Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра математики

Кваліфікаційна робота

освітній ступінь – бакалавр

на тему: **«Розробка чат боту для задачі планування»**

Виконав: студент 4-го року навчання,
Освітньої програми «Прикладна математика», 113

Статейко Артем Олександрович

Керівник: Козеренко С. О.
кандидат фіз.-мат. наук, ст. викладач

Рецензент: _____
(прізвище та ініціали)

Кваліфікаційна робота захищена
з оцінкою _____

Секретар ЕК _____
«____» _____ 20____ р.

Київ – 2024

Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра математики

ЗАТВЕРДЖУЮ

Зав. кафедри математики,

Доцент, кандидат ф.-м. наук

Руслан Костянтинович Чорней

_____

“____” _____2023

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на кваліфікаційну роботу

студенту 4-го курсу, факультету інформатики

Статейко Артем Олександрович

Тема: Розробка чат боту для задачі планування.

Зміст ТЧ до кваліфікаційної роботи:

1. Анотація
2. Вступ
3. Огляд існуючих технологій
4. Розробка чат-боту для збору даних
5. Теорія та реалізація генетичного алгоритму

6. Розробка фінальної програми та тестування
7. Висновок

Дата видачі “____” _____ 2024 р. Керівник:_____

Завдання отримано:_____

Календарний план виконання кваліфікаційної роботи

Тема: Розробка чат боту для задачі планування.

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	17.11.2023	
2.	Огляд технічної літератури за темою роботи	11.2023-12.2023	
	Розробка та запуск користувацького інтерфейсу з заглушкою	01.2024	
	Аналіз отриманих даних та розробка алгоритму оптимізації	02.2024-03.2024	
3.	Написання текстової частини	03.2024-05.2024	
4.	Внесення змін до кваліфікаційної роботи відповідно до зауважень наукового керівника	11.2023-05.2024	
5.	Попередній захист роботи	15.05.2024	
6.	Проходження перевірки на плагіат	27.05.2024	
7.	Захист роботи	03.06.2024	

Студент Статейко А.О. _____

Керівник Козеренко С. О. _____

“_____”_____ 2024 р.

ЗМІСТ

стор.

Анотація

Кваліфікаційна робота присвячена розробці алгоритму для створення розкладів. Для збору тестових даних та, внаслідок визначення задачі, був використаний ChatGPT3.5. Так само він буде використовуватися в кінцевому проекті замість користувацького інтерфейсу. В результаті було використано модифікований генетичний алгоритм для оптимізації алгоритму пошуку оптимального розкладу. Програмування усіх алгоритмів відбувалося за допомогою мови програмування Python та бібліотек spaCy та spaCy-llm.

Вступ

В сучасному світі, де час є одним із найдорожчих ресурсів, завжди постає питання оптимізації використання цього ресурсу. І не важливо в якій сфері ви працюєте, оптимізація розкладів може допомогти не лише краще використовувати час, а й щоб розвантажитись від втоми та стресу безпосередньо пов'язаного з продуктивністю та якістю життя. Наукові дослідження, такі як роботи Джонсона та Пінедо, полегшують шлях до створення алгоритмів з урахуванням часу обмеження, вимог щодо перерв або інших факторів-показників запобігання імплантації.

Дуже важливим аспектом оптимізації графіків є врахування людського фактору під час планування. Наприклад, у дослідженні Бейкера і Трича «Принципи послідовності та планування» наголошується на необхідності визначення пріоритетів завдань і ефективного управління робочим навантаженням для підвищення загальної продуктивності при зниженні рівня стресу. Важливість оптимізації планування для окремих людей полягає в здатності алгоритмів враховувати унікальні потреби та риси кожного користувача, роблячи їх повсякденне життя більш гармонійним і продуктивним. Тому дослідження в області розробки алгоритмів оптимізації розкладу є актуальним і важливим завданням, яке може значною мірою підвищити ефективність управління часом, а також загальний добробут людей.

Дана кваліфікаційна робота мала на меті дослідження відомих алгоритмів оптимізації розкладів та розробка власного алгоритму, адаптованого під конкретно наші данні та умови.

Для реалізації мети роботи постали наступні задачі: проаналізувати літературу по заданій тематиці та дослідити вже наявні алгоритми оптимізації разом з їх варіаціями, знайти алгоритм, котрий можна буде використати для оптимізації, та реалізувати цей алгоритм в чат-боті.

1. Огляд існуючих технологій

1.1 Чат-боти та вилучення даних

Історія чат-ботів бере початок ще у 60-х роках минулого сторіччя. Тоді Джозеф Вайценбаум розробив першого чат-бота під назвою ELIZA. Її мета була з'імітувати розмову психотерапевта, тож вона була однією з перших програм, котру ми могли б назвати чат-ботом.

Ще одним дуже критичним моментом для історії чат-ботів стало перше проходження тесту Тьюрінга чат-ботом PARRY, написаним психіатором Кеннетом Колбі. Цей тест полягав в тому, що людина, котра окремо поспілкується з однією іншою людиною та чат-ботом, не маючи інформації з ким саме вона поспілкувалася не зможе чітко відповісти хто з них чат-бот.

Поява інтернету, покращення обчислювального обладнання та збільшення потужностей в 1990-х-2000-х дозволило розробляти більш комплексні та складні чат-боти. Одним з таких є A.L.I.C.E (Artificial Linguistic Internet Computer Entity), заснований на AIML (Artificial Intelligence Markup Language). І чат-бота, і мову розмітки розробив Річард Уоллес в 1995 році, що принесло йому 3 премії Лобнера, котрі видають найбільш розвинутому штучному інтелекту.

Впровадження машинного навчання та обробки природної мови (NLP) започаткувало сучасну еру чат-ботів. У 2010-х роках такі платформи, як WhatsApp, Slack та Facebook Messenger, надали API для створення чат-ботів, що значно спростило розробку та використання цих ботів. Для цього періоду часу однією з ключових подій в цій галузі стала поява OpenAI GPT (Generative Pre-trained Transformer), яка має високу здатність розуміти і

генерувати людську мову. Нові можливості для створення більш інтелектуальних та адаптивних чат-ботів були надані GPT-3 та GPT-4

Основними задачами обробки природної мови є:

1. Токенизація. Поділ тексту на окремі токени чи слова
2. Лемантизація та стеммінг. Приведення слова до базової форми
3. POC розмітка. Визначення частини мови для слова
4. Розпізнавання іменованих сутностей. Ідентифікація класифікації сутності (ім’я, дата, тощо)
5. Аналіз синтаксичної структури

На сьогодні чат-ботів можна розбити на декілька груп за їх характеристиками. Основними серед таких є:

- Складність. Прості чат-боти працюють мають обмежений функціонал та працюють за заздалегідь визначеними правилами. Складні використовують ШІ для подальшого навчання на основі взаємодій з користувачем
- Метод взаємодії. Голосовий чи текстовий
- Призначення. Розважальні, інформаційні тощо.

В нашому випадку ми будемо використовувати OpenAI GPT-4. Це нова потужна модель, котра генерує тексти, майже не відрізнити від людських. Крім того для більш детального аналізу повідомлень користувача я буду використовувати бібліотеки spaCy та spaCy-llm. Вона надає основні інструменти для виконання основних завдань NLP та дозволяє розробляти високоефективні системи для обробки та аналізу тексту.

1.2 Методи оптимізації розкладів

Оптимізація розкладів завжди є важливою задачею в будь-якій галузі. Її основна ціль – розподіл якихось ресурсів таким чином, щоб максимізувати ефективність чогось чи досягти якихось певних умов.

Традиційні методи програмування:

- Лінійне прогламування (LP): цей метод використовується для оптимізації завдань з лінійними обмеженнями та лінійною цільовою функцією. Коли відносини між змінними може бути виражені лінійними рівняннями чи нерівностями, лінійне програмування особливо корисно. Симплекс-метод є популярним алгоритмом на вирішення завдань лінійного програмування.
- Цілочисельне програмування (IP): Цей метод використовує лише цілі змінні, що розширює лінійне програмування. При складанні розкладів для людей або машин, наприклад, використовується цілочисельне програмування.
- Комбінаторні алгоритми. Методи, які досліджують усі можливі варіанти для ухвалення кращого рішення, належать до цієї категорії. Прикладом є метод гілок та кордонів, який використовується для вирішення задач комбінаторної оптимізації, коли всі можливі рішення систематично перебираються, а потім відсіваються свідомо невигідні гілки.

Метаевристичні методи:

- Генетичні алгоритми. Метод оптимізації, в основі якого лежать генетичні оператори (природний відбі, мутація тощо). Генетичні алгоритми добре справляються зі складними завданнями оптимізації,

коли традиційні методи можуть виявитися неефективними. Вони дозволяють уникати локальних рішень та досліджувати широкий спектр рішень.

- Метод відпалу. Цей метод ґрунтується на аналогії з процесом відпалу в металургії, де поступове охолодження металу призводить до досягнення стабільного стану з мінімальною енергією. Метод імітації відпалу використовує випадкове блукання простором рішень з ймовірністю прийняття гірших рішень на ранніх етапах, що дозволяє уникати потрапляння в локальні мінімуми.
- Мурашиний алгоритм. Метод, натхненням для створення якого стала поведінка мурах при пошуку шляху до джерела їжі. Алгоритми мурах використовують колекцію штучних мурах для прийняття рішень. Ці алгоритми використовують можливість вибору шляхів на основі феромонів, які залишають інші мурахи. Метод ACO успішно використовується для складання розкладів та маршрутизації.

Серед усіх методів генетичний алгоритм найкраще підходить під умови нашої задачу. Він часто показує себе набагато кращі результати, ніж інші еврістичні методи-побратими, і він має набагато менше обмежень для застосування ніж нееврістичні методи.

2. Збір даних за допомогою чатботу

2.1 Визначення задачі чат-боту

Однією з перших і найважливіших завдань розробки будь-якої моделі оптимізації є чітке формулювання проблеми. Ми не мали чіткого уявлення про те, які дані будуть використовуватися, а також за якими параметрами ми будемо оптимізовувати наш алгоритм. Щоб усунути цю невизначеність, було ухвалено рішення запустити Telegram-бота, який взаємодіяв би з користувачами та створював враження, що він уже здатний керувати розкладом.

Тобто його основною метою був збір повідомлень, котрі користувачі відправляли б вже повністю робочому застосунку. Для цього був використаний Telegram API, що дозволило досить швидко розгорнути бота та під'єднати його до GPT-4.

Для обробки повідомлень користувачів та відповіді були використані OpenAI API и библиотека spaCy.

Після завершення розробки, ми одразу запустили цього телеграм-бота та попросили інших людей почати «використовувати» його. І одразу на початковому етапі це допомогло визначити декілька типів повідомлень, котрі цікавлять нас і котрі ми будемо опрацьовувати за допомогою NLP:

- Повідомлення в яких користувач просить додати/змінити/прибрати якусь подію/події в свій розклад. (Сценарій модифікації)
- Повідомлення яких він просить надати інформацію з вже створеного розкладу. (Сценарій надання інформації)

Таким чином цей метод дозволив велику кількість реальних даних, які потім аналізувалися, щоб визначити основні вимоги та шаблони.

2.2 Налаштування токенізації

Ми вже зробили перший крок в налаштуванні токенізації повідомлень користувача, коли розподілили їх на два (три, ще повідомлення, котрі не стосуються розкладів) типи сценарієв.

Перший тип в свою чергу складається з підкатегорій, котрі розрізняються типом події/подій, котрі користувач хоче додати в розклад:

- Подія із жорсткими обмеженнями часу
  Приклад: «Завтра з сьомої до дев'ятої години ранку в мене онлайн-мітінг з тім-лідом»
- Повторювана подія з жорсткими обмеженнями часу
  Приклад: «Я працюю з 9 до 18 кожен понеділок, вівторок, середу та суботу»
- Подія-дедлайн, до котрої треба виконати якісь задачі.
  Приклад: «Мені треба ще три рази сходити в той спортивний зал за цей місяць, щоб до кінця витратити свій абонемент. Я займаюсь по три години за раз»
- Повторювана подія/події.
  Приклад: «Я хочу кожен тиждень приділяти 2 години на прибирання своєї кімнати»
- Відміна події/подій
  Приклад: «Мені більше не треба приділяти час на прибирання»

Таким чином ми можемо побачити, що в кожної події можуть бути різні атрибути в залежності від її типа:

- Для подій з жорсткими обмеженнями часу в нас завжди будуть мати або час початку та тривалість, або час початку та кінця події

- Для події з дедлайном в нас будуть завжди час самого дедлайну та кількість роботи, котру треба виконати до нього, та можливо час, коли користувач отримує можливість виконувати роботу з цієї події. Сама робота може бути подана користувачем в різних форматах:
    - час витрачений на одну сесію роботи та кількість таких сесій, котрі ми маємо розподілити в розкладі до дедлайну, як в прикладі зверху
    - кількість часу, котру треба приділити якійсь події до дедлайну, з обмеженнями по мінімальному та максимальному часу витрачену на один сеанс безперервної роботи.
      Наприклад: «Мені треба десь 40 годин щоб розібратися з даними до зустрічі з інвесторами дев'ятого серпня. Я можу займатися цим від 2 до 4 безперервно»

Другий тип в основному складається з типових запитів щодо розкладу, які можна поділити на прямі запити інформації щодо розкладу на день, тиждень тощо.

Тепер, коли ми визначили різні типи повідомлень і зрозуміло що саме нам потрібно, ми можемо визначимо яким чином ми будемо розрізняти повідомлення різних типів. Повідомлення другого типу відрізняється від першого типу тим, що зазвичай містять в собі дієслово наказового відмінку, котре буде синонімом слова «покажи», або саме повідомлення є питанням, котре ми можемо визначити за наявністю відповідного знака чи прислівника часу. Тож ми одразу визначаємо дві категорії слів, котрі буде шукати алгоритм в тексті: синоніми слова покажи в наказовому відмінку («show_order») та прислівники часу чи місця («question_when»).

Перший тип повідомлень в свою чергу не має таких чітких категорій слів, котрі могли б вказати, що це саме перший тип, тому поки з двома категоріями шуканої інформації ми покладаємося на визначення типу повідомлення лише на наявність в вихідному масиві сутностей категорій «show_order» та «question_when».

Далі, ми дивимося в масив сутностей на наявність інших entity, необхідність яких визначається вже відомим типом повідомлення. Для другого типу ми можемо легко сформулювати категорії для цих сутностей:

- «timestamp» - точка в часі
- «time_period» - проміжок часу

Нам буде необхідна хоча б одна сутність з цих двох категорій, бо ці сутності будуть позначати інформацію за який саме день чи період хоче користувач. Якщо їх буде більше, то ми будемо вважати, що користувач хоче отримати інформацію по всьому переліченому та категоризованому як «timestamp» чи «time_period»

Для першого типу в нас знову не все так просто, бо реакцією на таке повідомлення буде не просте виведення даних для користувача, а додавання події з цього повідомлення в розклад, а тому ми хочемо одразу продумувати яким чином ми будемо збирати та зберігати дані. Також в першому типі в нас є інші підкатегорій, котрі відрізняються між собою деякими потребуючими змінними. Але почнемо ми з необхідних категорій, котрі є збіжними:

- «task_name» - назва завдання, котрим хоче/має зайнятися користувач Сутність такої категорії необхідна для всіх підкатегорій
- «duration» - тривалість цього завдання

Може бути відсутня для подій з жорсткими умовами. Має бути одна чи три для подій-дедлайнів

- «repeat_flag» - слова «кожен», «кожний»
  Необхідний для повторюваних подій. Його наявність визначає ці підкатегорії
- «time_point» - точка в часі
  Може бути відсутня лише в випадку повторюваної події, якщо є, то позначає день/дні тижня для якого буде повторюватися подія. Може бути дві і більше для подій з жорсткими умовами, якщо немає «duration»
- «repeat_period» - період за який повторюються події
  Необхідний для повторюваних подій
- «num_b_dl» - кількість сесій роботи до дедлайну/в період часу
  Необхідний для події-дедлайн та повторюваної події

Таким чином, ми отримаємо опис кожної події, котрий співпадає тільки з певною підкатегорію, бо «repeat_flag» точно визначає чи це повторювана подія чи ні. А далі, якщо вона не повторювана, то ми перевіряємо існування «num_b_dl», котрий може бути тільки в події-дедлайн, і на кількість наявних «duration» та «time_point», бо жорсткі умови завжди матимуть хоча б один «time_point» та або другий (2,0), або «duration» (1,1), а подія-дедлайн може бути складена лише з «num_b_dl», «duration» та «time_point» як мінімум.

Для відокремлення повторюваних подій ми скористаємося тією самою відмінністю в можливих категоріях суб’єктів. Повторювана подія з жорсткими умовами не буде мати «num_b_dl».

Наприкінці аналізу ми змогли прийти до деяких основних висновків:

- Більшість користувачів, котрі користувалися ботом, висловлювали ідею того, що оптимальність розкладу в основному для них залежить від розподілу навантажень на якийсь період часу, тобто ми будемо мінімізовувати квадратичне відхилення напруження на один період часу.
- Коли йдеться про оптимізацію розкладу, щоб встигнути на всі заходи чи виконати певну кількість завдань до дедлайну, часто виникають запити на оптимізацію розкладу, що тільки наголошує на необхідності ефективних алгоритмів планування та оптимізації
- Також певні користувачі звернули увагу на додатковому опрацюванні їх особистих переваг, таких як бажання чи не бажання працювати в певні години

Ці результати дозволили визначити конкретні вимоги до системи оптимізації розкладів та визначити важливі елементи, які слід враховувати під час розробки генетичних алгоритмів.

3. Теорія та реалізація генетичного алгоритму

3.1 Постановка задачі

Повертаючись до наших даних, якщо ми візьмемо лише поки події з жорсткими обмеженнями часу повторювані чи ні, покладемо їх на шкалу часу та додамо додаткові події-обмеження на те, що наша модель не має можливості правильно оптимізовувати, таке як сон чи прийом їжі, то ми отримаємо шкалу часу на які в нас є вільні від подій з жорсткими умовами періоди часу.

Також в нас залишаються повторювані події та події з дедлайном, котрі ми маємо зробити за ці вільні періоди часу. Їх ми зводимо до одного виду: повторювані події ми перетворюємо в велику кількість подій з початком та дедлайном. Таким чином ми отримуємо щось схоже на проблему наповнення кількох рюкзаків, суть якої набрати в рюкзаки, які мають обмеження по вазі, предметів на найбільшу вартість.

В нас є періоди часу, котрі виконують роль рюкзаку, та події в ролі предметів, але в нашому випадку ми хочемо максимально рівно розподілити наші події, щоб зменшити різницю між навантаженнями на кожен день. Для цього ми робимо матрицю x_i_j розмірністю n*m, де n – це кількість вільних періодів, m – кількість різних подій, котрі ми маємо розподілити. І визначаємо проблему мінімізації, де я використовую квадрат для мінімізації розкиду

$$min \sum_{i=1}^{n} \sum_{j=1}^{m} (x_{ij} * w_j - c_i * \frac{\sum_{j_1=1}^{m} a_{j_1} * w_{j_1}}{\sum_{i_1=1}^{n} c_{i_1}})^2$$

Обмеження:

Період часу обмежений і ми не можемо за нього виходити

$$\sum_{j=1}^{m} x_{ij} * w_j \ < c_i, i = 1, 2, \dots n$$

Для кожної події будуть обмеження в які періоди часу їх можна розподілити

$$\sum_{i=1}^{s_j - 1} x_{ij} \ + \sum_{i=d_j}^{n} x_{ij} \leq 0, j = 1, 2, \dots m$$

Кожна подія має пройти стільки разів скільки потрібно

$$\sum_{i=1}^{n} x_{ij} = a_j, j = 1, 2, \dots m$$

Де:

x_{ij} – це показник матриці, котрий може приймати цілі значення від 0 до a_j. Він показує скільки подій j буде в періоді і

a_j – список значень, котрі відповідають скілько всього подія має пройти

w_j – список значень довжин подій

c_i – список ємностей періодів часу

s_j – список порядкових номерів періодів часу з якого можна починати проводити певні події

d_j- список порядкових номерів періодів часу, котрі є дедлайном для певнох події

3.2 Вступ до генетичних алгоритмів

Генетичний алгоритм – це один із методів оптимізації задач, який використовує механізми біологічної еволюції для послідовного підбору, комбінування та зміни параметрів для вирішення задач оптимізації та моделювання, а тому більшість означень, котрі використовуються для опису генетичних алгоритмів, запозичені з генетики. «Батьком-засновником» генетичних алгоритмів вважається Джон Холланд. Його книга «Адаптація в природних і штучних системах» є однією з найважливіших робіт в цій галузі досліджень.

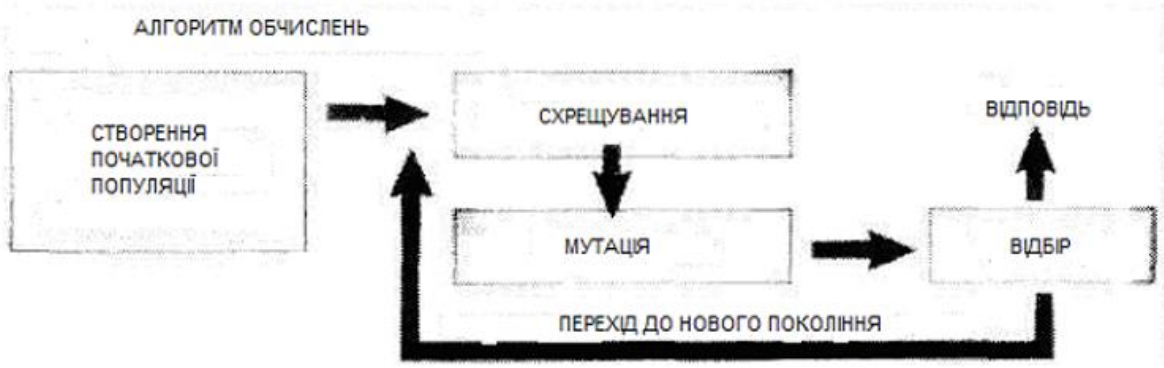
Основні поняття в генетичних алгоритмах:

- Популяція – множина особин
- Хромосома– послідовність генів
- Ген – елемент генотипу, зокрема хромосоми
- Генотип – набір хромосом
- Життєвий цикл популяції – циклічне створення нової популяції за допомогою генетичних операторів (зазвичай це селкція, схрещування та мутації)
- Схрещування – генетичний оператор, котрий злучає генетичну інформацію батьків, щоб отримати нову особу - дитину
- Мутація – генетичний оператор, котрий випадково змінює наші гени, щоб підтримувати генетичне різноманіття
- Селекція – процес відбору батьків з поточної популяції

Для того, що почати моделювати еволюційний процес ми спочатку створюємо початкову популяцію і запускаємо життєвий цикл популяцій, де

ми за допомоги схрещення та випадкових мутацій створюємо нову популяцію на заміну старій. Повний процес виглядає якось так:

1. Ініціалізація початкової популяції
2. Оцінка популяції за допомогою функції допасованості
3. Відбір індивідів з популяції (селекція)
4. Застосування генетичних операторів (схрещення та/або мутація) для оновлення популяції
5. Оцінка популяції
6. Якщо критерії зупинки алгоритму не виконаний, то повертаємося на крок 3. Критерієм зупинки може бути кількість поколінь чи необхідне значення оціночної функції



3.3 Реалізація нашої версії генетичного алгоритму

В нашому випадку хромосома буде мати вигляд матриці розмірності n*m, а генам будуть відповідати самі події, точніше одновимірний масив (g[0..n]), котрий показує скільки раз подія відбулася в певному періоді.

Першим кроком до створення генетичного алгоритму зазвичай є функція створення першої популяції (pop_init), але наразі ми подивимося на іншу функцію (time_relief), котра буде використовуватися в деяких функціях далі.

Ця функція буде використовуватися в випадку невдачної мутації чи схрещення, коли один з періодів стає переповненим. Ми дивимося на всі події котрі вже є в цьому періоді, а потім дивимося для кожної події куди ми можемо перемістити її, щоб не переповнити інший. Далі ми випадково обираємо подію і куди вона переходить, якщо період все ще заповнений, повторюємо дії.

```
def time_relief():
    not_feasible = True
    while not_feasible:
        not_feasible = False
        for i=1 to n:
            if c[i] < sum(x[i, j]*w[j] for j in range(m))
                not_feasible = True
                potential_switch = []
                for j=1 to m:
                    for l = s[j]-1 to d[j]:
                        if c[l] > sum(x[l, j]*w[j] for j in range(m)) + w[j]
```

```
                                    then potential_switch.append(l, j)
                        if potential_switch in not empty:
                                ind = rand(0, potential_switch.length)
                                x[i, potential_switch[1, ind]] -=1
                                x[potential_switch[2, ind], potential_switch[1, ind]] +=1
                        else print(«All filled»)
```

Тепер, коли ми доробили цю допоміжну функцію, ми можемо приступати до генерування першої популяції (pop_init). В нашому випадку ми просто випадково розкидуємо наші події по будь-яким періодам доступним цим подіям, а потім визиваємо функцію time_relief для перевірки та покращення розкладу.

```
def pop_init ()
population = []
    for i=1 to popsize:
        x = [n x m]
        for j=1 to m:
            count = 0
            while count < a[j]
                x[s[j] + rand(0, d[j] – s[j]), j] +=1
        time_relief(x)
        population.append(x)
```

Наступним кроком буде створення алгоритму схрещення (Crossover). Через те, що в нас один ген має вигляд строки з цілими числами, ми не можемо користуватися більш традиційними методами кроссоверу, такими як one-point чи two-point crossover, але ми можемо і самі визначити спосіб

схрещення. В цьому випадку я б хотів використати uniform crossover, котрий полягає в тому, що кожен ген дитини обирається випадково з двох генів батьків. Також батьків для схрещення я з початку буду обирати за допомогою дуелей між випадково обраними хромосомами, і при кожному схрещенні буде ймовіність, що батьки незмінно пройдуть цей етап і залишаться далі в популяції (crossover rate).

```
def duel(population):
    a = population[rand()]
    b = population[rand()]
    if fitness_function(a) > fitness_function(b)
            then return a;
            else return b;


def uni_cross(parent1, parent2, cross_rate)
    child1 = []
    child2 = []
    if rand() < cross_rate
    then
            for j=1 to m
                    if rand() > 0.5 :
                            for i=1 to n:
                            child1[i, j] = parent1[i,j]
                            child2[i, j] = parent2[i,j]
```

```
                else
                        for i=1 to n:
                        child1[i, j] = parent2[i,j]
                        child2[i, j] = parent1[i,j]
        else
                child1 = parent1
                child2 = parent2
        time_relief(child1)
        time_relief(child2)
```

Останнім кроком перед можливо фінальною оцінкою нашої популяції є мутація. Мутувати в нас можуть усі гени, якщо це дозволяють умови.

```
def mutate(mutation_rate):
    for j=0 to m:
            ind = rand(0, a[j])
            count = 0
            i = s[j]
            while ind > count:
                    count += x[i, j]
                    i += 1
            potential_switch = []
            for i=s[j] to d[j]
                    if c[i] > sum(x[i, j]*w[j] for j in range(m)) + w[j]
                            then potential_switch.append(l, j)
```

```
if potential_switch in not empty:
        ind = rand(0, potential_switch.length)
        x[i, potential_switch[1, ind]] -=1
        x[potential_switch[2, ind], potential_switch[1, ind]] +=1
```

Таким чином ми отримаємо нову популяцію, котра далі знову піде по тому самому шляху, якщо вона не є останньою популяцією.

4. Розробка фінальної програми та тестування

4.1 Фінальна програма

У фінальній програмі перед нами постала задача поєднати генетичний алгоритм, котрий по суті є бек-ендом, з вже працюючим чат-ботом. Для цього ми використовуємо ті іменовані сутності, котрі ми знаходили в повідомлення за допомогою NLP. Таким чином ми отримаємо дані, котрі при деякій обробці можуть бути використані нашим алгоритмом для складення розкладу.

Фінальний додаток складається з декількох основних частин, кожна з яких виконує певну функцію:

- Conversation handler:
  Інтерфейс користувача: Telegram-бот служить основним інтерфейсом для взаємодії з користувачами, дозволяючи їм вводити дані та отримувати результати оптимізації.
  Обробка повідомлень: бот аналізує текстові повідомлення за допомогою OpenAI та spaCy, витягує ключові дані та визначає наміри користувачів.
- Серверна частина:
  Збір та зберігання інформації: Дані, надані користувачами, такі як поточні розклади, переваги та обмеження, приймаються, обробляються та зберігаються на серверній частині.
  Генетичний алгоритм: отримані дані використовуються для оптимізації розкладів модулем генетичного алгоритму.
  Логування: для подальшого аналізу та покращення системи ведення логів взаємодій користувачів з ботом та виконаних операцій.

4.2 Оцінка роботи чат-боту та ідеї для подальших покращень алгоритму

Для оцінки працездатності нашого чат-боту, я провів тестування серед друзів і родичів, котрі погодилися на це. Для цього я два тижні збирав фідбек та слідкував за часом виконання генетичного алгоритму для кожного з тестерів.

Тут показан один із розкладів на тиждень складених для мого друга. Ми бачимо, що кожен день в нього починається з 10, на кожному дні є події із жорсткими умовами (час прийому їжі), а майже весь інший час в нього регулюється чат-ботом. І майже в кожному вільному проміжку часу в нього є якась подія, що доказує, що алгоритм в нас працює і рівномірно розподіляє задачі по періодам часу.

Звісно в нашому алгоритмі не все так ідеально. Особливо коли розклад починає заповнюватися занадто сильно на якомусь періоді часу. Через те, що функція кроссоверу не гарантує те, що дитина буде підходити під умови задачі.

Також користувачі (чи їх дані) вже підкинули декілька ідей по оптимізації мого алгоритму:

- Зміна одновимірної цільової функції на багатовимірну
  Переглядаючи розклади своїх друзів я не міг не помітити, що іноді, оптимальним рішенням розкладу стає не найлогічніше рішення виконувати якісь маленькі дії по декілька разів за раз. Для вирішення цієї проблеми можна додати до кожного заняття додаткові ваги, щоб покращити оцінку розкладів з різноманітними завданнями в один період часу (наприклад двічі повчитися по дві години буде сприйматися

гірше ніж повчитися дві години та три години поприбирати) ти змінити цільову функцію, щоб вона враховувала ці ваги.

- Зняття умови дедлайну

  Поки що при використанні нашого алгоритму оптимізації в нас є умова, що дедлайн якоїсь задачі може бути лише на кінці якогось періоду часу. Я б хотів прибрати це обмеження, бо це приводить до того, що алгоритм не розглядає деяке поле рішень
- Перехід від задач з чітко виставленим часом виконання до задач з плаваючим

Висновок

Ця кваліфікаційна робота мала на меті розробку чат-боту для оптимізації Дана кваліфікаційна робота була присвячена пошуку алгоритмів спрямованих на оптимізацію розкладів. Під час цього було проаналізовано літературу, котра стосується цієї теми, та обрано генетичний алгоритм, котрий ми реалізували. З цією метою була визначена задача оптимізації та її характеристики (обмеження).

Результатом виконання роботи став телеграм бот, котрий буде допомогати у складанні розкладів. Програмування боту та алгоритму було здійснено на мові програмування Python.

Список літераури

1. Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included.
2. Pinedo, M. (1995). Scheduling: Theory, Algorithms, and Systems.
3. Baker, K. R., Trietsch, D. (2009). Principles of Sequencing and Scheduling.
4. John H. Holland (1992). Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence
5. Poli, R., Langdon, W. B., McPhee, N. F. (2008). A Field Guide to Genetic Programming
6. Kononyuk, A. (2008) Genetic algorithms
7. Huang, M. H., Rust, R. T. (2018). Artificial Intelligence in Service. Journal of Service Research
8. Dorigo, M., Stützle, T. (2004). Ant Colony Optimization. MIT Press
9. Shawar, B. A., Atwell, E. (2007). Chatbots: Are they Really Useful? LDV Forum
10. Haupt, R. L., Haupt, S. E. (2004). Practical Genetic Algorithms. Wiley.
11. Telegram API documentation. Telegram
12. OpenAI API documentation. OpenAI