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

Spis treści

1 Wstęp **1**
- 1.1 Cel i zakres pracy . . . 1
- 1.2 Układ pracy . . . 2

2 Powiązane prace **3**
- 2.1 Inspiracja podejściem iteracyjnym . . . 3
  - 2.1.1 Adaptacja . . . 3
  - 2.1.2 Korzyści z podejścia iteracyjnego . . . 4
- 2.2 Job-Shop Problem . . . 5
  - 2.2.1 Podobieństwa do problemu układania planu lekcji . . . 5
  - 2.2.2 Różnice i specyfika problemu szkolnego . . . 7
  - 2.2.3 Gotowe rozwiązania . . . 7
- 2.3 Istniejące kompleksowe rozwiązania . . . 8
  - 2.3.1 Plan lekcji Optivum . . . 9

3 Problem układania planu lekcji i algorytm jego rozwiązania **15**
- 3.1 Sformułowanie problemu optymalizacyjnego . . . 15
  - 3.1.1 Dane i szukane . . . 17
  - 3.1.2 Ograniczenia . . . 19
- 3.2 Poprzednie podejścia . . . 20
  - 3.2.1 Programowanie zero-jedynkowe . . . 21
  - 3.2.2 Grafowe sieci neuronowe . . . 22
- 3.3 Struktura algorytmu i wybór technologii . . . 23
- 3.4 Algorytm zachłanny . . . 23
  - 3.4.1 Bloki lekcyjne . . . 24
  - 3.4.2 Działanie . . . 25
  - 3.4.3 Przykładowe rezultaty . . . 27
- 3.5 Algorytm ewolucyjny . . . 27
  - 3.5.1 Cel algorytmu . . . 28
  - 3.5.2 Kodowanie i ograniczenia twarde . . . 29
  - 3.5.3 Generowanie populacji początkowej . . . 30
  - 3.5.4 Przystosowanie . . . 31
  - 3.5.5 Selekcja . . . 33
  - 3.5.6 Krzyżowanie . . . 34
  - 3.5.7 Mutacja . . . 35
  - 3.5.8 Przykładowe rezultaty . . . 36

3.6 Solver programowania liniowego z ograniczeniami . . . 36
3.6.1 Reprezentacja interwałów . . . 36
3.6.2 Zmienne decyzyjne . . . 37
3.6.3 Ograniczenia . . . 37
3.6.4 Funkcja celu . . . 38
3.6.5 Transformacja interwałów na przypisania końcowe . . . 38
3.7 Wyniki . . . 40
3.7.1 Statystyki planu . . . 40
3.7.2 Porównanie z ręcznie ułożonym planem . . . 40

4 Aplikacja **41**
4.1 Specyfikacja wymagań . . . 41
4.1.1 Wymagania funkcjonalne . . . 41
4.1.2 Wymagania niefunkcjonalne . . . 42
4.1.3 Przypadki użycia . . . 42
4.2 Projekt . . . 42
4.2.1 Projekt bazy danych . . . 42
4.2.2 Projekt interfejsu . . . 42
4.2.3 Sposób integracji z algorytmem . . . 42
4.3 Implementacja . . . 42
4.3.1 Wybór narzędzi . . . 42
4.3.2 Implementacja bazy danych w Django . . . 43
4.3.3 Implementacja API w Django . . . 43
4.3.4 Implementacja interfejsu w React . . . 43
4.3.5 Integracja z algorytmem . . . 43

5 Testowanie i ewaluacja rozwiązania **45**
5.1 Scenariusze testowe . . . 45
5.1.1 Scenariusz 1 . . . 45
5.1.2 Scenariusz 2 . . . 45
5.2 Testowanie aplikacji . . . 45
5.2.1 Funkcjonalność 1 . . . 45
5.2.2 Funkcjonalność 2 . . . 45

6 Wnioski i perspektywy rozwoju **47**

1. Wstęp

Jednym z corocznych wyzwań placówek oświatowych jest ułożenie dobrego planu lekcji. Pomimo postępu technologicznego, proces ten nadal sprawia istotne trudności nawet najlepiej zorganizowanym szkołom. Problem można podzielić na dwa zasadnicze 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 liczbie uczniów. Dostępne są jedynie prognozy które pozwalają na określenie liczby klas, co zapewnia to ciągłość nauczania jednego nauczyciela z roku na rok dla danej klasy.

2. **Przypisania godzin rozpoczęcia i zakończenia lekcji oraz sal, w których będą się odbywać.** Mając już informację o liczbie godzin lekcyjnych, które każda klasa musi odbyć z każdym nauczycielem, możemy przejść do tworzenia właściwego planu. Głównym ograniczeniem jest stworzony w etapie pierwszym przydział godzin. Proces ten, wykonywany ręcznie, może zajmować dziesiątki godzin pracy, co często skutkuje chaosem organizacyjnym w pierwszych tygodniach roku szkolnego.

Drugi etap stanowi klasyczny problem harmonogramowania z ograniczeniami twardymi i miękkimi. W niniejszej pracy zaproponowałem podejście łączące algorytm ewolucyjny, inspirowany mechanizmami biologicznej ewolucji (mutacja, krzyżowanie, selekcja), z technikami programowania ograniczeń. Takie połączenie umożliwia osiągnięcie wysokiej jakości rozwiązań przy akceptowalnym czasie obliczeń, oferując praktyczny kompromis między optymalnością a wydajnością.

1.1. Cel i zakres pracy

Głównym celem pracy jest opracowanie i implementacja aplikacji webowej wspomagającej automatyczne układanie planu lekcji z wykorzystaniem algorytmu ewolucyjnego. Aplikacja ma umożliwiać generowanie planów uwzględniających:

- Ograniczenia fizyczne (dostępność sal, unikanie kolizji zajęć)
- Ograniczenia jakościowe (ciągłość zajęć, brak okienek)
- Ograniczenia specyficzne definiowane przez użytkownika (bloki lekcyjne)

Algorytm ma minimalizować liczbę okienek w planach nauczycieli, redukując tym samym czas ich przebywania w szkole.

Dla osiągnięcia postawionego celu konieczne jest wykonanie następujących zadań:

- Zaprojektowanie i implementacja bazy danych do przechowywania ograniczeń, specyfikacji bloków lekcyjnych oraz wyników generowania planów.
- Zaprojektowanie i wykonanie interfejsu użytkownika umożliwiającego definiowanie wszystkich niezbędnych ograniczeń oraz wizualizację końcowych planów lekcji.
- Opracowanie algorytmu do układania planu lekcji wykorzystującego:
  - Algorytm zachłanny do tworzenia struktury bloków lekcyjnych.
  - Algorytm ewolucyjny do optymalizacji rozkładu godzinowego.
  - Solver programowania ograniczeń do szczegółowego harmonogramowania.
- Integracja aplikacji webowej z algorytmem.

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.

1.2. 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

Struktura rozdziału?

2.1. Inspiracja podejściem iteracyjnym

Praca „A mixed-integer programming approach for solving university course timetabling problems" [12] przedstawia innowacyjne podejście do rozwiązywania problemów harmonogramowania poprzez dekompozycję na prostsze podproblemy i iteracyjne dodawanie ograniczeń. Autorzy wykorzystują strategię polegającą na:

1. Uzyskaniu rozwiązania początkowego z podstawowym zestawem ograniczeń
2. Iteracyjnym „wstrzykiwaniu" kolejnych ograniczeń z wykorzystaniem zmiennych sztucznych
3. Stosowaniu heurystyk redukcji zmiennych dla kontroli złożoności obliczeniowej
4. Utrzymywaniu wykonalności rozwiązania na każdym etapie

2.1.1. Adaptacja

W mojej pracy zaadaptowałem i rozwinąłem tę fundamentalną ideę, dostosowując ją do specyfiki problemu układania planu lekcji.

Dekompozycja problemu

Podobnie jak w pracy źródłowej, gdzie autorzy rozbijają problem na warstwy ograniczeń, w moim rozwiązaniu zastosowałem trójetapową dekompozycję:

$$\text{Problem pełny} \longrightarrow \text{Etap 1} \longrightarrow \text{Etap 2} \longrightarrow \text{Etap 3}$$

gdzie każdy etap rozwiązuje wyraźnie wydzielony podproblem o mniejszej złożoności.

Rozwiązanie początkowe z prostymi ograniczeniami

Analogicznie do podejścia z tej pracy, gdzie zaczynają od podstawowych ograniczeń równościowych, w moim algorytmie:

- **Etap 1 (algorytm zachłanny)**: Koncentruje się wyłącznie na ograniczeniach głównych i strukturalnych bloków lekcyjnych (3.1):

  $$\mathcal{L} = \text{Greedy}(\mathcal{W}, \mathcal{B})$$

  pomijając tymczasowo ograniczenia czasowe i przestrzenne.

- **Etap 2 (algorytm ewolucyjny)**: Dodaje ograniczenia dostępności nauczycieli i równomierności rozkładu:
$$S_{\text{best}} = \text{Evolutionary}(\mathcal{L}, V, A)$$
ale wciąż operuje na poziomie dni, a nie konkretnych slotów czasowych.

Hierarchiczna dekompozycja problemu

Podczas gdy w pracy [12] dekompozycja dotyczy głównie warstw ograniczeń, w moim rozwiązaniu wprowadziłem dekompozycję całego problemu:

Poziom 1: Struktura bloków lekcyjnych $\mathcal{L}$
Poziom 2: Rozkład tygodniowy S_{best}
Poziom 3: Przypisania szczegółowe $\mathcal{Z}$

Każdy poziom rozwiązuje inny problem i wykorzystuje odpowiednie do tego celu algorytmy.

Kombinacja metod optymalizacyjnych

W odróżnieniu od pracy źródłowej, która wykorzystuje wyłącznie solver MIP z heurystykami, w moim podejściu zastosowałem wiele technik:

- **Algorytm zachłanny** — dla problemów konstrukcyjnych (tworzenie bloków)
- **Algorytm ewolucyjny** — dla dyskretnego przydziału bloków do dni
- **Solver CP** — dla problemów szeregowania

2.1.2. Korzyści z podejścia iteracyjnego

Redukcja złożoności obliczeniowej

Pełny problem harmonogramowania wymagałby liczby zmiennych decyzyjnych proporcjonalnych do:
$$|\mathcal{T}| \times |\mathcal{C}| \times |\mathcal{S}| \times 5 \times H \times |\mathcal{R}|$$
co dla typowej szkoły daje setki milionów potencjalnych zmiennych.

W podejściu dekompozycyjnym:

- **Etap 1**: Operuje na $|\mathcal{W}|$ wymaganiach
- **Etap 2**: Operuje na $5 \times |\mathcal{L}|$ zmiennych
- **Etap 3**: Rozwiązuje 5 niezależnych podproblemów o rozmiarze 20% oryginalnego problemu

Każdy etap można modyfikować lub wymieniać niezależnie. Jest to ważna zaleta podczas rozwijania oprogramowania. W szczególności jeśli takie oprogramowanie może być w przyszłości rozwijane przez więcej niż jednego dewelopera.

Podejście dekompozycyjne zaprezentowane zarówno w pracy [12], jak i w mojej pracy, demonstruje fundamentalną zasadę inżynierii oprogramowania: „dziel i zwyciężaj”.

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 [15].

Posiadamy zbiór maszyn:

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

oraz zbiór zadań (prac):

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

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

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

które muszą zostać wykonane w ściśle określonej kolejności:

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

Każda operacja O_{ij} jest przypisana do dokładnie jednej maszyny 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 (ang. *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:

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

Ograniczenia precedencji:

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

Dla każdej pary różnych operacji (O_{ij}, O_{kl}) przypisanych do tej samej maszyny zachodzi jedno z dwóch:

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

Funkcja celu:

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

2.2.1. Podobieństwa do problemu układania planu lekcji

Problem harmonogramowania dla szkół można rozpatrywać jako specyficzną odmianę JSSP z wieloma zasobami. Podstawowe podobieństwa strukturalne obejmują:

Struktura zasobów

W JSSP mamy zbiór maszyn M, natomiast w problemie planu lekcji dysponujemy trzema typami zasobów:

$$\{\mathcal{T}, \mathcal{C}, \mathcal{R}\}$$

gdzie:

- $\mathcal{T}$ — zbiór nauczycieli (maszyny typu „nauczyciel")
- $\mathcal{C}$ — zbiór klas (maszyny typu „klasa)"
- $\mathcal{R}$ — zbiór sal (maszyny typu „sala")

Struktura zadań

W klasycznym JSSP zadania J_i składają się z operacji O_{ij}. W problemie planu lekcji:

$$\text{Lekcja } z_i = (d_i, h_i, c_i, t_i, s_i, r_i)$$

można traktować jako operację wymagającą jednoczesnego dostępu do trzech zasobów: nauczyciela, klasy i sali. gdzie d_i to dzień lekcji, h_i to godzina rozpoczęcia lekcji, c_i to klasa, t_i to nauczyciel, s_i to przedmiot, a r_i to sala.

Ograniczenia nakładania się

Analogia do ograniczenia w JSSP:

$$\forall \text{pary lekcji } (z_i, z_j) \text{ współdzielących zasób } \mathcal{R}:$$

$$h_i \neq h_j \ \vee \ d_i \neq d_j \ \vee \ r_i \neq r_j$$

co odpowiada warunkowi w JSSP:

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

Analogiczne ograniczenia należałoby definiować dla pozostałych zasobów $\mathcal{T}$ i $\mathcal{C}$.

Reprezentacja interwałowa

Kluczowym podobieństwem jest wykorzystanie zmiennych interwałowych. W JSSP operację O_{ij} reprezentujemy jako interwał:

$$I_{ij} = (S_{ij}, p_{ij}, C_{ij})$$

gdzie $C_{ij} = S_{ij} + p_{ij}$.

W problemie planu lekcji, lekcję z_i reprezentujemy jako:

$$I_i = (h_i, v_i, h_i + v_i)$$

gdzie v_i to liczba godzin (czas trwania), a h_i to slot rozpoczęcia.

2.2.2. Różnice i specyfika problemu szkolnego

Pomimo strukturalnych podobieństw, problem układania planu lekcji wprowadza istotne modyfikacje:

Wielozasobowość

Podczas gdy w klasycznym JSSP każda operacja wymaga jednej maszyny, lekcja wymaga **jednoczesnego** dostępu do trzech zasobów:

$$\text{Lekcja } z_i \text{ wymaga: } (t_i, c_i, r_i) \in \mathcal{T} \times \mathcal{C} \times \mathcal{R}$$

Ograniczenia miękkie i jakościowe

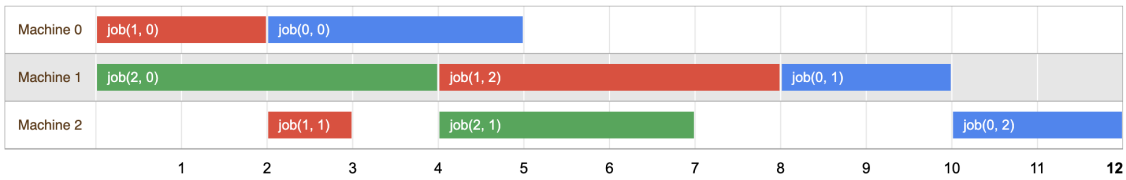
Problem szkolny wprowadza bogaty zestaw ograniczeń jakościowych niewystępujących w klasycznym JSSP:

- Brak okienek dla uczniów (ciągłość zajęć)
- Równomierny rozkład godzin w tygodniu
- Ograniczenia prawne (max 2 godziny tego samego przedmiotu dziennie)
- Dostępność nauczycieli

Innowacją w podejściu jest wprowadzenie bloków lekcyjnych $\mathcal{L}$, które grupują lekcje z_i odbywające się równocześnie, co stanowi rozszerzenie klasycznej koncepcji operacji w JSSP. Umożliwia to znacznie łatwiejsze definiowanie ograniczeń.

2.2.3. Gotowe rozwiązania

Istnieje wiele istniejących rozwiązań JSSP, jednak szczególnie inspirujące okazało się podejście zaprezentowane w dokumentacji Google OR-Tools [10], które wykorzystuje zmienne interwałowe do modelowania operacji.



Rysunek 2.1: Wizualizacja rozwiązania klasycznego problemu JSSP z wykorzystaniem zmiennych interwałowych [10]

Wyraźnie widać te inspiracje w sekcji trzeciej algorytmu dotyczącej solvera CP (3.6), gdzie zastosowano:

- Reprezentację lekcji jako interwałów czasowych
- Ograniczenia NoOverlap dla zasobów (nauczyciele, klasy, sale)

- Optional intervals dla przypisań sal
- Specjalistyczne ograniczenia ciągłości czasowej

Kluczowe koncepcje zaadaptowane w mojej pracy:

Zmienne interwałowe

- W JSSP: IntervalVar($start, duration, end$).
- W planie lekcji: IntervalVar($h_i, v_i, h_i + v_i$), gdzie h_i to slot czasowy (3.1), a v_i to długość trwania lekcji.

Ograniczenie NoOverlap

- W JSSP: AddNoOverlap([$\texttt{interval}_1, \ldots, \texttt{interval}_n$]) dla operacji na tej samej maszynie [8].
- W planie lekcji: AddNoOverlap zastosowane dla:
  - Wszystkich lekcji tej samej klasy
  - Wszystkich lekcji tego samego nauczyciela
  - Wszystkich lekcji w tej samej sali

Optional Intervals

Rozszerzenie o interwały opcjonalne pozwala na modelowanie przypisań sal do nauczycieli w blokach lekcyjnych:

$$\text{OptionalIntervalVar}(start, duration, end, presence)$$

Takie interwały są brane pod uwagę w ograniczeniach tylko i wyłącznie, jeśli $presence = 1$; Oznaczna to, że interwał jest „aktywny”.

2.3. Istniejące kompleksowe rozwiązania

Zadanie układania planu lekcji jest powszechnym wyzwaniem dla placówek edukacyjnych na całym świecie, w tym Polsce, co zaowocowało rozwojem komercyjnych rozwiązań tego problemu. Na polskim rynku dominuje kilka systemów, wśród których szczególną pozycję zajmuje firma Vulcan, oferująca zintegrowany system zarządzania oświatą. Obok niej funkcjonują takie rozwiązania jak Dobry Plan, czy Librus.

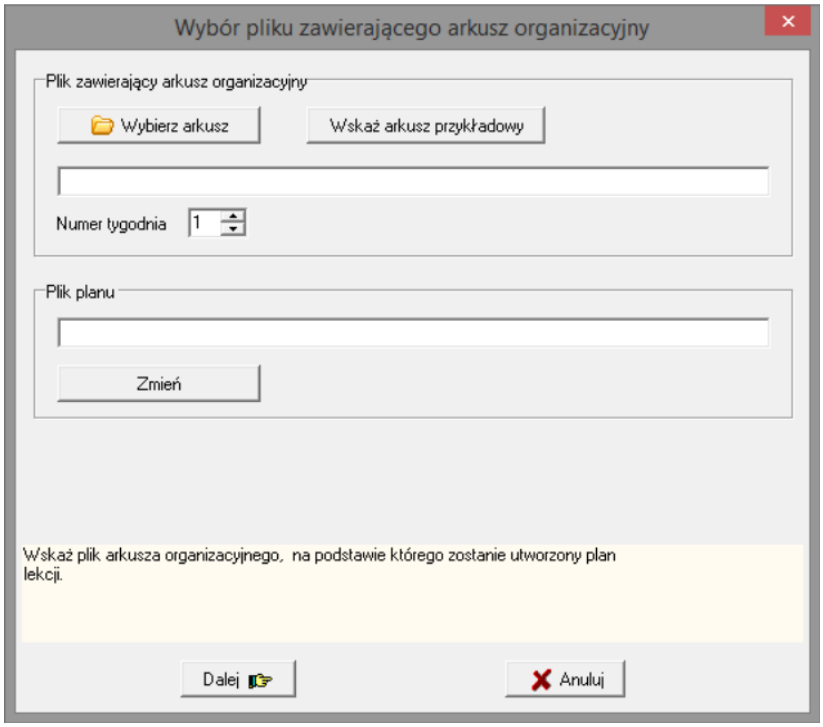
2.3.1. Plan lekcji Optivum

Vulcan, jako jeden z najstarszych na rynku dostawców, opracował kompleksowe rozwiązanie obejmujące nie tylko układanie planu lekcji, ale także dziennik elektroniczny, sekretariat i inne moduły zarządzania szkołą. Jego popularność w polskich szkołach wynika z lat doświadczenia w dostosowywaniu systemu do specyficznych wymagań polskiego systemu edukacji.

Aplikacja „Plan lekcji Optivum" firmy Vulcan to zaawansowane narzędzie wspomagające proces tworzenia szkolnego planu zajęć. Jego główną zaletą jest elastyczność w definiowaniu skomplikowanych ograniczeń, w tym szczegółowe zarządzanie podziałami uczniów na grupy.

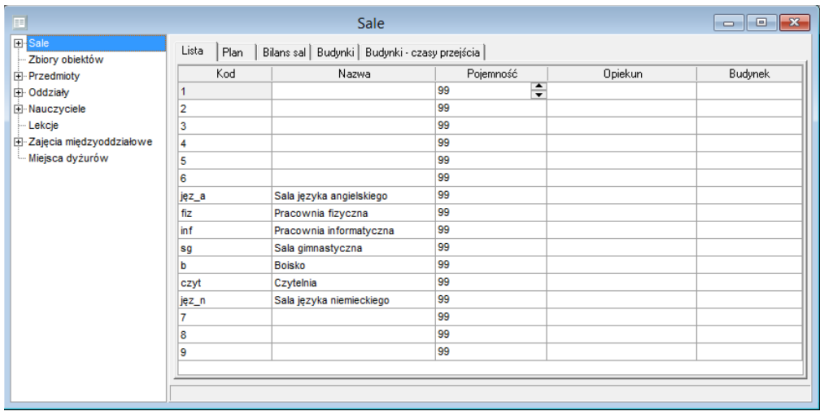
Proces rozpoczyna się od zaimportowania danych z arkusza organizacyjnego, a kończy na publikacji gotowego planu [14].

Podstawą do tworzenia planu są dane zaimportowane z arkusza organizacyjnego. Kluczowym wymaganiem, które zostało mi przedstawione przez liceum, które zaopatrzyło mnie w dane do tej pracy jest możliwość importowania podobnych plików w aplikacji. W ten sposób pozbywamy się żmudnego procesu ręcznego wprowadzania ograniczeń.



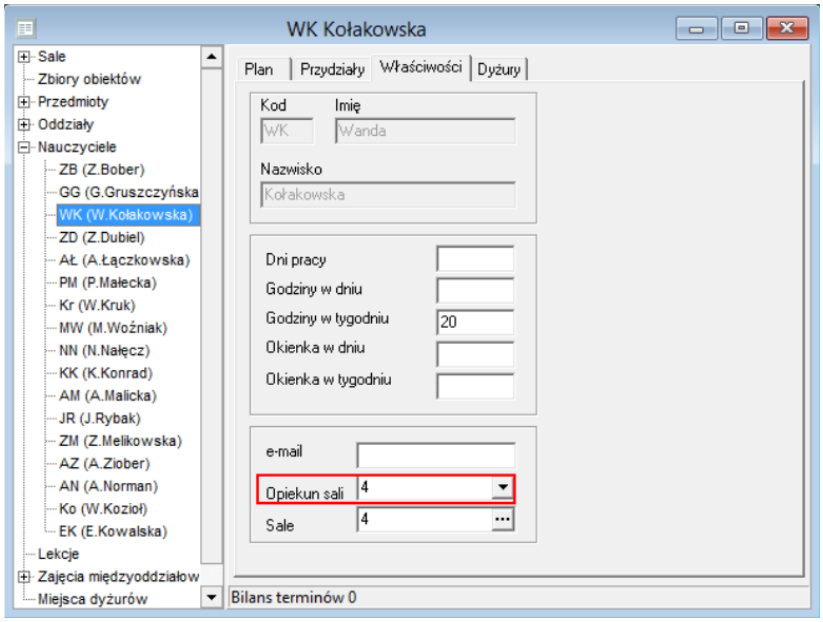
Rysunek 2.2: Zrzut ekranu importowania arkusza organizacyjnego do programu „Plan lekcji Optivum" [14]

W następnym etapie użytkownik definije zasoby lokalne zaczynając od sal oraz preferencji. Dla zajęć grupowych kluczowe jest poprawne zdefiniowanie sal. Jeśli kilka grup ma korzystać z jednego dużego pomieszczenia (sala gimnastyczna, pracownia), należy je podzielić na części (przykładowo: salagim1, salagim2) i traktować jako odrębne sale. Dla zajęć poza szkołą (basen) wykorzystuje się tzw. „salę pozorną".



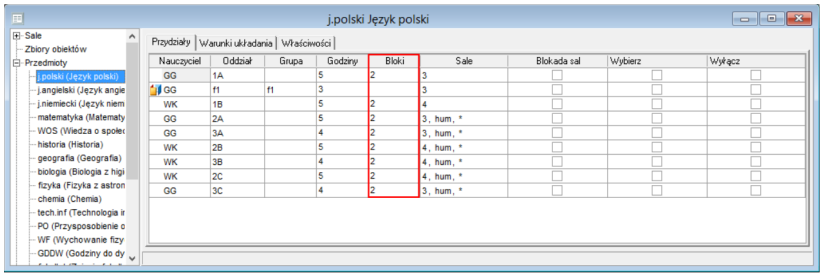
Rysunek 2.3: Zrzut ekranu wprowadzania sal do programu „Plan lekcji Optivum” [14]

Rysunek 2.4: Zrzut ekranu wprowadzania preferencji sal względem przedmiotów do programu „Plan lekcji Optivum” [14]



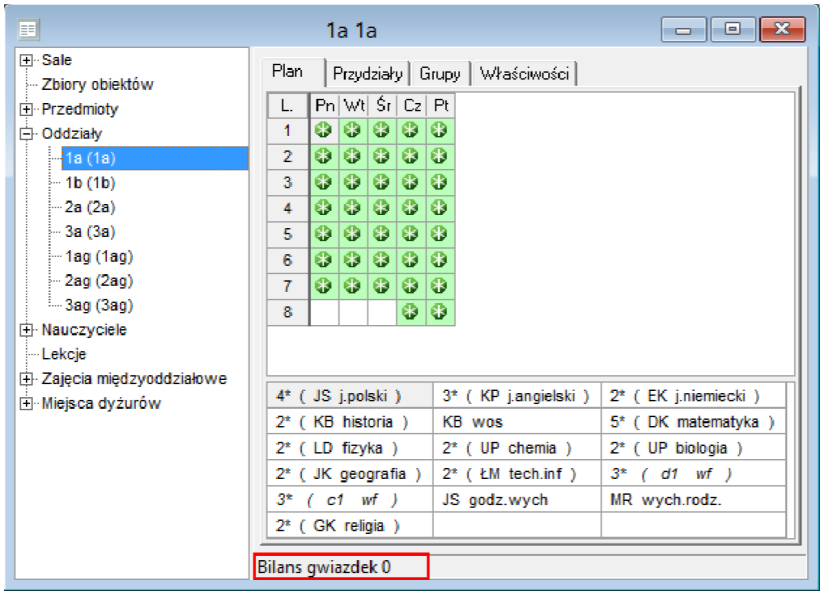
Rysunek 2.5: Zrzut ekranu wprowadzania preferencji sal względem nauczycieli do programu „Plan lekcji Optivum” [14]

Po wprowadzeniu sal oraz ich preferencji użytkownik jest poproszony o wprowadzenie ewentualnych podziałów na bloki. Decyzja o rozkładzie godzin w tygodniu ma bezpośredni wpływ na grupy. Dla przydziału 4-godzinnego WF-u, podział na bloki 2,1,1 oznacza, że jedna z lekcji (przykładowo dla dziewcząt) będzie dwugodzinnym blokiem, a pozostałe pojedynczymi.



Rysunek 2.6: Zrzut ekranu wprowadzania bloków przedmiotów do programu „Plan lekcji Optivum” [14]

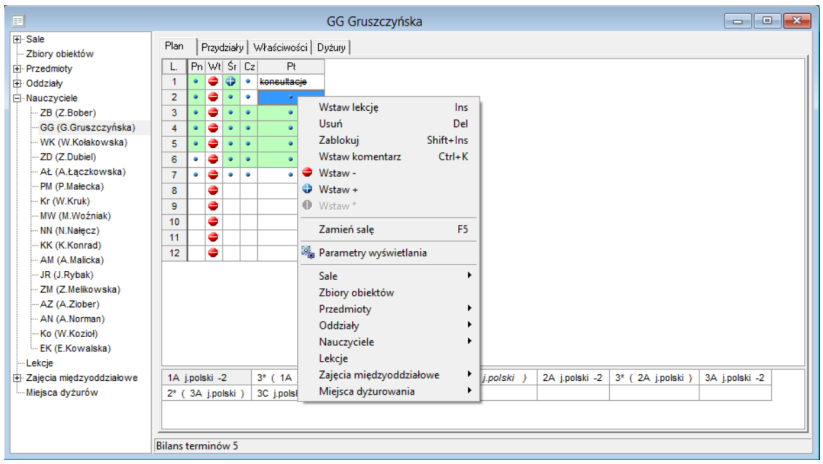
W kolejnym etapie wprowadzane są terminy odbycia zajęć w poszczególnych klasach. Program na bieżąco wylicza „Bilans gwiazdek” — różnicę między liczbą gwiazdek a minimalną liczbą potrzebną do ułożenia planu. Dla oddziałów z podziałami na grupy, prawidłowe rozmieszczenie gwiazdek jest kluczowe, aby uniknąć okienek lub niemożności ułożenia planu.



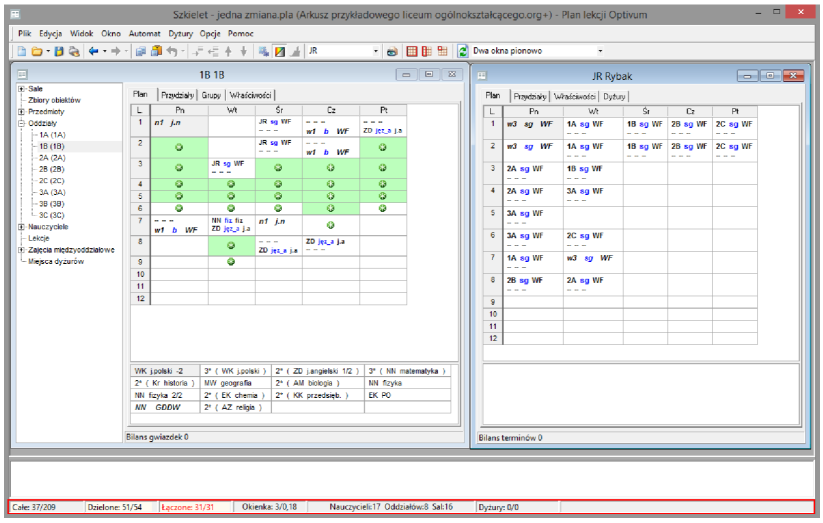
Rysunek 2.7: Zrzut ekranu wprowadzania terminów zajęć dla poszczególnych klas do programu „Plan lekcji Optivum” [14]

Następnie definiuje się dostępność nauczycieli. Wybrane terminy nauczyciela można zablokować lub wskazać jako szczególnie pożądane poprzez odpowiednio symbole $\ominus$ oraz $\oplus$.

Przed automatycznym ułożeniem całego planu, zaleca się ręczne lub automatyczne umieszczenie lekcji uznanych za najtrudniejsze, do których należą zajęcia dzielone na grupy i międzyoddziałowe.



Rysunek 2.8: Zrzut ekranu wprowadzania dostępności nauczycieli do programu „Plan lekcji Optivum” [14]



Rysunek 2.9: Zrzut ekranu wprowadzania „trudnych” lekcji do programu „Plan lekcji Optivum” [14]

Po ułożeniu „trudnych” lekcji uruchamia się automat dla całego planu. Jeśli automat nie poradzi sobie z ułożeniem wszystkich lekcji (przykładowo z powodu zbyt restrykcyjnych warunków dla grup), należy analizować nieułożone lekcje i łagodzić parametry. Czasami pomaga kilkakrotne wykonanie minimalizacji okienek i układania całego planu.

Narzędzie „Plan lekcji Optivum” jest bardzo obszernym narzędziem oferującym wiele możliwości. Bierze pod uwagę praktycznie każdy możliwy scenariusz, który może wystąpić w polskiej szkole, co jest rezultatem wieloletniej obecności na rynku oraz doświadczenia deweloperów.

Aplikacja znakomicie ilustruje cenę uniwersalności, którą jest konieczność stworzenia rozbudowanej aplikacji wymagającej od użytkowników definiowania wielu ograniczeń, nawet tych rzadko spotykanych w przeciętnej szkole. Kolejnym kosztem takiego podejścia jest konieczność specjalistycznych szkoleń — Vulcan oferuje kosztowne szesnastogodzinne szkolenia

online poświęcone wyłącznie obsłudze aplikacji do układania planu lekcji.

Podsumowując, „Plan lekcji Optivum" to doskonałe narzędzie dla dużych placówek, które potrzebują sprawdzonego i kompleksowego rozwiązania. Niemniej jednak dla małych i średnich szkół, które nie dysponują odpowiednimi funduszami ani czasem na obsługę tak rozbudowanego systemu, może okazać się zbyt skomplikowane i kosztowne.

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.

Słownik oznaczeń:

- $\mathcal{C}$ — zbiór klas
- $\mathcal{T}$ — zbiór nauczycieli
- $\mathcal{S}$ — zbiór przedmiotów
- $\mathcal{R}$ — zbiór sal
- $\vec{R_i}$ — zbiór przedmiotów obsługiwanych przez i-tą salę

- $\mathcal{W}$ — zbiór wymagań głównych
- $\mathcal{L}$ — zbiór bloków lekcyjnych
- $\mathcal{B}$ — zbiór bloków przedmiotów
- H — liczba slotów czasowych w dniu
- V — wektor godzin bloków
- S — macierz przydziału do dni, kodowanie osobnika
- $\mathfrak{P}$ — wielkość populacji w algorytmie ewolucyjnym

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 powinien być zbior przypisań $\mathcal{Z}$. Każde takie przypisanie powinno mieć 6 wartości:

$$z_i \triangleq (d_{z_i}, h_{z_i}, c_{z_i}, t_{z_i}, s_{z_i}, r_{z_i}) \in \mathcal{Z}_d, \quad \forall i \in \{1, 2, \ldots, |\mathcal{Z}_d|\} : \begin{cases} d_{z_i} & \in \{1,2,3,4,5\} \\ h_{z_i} & \in \{1,2,\ldots,H\} \\ c_{z_i} & \in \mathcal{C} \\ t_{z_i} & \in \mathcal{T} \\ s_{z_i} & \in \mathcal{S} \\ r_{z_i} & \in \mathcal{R} \end{cases}$$

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

⋮

- 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** $\mathcal{L}$ z ograniczeń głównych $\mathcal{W}$ na podstawie $\mathcal{B}$:
   - 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.**
   - Tworzenie macierzy S.
   - Przykładowo: dwie godziny matematyki w poniedziałek, godzina matematyki w środę, dwie godziny matematyki w czwartek i zero w piątek.
3. **Tworzenie samych przypisań $\mathcal{Z}$ dla każdego dnia tygodnia osobno.**

3.1.1. Dane i szukane

Wymagania główne

Warto zacząć od przedstawienia sposobu reprezentacji zmiennych w kodzie zaczynając od wymagań głównych. Ilość godzin tygodniowo odbytych przez klasę z danym nauczycielem w ramach danego przedmiotu jest z góry ustalona. Aby łatwiej zrozumieć na czym polegają takie przypisania warto spojrzeć na dotychczasowy sposób przypisywania ilości godzin nauczycieli do klas (Rysunek 3.1) w liceum, które dostarczyło dane na potrzeby 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

Rysunek 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.

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

$$w_i \triangleq (t_{w_i}, c_{w_i}, s_{w_i}, g_{w_i}) \in \mathcal{W}, \quad \forall i \in \{1, 2, \ldots, |\mathcal{W}|\} : \begin{cases} t_{w_i} & \in \mathcal{T} \\ c_{w_i} & \in \mathcal{C} \\ s_{w_i} & \in \mathcal{S} \\ g_{w_i} & \in \mathbb{N}^+ \end{cases} \tag{3.1}$$

gdzie g_{w_i} to liczba wymaganych tygodniowo godzin przedmiotu s_{w_i} przeprowadzonego przez nauczyciela t_{w_i} dla klasy c_{w_i}.

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 niedostepny.

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

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

$$b_i \triangleq \left(\vec{S_{b_i}}, \vec{C_{b_i}}, \vec{G_{b_i}}, \texttt{is\_aggregating}_{b_i}, \texttt{max\_weekly}_{b_i}\right) \in \mathcal{B}$$

$$\forall i \in \{1, 2, \ldots, |\mathcal{B}|\} : \begin{cases} \vec{S_{b_i}} & \subset \mathcal{S} \\ \vec{C_{b_i}} & \subseteq \mathcal{C} \\ \vec{G_{b_i}} & \in \mathbb{N}^{|\vec{S_b}|} \\ \texttt{is\_aggregating}_{b_i} & \in \{0, 1\} \\ \texttt{max\_weekly}_{b_i} & \in \mathbb{N} \end{cases}$$

gdzie

- $\vec{S_{b_i}}$ to podzbiór wszystkich przedmitów, których dotyczy blok b_i,
- $\vec{C_{b_i}}$ to podzbiór wszystkich klas, których dotyczy blok b_i,
- $\vec{G_{b_i}}$ to wektor oczekiwanej liczby przedmiotów w bloku.
- `is_aggregating` to wartość binarna informująca o tym czy blok jest *agregujący*.
- `max_weekly` to liczba naturalna która informuje o maksymalnej liczbie bloków. Jeśli `max_weekly` $= 0$ to oznacza to, że liczba bloków jest nielimitowana.

Przykładowy blok przedmiotów:

- $\vec{S_b}$ = *(język angielski, informatyka)*,
- $\vec{C_b}$ = (IA, IB, IC),

- $\vec{G_b} = (1, 1)$,
- `is_aggregating` $= 0$, *(To nie jest blok argumentujący)*
- `max_weekly` $= 0$, *nielimitowana tygdniowo liczba godzin*

lub:

- $\vec{S_b}$ = *(język angielski)*,
- $\vec{C_b} = \mathcal{C}$
- $\vec{G_b} = (2)$,
- `is_aggregating` $= 0$
- `max_weekly` $= 0$

3.1.2. Ograniczenia

Wcześniej wspomniane ograniczenia można podzielić na 4 kategorie:

Ograniczenia fizyczne

- Żaden nauczyciel nie może być w 2 miejscach na raz.

$$\forall i, j \in \{1, 2, \ldots, |\mathcal{Z}|\} : t_{z_i} \neq t_{z_j} \implies \left(d_{z_i} = d_{z_j} \wedge h_{z_i} = h_{z_j} \wedge r_{z_i} \neq r_{z_j} \wedge i \neq j\right)$$

- Nauczyciel musi być dostępny.

$$\forall i \in \{1, 2, \ldots, |\mathcal{Z}|\} : a_{t_{z_i}, d_{z_i}} = 1$$

  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.

$$\forall i, j \in \{1, 2, \ldots, |\mathcal{Z}|\}, \exists k \in \{1, 2, \ldots |\mathcal{B}|\} :$$
$$\left(d_{z_i} = d_{z_j} \wedge h_{z_i} = h_{z_j} \wedge c_{z_i} = c_{z_j} \wedge i \neq j\right) \implies c_{z_i} \in \vec{C_{b_k}} \wedge s_{z_i}, s_{z_j} \in \vec{S_{b_k}}$$

- W żadnej sali nie mogą odbywać się 2 lekcje na raz.

$$\forall i, j \in \{1, 2, \ldots, |\mathcal{Z}|\}, \exists k \in \{1, 2, \ldots |\mathcal{B}|\} :$$
$$\left(d_{z_i} = d_{z_j} \wedge h_{z_i} = h_{z_j} \wedge r_{z_i} = r_{z_j} \wedge i \neq j\right) \implies s_{z_i} = s_{z_j} \wedge s_{z_i}, s_{z_j} \in \vec{S_{b_k}}$$

Ograniczenia prawne

[1, 2, 3]

- Uczeń nie może mieć więcej niż 2 godzin lekcyjnych tego samego przedmiotu dziennie.

$$\forall d \in \{1,2,3,4,5\}, c \in \mathcal{C}, s \in \mathcal{S} : |\{z \in \mathcal{Z} : d_z = d \wedge c_z = c \wedge s_z = s\}| \leq 2$$

- Jeśli danego dnia mają zostać przeprowadzone 2 godziny jednego przedmiotu, to uczeń musi je mieć bezpośrednio po sobie.

$$\forall i, j \in \{1,2,\ldots,|\mathcal{Z}|\}, c \in \mathcal{C}, s \in \mathcal{S} :$$
$$\left(d_{z_i} = d_{z_j} \wedge c_{z_i} = c_{z_j} \wedge s_{z_i} = s_{z_j} \wedge i \neq j\right) \implies \left|h_{z_i} - h_{z_j}\right| = 1$$

Ograniczenia jakościowe

- Brak okienek dla uczniów. Nie istnieje taka para przypisań, dla których różnica między slotami czasowymi jest większa niż suma wszystkich lekcji danego dnia.

$$\nexists i, j \in \{1,2,\ldots,|\mathcal{Z}|\}, c \in \mathcal{C} :$$
$$d_{z_i} = d_{z_j} \wedge c_{z_i} = c_{z_j} \wedge \left|h_{z_i} - h_{z_j}\right| > |\{z_k : d_{z_k} = d_{z_i} \wedge c_{z_i}\}|$$

- Odpowiednie przypisanie sal. Lekcje wychowania fizycznego muszą odbyć się w przeznaczonych do tego salach, podobnie lekcji informatyki itd.

$$\forall z \in \mathcal{Z} : s_z \in \vec{R_{r_z}}$$

- Dla bloków lekcyjnych, które odbywają się 3 razy w tygodniu konieczne jest stworzenie jednego bloku dwugodzinnego.

$$\forall w \in \mathcal{W}, \exists d \in \{1,2,3,4,5\} :$$
$$g_w = 3 \implies |\{z \in \mathcal{Z} : c_w = c_z \wedge s_w = s_z \wedge t_w = t_z \wedge d_z = d\}| = 2$$

Ograniczenie główne

Zgodnie z definicją przyjętą w 3.1 tygodniowa liczba godzin każdego przedmiotu musi być równa tej ustalonej w zbiorze wymagań:

$$\forall w \in \mathcal{W} : |\{z \in \mathcal{Z} : c_w = c_z \wedge t_w = t_z \wedge s_w = s_z\}| = g_w$$

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 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:

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

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

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

- Ograniczenie prowadzenia maksymalnie jednej lekcji dla wszystkich nauczycieli:

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

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

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

- Ograniczenie maksymalnie jednej lekcji w pokoju:

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

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 [13]. W przyjętym modelu problem został przedstawiony w postaci grafu, gdzie węzły reprezentowały wymagania główne (**??**), 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. Struktura algorytmu i wybór technologii

W świetle przeprowadzonych eksperymentów i poprzednich prób rozwiazania 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 przedmitów 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 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.

3.4.2. Działanie

Dane wejściowe

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

Klasyfikacja bloków

- **Bloki agregujące** — tworzą wyższy poziom hierarchii, łącząc istniejące bloki i wymagania w jeszcze większe bloki

$$\mathcal{B}_{\text{agg}} = \{b \in \mathcal{B} : \texttt{is\_aggregating}_b = 1\}$$

- **Bloki wieloklasowe** — łączą wymagania z wielu klas dla tych samych przedmiotów

$$\mathcal{B}_{\text{multi}} = \left\{b \in \mathcal{B} : \left|\vec{S}_b\right| = 1\right\}$$

- **Bloki pojedyncze** — obejmują pojedyncze klasy

$$\mathcal{B}_{\text{single}} = \mathcal{B} \cap (\mathcal{B}_{\text{multi}} \cup \mathcal{B}_{\text{multi}})$$

Inicjalizacja

$$\begin{aligned} \mathcal{L} &\leftarrow \emptyset \quad \text{(zbiór bloków lekcyjnych)} \\ V &\leftarrow \emptyset \quad \text{(wektor godzin bloków)} \\ U : \mathcal{W} &\to \mathbb{N} \quad \text{(funkcja wykorzystania godzin wymagań)} \\ U(w) &\leftarrow 0 \quad \forall w \in \mathcal{W} \end{aligned}$$

Tworzenie grup wieloklasowych

Dla każdego bloku $b \in \mathcal{B}_{\text{multi}}$ tworzymy zbiór wymagań zgodnych z definicją bloku:

$$\begin{cases} \mathcal{W}_b = \left\{w \in \mathcal{W} : s_w \in \vec{S}_b \wedge c_w \in \vec{C}_b\right\} \\ |\{w \in \mathcal{W}_b : s_w = s\}| = g_b \qquad\qquad \forall s \in \vec{S}_b \end{cases}$$

Grupy te stanowią podstawę do tworzenia bloków lekcyjnych międzyklasowych.

Agregacja bloków

Dla każdego bloku $b \in \mathcal{B}_{\text{agg}}$:

1. Wybieramy maksymalnie duży zbiór wymagań $\mathcal{W}_{\text{agg}} \subseteq \mathcal{W}$ spełniający:

$$\forall w \in \mathcal{W}_{\text{agg}} : c_w \in \vec{C}_b \wedge s_w \in \vec{S}_b$$

   - Nauczyciele w $\mathcal{W}_{\text{agg}}$ są różni.
   - Wymagania nie były wcześniej wykorzystane.

   Robimy to poprzez łączenie odpowiednich bloków $\mathcal{W}_b$ dla $b \in \mathcal{B}_{\text{multi}}$.

2. Określamy liczbę godzin: $v = \min\{g_w - U(w) : w \in \mathcal{W}_{\text{agg}}\}$
3. Dodajemy do wyników: $\mathcal{L} \leftarrow \mathcal{L} \cup \{\mathcal{W}_{\text{agg}}\}$, $V \leftarrow V \cup \{v\}$
4. Aktualizujemy wykorzystanie: $U(w) \leftarrow U(w) + v$ dla $w \in \mathcal{W}_{\text{agg}}$

Przetwarzanie bloków wieloklasowych

Dla każdej utworzonej wcześniej grupy $\mathcal{W}_b$:

1. Jeśli wymagania nie zostały w pełni wykorzystane, tworzymy blok
2. Liczba godzin: $v = \min\{g_w - U(w) : w \in \mathcal{W}_b\}$
3. Dodajemy do wyników i aktualizujemy wykorzystanie

Generowanie bloków pojedynczych

Dla każdego bloku $b \in \mathcal{B}_{\text{single}}$ i klasy $c \in \vec{C}_b$:

1. Znajdujemy wymagania: $\mathcal{W}_c = \{w \in \mathcal{W} : c_w = c \wedge s_w \in \vec{S}_b\}$
2. Jeśli zbiór spełnia warunki bloku co do ilości przemiotów, tworzymy blok lekcyjny
3. W przeciwnym przypadku generujemy wszystkie poprawne kombinacje wymagań
4. Dla każdej poprawnej kombinacji $\mathcal{W}' \subseteq \mathcal{W}_c$ tworzymy blok

Iteracyjna alokacja pozostałych godzin

1. Dopóki istnieją nie w pełni wykorzystane wymagania w blokach pojedynczych:
   (a) Równomiernie zwiększamy liczbę godzin we wszystkich możliwych blokach. Równomiernie oznacza, że iteruję się przez wszystkie bloki i ziększam ich przypisanie o 1, gwarantując tym samym, że wszystkie zostaną użyte.
   (b) Respektujemy ograniczenia maksymalnej liczby bloków tygodniowo

Finalizacja

1. Dla każdego wymagania $w \in \mathcal{W}$ z niewykorzystanymi godzinami:
2. Tworzymy blok jednostkowy: $\mathcal{L} \leftarrow \mathcal{L} \cup \{\{w\}\}$
3. $V \leftarrow V \cup \{g_w - U(w)\}$

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 zbiorem wymagań głównych:
$$\mathcal{L} = \left\{L_1, L_2, \ldots, L_{|\mathcal{L}|}\right\}, \quad \forall i \in \{1, 2, \ldots, |\mathcal{L}|\} : L_i \subset \mathcal{W}$$
gdzie $|\mathcal{L}|$ 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.
$$V = \begin{bmatrix} v_1 & v_2 & \cdots & v_{|\mathcal{L}|} \end{bmatrix}, \quad \forall i \in \{1, 2, \ldots, |\mathcal{L}|\} : v_i \in \mathbb{N}^+$$

Przy czym gwarantowane jest spełnienie wymagania głównego:

$$\forall w \in \mathcal{W} : \sum_{i:w \in L_i} v_i = g_w$$

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 w pracy „Genetic algorithms” [11]. W odróżnieniu od podejść gradientowych lub zachłannych algorytm genetyczny eksploruje wiele konkurujących rozwiązań równolegle oraz uniknięcie 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 (3.2). 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 $5 \times |\mathcal{L}|$.

$$S = \begin{bmatrix} s_{1,1} & s_{1,2} & s_{1,3} & \cdots & s_{1,|\mathcal{L}|} \\ s_{2,1} & s_{2,2} & s_{2,3} & \cdots & s_{2,|\mathcal{L}|} \\ s_{3,1} & s_{3,2} & s_{3,3} & \cdots & s_{3,|\mathcal{L}|} \\ s_{4,1} & s_{4,2} & s_{4,3} & \cdots & s_{4,|\mathcal{L}|} \\ s_{5,1} & s_{5,2} & s_{5,3} & \cdots & s_{5,|\mathcal{L}|} \end{bmatrix}, \quad S \in \mathbb{N}^2 \tag{3.2}$$

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.
$$\forall i \in \{1, 2, \ldots, |\mathcal{L}|\} : \sum_{d=1}^{5} s_{d,i} = v_i$$
2. Maksymalna liczba godzin każdego bloku w pojedynczym dniu nie może być większa niż 2.
$$\forall i \in \{1, 2, \ldots, |\mathcal{L}|\}, \forall d \in \{1, 2, 3, 4, 5\} : s_{d,i} \leq 2$$
3. Dla bloków 3-godzinnych konieczne jest przedzielenie 2 godzin do jednego dnia.
$$\forall i \in \{1, 2, \ldots, |\mathcal{L}|\} : v_i = 3 \implies \exists d \in \{1, 2, 3, 4, 5\} \text{ takie, że } s_{d,i} = 2$$

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.
   $$\forall d \in \{1,2,3,4,5\} : a'_d = \min_{t \in T_i} a_{t,d}$$
   gdzie $T_i = \{t_{w_j} : w_j \in L_i\}$ to zbiór nauczycieli w bloku L_i.

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` [5], 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:

$$X = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_{|\mathcal{L}|} \end{bmatrix}$$

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:

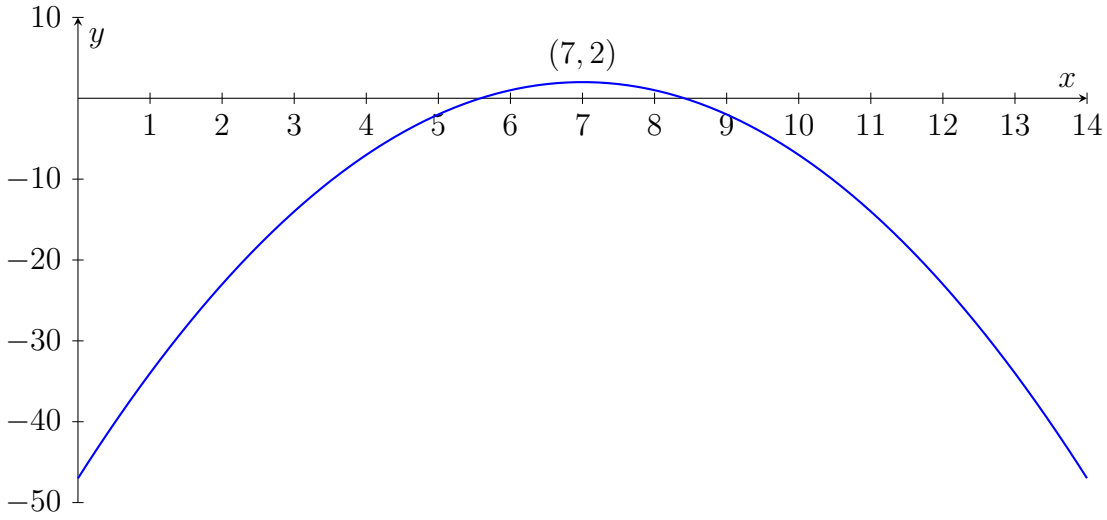
- Równomierność rozkładu godzin pracy nauczycieli w ciągu tygodnia
- Równomierność rozkładu godzin lekcyjnych klas w ciągu tygodnia
- Liczba dni, w których nauczyciele muszą pojawić się w szkole
- Różnica między najdłuższym i najkrótszym dniem lekcyjnym dla klas

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-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.



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:

$$X_i = \begin{bmatrix} x_{i,1} & x_{i,2} & x_{i,3} & x_{i,4} & x_{i,5} \end{bmatrix}$$

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

Problemem funkcji kwadradowej jest fakt, że osobniki otrzymują dużą karę, 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:

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

Następnie wyznaczam sumę wartości funkcji dla wszystkich argumentów tego wektora:

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

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:

$$X_i = \begin{bmatrix} x_{i,1} & x_{i,2} & x_{i,3} & x_{i,4} & x_{i,5} \end{bmatrix}$$

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 3.1.2, a więc modyfikacje funkcji nie są potrzebne.

Analogicznie do nauczycieli tworzymy kolejno wektory i wartości:

$$F_{c_{C_i}} = \sum_{d=1}^{5} f(x_{i,d}) \qquad F_{c_C} = \sum_{i=1}^{|\mathcal{C}|} F_{c_{C_i}}$$

gdzie $|\mathcal{C}|$ 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 przywyższa 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 $|\mathcal{C}|$. 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:

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

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

3.5.5. Selekcja

Zastosowałem traducyjną metodę ruletkową. Polega ona na losowaniu osobników, którzy posłużą jako rodzice z prawdopodobieństwami, które sa proporcjonalne do wartości ich funkcji przystowania. Mając wektor ocen $F_{c\ \text{total}} = \begin{bmatrix} F_{c_1} & F_{c_1} & \cdots & F_{c_\mathfrak{P}} \end{bmatrix}$, 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:

$$P = \begin{bmatrix} \sigma\left(F_{c_1}\right) & \sigma\left(F_{c_2}\right) & \cdots & \sigma\left(F_{c_\mathfrak{P}}\right) \end{bmatrix}, \quad \sigma(F_{c_i}) = \frac{\exp\left[F_{c_i}\right]}{\sum_{j=1}^{\mathfrak{P}} \exp\left[F_{c_j}\right]}$$

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 zagwarantowałoby 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: $S_{\text{child}} = \mathbf{0}_{5\times|\mathcal{L}|}$
2. Dla każdego nauczyciela $i \in \{1, 2, \ldots, |\mathcal{T}|\}$:
   (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: $S_{\text{child}} = \mathbf{0}_{5\times|\mathcal{L}|}$
2. Dla każdej klasy $i \in \{1, 2, \ldots, |\mathcal{C}|\}$:
   (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 niezmieniony w następnej generacji. Jest to osobik o największej wartości funkcji przystosowania:

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

3.5.8. Przykładowe rezultaty

Przykładowe macierze z opisami

3.6. Solver programowania liniowego z ograniczeniami

W trzecim etapie algorytmu wykorzystuję solver programowania liniowego z ograniczeniami (Constraint Programming — CP), konkretnie implementację CP-SAT z pakietu *Google OR-Tools*. W odróżnieniu od czystego programowania liniowego (MIP), solver CP operuje na szerszej klasie ograniczeń, w tym na zmiennych interwałowych, co czyni go idealnym narzędziem do problemów harmonogramowania. W odpowiedzi na wyzwania napotkane w podejściu opisanym w podrozdziale 3.2.1 zdecydowałem się na wykorzystanie zmiennych interwałowych opisanych w dokumentacji OR-Tools [9].

Zalety podejścia CP:

- **Naturalne modelowanie**: Zmienne interwałowe bezpośrednio reprezentują zajęcia o określonym czasie rozpoczęcia i zakończenia.
- **Efektywność pamięciowa**: W porównaniu z reprezentacją binarną z podrozdziału 3.2.1, podejście interwałowe redukuje liczbę zmiennych decyzyjnych.
- **Specjalizowane ograniczenia**: Solver oferuje gotowe implementacje ograniczeń typowych dla harmonogramów — nakładanie się.

Proces jest wykonywany niezależnie dla każdego dnia tygodnia $d \in \{1, 2, 3, 4, 5\}$, reprezentowanego przez wiersze macierzy S_{best}. Dla każdego dnia rozwiązujemy podproblem zawierający tylko te bloki L_i, dla których $S_{\text{best}_{d,i}} > 0$.
Takie podejście:

- Redukuje wymagania pamięciowe o ≈ 5.
- Umożliwia równoległe przetwarzanie dni, jeśli nie ogranicza nas pamięć.
- Upraszcza strukturę ograniczeń.

3.6.1. Reprezentacja interwałów

Dla modelowania zajęć używamy zmiennych interwałowych, które reprezentują bloki czasowe o określonej długości:

- **Interwał podstawowy** dla bloku lekcyjnego $L_i \in \mathcal{L}$:

$$\begin{cases} s_i & \in [0, H] \\ d_i & = v_i \\ e_i & \in [0, H] \\ e_i & = s_i + d_i \end{cases}$$

gdzie s_i to czas rozpoczęcia, e_i to czas zakończenia, a d_i to czas trwania.

- **Interwał opcjonalny** dla przypisania sali:

$$\begin{cases} p_{i,j,r} & \in \{0,1\} \\ p_{i,j,r} & \implies e_i = s_i + d_i \end{cases}$$

3.6.2. Zmienne decyzyjne

Interwały bloków lekcyjnych

Dla każdego bloku $L_i \in \mathcal{L}$ z liczbą godzin $v_i > 0$ definiujemy:

$$I_i = (s_i, v_i, e_i)$$

Przypisania sal

Dla każdej możliwej kombinacji (blok, nauczyciel, sala) definiujemy interwał opcjonalny:

$$O_{L_i,t,r} = (s_i, v_i, e_i, p_{L_i,t,r}), \quad \begin{cases} L_i & \in \mathcal{L} \\ t & \in \mathcal{T} \\ r & \in \mathcal{R} \end{cases}$$

gdzie brane pod uwagę są tylko mozliwe kombinacje. Przykładowo blok lekcyjny z jednym wymaganiem głównym matematyki, nauczyciel matematyki oraz sala do matematyki.

3.6.3. Ograniczenia

Ograniczenie przypisania sal

Dla każdego bloku L_i musi być przypisana odpowiednia liczba sal:

$$\forall L_i \in \mathcal{L} : \sum_{r \in \mathcal{R}, t \in \mathcal{T}} p_{L_i,t,r} = |T_i|$$

Ciągłość zajęć dla klas

Dla każdej klasy $c \in \mathcal{C}$ i dnia $d \in \{1, 2, \ldots, 5\}$:

$$\mathcal{L}_{c,d} = \{L_i \in \mathcal{L} : \exists w \in L_i \text{ takie, że } c_w = c \wedge s_{\text{best}_{d,i}} > 0\}$$

Nienakładanie się zajęć [8]**:**

$$\text{AddNoOverlap}(\{I_i : L_i \in \mathcal{L}_{c,d}\})$$

Brak okienek:

$$\begin{cases} \text{start}_c = \min\{s_i : L_i \in \mathcal{L}_{c,d}\} \\ \text{end}_c = \max\{e_i : L_i \in \mathcal{L}_{c,d}\} \\ \text{end}_c - \text{start}_c = \sum_{L_i \in \mathcal{L}_{c,d}} v_i \end{cases}$$

Nakładanie się nauczycieli

Dla każdego nauczyciela $t \in \mathcal{T}$ i dnia $d \in \{1, 2, \ldots, 5\}$:

$$\mathcal{L}_{t,d} = \{L_i \in \mathcal{L} : t \in T_i \wedge s_{\text{best}_{d,i}} > 0\}$$

$$\text{AddNoOverlap}(\{I_i : L_i \in \mathcal{L}_{t,d}\})$$

Kolizje sal

Dla każdej sali $r \in \mathcal{R}$ i dnia $d \in \{1, 2, \ldots, 5\}$:

$$\mathcal{O}_{r,d} = \{O_{L_i,t,r} : L_i \in \mathcal{L}, t \in \mathcal{T}, s_{\text{best}_{d,i}} > 0\}$$

$$\text{AddNoOverlap}(\mathcal{O}_{r,d})$$

Sekwencjonowanie przedmiotów

Ograniczenie dotyczące dwuch takich samych przedmiotów jednego dnia jest łatwo rozwiązywane poprzez zastosowanie interwałów, gdyż z definicji mamy zagwarantowane, że:

$$\forall i \in \{1, 2, \ldots, |\mathcal{L}|\} : v_i = 2 \implies e_i - s_i = 2$$

W ten sposób jeśli do danego dnia są przypisane dwie godziny i-tego bloku, to mamy gwarancję, że następują one po sobie.

3.6.4. Funkcja celu

Minimalizujemy całkowity czas przebywania nauczycieli w szkole:

$$\forall t \in \mathcal{T}, d \in \{1, 2, \ldots, 5\} : \begin{cases} \text{start}_{t,d} & = \min\{s_i : L_i \in \mathcal{L}_{t,d}\} \\ \text{end}_{t,d} & = \max\{e_i : L_i \in \mathcal{L}_{t,d}\} \\ \text{duration}_{t,d} & = \text{end}_{t,d} - \text{start}_{t,d} \end{cases}$$

$$F_c = \sum_{t \in \mathcal{T}} \sum_{d=1}^{5} \text{duration}_{t,d}$$

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

3.6.5. Transformacja interwałów na przypisania końcowe

Po znalezieniu rozwiązania przez solver CP-SAT, konieczne jest przekształcenie zmiennych interwałowych na ostateczne przypisania w zbiorze $\mathcal{Z}$. Proces ten obejmuje ekstrakcję wartości zmiennych decyzyjnych i mapowanie na strukturę danych planu lekcji.

Ekstrakcja rozwiązań z solvera

Dla każdego interwału I_i w rozwiązaniu zapisujemy następujące wartości:

- L_i
- s_i
- e_i
- v_i
- d — aktualnie przetwarzany dzień tygodnia
- $\mathcal{R}_i = \{r \in \mathcal{R} : \exists t \in \mathcal{T}$ takie, że $p_{i,t,r} = 1\}$ — zbiór aktywnych sal, które zostały przypisane do bloku L_i.

Mapowanie na strukturę lekcji

Dla każdych takich wartości tworzymy konkretne lekcje w zbiorze $\mathcal{Z}$. Wpierw tworzymy funkcję, która będzie przypisywać nauczycieli i prowadzone przez nich przedmioty s do odpowiednich sal.

$$f : T_i \to \mathcal{R}_i$$
$$\forall t \in T_i : f(t) = r \in \mathcal{R}_i \text{ takie, że } r \text{ obsługuje przedmiot } s \in \vec{S}_{L_i}$$

Konieczne jest użycie takiej funkcji, ponieważ każdy nauczyciel w jednym bloku może mieć przypisane wiele klas. Musimy stworzyć wiele przypisań, które będą miały taką samą salę r dla każdej klasy.

Dla każdego wymagania $w \in L_i$ i każdego slotu czasowego $h \in [1, e_i - s_i]$:

$$z = (d, h, c_w, t_w, s_w, f(t_w))$$
$$\mathcal{Z}_d \leftarrow \mathcal{Z}_d \cup \{z\}$$

Przykład transformacji

Rozważmy blok $L_i = \{w_1, w_2\}$ z $v_i = 2$, gdzie:

- $w_1 = (t_1, c_1, s_1, 3)$ — nauczyciel t_1, klasa c_1, wychowanie fizyczne
- $w_2 = (t_2, c_1, s_2, 3)$ — nauczyciel t_2, klasa c_1, wychowanie fizyczne

Po rozwiązaniu otrzymujemy przypisania:

$$z_1 = (1, 2, c_A, t_1, s_1, r_1)$$
$$z_2 = (1, 3, c_A, t_1, s_1, r_1)$$
$$z_3 = (1, 2, c_A, t_2, s_2, r_2)$$
$$z_4 = (1, 3, c_A, t_2, s_2, r_2)$$

Jest to równoznaczne z leckją wychowania fizycznego podzieloną na dwie grupy, które odbywają się w tym samym czasie.

Proces ten powtarzamy dla wszystkich dni tygodnia, tworząc kompletny plan lekcji $\mathcal{Z} = \bigcup_{d=1}^{5} \mathcal{Z}_d$ spełniający wszystkie ograniczenia i optymalizujący funkcję celu.

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

Najważniejszym aspektem działania aplikacji są funkcjonalności, które ona posiada. Takie wymagania nazywamy funkcjonalnymi — definiują one konkretne zachowania i funkcje, które system musi porafić wykonać.

W mojej aplikacji mogę podzielić takie wymagania na 4 grupy:

- **Obsługa wielu planów lekcji i konfiguracji:**
  - Tworzenie i przechowywanie wielu planów lekcji.
  - Możliwość pracy z wieloma zestawami zasobów i ograniczeń równolegle. Przykładowo dla obecnego roku szkolnego mamy zbiór ograniczeń głównych $\mathcal{W}_{2025/2026}$, ale dla następnego jest to juz zupełnie inne zbiór ograniczeń $\mathcal{W}_{2026/2027}$.
- **Wymagania dotyczące wprowadzania danych:**
  - Import plików `txt` oraz `csv` z zasobami i wymaganiami w formacie używanym przez placówkę, która dostarczyła dane na potrzeby tej pracy.
  - Możliwość ręcznego wprowadzenia i edycji wszystkich typów danych:
    * Nauczyciele, klasy, przedmioty, sale.
    * Wymagania główne i bloki przedmiotów.
    * Dostępności i możliwości sal.
- **Wymagania dotyczące generowania planu lekcji:**
  - Automatyczne generowanie kompletnych planów na podstawie zdefiniowanych ograniczeń.
  - Możliwość sterowania czasem generowania planu poprzez wprowadzanie liczby generacji w algorytmie ewolucyjnym.
- **Wymagania dotyczące wglądu na utworzone plany:**
  - Wymagany jest wgląd na plan z perspektywy:
    * Nauczycieli
    * Klas
  - Wymagana jest możliwość wglądu w wiele różnych planów lekcji.

4.1.2. Wymagania niefunkcjonalne

Wymagania niefunkcjonalne określają jakościowe cechy systemu, wpływające na jego użyteczność, wydajność i niezawodność.

Dla rozpatrywanej aplikacji przyjąłem 2 grupy tych wymagań:

- **Wymagania wydajnościowe:**
  - Czas generowania kompletnego planu nie powinien przekraczać 10 minut dla typowych przypadków.
  - Aplikacja musi obsługiwać realistyczne rozmiary danych: do 1000 wymagań głównych, 100 nauczycieli, 50 klas i 100 sal.
  - Zużycie pamięci operacyjnej nie może przekraczać 8GB podczas generowania planu.
- **Wymagania co do interfejsu:**
  - Interfejs powinien być intuicyjny dla użytkowników zaznajomionych z arkuszami kalkulacyjnymi.
  - Spójność interfejsu we wszystkich modułach aplikacji.
  - Przejrzystość, minimalizm, wygoda i prostota w użytkowaniu interfejsu.

Uwzględnienie zarówno wymagań funkcjonalnych, jak i niefunkcjonalnych pozwala na stworzenie kompletnej aplikacji, która nie tylko realizuje założone funkcje, ale także zapewnia komfortowe i niezawodne środowisko pracy dla docelowych użytkowników.

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 - addnooverlap, (2025). Ostatnio otworzono 25 listopada 2025.

[9] Google. Documentation, or-tools - intervalvar, (2025). Ostatnio otworzono 15 listopada 2025.

[10] Google. Documentation, or-tools - job shop, (2025). Ostatnio otworzono 25 listopada 2025.

[11] J. H. Holland. Genetic algorithms. *Scientific american*, 267(1):66–73, 1992.

[12] 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.

[13] 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.

[14] Vulcan. Plan lekcji optivum. układanie planu lekcji - krok po kroku, (2025). Ostatnio otworzono 25 listopada 2025.

[15] 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

2.1 Wizualizacja rozwiązania klasycznego problemu JSSP z wykorzystaniem zmiennych interwałowych [10] . . . 7
2.2 Zrzut ekranu importowania arkusza organizacyjnego do programu „Plan lekcji Optivum” [14] . . . 9
2.3 Zrzut ekranu wprowadzania sal do programu „Plan lekcji Optivum” [14] . . . 10
2.4 Zrzut ekranu wprowadzania preferencji sal względem przedmiotów do programu „Plan lekcji Optivum” [14] . . . 10
2.5 Zrzut ekranu wprowadzania preferencji sal względem nauczycieli do programu „Plan lekcji Optivum” [14] . . . 10
2.6 Zrzut ekranu wprowadzania bloków przedmiotów do programu „Plan lekcji Optivum” [14] . . . 11
2.7 Zrzut ekranu wprowadzania terminów zajęć dla poszczególnych klas do programu „Plan lekcji Optivum” [14] . . . 11
2.8 Zrzut ekranu wprowadzania dostępności nauczycieli do programu „Plan lekcji Optivum” [14] . . . 12
2.9 Zrzut ekranu wprowadzania „trudnych” lekcji do programu „Plan lekcji Optivum” [14] . . . 12

3.1 Zrzut ekranu z arkusza kalkulacyjnego przedstawiający przypisanie godzinowe nauczycieli dla każdej klasy. . . . 17
3.2 Wykres funkcji użytej do ewaluacji rozkładu godzin nauczycieli i klas na przestrzeni tygodnia. . . . 32

Spis tabel