 1
    else:
        return 0
```

Funkcja poniżej sumuje ilość niezerowych wierszy w macierzy wejściowej. Wykorzystywana jest funkcja wyżej.

```
def evaluate_1(individual: List[List[int]]):
    sum = 0
    for elem in individual:
        sum += if_row_has_zero(elem)
    return sum
```

Przechodzimy do kroku nr 8. Oceniamy dopasowanie każdego chromosomu w nowej populacji. W tym celu wykonujemy ocenę dla starej i nowej populacji, a wyniki zapisujemy w

dwóch listach w formacie krotki - chromosom i suma funkcji celu. Łączymy obie listy, a później wykonujemy sortowanie listy według wartości decyzji o wyjściu do sklepu w kolejności rosnącej.

```
def replace_old_pop_with_new_one(old_population: List[List[List[int]]], new_population: List[List[List[int]]]):
    result_for_old = []
    for elem in old_population:
        pair = elem, evaluate_1(elem)
        result_for_old.append(pair)
```

```
    result_for_new = []
    for elem in new_population:
        pair = elem, evaluate_1(elem)
        result_for_new.append(pair)

    nowa_lista = []
    for i in range(0, len(result_for_new)):
        nowa_lista.append(result_for_new[i])
    for i in range(0, len(result_for_old)):
        nowa_lista.append(result_for_old[i])

    nowa_lista = sorted(nowa_lista, key=lambda t: t[1], reverse=False)

    return nowa_lista
```

Stworzyliśmy także dodatkową funkcję, która służy do zmiany formatu osobnika. Z listy osobników, gdzie osobnik jest postaci (decyzja, produkty, bilans) przechodzimy na postać macierzową.

```
def pull_parents_form_parents_longer(old_population):
    parents = []
    for i in range(len(old_population)):
        parents.append([])
        for j in range(len(old_population[i])):
            parents[i].append(old_population[i][j][1])
    return parents
```

Jest to kolejna funkcja, która ma za zadanie przeformatować rozwiązanie. Końcowo otrzymamy rozwiązanie, które ma postać finalną, czyli lista list decyzji, macierzy produktów oraz bilansu.

```
def change_new_popul_to_other_format(new_population):
    formatted = []

    for elem in (new_population):
        list = []
        for j in range(len(elem[0])):
            decision = if_row_has_zero(elem[0][j])
            list.append([decision, elem[0][j], 0])
        formatted.append(list)

    return formatted
```

Ostatnim elementem do implementacji jest złożenie funkcji do realizacji algorytmu genetycznego. Funkcja dostaje górny zakres, listę produktów, kalendarz, długość populacji, prawdopodobieństwo mutacji oraz liczbę zmian w mutacji. Zaczynamy od utworzenia początkowej populacji. Następnie w pętli while dopóki nie zostanie osiągnięty zakres górny, obliczamy wartości funkcji celu i zapisujemy wraz z osobnikami. Kolejno wybieramy połowę naszej populacji, dzielimy je losowo na pary i wykonujemy krzyżowanie. Część nowej populacji ulegnie mutacji zgodnie z zadanym prawdopodobieństwem. Następnym krokiem jest sprawdzenie czy nowe osobniki spełniają założenia i wpisanie ich do starych osobników. Po wykonaniu się pętli otrzymujemy rozwiązanie końcowe - zminimalizowaną liczbę wyjść do sklepu, listę zakupionych produktów oraz bilans kaloryczny.

```python
def genetic_algo(upper_bound, lista_produktow, terminarz, len_init_population, probability=0.1, liczba_zamian=None):
    global najlepsze_rozwiazanie
    # osobniki poczatkowe
    init_specimen = init_population(len_init_population, terminarz, lista_produktow)

    i = 0
    while i < upper_bound:
        # osobniki poczatkowe plus wartosc f celu
        init_specimen_f_celu = eval_init_population(init_specimen)
```

```python
        # zwraca polowe dlugosci pierwotnej listy osobników najlepszych osobników
        parents = choose_parents(init_specimen_f_celu, len_init_population)
        copy_parents = copy.deepcopy(parents)

        # łaczenie wybranych w losowaniu osobników w pary
        pairs = return_pairs(parents)

        # krzyżowanie rodziców - nowa populacja
        offspring = crossover(pairs)

        # mutowanie osobników z prawdopodobienstwem wystapienia mutacji = probability (wartosc domyslna  = 0.1)
        # oraz liczba zamian przy niej wykonywanych = liczba_zamian (wartosc domysna ~= 30 zamian na tydzień)
        offs_mut = mutation(offspring, prob_od_mut=probability, changes_no=liczba_zamian)

        # sprawdzenie poprawności otrzymanych po krzyzowaniu i mutacji osobników (bilans kaloryczny oraz upper bound lodowki)
        offspings_checked = check_offspring(offs_mut, lista_produktow)
```

```python
        # dodanie do starych osobnikow, nowo powstalych
        x = replace_old_pop_with_new_one(pull_parents_form_parents_longer(copy_parents), offspings_checked)

        if x[0][1] < najlepsze_rozwiazanie:
            print('iteracja = ', i, 'f.celu = ', x[0][1])
            najlepsze_rozwiazanie = x[0][1]

        # len(x) jest miedzy [len(rodzice), 2*len(rodzice)]
        init_specimen = change_new_popul_to_other_format(x)
        i += 1

    return x
```

4. Aplikacja

Projekt podzielony jest zgodnie z praktykami programistycznymi. W folderze scr znajdują się osobno pliki: data_structures.py, genetic_algo.py i tabu_solution.py. Znajdują się tam odpowiednio ogólne struktury danych wykorzystane w obu algorytmach oraz implementację dwóch algorytmów, które wywoływane są w pliku main.py.

Parametry możemy ustawić w odpowiednich miejscach na początku pliku main.py.
Mamy możliwość zmieniać wagę i kaloryczność danego produktu ([waga, kaloryczność]).
Możemy zmienić także badany okres czasu - zmienna calendar. Podajemy początkowy i końcowy czas - format: dzień, miesiąc, rok

```
lista_produktow3 = np.array([[4.900e-01, 6.580e+02],
                             [4.400e-01, 8.960e+02],
                             [1.000e-01, 8.500e+02],
                             [2.200e-01, 1.145e+03],
                             [2.700e-01, 5.880e+02],
                             [1.700e-01, 8.650e+02],
                             [6.000e-02, 7.740e+02],
                             [1.500e-01, 9.500e+02],
                             [4.500e-01, 6.980e+02],
                             [2.200e-01, 9.710e+02]])


class Data:
    lista_produktow = lista_produktow3
    kalendarz = ds.return_calendar(3, 1, 2022, 25, 1, 2022)
```

Poniżej znajdują się ustawienia parametrów algorytmów.

```
class ParamsToGeneticAlgo:
    iteration = 1000
    prawdopobienstwo_mutacji = 0.9
    ilosc_osobnikow_do_reprodukcji = 20


class ParamsToTabuSearch:
    iteration = 100
```

W pliku main.py na końcu znajduje się funkcja główna, w której możemy wybrać, który algorytm chcemy testować. Algorytm Tabu Search zostanie wybrany, gdy zmienna wybierz_metode_genetic_algo będzie False, natomiast ustawienie True uruchomi nam algorytm genetyczny. Uruchomienie pliku main.py spowoduje działanie całej aplikacji.

```python
if __name__ == '__main__':

    params_to_genetic_algo = ParamsToGeneticAlgo
    data = Data

    params_to_tabu_search = ParamsToTabuSearch

    # Tu wybierz którą metodę chcesz liczyć
    # True = algorytm genetyczny
    # False = Tabu Search
    wybierz_metode_genetic_algo: bool = False

    if wybierz_metode_genetic_algo is True:
        genetic_algo_print(iteracje=params_to_genetic_algo.iteration, lista_produktow=data.lista_produktow,
                           calendar=data.kalendarz,
                           ilosc_osobnikow_pierw=params_to_genetic_algo.ilosc_osobnikow_do_reprodukcji,
                           prawdopodobienstwo_wyst_mutacji=params_to_genetic_algo.prawdopobienstwo_mutacji)
    else:
        tabu_search_print(data, params_to_tabu_search)
```

Rozwiązanie pojawi się w terminalu. Wszystko powinno być opisane jednoznacznie. W wyniku otrzymamy kolejno ilość wyjść do sklepu (0 - nie idziemy, 1 - idziemy), listę produktów zakupionych (0 - dany produkt nie kupiony, 1 - dany produkt kupiony) oraz wartość bilansu kalorycznego (0 - bilans dzienny spełniony, wartość ujemna - tyle kalorii zabrakło).

5. Eksperymenty

W ramach badań nad algorytmami zostały wykonane testy dla różnych parametrów algorytmów, jak i zmiany wartości ograniczeń. Poniżej znajduje się udokumentowana część wyników takich eksperymentów.

Ilość iteracji: 500; Ilość dni 14

```
======= TABU SEARCH =======


[[5.600e-01 1.335e+03]
 [2.500e-01 1.274e+03]
 [3.800e-01 6.570e+02]
 [7.000e-02 1.258e+03]
 [1.100e-01 8.410e+02]
 [5.500e-01 1.146e+03]
 [3.200e-01 9.850e+02]
 [2.000e-02 8.490e+02]
 [1.400e-01 1.235e+03]
 [1.000e-01 1.343e+03]]
Dla iteracji:  500
Początkowe rozwiazanie:  14
Najlepsze rozwiazanie   : 6 , it: 4
Najlepsze rozwiazanie   : 5 , it: 111
d: 1  lst: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 1  lst: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
d: 1  lst: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 1  lst: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 1  lst: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

Ilość iteracji: 1000; Ilość dni 14

```
======= TABU SEARCH =======



[[5.600e-01 1.335e+03]
 [2.500e-01 1.274e+03]
 [3.800e-01 6.570e+02]
 [7.000e-02 1.258e+03]
 [1.100e-01 8.410e+02]
 [5.500e-01 1.146e+03]
 [3.200e-01 9.850e+02]
 [2.000e-02 8.490e+02]
 [1.400e-01 1.235e+03]
 [1.000e-01 1.343e+03]]
Dla iteracji:  1000
Początkowe rozwiazanie:  14
Najlepsze rozwiazanie   : 5 , it: 150
d: 1  lst: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
d: 1  lst: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 1  lst: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 1  lst: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 1  lst: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

Ilość iteracji: 500; Ilość dni 14; Prawdopodobieństwo mutacji 0.1

```
======= Algorytm genetyczny =======


[[5.600e-01 1.335e+03]
 [2.500e-01 1.274e+03]
 [3.800e-01 6.570e+02]
 [7.000e-02 1.258e+03]
 [1.100e-01 8.410e+02]
 [5.500e-01 1.146e+03]
 [3.200e-01 9.850e+02]
 [2.000e-02 8.490e+02]
 [1.400e-01 1.235e+03]
 [1.000e-01 1.343e+03]]
kryterium stopu, iteracje = 500
iteracja =  0 f.celu =  14
iteracja =  45 f.celu =  13
iteracja =  60 f.celu =  12
iteracja =  80 f.celu =  11
iteracja =  94 f.celu =  10
iteracja =  272 f.celu =  9
Najlepsze rozwiązanie:    9
d: 1  lst: [1, 1, 1, 1, 1, 1, 1, 0, 1, 0]
d: 1  lst: [0, 1, 0, 1, 1, 1, 0, 1, 1, 0]
d: 1  lst: [0, 1, 1, 1, 0, 0, 1, 0, 0, 0]
d: 1  lst: [0, 0, 0, 0, 1, 1, 0, 0, 0, 0]
d: 1  lst: [0, 1, 0, 0, 1, 0, 0, 1, 1, 0]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 1  lst: [1, 1, 1, 1, 1, 1, 1, 1, 0, 1]
d: 1  lst: [0, 0, 0, 0, 1, 0, 0, 1, 0, 1]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 1  lst: [0, 0, 1, 0, 0, 1, 0, 1, 1, 1]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 1  lst: [0, 0, 0, 1, 0, 1, 1, 0, 1, 0]
```

Ilość iteracji: 1000; Ilość dni 14; Prawdopodobieństwo mutacji 0.1

```
======= Algorytm genetyczny =======


[[5.600e-01 1.335e+03]
 [2.500e-01 1.274e+03]
 [3.800e-01 6.570e+02]
 [7.000e-02 1.258e+03]
 [1.100e-01 8.410e+02]
 [5.500e-01 1.146e+03]
 [3.200e-01 9.850e+02]
 [2.000e-02 8.490e+02]
 [1.400e-01 1.235e+03]
 [1.000e-01 1.343e+03]]
kryterium stopu, iteracje = 1000
iteracja =  0 f.celu =  14
iteracja =  49 f.celu =  13
iteracja =  114 f.celu =  12
iteracja =  121 f.celu =  11
iteracja =  564 f.celu =  10
Najlepsze rozwiązanie:    10
d: 1  lst: [1, 1, 1, 1, 0, 1, 1, 1, 0, 0]
d: 1  lst: [1, 1, 1, 0, 0, 0, 1, 0, 1, 1]
d: 1  lst: [1, 1, 1, 0, 0, 1, 0, 0, 0, 1]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 1  lst: [0, 1, 1, 1, 0, 1, 1, 1, 0, 0]
d: 1  lst: [0, 0, 0, 0, 1, 0, 1, 1, 0, 1]
d: 1  lst: [1, 0, 0, 0, 0, 0, 1, 1, 0, 0]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 1  lst: [1, 0, 0, 1, 0, 1, 1, 0, 1, 1]
d: 1  lst: [0, 0, 0, 1, 0, 1, 1, 0, 1, 1]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 1  lst: [0, 0, 1, 1, 1, 0, 0, 1, 0, 0]
d: 1  lst: [1, 0, 1, 1, 0, 1, 0, 0, 1, 0]
```

Ilość iteracji: 1000; Ilość dni 14; Prawdopodobieństwo mutacji 0.5

```
======= Algorytm genetyczny =======


[[5.600e-01 1.335e+03]
 [2.500e-01 1.274e+03]
 [3.800e-01 6.570e+02]
 [7.000e-02 1.258e+03]
 [1.100e-01 8.410e+02]
 [5.500e-01 1.146e+03]
 [3.200e-01 9.850e+02]
 [2.000e-02 8.490e+02]
 [1.400e-01 1.235e+03]
 [1.000e-01 1.343e+03]]
kryterium stopu, iteracje = 1000
iteracja =  0 f.celu =  14
iteracja =  7 f.celu =  13
iteracja =  26 f.celu =  12
iteracja =  37 f.celu =  11
iteracja =  41 f.celu =  10
iteracja =  81 f.celu =  9
Najlepsze rozwiązanie:    9
d: 1  lst: [1, 1, 0, 1, 1, 0, 0, 1, 1, 1]
d: 1  lst: [1, 1, 1, 1, 0, 0, 1, 0, 0, 0]
d: 1  lst: [1, 1, 1, 1, 0, 0, 1, 0, 0, 0]
d: 1  lst: [1, 1, 1, 1, 0, 1, 0, 1, 0, 1]
d: 1  lst: [1, 1, 0, 0, 0, 1, 1, 0, 1, 1]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 1  lst: [1, 1, 0, 0, 1, 1, 1, 0, 0, 0]
d: 1  lst: [0, 1, 1, 1, 1, 1, 1, 0, 0, 0]
d: 1  lst: [1, 1, 0, 1, 1, 0, 1, 0, 0, 0]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 1  lst: [0, 0, 0, 0, 1, 0, 1, 0, 1, 1]
```

Ilość iteracji: 1000; Ilość dni 14; Prawdopodobieństwo mutacji 0.9

```
======= Algorytm genetyczny =======


[[5.600e-01 1.335e+03]
 [2.500e-01 1.274e+03]
 [3.800e-01 6.570e+02]
 [7.000e-02 1.258e+03]
 [1.100e-01 8.410e+02]
 [5.500e-01 1.146e+03]
 [3.200e-01 9.850e+02]
 [2.000e-02 8.490e+02]
 [1.400e-01 1.235e+03]
 [1.000e-01 1.343e+03]]
kryterium stopu, iteracje = 1000
iteracja =  0 f.celu =  14
iteracja =  7 f.celu =  13
iteracja =  16 f.celu =  12
iteracja =  34 f.celu =  11
iteracja =  55 f.celu =  10
iteracja =  105 f.celu =  9
iteracja =  648 f.celu =  8
Najlepsze rozwiązanie:    8
d: 1  lst: [1, 0, 1, 1, 0, 0, 0, 1, 1, 0]
d: 1  lst: [1, 0, 1, 0, 1, 1, 1, 0, 1, 0]
d: 1  lst: [1, 1, 1, 1, 0, 0, 1, 1, 0, 0]
d: 1  lst: [1, 1, 1, 0, 1, 1, 1, 0, 0, 1]
d: 1  lst: [0, 1, 0, 0, 1, 1, 1, 0, 1, 0]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 1  lst: [1, 0, 1, 1, 0, 1, 1, 1, 1, 1]
d: 1  lst: [1, 1, 0, 1, 1, 0, 0, 1, 1, 0]
d: 1  lst: [1, 1, 0, 0, 0, 1, 0, 0, 0, 0]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Ilość iteracji: 10 000; Ilość dni 14; Prawdopodobieństwo mutacji 0.9

```
======= Algorytm genetyczny =======


[[6.000e-02 9.300e+02]
 [4.000e-01 8.160e+02]
 [3.000e-01 1.242e+03]
 [5.000e-01 1.498e+03]
 [6.400e-01 8.930e+02]
 [2.800e-01 1.134e+03]
 [1.700e-01 6.590e+02]
 [6.100e-01 1.315e+03]
 [1.800e-01 3.330e+02]
 [2.100e-01 8.580e+02]]
kryterium stopu, iteracje = 10000
iteracja =  0 f.celu =  14
iteracja =  14 f.celu =  13
iteracja =  24 f.celu =  12
iteracja =  34 f.celu =  11
iteracja =  132 f.celu =  10
iteracja =  4259 f.celu =  9
Najlepsze rozwiązanie:    9
d: 1  lst: [1, 1, 1, 1, 1, 0, 0, 0, 0, 1]
d: 1  lst: [0, 1, 0, 1, 1, 0, 1, 0, 1, 1]
d: 1  lst: [1, 1, 1, 0, 1, 1, 1, 1, 0, 0]
d: 1  lst: [1, 1, 1, 1, 1, 1, 0, 1, 0, 1]
d: 1  lst: [1, 1, 1, 1, 1, 1, 0, 0, 0, 1]
d: 1  lst: [1, 1, 0, 1, 1, 1, 0, 1, 1, 0]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 1  lst: [1, 0, 1, 1, 0, 1, 1, 1, 1, 1]
d: 1  lst: [0, 0, 0, 0, 1, 0, 0, 0, 0, 1]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 1  lst: [1, 0, 0, 1, 1, 0, 0, 1, 0, 1]
```

Ilość iteracji: 1000; Ilość dni 14; Prawdopodobieństwo mutacji 0.9; lista produktów nr 2

```
======= Algorytm genetyczny =======


[[6.000e-02 9.300e+02]
 [4.000e-01 8.160e+02]
 [3.000e-01 1.242e+03]
 [5.000e-01 1.498e+03]
 [6.400e-01 8.930e+02]
 [2.800e-01 1.134e+03]
 [1.700e-01 6.590e+02]
 [6.100e-01 1.315e+03]
 [1.800e-01 3.330e+02]
 [2.100e-01 8.580e+02]]
kryterium stopu, iteracje = 1000
iteracja =  0 f.celu =  14
iteracja =  12 f.celu =  13
iteracja =  19 f.celu =  12
iteracja =  27 f.celu =  11
iteracja =  107 f.celu =  10
iteracja =  468 f.celu =  9
Najlepsze rozwiązanie:    9
d: 1  lst: [0, 0, 0, 1, 1, 1, 0, 0, 0, 1]
d: 1  lst: [1, 1, 1, 0, 1, 0, 0, 0, 1, 1]
d: 1  lst: [1, 0, 1, 0, 1, 1, 0, 0, 0, 1]
d: 1  lst: [0, 1, 1, 0, 1, 1, 0, 0, 1, 1]
d: 1  lst: [1, 1, 1, 1, 0, 0, 1, 0, 1, 1]
d: 1  lst: [1, 1, 0, 1, 1, 0, 1, 1, 0, 1]
d: 1  lst: [1, 1, 0, 0, 1, 0, 1, 1, 0, 0]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 1  lst: [1, 0, 0, 1, 0, 0, 1, 1, 0, 1]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 1  lst: [1, 1, 1, 0, 1, 0, 1, 1, 0, 1]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Ilość iteracji: 1000; Ilość dni 29

```
======= TABU SEARCH =======



[[5.600e-01 1.335e+03]
 [2.500e-01 1.274e+03]
 [3.800e-01 6.570e+02]
 [7.000e-02 1.258e+03]
 [1.100e-01 8.410e+02]
 [5.500e-01 1.146e+03]
 [3.200e-01 9.850e+02]
 [2.000e-02 8.490e+02]
 [1.400e-01 1.235e+03]
 [1.000e-01 1.343e+03]]
Dla iteracji:  1000
Początkowe rozwiazanie:  29
Najlepsze rozwiazanie   : 10 , it: 13
```

```
d: 1  lst: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
d: 1  lst: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 1  lst: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 1  lst: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 1  lst: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 1  lst: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 1  lst: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 1  lst: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 1  lst: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 1  lst: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d: 0  lst: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

Ilość iteracji: 1000; Ilość dni 29; lista produktów nr 2

```
 ======= TABU SEARCH =======


 [[6.000e-02 9.300e+02]
  [4.000e-01 8.160e+02]
  [3.000e-01 1.242e+03]
  [5.000e-01 1.498e+03]
  [6.400e-01 8.930e+02]
  [2.800e-01 1.134e+03]
  [1.700e-01 6.590e+02]
  [6.100e-01 1.315e+03]
  [1.800e-01 3.330e+02]
  [2.100e-01 8.580e+02]]
 Dla iteracji:  1000
 Początkowe rozwiazanie:  29
```

Algorytm nie znalazł lepszego rozwiązania

Ilość iteracji: 1000; Ilość dni 29; lista produktów nr 3

```
======= TABU SEARCH =======


[[4.900e-01 6.580e+02]
 [4.400e-01 8.960e+02]
 [1.000e-01 8.500e+02]
 [2.200e-01 1.145e+03]
 [2.700e-01 5.880e+02]
 [1.700e-01 8.650e+02]
 [6.000e-02 7.740e+02]
 [1.500e-01 9.500e+02]
 [4.500e-01 6.980e+02]
 [2.200e-01 9.710e+02]]
Dla iteracji:  1000
Początkowe rozwiazanie:  29
Najlepsze rozwiazanie   : 12 , it: 829
```

Ilość iteracji: 1000; Ilość dni 29; lista produktów nr 3 - inna próba
W tym przypadku nie działa

```
======= TABU SEARCH =======


[[4.900e-01 6.580e+02]
 [4.400e-01 8.960e+02]
 [1.000e-01 8.500e+02]
 [2.200e-01 1.145e+03]
 [2.700e-01 5.880e+02]
 [1.700e-01 8.650e+02]
 [6.000e-02 7.740e+02]
 [1.500e-01 9.500e+02]
 [4.500e-01 6.980e+02]
 [2.200e-01 9.710e+02]]
Dla iteracji:  1000
Początkowe rozwiazanie:  29
```

```
d: 1  lst: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
d: 1  lst: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
d: 1  lst: [1, 1, 1, 1, 0, 1, 1, 1, 1, 0]
d: 1  lst: [1, 1, 0, 0, 1, 0, 0, 1, 1, 0]
d: 1  lst: [1, 0, 0, 0, 0, 1, 1, 0, 0, 0]
d: 1  lst: [1, 0, 1, 0, 1, 0, 0, 0, 0, 1]
d: 1  lst: [0, 1, 0, 1, 1, 0, 0, 0, 1, 0]
d: 1  lst: [0, 1, 0, 0, 0, 0, 0, 1, 1, 0]
d: 1  lst: [0, 0, 0, 1, 1, 1, 0, 1, 1, 0]
d: 1  lst: [0, 0, 0, 0, 1, 0, 1, 0, 1, 1]
d: 1  lst: [1, 0, 0, 0, 1, 0, 1, 0, 0, 0]
d: 1  lst: [1, 0, 0, 0, 0, 0, 0, 1, 1, 0]
d: 1  lst: [0, 1, 0, 0, 1, 1, 0, 0, 0, 1]
d: 1  lst: [0, 1, 1, 0, 0, 0, 1, 0, 1, 0]
d: 1  lst: [0, 0, 0, 0, 1, 1, 0, 0, 1, 1]
d: 1  lst: [1, 0, 0, 1, 0, 0, 1, 0, 0, 1]
d: 1  lst: [1, 0, 0, 1, 1, 0, 1, 0, 0, 0]
d: 1  lst: [0, 1, 0, 1, 0, 0, 1, 1, 0, 0]
d: 1  lst: [0, 0, 0, 1, 0, 0, 1, 1, 1, 0]
d: 1  lst: [0, 1, 1, 0, 0, 0, 0, 1, 1, 0]
d: 1  lst: [0, 0, 0, 1, 1, 0, 0, 1, 0, 0]
d: 1  lst: [0, 1, 0, 0, 0, 1, 0, 0, 1, 0]
d: 1  lst: [1, 0, 0, 1, 1, 0, 1, 0, 0, 0]
d: 1  lst: [0, 1, 0, 0, 0, 0, 1, 1, 0, 0]
d: 1  lst: [0, 0, 0, 1, 1, 0, 1, 1, 0, 0]
d: 1  lst: [1, 1, 1, 0, 0, 0, 0, 0, 0, 1]
d: 1  lst: [0, 0, 0, 0, 1, 1, 0, 0, 0, 1]
d: 1  lst: [0, 0, 0, 1, 1, 0, 1, 1, 0, 0]
d: 1  lst: [1, 0, 0, 0, 1, 0, 0, 1, 0, 1]
```

Ilość iteracji: 1000; Ilość dni 29; Prawdopodobieństwo mutacji 0.1

```
======= Algorytm genetyczny =======


[[4.900e-01 6.580e+02]
 [4.400e-01 8.960e+02]
 [1.000e-01 8.500e+02]
 [2.200e-01 1.145e+03]
 [2.700e-01 5.880e+02]
 [1.700e-01 8.650e+02]
 [6.000e-02 7.740e+02]
 [1.500e-01 9.500e+02]
 [4.500e-01 6.980e+02]
 [2.200e-01 9.710e+02]]
kryterium stopu, iteracje = 1000
iteracja =  0 f.celu =  29
iteracja =  107 f.celu =  28
iteracja =  129 f.celu =  27
iteracja =  130 f.celu =  26
iteracja =  707 f.celu =  25
Najlepsze rozwiązanie:    25
```

```
d: 1  lst: [1, 1, 1, 1, 1, 1, 1, 1, 0, 1]
d: 1  lst: [1, 1, 0, 1, 1, 1, 1, 1, 0, 1]
d: 1  lst: [1, 0, 1, 0, 0, 1, 0, 1, 1, 1]
d: 1  lst: [0, 0, 1, 1, 0, 1, 1, 1, 0, 1]
d: 1  lst: [0, 1, 1, 1, 1, 0, 0, 0, 0, 0]
d: 1  lst: [0, 0, 1, 1, 0, 1, 0, 0, 1, 1]
d: 1  lst: [1, 0, 0, 1, 0, 1, 0, 0, 1, 0]
d: 1  lst: [0, 0, 0, 0, 0, 1, 1, 0, 0, 0]
d: 1  lst: [0, 0, 0, 0, 0, 1, 0, 1, 0, 0]
d: 1  lst: [1, 1, 0, 1, 1, 0, 0, 0, 1, 1]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 1  lst: [0, 0, 0, 0, 1, 1, 1, 1, 1, 1]
d: 1  lst: [0, 1, 0, 1, 0, 0, 1, 0, 0, 0]
d: 1  lst: [0, 1, 1, 1, 1, 0, 0, 1, 0, 1]
d: 1  lst: [0, 1, 1, 0, 0, 0, 0, 1, 0, 1]
d: 1  lst: [0, 1, 0, 0, 0, 0, 1, 1, 1, 1]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 1  lst: [0, 1, 1, 1, 0, 1, 1, 0, 0, 1]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 1  lst: [1, 1, 1, 0, 0, 1, 1, 1, 1, 0]
d: 1  lst: [0, 1, 0, 0, 0, 1, 1, 0, 0, 0]
d: 1  lst: [1, 0, 0, 0, 1, 1, 0, 0, 1, 0]
d: 1  lst: [0, 0, 0, 1, 1, 1, 0, 0, 0, 1]
d: 1  lst: [0, 1, 0, 1, 0, 1, 1, 1, 0, 1]
d: 1  lst: [0, 0, 0, 1, 0, 1, 0, 0, 0, 0]
d: 1  lst: [1, 0, 0, 0, 1, 0, 0, 1, 1, 0]
d: 1  lst: [0, 0, 1, 1, 0, 0, 0, 1, 0, 1]
d: 1  lst: [0, 0, 1, 0, 1, 1, 0, 1, 0, 0]
```

Ilość iteracji: 1000; Ilość dni 29; Prawdopodobieństwo mutacji 0.5

```
======= Algorytm genetyczny =======


[[4.900e-01 6.580e+02]
 [4.400e-01 8.960e+02]
 [1.000e-01 8.500e+02]
 [2.200e-01 1.145e+03]
 [2.700e-01 5.880e+02]
 [1.700e-01 8.650e+02]
 [6.000e-02 7.740e+02]
 [1.500e-01 9.500e+02]
 [4.500e-01 6.980e+02]
 [2.200e-01 9.710e+02]]
kryterium stopu, iteracje = 1000
iteracja =  0 f.celu =  29
iteracja =  25 f.celu =  28
iteracja =  39 f.celu =  27
iteracja =  88 f.celu =  26
iteracja =  171 f.celu =  25
Najlepsze rozwiązanie:    25
```

```
d: 1  lst: [1, 0, 1, 1, 1, 0, 1, 0, 0, 1]
d: 1  lst: [0, 1, 0, 0, 1, 1, 0, 1, 1, 1]
d: 1  lst: [1, 1, 1, 0, 1, 1, 0, 1, 1, 0]
d: 1  lst: [1, 1, 1, 1, 0, 1, 1, 1, 1, 1]
d: 1  lst: [1, 1, 0, 1, 1, 0, 1, 1, 0, 1]
d: 1  lst: [0, 0, 0, 0, 1, 1, 0, 1, 0, 0]
d: 1  lst: [0, 0, 1, 0, 0, 0, 0, 0, 1, 0]
d: 1  lst: [1, 0, 1, 0, 0, 1, 1, 1, 0, 0]
d: 1  lst: [0, 1, 1, 1, 1, 1, 0, 0, 0, 1]
d: 1  lst: [0, 0, 1, 1, 1, 0, 1, 0, 1, 0]
d: 1  lst: [0, 0, 1, 1, 0, 0, 0, 0, 1, 1]
d: 1  lst: [1, 0, 0, 1, 0, 0, 0, 0, 1, 1]
d: 1  lst: [1, 1, 0, 0, 1, 1, 0, 0, 0, 0]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 1  lst: [1, 1, 1, 1, 0, 0, 1, 1, 0, 0]
d: 1  lst: [1, 0, 0, 0, 0, 0, 1, 0, 0, 1]
d: 1  lst: [1, 0, 1, 1, 0, 1, 1, 0, 0, 1]
d: 1  lst: [0, 1, 0, 0, 1, 0, 0, 0, 0, 1]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 1  lst: [1, 1, 0, 0, 0, 0, 1, 0, 0, 1]
d: 1  lst: [1, 0, 1, 0, 0, 1, 1, 1, 1, 1]
d: 1  lst: [1, 0, 0, 0, 0, 0, 0, 1, 1, 1]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 1  lst: [0, 0, 0, 1, 0, 0, 0, 1, 1, 1]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 1  lst: [0, 1, 1, 1, 0, 0, 0, 1, 0, 1]
d: 1  lst: [1, 1, 0, 0, 0, 0, 1, 0, 0, 0]
d: 1  lst: [0, 1, 0, 0, 0, 0, 1, 0, 0, 0]
d: 1  lst: [1, 1, 0, 0, 0, 1, 0, 0, 1, 1]
```

Ilość iteracji: 1000; Ilość dni 29; Prawdopodobieństwo mutacji 0.9

```
======= Algorytm genetyczny =======


[[4.900e-01 6.580e+02]
 [4.400e-01 8.960e+02]
 [1.000e-01 8.500e+02]
 [2.200e-01 1.145e+03]
 [2.700e-01 5.880e+02]
 [1.700e-01 8.650e+02]
 [6.000e-02 7.740e+02]
 [1.500e-01 9.500e+02]
 [4.500e-01 6.980e+02]
 [2.200e-01 9.710e+02]]
kryterium stopu, iteracje = 1000
iteracja =  0 f.celu =  29
iteracja =  18 f.celu =  28
iteracja =  34 f.celu =  27
iteracja =  59 f.celu =  26
iteracja =  359 f.celu =  25
iteracja =  479 f.celu =  24
Najlepsze rozwiązanie:    24
```

```
d: 1  lst: [1, 1, 1, 1, 0, 1, 0, 1, 1, 1]
d: 1  lst: [1, 1, 0, 1, 1, 1, 1, 0, 0, 1]
d: 1  lst: [1, 1, 1, 0, 0, 0, 0, 1, 1, 1]
d: 1  lst: [0, 0, 1, 1, 0, 1, 0, 0, 0, 1]
d: 1  lst: [1, 0, 1, 0, 0, 0, 0, 0, 0, 1]
d: 1  lst: [1, 1, 0, 0, 0, 0, 0, 0, 1, 1]
d: 1  lst: [0, 0, 1, 1, 1, 1, 1, 0, 0, 1]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 1  lst: [1, 0, 0, 0, 1, 1, 0, 0, 0, 1]
d: 1  lst: [0, 0, 1, 0, 1, 0, 0, 0, 1, 0]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 1  lst: [0, 0, 1, 0, 0, 1, 1, 0, 0, 1]
d: 1  lst: [0, 1, 1, 0, 1, 1, 0, 1, 0, 0]
d: 1  lst: [0, 0, 1, 1, 1, 1, 1, 1, 1, 0]
d: 1  lst: [1, 1, 0, 1, 0, 0, 1, 0, 0, 0]
d: 1  lst: [0, 1, 0, 0, 1, 0, 1, 1, 1, 0]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 1  lst: [1, 1, 1, 1, 1, 0, 1, 0, 0, 1]
d: 1  lst: [1, 0, 0, 1, 1, 1, 1, 0, 0, 0]
d: 1  lst: [1, 1, 1, 0, 0, 0, 1, 1, 0, 1]
d: 1  lst: [1, 0, 0, 1, 1, 0, 0, 1, 0, 0]
d: 1  lst: [1, 1, 0, 0, 0, 1, 0, 1, 1, 0]
d: 1  lst: [1, 0, 1, 0, 0, 1, 0, 1, 1, 0]
d: 1  lst: [1, 1, 0, 1, 0, 1, 0, 1, 0, 1]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d: 1  lst: [0, 0, 1, 0, 1, 0, 1, 1, 1, 1]
d: 1  lst: [1, 0, 0, 1, 0, 1, 1, 1, 0, 1]
d: 1  lst: [0, 1, 0, 1, 0, 0, 0, 1, 0, 0]
d: 0  lst: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

6. Wnioski

Tabu Search:

Dla zaimplementowanego przez nas algorytmu Tabu Search, jest bardziej problematyczne znalezienie rozwiązania przy zadanej większej ilości dni. Również w zależności od danych wejściowych (listy produktów) zależy czy algorytm znajdzie poprawione rozwiązanie. Otrzymane wyniki wydają się być poprawne, różnią się jednak bardzo od tych otrzymanych a algorytmie genetycznym. Zwiększenie ilości iteracji zwiększało prawdopodobieństwo znalezienia lepszego rozwiązania.

Algorytmy Genetyczne

Zaimplementowany przez nas algorytm genetyczny w każdym przypadku znajdował poprawione rozwiązanie. TO jak bardzo zminimalizowana była wartość funkcji celu zależało w największej części od prawdopodobieństwa wystąpienia mutacji po krzyżowaniu.
Nie wpływało ono jednak tak bardzo na szybkość znalezienia tego rozwiązania, co na to, że rozwiązanie prawdopodobnie nie lądowało w minimum lokalnym (mogło z niego wyjść dzięki mutacji).
Zwiększenie ilości iteracji zwiększało prawdopodobieństwo znalezienia lepszego rozwiązania.

7. Podsumowanie

Wykonanie ćwiczenia pozwoliło przybliżyć nam temat algorytmów, w szczególności Tabu Search i algorytmu genetycznego. Poza typową implementacją algorytmu przekonaliśmy się, że rozwiązywanie danego problemu może być nie być całkiem proste i trzeba zdecydować, jak uprościć wybrane zagadnienie. W procesie tworzenia kodu w naszym przypadku niejednokrotnie spotykaliśmy przeszkody w postaci trudnego dopasowania problemu i ograniczeń do algorytmu, który powinien się wykonać i finalnie dać nam poprawne rozwiązanie. Podczas implementacji zrezygnowaliśmy m.in. z uwzględniania przydatności danego produktu, uwzględniania świąt i niedziel nie handlowych oraz musieliśmy podnieść ograniczenie udźwigu plecaka. Problematycznym ograniczeniem była choćby kaloryczność, przez co trudno było nam poprawiać dane rozwiązanie.

8. Literatura

- Wykład z Badań operacyjnych 2
- Tabu Search | Python | Np-hard | Metaheuristics | Heuristics | mathematical optimization| Scheduling problem | The Startup (medium.com)
- Tabu Search — gentle introduction | by Mohanad Kaleia | Medium
- Genetic Algorithm for Traveling Salesman Problem with Modified Cycle Crossover Operator (hindawi.com)
- Microsoft Word - 10026 - pdfMachine from Broadgun Software, http://pdfmachine.com, a great PDF writer utility! (mnkjournals.com)

9. Podział pracy

	Barbara Pobiedzińska	Marcin Biela	Tomasz Brania
Model Matematyczny	:)	:)	:)
Tabu Search	:D	:D	:)
Algorytm genetyczny	:D	:D	:)
Eksperymenty	:)	:)	:)
Struktura projektu i obsługa GitHuba	:|	:D	:|
Dokumentacja	:|	:|	:D

Link do githuba klik :)