Skład zespołu:
1. Adrian Kotowski
2. Kamil Hajduk
3. Michał Bogucki

Prowadzący:
Grzegorz Blinowski

SPRAWOZDANIE
PSZT GB.AE.2

I. Treść zadania

GB.AE.2

Napisać program "Plan szkoleń" automatycznie przydzielający grupy ludzi (uczestników) do sal szkoleniowych. W pewnym przedziale czasowym organizowane są szkolenia (spotkania). Szkolenia są poświęcone różnym zagadnieniom (T1, T2, …). W każdym szkoleniu jednorazowo bierze udział określona i znana liczba osób T1:4, T2:10, itd. - liczba osób przydzielonych do szkolenia może być mniejsza, ale nie może być większa. Do dyspozycji jest pewna liczba sal, każda o określonej pojemności (np. 4-20) miejsc: S1:4, S2:20, S3:10, Oczywiście, w jednej sali na raz może odbywać się tylko jedno szkolenie. Każdy uczestnik określa jakie zagadnienia szkoleń go interesują: U1:T1,T2; U2:T3, U3:T1,T3; itd. Liczba szkoleń danego typu jest określona z góry: T#1:2 T#2:2, T#3:4, itd. Sumarycznie może się odbyć mniej szkoleń niż założono, ale nie więcej. Szkolenie każdego typu trwa określoną stałą liczbę godzin: TG1:4, TG2:8, TG3:3, itd. (szkolenie to ciągły blok czasowy, podział jest niedozwolony). Dodatkowo, uczestnicy mogą wyrazić preferencje co do "towarzystwa", np. uczestnik U1 chce być w grupie z U2 – U6 (preferencje te powinny być brane pod uwagę, ale nie muszą być bezwzględnie spełnione). Wynikiem działania programu powinien być grafik szkoleń, tj mapowanie uczestników na sale i szkolenia. Należy przyjąć, że szkolenia mogą odbywać się np. w godzinach 8:00 – 18:00, i że sumarycznie grafik szkoleń może objąć więcej niż jeden dzień, ale liczba dni jest ograniczona (np. maksimum 3). Uwaga #1: warunki mogą powodować, że określeni uczestnicy, nie będą mogli uczestniczyć we wszystkich wybranych przez siebie szkoleniach. Uwaga #2: jak wyżej wspomniano liczba szkoleń każdego typu jest ograniczona, należy to kryterium rozumieć np. jako wynikające z określonych zasobów "trenderów". Grafik powinien być w miarę możności pozbawiony okienek i minimalizować czas trwania wszystkich szkoleń łącznie.

II. Interpretacja treści zadania.

Analizując treść powyższego zadania dokonano dodatkowych założeń, podzielono problem na dwa mniejsze (etapy):

1. Dobór uczestników do realizacji kursów
2. Przydzielenie realizacji do sali w dostępnym dla niej terminie

I. Etap: Dobór uczestników do realizacji kursów

Algorytm ewolucyjny operuje na parach uczestnik-kurs realizacji.
Każde pojedyncze wywołanie algorytmu zwraca parę, która oznacza faktyczne przypisanie uczestnika do realizacji kursu.
Algorytm wyklucza przypisanie uczestnika do realizacji niepreferowanego przez niego kursu.
Niemożliwe jest zapisanie na te same realizacje kursu oraz dwie różne realizacje tego samego kursu. Zapis na wiele realizacji różnych kursów jest dopuszczalny.
Uwzględniono limit miejsc na poszczególne realizację.

Wartość wylosowanego potomka uczestnik-realizacja określa **funkcja celu nr 1: F1** (opisana dokładniej w innym paragrafie). Uwzględniono w niej takie wartości jak ilość preferowanych kursów przez uczestnika i w mniejszym stopniu jego preferencje osobowe.

II. Etap: Przydzielenie realizacji kursów do terminów sal

W tym etapie algorytm dobiera pary realizacja kursu-sala.
Wynikiem algorytmu jest podobnie jak wcześniej dodanie analogicznego przypisania.
Koniecznym jest pilnowanie warunku, żeby liczba miejsc na realizacji nie była większa od liczby miejsc na sali i aby sala miała wolny odpowiedni slot czasowy.

Potomek w drugim etapie ma też swoją **własną funkcję celu F2** (opisana dokładniej w innym paragrafie). Uwzględnia ona liczbę osób zapisanych na kurs, opłatę uczestnika za kurs, koszt wynajmu sali, koszt prelegenta.

Określanie godzin w grafiku dla realizacji przypisanej do sali przebiega na zasadzie stopniowego zapełniania godzin każdego dnia po kolei.
****
Dodatkowo niewskazane jest zapisanie się na realizacje odbywające się w tym samych czasie. Każda sala jest dostępna przez 10h dziennie od 8 do 18 przez trzy dni. W sali nie mogą się odbywać jednocześnie więcej niż jedna realizacja kursu.

Efektem końcowym jest grafik zajęć dla każdej sali, lista uczestników wraz z kursami, na które udało im się zapisać oraz wszystkie realizacje, które były brane pod uwagę podczas tworzenia grafiku.

III. Krótki opis funkcjonalny - “black-box”

3.1 Dane wejściowe

Schemat danych wejściowych
****

```
500         //tyle ma być uczestników
 40         //tyle ma być realizacji wszystkich kursów
  8         //tyle ma być wszystkich dostępnych sal
```

****
Dane uczestnika nr 1
Dane uczestnika nr 2
…
Dane uczestnika nr 500 //tyle ile zadeklarowano na początku
****
Dane realizacji nr 1
Dane realizacji nr 2
....
Dane realizacji nr 40 //tyle ile zadeklarowano na początku
****
Dane sali nr 1
Dane sali nr 2
....
Dane sali nr 8 //tyle ile zadeklarowano na początku
****

Format danych wejściowych Uczestnika:

1	Id uczestnika
(T)**1** ,…. ,(T)**n**	Zadeklarowane kursy
(P)**1** , … , (P)**m**	Preferowani współuczestnicy

Format danych wejściowych Realizacji:

8	Id realizacji
(T)**2**	Typ kursu
5	Długość trwania kursu
30	Max liczba uczestników
28	Liczba zapisanych uczestników
600	Koszt kursu
10000	Koszt prelegenta
5	Przydzielona sala (default **-1**)

Format danych wejściowych Sali:

(R) **1**	Id Sali
60	Liczba krzeseł
250	Godzinowy koszt wynajmu
6	Liczba zajętych godzin (max **30**)

3.2 Dane wyjściowe

3.2.1 Uczestnik:

```
Participant ID #9
Course you are enrolled on:
        Realization: #21,  Room: #7
        Realization: #25,  Room: #6
        Realization: #15,  Room: #0
        Realization: #19,  Room: #7

Participant ID #10
Course you are enrolled on:
        Realization: #16,  Room: #0

Participant ID #11
Course you are enrolled on:
        Realization: #31,  Room: #2
        Realization: #5,   Room: #5
```

3.2.2 Realizacja:

```
Realization:      38
    Fitness:      1600
    courseType    5
    courseLength  2
    Lease cost    250
    maxPartAmount 40
    enrolled      17
    hiringCost    3000
    courseCost    300
    roomId        7


Realization:      39
    Fitness:      3600
    courseType    5
    courseLength  2
    Lease cost    150
    maxPartAmount 40
    enrolled      23
    hiringCost    3000
    courseCost    300
    roomId        2




Realizations assigned: 35/40
```

3.2.3 Sala:

```
-----------------------------------------------
Room #2
    Seats: 35
    Lease cost: 150
    Occupied hours: 27
Hours schedule:
Day 1
    8:15    ->  #31     ppl 30/35
    9:15    ->  #31     ppl 30/35
    10:15   ->  #31     ppl 30/35
    11:15   ->  #31     ppl 30/35
    12:15   ->  #31     ppl 30/35
    13:15   ->  #31     ppl 30/35
    14:15   ->  #31     ppl 30/35
    15:15   ->  #17     ppl 28/35
    16:15   ->  #17     ppl 28/35
    17:15   ->  #-1
Day 2
    8:15    ->  #7      ppl 30/35
    9:15    ->  #7      ppl 30/35
    10:15   ->  #7      ppl 30/35
    11:15   ->  #7      ppl 30/35
    12:15   ->  #7      ppl 30/35
    13:15   ->  #7      ppl 30/35
    14:15   ->  #7      ppl 30/35
    15:15   ->  #39     ppl 23/35
    16:15   ->  #39     ppl 23/35
    17:15   ->  #-1
Day 3
    8:15    ->  #12     ppl 30/35
    9:15    ->  #12     ppl 30/35
    10:15   ->  #12     ppl 30/35
    11:15   ->  #12     ppl 30/35
    12:15   ->  #12     ppl 30/35
    13:15   ->  #12     ppl 30/35
    14:15   ->  #12     ppl 30/35
    15:15   ->  #4      ppl 21/35
    16:15   ->  #4      ppl 21/35
    17:15   ->  #-1


-----------------------------------------------
Room #3
    Seats: 55
    Lease cost: 350
    Occupied hours: 3
Hours schedule:
```

IV. Opis i uzasadnienie przyjętego rozwiązania.

Celem wspomnianych dwóch etapów projektu jest ustalenie dwóch zbiorów przypisań, rozumianych jako dwa zbiory osobników.
Realizując dobór uczestników do realizacji kursów przed przydzieleniem realizacji do sal, można uwzględnić otrzymane wcześniej dane o liczbie zapisanych na realizację w funkcji celu kolejnego etapu.

Wybrano **algorytm 1+1** ze względu na chęć rezygnacji z krzyżowania rodziców. Dzięki temu ograniczono złożoność obliczeniową pojedynczego wywołania algorytmu.

Na danym etapie działa pętla wywołująca w każdej iteracji algorytm ewolucyjny adekwatny do etapu.
Algorytm nastawiony jest pod kątem eksploatacji, czyli wyszukiwania maksimów lokalnych. Standardowo algorytm szuka najlepszego potomka, który jednocześnie pomyślnie przechodzi walidację, czyli jego rzeczywisty wymiar ma sens z punktu widzenia układania grafiku. Algorytm kończy działanie zwracając najlepszego znalezionego osobnika albo zwraca flagę braku osobnika w przypadku nie znalezienia jakiegokolwiek “poprawnego”. Ta druga opcja może wystąpić po jakimś czasie, czego dowodem jest przykład.
Jeśli na początku 60 uczestników pozwoli się zapisać na średnio 7 realizacji kursów na 15 to w naszej przestrzeni dwuwymiarowej ok. 50% osobników będzie poprawna.
Po uruchomieniu symulacji liczba poprawnych osobników zacznie spadać do ok. 0% ponieważ osobniki które do tej pory były poprawne przestaną nimi być bo np.:

- uczestnik został właśnie przypisany do jakiejś realizacji tego kursu
- nie można przypisać uczestnika do realizacji bo brakuje na nią wolnych miejsc

Zatem w pewnym momencie algorytm zacznie ciągle zwracać flagę braku potomka. Zliczając takie sytuacje w ostatnich kilku iteracjach przerwa się pętlę stosując jednocześnie **warunek stopu** dla całego mechanizmu.

Funkcja celu potomka uczestnik-realizacja **F1**

Jeśli realizacja jest typem kursu który deklarował uczestnik to

F1(uczestnik, realizacja) = LDK*590 + (LWZ^2)*30

W powyższej funkcji celu uwzględnimy dwa czynniki:

- **LDK (Liczba zadeklarowanych kursów przez uczestnika)**
  Preferuje się uczestników z większą ilością wybranych kursów, ponieważ Ci uczestnicy generują większy zysk i to może zachęcać innych uczestników, aby deklarowali więcej kursów, by czuć się preferowani w zapisach.
- **LWZ (Liczba preferowanych znajomych)**
  Wspiera się sytuacje, gdzie jak najwięcej osób może się znaleźć z jak największa liczbą swoich znajomych na realizacjach kursów.

Skutkiem ubocznym takiej funkcji celu jest zmniejszenie szans na zapis osób bez preferencji osób i mniejszą ilością zadeklarowanych kursów.

Funkcja celu potomka realizacja-sala **F2**

F2(realizacja, sala) = (ZYSK - STRATY) =
[(LZU * Koszt_kursu) - (KP + KNS * Dlugosc_kursu)]

W powyższej funkcji celu uwzględnimy dwa czynniki:

- **LZU (Liczba zapisanych uczestników)**
  Skoro z każdego zapisanego mamy zysk z wpisowego, to im więcej zapisanych tym lepiej.
- **KP (Koszt prelegenta)**
  Wymagane jest wynagrodzenie dla prowadzącego zajęcia.
- **KNS (Koszt najmu sali)**
  W przeliczeniu na godzinę

Wartość funkcji może być ujemna, co powoduje, że algorytm za wszelką cenę unika takich sytuacji. Zatem nie opłacalne realizacje kursu nie startują.

V. Planowany podział na komponenty i sposób komunikacji między nimi:
jakie klasy, co zawierają (metody, kontenery, zmienne, w tym flagi),

UCZESTNIK

{
INT **ID_Uczestnika**
Map<INT **ID_kursu**, INT **ID_Realizacji**> **Deklaracje**
//Inicjalizowane: -1 oznacza **niesprawdzony,** wartość indeksu przypisanej realizacji jest nieujemna
ArrayList <INT **ID_Uczestnika**> **PreferencjeWobecKolegów**

//Opis metod na końcu
String **DrukujUczestników()**
Static Void **GenerujUżytkowników()**
Static Void **GenerujPreferencje()**
}

REALIZACJA

{
INT **ID_realizacji**
INT **Typ_kursu**
INT **Dlugosc** // (w h)
INT **Liczba_miejsc**
INT **Liczba_zapisanych**
INT **Koszt_prelegenta**

INT **Cena_kursu**
INT **Sala** // początkowo będzie to -1 (czyli oznaczenie braku przypisania)
// Nie potrzeba tu czasu, realizacje będą przydzielone w obiekcie sala
DOUBLE **WartośćFunkcjiPrzystosowania**

//Opis metod na końcu
Static Void **GenerujRealizacje()**
String **DrukujRealizacje()**
Double **PrzeliczPonownieLokalnąFunkcjePrzystosowania()**
Double **ZwróćLokalnąWartośćPrzystosowania()**
}

SALA

{
INT **ID_Sali**
INT **Pojemnosc**
INT **KosztNajmu** // za 1 h
INT **Obciazenie** // na ile h sala jest już zajęta
Tablica [ ] [ ] (INT) **Grafik**[3][10] // (-1) - sala wolna, nieujemna licz. calk. - przypis realizacji
Tablica [ ] (INT) **WolneGodziny**[3] // liczba dostępnych godzin danego dnia
Static Tablica [ ] (INT) **ZbiórMożliwejLiczbyMiejsc** [6]
Static Tablica [ ] (INT) **ZbiórMożliwychKosztównajmu** [6]

//Opis metod na końcu
Void **ObliczZajętośćSali()**
Static Void **GenerujSala()**
String **DrukujSale()**
Int **ZwróćGrafik()**

}

PLANISTA

{
Static ArrayList <Uczestnik> **Uczestnicy**
Static ArrayList <TypKursu> **TypyKursów**
Static ArrayList <Realizacja> **Realizacje**
Static ArrayList <Sala> **Sale**

// Poniższe dane potrzebne do wyznaczania zakresów list
Static INT **LiczbaUczwstników**
Static INT **LiczbaKursów**
Static INT **LiczbaRealizacji**
Static INT **LiczbaSal**
Static DOUBLE **GlobalnaWartośćFunkcjiPrzystosowania**

//Opis metod na końcu
Static Void **GenerujDaneWejściowe ()**

Static Double **ObliczGlobalnąWarośćFunkcjiPrzystowania()**
Static Void **ZapiszRealizacjeDoPliku()**
Static Void **ZapiszGrafikDoPliku()**
Static Void **ZapiszUczestnikówDoPliku()**
}

UCZESTNIK_REALIZACJA

// Potomek w pierwszym algorytmie ewolucyjnym
{
INT **IdUczestnika**
INT **IdRealizacji**
DOUBLE **LokalnaWartośćFunkcjiPrzystosowaniaF1**

//Opis metod na końcu
Static Double **ObliczLokalnąWarośćFunkcjiPrzystowania()**
}

DOBOR_PIERWSZY

// Klasa odpowiedzialna za dobór uczestnika do sali
{
UczestnikRealizacja **Obecny** //rodzic
UczestnikRealizacja **Następny** //potomek
DOUBLE **SigmaUczestnika**
DOUBLE **SigmaRealizacji**
INT **LiczbaSukcesów**
INT **LiczbaPrób**
DOUBLE **ProporcjaSukcesów** //Sukces oznacza uzyskanie lepszego potomka
INT **LiczbaSukcesowDlaPętli** // w ostatnich M wyborach potomków
INT **LiczbaWywołańPętli** // liczba ostatnio rozważanych wyborów potomków w jednym wywołaniu algorytmu
DOUBLE **ProporcjaSukcesówDlaPętli** // = Liczba_sukcesow / M
DOUBLE **C1** = **0.82** // Parametr algorytmu ewolucyjnego
DOUBLE **C2** = **1.2** // Parametr algorytmu ewolucyjnego

//Opis metod na końcu
VOID **PetlaWywolanAlgorytmu()**
UCZESTNIK_REALIZACJA **AlgorytmEwolucyjny()**
BOOLEAN **SprawdzWalidacjePotomka**(UCZESTNIK_REALIZACJA Potomek)
VOID **PrzypiszPotomka**(UCZESTNIK_REALIZACJA Potomek)
VOID **AktualizujSigmy()**
}

REALIZACJA_SALA

{
INT **IdRealizacji**
INT **IdSali**
DOUBLE **LokalnaWartoscFuncjiPrzystosowania**

//Opis metod na końcu
Static Double **ObliczLokalnąWarośćFunkcjiPrzystowania()**
}

DOBOR_DRUGI

{ //Algorytm odpowiedzialny za przypisanie realizacja do sali
RealizacjaSala **Obecny** //rodzic
RealizacjaSala **Następny** //potomek
DOUBLE **SigmaUczestnika**
DOUBLE **SigmaRealizacji**
INT **LiczbaSukcesów**
INT **LiczbaPrób**
DOUBLE **ProporcjaSukcesów** //Sukces oznacza uzyskanie lepszego potomka
INT **LiczbaSukcesowDlaPętli** // w ostatnich M wyborach potomków
INT **LiczbaWywołańPętli** // liczba ostatnio rozważanych wyborów potomków w jednym wywołaniu algorytmu
DOUBLE **ProporcjaSukcesówDlaPętli** // = Liczba_sukcesow / M
DOUBLE **C1** = **0.82** // Parametr algorytmu ewolucyjnego
DOUBLE **C2** = **1.2** // Parametr algorytmu ewolucyjnego

//Opis metod na końcu
VOID **PetlaWywolanAlgorytmu()**
UCZESTNIK_REALIZACJA **AlgorytmEwolucyjny()**
BOOLEAN **SprawdzWalidacjePotomka**(REALIZACJA _SALA Potomek)
VOID **PrzypiszPotomka**(REALIZACJA _SALA Potomek)
VOID **AktualizujSigmy()**
BOOLEAN **PodmieńPotomkaNaLepszego**(REALIZACJA _SALA Potomek)

}

VI. Zarys koncepcji implementacji (funkcje, algorytmy, obiekty komunikacyjne)

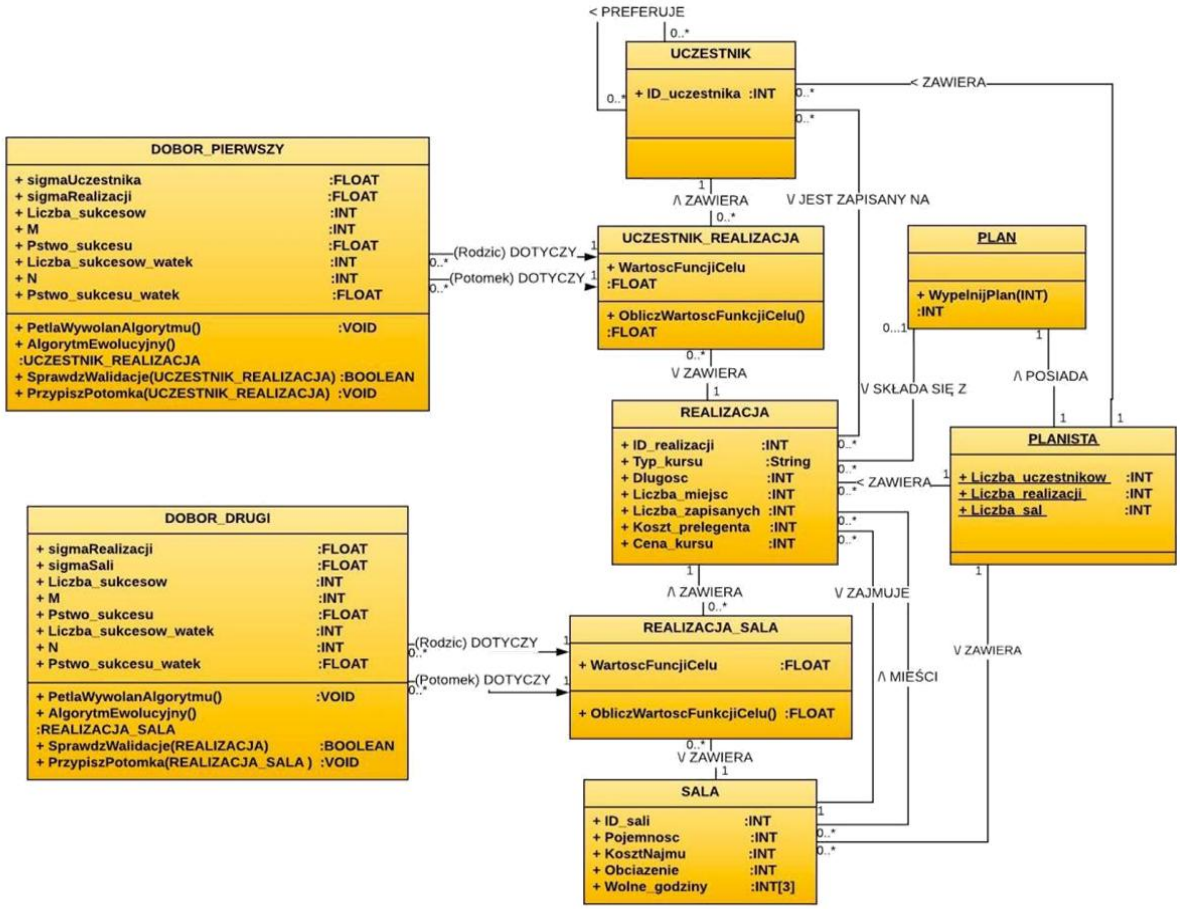
Główną klasą programu jest klasa **Planista** zawierająca listy wszystkich obiektów klas z na których będą operować algorytmy ewolucyjne.

Klasy **Dobor_pierwszy** i **Dobor_drugi** realizują funkcjonalność algorytmów ewolucyjnych i zawierają metody manipulujące danymi przechowywanymi w klasie **Planista.**

Klasy **Uczestnik_Realizacja** i **Realizacja_Sala** stanowią klasy asocjacyjne wskazanych w tytule klas. Wykorzystywane są przez klasy **Dobor_pierwszy** i **Dobor_drugi** jako obiekty reprezentujące potomka.

Klasy **Uczestnik**, **Realizacja** i **Sala** zawierają główne dane operacyjne symulacji.

Wprowadzane obiekty wrzucane są na listy z **numerami id od 0,** co upraszcza odwołania do tych obiektów na listach.



Rys. Diagram encji programu

VII. Opis metod

SALA

{

int **GenerujSale()**

//Z dostępnych zbiorów losowane są wartość Liczby Siedzeń oraz Kosztu Najmu

}

UCZESTNIK_REALIZACJA

{

FLOAT **ObliczWartoscFunkcjiCelu()**

// wartość wyliczana w oparciu o składowe odpowiednich obiektów Uczestnik i Realizacja, wskazanych przez indeksy zawarte w polach tego obiektu

}

DOBOR_PIERWSZY

{

VOID **PetlaWywolanAlgorytmu()**

// uruchamiany jest w pętli AlgorytmPierwszy() (dopóki Pstwo_sukcesu_watek > 0.01)

UCZESTNIK_REALIZACJA **AlgorytmEwolucyjny()**

// Dzialanie algorytmu (Standardowy 1+1)

- Losowanie pierwszego rodzica (dopóki któryś przejdzie walidację) i wyznaczenie jego wartość funkcji celu
- Wybieranie potomka w oparciu o odchylenie
- Sprawdzanie walidacji potomka
- Porównanie wartość funkcji celu rodzica i potomka, wybranie lepszego
- Co M wyborów potomka aktualizowana jest sigma
- Powtarzanie sekwencji do momentu aż sigma osiągnie odpowiednio małą wartość
- Algorytm znajduje najlepszego potomka (maksimum lokalne), jednocześnie przypisując uczestnika do realizacji kursu (parę odpowiadająca potomkowi)
  W razie nie znalezienia nawet jednego potomka spełniającego walidację zwracany jest potomek kontrolny z wartościami atrybutów = -1 i kończony jest aktualny etap programu

BOOLEAN **SprawdzWalidacjePotomka**(UCZESTNIK_REALIZACJA Potomek)

// Funkcja sprawdza kolejno czy dany zapis jest

- preferowany przez użytkownika
- czy jest miejsce na dany kurs
- czy nie przypisano już danego uczestnika do realizacji danego kursu
- jeśli coś nie jest spełnione to potomek jest odrzucany przed porównaniem z rodzicem

VOID **PrzypiszPotomka(**UCZESTNIK_REALIZACJA Potomek**)**

- zmiena wartości w słowniku potomka zapisy (kurs-realizacja) z „-1” na wartość indeksu realizacji

- liczba zapisanych na realizacje +=1

// **********

}

REALIZACJA_SALA

{

FLOAT **ObliczWartoscFunkcjiCelu()**

// wartość wyliczana w oparciu o składowe odpowiednich obiektów Realizacja i Sala wskazanych przez indeksy zawarte w polach tego obiektu

}

DOBOR_DRUGI

{

VOID **PetlaWywolanAlgorytmu()**

// uruchamienie w pętli AlgorytmPierwszy()
(dopóki Pstwo_sukcesu_watek > 0.1 lub licznIteracji < 100)

REALIZACJA_SALA **AlgorytmEwolucyjny()**

// Dzialanie algorytmu (Standardowy 1+1)

- Losujemy pierwszego rodzica (losujemy dopóki któryś przejdzie walidację) i wyliczamy jego wartość funkcji celu
- Sprawdzamy walidację potomka
- Porównujemy wartość funkcji celu rodzica i potomka, wybieramy lepszego
- Co M wyborów potomka aktualizujemy sigmę
- Powtarzamy sekwencje do momentu aż gdy sigma osiągnie odpowiednio małą wartość
- Algorytm znajduje najlepszego potomka (maksimum lokalne), jednocześnie przypisując realizację kursu do sali (parę odpowiadająca potomkowi)
- W razie nie znalezienia nawet jednego potomka spełniającego walidację zwracamy potomka kontrolnego z wartościami atrybutów = -1 i koniec algorytmu

BOOLEAN **SprawdzWalidacjePotomka**(REALIZACJA_SALA Potomek)

// Funkcja sprawdza kolejno czy:

- przypisanie realizacji do sali nie spowoduje strat
- dana realizacja kursu nie jest już przypisana do jakiejś sali
- sala pomieści wszystkich zapisanych na kurs
- czy sala ma jeszcze wolne terminy w jakimkolwiek dniu
- jeśli coś nie jest spełnione to potomek jest odrzucany przed porównaniem z rodzicem

VOID **PrzypiszPotomka**(REALIZACJA_SALA Potomek)

- zmieniamy grafik sali z -1 (nieprzypisana) na wartość indeksu realizacji
- liczba godzin zarezerwowania sali += liczba godzin realizacji kursu

// **********

}

PLANISTA

{

Static Void **GenerujDaneWejściowe ()**

// Generowane są losowa dane dotyczące deklaracji uczestników, typów kursów i ich realizacji oraz dostępnych sal

Static Double **ObliczGlobalnąWarośćFunkcjiPrzystowania()**

//Sumowanie lokalnych wartości funkcji celu każdej przypisanej realizacji

Static Void **ZapiszRealizacjeDoPliku()**

//zapisuje realizację do pliku w celu czytelnej prezentacji wyników działania programu

Static Void **ZapiszGrafikDoPliku()**

//zapisuje grafiku do pliku w celu czytelnej prezentacji wyników działania programu

Static Void **ZapiszUczestnikówDoPliku()**

//zapisuje uczestników do pliku w celu czytelnej prezentacji wyników działania programu

}