

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

Streszczenie

Bardzo fajny algorytm w bardzo fajnej aplikacji

Abstract

Very nice algorytm w bardzo nice aplikacji

Spis treści

1	Wstęp	1
1.1	Cel i zakres pracy	1
1.2	Układ pracy	2
2	Powiązane prace	3
2.1	A mixed-integer programming approach for solving university course timeta- bling problems.	3
2.2	Job-Shop Problem	3
2.3	Istniejące rozwiązania	4
2.3.1	Vulcan	4
2.3.2	Dobry Plan??	4
3	Problem układania planu lekcji i algorytm jego rozwiązania	5
3.1	Sformułowanie problemu optymalizacyjnego	5
3.1.1	Ograniczenia	7
3.1.2	Dane i szukane	8
3.2	Poprzednie podejścia	10
3.2.1	Programowanie zero-jedynkowe	10
3.2.2	Graflowe sieci neuronowe	11
3.3	Struktura algorytmu i wybór technologii	12
3.4	Algorytm zachłanny	12
3.4.1	Bloki lekcyjne	13
3.4.2	Działanie	14
3.4.3	Przykładowe rezultaty	15
3.5	Algorytm ewolucyjny	15
3.5.1	Cel algorytmu	17
3.5.2	Kodowanie i ograniczenia twarde	17
3.5.3	Generowanie populacji początkowej	18
3.5.4	Przystosowanie	19
3.5.5	Selekcja	22
3.5.6	Krzyżowanie	22
3.5.7	Mutacja	23
3.5.8	Przykładowe rezultaty	24
3.6	Solver liniowy	24
3.6.1	Czym są interwały	25
3.6.2	Zmienne decyzyjne oraz wymagania ilościowe sal	25

3.6.3	Ograniczenia	26
3.6.4	Funkcja celu	27
3.7	Wyniki	27
3.7.1	Statystyki planu	27
3.7.2	Porównanie z ręcznie ułożonym planem	28
4	Aplikacja	29
4.1	Specyfikacja wymagań	29
4.1.1	Wymagania funkcjonalne	29
4.1.2	Wymagania нефункционалне	29
4.1.3	Przypadki użycia	29
4.2	Projekt	29
4.2.1	Projekt bazy danych	29
4.2.2	Projekt interfejsu	29
4.2.3	Sposób integracji z algorytmem	29
4.3	Implementacja	29
4.3.1	Wybór narzędzi	29
4.3.2	Implementacja bazy danych w Django	29
4.3.3	Implementacja API w Django	29
4.3.4	Implementacja interfejsu w React	29
4.3.5	Integracja z algorytmem	29
5	Testowanie i ewaluacja rozwiązania	31
5.1	Scenariusze testowe	31
5.1.1	Scenariusz 1	31
5.1.2	Scenariusz 2	31
5.2	Testowanie aplikacji	31
5.2.1	Funkcjonalność 1	31
5.2.2	Funkcjonalność 2	31
6	Wnioski i perspektywy rozwoju	33

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. **Przypisania godzin rozpoczęcia i skończenia lekcji oraz sal, w których będą się odbywać.** Mając już informację o ilości godzin lekcyjnych, które każda klasa musi odbyć z każdym nauczycielem, możemy przejść do tworzenia właściwego planu. Głównym ograniczeniem jest stworzony w etapie 1. przydział godzin. Istnieje jednak wiele innych ograniczeń, które sprawiają, że proces ręcznego układania takiego planu 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 harmonogramowaniem z twardymi i miękkimi ograniczeniami. 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 zaprogramowanie problemu programowania liniowego, który można rozwiązać w satysfakcjonującym czasie. Umożliwia to jednoczesne zachowanie wysokiej jakości rozwiązania, poprzez użycie pakietów optymalizacyjnych, 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, która będzie wykorzystywać algorytm ewolucyjny przy automatycznej generacji planu. Aplikacja musi umożliwiać użytkownikowi na automatyczne generowanie planu na podstawie wprowadzonych danych. Pod uwagę będą brane ograniczenia fizyczne takie jak dostępność sal i kolizje zajęć, ograniczenia jakościowe takie jak ciągłość 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.

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 będzie używał algorytmu genetyczny, algorytmu zachłannego oraz solvera liniowego.

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

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

Na przestrzeni lat wykształcono wiele podejść do rozwiązywania takich problemów. Jednym z nich jest hybrydowe podejście, opierające się na jednoczesnym zastosowaniu metod sztucznej inteligencji oraz metod programowania całkowitoliczbowego.

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

Rappos, E., Thiémarc, 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 [10], którą luźno się interesowałem — też rozbiła problem na 2 etapy. Też stopniowo rozwiązywała problem i stopniowo dodawała ograniczenia do solvera liniowego.

Jak będzie trzeba więcej.

2.2. Job-Shop Problem

Problem harmonogramowania typu *Job-Shop* (JSSP) jest jednym z najbardziej klasycznych i istotnych problemów optymalizacji kombinatorycznej w badaniach operacyjnych i zarządzaniu produkcją. Ze względu na bardzo szerokie zastosowania inżynierskie był intensywnie studiowany przez wielu autorów. Klasyczny wariant można opisać następująco [12].

Posiadamy zbiór maszyn:

$$M = \{M_1, M_2, \dots, M_m\}$$

oraz zbiór zadań (prac):

$$J = \{J_1, J_2, \dots, J_n\}.$$

Każde zadanie (job) J_i składa się z liniowo uporządkowanej sekwencji operacji:

$$O_i = \{O_{i1}, O_{i2}, \dots, O_{in_i}\},$$

które muszą zostać wykonane w ściśle określonej kolejności technologicznej:

$$O_{i1} \prec O_{i2} \prec \dots \prec O_{in_i}.$$

Każda operacja O_{ij} jest przypisana do dokładnie jednej maszyny poprzez odwzorowanie:

$$\mu : (i, j) \mapsto k \quad \text{tak że } O_{ij} \text{ musi być przetwarzana na } M_{\mu(i,j)},$$

oraz posiada znany czas przetwarzania:

$$p_{ij} \in \mathbb{N}^+.$$

Celem jest wyznaczenie kolejności (szeregowania) i czasów rozpoczęcia wszystkich operacji na maszynach tak, aby zminimalizować maksymalny czas zakończenia dowolnego zadania (tzw. *makespan*):

$$C_{\max} = \max_i C_i,$$

gdzie

$$C_i = S_{in_i} + p_{in_i},$$

a S_{ij} oznacza czas rozpoczęcia operacji O_{ij} .

Zmienne decyzyjne.

$$S_{ij} \in \mathbb{N}_0 \quad \forall i = 1, \dots, n, \quad \forall j = 1, \dots, n_i.$$

Ograniczenia precedencji.

$$S_{i,j+1} \geq S_{ij} + p_{ij} \quad \forall i, \quad \forall j = 1, \dots, n_i - 1.$$

Ograniczenia nieprzerwalności. Każda operacja po rozpoczęciu zajmuje swoją maszynę nieprzerwanie przez p_{ij} jednostek czasu (modelowane implicit przez definicję przedziału $[S_{ij}, S_{ij} + p_{ij})$).

Ograniczenia dysjunktywne maszyn. Dla każdej pary różnych operacji (O_{ij}, O_{kl}) przypisanych do tej samej maszyny ($\mu(i, j) = \mu(k, l) = k$) zachodzi jedno z dwóch:

$$S_{ij} + p_{ij} \leq S_{kl} \quad \vee \quad S_{kl} + p_{kl} \leq S_{ij}.$$

W liniowym sformułowaniu wprowadza się binarne zmienne $y_{(ij),(kl)}$:

$$S_{ij} + p_{ij} \leq S_{kl} + M \cdot y_{(ij),(kl)}, \quad S_{kl} + p_{kl} \leq S_{ij} + M \cdot (1 - y_{(ij),(kl)}),$$

gdzie M jest wystarczająco dużą stałą.

Funkcja celu.

$$\min C_{\max} \quad \text{przy} \quad C_{\max} \geq S_{in_i} + p_{in_i} \quad \forall i.$$

2.3. Istniejące rozwiązania

2.3.1. Vulcan

2.3.2. Dobry Plan??

3. Problem układania planu lekcji i algorytm jego rozwiązania

Opis struktury rozdziału.

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.
- **Slot czasowy:** Czas w którym odbywa się lekcja; przykładowo slot zerowy może odbywać się od 7:00 do 7:45.
- **Blok lekcyjny:** Grupa dwóch lub więcej lekcji, które odbywają się w tym samym slocie 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 powinno być pięć zbiorów przypisań \mathcal{Z}_d dla każdego roboczego dnia tygodnia $d \in \{1, 2, 3, 4, 5\}$. Każde takie przypisanie powinno mieć 5 wartości:

$$z_i \triangleq (d, h, c, t, s, r) \in \mathcal{Z}_d, \quad \begin{cases} h & \in \{1, 2, \dots, H\} \\ c & \in \mathcal{C} \\ t & \in \mathcal{T} \\ s & \in \mathcal{S} \\ r & \in \mathcal{R} \\ i & \end{cases}$$

gdzie:

- \mathcal{C} to zbiór klas,
- \mathcal{T} to zbiór nauczycieli,
- \mathcal{S} to zbiór przedmiotów,
- \mathcal{R} to zbiór sal,
- H to wartość rozpatrywanego horyzontu czasowego.

Przykładowe interpretacje takich przypisań:

- Poniedziałek, Slot czasowy 0, IA, Sala nr 1, nauczyciel francuskiego 1, język francuski
- Poniedziałek, Slot czasowy 0, IIA, Sala nr 1, nauczyciel francuskiego 1, język francuski
- \vdots
- Piątek, Slot czasowy 10, IVC, Sala nr 15, nauczyciel fizyki 2, fizyka

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

Proces tworzenia takich przypisań można rozbić na 3 etapy:

1. Tworzenie bloków lekcyjnych z ograniczeń głównych zdefiniowanych w 3.1.1:

- Łączenie ograniczeń głównych w lekcje międzyklasowe.
- Łączenie lekcji międzyklasowych w bloki lekcyjne.
- Łączenie ograniczeń głównych w bloki lekcyjne.

2. Przydział bloków lekcyjnych do poszczególnych dni tygodnia.

- Przykładowo: dwie godziny matematyki w poniedziałek, godzina matematyki w środę, dwie godziny matematyki w czwartek i zero w piątek.

3. Tworzenie samych przypisań dla każdego dnia tygodnia 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. Bardzo rzadko dochodzi do sytuacji, gdzie nauczyciel faktycznie jest w stanie wyrobić pełen etat w jednej szkole, w szczególności w małych wsiach i miastach. 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

[1, 2, 3]

- Uczeń nie może mieć więcej niż 2 godzin lekcyjnych tego samego przedmiotu dziennie.
- Jeśli danego dnia mają zostać przeprowadzone 2 godziny jednego przedmiotu, to uczeń musi je mieć 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 potrzeby tej pracy.

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.

	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	BA	
1	Projekt 2025/2026	psychol	jęz-tur	spot-pr	polit	b-ch	eko-men		psych-prawn	jęz-ekonomi	polit-biol-	eko-men					dysl	jęz-tur	spot-pr	polit	b-ch	eko-men		dysl	jęz-tur	spot-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	III	IV	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																						0.00	0.00	0.00	0.00	0.00	0.00	
17																							0.00	0.00	0.00	0.00	0.00	0.00	
18																													

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

Zbiór takich wymagań \mathcal{W} definiuje się następująco:

$$w_i \triangleq (t_i, c_i, s_i, g_i) \in \mathcal{W}, \quad \begin{cases} t & \in \mathcal{T} \\ c & \in \mathcal{C} \\ s & \in \mathcal{S} \\ g & \in \mathbb{N}^+ \\ i & \in \{1, 2, \dots, |\mathcal{W}|\} \end{cases}$$

gdzie g to liczba wymaganych tygodniowo godzin przedmiotu s przeprowadzonego przez nauczyciela t dla klasy c .

3.1.2. Dane i szukane

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: $\mathcal{C}, \mathcal{T}, \mathcal{S}, \mathcal{R}, \mathcal{W}$.

Ponadto każdy nauczyciel ma też zdefiniowaną dostępność, która jest reprezentowana przez macierz A o wymiarach $|\mathcal{T}|$ na 5, gdzie $|\mathcal{T}|$ to liczba wszystkich nauczycieli. Macierz jest zero-jedynkowa, gdzie 1 oznacza, że t -ty nauczyciel jest dostępny d -tego dnia tygodnia, a

0 że jest niedostępny.

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{|\mathcal{T}|,1} & a_{|\mathcal{T}|,2} & a_{|\mathcal{T}|,3} & a_{|\mathcal{T}|,4} & a_{|\mathcal{T}|,5} \end{bmatrix}, \quad \forall t \in \{1, 2, \dots, |\mathcal{T}|\}, \forall d \in \{1, 2, 3, 4, 5\} : a_{t,d} \in \{0, 1\}$$

W celu tworzenia bloków mamy także dostęp do zbioru bloków przedmiotów \mathcal{B} :

$$(\vec{S}_b, \vec{C}_b, \vec{G}_b, \text{is_aggregating}, \text{max_weekly}) \in \mathcal{B}, \quad \begin{cases} \vec{S}_b & \subset \mathcal{S} \\ \vec{C}_b & \subseteq \mathcal{C} \\ \vec{G}_b & \in \mathbb{N}^{|\vec{S}_b|} \\ \text{is_aggregating} & \in \{0, 1\} \\ \text{max_weekly} & \in \mathbb{N} \end{cases}$$

gdzie

- \vec{S}_b to podzbiór wszystkich przedmiotów, których dotyczy blok b ,
- \vec{C}_b to podzbiór wszystkich klas, których dotyczy blok b ,
- \vec{G}_b to wektor oczekiwanej liczby przedmiotów w bloku.
- is_aggregating to wartość binarną informującą o tym czy blok jest *agregujący*.
- max_weekly to liczba naturalna która informuje o maksymalnej liczbie bloków. Jeśli $\text{max_weekly} = 0$ to oznacza to, że liczba bloków jest nielimitowana.

Przykładowy blok przedmiotów:

- $\vec{S}_b = (\text{język angielski}, \text{informatyka})$,
- $\vec{C}_b = (\text{IA}, \text{IB}, \text{IC})$,
- $\vec{G}_b = (1, 1)$,
- $\text{is_aggregating} = 0$, (*To nie jest blok argumentujący*)
- $\text{max_weekly} = 0$, *nielimitowana tygodniowo liczba godzin*

lub:

- $\vec{S}_b = (\text{język angielski})$,
- $\vec{C}_b = \mathcal{C}$
- $\vec{G}_b = (2)$,
- $\text{is_aggregating} = 0$
- $\text{max_weekly} = 0$

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 pakietu optymalizacyjnego *IBM ILOG CPLEX*. Problem zdefiniowałem używając zmiennych binarnych, tworząc sześćo 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 przedmiotów, 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:

$$X = \{(t, c, d, h, s, r) : x_{t,c,d,h,s,r}\}, \quad \begin{cases} t & \in \{1, 2, \dots, \mathfrak{T}\} \\ c & \in \{1, 2, \dots, \mathfrak{C}\} \\ d & \in \{1, 2, \dots, 5\} \\ h & \in \{1, 2, \dots, \mathfrak{H}\} \\ s & \in \{1, 2, \dots, \mathfrak{S}\} \\ r & \in \{1, 2, \dots, \mathfrak{R}\} \end{cases}$$

$$\forall t \in \{1, 2, \dots, \mathfrak{T}\}, c \in \{1, 2, \dots, \mathfrak{C}\}, d \in \{1, 2, \dots, 5\}, \\ h \in \{1, 2, \dots, \mathfrak{H}\}, s \in \{1, 2, \dots, \mathfrak{S}\}, r \in \{1, 2, \dots, \mathfrak{R}\} : \quad x_{(t,c,d,h,s,r)} \in \{0, 1\}$$

gdzie

- \mathfrak{T} to liczba nauczycieli,
- \mathfrak{C} to liczba klas,
- H to wartości horyzontu,

- \mathfrak{S} to liczba przedmiotów,
- \mathfrak{R} to liczba dostępnych sal,
- d reprezentuje dzień tygodnia,
- h reprezentuje slot czasowy.

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 niezmienniej długości jednego slotu czasowego. Bardzo łatwo można definiować ograniczenia fizyczne.

- Ograniczenie prowadzenia maksymalnie jednej lekcji dla wszystkich nauczycieli:

$$\forall t \in \{0, 1, 2, \dots, \mathfrak{T}\}, \quad \sum_{c=0}^{\mathfrak{C}} \sum_{s=0}^{\mathfrak{S}} \sum_{d=1}^5 \sum_{h=0}^H \sum_{r=0}^{\mathfrak{R}} x_{t,c,s,d,h,r} = 1$$

- Ograniczenie posiadania maksymalnie jednej lekcji w jednym slotcie czasowym dla wszystkich uczniów:

$$\forall c \in \{0, 1, 2, \dots, \mathfrak{C}\}, \quad \sum_{t=0}^{\mathfrak{T}} \sum_{s=0}^{\mathfrak{S}} \sum_{d=1}^5 \sum_{h=0}^H \sum_{r=0}^{\mathfrak{R}} x_{t,c,s,d,h,r} = 1$$

- Ograniczenie maksymalnie jednej lekcji w pokoju:

$$\forall r \in \{0, 1, 2, \dots, \mathfrak{R}\}, \quad \sum_{t=0}^{\mathfrak{T}} \sum_{s=0}^{\mathfrak{S}} \sum_{d=1}^5 \sum_{h=0}^H \sum_{c=0}^{\mathfrak{C}} x_{t,c,s,d,h,r} = 1$$

Tak zdefiniowany problem bardzo szybko znajdował rozwiązania, które spełniały wszystkie ograniczenia fizyczne.

Problemem tego rozwiązania jest niemożność wprowadzenia bloków lekcyjnych przy jednoczesnym zachowaniu prostoty obliczeniowej. Kolejnym wyzwaniem było również wprowadzenie minimalizacji liczby okienek dla nauczycieli oraz wprowadzenie wymagania ciągłości zajęć dla wszystkich klas. Zmienne binarne nie oferują wystarczającej wszechstronności, która jest potrzebna w układaniu planów zajęć.

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 [11]. 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 kilkadziesiąt epokach wykazywał zdolność do identyfikowania, które lekcje 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. Struktura algorytmu i wybór technologii

W świetle przeprowadzonych eksperymentów i poprzednich prób rozwiązania problemu zauważyłem, że poleganie wyłącznie na metodach inteligentnych lub MIP nie doprowadzi do sensownych rezultatów. Główna innowacja zaproponowanego rozwiązania leży w dekompozycji oryginalnego zadania na trzy sekwencyjne fazy, co nie było obecne w moich pierwszych próbach. 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.

Zdecydowałem się na użycie trzech następujących technik:

1. Algorytmu zachłannego do tworzenia bloków lekcyjnych z ograniczeń głównych.
2. Algorytmu ewolucyjnego do przydziału bloków lekcyjnych do poszczególnych dni tygodnia.
3. Programowania liniowego do tworzenia przypisań opisanych w sekcji 3.1 dla każdego dnia 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. Algorytm zachłanny

Algorytm zachłanny (ang. *greedy*) polega na iteracyjnym podejmowaniu lokalnie najlepszej decyzji bez cofania się i bez globalnego przeszukiwania przestrzeni rozwiązań [4]. Choć w wielu problemach algorytm zachłanny daje rozwiązania przybliżone, w zadaniach konstrukcyjnych, w których celem jest wyłącznie znalezienie wszystkich maksymalnych (niedających się już powiększyć) struktur, jego prostota jest zaletą: eliminuje zbędną warstwę optymalizacji.

W etapie budowy bloków lekcyjnych nie maksymalizujemy funkcji celu ani nie porównujemy jakości wariantów — każdy poprawny (zgodny z ograniczeniami) blok jest akceptowalny. Celem jest:

1. Pokrycie wszystkich wymagań głównych.
2. Rozpatrzenie wszystkich bloków przedmiotów.
3. Utworzenie możliwie największych spójnych bloków (maksymalnych względem liczby składowych wymagań głównych).
4. Uniknięcie duplikacji i kolizji (spełnienie ograniczeń).

Problem sprowadza się do iterowania się po blokach przedmiotów i tworzenia wszystkich, maksymalnie dużych, bloków lekcyjnych. Próba użycia metod optymalizacyjnych (MIP, metaheurystyki) na tym etapie wprowadzałaby koszt obliczeniowy bez zysku jakościowego: już sam algorytm zachłanny pozwala na stworzenie maksymalnie dużych bloków. Algorytm zachłanny pozwala szybko eksplorować przestrzeń lokalną: dla każdej potencjalnej konfiguracji rozszerzamy blok dopóki wszystkie warunki (brak konfliktu nauczycieli, zgodność klas, ...) pozostają spełnione. Po wyczerpaniu możliwości blok osiąga swoją maksymalną wielkość. Wielkość innych bloków nie wpływa na ten wynik, co oznacza, że rozwiązanie optymalne lokalnie jest również optymalne globalnie.

3.4.1. Bloki lekcyjne

Operując na blokach lekcyjnych dużo 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 lekcje 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 są 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 lekcje 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.

3.4.2. Działanie

Dane wejściowe

- \mathcal{B}
- \mathcal{W}

Klasyfikacja bloków

- **Bloki pojedyncze** — obejmują pojedyncze klasy
- **Bloki wieloklasowe** — łączą wymagania z wielu klas dla tych samych przedmiotów
- **Bloki agregujące** — tworzą wyższy poziom hierarchii, łącząc istniejące bloki i wymagania w jeszcze większe bloki

Działanie

1. Inicjalizacja struktur danych:

- Słownik śledzący wykorzystanie godzin wymagań
- Listy przechowujące pogrupowane wymagania

2. Tworzenie bloków wieloklasowych:

- Grupowanie wymagań według przedmiotów i klas zdefiniowanych w blokach wieloklasowych
- Każda grupa tworzy potencjalny blok lekcyjny

3. Agregacja bloków:

- Łączenie bloków wieloklasowych w większe jednostki

- Unikanie konfliktów nauczycieli poprzez śledzenie wykorzystanych pedagogów
- Przydział godzin oparty na minimalnym wymaganiu w grupie

4. Generowanie bloków pojedynczych:

- Tworzenie kombinacji wymagań dla pojedynczych klas
- Uwzględnienie przypadków z wieloma nauczycielami tego samego przedmiotu
- Walidacja zgodności z definicjami bloków

5. Iteracyjna alokacja godzin:

- Równomierny przydział pozostałych godzin do bloków
- Respektowanie maksymalnych limitów tygodniowych
- Gwarancja wykorzystania wszystkich wymaganych godzin

6. Finalizacja:

- Dodanie pozostałych wymagań jako bloków jednostkowych
- Zwrócenie listy bloków i odpowiadających im godzin

Wynikiem działania algorytmu zachłannego jest efektywna reprezentacja bloków gotowa do dalszego generowania planu. Algorytm zwraca:

- Zbiór bloków lekcyjnych \mathcal{L} , gdzie każdy blok L_i jest wekrotem zbioru wymagań głównych:

$$\mathcal{L} = \{L_1, L_2, L_{\mathfrak{N}}\}, \quad \forall i \in \{1, 2, \dots, \mathfrak{N}\} : L_i \subset \mathcal{W}$$

gdzie \mathfrak{N} to liczba wszystkich bloków.

- Macierz wymagań głównych bloków — liczby godzin tygodniowych, gdzie v_i oznacza liczbę godzin przeznaczonych na blok b_i na przestrzeni tygodnia.

$$V = \begin{bmatrix} v_1 & v_2 & \cdots & v_{\mathfrak{N}} \end{bmatrix}, \quad \forall i \in \{1, 2, \dots, \mathfrak{N}\}, \quad v_i \in \mathbb{N}^+$$

3.4.3. Przykładowe rezultaty

3.5. Algorytm ewolucyjny

Algorytm ewolucyjny jest populacyjną metodą przeszukiwania inspirowaną mechanizmami doboru naturalnego: selekcją, krzyżowaniem i mutacją. Klasyczne podstawy teorii zaprezentował Holland [9]. W odróżnieniu od podejść gradientowych lub zachłannych algorytm genetyczny eksploruje wiele konkurujących rozwiązań równolegle, co ułatwia balansowanie kilku kryteriów jakości oraz unikanie wczesnego utknięcia na lokalnych wierzchołkach funkcji celu.

W mojej pracy algorytm ewolucyjny pełni rolę drugiego etapu dekompozycji: po redukcji problemu do listy bloków lekcyjnych pozostaje dyskretne zadanie rozdziału tygodniowych godzin każdego bloku na pięć dni roboczych przy prostych ograniczeniach.

Zastosowany algorytm ewolucyjny wnosi:

1. **Naturalną obsługę wielu kryteriów** poprzez dodawanie komponentów funkcji przystosowania.
2. **Specjalistyczne operatory** krzyżowania i mutacji działające na poziomie kolumn (bloków), co zachowuje poprawność bez kosztownych procedur naprawczych.
3. **Niską złożoność obliczeniową** dzięki małej liczbie zmiennych w jednym osobniku (5 wartości na blok) i możliwości jednoczesnego oceniania całej populacji używając operacji na macierzach.
4. **Łatwe dostrajanie wag** (priorytety klas kontra nauczyciele) bez zmian w strukturze modelu.

Ogólna struktura algorytmu:

Algorithm 3.1: “The general scheme of an evolutionary algorithm in pseudocode” [6]

```

1 INITIALISE population with random individuals;
2 EVALUATE each individual;
3 repeat
4   SELECT parents;
5   RECOMBINE pairs of parents;
6   MUTATE the resulting offspring;
7   EVALUATE new individuals;
8   SELECT individuals for the next generation;
9 until TERMINATION CONDITION is satisfied;
```

gdzie w mojej pracy:

- **INITIALISE** reprezentuje budowę pierwszej populacji (3.5.3) przez losowe generowanie kolumn zgodnie z dostępnością i wymaganiami bloków.
- **EVALUATE** oblicza wartości funkcji przystosowania (3.5.4) dla każdego osobnika według zdefiniowanych metryk.
- **SELECT parents** realizuje selekcję probabilistyczną (3.5.5) z przewagą lepiej ocenionych osobników.
- **RECOMBINE pairs of parents** tworzy potomków (3.5.6) przez kolumnowe dziedziczenie struktur bloków (bez łamania ograniczeń).
- **MUTATE the resulting offspring** wprowadza różnorodność przez rekonstrukcję wybranych kolumn (3.5.7) z rozkładu wielomianowego.
- **SELECT individuals for the next generation** implementuje elitaryzm i utrzymuje stały rozmiar populacji.
- **TERMINATION CONDITION** kończy proces po ustalonej liczbie generacji.

3.5.1. Cel algorytmu

Algorytm ma na celu przydział godzin lekcyjnych z macierzy bloków V do pięciu roboczych dni tygodnia reprezentowanych przez macierz S (3.1). Dla każdego bloku generowanych jest pięć wartości całkowitoliczbowych reprezentujących liczbę godzin lekcyjnych przydzielonych do poszczególnych dni, przy zachowaniu wymagań wynikających z poprzedniego etapu przetwarzania. Końcowy przydział godzin powinien mieć następujące cechy:

- Spełnione ograniczenia twarde
- Równomierny przydział bloków na przestrzeni tygodnia dla klas
- Równomierny przydział bloków na przestrzeni tygodnia dla nauczycieli

Dane wejściowe

- \mathcal{L}
- V
- A
- \mathcal{B}
- \mathcal{C}
- \mathcal{T}
- \mathcal{S}

Dane wyjściowe

- Poprawna macierz S_{best} :
 - spełniająca wszystkie ograniczenia twarde,
 - posiadająca satysfakcjonującą wartość funkcji przystosowania.

3.5.2. Kodowanie i ograniczenia twarde

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

$$S = \begin{bmatrix} s_{1,1} & s_{1,2} & s_{1,3} & \cdots & s_{1,\mathfrak{N}} \\ s_{2,1} & s_{2,2} & s_{2,3} & \cdots & s_{2,\mathfrak{N}} \\ s_{3,1} & s_{3,2} & s_{3,3} & \cdots & s_{3,\mathfrak{N}} \\ s_{4,1} & s_{4,2} & s_{4,3} & \cdots & s_{4,\mathfrak{N}} \\ s_{5,1} & s_{5,2} & s_{5,3} & \cdots & s_{5,\mathfrak{N}} \end{bmatrix}, \quad S \in \mathbb{N}^2 \quad (3.1)$$

gdzie $s_{d,i}$ oznacza liczbę godzin i -tego bloku lekcyjnego L_i 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 .

$$\forall i \in \{1, 2, \dots, \mathfrak{N}\} : \sum_{d=1}^5 s_{d,i} = v_i$$

2. Maksymalna liczba godzin każdego bloku w pojedynczym dniu nie może być większa niż 2.

$$\forall i \in \{1, 2, \dots, \mathfrak{N}\}, \forall d \in \{1, 2, 3, 4, 5\} : s_{d,i} \leq 2$$

3. Dla bloków 3-godzinnych konieczne jest przedzielenie 2 godzin do jednego dnia.

$$\forall i \in \{1, 2, \dots, \mathfrak{N}\} : v_i = 3 \implies \exists d \in \{1, 2, 3, 4, 5\} \text{ takie, że } s_{d,i} = 2$$

3.5.3. Generowanie populacji początkowej

Generowanie osobników odbywa się losowo z uwzględnieniem wymagań bloków oraz dostępności nauczycieli. Wykorzystuję w tym celu rozkład wielomianowy [7], 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.

$$\forall d \in \{1, 2, 3, 4, 5\} : a'_d = \bigwedge_{\substack{t \in \bigcup \\ w_i \in L_i}} a_{t,d}$$

gdzie t_j to identyfikator j -tego nauczyciela w bloku. W dalszej części algorytmu wartości logiczne traktujemy jako wartości zerojedynekowe.

2. **Jeśli $v_i = 3$ to:**

- (a) $X_i \leftarrow [0 \ 0 \ 0 \ 0 \ 0]$
- (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:

$$P = \begin{bmatrix} \frac{a'_1}{\sum_{i=1}^5 a'_i} & \frac{a'_2}{\sum_{i=1}^5 a'_i} & \frac{a'_3}{\sum_{i=1}^5 a'_i} & \frac{a'_4}{\sum_{i=1}^5 a'_i} & \frac{a'_5}{\sum_{i=1}^5 a'_i} \end{bmatrix}$$

$$P = [p_1 \ p_2 \ p_3 \ p_4 \ p_5]$$

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` [5], generowana jest próbka:

$$\text{numpy.random.multinomial}(r, [p_1 \ p_2 \ \cdots \ p_5])$$

Wynikiem tej funkcji jest macierz $X_i = [x_0 \ x_1 \ \cdots \ x_5]$, która ze względu na własności rozkładu spełnia:

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

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. $X'_i \leftarrow \text{numpy.random.multinomial}(x_d - 2, P)$
- 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: $S = X^T$.

3.5.4. Przystosowanie

Ocena jakości osobników odbywa się na podstawie czterech niezależnych metryk:

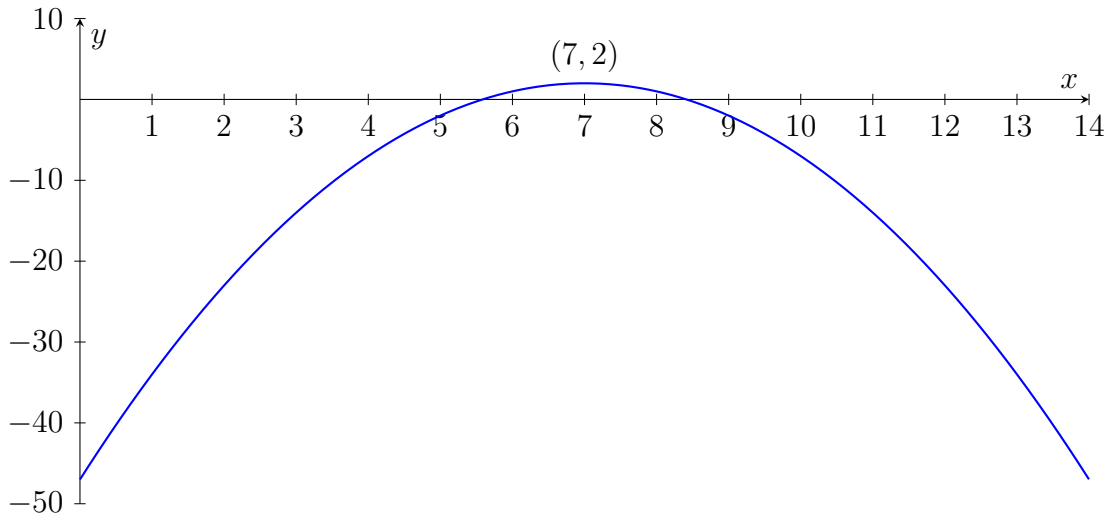
- Równomierność rozkładu godzin pracy nauczycieli w ciągu tygodnia
- Równomierność rozkładu godzin lekcyjnych klas w ciągu tygodnia
- Liczba dni, w których nauczyciele muszą pojawić się w szkole
- Różnica między najdłuższym i najkrótszym dniem lekcyjnym dla klas

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-godzinny kosztom 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.



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

Ocena względem nauczycieli

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

$$T_i = [t_{i,1} \ t_{i,2} \ t_{i,3} \ t_{i,4} \ t_{i,5}]$$

gdzie $t_{i,d}$ oznacza liczbę godzin lekcyjnych do przeprowadzenia przez i -tego nauczyciela d -tego dnia.

Problemem funkcji kwadratowej jest fakt, że osobniki otrzymują dużą karę, kiedy zgodnie z przydziałem nauczyciel ma dzień wolny — jest to niezgodne z założeniami. Wolimy, aby nauczyciel miał cztery dni 8-godzinne niż pięć dni po 6 lub 7 godzin. Aby osiągnąć taką

właściwość zeruje wartość funkcji dla $x = 0$, co skutkuje następującą funkcją:

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

Następnie wyznaczam sumę wartości funkcji dla wszystkich argumentów tego wektora:

$$F_{c_{T_i}} = \sum_{d=1}^5 f_T(t_{i,d})$$

gdzie $|\mathcal{T}|$ to liczba wszystkich nauczycieli.

Końcowa ocena wszystkich nauczycieli to najzwyczajniej suma wszystkich wartości:

$$F_{c_T} = \sum_{i=1}^{|\mathcal{T}|} F_{c_{T_i}}$$

Ocena względem klas

Ocena równomierności z respektem do klas jest analogiczna do poprzedniej. Tworzony jest wektor obciążenia godzinowego:

$$G_i = [g_{i,1} \ g_{i,2} \ g_{i,3} \ g_{i,4} \ g_{i,5}]$$

gdzie $g_{i,d}$ oznacza liczbę godzin lekcyjnych przez i -tej klasy d -tego dnia.

W tym przypadku dni wolne nie są zgodne ze względu na ograniczenia opisane w 3.1.1, a więc modyfikacje funkcji nie są potrzebne.

Analogicznie do nauczycieli tworzymy kolejno wektory i wartości:

$$F_{c_{G_i}} = \sum_{d=1}^5 f(g_{i,d}) \qquad F_{c_G} = \sum_{i=1}^{\mathfrak{G}} F_{c_{G_i}}$$

gdzie \mathfrak{G} to liczba wszystkich klas.

Normalizacja i wagi

Wyznaczanie końcowej wartości funkcji przystosowania wymaga uwzględnienia stosunku liczby nauczycieli do liczby uczniów. W typowych placówkach oświatowych liczba nauczycieli przywysza liczbę klas. Bezpośrednie dodawanie ocen $F_{c_T} + F_{c_G}$ prowadziłoby do znaczącej dominacji oceny nauczycieli w ocenie końcowej. W celu rozwiązania tego problemu zastosowałem normalizację poprzez dzielenie każdej składowej przez odpowiednio $|\mathcal{T}|$ i \mathfrak{G} . Takie podejście gwarantuje, że wpływ pojedynczej klasy na ocenę końcową jest porównywalny z wpływem pojedynczego nauczyciela.

Kolejnym aspektem wymagającym uwzględnienia jest relatywna ważność obu kryteriów oceny. W praktyce, wymagania równomiernego rozkładu dla każdej klasy są ważniejsze niż równomierny rozkład godzin nauczycieli. Aby umożliwić sterowanie tymi preferencjami należy zaimplementować wagi, co prowadzi do następującej postaci funkcji przystosowania:

$$F_c = \frac{\alpha_T}{|\mathcal{T}|} F_{c_T} + \frac{\alpha_G}{\mathfrak{G}} F_{c_G}, \quad \alpha_T, \alpha_G \in \mathbb{R}^+$$

gdzie α_T to waga oceny nauczycieli, a α_G to waga oceny klas.

3.5.5. Selekcja

Zastosowałem tradycyjną metodę ruletkową. Polega ona na losowaniu osobników, którzy posłużą jako rodzice z prawdopodobieństwami, które są proporcjonalne do wartości ich funkcji przystosowania. Mając wektor ocen $F_{c \text{ total}} = [F_{c_1} \ F_{c_1} \ \dots \ F_{c_{\mathfrak{P}}}]$, gdzie \mathfrak{P} to rozmiar populacji, sortujemy go i dzielimy go na pół. Drugą połowę populacji, która ma najgorsze wyniki, usuwam i na jej miejsce wstawiam potomków rodziców wylosowanych zgodnie z prawdopodobieństwami:

$$P = [\sigma(F_{c_1}) \ \sigma(F_{c_2}) \ \dots \ \sigma(F_{c_{\mathfrak{P}}})], \quad \sigma(F_{c_i}) = \frac{\exp[F_{c_i}]}{\sum_{j=1}^{\mathfrak{P}} \exp[F_{c_j}]}$$

W celu zamiany funkcji przystosowania na prawdopodobieństwa użyłem funkcji SoftMax.

3.5.6. Krzyżowanie

Największym wyzwaniem podczas projektowania algorytmu okazało się opracowanie krzyżowania osobników, które zagwarantowałyby spójność z nałożonymi ograniczeniami. Niemożność zaimplementowania takich rozwiązań jak proste modyfikowanie krzyżowania jednopunktowego zmusiło mnie do opracowania specjalistycznych metod.

Z uwagi na dwie składowe funkcji zdecydowałem się na zaimplementowanie dwóch niezależnych metod krzyżowania osobników. Takie podejście umożliwia równoczesne dziedziczenie cech związanych z optymalnym rozkładem zajęć zarówno z perspektywy nauczycieli, jak i klas. Pozwala to także na stworzenie dwóch różnych potomków z dwóch rodziców.

Względem oceny nauczycieli

Funkcja krzyżowania uwzględniająca ocenę rozkładu godzin nauczycieli definiuje się następująco:

Dane wejściowe:

- S_1 — pierwszy rodzic
- S_2 — drugi rodzic
- $F_{T_{S_1}}$ — ocena nauczycieli pierwszego osobnika
- $F_{T_{S_2}}$ — ocena nauczycieli drugiego osobnika

Proces:

1. Inicjalizacja macierzy potomka: $S_{\text{child}} = \mathbf{0}_{5 \times \mathfrak{N}}$
2. Dla każdego nauczyciela $i \in 1, 2, \dots, |\mathcal{T}|$:
 - (a) Jeżeli $F_{T_{i,S_1}} > F_{T_{i,S_2}}$, to dla każdego bloku j prowadzonego przez nauczyciela i :
 - i. Przepisz kolumnę j z macierzy S_1 do kolumny j macierzy S_{child}
 - (b) W przeciwnym przypadku, dla każdego bloku j prowadzonego przez nauczyciela i :

- i. Przepisz kolumnę j z macierzy S_2 do kolumny j macierzy S_{child}

Wygenerowany w ten sposób osobnik S_{child} ma cechy, które gwarantują jak najlepsze przypisanie nauczycieli wśród obu rodziców. Następną zaletą tego podejścia jest fakt, że gwarantuje to też także spełnienie wszystkich wymagań, jako że wszystkie wymagania odnoszą się do indywidualnych bloków, a tych nie zmieniamy, tylko przepisujemy.

Względem oceny klas

Analogicznie definiujemy krzyżowanie względem oceny klas:

Dane wejściowe:

- S_1 — pierwszy rodzic
- S_2 — drugi rodzic
- $F_{G_{S_1}}$ — ocena klas pierwszego osobnika
- $F_{G_{S_2}}$ — ocena klas drugiego osobnika

Proces:

1. Inicjalizacja macierzy potomka: $S_{\text{child}} = \mathbf{0}_{5 \times \mathfrak{N}}$
2. Dla każdej klasy $i \in 1, 2, \dots, \mathfrak{G}$:
 - (a) Jeżeli $F_{G_{i,S_1}} > F_{G_{i,S_2}}$, to dla każdego bloku j prowadzonego dla klasy i :
 - i. Przepisz kolumnę j z macierzy S_1 do kolumny j macierzy S_{child}
 - (b) W przeciwnym przypadku, dla każdego bloku j prowadzonego przez nauczyciela i :
 - i. Przepisz kolumnę j z macierzy S_2 do kolumny j macierzy S_{child}

Wygenerowany w ten sposób osobnik S_{child} ma cechy, które gwarantują jak najlepszy rozkład godzin lekcyjnych dla klas wśród obu rodziców. Podobnie w tym przypadku nie modyfikujemy przydziału godzin w indywidualnych blokach, więc zachowujemy zgodność z ograniczeniami.

3.5.7. Mutacja

Analogicznie do przypadku krzyżowania, zastosowanie prostych metod mutacji (takich jak losowa zamiana pojedynczych przydziałów) nie jest możliwe ze względu na wysokie ryzyko naruszenia ograniczeń. W szczególności, modyfikacja pojedynczych wartości macierzy S może prowadzić do niezgodności z warunkiem głównym bloków.

W odpowiedzi na to wyzwanie, przyjąłem strategię mutacji operującą na poziomie bloków — kolumn macierzy przydziałów, podobną do podejścia zastosowanego w funkcji krzyżowania. Pozwana to na zachowania zgodności z ograniczeniami.

Procedura mutacji definiuje się następująco:

1. Z populacji wybieranych jest losowo n osobników.
2. Dla każdego wybranego osobnika wybierany jest losowy podzbiór kolumn (bloków lekcyjnych).
3. Dla każdej wybranej kolumny j wylosuj nowy przydział zgodnie z procedurą opisaną w podrozdziale 3.5.3.

Takie podejście gwarantuje, że zmutowane osobniki zachowują zgodność z podstawowymi ograniczeniami problemu, jednocześnie wprowadzając do populacji nowe warianty rozkładów godzinowych wybranych bloków lekcyjnych.

W celu zwiększenia stabilności procesu ewolucyjnego zaimplementowałem także elitarnego osobnika, który zawsze pojawia się niezmienny w następnej generacji. Jest to osobnik o największej wartości funkcji przystosowania:

$$S_{\text{best}} = S_i, \quad i = \arg \max_{j \in \{1, 2, \dots, \mathfrak{P}\}} F_{c_j}$$

3.5.8. Przykładowe rezultaty

Przykładowe macierze z opisami

3.6. Solver liniowy

Wyjaśnienie czym jest solver liniowy

Opisany poniżej proces jest wykonywany niezależnie dla każdego dnia tygodnia, reprezentowanego przez wiersze macierzy S_{best} . Takie podejście umożliwia dekompozycję problemu na pięć mniejszych, niezależnych podproblemów, co znacząco redukuje wymagania obliczeniowe i

zużycie pamięci RAM.

3.6.1. Czym są interwały

W odpowiedzi na wyzwania napotkane w podejściu opisanym w podrozdziale 3.2.1 zdecydowałem się na eksplorację innych sposobów reprezentacji lekcji niż zmienne binarne. W tym procesie napotkałem się na opisane przez Google `Interval Variables` [8]. Są one sposobem reprezentacji interwałów poprzez trzy powiązane zmienne, z fundamentalnym ograniczeniem:

Zmienne:

Ograniczenie:

- s — czas rozpoczęcia $e - s = d$
- d — czas trwania
- e — czas zakończenia

OR-Tools oferuje także `OptionalIntervals`, które reprezentuje interwał z dodatkową zmienną `is_present`, która decyduje o tym czy interwał jest aktywny przy sprawdzaniu ograniczeń. Traktowana jest jako zmienna zerojedynkowa przy dodawaniu:

Zmienne:

Ograniczenie:

- s — czas rozpoczęcia $\text{is_present} \implies e - s = d$
- d — czas trwania
- e — czas zakończenia
- `is_present` — zmienna logiczna

3.6.2. Zmienne decyzyjne oraz wymagania ilościowe sal

Dla każdego bloku lekcyjnego b_i dla którego $v_i > 0$ definiujemy:

1. $s_i \leftarrow \text{NewIntVar}(0, \text{horizon})$
2. $e_i \leftarrow \text{NewIntVar}(0, \text{horizon})$
3. $\text{Interval}_i \leftarrow \text{NewIntervalVar}(s_i, s_{\text{best}_{d,i}}, e_i)$

Dla każdego przedmiotu z oraz liczby nauczycieli $|\mathcal{T}|_{i,z}$ prowadzących ten przedmiot w bloku b_i definiujemy:

1. $\forall r \in \{1, 2, \dots, \mathfrak{R}_z\}$:
 - (a) $\text{is_present}_{i,j,r} \leftarrow \text{NewBoolVar}()$
 - (b) $\text{OptionalInterval}_{i,j,r} \leftarrow \text{NewOptionalIntervalVar}(s_i, s_{\text{best}_{d,i}}, e_i, \text{is_present}_{i,j,r})$

gdzie $j \in \{1, 2, \dots, |\mathcal{T}|_{i,z}\}$, oraz \mathfrak{R}_z to liczba sal obsługujących przedmiot z .

2. Na tym etapie dodajemy także ograniczenie do modelu:

$$\sum_{r=1}^{\mathfrak{R}_z} \sum_{j=1}^{|\mathcal{T}|_{i,z}} \text{is_present}_{i,j,r} = |\mathcal{T}|_{i,z}$$

Wszystkie zdefiniowane ograniczenia w tens sposób możemy wyrazić językiem naturalnym jako: “Liczba sal przypisanych do bloku musi być równa liczbie nauczycieli w bloku b_i oraz przypisane sale muszą obsługiwać przedmioty prowadzone w ramach bloku b_i ”.

3.6.3. Ograniczenia

Dwa takie same przedmioty muszą być obok siebie

Jedną z licznych zalet stosowania interwałów do reprezentacji bloków lekcyjnych jest fakt, że sama ich definicja gwarantuje nam następstwo takich samych bloków. Tworzymy bloki o długości $s_{d,i}$, co za tym idzie jeśli dwie godziny są przypisane do danego dnia

Nienakładanie się lekcji dla każdej klasy i brak okienek dla uczniów

Dla każdej klasy g wykonujemy następującą procedurę:

1. Tworzymy listę indeksów bloków/interwałów D_g , których bloki obejmują rozpatrywaną klasę g .
2. Dodajemy ograniczenie `model.AddNoOverlap([Intervali for i in D_g])`
3. Tworzymy dwie zmienne pomocnicze:
 - `day_start ← model.NewIntVar(0, horizon)`
 - `day_end ← model.NewIntVar(0, horizon)`

4. Dodajemy następujące ograniczenia:

$$\begin{aligned} \text{(a) } \text{day_start} &= \min_{i \in D_g} \{s_i\} \\ \text{(b) } \text{day_end} &= \max_{i \in D_g} \{e_i\} \\ \text{(c) } \text{day_end} - \text{day_start} &= \sum_{i \in D_g} v_i \end{aligned}$$

Nienakładanie się lekcji dla każdego nauczyciela

Dla każdego nauczyciela t wykonujemy:

1. Tworzymy listę indeksów bloków/interwałów D_t , których bloki obejmują rozpatrywanego nauczyciela t .
2. Dodajemy ograniczenie `model.AddNoOverlap([Intervali for i in D_t])`

Nienakładanie się lekcji w jednej sali

Dla każdej sali r wykonujemy:

1. Tworzymy listę wszystkich istniejących opcjonalnych interwałów

$$\text{room_intervals}_r = [\text{OptionalInterval}_{i,j,r}], \quad \forall i \in \{1, 2, \dots, \mathfrak{N}\}, \forall j \in \{1, 2, \dots, |\mathcal{T}|\}$$

2. Dodajemy ograniczenie `model.AddNoOverlap(room_intervals_r)`

3.6.4. Funkcja celu

Funkcja celu ma na celu minimalizację czasu spędzanego przez nauczycieli w szkole, co pośrednio redukuje liczbę okienek. Możemy w tym celu użyć bardzo podobnej architektury jak w ograniczeniu, które gwarantuje ciągłość lekcji dla każdej klasy.

Stworzymy zmienne dla każdego nauczyciela t :

- `day_start` \leftarrow `model.NewIntVar(0, horizon)`
- `day_end` \leftarrow `model.NewIntVar(0, horizon)`
- `day_durationt` = `day_end` – `day_start`

Dodajmy następujące ograniczenia:

1. `day_start` = $\min_{i \in D_t} \{s_i\}$
2. `day_end` = $\max_{i \in D_t} \{e_i\}$

Funkcja celu ma postać:

$$F_c = \sum_{t=1}^{|\mathcal{T}|} \text{day_duration}_t$$

Minimalizacja tej funkcji prowadzi do kompaktowego układania zajęć, redukując okienka w planach nauczycieli.

3.7. Wyniki

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

3.7.1. Statystyki planu

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

3.7.2. 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.

Bibliografia

- [1] Ustawy karta nauczyciela z dnia 26 stycznia 1982r. Dz. U. z 2021 r. poz. 1762 oraz z 2022 r. poz. 935, 1982.
- [2] Ustawy prawo oświatowe z dnia 14 grudnia 2016r. Dz. U. z 2017 r. poz. 60, 949 i 2203, z 2018 r. poz. 2245 oraz z 2019 r. poz. 1287, 2016.
- [3] Obwieszczenie ministra edukacji narodowej z dnia 4 września 2020 r. w sprawie ogłoszenia jednolitego tekstu rozporządzenia ministra edukacji narodowej i sportu w sprawie bezpieczeństwa i higieny w publicznych i niepublicznych szkołach i placówkach. Dz.U. 2020 poz. 1604, 2020.
- [4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2022.
- [5] N. Developers. Documentation, numpy.random.multinomial, (2025). Ostatnio otworzono 14 listopada 2025.
- [6] A. E. Eiben and J. E. Smith. What is an evolutionary algorithm? In *Introduction to evolutionary computing*, pages 15–35. Springer, 2015.
- [7] W. Feller et al. *An introduction to probability theory and its applications*, volume 963. Wiley New York, 1971.
- [8] Google. Documentation, or-tools - intervalvar, (2025). Ostatnio otworzono 15 listopada 2025.
- [9] J. H. Holland. Genetic algorithms. *Scientific american*, 267(1):66–73, 1992.
- [10] 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.
- [11] 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.
- [12] H. Xiong, S. Shi, D. Ren, and J. Hu. A survey of job shop scheduling problem: The types and models. *Computers & Operations Research*, 142:105731, 2022.

Spis rysunków

3.1	Zrzut ekranu z arkusza kalkulacyjnego przedstawiający przypisanie godzinowe nauczycieli dla każdej klasy	8
3.2	Wykres funkcji użytej do ewaluacji rozkładu godzin nauczycieli i klas na przestrzeni tygodnia.	20

Spis tabel