

Politechnika Wrocławskiego
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

Slowa kluczowe: planowanie lekcji, algorytm ewolucyjny, aplikacja webowa, harmonogramowanie

WROCŁAW (2025)

Streszczenie

Problem automatycznego układania szkolnego planu lekcji jest złożonym zagadnieniem optymalizacyjnym, z którym corocznie mierzą się placówki oświatowe. Konieczność równoczesnego spełnienia wielu ograniczeń — dostępności nauczycieli, niekolizyjności zajęć, ciągłości lekcji oraz minimalizacji okienek — sprawia, że ręczne tworzenie planu jest procesem niezwykle czasochłonnym.

Niniejsza praca ma na celu usprawnienie tego procesu poprzez zaprojektowanie i implementację algorytmu oraz funkcjonalnej aplikacji webowej. Kluczową innowacją jest dekompozycja problemu na trzy sekwencyjne etapy: grupowanie pojedynczych zajęć w bloki lekcyjne, optymalny przydział tych bloków do dni tygodnia za pomocą algorytmu ewolucyjnego oraz finalne przypisanie do nich konkretnych sal i terminów.

Proponowane rozwiązanie zostało w pełni zaimplementowane jako aplikacja webowa używająca Reacta oraz Django. Sercem systemu jest moduł optymalizacyjny napisany w Pythonie, który do efektywnego rozwiązywania problemu harmonogramowania wykorzystuje bibliotekę Google OR-Tools, a do obliczeń w algorytmie ewolucyjnym bibliotekę NumPy.

Abstract

The automatic creation of a school timetable is a complex optimization problem that educational institutions face annually. The need to simultaneously satisfy multiple constraints — teacher availability, non-overlapping lessons, continuity of classes, and minimization of teacher gaps — makes manual schedule creation a laborious process.

This thesis aims to streamline this process by designing and implementing an algorithm and a functional web application. The key innovation is the decomposition of the problem into three sequential stages: grouping individual lessons into instructional blocks, optimally allocating these blocks to weekdays using an evolutionary algorithm, and the final assignment of specific rooms and time slots to them.

The proposed solution has been fully implemented as a web application using React and Django. The core of the system is an optimization module written in Python, which utilizes the Google OR-Tools library for efficiently solving the scheduling problem and the NumPy library for computations within the evolutionary algorithm.

Spis treści

1 Wstęp	1
1.1 Cel i zakres pracy	1
1.2 Aspekt systemowy	2
1.3 Układ pracy	2
2 Powiązane prace	5
2.1 Inspiracja podejściem iteracyjnym	5
2.1.1 Definicja problemu harmonogramowania	5
2.2 Problem harmonogramowania zadań typu job-shop	7
2.2.1 Podobieństwa do problemu układania planu lekcji	8
2.3 Istniejące kompleksowe rozwiązania	9
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	16
3.1.2 Ograniczenia	19
3.1.3 Funkcja celu	20
3.2 Poprzednie podejścia	21
3.2.1 Programowanie mieszanocalkowitoliczbowe	21
3.2.2 Grafowe sieci neuronowe	23
3.3 Dekompozycja problemu i wybór technik	23
3.3.1 Problem grupowania	25
3.3.2 Problem alokacji	28
3.3.3 Problem harmonogramowania	29
3.4 Algorytm zachłanny	30
3.4.1 Działanie	31
3.4.2 Przykładowy wynik	33
3.5 Algorytm ewolucyjny	34
3.5.1 Kodowanie i ograniczenia twardye	34
3.5.2 Generowanie populacji początkowej	35
3.5.3 Przystosowanie	35
3.5.4 Selekcja	38
3.5.5 Krzyżowanie	38
3.5.6 Mutacja	39
3.5.7 Przykładowy wynik	40

3.6	Solver programowania liniowego z ograniczeniami	41
3.6.1	Zmienne decyzyjne	41
3.6.2	Ograniczenia	41
3.6.3	Funkcja celu	42
3.6.4	Transformacja przedziałów na przypisania końcowe	42
3.6.5	Przykładowy wynik	44
4	Aplikacja	45
4.1	Specyfikacja wymagań	45
4.1.1	Wymagania funkcjonalne	45
4.1.2	Wymagania niefunkcjonalne	45
4.1.3	Przypadki użycia	46
4.1.4	Diagramy sekwenacji	55
4.2	Projekt	57
4.2.1	Architektura aplikacji	57
4.2.2	Projekt struktury bazy danych	58
4.2.3	Projekt interfejsu	65
4.3	Implementacja	66
4.3.1	Wybór narzędzi	66
4.3.2	Implementacja bazy danych w Django	68
4.3.3	Implementacja API	68
4.3.4	Implementacja interfejsu webowego	69
4.3.5	Integracja z algorytmem	76
5	Weryfikacja i walidacja rozwiązania	77
5.1	Scenariusze testowe	77
5.1.1	Scenariusz testowy generowania planu lekcji z określonymi parametrami	77
5.1.2	Scenariusz przeglądu planów lekcji	77
5.1.3	Scenariusz importowania i edytowania wymagań głównych	77
5.1.4	Scenariusz edytowania dostępności nauczycieli	77
5.2	Testowanie aplikacji	80
5.2.1	Test generowania planu lekcji z określonymi parametrami	80
5.2.2	Test przeglądania planów lekcji	80
5.2.3	Test importowania i edytowania wymagań głównych	81
5.2.4	Test edytowania dostępności nauczycieli	82
5.3	Analiza jakości wyników generowania planu lekcji	83
5.3.1	Omówienie danych wejściowych	83
5.3.2	Omówienie całego planu	83
5.3.3	Omówienie szczególnych przypadków	88
5.3.4	Wydajność generowania planu lekcji	91
6	Wnioski i perspektywy rozwoju	93
6.1	Problemy rozwiązania	93
6.2	Perspektywy rozwoju	93
6.3	Wnioski	94

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 dla zadań programowania z ograniczeniami do szczegółowego harmonogramowania,
- integracja aplikacji webowej z algorytmem.

1.2. Aspekt systemowy

W pracy zastosowałem podejście systemowe, ponieważ stworzenie aplikacji do generowania planu lekcji wymaga spojrzenia wykraczającego poza sam algorytm optymalizacyjny. Takie podejście obejmuje analizę wymagań, ich formalizację, projekt i implementację rozwiązania oraz końcową walidację, co pozwala traktować aplikację jako kompletny system informatyczny, a nie jedynie algorytm.

Podejście systemowe pomogło mi także z dogłębną analizą problemu oraz jego dalszą dekompozycją. Modelowanie części algorytmu jako obiektów wejściowo-wyjściowych znacząco ułatwiało proces inżynierski.

Zróżnicowanie uwarunkowań istotnie wpływa na projekt systemu. Należy uwzględnić zarówno wymagania prawne, fizyczne ograniczenia szkoły (dostępność sal i nauczycieli), jak i kwestie związane z komfortem uczniów, takie jak równomierne rozłożenie obciążień dydaktycznych. Wszystkie te czynniki muszą być odzwierciedlone w modelu danych oraz w module optymalizacyjnym.

Istotny jest również aspekt użytkowy: system musi być praktyczny i intuicyjny w obsłudze. Dlatego projekt obejmuje także ergonomię interfejsu oraz sposób prezentacji danych.

Podejście systemowe jest konieczne, aby stworzyć spójne, skalowalne i odpowiadające rzeczywistym potrzebom szkoły narzędzie.

1.3. Układ pracy

Praca dzieli się na sześć zasadniczych rozdziałów, które kolejno wprowadzają w tematykę, przedstawiają stan wiedzy, prezentują autorskie rozwiązanie, opisują jego praktyczną implementację, weryfikują jej skuteczność oraz podsumowują uzyskane rezultaty i wskazują kierunki dalszego rozwoju.

Rozdział 2 poświęcony jest przeglądowi powiązanych prac i stanowi teoretyczne zaplecze dla dalszych rozważań. W jego ramach dokonałem analizy istniejących podejść naukowych i komercyjnych. Szczególnie dokładnie omówiłem podejście iteracyjne programowania mieszanego

całkowitoliczbowego, które było inspiracją dla architektury aplikacji. Następnie przedstawiłem klasyczny problem harmonogramowania zadań typu job-shop oraz pokazałem, jak mój problem jest jego odmianą. Na koniec, w celu zrozumienia rzeczywistych wymagań i wyzwań, dokonałem szczegółowego przeglądu wiodącego na rynku polskim systemu komercyjnego.

Centralną i kluczową częścią pracy jest rozdział 3, który w całości dotyczy problemu układania planu lekcji i proponowanego algorytmu jego rozwiązania. Rozdział zaczyna się od formalnego sformułowania problemu optymalizacyjnego, włączając w to definicję terminologii, danych wejściowych, szukanych, ograniczeń i funkcji celu. Następnie, w oparciu o doświadczenie z wcześniejszych prób rozwiązania, zaprezentowałem kluczową innowację pracy: hierarchiczną dekompozycję oryginalnego, złożonego problemu na trzy mniejsze etapy. Dla każdego z tych etapów opisałem dobrą i zasadę działania techniki adekwatnej do danego podproblemu oraz przykładowe wyniki na rzeczywistych danych, co pozwoli na prześledzenie logiki całego procesu generowania planu.

Rozdział 4 koncentruje się na praktycznym aspekcie pracy, czyli projekcie oraz implementacji aplikacji webowej. Zaczyna się on od specyfikacji wymagań, przypadków użycia oraz diagramów sekwencji. W dalszej części szczegółowo opisałem projekt systemu obejmujący jego ogólną architekturę, projekt bazy danych oraz omówienie projektu interfejsu użytkownika. Ostatnia część tego rozdziału opisuje implementację oraz uzasadnia wybór stosu technologicznego.

Rozdział 5 służy weryfikacji i walidacji stworzonego rozwiązania. Jego pierwsza część zawiera formalne scenariusze testowe kluczowych funkcjonalności aplikacji, a następna dokumentuje testy przeprowadzone na ich podstawie. Najważniejszą część tego rozdziału stanowi analiza jakościowa wyników. Obejmuje ona omówienie ograniczeń posiadanych danych, analizę całościową wygenerowanego planu lekcji, a także ocenę kluczowego wskaźnika — liczby okienek nauczycieli. Dalej szczegółowo analizuję krytyczny przypadek klasy o profilu łączonym oraz omawiam wydajność czasową całego procesu optymalizacyjnego.

Ostatni rozdział pracy 6 stanowi podsumowanie oraz prezentuje perspektywy rozwoju. Najpierw omówiłem ograniczenia i problemy bieżącej wersji rozwiązania. Następnie zaprezentowałem konkretne kierunki dalszych prac, które pomogą w rozwiązaniu wcześniej wspomnianych problemów. Rozdział, a tym samym cała praca, zamknie się sformułowaniem ostatecznych wniosków, potwierdzających trafność przyjętej metody dekompozycji oraz oceniających praktyczną przydatność opracowanego systemu.

2. Powiązane prace

Rozdział prezentuje przegląd literatury w trzech wyraźnie oddzielonych obszarach, które stanowią fundament dla rozwiązania zaproponowanego w pracy:

- Podejścia algorytmiczne i metodyczne — analiza pracy naukowej przedstawiającej zaawansowaną, iteracyjną metodę optymalizacji, która posłużyła jako kluczowa inspiracja architektoniczna.
- Podstawy teoretyczne — formalne wprowadzenie do klasycznego problemu harmonogramowania zadań typu job-shop, który stanowi teoretyczną ramę dla problemu harmonogramowania i uzasadnia użycie technik programowania z ograniczeniami.
- Rozwiązania praktyczne i komercyjne — szczegółowy przegląd wiodącego na rynku polskim systemu, służący zrozumieniu rzeczywistych wymagań, złożoności problemu oraz ustaleniu punktu odniesienia dla oceny praktyczności własnego rozwiązania.

2.1. Inspiracja podejściem iteracyjnym

Praca „A mixed-integer programming approach for solving university course timetabling problems” [13] przedstawia innowacyjne podejście do rozwiązywania problemów harmonogramowania poprzez dekompozycję na prostsze podproblemy i iteracyjne dodawanie ograniczeń. Autorzy wykorzystują strategię polegającą na:

- Uzyskaniu rozwiązania początkowego z podstawowym zestawem ograniczeń.
- Iteracyjnym „wstrzykiwaniem” kolejnych ograniczeń z wykorzystaniem zmiennych sztucznych.
- Stosowaniu heurystyk redukcji zmiennych dla kontroli złożoności obliczeniowej.

2.1.1. Definicja problemu harmonogramowania

W przedstawionym podejściu problem harmonogramowania zajęć uniwersyteckich definiowany jest jako zadanie przydzielenia zajęć do dostępnych sal i terminów oraz przypisania studentów do odpowiednich grup zajęciowych, z uwzględnieniem złożonych ograniczeń i preferencji. Głównymi komponentami problemu są:

- **Zajęcia:** Reprezentują pojedyncze lekcje dydaktyczne, które muszą zostać zaplanowane. Każde zajęcie l musi zostać przypisane do slotu czasowego h z listy dopuszczalnych slotów H_l oraz, w wymaganych przypadkach, przypisanie sali r z listy dostępnych pomieszczeń R_l .

- **Ograniczenia dystrybucyjne:** Stanowią zbiór reguł określających relacje pomiędzy różnymi zajęciami. Wyróżnia się ograniczenia twardie, które muszą być bezwzględnie spełnione, oraz miękkie, których naruszenie generuje kary w funkcji celu. Przykładowo wymóg rozpoczętania zajęć w tym samym czasie, zakaz nakładania się terminów czy wymóg zachowania minimalnego odstępu czasowego.
- **Struktura kursów i konfiguracji:** Każdy kurs posiada hierarchiczną strukturę, w ramach której student musi wybrać jedną konfigurację k , za następnie po jednym zajęciu z każdej części składowej P_k danej konfiguracji.
- **Konflikty studentów:** Powstają gdy student jest przypisany do zajęć odbywających się w tym samym czasie lub gdy czas pomiędzy zajęciami jest niewystarczający na przemieszczenie się między salami.

Celem optymalizacji jest znalezienie rozwiązania spełniającego wszystkie twardie ograniczenia przy minimalizacji łącznej kary, uwzględniającej naruszenia ograniczeń miękkich oraz konflikty studentów.

Kluczowym elementem przedstawionej w pracy metody jest dekompozycja oryginalnego, złożonego problemu na szereg prostszych podproblemów, rozwiązywanych sekwencyjnie w procesie iteracyjnym.

Podstawowy model MIP

Podstawę stanowi sformułowanie problemu mieszanocałkowitoliczbowego (MIP) z czterema typami zmiennych decyzyjnych:

- zmienne $x_{c,h}$ określają czy zajęcia l są przypisane do terminu h ,
- zmienne $y_{c,r}$ określają czy zajęcia l są przypisane do sali r ,
- zmienne $z_{s,l}$ określają czy student s jest przypisany do zajęć l ,
- zmienne $Z_{s,k}$ określają czy student s wybiera konfigurację k .

Iteracyjne wstrzykiwanie ograniczeń

Model obejmuje fundamentalne ograniczenia zapewniające poprawność podstawowej struktury rozwiązania, takie jak wymóg przypisania każdych zajęć do dokładnie jednego terminu i sali, spełnienie limitów liczby studentów w grupach oraz zakaz równoczesnego użytkowania tej samej sali przez różne zajęcia. Ze względu na jego bardzo dużą złożoność, autorzy stosują podejście wieloetapowe:

- **Rozwiążanie początkowe:** Optymalizacja rozpoczyna się od uproszczonego modelu zawierającego jedynie fundamentalne ograniczenia oraz podstawowe składniki funkcji celu.
- **Dodawanie ograniczeń dystrybucyjnych:** Twarde ograniczenia dystrybucyjne są dodawane do modelu w formie ograniczeń wykluczających niedopuszczalne kombinacje przypisań czasowych $x_{c,h}$ i salowych $y_{c,r}$.

- **Obsługa ograniczeń specjalnych:** Dla czterech szczególnie złożonych typów ograniczeń („MaxDays”, „MaxDayLoad”, „MaxBreaks”, „MaxBlock”) stosowane jest podejście z użyciem „leniwych” ograniczeń (ang. *lazy constraints*). Takie ograniczenia są dodawane dopiero gdy zostanie znalezione rozwiązanie całkowitoliczbowe, które je narusza.
- **Iteracyjne uwzględnianie konfliktów studentów:** Podobne podejście stosowane jest dla uwzględnienia w funkcji celu kar za konflikty studentów. Początkowo pomija się najbardziej złożone składowe związane z konfliktami, a następnie iteracyjnie dodaje się do modelu te ograniczenia, które zostały naruszone w poprzednich uruchomieniach.

Strategie redukcji złożoności

Aby utrzymać złożoność obliczeniową na akceptowalnym poziomie, autorzy stosują zaawansowane strategie redukcji modelu:

- **Eliminacja zmiennych:** identyfikacja i usuwanie zmiennych, których wartości można ustalić z góry na podstawie analizy struktury problemu.
- **Agregacja ograniczeń:** grupowanie podobnych ograniczeń w pojedyncze, bardziej zwarte warunki, co znaczaco redukuje rozmiar modelu.
- **Leniwe ograniczenia:** najbardziej złożone ograniczenia są początkowo pomijane, a następnie dodawane tylko gdy zostały naruszone w poprzednich iteracjach.

Podejście to pozwala na stopniowe poprawianie jakości rozwiązania przy kontrolowanym wzroście złożoności obliczeniowej, zachowując wykonalność rozwiązania na każdym etapie procesu optymalizacji.

2.2. Problem harmonogramowania zadań typu job-shop

Problem harmonogramowania typu job-shop (ang. *Job-Shop Scheduling Problem* — JSSP) należy do klasycznych zagadnień optymalizacji kombinatorycznej w badaniach operacyjnych i zarządzaniu produkcją. Ze względu na szerokie zastosowania inżynierskie był intensywnie analizowany w literaturze. Klasyczny wariant, podsumowany m.in. w „A survey of job shop scheduling problem: The types and models” [16], można sformalizować następująco.

Rozważamy zbiór maszyn:

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

zbiór zadań (prac):

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

oraz dla każdego zadania J_i liniowo uporządkowaną sekwencję operacji:

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

Każda operacja O_{ij} jest przypisana do dokładnie jednej maszyny (oznaczanej $\mu(i, j) \in \{1, \dots, m\}$) oraz posiada znany, dodatni czas przetwarzania $p_{ij} \in \mathbb{N}^+$.

Zmiennymi decyzyjnymi są czasy rozpoczęcia operacji $S_{ij} \in \mathbb{N}_0$. Definiujemy czas zakończenia operacji jako $C_{ij} = S_{ij} + p_{ij}$, a czas zakończenia zadania J_i jako $C_i = C_{i n_i}$.

Celem jest minimalizacja maksymalnego czasu zakończenia (ang. *makespan*):

$$\min C_{\max} \quad \text{przy} \quad C_{\max} \geq C_i \quad \forall i \in \{1, \dots, n\}$$

Model spełnia następujące ograniczenia:

- **Ograniczenia kolejnościowe (precedencji):** dla każdego zadania J_i i $j = 1, \dots, n_i - 1$ musi zachodzić

$$S_{i,j+1} \geq S_{ij} + p_{ij}.$$

- **Ograniczenia zasobowe (brak nakładania na tej samej maszynie):** dla każdej pary różnych operacji (O_{ij}, O_{kl}) takich, że $\mu(i, j) = \mu(k, l)$ obowiązuje dysjunkcja czasowa

$$S_{ij} + p_{ij} \leq S_{kl} \quad \text{lub} \quad S_{kl} + p_{kl} \leq S_{ij}, \quad (2.1)$$

co w praktyce implementuje się poprzez zmienne binarne i duże stałe M (formulacje MIP) albo jako ograniczenia dysjunktywne.

2.2.1. Podobieństwa do problemu układania planu lekcji

Problem planowania zajęć szkolnych można postrzegać jako rozszerzenie JSSP z wieloma typami zasobów. Podstawowe podobieństwa strukturalne obejmują:

Struktura zasobów

W JSSP rozważany jest zbiór maszyn M . W problemie planu lekcji występują trzy typy zasobów:

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

gdzie:

- \mathcal{T} — zbiór nauczycieli (zasób typu „nauczyciel”),
- \mathcal{C} — zbiór klas (zasób typu „klasa”),
- \mathcal{R} — zbiór sal (zasób typu „sala”).

Struktura zadań

W klasycznym JSSP każde zadanie J_i składa się z operacji O_{ij} wymagających jednego zasobu (maszyny). W układaniu planu zajęć pojedyncza lekcja wymaga jednoczesnego dostępu do trzech zasobów: nauczyciela, klasy i sali. Reprezentację lekcji można zapisać jako

$$z_i = (d_i, h_i, c_i, t_i, s_i, r_i),$$

gdzie d_i to dzień, h_i to slot czasowy (przykładowo zerowy: 7:00–7:45), c_i to klasa, t_i to nauczyciel, s_i to przedmiot, a r_i to sala.

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

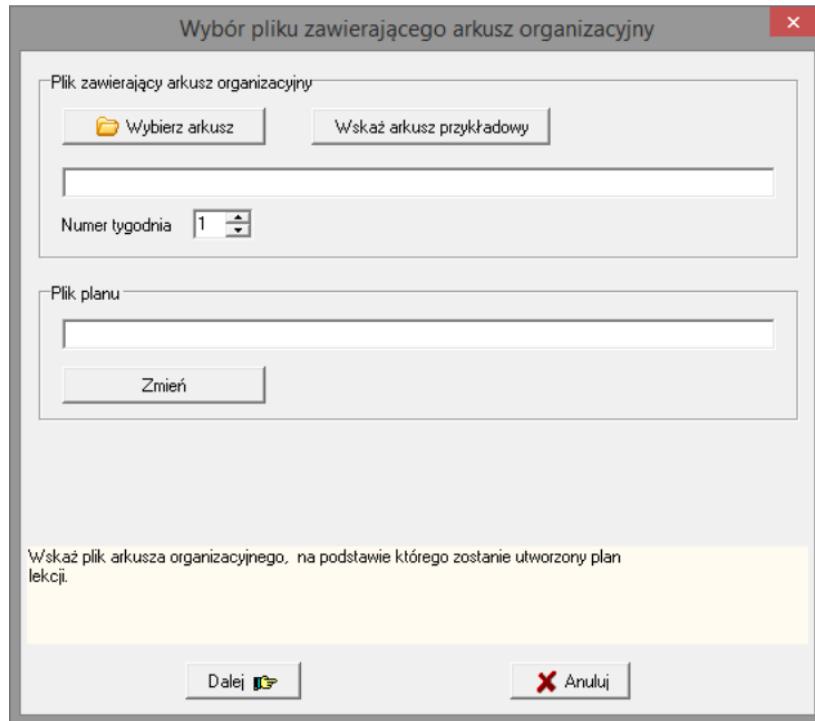
Vulcan, jako jeden z najstarszych na rynku dostawców, opracował kompleksowe rozwiązanie obejmujące nie tylko układanie planu lekcji, ale także dziennik elektroniczny, sekretariat i inne moduły zarządzania szkołą. Jego popularność w polskich szkołach wynika z lat doświadczenia w dostosowywaniu systemu 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 zainportowania danych z arkusza organizacyjnego, a kończy na publikacji gotowego planu [15].

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

W kolejnym etapie użytkownik definiuje 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 2.1: Zrzut ekranu importowania arkusza organizacyjnego do programu „Plan lekcji Optivum” [15]

Rysunek 2.2: Zrzut ekranu wprowadzania sal do programu „Plan lekcji Optivum” [15]

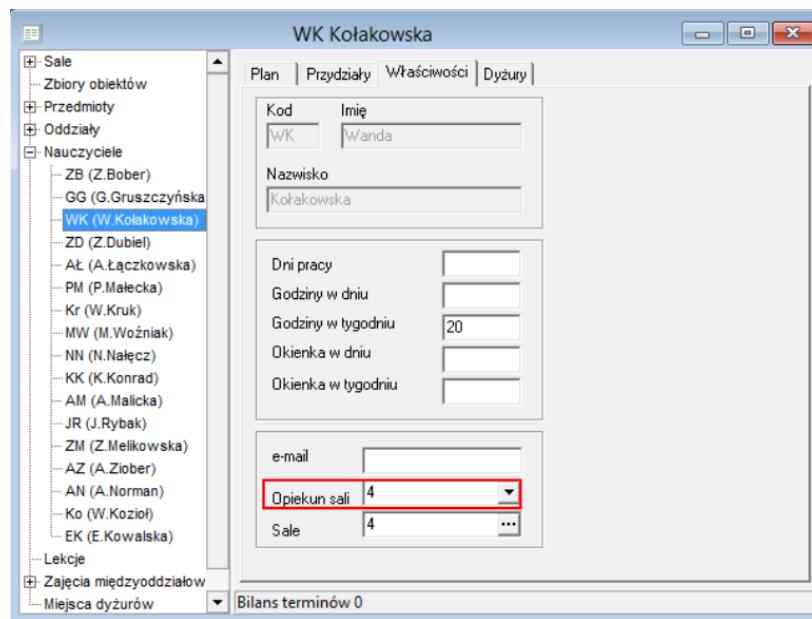
Po wprowadzeniu sal oraz ich preferencji użytkownik jest poproszony o wprowadzenie ewentualnych podziałów na bloki. Decyzja o rozkładzie godzin w tygodniu ma bezpośredni wpływ na grupy. Dla przydziału 4-godzinnego WF-u, podział na bloki 2,1,1 oznacza, że jedna z lekcji (przykładowo dla dziewcząt) będzie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

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.

Następnie definiuje się dostępność nauczycieli. Wybrane terminy nauczyciela można



Rysunek 2.3: Zrzut ekranu wprowadzania preferencji sal względem przedmiotów do programu „Plan lekcji Optivum” [15]



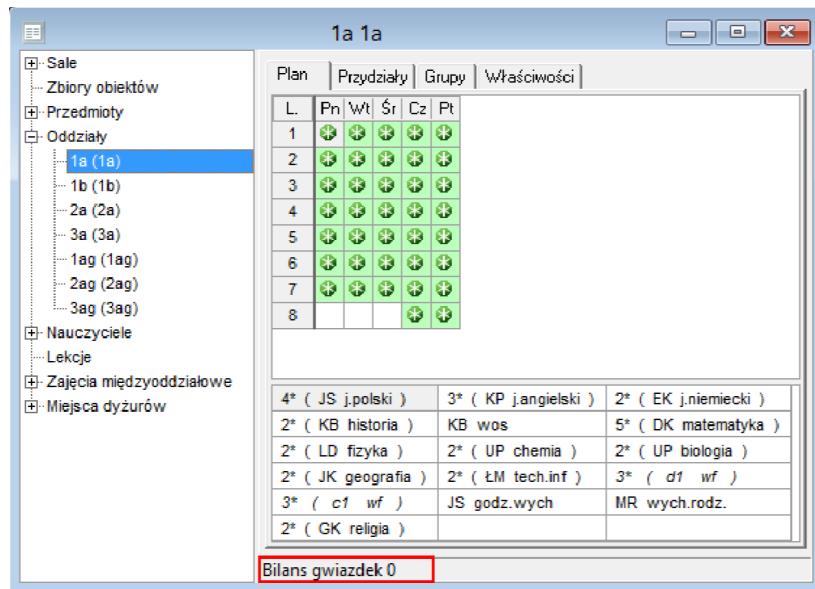
Rysunek 2.4: Zrzut ekranu wprowadzania preferencji sal względem nauczycieli do programu „Plan lekcji Optivum” [15]

j.polski Język polski								
Przydziel			Warunki układania			Właściwości		
Nauczyciel	Oddział	Grupa	Godziny	Blok	Sale	Blokada sal	Wybierz	Wyłącz
GG	1A	5	2	3	3	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
GG	f1	f1	3	4	4	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
WK	1B	5	2	3	3	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
GG	2A	5	2	3	3, hum.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
GG	3A	4	2	3	3, hum.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
GG	2B	5	2	4	4, hum.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
WK	3B	4	2	4	4, hum.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
WK	2C	5	2	4	4, hum.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
GG	3C	4	2	3	3, hum.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

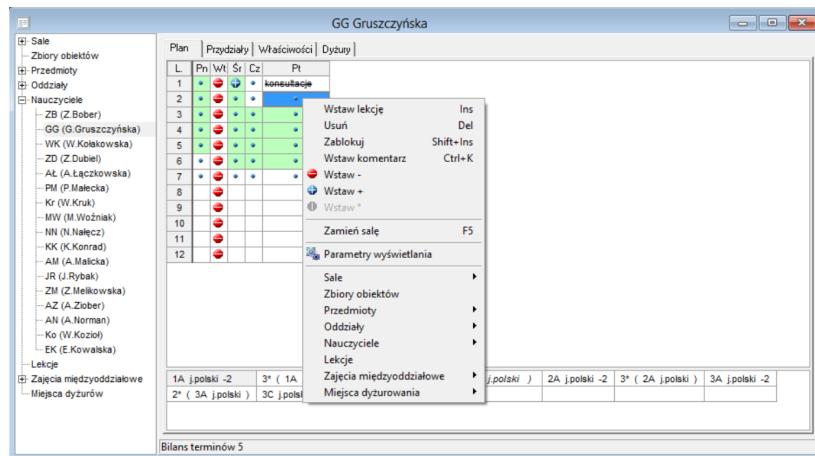
Rysunek 2.5: Zrzut ekranu wprowadzania bloków przedmiotów do programu „Plan lekcji Optivum” [15]

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



Rysunek 2.6: Zrzut ekranu wprowadzania terminów zajęć dla poszczególnych klas do programu „Plan lekcji Optivum” [15]



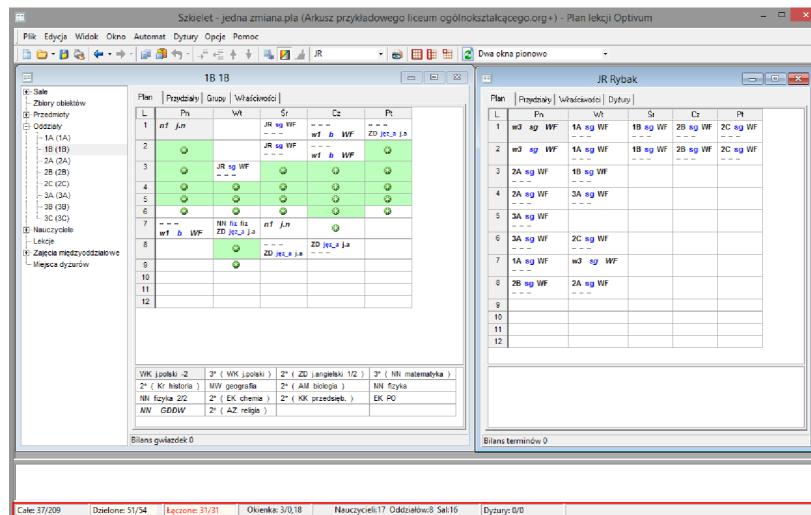
Rysunek 2.7: Zrzut ekranu wprowadzania dostępności nauczycieli do programu „Plan lekcji Optivum” [15]

międzyoddziałowe.

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 przeanalizować nieułożone lekcje i złagodzić 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.

Ceną uniwersalności, jest konieczność stworzenia rozbudowanej aplikacji wymagającej od



Rysunek 2.8: Zrzut ekranu wprowadzania „trudnych” lekcji do programu „Plan lekcji Optimum” [15]

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 Optimum” 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 usługę tak rozbudowanego systemu, może okazać się zbyt skomplikowane i kosztowne.

3. Problem układania planu lekcji i algorytm jego rozwiązania

Rozdział przedstawia kompleksowe rozwiązanie problemu układania szkolnego planu lekcji. Jego struktura odzwierciedla logikę stopniowego konstruowania finalnego algorytmu.

- **sformułowanie problemu** — rozpoczynam od precyzyjnej definicji terminologii, danych wejściowych, oczekiwanej wyniku oraz formalnego opisu ograniczeń i funkcji celu. Stanowi to fundament dla dalszych rozważań,
- **analiza poprzednich podejść** — krótko omawiam wcześniejsze, nieskuteczne próby rozwiązania, aby uzasadnić konieczność opracowania nowej, hybrydowej metody,
- **koncepcja dekompozycji i dobór technik** — przedstawiam kluczową innowację pracy: podział złożonego problemu na trzy mniejsze, sekwencyjne etapy. Dla każdego z nich uzasadniam wybór najodpowiedniejszej techniki algorytmicznej,
- **szczegółowy opis algorytmów** — kolejno, w osobnych sekcjach, szczegółowo opisuję działanie każdego z trzech algorytmów składowych, ilustrując je pseudokodami i przykładami wyników na rzeczywistych danych.

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”,
- **sala**: Miejsce, w którym prowadzone są zajęcia; przykładowo „Sala Gimnastyczna 1”, „2”,
- **przedmiot**: Temat zajęć prowadzonych przez nauczyciela; przykładowo „Wychowanie Fizyczne”, „Matematyka”,
- **lekcia**: 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,
- **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 efektywnym rozwiązaniem jednego wielkiego problemu programowania całkowitoliczbowego w sensownym czasie przy użyciu komputera z przeciętną specyfikacją.

3.1.1. Dane i szukane

Słownik podstawowych oznaczeń

- \mathcal{C} — zbiór klas,
- \mathcal{T} — zbiór nauczycieli,
- \mathcal{S} — zbiór przedmiotów,
- \mathcal{R} — zbiór sal,
- \vec{R}_r — zbiór przedmiotów obsługiwanych przez salę r ,
- H — liczba slotów czasowych w dniu.

Wymagania główne

Warto zacząć od przedstawienia sposobu reprezentacji wymagań głównych. Liczba 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 rysunek 3.1 przedstawiający dotychczasowy sposób przypisywania liczby godzin nauczycieli do klas 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	BA
1	Projekt 2025/2026	psychol	jęz-tur	społ-pr	polit	b-ch	eko-men		psych-prawni	jez-ekonomiczni	polit-biol	eko-men																	
2		I	I	I	I	I	I		II	II	II	II	II	II	II	II	II	II	II	II	II	II	IV	IV	IV	IV	IV		
3		A	B	C	D	F	G		AC	BG	DF	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							
5																													
6	Jp1																												
7																													
8	Jp2																												
9																													
10	Jp3																												
11																													
12	Jp3																												
13																													
14	Jp4																												
15																													
16	Jp5																												
17																													
18																													

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 liczba godzin, która będzie poświęcona na prowadzenie tego przedmiotu.

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

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

gdzie $g_{w,i}$ to liczba wymaganych tygodniowo godzin przedmiotu $s_{w,i}$ przeprowadzonego przez nauczyciela $t_{w,i}$ dla klasy $c_{w,i}$.

Dostępność nauczycieli

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.

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

Bloki przedmiotów

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

$$b_i \triangleq (\vec{S}_{b,i}, \vec{C}_{b,i}, \vec{G}_{b,i}, \gamma_{b,i}, \epsilon_{b,i}) \in \mathcal{B}$$

$$\forall i \in \{1, 2, \dots, |\mathcal{B}|\} : \begin{cases} \vec{S}_{b,i} & \subset \mathcal{S} \\ \vec{C}_{b,i} & \subseteq \mathcal{C} \\ \vec{G}_{b,i} & \in \mathbb{N}^{|\vec{S}_{b,i}|} \\ \gamma_{b,i} & \in \{0, 1\} \\ \epsilon_{b,i} & \in \mathbb{N} \end{cases}$$

gdzie

- $\vec{S}_{b,i}$ to podzbiór wszystkich przedmiotów, 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 godzin przedmiotów w bloku,
- γ to wartość binarna informująca o tym czy blok jest *agregujący* — służący do łączenia innych bloków w jeszcze większe bloki,
- ϵ to liczba naturalna która informuje o maksymalnej liczbie bloków. Jeśli $\epsilon = 0$ to, liczba bloków jest nielimitowana.

Przykładowy blok przedmiotów:

- $\vec{S}_b = (\text{język angielski, informatyka}),$
- $\vec{C}_b = (\text{IA, IB, IC}),$
- $\vec{G}_b = (1, 1),$
- $\gamma = 0,$
- $\epsilon = 0.$

lub:

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

Szukane

Oczekiwanym rezultatem działania algorytmu powinien być zbiór przypisań \mathcal{Z} . Każde takie przypisanie powinno mieć 6 wartości:

$$z_i \triangleq (d_{z,i}, h_{z,i}, c_{z,i}, t_{z,i}, s_{z,i}, r_{z,i}) \in \mathcal{Z}, \quad \forall i \in \{0, 1, \dots, |\mathcal{Z}_d|\} : \begin{cases} d_{z,i} \in \{1, 2, 3, 4, 5\} \\ h_{z,i} \in \{0, 1, \dots, H\} \\ c_{z,i} \in \mathcal{C} \\ t_{z,i} \in \mathcal{T} \\ s_{z,i} \in \mathcal{S} \\ r_{z,i} \in \mathcal{R} \end{cases}$$

Przykładowe interpretacje takich przypisań:

- poniedziałek, Slot czasowy 0, IA, nauczyciel francuskiego 1, język francuski, Sala nr 1,
- poniedziałek, Slot czasowy 0, IIA, nauczyciel francuskiego 1, język francuski, Sala nr 1,
- piątek, Slot czasowy 10, IVC, nauczyciel fizyki 2, fizyka, Sala nr 15.

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

3.1.2. Ograniczenia

Wcześniej wspomniane ograniczenia można podzielić na 4 kategorie:

Ograniczenia fizyczne

- żaden nauczyciel nie może być w 2 miejscach na raz,

$$\forall i, j \in \{1, 2, \dots, |\mathcal{Z}|\} : t_{z,i} = t_{z,j} \implies [i = j \vee d_{z,i} \neq d_{z,j} \vee h_{z,i} \neq h_{z,j} \vee (d_{z,i} = d_{z,j} \wedge h_{z,i} = h_{z,j} \wedge r_{z,i} = r_{z,j})]$$

- nauczyciel musi być dostępny,

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

- żaden uczeń nie może być w 2 miejscach na raz,

$$\forall i, j \in \{1, 2, \dots, |\mathcal{Z}|\}, \exists k \in \{1, 2, \dots, |\mathcal{B}|\} : (d_{z,i} = d_{z,j} \wedge h_{z,i} = h_{z,j} \wedge c_{z,i} = c_{z,j} \wedge i \neq j) \implies c_{z,i} \in \vec{C}_{b,k} \wedge s_{z,i}, s_{z,j} \in \vec{S}_{b,k}$$

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

$$\forall i, j \in \{1, 2, \dots, |\mathcal{Z}|\}, \exists k \in \{1, 2, \dots, |\mathcal{B}|\} : (d_{z,i} = d_{z,j} \wedge h_{z,i} = h_{z,j} \wedge r_{z,i} = r_{z,j} \wedge i \neq j) \implies s_{z,i} = s_{z,j} \wedge s_{z,i}, s_{z,j} \in \vec{S}_{b,k}$$

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.

$$\begin{aligned} \forall i, j \in \{1, 2, \dots, |\mathcal{Z}|\}, c \in \mathcal{C}, s \in \mathcal{S} : \\ (d_{z,i} = d_{z,j} \wedge c_{z,i} = c_{z,j} \wedge s_{z,i} = s_{z,j} \wedge i \neq j) \implies |h_{z,i} - h_{z,j}| = 1 \end{aligned}$$

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,

$$\begin{aligned} \nexists i, j \in \{1, 2, \dots, |\mathcal{Z}|\}, c \in \mathcal{C} : \\ d_{z,i} = d_{z,j} \wedge c_{z,i} = c_{z,j} \wedge |h_{z,i} - h_{z,j}| > |\{z_k \in \mathcal{Z} : d_{z,k} = d_{z,i} \wedge c_{z,k}\}| \end{aligned}$$

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

$$\forall z_i \in \mathcal{Z} : s_{z,i} \in \vec{R}_{r_z}$$

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

$$\begin{aligned} \forall w_i \in \mathcal{W}, \exists d \in \{1, 2, 3, 4, 5\} : \\ g_w = 3 \implies |\{z_j \in \mathcal{Z} : c_{w,i} = c_{z,j} \wedge s_{w,i} = s_{z,j} \wedge t_{w,i} = t_{z,j} \wedge d_{z,j} = d\}| = 2 \end{aligned}$$

Ograniczenie główne

Zgodnie z definicją przedstawioną powyżej, tygodniowa liczba godzin każdego przedmiotu musi być równa tej ustalonej w zbiorze wymagań:

$$\forall w_i \in \mathcal{W} : |\{z_j \in \mathcal{Z} : c_{w,i} = c_{z,j} \wedge t_{w,i} = t_{z,i} \wedge s_{w,i} = s_{z,i}\}| = g_{w,i}$$

3.1.3. Funkcja celu

Plan powinien posiadać możliwie najmniejszą liczbę okienek nauczycieli, co za tym idzie nauczyciele powinni spędzać możliwie najmniej czasu w szkole. Efekt ten możemy osiągnąć minimalizując czas od pierwszej do ostatniej lekcji przeprowadzonej przez nauczyciela każdego dnia:

$$F = \sum_{d=1}^5 \sum_{t \in \mathcal{T}} \left(\max_{\substack{i \in \{1, 2, \dots, |\mathcal{Z}|\} \\ t=t_{z,i} \wedge d=d_{z,i}}} h_{z,i} - \min_{\substack{i \in \{1, 2, \dots, |\mathcal{Z}|\} \\ t=t_{z,i} \wedge d=d_{z,i}}} h_{z,i} \right)$$

3.2. Poprzednie podejścia

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

3.2.1. Programowanie mieszanocałkowitoliczbowe

Moim pierwszym podejściem była próba użycia tylko i wyłącznie pakietu optymalizacyjnego *IBM ILOG CPLEX*. Problem zdefiniowałem używając zmiennych binarnych inspirując się podejściem opisany w sekcji 2.1, tworząc sześciowymiarową macierz:

- wymiar nauczycieli,
- wymiar klas,
- wymiar przedmiotów,
- wymiar dni,
- wymiar slotów czasowych,
- 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 krótkie 6 liczb całkowitych. W ten sposób zmienne, które nie posiadają klucza, nie powstają w pamięci. Przykładowo krotka reprezentująca (nauczyciel francuskiego, IIIA, poniedziałek, slot czasowy 0, niemiecki, siłownia) nie jest w zbiorze możliwych kluczy.

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

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

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 sum w ten sam sposób jak to robimy korzystając ze zmiennych decyzyjnych w postaci macierzowej. 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:

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

- Ograniczenie posiadania maksymalnie jednej lekcji w jednym slocie 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 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 rozwiązań. Wybrałem niekonwencjonalną reprezentację problemu harmonogramowania używając grafowych sieci neuronowych [14]. 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 wykonalności rozwiązania sprowadziła 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 kilkudziesięciu cyklach uczenia 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 (lekcia 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. Dekompozycja problemu i wybór technik

W niniejszej pracy zaadaptowałem oraz rozwiniąłem ideę przedstawioną w podrozdziale 2.1, dostosowując ją do specyfiki mojego problemu. Zdecydowałem się na podział zadania na trzy mniejsze, kolejno realizowane etapy, z których każdy rozwiązuje wyraźnie wydzielony podproblem o niższej złożoności obliczeniowej niż problem kompletny.

Analiza moich wcześniejszych prób rozwiązania tego zagadnienia wskazuje, że oparcie się wyłącznie na metodach inteligentnych bądź na MIP nie prowadzi do satysfakcjonujących rezultatów. Zasadnicza innowacja przedstawionego przeze mnie podejścia polega na dekompozycji oryginalnego problemu na trzy podproblemy rozwiązywane sekwencyjnie odpowiednimi technikami, każda używająca adekwatnej techniki. Jest to rozwinięcie względem moich wcześniejszych koncepcji.

Aby móc przeprowadzić dekompozycję zdecydowałem się na wprowadzenie następujących pojęć:

- **Blok lekcyjny:** Grupa dwóch lub więcej wymagań głównych, które reprezentują lekcje są w stanie odbywać 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. Na potrzeby tej pracy zbiory jednego wymagania głównego też traktujemy jako pojedyncze bloki lekcyjne.
- **Przydział godzinowy:** Posiadając już zbiór takich bloków lekcyjnych macierz ich przypisań nazywamy przydziałem godzinowym. Każda kolumna takiej macierzy ma 5 wartości i reprezentuje liczbę godzin lekcyjnych przeprowadzoną każdego dnia. Przykładowo kolumna $[0 \ 1 \ 0 \ 0 \ 0]^T$ reprezentuje jedną godzinę bloku we wtorek.

Dekompozycja prezentuje się następująco:

- Etap 1 — **Problem grupowania:**

Dane wejściowe: Zbiór wymagań głównych \mathcal{W} , zbiór definicji bloków przedmiotów \mathcal{B} oraz zasoby szkolne $\mathcal{T}, \mathcal{C}, \mathcal{S}$.

Wynik: Zbiór bloków lekcyjnych \mathcal{L} i wektor ich wymaganych godzin tygodniowo V .

Cel: Grupowanie pojedynczych wymagań głównych w możliwe do zrealizowania bloki lekcyjne.

- Etap 2 — **Problem alokacji:**

Dane wejściowe: Bloki lekcyjne \mathcal{L} , wektor godzin V , macierz dostępności nauczycieli A oraz zasoby szkolne \mathcal{T} i \mathcal{C} .

Wynik: Przydział godzinowy S określający liczbę godzin bloków każdego dnia.

Cel: Optymalny przydział godzin bloków lekcyjnych na pięć dni roboczych.

- Etap 3 — **Problem harmonogramowania:**

Dane wejściowe: Macierz S , bloki lekcyjne \mathcal{L} oraz zasoby sal \mathcal{R} .

Wynik: Zbiór przypisań \mathcal{Z} określających konkretne sloty czasowe i sale.

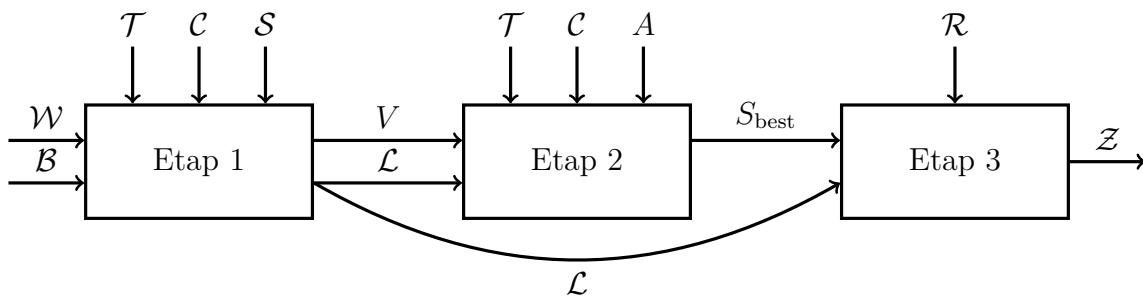
Cel: Przypisanie terminów i pomieszczeń dla wszystkich zajęć.

Pełne sformułowanie problemu harmonogramowania prowadziłoby do liczby zmiennych decyzyjnych proporcjonalnej do wyrażenia

$$|\mathcal{T}| \times |\mathcal{C}| \times |\mathcal{S}| \times 5 \times H \times |\mathcal{R}|$$

co w przypadku typowej szkoły skutkowałoby powstaniem nawet kilku milionów potencjalnych zmiennych. Tak duża skala zadania stanowi istotne wyzwanie obliczeniowe i uzasadnia konieczność zastosowania wieloetapowego podejścia.

Na rysunku 3.2 przedstawiłem diagram dekompozycji problemu:



Rysunek 3.2: Dekompozycja problemu

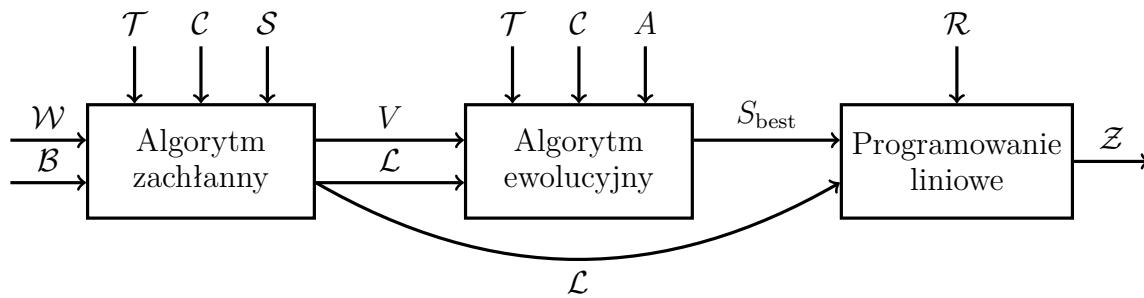
Dla każdego etapu dobrąłem technikę optymalizacyjną dopasowaną do specyfiki problemu.

Etap pierwszy charakteryzuje się niewielką przestrzenią decyzyjną oraz prostymi regułami tworzenia kombinacji wymagań głównych. Tworzenie bloków sprowadza się do grupowania wszystkich wymagań głównych, które spełniają kryteria opisane w blokach przedmiotów. Z uwagi na niską złożoność problemu oraz prostotę implementacyjną zdecydowałem się na użycie algorytmu zachłannego. Chociaż podejście to nie gwarantuje uzyskania optymalnych kombinacji, w kontekście niewielkiej przestrzeni decyzyjnej różnice w wynikach są minimalne.

Kontrastowo, drugi etap charakteryzuje się bardzo dużą przestrzenią decyzyjną — liczba możliwych kombinacji jest ogromna. Wymaga to zastosowania metody optymalizacyjnej zdolnej do równoległej ewaluacji wielu potencjalnych rozwiązań. Biorąc pod uwagę tę potrzebę oraz naturalną reprezentację danych wyjściowych w postaci macierzy, zdecydowałem się na użycie algorytmu ewolucyjnego. Podejście to upraszcza definiowanie funkcji celu (nie wymaga znajomości gradientu), zapewnia efektywne przeszukiwanie ogromnej liczby kombinacji i umożliwia bezpośrednie wykorzystanie formatu danych wyjściowych jako kodowania osobników.

Trzeci etap sprowadza się do problemu harmonogramowania, który jak omówiono wcześniej można przekształcić do postaci opisanej w podrozdziale 2.2. Choć istnieje wiele podejść rozwiązujących tego typu zadania, zdecydowałem się na zastosowanie programowania liniowego z ograniczeniami. Metoda ta nie wymaga projektowania własnego algorytmu rozwiązującego, co znacząco upraszcza proces implementacji, a jednocześnie gwarantuje wysoką jakość otrzymywanych harmonogramów. W szczególności wykorzystałem zmienne przedziałowe, które ułatwiają modelowanie złożonych zależności czasowych i ograniczeń między lekcjami.

Taki dobór technik tworzy hybrydowe podejście, które wykorzystuje mocne strony każdej metody, zapewniając kompromis między jakością rozwiązania a czasem obliczeń. Na rysunku 3.3 przedstawiłem strukturę algorytmu z wpisany odpowiednimi nazwami technik.



Rysunek 3.3: Struktura algorytmu z doborem technik

3.3.1. Problem grupowania

Bloki lekcyjne

Operując na blokach lekcyjnych dużo łatwiej zdefiniować ograniczenia fizyczne. Rozważmy trzy lekcje: język niemiecki, język francuski oraz język rosyjski. Gdybyśmy chcieli zdefiniować dla nich ograniczenie mówiące, że żaden uczeń nie może mieć w tym samym czasie dwóch lekcji, musielibyśmy osobno zadbać o to, aby żadna z tych lekcji nie nakładała się na lekcje obowiązkowe danej klasy, takie jak matematyka czy fizyka, a jednocześnie umożliwić nakładanie się lekcji między sobą, ponieważ każdy uczeń wybiera tylko jeden z tych trzech języków. Grupując takie wymagania w jeden blok trzech lekcji, możemy traktować go jak pojedynczą lekcję przy definiowaniu ograniczeń: blok nie może pokrywać się czasowo z innymi lekcjami, ale jego wewnętrzna struktura nie wymaga dodatkowych wyjątków.

Drugą zaletą podejścia blokowego jest uproszczenie modelowania lekcji przeprowadzanych jednocześnie dla wielu klas. Sytuacje takie są częste, szczególnie w przypadku przedmiotów o małej liczbie uczniów. Często w szkołach brakuje uczniów zapisanych na, przykładowo,

język rosyjski, aby uzasadnić indywidualne lekcje prowadzone przez nauczyciela z tylko i wyłącznie jedną klasą. Definiowanie ograniczeń dla każdej lekcji z osobna wymagałoby licznych wyjątków, natomiast połączenie wymagań głównych w jeden wspólny blok pozwala traktować taką lekcję jak każde inne zajęcia w systemie, bez potrzeby dodatkowych reguł.

Kolejną korzyścią jest możliwość łączenia bloków w struktury jeszcze wyższego poziomu. Wyobraźmy sobie trzy klasy: IIIA, IIIB i IIIC oraz trzy przedmioty: język niemiecki, francuski i rosyjski. Z powodu małej liczby uczniów zainteresowanych rosyjskim i francuskim, szkoła organizuje zajęcia w następujący sposób:

- trzy klasy uczestniczą wspólnie w lekcji języka rosyjskiego,
- IIIA i IIIB mają lekcję języka francuskiego razem, a klasa IIIC osobno z innym nauczycielem,
- każda klasa ma indywidualną lekcję języka niemieckiego, przy czym klasy IIIA i IIIC mają tego samego nauczyciela.

Z analizy wynika, że wszystkie zajęcia poza jedną lekcją języka niemieckiego mogą być prowadzone równocześnie. Tworząc wieloklasowe bloki lekcyjne, możemy je następnie łączyć w większe bloki: blok języka rosyjskiego można połączyć z blokiem języka francuskiego dla klas IIIA i IIIB oraz lekcją francuskiego klasy IIIC, a następnie dodać do tego dwie lekcje języka niemieckiego, pozostawiając jedynie trzecią lekcję poza blokiem.

Istnieją również lekcje, z których nie da się zbudować większych bloków. Przykładem są zajęcia, w których musi uczestniczyć cała klasa — takie jak język polski. W czasie ich trwania żadne inne grupowe lekcje nie mogą być prowadzone, ponieważ wszyscy uczniowie są obecni na tej samej lekcji. Na potrzeby dalszego etapu dekompozycji traktujemy takie zajęcia jako bloki jednostkowe — zbiory zawierające pojedyncze wymaganie główne.

Przestrzeń decyzyjna

Powyższe przykłady pokazują, jak niewielka jest przestrzeń decyzyjna w tym etapie. W praktyce jedyne decyzje, jakie muszą zostać podjęte, dotyczą sytuacji, w których możliwe jest łączenie bloków na różne sposoby. Przykładowo, jeśli łączymy lekcje języka niemieckiego dla kilku klas, musimy zdecydować, którą z lekcji dodać do tworzonego bloku.

Są to decyzje o marginalnym wpływie na wynik — potencjalny zysk z wyboru jednej opcji zamiast innej jest minimalny. Z tego powodu zastosowałem algorytm zachłanny, który zawsze wybiera pierwszą możliwą opcję prowadzącą do utworzenia jak największego bloku lekcyjnego.

Podobnie jak w pracy omówionej w podrozdziale 2.1, gdzie autorzy zmniejszali złożoność przez agregację ograniczeń, moje podejście wykorzystuje bloki lekcyjne do scalenia wielu ograniczeń, co istotnie redukuje przestrzeń decyzyjną na dalszych etapach rozwiązania.

Oznaczenia

Aby w pełni zrozumieć działanie algorytmu, należy formalnie zdefiniować pojęcie bloku lekcyjnego. Blok lekcyjny L to zbiór wymagań głównych realizowanych jednocześnie — w ramach jednej lub kilku lekcji. Zbiór takich bloków lekcyjnych \mathcal{L} definiujemy następująco:

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

Przykładowe interpretacje bloków lekcyjnych:

- 3 godziny języka angielskiego z klasą IA prowadzone przez nauczyciela angielskiego 1,
- 3 godziny języka angielskiego z klasą IA prowadzone przez nauczyciela angielskiego 2,

w przypadku różnych przedmiotów:

- 3 godziny języka angielskiego z klasą IA prowadzone przez nauczyciela angielskiego 1,
- 2 godziny informatyki z klasą IA prowadzone przez nauczyciela informatyki 1,

lub dla różnych klas:

- 1 godzina języka francuskiego z klasą IA prowadzona przez nauczyciela francuskiego 1,
- 1 godzina języka francuskiego z klasą IB prowadzona przez nauczyciela francuskiego 1,

Aby poprawnie modelować pracę z blokami lekcyjnymi, konieczne jest określenie liczby godzin przypisanych do każdego bloku. Każde wymaganie główne w_i posiada tygodniową liczbę godzin $g_{w,i}$, które należy rozdzielić pomiędzy bloki lekcyjne tak, aby suma godzin bloków pokrywających dane wymaganie była równa liczbie godzin wynikającej z wymagania głównego. Macierz przydziałów godzin do bloków prezentuje się następująco:

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

gdzie v_i oznacza liczbę godzin przypisanych do bloku L_i . Aby przydział godzin był poprawny, musi zostać spełniony następujący warunek zgodności z wymaganiami głównymi:

$$\forall w_i \in \mathcal{W} : \sum_{\substack{L_j \in \mathcal{L} \\ w_i \in L_j}} v_j = g_{w,i}$$

Funkcja celu

Celem pierwszego etapu jest maksymalizacja liczby bloków składających się z więcej niż jednego wymagania głównego oraz równe przedzielenie godzin pomiędzy te bloki. Jako że istnieje jeden równomierny przydział godzin, to nie uwzględniamy go w funkcji celu. Odpowiada temu funkcja celu:

$$F = |\{L \in \mathcal{L} : |L| > 1\}|$$

Algorytm zachłanny iteracyjnie przechodzi przez zbiór bloków przedmiotów \mathcal{B} tworząc wszystkie możliwe bloki lekcyjne spełniające kryteria opisane w tych blokach. W ten sposób gwarantujemy akceptowalną wartość funkcji celu.

3.3.2. Problem alokacji

Oznaczenia

Algorytm ma na celu alokację godzin lekcyjnych z macierzy V do pięciu roboczych dni tygodnia. Dla każdego bloku lekcyjnego L_i przydzielanych jest pięć wartości reprezentujących liczbę godzin lekcyjnych przydzielonych do poszczególnych dni, przy zachowaniu wymagań wynikających z poprzedniego etapu przetwarzania. Końcowy przydział godzin S definiujemy następująco:

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

gdzie $s_{d,i}$ oznacza liczbę godzin i -tego bloku lekcyjnego L_i przydzielonych do d -tego dnia tygodnia. Najlepszego osobnika na końcu działania algorytmu oznaczam przez S_{best} .

Najważniejszym wymaganiem jakie musi spełnić macierz S jest wymaganie co do liczby godzin zdefiniowanej w macierzy V :

$$\forall i \in \{1, 2, \dots, |\mathcal{L}|\} : \sum_{d=1}^5 s_{d,i} = v_i \quad (3.2)$$

Funkcja celu

Poniżej przedstawiam jedynie intuicyjny sens funkcji celu. Jej pełne omówienie zaprezentuję w sekcji dotyczącej funkcji przystosowania osobników w algorytmie ewolucyjnym.

Aby ocenić jakość wygenerowanego rozwiązania, wprowadzamy pojęcie obciążenia dydaktycznego dla klasy (lub analogicznie dla nauczyciela). Obciążenie to opisuje liczbę godzin zajęć przypadających na każdy z dni tygodnia.

Niech $s_{d,i}$ oznacza liczbę godzin bloku L_i zaplanowaną w dniu d . Wówczas obciążenie dydaktyczne klasy c definiujemy jako wektor:

$$\boldsymbol{\beta}_c = \begin{bmatrix} \beta_{c,1} \\ \beta_{c,2} \\ \vdots \\ \beta_{c,5} \end{bmatrix}, \quad \beta_{c,d} = \sum_{\substack{L_i \in \mathcal{L} \\ c \in \{c_{w,j} : w_j \in L_i\}}} s_{d,i}$$

czyli sumę wszystkich godzin bloków, w których uczestniczy klasa C , przypadających na dzień d .

Celem jest równomierne rozłożenie zajęć między dni tygodnia, tak aby uniknąć bardzo krótkich oraz bardzo przeładowanych dni. Miarą nierównomierności jest różnica między dniem najbardziej obciążonym a dniem najmniej obciążonym:

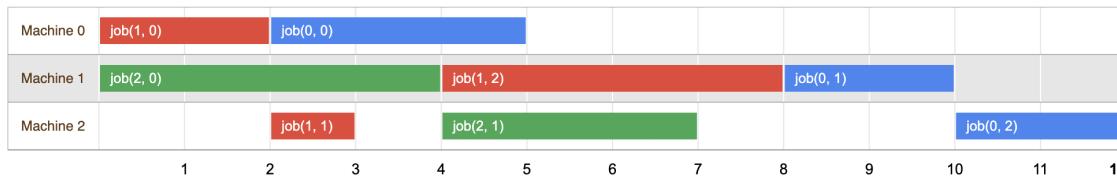
$$F = \max_{d \in \{1, 2, \dots, 5\}} \beta_{c,d} - \min_{d \in \{1, 2, \dots, 5\}} \beta_{c,d} \quad (3.3)$$

Analogicznie można definiować funkcje celu dla nauczycieli.

3.3.3. Problem harmonogramowania

Trzeci etap, podobnie jak poprzedni, cechuje się dużą przestrzenią decyzyjną. Istnieje wiele wykonalnych planów, z których część powoduje liczne okienka w harmonogramach nauczycieli. Z tego powodu zdecydowałem się zastosować programowanie liniowe z ograniczeniami (ang. *Constraint Programming*, CP), które w praktyce dostarcza wysokiej jakości rozwiązania bez potrzeby projektowania własnego algorytmu przeszukiwania.

Aby zapewnić poprawność i przejrzystość modelu, sformułowałem problem harmonogramowania jako specyficzny wariant JSSP z wykorzystaniem zmiennych przedziałowych. Na rysunku 3.4 przedstawiono przykładową wizualizację rozwiązania JSSP.



Rysunek 3.4: Wizualizacja rozwiązania klasycznego problemu JSSP z wykorzystaniem zmiennych przedziałowych [10]

Podejście oparte na przedziałach oferuje większą elastyczność niż model binarny z podrozdziału 3.2.1, a jednocześnie upraszcza definicję kluczowych ograniczeń.

Dzięki temu, że w etapie pierwszym utworzyłem bloki lekcyjne, które nie nachodzą na siebie, mogę zdefiniować zadania z prostymi ograniczeniami nienakładania. Dodatkowo, dzięki drugiemu etapowi układam plan niezależnie dla każdego dnia tygodnia $d \in \{1, 2, 3, 4, 5\}$, reprezentowanego przez wiersze macierzy S_{best} . Dla każdego dnia rozwiążę podproblem zawierający tylko te bloki L_i , dla których $S_{\text{best},d,i} > 0$.

Takie podejście:

- Redukuje wymagania pamięciowe około pięciokrotnie.
- Umożliwia równoległe przetwarzanie dni, o ile nie ogranicza nas pamięć.
- Upraszczają strukturę ograniczeń, ponieważ nie ma zależności między dniami.

Zmienne decyzyjne i ograniczenia

Wszystkie zmienne definiuję osobno dla każdego dnia $d \in \{1, 2, \dots, 5\}$. Dla każdego bloku lekcyjnego L_i , dla którego $s_{d,i} > 0$, wprowadzam zmienną przedziałową

$$I_i = (\mathbf{s}_i, s_{d,i}, \mathbf{e}_i), \quad \mathbf{e}_i = \mathbf{s}_i + s_{d,i}, \quad \mathbf{s}_i, \mathbf{e}_i \in [0, H]$$

gdzie \mathbf{s}_i i \mathbf{e}_i oznaczają odpowiednio slot rozpoczęcia i zakończenia przedziału dla bloku L_i .

Ograniczenia nienakładania się, implementuję przy użyciu funkcji `AddNoOverlap` [9]. Funkcja przyjmuje zbiór przedziałów i generuje wszystkie potrzebne ograniczenia bez konieczności ręcznego ich definiowania. Operację tę wykonuję niezależnie dla zbiorów przedziałów powiązanych z każdą klasą i każdym nauczycielem.

Kolejną zaletą przedziałów jest możliwość użycia ich wariantu opcjonalnego do przypisywania sal. Dla każdej dopuszczalnej kombinacji (blok L_i , nauczyciel t , przedmiot s , sala r) definiuję opcjonalny przedział

$$O_{i,t,s,r} = (\mathbf{s}_i, s_{d,i}, \mathbf{e}_i, p_{i,t,s,r}), \quad p_{i,t,s,r} \in \{0, 1\}$$

gdzie rozważam wyłącznie kombinacje spełniające wymagania (np. blok z matematyką, nauczyciel matematyki, matematyka, sala do matematyki), co eliminuje niemożliwe kombinacje (analogicznie jak w podejściu z podrozdziału 2.1). Ograniczenia nienakładania dla sal obowiązują tylko wtedy, gdy $p_{i,s,t,r} = 1$.

W praktyce musimy narzucić, aby suma odpowiednich zmiennych $p_{i,t,s,r}$ była równa wymaganej liczbie sal dla danego bloku. Przykładowo, jeśli blok L_1 zawiera jedno wymaganie główne z przedmiotu s_1 prowadzone przez nauczyciela t_1 , a szkoła dysponuje pięcioma salami obsługującymi ten przedmiot r_1, r_2, \dots, r_5 , to

$$\sum_{i=1}^5 p_{1,t_1,s_1,r_i} = 1$$

Ograniczenie to zapewnia, że dokładnie jedna sala matematyki zostanie przypisana do danego bloku. Sale, które nie obsługują matematyki, nie mogą zostać przypisane, ponieważ nie definiuję dla nich zmiennych decyzyjnych, co także eliminuje niemożliwe kombinacje.

Analogicznie do zwykłych przedziałów, nienakładanie dla sal tworzę poprzez `AddNoOverlap`, przekazując zbiór wszystkich opcjonalnych przedziałów dla danej sali i powtarzając tę operację dla każdego $r \in \mathcal{R}$.

Pozostałe ograniczenia omawiam szczegółowo w kolejnych sekcjach.

Funkcja celu

Celem tej fazy jest minimalizacja całkowitego czasu przebywania nauczycieli w szkole. Wprowadzam pomocnicze zmienne

$$\forall t \in \mathcal{T} : \begin{cases} \tau_{\text{start},t} &= \min\{\mathbf{s}_i : i \in \{1, \dots, |L|\} \wedge t \in T_i\} \\ \tau_{\text{end},t} &= \max\{\mathbf{e}_i : i \in \{1, \dots, |L|\} \wedge t \in T_i\} \\ \Delta_t &= \tau_{\text{end},t} - \tau_{\text{start},t} \end{cases}$$

Wielkość Δ_t reprezentuje czas obecności nauczyciela w szkole (od początku pierwszego do końca ostatniego przedziału). Minimalizuję sumę tych czasów

$$F = \sum_{t \in \mathcal{T}} \Delta_t$$

co prowadzi do kompaktowego układania zajęć i redukcji okienek w planach nauczycieli.

3.4. Algorytm zachłanny

Algorytm zachłanny (ang. *greedy*) polega na iteracyjnym podejmowaniu lokalnie najlepszej decyzji bez cofania się i bez globalnego przeszukiwania przestrzeni rozwiązań [5].

3.4.1. Działanie

Klasyfikacja bloków

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

$$\mathcal{B}_{\text{agg}} = \{b \in \mathcal{B} : \gamma_b = 1\}$$

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

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

- **Bloki pojedyncze** — obejmują pojedyncze klasy

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

Procedura

Procedurę generowania bloków można podzielić na parę faz:

- **Tworzenie grup wieloklasowych** — grupowanie wymagań głównych dotyczących tego samego przedmiotu dla różnych klas;
przykładowo język rosyjski dla klasy IIIA, IIIC, IID.
- **Agregacja bloków** — łączenie powstałych bloków międzyklasowych z innymi wymaganiami, które mogą odbywać się równolegle;
przykładowo wyżej wspomniany język rosyjski oraz język niemiecki dla klasy IIIA.
- **Tworzenie bloków pojedynczych** — formowanie bloków wewnętrz pojedynczych klas;
przykładowo podział wychowania fizycznego na grupy chłopięce i dziewczęce.
- **Tworzenie bloków jednostkowych** — utworzenie jednoelementowych zbiorów dla pozostałych wymagań, niezgrupowanych w poprzednich fazach;
przykładowo lekcja języka polskiego w której musi uczestniczyć cała klasa.
- **Przydział godzin lekcyjnych do bloków** — rozdzielenie godzin z wymagań głównych do bloków tak, aby każdy blok w obrębie klasy dostał przynajmniej jedną godzinę.

Szczegółowy opis działania przedstawiłem w pseudokodzie [3.1](#).

Pseudokod 3.1: Generowanie bloków lekcyjnych z wymagań

Wejście : Zbiory wymagań \mathcal{W} , bloków $\mathcal{B}_{\text{single}}$, $\mathcal{B}_{\text{multi}}$, \mathcal{B}_{agg}

Wyjście : Zbiór bloków lekcyjnych \mathcal{L} , wektor godzin V

- 1 $\mathcal{L} \leftarrow \emptyset, V \leftarrow \emptyset, \forall w \in \mathcal{W} : U(w) \leftarrow 0;$
- 2 **Dla** każdego bloku $b \in \mathcal{B}_{\text{multi}}$ **wykonaj:**
- 3 $\mathcal{W}_b \leftarrow \{w \in \mathcal{W} : s_w \in \vec{S}_b \wedge c_w \in \vec{C}_b\}, \quad \forall s \in \vec{S}_b : |\{w \in \mathcal{W}_b : s_w = s\}| = g_b ;$
- 4 **Dla** każdego bloku $b \in \mathcal{B}_{\text{agg}}$ **wykonaj:**
- 5 Tworzymy maksymalnie duży zbiór $\mathcal{W}_{\text{agg}} \subset \mathcal{W}$:
$$\begin{cases} c_{w,i} \in \vec{C}_b & \forall w_i \in \mathcal{W}_{\text{agg}} \\ s_{w,i} \in \vec{S}_b & \forall w_i \in \mathcal{W}_{\text{agg}} \\ U(w_i) = 0 & \forall w_i \in \mathcal{W}_{\text{agg}} \end{cases}$$

oraz nie występują kolizje nauczycieli ;
- 6 $\mathcal{L} \leftarrow \mathcal{L} \cup \{\mathcal{W}_{\text{agg}}\} ;$
- 7 $V \leftarrow V \cup \{\min\{g_w - U(w) : w \in \mathcal{W}_{\text{agg}}\}\} ;$
- 8 **Dla** $w \in \mathcal{W}_{\text{agg}}$ **wykonaj:**
- 9 $U(w) \leftarrow U(w) + v ;$
- 10 **Dla** każdego bloku $b \in \mathcal{B}_{\text{multi}}$ **wykonaj:**
- 11 **Jeśli** $\forall w \in \mathcal{W}_b : U(w) < g_w$ **to:**
- 12 $\mathcal{L} \leftarrow \mathcal{L} \cup \{\mathcal{W}_b\} ;$
- 13 $V \leftarrow V \cup \{\min\{g_w - U(w) : w \in \mathcal{W}_b\}\} ;$
- 14 **Dla** $w \in \mathcal{W}_b$ **wykonaj:**
- 15 $U(w) \leftarrow U(w) + v ;$
- 16 **Dla** każdego bloku $b \in \mathcal{B}_{\text{single}}$ **wykonaj:**
- 17 **Dla** każdej klasy $c \in \vec{C}_b$ **wykonaj:**
- 18 $\mathcal{W}_c \leftarrow \{w : c_w = c \wedge s_w \in \vec{S}_b \wedge w \in \mathcal{W}\} ;$
- 19 **Jeśli** \mathcal{W}_c spełnia warunki bloku co do liczby przedmiotów **to:**
- 20 $\mathcal{L} \leftarrow \mathcal{L} \cup \{\mathcal{W}_c\} ;$
- 21 **W przeciwnym przypadku:**
- 22 **Dla** każdej poprawnej kombinacji $\mathcal{W}'_c \subseteq \mathcal{W}_c$ **wykonaj:**
- 23 $\mathcal{L} \leftarrow \mathcal{L} \cup \{\mathcal{W}'_c\} ;$
- 24 **Dopóki** \exists nie w pełni wykorzystane wymagania w blokach pojedynczych **wykonuj:**
- 25 **Dla** każdego bloku pojedynczego $\mathcal{W}' \in \mathcal{L}$ **wykonaj:**
- 26 **if** można zwiększyć liczbę godzin **then**
- 27 Zwiększą wartość v odpowiadającą blokowi \mathcal{W}' o 1 ;
- 28 **Dla** każdego $w \in \mathcal{W}'$ **wykonaj:**
- 29 $U(w) \leftarrow U(w) + 1 ;$
- 30 **Dla** $w \in \mathcal{W}$ **wykonaj:**
- 31 **if** $U(w) < g_w$ **then**
- 32 $\mathcal{L} \leftarrow \mathcal{L} \cup \{\{w\}\}, V \leftarrow V \cup \{g_w - U(w)\} ;$
- 33 **Zwróć** \mathcal{L}, V

3.4.2. Przykładowy wynik

Dla danych udostępnionych przez szkołę liczba wymagań głównych wynosi $|\mathcal{W}| = 378$, natomiast liczba zdefiniowanych bloków przedmiotów $|\mathcal{B}| = 47$. Zastosowany algorytm zachłanny wygenerował $|\mathcal{L}| = 302$ bloków lekcyjnych. Średni czas wykonywania dla pięciu niezależnych uruchomień wyniósł 0.84s, przy czym pomiar czasu obejmował również czas pobierania danych z bazy.

Poniżej fragment otrzymanego zbioru bloków oraz odpowiadających wartości macierzy V :

$$\mathcal{L} = \left\{ \begin{array}{l} (\{\text{Jn1, IV_A, J.Niem, 2}\}, \{\text{Jn1, IV_F, J.Niem, 2}\}, \dots) \\ (\{\text{Jr1, IV_A, J.Ros, 2}\}, \{\text{Jr1, IV_G, J.Ros, 2}\}, \dots) \\ (\{\text{Jf2, IV_D, J.Fran, 2}\}, \{\text{Jf1, IV_F, J.Fran, 2}\}, \dots), \dots, \\ (\{\text{Ja5, I_A, J.Ang, 2}\}, \{\text{Inf2, I_A, Informatyka, 2}\}), \dots, \\ (\{\text{Chem1, I_G, Chemia, 1}\}), \dots \end{array} \right\}$$

$$V = \begin{bmatrix} 2 & \dots & 1 & \dots & 1 & \dots \end{bmatrix}$$

W wynikach możemy zaobserwować działanie algorytmu. Pierwszym elementem w \mathcal{L} jest komparatywnie duży, wieloklasowy blok lekcyjny powstały w fazie agregacji. Obejmuje on między innymi:

- Lekcję j. niemieckiego z klasami IVA oraz IVF prowadzoną przez nauczyciela Jn1,
- Lekcję j. niemieckiego z klasami IVC oraz IVG prowadzoną przez nauczyciela Jn2,
- Lekcję j. rosyjskiego z klasami IVA, IVC, IVD, IVF oraz IVG przez nauczyciela Jr1,
- Lekcję j. francuskiego z klasą IVA oraz IVF prowadzoną przez nauczyciela Jf1,
- Lekcję j. francuskiego z klasą IVD prowadzoną przez nauczyciela Jf2.

Dla tego bloku wartość v_1 wynosi:

$$v_1 = \min_{\forall w_i \in L_1} g_{w,i} - U(w) = 2$$

ponieważ na początku wszystkie wymagania mają $U(w) = 0$.

Drugim typem bloków są bloki pojedyncze. Jeden z przykładów widocznych w wynikach obejmuje:

- Lekcję j. angielskiego z klasą IA prowadzoną przez nauczyciela Ja5
- Lekcję informatyki z klasą IA prowadzoną przez nauczyciela Inf2

W tym przypadku $v = 1$, ponieważ klasa posiada dwie grupy języka angielskiego oraz dwie grupy informatyki, a pełne wymaganie informatyki musi zostać rozdzielone między odpowiadające mu bloki. Obrazuje to konieczność iteracyjnego przydzielania wymaganych godzin do bloków — gdybyśmy chcieli przypisać obie godziny do pierwszego bloku to druga grupa informatyki nie miałaby możliwości odbycia lekcji.

Ostatnią kategorię stanowią bloki jednostkowe, przykładowo lekcja chemii dla klasy IG. Bloki takie pojawiają się w przypadku wymagań, które nie mogą być dołączone do żadnego większego bloku.

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 unikniecie wczesnego utknięcia na lokalnych wierzchołkach funkcji celu.

Ogólna struktura algorytmu:

Pseudokod 3.2: Ogólny schemat algorytmu ewolucyjnego [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 przez losowe generowanie kolumn zgodnie z dostępnością nauczycielami i wymaganiami bloków.
- EVALUATE oblicza wartości funkcji przystosowania dla każdego osobnika.
- SELECT parents realizuje selekcję ruletkową.
- RECOMBINE tworzy potomków przez kolumnowe dziedziczenie przydziału bloków.
- MUTATE wprowadza różnorodność przez rekonstrukcję wybranych kolumn.
- SELECT implementuje elitaryzm i utrzymuje stały rozmiar populacji.
- TERMINATION CONDITION kończy proces po ustalonej liczbie generacji.

3.5.1. Kodowanie i ograniczenia twardé

Każdy osobnik w populacji jest reprezentowany przez macierz przydziałów S przyjętą w definicji oznaczenia (3.1).

Tak przyjęta reprezentacja umożliwia prostą weryfikację następujących ograniczeń:

- suma przypisań i -tego bloku lekcyjnego musi być równa v_i . Zostało to opisane w warunku (3.2),
- maksymalna liczba godzin każdego bloku w pojedynczym dniu nie może być większa niż 2,

$$\forall i \in \{1, 2, \dots, |\mathcal{L}|\}, \forall d \in \{1, 2, 3, 4, 5\} : s_{d,i} \leq 2$$

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

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

3.5.2. 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 etapów wykonywanych dla każdego i -tego bloku przedstawionych w pseudokodzie 3.3.

Pseudokod 3.3: Generowanie rozkładu zajęć dla bloku i

Wejście : $v_i, a_{t,d}, T_i$

Wyjście : Macierz przydziałów $X_i = [x_1, x_2, x_3, x_4, x_5]$

1 Dla $d \in \{1, 2, \dots, 5\}$ wykonaj:

2 $\alpha_d \leftarrow \min_{t \in T_i} a_{t,d} ;$

3 Jeśli $v_i = 3$ to:

4 $X_i \leftarrow [0, 0, 0, 0, 0] ;$

5 Wylosuj dni k, l spośród tych, dla których $\alpha_k = \alpha_l = 1$;

6 $x_k \leftarrow 2, x_l \leftarrow 1 ;$

7 Zwróć X_i ;

8 W przeciwnym przypadku:

9 $P \leftarrow \left[\frac{\alpha_1}{\sum \alpha_d}, \dots, \frac{\alpha_5}{\sum \alpha_d} \right] ;$

10 $X_i \leftarrow \text{numpy.random.multinomial}(v_i, P) ;$

11 Dopóki $\exists d : x_d > 2$ wykonuj:

12 Dla $\forall d \in \{1, 2, \dots, 5\} : x_d > 2$ wykonaj:

13 $X'_i \leftarrow \text{numpy.random.multinomial}(x_d - 2, P) ;$

14 $x_d \leftarrow 2 ;$

15 $X_i \leftarrow X_i + X'_i ;$

16 Zwróć X_i

Przez $T_i = \{t_{w,j} : w_j \in L_i\}$ oznaczono zbiór nauczycieli prowadzących zajęcia w bloku L_i .

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

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

Osobnik populacji reprezentowany jest przez transpozycję tej macierzy $S = \mathbf{X}^T$.

3.5.3. Przystosowanie

Ocena jakości osobników odbywa się na podstawie dwóch wzkaźników:

- wskaźnika równomierności obciążeń dydaktycznych lekcyjnych klas,
- wskaźnika równomierności obciążień dydaktycznych pracy nauczycieli.

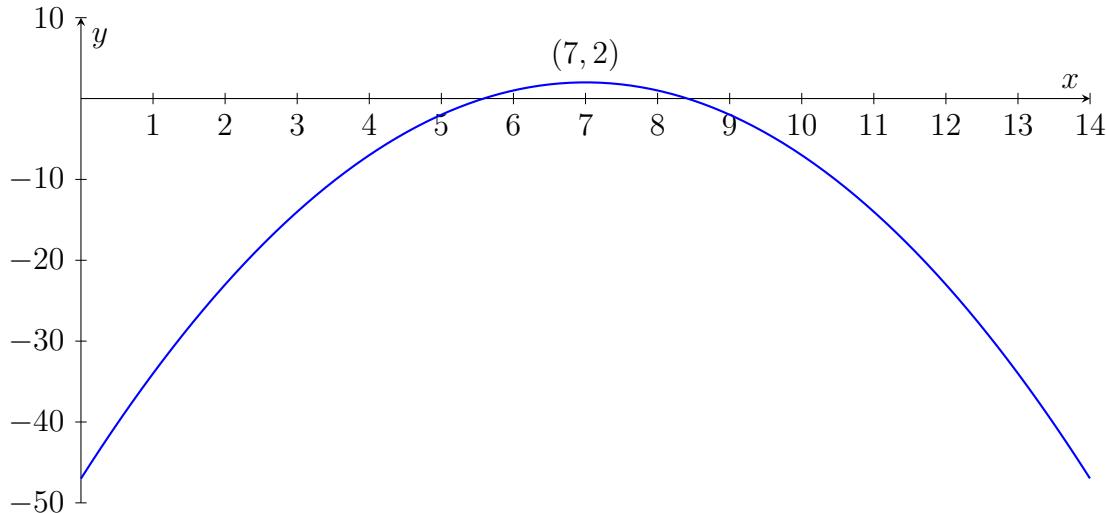
Aby móc niezależnie oceniać równomierność obciążenia dydaktycznego w poszczególnych dniach tygodnia, konieczne było zaprojektowanie funkcji, która jako argument przyjmuje dzienne wartości obciążień $\beta_{c,d}$ dla klas oraz $\beta_{t,d}$ dla nauczycieli.

W początkowej fazie prac rozważyłem zastosowanie funkcji gęstości rozkładu normalnego o wartości oczekiwanej $\mu = 8$, odpowiadającej ośmiogodzinnemu dniowi pracy lub nauki. Szybko okazało się jednak, że funkcja ta jedynie nagradza dobre rozwiązania, lecz nigdy nie karze wyraźnie rozkładów niepożądanych. W rezultacie algorytm tracił możliwość korygowania błędów, gdyż nawet bardzo nierównomierne rozkłady nie były wystarczająco penalizowane.

W odpowiedzi na te wyzwania zaprojektowałem funkcję kwadratową przedstawioną na rysunku 3.5:

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

która silnie karze zarówno zbyt małe, jak i zbyt duże obciążenia oraz nagradza wartości zbliżone do $x = 7$. Wierzchołek funkcji został przesunięty z 8 na 7, ponieważ pięciogodzinny dzień pracy jest znacznie bardziej akceptowalny niż dzień dziesięciogodzinny. Nawet w przypadkach, gdy liczba godzin zajęć przekracza 35 tygodniowo (co uniemożliwia osiągnięcie maksymalnej wartości funkcji przy równomiernym rozkładzie), funkcja zachowuje zdolność odróżniania rozkładów bardziej wyrównanych od tych skrajnie nierównych. Wynika to z faktu, że pochodna funkcji na przedziale $[7, \infty)$ maleje, przez co rozwiązania z jednym dniem jedenastogodzinnym oceniane są gorzej niż rozwiązania z dwoma dniami dziewięciogodzinnymi.



Rysunek 3.5: Wykres funkcji użytej do ewaluacji rozkładu godzin nauczycieli i klas na przestrzeni tygodnia.

Ocena względem nauczycieli

Dla każdego nauczyciela t tworzony jest wektor dziennego obciążenia dydaktycznego β_t .

Zastosowanie funkcji $f(x)$ bez modyfikacji powodowałoby silne karanie dni wolnych ($x = 0$), co jest niepożądane — nauczyciel może mieć dzień wolny i nie powinno to stanowić wady planu. Aby temu zapobiec, zmodyfikowałem funkcję w następujący sposób:

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

Ocena dla nauczyciela t to suma wartości funkcji dla wszystkich dni tygodnia:

$$F_t = \sum_{d=1}^5 f_T(\beta_{t,d})$$

Ocena wszystkich nauczycieli jest równa:

$$F_T = \sum_{t \in \mathcal{T}} F_t$$

Ocena względem klas

Proces jest analogiczny, przy czym dla klas dni wolne są niepożądane, więc nie stosuję dodatkowych modyfikacji funkcji.

Oceny mają postać:

$$\begin{aligned} F_c &= \sum_{d=1}^5 f(\beta_{c,d}) \\ F_C &= \sum_{c \in \mathcal{C}} F_c \end{aligned}$$

Normalizacja i wagi

Wyznaczanie końcowej wartości funkcji przystosowania wymaga uwzględnienia stosunku liczby nauczycieli do liczby uczniów. W typowych placówkach oświatowych liczba nauczycieli przywyjsza liczbę klas. Bezpośrednie dodawanie ocen $F_T + F_C$ 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 obciążień nauczycieli. Aby umożliwić sterowanie tymi preferencjami należy zaimplementować wagi, co prowadzi do następującej postaci funkcji przystosowania dla i -tego osobnika:

$$F_i = \frac{\theta_T}{|\mathcal{T}|} F_{T,i} + \frac{\theta_C}{|\mathcal{C}|} F_{C,i}, \quad \theta_T, \theta_C \in \mathbb{R}^+$$

gdzie θ_T to waga oceny nauczycieli, a θ_C to waga oceny klas.

3.5.4. Selekcja

Zastosowałem tradycyjną metodę ruletkową. Polega ona na losowaniu osobników, którzy posługują jako rodzice z prawdopodobieństwami, które są proporcjonalne do wartości ich funkcji przystosowania. Mając wektor ocen $F_{c\text{ total}} = [F_1 \ F_2 \ \dots \ F_{\frac{P}{2}}]$, gdzie $\frac{P}{2}$ 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:

$$P = [\sigma(F_1) \ \sigma(F_2) \ \dots \ \sigma(F_{\frac{P}{2}})], \quad \sigma(F_i) = \frac{\exp[F_i]}{\sum_{j=1}^{\frac{P}{2}} \exp[F_j]}$$

W celu zamiany funkcji przystosowania na prawdopodobieństwa użyłem funkcji SoftMax.

3.5.5. 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 przystosowania zdecydowałem się na zaimplementowanie dwóch niezależnych metod krzyżowania osobników widocznych w pseudokodach 3.4 oraz 3.5. Takie podejście umożliwia równoczesne dziedziczenie cech związanych z optymalnym obciążeniem dydaktycznym zarówno nauczycieli, jak i klas. Pozwala to także na stworzenie dwóch różnych potomków z dwóch rodziców. Wygenerowany w ten sposób osobniki S' mają cechy, które gwarantują jak najlepsze przypisanie nauczycieli lub klas wśród obu rodziców. Nastecną 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.

Pseudokod 3.4: Krzyżowanie uwzględniające ocenę obciążenia dydaktycznego nauczycieli

Wejście : S_1, S_2 , oraz $F_{t,1}, F_{t,2}$ dla wszystkich nauczycieli $t \in \mathcal{T}$

Wyjście : Macierz potomka S'

- 1 $S' \leftarrow 0_{5 \times |\mathcal{L}|}$;
- 2 **Dla** $t \in \mathcal{T}$ **w wykonaj:**
 - 3 **Jeśli** $F_{t,1} > F_{t,2}$ **to:**
 - 4 **Dla** każdego bloku L_i prowadzonego przez nauczyciela t **w wykonaj:**
 - 5 **Dla** $d \in \{1, 2, \dots, 5\}$ **w wykonaj:**
 - 6 $S'_{d,i} \leftarrow S_{1,d,i}$;
 - 7 **W przeciwnym przypadku:**
 - 8 **Dla** każdego bloku L_i prowadzonego przez nauczyciela t **w wykonaj:**
 - 9 **Dla** $d \in \{1, 2, \dots, 5\}$ **w wykonaj:**
 - 10 $S'_{d,i} \leftarrow S_{2,d,i}$;
 - 11 **Zwróć** S'

Pseudokod 3.5: Krzyżowanie uwzględniające ocenę obciążenia dydaktycznego klas

Wejście : S_1, S_2 , oraz $F_{c,1}, F_{c,2}$ dla wszystkich klas $c \in \mathcal{C}$

Wyjście : Macierz potomka S'

1 $S' \leftarrow \mathbf{0}_{5 \times |\mathcal{L}|}$;

2 Dla $t \in \mathcal{T}$ wykonaj:

3 | Jeśli $F_{c,1} > F_{c,2}$ to:

4 | | Dla każdego bloku L_i dla klasy c wykonaj:

5 | | | Dla $d \in \{1, 2, \dots, 5\}$ wykonaj:

6 | | | | $S'_{d,i} \leftarrow S_{1,d,i}$;

7 | W przeciwnym przypadku:

8 | | Dla każdego bloku L_i prowadzonego przez nauczyciela t wykonaj:

9 | | | Dla $d \in \{1, 2, \dots, 5\}$ wykonaj:

10 | | | | $S'_{d,i} \leftarrow S_{2,d,i}$;

11 Zwróć S'

3.5.6. 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. Pozwala 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 punkcie 3.5.2.

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 osobnik o największej wartości funkcji przystosowania:

$$S_{\text{best}} = S_i, \quad i = \arg \max_{j \in \{1, 2, \dots, |\mathcal{P}|\}} F_j$$

3.5.7. Przykładowy wynik

Dla danych wejściowych przedstawionych w poprzednim przykładzie, $\mathfrak{P} = 1000$, $\theta_T = 1$ oraz $\theta_G = 2$ algorytm wygenerował następującego najlepszego osobnika:

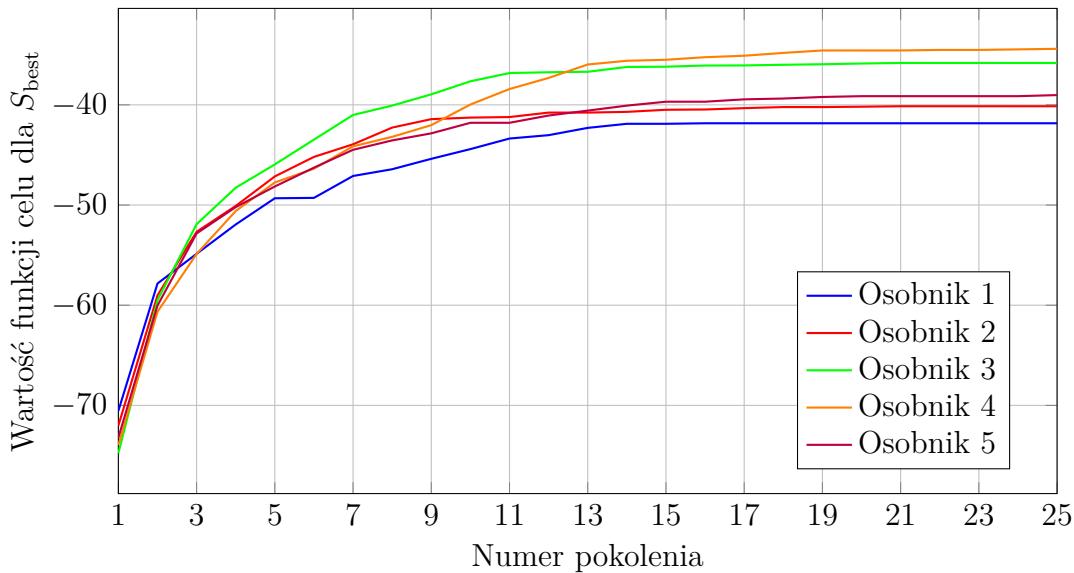
$$S_{\text{best}} = \begin{bmatrix} 1 & \dots & 0 & \dots & 0 & \dots \\ 0 & \dots & 1 & \dots & 0 & \dots \\ 0 & \dots & 0 & \dots & 1 & \dots \\ 0 & \dots & 0 & \dots & 0 & \dots \\ 1 & \dots & 0 & \dots & 0 & \dots \end{bmatrix}$$

Interpretacja macierzy przydziału jest następująca:

- blok językowy (niemiecki, francuski, rosyjski) został umieszczony po jednej godzinie w poniedziałek oraz piątek,
- jedna godzina wspólnego bloku informatyki i języka angielskiego została przydzielona do wtorku,
- jedna godzina chemii została przypisana do środy.

Na rysunku 3.6 przedstawiłem wartości funkcji celu dla pięciu niezależnych uruchomień algorytmu genetycznego dla 25 pokoleń, przy następujących parametrach:

- $\mathfrak{P} = 1000$,
- $\theta_T = 1$,
- $\theta_G = 2$.



Rysunek 3.6: Wykres wartości funkcji celu dla pięciu uruchomień algorytmu genetycznego

W żadnym przypadku funkcja celu nie osiągnęła wartości dodatnich. Wynika to z faktu, że w danych wejściowych występują klasy oraz nauczyciele, którzy mają liczbę bloków znacząco mniejszą niż 35. W praktyce uniemożliwia to uzyskiwanie dodatnich wartości funkcji celu, ponieważ algorytm nie ma wystarczającej liczby bloków, by spełnić wszystkie wymagania strukturalne konieczne do uzyskania dodatniego wyniku.

3.6. Solver programowania liniowego z ograniczeniami

Programowanie z ograniczeniami (ang. *Constraint Programming*, CP) jest sposobem modelowania i rozwiązywania problemów kombinatorycznych, w którym rozwiązanie opisuje się za pomocą zmiennych decyzyjnych, dziedzin tych zmiennych oraz ograniczeń wiążących je ze sobą. Zamiast tradycyjnego definiowania funkcji w postaci algebraicznej, jak w klasycznym programowaniu liniowym, w CP kluczową rolę odgrywają relacje logiczne i strukturalne, takie jak nienakładanie się zdarzeń, zależności kolejnościowe czy ograniczenia zasobów.

Solver CP rozwiązuje problem poprzez naprzemienne stosowanie dwóch mechanizmów:

- propagacji ograniczeń, która zawęża dziedziny zmiennych poprzez analizę warunków narzuconych w modelu,
- przeszukiwania przestrzeni rozwiązań, w którym solver wybiera zmienną, przypisuje jej wartość i rekurencyjnie kontynuuje proces, stosując strategię gałęzi i ograniczeń.

3.6.1. Zmienne decyzyjne

Zmienne decyzyjne są definiowane zgodnie z definicjami przedstawionymi w punkcie 3.3.3.

3.6.2. Ograniczenia

Każde z poniższych ograniczeń nakładamy niezależnie dla każdego dnia $d \in \{1, 2, \dots, 5\}$.

Ograniczenie przypisania sal

Dla każdego bloku L_i musi być przypisana odpowiednia liczba sal. W tym celu tworzymy następującą funkcję pomocniczą dla każdego bloku i :

$$f_i(s) = |\{t_{w,j} : s = s_{w,j} \wedge w_j \in L_i\}|$$

która zwraca liczbę nauczycieli prowadzących przedmiot s w bloku L_i .

$$\forall L_i \in \mathcal{L}, s \in S_i : \sum_{r \in \mathcal{R}, t \in \mathcal{T}} p_{L_i, s, t, r} = f_i(s)$$

gdzie $S_i = \{s_{w,j} : w_j \in L_i\}$ to zbiór przedmiotów prowadzonych w bloku L_i . Z uwagi na to, że zmienne decyzyjne są tworzone tylko i wyłącznie dla poprawnych kombinacji gwarantuje to tym samym przypisanie poprawnych sal.

Ciągłość zajęć dla klas

Dla każdej klasy $c \in \mathcal{C}$:

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

Nienakładanie się zajęć [9]:

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

Brak okienek:

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

Nakładanie się nauczycieli

Dla każdego nauczyciela $t \in \mathcal{T}$:

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

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

Kolizje sal

Dla każdej sali $r \in \mathcal{R}$:

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

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

Kolejność przedmiotów

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

$$\forall i \in \{1, 2, \dots, |\mathcal{L}|\} : s_{\text{best},d,i} = 2 \implies e_i - s_i = 2$$

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

3.6.3. Funkcja celu

Funkcja celu jest definiowana zgodnie z definicją przedstawioną w sekcji 3.3.3.

3.6.4. Transformacja przedziałów na przypisania końcowe

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

Ekstrakcja rozwiązań z solvera

Dla każdego przedziału I_i w rozwiązaniu zapisujemy następujące wartości:

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

Odwzorowanie na strukturę lekcji

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

$$f : T_i \rightarrow \mathcal{R}_i$$

$\forall t \in T_i : f(t) = r \in \mathcal{R}_i$ takie, że r obsługuje przedmiot prowadzony przez nauczyciela t

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 $h \in [1, e_i - s_i]$:

$$\begin{aligned} z &\leftarrow (d, h, c_w, t_w, s_w, f(t_w)) \\ \mathcal{Z}_d &\leftarrow \mathcal{Z}_d \cup \{z\} \end{aligned}$$

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.

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:

$$\begin{aligned}z_1 &= (1, 2, c_A, t_1, s_1, r_1) \\z_2 &= (1, 3, c_A, t_1, s_1, r_1) \\z_3 &= (1, 2, c_A, t_2, s_2, r_2) \\z_4 &= (1, 3, c_A, t_2, s_2, r_2)\end{aligned}$$

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

3.6.5. Przykładowy wynik

Dla danych wejściowych przedstawionych w poprzednim przykładzie otrzymałem następujące przypisania:

$$\mathcal{Z} = \left\{ \begin{array}{l} (1, 3, \text{IVA}, \text{Jn1}, \text{J.Niem}, 11), \\ (1, 3, \text{IVF}, \text{Jn1}, \text{J.Niem}, 11), \\ (1, 3, \text{IVA}, \text{Jr1}, \text{J.Ros}, 19), \\ (1, 3, \text{IVG}, \text{Jr1}, \text{J.Ros}, 19), \\ (1, 3, \text{IVD}, \text{Jf2}, \text{J.Fran}, 20), \\ (1, 3, \text{IVF}, \text{Jf2}, \text{J.Fran}, 20), \\ \vdots \\ (2, 7, \text{IA}, \text{Ja5}, \text{J.Ang}, 9) \\ (2, 7, \text{IA}, \text{Inf2}, \text{Informatyka}, \text{Sala Informatyczna 2}), \\ \vdots \\ (3, 7, \text{IG}, \text{Chem1}, \text{Chemia}, \text{Sala do Chemii 1}) \end{array} \right\}$$

Wszystkie lekcje zostały przypisane zgodnie z określonymi ograniczeniami. Nauczyciele otrzymali zajęcia zgodne z wymaganiami głównymi, a przedmioty, w szczególności informatyka oraz chemia, zostały przydzielone do odpowiednio wyposażonych sal dydaktycznych.

4. Aplikacja

Rozdział prezentuje praktyczną realizację rozwiązania teoretycznego przedstawionego wcześniej. Jego struktura odzwierciedla proces inżynierii oprogramowania, przechodzący od wymagań, przez projekt, aż do implementacji:

- Specyfikacja wymagań — formalizuje oczekiwania wobec systemu, dzieląc je na funkcjonalne i niefunkcjonalne, a następnie konkretyzując za pomocą diagramów przypadków użycia i sekwencji.
- Projekt systemu — przedstawia wysokopoziomową architekturę aplikacji, szczegółowy projekt bazy danych oraz makietę interfejsu użytkownika.
- Implementacja — opisuje konkretny stos technologiczny, sposób realizacji poszczególnych komponentów oraz integrację algorytmu z aplikacją webową.

4.1. Specyfikacja wymagań

4.1.1. Wymagania funkcjonalne

Aplikacja musi realizować cztery kluczowe grupy funkcjonalności:

- **obsługa wielu planów i konfiguracji** — możliwość pracy z wieloma zestawami zasobów i ograniczeń oraz wieloma planami równolegle,
- **wprowadzanie danych** — import plików txt i csv oraz ręczna edycja nauczycieli, klas, przedmiotów, sal, wymagań głównych i bloków przedmiotów,
- **generowanie planu** — automatyczne tworzenie planów z możliwością kontroli parametrów algorytmu ewolucyjnego,
- **przeglądanie planów** — wizualizacja planów z perspektywy nauczycieli i klas dla wielu wariantów.

4.1.2. Wymagania niefunkcjonalne

System musi spełniać następujące wymagania jakościowe:

- **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.

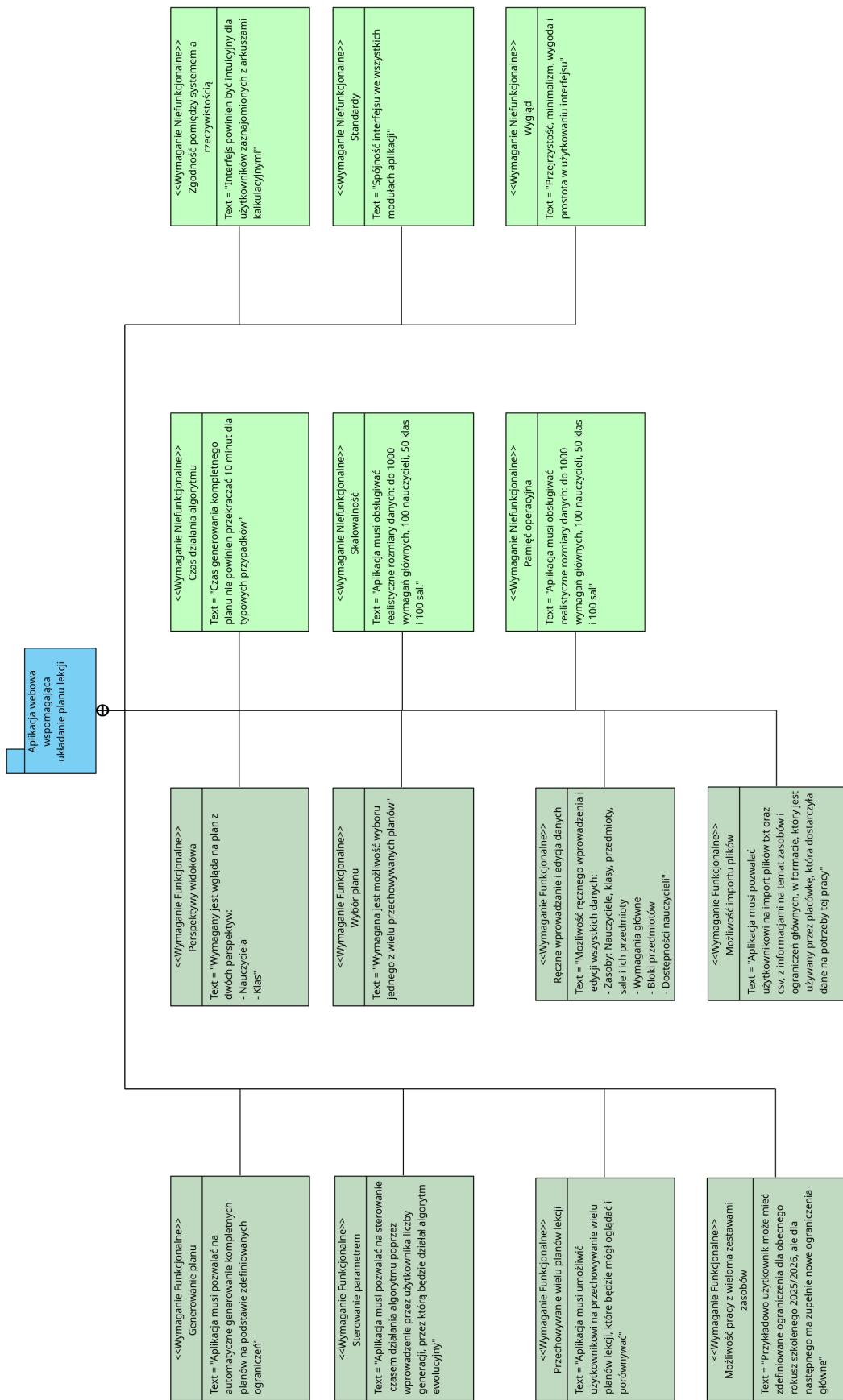
Szczegółowa specyfikacja wymagań została przedstawiona na rysunku diagramu wymagań [4.1](#).

4.1.3. Przypadki użycia

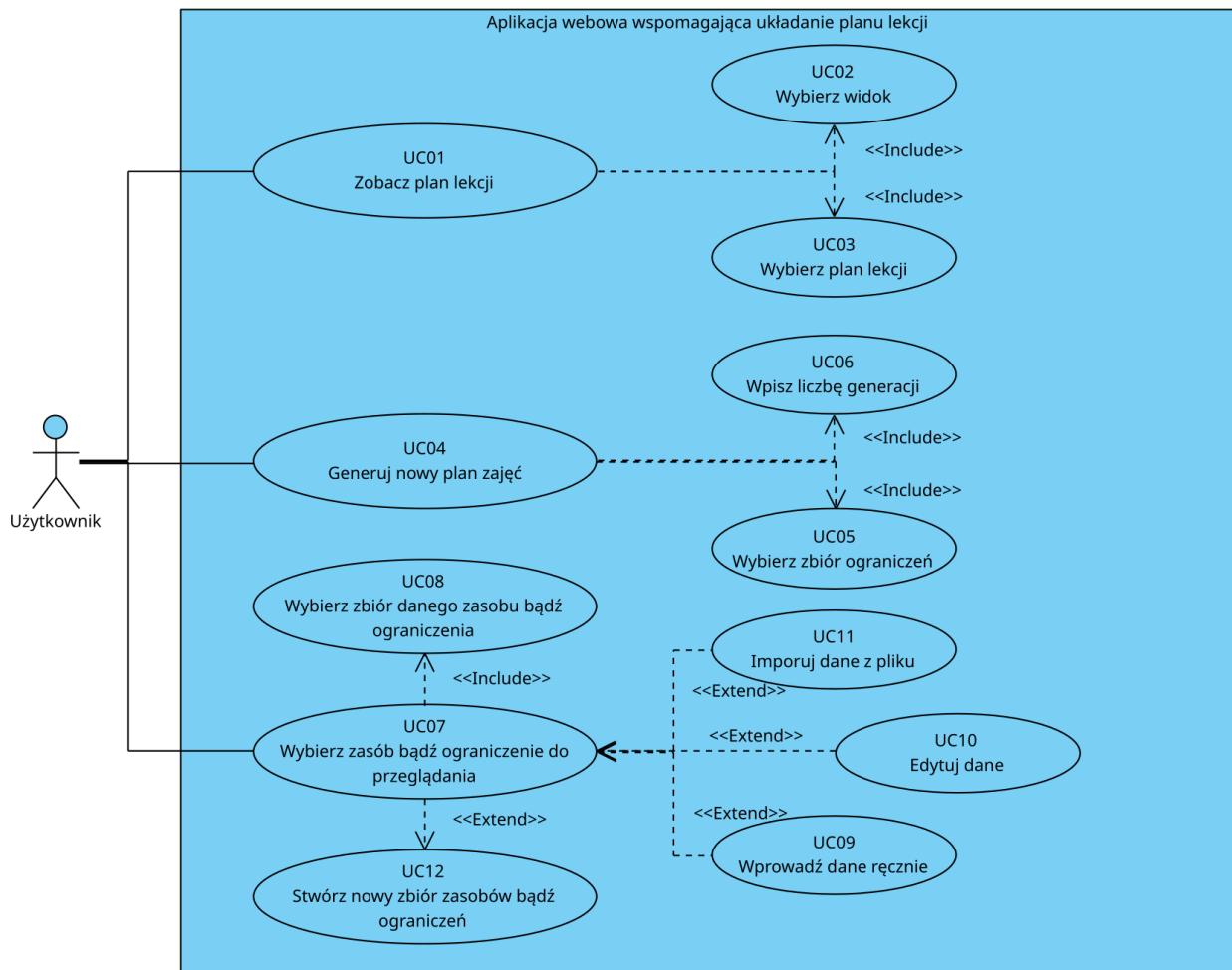
Diagram przypadków użycia pozwala na zwizualizowanie wcześniej omawianych wymagań funkcjonalnych w formie graficznej, ukazując interakcje między użytkownikami a systemem. Ułatwia on powiązanie specyfikacji wymagań z projektem architektury, stanowiąc punkt wyjścia do dalszych etapów analizy i implementacji. W niniejszej sekcji omawiam kluczowe przypadki użycia zilustrowane na diagramie, obejmujące zarówno generowanie planów lekcji, ich przeglądanie, jak i zarządzanie danymi wejściowymi systemu.

Identyfikator	UC01
Przypadek użycia	Zobacz plan lekcji
Aktor	Użytkownik
Cel	Sprawdzenie wygenerowanych planów
Składowe przypadki użycia	Wybierz widok Wybierz plan lekcji
Warunki wstępne	Wprowadzone zasoby: nauczyciele, sale, klasy, przedmioty Przynajmniej jeden wygenerowany plan
Stan końcowy	Wyświetlony plan lekcji
Scenariusz	<ol style="list-style-type: none"> 1. Użytkownik przechodzi w widok przeglądania planów lekcji 2. Użytkownik wybiera plan lekcji 3. Użytkownik wybiera widok 4. System wyświetla odpowiedni plan lekcji z danej perspektywy

Tabela 4.1: Przypadek użycia UC01



Rysunek 4.1: Diagram wymagań



Rysunek 4.2: Diagram przypadków użycia

Identyfikator	UC02
Przypadek użycia	Wybierz widok
Aktor	Użytkownik
Cel	Wybranie perspektywy oglądania planu lekcji
Warunki wstępne	Wprowadzone zasoby: nauczyciele, sale, klasy, przedmioty Przynajmniej jeden wygenerowany plan
Stan końcowy	Wybrany nauczyciel bądź klasa Plan wyświetlony z perspektywy wybranej klasy bądź nauczyciela
Scenariusze	1a. Użytkownik rozwija listę dostępnych nauczycieli 2a. Użytkownik wybiera konkretnego nauczyciela 3a. System wyświetla wybranego nauczyciela 1b. Użytkownik rozwija listę dostępnych klas 2b. Użytkownik wybiera konkretną klasę 3b. System wyświetla wybraną klasę

Tabela 4.2: Przypadek użycia UC02

Identyfikator	UC03
Przypadek użycia	Wybierz plan lekcji
Aktor	Użytkownik
Cel	Wybranie planu lekcji do podglądu
Składowe przypadki użycia	Wybierz zbiór ograniczeń Wybierz liczbę generacji
Warunki wstępne	Wprowadzone zasoby: nauczyciele, sale, klasy, przedmioty Przynajmniej jeden wygenerowany plan
Stan końcowy	Wybrany plan lekcji Możliwość wybrania w systemie widoku
Scenariusz	1. Użytkownik rozwija listę dostępnych planów lekcji 2. Użytkownik wybiera konkretnego plan lekcji 3. System wyświetla nazwę wybranego planu lekcji 4. System umożliwia wybranie widoku

Tabela 4.3: Przypadek użycia UC03

Identyfikator	UC04
Przypadek użycia	Generuj nowy plan lekcji
Aktor	Użytkownik
Cel	Wygenerowanie nowego planu lekcji na podstawie wybranych ograniczeń
Warunki wstępne	Wprowadzone zasoby: nauczyciele, sale, klasy, przedmioty Wprowadzone wymagania główne Wprowadzone dostępności nauczycieli Wprowadzone bloki przedmiotów
Stan końcowy	Wyświetlony komunikat o sukcesie operacji
Scenariusz	<ol style="list-style-type: none"> 1. Użytkownik wybiera zestaw ograniczeń 2. Użytkownik wpisuje liczbę generacji 3. Użytkownik wybiera opcję generowania planu 4. System generuje plan 5. System wysyła powiadomienie o zakończeniu procesu

Tabela 4.4: Przypadek użycia UC04

Identyfikator	UC05
Przypadek użycia	Wybierz zbiór ograniczeń
Aktor	Użytkownik
Cel	Wybranie zbioru ograniczeń służącego do wygenerowania nowego planu lekcji
Warunki wstępne	Przynajmniej jeden istniejący zbiór ograniczeń
Stan końcowy	Wybrany zestaw ograniczeń
Scenariusz	<ol style="list-style-type: none"> 1. Użytkownik rozwija listę dostępnych zestawów ograniczeń 2. Użytkownik wybiera konkretny zestaw ograniczeń 3. System wyświetla nazwę wybranego zestawu ograniczeń

Tabela 4.5: Przypadek użycia UC05

Identyfikator	UC06
Przypadek użycia	Wpisz liczbę generacji
Aktor	Użytkownik
Cel	Wpisanie liczby generacji służącej do wygenerowania planu lekcji
Warunki wstępne	Przynajmniej jeden istniejący zbiór ograniczeń
Stan końcowy	Widoczna wpisana liczba generacji Możliwa opcja wygenerowania planu lekcji
Scenariusz	<ol style="list-style-type: none"> 1. Użytkownik wpisuje liczbę generacji 2. System wyświetla wpisaną liczbę 3. System umożliwia wybranie opcji generowania planu lekcji

Tabela 4.6: Przypadek użycia UC06

Identyfikator	UC07
Przypadek użycia	Wybierz zasób bądź ograniczenie do przeglądania
Aktor	Użytkownik
Cel	Wybranie zasoby bądź ograniczenia do przeglądania, edytowania bądź wprowadzania
Składowe przypadki użycia	Wybierz zbiór danego zasobu bądź ograniczenia Stworzy nowy zbiór danego zasobu bądź ograniczenia
Warunki wstępne	—
Stan końcowy	Widoczny widok zasobu bądź ograniczenia
Scenariusz	<ol style="list-style-type: none"> 1. Użytkownik rozwija listę dostępnych zasobów i ograniczeń 2. Użytkownik wybiera jeden z nich 3. System wyświetla widok danego zasobu 4a. Użytkownik wybiera zbiór danego zasobu bądź ograniczenia 4b. Użytkownik tworzy nowy zbiór danego zasobu bądź ograniczenia 5a. Użytkownik importuje dane z pliku 5b. Użytkownik edytuje dane 5c. Użytkownik wprowadza dane ręcznie

Tabela 4.7: Przypadek użycia UC07

Identyfikator	UC08
Przypadek użycia	Wybierz zbiór danego zasobu bądź ograniczenia
Aktor	Użytkownik
Cel	Wybranie zbioru zasobu bądź ograniczenia do wglądu
Warunki wstępne	Przynajmniej jeden istniejący zbiór zasobu bądź ograniczenia
Stan końcowy	Widoczna nazwa zbioru Widoczny wybrany zbiór zasobów bądź ograniczeń
Scenariusz	<ol style="list-style-type: none"> 1. Użytkownik rozwija listę dostępnych zbiorów 2. Użytkownik wybiera konkretny zbiór 3. System wyświetla nazwę wybranego zbioru 4. System wyświetla dany zapisane w bazie danych

Tabela 4.8: Przypadek użycia UC08

Identyfikator	UC09
Przypadek użycia	Wprowadź ręcznie dane
Aktor	Użytkownik
Cel	Ręczne wprowadzenie nowych zasobów bądź ograniczeń
Warunki wstępne	Przynajmniej jeden istniejący zbiór zasobu bądź ograniczenia
Stan końcowy	Widoczne nowe ograniczenie/zasób Nowe ograniczenie/zasób zapisany w bazie danych
Scenariusz	<ol style="list-style-type: none"> 1. Użytkownik wybiera opcję dodania nowego zasobu bądź ograniczenia 2. Użytkownik wpisuje odpowiednie dane 3. Użytkownik zapisuje dane 3. System wyświetla powiadomienie o powodzeniu zapisu

Tabela 4.9: Przypadek użycia UC09

Identyfikator	UC10
Przypadek użycia	Edytuj dane
Aktor	Użytkownik
Cel	Edytowanie istniejących danych
Warunki wstępne	Przynajmniej jeden istniejący zbiór zasobu bądź ograniczenia
Stan końcowy	Widoczne zmiany Zmiany zapisane w bazie danych
Scenariusz	<ol style="list-style-type: none"> 1. Użytkownik edytuje wiersz zasobu bądź ograniczenia 2. Użytkownik zapisuje dane 3. System wyświetla powiadomienie o powodzeniu zapisu

Tabela 4.10: Przypadek użycia UC10

Identyfikator	UC11
Przypadek użycia	Importuj dane z pliku
Aktor	Użytkownik
Cel	Import danych z pliku txt lub csv
Warunki wstępne	Przynajmniej jeden istniejący zbiór zasobu bądź ograniczenia Użytkownik posiada plik we właściwym formacie
Stan końcowy	Widoczne dane, które były w pliku Dane zapisane w bazie danych
Scenariusz	<ol style="list-style-type: none"> 1a. Użytkownik wybiera opcję wybrania pliku źródłowego 2a. Użytkownik wybiera i zatwierdza wybrany plik źródłowy 1b. Użytkownik otwiera eksplorator plików 2b. Użytkownik przeciąga i upuszcza wybrany plik źródłowy 3. System wyświetla dane w pliku 4. Użytkownik zapisuje dane 5. System wyświetla powiadomienie o powodzeniu zapisu

Tabela 4.11: Przypadek użycia UC11

Identyfikator	UC12
Przypadek użycia	Stwórz nowy zbiór zasobów bądź ograniczeń
Aktor	Użytkownik
Cel	Stworzenie nowego zbioru zasobów bądź ograniczeń
Warunki wstępne	—
Stan końcowy	Wybrany zbiór zasobów bądź ograniczeń Widoczny nowy zbiór zasobów bądź ograniczeń
Scenariusz	Użytkownik wpisuje nazwę nowego zbioru zasobów bądź ograniczeń Użytkownik wybiera opcję stworzenia nowego zbioru zasobów bądź ograniczeń System wyświetla powiadomienie o powodzeniu zapisu System wybiera nowy zbiór zasobów bądź ograniczeń

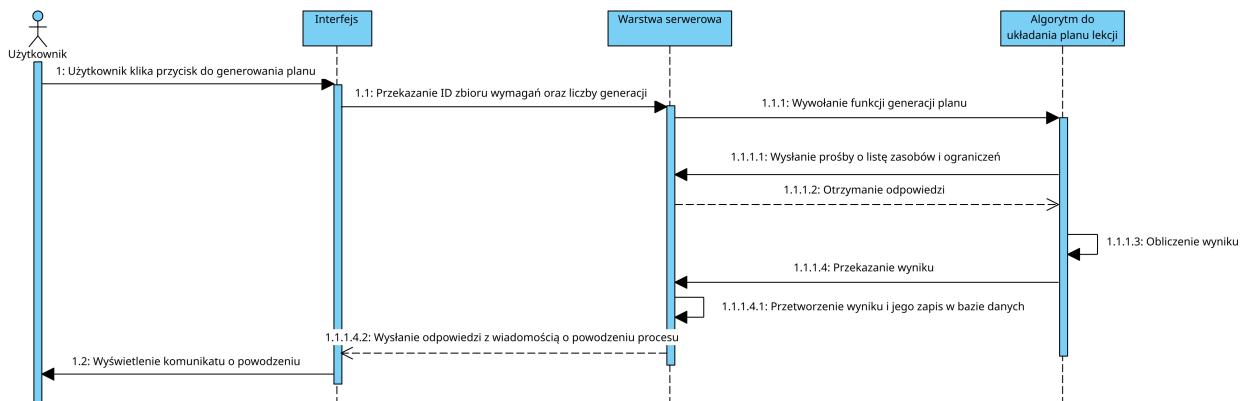
Tabela 4.12: Przypadek użycia UC12

4.1.4. Diagramy sekwencji

Diagramy sekwencji stanowią kolejny element modelowania systemu, ukazując sekwencyjny przepływ komunikatów między obiektami w czasie. Pozwalają prześledzić interakcje między obiektami w systemie — od inicjacji akcji w interfejsie, poprzez przetwarzanie w backendzie, aż do generowania planu przez algorytmy. W niniejszej sekcji przedstawiam scenariusze współpracy pomiędzy głównymi komponentami aplikacji: interfejsem, warstwą serwerową oraz algorytmem.

Diagram sekwencji generowania planu lekcji

Najważniejszym do zaprojektowania jest diagram sekwencji generowania planu lekcji. Jego poprawne zaprojektowanie ma istotny wpływ na czas oczekiwania użytkownika, ponieważ nadmiarowa komunikacja między interfejsem a backendem może znacząco wydłużyć proces. Diagram przedstawia minimalny, konieczny zestaw komunikatów prowadzących do wywołania algorytmu optymalizacyjnego i zwrócenia użytkownikowi gotowego planu.

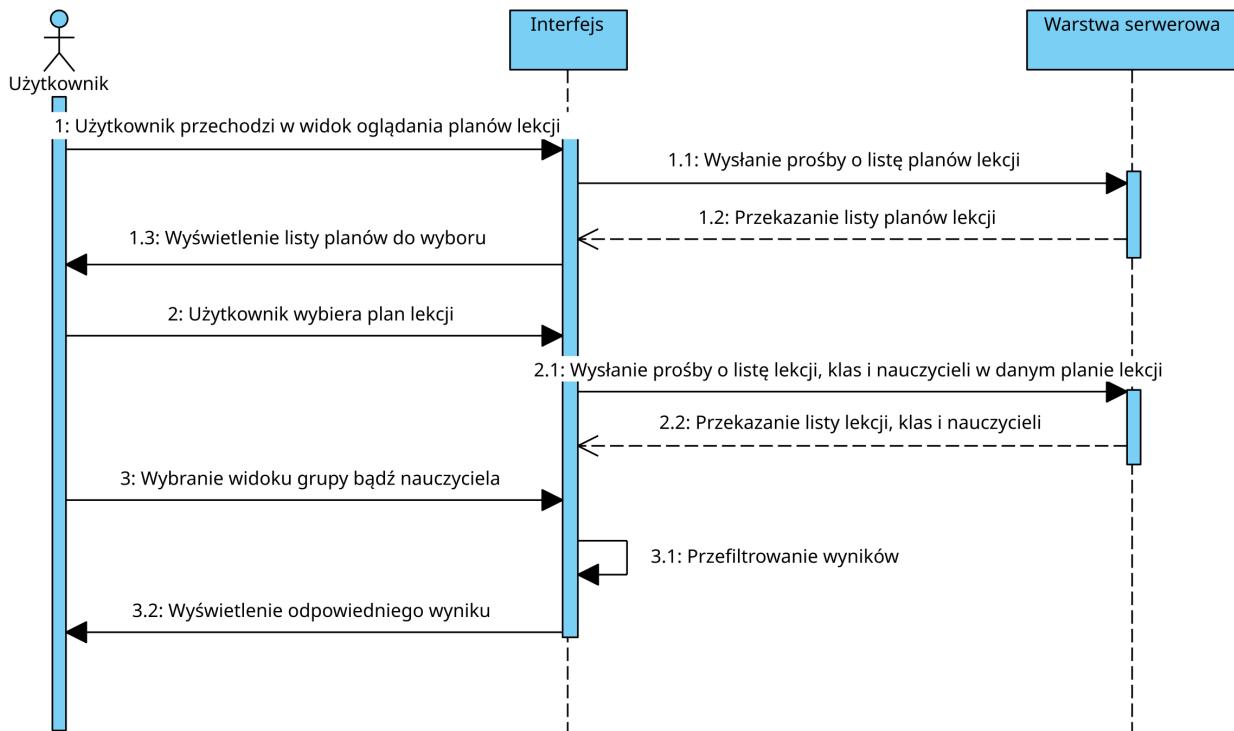


Rysunek 4.3: Diagram sekwencji generowania planu lekcji

Diagram sekwencji przeglądania planów lekcji

Drugim istotnym scenariuszem jest przeglądanie wygenerowanych planów. W tym przypadku kluczowym wymaganiem jest zapewnienie maksymalnej responsywności interfejsu. Ciągłe ponawianie zapytań GET przy każdej zmianie widoku byłoby nieefektywne i prowadziłoby do zauważalnych opóźnień.

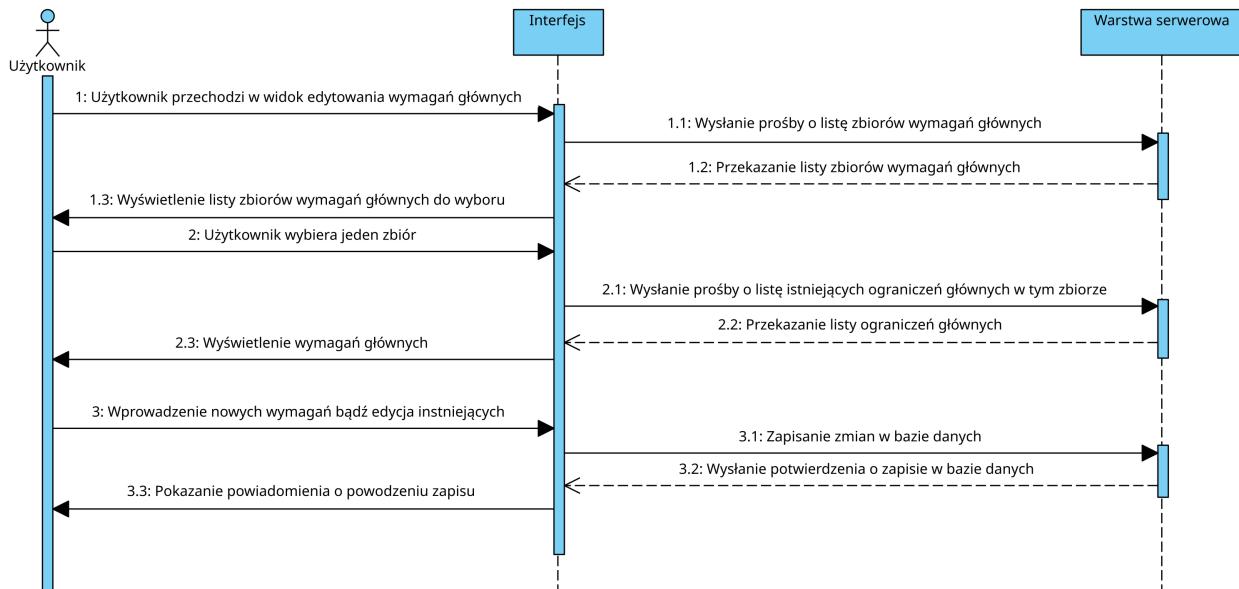
Z tego powodu zastosowałem inne rozwiązanie — po wybraniu konkretnego planu aplikacja wykonuje jedno większe zapytanie do bazy danych, a następnie cała dalsza filtracja odbywa się po stronie klienta. Minimalnie wydłuża to czas pierwszego ładowania, ale zapewnia pełną płynność późniejszej interakcji.



Rysunek 4.4: Diagram sekwencji dla przypadku użycia oglądania planu lekcji

Diagram sekwencji wprowadzania i edycji danych

Trzeci scenariusz obejmuje wprowadzanie lub edycję danych, takich jak wymagania główne. Kluczowym założeniem było ograniczenie liczby zapytań do serwera przy jednoczesnym zapewnieniu użytkownikowi informacji zwrotnej o powodzeniu operacji.



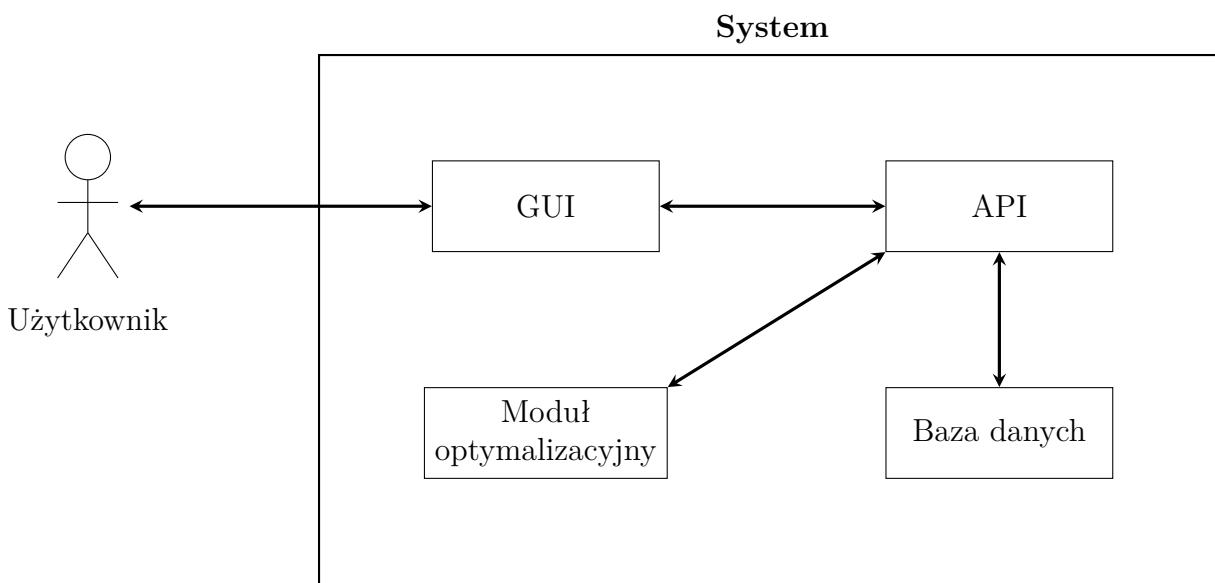
Rysunek 4.5: Diagram sekwencji edycji bądź wprowadzania wymagań głównych

4.2. Projekt

4.2.1. Architektura aplikacji

Wybór odpowiedniej architektury aplikacji jest kluczowy dla zapewnienia jej skalowalności, elastyczności i możliwości dalszego rozwoju. W celu spełnienia wszystkich wymagań projektowych podzieliłem aplikację na cztery główne komponenty:

- **Interfejs użytkownika** (ang. *Graphical User Interface*, GUI) — odpowiada za komunikację z użytkownikiem, umożliwiając wygodną obsługę systemu bez konieczności znajomości szczegółów działania jego wewnętrznych mechanizmów.
- **Warstwę sewerową** — umożliwia sprawną integrację wszystkich komponentów poprzez interfejs programowania aplikacji (ang. *Application Programming Interface*, API).
- **Bazę danych** — przechowuje wszystkie zasoby, ograniczenia oraz plany lekcji w ustrukturyzowany sposób.
- **Moduł optymalizacyjny** — realizuje algorytmy generowania planów lekcji, przetwarzając dane z bazy i uwzględniając wszystkie wymagania i ograniczenia.



Rysunek 4.6: Architektura aplikacji

Taki podział umożliwia niezależny rozwój każdego z komponentów, ułatwia testowanie i utrzymanie systemu, a także pozwala na łatwe skalowanie oraz integrację dodatkowych funkcjonalności w przyszłości.

4.2.2. Projekt struktury bazy danych

Projekt bazy danych stanowi kluczowy element architektury aplikacji. Jego celem jest umożliwienie wielokrotnego wykorzystania zasobów szkolnych przy jednoczesnej obsłudze zmieniających się wymagań w kolejnych generacjach planów lekcji.

Aby zapewnić elastyczność, zastosowałem mechanizm zbiorów zasobów oraz zbiorów wymagań, dzięki czemu każdy wygenerowany plan odwołuje się do spójnej konfiguracji danych.

Poniżej opisałem strukturę bazy danych, dzieląc ją na cztery grupy tabel: tabele podstawowe, tabele wymagań, tabele agregujące, oraz tabele wyników.

Tabele podstawowe

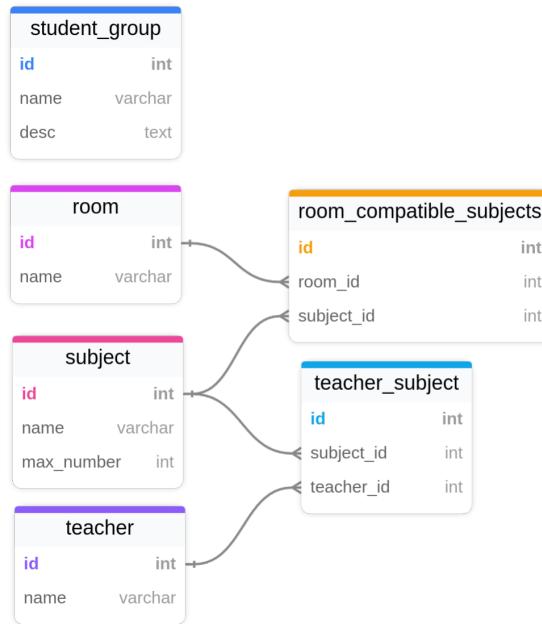
Tabele podstawowe definiują fundamentalne zasoby szkoły, stanowiąc punkt odniesienia dla całego systemu:

Tabela	Relacja
subject	
student_group	
teacher	M:N do subject
room	M:N do subject

Tabela 4.13: Tabele podstawowe zasobów szkolnych

gdzie:

- **subject** przechowuje informacje o przedmiotach.
- **student_group** przechowuje informacje o klasach.
- **teacher** przechowuje informacje o nauczycielach.
 - Relacja M:N z nauczycielami określa jakich przedmiotów może uczyć dany nauczyciel.
- **room** przechowuje informacje o salach.
 - Relacja M:N z przedmiotami określa jakie przedmioty mogą być prowadzone w danej sali.



Rysunek 4.7: Diagram tabel podstawowych w bazie danych

Tabele wymagań

Tabele wymagań definiują, co dokładnie należy umieścić w planie lekcji. W przeciwieństwie do tabel zasobów nie opisują one tego, co istnieje w szkole, lecz to, jakie lekcje muszą zostać wygenerowane.

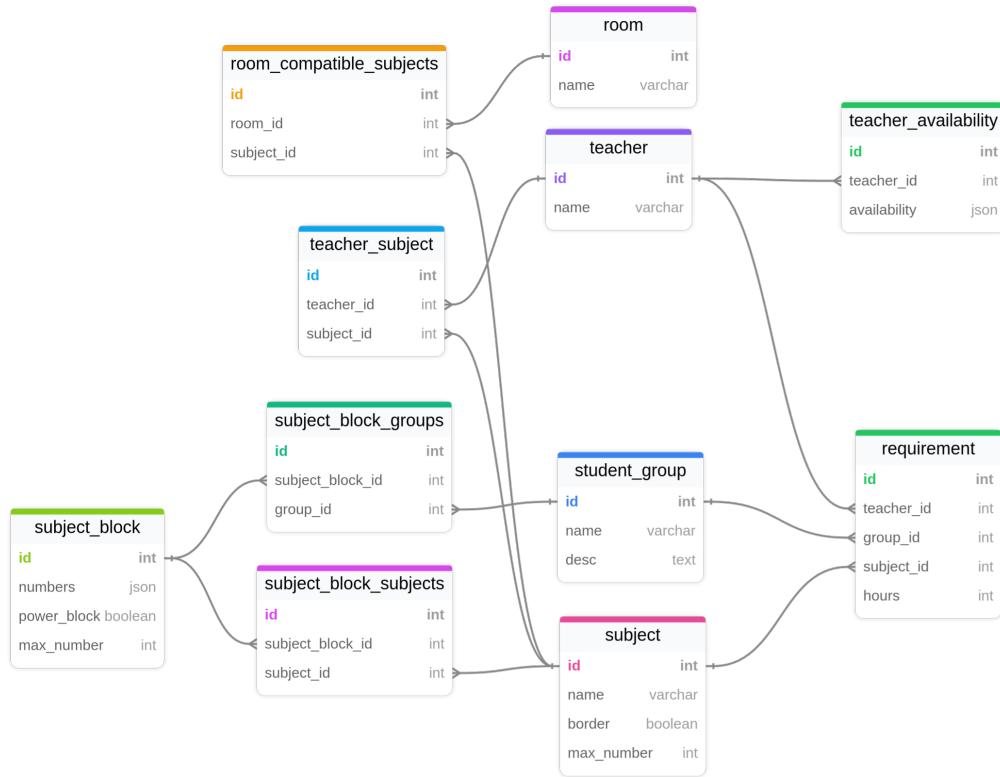
Tabela	Relacje
requirement	N:1 z teacher , student_group , subject
subject_block	M:N z subject , M:N z student_group
teacher_availability	N:1 z teacher

Tabela 4.14: Tabele definiujące wymagania i ograniczenia

gdzie:

- **requirement** każdy rekord opisuje jedno wymaganie główne.
 - Zawiera klucze obce do tabel przedmiotów, klas, nauczycieli.
- **subject_block** przechowuje informacje na temat bloków przedmiotów.
 - Relacja M:N z przedmiotami określa jakie przedmioty powinny być w bloku lekcyjnym.
 - Relacja M:N z klasami określa jakie klasy powinny być w bloku lekcyjnym.
- **teacher_availability** przechowuje dostępności nauczycieli. Każdy rekord dotyczy innego nauczyciela.

- Zawiera klucz obcy do tabeli nauczycieli.



Rysunek 4.8: Diagram tabel podstawowych i wymagań w bazie danych

Zbiory danych

Mechanizm zbiorów umożliwia grupowanie zasobów i wymagań dla różnych kontekstów:

Tabela	Relacja
subject_pool	M:N z subject
student_group_pool	1:N z student_group
teacher_pool	M:N z teacher
room_pool	1:N z room

Tabela 4.15: Tabele zbiorów zasobów

Dla nauczycieli i przedmiotów zastosowałem relację wiele-do-wielu, ponieważ zasoby te charakteryzują się względną stabilnością pomiędzy latami szkolnymi — większość nauczycieli

cieli i przedmiotów pozostaje niezmienna, a jedynie dodawane są nowe rekordy dla nowo zatrudnionych pedagogów lub wprowadzanych przedmiotów.

W przypadku sal przyjęto relację jeden-do-wielu, co wynika z całkowitej niezmienności tej puli zasobów pomiędzy okresami szkolnymi. Natomiast dla klas zastosowałem relację jeden-do-wielu, odzwierciedlającą coroczną całkowitą wymianę składu grup uczniowskich — klasy z poprzedniego roku nie są ponownie wykorzystywane w kolejnych okresach, co eliminuje potrzebę stosowania relacji wiele-do-wielu.

Szczególną rolę pełni tabela **requirement_set** (zbiór wymagań głównych), ponieważ agreguje wszystkie inne zasoby oraz ogarniczenia.

Tabela	Relacja
requirement_set	1:N z requirement , teacher_availability , subject_block N:1 z subject_pool , student_group_pool , teacher_pool , room_pool

Tabela 4.16: Tabela requirement_set

gdzie:

- **requirement_set** każdy rekord opisuje jedno wymaganie główne.
 - Posiada relację 1:N z tabelą **requirement**, **teacher_availability** oraz **subject_block**.
 - Zawiera klucze obce do tabel **subject_pool**, **student_group_pool**, **teacher_pool** oraz **room_pool**.

Ta struktura umożliwia generowanie kompletnego planu lekcji na podstawie pojedynczego rekordu zbioru wymagań.

Zaprezentowany model bazy danych zapewnia elastyczność niezbędną do obsługi zmieniających się wymagań szkolnych poprzez mechanizm zbiorów. Struktura umożliwia wielokrotne wykorzystanie zasobów pomiędzy różnymi konfiguracjami planów, zachowując przy tym prostotę i wydajność.

Tabele wyników

Tabele przechowujące wygenerowane plany lekcji i ich przypisania:

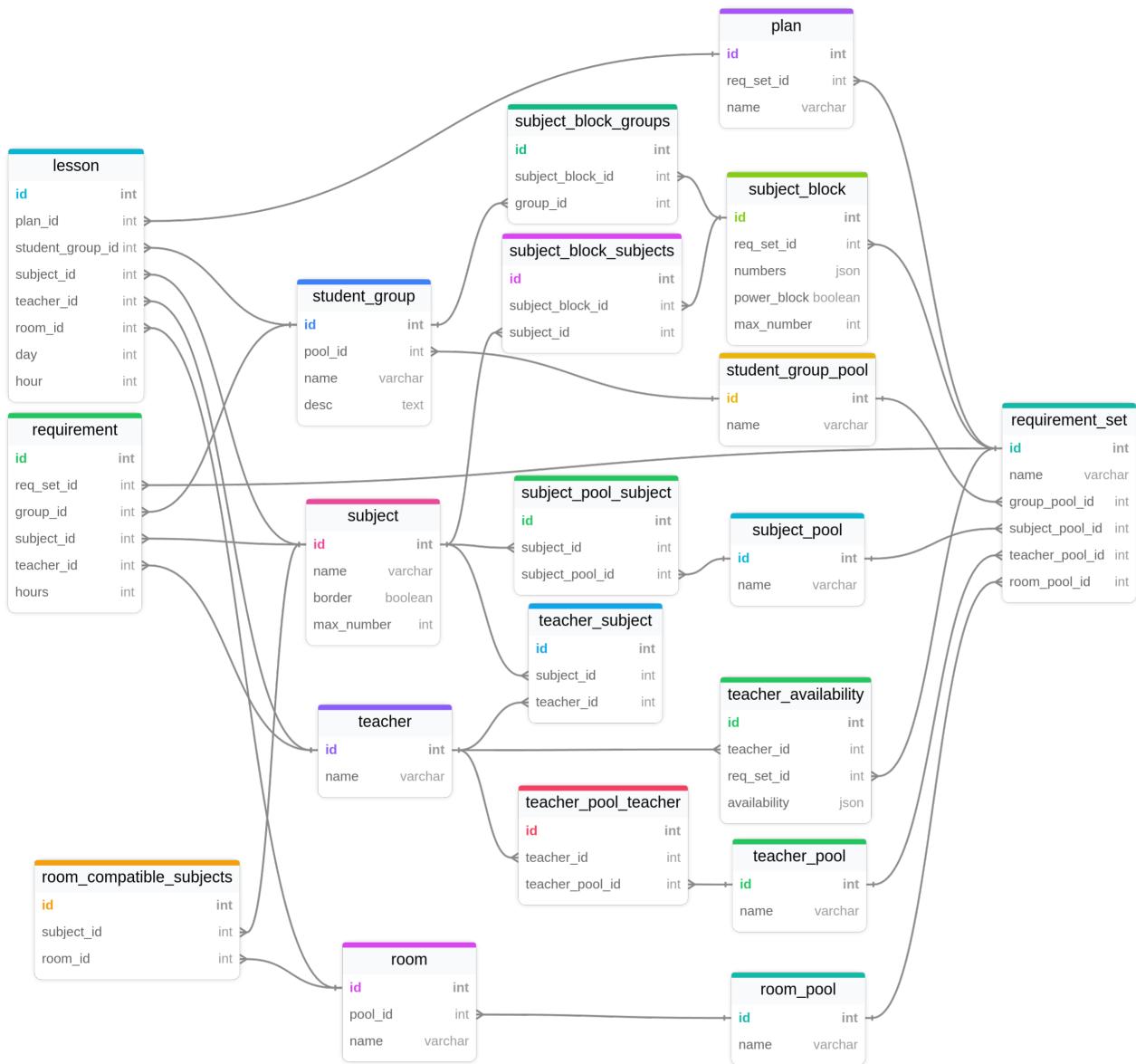
Tabela	Relacje
plan	N:1 z requirement_set
lesson	N:1 z teacher , subject , room , student_group , plan

Tabela 4.17: Tabele przechowujące wygenerowane plany lekcji

gdzie:

- **plan** przechowuje plany lekcji dla konkretnego zbioru wymagań (klucz obcy do odpowiedniej tabeli).

- **lesson** przechowuje przypisania. Posiada klucz obcy do wszystkich zasobów oraz planu lekcji.



Rysunek 4.9: Diagram pełnego projektu bazy danych

Tabela	Kolumna	Opis
student_group	<code>id</code> <code>pool_id</code> <code>name</code> <code>desc</code>	Unikalny identyfikator klasy Klucz obcy do zbioru klas Nazwa klasy („IIIA”) Opis klasy (opcjonalny)
room	<code>id</code> <code>pool_id</code> <code>name</code>	Unikalny identyfikator sali Klucz obcy do zbioru sal Nazwa sali („Sala 101”)
subject	<code>id</code> <code>name</code>	Unikalny identyfikator przedmiotu Nazwa przedmiotu („Matematyka”)
teacher	<code>id</code> <code>name</code>	Unikalny identyfikator nauczyciela Imię i/lub nazwisko nauczyciela
requirement_set	<code>id</code> <code>teacher_pool_id</code> <code>room_pool_id</code> <code>group_pool_id</code> <code>subject_pool_id</code> <code>name</code>	Unikalny identyfikator zbioru wymagań Klucz obcy do zbioru nauczycieli Klucz obcy do zbioru sal Klucz obcy do zbioru klas Klucz obcy do zbioru przedmiotów Nazwa zbioru wymagań
subject_block	<code>id</code> <code>req_set_id</code> <code>numbers</code> <code>power_block</code> <code>max_number</code>	Unikalny identyfikator bloku przedmiotów Klucz obcy do zbioru wymagań Konfiguracja liczby godzin (JSON) Flaga bloku agregującego Maksymalna liczba bloków tygodniowo
requirement	<code>id</code> <code>req_set_id</code> <code>teacher_id</code> <code>group_id</code> <code>subject_id</code> <code>hours</code>	Unikalny identyfikator wymagania Klucz obcy do zbioru wymagań Klucz obcy do nauczyciela Klucz obcy do klasy Klucz obcy do przedmiotu Liczba wymaganych godzin tygodniowo
teacher_availability	<code>id</code> <code>teacher_id</code> <code>req_set_id</code> <code>availability</code>	Unikalny identyfikator dostępności Klucz obcy do nauczyciela Klucz obcy do zbioru wymagań Macierz dostępności (JSON)
plan	<code>id</code> <code>req_set_id</code> <code>name</code>	Unikalny identyfikator planu Klucz obcy do zbioru wymagań Nazwa planu lekcji
lesson	<code>id</code> <code>plan_id</code> <code>teacher_id</code> <code>subject_id</code> <code>room_id</code> <code>student_group_id</code> <code>day</code> <code>hour</code>	Unikalny identyfikator lekcji Klucz obcy do planu Klucz obcy do nauczyciela Klucz obcy do przedmiotu Klucz obcy do sali Klucz obcy do klasy Dzień tygodnia Slot czasowy

Tabela 4.18: Struktura najważniejszych tabel w bazie danych

4.2.3. Projekt interfejsu

Projekt interfejsu użytkownika opracowałem z uwzględnieniem wymagań funkcjonalnych. Architektura nawigacji opiera się na logicznym przepływie pracy charakterystycznym dla procesu układania planu lekcji, zapewniając intuicyjne środowisko dla użytkowników zaznajomionych z arkuszami kalkulacyjnymi.

Nawigacja

System nawigacji został zaprojektowany jako hierarchiczny, z głównym menu umożliwiającym dostęp do wszystkich kluczowych funkcjonalności. Nawigacja górnego poziomu obejmuje przejście między głównymi modułami aplikacji, podczas gdy niższe poziomy zapewniają dostęp do szczegółowych funkcji w obrębie modułu wprowadzania danych.

Strona główna

Strona główna pełni funkcję centralnego hubu aplikacji. Umożliwia bezpośrednie przejście do trzech kluczowych sekcji:

- podglądu istniejących planów lekcji,
- wprowadzania i edycji ograniczeń głównych,
- modułu generowania nowych planów.

Podgląd planów lekcji

Strona podglądu planu lekcji umożliwia przeglądanie wielu wariantów planów, co realizuje wymaganie co do możliwości przechowywania wielu planów lekcji. Realizuje także wymagania dotyczące wieloperspektywicznego wglądu w wygenerowane plany — umożliwia wgląd z perspektywy:

- nauczycieli — prezentujący indywidualne plany zajęć każdego pedagoga,
- klas — ukazujący rozkład zajęć dla poszczególnych grup uczniowskich.

Moduły wprowadzania danych

Aplikacja zawiera kompleksowy zestaw stron do wprowadzania wszystkich typów danych wymaganych do generowania planu:

- **Strony zasobów** umożliwiają zarządzanie czterema podstawowymi zasobami: przedmiotami, nauczycielami, klasami oraz salami. Interfejs powinien zapewniać możliwość ręcznego wprowadzania, edycji i usuwania rekordów, a także importu danych z plików tekstowych w ustalonych formatach.
- **Strona ograniczeń głównych** dedykowana jest definiowaniu wymagań głównych, czyli przypisań godzinowych nauczycieli do klas dla konkretnych przedmiotów. Interfejs powinien nawiazywać wygólem i funkcjonalnością do arkuszy kalkulacyjnych, co ułatwia migrację z dotychczas używanych narzędzi.

- **Strona dostępności nauczycieli** zapewnia graficzny interfejs do określania dni, w których poszczególni nauczyciele są dostępni do prowadzenia zajęć. Wykorzystanie intuicyjnych checkboxów z możliwością blokowania i preferowania konkretnych dni.
- **Strona bloków przedmiotów** umożliwia konfigurację bloków przedmiotów, zgodnie z koncepcją przedstawioną w rozdziale 3. Interfejs powinien pozwalać na definiowanie składu bloków, klas objętych blokiem, ograniczeń liczbowych oraz flagi agregacji.

Moduł generowania planu

Strona generowania planu lekcji powinna integrować wszystkie wprowadzone dane i ograniczenia, oferując proste dla użytkownika sterowanie procesem optymalizacji. Docelowi użytkownicy nie są osobami zaznajomionymi z technologią, a tym bardziej algorytmami ewolucyjnymi — nie powinni mieć pełnej kontroli nad wszystkimi parametrami. Jedyny parametr, który jest ważny z ich punktu widzenia to liczba generacji algorytmu genetycznego, która jest proporcjonalna do czasu oczekiwania na generację planu.

Spójność i ergonomia

Wszystkie strony aplikacji powinny utrzymywać spójny schemat wizualny oparty na zasadach minimalizmu i przejrzystości. Projekt interfejsu prioritetyzuje wygodę użytkowania poprzez ograniczenie liczby koniecznych kliknięć do wykonania kluczowych operacji oraz zapewnienie natychmiastowego dostępu do najczęściej używanych funkcjonalności.

Wszystkie strony powinny również oferować powiadomienia o wszystkich wykonanych działaniach, tak aby użytkownik wiedział czy jego działania kończą się sukcesem.

4.3. Implementacja

4.3.1. Wybór narzędzi

W wyborze stosu technologicznego kierowałem się produktywnością rozwoju aplikacji, skalowalnością rozwiązania oraz integracją z innymi komponentami systemu.

Interfejs webowy

Frontend to część aplikacji odpowiadająca za interfejs użytkownika i działającą po stronie przeglądarki internetowej. Framework frontendowy to zestaw narzędzi, bibliotek i konwencji programistycznych, które ułatwiają i przyspieszają tworzenie oprogramowania, zapewniając gotowe rozwiązania dla typowych problemów.

Decyzja o wykorzystaniu Reacta [12] jako frameworka frontendowego wynika z kilku kluczowych czynników. Po pierwsze, posiadam znaczące doświadczenie w rozwoju aplikacji przy użyciu tej technologii, co pozwoliło na efektywną implementację interfejsu użytkownika. Po drugie, React jest jednym z najbardziej popularnych i dojrzałych frameworków frontendowych, co gwarantuje obszerną dokumentację oraz aktywną społeczność wspierającą rozwój. Po trzecie, React oferuje responsywne zarządzanie stanem aplikacji, co umożliwia dynamiczną

aktualizację widoków planów lekcji bez konieczności przeładowywania strony, zapewniając płynność interakcji użytkownika podczas przeglądania planów lekcji.

Warstwa serwerowa

Wybór Pythona jako głównego języka programowania dla całego systemu był decyzją, podyktowaną koniecznością sprawnej integracji modułu optymalizacyjnego z warstwą aplikacji webowej. Implementacja algorytmu generowania planu lekcji w Pythonie umożliwiła wykorzystanie pakietów optymalizacyjnych Google OR-Tools oraz biblioteki NumPy, eliminując potrzebę tworzenia skomplikowanych interfejsów międzyjęzykowych.

Backend to część aplikacji działająca po stronie serwera, jest odpowiedzialna za logikę, komunikację z bazą danych oraz zapewnianie API dla frontendu. Stanowi on fundament aplikacji webowej, podczas gdy frontend odpowiada za prezentację danych użytkownikowi. Podobnie jak w przypadku frontendu istnieje wiele frameworków ułatwiających tworzenie tej części aplikacji.

Wybór Django [8] jako frameworka backendowego został dokonany po analizie alternatywnego rozwiązania w Pythonie — FastAPI. Pomimo że FastAPI oferuje wydajność i nowoczesne funkcje, Django zapewnia kompleksowe podejście, które jest lepiej dostosowane do złożonych aplikacji webowych. Moje wcześniejsze doświadczenie z Django pozwoliło na efektywną pracę oraz sprawne wykorzystanie technik ORM (ang. *Object-Relational Mapping*). Dodatkowo, skalowalność Django i jego bezpieczeństwo zapewniają solidne fundamenty dla aplikacji, która może ewoluować w przyszłości.

Baza danych

PostgreSQL został wybrany jako system zarządzania bazą danych ze względu na jego wsparcie oraz doskonałą integrację z Django. Kluczowym czynnikiem była kompatybilność z biblioteką Django-Tenants, która umożliwia przyszłą implementację architektury opisanej w jej dokumentacji [4]. Ta funkcjonalność pozwoli w przyszłości na łatwe dodanie obsługi wielu niezależnych użytkowników lub instytucji w obrębie pojedynczej instancji aplikacji. Dodatkowo, wsparcie dla niestandardowych typów danych, między innymi JSON, w PostgreSQL idealnie odpowiada potrzebom przechowywania skomplikowanych struktur danych, takich jak macierze dostępności nauczycieli czy konfiguracje bloków przedmiotów.

Moduł optymalizacyjny

Moduł optymalizacyjny został zaimplementowany w języku Python, co umożliwiło łatwą integrację z warstwą serwerową oraz szybkie prototypowanie rozwiązań. W przypadku algorytmu ewolucyjnego wykorzystałem bibliotekę NumPy, która dzięki zoptymalizowanym operacjom macierzowym znaczaco przyspiesza obliczenia i umożliwia efektywne przetwarzanie populacji rozwiązań.

Algorytm zachłanny oraz algorytm ewolucyjny zostały zaimplementowane od zera, przy użyciu wspomnianej wyżej biblioteki, co pozwoliło na pełną kontrolę nad ich działaniem. W trzecim etapie użyłem pakietu Google OR-Tools, który oferuje bogaty zestaw narzędzi do modelowania problemów programowania z ograniczeniami, w tym zmienne przedziałowe i

ograniczenia typu `NoOverlap`. Jego dobra dokumentacja, wysoka wydajność oraz natywna integracja z Pythonem sprawiają, że stanowi on naturalny wybór w tego typu zastosowaniach.

4.3.2. Implementacja bazy danych w Django

Technika ORM stosowana w Django oferuje intuicyjny sposób definiowania relacji między modelami.

Relacje wiele-do-jednego są definiowane klasą `ForeignKey`. Relacje wiele-do-wielu implementuje się za pomocą klasy `ManyToManyField`, która automatycznie generuje niezbędne tabele pośredniczące w bazie danych.

Każda tabela w systemie, z wyjątkiem automatycznie generowanych tabel dla relacji wiele-do-wielu, jest definiowana przez dedykowaną klasę w pliku `models.py`. Na rysunku 4.10 przedstawiłem przykładową implementację modelu reprezentującego sale.

Tabela sal

```
1 class Room(models.Model):
2     pool = models.ForeignKey(RoomPool, on_delete=models.CASCADE)
3     name = models.CharField(max_length=255)
4     compatible_subjects = models.ManyToManyField(Subject)
```

Rysunek 4.10: Fragment kodu służący do definicji tabeli sal

Implementacja dla reszty tabel opisanych w punkcie 4.2.2 jest analogiczna.

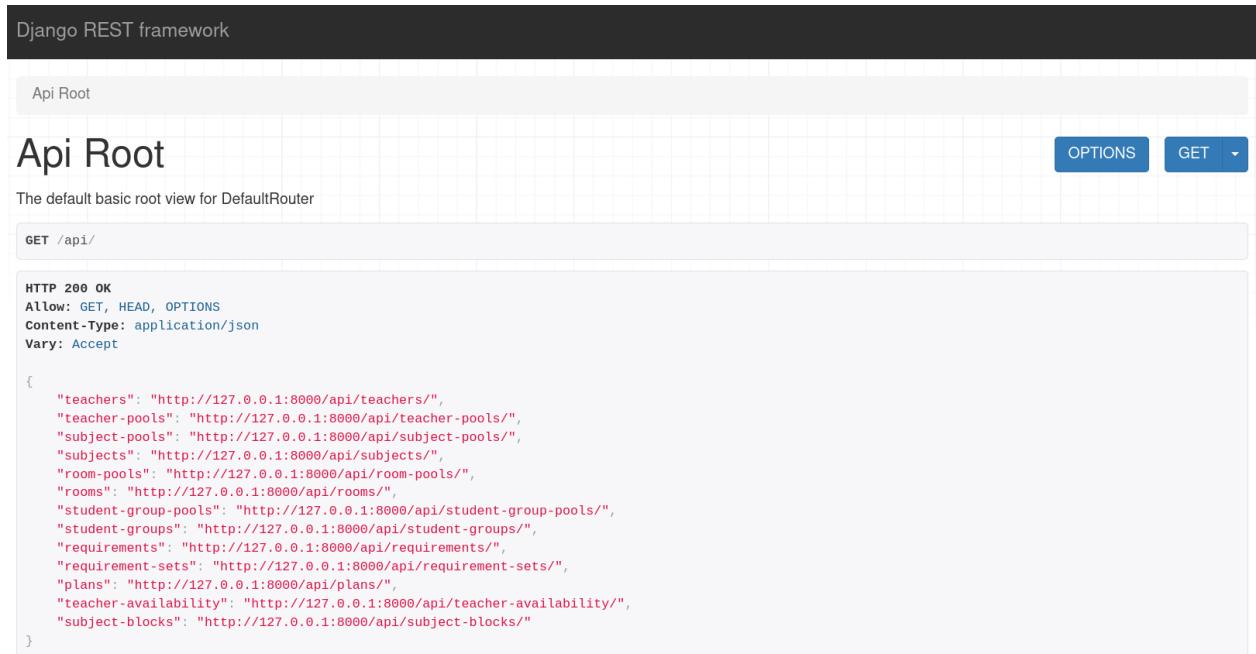
4.3.3. Implementacja API

Aby zrozumieć implementację API w Django należy znać parę pojęć:

- **Serializacja** (ang. *Serializing*) to proces konwersji obiektów Django na format danych przesyłany przez API oraz konwersja danych przychodzących na obiekty Django.
- **Widok** (ang. *View*) to komponent odpowiedzialny za przetwarzanie żądań HTTP i zwracanie odpowiedzi, implementujący logikę biznesową dla konkretnych punktów końcowych.
- **Punkt końcowy** (ang. *Endpoint*) to unikalny adres URL w API, który odpowiada na żądania HTTP i udostępnia konkretną funkcjonalność.

Warstwa API została zaimplementowana z wykorzystaniem Django REST Framework — elastycznego zestawu narzędzi do budowy interfejsów REST w Django. Znaczco przyspieszyło to proces rozwijania aplikacji poprzez dostarczenie gotowych komponentów do obsługi operacji CRUD, validacji danych, autentykacji i serializacji. Jedną z kluczowych zalet frameworka są widoki `ModelViewSet`, które automatycznie generują kompletny zestaw punktów końcowych

dla modelu na podstawie minimalnej konfiguracji, eliminując konieczność ręcznego implementowania rutynowych operacji. Takie widoki są również modyfikowalne, co pozwala na uwzględnienie niestandardowych przypadków, takich jak tworzenie wielu wymagań głównych na raz w celu poprawienia wydajności, z relatywną łatwością.



Rysunek 4.11: Automatycznie wygenerowana lista punktów końcowych stworzonych przy użyciu ModelViewSet

Na rysunku 4.12 przedstawiłem przykładową implementację punktów końcowych dotyczących tabeli `room_pool` używając klas `ModelViewSet`.

```

1 class RoomPoolSerializer(serializers.ModelSerializer):
2     class Meta:
3         model = RoomPool
4         fields = [ "id" , "name" ]
5
6 class RoomPoolViewSet(ModelViewSet):
7     queryset = RoomPool.objects.all()
8     serializer_class = RoomPoolSerializer

```

Rysunek 4.12: Fragment kodu służący do implementacji punktów końcowych dla zbioru sal

4.3.4. Implementacja interfejsu webowego

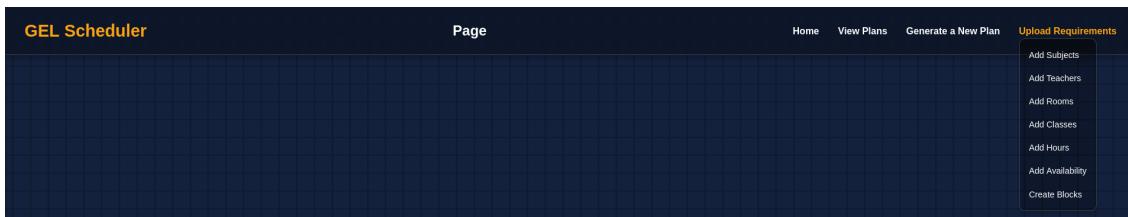
Komponentowa architektura Reacta pozwoliła na zbudowanie modułarnego interfejsu, gdzie poszczególne widoki (takie jak edycja wymagań, konfiguracja zasobów czy podgląd planów)

zostały wydzielone jako niezależne komponenty, które w połączeniu z komponentem nawigacji tworzą konkretne strony. Połączenie z backendem realizowane jest poprzez zdefiniowane powyżej API, zapewniając płynną komunikację między warstwą interfejsu a warstwą serwerową.

Nawigacja

Nawigacja przedstawiona na rysunku 4.13 składa się z trzech elementów:

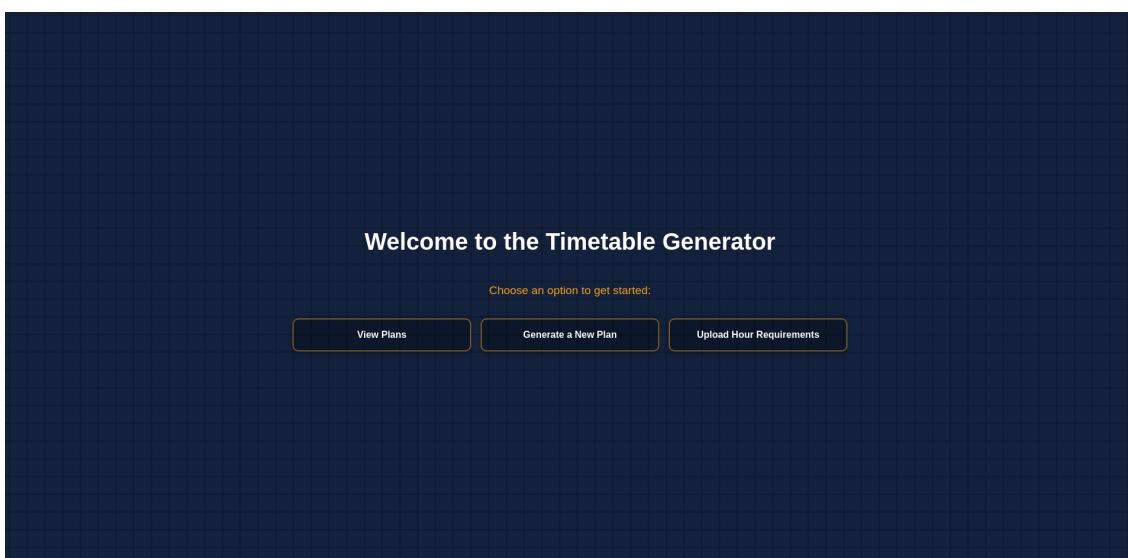
- Po lewej stronie znajduje się nazwa aplikacji.
- Na środku wyświetlana jest aktualna nazwa podstrony, co pomaga użytkownikowi szybko zorientować się w kontekście.
- Po prawej dostępne są główne sekcje aplikacji, w tym rozwijane menu „Upload Requirements”, umożliwiające przejście do konkretnych modułów wprowadzania danych.



Rysunek 4.13: Nawigacja interfejsu webowego

Strona główna

Strona główna przedstawiona na rysunku 4.14 zapewnia szybki dostęp do kluczowych funkcji aplikacji: przeglądania istniejących planów lekcji, generowania nowych oraz wprowadzania wymagań głównych.



Rysunek 4.14: Strona główna interfejsu webowego

Podgląd planów lekcji

Widok planów lekcji można przeglądać zarówno z perspektywy klas, jak i nauczycieli. W widoku klas każda lekcja prezentowana jest wraz z przypisanym nauczycielem, natomiast w widoku nauczyciela wyświetlane są klasy, z którymi prowadzi zajęcia.

Implementacja zaprezentowana na rysunku 4.15 oraz 4.16 spełnia wymagania funkcjonalne co do przeglądania planów.

GEL Scheduler		Lesson View						Home		View Plans		Generate a New Plan		Upload Requirements		
		Plan generated using		Select Group		Mat1										
Time Slot	Monday	Tuesday	Wednesday		Thursday		Friday									
1			Matematyka II_BG		Matematyka III_D		Matematyka I_B		13							
2	Matematyka II_BG	Matematyka III_D	Matematyka III_D		Matematyka III_D		Matematyka I_B		13							
3	Matematyka II_BG	Matematyka III_D	Matematyka III_D		Matematyka III_D		Matematyka II_BG		15							
4	Matematyka III_D				Matematyka I_D		Matematyka I_D		13							
5	Matematyka I_D	Matematyka I_D	Matematyka I_B		Matematyka I_D		Matematyka I_D		15							
6		Matematyka I_D			Matematyka I_D		Matematyka II_BG		15							
9																

Rysunek 4.15: Podgląd planu z perspektywy nauczyciela

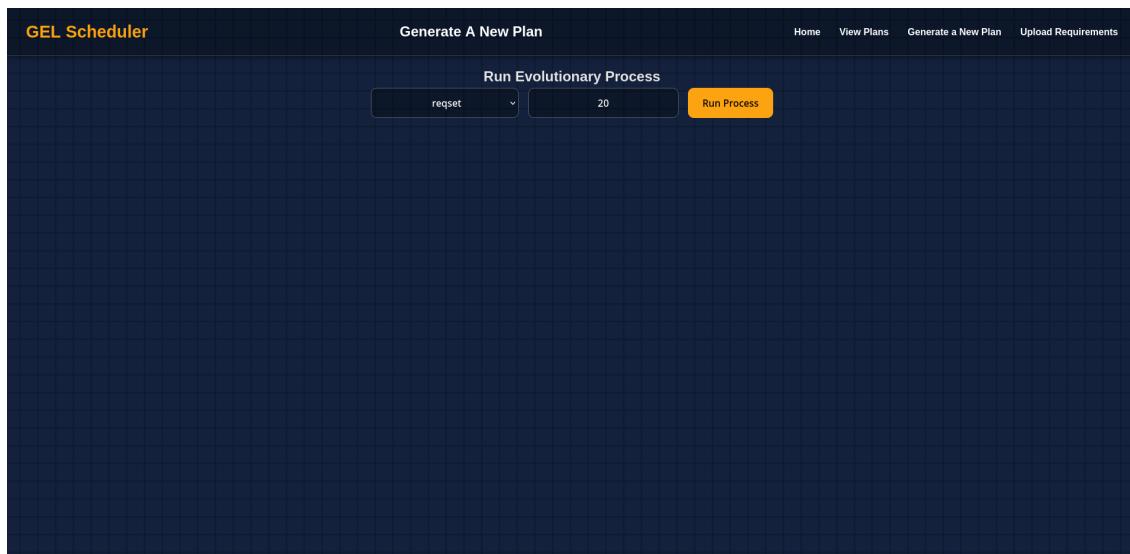
GEL Scheduler		Lesson View						Home		View Plans		Generate a New Plan		Upload Requirements		
		Plan generated using		IV_G		Select Teacher										
Time Slot	Monday	Tuesday	Wednesday		Thursday		Friday									
1	Geografia Geo1	J.Polski Jp1	J.Angielski Ja3	J.Angielski Ja8	Matematyka Mat5	Historia His3	Wych.Fiz. Wf2 Sala Gimnastyczna 1	Wych.Fiz. Wf3 Sala Gimnastyczna 2	13							
2	Geografia Geo1	J.Polski Jp1	Geografia Geo1		Matematyka Mat5	Wych.Fiz. Wf2 Sala Gimnastyczna 1	Wych.Fiz. Wf3 Sala Gimnastyczna 2	J.Polski Jp1	13							
3	J.Angielski Ja3	J.Angielski Ja8	Wych.Fiz. Wf2 Sala Gimnastyczna 1	Wych.Fiz. Wf3 Sala Gimnastyczna 2	J.Polski Jp1	Historia His3	Wych.Fiz. Wf2 Sala Gimnastyczna 1	Wych.Fiz. Wf3 Sala Gimnastyczna 2	1							
4	J.Francuski Jf1		Matematyka Mat5		Matematyka Mat5	J.Polski Jp1	Wych.Fiz. Wf2 Sala Gimnastyczna 1	Wych.Fiz. Wf3 Sala Gimnastyczna 2	4							
5	Matematyka Mat5	Religia Kat Rk1	Matematyka Mat5		Matematyka Mat5	J.Angielski Ja3	J.Angielski Ja8	J.Angielski Ja3	22							
6	J.Niemiecki Jn2	J.Rosyjski Jr1				J.Angielski Ja3	J.Angielski Ja8	J.Angielski Ja3	11							
7																

Rysunek 4.16: Podgląd planu z perspektywy klasy

Widok generowania planów lekcji

Widok umożliwia użytkownikowi określenie liczby generacji, jakie algorytm powinien wykonać podczas tworzenia planu lekcji. Dzięki temu można dostosować czas działania i potencjalną jakość rozwiązania do bieżących potrzeb.

Implementacja zaprezentowana na rysunku 4.17 spełnia wymagania funkcjonalne co do generowania planu lekcji.



Rysunek 4.17: Widok generowania planu lekcji

Widoki wprowadzania zasobów

Widoki wprowadzania zasobów pozwalają na tworzenie, i wybieranie ich zbiorów. Implementacje przedstawione na rysunkach 4.18, 4.19, 4.20 i 4.21 spełniają wymagania funkcjonalne co do dodawania zasobów.

Subject Name	Actions
J.Polski	x
J.Angielski	x
Historia	x
WOS	x
Etyka	x
Filozofia	x
Chemia	x
Fizyka	x
Informatyka	x
Biznes i zarządzanie	x
Religia	x

Rysunek 4.18: Widok dodawania przedmiotów

Teacher Name	Subjects (multi-select)	Actions
jp1	J.Polski J.Angielski Historia WOS Etyka Filozofia Chemia Fizyka Informatyka Biznes i zarządzanie Religia	x
jp2	J.Polski J.Angielski Historia WOS Etyka Filozofia Chemia Fizyka Informatyka Biznes i zarządzanie Religia	x
jp3	J.Polski J.Angielski Historia WOS Etyka Filozofia Chemia Fizyka Informatyka Biznes i zarządzanie Religia	x
jp4	J.Polski J.Angielski Historia WOS Etyka Filozofia Chemia Fizyka Informatyka Biznes i zarządzanie Religia	x

Rysunek 4.19: Widok dodawania nauczycieli

Room Name	Compatible Subjects (multi-select)	Actions
1	J.Polski J.Angielski Historia WOS	X
2	J.Polski J.Angielski Historia WOS	X
3	J.Polski J.Angielski Historia WOS	X
4	J.Polski J.Angielski Historia WOS Chemia Fizyka	X

Rysunek 4.20: Widok dodawania sal

Group Name	Description	Actions
I_A	psychol	X
I_B	jez-tur	X
I_C	społ-pr	X
I_D	polit	X
I_F	b-ch	X
I_G	eko-men	X
II_AC	psych-prawna	X
II_BG	jez-ekonomiczna	X
II_DF	polit-biol-chem	X
II_G	eko-men	X
III_A	dypł	X

Rysunek 4.21: Widok dodawania klas

Widoki wprowadzania wymagań

Widok wprowadzania wymagań głównych przedstawiony na rysunku 4.22, który przypomina arkusz kalkulacyjny pokazany na rysunku 3.1, co spełnia wymagania niefunkcjonalne. Górný wiersz przewija się razem ze stroną ułatwiając nawigację i wpisywanie wymagań. Podobnie jak widoki wprowadzania zasobów pozwala na wybranie zbioru wymagań głównych, co realizuje wymagania funkcjonalne co do pracy z wieloma zestawami zasobów.

Na podobnej zasadzie działają widoki wprowadzania dostępności nauczycieli i bloków przedmiotów przedstawione odpowiednio na rysunkach 4.23 i 4.24.

Rysunek 4.22: Widok dodawania wymagań głównych

Teacher	Day 1	Day 2	Day 3	Day 4	Day 5
Jp1	✓	✓	✓	✓	■
Jp2	✓	✓	✓	✓	✓
Jp3	✓	✓	✓	✓	✓
Jp4	✓	✓	✓	✓	✓
Jp5	✓	✓	✓	✓	✓
Ja1	✓	✓	✓	✓	✓
Ja2	✓	✓	✓	✓	✓
Ja3	✓	✓	✓	✓	✓
Ja4	✓	✓	✓	✓	✓
Ja5	✓	✓	✓	✓	✓
Ja6	✓	✓	✓	✓	✓
Ja7	✓	✓	✓	✓	✓
Ja8	✓	✓	✓	✓	✓

Rysunek 4.23: Widok wprowadzania dostępności nauczycieli

Create Lesson Blocks					
Block 1			Block 2		
Subjects		Applicable Groups			
J.Polski	J.angielski	Istoria	I_A, I_B, I_C, I_D, II_F, II_G, II_AC, II_BG, II_DF, III_G, III_A, III_B, III_C, III_D, III_F, III_G, IV_A, IV_B, IV_C, IV_D, IV_F, IV_G		
WOS	Etyka	Filozofia			
Chemia	Fizyka	Informatyka			
Biznes i zarządzanie	Religia	J.Niemiecki			
J.Francuski	J.Rosyjski z elem. Kult.	J.Rosyjski			
Biologia	Wych.Fiz.	EdB			
Matematyka	Geografia	WDZ			
Zajęcia kreatywne	Civil Society	Creative Writing			
Subjects		Applicable Groups			
		I_A, I_B, I_C, I_D			

Rysunek 4.24: Widok dodawania bloków przedmiotów

4.3.5. Integracja z algorytmem

Dzięki adekwatnemu wyborowi technologii, proces integracji modułu optymalizacyjnego z warstwą serwerową został znaczco uproszczony. Zaimplementowałem dedykowany punkt końcowy, który przyjmuje żądania typu POST zawierające parametry konfiguracyjne algorytmu, w tym liczbę generacji dla algorytmu ewolucyjnego oraz ID zbioru wymagań. Dalszy proces jest opisany na diagramie sekscencji ??

5. Weryfikacja i walidacja rozwiązania

Rozdział stanowi krytyczną ocenę wdrożonego systemu, podzieloną na dwie kluczowe perspektywy:

- testowanie funkcjonalności — weryfikacja, czy aplikacja działa zgodnie z założeniami projektowymi. Zawiera formalne scenariusze testowe oraz ich praktyczną realizację,
- analiza jakości wyniku — walidacja, czy wygenerowane plany lekcji są użyteczne i poprawne jakościowo. Obejmuje porównanie z planem ręcznym, analizę krytycznych przypadków oraz ewaluację kluczowego wskaźnika efektywności, liczby okienek nauczycieli.

5.1. Scenariusze testowe

5.1.1. Scenariusz testowy generowania planu lekcji z określonymi parametrami

Scenariusz zaprezentowany w tabeli 5.1 opisuje proces wygenerowania planu lekcji przy użyciu zaimplementowanego modułu optymalizacyjnego.

5.1.2. Scenariusz przeglądu planów lekcji

Scenariusz zaprezentowany w tabeli 5.2 opisuje proces przeglądania istniejących planów lekcji wygenerowanych w systemie.

5.1.3. Scenariusz importowania i edytowania wymagań głównych

Scenariusz opisuje proces masowego importu wymagań głównych oraz ich późniejszej ręcznej edycji.

5.1.4. Scenariusz edytowania dostępności nauczycieli

Scenariusz opisuje proces modyfikacji dostępności nauczycieli po wcześniejszym zdefiniowaniu wymagań głównych.

Tabela 5.1: Generowanie planu lekcji z określonymi parametrami

Scenariusz	Generowanie planu lekcji z określonymi parametrami
Opis	Test weryfkuje poprawność procesu generowania planu lekcji przy zadanej liczbie generacji oraz pełnym zestawie danych wejściowych.
Widok początkowy	Strona główna
Wymagania	Wprowadzone zasoby: nauczyciele, klasy, sale, przedmioty Zdefiniowane wymagania główne dla każdej klasy Wprowadzone dostępności nauczycieli Zdefiniowane bloki przedmiotowe
Kroki testowe	1. Przejdź do widoku generowania planu lekcji 2. Wprowadź liczbę generacji 3. Wybierz dostępny zestaw ograniczeń 4. Uruchom proces generowania planu
Oczekiwany rezultat	System generuje kompletny plan lekcji zgodny ze wszystkimi ograniczeniami. System pokazuje użytkownikowi powiadomienie o zakończeniu procesu.

Tabela 5.2: Scenariusz przeglądu planów lekcji

Scenariusz	Przegląd planów lekcji
Opis	Test weryfkuje możliwość przeglądania wielu wygenerowanych planów lekcji oraz podglądu ich szczegółów z perspektywy klasy i nauczyciela.
Widok początkowy	Strona główna
Wymagania	Wprowadzone zasoby: nauczyciele, klasy, sale, przedmioty W systemie dostępne są co najmniej dwa wygenerowane plany lekcji
Kroki testowe	1. Przejdź do widoku przeglądania planów lekcji 2. Wybierz najnowszy dostępny plan 3. Wyświetl plan lekcji dla wybranej klasy 4. Wyświetl plan lekcji dla konkretnego nauczyciela
Oczekiwany rezultat	Użytkownik może poprawnie przeglądać różne warianty planów lekcji oraz widoki szczegółowe, a system prezentuje dane zgodnie z zapisanym stanem.

Tabela 5.3: Scenariusz importowania i edytowania wymagań głównych

Scenariusz	Importowanie i edytowanie wymagań głównych
Opis	Test weryfkuje poprawność importu danych z pliku CSV oraz poprawność procesu edycji wymagań godzinowych dla klas i nauczycieli.
Widok początkowy	Strona główna
Wymagania	Wprowadzone zasoby: nauczyciele, klasy, przedmioty Dostępny plik CSV zawierający wymagania główne
Kroki testowe	<ol style="list-style-type: none"> 1. Przejdź do widoku dodawania wymagań głównych 2. Wybierz odpowiednie zbiory zasobów 3. Przeciągnij plik CSV do aplikacji 4. Edytuj wymagania godzinowe dla wybranej klasy i nauczyciela 5. Zapisz zmiany
Oczekiwany rezultat	System poprawnie importuje dane z pliku CSV oraz umożliwia ich edycję i zapisanie nowych wartości.

Tabela 5.4: Scenariusz edytowania dostępności nauczycieli

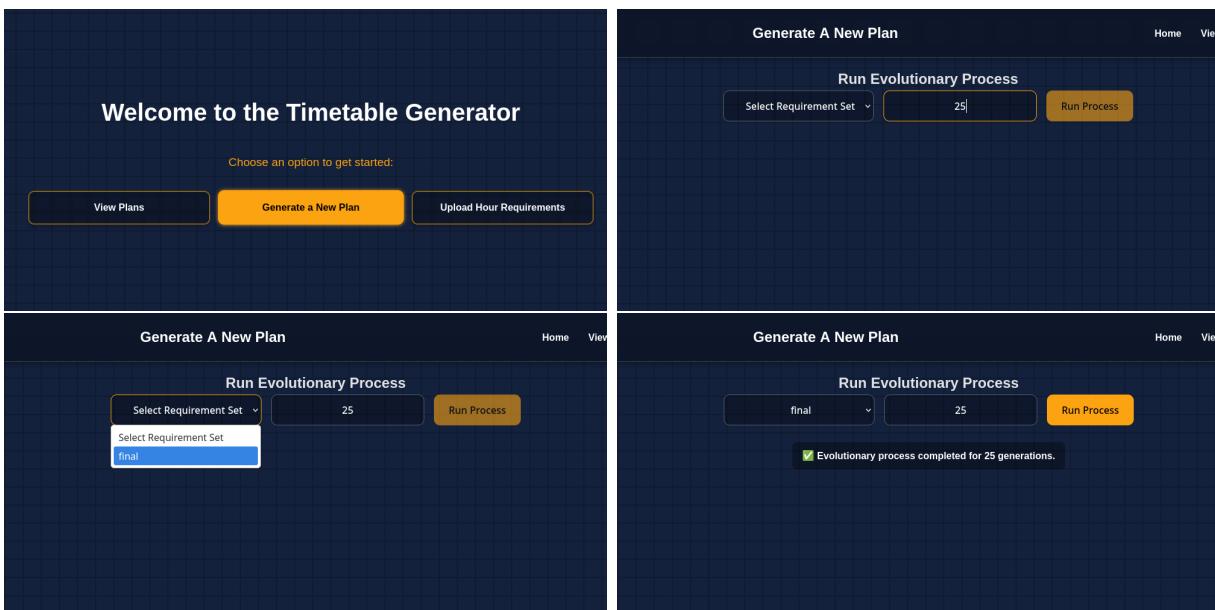
Scenariusz	Edytowanie dostępności nauczycieli
Opis	Test sprawdza możliwość modyfikacji godzin dostępności nauczycieli z poziomu odpowiedniego widoku systemu.
Widok początkowy	Widok wprowadzania wymagań głównych
Wymagania	Wprowadzone zasoby: nauczyciele
Kroki testowe	<ol style="list-style-type: none"> 1. Rozwiń wstążkę nawigacji 2. Przejdź do widoku dostępności nauczycieli 3. Wybierz dostępny zbiór ograniczeń 4. Edytuj dostępność wybranego nauczyciela 5. Zapisz zmiany
Oczekiwany rezultat	System zapisuje zmodyfikowane dane dostępności. System pokazuje powiadomienie o powodzeniu operacji

5.2. Testowanie aplikacji

5.2.1. Test generowania planu lekcji z określonymi parametrami

Test potwierdza poprawność procesu generowania planu zgodnie ze scenariuszem przedstawionym w tabeli 5.1. Jak widać na rysunku 5.1 aplikacja pozwala użytkownikowi na zdefiniowanie liczby generacji i spełnia wszystkie wymagania. Kolejno wykonywane kroki to:

- przejdźcie do widoku generowania planów lekcji,
- wpisywanie liczby generacji,
- wybieranie zbioru ograniczeń,
- uruchomienie procesu oraz otrzymane powiadomienie.

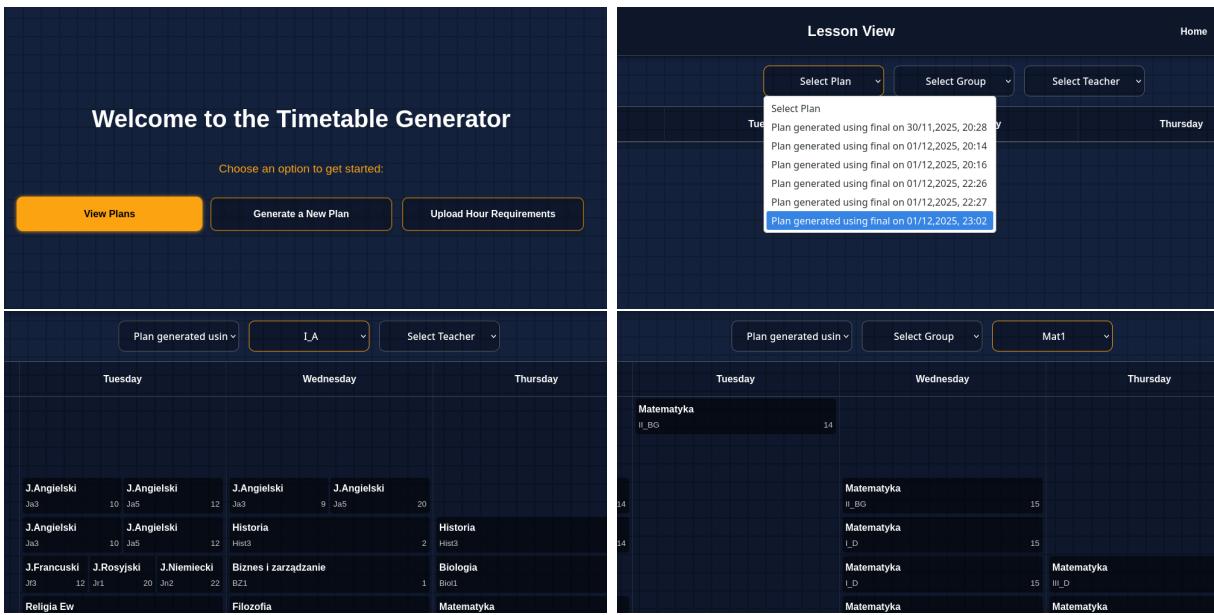


Rysunek 5.1: Zrzuty ekranów wykonanych kroków dla scenariusza generowania planu lekcji

5.2.2. Test przeglądania planów lekcji

Test weryfkuje realizację wymagań dotyczących wieloperspektywnego wglądu w plan. Na rysunku 5.2 przedstawiłem proces testowania tej funkcjonalności zgodnie ze scenariuszem przedstawionym w tabeli 5.2. Użytkownik kolejno:

- przechodzi do widoku przeglądania planów lekcji,
- wybiera najnowszy dostępny plan,
- wybiera perspektywę konkretnej klasy,
- wybiera perspektywę konkretnego nauczyciela.

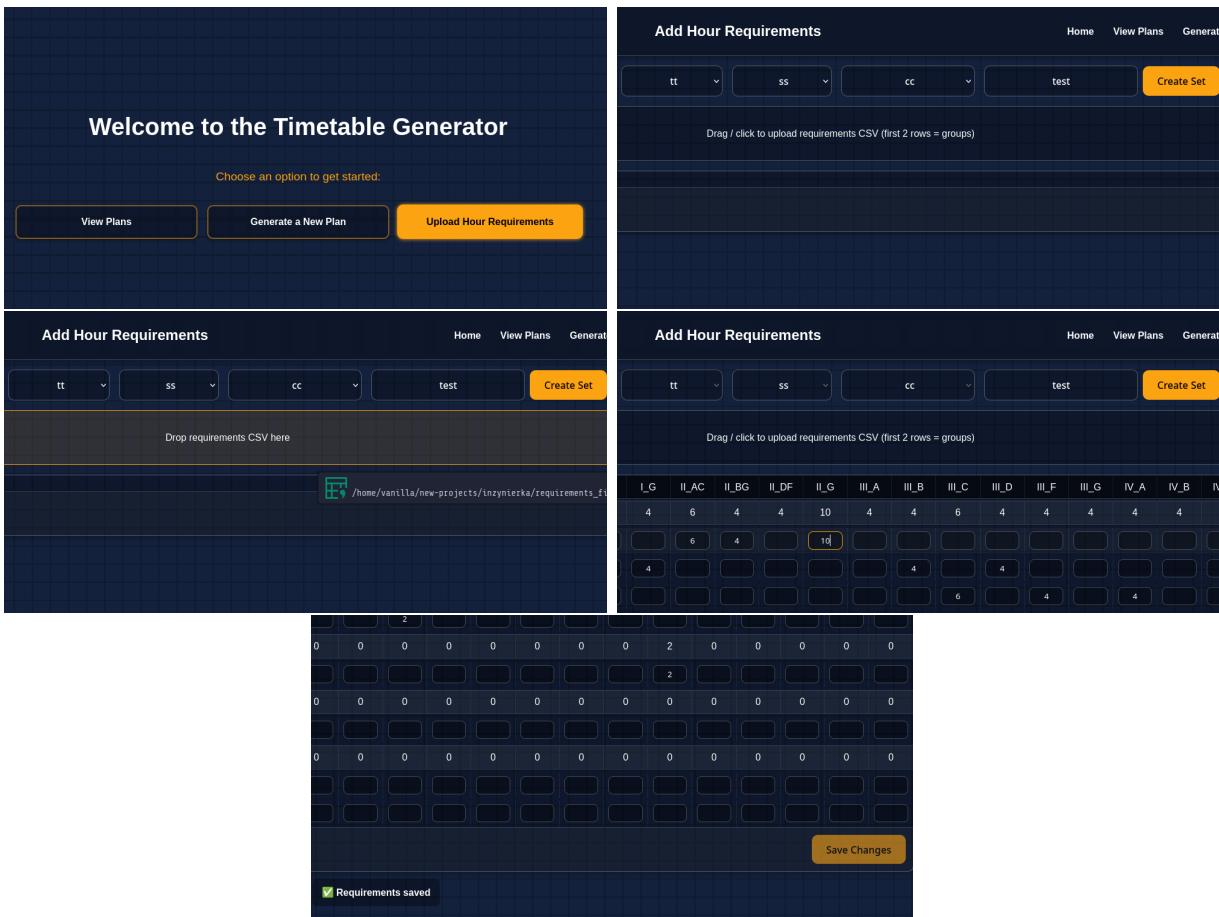


Rysunek 5.2: Zrzuty ekranów wykonanych kroków dla scenariusza przeglądania planów lekcji

5.2.3. Test importowania i edytowania wymagań głównych

Test weryfikuje możliwość importowania wymagań głównych z pliku csv oraz możliwość ich modyfikowania. Na rysunku 5.3 przedstawiłem kroki użytkownika testującego aplikację zgodnie ze scenariuszem przedstawionym w tabeli 5.3. Użytkownik kolejno:

- przychodzi do widoku dodawania wymagań głównych,
- wybiera odpowiednie zbiory zasobów,
- importuje plik csv,
- edytuje pojedyncze wymaganie główne,
- zapisuje zmiany i dostaje powiadomienie.

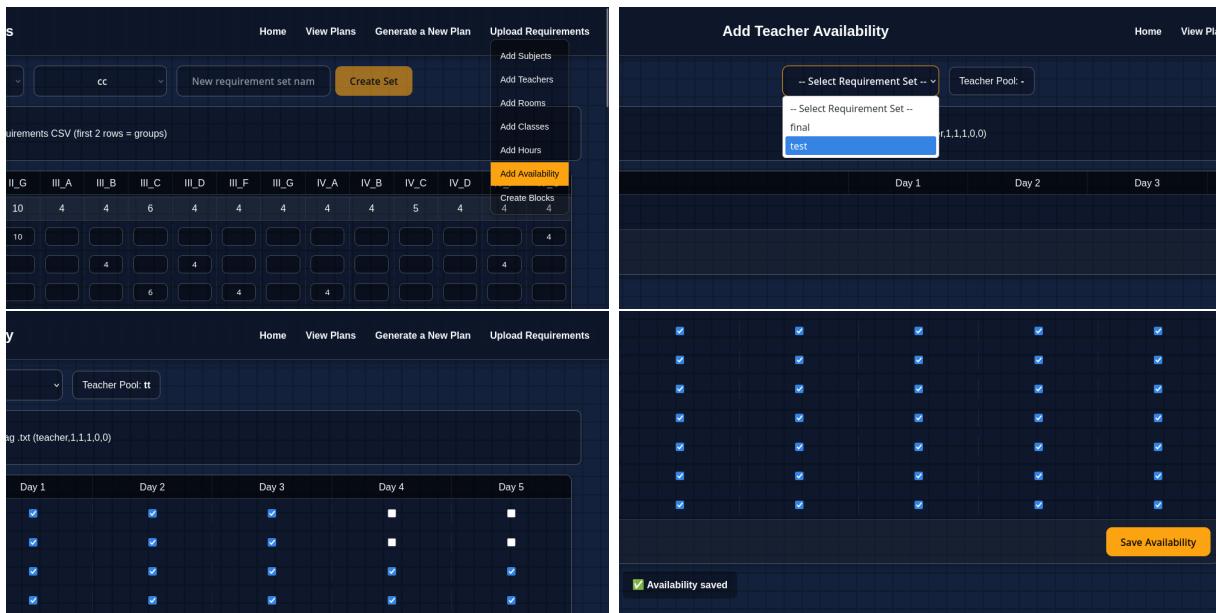


Rysunek 5.3: Zrzuty ekranów wykonanych kroków dla scenariusza importowania i edytowania wymagań głównych

5.2.4. Test edytowania dostępności nauczycieli

Test sprawdza poprawność modyfikacji dostępności zgodnie ze scenariuszem przedstawionym w tabeli 5.4. Rysunek 5.3 przedstawia proces:

- wybrania widoku edytowania dostępności nauczycieli,
- wybierania odpowiedniego zbioru wymagań,
- edytowania dostępności pierwszych dwóch nauczycieli,
- zapisania wyniku oraz otrzymania powiadomienia o sukcesie operacji.



Rysunek 5.4: Zrzuty ekranów wykonanych kroków dla scenariusza edytowania dostępności nauczycieli

5.3. Analiza jakości wyników generowania planu lekcji

5.3.1. Omówienie danych wejściowych

Dane przekazane przez placówkę oświatową pochodzą z początkowej fazy procesu układania planu lekcji i uległy zmianom po uzyskaniu pełnych informacji o uczniach. W szczególności nie obejmują one lekcji wychowawczych (LW) oraz zajęć dodanych na końcowym etapie tworzenia planu (np. EO, EZ, ...). W konsekwencji nie było możliwe pełne zamodelowanie tych elementów ani uwzględnienie ich podczas optymalizacji planu.

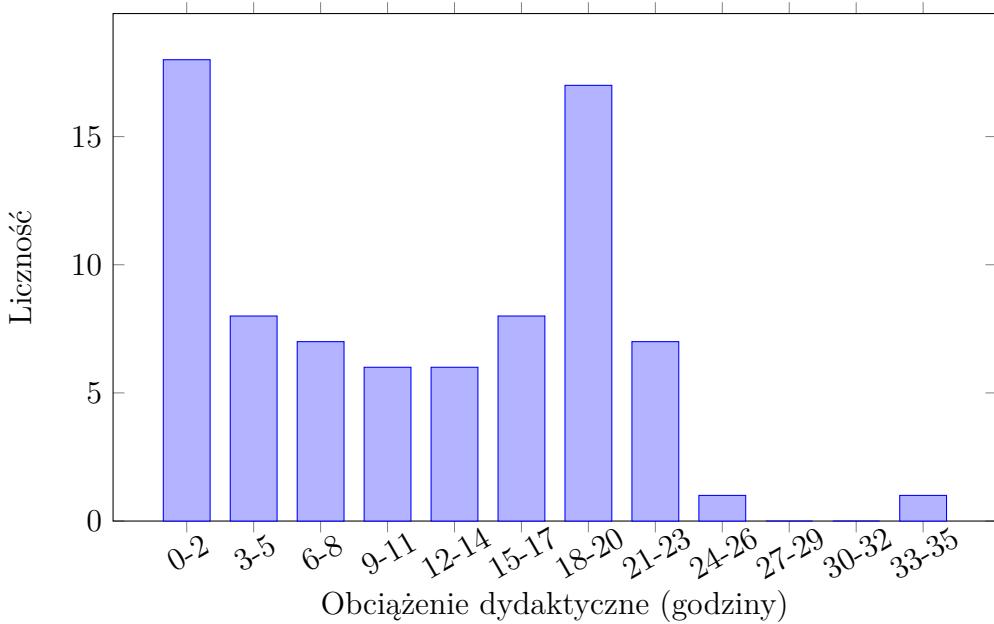
Istotnym ograniczeniem była również kwestia ochrony danych osobowych nauczycieli — nie uzyskałem dostępu do imion i nazwisk, co uniemożliwiło identyfikację sytuacji, w których jeden nauczyciel prowadzi więcej niż jeden przedmiot. Tym samym nie można było zamodelować potencjalnych konfliktów wynikających z równoległego prowadzenia zajęć w różnych klasach przez tego samego nauczyciela.

5.3.2. Omówienie całego planu

Plan omówiony poniżej został wygenerowany używając osobnika przedstawionego w punkcie 3.5.7.

Ocenianie jakości wyników należy zacząć od analizy obciążen dydaktycznych. Na rysunku 5.5 przedstawiono histogram wymaganej tygodniowej liczby godzin lekcyjnych dla nauczycieli. Analogicznie, na rysunku 5.6 zaprezentowano histogram liczby obowiązkowych godzin lekcyjnych dla klas.

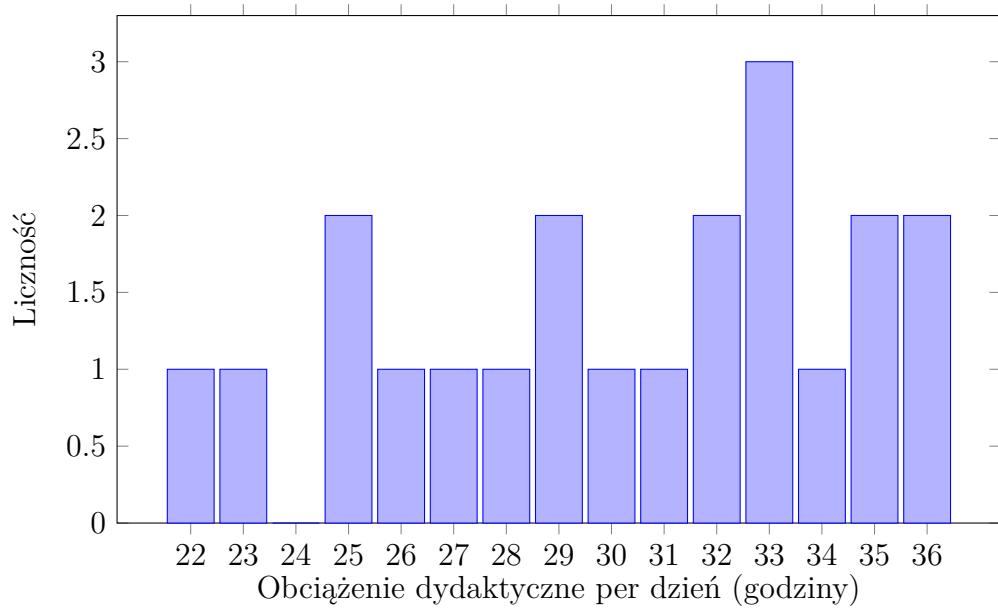
Z przedstawionych histogramów wynika, że wielu nauczycielom przypisano tygodniowe obciążenie równe zero. Wynika to z faktu, że szkoła dysponuje większą liczbą nauczycieli, niż



Rysunek 5.5: Histogram obciążeń dydaktycznych nauczycieli tygodniowo

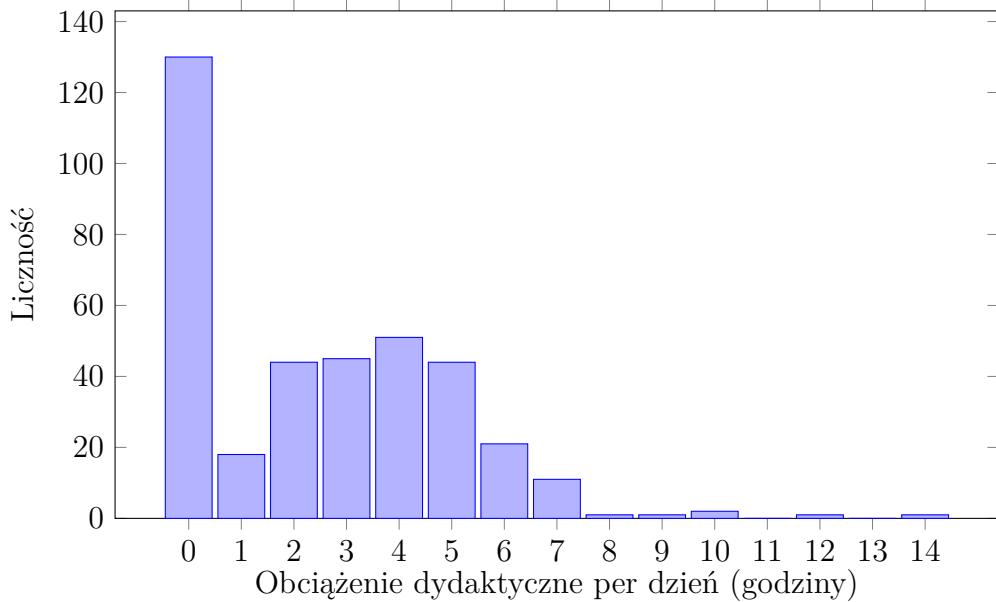
jest w praktyce potrzebne do realizacji aktualnej liczby zajęć. W rezultacie wiele nauczycieli nie zostaje przypisanych do prowadzenia lekcji. Ponadto przez anonimowość nazwisk nauczycieli, niektórzy nauczyciele zostali w modelu sztucznie rozbici na dwie lub trzy osoby. Powoduje to sztuczne zaniżenie wymaganej tygodniowo liczby godzin od poszczególnych nauczycieli.

Z kolei, ze względu na nieuwzględnienie części lekcji opisanych wcześniej, wymagana tygodniowa liczba godzin dla klas jest również sztucznie zaniżona. Mimo tego można zaobserwować, że niektóre klasy mają znacząco niższe obciążenia tygodniowe niż pozostałe. Wynika to przede wszystkim z faktu, że klasy maturalne realizują istotnie mniejszą liczbę zajęć niż, przykładowo, klasy pierwsze.



Rysunek 5.6: Histogram obciążen dydaktycznych uczniów tygodniowo

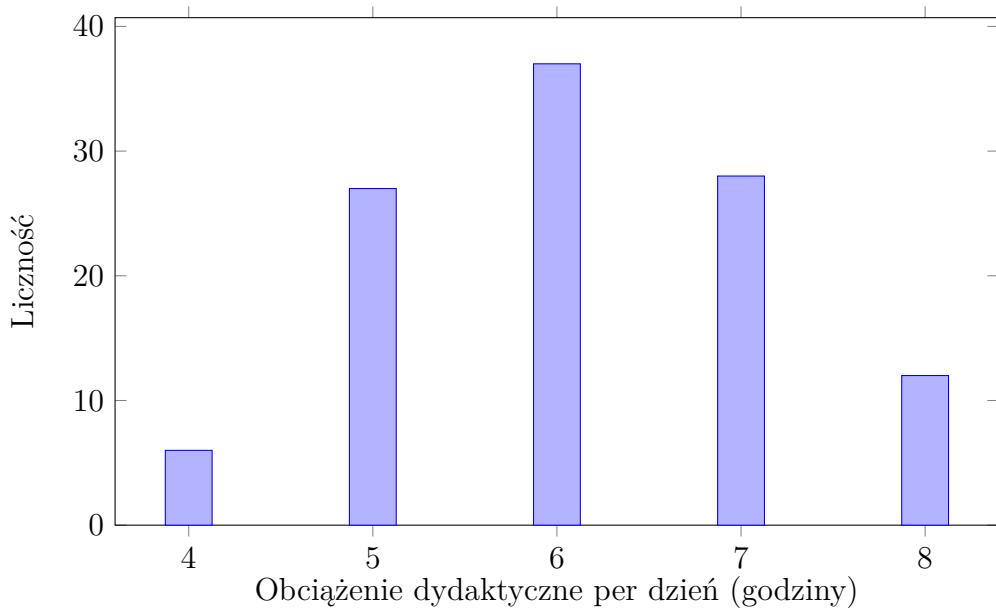
Biorąc pod uwagę te obserwacje, możemy przejść do analizy dziennego obciążenia dydaktycznego. Na rysunkach 5.7 oraz 5.8 przedstawiono odpowiednio histogram dziennych obciążeń nauczycieli i klas.



Rysunek 5.7: Histogram obciążień dydaktycznych nauczycieli

Z analizy wyników wynika, że nauczyciele mają liczne dni wolne. Jest to zamierzony efekt, osiągnięty dzięki modyfikacji funkcji przystosowania w algorytmie ewolucyjnym. Niestety, ze względu na zastosowane wagie $\theta_T = 1$ i $\theta_G = 2$, algorytm w większym stopniu optymalizował obciążenia klas niż nauczycieli. W rezultacie pojawiły się dwa przypadki, w których nauczyciel ma zaplanowane 12 lub 14 godzin zajęć w jednym dniu.

Zastosowane wagie spowodowały jednocześnie, że rozkład obciążień uczniów jest bardzo korzystny. Nie pojawiają się wartości skrajne ani nieakceptowalne. Minimalne obciążenie, wynoszące 4 godziny, dotyczy klas maturalnych, które mają stosunkowo niską liczbę wymaganych godzin tygodniowo (22 lub 23). Wszystkie pozostałe wyniki mieszczą się w przedziale od 4 do 8 godzin dziennie, co umożliwia tworzenie praktycznych i realistycznych planów lekcji.



Rysunek 5.8: Histogram obciążen dydaktycznych uczniów

Aby ocenić jakość wygenerowanych harmonogramów, przeanalizowałem także liczbę okienek nauczycieli w planie zoptymalizowanym pod kątem minimalizacji ich liczby oraz w planie, który jedynie spełnia wszystkie ograniczenia, lecz nie został poddany optymalizacji. Wyniki zestawiono w tabeli 5.5. Ze względu na brak dostępu do szczegółowych danych dotyczących planu ułożonego ręcznie przez szkołę możliwe było jedynie porównanie wariantów wygenerowanych przez algorytm.

Tabela 5.5: Statystyki dla generowanych planów

Dzień tygodnia	Pon	Wt	Śr	Czw	Pt	Całosciowo
Liczba bloków	132	133	133	135	140	673
Liczba okienek przy minimalizacji	65	41	68	59	72	305
Liczba okienek przy minimalizacji per nauczyciel	1.05	0.98	1.10	0.95	1.16	5.24
Liczba okienek bez minimalizacji	172	157	141	139	133	742
Liczba okienek bez minimalizacji per nauczyciel	2.77	2.53	2.27	2.24	2.15	11.96

5.3.3. Omówienie szczególnych przypadków

Szczególnej analizie poddałem klasę o profilach łączonych, ponieważ to właśnie dla nich ręczne ułożenie planu jest najbardziej wymagające. W moim zbiorze danych klasa IIAC łączy profil psychologiczny z prawnym, co oznacza konieczność równoległego prowadzenia bloków lekcyjnych dotyczących obu ścieżek kształcenia. Wygenerowany plan lekcji dla tej klasy przedstawiłem na rysunku 5.9. Dla porównania, na rysunku 5.10 zaprezentowałem plan opracowany ręcznie przez szkołę.

Analizując wygenerowany plan można zauważyc, że algorytm poprawnie odwzorował wymagane bloki lekcyjne, takie jak jednocześnie prowadzenie języka biologii i wiedzy o społeczeństwie, a także prawidłowo przydzielił sale o specjalnym przeznaczeniu (sale gimnastyczne, pracownie informatyczne, ...). Plan charakteryzuje się również równomiernym rozkładem obciążen dydaktycznych w ciągu tygodnia.

Time	Slot	Monday	Tuesday	Wednesday	Thursday	Friday
1					J.Polski Jp1	
2		J.Angielski Jr1	J.Angielski Ja1		J.Polski Jp1	J.Polski Jp1
3		J.Rosyjski Jr1	J.Angielski Ja1	Biznes i zarządzanie BZ1	J.Polski Jp1	J.Polski Jp1
4		Biology Bio2		J.Rosyjski Jr1	J.Angielski Ja1	Matematyka Mat4
5		J.Angielski Ja1	Informatyka Inr1 Sala informatyczna 2	J.Rosyjski Jr1	J.Angielski Ja7	Wych.Fiz. Wf2 Sala Gimnastyczna 1
6		Biology Bio2	WOS WO3	Biznes i zarządzanie BZ1	J.Polski Jp1	Wych.Fiz. Wf2 Sala Gimnastyczna 1
7		J.Polski Jp1		Biologia Biol2	Religia Kat Rel1	Wych.Fiz. Wf4
8		Historia Hist1		J.Angielski Ja7	Matematyka Mat4	Wych.Fiz. Wf4
9		Historia Hist1		J.Rosyjski Jr1	Chemia Chem2	Wych.Fiz. Wf4
10		Zajęcia kreatywne Zk2		Biologia Biol2	WOS WO3	Wych.Fiz. Wf4
11				Historia Hist1		Wych.Fiz. Wf4

Rysunek 5.9: Wygenerowany plan lekcji dla klasy IIAC

Nr	Pn	Wt	Śr	Cz	Pt
7.10 - 7.55				Rel.ew/ Mr/ 13/	
8.00 - 8.45	Et/WF Dc/Pi 14/		Inf/Ang PH/Wa 11/37	Geo IM 2	WF/WF BH/Pi
8.50 - 9.35	Mat AG 16		Ang/Inf IP/PH 36/11	His IM 2	WF/WF BH/Pi
9.40 - 10.25	Fiz MH 15	WF/ BH/	Pol AT 13	His IM 2	Pol AT 13
10.35 - 11.20	Pol AT 13	Biz Gr 02	Pol AT 13	Ros/Ang PM/IP 33/35	Pol AT 13
11.30 - 12.15	Pol AT 13	Ang/Ros Wa/PM 37/33	Biol/Wos BW/DC 25/7	Ros/Ang PM/IP 33/35	EO IM 2
12.25 - 13.10	Geo IM 2	Ang/Ros Wa/PM 37/33	Mat AG 16	Chem JP 6	EO IM 2
13.25 - 14.10	His IM 2	Biol BW 25	Mat AG 16	Chem JP 6	Biol BW 25
14.15 - 15.00	EZ/ Sc/ 25/	Mat AG 16	Biz Gr 2	Zaj. Kształ. Kr IM 2	Biol/Wos BW/DC 25/24
15.05 - 15.50		LW IM 2			Rel.kat./ Br/ 24/

Rysunek 5.10: Plan lekcji dla klasy IIAC stworzony przez szkołę

Kolejnym istotnym aspektem jest prawidłowe zaplanowanie zajęć prowadzonych jednocześnie dla wielu klas. Najczęściej występują tu bloki językowe obejmujące język francuski, niemiecki oraz rosyjski. Przykład takiego bloku, realizowanego w środy, przedstawiono na rysunku 5.11.

Wyniki potwierdzają wysoką skuteczność algorytmu — średnio około pięciu okienek w ciągu całego tygodnia na nauczyciela można uznać za wynik satysfakcjonujący.

5.3.4. Wydajność generowania planu lekcji

Testy wydajnościowe przeprowadzono na następującej konfiguracji sprzętowej:

- **procesor:** AMD Ryzen 7 2700X (16 wątków), 3.875 GHz,
- **karta graficzna:** AMD Radeon RX 7700 XT,
- **pamięć RAM:** 16GB,
- **system operacyjny:** Linux 6.17.9.arch1-1.

Przy czym warto zaznaczyć, że algorytm nie korzysta z karty graficznej.

Czas generowania bloków lekcyjnych jest znacznie mniejszy w porównaniu do pozostałych etapów (poniżej jednej sekundy). Średni czas działania algorytmu ewolucyjnego, obliczony na podstawie pięciu niezależnych prób, wyniósł 95 sekund. Następnie wygenerowanie pełnego planu zajęć wraz z minimalizacją liczby okienek zajęło 6 minut i 52 sekundy.

Łączny czas potrzebny do uzyskania kompletnego planu lekcji dla rzeczywistych danych wyniósł zatem około 8.45 minuty. Jest to bardzo dobry rezultat, biorąc pod uwagę złożoność problemu, liczne zależności oraz obecność klas o profilach łączonych. Ponadto czas ten mieści się w wymaganym limicie 10 minut, określonym w wymaganiach niefunkcjonalnych systemu.

Time Slot	Wednesday			Wednesday			Wednesday			Wednesday			Wednesday			Wednesday		
1	J.Rosyjski	J.Niemiecki	J.Rosyjski	J.Francuski	J.Niemiecki	J.Rosyjski	J.Francuski	J.Niemiecki	J.Rosyjski	J.Francuski	J.Niemiecki	J.Rosyjski	J.Francuski	J.Niemiecki	J.Rosyjski	J.Francuski	J.Niemiecki	J.Rosyjski
2	J.Rosyjski	J.Niemiecki	J.Rosyjski	J.Francuski	J.Niemiecki	J.Rosyjski	J.Francuski	J.Niemiecki	J.Rosyjski	J.Francuski	J.Niemiecki	J.Rosyjski	J.Francuski	J.Niemiecki	J.Rosyjski	J.Francuski	J.Niemiecki	J.Rosyjski
3	Chem1	Sala do Chemicznej 1	His12	Informatyka	Historia	1	Informatyka	Sala Informatyczna 2	Biol2	Biol2	Fiz1	Fizyka	Fiz1	21	Fizyka	Fiz1	21	Fizyka
4	Wych.Fiz.	Wych.Fiz.	Geografia	Informatyka	Chemia	Chem1	Geografia	Sala Informatyczna 2	Chem1	Chem1	Chem1	Chemia	Chem1	Chem1	Geografia	Geografia	Geografia	Geografia
5	J.Francuski	J.Francuski	Fizyka	J.Angielski	J.Angielski	J.Angielski	Fizyka	J.Angielski	J.Angielski	J.Angielski	J.Angielski	Informatyka	Informatyka	Informatyka	Religia Kat	Religia Kat	Religia Kat	Religia Kat
6	Religia Kat	Religia Kat	J.Polski	Religia Kat	Religia Kat	Religia Kat	J.Polski	Religia Kat	Religia Kat	Religia Kat	Religia Kat	Religia Kat	Religia Kat	Religia Kat	Religia Kat	Religia Kat	Religia Kat	Religia Kat
7	Geografia	Geografia	J.Polski	Matematyka	J.Angielski	J.Angielski	J.Polski	Matematyka	J.Angielski	J.Angielski	J.Angielski	J.Angielski	J.Polski	J.Polski	J.Polski	J.Polski	J.Polski	J.Polski
8	Geografia	Geografia	J.Angielski	Biologia	Matematyka	Matematyka	J.Angielski	Biologia	Matematyka	Matematyka	Matematyka	Matematyka	Wych.Fiz.	Wych.Fiz.	Wych.Fiz.	Wych.Fiz.	Wych.Fiz.	Wych.Fiz.
9	Geo1	Geo1	Ja4	Biol1	Mat6	Mat6	Ja4	Biol1	Mat6	Mat6	Mat6	Mat6	Wys3 Sala Gimnastyczna 1					

Rysunek 5.11: Zestawienie środowych planów lekcji dla kolejno klas IIIA, IIIC, IIID, IIIF i IIIG

6. Wnioski i perspektywy rozwoju

6.1. Problemy rozwiązania

Choć system realizuje wszystkie cele, które założyłem w tej pracy, jego obecna wersja posiada ograniczenia, które mogą utrudniać wykorzystanie w warunkach rzeczywistych.

Pierwszym problemem jest obsługa bloków lekcyjnych obejmujących tylko część klasy. W praktyce takie zajęcia powinny być umieszczane wyłącznie na początku lub końcu dnia, aby uniknąć powstawania okienek dla uczniów. System nie narzuca jeszcze tego warunku, co może prowadzić do niepożądanych okienek.

Uproszczony jest również mechanizm przypisywania sal. Obecnie wykorzystuje on tzw. białe listy obsługiwanych przedmiotów, co jest rozwiązaniem poprawnym, ale mało elastycznym. W wielu sytuacjach potrzebne jest rozróżnienie między salą „dozwoloną” a „preferowaną”. Przykładowo nic nie stoi na przeszkodzie, aby w sytuacji w której brakuje sal zajęcia języka polskiego odbyły się w sali do matematyki; natomiast zajęcia wychowania fizycznego mogą odbyć się tylko i wyłącznie w dedykowanych salach.

Kolejne ograniczenie dotyczy dostępności nauczycieli, które są definiowane w skali dnia. W rzeczywistości dostępności powinny być opisane per slot czasowy.

Ograniczenia dotyczą również samego procesu generowania planów. Algorytm ewolucyjny stosowany do alokacji bloków lekcyjnych pomiędzy dniami działa poprawnie, jednak jego natura heurystyczna powoduje, że nie gwarantuje optymalności.

Ponadto generowanie planów lekcji jest realizowane lokalnie (niezależnie dla każdego dnia) co redukuje złożoność obliczeniową, ale może prowadzić do utraty globalnej (tygodniowej) optymalności.

6.2. Perspektywy rozwoju

Dalszy rozwój aplikacji powinien w pierwszej kolejności skoncentrować się na rozwiązaniu problemów zidentyfikowanych w poprzedniej sekcji. Ich eliminacja jest kluczowa, aby system mógł zostać wykorzystany w realnych warunkach szkolnych.

Jednym z naturalnych kierunków rozbudowy jest ponowna analiza sposobu przydzielania lekcji do dni tygodnia. Obecne podejście heurystyczne jest szybkie, jednak zastosowanie metod programowania całkowitoliczbowego mogłoby zapewnić rozwiązania bliższe optymalnym, szczególnie przy bardziej wymagających konfiguracjach danych.

Warto również rozszerzyć model sal lekcyjnych poprzez wprowadzenie funkcji kar i nagród. Pozwoliłoby to nie tylko uwzględnić twarde ograniczenia, ale też modelować preferencje. Przykładowo ograniczyć korzystanie z nieoptimalnych pomieszczeń lub promować nowoczesne sale.

Kolejnym istotnym elementem jest zmiana sposobu reprezentacji dostępności nauczycieli. Zamiast ograniczeń na poziomie dni, powinny być one definiowane w skali slotów czasowych.

Integracja takich informacji w ostatnim etapie algorytmu umożliwia bardziej realistyczne planowanie.

Wartym analizy kierunkiem rozwoju jest również dodanie czwartego etapu generowania planu lekcji. Móglby on polegać na zebraniu dziennych harmonogramów i potraktowaniu ich jako punktu wyjścia dla solwera programowania liniowego z ograniczeniami. Ponieważ bazowe rozwiązanie jest już poprawne, dodatkowa optymalizacja nie powinna znacząco zwiększyć czasu obliczeń, a może wyraźnie poprawić globalną jakość tygodniowego planu.

Poza stroną algorytmiczną, dalsze prace wymagają rozbudowy warstwy użytkowej. System powinien obsługiwać wielu użytkowników, co wymaga implementacji kont opartych o przykładowo adresy e-mail i hasła. Pozwoliłoby to na wdrożenie aplikacji na rynku.

Interfejs aplikacji również powinien zostać znacząco rozwinięty. Obecna wersja skupia się na funkcjonalności, natomiast pełnoprawne wdrożenie wymaga bardziej intuicyjnego, przejrzystego i przyjaznego użytkownikowi UI. W szczególności konieczne jest zapewnienie narzędzi do ręcznej edycji wygenerowanego planu — poprawy sal, wymiany pojedynczych lekcji czy reorganizacji fragmentów tygodnia. Tego typu funkcjonalność jest niezbędna, ponieważ nawet najlepszy algorytm nie jest w stanie uwzględnić wszystkich specyficznych preferencji i wyjątków charakterystycznych dla danej szkoły.

6.3. Wnioski

Zastosowany algorytm zachłanny do generowania bloków lekcyjnych okazał się trafnym wyborem. Jego prostota znacząco ułatwia konstrukcję ograniczeń i eliminuje potrzebę stosowania bardziej złożonych metod. Pozwala również na pojedyncze wywołanie funkcji `NoOverlap` dla danej klasy, ponieważ bloki lekcyjne, w przeciwieństwie do pojedynczych zajęć, nie mogą na siebie zachodzić (przykładowo francuski i niemiecki). Dzięki temu proces tworzenia harmonogramu jest bardziej przejrzysty i obliczeniowo efektywny.

Uzyskany plan lekcji, choć wymagałby pewnych ręcznych korekt, jest wystarczająco bliski rozwiązaniu praktycznemu. Podejście które przedstawiłem, oparte na podziale problemu na trzy mniejsze podproblemy, okazało się efektywne przy dostępnej strukturze danych wejściowych. Koncepcja ta jest na tyle dobrze uzasadniona i elastyczna, że z powodzeniem może być stosowana również w przyszłych działaniach i projektach.

Praca zawiera propozycję wieloetapowego modelu generowania planu lekcji. Pokazałem, że takie połączenie umożliwia uzyskanie poprawnych rozwiązań mimo złożonych zależności między zajęciami. Eksperymenty wykazały, że rozdzielenie problemu na trzy podproblemy znacząco poprawia efektywność obliczeń i upraszcza formułowanie ograniczeń.

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] B. P. Carneiro. Django-tenants documentation, (2025). Ostatnio otworzono 27 listopada 2025.
- [5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2022.
- [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] D. S. Foundation. The web framework for perfectionists with deadlines., (2025). Ostatnio otworzono 27 listopada 2025.
- [9] Google. Documentation, or-tools - addnooverlap, (2025). Ostatnio otworzono 25 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] Meta. React, (2025). Ostatnio otworzono 27 listopada 2025.
- [13] 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.
- [14] 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.

- [15] Vulcan. Plan lekcji optivum. układanie planu lekcji - krok po kroku, (2025). Ostatnio otworzono 25 listopada 2025.
- [16] 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	Zrzut ekranu importowania arkusza organizacyjnego do programu „Plan lekcji Optivum” [15]	10
2.2	Zrzut ekranu wprowadzania sal do programu „Plan lekcji Optivum” [15]	10
2.3	Zrzut ekranu wprowadzania preferencji sal względem przedmiotów do programu „Plan lekcji Optivum” [15]	11
2.4	Zrzut ekranu wprowadzania preferencji sal względem nauczycieli do programu „Plan lekcji Optivum” [15]	11
2.5	Zrzut ekranu wprowadzania bloków przedmiotów do programu „Plan lekcji Optivum” [15]	11
2.6	Zrzut ekranu wprowadzania terminów zajęć dla poszczególnych klas do programu „Plan lekcji Optivum” [15]	12
2.7	Zrzut ekranu wprowadzania dostępności nauczycieli do programu „Plan lekcji Optivum” [15]	12
2.8	Zrzut ekranu wprowadzania „trudnych” lekcji do programu „Plan lekcji Optivum” [15]	13
3.1	Zrzut ekranu z arkusza kalkulacyjnego przedstawiający przypisanie godzinowe nauczycieli dla każdej klasy.	17
3.2	Dekompozycja problemu	24
3.3	Struktura algorytmu z doborem technik	25
3.4	Wizualizacja rozwiązania klasycznego problemu JSSP z wykorzystaniem zmiennych przedziałowych [10]	29
3.5	Wykres funkcji użytej do ewaluacji rozkładu godzin nauczycieli i klas na przestrzeni tygodnia.	36
3.6	Wykres wartości funkcji celu dla pięciu uruchomień algorytmu genetycznego	40
4.1	Diagram wymagań	47
4.2	Diagram przypadków użycia	48
4.3	Diagram sekwencji generowania planu lekcji	55
4.4	Diagram sekwencji dla przypadku użycia oglądania planu lekcji	56
4.5	Diagram sekwencji edycji bądź wprowadzania wymagań głównych	56
4.6	Architektura aplikacji	57
4.7	Diagram tabel podstawowych w bazie danych	59
4.8	Diagram tabel podstawowych i wymagań w bazie danych	60
4.9	Diagram pełnego projektu bazy danych	63
4.10	Fragment kodu służący do definicji tabeli sal	68

4.11 Automatycznie wygenerowana lista punktów końcowych stworzonych przy użyciu ModelViewSet	69
4.12 Fragment kodu służący do implementacji punktów końcowych dla zbioru sal . .	69
4.13 Nawigacja interfejsu webowego	70
4.14 Strona główna interfejsu webowego	70
4.15 Podgląd planu z perspektywy nauczyciela	71
4.16 Podgląd planu z perspektywy klasy	71
4.17 Widok generowania planu lekcji	72
4.18 Widok dodawania przedmiotów	73
4.19 Widok dodawania nauczycieli	73
4.20 Widok dodawania sal	74
4.21 Widok dodawania klas	74
4.22 Widok dodawania wymagań głównych	75
4.23 Widok wprowadzania dostępności nauczycieli	76
4.24 Widok dodawania bloków przedmiotów	76
5.1 Zrzuty ekranów wykonanych kroków dla scenariusza generowania planu lekcji .	80
5.2 Zrzuty ekranów wykonanych kroków dla scenariusza przeglądania planów lekcji	81
5.3 Zrzuty ekranów wykonanych kroków dla scenariusza importowania i edytowania wymagań głównych	82
5.4 Zrzuty ekranów wykonanych kroków dla scenariusza edytowania dostępności nauczycieli	83
5.5 Histogram obciążeń dydaktycznych nauczycieli tygodniowo	84
5.6 Histogram obciążień dydaktycznych uczniów tygodniowo	85
5.7 Histogram obciążień dydaktycznych nauczycieli	86
5.8 Histogram obciążień dydaktycznych uczniów	87
5.9 Wygenerowany plan lekcji dla klasy IIAC	89
5.10 Plan lekcji dla klasy IIAC stworzony przez szkołę	90
5.11 Zestawienie śródrodznych planów lekcji dla kolejno klas IIIA, IIIC, IID, IIIF i IIIG	92

Spis tabel

4.1 Przypadek użycia UC01	46
4.2 Przypadek użycia UC02	49
4.3 Przypadek użycia UC03	49
4.4 Przypadek użycia UC04	50
4.5 Przypadek użycia UC05	50
4.6 Przypadek użycia UC06	51
4.7 Przypadek użycia UC07	51
4.8 Przypadek użycia UC08	52
4.9 Przypadek użycia UC09	52
4.10 Przypadek użycia UC10	53
4.11 Przypadek użycia UC11	53
4.12 Przypadek użycia UC12	54
4.13 Tabele podstawowe zasobów szkolnych	58
4.14 Tabele definiujące wymagania i ograniczenia	59
4.15 Tabele zbiorów zasobów	60
4.16 Tabela requirement_set	61
4.17 Tabele przechowujące wygenerowane plany lekcji	61
4.18 Struktura najważniejszych tabel w bazie danych	64
5.1 Generowanie planu lekcji z określonymi parametrami	78
5.2 Scenariusz przeglądu planów lekcji	78
5.3 Scenariusz importowania i edytowania wymagań głównych	79
5.4 Scenariusz edytowania dostępności nauczycieli	79
5.5 Statystyki dla generowanych planów	88