|  |  |  |  |
| --- | --- | --- | --- |
| agh_nzw_s_pl_1w_wbr_rgb_150ppi  **NAZWA WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**  Informatyka w sterowaniu i zarządzaniu  Głębokie uczenie i inteligencja obliczeniowa  Grupa 2 (śr 13:30 – 15:45)  *Układanie planu dla szkoły językowej* | | | |
| ***L.p.*** | **Członek** | **Numer albumu** | **Adres e-mail** |
| *1* | Patryk Chorąży | 402569 | pchorazy@student.agh.edu.pl |
| *2* | Rafał Kośla | 400332 | rkosla@student.agh.edu.pl |
| *3* | Artur Mzyk | 400658 | arturmzyk@student.agh.edu.pl |
| *4* | Joanna Nużka | 400561 | [joannanuzka@student.agh.edu.pl](mailto:joannanuzka@student.agh.edu.pl) |
| *5* | Adrian Poniatowski | 401346 | [adrianponiat@student.agh.edu.pl](mailto:adrianponiat@student.agh.edu.pl) |
| *6* | Wojciech Poniewierka | 402224 | wponiewierka@student.agh.edu.pl |

Spis treści

[1. Wstęp 3](#_Toc129799542)

[1.1. Cel projektu 3](#_Toc129799543)

[1.2. Założenia projektu 3](#_Toc129799544)

[2. Badany problem 3](#_Toc129799545)

[2.1. Opis problemu 3](#_Toc129799546)

[2.2. Przyjęte oznaczenia 4](#_Toc129799547)

[2.3. Postać rozwiązania 5](#_Toc129799548)

[2.4. Ograniczenia 5](#_Toc129799549)

[2.5. Funkcja celu 6](#_Toc129799550)

[2.6. Dane wejściowe 6](#_Toc129799551)

[2.7. Przegląd literatury 6](#_Toc129799552)

[3. Propozycja rozwiązania 6](#_Toc129799553)

[3.1. Algorytm 6](#_Toc129799554)

[3.1.1. Schemat podstawowego algorytmu 6](#_Toc129799555)

[3.1.2. Adaptacja algorytmu 6](#_Toc129799556)

[3.2. Pseudokod 6](#_Toc129799557)

[4. Aplikacja 6](#_Toc129799558)

[5. Eksperymenty 7](#_Toc129799559)

[6. Podsumowanie/wnioski 7](#_Toc129799560)

[7. Spis literatury 7](#_Toc129799561)

[8. Podział pracy 7](#_Toc129799562)

# Wstęp

## Cel projektu

Celem projektu jest zbudowanie aplikacji, która będzie za pomocą algorytmu genetycznego optymalizowała problem postawiony w tytule pracy, czyli „Układanie planu dla szkoły językowej”. Aplikacja zostanie rozbudowana o interfejs graficzny, który pozwoli na dynamiczną zmianę danych wejściowych oraz parametrów algorytmu. Dobór odpowiednich parametrów jest bardzo ważny z punktu widzenia optymalizacji, gdyż algorytm genetyczny daje jedynie rozwiązania bliskie optymalnemu. Odpowiednie dostrojenie – zazwyczaj  
w sposób empiryczny – może pozwolić na osiągnięcie znacznie lepszych rozwiązań niż  
w przypadku ustawienia losowej konfiguracji parametrów.

## Założenia projektu

Dane wejściowe wprowadzane są do aplikacji w postaci plików CSV, zaś parametry ustawiane w interfejsie użytkownika. Uczniowie dzieleni są na grupy w miarę możliwości na tym samym poziomie językowym – poziom ucznia może się różnić od poziomu grupy maksymalnie o jeden. Zakładamy również, że istnieje rozwiązanie problemu dla wprowadzonych danych – szkoła posiada wystarczająco dużo sal i nauczycieli, aby każda grupa uczniów mogła mieć zajęcia w godzinach pracy szkoły.

# Badany problem

## Opis problemu

Rozważanym problemem jest ułożenie harmonogramu zajęć dla szkoły językowej. Jest to zagadnienie często podejmowane w innych pracach, zwłaszcza w kontekście uczelni czy szkół. Celem jest ustawienie lekcji w taki sposób, aby spełnione były wszystkie ograniczenia czasowe i te dotyczące zasobów zarówno ludzkich, jak i przestrzennych, oraz aby wartość przyjętej funkcji celu była jak najniższa. Ustalenie, że dana lekcja odbywa się w danym terminie, polega na powiązaniu ze sobą następujących danych:

* przedziałów czasowych formujących zajęcia,
* lektora,
* sali,
* uczniów o odpowiednim poziomie.

Szkoła językowa ma określoną liczbę uczniów, lektorów i sal. Są one niezależne od siebie. Liczba uczniów jest odczytywana z danych wejściowych – z uczniem powiązany jest jego poziom językowy. Liczba lektorów jest odczytywana z danych wejściowych – z lektorem są powiązane jego preferowane godziny pracy. Natomiast liczba sal jest pobierana z interfejsu graficznego.

Liczebność grup jest taka sama dla każdej z nich i pobierana jest z interfejsu graficznego. Liczba grup zależy od liczby przedziałów czasowych, w których jest otwarta szkoła, oraz od dostępności lektorów i sal.

Każdy uczeń i każda grupa mają określony poziom języka: 1, 2 lub 3. Do danej grupy mogą zostać zapisani uczniowie, których poziom różni się co najwyżej o 1 od poziomu tejże grupy. Za przypisanie ucznia do nieodpowiedniego poziomu jest przewidziana kara w funkcji celu. Kara w przypadku prawidłowego przypisania wynosi 0.

Dany uczeń może mieć tylko jedne zajęcia w ciągu tygodnia.

Dany lektor może uczyć dowolną grupę, gdyż jako wykształcony lingwistycznie człowiek potrafi doskonale język i poziom nie jest dla niego problemem.

Czas, w którym szkoła jest otwarta, jest podzielony na przedziały 15-minutowe. Pozwala to na elastyczne rozłożenie godzin zajęć i nie wymusza rozpoczynania ich o pełnej godzinie zegarowej. Przykładowo, jeśli zajęcia trwają godzinę, to zajmują one 4 takie przedziały. Długość – liczba przedziałów czasowych, które zajmują dane zajęcia – jest pobierana z interfejsu graficznego.

Dany lektor ma swoje preferowane godziny pracy. Funkcja celu uwzględnia karę za pracę lektora poza jego preferowanymi godzinami. Lektor nie potrzebuje przerwy pomiędzy zajęciami, gdyż czas funkcjonowania szkoły nie jest długi. Brak przerw nie powinien mocno wpływać na wydajność lektora.

Celem jest minimalizacja czasu spędzonego przez lektora w szkole językowej.

## Przyjęte oznaczenia

Zbudowano model problemu i zastosowano w nim następujące oznaczenia:

– rozwiązanie,

– liczba przedziałów czasowych (ang. *time slots*) na przestrzeni tygodnia, ,

– indeks przedziału czasowego, ,

– długość trwania zajęć (ang. *duration*), ,

– liczba lektorów (ang. *lectors*), ,

– liczba sal (ang. *rooms*), ,

– liczba zajęć (ang. *classes*) w przedziale czasowym , ,

– indeks zajęć w przedziale czasowym , ,

– indeks lektora powiązanego z przedziałem czasowym i zajęciami , ,

– indeks sali powiązanej z przedziałem czasowym i zajęciami , ,

– indeks lektora, ,

– zbiór bloków (ang. *blocks*), w których zajęcia prowadzi lektor ,

– zbiór tzw. okienek (ang. *windows*) lektora ,

– zbiór preferowanych (ang. *prefered*) przedziałów czasowych lektora ,

– zbiór bloków (ang. *blocks*), które są prowadzone w sali ,

– zbiór tzw. okienek (ang. *windows*) sali ,

– liczba grup (ang. *groups*), ,

– maksymalny rozmiar grupy (ang. *maximal group size*), ,

– liczba uczniów, ,

– liczba uczniów o znajomości (ang. *knowledge*) języka na poziomie 1, ,

– liczba uczniów o znajomości (ang. *knowledge*) języka na poziomie 2, ,

– liczba uczniów o znajomości (ang. *knowledge*) języka na poziomie 3, ,

– współczynnik kary za liczbę przedziałów czasowych przepracowanych przez lektora poza jego preferowanymi godzinami pracy, ,

– współczynnik kary za liczbę przedziałów czasowych będących tzw. okienkiem dla lektora, ,

– współczynnik kary za nieprawidłowe przypisanie ucznia do grupy językowej, .

## Postać rozwiązania

Z matematycznego punktu widzenia postać rozwiązania to wektor o stałej długości . Jego elementami są wektory o zmiennej długości . Z kolei ich elementami są piątki uporządkowane w postaci .

Z programistycznego punktu widzenia postać rozwiązania to -elementowa lista  
-elementowych list krotek 5-elementowych.

## Ograniczenia

Liczba grup jest ograniczona przez liczbę uczniów i maksymalny rozmiar grupy :

.

Do danej grupy mogą zostać przypisani tylko uczniowie, których poziom różni się co najwyżej o 1 od poziomu tejże grupy:

Dana grupa musi mieć zajęcia w sąsiednich przedziałach czasowych, które formują razem blok zajęciowy, co jest intuicyjne:

gdzie

Dany uczeń może mieć zajęcia tylko raz w ciągu tygodnia:

Lektor może prowadzić tylko jedne zajęcia w tym samym czasie:

W jednej sali mogą się odbywać tylko jedne zajęcia w tym samym czasie:

## Funkcja celu

Funkcja celu uwzględnia trzy elementy: liczba slotów nauczyciela poza jego preferowanymi godzinami pracy, liczba okienek podczas pracy szkoły oraz liczba uczniów przypisanych do grupy na złym poziomie językowym. Jest ona opisana następującym równaniem:

## Dane wejściowe

Danymi wejściowymi są:

* lista uczniów wraz z przypisanym poziomem,
* lista nauczycieli wraz z preferowanymi godzinami pracy,
* lista klas,
* godziny pracy szkoły,
* parametry funkcji celu: wagi poszczególnych elementów,
* parametry algorytmu genetycznego: wybrany sposób selekcji, krzyżowania i mutacji, prawdopodobieństwa krzyżowania i mutacji, rozmiar populacji oraz maksymalna liczba iteracji.

## Przegląd literatury

Przegląd literatury.

# Propozycja rozwiązania

Dokładny opis proponowanego rozwiązania (proponowanych rozwiązań).

## 3.1. Algorytm

### 3.1.1. Schemat podstawowego algorytmu

Wykorzystany w projekcie algorytm to algorytm genetyczny. Pozwala on na dążenie do lokalnego ekstremum, ale dalej możliwa jest większa eksploracja przestrzeni możliwych wyników w celu poszukiwania lepszych rozwiązań. Schemat jego działania można opisać w kilku krokach:

1. Utworzenie populacji początkowej – zbiór losowych rozwiązań problemu
2. Obliczenie dopasowania każdego osobnika w populacji, np. poprzez obliczenie wartości funkcji celu
3. Selekcja osobników do tworzenia kolejnej generacji – im wyższa wartość dopasowania, tym większa szansa na wybranie danego rozwiązania
4. Krzyżowanie – wybrane osobniki tworzą nowe poprzez wymianę części swoich genów
5. Mutacja – losowa zmiana niektórych genów, by dać szansę na rozwinięcie się nowych, korzystnych cech w populacji
6. Jeżeli nie osiągnięto zadowalających rezultatów i nie przekroczono maksymalnej liczby iteracji – powrót do kroku 2.

### 3.1.2. Adaptacja algorytmu

Sposób generowania rozwiązań, sposób zakodowania rozwiązań (np. w algorytmie genetycznym sposób zrealizowania selekcji, krzyżowań i mutacji).

**Rozwiązania początkowe**

Pierwszym krokiem w generacji rozwiązań początkowych jest podział studentów na grupy   
o tym samym poziomie. Tworzone grupy mają w miarę możliwości maksymalną dopuszczalną liczbę studentów. Jeśli nie jest to możliwe, to na każdym poziomie tworzona jest maksymalnie jedna niepełna grupa.

Następnie wywoływana jest funkcja łącząca dwie grupy studentów różniące się poziomem   
o jeden. Mogą one zostać połączone tylko jeśli suma studentów, którzy do nich należą nie przekracza maksymalnej dopuszczalnej. Grupy są łączone z pewnym prawdopodobieństwem, które może być zmieniane przez użytkownika.

Dla każdej z grup losowany jest jeden z dopuszczalnych slotów początkowych, lektor i sala. Rezerwowana również jest odpowiednia liczba kolejnych slotów, aby utworzyć pełne zajęcia. Jako rozwiązanie zwracany jest słownik, w którym każdemu slotowi przyporządkowana jest lista grup (wraz z informacją o nauczycielu i sali) mających wtedy zajęcia.

**Selekcja**

Selekcje pozwalają na wybór odpowiednich osobników przy tworzeniu kolejnej populacji. Im lepsza wartość dopasowania danego rozwiązania, tym większą ma on szansę na wybranie. Zastosowano 3 podejścia przy problemie selekcji.

Selekcja best

W tym przypadku wybranymi do rozmnażania osobnikami są zawsze dwa najlepsze rozwiązania. Sposób ten jest eksperymentalny, ponieważ mocno ogranicza możliwość eksploracji przestrzeni rozwiązań.

Selekcja ruletkowa

Selekcja ruletkowa jest jedną z częściej używanych metod selekcji. W tym algorytmie im większa wartość dopasowania rozwiązania, tym większa szansa na wybranie go do reprodukcji. Co ważne w naszym przypadku algorytm ma minimalizować wartości funkcji celu, więc należy wprowadzić zmianę: zamiast liczyć wartość funkcji dopasowania, liczymy jej odwrotność. W ten sposób im mniejsza wartość dopasowania, tym większa szansa na wylosowanie. Algorytm selekcji ruletkowej można przedstawić za pomocą pseudokodu:

def rulette\_selection(population):

inverted\_fitnesses = [1/fitness(solution) for solution in population]

random\_point = random\_uniform(min=0, max=sum(inverted\_fitnesses))

chosen\_solution = 0

partial\_sum = inverted\_fitnesses[0]

while partial\_sum < random\_point:

chosen\_soluion = chosen\_solution + 1

partial\_sum += inverted\_fitnesses[chosen\_solution]

return chosen\_solution

W tym przypadku funkcja zwróci nam jedno wybrane rozwiązanie z podanej w argumencie listy. Aby dostać dwa rozwiązania musimy wywołać taką funkcję dwa razy, gdzie za drugim razem podajemy listę bez wylosowanego wcześniej osobnika, by uniknąć wylosowania dwa razy tego samego rozwiązania.

Selekcja turniejowa

Selekcja turniejowa jest popularną metodą, ze względu na swoja prostotę i skuteczność. Algorytm polega na wybraniu *n* osobników z populacji, a następnie spośród nich wybraniu osobnika z najlepszą wartością dopasowania. Z racji minimalizacji funkcji celu, dla nas najlepszą wartością jest wartość minimalna. Osobniki wybierane do „turnieju” są losowane z takim samym prawdopodobieństwem. Warto dodać, że metoda ta przyjmuje jeden parametr: *n*, a w naszym problemie, po kilku testach przyjęliśmy, że wartość ta będzie wynosić 3.

**Krzyżowanie**

**Mutacja**

Zaimplementowano dwie metody mutacji: przez przesunięcie grupy o jeden slot lub przez zmianę nauczyciela dla jednej grupy.

W mutacji przez przesunięcie grupy o jeden slot najpierw losowana jest jedna z grup. Następnie sprawdzana jest możliwość jej przesunięcia – czy przypisani do niej nauczyciel   
i sala są wolni w poprzednim lub następnym slocie oraz czy slot ten mieści się w godzinach pracy szkoły. Jeśli przesunięcie nie jest możliwe, losowana jest kolejna grupa. Jeśli istnieje możliwość jego dokonania, losowany jest kierunek ze zbioru możliwych (w górę lub w dół). W zależności od wylosowanego kierunku do słownika będącego rozwiązaniem grupa jest dodawana do slotu, który poprzedza jej pierwszy lub znajduje się za ostatnim oraz usuwana   
z ostatniego lub pierwszego slotu.

W mutacji poprzez zmianę nauczyciela losowana jest jedna z grup należących do rozwiązania. Następnie sprawdzane jest, czy w godzinach jej zajęć dostępni są inni nauczyciele. Jeśli tak, losowany jest jeden z nich i następuje zmiana lektora dla tej grupy. Jeśli nie, losowana jest inna grupa.

## 3.2. Pseudokod

Poniżej znajduje się pseudokod algorytmu, który zwraca listę najlepszych rozwiązań dla każdej iteracji:

def genetic\_algorithm():

initial\_population = make\_initial\_population()

best\_solutions = []

current\_population = initial\_population

for i from 0 to num\_of\_generations:

best\_solutions.append(get\_best\_solution(current\_population))

new\_population = []

for j from 0 to size(population) / 2:

selected = selection(current\_population)

crossovered = crossover(selected)

mutated = mutate(crosovered)

new\_population.append(mutated)

current\_population = get\_best\_n\_solutions(current\_population, new\_population)

W powyższym pseudokodzie metoda *get\_best\_solution* wybiera najlepsze rozwiązanie z danej populacji na podstawie wartości funkcji celu. Podobnie działa funkcja *get\_best\_solutions*, lecz wybiera ona *n* najlepszych rozwiązań, gdzie *n* jest rozmiarem populacji.

# 4. Aplikacja

Sama aplikacja składa się z graficznego interfejsu użytkownika, dzięki któremu można wprowadzić dane startowe algorytmu oraz części odpowiedzialne za sam algorytm. Do stworzenia GUI została użyta biblioteka PyQt5, natomiast część algorytmiczna została napisana w czystym Pythonie. Aby uruchomić aplikację wystarczy uruchomić skrypt pythonowy o nazwie main.py, następnie dobrać odpowiednie parametry i kliknąć przycisk Start.

# Eksperymenty

Badanie jakości algorytmu, różnych rozwiązań. Pokazać na przykładzie wyniki algorytmu.

# Podsumowanie/wnioski

Podsumowanie/wnioski.

# Spis literatury

# Podział pracy

Wkład każdego członka zespołu: etap realizacji i procentowy udział.