Politechnika Wrocławska
Wydział Informatyki i Telekomunikacji

Kierunek: **Inżynieria Systemów (INS)**

PRACA DYPLOMOWA INŻYNIERSKA

Zastosowanie algorytmu ewolucyjnego w aplikacji webowej wspomagającej układanie planu lekcji

Igor Kowalczyk

Opiekun pracy
Dr inż. Donat Orski

Słowa kluczowe: 3-6 słów kluczowych

WROCŁAW (2025)

Streszczenie

Bardzo fajny algorytm w bardzo fajnej aplikacji

Abstract

Very nice algorytm w bardzo nice aplikacji

Contents

1. Wstęp

Jednym z corocznych wyzwań placówek oświatowych jest ułożenie dobrego planu lekcji. Jest to nieodłączny proces towarzyszący prowadzeniu szkoły, który do tej pory sprawia problemy nawet najlepszym ośrodkom. Można go podzielić na 2 etapy:

1. **Przydział godzinowy nauczycieli do każdej klasy.** Przydział nauczycieli do klas jest zadaniem, które wykonuje się przed rekrutacją. W praktyce oznacza to rozpoczęcie pracy nad planem bez informacji o dokładnej ilości uczniów. Dostępne są jedynie prognozy które pozwalają na określenie ilości klas, co zapewnia to ciągłość nauczania jednego nauczyciela z roku na rok dla danej klasy.

2. **Przydział lekcji do slotów czasowych i sal.** Mając już informację o ilości godzin lekcyjnych, które każda klasa musi odbyć z każdym nauczycielem, możemy przejść do tworzenia właściwego planu. Głównym ograniczeniem jest stworzona w etapie 1. rozpiska. Istnieje jednak wiele innych ograniczeń, które sprawiają, że proces ręcznego układania takiego harmonogramu zajmuje niejednokrotnie dziesiątki godzin. W efekcie pierwszy miesiąc w szkołach bywa bardzo chaotyczny ze względu na ciągłe zmiany godzin i sal.

Drugi etap jest niczym innym jak problemem harmonogramowania z twardymi i miękkimi ograniczeniami. Na przestrzeni lat wykształcono wiele podejść do rozwiązywania takich problemów. Jednym z nich jest hybrodowe podejście, opierające się na jednoczesnym zastosowaniu metod sztucznej inteligencji oraz metod programowania całkowitoliczbowego. Metodą którą będę realizował w tej pracy jest algorytm ewolucyjny, inspirowany biologiczną ewolucją. Używa takich samych metod do tworzenia najlepszych osobników jak natura: mutacja, krzyżowanie i selekcja. Zastosowanie takiego podejścia pozwala na stworzenie problemu liniowego, który można rozwiązać w satysfakcjonującym czasie. Umożliwia to jednoczesne zachowanie wysokiej jakości rozwiązania, poprzez użycie solvera liniowego, oraz zmniejszenie czasu obliczeń w porównaniu do rozwiązań realizowanych tylko i wyłącznie przy użyciu programowania całkowitoliczbowego.

1.1. Cel i zakres pracy

Celem pracy jest opracowanie i stworzenie aplikacji webowej wspomagającej układanie planu lekcji. Konieczne w tym celu jest:

- Zaprojektowanie i implementacja bazy danych do przechowywania ograniczeń, specyfikacji bloków lekcyjnych oraz końcowych wyników.

- Zaprojektowanie i wykonanie interfejsu graficznego, pozwalającego użytkownikowi na wprowadzenie wszystkich potrzebnych ograniczeń oraz zobaczenie końcowego planu lekcji.

- Zintegrowanie aplikacji webowej z algorytmem do układania planu lekcji, który będze używał algorytmu genetyczny, algorytmu zachłannego oraz solvera liniowego.

Aplikacja musi umożliwiać użytkownikowi na automatyczne generowanie planu na podstawie wprowadzonych danych. Pod uwagę będą brane ogarniczenia fizyczne takie jak dostępność sal i kolizje zajęć, ograniczenia jakościowe takie jak ciągłośc zajęć, ale także ograniczenia definiowane przez użytkownika. Realizując te ograniczenia algorytm powinien minimalizować ilość okienek nauczycieli, tak aby zmniejszyć czas spędzony przez nauczycieli w szkołach.

Coś o podejściu systemowym? Aplikacja będzie przetestowana na rzeczywistych danych, co pozowli na sprawdzenie rozwiązania i zweryfikowanie, czy algorytm nadaje się do układania tego typu planów.

1.1.1. Dane wejściowe

Napisałem tą sekcję jeszcze przed otrzymaniej maila od pana na temat zmiany struktury pracy, także nie wiem jeszcze gdzie ją dam

Aby móc umożliwić użytkownikowi stworzenie ograniczenia głównego (3.1.1) potrzebujemy określonych danych wejściowych:

- Lista przedmiotów.
- Lista nauczycieli oraz przedmioty, które mogą prowadzić.
- Lista klas.

Mając te informacje użytkownik może stworzyć tabelę, która będzie służyć do wprowadzenia tygodniowych godzin lekcyjnych. Nie jest to wystarczająco, aby ułożyć plan zajęć. W tym celu potrzebujemy więcej informacji:

- Informacje o dostępności nauczyciela w dane dni.
- Informacje o blokach lekcyjnych:
  - Jakie klasy mogą występować w jednych bloku.
  - Jakie przedmioty mogą występować w jednym bloku.
- Listę sal oraz przedmiotów, które mogą być w nich prowadzone.

1.2. Układ pracy

Zarysuj strukturę swojej pracy dyplomowej. Ogólnie przedstawienie pracy. Przykładowo: „Praca dzieli się na 7 rozdziałów (...)". Rozdział **??** dotyczy (...). Temat został rozwinięty w **??**.

2. Powiązane prace

2.1. A mixed-integer programming approach for solving university course timetabling problems.

Rappos, E., Thiémard, E., Robert, S. and Hêche, J.F., 2022. A mixed-integer programming approach for solving university course timetabling problems. Journal of Scheduling, 25(4), pp.391-404.

Fajna praca [4], którą luźno się interesowałem — też rozbijała problem na 2 etapy. Też stopniowo rozwiązywała problem i stopniowo dodawała ograniczenia do solvera liniowego.

2.2. Istniejące rozwiązania

2.2.1. Vulcan

2.2.2. Dobry Plan??

Jak będzie trzeba więcej.

2.2.3. Flow-shop scheduling

Inspirowałem się rozwiązaniem tego problemu, więc warto zawrzeć.

3. Problem układania planu lekcji i algorytm jego rozwiązania

Głównym elementem systemu jest wieloetapowy algorytm generowania planu lekcji. Jego główna innowacja leży w dekompozycji oryginalnego zadania na trzy sekwencyjne fazy. Zadaniem pierwszych dwóch jest zmniejszenie przestrzeni decyzyjnej do coraz mniejszej skali, tak aby w ostatniej fazie można było sformułować i rozwiązać problem programowania całkowitoliczbowego (MIP). Rezultatem takiego podejścia jest redukcja liczby zmiennych decyzyjnych, potrzebnych do zdefiniowania ograniczeń w trzecim etapie.

Chociaż czyste podejście MIP prowadzi do teoretycznie optymalnych rozwiązań, w praktyce jego zastosowanie do pełnego problemu jest niemożliwe. Złożoność obliczeniowa i pamięciowa przekracza możliwości przeciętnych komputerów, co uniemożliwia efektywny rozwój takiego rozwiązania. Ponadto takie rozwiązanie jest też trudne do zaprogramowania ze względu na bloki lekcyjne.

Zaprezentowane podejście hybrydowe pozwala pokonać problem złożoności obliczeniowej, oferując praktyczny kompromis między optymalnością a czasem obliczeń.

3.1. Sformułowanie problemu optymalizacyjnego

Na potrzeby pracy warto ujednolicić terminologię, z uwagi na to, że w języku potocznym niektóre z tych terminów są używane zamiennie:

- **Klasa**: Grupa uczniów; przykładowo “IIA”, “IVC”, . . . Ze względu na angielską nazwę *class*, która koliduje ze składnią języków programowania, w kodzie często odnoszę się do klas jako `student_group`.
- **Sala**: Miejsce, w którym prowadzone są zajęcia; przykładowo “Sala Gimnastyczna 1”, “2”, . . .
- **Przedmiot**: Temat zajęć prowadzonych przez nauczyciela; przykładowo “Wychowanie Fizyczne”, “Matematyka”, . . .
- **Lekcja**: Zajęcia prowadzone przez jednego nauczyciela, w jednej sali, z jedną lub więcej klas, które są na temat jednego przedmiotu.
- **Blok lekcyjny**: Grupa dwóch lub więcej lekcji, które są w tym samym okienku czasowym. Mogą one dotyczyć jednej klasy oraz wielu nauczycieli, jednego nauczyciela i wielu klas, lub też wielu klas i wielu nauczycieli.
- **Okienko**: Przerwa między dwoma lekcjami klasy lub nauczyciela. Występuje gdy zajęcia nie są przeprowadzane bezpośrednio po sobie.

Problem optymalizacyjny w tej pracy polega na przypisaniu lekcji do odpowiednich slotów czasowych i sal przy jednoczesnym spełnieniu wymagań. W rzeczywistości sformułowanie takiego zadania i wyznaczenie jego rozwiązania stanowi duże wyzwanie. Istnieją ograniczenia, które są różne dla każdej klasy, co utrudnia formułowanie problemu — wiele lekcji jest realizowanych w blokach, które są definiowane każdy z osobna. Przez te wyjątki nie jest możliwym wykorzystanie prostych algorytmów. Nie jest także możliwym rozwiązanie jednego wielkiego problemu programowania całkowitoliczbowego w sensownym czasie przy użyciu komputera z przeciętną specyfikacją.

Oczekiwanym rezultatem działania algorytmu powinna być lista przypisań. Każde takie przypisanie powinno mieć 5 wartości:

1. Slot czasowy
2. Klasa
3. Sala
4. Nauczyciel
5. Przedmiot

W przypadku lekcji, która obejmują więcej klas niż jedna, należy stworzyć przypisanie dla każdej klasy osobno.

3.1.1. Ograniczenia

Wcześniej wspomniane ograniczenia można podzielić na 4 kategorie:

Ograniczenia fizyczne

- Żaden nauczyciel nie może być w 2 miejscach na raz.
- Nauczyciel musi być dostępny. Ze względu na charakter pracy nauczyciele często są zmuszeni do pracy w wielu miastach w wielu szkołach. Układając plan musimy brać pod uwagę ich dostępność.
- Żaden uczeń nie może być w 2 miejscach na raz.
- W żadnej sali nie mogą odbywać się 2 lekcje na raz.

Ograniczenia prawne

- Maksymalnie 9 godzin lekcyjnych dziennie.
- Nie więcej niż 2 godziny lekcyjne tego samego przedmiotu dziennie.
- Jeśli danego dnia mają zostać przeprowadzone 2 godziny jednego przedmiotu, to muszą one wystąpić bezpośrednio po sobie.

Ograniczenia jakościowe

- Brak okienek dla uczniów.
- Równomierny rozkład godzin na przestrzeni tygodnia. Nie może wystąpić sytuacja gdzie jednego dnia uczeń ma dwie lekcje, a następnego dziesięć.
- Odpowiednie przypisanie sal. Lekcje wychowania fizycznego muszą odbyć się w przeznaczonych do tego salach, podobnie lekcji informatyki itd.

Ograniczenie główne

Ilość godzin tygodniowo odbytych przez klasę z danym nauczycielem w ramach danego przedmiotu musi być równa ilości przypisanej w pierwszym etapie układania planu. Aby łatwiej zrozumieć na czym polega takie przypisanie warto spojrzeć na dotychczasowy sposób przypisywania ilości godzin nauczycieli do klas (Rysunek 3.1) w liceum, które dostarczyło dane na potrzebu tej pracy.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB
1	Projekt 2025/2026	psychol	jęz-tur	społ-pr	polit	b-ch	eko-men		psych-prawn	jez-ekonomi	polit-biol-	eko-men				dypl	jęz-tur	społ-pr	polit	b-ch	eko-men		dypl	jęz-tur	społ-pr	polit	b-ch	eko-men
2		I	I	I	I	I	I		II	II	II	II	II	II		III	III	III	III	III	III		IV	IV	IV	IV	IV	IV
3		A	B	C	D	F	G		AC	BG	DF	G				A	B	C	D	F	G		A	B	C	D	F	G
4	J.Polski	6	4	6	4	4	4		6	4	4	4	0	0		4	4	6	4	4	4		4	4	5	4	4	4
5																							3.16	3.16	3.95	3.16	3.16	3.16
6	Jp1		4						6	4		4																4
7																							0.00	0.00	0.00	0.00	0.00	3.16
8	Jp2						4										4		4								4	
9																							0.00	0.00	0.00	0.00	3.16	0.00
10	Jp3			6														6		4			4					
11																							3.16	0.00	0.00	0.00	0.00	0.00
12	Jp3				4	4															4			4	5			
13																							0.00	3.16	3.95	0.00	0.00	0.00
14	Jp4	6									4					4										4		
15																							0.00	0.00	0.00	3.16	0.00	0.00
16	Jp5																											
17																							0.00	0.00	0.00	0.00	0.00	0.00

Figure 3.1: Zrzut ekranu z arkusza kalkulacyjnego przedstawiający przypisanie godzinowe nauczycieli dla każdej klasy

Każdy nauczyciel jest przypisany do prowadzonych przez niego przedmiotów. Następnie w odpowiednim wierszu nauczyciela, pod odpowiednim przedmiotem, w kolumnie każdej klasy definiowana jest ilość godzin, która będzie poświęcona na prowadzenie tego przedmiotu.

3.2. Poprzednie podejścia

Aby w pełni zrozumieć mój wybór narzędzi warto szybko przetoczyć historię moich poprzednich podejść do rozwiązania tego problemu.

3.2.1. Programowanie zero-jedynkowe

Moim pierwszym podejściem była próba użycia tylko i wyłącznie solvera liniowego. Użyłem w tym celu pakietu *IBM ILOG CPLEX*. Problem zdefiniowałem używając zmiennych binarnych, tworząc sześcio wymiarową macierz:

1. Wymiar nauczycieli
2. Wymiar klas
3. Wymiar przedmiotów
4. Wymiar dnia
5. Wymiar slotu czasowego
6. Wymiar sal

Jak można zauważyć złożoność pamięciowa takiego podejścia uniemożliwia jego efektywne skalowanie. Nawet dla danych średniej szkoły, mającej mniej niż 100 sal, nauczycieli, klas i przedmitów, taka macierz zajmowała setki GB pamięci RAM. Rozwiązaniem tego problemu było zastosowanie słownika z wartościami jako zmienne binarne i kluczami jako krotki 6 liczb całkowitych. W ten sposób pozbywam się wszystkich niemożliwych wartości, przykładowo wychowania fizycznego z nauczycielem matematyki. Używając tej metody nadal możemy używać intuicji, która towarzyszy z macierzą. Musimy tylko sprawdzać czy dane zmienne binarne faktycznie istnieją.

Jest to intuicyjny sposób poradzenia sobie z problemem harmonogramowania zajęć o stałem długości. Bardzo łatwo można definiować ograniczenia fizyczne. Przykładowo ograniczenie prowadzenia maksymalnie jednej lekcji dla wszystkich nauczycieli:

$$\forall t \in \{0, 1, 2, \ldots, t_{\max}\} \quad \sum_{c=0}^{c_{\max}} \sum_{s=0}^{s_{\max}} \sum_{d=0}^{4} \sum_{h=0}^{h_{\max}} \sum_{r=0}^{r_{\max}} x_{t,c,s,d,h,r} = 1$$

Gdzie:

- t to indeks nauczyciela i $t_{\max}$ to liczba nauczycieli minus jeden (indeksowanie zaczynamy od 0)
- c to indeks klasy i podobnie $c_{\max}$ to liczba klas minus jeden
- d to dzień, gdzie 1 oznacza poniedziałek, a 4 piątek
- h to slot czasowy, a $h_{\max}$ to horyzont

- r to indeks sali, a $r_{\max}$ to liczba wszystkich sal minus jeden

Problemem tego rozwiązania jest niemożność wprowadzenia bloków lekcyjnych przy jednoczesnym zachowaniu prostoty obliczeniowej. Bardzo ciężkim okazało się również wprowadzenie minimalizacji liczby okienek. Zmienne binarne nie oferują wystarczającej wszechstronności.

3.2.2. Grafowe sieci neuronowe

W odpowiedzi na problemy z MIP, zdecydowałem się na zbadanie alternatywnych metod rozwiązania. Wybrałem niekonwencjonalną reprezentację problemu harmonogramowania używając grafowych sieci neuronowych [5]. W przyjętym modelu problem został przedstawiony w postaci grafu, gdzie węzły reprezentowały wymagania główne (3.1.1), a krawędzie łączyły zajęcia o tych samych nauczycielach lub tych samych klasach.

Taka reprezentacja okazała się szczególnie atrakcyjna pod względem implementacji funkcji celu. Ocena dopuszczalności rozwiązania sprowadzała się do weryfikacji spełnienia ograniczeń, które można było w prosty sposób zamodelować za pomocą funkcji kary Jednocześnie można nagradzać model za przypisywanie lekcji do poprawnych bloków. Początkowe rezultaty były bardzo obiecujące — model już po kilkadziesięciu epokach wykazywał zdolność do identyfikowania, które lekcji powinny być w blokach, a które wymagają rozdzielenia w celu uniknięcia kolizji.

Główną wadą w tym podejściu okazał się brak gwarancji spełnienia ograniczeń twardych oraz trudności przy układaniu planu iteracyjnie (lekcja po lekcji). Eksperymenty wykazały, że przy zastosowaniu zbyt wysokich współczynników kary, proces uczenia nie przynosił rezultatów. Zbyt niskie natomiast powodowały, że model preferował optymalizację nagrody za grupowanie lekcji kosztem naruszenia ograniczeń.

3.3. Wybór metod i technologi

W świetle przeprowadzonych eksperymentów i poprzednich prób rozwiazania problemu zauważyłem, że poleganie wyłącznie na metodach inteligentnych lub MIP nie doprowadzi do sensownych rezultatów. Zdecydowałem się na użycie:

- Algorytmu zachłannego do łączenia lekcji w bloki, gdyż jest to najefektywniejsza i najprostsza metoda do tego zadania.
- Algorytmu ewolucyjnego do przydziału godzin do każdego dnia tygodnia.
- Solvera liniwego do ułożenia samego planu dla każdego dnia tygodnia osobno.

Ze względu na prostotę integracji z backendem aplikacji użyłem języka programowania Python. Otwarta natura narzędzia Google OR-Tools oraz jego wygodne API w Pythonie skłoniło mnie do decyzji przeciwko CPLEX i Gurobi.

3.4. Dane i parametry

Opis działania algorytmu warto zacząć od przedstawienia sposobu reprezentacji wymagań w kodzie zaczynając od wymagań głównych. Zaczynając procedurę generowania planu lekcji zaczynamy od pobrania z bazy danych odpowiednich rekordów. Do otrzymanych w ten sposób rezultatów należy: lista wymagań głównych, lista nauczycieli, lista sal, lista klas, lista przedmiotów, lista dostępności nauczycieli,

Podczas wszystkich etapów tworzenia planu lekcji mamy dostęp do list:

- ID nauczycieli,
- ID klas,
- ID przedmiotów,
- ID sal.

Ponadto mamy do dyspozycji listę wymagań głównych, gdzie każde z nich jest reprezentowane klasą [3] w Pythonie, która ma poniższe atrybuty:

- ID nauczyciela,
- ID klasy,
- ID przedmiotu,
- niezerowa liczba godzin.

Każdy nauczyciel ma też zdefiniowaną dostępność, która jest reprezentowana przez macierz A o wymiarach T na 5, gdzie T to liczba wszystkich nauczycieli. Macierz jest zero-jedynkowa, gdzie 1 oznacza, że t-ty nauczyciel jest dostępny i-tego dnia tygodnia, a 0 że nie jest dostepny.

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} \\ \vdots & \vdots & \vdots & \vdots & \\ & & & & \\ a_{T,1} & a_{T,2} & a_{T,3} & a_{T,4} & a_{T,5} \end{bmatrix}, \quad \forall t \in \{1, 2, \ldots, T\}, \forall i \in \{1, 2, 3, 4, 5\}, \quad a_{t,i} \in \{0, 1\}$$

W celu tworzenia bloków mamy także dostęp do klasy bloku przedmiotów, która ma następujące atrybuty:

- Listę ID przedmiotów.
- Listę liczby tych przedmiotów w bloku.
- Klas których dotyczy blok.
- Wartość binarną informującą o tym czy blok jest *agregujący*.
- Liczbę naturalną z informacją o maksymalnej ilości takich bloków tygodniowo.

3.5. Struktura algorytmu

Pierwszy etap algorytmu służy do stworzenia bloków lekcyjnych.

3.6. Algorytm zachłanny

Opowiedzenie czym jest algorytm zachłanny najlepiej w oparciu o jakąś pracę naukową.

3.6.1. Bloki lekcyjne

Operując na blokach lekcyjnych duży łatwiej zdefiniować ograniczenia fizyczne. Weźmy na przykład 3 lekcje: Język Niemiecki, Język Francuski oraz Język Rosyjski. Gdybyśmy mieli definiować dla nich ograniczenie określające, że żaden uczeń nie może mieć przypisanych dwóch lub więcej lekcji w tym samym czasie, musielibyśmy zapewnić, że żadna z tych lekcji nie pokrywa się z innymi lekcjami tej klasy, takimi jak Matematyka czy Fizyka, przy jednoczesnym zapewnieniu, że te lekcji mogą się na siebie nałożyć. Te zajęcia stanowią obowiązkowy język dodatkowy, który uczniowie wybierają każdy z osobna. Każdy uczeń może wybrać tylko jeden język, co za tym idzie lekcje mogą odbywać się niezależnie od siebie. Grupując takie ograniczenia główne w bloki 3 lekcji możemy potraktować taki blok jak każdą inną lekcję przy definiowaniu ograniczeń — nie może być przypisany do tego samego slotu czasowego z żadną inną lekcją.

Następną zaletą jest prostota w projektowaniu ograniczeń dla lekcji, które odbywają się dla więcej niż jednej klasy. Często w szkołach brakuje uczniów zapisanych na przykładowo Język Rosyjski w jednej klasie, aby uzasadnić indywidualną lekcję prowadzoną przez nauczyciela z tylko i wyłącznie jedną klasą. W takich przypadkach szkoła definiuje lekcje, które nauczyciel prowadzi dla wielu klas jednocześnie. Podobnie jak w poprzednim przykładzie, definiowanie ograniczeń dla każdej lekcji z osobna wiąże się z wyjątkami. Jeśli natomiast połączymy wymagania główne dla paru klas w jeden blok, możemy go traktować tak jak każdą inną lekcję.

Kolejną zaletą tego rozwiązania jest możliwość łączenia bloków w jeszcze większe bloki. Wyobraźmy sobie sytuację, w której mamy trzy klasy: IIIA, IIIB i IIIC, oraz 3 przedmioty do przeprowadzenia: Język Niemiecki, Język Francuski i Język Rosyjski. Z uwagi na niską liczbę uczniów zapisanych na rosyjski i francuski te zajęcia sa prowadzone w następujących grupach:

- wszystkie 3 klasy mają razem Język Rosyjski,
- klasa IIIA i IIIB mają razem Język Francuski, a klasa IIIC ma indywidualnie z innym nauczycielem,
- każda klasa ma indywidualnie Język Niemiecki, z czego klasa IIIA i IIIC mają tego samego nauczyciela.

Jak można łatwo zauważyć wszystkie te lekcji poza jedną lekcją języka niemieckiego mogą odbyć się jednocześnie. Po stworzeniu wieloklasowych bloków lekcyjnych możemy je łączyć dalej. Język Rosyjski może być połączony z blokiem Języka Francuskiego dla klas IIIA i

IIIB oraz lekcją klasy IIIC. Do tego możemy też dodać dwie lekcje języka Niemieckiego pozostawiając ostatnią lekcję poza blokiem ze względu na kolizję nauczycieli.

3.6.2. Działanie

Pseudokod i dokładne działanie.
Ważnym jest wyjście tego algorytmu:

- Lista bloków,
- Wymagana tygodniowa liczba godzin każdego bloku.

Efektem tych działań jest lista krotek takich wymagań oraz lista przypisań godzinowych.

- Lista bloków: $[(\text{Req } 1, \text{Req } 2, \text{Req } 3)]$
- Macierz wymaganych odbytych godzin każdego bloku na przestrzeni tygodnia.

$$V = \begin{bmatrix} v_1 & v_2 & \cdots & v_N \end{bmatrix}, \quad \forall i \in \{1, 2, \ldots, N\}, \quad v_i \in \mathbb{N}$$

gdzie N to liczba wszystkich bloków.

- Na potrzeby dalszych rozważań możemy przyjąć, że mamy dostęp także do macierzy

$$B = \begin{bmatrix} b_1 & b_2 & \cdots & b_N \end{bmatrix}, \quad \forall i \in \{1, 2, \ldots, N\}, \quad b_i \in \mathbb{N}^+$$

gdzie b_i to liczba wymagań w i-tym bloku. W faktycznym kodzie nie definiuję tej macierzy, tylko w razie potrzeby sprawdzam liczbę elementów w krotce.

3.7. Algorytm ewolucyjny

Opowiedzenie czym jest algorytm ewolucyjny najlepiej w oparciu o jakąś pracę naukową.

3.7.1. Cel algorytmu

Algorytm ma na celu przydział godzin lekcyjnych z macierzy bloków V do pięciu roboczych dni tygodnia. Dla każdego bloku generowanych jest pięć wartości całkowitoliczbowych reprezentujących liczbę godzin lekcyjnych przydzielonych do poszczególnych dni, przy zachowaniu wymagań wynikających z poprzedniego etapu przetwarzania.

3.7.2. Kodowanie

Każdy osobnik w populacji jest reprezentowany przez macierz przydziałów S o wymiarach $5 \times N$.

$$S = \begin{bmatrix} s_{1,1} & s_{1,2} & s_{1,3} & \cdots & s_{1,B} \\ s_{2,1} & s_{2,2} & s_{2,3} & \cdots & s_{2,B} \\ s_{3,1} & s_{3,2} & s_{3,3} & \cdots & s_{3,B} \\ s_{4,1} & s_{4,2} & s_{4,3} & \cdots & s_{4,B} \\ s_{5,1} & s_{5,2} & s_{5,3} & \cdots & s_{5,B} \end{bmatrix}$$

gdzie $s_{d,i}$ oznacza liczbę godzin i-tego bloku lekcyjnego przydzielonych do d-tego dnia tygodnia.

Tak przyjęta reprezentacja umożliwia prostą weryfikację następujących ograniczeń:

1. Całkowita liczba godzin każdego bloku musi odpowiadać wymaganiom określonym w macierzy V.
$$\forall i \in \{1, 2, \ldots, B\}, \quad \sum_{d=1}^{5} p_{d,i} = v_i$$
2. Maksymalna liczba godzin danego bloku w pojedynczym dniu nie może być większa niż 2.
$$\forall i \in \{1, 2, \ldots, B\}, \forall d \in \{1, 2, 3, 4, 5\}, \quad p_{d,i} \leq 2$$
3. Dla bloków 3-godzinnych konieczne jest przedzielenie 2 godzin do jednego dnia.
$$\forall i \in \{1, 2, \ldots, B\}, \quad v_i = 3 \implies \exists d \in \{1, 2, 3, 4, 5\} \text{ takie, że } s_{d,i} = 2$$

3.7.3. Generowanie populacji początkowej

Generowanie osobników odbywa się losowo z uwzględnieniem wymagań bloków oraz dostępności nauczycieli. Wykorzystuję w tym celu rozkład wielomianowy [2], który zapewnia spełnienie podstawowych ograniczeń. Proces generowania pojedynczego osobnika składa się z następujących etapów wykonywanych dla każdego i-tego bloku:

1. **Zagregowanie dostępności nauczycieli:**
Dla każdego dnia wyznaczana jest dostępność wszystkich nauczycieli przypisanych do i-tego bloku.
$$\forall d \in \{1, 2, 3, 4, 5\}, \quad a'_d = \bigwedge_{j=1}^{b_i} a_{t_j,d}$$
gdzie t_j to identyfikator j-tego nauczycielaw bloku. W dalszej części algorytmu wartości logiczne traktujemy jako wartości zerojedynkowe.
2. **Jeśli** $v_i = 3$ **to:**
   (a) $X_i \leftarrow \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \end{bmatrix}$
   (b) $k, l \leftarrow$ losowe dni, dla których $a'_i = a'_j = 1$
   (c) $x_k \leftarrow 2, x_l \leftarrow 1$

   Po wykonaniu tych kroków generowanie dla bieżącego bloku jest zakończone.
3. **Stworzenie macierzy prawdopodobieństw wystąpienia bloku w danym dniu:**
Tworzymy wektor prawdopodobieństw:
$$P = \begin{bmatrix} \frac{a'_1}{\sum_{i=1}^{5} a'_i} & \frac{a'_2}{\sum_{i=1}^{5} a'_i} & \frac{a'_3}{\sum_{i=1}^{5} a'_i} & \frac{a'_4}{\sum_{i=1}^{5} a'_i} & \frac{a'_5}{\sum_{i=1}^{5} a'_i} \end{bmatrix}$$
$$P = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 & p_5 \end{bmatrix}$$
tak stworzone prawdopodobieństwa zapewniają, że $p_d = 0$ jeśli którykolwiek nauczyciel jest niedostępny d-tego dnia oraz $\sum_{d=1}^{5} p_d = 1$.

4. **Losowanie z rozkładu wielomianowego:**
   Korzystając z implementacji biblioteki `numpy` [1], generowana jest próbka:

   $$\texttt{numpy.random.multinomial(r, } \begin{bmatrix} p_1 & p_2 & \cdots & p_5 \end{bmatrix} \texttt{)}$$

   Wynikiem tej funkcji jest macierz $X_i = \begin{bmatrix} x_0 & x_1 & \cdots & x_5 \end{bmatrix}$, która ze względu na własności rozkładu spełnia:

   $$\begin{cases} x_d \in \mathbb{N} & \forall d \in \{1, 2, \ldots, 5\} \\ \sum_{d=1}^{5} x_d = v_i \end{cases}$$

5. **Korekcja przekroczeń limitu dziennego:**
   Dopóki $\exists d \in \{1, 2, 3, 4, 5\} : x_d > 2$:

   (a) Dla każdego $d \in \{1, 2, 3, 4, 5\}$ spełniającego $x_d > 2$:

      i. $X'_i \leftarrow$ `numpy.random.multinomial(`$x_d - 2$`,` P`)`
      ii. $x_d \leftarrow 2$
      iii. $X_i \leftarrow X_i + X'_i$

Po wykonaniu powyższej procedury dla wszystkich bloków, uzyskane wektory X_i łączy się w macierz:

$$X = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_N \end{bmatrix}$$

Osobnik populacji reprezentowany jest przez transpozycję tej macierzy: $S = X^T$.

3.7.4. Przystosowanie

Ocena jakości osobników odbywa się na podstawie czterech niezależnych metryk:

- Równomierność rozkładu godzin pracy nauczycieli w ciągu tygodnia
- Równomierność rozkładu godzin lekcyjnych klas w ciągu tygodnia
- Liczba dni, w których nauczyciele muszą pojawić się w szkole
- Różnica między najdłuższym i najkrótszym dniem lekcyjnym dla klas

W początkowej fazie prac wykorzystałem funkcję gęstości rozkładu normalnego z wartością oczekiwaną $\mu = 8$, co wynika z ośmiogodzinnego dnia pracy. Okazało się jednak, że takie podejście zapewniało wyłącznie nagrody, nie oferując mechanizmu karania niepożądanych rozwiązań. Ograniczało to zdolność algorytmu do korekcji błędów. W odpowiedzi na te wyzwania zaprojektowałem funkcję kwadratową:

$$f(x) = -(7 - x)^2 + 2$$

Jak widać na wykresie funkcji 3.2, jej głównym zadaniem jest karanie dni o skrajnym obciążeniu dydaktycznym zarówno dla nauczycieli, jak i uczniów. Wierzchołek funkcji umieściłem w punkcie $x = 7$, a nie $x = 8$, ponieważ pięciogodzinny dzień pracy jest znacznie bardziej akceptowalny niż dziesięciogodzinny. Funkcja nagradza wartości z przedziału $(5, 9)$, przy czym wartości bliskie 7 otrzymują maksymalną ocenę. Wartości spoza tego przedziału są znacząco karane, co zapewnia spełnienie dwóch kluczowych warunków: brak ponad dziesięciogodzinnych dni pracy dla nauczycieli oraz utrzymanie około siedmiogodzinnych dni lekcyjnych dla uczniów.

Nawet w przypadkach, gdy program nauczania przekracza 35 godzin tygodniowo (co uniemożliwia dodatnią wartość tej funkcji przy równomiernym rozłożeniu), funkcja zachowuje swoją przydatność w zapewnianiu zgodności z wymaganiami równomiernych rozkładów godzin. Ze względu na malejącą pochodną w przedziale $[7, +\infty)$, rozwiązania z jednym dniem 7-godzinnym kosztem dnia 11-godzinnego otrzymują gorszą ocenę niż rozwiązania z dwoma dniami 9-godzinnymi, co promuje bardziej zrównoważony rozkład obciążenia. będą gorzej oceniane niż osobniki z dwoma dniami o 9 godzinach.

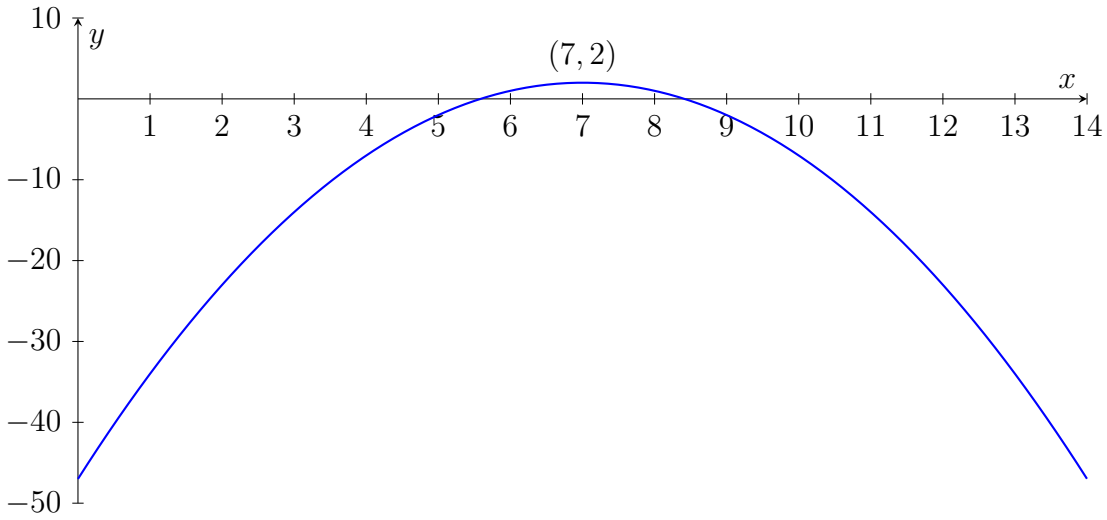

Figure 3.2: Wykres funkcji użytej do ewaluacji rozkładu godzin nauczycieli i klas na przestrzeni tygodnia.

Ocena równomierności rozpoczyna się od obliczenia dla każdego nauczyciela wektora obciążenia godzinowego:

$$\begin{bmatrix} t_{i,1} & t_{i,2} & t_{i,3} & t_{i,4} & t_{i,5} \end{bmatrix}$$

gdzie $t_{i,d}$ oznacza liczbę godzin lekcyjnych do przeprowadzenia i-tego nauczyciela d-tego dnia. Następnie wyznaczam sumę wartości funkcji dla wszystkich argumentów tego wektora. Problemem jest to, że osobniki są karane, kiedy zgodnie z przydziałem nauczyciel ma dzień wolny — jest to niezgodne z założeniami. Wolimy, aby nauczyciej miał cztery dni 8-godzinne niż pięć dni po 6 lub 7 godzin. Aby osiągnąć taką właściwość zeruję wartość funkcji dla $x = 0$, co skutkuje następującą funkcją:

$$\begin{cases} f(x) = 0 & \text{x} = 0 \\ f(x) = -(7 - x)^2 + 2 & \text{wpp.} \end{cases}$$

3.7.5. Krzyżowanie

3.7.6. Mutacja

3.7.7. Jaki jest oczekiwany najlepszy osobnik

3.8. Solver liniowy do układania planu dla każdego dnia osobno

3.8.1. Czym są interwały

Wyjaśnienie jak reprezentuję bloki zajęć przez intervały

3.8.2. Ograniczenia

- Nienakładanie się lekcji dla każdej klasy
- Nienakładanie się lekcji dla każdego nauczyciela
- Nienakładanie się lekcji w jendym pokoju
- Lekcje kończące/zaczynające
- Brak okienek dla uczniów

3.8.3. Funkcja celu

Jak reprezentujemy okienka nauczycieli. Dlaczego interwały w tym pomagają.

3.9. Wyniki

- Dlaczego moje wyniki są wspaniałe
- Średni czas potrzebny na generację planu

3.9.1. Statystyki planu

- Ilość okienek
- Rozkład lekcji w tygodniu
- Lekcje początkujące/kończące
- Statystyki nauczycieli, godziny w szkole do godzin lekcyjnych (płatnych)

3.9.2. Porównanie z ręcznie ułożonym planem

Porównanie z planem, który szkoła ułożyła ręcznie.

4. Aplikacja

4.1. Specyfikacja wymagań

4.1.1. Wymagania funkcjonalne

4.1.2. Wymagania niefunkcjonalne

4.1.3. Przypadki użycia

4.2. Projekt

4.2.1. Projekt bazy danych

4.2.2. Projekt interfejsu

4.2.3. Sposób integracji z algorytmem

4.3. Implementacja

4.3.1. Wybór narzędzi

Dlaczego React+Django. Dlaczego PostgreSQL

4.3.2. Implementacja bazy danych w Django

4.3.3. Implementacja API w Django

4.3.4. Implementacja interfejsu w React

4.3.5. Integracja z algorytmem

5. Testowanie i ewaluacja rozwiązania

5.1. Scenariusze testowe

5.1.1. Scenariusz 1

5.1.2. Scenariusz 2

5.2. Testowanie aplikacji

5.2.1. Funkcjonalność 1

5.2.2. Funkcjonalność 2

6. Wnioski i perspektywy rozwoju

Zakończenie, podsumowuje najważniejsze wnioski, podaje możliwości dalszego rozwinięcia wykonanych prac i wskazuje obszar potencjalnego zastosowania pracy. Rezultaty pracy mają charakter poznawczy, mogą mieć charakter użytkowy. Należy dokonać analizy uzyskanych wyników. Rezultaty powinny charakteryzować się oryginalnością, a nawet w pewnym stopniu nowatorstwem. Praca zawiera (...). Zostało pokazane (...). Eksperymenty wykazały (...). Tu piszemy wnioski i obserwacje.

Widzimy, że (...). Z tego powodu przyszła praca powinna obejmować (...).

Na pewno będę miał sporo rzeczy, które wiem, że będę chciał poprawić w przyszłości.

Bibliography

[1] N. Developers. Documentation, numpy.random.multinomial, (2025). Ostatnio otworzono 14 listopada 2025.

[2] W. Feller et al. *An introduction to probability theory and its applications*, volume 963. Wiley New York, 1971.

[3] D. S. Foundation. Documentation, models, (2025). Ostatnio otworzono 12 listopada 2025.

[4] E. Rappos, E. Thiémard, S. Robert, and J.-F. Hêche. A mixed-integer programming approach for solving university course timetabling problems. *Journal of Scheduling*, 25(4):391–404, 2022.

[5] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pages 593–607. Springer, 2018.

List of Figures

List of Tables