

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ННК “Інститут прикладного системного аналізу”  
(повна назва інституту/факультету)

Кафедра Системного проектування  
(повна назва кафедри)

«До захисту допущено»

Завідувач кафедри

\_\_\_\_\_ А.І.Петренко  
(підпис) (ініціали, прізвище)

“ \_\_\_\_ ” \_\_\_\_\_ 2016 р.

**Дипломна робота**

першого (бакалаврського) \_\_\_\_\_ рівня вищої освіти  
(першого (бакалаврського), другого (магістерського))

зі спеціальності 7.05010102, 8.05010102 Інформаційні технології проектування  
7.05010103, 8.05010103 Системне проектування  
(код та назва спеціальності)

на тему: Дослідження алгоритмів оптимізації індивідуальних календарних  
планів навчання

Виконав: студент 4 курсу, групи ДА-21  
(шифр групи)

\_\_\_\_\_ Севідов Павло Миколайович \_\_\_\_\_  
(прізвище, ім'я, по батькові) (підпис)

Керівник \_\_\_\_\_ к.т.н., доц. Кисельов Г.Д. \_\_\_\_\_  
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант економічний проф. док. ек. н. Семенченко Н. В. \_\_\_\_\_  
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент \_\_\_\_\_ д.т.н., проф., Бідюк П.І \_\_\_\_\_  
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Нормоконтроль \_\_\_\_\_ ст. викладач Бритов О.А. \_\_\_\_\_  
(підпис)

Засвідчую, що у цій дипломній роботі  
немає запозичень з праць інших авторів  
без відповідних посилань.

Студент \_\_\_\_\_  
(підпис)

Київ – 2016 року

**Національний технічний університет України  
«Київський політехнічний інститут»**

Факультет (інститут) ННК “Інститут прикладного системного аналізу”  
(повна назва)

Кафедра Системного проектування  
(повна назва)

Рівень вищої освіти Перший(Бакалаврський)  
(перший (бакалаврський), другий (магістерський) або спеціаліста)

Спеціальність 7.05010102, 8.05010102 Інформаційні технології проектування  
7.05010103, 8.05010103 Системне проектування  
(код і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри

А.І.Петренко  
(підпис) (ініціали, прізвище)

«   »                      2016 р.

**ЗАВДАННЯ**

**на дипломний проект (роботу) студенту**

Севідову Павлу Миколайовичу  
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) ) Дослідження алгоритмів оптимізації індивідуальних план-графіків навчання

керівник проекту (роботи) ) Кисельов Г.Д., к.т.н., доц.,  
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від 12 травня 2016 р. № 50-ст

2. Строк подання студентом проекту (роботи) \_\_\_\_\_

3. Вихідні дані до проекту (роботи)

Дослідити алгоритми оптимізації індивідуальних план-графіків студента на основі учбового плану, враховуючи власні зайняття.

Для реалізації використовувати технологію Spring framework.

Вхідними даними є учбовий план і індивідуальні заняття.

Можливість вибору алгоритму оптимізації.

Передбачити можливість додання нових завдань протягом семестру.

Результати обчислень мають включати наступні пункти: виведення оптимізованого план-графіку студента, графічного зображення графіку значення фітнес функції від ітерації для всіх алгоритмів, графічного зображення графіку фітнес функції від ймовірності мутації, графічного зображення графіку фітнес функції від розміру популяції.

4. Зміст розрахунково-пояснювальної записки (перелік завдань, які потрібно розробити)

1. Постановка задачі та аналіз предметної області
2. Розробка вимог до характеристик об'єкту проектування. Вибір алгоритму
3. Вибір інструментальної бази розробки
4. Результати роботи розроблюваної програми
5. Функціонально-вартісний аналіз програмного продукту
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслеників, плакатів тощо)
  1. Блок-схема алгоритму оптимізації план-графіку - плакат
  2. UML-діаграма класів розробленого додатку – плакат.
  3. Результати роботи програми - плакат

6. Консультанти розділів проекту (роботи)\*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний розділ	проф. док. ек. н. Семенченко Н. В.		

7. Дата видачі завдання 01.02.2016

#### Календарний план

№ з/п	Назва етапів виконання дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Отримання завдання	01.02.2016	
2	Збір інформації	15.02.2016	
3	Вивчення алгоритмів оптимізації та вибір варіанту для розробки	28.02.2016	
4	Розробка алгоритму та структури системи	10.03.2016	
5	Розробка програмної моделі	15.03.2016	
6	Тестування додатку та отримання результатів	25.03.2016	
7	Оформлення дипломної роботи	31.05.2016	
8	Отримання допуску до захисту та подача роботи в ДЕК		

Студент

\_\_\_\_\_

(підпис)

П.М. Севідов

(ініціали, прізвище)

Керівник проекту (роботи)

\_\_\_\_\_

(підпис)

Г.Д. Кисельов

(ініціали, прізвище)

\* Консультантом не може бути зазначено керівника дипломного проекту (роботи).

## АНОТАЦІЯ

бакалаврської дипломної роботи Севідова Павла Миколайовича  
на тему «Дослідження алгоритмів оптимізації індивідуальних календарних  
планів навчального процесу»

У даній роботі ставиться завдання розглянути алгоритми оптимізації індивідуальних планів-графіків навчання, для вирішення проблеми управління часу студента. Було досліджено оптимальність використання генетичних алгоритмів для вирішення задачі багатокритеріальної оптимізації календарного плану навчального процесу.

Різноманітність предметів і робіт призводить до неоптимального використання власного часу. Наслідком даної проблеми є так званий «синдром студента». Це призводить до втрати часу, яке було виділено при оцінці трудомісткості і ризиків проекту, і збільшення рівня стресу. Рішення даної проблеми є актуальним, поліпшить успішність, зменшить навантаження на студентів і викладачів. Головна мета - забезпечити своєчасну здачу запланованих навчальним планом робіт, ефективно розподіляючи найважливіший ресурс кожного студента - час. Всі наявні рішення не враховують індивідуальний розклад студента і рівень його щоденної завантаженості власними справами при формуванні графіка виконання навчального плану.

Результат роботи - реалізація алгоритмів в системі створення індивідуального календарного плану, порівняльний аналіз алгоритмів, порівняльні дані оптимізації та демонстрація результатів оптимізації календарного плану виконання навчального процесу на 1 семестр.

Загальний обсяг роботи 100 с., 42 рис., 10 табл., 1 додаток на 3 стр., 26 посилань.

**Ключові слова:** алгоритми оптимізації, генетичний алгоритм, план-графік, фітнес функція, Spring framework.

## АННОТАЦИЯ

бакалаврской дипломной работы Севидова Павла Николаевича  
на тему: «Исследование алгоритмов оптимизации индивидуальных  
календарных планов учебного процесса»

В данной работе ставится задача рассмотреть алгоритмы оптимизации индивидуальных план-графиков обучения, для решения проблемы управлением времени студента. Было исследовано оптимальность использования генетических алгоритмов для решения задачи многокритериальной оптимизации календарного плана учебного процесса.

Разнообразии предметов и работ приводит к неоптимального использования собственного времени. Следствием данной проблемы является так называемый «синдром студента». Это приводит к потере времени, которое было выделено при оценке трудоемкости и рисков проекта, и увеличению уровня стресса. Решение данной проблемы является актуальным, улучшит успеваемость, уменьшит нагрузку на студентов и преподавателей. Главная цель - обеспечить своевременную сдачу запланированных учебным планом работ, эффективно распределяя важнейший ресурс каждого студента - время. Все имеющиеся решения не учитывают индивидуальное расписание студента и уровень его ежедневной загруженности собственными делами при формировании графика выполнения учебного плана.

Результат работы – реализация алгоритмов в системе создания индивидуального календарного плана, сравнительный анализ алгоритмов, сравнительные данные оптимизации и демонстрация результатов оптимизации календарного плана выполнения учебного процесса на 1 семестр.

Общий объем работы 100 с., 42 рис., 10 табл., 1 приложение на 3 стр., 26 источников.

**Ключевые слова:** алгоритмы оптимизации, генетический алгоритм, план-график, фитнес функция, Springframework.

## ABSTRACT

on Pavel Sevidov bachelor's degree

thesis: "Algorithms for individual training schedule optimization"

In this work we consider the task of optimization algorithms of individual training schedules to solve the problem of student time management. It was investigated the optimal use of genetic algorithms for solving the problem of multi-criteria optimization schedule of the educational process.

The variety of subjects and activities leads to suboptimal use of his time. The consequence of this problem is the so-called "student syndrome". This leads to a loss of time that has been allocated in assessing risks and design complexity, and increased stress levels. The solution to this problem is urgent, improve performance, reduce the burden on students and teachers. The main purpose - to ensure the timely delivery of the planned curriculum work, effectively distributing the most important resource of every student - time. All the existing solutions do not take into account the student's individual schedule and the level of its daily congestion own affairs in forming schedule of the curriculum.

Result of work - implementation of algorithms in the system to create an individual schedule, the comparative analysis of algorithms, optimization and comparative data to demonstrate results in the 1st semester optimizing schedule performance of the educational process.

Bachelor's work size 100 p., 42 pic., 10 tables, 1 appendix for 3 p., 26 sources.

**Keywords:** optimization algorithms, genetic algorithm, schedule, fitness function, Springframework.

## ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ .....	10
ВСТУП .....	11
1 ПОСТАНОВКА ЗАДАЧІ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.....	13
1.1 Основи теорії розкладів .....	13
1.2 Завдання знаходження допустимого розкладу .....	15
1.3 Класифікація завдань TP .....	18
1.4 Висновок до розділу 1 .....	20
2 РОЗРОБКА ВИМОГ ДО ХАРАКТЕРИСТИК ОБ’ЄКТУ ПРОЕКТУВАННЯ. ВИБІР АЛГОРИТМУ .....	22
2.1 Розробка математичної моделі .....	22
2.2 Критерії оптимізації .....	23
2.2.1 Hard Stop критерії .....	23
2.2.2 Soft Stop критерії.....	23
2.3 Об’єктна модель предметної області .....	24
2.4 Функціональні аспекти системи. Діаграма прецедентів. ....	25
2.5 Структурні особливості взаємодії об’єктів. Діаграма послідовностей.....	26
2.6 Опис потоків даних. Data Flow Diagram .....	27
2.7 Опис алгоритму планування .....	28
2.7.1 Опис генетичного алгоритму .....	28
2.7.2 Етапи генетичного алгоритму .....	30
2.7.3 Створення початкової популяції .....	30
2.7.4 Відбір.....	31
2.7.5 Розмноження .....	31
2.7.6 Мутації.....	32
2.8 Види кросинговеру .....	32
2.9 Види мутації .....	42
2.10 Висновки до розділу 2 .....	45
3 ВИБІР ІНСТРУМЕНТАЛЬНОЇ БАЗИ РОЗРОБКИ.....	47

3.1.	Вибір бази даних. Порівняння MySQL і PostgreSQL.....	47
3.1.1.	Коли використовувати MySQL.....	49
3.1.2.	Коли використовувати PostgreSQL.....	49
3.2.	Веб фреймворк Spring Framework.....	50
3.3.	Шаблон проектування MVC .....	52
3.4.	Опис Spring web MVC фреймворку .....	53
3.4.1.	Обробник запитів DispatcherServlet.....	54
3.5.	Авторизація користувача .....	55
3.6.	Об'єктно-реляційна проекція. Hibernate.....	58
3.6.1.	Компоненти Hibernate.....	60
3.7.	Тестування .....	61
3.7.1.	Приклад тестування.....	62
3.8.	Синхронізація з Google сервісами .....	63
3.8.1.	GoogleApiClient .....	63
3.8.2.	GooglecalendarApi .....	65
3.9.	Інтерфейс користувача.....	68
3.10.	Висновки до розділу 3 .....	70
4.	РЕЗУЛЬТАТИ РОБОТИ РОЗРОБЛЮВАНОЇ ПРОГРАМИ.....	72
4.1.	Порівняння з результатами роботи алгоритму імітації відпалу.....	75
4.2.	Висновки до розділу 3 .....	75
5	ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ.....	77
5.1.	Постановка задачі.....	78
5.1.1.	Обґрунтування функцій програмного продукту.....	78
5.1.2.	Варіанти реалізації основних функцій .....	79
5.2.	Обґрунтування системи параметрів ПП .....	81
5.2.1.	Опис параметрів .....	81
5.2.2.	Кількісна оцінка параметрів.....	81
5.2.3.	Аналіз експертного оцінювання параметрів .....	83
5.3.	Аналіз рівня якості варіантів реалізації функцій .....	86



5.4. Економічний аналіз варіантів розробки ПП .....	87
5.5. Вибір кращого варіанта ПП техніко-економічного рівня .....	91
5.6. Висновки до розділу 5 .....	91
ВИСНОВКИ .....	93
ПЕРЕЛІК ПОСИЛАНЬ.....	95
ДОДАТОК А .....	98
Реалізація генетичного алгоритму .....	98

## ПЕРЕЛІК СКОРОЧЕНЬ

ПЗ – програмне забезпечення.

API – прикладний програмний інтерфейс (англ. application programming interface).

REST – Передача стану подання (англ. Representation State Transfer).

XML – Розширювана мова розмітки (англ. Extensible Markup Language).

UI – засіб зручної взаємодії користувача з інформаційною системою (англ. User Interface).

SQL (мова структурованих запитів) — декларативна мова програмування для взаємодії користувача з базами даних, що застосовується для формування запитів, оновлення і керування реляційними БД, створення схеми бази даних і її модифікації, системи контролю за доступом до бази даних (англ. Structured query language).

MVC — архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення (англ. Model-view-controller).

БД – база даних (англ. database).

IoC — це принцип побудови програми, при якому її частини отримують потік керування (викликаються) із загальної спільно використовуваної бібліотеки (англ. Inversion of Control, IoC).

## ВСТУП

Люди протягом усього свого життя стикаються з задачею оптимального планування свого часу. Кожен з щодня займається плануванням власного часу. Майже всі люди керуються при цьому “схожим алгоритмами”, намагаючись виконати всі важливі справи та вчасно, виділивши при цьому максимальний час на дозвілля та відпочинок.

Часто проблема формування календарного плану виникає у студентів. Різноманітність навчальних предметів і робіт призводить до неоптимального використання власного часу. Вирішення даної проблеми є актуальним, адже покращує успішність, зменшує навантаження на студентів і викладачів.

Складнощі при формуванні розкладів з'являються тоді, коли робіт стає багато, потрібно врахувати безліч додаткових умов і / або скласти розклад не для однієї людини, а для цілого колективу, коли існує сотні робіт і виконавців.

Аналізуючи подібні рішення, можливо сказати, що майже не існує систем, що враховують індивідуальний розклад користувача. Багато людей використовують Google календар, для ведення власного розкладу. Даний сервіс є найпопулярнішим в світі, за даними на офіційному сайті Google Calendar. Тому було обрано за основу індивідуального розкладу дані з публічного календаря Google.

У процесі вирішення таких завдань, були вироблені спільні рекомендації, принципи і методики складання розкладів. Надалі подібні завдання стали досліджуватися в рамках спеціального розділу науки - теорії розкладів.

**Теорія розкладів** - це розділ дослідження операцій, в якому будуються і аналізуються математичні моделі календарного планування (тобто упорядкування в часі) різних цілеспрямованих дій з урахуванням цільової функції і різних обмежень.

Змістовно багато завдань ТР є оптимізаційними, тобто складаються у виборі (знаходженні) серед безлічі допустимих розкладів (розкладів, що допускаються умовами завдання) тих рішень, на яких досягається "оптимальне"

значення цільової функції. Зазвичай під "оптимальністю" розуміється мінімальне або максимальне значення деякої цільової функції. Допустимість розкладу розуміється в сенсі його здійсненності, а оптимальність - в сенсі його доцільності.

**Приклад.** Необхідно побудувати будинок якомога швидше, при цьому послідовність робіт повинна бути дотримана. Для даної задачі допустимий розклад то, при якому буде дотримана послідовність робіт, а оптимальний розклад - це допустимий розклад, при якому будинок буде побудований в мінімальні терміни.

**Календарний план** – проектно-технологічний документ, який визначає послідовність, інтенсивність, тривалість робіт, також їх взаємозв'язок з іншими видами робіт, необхідність в матеріальних, технічних, трудових або фінансових ресурсах.

Даний тип завдань є NP складною і не існує точної математичної моделі для знаходження оптимального рішення. Тому доцільно використання стохастичних і евристичних методів. Дані методи дозволяють знайти «гарне» рішення за допустимий час. Головним недоліком, що можливо знайти – це велика ймовірність не виходження з локального мінімуму, що унеможливорює подальший пошук оптимального рішення.

**Основна задача планування** – полягає в створенні певного порядку дій та оптимальному розподілі ресурсів (вільний час студента), що необхідні для досягнення поставленої мети (успішне та своєчасне виконання студентом усіх робіт, що передбачені навчальним процесом з врахуванням власного графіку). Сформований графік робіт повинен задовольняти всім наявним hard stop критеріям, а також якомога менше порушувати умови soft stop критеріїв.

# 1 ПОСТАНОВКА ЗАДАЧІ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Завдання аналіз алгоритмів для планування власного розкладу виникло в результаті проблеми спланувати час виконання великої кількості задач при в рахуванні різних обмежуючих критеріїв, що звісно не посилі виконати звичайній людині без допомоги комп'ютера.

На сьогоднішній день існує багато рішень для планування розкладів, вони мають багато переваг але не позбавлені недоліків. Більшість з них орієнтовані на створення розкладів для шкіл, університетів та інших навчальних закладів. Серед них є аналоги, що відповідають майже всім вимогам але вартість таких сервісів занадто велика, а безкоштовні - з поганим функціоналом та інтерфейсом користувача.

Перший розділ містить в собі постановку задачі. В цьому розділі досліджується проблема теорії розкладів, розглядаються існуючі рішення та наводяться їх недоліки та переваги, у кінці робиться висновок по особливостям предметної області та формується вимоги до програмного продукту.

## 1.1 Основи теорії розкладів

Для побудови математичної моделі для нашого додатку проаналізуємо дисципліну яка займається вивченням розкладів. Які є особливості цієї предметної області, типи розв'язуваних задач, проблеми планування та інше. Розбір даного предмету дасть нам необхідну інформацію для винесення обґрунтованих рішення щодо побудови математичної моделі планувальника.

Часто ми плануємо наші дії в порядку зростання крайніх термінів виконання робіт (справ). Наприклад, студенти під час екзаменаційної сесії вчать предмет з найменшим директивним терміном (найближчий за датою здачі іспит). Якщо перший іспит необхідно здати 12-го січня, другий 15-го, а третій 19-го, при цьому на кожен іспит необхідно 3 дні, то більшість студентів складуть такий

розклад: "з 9-го по 11-е я готуюся до першого іспиту, з 12-го по 14-е до другого і з 16-го по 18-е до третього".

Складнощі при складанні розкладів з'являються тоді, коли робіт стає багато, потрібно врахувати безліч додаткових умов і / або скласти розклад не для однієї людини, а для цілого колективу. Уявіть собі сотні робіт і десятки виконавців, для яких необхідно скласти розклад.

У процесі вирішення таких завдань, були вироблені спільні рекомендації, принципи і методики складання розкладів. Надалі подібні завдання стали досліджуватися в рамках спеціального розділу науки - теорії розкладів [1].

Ця наука виникла не на порожньому місці, а виникла на стику інших областей наукового знання. Перш ніж дати визначення, що таке Теорія розкладів, опишемо одну важливу область математики.

**Дослідження операцій** (ДО) - науковий метод вироблення кількісно обґрунтованих рекомендацій щодо прийняття рішень [4]. Важливість кількісного фактора в ДО і цілеспрямованість сформульованих рекомендацій дозволяють визначити ДО як теорію прийняття оптимальних рішень. ДО сприяє перетворенню мистецтва прийняття рішень в математичну дисципліну. Термін "ДО" виник в результаті буквального перекладу виразу "operation research", введеного в кінці 30-х років 20-го століття як умовне найменування одного з підрозділів британських ВПС, що займається питаннями використання радіолокаційних установок в загальній системі оборони. Спочатку ДО було пов'язано з вирішенням завдань військового змісту, але вже з кінця 40-х років минулого століття воно використовується для вирішення технічних, техніко-економічних завдань, а також завдань управління на різних рівнях.

Тепер ми можемо дати визначення теорії розкладів.

**Теорія розкладів** - це розділ дослідження операцій, в якому будуються і аналізуються математичні моделі календарного планування (тобто упорядкування в часі) різних цілеспрямованих дій з урахуванням цільової функції і різних обмежень.

Змістовно багато завдань TP є оптимізаційними, тобто складаються у виборі (знаходженні) серед безлічі допустимих розкладів (розкладів, що допускаються умовами задачі) тих рішень, на яких досягається "оптимальне" значення цільової функції. Зазвичай під "оптимальністю" розуміється мінімальне або максимальне значення деякої цільової функції. Допустимість розкладу розуміється в сенсі його здійсненності, а оптимальність - в сенсі його доцільності.

**Приклад.** Необхідно побудувати будинок якомога швидше, при цьому послідовність робіт повинна бути дотримана. Для даної задачі допустимий розклад то, при якому буде дотримана послідовність робіт, а оптимальний розклад - це допустимий розклад, при якому будинок буде побудований в мінімальні терміни.

Інший тип завдань полягає в пошуку допустимого розкладу, який задовольняє всім умовам. Наведемо приклади обох типів завдань.

## 1.2 Завдання знаходження допустимого розкладу

На одному процесорі потрібно виконати безліч  $N = \{1, 2, \dots, N\}$  завдань. Для кожного завдання  $j \in N$  визначені тривалість виконання  $p_j > 0$ , час надходження завдання на процесор  $r_j \geq 0$  і крайній директивний термін  $D_j > 0$ , до якого завдання повинне бути виконане. Процесор готовий до виконання завдань з моменту часу 0 і може виконувати одночасно лише одне завдання. Переривання при виконанні будь-якого завдання заборонені. Необхідно побудувати допустимий розклад виконання завдань, при якому всі умови задачі дотримані. Тобто необхідно для кожного вимоги  $j \in N$  визначити момент початку виконання  $S_j$  такий, що  $S_j \geq r_j$  і момент закінчення виконання  $C_j = S_j + p_j \leq D_j$ . Причому якщо  $S_j < S_i$ , то  $S_j + p_j \leq S_i$ , де  $S_i$  – момент початку виконання іншого завдання  $i \in N$ ,  $i \neq j$  [9], [10].

У задачах ТР час задається в умовних одиницях. Параметри  $p_j, r_j, D_j, C_j, S_j$  можуть вимірюватися в хвилинах, годинах, днях і т.п. З точки зору обчислювального процесу два приклади -  $p_1 = 2$  хвилини,  $p_2 = 3$  хвилини,  $r_1 = 0$  - нульова хвилина,  $r_2 = 1$  - перша хвилина і, приклад, де  $p_1 = 2$  години,  $p_2 = 3$  години,  $r_1 = 0$  - нульовий час,  $r_2 = 1$  - першу годину, - є ідентичними, тому одиницю виміру часу при визначенні завдань опускають.

Модифікуємо попередню задачу наступним чином. Нехай крайні терміни  $D_j$  не задані (тобто  $D_j = +\infty, j = 1, 2, \dots, N$ ). Всі інші умови залишаються тими ж. Позначимо  $C_j$  - момент закінчення виконання завдання  $j$ , тобто  $C_j = S_j + p_j$ , де  $S_j$  - момент початку виконання завдання  $j$ . Необхідно побудувати допустимий розклад, при якому значення функції  $\sum_{j=1}^n C_j$  буде мінімальним [11].

Тепер з оглядкою на дві наведені завдання можна дати наступні два визначення.

**Визначення 1** Завдання, в якій всі вхідні дані повністю визначені, називається індивідуальним завданням.

**Визначення 2** Масова завдання - безліч індивідуальних завдань.

Надалі ми будемо називати масову задачу просто завданням (наприклад, "Завдання комівояжера"), а індивідуальну задачу - прикладом.

Таблица 1.1: Пример. Исходные данные, допустимое и оптимальное расписания

$j$	1	2	3	4
$p_j$	6	2	2	3
$r_j$	0	1	2	11
$D_j$	7	10	9	15
Допустимое расписание. Время начала выполнения $S_j$	0	8	6	11
Допустимое расписание. Время окончания выполнения $C_j$	6	10	8	14
Оптимальное расписание. Время начала выполнения $S_j$	5	1	3	11
Оптимальное расписание. Время окончания выполнения $C_j$	11	3	5	14

Рис. 1.1. Вихідні данні

На рисунку 1.1 представлені допустиме і оптимальне розкладу для обох завдань на одному і тому ж прикладі. Схематично отримані розкладу зображені



на рисунку 1.2. На цьому малюнку розкладу представ лени у вигляді набору прямокутників, розташованих уздовж віртуальної тимчасової осі  $t$ . Кожен прямокутник відповідає деякому завданню, а довжина прямокутників відповідає тривалості виконання завдань, а їх розташування на осі вказує час початку і закінчення виконання кожного завдання.

Для першого завдання існує єдиний допустимий розклад, при якому порядок виконання завдань визначено наступним чином: (1, 3, 2, 4). Спочатку виконується перше завдання, потім третє, друге і четверте. Очевидно, що для даного прикладу будь-який інший порядок виконання завдань неприпустимий. Наприклад, при порядку обслуговування (2, 1, 3, 4) буде порушено умова  $C_1 \leq D_1$ , де  $D_1 = 7$  і  $C_1 = 9$ .

Для другого завдання схематично представлено оптимальний розклад. На цьому ж малюнку для другого завдання зображено допустимий (в термінах другого завдання), але неоптимальний розклад, заданий порядком (1, 2, 3, 4). Легко переконатися, що при оптимальному розкладі, заданому порядком обслуговування (2, 3, 1, 4) маємо  $\sum_{j=1}^4 C_j = 11 + 3 + 5 + 14 = 33$ , а при розкладі, заданому як (1, 2, 3, 4), маємо  $\sum_{j=1}^4 C_j = 6 + 8 + 10 + 14 = 38$ .

Завдання ТР (як завдання розділу Дослідження Операцій) мають ряд рис, які обумовлюють методику їх складання і рішення. По-перше, навіть для простих параметричних задач не вдається уявити.

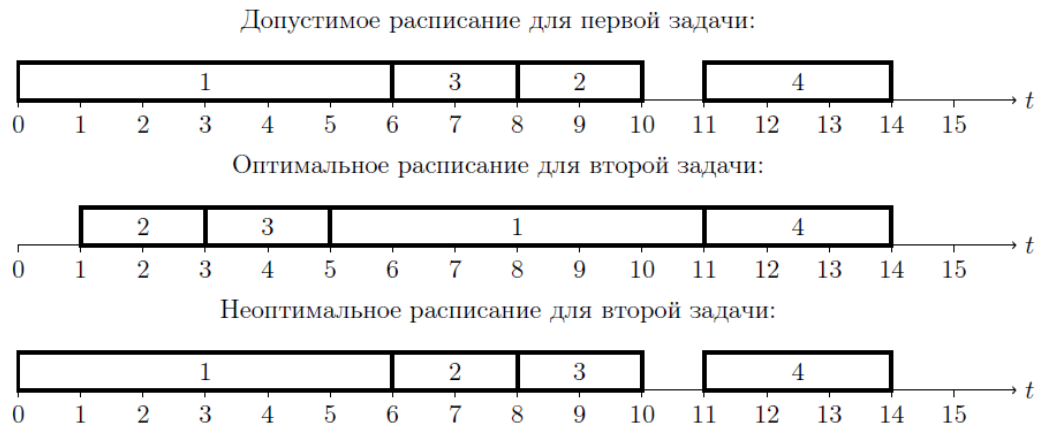


Рис. 1.2. Графічне представлення розкладів.

рішення у вигляді аналітичного виразу від відповідних параметрів (у вигляді формули). Тому завдання ТР, в переважній більшості, не піддаються аналітичному рішенню і повинні вирішуватися чисельно. По-друге, більшість завдань ТР містить в своїх формулюваннях велика кількість числового матеріалу, що не зводиться до аналітичних виразів. Тому чисельне рішення цих задач, за небагатьма винятками, можливо лише за допомогою комп'ютера [2].

Для вирішення завдань ТР необхідно розробити алгоритм рішення. Тобто послідовність дій, виконуваних комп'ютером, за допомогою яких можна побудувати шукане розклад (допустиме або оптимальне).

Зустрічаються на практиці завдання складання розкладів містять тисячі, а часом і мільйони завдань. Тому основна мета при дослідженні моделей (завдань) ТР - це побудова ефективних, тобто швидких, алгоритмів рішення. Рішення прикладу має бути отримано за "розумний" час.

### 1.3 Класифікація завдань ТР

Наведемо деякі способи класифікації задач ТР [3]:

- За типом даного завдання:
  - Завдання упорядкування. У цих завданнях вже задано розподіл робіт за виконавцями, а також визначені всі параметри робіт

- (тривалість виконання, час надходження та т.д.). Необхідно скласти розклад (або порядок) виконання робіт кожним виконавцем;
- Завдання узгодження. Основна увага в цих завданнях приділяється вибору тривалості виконання робіт, часу надходження і іншим параметрам;
  - Завдання розподілу на прикладі пошуку оптимального розподілу робіт за виконавцями.
- За типом цільової функції:
    - Завдання з сумарними критеріями оптимізації. У попередньому розділі ми навели приклад такого завдання, в якій необхідно було мінімізувати сумарне значення моментів закінчення обслуговування робіт  $\sum_{j=1}^n C_j$ ;
    - Завдання з  $\min\max$  (мінімаксними) критеріями оптимізації. Відмінність цих завдань від завдань з сумарними критеріями полягає в тому, що потрібно мінімізувати не сума деяких значень, а лише максимальне з них. Наприклад, якщо в згаданій задачі необхідно мінімізувати максимальне значення  $C_{\max}$ , де  $C_{\max} = \max(j \in N) C_j$ , то ми отримаємо одну з тривіальних завдань цього класу;
    - Багатокритеріальні задачі оптимізації. Якщо в досліджуваних завданнях необхідно побудувати оптимальне рішення з точки зору декількох цільових установок (функцій), то такі завдання називаються Багатокритеріальний. Наприклад, якщо в згаданій задачі необхідно не тільки мінімізувати значення  $\sum_{j=1}^n C_j$ , але мінімізувати і час простою процесора (Приладу), то це багатокритеріальна задача;
    - Завдання на побудову допустимого розкладу. В попередньому розділі було дано приклад такого завдання. Необхідно відзначити, що даний клас задач можна звести до оптимізаційних задач,

ввівши спеціальну функцію штрафу, яку потрібно мінімізувати.

Проте, прийнято виділяти такі задачі в окремий клас.

- За способом завдання вхідної інформації:
  - Детерміновані задачі (off-line). Для таких задач характерно, що всі вхідні дані завдання точно відомі, тобто дані значення всіх параметрів до початку її вирішення;
  - Динамічні задачі (on-line). Для даних завдань розкладу будуються в режимі реального часу, тобто перед початком виконання завдання ми не знаємо значення всіх параметрів. Розклад будується частинами в міру надходження нової інформації. При цьому в будь-який момент може бути знадобитися відповідь про якість побудованого "часткового" розкладу.
- Згідно розділу TP. В рамках TP прийнято виділяти наступні розділи:
  - Планування мереж або побудова розкладу для проекту, Project scheduling (PS);
  - Календарне планування або побудова розкладу для приборів, Machine scheduling (MS);
  - Складання тимчасових таблиць (Time Tabling);
  - Доставка товарів в магазини (Shop-Floor Scheduling);
  - Складання розкладів руху транспортних засобів (Transport Scheduling), Циклічні розкладу для транспортних засобів (Vehicle Routing);
  - Складання розкладів спортивних заходів (Sports scheduling).

## **1.4 Висновок до розділу 1**

В результаті розбору теорії розкладів, з'ясовано це є розділ дослідження операцій, який займається аналізом математичної моделі календарного планування й метою є знаходження оптимального та допустимого розкладу з існуючих варіантів. Для цього використовується цільова функція яка враховує вибрані критерії оптимізації.

Оскільки задача планування розкладу в нашому випадку є багатокритеріальною, потрібно враховувати навантаження дня, сумарне значення моментів закінчення завдань, а також пріоритети задач. Для цього підходять алгоритми оптимізації при великій кількості умов та критеріїв, наприклад як генетичний алгоритм та імітації відпалу. Також сервіс повинен підтримувати інтеграцію з електронними календарями, як Google Calendar, для зручності отримання вже існуючого розкладу людини для подальшого планування оптимального рішення.

Отже, потрібно створити власну систему. Створення веб сервісу в повному обсязі задовільнить всі критерії, а також має ряд переваг для кінцевого користувача: зручність доступу, не потребує попереднього встановлення, складні математичні обчислювання виконуються на серверній частині.

## 2 РОЗРОБКА ВИМОГ ДО ХАРАКТЕРИСТИК ОБ'ЄКТУ ПРОЕКТУВАННЯ. ВИБІР АЛГОРИТМУ

### 2.1 Розробка математичної моделі

$N = \{1, \dots, n\}$  – множина робіт на один семестр. Завдання виконуються послідовно одним студентом (з перериваннями або без, *MachineSchedulingProblem*).

Кожна робота характеризується наступними параметрами:

$p_j > 0$  – тривалість виконання  $j$ -ої роботи.

$D_j > 0$  – крайній термін здачі  $j$ -ої роботи.

$W_j > 0$  – вага  $j$ -ої роботи (залежить від типу предмета: залік, диференціальний залік, екзамен та від типу роботи: лабораторна робота, реферат, розрахункова робота, курсова робота, дипломна робота і т.д.)

$r_j \geq 0$  – час отримання роботи (не всі роботи видаються в 0-ий момент часу, видача деякий робіт відбувається протягом семестру)

$S_j > 0$  – момент початку (start) виконання  $j$ -ої роботи (величина, яку необхідно визначити для кожної роботи).

$S_j > r_j$  – роботу можна починати виконувати тільки після її отримання.

$Z_{jk}$ , де  $k = 1, \dots, m$  – множина моментів часу до  $k$ -ої можливості здачі  $j$ -ої роботи.

Момент завершення виконання роботи (hardstop критерій):

$$C_j = S_j + p_j + z_{j1} \leq D_j \quad (2.1)$$

Якщо  $S_j < S_i$ , то  $S_j + p_j \leq S_i$ , где  $i \neq j$  (роботи не можуть виконуватися паралельно).

**Задача полягає в оптимізації суми штрафних функцій (сума дат завершення кожної роботи).**

$$\sum_j^n C_j \rightarrow \min \quad (2.2)$$

Ще одна величина, яка підлягає оптимізації – це показник рівномірності навантажень на кожен робочий день ( $N_i$ ).

## 2.2 Критерії оптимізації

Дана багатокритеріальна задача оптимізації має відповідати певним вимогам, порушення деяких з них унеможлиблює отримання результату. В даному контексті можна виділити два основних види критеріїв: HardStop і SoftStop.

### 2.2.1 Hard Stop критерії

HardStop – критерії, які заборонено порушувати для даної задачі.

Перелік Hard Stop критеріїв:

- Кожна робота повинна бути закінчена і здана до крайнього строку (deadline).
- Виконану роботу можна здати лише в той день, коли є пара з викладачем, який приймає роботу.
- Почати виконання роботи можливо за умови, якщо немає інших справ в цей час.

### 2.2.2 Soft Stop критерії

Softstop – критерії, які не бажано порушувати, порушення призводить до погіршення результату.

Перелік Soft stop критеріїв:

- Обов'язкові задачі з високим пріоритетом повинні виконуватися якомога раніше.
- Рівномірність завантаженості кожного робочого дня (показник завантаженості не повинен перевищувати допустиму норму).
- Сума дат завершення робіт мінімальна. Ця умова відображає бажання студента виконати всі роботи якомога швидше.

## 2.3 Об'єктна модель предметної області

Представлення задачі в вигляді моделі «сутність-зв'язок» (ER-моделі), надає розуміння про присутні в системі об'єкти, та їх взаємозв'язки. Так, систему доцільно розділити на чотири основні сутності, як показано на рисунку 2.1.

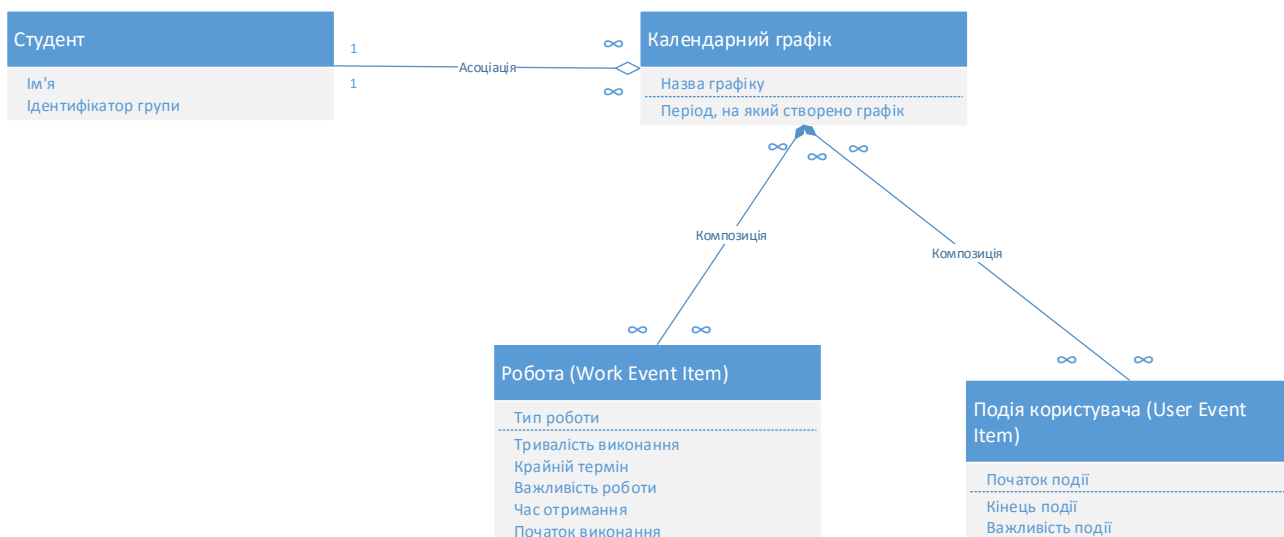


Рис.2.1. Діаграма об'єктів та їх взаємозв'язків (ERD)

Календарний графік включає множину робіт (**WorkEventItem**), яка має наступні атрибути:

- Тип роботи (лабораторна робота, розрахункова робота, додаткова робота, курсова робота, дипломна робота, магістерська робота).
- Крайній термін – дата, до якої необхідно виконати та здати роботу.
- Важливість роботи або пріоритет (низький, середній, високий).
- Початок виконання – величина, значення якої необхідно визначити при формуванні оптимального календарного графіку.

Календарний графік включає множину подій користувача (**UserEventItem**). До таких подій належать ті, які відносяться до власних справ користувача (пов'язані з його працевлаштуванням або дозволенням).



## 2.4 Функціональні аспекти системи. Діаграма прецедентів.

Діаграма прецедентів є графом, що складається з множини акторів, прецедентів (варіантів використання) обмежених границею системи (прямокутних), асоціації між акторами та прецедентами, відношення серед прецедентів та відношень узагальнення між акторами. Такі діаграми відображають елементи моделі варіантів використання, на прикладу рисунку 2.2.

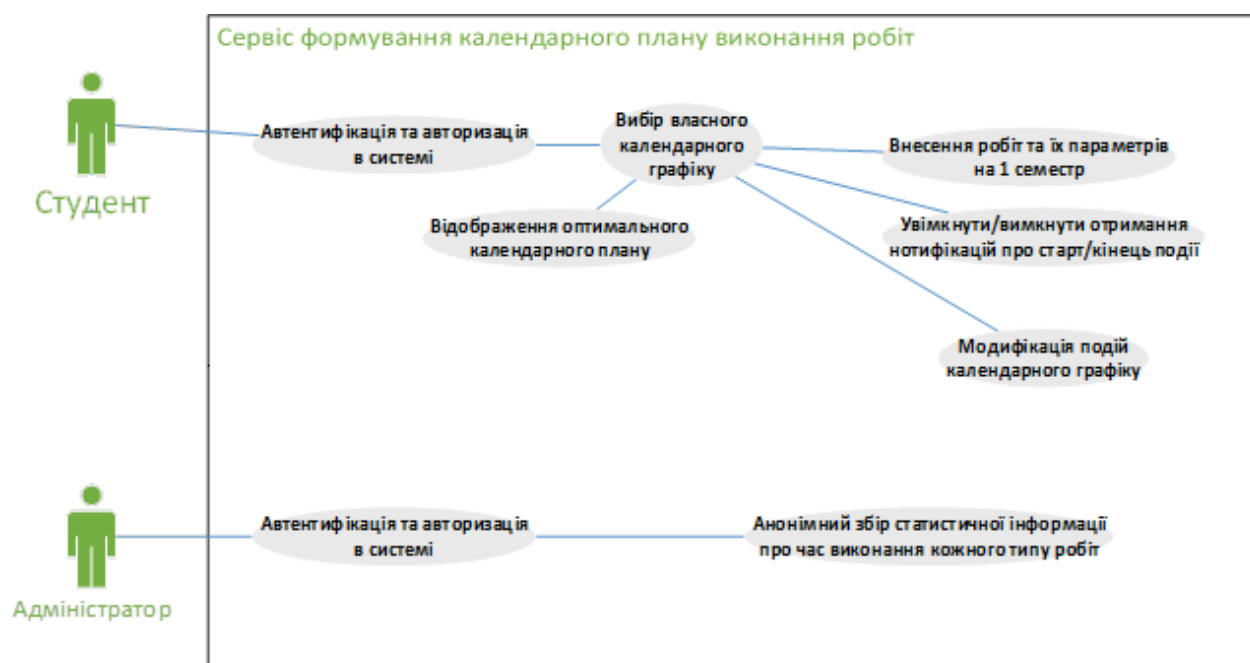


Рис. 2.2. Діаграма прецедентів

### Ролі в системі:

- Студент.
- Адміністратор.

Кожний користувач повинен пройти етап автентифікації та авторизації перед використанням системи задля підтвердження особистості та отримання відповідних прав доступу.

Студент може додавати власні роботи (наприклад додаткові роботи) до вже існуючих робіт для всієї групи. На основі переліку всіх запланованих на 1 семестр робіт та власного календарного графіку студент має можливість отримати оптимальний календарний план виконання робіт.

Адміністратор має доступ до статистичних даних (середній час виконання певного типу роботи в групі, середня завантаженість групи та інші).

## 2.5 Структурні особливості взаємодії об'єктів. Діаграма послідовностей.

Діаграма послідовності – в UML, відображає взаємодії об'єктів, підпорядкованих за часом. Зокрема, такі діаграми відображають задіяні сутності та послідовність відправлених ними повідомлень.

Діаграма послідовності взаємодії студента з системою (рис. 2.3.):

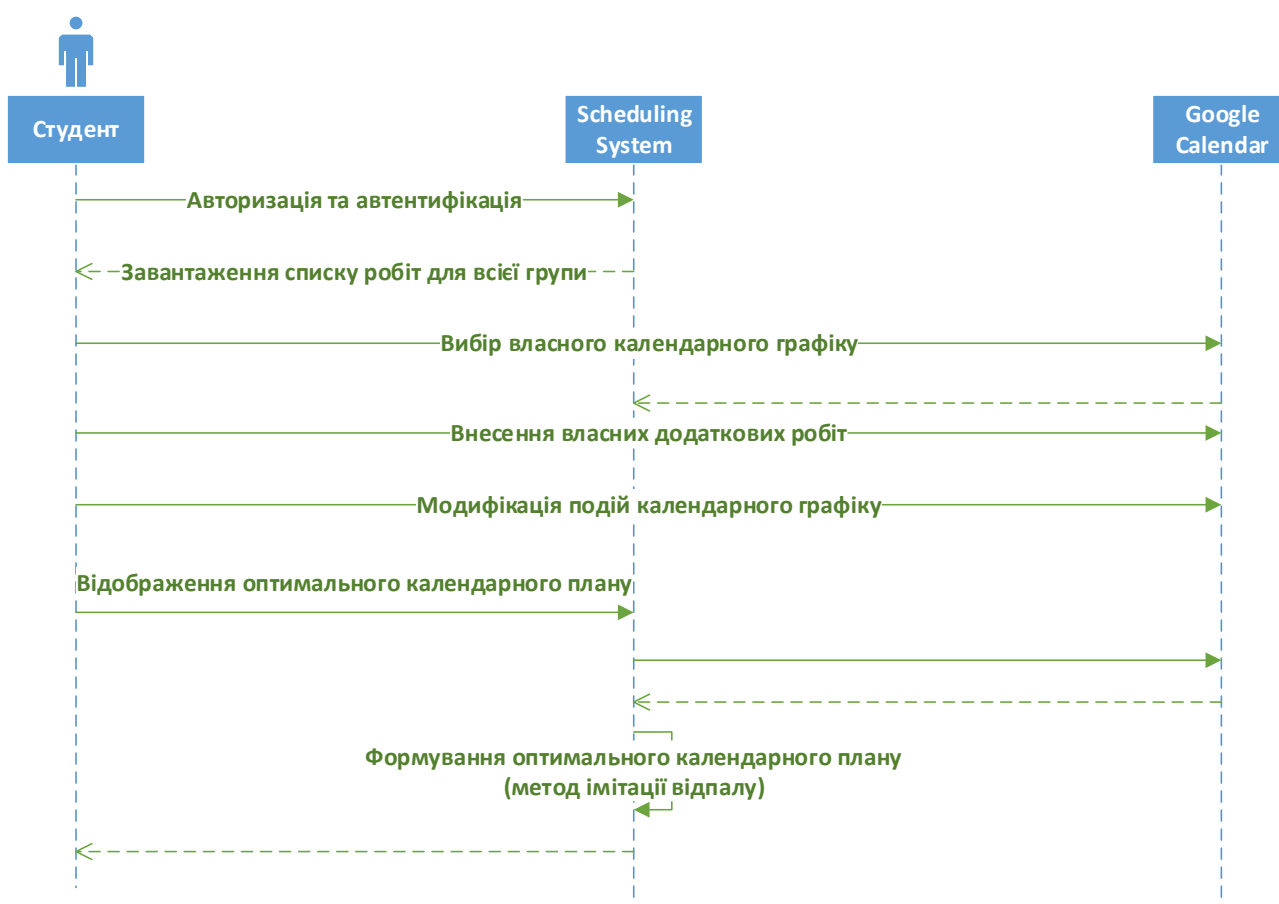


Рис. 2.3. Взаємодія студента з системою

Діаграма послідовності взаємодії адміністратора з системою (рис. 2.4.):

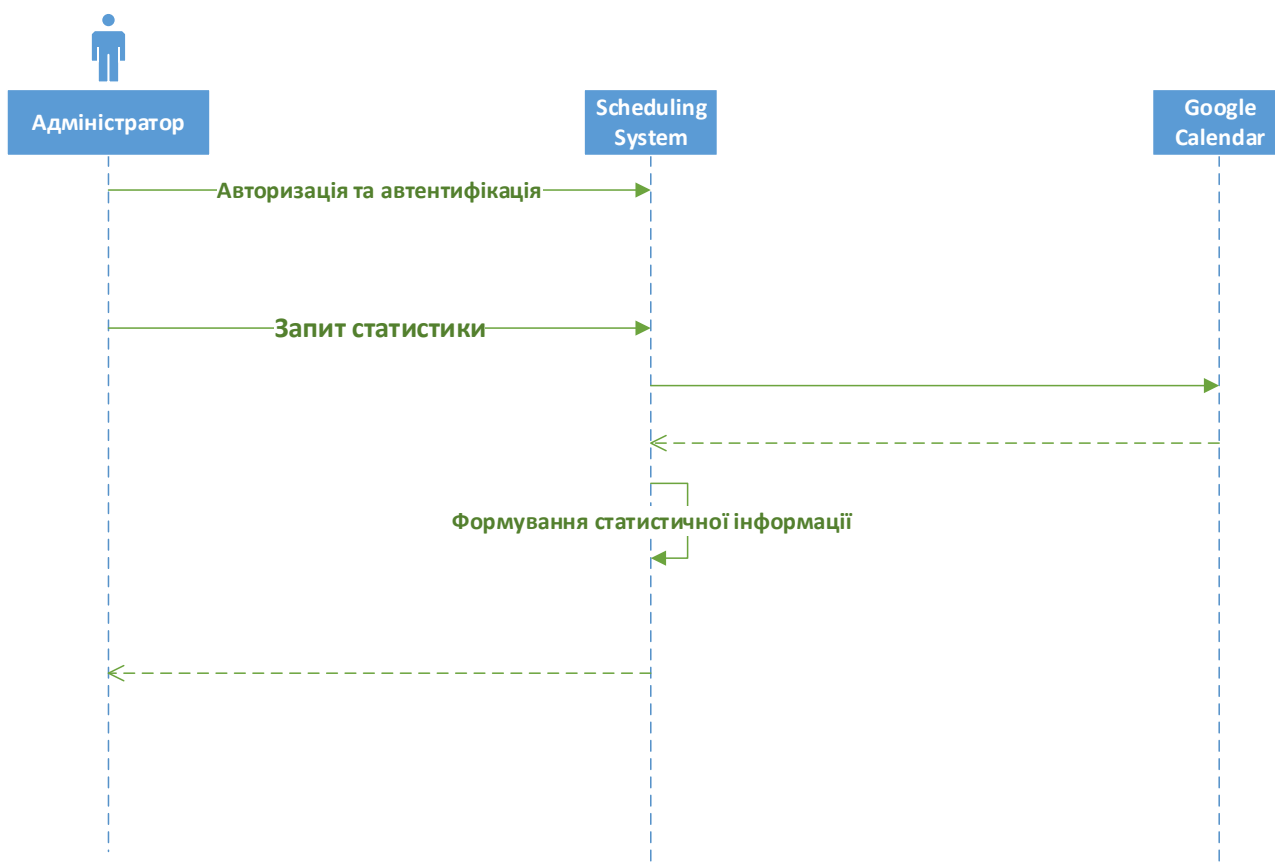


Рис. 2.4. Взаємодія адміністратора з системою

## 2.6 Опис потоків даних. Data Flow Diagram

**DataFlow** – методологія графічного структурного аналізу, що описує зовнішні по відношенню до системи джерела та адресати даних, логічні функції та сховища даних, до яких здійснюється доступ. Представляє собою ієрархію функціональних процесів, що пов’язані між собою потоками даних.

**Діаграма потоків даних (DataFlowdiagram, DFD)** – один з основних інструментів структурного аналізу і проектування інформаційних систем (рис 2.5.).

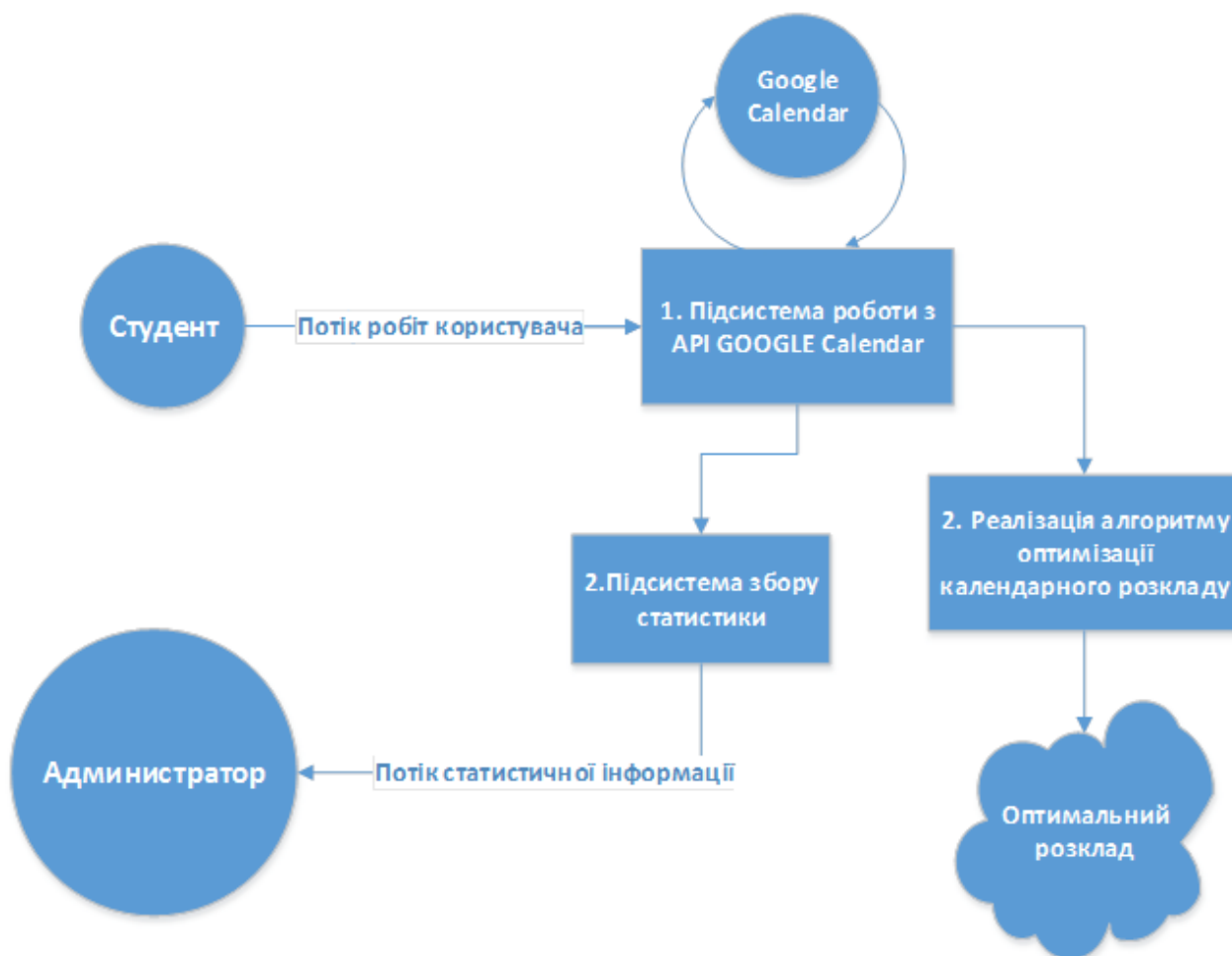


Рис. 2.5. Опис потоків даних

## 2.7 Опис алгоритму планування

Оскільки в нашому випадку задача багатокритеріальної оптимізації, то розглянемо евристичний алгоритм, а саме генетичний, який дозволяє виконати оптимізацію за великою кількістю критеріїв та обмежуючих умов.

### 2.7.1 Опис генетичного алгоритму

**Генетичний алгоритм** (genetic algorithm) — це еволюційний алгоритм пошуку, що використовується для вирішення задач оптимізації і моделювання шляхом послідовного підбору, комбінування і варіації шуканих параметрів з використанням механізмів, що нагадують біологічну еволюцію [6].

Задача кодується таким чином, щоб її вирішення могло бути представлено в вигляді масиву подібного до інформації складу хромосоми. Цей масив часто

називають саме так «хромосома». Випадковим чином в масиві створюється деяка кількість початкових елементів «осіб», або початкова популяція. Особи оцінюються з використанням функції пристосування, в результаті якої кожній особі присвоюється певне значення пристосованості, яке визначає можливість виживання особи. Після цього з використанням отриманих значень пристосованості вибираються особи допущені до схрещення (селекція). До осіб застосовується «генетичні оператори» (в більшості випадків це оператор схрещення (crossover) і оператор мутації (mutation)), створюючи таким чином наступне покоління осіб. Особи наступного покоління також оцінюються застосуванням генетичних операторів і виконується селекція і мутація. Так моделюється еволюційний процес, що продовжується декілька життєвих циклів (поколінь), поки не буде виконано критерій зупинки алгоритму (рис. 2.6). Таким критерієм може бути [15]:

1. знаходження глобального, або над оптимального вирішення;
2. вичерпання числа поколінь, що відпущені на еволюцію;
3. вичерпання часу, відпущеного на еволюцію.

Генетичні алгоритми можуть використати для пошуку рішень в дуже великих і важких просторах пошуку.

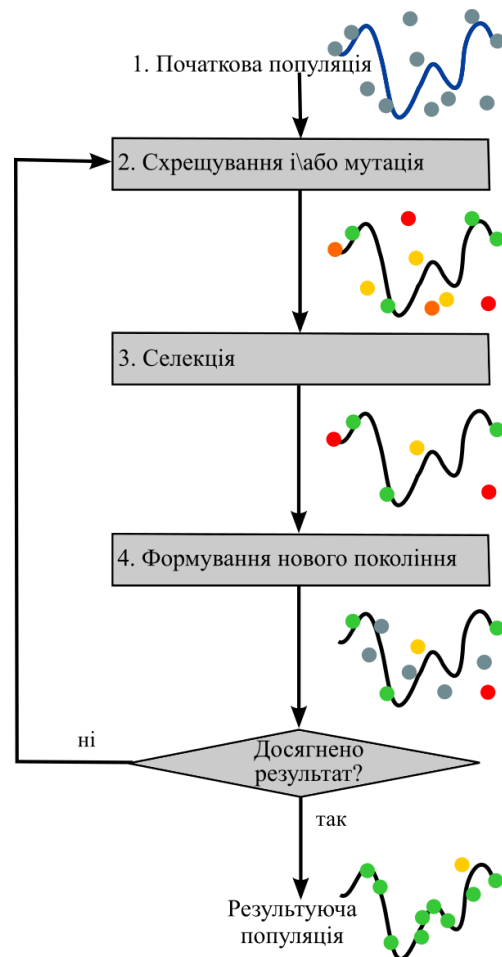


Рис. 2.6. Загальна схема роботи генетичного алгоритму

### 2.7.2 Етапи генетичного алгоритму

Можна виділити такі етапи генетичного алгоритму:

1. Створення початкової популяції:
2. Обчислення функції пристосованості для осіб популяції (оцінювання)
3. Повторювання до виконання критерію зупинки алгоритму:
  1. Вибір індивідів із поточної популяції (селекція)
  2. Схрещення або/та мутація
  3. Обчислення функції пристосовуваності для всіх осіб
  4. Формування нового покоління

### 2.7.3 Створення початкової популяції

Перед першим кроком необхідно випадковим чином створити деяку початкову популяцію. Навіть якщо популяція виявиться абсолютно

неконкурентоздатною, генетичний алгоритм все одно достатньо швидко переведе її в придатну для життя популяцію. Таким чином, на першому кроці можна не старатися зробити надто пристосованих осіб, достатньо, щоб вони відповідали формату осіб популяції, і на них можна було порахувати функцію пристосованості (фітнес функцію). Наслідком першого кроку є популяція  $H$ , що налічує  $N$  осіб.

#### 2.7.4 Відбір

На етапі відбору необхідно із всієї популяції вибрати її певну долю, яка залишиться в «живих» на цьому етапі популяції. Є декілька способів провести відбір. Ймовірність виживання особи  $h$  повинна залежати від значення її пристосованості  $Fitness(h)$ . Сама ж доля відібраних  $s$  зазвичай є параметром генетичного алгоритму, і її просто задають заздалегідь. Внаслідок відбору із  $N$  осіб популяції  $H$  повинні залишитись  $s_N$  осіб, які ввійдуть в наступну популяцію  $H'$ . Решта осіб «загине» [18].

#### 2.7.5 Розмноження

Розмноження в генетичних алгоритмах зазвичай статеве — щоб «народити» нащадку, необхідно декілька батьків, зазвичай потрібна участь двох. Розмноження в різних алгоритмах описується по різному — воно, звісно, залежить від формату осіб. Головна вимога до розмноження — щоб нащадок чи нащадки мали можливість успадкувати риси всіх батьків, «змішавши» їх якимось достатньо розумним чином.

Розмноження або оператор рекомбінації застосовують відразу ж після оператора відбору батьків для отримання нових особин-нащадків. Сенса рекомбінації полягає в тому, що створені нащадки повинні наслідувати генну інформацію від обох батьків. Розрізняють дискретну рекомбінацію і кросинговер.

Приклад операції розмноження: Вибрати  $\frac{(1-s)p}{2}$  пар гіпотез із  $H$  і провести з ними розмноження, отримавши по два нащадку від кожної пари (якщо

розмноження описано так, щоб давати одного нащадку, необхідно вибрати  $(1 - s)p$  пар), і додати цих нащадків в  $N'$ . В результаті  $N'$  буде складатися з  $N$  осіб.

Особи для розмноження зазвичай вибираються із всієї популяції  $N$ , а не із тих, що вижили на першому кроці (хоча останній варіант теж має право на існування). Справа в тому, що головна проблема генетичних алгоритмів — недостача різноманітності (diversity) в особах. Достатньо швидко виділяється єдиний генотип, який являє собою локальний максимум і згодом всі елементи популяції програють йому в відборі, і вся популяція «забивається» копіями цієї особи. Існують різні способи боротьби із таким небажаним ефектом; один з них — вибір для розмноження не з самих «пристосованих», а взагалі зі всіх осіб.

### 2.7.6 Мутації

До мутацій відноситься все те ж, що і до розмноження: є деяка доля мутантів  $m$ , що є параметром генетичного алгоритму, і на кроці мутацій необхідно вибрати  $m_N$  осіб, а згодом змінити їх згідно з задалегідь заданими операціями мутації.

## 2.8 Види кросинговеру

**Одноточковий кросинговер** (Single-point crossover) моделюється наступним чином. Нехай є дві батьківські особини з хромосомами  $X = x_i, i \in [0, L]$  і  $Y = y_i, i \in [0, L]$  [20]. Випадковим чином визначається точка всередині хромосоми (точка розриву), в якій обидві хромосоми діляться на дві частини і обмінюються ними. Такий тип кросинговеру називаються одноточковим, так як при ньому батьківські хромосоми розділяються тільки в одній випадковій точці. Принцип роботи одноточкового кросинговеру зображений на рисунку 2.7.



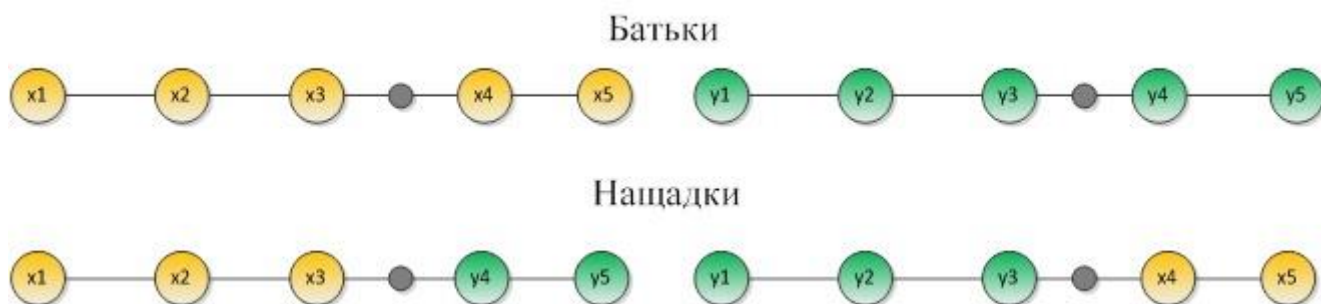


Рис. 2.7. Принцип дії односточкового кросинговеру

У двоточковому кросинговері (і в багатоточковому кросинговері також) хромосоми розглядаються як цикли, які формуються з'єднанням кінців лінійної хромосоми разом. Для заміни сегменту одного циклу сегментом іншого циклу потрібно вибрати дві точки розрізу. З цієї точки зору, односточковий кросинговер може бути розглянутий як кросинговер з двома точками, але з однією точкою розриву, зафіксованій на початку рядка. Отже, двоточковим кросинговером можна вирішувати ті ж задачі, що і односточковим, але більш детально. Хромосома, що розглядається як цикл, може містити більшу кількість стандартних блоків, так як вони можуть здійснити «циклічне повернення» в кінці рядка. На даний момент багато дослідників погоджуються, що двоточковий кросинговер кращий, ніж односточковий. Принцип роботи двоточкового кросинговеру зображений на рисунку 2.8.

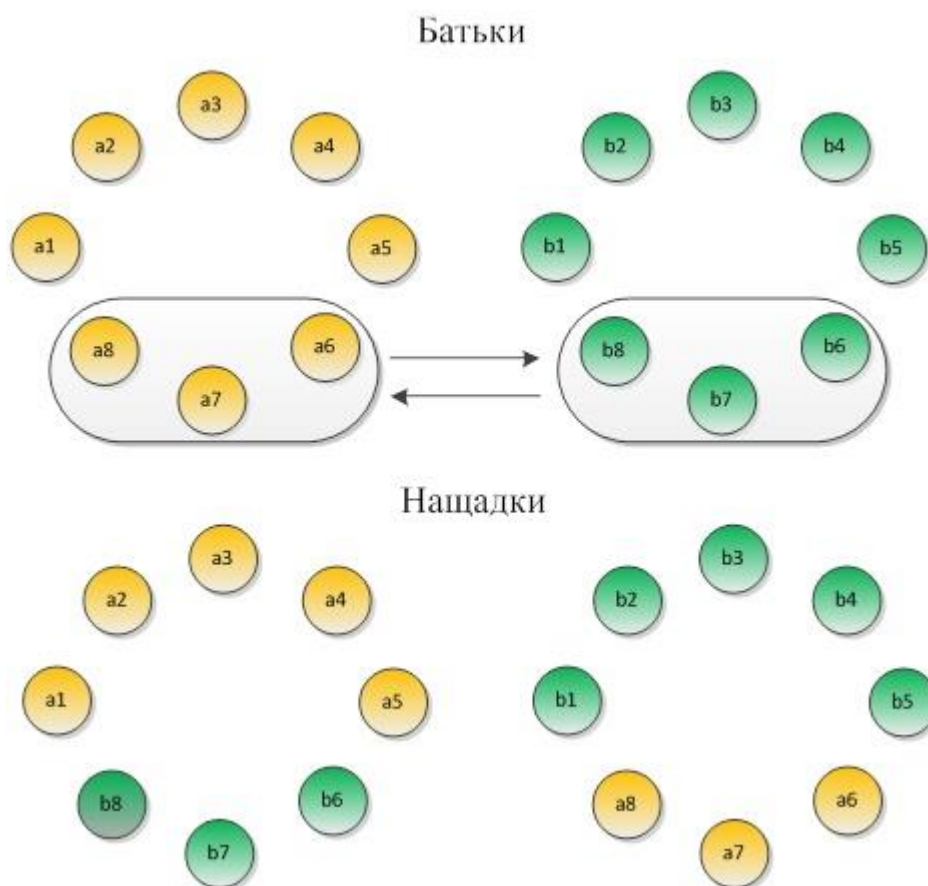


Рис. 2.8. Принцип дії двоточкового кросинговеру

Для багатоточкового кросинговеру (Multi-point crossover), вибираємо  $m$  точок розриву  $k_i \in [1, Nvar]$ ,  $i = (1, \dots, m)$ ,  $Nvar$  - кількість змінних (генів) у особині. Точки розриву вибираються випадково без повторень і сортуються в порядку зростання. При кросинговері походить обмін ділянками хромосом, обмеженими точками розриву і таким чином отримують двох нащадків. Ділянка особини з першим геном до першої точки розриву в обміні не бере участь. Порівняємо наступні дві особини по 11 двійковим генам (рис. 2.9).

Особина 1	0	1	1	1	0	0	1	1	0	1	0
Особина 2	1	0	1	0	1	1	0	0	1	0	1

Рис. 2.9. Дві двійкові батьківські особини.

Оберемо такі точки розриву кросинговеру: 2, 6 і 10. Отримаємо таких нащадків (рис. 2.11):

Нашадок 1	0	1	1	0	1	1	1	1	0	1	1
Нашадок 2	1	0	1	1	0	0	0	0	1	0	0

Рис. 2.10. Дві двійкові особини нащадків.

Застосування багатоточкового кросинговеру вимагає введення декількох змінних (точок розриву), і для відтворення вибираються особини з найбільшою пристосованістю.

Одноточковий і багатоточковий кросинговер визначають точки розрізу, які поділяють особини на частини. Однорідний кросинговер (Uniform crossover) створює маску (схему) особини, в кожному локусі якої знаходиться потенційна точка кросинговеру. Маска кросинговеру має ту ж довжину, що і перехресні особини. Створити маску можна наступним чином: введемо деяку величину  $0 < p_0 < 1$ , і якщо випадкове число більше  $p_0$ , то на  $n$ -у позицію маски ставимо 0, інакше - 1. Таким чином, гени маски є випадковими двійковими числами (0 або 1). Згідно з цими значеннями, геном нащадку стає перша (якщо ген маски = 0) або друга (якщо ген маски = 1) особина-батько. Наприклад, розглянемо особини (табл. 2.1):

Табл. 2.1. Дві батьківські особини.

Особина 1	0	1	1	1	0	0	1	1	0	1	0
Особина 2	1	0	1	0	1	1	0	0	1	0	1

Для кожного створеного потомку попередньо створюється маска із 11 випадково обраних елементів із множини  $\{0,1\}$  (табл. 2.2):

Табл. 2.2. Маски двох особин.

Маска 1	0	1	1	0	0	0	1	1	0	1	0
Маска 2	1	0	0	1	1	1	0	0	1	0	1

Створимо нащадків за наступним правилом: якщо на  $i$ -му місці з відповідній масці стоїть 1, то ген першого батька переходить до потомку, інакше унаслідується ген другого батька. Отримаємо наступні особини (табл. 2.3):

Табл. 2.3. Два нащадки.

Нащадок 1	1	1	1	0	1	1	1	1	1	1	1
Нащадок 2	0	0	1	1	0	0	0	0	0	0	0

Однорідний кросинговер дуже схожий на багатоточковий, але рядок випадкових бітових значень в ньому довший. Це гарантує, що в потомках будуть чергуватися короткі рядки особин-батьків. Алгоритм однорідного кросинговеру для двійкових рядків повністю ідентичний дискретному відтворенню для дійсних хромосом. Такий вид кросинговеру ще називають *уніфікованим*.

Інший варіант кросинговеру, "**Розрізання і склеювання**" (cutandsplice) підхід, призводить до зміни довжини генів дітей (рис. 2.11). Причина цієї відмінності в тому, що кожен батько рядок має окремий вибір точки кросинговеру.

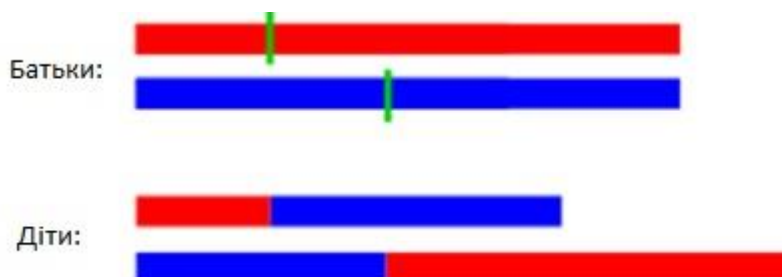


Рис. 2.11. Принцип дії «розрізання і склеювання» кросинговеру

**Однорідний кросинговер** використовує фіксоване співвідношення змішування між двома батьками. На відміну від одно- і двоточкового кросинговеру, однорідний кросинговер дозволяє батьківським хромосомам вносити свій внесок на генному рівні, а не рівні сегмента.

Якщо співвідношення змішування **0,5**, то потомство має приблизно половину генів від першого батька, а інша половина від другого з батьків, хоча точки кросинговеру можуть бути обрані випадковим чином, як показано нижче (рис. 2.12):



Рис. 2.12. Схема однорідного кросинговеру

Однорідний кросинговер оцінює кожен біт в батьківських рядків для обміну з ймовірністю 0,5. Емпіричні дані свідчать про те, що це більш розвідувальний підхід до кросинговеру, ніж традиційні експлуататорний підхід, який підтримує більш довгу схем. Це забезпечує більш повний пошук на просторі параметрів з підтримкою обміну інформації. На жаль, немає задовільної теорії не існує, щоб пояснити розбіжності між однорідним кросинговером і традиційних підходів.

В однорідний схемі кросинговеру окремих бітів в рядку порівнюються між двома батьками. Біти міняються місцями з фіксованою ймовірністю, як правило, 0,5.

В напів-однорідній схемі кросинговеру, рівно половина бітів невідповідних міняються місцями. Таким чином, в першу чергу відстань Хеммінга (число різних бітів) обчислюється. Це число ділиться на дві частини.

Отримане число є скільки бітів, які не відповідають між двома батьками будуть обмінені.

Трьох батьківський кросинговер. У цій техніці, дитину отримують з трьох, випадковим чином обраних, батьків. Кожен біт першого батька порівнюється з тим же бітом другого з батьків. Коли ці біти однакові він використовується в дитині, в іншому випадку біт від третього батька використовується в потомстві. Наприклад, наступні три батьків:

$p_1$  110100010

$p_2$  011001001

$p_3$  110110101

дасть наступне потомство:

$Op_1p_2p_3$  110100001

Кросинговер зі зменшенням заміни (Crossover with reduced surrogate). Оператор зменшення заміни обмежує кросинговер, щоб завжди, коли це можливо, створювати нові особини. Це здійснюється за рахунок обмеження на вибір точки розрізу: точки розрізу повинні з'являтися тільки там, де гени розрізняються.

Як було показано вище, кросинговер генерує нове рішення (у вигляді особини-нащадку) на основі двох наявних, комбінуючи їх частини. Тому число різних рішень, які можуть бути отримані кросинговером при використанні однієї і тієї ж пари готових рішень, обмежена. Відповідно, простір, яке ГА може покрити, використовуючи лише кросинговер, жорстко залежить від генофонду популяції. Чим різноманітніше генотипи рішень популяції, тим більше простір покриття. При виявленні локального оптимуму відповідний йому генотип буде прагнути зайняти всі позиції в популяції, і алгоритм може зійтися до помилкового оптимуму. Тому в генетичному алгоритмі важливу роль відіграють мутації. Існує кілька способів мутації генів. Питання про вибір відповідного оператора мутації вирішується в рамках поставленого завдання [18].

**Впорядкований кросинговер.** Залежно від того, як хромосома являє собою рішення, прямий обмін не може бути можливим. Один з таких випадків,

коли хромосома являє собою впорядкований список, наприклад, як упорядкований список міст, які будуть пройдені для завдання комівояжера. Наш випадок – розклад, як раз підпадає під такий тип, тому що являє собою впорядкований список днів які містять в собі задачі.

Є багато методів кросинговеру для упорядкованих хромосом. Уже згадувалося, що N-точковий перетин може бути застосований для упорядкованих хромосом також, але для цього завжди потрібен відповідний процес ремонту хромосоми. Кілька прикладів для операторів кросинговеру (також операторів мутації), що зберігають обмеження упорядкованості:

### 1. Кросинговер часткової відповідності (PMX partially matched crossover)

Цей метод кросинговер створює нащадку, вибираючи підпоследовність генів від одного з батьків та зберігаючи порядок і положення якомога більшого числа генів іншого батька.

Підпоследовність елементів вибирається шляхом вибору двох випадкових точок перетину, які служать в якості кордонів для операції обміну.

Опис алгоритму (рис. 2.13):

1. Вибір випадкового сегмента копіювання його з  $P_1$  до дитини.
2. Починаючи з першої точки кросинговеру пошукайте ці елементи в сегменті батька  $P_2$ , що не були скопійовані.
3. Для кожного з них  $i$  дивимось в потомку, щоб побачити, що елемент  $j$  був скопійований на своєму місці від  $P_1$
4. Помістіть  $i$  в положення, займане  $j$  в  $P_2$ , так як ми знаємо, що ми не будемо класти туди  $j$  (так як  $j$  вже в потомку)
5. Якщо місце, займане  $j$  в  $P_2$ , вже заповнені в потомку елементом  $k$ , покласти  $i$  в положення займане  $k$  в  $P_2$ .

6. Розібравшись з елементами сегмента батька  $P_1$ , інша частина нащадку може бути заповнена елементами з  $P_2$

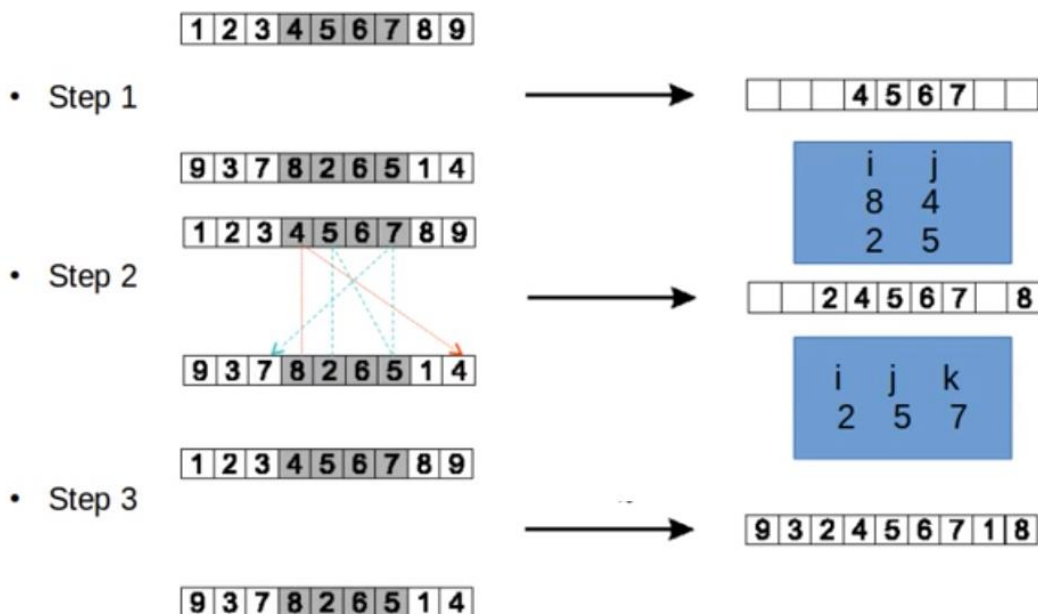


Рис. 2.13. Приклад РМХ кросинговеру

## 2. Циклічний кросинговер (СХ)

Починаючи з будь-якого гена  $i$  в 1-му батьку,  $i$ -й ген в батьку 2 заміщається їм. Те ж саме повторюється для переміщеного гена, поки ген, який дорівнює першому вставленому гену не заміниться й цикл замкнеться.

Опис алгоритму (рис.2.14):

1. Складіть цикл елементів з  $P_1$  наступним чином
  - a. Почніть з першого елемента
  - b. Подивіться на елемент в тому ж положенні в  $P_2$
  - c. Перейти до позиції з тим же елементом в  $P_1$
  - d. Додайте цей елемент в цикл
  - e. Повторити кроки від b до d, поки не вернемось на перший елемент
2. Помістіть елементи циклу у нащадок на позиції, які вони мають в першому батьку
3. Береться наступний цикл від другого з батьків



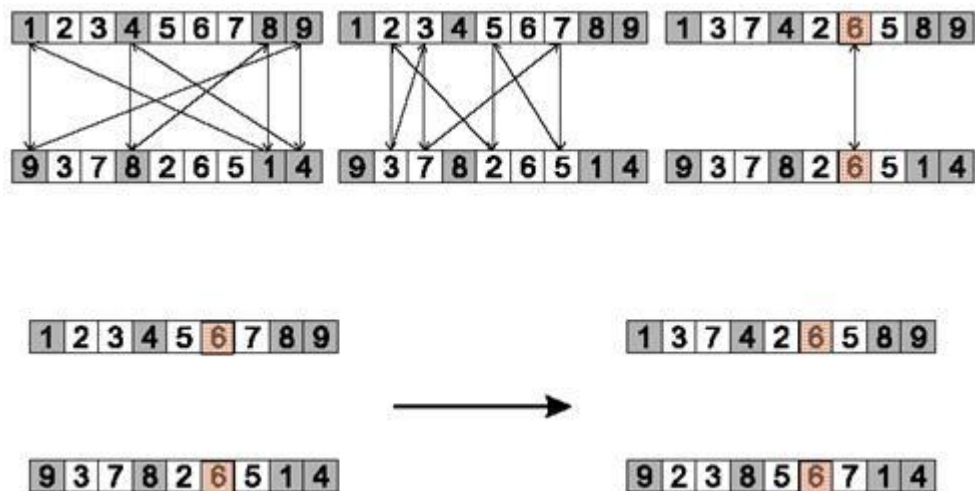


Рис. 2.14. Приклад циклічного кросинговеру

3. Впорядкований кросинговер (OX1): Частина одного з батьків береться за основу нащадку. Далі в отриманий нащадок недостаючи гени беруться з другого батька зберігаючи порядок та пропускаючи ті, що вже є в частині від першого батька (рис. 2.15). Таким чином не порушуються жодне з обмежень впорядкованого хромосому.

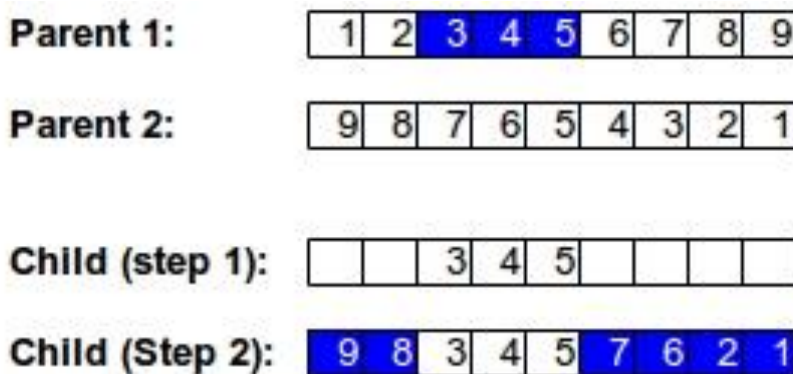


Рис. 2.15. Приклад впорядкованого кросинговеру

4. Модифікація впорядкованого кросинговеру (OX2)
5. Позиційний кросинговер (POS)
6. Кросинговер голосування рекомбінацій (VR)
7. Кросинговеру змінного положення (AP)
8. Послідовно конструктивний кросинговер (SCX)

## 2.9 Види мутації

Після процесу відтворення відбуваються мутації (mutation). Даний оператор необхідний для «вибивання» популяції з локального екстремуму і перешкоджає передчасної збіжності. Це досягається за рахунок того, що змінюється випадково обраний ген в хромосомі (рис. 2.16).

$$x_1 x_2 \dots x_{n-1} x_n x_{n+1} \dots x_m \longrightarrow x_1 x_2 \dots x_{n-1} \bar{x}_n x_{n+1} \dots x_m$$

Рис. 2.16. Мутація в точці  $x_n$

Так само, як і кросинговер, мутації можуть проводитися не тільки по одній випадковій точці. Можна вибрати для зміни кілька точок в хромосомі, причому їх число також може бути випадковим. Використовують і мутації зі зміною відразу деякої групи послідовних точок. Імовірність мутації  $p_m$  (як правило,  $p_m \ll 1$ ) може бути або фіксованим випадковим числом на відрізку  $[0; 1]$ , або функцією від будь-якої характеристики розв'язуваної задачі. Наприклад, можна покласти ймовірність мутації генів, обернено пропорційного числу всіх генів в особини (розмірності). Оптимальне значення ймовірності мутації обговорюється в різних статтях. Так, наприклад, мутація з фіксованою ймовірністю призводить до гарних результатів для широкого класу тестових функцій (наприклад, для унімодальних функцій). Для мультимодальних функцій застосовують само-адаптовану оцінку ймовірності. Нижче наведені основні варіанти мутації.

**Мутація для особин з матеріальними генами (Real valued mutation).** На рис. 2.17. показана можлива мутація для особин з речовими генами в двовимірному просторі.

Для мутації особин з речовими числами необхідно визначити величину кроку мутації - число, на яке зміниться значення гена при мутації.

Зазвичай визначення кроку мутації представляє деяку складність. Оптимальний розмір кроку повинен змінюватися протягом усього процесу

пошуку. Найбільш придатні маленькі кроки, але іноді великі кроки можуть призвести до прискорення процесу. Гени можуть мутувати згідно наступним правилом:

$$\text{нова змінна} = \text{стару змінну} \pm \alpha \cdot \delta,$$

де знаки + або - вибираються з однаковою ймовірністю,  $\alpha = 0,5 \times \times$  пошукове простір (інтервал зміни даної змінної),

$$\delta = \sum a(i)2^{-i} \quad (2.3)$$

$a(i) = 1$  з ймовірністю  $\frac{1}{m}$ , в іншому випадку  $a(i) = 0$ ,  $m$  — параметр.

Нова особина, що вийшла при такій мутації, в більшості випадків не набагато відрізняється від старої. Це пов'язано з тим, що ймовірність маленького кроку мутації вище, ніж вірогідність великого кроку. При  $m = 20$ , даний алгоритм мутації придатний для локалізації оптимуму з точністю  $\alpha \cdot 2^{-19}$ .

Двійкова мутація (Binary mutation). Для особин, кодованих двійковим кодом або кодом Грея, мутація полягає у випадковому інвертуванні гена (0 замінюється 1 і навпаки). Ефект мутації залежить від застосованого способу кодування генів. Так, в одних завданнях при мутації найкращий ефект досягається в разі, коли особини закодовані кодом Грея, а в інших - за допомогою двійкового коду.

Щільність мутації (Density mutation). Стратегія мутації з використанням поняття щільності полягає в мутації кожного гена нащадку із заданою вірогідністю. Таким чином, крім ймовірності застосування мутації до самого нащадкові використовується ще ймовірність застосування мутації до кожного його гену, величину якої вибирають з таким розрахунком, щоб в середньому мутували від 1 до 10% генів.

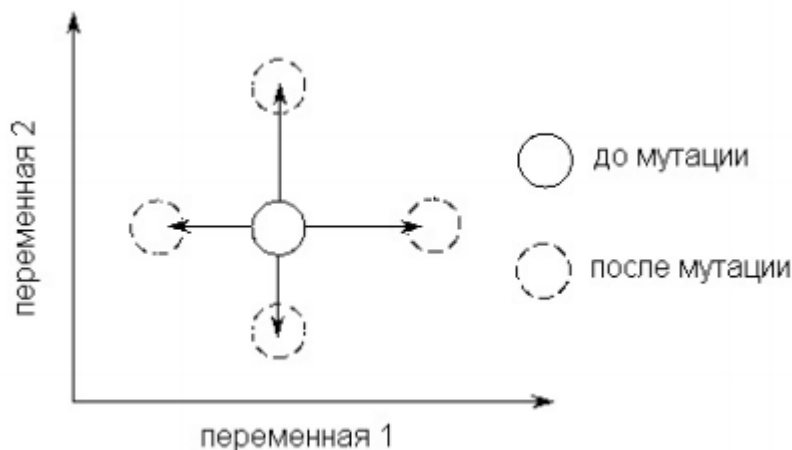


Рис. 2.17. Мутація для особин з матеріальними генами

**Інші види мутацій [19].** Нехай особина  $t$  представлена такою послідовністю генів  $t_i: t = t_1, \dots, t_k$ . Тоді можна застосувати наступні оператори мутації:

1. *Приєднання* (рис. 2.18) випадкового гена з сукупності всіляких значень генів до кінця послідовності:  $t \rightarrow t_1, \dots, t_k, s$ .

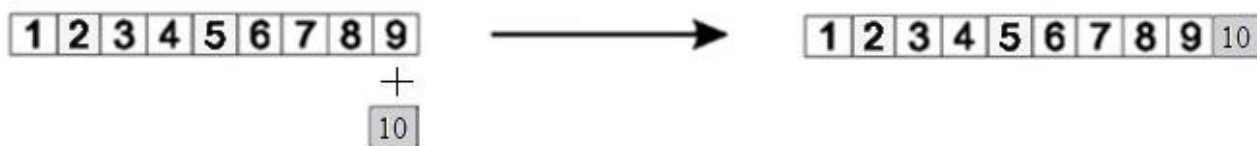


Рис. 2.18. Приклад приєднання.

2. *Вставка* (рис. 2.19) випадкового гена з сукупності всіляких значень генів в випадково обрану позицію в послідовності:  $t \rightarrow t_1, \dots, t_{i-1}, s, t_i, \dots, t_k$ .

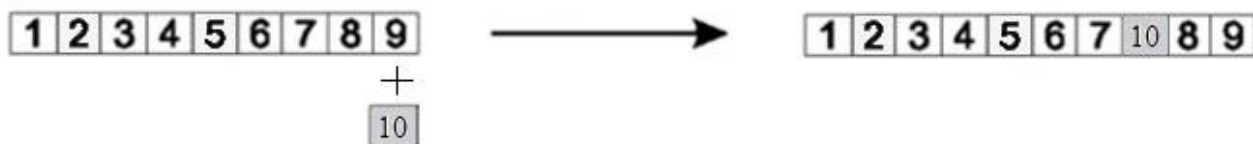


Рис. 2.19. Приклад вставки.

3. *Видалення* (рис. 2.20) випадково обраного гена з послідовності:  $t \rightarrow t_1, \dots, t_{i-1}, t_{i+1}, \dots, t_k$ .



Рис. 2.20. Приклад видалення.

4. *Обмін* (рис. 2.21) місцями в послідовності двох сусідів одного випадково обраного гена:  $t \rightarrow t_1, \dots, t_{i+1}, t_i, t_{i-1}, \dots, t_k$ .



Рис. 2.21. Приклад обміну.

5. *Інверсія* (рис. 2.22) послідовності генів на проміжку заданої довжини:  $t \rightarrow t_1, \dots, t_{i+2}, t_{i+1}, t_i, t_{i-1}, t_{i-2}, \dots, t_k$ .



Рис. 2.22. Приклад інверсії.

6. *Скремблювання* (рис. 2.23) послідовності генів на проміжку. Тобто випадковому перемішуванню генів:  $t \rightarrow t_1, \dots, t_i, t_{i-2}, t_{i+2}, t_{i+1}, t_{i-1}, \dots, t_k$ .



Рис. 2.23. Приклад скремблювання.

Слід зауважити, що мутація 1 є окремим випадком мутації 2. Для особин з фіксованим розміром (кількість генів в послідовності) можливе застосування в чистому вигляді лише 4 мутації, а 1 і 2 мутації повинні застосовуватися в поєднанні з мутацією 3.

## 2.10 Висновки до розділу 2

В даному розділі було наведено математичний опис предметної області. Була проведена об'єктна декомпозиція та представлення взаємозв'язків між

сутностями предметної області результатом якої є ERD діаграми. Створена діаграма прецедентів, що відображає основні функціональні аспекти системи та відносини розширення між акторами системи. Діаграми послідовностей представляють порядок взаємодії сутностей системи та їх обмін повідомленнями. Проведено графічний структурний аналіз за допомогою Data Flow методології.

Проведено детальний аналіз особливостей генетичного алгоритму. Варіації етапів генетичного алгоритму дали можливість підібрати варіант, що не порушує наявних критеріїв. Було обрано впорядкований кросинговер (OX1). В якому частина одного з батьків береться за основу нащадку. Далі в отриманий нащадок недостаючи гени беруться з другого батька зберігаючи порядок та пропускаючи ті, що вже є в частині від першого батька. Таким чином не порушуються жодне з обмежень. Найкращими показниками простоти реалізації і швидкодії відповідає метод мутації обміну, що змінює в послідовності двох сусідів одного випадково обраного гена.

## 3 ВИБІР ІНСТРУМЕНТАЛЬНОЇ БАЗИ РОЗРОБКИ

### 3.1. Вибір бази даних. Порівняння MySQL і PostgreSQL

Бази даних - це спеціально розроблене сховище для різних типів даних. Кожна база даних, має певну модель (реляційна, документно-орієнтована), яка забезпечує зручний доступ до даних. Системи управління базами даних (СУБД) - спеціальні програми (або бібліотеки) для управління базами даних різних розмірів і форм.

Для використання СУБД в даній системі, вона повинна забезпечувати реляційну модель роботи з даними. Сама модель має на увазі певний тип зв'язку між сутностями з різних таблиць. Щоб зберігати і працювати з даними, такий тип СУБД повинен мати певну структуру (таблиці). У таблицях кожен стовпець може містити дані різного типу. Кожен запис складається з безлічі атрибутів (стовпців) і має унікальний ключ, що зберігається в тій же таблиці - всі ці дані взаємопов'язані між собою, як описано в реляційної моделі. Додатковою вимогою є безкоштовне розповсюдження.

Даним критеріям відповідають такі бази даних: MySQL і PostgreSQL. Далі у таблиці 1 наведено порівняння найбільш вживаних особливостей і можливостей цих СУБД.

Таблиця 3.1 - це не вичерпний список особливостей, типів даних або проблем продуктивності, що стосується цих двох систем баз даних - вона лише дає деяке уявлення про те, що кожна з них може запропонувати. З таблиці бачимо, що PostgreSQL пропонує повні особливості та можливості традиційних додатків баз даних, в той час як MySQL зосереджується на більш швидкому виконанні (роботі) для веб додатків. Розвиток індустрії «відкритих початкових кодів» принесе більшу кількість особливостей і можливостей в наступних версіях обох баз даних [20].

Таблиця 3.1. Порівняння MySQL і PostgreSQL

Особливості	PostgreSQL	MySQL
ANSI SQL сумісність	Близька до стандарту ANSI SQL	Слід деяким стандартам ANSI SQL
Швидкість роботи	Повільніше	Швидше
Вкладені запити	Так	Ні
Транзакції	Так	Так, проте повинен використовуватися тип таблиці InnoDB
Відповідь бази даних	Так	Так
Підтримка зовнішніх ключів	Так	Ні
Уявлення	Так	Ні
Збережені процедури	Так	Ні
Тригери	Так	Ні
Unions	Так	Ні
Повні Joins	Так	Ні
Обмежувачі цілісності	Так	Ні
Підтримка Windows	Так	Так
Вакуум (очищення)	Так	Ні
ODBC	Так	Так
JDBC	Так	Так
Різні типи таблиць	Ні	Так



### 3.1.1. Коли використовувати MySQL

Спочатку, розглянуто потреби додатків в термінах вимог бази даних. Якщо при створенні веб додатку, головне це продуктивність і швидкість - MySQL буде кращим вибором, тому що вона швидка і розроблена для того, щоб добре працювати з веб серверами. Однак, для створення програми, яка вимагає виконання транзакцій і наявності зовнішніх ключів, кращим вибором стане PostgreSQL.

Навіть при тому, що MySQL в повному обсязі сумісна з ANSI SQL стандартом, в той час PostgreSQL ближче до ANSI SQL стандарту, MySQL ближче до ODBC стандарту.

Деякі плюси використання MySQL:

1. MySQL швидше PostgreSQL.
2. Дизайн і планування бази даних дещо простіше.
3. Можна створити простий веб сайт з використанням бази.
4. Відповіді на запити MySQL були добре протестовані.
5. Немає потреби використовувати методи очищення.

### 3.1.2. Коли використовувати PostgreSQL

Багато веб-розробників використовують в своїй роботі PostgreSQL, так як вважають, що додаткові особливості і можливості знижують продуктивність і швидкість роботи. Однак, PostgreSQL має багато переваг над MySQL.

Наприклад, деякі з особливостей, які часто використовуються - зовнішні ключі, тригери та подання. Вони дозволяють приховувати складність бази даних від додатка, таким чином уникаючи створення складних команд SQL. Існує чимало розробників, які вважають за краще багаті функціональні можливості SQL команд PostgreSQL. Одне з найбільш відчутних відмінностей між MySQL і PostgreSQL - неможливість створення вкладених під запитів (селектів) в MySQL. PostgreSQL відповідає багатьма SQL стандартам ANSI, таким чином дозволяючи створення складних команд SQL.

Кілька причин використовувати PostgreSQL:

1. Складний дизайн бази даних.
2. Переїзд з Oracle, Sybase або MSSQL.
3. Складні набори правил.
4. Використання процедурних мов на сервері.
5. Транзакції
6. Використання збережених процедур.
7. Використання географічних даних.
8. R-Trees (наприклад, використання індексів).
9. Нова функціональність додається без зупинки сервера.
10. Надійність

Підсумовуючи вище сказане, обрано базу даних PostgreSQL. Використання зовнішніх ключів необхідна умова для зв'язку між об'єктами системи. Також об'єктно-реляційна система керування базами даних PostgreSQL, дозволить повністю зосередитися на розробці програми, уникаючи будь-які проблеми пов'язані зі збереженням даних.

## **3.2. Веб фреймворк Spring Framework**

Платформа Spring - популярна платформа додатків з відкритим кодом, призначена для спрощення розробки на J2EE. Вона складається з контейнера, платформи управління елементами і набору інтегрованих служб для веб-інтерфейсів користувача, транзакцій і збереження стану. До складу платформи Spring Spring Web Входить MVC - розширювана платформа MVC для створення веб-додатків [21].

Незважаючи на те, що Spring Framework не забезпечував якусь конкретну модель програмування, він є широко поширеним в Java-співтоваристві головним чином як альтернатива і заміна моделі Enterprise JavaBeans. Spring Framework надає велику свободу Java-розробникам в проектуванні; крім того, він надає добре документовані і легкі у використанні засоби вирішення проблем, що виникають при створенні додатків корпоративного масштабу.

Spring Framework забезпечує вирішення багатьох завдань, з якими стикаються Java-розробники і організації, які хочуть створити інформаційну систему, засновану на платформі Java. Через широкую функціональність важко визначити найбільш значущі структурні елементи, з яких він складається. Spring Framework НЕ повністю пов'язані з платформою Java Enterprise, незважаючи на його масштабну інтеграцію з нею, що є важливою причиною його популярності. Spring Framework, ймовірно, найбільш відомий як джерело розширень (функції), потрібних для ефективної розробки складних бізнес-додатків поза великовагових програмних моделей, які історично були домінуючими в промисловості. Ще одне його достоїнство в тому, що він ввів раніше невикористовувані функціональні можливості в сьгоднішні панівні методи розробки, навіть поза платформи Java.

Цей фреймворк пропонує послідовну модель і робить її придатною до більшості типів додатків, які вже створені на основі платформи Java. Вважається, що Spring Framework реалізує модель Розробки, засновану на кращих стандартах індустрії, і робить її доступною в багатьох областях Java.

Основна перевага Spring'a - можливість розробки програми як набору слабосвязаних (пухкий зв'язком) компонентів. Чим менше компоненти програми знають один про одного, тим простіше розробляти новий і підтримувати існуючий функціонал програми. Класичний приклад - управління транзакціями. Spring дозволяє вам керувати транзакціями абсолютно незалежно від основної логіки взаємодії з БД. Зміна цієї логіки не порушить транзакційність, так само як зміна логіки управління транзакціями не зламає логіку програми. Spring модульність заохочує. Компоненти можна додавати і видаляти (майже) незалежно один від одного. В принципі, додаток можна розробити таким чином, що воно навіть не буде знати, що управляється Spring'ом. Також Spring помітно спрощує модульне тестування (модульного тестування): в компонент, розроблений для роботи в ІоС контейнері дуже легко інжектувати фейковий залежності і перевірити роботу тільки цього компонента. Ну, і як приємне доповнення, весна сильно полегшує ініціалізацію і настройку

компонентів додатка, дозволяючи гнучко налаштовувати додаток без істотних змін Java-коду.

Заохочення слабкої зв'язаності компонентів, і, як наслідок:

1. спрощення ініціалізації і налаштування компонентів,
2. спрощення модульного тестування,
3. спрощення розробки та підтримки програми в цілому.

Тож, так як Spring був розроблений як альтернатива EJB прямо з самого початку, і пропонує багато можливостей, що полегшують розробку. Використання цього фреймворку значно збільшить якість розробки.

### 3.3. Шаблон проектування MVC

Принцип MVC у веб-програмуванні (Model - View - Controller, Модель - Подання (Вид) - Контролер) - одна з найбільш вдалих ідей на сьогоднішній день [22]. Принцип MVC інтуїтивно зрозумілий на перший погляд, але не дуже простий при поглибленні. Спочатку розглянемо, для чого він призначений.

Принцип MVC, дозволяє розділити реалізацію логіки докладання, зовнішній вигляд (графічний інтерфейс, GUI) і взаємодія з користувачем.

Це призводить до більш структурованого коду, дозволяє працювати над проектом більш спеціалізованим людям, спрощує підтримку коду, робить його більш логічним і зрозумілим. Зміна в одному з компонентів мінімально впливає на інші. Можна до однієї моделі підключати різні види, різні контролери.

Розглянемо докладніше компоненти.

**Model** (Модель) - містить так звану "Бізнес-логіку" - обробку і верифікацію даних, звернення до баз даних, представляє внутрішній устрій системи. Модель не повинна безпосередньо взаємодіяти з користувачем.

**View** (Вид, Подання) описує зовнішній вигляд програми.

**Controller** (Контролер) - сполучна ланка між моделлю і видом, отримує дані від користувача, передає їх моделі, отримує оброблений результат і передає його в уявлення.

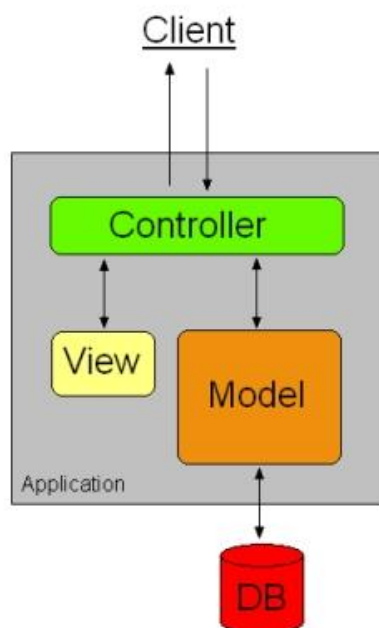


Рис. 3.1. Модель MVC.

Springframework надає гарну можливість для реалізації архітектури MVC за допомогою Spring web MVC framework.

### 3.4. Опис Spring web MVC фреймворку

Spring web MVC framework забезпечує архітектуру модель-уявлення-контролер (MVC) і готові компоненти, які можуть бути використані для розробки гнучких і слабо пов'язаних веб-додатків. Шаблон MVC призводить до поділу різних аспектів застосування (вхідні логіка, бізнес-логіки і логіки UI), забезпечуючи при цьому слабкий зв'язок між цими елементами.

1. **Model** інкапсулює дані додатка і в цілому вони будуть складатися з POJO.
2. **View** відповідає за надання даних моделі і в цілому він генерує HTML, що браузер клієнта може інтерпретувати.
3. **Controller** відповідає за обробку запитів користувачів і побудови відповідної моделі і передає його в уявлення для відображення на екран.

### 3.4.1. Обробник запитів `DispatcherServlet`

Spring Web model-view-controller (MVC) framework розроблений навколо `DispatcherServlet`, який обробляє всі HTTP-запити і відповіді. Обробка запиту робочого процесу в Spring Web MVC `DispatcherServlet` показано на рисунку 3.2:

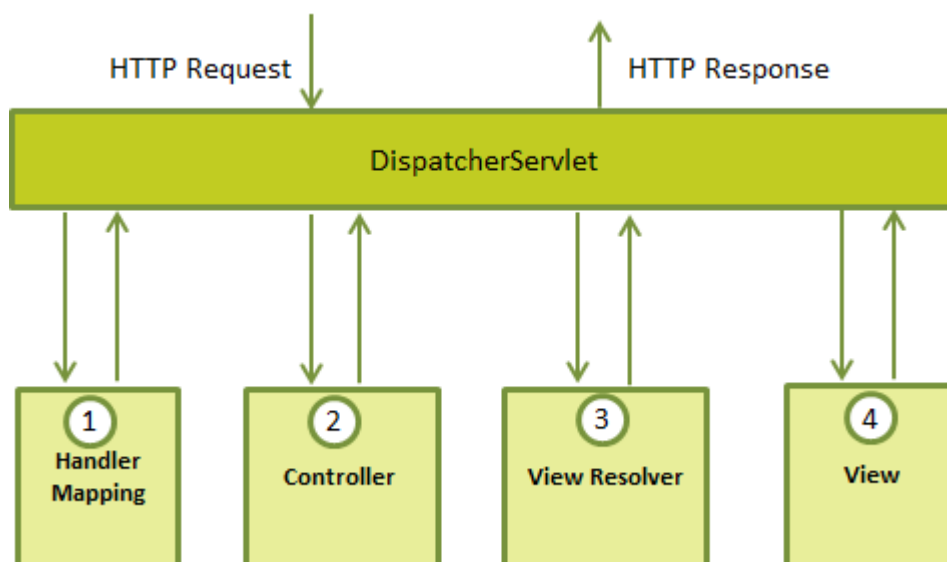


Рис. 3.2. Обробка запиту робочого процесу в Spring Web MVC `DispatcherServlet`

Нижче наводиться послідовність подій, відповідних вхідного запиту HTTP до `DispatcherServlet`:

1. Після отримання запиту HTTP, `DispatcherServlet` консультиється `theHandlerMapping` для виклику відповідного *Контролера*.
2. *Контролер* приймає запит і викликає відповідні методи сервісів на основі використовуваного методу GET або POST. Метод служби встановлює дані моделі, засновані на певній бізнес-логіці і повертає ім'я перегляду в `DispatcherServlet`.
3. `DispatcherServlet` буде приймати допомогу від `ViewResolver` пікап певний вид для запиту.
4. Після того, як перегляд завершено, `DispatcherServlet` передає дані моделі з точки зору, яка, нарешті, винесеному в браузері.

### 3.5. Авторизація користувача

Користувач має три рівня доступу до додатка: незнайомец, зареєстрований і адміністратор. Для забезпечення даної функціональності потрібно зробити можливість авторизації і автентифікації в ресурсі. Spring framework надає таку можливість за допомогою Spring security [23].

Spring Security надає комплексні послуги безпеки для J2EE на основі корпоративних програмних додатків. Існує особливий акцент на підтримці проектів, створених з використанням Spring Framework, яка є провідним рішенням J2EE для розробки програмного забезпечення підприємства.

Для використання Spring Security є багато причин, але основною є те, що можливостям безпеки J2EE's Servlet Specification або EJB Specification не вистачає глибини, необхідної для типових сценаріїв корпоративних додатків. Незважаючи на це, згадуючи ці стандарти, важливо визнати, що вони не переносяться на WAR або EAR рівні. Тому, при переході до серверних середовищ, як правило, займає багато роботи, зміна конфігурації безпеки застосування до нового середовища. Використання Spring Security долає ці проблеми, а також приносить десятки інших корисних налаштовуваних функцій безпеки.

Дві основні галузі безпеки додатків є "автентифікація" і "авторизація" (або "контролю доступу"). Це дві основні області, Spring Security. «Автентифікація» являє собою процес встановлення ким user є для системи (а "user" зазвичай означає користувача, пристрій або будь-якої іншої системи, яка може виконати дію в додатку). "Авторизація" відноситься до процесу прийняття рішення чи користувач той за кого себе видає, або чи дозволено виконувати дії в вашому додатку.

На рівні автентифікації Spring Security підтримує широкий спектр моделей автентифікації. Більшість з цих моделей автентифікації або надаються третіми особами, або розробляються відповідними органами стандартизації, такими як Engineering Task Force Internet. Крім того, Spring Security надає свій власний

набір функцій автентифікації. Зокрема, в даний час Spring Security підтримує інтеграцію автентифікації з усіма цими технологіями:

1. HTTP Basic заголовків перевірки автентичності (стандарт RFC IETF основі)
2. HTTP Digest заголовки автентифікації (стандартом RFC IETF основі)
3. HTTP X.509 клієнт обміну сертифікатів (і IETF RFC на основі стандарт)
4. LDAP (дуже поширений підхід до потреб автентифікації крос-платформних, особливо в великих середовищах)
5. Автентифікація на основі форм (для простих потреб користувальницького інтерфейсу)
6. Автентифікація OpenID
7. Автентифікація на основі заздалегідь встановлених заголовків запиту (наприклад, Computer Associates Siteminder)
8. JA-SIG Центральна служба перевірки справжності (інакше відомий як CAS, який є популярним відкритим вихідним кодом єдиного входу системи)
9. Прозора поширення контексту автентифікації для віддаленого виклику методу (RMI) і HttpInvoker (а протокол Spring Remoting)
10. Автоматичний "пам'ятає мене" автентифікації (так що ви можете поставити галочку в полі, щоб уникнути повторної автентифікації протягом заданого періоду часу)
11. Анонімна перевірка справжності (що дозволяє кожен виклик автоматично припускати певну ідентичність безпеки)
12. Run-автентифікацією (що корисно, якщо один виклик повинен приступити до іншої ідентичності безпеки)
13. Java автентифікації і авторизації обслуговування (JAAS)
14. JEE container authentication (так що ви можете використовувати Container Managed Authentication при бажанні)
15. Kerberos
16. Java Open Source Single Sign On (JOSSO)



17. OpenNMS Management Network Platform
18. AppFuse
19. AndroMDA
20. Mule ESB
21. Пряма веб-запиту (ДСР)
22. Grails
23. Tapestry
24. JTrac
25. Jasypt
26. Roller
27. Elastic Path
28. Atlassian Crowd

Багато незалежних постачальників програмного забезпечення (ISV) використовують Spring Security через гнучкий вибір моделей аутентифікації. Це дозволяє їм швидко інтегрувати свої рішення з тим, що потрібно їх кінцевим клієнтам, без проведення багатьох змін або вимагання клієнта змінити середовище розробки. При незадоволенні перерахованих вище механізмів аутентифікації, Spring Security є відкритою платформою, і досить просто написати свій власний механізм аутентифікації. Багатьом корпоративним користувачем Spring Security необхідно інтегрувати з "legacy" системами, які не дотримуються будь-яких конкретних стандартів безпеки і Spring Security легко інтегрується з такими системами.

Коли недостатньо простого процесу автентифікації, і необхідно диференціювати рівні безпеки, на основі способів взаємодії користувача з додатком. Наприклад, забезпечення запитів тільки через HTTPS, щоб захистити паролі від підслуховування або кінцевих користувачів атак шахраїв. Це особливо корисно для захисту процесів відновлення пароля від атак методом "трубої сили", або просто, щоб зробити його більш складним для людей, або щоб дублювати ключовий зміст додатку. Spring Security повністю підтримує автоматичну "channel security", разом з інтеграцією JCAPTCHA для переконання,

що користувач людина. Незалежно від того, як була проведена перевірка справжності, Spring Security забезпечує глибокий набір можливостей авторизації.

### 3.6. Об'єктно-реляційна проєкція. Hibernate

Основна проблема при роботі з реляційними базами даних – зв'язування об'єктів розроблювальної системи з таблицями БД. Працюючи, виникає необхідність пам'ятати особливості СУБД, для пов'язаних об'єктів постійне контролювання даних в таблицях. Рішенням цієї проблеми є використання об'єктно-реляційних проєкцій. Найкращим рішенням для мови програмування Java є Hibernate.

**Hibernate** — засіб відображення між об'єктами та реляційними структурами (object-relational mapping, ORM) для платформи Java [23]. Hibernate є вільним програмним забезпеченням, яке поширюється на умовах GNU Lesser General Public License. Hibernate надає легкий для використання каркас (фреймворк) для відображення між об'єктно-орієнтованою моделлю даних і традиційною реляційною базою даних.

Метою Hibernate є звільнення розробника від значних типових завдань із програмування взаємодії з базою даних. Розробник може використовувати Hibernate як при розробці з нуля, так і для вже існуючої бази даних.

Hibernate піклується про зв'язок класів з таблицями бази даних (і типів даних мови програмування із типами даних SQL), і надає засоби автоматичної побудови SQL запитів й зчитування/запису даних, і може значно зменшити час розробки, який зазвичай витрачається на ручне написання типового SQL і JDBC коду. Hibernate генерує SQL виклики і звільняє розробника від ручної обробки результуючого набору даних, конвертації об'єктів і забезпечення сумісності із різними базами даних.

Hibernate забезпечує прозору підтримку збереження даних, тобто їхньої персистентності (англ. persistence) для «POJO»-об'єктів, себто для звичайних Java-об'єктів; єдина суворя вимога до класу, що зберігається — конструктор за

замовчанням (Для коректної поведінки у деяких застосуваннях потрібно приділити особливу увагу до методів `equals()` і `hashCode()`).

JPA 2 є частиною платформи Java EE 6.0. Persistence в JPA доступна в контейнерах, таких як EJB 3 або більш сучасні CDI (Java Context і Dependency Injection), а також в автономних додатках Java SE, які виконуються за межами певного контейнера. Наступні програмні інтерфейси і артефакти доступні в обох середовищах.

Mapping (зіставлення, буквально — картування) Java класів з таблицями бази даних здійснюється за допомогою конфігураційних XML файлів або Java анотацій. При використанні файлу XML, Hibernate може генерувати скелет вихідного коду для класів тривалого зберігання (`persistent`). У цьому немає необхідності, якщо використовується анотація. Hibernate може використовувати файл XML або анотації для підтримки схеми бази даних.

Забезпечуються можливості з організації відношення між класами «один-до-багатьох» і «багато-до-багатьох». На додаток до управління зв'язками між об'єктами, Hibernate також може керувати рефлексивними асоціаціями, де об'єкт має зв'язок «один-до-багатьох» з іншими примірниками свого власного типу даних.

Hibernate підтримує відображення користувацьких типів значень. Це робить можливим такі сценарії:

1. Перевизначення типу за замовчуванням SQL, який Hibernate вибирає при відображенні стовпчика властивості.
2. Картування перераховуваного типу Java до колонок БД, так ніби вони є звичайними властивостями.
3. Картування однієї властивості в декілька колонок.

Hibernate забезпечує прозоре збереження POJO (Plain Old Java Objects — простих старих об'єктів Java). Єдина сувора вимога для персистентного класу — конструктор без аргументів, не обов'язково публічний. Для правильної поведінки деяких програм також потрібна особлива увага до методів `equals()` і `hashCode()`.

Колекції об'єктів даних, як правило, зберігаються у вигляді колекцій Java-об'єктів, таких як набір (Set) і список (List). Підтримуються узагальнені класи (Generics), введені в Java 5. Hibernate може бути налаштований на «ледачі» (відкладені) завантаження колекцій. Відкладені завантаження є варіантом за замовчуванням, починаючи з Hibernate 3.

Зв'язані об'єкти можуть бути налаштовані на каскадні операції. Наприклад, батьківський клас, Album (музичний альбом), може бути налаштований на каскадне збереження і/або видалення свого нащадку Track. Це може скоротити час розробки і забезпечити цілісність. Функція перевірки зміни даних (dirty checking) дозволяє уникнути непотрібного запису дій в базу даних, виконуючи SQL оновлення тільки при зміні полів персистентних об'єктів.

### 3.6.1. Компоненти Hibernate

#### 1. EntityManagerFactory

Фабрика entity manager забезпечує екземпляри диспетчера об'єктів, всі екземпляри виконані з можливістю підключення до тієї же бази даних, щоб використовувати одні і ті ж параметри за замовчуванням, як це визначено в конкретній реалізації, і т.д. Ви можете підготувати кілька фабрик entity manager отримати доступ до кількох сховищ даних. Цей інтерфейс аналогічний theSessionFactory в рідному Hibernate.

#### 2. EntityManager

EntityManager API використовується для доступу до бази даних в конкретній одиниці роботи. Він використовується для створення та видалення стійких зразок сутностей, щоб знайти об'єкти по їх первинному ключу ідентичності, і для запиту по всім організаціям. Цей інтерфейс схожий на сесії в Hibernate.

#### 3. Persistence context

Контекст persistence являє собою набір екземплярів об'єкта, в якому для будь-якої постійної ідентичності об'єкта є унікальний екземпляр об'єкта. У persistence context, екземпляри сутностей і їх життєвий цикл управляється конкретним

менеджером сутностей. Обсяг цього контексту може бути або транзакція, або розширена одиниця роботи.

Безліч типів сутностей, які можуть управлятися за допомогою даного менеджера об'єкта визначається persistence блоком. Блок persistence визначає набір всіх класів, які пов'язані або згрупованих додатком, і які повинні бути в їх спільно розташованими відображення в одному сховищі даних.

### 3.7. Тестування

Для забезпечення цілісності роботи додатка на всіх етапах розробки, гарною практикою є використання тестування. Найкращим рішенням є JUnit.

JUnit є відкритим вихідним фреймворком розробленим з метою написання та запуску тестів в мові програмування Java. JUnit, спочатку написаний Еріх Гамма і Кент Бек, зіграв важливу роль в еволюції test-driven development (TDD), який є частиною більшого проекту програмного забезпечення парадигми, відомої як Extreme Programming (XP).

JUnit має графічний користувальницький інтерфейс (GUI), що робить можливим писати і вихідний код тесту швидко і легко. JUnit дозволяє розробнику поступово будувати тестові набори для вимірювання прогресу і виявлення ненавмисних побічних ефектів. Тести можуть працювати безперервно. Результати відразу ж надаються. JUnit показує прогрес тесту в барі, який зазвичай зелений, але стає червоним, коли тест не пройдений. Триваючий список невдалих випробувань з'являється в просторі поблизу нижньої частини вікна дисплея. Кілька тестів можуть бути запущені одночасно. Ні суб'єктивних людських суджень або інтерпретації результатів випробувань не потрібно. Простота JUnit дозволяє розробнику програмного забезпечення, щоб легко виправити помилки під час їх виявлення.

Хоча JUnit спочатку була написана для Java, продовженнях розробили для кількох інших мов програмування. Вся сім'я споріднених структур тестування називається XUnit [24].

### 3.7.1. Приклад тестування

Припустимо у нас є клас, в якому є метод, які виконує якісь дії, наприклад підсумовує числа, це і буде наша логіка, яку потрібно протестувати.

```
public class Calculate {
    public int calA (int a, int b) {
        return a + b;
    }
}
```

Як ви повинні бачити цей клас коли тестуєте?

1) Ви не знаєте, які маніпуляції виконують методи класу, ви бачите метод і знаєте що він повертає, також ви знаєте що він робить але не знаєте як, а так само ви знаєте що метод приймає на вхід.

А якщо бути точніше, то ось об'єкт тестування:

```
public int calA (int a, int b)
```

2) Ви повинні передати в цей метод всілякі дані і спробувати зробити так що б тест завалився, це головна мета тестування.

Unit тест з технічного боку - це клас який лежить в тестовому ресурсі і який призначений тільки для тестування логіки, а не для використання в production коді.

Приклад JUnit тесту:

```
@Test
public void testMultiply () {
    // Тестувальний клас
    MyClass tester = new MyClass ();
```

```
// Перевіряється метод
assertEquals ( "10 x 5 must be 50", 50, tester.multiply (10, 5));
}
```

У JUnit передбачається, що всі тестовані методи можуть бути виконані в довільному порядку. Тому тести не повинні залежати від інших тестів.

Для того щоб вказати що дані метод є тестовим його потрібно проанотувати `@Test` після чого даний метод можна буде запускати в окремому потоці для проведення тестування.

### 3.8. Синхронізація з Google сервісами

Робота з Google сервісами передбачає ряд вимог, накладених з боку постачаника. Такими є використання GoogleApiClient Google Calendar API.

#### 3.8.1. GoogleApiClient

Для використання Google сервісів на стороні серверу потрібен клієнт і Google надає API для роботи з ним. Давайте ознайомимося з ним. Це є головною точкою входу для інтеграції сервісів Google Play [25].

GoogleApiClient використовується з різними статичними методами. Деякі з цих методів вимагають, щоб GoogleApiClient бути підключені, деякі з них будуть чекати в черзі викликів, перш ніж GoogleApiClient буде підключений; перевірити відповідну документацію по API, щоб визначити, чи потрібно бути пов'язані між собою.



Рис. 3.3. Зручний доступ до Google API за допомогою Java

Бібліотека Google API-клієнт для Java надає функціональні можливості, загальні для всіх Google API, наприклад, HTTP транспорту, обробки помилок, автентифікації, JSON синтаксичного аналізу, медіа-завантаження / скачування. Бібліотека включає в себе потужну бібліотеку OAuth 2.0; легкі, ефективні моделі даних XML і JSON, які підтримують будь-яку схему даних; і підтримка для буферів протоколу.

Для виклику Google API, використовуючи клієнтські бібліотеки Google, для Java, вам потрібно згенеровану бібліотеку Java для Google API з якої ви звертаєтесь. Ці згенеровані бібліотеки включають в себе основу google-api-java-client бібліотеки разом з інформацією API-специфічною, таку як коренева URL. Вони також включають в себе класи, які мають об'єкти в контексті API, і що можуть бути використані для створення переходів між об'єктами JSON і об'єктів Java.

Ось приклад, який використовує Calendar API Client Library для Java, щоб зробити виклик Google Calendar API:

```
// Show events on user's calendar.
View.header("Show Calendars");
CalendarList feed = client.calendarList().list().execute();
View.display(feed);
```

Бібліотека включає в себе потужну бібліотеку автентифікації, яка може зменшити обсяг коду, який потрібно обробляти OAuth 2.0. Іноді кілька рядків все, що вам потрібно. Наприклад:

```
/** Authorizes the installed application to access user's protected data.
 */
private static Credential authorize() throws Exception {
    // load client secrets
    GoogleClientSecrets clientSecrets = GoogleClientSecrets.load(JSON_FACTORY,
        new InputStreamReader(CalendarSample.class.getResourceAsStream("/client_secrets.json")));
    // set up authorization code flow
    GoogleAuthorizationCodeFlow flow = new GoogleAuthorizationCodeFlow.Builder(
```



```

    httpTransport, JSON_FACTORY, clientSecrets,
    Collections.singleton(CalendarScopes.CALENDAR)).setDataStoreFactory(d
ataStoreFactory)
    .build();
    // authorize
    return new AuthorizationCodeInstalledApp(flow, new LocalServerReceiver()).aut
horize("user");
}

```

### Підтримувані середовища

Бібліотека Google API Client Library для Java підтримує ці Java-середовища:

1. Java 5 або вище, стандарт (SE) і підприємств (EE).
2. Google App Engine.
3. Android 1.5 або вище

Не підтримується: Google Web Toolkit (GWT), Java Mobile (ME), і Java 1.4 (або більш ранніх версій).

### **3.8.2. GoogleCalendarApi**

Мільйони людей використовують Google Calendar, щоб відслідковувати їх події. Calendar API дозволяє інтегрувати додаток з Google Calendar, створюючи нові шляхи для вас, щоб залучити ваших користувачів.

Наприклад, додаток для туризму може автоматично додавати маршрути в календарі користувача. Ви можете використовувати Google Calendar API, щоб знайти і переглядати публічні події календаря. Якщо ви авторизований, ви також можете отримати доступ і змінювати особисті календарі та події на цих календарів.

Використовуйте Calendar API для досягнення більш глибокої інтеграції з Google Calendar. Мобільні додатки, веб-додатки та інші системи можуть створювати, переглядати, або синхронізувати з даними календаря [26].

Calendar API є REST API, які можуть бути доступні через явні виклики HTTP або за допомогою Google Client Libraries; API-інтерфейс надає більшість функцій, доступних в веб версії Google Calendar.

### Базові концепції

Кожен користувач календаря пов'язаний з первинним календарем, а також ряд інших календарів, до яких вони також можуть отримати доступ. Користувачі можуть створювати події і запрошувати інших користувачів, як показано на рисунку 3.4:

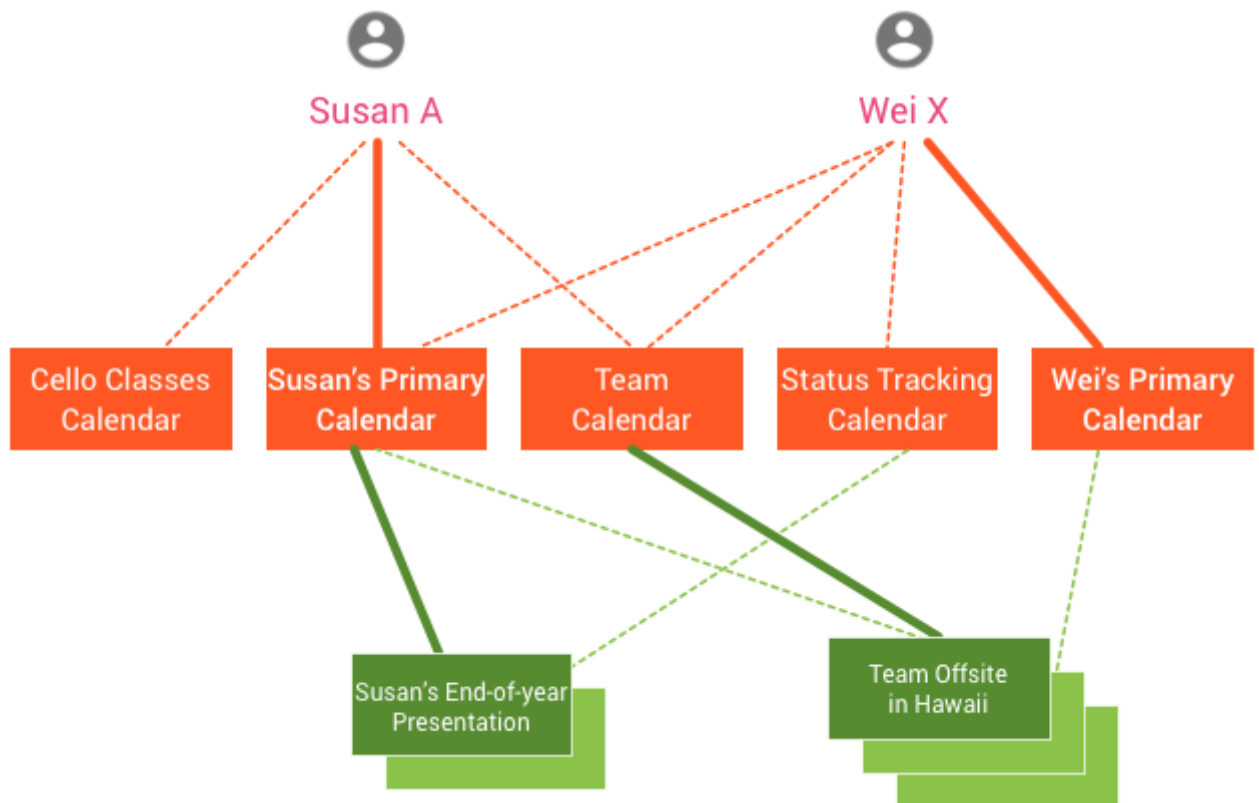


Рис. 3.4. Можливі дії користувачів за допомогою GoogleCalendarAPI

Цей приклад показує два користувача, Susan A і Wei X. Кожен з них має основний календар і кілька інших пов'язаних з ними календарів. У прикладі також показані дві події: кінець-року і команда на виїзді.

Ось деякі факти, представлені на діаграмі:

1. календар Сьюзен включає її основний календар, а також календарі для своєї команди і уроки гри на віолончелі.
2. календар Вей включає в свій основний календар, а також календар команди, стан календаря відстеження і основного календаря Сьюзен.

3. Представлення подій з вичерпаним роком показує Сьюзен як організатор і Вей в якості слухача.
4. Команда за межами майданчика на Гаваях події має календар команди в якості організатора (це означає, він був створений в цьому календарі) і скопійований Сьюзен і Вей в якості відвідувачів.

### API ресурси

Google Calendar використовує такі ресурси:

**Event** - подія на календарі, що містить таку інформацію, як назва, час початку і закінчення, і учасники. Події можуть бути як окремі події або події, що повторюються. Подія представлено ресурсом Event. Колекція подій для даного календаря містить всі ресурси для цієї події календаря.

**Calendar** - календар являє собою сукупність подій. Кожен календар має пов'язані з ним метадані, такі як календар або опис календаря за умовчанням часового поясу. Метадані для одного календаря представлені календар ресурсу. Колекція календарів містить ресурси календарів для всіх існуючих календарів.

**Calendar List** - список всіх календарів на користувача календарів в календарі UI. Метадані для одного календаря, який з'являється в календарі списку представлений CalendarListEntry ресурсу. Ці метадані включають певні користувачем властивості календаря, наприклад, його колір або повідомлень про нові події. Колекція CalendarList містить всі CalendarListEntry ресурси для даного користувача.

**Setting** - переваги (preference) користувача з Calendar UI, такі як часовий пояс користувача. Екземпляр переваги (preference) користувача представлений Setting Resource. Settings містить всі налаштування Settings ресурсів для даного користувача.

**ACL** - Правило контролю доступу надання користувачеві (або групі користувачів) певного рівня доступу до календаря. Екземпляр правила контролю

доступу (ACL ) представлений ACL ресурсом. Колекція ACL для даного календаря містить всі ACL ресурси, які надають доступ до цього календаря.

**Color** - колір, представлений в Calendar UI. Colors ресурс являє собою сукупність всіх кольорів, доступних в інтерфейсі календаря, в двох групах: кольори доступні для подій і кольорів доступних для календарів.

**Free / Busy** - Час, коли календар вже має заплановані події вважається "busy", час, коли календар не має подій вважається "free". Ресурс FreeBusy дозволяє запитувати для безлічі зайнятих часу для даного календаря або набір календарів.

### 3.9. Інтерфейс користувача

В даному розділі розглянемо основні елементи інтерфейсу розробленого прототипу, які дозволяють користувачу ефективно вирішувати задачу створення індивідуального календарного плану навчального процесу. Першим етапом роботи є автентифікація в системі, що відбувається в окремому діалоговому вікні. Користувач повинен ввести власні логін та пароль для надання доступу для роботи в інформаційній системі (рис. 3.5).

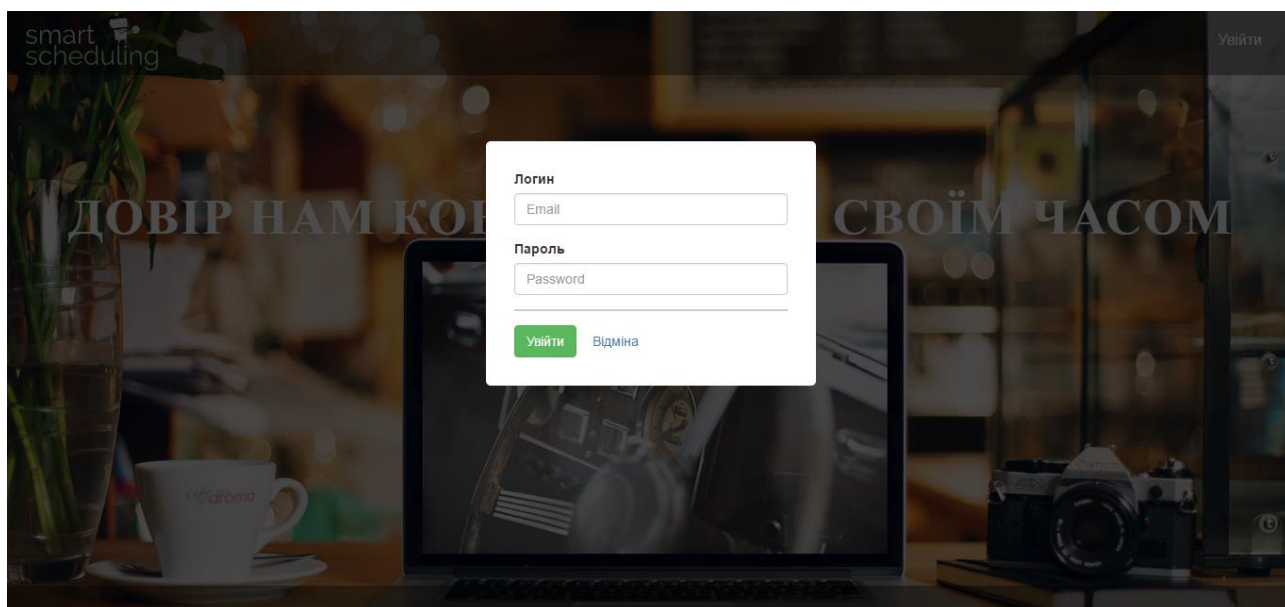
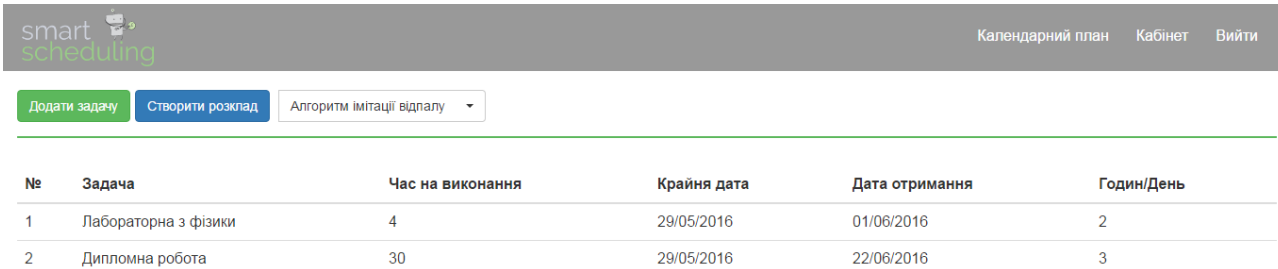


Рис. 3.5. Авторизація користувача в системі

Після процесу автентифікації користувач потрапляє у власний кабінет, де він має змогу переглядати задачі навчального плану (рис. 3.6).



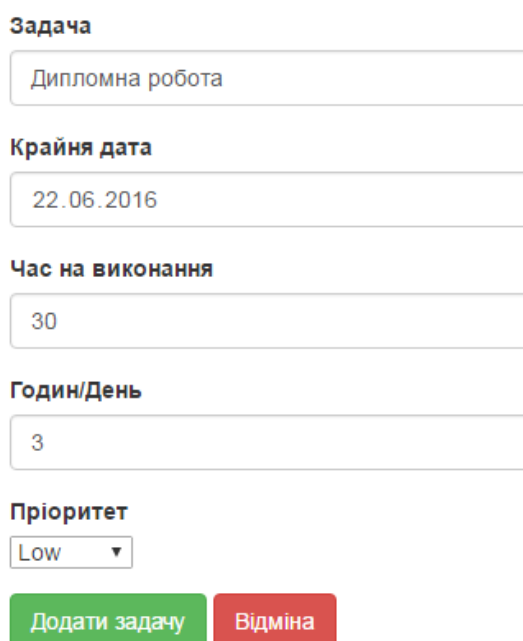
The screenshot shows the 'smart scheduling' user cabinet. At the top right, there are links for 'Календарний план', 'Кабінет', and 'Вийти'. Below the header, there are buttons for 'Додати задачу' (Add task), 'Створити розклад' (Create schedule), and a dropdown menu for 'Алгоритм імітації відпалу' (Imitation algorithm). The main content is a table with the following data:

№	Задача	Час на виконання	Крайня дата	Дата отримання	Годин/День
1	Лабораторна з фізики	4	29/05/2016	01/06/2016	2
2	Дипломна робота	30	29/05/2016	22/06/2016	3

Рис. 3.6. Кабінет користувача. Перегляд задач навчального плану.

За допомогою окремого діалогового вікна (рис. 3.7) у користувача є можливість додавати задачі до свого плану. Для цього необхідно вказати наступні дані про задачу:

- Назва задачі.
- Крайній строк задачі.
- Час на виконання даної задачі.
- Бажана кількість годин, яка буде витратиться на дану задачу.
- Пріоритет задачі.



The dialog window for adding a task contains the following fields and controls:

- Задача:** Text input field containing 'Дипломна робота'.
- Крайня дата:** Text input field containing '22.06.2016'.
- Час на виконання:** Text input field containing '30'.
- Годин/День:** Text input field containing '3'.
- Пріоритет:** Dropdown menu with 'Low' selected.
- Buttons: 'Додати задачу' (Add task) and 'Відміна' (Cancel).

Рис. 3.7. Діалогове вікно для додавання задач до календарного плану

Також в кабінеті є можливість обирати алгоритм оптимізації календарного плану.

Після натискання на кнопку “Створити розклад” користувач автоматично потрапляє на сторінку зі сформованим календарним планом (рис. 3.8), де він має можливість переглянути які задачі йому необхідно виконувати сьогодні та на який день запланована та чи інша активність. Фінальний розклад представлений у вигляді календаря з можливістю переглядати різні представлення (увесь місяць, тиждень, окремий день). Синім кольором в полі дня позначені задачі, які повинні бути виконаними в даний день. Також є змога редагувати готовий календарний план за власним бажанням (перетягувати задачі з одного дня на інший, змінювати час початку та кінця виконання задачі, змінювати тривалість задачі).

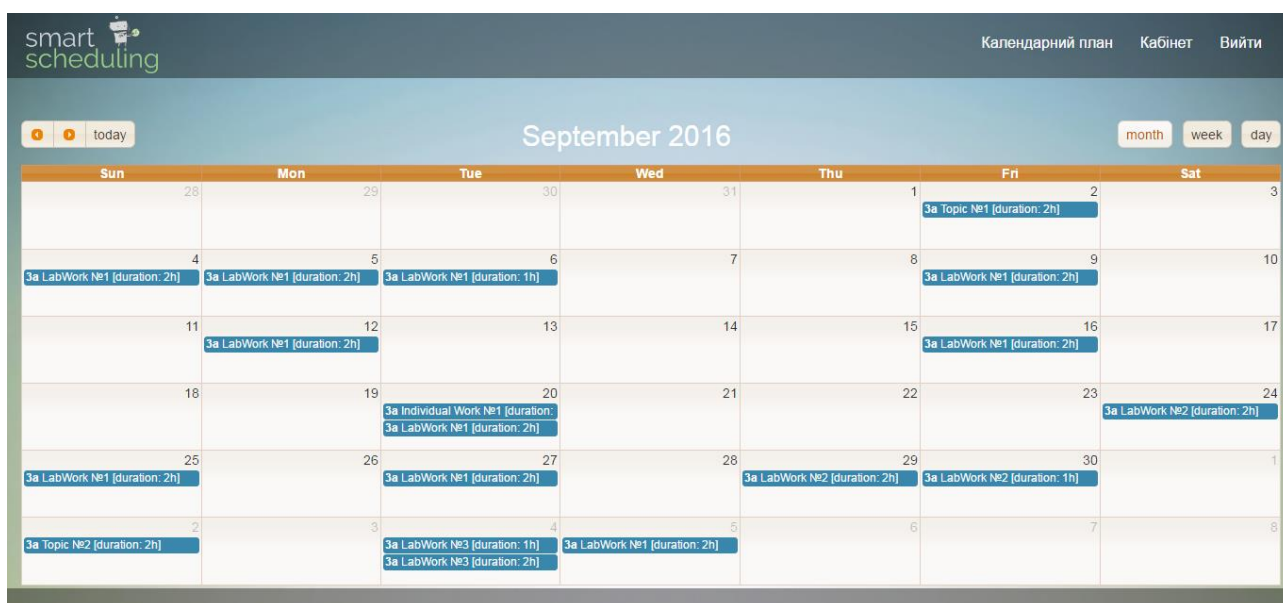


Рис. 3.8. Оптимізований індивідуальний календарний план

### 3.10. Висновки до розділу 3

В даному розділі розглянуті основні переваги обраних технологій для реалізації прототипу системи та їх порівняльна характеристика з іншими альтернативами. Для вирішення задачі довгострокового збереження даних була обрана БД PostgreSQL. Взаємодія клієнта та сервера реалізована з

використанням Spring MVC та Spring REST для реалізації API. Для зменшення зв'язаності модулів та спрощення тестування прототипу використовувався Spring IOC контейнер. Для реалізації шару сервісів, що відповідають за доступ до даних використовувався фреймворк Spring Data. Контроль якості розроблюваного додатку забезпечувався модульним тестуванням з використанням фреймворку JUnit.

## 4. РЕЗУЛЬТАТИ РОБОТИ РОЗРОБЛЮВАНОЇ ПРОГРАМИ

Результатами роботи програми є сформований календарний план. Проведено оптимізацію індивідуального план-графіку. Як бачимо на рисунку 3.8 зображено оптимізований календарний план.

Результат роботи генетичного алгоритму сильно залежить від того, яким чином налаштовані його параметри. Основними параметрами ГА є:

1. Імовірність мутації
2. Розмір популяції

Відзначимо, що вищенаведений список може бути легко розширений, але перераховані параметри присутні практично в будь-якій реалізації ГА. Різні параметри впливають на різні аспекти еволюційного пошуку, серед яких можна виділити два найбільш загальних:

1. Дослідження простору пошуку (exploration).
2. Використання знайдених «хороших» рішень (exploitation).

Перший аспект відповідає за здатність ГА до ефективного пошуку рішення і характеризує здатність алгоритму уникати локальних екстремумів. Другий аспект важливий для поступового поліпшення наявних результатів від покоління до покоління на основі вже знайдених «проміжних» рішень. Нехтування дослідними здібностями призводить до істотного збільшення часу роботи ГА і погіршення результатів через «застрявання» алгоритму в локальних екстремуму. В результаті стає можливою *передчасна збіжність* генетичного алгоритму (також кажуть про *виродження популяції*), коли рішення ще не знайдено, але в популяції практично всі особини стають однаковими і довгий час (порядку декількох десятків і сотень поколінь) не спостерігається поліпшення пристосованості.

Ігнорування знайдених рішень може привести до того, що робота ГА буде нагадувати випадковий пошук, що також негативно позначається на ефективності пошуку й про якість наданих рішень.



Основна мета в налаштуванні параметрів ГА і, одночасно, необхідна умова для стабільного отримання хороших результатів роботи алгоритму - це досягнення **балансу між дослідженням простору пошуку та використанням знайдених рішень.**

Взаємозв'язок між параметрами генетичного алгоритму, а також їх вплив на еволюційний процес має складний характер. На рис. 4.1 схематично зображено вплив зміни деяких параметрів ГА на характеристики еволюційного пошуку.

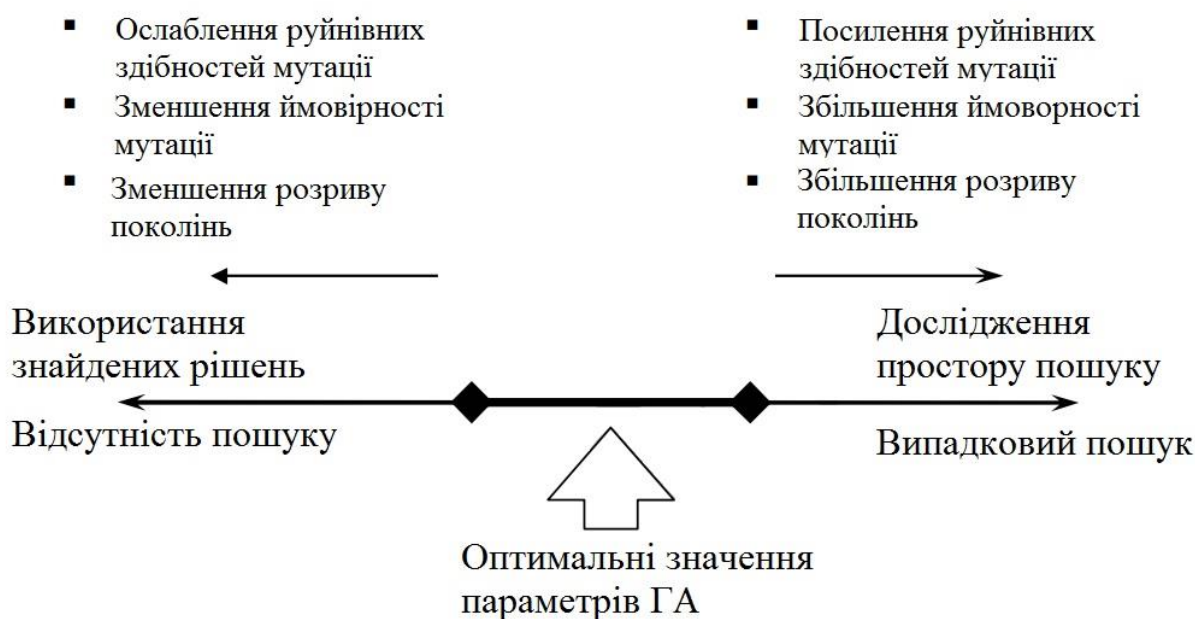


Рис. 4.1. Вплив параметрів ГА на характеристики еволюційного пошуку

Неправильне налаштування параметрів може стати причиною різних проблем в роботі ГА. Проведемо аналіз параметрів.



Рис. 4.2. Залежність значення фітнес функції від ймовірності мутації

Насамперед розглянемо залежність найкращого значення фітнес функції з популяції до ймовірності мутації (рис. 4.2). З попередніх розділів відомо, що ймовірність мутації допомагає ГА вийти з локального екстремуму. Експериментальним шляхом біло встановлено, що при збільшенні ймовірності, збільшується «випадковість» генерації значень нового покоління. Результатом даних досліджень є вибір значення ймовірності мутації в межах від 0.1 до 0.25.

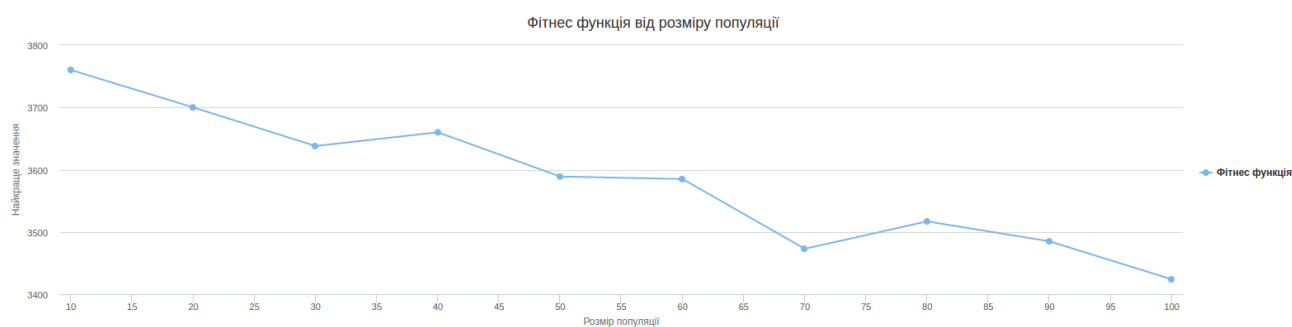


Рис. 4.3. Залежність значення фітнес функції від розміру популяції

Також розглянемо залежність найкращого значення фітнес функції від розміру популяції (рис. 4.3). Як було розглянуто, кожна популяція складається з певної кількості різних особин. При збільшенні популяції, збільшується варіативність особин, що в свою чергу збільшує шанси досягти оптимального результату. Мінусом збільшення популяції є уповільнення роботи алгоритму, тому потрібно знайти компромісне рішення. Як бачимо з рисунку 3.11 таким значенням є розмір популяції в межах від 50 до 70.

Генетичний алгоритм в результаті своєї роботи мінімізував значення фітнес функції з позначки 4825.098 до 3532.36, що складає

$$\frac{(4825.098 - 3532.36)}{4825.098} * 100\% = 27\% \quad (4.1)$$

Тобто початковий розклад було оптимізовано на 27%.

## 4.1. Порівняння з результатами роботи алгоритму імітації відпалу

Розглянемо також результати ще одного тестування, проте вже в порівнянні з результатами роботи алгоритму імітації відпалу (рис. 4.4).

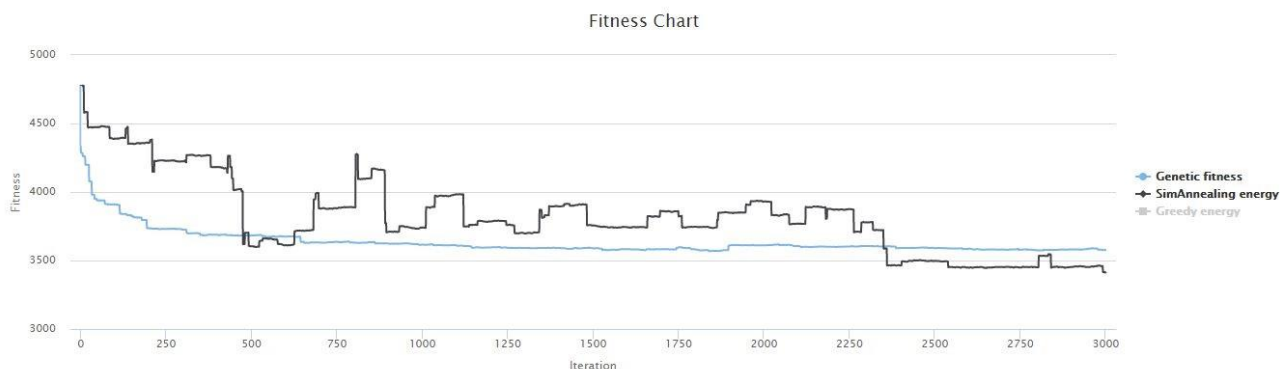


Рис. 4.4. Порівняння з результатами роботи алгоритму імітації відпалу

Алгоритм імітації відпалу в результаті своєї роботи мінімізував значення функції енергії з позначки 4825.098 до 3387.516, що складає

$$\frac{(4795.110 - 3412.758)}{4795.110} * 100\% = 29\% \quad (4.2)$$

Генетичний алгоритм в результаті своєї роботи мінімізував значення функції енергії з позначки 4825.098 до 3387.516, що складає

$$\frac{(4795.110 - 3570.550)}{4795.110} * 100\% = 26\% \quad (4.3)$$

## 4.2. Висновки до розділу 3

В даному розділі наведено порівняльну характеристику роботи алгоритмів у вигляді графіків. Для генетичного алгоритму було відображено залежність параметрів від найкращого значення функції. При збільшенні розміру популяції, зростає ймовірність досягнути оптимального рішення і досягається краще значення фітнес функції.

Також було наведено порівняльну характеристику з алгоритмом імітації відпалу. Фінальна оптимізація генетичного алгоритму менша на 3%. Можливо досягнути кращу мінімізацію збільшуючи розмір популяції, але постає інша проблема – задіяння неймовірної кількості пам'яті. Це призводить до частих запусків збирача сміття, в результаті чого просідання в швидкості видачі готового результату кінцевому користувачу.

## 5 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі проводиться оцінка основних характеристик програмного продукту, SmartScheduling. Інтерфейс користувача був розроблений за допомогою мови програмування Java у середовищі розробки IntelliJIDEA. Інтерфейс користувача створений за допомогою технології Spring.

Програмний продукт призначено для використання на персональних комп'ютерах під управлінням будь-якої операційної системи.

Нижче наведено аналіз різних варіантів реалізації модулю з метою вибору оптимальної, з огляду при цьому як на економічні фактори, так і на характеристики продукту, що впливають на продуктивність роботи і на його сумісність з апаратним забезпеченням. Для цього було використано апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) – це технологія, яка дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. Як прямі, так і побічні витрати розподіляються по продуктам та послугам у залежності від потрібних на кожному етапі виробництва обсягів ресурсів. Виконані на цих етапах дії у контексті метода ФВА називаються функціями.

Мета ФВА полягає у забезпеченні правильного розподілу ресурсів, виділених на виробництво продукції або надання послуг, на прямі та непрямі витрати. У даному випадку – аналізу функцій програмного продукту й виявлення усіх витрат на реалізацію цих функцій.

Фактично цей метод працює за таким алгоритмом:

– визначається послідовність функцій, необхідних для виробництва продукту. Спочатку – всі можливі, потім вони розподіляються по двом групам: ті, що впливають на вартість продукту і ті, що не впливають. На цьому ж етапі оптимізується сама послідовність скороченням кроків, що не впливають на цінність і відповідно витрат.

- для кожної функції визначаються повні річні витрати й кількість робочих часів.
- для кожної функції на основі оцінок попереднього пункту визначається кількісна характеристика джерел витрат.
- після того, як для кожної функції будуть визначені їх джерела витрат, проводиться кінцевий розрахунок витрат на виробництво продукту.

## **5.1. Постановка задачі**

У роботі застосовується метод ФВА для проведення техніко-економічний аналізу розробки.

Відповідно цьому варто обирати і систему показників якості програмного продукту.

Технічні вимоги до продукту наступні:

- програмний продукт повинен функціонувати на персональних комп'ютерах із стандартним набором компонент;
- забезпечувати високу швидкість обробки великих об'ємів даних у реальному часі;
- забезпечувати зручність і простоту взаємодії з користувачем або з розробником програмного забезпечення у випадку використання його як модуля;
- передбачати мінімальні витрати на впровадження програмного продукту.

### **5.1.1. Обґрунтування функцій програмного продукту**

Головна функція  $F_0$  – розробка програмного продукту, який аналізує процес за вхідними даними та будує його модель для подальшого прогнозування. Виходячи з конкретної мети, можна виділити наступні основні функції ПП:

- $F_1$  – вибір мови програмування;
- $F_2$  – вибір оптимальної СКБД;
- $F_3$  – інтерфейс користувача.

Кожна з основних функцій може мати декілька варіантів реалізації.

Функція  $F_1$ :

- а) мова програмування C#;
- б) мова програмування Java;

Функція  $F_2$ :

- а) MySQL;
- б) PostgreSQL.

Функція  $F_3$ :

- а) інтерфейс користувача, створений за технологією ASPNet;
- б) інтерфейс користувача, створений за технологією Spring.

### 5.1.2. Варіанти реалізації основних функцій

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1). На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій (таблиця 4.1).

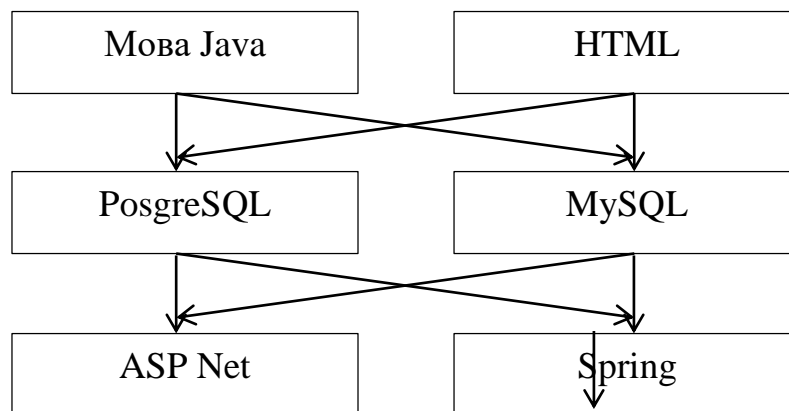


Рисунок 5.1 – Морфологічна карта

Морфологічна карта відображує всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів ПП.

Таблиця 5.1 – Позитивно-негативна матриця

Основні функції	Варіанти реалізації	Переваги	Недоліки
<i>F1</i>	<i>A</i>	Кросплатформений	Низька швидкодія
	<i>B</i>	Займає менше часу при написанні коду	Більший час на виконання операцій
<i>F2</i>	<i>A</i>	Безкоштовність	Відсутність вкладених запитів
	<i>B</i>	Надійність, внесення змін без перезапуску, безкоштовність	Необхідність додаткової інсталяції, низький рівень користувацької підтримки
<i>F3</i>	<i>A</i>	Простота створення	Відсутність кросплатформеності
	<i>B</i>	Простота створення,	Кросплатформеність

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

#### Функція *F1*:

Оскільки розрахунки проводяться з великими об'ємами вхідних даних, то час виконання програмного коду є дуже необхідним, тому варіант а) має бути відкинтий.

#### Функція *F2*:

Вибір СКБД не відіграє велику роль у даному програмному продукту, тому вважаємо варіанти а) та б) гідними розгляду.

#### Функція *F3*:

Оскільки, програмний продукт реалізується на мові Java, використовуємо варіант Б як єдиний можливий.

Таким чином, будемо розглядати такі варіанти реалізації ПП:

1. F1б – F2а – F3б
2. F1б – F2б – F3б



Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

## **5.2. Обґрунтування системи параметрів ПП**

### **5.2.1. Опис параметрів**

На підставі даних про основні функції, що повинен реалізувати програмний продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- *X1* – швидкодія мови програмування;
- *X2* – об'єм пам'яті для збереження даних;
- *X3* – час обробки даних;
- *X4* – потенційний об'єм програмного коду.

*X1*: Відображає швидкодію операцій залежно від обраної мови програмування.

*X2*: Відображає об'єм пам'яті в оперативній пам'яті персонального комп'ютера, необхідний для збереження та обробки даних під час виконання програми.

*X3*: Відображає час, який витрачається на дії.

*X4*: Показує розмір програмного коду який необхідно створити безпосередньо розробнику.

### **5.2.2. Кількісна оцінка параметрів**

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у табл. 5.2.

Таблиця 5.2 – Основні параметри ПП

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	Оп/мс	19000	11000	2000
Об'єм пам'яті для збереження даних	X2	Мб	32	16	8
Час обробки запитів користувача	X3	мс	1000	420	60
Потенційний об'єм програмного коду	X4	кількість строк коду	2000	1500	1000

За даними таблиці 5.2 будуються графічні характеристики параметрів – рис. 5.2 – рис. 5.5.

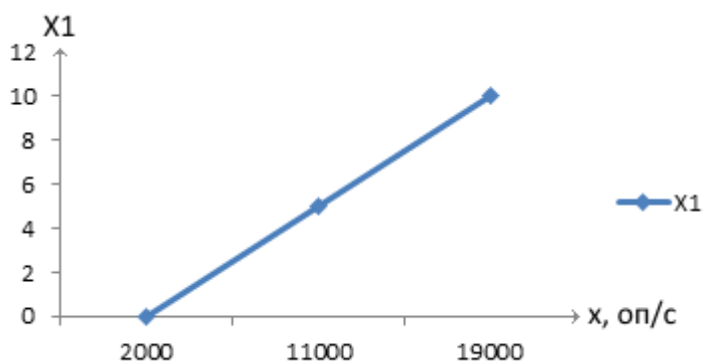


Рисунок 5.2 – X1, швидкодія мови програмування

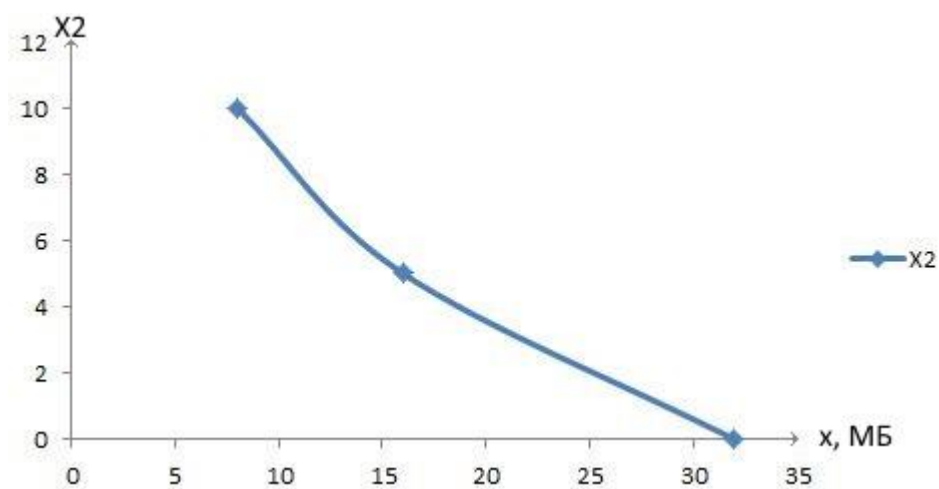


Рисунок 5.3 – X2, об'єм пам'яті для збереження даних

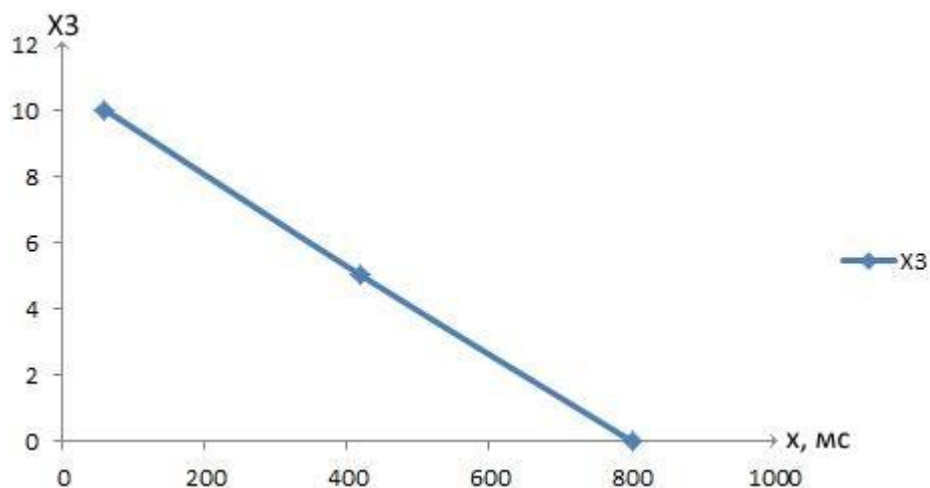


Рисунок 5.4 – X3, час виконання запитів користувача

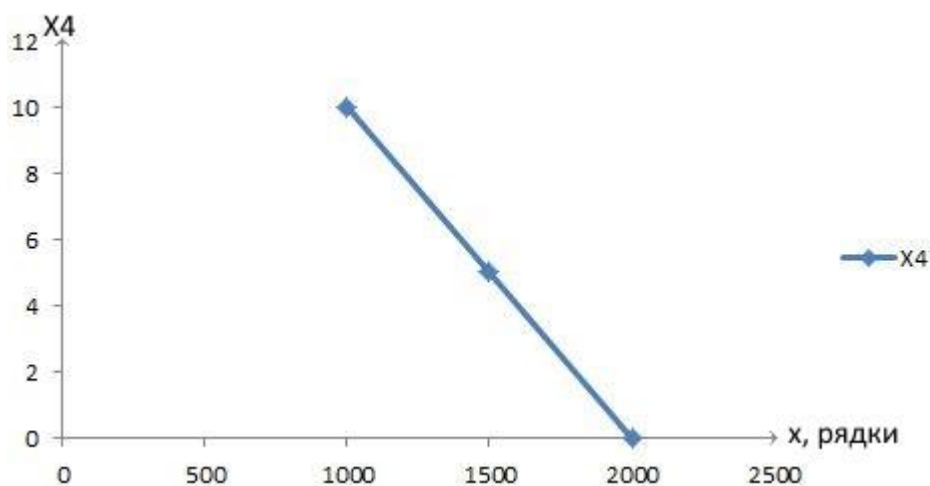


Рисунок 5.5 – X4, потенційний об'єм програмного коду

### 5.2.3. Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

– визначення рівня значимості параметра шляхом присвоєння різних рангів;

- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 5.3 – Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів $R_i$	Відхилення $\Delta_i$	$\Delta_i^2$
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	Оп/мс	4	3	4	4	4	4	4	27	0,75	0,56
X2	Об'єм пам'яті для збереження даних	Мб	4	4	4	3	4	3	3	25	-1,25	1,56
X3	Час обробки запитів користувача	Мс	2	2	1	2	1	2	2	12	-14,25	203,06
X4	Потенційний об'єм програмного коду	кількість строк коду	5	6	6	6	6	6	6	41	14,75	217,56
	Разом		15	15	15	15	15	15	15	105	0	420,75

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

- а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^N r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 105 \quad (5.1)$$

де  $N$  – число експертів;

$n$  – кількість параметрів.

- б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 26,25 \quad (5.2)$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T \quad (5.3)$$

Сума відхилень по всіх параметрах повинна дорівнювати 0;

г) загальна сума квадратів відхилення:

$$S = \sum_{i=1}^N \Delta_i^2 = 420,75 \quad (5.4)$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 420,75}{7^2(5^3 - 5)} = 1,03 > W_k = 0,67$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 5.4.

Таблиця 5.4 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	=	>	=	<	=	<	<	<	0,5
X1 і X3	<	<	<	<	<	<	<	<	0,5
X1 і X4	>	>	>	>	>	>	>	>	1,5
X2 і X3	<	<	<	<	<	<	<	<	0,5
X2 і X4	>	>	>	>	>	>	>	>	1,5
X3 і X4	>	>	>	>	>	>	>	>	1,5

Числове значення, що визначає ступінь переваги  $i$ -го параметра над  $j$ -тим,  $a_{ij}$  визначається по формулі:

$$a_{ij} = \begin{cases} 1,5 \text{ при } X_i > X_j \\ 1,0 \text{ при } X_i = X_j \\ 0,5 \text{ при } X_i < X_j \end{cases}$$

З отриманих числових оцінок переваги складемо матрицю  $A = \| a_{ij} \|$ .

Для кожного параметра зробимо розрахунок вагомості  $K_{ei}$  за наступними формулами:

$$K_{Bi} = \frac{b_i}{\sum_{i=1}^n b_i}, \text{де } b_i = \sum_{i=1}^N a_{ij}.$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятись від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{Bi} = \frac{b'_i}{\sum_{i=1}^n b'_i}, \text{де } b'_i = \sum_{i=1}^N a_{ij} b_j.$$

Як видно з таблиці 5.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 5.5 – Розрахунок вагомості параметрів

Параметри $x_i$	Параметри $x_j$				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	$b_i$	$K_{Bi}$	$b_i^1$	$K_{Bi}^1$	$b_i^2$	$K_{Bi}^2$
X1	1,0	0,5	0,5	1,5	3,5	0,219	22,25	0,216	100	0,215
X2	1,5	1,0	0,5	1,5	4,5	0,281	27,25	0,282	124,25	0,283
X3	1,5	1,5	1,0	1,5	5,5	0,344	34,25	0,347	156	0,348
X4	0,5	0,5	0,5	1,0	2,5	0,156	14,25	0,155	64,75	0,154
Всього:					16	1	98	1	445	1

### 5.3. Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів X2 (об'єм пам'яті для збереження даних) та X1 (швидкодія мови програмування) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра X3 (час обробки даних) обрано не найгіршим (не максимальним), тобто це значення відповідає або варіанту а) 1000 мс або варіанту б) 80мс.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^n K_{ei,j} B_{i,j} \quad (5.5)$$

де  $n$  – кількість параметрів;  $K_{ei}$  – коефіцієнт вагомості  $i$ -го параметра;  $B_i$  – оцінка  $i$ -го параметра в балах.

Таблиця 5.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1(X1)	A	11000	3,6	0,215	0,774
F2(X2)	A	16	3,4	0,283	0,962
F3(X3,X4)	A	800	2,4	0,348	0,835
	Б	80	1	0,154	0,154

За даними з таблиці 5.6 за формулою

$$K_K = K_{ТУ}[F_{1k}] + K_{ТУ}[F_{2k}] + \dots + K_{ТУ}[F_{zk}],$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 0,774 + 0,962 + 0,835 = 2,57$$

$$K_{K2} = 0,774 + 0,962 + 0,154 = 1,89$$

Як видно з розрахунків, кращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

#### 5.4. Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;
2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань. Загальна трудомісткість обчислюється як

$$T_0 = T_P \cdot K_{\Pi} \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М}, \quad (5.1)$$

де  $T_P$  – трудомісткість розробки ПП;  $K_{\Pi}$  – поправочний коефіцієнт;  $K_{СК}$  – коефіцієнт на складність вхідної інформації;  $K_M$  – коефіцієнт рівня мови програмування;  $K_{СТ}$  – коефіцієнт використання стандартних модулів і прикладних програм;  $K_{СТ.М}$  – коефіцієнт стандартного математичного забезпечення.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру ступеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює:  $T_P = 90$  людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання:  $K_{\Pi} = 1.7$ . Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1:  $K_{СК} = 1$ . Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта  $K_{СТ} = 0.8$ . Тоді, за формулою 5.1, загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 90 \cdot 1.7 \cdot 0.8 = 122.4 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто  $T_P = 27$  людино-днів,  $K_{\Pi} = 0.9$ ,  $K_{СК} = 1$ ,  $K_{СТ} = 0.8$ :

$$T_2 = 27 \cdot 0.9 \cdot 0.8 = 19.44 \text{ людино-днів.}$$



Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$T_I = (122.4 + 19.44 + 4.8 + 19.44) \cdot 8 = 1328,64 \text{ людино-годин};$$

$$T_{II} = (122.4 + 19.44 + 6.91 + 19.44) \cdot 8 = 1345.52 \text{ людино-годин};$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 7000 грн., один фінансовий аналітик з окладом 9500грн. Визначимо зарплату за годину за формулою:

$$C_q = \frac{M}{T_m \cdot t} \text{ грн.},$$

де  $M$  – місячний оклад працівників;  $T_m$  – кількість робочих днів тиждень;  $t$  – кількість робочих годин в день.

$$C_q = \frac{7000 + 7000 + 9500}{3 \cdot 21 \cdot 8} = 46,62 \text{ грн.}$$

Тоді, розрахуємо заробітну плату за формулою

$$C_{зп} = C_q \cdot T_i \cdot K_d,$$

де  $C_q$  – величина погодинної оплати праці програміста;  $T_i$  – трудомісткість відповідного завдання;  $K_d$  – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

$$I. \quad C_{зп} = 46,62 \cdot 1328.64 \cdot 1.2 = 74340,57 \text{ грн.}$$

$$II. \quad C_{зп} = 46,62 \cdot 1345.52 \cdot 1.2 = 75285,04 \text{ грн.}$$

Відрахування на єдиний соціальний внесок в залежності від групи професійного ризику (II клас) становить 22%:

$$I. \quad C_{вд} = C_{зп} \cdot 0.22 = 74340,57 \cdot 0.22 = 16354.93 \text{ грн.}$$

$$II. \quad C_{вд} = C_{зп} \cdot 0.22 = 75285,04 \cdot 0.22 = 16562.71 \text{ грн.}$$

Тепер визначимо витрати на оплату однієї машино-години. ( $C_M$ )

Так як одна ЕОМ обслуговує одного програміста з окладом 7000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$C_T = 12 \cdot M \cdot K_3 = 12 \cdot 7000 \cdot 0,2 = 16800 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$C_{3П} = C_{Г} \cdot (1 + K_3) = 16800 \cdot (1 + 0.2) = 20160 \text{ грн.}$$

Відрахування на єдиний соціальний внесок:

$$C_{ВІД} = C_{3П} \cdot 0.22 = 20160 \cdot 0.22 = 4435.20 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 10000 грн.

$$C_A = K_{ТМ} \cdot K_A \cdot Ц_{ПР} = 1.15 \cdot 0.25 \cdot 10000 = 2875 \text{ грн.,}$$

де  $K_{ТМ}$  – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;  $K_A$  – річна норма амортизації;  $Ц_{ПР}$  – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$C_P = K_{ТМ} \cdot Ц_{ПР} \cdot K_P = 1.15 \cdot 10000 \cdot 0.05 = 575 \text{ грн.,}$$

де  $K_P$  – відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$T_{ЕФ} = (D_K - D_B - D_C - D_P) \cdot t_3 \cdot K_B = (365 - 104 - 8 - 16) \cdot 8 \cdot 0.9 = 1706.4 \text{ годин,}$$

де  $D_K$  – календарна кількість днів у році;  $D_B$ ,  $D_C$  – відповідно кількість вихідних та святкових днів;  $D_P$  – кількість днів планових ремонтів устаткування;  $t$  – кількість робочих годин в день;  $K_B$  – коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$C_{ЕЛ} = T_{ЕФ} \cdot N_C \cdot K_3 \cdot Ц_{ЕН} = 1706,4 \cdot 0,156 \cdot 0,9733 \cdot 2,0218 = 523.83 \text{ грн.,}$$

де  $N_C$  – середньо-споживча потужність приладу;  $K_3$  – коефіцієнтом зайнятості приладу;  $Ц_{ЕН}$  – тариф за 1 кВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$C_H = Ц_{ПР} \cdot 0.67 = 10000 \cdot 0,67 = 6700 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$C_{ЕКС} = C_{3П} + C_{ВІД} + C_A + C_P + C_{ЕЛ} + C_H$$

$$C_{ЕКС} = 20160 + 4435.20 + 2875 + 575 + 523.83 + 6700 = 35269.03 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$C_{М-Г} = C_{ЕКС} / T_{ЕФ} = 35269.03 / 1706,4 = 20,67 \text{ грн/час.}$$

Оскільки в даному випадку всі роботи, які пов'язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$C_M = C_{M-G} \cdot T$$

$$I. \quad C_M = 20,67 * 1328,64 = 27462,99 \text{ грн.};$$

$$II. \quad C_M = 20,67 * 1345,52 = 27811,9 \text{ грн.};$$

Накладні витрати складають 67% від заробітної плати:

$$C_H = C_{ЗП} \cdot 0,67$$

$$I. \quad C_H = 74340,57 * 0,67 = 49808,18 \text{ грн.};$$

$$II. \quad C_H = 75285,04 * 0,67 = 50440,98 \text{ грн.};$$

Отже, вартість розробки ПП за варіантами становить:

$$C_{ПП} = C_{ЗП} + C_{Вид} + C_M + C_H$$

$$I. \quad C_{ПП} = 74340,57 + 16354,93 + 27462,99 + 49808,18 = 167966,67 \text{ грн.};$$

$$II. \quad C_{ПП} = 75285,04 + 16562,71 + 27811,9 + 50440,98 = 170100,63 \text{ грн.};$$

## 5.5. Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$K_{TEPj} = K_{Kj} / C_{Фj},$$

$$K_{TEP1} = 2,57 / 167966,67 = 0,15 \cdot 10^{-4};$$

$$K_{TEP2} = 1,89 / 183561,43 = 0,1 \cdot 10^{-4};$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня  $K_{TEP1} = 0,15 \cdot 10^{-4}$ .

## 5.6. Висновки до розділу 5

В даному розділі проведено повний функціонально-вартісний аналіз ПП, який було розроблено в рамках дипломного проекту. Процес аналізу можна умовно розділити на дві частини.

В першій з них проведено дослідження ПП з технічної точки зору: було визначено основні функції ПП та сформовано множину варіантів їх реалізації; на основі обчислених значень параметрів, а також експертних оцінок їх важливості

було обчислено коефіцієнт технічного рівня, який і дав змогу визначити оптимальну з технічної точки зору альтернативу реалізації функцій ПП.

Другу частину ФВА присвячено вибору із альтернативних варіантів реалізації найбільш економічно обґрунтованого. Порівняння запропонованих варіантів реалізації в рамках даної частини виконувалось за коефіцієнтом ефективності, для обчислення якого були обчислені такі допоміжні параметри, як трудомісткість, витрати на заробітну плату, накладні витрати.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості  $K_{\text{TEP}} = 0,15 \cdot 10^{-4}$ .

Цей варіант реалізації програмного продукту має такі параметри:

- мова програмування – Java;
- СКБД PostgreSQL;
- інтерфейс користувача, створений за технологією Spring.

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, функціонал і швидкодію.

## ВИСНОВКИ

Результатом даної роботи є аналіз особливостей та ефективності використання генетичного алгоритму для вирішення задачі створення та оптимізації календарного плану навчального процесу. Продуктом даного аналізу є порівняльна характеристика показників роботи та результатів оптимізації розкладу на один навчальний семестр.

Оскільки жодна з існуючих систем не орієнтована на врахування індивідуального розкладу користувачів при формуванні навчального плану, то було розроблено веб-орієнтований прототип такої системи та її інтеграція з сервісом Google Calendar, що використовується користувачами для управління особистим розкладом. Власне рішення включає в себе реалізацію обраного алгоритму.

В другому розділі зроблено аналіз вимог до характеристик об'єкту проектування, а саме розроблено математичної моделі. Виявлено критерії оптимізації, їх поділено на два основних типи hard stop і soft stop. Для даної задачі створено об'єктну модель, а саме представлено в вигляді моделі «сутність-зв'язок» (ER-моделі). Так, систему доцільно розділено на чотири основні сутності: студент, календарний графік, робота, подія користувача. Розглянуто функціональні аспекти системи і на основі цих даних зроблено діаграму прецедентів, що відображає дві основні ролі: студент і адміністратор. Також розглянуто структурні особливості взаємодії об'єктів, в результаті отримано діаграму послідовностей, що відображає взаємодії основних об'єктів системи, підпорядкованих за часом. Далі проведено опис алгоритму планування і детально розглянуто генетичний алгоритм і його особливості. Детально описано всі етапи алгоритму, наведено можливі варіації генетичного алгоритму.

Варіації етапів генетичного алгоритму дали можливість підібрати варіант, що не порушує наявних критеріїв. Було обрано впорядкований кросинговер (OX1). В якому частина одного з батьків береться за основу нащадку. Далі в отриманий нащадок недостаючи гени беруться з другого батька зберігаючи

порядок та пропускаючи ті, що вже є в частині від першого батька. Таким чином не порушуються жодне з наявних критеріїв. Найкращими показниками простоти реалізації і швидкодії відповідає метод мутації обміну, що змінює в послідовності двох сусідів одного випадково обраного гена.

В третьому розділі проведено аналіз інструментальної бази розробки системи. Спочатку розглянуто можливі рішення для збереження даних, для цього проведено детальний аналіз баз даних MySQL і PostgreSQL. Результатом є вибір СКБД PostgreSQL. Також розглянуто можливі архітектурні рішення для створення системи і обрано кращий шаблон проектування для створення веб ресурсів – MVC. Для наявних вимог найкращим рішенням Spring framework. Розглянуто можливості інтеграції з найпопулярнішим сервісом для планування Google Calendar.

Для генетичного алгоритму було відображено залежність параметрів від найкращого значення функції. При збільшенні розміру популяції, зростає ймовірність досягнути оптимального рішення і досягається краще значення фітнес функції.

Також було наведено порівняльну характеристику з алгоритмом імітації відпалу. Фінальна оптимізація генетичного алгоритму менша на 3%. Можливо досягнути кращу мінімізацію збільшуючи розмір популяції, але постає інша проблема – задіяння неймовірної кількості пам'яті. Це призводить до частих запусків збирача сміття, в результаті чого просідання в швидкості видачі готового результату кінцевому користувачеві.

## ПЕРЕЛІК ПОСИЛАНЬ

1. Лазарев А.А. Решение NP-трудной задачи теории расписаний минимизации суммарного запаздывания // Журнал Вычислительной математики и математической физики – 2007. том 47, N.6. – С. 1087–1098
2. Лазарев А.А., Гафаров Е.Р. Теория расписаний. Минимизация суммарного запаздывания для одного прибора. // Научное издание, М.: Вычислительный центр им. А.А. Дороницына РАН, 2006. - 134 с.
3. Лазарев А.А., Гафаров Е.Р. Теория расписаний. Исследование задач с отношениями предшествования и ресурсными ограничениями. // Научное издание, М.: Вычислительный центр им. А.А. Дороницына РАН, 2007. – 80 с.
4. Лазарев А.А., Гафаров Е.Р. Теория расписаний задачи и алгоритмы. М.: Наука, 2009.
5. Танаев В.С., Шкурба В.В. Введение в теорию расписаний. М.: Наука, 1975.
6. Танаев В.С., Гордон В.С., Шафранский Я.М. Теория расписаний. Одностадийные системы. М.: Наука. Гл. ред. физ.-мат. лит., 1984. 384 с.
7. Танаев В.С., Сотсков Ю.Н., Струевич В.А. Теория расписаний. Многостадийные системы. – М.: Наука. Гл. ред. физ.-мат. лит., 1989. – 328 с.
8. Танаев В.С., Ковалев М.Я., Шафранский Я.М. Теория расписаний. Групповые технологии. Минск: Институт технической кибернетики НАН Беларуси, 1998. 290 с.
9. Alon N., Woeginger G.J., Yadid T. Approximation schemes for scheduling on parallel machines // J. of Scheduling.– 1998.– V. 1.– P. 55 – 66.
10. Van de Akker J.M., Hoogeveen J.A., van de Velde S.L. Parallel machine scheduling by column generation // Oper. Res.– 1999.– V. 47, N 6.– P. 862 – 872.

11. Van de Akker J.M., Hurkens C.A.J., Savelsbergh M.W.P. Timeindexed formulations for single-machine scheduling problems: column generation // INFORMS J. on Computing.– 2000.– V. 12, N 2.– P. 111 – 124.
12. Baptiste Ph., Le Pape C., Nuijten W. Constraint-based scheduling: applying constraint programming to scheduling problems // Kluwer Academic Publishers, 2001.– 198 p.
13. Brooks G.N., White C.R. An algorithm for finding optimal or near – optimal solutions to the production scheduling problem // J. Ind. Eng.– 1965.– V. 16, N 1.– P. 34 – 40.
14. Brucker P. Scheduling Algorithms. Germany: Springer-Verlag 2001. 365 p.
15. Brucker P., Knust S. Complex scheduling Springer-Verlag Berlin, Heidelberg, Germany, 2006.
16. Carlier J. The one-machine sequencing problem // European J. of Oper. Res.– 1982.– V. 11, N 1.– P. 42 – 47.
17. Oracle – Hardware and Software docs. – Режим доступа: [www.oracle.com/](http://www.oracle.com/). - Дата доступа: 19.05.2016.
18. Wikipedia. – Режим доступа: [https://en.wikipedia.org/wiki/Crossover\\_\(genetic\\_algorithm\)](https://en.wikipedia.org/wiki/Crossover_(genetic_algorithm)) . - Дата доступа: 08.05.2016.
19. Nouredin Sadawi. – Режим доступа: <https://www.youtube.com/channel/UCNYv4HA3WjV3gZGLfBehRWQ>. - Дата доступа: 27.05.2016.
20. SiteMarker.Ru Интернет технологии. – Режим доступа: <http://sitemaker.ru/technologies/database/mysqlvspostgresql/>. - Дата доступа: 19.05.2016.
21. Spring Framework Overview. – Режим доступа: [http://www.tutorialspoint.com/spring/spring\\_overview.htm](http://www.tutorialspoint.com/spring/spring_overview.htm). - Дата доступа: 02.06.2016.
22. Принцип MVC в веб программировании. – Режим доступа: <http://folkprog.net/printsip-mvc-u-web-programmirovanii/>. - Дата доступа:



03.06.2016.

23. Spring security. – Режим доступа: <http://projects.spring.io/spring-security/>. -

Дата доступа: 23.04.2016.

24. Hibernate (framework). – Режим доступа:

[https://en.wikipedia.org/wiki/Hibernate\\_\(framework\)](https://en.wikipedia.org/wiki/Hibernate_(framework)). - Дата доступа:

13.10.2015.

25. Unit тестирование с JUnit. – Режим доступа: <http://devcolibri.com/864>. -

Дата доступа: 30.05.2016.

26. JUnit. – Режим доступа: <https://ru.wikipedia.org/wiki/JUnit>. - Дата доступа:

27.04.2016.

27. GoogleApiClient. – Режим доступа:

<https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient>. - Дата доступа: 14.04.2016.

28. Google Calendar API. – Режим доступа:

<https://developers.google.com/google-apps/calendar/>. - Дата доступа:

05.06.2016.

## ДОДАТОК А

### Реалізація генетичного алгоритму

```

import com.redkite.algorithm.Algorithm;
import com.redkite.algorithm.ChartDataSuit;
import com.redkite.algorithm.geneticmodel.GenSchedule;
import com.redkite.algorithm.model.Schedule;
import com.redkite.algorithm.model.Semester;
import com.redkite.algorithm.model.SubTask;
import com.redkite.entities.chart.ChartData;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Random;

public class GeneticAlgorithm implements Algorithm, ChartDataSuit {
    private List<GenSchedule> population;
    private final double crossoverSize = 0.5;
    private final Integer populationSize = 20;
    private Integer iteration = 0;
    private final Integer maxIterations = 3000;
    private final double mutationProbability = 0.5;
    private final Random random = new Random();
    private GenSchedule initial;
    private ChartData fitnessAndIterationData = new ChartData("Genetic fitness");

    @Override
    public Schedule doCalculation(Semester semester, List<SubTask> subTasks) {
        createInitPopulation(semester, subTasks);
        if (initial == null) {
            initial = getBest();
        }

        for (int i = 0; i < maxIterations; i++) {
            iteration = i;
            double value = i == 0 ? initial.getOptValue() : getBest().getOptValue();
            fitnessAndIterationData.getData().add(new Number[]{iteration, value});
            System.out.printf("Curr iteration %d, best value = %f \n", i, value);

            List<GenSchedule> parents = chooseParents();
            //
            for (int j = 0; j < parents.size(); j += 2) {
                GenSchedule p1 = parents.get(j);
                GenSchedule p2 = parents.get((j + 1) % parents.size());
            }
        }
    }
}

```

```

        population.add((GenSchedule) p1.doCrossover(p2, (int)
(p2.getDaysWithTasks().size() * crossoverSize)));
    }
    population = selectNewGeneration();
    if (random.nextDouble() <= mutationProbability) {
        GenSchedule schedule = population.get(random.nextInt(population.size()));
        schedule.doMutation();
    }

}
return population.get(0);
}

@Override
public double getInitialOptimizedValue() {
    return initial.getOptValue();
}

@Override
public double getFinalOptimizedValue() {
    return getBest().getOptValue();
}

@Override
public void setInitialSchedule(Schedule initialSchedule) {
    initial = new GenSchedule(initialSchedule);
}

private GenSchedule getBest() {
    Collections.sort(population, (o1, o2) -> (int) (o2.getFitness() - o1.getFitness()));
    return population.get(0);
}

private List<GenSchedule> chooseParents() {
    int size = population.size();
    int parentsNumber = size % 2 == 0 ? size / 2 : (size - 1) / 2;
    List<GenSchedule> parents = new ArrayList<>();
    double sum = population.stream().mapToDouble(GenSchedule::getFitness).sum();
    for (int i = 0; i < parentsNumber; i++) {
        int r = random.nextInt((int) sum);
        double partSum = 0;
        int chosenChr = 0;
        for (int j = 0; j < size; j++) {
            partSum += population.get(j).getFitness();

```

```

        if (partSum > r) {
            chosenChr = j;
            break;
        }
    }
    parents.add(population.get(chosenChr));
}
return parents;
}

```

```

private List<GenSchedule> selectNewGeneration() {
    int size = population.size();
    // sort ascending
    Collections.sort(population, (o1, o2) -> (int) (o1.getFitness() - o2.getFitness()));
    return population.subList(size / 2, size);
}

```

```

private void createInitPopulation(Semester semester, List<SubTask> subTasks) {
    population = new ArrayList<>(populationSize);
    int size;
    if (initial != null) {
        size = populationSize - 1;
        population.add(initial);
    } else {
        size = populationSize;
    }
}

```

```

for (int i = 0; i < size; i++) {
    GenSchedule schedule = new GenSchedule(semester, subTasks);
    population.add(schedule);
}

```

```

}

```

```

public ChartData getFitnessAndIterationData() {
    return fitnessAndIterationData;
}

```

```

}

```