MMWD

Automatyka i Robotyka 2018/19

Skład zespołu: Bartłomiej Gondek, Bartosz Bywalec, Rafał Bieszczad, Rafał Siniewicz
Opiekun: dr .inż. Joanna Kwiecień

1. Cel projektu:

Projekt ma na celu stworzenie aplikacji do generowania planu zajęć. Docelowo szukamy optymalnego dopasowania planu zajęć dla określonej liczby studentów, tak, aby zmaksymalizować łączny czas, który będzie można przeznaczyć na pracę (lub inne zajęcia wymagające stosunkowo dużej ilości godzin w tygodniu).

2. Opis i założenia:

Do stworzenia jak najlepszego planu zajęć używamy algorytmów genetycznych.

Parametry problemu:

n studentów, k przedmiotów
Dla każdego przedmiotu m_i grup laboratoryjnych/audytoryjnych (i = 1..k),
maksymalne liczebności grup dla każdego przedmiotu p_i (i = 1..k),
terminy zajęć dla poszczególnych grup

Przykładowo:

n = 100 studentów, k = 7 przedmiotów

Podstawy Robotyki:	m_1 = 10 grup laboratoryjnych	p_1 = 12 osób
Technika Mikroprocesorowa:	m_2 = 8 grup laboratoryjnych	p_2 = 15 osób
Teoria Sterowania:	m_3 = 5 grup audytoryjnych	p_3 = 25 osób

Szczególne przypadki:

- przedmioty w ramach których prowadzony jest tylko wykład
- przedmioty z których prowadzone są w jednym semestrze i ćwiczenia laboratoryjne i audytoryjne

Zapis problemu:

Macierz przyporządkowania o n wierszach i $m_1 + m_2 + \dots m_k$ kolumnach o wartościach $M_{ij} = \{0,1\}$

Ograniczenia:

- Każdy student przyporządkowany dokładnie do 1 grupy z danego przedmiotu
- Liczebność każdej z grup nie może przekroczyć maksymalnej liczebności grupy dla danego przedmiotu

- Liczebność każdej z grup nie może być mniejsza niż minimalna liczebność grupy dla danego przedmiotu
- Żaden student nie może mieć jednocześnie więcej niż 1 zajęcia

Funkcja celu: Suma czasu, który może być przeznaczony na pracę w tygodniowym planie studenta dla wszystkich studentów podzielona przez liczbę studentów -> max.:

$$f = \frac{\sum_{i=1}^{n} d_i}{n}$$

n - liczba studentów
d_i - sumaryczny czas, który można przeznaczyć na pracę w tygodniowym planie i- tego studenta

3. Algorytm

a. Idea i adaptacja

Algorytm genetyczny to rodzaj heurystyki przeszukującej przestrzeń alternatywnych rozwiązań problemu w celu wyszukania rozwiązań najlepszych. W danym problemie definiowane jest środowisko, w którym istnieje populacja osobników. W skrócie działanie algorytmu można przedstawić następująco:

1. Losowanie populacji początkowej.

2. Selekcja- najlepiej przystosowane osobniki biorą udział w procesie reprodukcji.

3. Genotypy wybranych osobników poddawane są operatorom ewolucyjnym:

   i. krzyżowanie- w naszym algorytmie polega ono na zamianie planów określonej liczby (COMB_ROWS_TO_REPLACE) studentów w dwóch macierzach z populacji, tzn. plan studenta nr X z macierzy A zamieniony z planem studenta nr X z macierzy B (zamiana wierszy w dwóch macierzach)
   ii. mutacja- w naszym algorytmie reprezentowana była przez zamianę grupy danego studenta na inną w ramach tego samego przedmiotu.

4. Narodziny kolejnego pokolenia. Aby utrzymać stałą liczbę osobników w populacji te najlepsze (według funkcji oceniającej fenotyp) są powielane, a najsłabsze usuwane. Jeżeli nie znaleziono dostatecznie dobrego rozwiązania, algorytm powraca do kroku drugiego. W przeciwnym wypadku wybieramy najlepszego osobnika z populacji - jego genotyp to uzyskany wynik.

W naszym programie zastosowaliśmy nieznaczną modyfikację algorytmu genetycznego. W klasycznym algorytmie genetycznym kolejność operacji jest następująca: selekcja, krzyżowanie, mutacja. Kolejność operacji w pętli głównej naszego programu to selekcja, mutacja (nie musi zachodzić w każdej pętli), krzyżowanie.

b. Schemat algorytmu podstawowego

i. Reprezentacja rozwiązania:

Osobnik w naszym modelu jest macierzą o wymiarach $n \times m$, gdzie n jest liczbą studentów (każdemu studentowi przyporządkowany jest jeden wiersz), a m jest sumą wszystkich grup w przedmiotach. Cała macierz zbudowana jest z bloków o wymiarach $n \times l$, gdzie l jest liczbą grup w danym przedmiocie. Każdy taki blok odpowiada jednemu przedmiotowi, a każda jego kolumna odpowiada danej grupie tego przedmiotu. Przykładowo: jednym z bloków w rozwiązaniu dla instancji odpowiadającej naszej obecnej sytuacji (przydział dla III roku Automatyki i Robotyki - 98 studentów) będzie blok przyporządkowany przedmiotowi Matematyczne Metody Wspomagania Decyzji. Będzie miał on wymiary 98×8 ponieważ mamy 98 studentów i w przypadku tego przedmiotu mamy 8 grup laboratoryjnych.
Macierz rozwiązania zawiera w sobie jedynie wartości logiczne (w kodzie występują dwie równoważne postaci rozwiązania i w zależności od reprezentacji będą w nich albo zera i jedynki, albo wartości logiczne wykorzystywane w języku Python 3 $True, False$). Każda jedynka i każde zero mówią tym czy student został przydzielony do danej grupy danego przedmiotu. Przykładowo: w macierzy rozwiązania w 53 wierszu, 15 kolumnie mamy jedynkę, kolumna ta jest piątą z kolei kolumną bloku przyporządkowanego przedmiotowi Podstawy Robotyki, więc student o numerze 53 na liście jest przydzielony do grupy 3a w przedmiocie Podstawy Robotyki (opierając się na idei stosowanej na uczelni (1a, 1b, 2a, 2b, 3a, itd…) piąta z kolei grupa to 3a).

ii. Wygenerowanie populacji początkowej:
Generowanie rozwiązania początkowego oparte jest o kilka zagnieżdżonych w sobie pętli.
Pierwsza z głównych iteruje się kolejno po przedmiotach (wydziela z rozwiązania bloki przyporządkowane przedmiotom), druga przechodzi po wierszach (wydziela z bloku wektor przyporządkowania dla danego studenta) i ostatnia wydziela kolejne informacje na temat przydziału do konkretnej grupy z wektora przydziału. W programie tworzy się pomocniczy wektor prawdopodobieństwa przydziału rozmiarowo zgodny z wektorem przyporządkowania. Program sprawdza czy student może zostać przydzielony do danej grupy w przedmiocie (sprawdzane są kolizje z poprzednio przydzielonymi przedmiotami i zajęciami audytoryjnymi) i odpowiednim miejscu wektora prawdopodobieństwa wstawiana jest liczba dostępnych miejsc dla odpowiadającej temu miejscu grupy laboratoryjnej, w przypadku gdy student nie może zostać przydzielony do danej grupy (przydział do tej grupy spowodowałby wygenerowanie rozwiązania niedopuszczalnego) w wektorze wstawiane jest zero. Jeśli grupa nie spełnia dolnego ograniczenia (jest do niej przyporządkowanych za mało studentów) odpowiedni element wektora mnożony jest przez czynnik liniowy (POSSIBILITY_FACTOR). Następnie, aby wektor reprezentował dyskretny rozkład prawdopodobieństwa każdy element wektora dzielony jest przez sumę jego elementów. Student przydzielany jest do możliwych grup zgodnie z prawdopodobieństwem określonym przez wcześniej zbudowany wektor. Gdyby program natrafił na sytuację w której wektor prawdopodobieństwa jest zerowy, zerowany jest cały blok rozwiązania i program próbuje przydzielić grupy w obrębie przedmiotu od początku, jeśli ten sam problem pojawił się kilka razy z rzędu (ilość określona jest parametrem SUBJECT_FAIL_FLAG) program wyzeruje cały obecnie rozpatrywany przedmiot oraz poprzedni i wróci do generowania poprzedniego bloku. Oczywiście przy rozpatrywaniu pierwszego bloku macierzy rozwiązania skok do poprzedniego bloku jest niemożliwy, wtedy po prostu zerowany jest jedynie pierwszy blok, ale jeśli zbyt dużo razy pojawi się problem wymagający cofnięcia przy pierwszym

rozwiązaniu (liczba określona parametrem MAX_SOLUTION_FAILS) cała próba wygenerowania rozwiązania jest zarzucana, aby uniknąć wejścia w nieskończoną pętlę.

Populacja początkowa (jak i kolejne pokolenia tej populacji) jest reprezentowana przez listę Population. Elementami listy Population są dwuelementowe listy postaci:
-pierwszy element to macierz przyporządkowania studentów wygenerowana za pomocą funkcji SolveMatrix(), czyli jest to właściwy osobnik populacji.
-zerowy element to wartość funkcji celu dla macierzy przyporządkownia, znajdującej się w pierwszym elemencie, obliczona za pomocą funkcji AverageTimePerStudent()
Zatem wygenerowanie populacji początkowej polega na wywołaniu funkcji SolveMatrix() i AverageTimePerStudent() określoną przez użytkownika ilość razy i zwiększeniu listy Population o wyniki tych funkcji w formie listy dwuelementowej opisanej powyżej.

iii. Generowanie kolejnych pokoleń osobników:
1.Selekcja. - Zastosowaliśmy system rankingu. W każdej iteracji głównego programu sortujemy listę Population według wartości funkcji celu osobników i pozostawiamy w niej zadaną przez użytkownika ilość najlepszych osobników z poprzedniego pokolenia. Resztę osobników usuwamy.

2.Mutacja - Po procesie selekcji może nastąpić mutacja na jednym losowym osobniku przeprowadzona z użyciem funkcji MutateMatrix().

MutateMatrix() wybiera losowo zadaną przez użytkownika ilość studentów. Następnie dla każdego studenta losuje przedmiot, w którym zajdzie zmiana i dla wylosowanego przedmiotu przyporządkowuje studenta do innej grupy niż był przyporządkowany dotychczas.

Po wywołaniu MutateMatrix() sprawdzamy czy wynik tej funkcji (zmutowany osobnik) spełnia ograniczenia zadania. Jeśli tak, to osobnik populacji, na którym była przeprowadzona mutacja jest zastępowany zmutowanym osobnikiem. Jeśli nie, to wywołujemy MutateMatrix() do momentu aż wynik funkcji będzie spełniał ograniczenia lub do momentu aż zostanie przekroczona maksymalna liczba prób.

To jak często zachodzi mutacja regulujemy parametrem MUTATION_PROBABILITY, który może przyjmować wartości z zakresu (0,100>, przy czym wartość 100 oznacza, że mutacja na jednym osobniku zachodzi w każdym pokoleniu, a np. wartość 20 tego współczynnika oznacza, że mutacja na jednym osobniku zajdzie średnio co piąte pokolenie.

3. Krzyżowanie - Krzyżowanie dwóch osobników polega na zastąpieniu w jednym osobniku (mainMatrix) zadaną liczbę losowo wybranych wierszy wierszami o odpowiadających numerach z drugiego osobnika (secondaryMatrix). W praktyce oznacza to zamianę przyporządkowania do grup ze wszystkich przedmiotów studentów o losowo wybranych numerach z pierwszej macierzy na odpowiadające im przyporządkowanie z drugiej macierzy. Kombinacje przeprowadzamy za pomocą funkcji CombineMatrix().

W programie wywołujemy funkcję CombineMatrix(), przekazując jej jako argumenty mainMatrix i secondaryMatrix dwóch osobników spośród osobników pozostałych w populacji po etapie selekcji (i ewentualnej mutacji) w aktualnym pokoleniu. Później za pomocą funkcji ograniczenia() sprawdzamy czy wynik tej funkcji spełnia ograniczenia zadania. Jeśli tak, to potomek jest dodawany do listy Population (wraz z obliczoną wartością funkcji celu). Jeśli nie, to ponownie wywołujemy CombineMatrix() dla tych samych argumentów mainMatrix i secondaryMatrix, do momentu aż wynik funkcji będzie spełniał ograniczenia lub do momentu aż zostanie przekroczona maksymalna liczba prób. Następnie program losuje kolejnych dwóch osobników i przeprowadza krzyżowanie do momentu aż lista Population będzie liczyła podaną na początku programu liczbę elementów lub do momentu aż zostanie przekroczona maksymalna liczba prób.

iv. Pseudokod

Główny program:

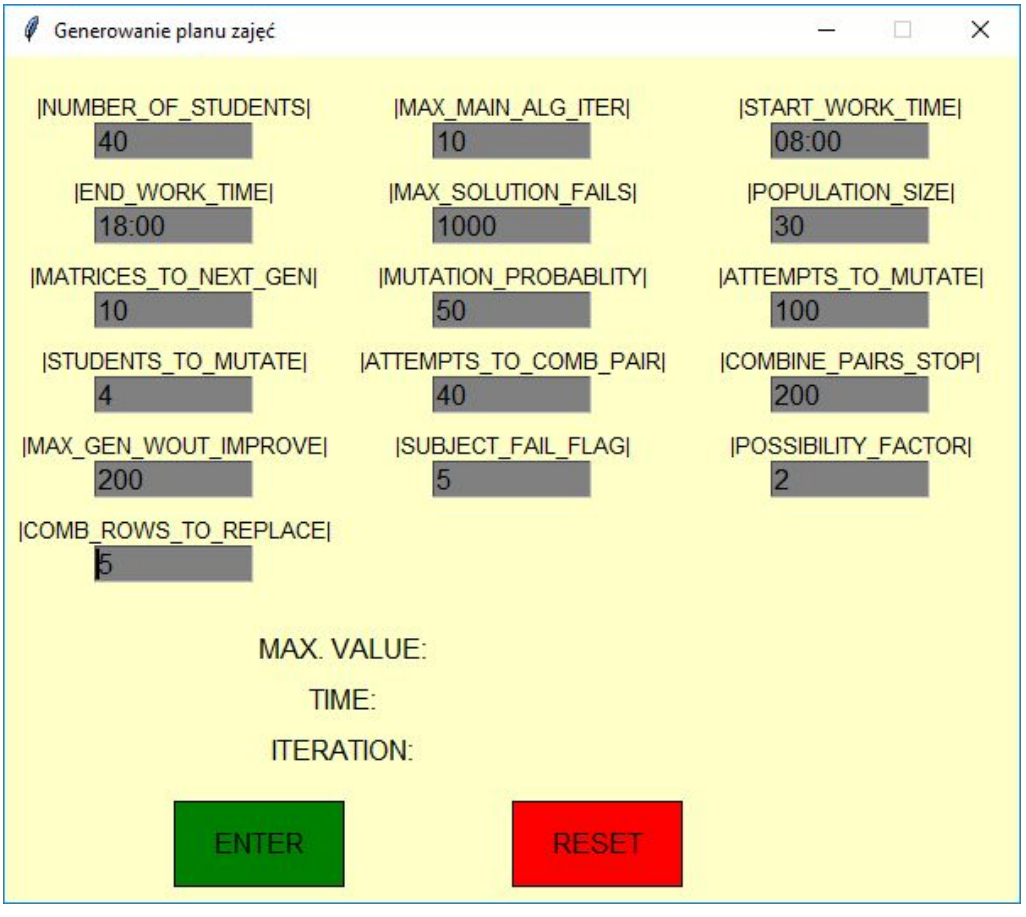
```
P := empty array                    // Population
Do POPULATION_SIZE -times:
        M := matrix  that meet the requirements
        t := average time for work in week per student for M
        Add array(t,M) to the end of P
sortDescending(P)
t_best := P[0][0]
M_best := P[0][1]
i1 := i2 := 0
while i1 < MAX_MAIN_ALG_ITER and i2 < MAX_GEN_WOUT_IMPROVE
        delete( P[j]) ,  ∀j ≥ MATRICES_TO_NEXT_GEN
        k := randInt(0, endRandIntRange)
        if k ε < 0,  MATRICES_TO_NEXT_GEN) then:
                i3 := 0
                while i3 < ATTEMPTS_TO_MUTATE
                        M_mut := MutateMatrix(P[k])
                        if ograniczenia( M_mut ) = TRUE then:
                                t_mut := average time for work in week per student for M_mut
                                P[k] := array( t_mut , M_mut )
                                i3 := ATTEMPTS_TO_MUTATE
                        i3 := i3 + 1
        i4 := 0
        while i4 < COMBINE_PAIRS_STOP
                (j1, j2) := randomTwoIntegers(0, MATRICES_TO_NEXT_GEN)
                i5 := 0
                while i5 < ATTEMPTS_TO_COMB_PAIR
                        M_comb := CombineMatrices(P[j1], P[j2])          // Krzyżowanie
                        if ograniczenia( M_comb ) = TRUE then:
                                t_comb := average time for work in week per student for M_comb
```

```
                    Add array(t,M) to the end of P
                    i5 := ATTEMPTS_TO_COMB_PAIR
        i5 := i5 +1
        if length(P) >= POPULATION_SIZE then:
                i4 := COMBINE_PAIRS_STOP
sortDescending(P)
if P[0][0] > t_best then:
        t_best := P[0][0]
        M_best := P[0][1]
        i2 := 0
else:
        i2 := i2+1
i1 :=0
```

4. Aplikacja

Główna część aplikacji (Graphical User Interface) została stworzona przy pomocy biblioteki graphics. Do rysowania wykresów użyto biblioteki matplotlib.

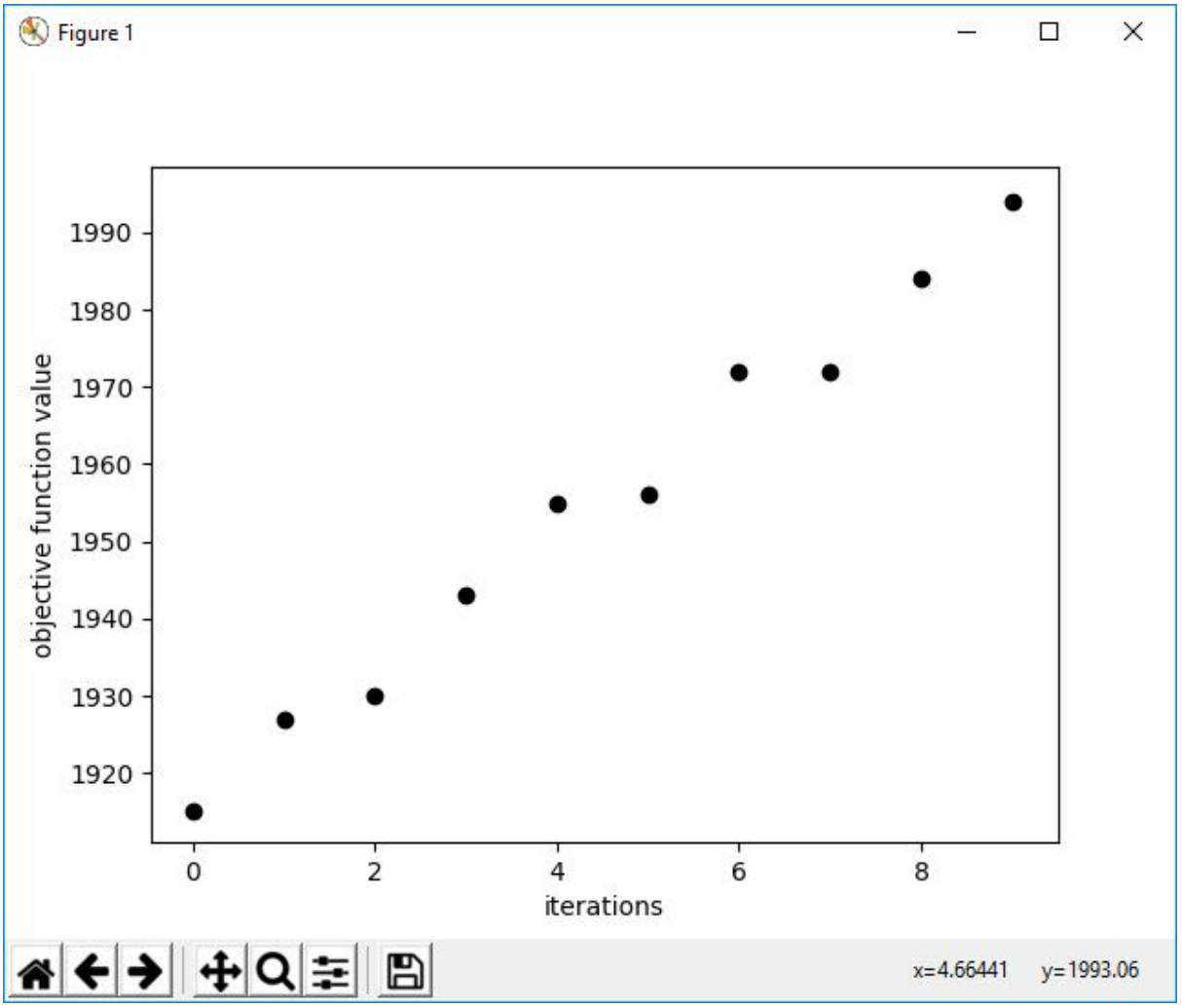
Zasadniczym zadaniem aplikacji jest możliwość sprawnej i łatwej komunikacji z programem, tj. wpisywania i modyfikowania zmiennych, generowanie rozwiązania, resetowanie ustawień, rysowanie wykresu wartości funkcji celu w kolejnych iteracjach oraz zapisywanie wyniku (macierzy reprezentującej plan zajęć oraz wartości funkcji celu) do pliku o nazwie output.txt.



Rys. 1. okno główne aplikacji

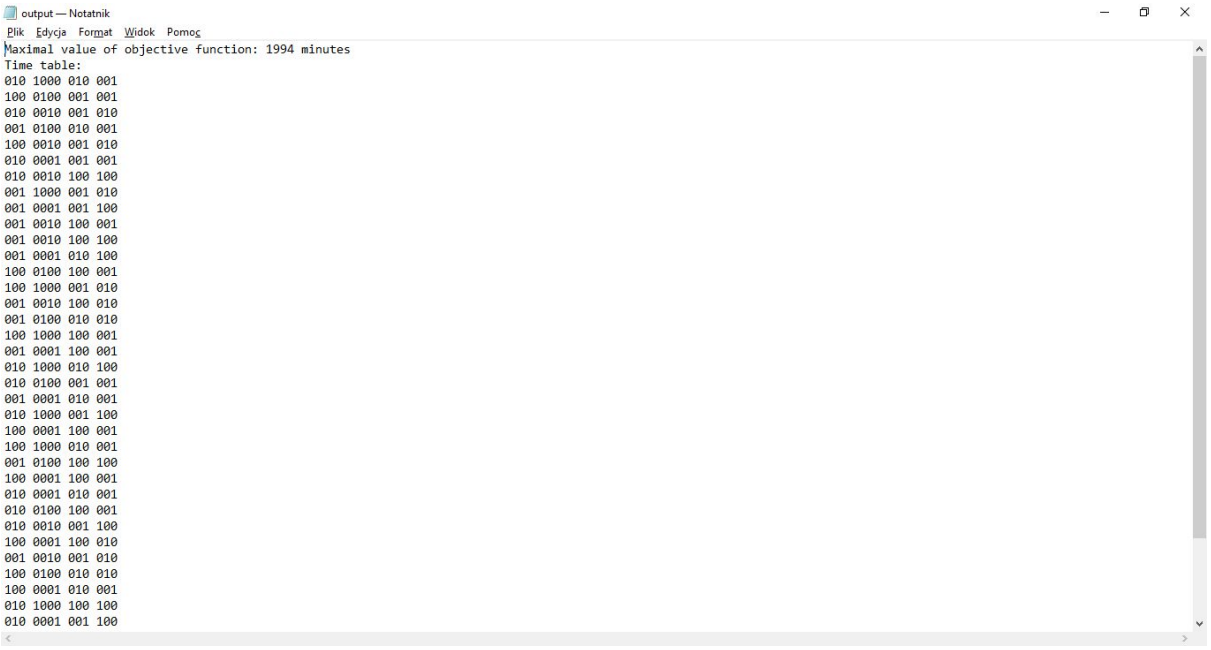
Opis okna:
W oknie GUI znajduje się 16 tzw. input boxów, tzn. okienek, do których można wpisywać różne wartości. Znaczenie każdej z nich jest omówione poniżej- aby ułatwić użytkownikowi korzystanie z aplikacji, każdy input box posiada wartość domyślną, którą można zmodyfikować.
Wciśnięcie przycisku ENTER powoduje rozpoczęcie pracy programu. Przycisk RESET odpowiada za przyporządkowanie wszystkim zmiennym wartości domyślnej oraz wyczyszczenie okna, w którym znajduje się wykres. Przy polu ITERATION wyświetlane są na bieżąco iteracje, dzięki czemu można zaobserwować przy której iteracji "jesteśmy" i przy której iteracji program się zakończył.

Legenda:
MAX. VALUE - oznacza max. wartość funkcji celu uzyskaną po przebiegu programu.
TIME - wyświetla czas, w którym program się wykonywał
NUMBER_OF_STUDENTS - liczba oznaczająca ilość studentów
MAX_MAIN_ALG_ITER - oznacza max. liczbę iteracji głównego programu
START_WORK_TIME - godzina od której najwcześniej studenci mogą rozpocząć pracę/ inną czynność wymagającą dużą ilość czasu- format: HH:MM
END_WORK_TIME - godzina do której najpóźniej studenci mogą rozpocząć pracę/ inną czynność wymagającą dużą ilość czasu- format: HH:MM
MAX_SOLUTION_FAILS - max. ilość prób wygenerowania osobnika spełniającego ograniczenia do populacji początkowej
POPULATION_SIZE - rozmiar populacji rozwiązań
MATRICES_TO_NEXT_GEN - określa ilu najlepszych osobników wg rankingu zostanie dopuszczonych do krzyżowania
MUTATION_PROBABILITY - prawdopodobieństwo wystąpienia mutacji wyrażone w procentach
ATTEMPTS_TO_MUTATE - max. liczba prób mutacji
STUDENTS_TO_MUTATE - ilość studentów wybranych do mutacji
ATTEMPTS_TO_COMB_PAIR - max. liczba prób krzyżowania wylosowanej pary osobników
COMBINE_PAIRS_STOP - max. liczba prób losowania pary osobników do krzyżowania
MAX_GEN_WOUT_IMPROVE - max. liczba kolejnych iteracji programu głównego bez polepszenia wartości funkcji celu
COMB_ROWS_TO_REPLACE - liczba wierszy wstawionych z drugiej macierzy do pierwszej podczas krzyżowania
SUBJECT_FAIL_FLAG - potrzebny podczas generacji populacji początkowej. Określa ilość prób przyporządkowania w danym przedmiocie
POSSIBILITY_FACTOR - parametr określający prawdopodobieństwo podczas generowania osobnika populacji początkowej



Rys. 2. okno przedstawiające wykres zmian funkcji celu w kolejnych iteracjach głównego programu

Opis okna:
Okno, w którym są rysowane wykresy również ma na celu jak najprostsze i intuicyjne przedstawienie zmian wartości funkcji celu w kolejnych iteracjach. Oś X oznacza ilość iteracji, a oś Y oznacza wartość funkcji celu. Wykres jest rysowany w czasie wykonywania programu, tak, aby można było na bieżąco śledzić zmiany wartości funkcji celu.



Rys. 3. plik output.txt

Opis:
Do pliku output.txt zapisywane są dwie informacje- max. wartość funkcji celu oraz wygenerowany plan zajęć w postaci macierzy. Wiersze macierzy oznaczają studentów, a zgrupowane kolumny- przedmiot. Pojedyncza kolumna oznacza grupę w danym przedmiocie, np. grupa 4b, z przedmiotu Matematyka.

5. Testy:

Testy zostały dla liczby studentów równej 40 oraz dla 7 przedmiotów(3 z grupami dziekańskimi (studenci odgórnie przyporządkowani do grup) i 4 z grupami laboratoryjnymi (studenci do przyporządkowania)). Czas trwania i godziny rozpoczęcia zajęć dla poszczególnych grup przedstawione są w tabeli poniżej:

Przedmiot	Czas trwania	Czas rozpoczęcia dla kolejnych grup			
Przedmioty z grupami do przydzielenia					
Przedmiot 1	1h 30min	Pon 12:30	Pon 14:00	Czw 14:30	
Przedmiot 2	2h 15min	Wt 12:30	Sr 12:00	Czw 14:00	Pt 13:45
Przedmiot 3	1h 30min	Pt 16:00	Pt 8:00	Czw 17:30	
Przedmiot 4	1h 30min	Pon 9:30	Czw 17:00	Wt 12:30	
Przedmioty z grupami dziekańskimi					
Przedmiot 5	1h 30min	Pon 17:30	Pon 9:15	Wt 14:00	

Przedmiot 6	1h 30min	Wt 16:00	Sr 15:00	Sr 12:00	
Przedmiot 7	1h 30min	Czw 11:00	Czw 8:00	Pon 10:00	

Tabela 1. Plan zajęć wykorzystany podczas testów

Zestaw parametrów wyjściowych:
POPULATION_SIZE = 30
MATRICES_TO_NEXT_GEN = 10
MUTATION_PROBABLITY = 20
STUDENTS_TO_MUTATE = 2
COMB_ROWS_TO_REPLACE = 5
MAX_GEN_WOUT_IMPROVE = 200
MAX_MAIN_ALG_ITER = 1000

Testy rozpoczęliśmy od sprawdzenia wpływu poszczególnych parametrów na wartość funkcji celu na koniec działania algorytmu. Dla każdego zestawu parametrów pięciokrotnie uruchomiliśmy program. Pierwszy, spradzanym parametrem była wielkość populacji , a także ilość osobników przechodzących do następnej iteracji .

Wpływ rozmiaru populacji na wynik algorytmu				
ozmiar populacji/ilość osobników echodzących do następnej iteracji	ajgorszy wynik	ajlepszy wynik	dni wynik	lepsze znalezione rozwiązanie
15/5	1304	1337	1318	367 dla populacji ącej 50 osobników
30/10	1297	1342	1319	
50/20	1339	1367	1354	

Tabela 2. Wpływ rozmiaru populacji na wynik algorytmu

Jak możemy zauważyć w tabeli 2 zmiana rozmiaru populacji z 15 na 30 osobników nie ma większego wpływu na wynik algorytmu, jednak przy zwiększeniu liczby osobników do 50 algorytm zwraca znacząco wyższy wynik (najgorsza wartość dla populacji równej 50 jest większa niż średnie wartości dla mniejszych populacji).
Kolejnym testowanym parametrem jest prawdopodobieństwo wystąpienia mutacji, a także ilość mutujących studentów podczas jednej iteracji (jeżeli mutacja odbywa się).

Wpływ ilości mutacji na wynik algorytmu				
wdopodobieństwo mutacji/ ilość osobników mutujących	ajgorszy wynik	ajlepszy wynik	dni wynik	ajlepsze znalezione rozwiązanie
20/2	1294	1329	1305	1346 dla rawdopodobieństwa utacji równego 80%
50/4	1297	1342	1319	
80/7	1308	1346	1323	

Tabela 3. Wpływ ilości mutacji na wynik algorytmu

Analizując wyniki przedstawione w tabeli 3 można zauważyć, że im większa ilość mutacji tym lepsze wyniki zwraca algorytm, wyniki najgorszy, najlepszy oraz średni zwiększają się wraz z wzrostem ilości mutacji.

Ostatnim sprawdzanym parametrem jest ilość wierszy, które zostaną zastąpione wierszami innego osobnika.

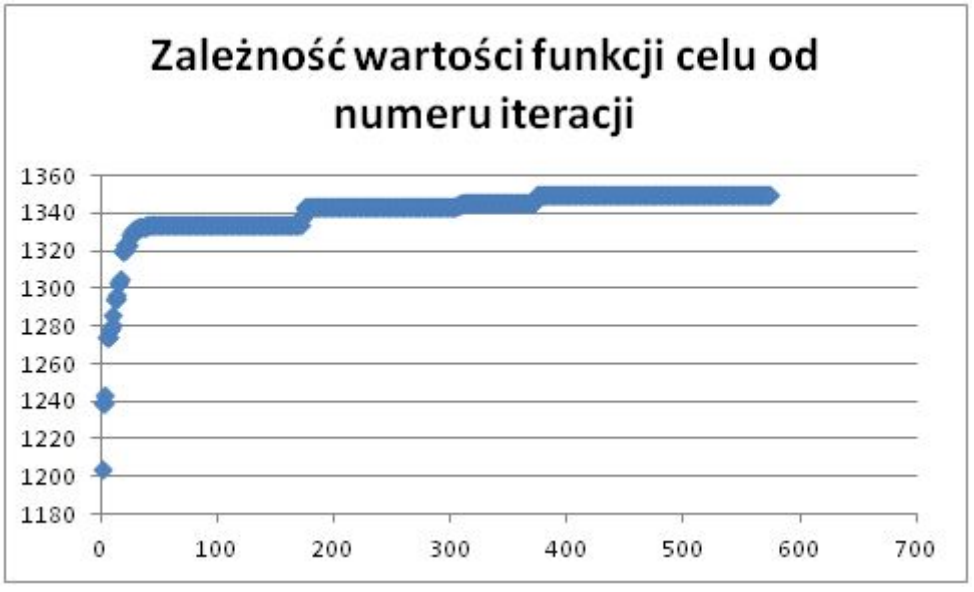
Wpływ ilości krzyżowanych wierszy na wynik algorytmu				
ość krzyżowanych jednorazowo wierszy	ajgorszy wynik	ajlepszy wynik	dni wynik	jlepsze znalezione rozwiązanie
5	1297	1342	1319	1356 dla 20 krzyżowanych orazowo osobników
10	1329	1346	1338	
20	1338	1356	1346	

Tabela 4. Wpływ ilości krzyżowań na wynik algorytmu

Podobnie jak dla mutacji można zauważyć, że wraz ze wzrostem krzyżowanych wierszy zwiększa się wartość funkcji celu zwracana przez algorytm. Jednak rozważanie większej liczby wierszy do krzyżowania niż 20 (dla 40 studentów) nie ma sensu, ponieważ krzyżowanie 21 wierszy byłoby analogiczne do krzyżowania 19, 22 do 18 itd.

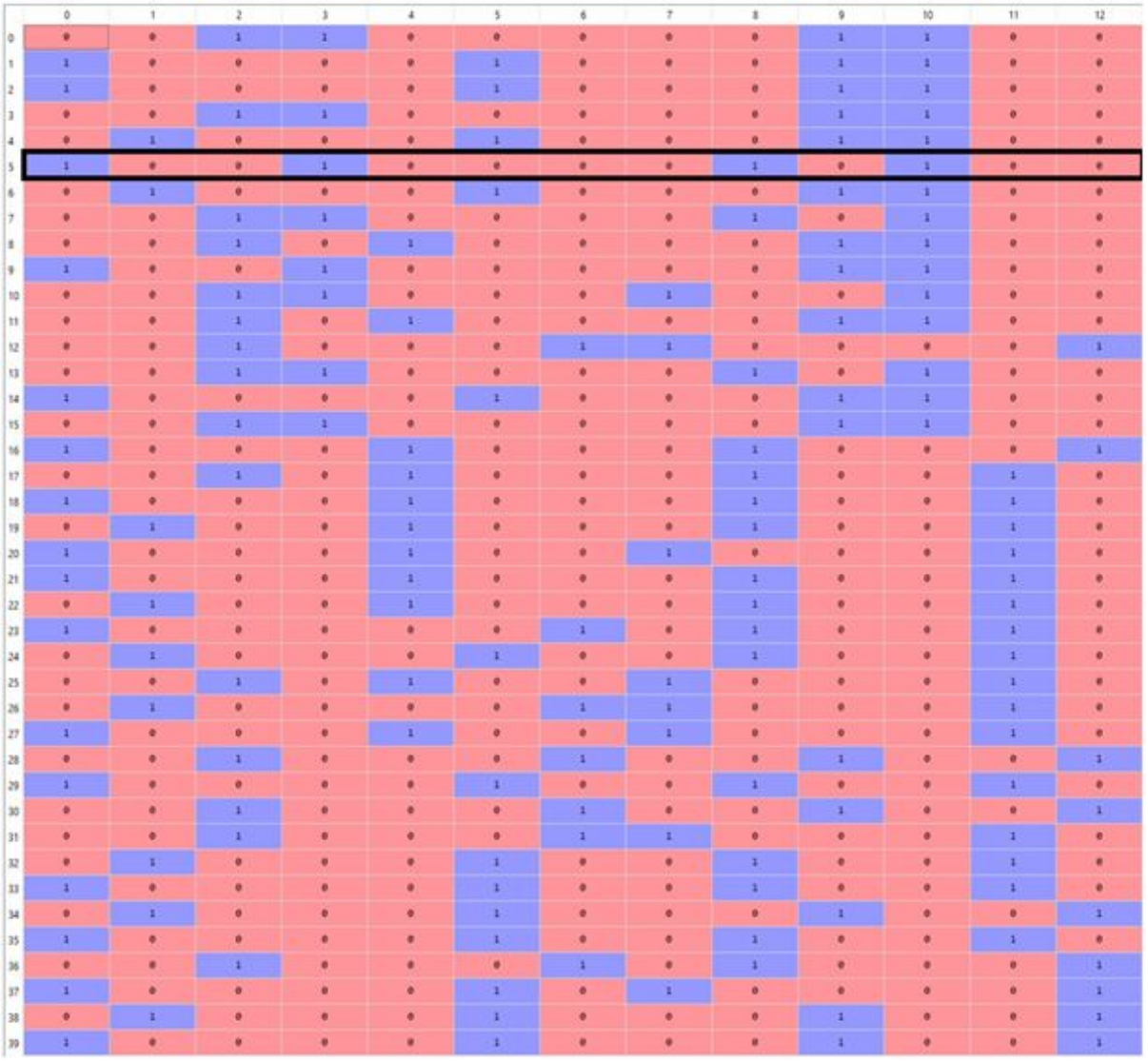
W celu zbadania zmienności chwilowej wartości funkcji celu dla najlepszego osobnika w trakcie trwania algorytmu ustawiliśmy poszczególne parametry na kolejno najlepszą i najgorszą wartość (zgodnie z powyższymi tabelami) i zapisywaliśmy aktualna najwyższą wartość funkcji celu dla każdej iteracji.

Zbiór najlepszych badanych parametrów: uzyskana wartość 1350 po 573 iteracjach



Rys. 4. Zależność wartości funkcji celu od iteracji algorytmu

Macierz będąca przyporządkowaniem studentów do grup:



Rys. 5. Macierz przyporządkowań studentów do grup

Plan zajęć dla studenta numer 5 (znajduje się on w 1 grupie dziekańskiej):

Poniedziałek	Wtorek	Środa	Czwartek	Piątek
9:30 – 11:00 przedmiot 4				8:00 – 9:30 przedmiot 3
12:30 – 14:00 przedmiot 1	12:30 – 14:45 przedmiot 2		11:00 – 12:30 przedmiot 7	

17:30 – 19:00 przedmiot 5	16:00 – 17:30 przedmiot 6			

Tabela 5. Plan zajęć dla studenta nr 5

Student ten nie ma czasu na pracę w poniedziałek, wtorek ani czwartek. W pracy może spędzić całą środę (10h) oraz 8,5h w piątek. Daje to w sumie 18,5 godzin przeznaczonych na prace, jest to 1110 minut, więc mniej niż średnia dla wszystkich studentów.

Zbiór najgorszych badanych parametrów: uzyskana wartość 1258 po 223 iteracjach

Rys. 6. Zależność wartości funkcji celu od iteracji algorytmu

W początkowych iteracjach wartość funkcji celu gwałtownie rośnie, by po osiągnięciu pewnego poziomu ustabilizować się, jedynie co jakiś czas odrobinę wzrastać.
W trakcie przeprowadzania testów zwracaliśmy uwagę na numery iteracji, w których kończyło się wykonywanie algorytmu. Podczas testów uruchomiliśmy algorytm około 40 razy tylko 7 razy wykonało się 1000 iteracji, więc tylko 7 razy zadziałało kryterium dotyczące maksymalnej liczby iteracji. W pozostałych przypadkach algorytm kończył się poprzez spełnienie kryterium dotyczące ilości iteracji bez poprawy najlepszej wartości funkcji celu. Dwa razy zdarzyło się, że algorytm wykonał tylko niewiele ponad 200 iteracji (223 oraz 238), ale w ponad pięćdziesięciu procentach przypadków wykonało się ponad 700 iteracji.

Na zakończenie testów zdecydowaliśmy się na sprawdzenie działania naszego algorytmu w „trudniejszych" warunkach. Zwiększyliśmy liczbę studentów (z 40 do 100) oraz grup w poszczególnych przedmiotach (z 3-4 do 8-9). Większa liczba grup równoważna jest z większą liczbą kolidujących ze sobą terminów, nie stanowiło to jednak problemu dla naszego algorytmu, który po 999. iteracjach zwrócił wynik 1423. Ta liczba nie może być oczywiście

porównywana do poprzednich (inne rozłożenie terminów dla poszczególnych grup), ale pokazuje że nasz algorytm jest w stanie poradzić sobie z bardziej skomplikowanymi przypadkami.

6. Wnioski

Algorytm genetyczny dzięki swojej uniwersalności i względnej prostocie okazał się dobrym narzędziem do utworzenia logicznej i spójnej “sieci” poleceń, dzięki której mogliśmy stworzyć program generujący jak najlepszy plan zajęć dla określonej grupy studentów, biorąc pod uwagę różne parametry. Najważniejszymi czynnikami, które wpływały na wartość funkcji celu były: ilość iteracji i rozmiar populacji. Duży wpływ na ostateczny wynik miały również: prawdopodobieństwo mutacji czy ilość wyselekcjonowanych osobników.