Національний університет

«Полтавська політехніка імені Юрія Кондратюка»

(повне найменування вищого навчального закладу)

Навчально-науковий інститут інформаційних технологій та робототехніки

(повна назва факультету)

Кафедра комп'ютерних та інформаційних технологій і систем

(повна назва кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістра

(освітньо-кваліфікаційний рівень)

на тему

Побудова застосунку для розрахування оптимального передаточного співвідношення для одношвидкісних велосипедів з використанням нечіткої логіки

Виконав: студент 6 курсу, групи 601-ТН

спеціальності 122 Комп'ютерні науки

(шифр і назва напряму)

Пташенко Т.В.

(прізвище та ініціали)

Керівник д.т.н., проф. Краснобаєв В.А.

(прізвище та ініціали)

Полтава – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ « ПОЛТАВСЬКА ПОЛІТЕХНІКА ІМЕНІ ЮРІЯ КОНДРАТЮКА»

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА РОБОТОТЕХНІКИ

КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА
спеціальність 122«Комп'ютерні науки» на тему
«Побудова застосунку для розрахування оптимального передаточного співвідношення для одношвидкісних велосипедів з використанням нечіткої логіки»

Студента групи 601-ТН Пташенка Тимура Вадимовича

Керівник роботи д.т.н., проф. Краснобаєв В.А.

Консультант
к. ф.-м. н. Двірна О.А.

Завідувач кафедри комп'ютерних та інформаційних технологій і систем
к.ф.-м.н. Двірна О.А.

Полтава – 2024

РЕФЕРАТ

Кваліфікаційна робота магістра: 64 сторінки, 10 рисунків, 4 таблиць, 2 додатки, 24 джерела.

Об’єкт дослідження: процес досліждення заходів для визначення оптимального передаточного співвідношення для одношвидкісних велосипедів та велосипедів з фіксованою передачею.

Предмет дослідження: розробка, реалізація та тестування програмного забезпечення для визначення оптимального передаточного співвідношення для одношвидкісних велосипедів та велосипедів з фіксованою передачею, яке враховує фізичні параметри, характеристики маршруту (рельєфу, покриття) та технічні особливості велосипеда.

Мета роботи: побудова застосунку, що дозволяє визначити оптимальне передаточне співвідношення для одношвидкісних велосипедів та велосипедів з фіксованою передачею шляхом аналізу індивідуальних даних користувача з використанням нечіткої логіки.

Методи: вивчення факторів, що впливають на вибір передатного числа, вибір основних, що впливають на його розрахунок, аналіз методів розрахунку передатного числа, аналіз існуючих додатків – калькуляторів, що використовуються в любительських вело-спільнотах та велосипедному спорті, визначення необхідних додаткових параметрів для оптимального розрахунку, вивчення можливості використання методів нечіткої логіки для створення застосунку.

Ключові слова: передаточне число, співвідношення, оптимізація, нечіткі множини, параметри, умови маршруту, функції належності, індивідуальні дані, оптимізація правил, аналіз GPS, аналіз GPX, аналіз TCX

ABSTRACT

Master's Qualification Work: 64 pages, 10 figures, 4 tables, 2 appendices, 24 references.

Object of study: the process of researching measures for determining the optimal gear ratio for single-speed bicycles and fixed-gear bicycles.

Subject of study: development, implementation, and testing of software for determining the optimal gear ratio for single-speed and fixed-gear bicycles, considering physical parameters, route characteristics (terrain, surface), and technical features of the bicycle.

Purpose of the work: development of an application that determines the optimal gear ratio for single-speed and fixed-gear bicycles by analyzing user-specific data using fuzzy logic.

Methods: study of factors influencing gear ratio selection; identification of key parameters affecting its calculation; analysis of existing methods for gear ratio calculation; review of existing calculators used in amateur cycling communities and professional cycling; determination of additional parameters required for optimal calculations; exploration of the feasibility of using fuzzy logic methods to create the application.

Keywords: gear ratio, optimization, fuzzy sets, parameters, route conditions, membership functions, user-specific data, rules optimization, GPS analysis, GPX analysis, TCX analysis.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БЗ – база знань.

БД – база даних.

БНЗ – база нечітких знань.

ЕС– експертна система.

НЛП – нечіткі логічні правила.

ПО – предметна область.

ІНЛ – інструментарій нечіткої логіки.

FLS – Fuzzy Logic System – система нечіткої логіки.

NFS – Neuro Fuzzy System – нейро-фаззі система.

NN – Neural Network – нейронна мережа

ФН – функція належності

Fixed-gear – велосипед з фіксованою передачею.

Single-speed – одношвидкісний велосипед.

ЗМІСТ

ВСТУП

Сучасна молодіжна культура відіграє важливу роль у формуванні трендів, серед яких особливе місце посідають одношвидкісні велосипеди та велосипеди з фіксованою передачею. Їх популярність значною мірою зумовлена стильним і мінімалістичним дизайном, що ідеально відповідає естетиці сучасного міського життя. Молодь, яка прагне підкреслити свою індивідуальність, усе частіше обирає ці велосипеди як елемент модного способу життя. Вони стали символом свободи, незалежності та самовираження, адже лаконічний вигляд, можливість кастомізації та асоціація з активним стилем роблять їх невід'ємною частиною урбаністичної культури. Окрім модного аспекту, ці велосипеди є практичним і екологічним вибором. Завдяки простоті конструкції та відсутності складних механізмів вони надійні, потребують мінімального обслуговування й дозволяють зменшити витрати на експлуатацію. Також вони сприяють зниженню впливу на довкілля, оскільки не продукують викидів, на відміну від автомобілів чи інших моторизованих засобів пересування. Це робить їх привабливими для тих, хто дбає про екологію та прагне робити свідомий вибір на користь природи.

Об'єктом дослідження є процес дослідження заходів для визначення оптимального передаточного співвідношення для одношвидкісних велосипедів та велосипедів з фіксованою передачею. Предметом дослідження є механізми і методи визначення такого співвідношення на основі аналізу даних про маршрут. Дослідження вирішує завдання розробки моделі нечіткої логіки, що враховує ключові параметри маршруту, такі як рельєф місцевості, тип дорожнього покриття, середня швидкість та частота обертання педалей. На основі цієї моделі створюється веб-додаток, який інтегрує ці дані, застосовуючи алгоритми нечіткої логіки, для визначення оптимального передаточного числа. Практична значимість роботи полягає у створенні інструменту, який дозволить велосипедистам отримувати персоналізовані рекомендації щодо налаштувань передаточного числа для максимального комфорту і ефективності поїздок. Результати дослідження сприятимуть не лише підвищенню зручності

використання таких велосипедів, але й популяризації їх як стильного та екологічно свідомого засобу пересування. У процесі роботи передбачається аналіз існуючих підходів, обґрунтування вибору технологій, розробка архітектури додатку та його тестування в реальних умовах. Це забезпечить ефективність розроблених методик і їх відповідність потребам користувачів.

Це питання висвітлювалось серед матеріалів Міжнародної науково-практичної конференції: Ptashchenko T., Dvirna O. Development of an application for selecting gear ratios for single-speed bicycles using fuzzy logic and artificial intelligence. Digital Economy and IT: Trends and Perspectives 2024: Матеріали Міжнародної науково-практичної конференції, 28-29 листопада 2024 року, Полтава: Національний університет імені Юрія Кондратюка, 2024. С. 91-94.

РОЗДІЛ 1.
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ЗАДАЧІ

1.1 Основні поняття та опис чинників

В останні роки спостерігається стрімке зростання популярності велосипедів з фіксованою передачею, що стало однією з яскравих рис сучасної молодіжної культури [2]. Свою історію ці велосипеди беруть з трекових перегонів, де вони використовувались ще з кінця XIX століття. Їх конструкція була ідеальною для змагань на велотреках завдяки простоті, надійності та відсутності зайвих деталей, таких як гальма чи перемикачі передач, що дозволяло досягати максимальної ефективності руху. З часом велосипеди з фіксованою передачею вийшли за межі велотреків, їх мінімалізм та спортивне коріння привернули увагу вело курьєрів, міських велосипедистів, любителів адреналіну, зробивши їх символом сучасного урбаністичного стилю [2].

Fixed-gear велосипеди — це одношвидкісні велосипеди, які не мають механізму вільного ходу, тобто педалі постійно обертаються разом із заднім колесом. Завдяки цьому оберти педалей безпосередньо передаються на заднє колесо. Ця проста конструкція, що позбавлена складної багатошвидкісної трансмісії, робить fixed-gear велосипеди надійними та менш вразливими до поломок, забезпечує максимально можливий ККД, а також знижує їх вагу, що приваблює багатьох велосипедистів [1].

Також такі велосипеди, як правило, можуть бути переобладнані в Single-speed велосипеди, або, просто кажучи, велосипеди з однією передачею та вільним ходом колеса, що забезпечує можливість обертання заднього колеса без руху педалей, що є більш комфортним варіантом для новачків але не забезпечує унікальних відчуттів від їзди, повної прив’язоності до руху. Вони стали популярними в урбанізованих середовищах, особливо серед молоді, адже як правило за відносно невелику ціну дозволяють поєднати практичність і стильність у пересуванні містом.

Хоча початково такі велосипеди призначались для професійного велоспорту, зараз вони часто використовуються для прогулянкових поїздок, розваг, поїздок на роботу. В особливості вони стали популярні серед велокур'єрів через повний контроль над кожним обертом колеса що допомогає в керуванні велосипедом в щільному трафіку, а також простий вигляд та конструкцію які дозволяють без переживань залишити велосипед під дощем в не самому благоприйнятному районі міста [2].

Рисунок 1.1 – Загальна будова велосипеду

Незважаючи на очевидні переваги, використання fixed-gear велосипедів має свої обмеження, які впливають на комфорт і безпеку їх експлуатації. Головна проблема полягає у відсутності можливості змінювати передатне число залежно від умов дороги — таких як нахил, тип і стан поверхні, погодні умови. Це робить fixed-gear велосипеди менш зручними для пересування в умовах різноманітного рельєфу.

Передаточне число — це відношення кількості зубців на передній зірці до кількості зубців на задній зірці велосипеда. Це значення визначає, скільки

оборотів робить заднє колесо за один повний оберт педалей. Наприклад, якщо передня зірка має 48 зубців, а задня — 16, то передаточне число буде 48:16, що дорівнює 3. Це означає, що заднє колесо робить 3 оберти за кожен оберт педалей. Чим вище передаточне число, тим більше обертів робить заднє колесо на один оберт педалей, але тим менший крутний момент при застосуванні однакової сили. Оскільки людина що використовує велосипед може активно змінювати прикладаєму до педалей силу під час їзди це характерно відрізняє велосипеди від інших видів транспорту, у яких би при схожих обставинах просто заглох би двигун, й дає можливість мати відносно широкий діапазон швидкості та поверхонь для пересування маючи лише одну механічну “швидкість”, але у цього усеодно є свої обмеження.

Передаточне число має велике значення при виборі передачі залежно від типу покриття та рельєфу місцевості. Наприклад, для швидкісної їзди на рівних дорогах або під час спусків вибирають високе передаточне число, яке дозволяє досягати великих швидкостей, оскільки заднє колесо робить більше обертів на один оберт педалей. Це передаточне число оптимальне для велошляхів, де не потрібно часто змінювати швидкість або сила педалювання повинна бути спрямована на підтримку швидкості.

Велосипедист, який використовує фіксовану передачу, часто стикається з проблемою надмірного фізичного навантаження на підйомах та високої швидкості педалювання на спусках, що може призвести до втрати контролю над велосипедом та зниження ефективності педалювання. Водночас подолання цих недоліків стає не лише фізичним викликом, а й своєрідною філософією, яка дозволяє велосипедисту краще розуміти свій потенціал, знаходити гармонію між технікою та тілом, а також приймати обмеження як частину досвіду, що формує витривалість і дисципліну [4].

Розглянемо таблицю передатних чисел з найбільш поширеними варіантами передач.

Final Drive Ratios Highlighted between 1.7:1 and 2.1:1

	11	12	13	14	15	16	17	18	19	20	21	22	23
20	1.818	1.667	1.538	1.429	1.333	1.250	1.176	1.111	1.053	1.000	0.952	0.909	0.870
22	2.000	1.833	1.692	1.571	1.467	1.375	1.294	1.222	1.158	1.100	1.048	1.000	0.957
24	2.182	2.000	1.846	1.714	1.600	1.500	1.412	1.333	1.263	1.200	1.143	1.091	1.043
26	2.364	2.167	2.000	1.857	1.733	1.625	1.529	1.444	1.368	1.300	1.238	1.182	1.130
28	2.545	2.333	2.154	2.000	1.867	1.750	1.647	1.556	1.474	1.400	1.333	1.273	1.217
30	2.727	2.500	2.308	2.143	2.000	1.875	1.765	1.667	1.579	1.500	1.429	1.364	1.304
31	2.818	2.583	2.385	2.214	2.067	1.938	1.824	1.722	1.632	1.550	1.476	1.409	1.348
32	2.909	2.667	2.462	2.286	2.133	2.000	1.882	1.778	1.684	1.600	1.524	1.455	1.391
33	3.000	2.750	2.538	2.357	2.200	2.063	1.941	1.833	1.737	1.650	1.571	1.500	1.435
34	3.091	2.833	2.615	2.429	2.267	2.125	2.000	1.889	1.789	1.700	1.619	1.545	1.478
35	3.182	2.917	2.692	2.500	2.333	2.188	2.059	1.944	1.842	1.750	1.667	1.591	1.522
36	3.273	3.000	2.769	2.571	2.400	2.250	2.118	2.000	1.895	1.800	1.714	1.636	1.565
38	3.455	3.167	2.923	2.714	2.533	2.375	2.235	2.111	2.000	1.900	1.810	1.727	1.652
39	3.545	3.250	3.000	2.786	2.600	2.438	2.294	2.167	2.053	1.950	1.857	1.773	1.696
40	3.636	3.333	3.077	2.857	2.667	2.500	2.353	2.222	2.105	2.000	1.905	1.818	1.739
42	3.818	3.500	3.231	3.000	2.800	2.625	2.471	2.333	2.211	2.100	2.000	1.909	1.826
44	4.000	3.667	3.385	3.143	2.933	2.750	2.588	2.444	2.316	2.200	2.095	2.000	1.913
46	4.182	3.833	3.538	3.286	3.067	2.875	2.706	2.556	2.421	2.300	2.190	2.091	2.000
47	4.273	3.917	3.615	3.357	3.133	2.938	2.765	2.611	2.474	2.350	2.238	2.136	2.043
48	4.364	4.000	3.692	3.429	3.200	3.000	2.824	2.667	2.526	2.400	2.286	2.182	2.087
49	4.455	4.083	3.769	3.500	3.267	3.063	2.882	2.722	2.579	2.450	2.333	2.227	2.130
50	4.545	4.167	3.846	3.571	3.333	3.125	2.941	2.778	2.632	2.500	2.381	2.273	2.174
52	4.727	4.333	4.000	3.714	3.467	3.250	3.059	2.889	2.737	2.600	2.476	2.364	2.261
53	4.818	4.417	4.077	3.786	3.533	3.313	3.118	2.944	2.789	2.650	2.524	2.409	2.304
54	4.909	4.500	4.154	3.857	3.600	3.375	3.176	3.000	2.842	2.700	2.571	2.455	2.348
56	5.091	4.667	4.308	4.000	3.733	3.500	3.294	3.111	2.947	2.800	2.667	2.545	2.435

Рисунок 1.2 – Співвідношення передатних чисел

Врахувати вплив всих чинників та вибрати з можливих варіантів найоптимальніший досить складно, особливо новачкам.

Актуальність дослідження проблеми оптимального передатного числа пояснюється тим, що кожен велосипедист має свої індивідуальні особливості: рівень фізичної підготовки, стиль їзди, параметри маршруту, який використовується, та інші фактори, що впливають на зручність їзди. Так, у місцях із значною кількістю підйомів та спусків використання єдиного фіксованого передатного числа може створювати значні труднощі. На крутих підйомах фіксована передача може виявитися надто важкою, що призведе до швидкої втоми та ризику травм через перенапруження м’язів. На спусках, навпаки, велосипедисту доводиться підтримувати дуже високу частоту педалювання, щоб зберігати контроль над велосипедом, що також може

викликати дискомфорт і перевантаження.

Вибір оптимального передатного числа стає особливо важливим у контексті різних поверхонь, по яких їздить велосипедист. Наприклад, на асфальтованих дорогах фіксоване передатне число може бути оптимально налаштоване для забезпечення високої швидкості. Проте, якщо велосипедист рухається по гравію чи нерівній місцевості, потрібно враховувати більші навантаження на колеса та можливі труднощі в управлінні. Це вимагає від велосипедиста регулярної адаптації своїх налаштувань та належної оцінки умов.

1.2 Загальна характеристика проблеми та актуальність дослідження

У зв’язку з цим виникає потреба у створенні додатку, що допомагатиме велосипедистам, особливо початківцям, обирати оптимальне передатне співвідношення для fixed-gear велосипедів. Такий додаток дозволив би значно спростити процес вибору передатного числа, мінімізуючи необхідність у великій кількості тестових поїздок та витрат на заміну компонентів велосипеда. Він стане корисним інструментом, що враховуватиме індивідуальні параметри кожного користувача, такі як вага, середня швидкість, фізична форма, тип поверхні та рельєф маршруту.

Додаток також може стати незамінним для спортсменів, які тренуються на велотреку. В умовах треку, де кожен грам ваги та кожна секунда мають вирішальне значення, точний підбір передатного числа є критичним для досягнення високих результатів [3]. Аналізуючи дані про середню швидкість, довжину треку та фізичну підготовку спортсмена, додаток може рекомендувати оптимальну конфігурацію для забезпечення максимальної ефективності педалювання.

Розробка додатку, що враховуватиме всі ці фактори, є актуальною і перспективною задачею. Алгоритми можуть аналізувати дані про маршрути (таких як нахили, довжина підйомів і спусків) та порівнювати їх із фізичними характеристиками велосипедиста, такими як вага, стиль їзди, рівень фізичної

підготовки тощо. Це дозволить створити персоналізовані рекомендації, які враховують не лише загальні показники, але й специфіку кожного індивідуума.

Крім того, додаток може враховувати вимоги велотреку, де педалювання має бути максимально плавним, а швидкість — стабільною. Завдяки цьому професійні трекові велосипедисти зможуть точно налаштовувати велосипеди для тренувань і змагань, оптимізуючи співвідношення швидкості та фізичних зусиль [3].

Велосипедист може вводити дані про маршрут перед початком поїздки, і на основі отриманих параметрів система зможе розрахувати оптимальне передатне число та надати рекомендації щодо зміни передатного числа та стилю їзди під час руху.

Також важливо зазначити, що такий додаток стане корисним не лише для початківців, але й для досвідчених велосипедистів, які зможуть оцінити, наскільки ефективним є обране співвідношення для їхніх маршрутів, а також адаптувати його у випадку зміни умов їзди або фізичної підготовки.

Загалом, розробка додатку для визначення оптимального передатного числа для fixed-gear та single-speed велосипедів є обґрунтованою і відповідає сучасним запитам як любителів, так і професійних велосипедистів. Цей додаток сприятиме покращенню ефективності та комфорту їзди, та підвищенню задоволення від використання фіксованої передачі.

РОЗДІЛ 2. АНАЛІЗ ФАКТОРІВ ВПЛИВУ ТА МЕТОДІВ РОЗРАХУНКУ ПЕРЕДАТНОГО ЧИСЛА

2.1 Фактори, що впливають на вибір передаточного числа

Процес вибору оптимального передатного числа є складним і багатогранним завданням, оскільки він залежить від численних змінних факторів, які можуть істотно вплинути на ефективність їзди та комфорт велосипедиста. Розглянемо детальніше основні фактори, що визначають вибір передатного числа [4]:

1. Фізичні параметри велосипедиста: вага, зріст, рівень фізичної підготовки, техніка педалювання — всі ці параметри мають безпосередній вплив на здатність підтримувати певну швидкість на різних ділянках траси. Наприклад, важчі велосипедисти можуть потребувати легшого передатного числа для подолання тієї ж відстані, ніж легші колеги, оскільки їм потрібно більше зусиль для підтримання оптимального темпу. Крім того, фізична підготовка велосипедиста також відіграє важливу роль: ті, хто має вищу витривалість і силу, можуть ефективніше використовувати вищі передатні числа. Однак новачки або менш підготовлені спортсмени можуть виявити, що легші передачі дозволяють їм легше справлятися з навантаженнями та підтримувати комфорт під час їзди.

Характер дороги**:** Середній нахил, різкі зміни висоти, наявність підйомів і спусків є критичними факторами, що впливають на витрати енергії та зусилля, які потрібно докласти для педалювання. Короткі, але круті підйоми потребують значно більше зусиль, ніж тривалі, пологі спуски. Велосипедистам доводиться адаптувати свої зусилля залежно від того, чи піднімаються вони на гору, чи спускаються вниз. Погані погодні умови, такі як вітер, також можуть значно впливати на вибір передачі, адже протидія вітру потребує додаткових зусиль і може вимагати легшого передатного числа для підтримання швидкості.

Зігдно ДСТУ 3587:2013 "Дороги автомобільні. Нормативи проектування"

[5] більшість доріг загального користування мають відносно не великі кути нахилу.

Для автомагістралей та основних доріг максимальний нахил складає 6% (приблизно 3,4 градуса).

Для вторинних доріг (з меншою інтенсивністю руху) максимальний нахил може бути до 8% (приблизно 4,6 градуса).

Для дорожніх підйомів у складних умовах (гірські райони, важкий рельєф) максимальний нахил може досягати 12-20%.

Це дозволяє пересуватися більшістю доріг загального користування маючи лише одне передаточне співвідношеня, проте ці показники достатні для того що б зробити відчутну різницю на різних місцевостях та ділянках дороги. Також треба враховувати людей котрі їздять по бездоріжжю у лісовій та гірській місцевості, де ці нормативи не застосовуються. Тому підбір оптимального передаточного співвідношення дуже важливий для комфортної їзди.

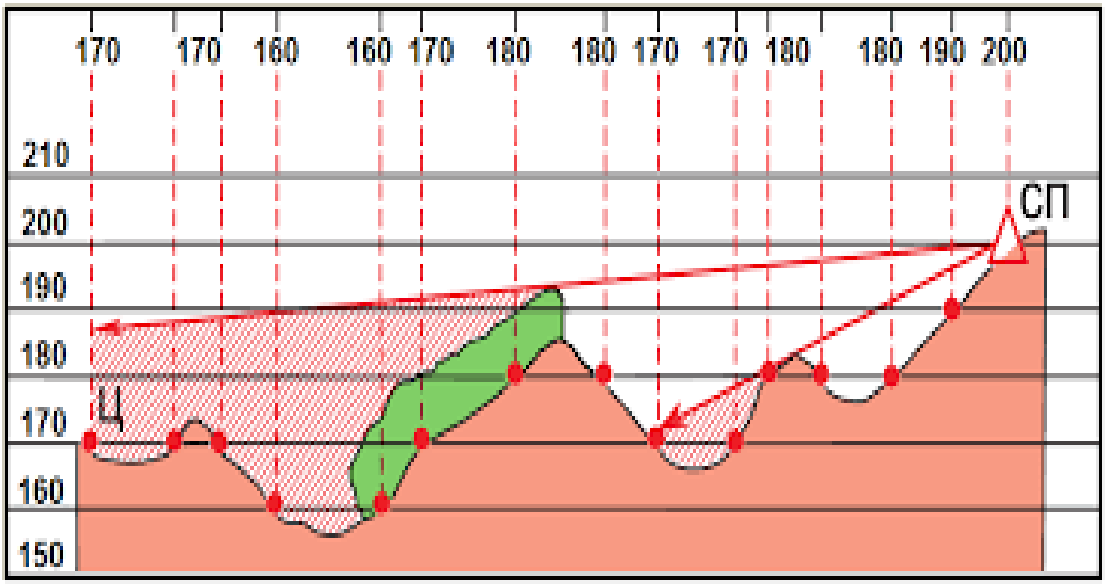

Рисунок 2.1 –Профіль дороги з характерними перепадами висот

2. Покриття дороги: дорожнє покриття справляє великий вплив на витрати енергії велосипедиста. Їзда на асфальті є значно легшою, ніж на гравійному покритті або по лісових стежках. Чим більш нерівна або слизька

поверхня, тим більше зусиль потрібно для подолання відстані. Наприклад, під час руху по нерівному або розмоклому ґрунту велосипедист може зіткнутися з додатковими труднощами, що вимагатиме використання легшого передатного числа. Додатково, неякісне покриття може призвести до втрати контролю над велосипедом, що є особливо небезпечним для фіксованих передач, оскільки відсутність механізму вільного ходу може ускладнити реакцію в критичних ситуаціях.

3. Бажана швидкість: велосипедисти можуть мати різні цілі під час поїздки, які варіюються від простої прогулянки до високошвидкісної їзди. Залежно від цього, вибір передатного числа може істотно змінюватися. Наприклад, якщо мета — максимальна швидкість на короткій дистанції, то краще використовувати вищу передачу. Водночас, для довгих поїздок, де важливо підтримувати енергію на протязі всього маршруту, бажано використовувати нижче передатне число, що дозволяє зменшити навантаження на ноги та підвищити комфорт під час тривалої їзди. Це особливо важливо для любителів, які прагнуть насолоджуватися поїздкою, не відчуваючи дискомфорту від перевантажень.

Навіть врахування такого фактору як опір повітря – вже впливатиме по різному на велосипедистів, знову ж – залежно від їх спортивної форми, статури, досвіду, маршруту та ін. Адже зі зростанням швидкості аеродинамічний опір зростає, так би мовити, випереджаючими темпами, оскільки він пропорційний квадрату швидкості. Інакше кажучи, при 45 км / год на аеродинаміку йде в 2,25 рази більше ват, ніж при 30 км / ч. Адже ви їдете швидше в півтора рази: 45 : 30 = 1,5 А опір виріс як квадрат швидкості: 1,5 х 1,5 = 2,25.

Незважаючи на значний вплив цих факторів, більшість існуючих калькуляторів передатного числа базуються на спрощених моделях, що використовують середні значення без урахування специфічних особливостей траси чи індивідуальних характеристик користувача. Це обмежує ефективність і універсальність таких підходів, особливо в умовах, коли траса має складний рельєф або коли необхідна точна адаптація до фізичних параметрів

велосипедиста.

Існуючі калькулятори можуть не враховувати, що на одній і тій же трасі в різні пори року або при різних погодних умовах можуть знадобитися різні налаштування. Оскільки навіть незначні зміни в рельєфі або типі покриття можуть вплинути на зусилля, які витрачає велосипедист, важливо мати можливість адаптувати передатне число в реальному часі.

Таким чином, для створення дійсно ефективного рішення необхідно розробити нові підходи, які б враховували не тільки зазначені фактори, а й інші параметри, такі як техніка їзди, особливості маршруту та навіть індивідуальні вподобання велосипедиста. Застосування сучасних технологій, таких як машинне навчання та аналітика даних, може суттєво підвищити точність рекомендацій і допомогти велосипедистам обирати оптимальне передатне число в різних умовах. Це дозволить значно підвищити комфорт, ефективність і безпеку їзди на фіксованих передачах, а також зробить процес вибору більш персоналізованим і адаптованим до потреб кожного окремого користувача.

2.2 Методи розрахунку передатного числа

Вибір оптимального передатного числа для велосипедів з фіксованою передачею традиційно базується на ряді класичних методів, які враховують фізичні характеристики велосипеда, параметри траси та фізичні здібності велосипедиста. Основні методи, що використовуються для розрахунку передачі, включають:

1. Розрахунок співвідношення зірок. Це один з найпростіших способів, який передбачає обчислення передатного числа на основі кількості зубців на передній та задній зірках. Співвідношення визначається як кількість зубців передньої зірки, поділена на кількість зубців задньої зірки. Цей підрахунок надає базове уявлення про те, як буде відчуватися педалювання на різних швидкостях.

Рисунок 2.2 – Розрахунок передатного числа по співвідношенню зірок

Розглянемо приклад для велосипеда з фіксованою передачею, що має передатне відношення 49/17. Це означає, що передня зірка має 49 зубців, а задня – 17. При такому співвідношенні передача становить $49 : 17 \approx 2{,}88$.

Іншими словами, за один повний оберт шатунів заднє колесо зробить 2,88 оберту. Припустимо, що велосипед обладнаний колесами з діаметром 28 дюймів, що відповідає приблизному обводу колеса (колесу з покришкою) у 2,2 метра. Тоді за один оберт шатунів велосипед проїжджає: $2{,}88 \times 2{,}2 \approx 6{,}34$ метра.

Це передатне відношення є компромісом між швидкістю та тяговими характеристиками, що підходить для використання на рівних дорогах або легких підйомах. При такій передачі велосипедист може підтримувати досить високу швидкість, зберігаючи комфортний рівень обертання педалей. Однак на крутих підйомах або при різких зупинках потрібна значна фізична сила, оскільки відсутність перемикачів передач і гальм на фіксованій передачі змушує працювати ногами як на розгін, так і на уповільнення. Таке співвідношення 49/17 історично стало популярним серед трекових гонщиків, а пізніше перекочувало в міське середовище, де його обирають за універсальність і ефективність [2]. Воно дозволяє відчути справжню динаміку руху і вимагає від велосипедиста оптимального контролю та майстерності.

Рисунок 2.3 - Передня зірка велосипедних шатунів для треку, кількість зубів 49

2. Врахування розміру колеса**.** Розмір колеса має великий вплив на ефективність передачі, оскільки він визначає, наскільки далеко велосипед рухається за один оберт педалей. Чим більший діаметр колеса, тим більша дистанція, яку проходить велосипед за один оберт. Це також слід враховувати при виборі передачі, адже занадто велике передатне число може ускладнити старт і підйом на крутій місцевості.

3. Аналіз фізичних характеристик велосипедиста**.** Індивідуальні параметри, такі як вага, зріст і фізична підготовка, відіграють важливу роль у виборі передатного числа. Наприклад, важчі велосипедисти можуть потребувати вищого передатного числа для подолання сили тяжіння на підйомах, тоді як легші велосипедисти можуть використовувати нижче передатне число для збереження енергії.

4. Оцінка умов маршруту**.** Стан дороги, нахил, наявність підйомів та спусків значно впливають на вибір передачі. При підйомі на крутих схилах краще використовувати нижче передатне число, що дозволяє легше долати опір. В той же час, на рівних ділянках можна використовувати вищі передатні числа для досягнення вищих швидкостей.

5. Пробний метод. Оскільки вибір оптимального передатного числа часто залежить від особистих уподобань, багато велосипедистів використовують пробний метод, підбираючи передачу шляхом практичного тестування на різних маршрутах. Це дозволяє врахувати особистий стиль їзди та адаптувати налаштування до індивідуальних потреб. Всі ці методи є важливими елементами, які дозволяють велосипедистам здійснити обґрунтований вибір передатного числа, адаптуючи його до своїх потреб і умов їзди. Хоча вони можуть здаватися простими, їх правильне застосування має вирішальне значення для забезпечення комфорту та безпеки під час їзди на велосипеді з фіксованою передачею.

Висновок: існує ряд факторів, як то фізичні параметри велосипедиста, характер і покриття дороги, бажана швидкісь та ін., що впливають на вибір передатного числа. Існуючі методи розрахунків передатного числа базуються на одному чи кількох факторах і не дають повного розуміння оптимального рішення з врахуванням необхідних і різних параметрів.

РОЗДІЛ 3. ВИКОРИСТАННЯ НЕЧІТКОЇ ЛОГІКИ ДЛЯ РОЗВ'ЯЗАННЯ ПРОБЛЕМИ

3.1 Обгрунтування вибору методу обробки даних

Застосування нечіткої логіки до вибору оптимального передатного числа є перспективним рішенням завдяки її здатності працювати з нечіткими даними та одночасно враховувати декілька факторів. Нечітка логіка дозволяє створювати алгоритми, які здатні адаптуватися до складних трас і індивідуальних потреб велосипедиста, не обмежуючись лише середніми значеннями.

У контексті вибору передатного числа нечітка логіка дозволяє встановлювати правила, які враховують параметри траси, дорожнього покриття, фізичні характеристики велосипедиста та інші параметри, на основі яких надаються рекомендації щодо оптимального передатного числа. Наприклад, для ділянок з крутими підйомами система може рекомендувати низький передатний коефіцієнт, тоді як для рівнинних ділянок — вищий. Це забезпечує більш точну і персоналізовану настройку, що підвищує ефективність їзди.

Основною перевагою нечіткої логіки є її здатність адаптуватися до різноманітних умов. Завдяки використанню нечітких змінних можна встановлювати правила для кожного параметра, що підлягає врахуванню [6]. Наприклад, вхідними даними можуть бути не лише показники швидкості та підйому, а й змінні, такі як втома велосипедиста або стан покриття дороги. У підсумку, вихідне значення передатного числа розраховується на основі багатофакторного аналізу, який враховує як індивідуальні особливості велосипедиста, так і специфіку маршруту.

Це дозволяє не тільки підвищити ефективність, а й знизити ризики, пов'язані з неправильним вибором передачі. Наприклад, у ситуаціях, коли велосипедисти стикаються з несподіваними змінами на трасі, такими як раптові підйоми або погіршення дорожніх умов, нечітка логіка може забезпечити швидке

коригування передатного числа, що сприятиме збереженню контролю та безпеки під час їзди.

Більше того, інтеграція нечіткої логіки в системи вибору передатного числа може призвести до розвитку нових алгоритмів, які зможуть вчитися на основі історичних даних про поїздки. Це дозволить системі покращувати свої рекомендації з часом, адаптуючись до особистих уподобань і стилю їзди кожного велосипедиста. Таким чином, застосування нечіткої логіки не тільки розширює можливості вибору передачі, але й відкриває нові горизонти для персоналізації та підвищення якості велосипедних поїздок.

3.2 Опис нечіткої логіки

Нечітка логіка (fuzzy logic) є розширенням класичної логіки, що дозволяє моделювати невизначеність, нечіткість і суб'єктивність у процесах прийняття рішень. Нечітка логіка — це просто гнучкий варіант і розширення класичної логіки. Нечітка логіка може представляти твердження, які є повністю істинними чи хибними, і також може представляти ті, що частково вірні [7].

Ця парадигма базується на концепції нечітких множин, введеної Лотфі Заде в 1965 році. Нечітка логіка стає особливо актуальною в ситуаціях, де традиційні бульові логічні системи (такі, що ґрунтуються на принципах двох значень: істина або хибність) виявляються недостатніми для адекватного опису реальних процесів [8].

У класичному понятті чіткої бінарної множини існує функція як сходинка, яка може дати системі лише два значення: 1 або 0. Л. Заде запропонував оперування нескінченною кількістю значень 0 та 1. Тому в функціях приналежності для елементу множини надано безліч значень в інтервалі [1; 0]. Такі множини й були названі нечіткими (fuzzy).

Основною причиною появи нової теорії стала наявність нечітких і приближенних суджень при описі людиною процесів, систем, об'єктів. Перш

ніж нечіткий підхід до моделювання складних систем отримав визнання в усьому світі, пройшло не одне десятиліття з моменту зародження цієї теорії.

Перший період (кінець 60-х – початок 70 рр.) характеризується розвитком теоретичного апарату нечітких множин (Л. Заде, Е. Мамдани, Беллман). Л. Заде визначив ряд операцій, які можна виконувати над нечіткими множинами, і запропонував узагальнення відомих методів логічного виведення modus ponens (той, що підтверджує) та modus tollens (метод від супротивного) в контексті нечітких множин. Ним було введено поняття лінгвістичної змінної. Припустивши, що в якості значень (термінів) виступають нечіткі множини, Л. Заде створив апарат для опису процесів інтелектуальної діяльності, включаючи нечіткість і невизначеність виразів. Для нечіткої підмножини характерною особливістю є те, що на визначеному проміжку [1; 0] існує безліч значень.

В другому періоді (70–80-е роки) з'являються перші практичні результати в області нечіткого управління складними технічними системами, прийняття рішень в медицині і економіці.

В третьому періоді, який почався після доведення Бартоломієм Коско знаменитої теореми FAT (Fuzzy Approximation Theorem) і продовжується з кінця 80-х років до даного часу, з'являються пакети програм для побудови нечітких експертних систем, а області застосування нечіткої логіки помітно розширюються: вона застосовується в автомобільній, аерокосмічній і транспортній промисловості, побутовій техніці, в сфері прийняття управлінських рішень, фінансів, аналітики та ін.

Таблиця 3.1 – області застосування нечіткої логіки

Сфера використання	Компанія	Нечітка логіка
1	2	3
Антиблокувальна система гальм	Nissan	Використання нечіткої логіки для керування гальмами в небезпечних випадках, залежить від швидкості автомобіля, швидкості колеса, прискорення

Продовження табл. 3.1

1	2	3
Автоматична коробка передач	NOK / Nissan	Нечітка логіка використовується для керування вприском палива і запалюванням на основі налаштування дросельної засувки, температури охолоджувача, об / хв і т.д.
Авто двигун	Honda, Nissan	Використовується я для вибору режиму в залежності від навантаження двигуна, стилю керування і дорожних умов.
Копірувальна машина	Canon	Використовується для регулювання напруження барабану в залежності від щільності зобрваження, вологості і температури.
Круїз-контроль	Nissan, Isuzu, Mitsubishi	Використовується для налаштування дросельної засувки, щоб встановити швидкість і прискорення автомобіля
Посудомийна машина	Matsushita	Використовується для регулювання циклу очищення, стратегії полоскання і миття залежать від кількості посуду і його забруднення
Керування ліфтом	Fujitec, Mitsubishi Electric, Toshiba	Використовується, щоб скоротити час очікування в залежності від пасажиропотоку
Фітнес управління	Omron	Нечіткі правила, що використовуються для перевірки працездатності своїх співробітників.
Микрохвильова піч	Mitsubishi Chemical	Встановлює силу випромінювання і стратегію приготування
Планшет	Hitachi, Sharp, Sanyo, Toshiba	Розпізнає рукописні символи

3.2.1 Нечіткі множини. Англійське слово fuzzy, від якого утворене прикметник Fuzzy (нечіткий), означає «ворс», оскільки малюнок на ворсистій тканині здається розмитим. Отже, кажучи «нечіткий», мають на увазі «неясний», «розмитий». Тобто коли означення повністю зрозуміле, але сказати, чи належить до цієї множини один або інший параметр однозначно, лише за допомогою слів «так» або «ні» не просто, оскільки йдеться про невизначені, нестрогі властивості

об’єктів дослідження. На відміну від цього світ, властивості якого можна строго визначити двома словами, наприклад «біле або чорне?», можна назвати чітким світом, логіку комп'ютерів, які мають справу з 0 і 1, чіткою логікою, а звичайні множини - чіткими множинами. Зауважимо, що нечіткі множини і нечітку логіку можна розглядати як розширення відповідних чітких понять.

Нечітка множина визначається за допомогою функції приналежності, що дозволяє елементам мати ступінь приналежності в інтервалі від 0 до 1 [9]. Це забезпечує гнучкість у формулюванні понять, які не можна точно охарактеризувати за допомогою бульової логіки. Формально, для нечіткої множини A, визначеної в універсальній множині X, функція приналежності μA описується як:

$$\mu A: X \rightarrow [0,1]$$

Функцією належності A (x) називається функція, що дозволяє для довільного елементу xX визначити ступінь його належності нечіткій множині. Для кожного конкретного значення x∈X величина μA(x) набуває певного значення із замкненого інтервалу [0; 1], яке називається ступенем належності елемента x нечіткій множині ~ A . Будемо вважати, що до множини ~ A не входять елементи (μA(x) / x) для яких μA (x) = 0. Функція належності μA (x) є одним з базових понять теорії нечітких множин. Більшість дослідників вважають, що вона є деяким не імовірнісним суб’єктивним виміром нечіткості, який відрізняється від імовірнісної міри. Тобто, ступінь належності μ A (x) елемента x до нечіткої множини ~ A інтерпретується як суб’єктивна міра того, наскільки елемент x∈X відповідає поняттю, зміст якого формалізується нечіткою множиною ~ A.

Класичний приклад — нечітка множина для терміна "високий зріст". Якщо \(h \) — це зріст особи в сантиметрах, то:

$$mu_A(h) = 0, if h < 170 \frac{(h - 170)}{10}, if 170 \leq h < 1801, if h \geq 180$$

Цей підхід дозволяє описувати складні й нечіткі категорії, що є надзвичайно корисним у багатьох прикладних ситуаціях.

3.2.2 Лінгвістичні змінні. Будь-які подання знань в більшості випадків реалізуються в лінгвістично вільних формах і конче пов'язані з необхідністю урахування суб'єктивізму джерел цих знань [10]. Лінгвістичні змінні використовуються для формулювання нечітких концепцій. Вони дозволяють виражати значення через терміни природної мови, що робить моделювання більш зрозумілим для користувача. Наприклад, змінна "температура" може бути описана такими термінами, як "холодно", "помірно", "гаряче". Кожна з цих лінгвістичних змінних представлена нечіткими множинами з відповідними функціями приналежності. Ступінь приналежності може трактуватися по різному, в залежності від задачі, в якій використовуються нечіткі множини. Можливі трактування ступеню приналежності: ступень відповідності поняттю А, імовірність, можливість, корисність, істинність, правдоподібність, значення функції та ін. Для кожного трактування ступеню приналежності розроблені свої методи побудови функцій приналежності. В деяких моделях м'яких розрахунків функції приналежності задаються достатньо довільно в параметричному вигляді. Кожна лінгвістична змінна складається з:

- назви;
- множини своїх значень, яка також називається базовою терм-множиною Т. Елементи базової терм-множини являють собою назви нечітких змінних;

- універсальної множини X;
- синтаксичного правила G, по якому генеруються нові терми з застосуванням слів природньої або формальної мови;
- семантичного правила P, яке кожному значенню лінгвістичної змінної ставить в відповідність нечітку підмножину множини X.

3.2.3 Нечіткі правила. Над нечіткими множинами можна виконувати різні операції. При цьому вони мають визначатись таким чином, щоб операції над нечіткими множинами узагальнювали відповідні операції над звичайними множинами, тобто щоб в окремому випадку, коли множина є чіткою, операції перетворювалися на звичайні операції теорії множин. Оскільки у теорії нечітких множин ступінь належності не обмежений бінарними значеннями 0 і 1, як у звичайній теорії множин, а може набувати значень з інтервалу [0; 1], таке узагальнення можна реалізувати різними способами, які дозволяють враховувати різноманітні смислові відтінки відповідних логічних зв'язок "І", "АБО", "НЕ" у різних предметних областях. Таким чином, будь-якій операції над звичайними множинами у теорії нечітких множин може відповідати кілька операцій.

Правила нечіткої логіки, які мають форму "Якщо ... , тоді ...", є основою для формулювання висновків у нечітких системах. Наприклад, правило: "Якщо температура висока, тоді швидкість вентилятора висока" може бути представлене в системі так:

$$R1: \text{Якщо} T \text{висока}, \text{тоді} V \text{висока}$$

де T — температура, а V — швидкість вентилятора. Нечіткі правила можуть комбінуватися, щоб сформулювати більш складні рішення, що враховують кілька вхідних змінних.

3.2.4 Процес виведення. Процес виведення в системах нечіткої логіки включає в себе три основні етапи: фазифікацію, застосування нечітких правил і дефазифікацію. Основою для проведення операції нечіткого логічного виводу є база правил, що вміщує нечіткі висловлювання в формі "Якщо - То" і функції приналежності для відповідних лінгвістичних термів. При цьому мають виконуватись наступні умови:

– існує хоча б одне правило для кожного лінгвістичного терма вихідної змінної;

– для любого терма вхідної змінної є хоча б одне правило, в якому цей терм використовується як передумова (ліва частина правила).

Інакше має місце не повна база нечітких правил.

Дефазифікація — це процес перетворення нечітких значень, отриманих на етапі виведення, у чіткі значення, які можуть бути використані для прийняття рішень.

Найпоширенішими методами виведення є метод Мамдані [11] та метод Такагі.

Метод Мамдані (Mamdani). В ньому використовується мінімаксна композиція нечітких множин.

Для кожного правила визначається ступінь активності, яка обчислюється на основі значень вхідних змінних. Вихідні нечіткі множини обчислюються шляхом взяття мінімуму між входами і ступенем приналежності виходу. Послідовність дій:

1. Процедура фазифікації: визначаються ступені істинності, тобто значення функцій приналежності для лівих частин кожного правила (передумов). Для бази правил з m правилами позначимо ступені істинності як

$$Aik(xk),\ i=1..m,\ k=1..n$$

2. Нечітке виведення. Спочатку визначаються рівні «відсічення» для

лівої частини кожного з правил, потім знаходяться «усічені» функції приналежності. :

$$B_i^*(y) = \min_i (alfa_i , B_i(y))$$

3. Композиція, або об'єднання отриманих усічених функцій, для чого використовується максимальна композиція нечітких множин. :

$$MF(y) = \max_i (B_i^*(y))$$

— функція приналежності підсумкової нечіткої множини.

4. Дефазифікація, або приведення до чіткості. Існує кілька методів дефазифікації. Наприклад, метод середнього центру, або центроїдний метод: геометричний сенс такого значення — центр ваги підсумкової фігури.

$$y = \int \mu(y)dy \int y \cdot \mu(y)dy$$

Цей метод дозволяє обчислити середнє значення виходу на основі функції приналежності.

- Метод Такагі**:** Використовується для комбінування виходів з кількох правил. В ньому виходи комбінуються шляхом операцій, таких як логічне "Або".

Sugeno и Takagi припускали, що виведення можна представити у вигляді лінійних функцій. Це ще більш жорстке обмеження, чим у алгоритму Цукамото, але при цьому також загальне виведення приймається як середньовиважене значення виведення кожного правила, що дозволяє відразу отримати чітке виведення без використання процесу дефазифікації [12].

3.2.5 Структура системи нечіткої логіки. Система нечіткої логіки складається з декількох ключових компонентів:

1. Вхідні змінні**:** Це параметри, які можуть бути чіткими або нечіткими, і зазвичай є даними, що підлягають аналізу.

2. Фаззифікація**:** Процес перетворення чітких значень у нечіткі, використовуючи відповідні функції приналежності.

3. Нечіткі правила**:** Це набір правил, які визначають, як вхідні дані впливають на вихідні дані.

4. Виведення: Генерація нечітких виходів на основі комбінації вхідних даних і правил.

5. Дефазифікація**:** Перетворення нечітких виходів у чіткі значення, які можуть бути використані для подальшої обробки або прийняття рішень.

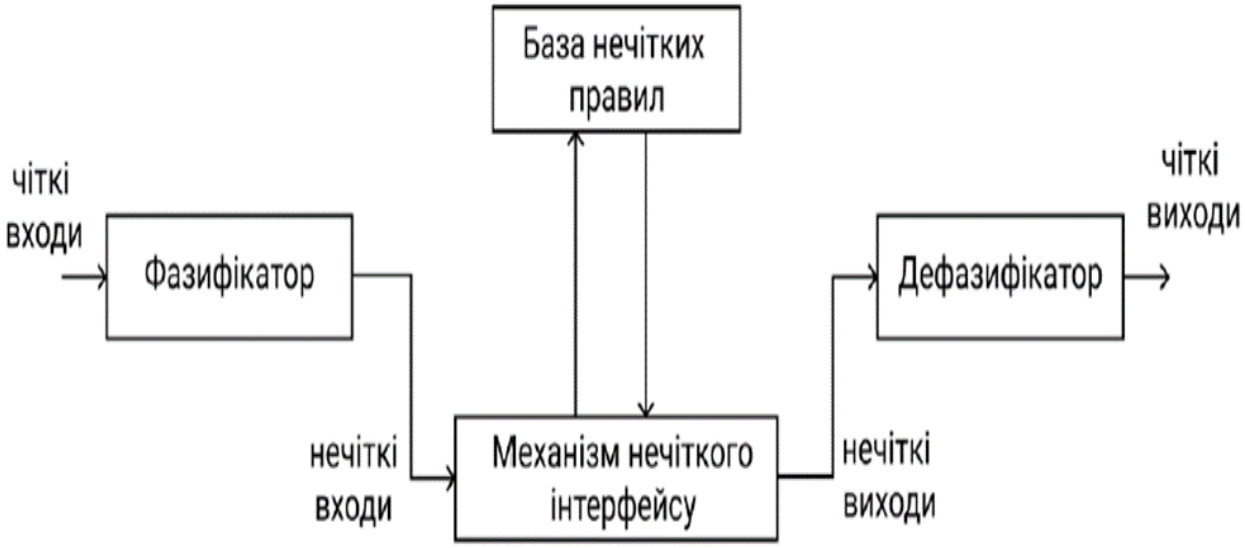

Рисунок 3.1 - Загальна структура нечіткої логіки

3.3 Чіткий проти нечіткого – порівняльний аналіз

Потужність та інтуїтивна простота нечіткої логіки як методології вирішення проблем гарантує її успішне використання в вбудованих системах контролю і аналізу інформації. На відміну від традиційної математики, що вимагає на кожному кроці моделювання точних і однозначних формулювань, нечітка логіка пропонує абсолютно новий підхід мислення, завдяки якому процес моделювання відбувається на високому рівні абстракції, при якому постулатами визначається лише мінімальний набір закономірностей.

Нечітка логіка, як метод моделювання складних систем, має численні

переваги, але також стикається з певними обмеженнями, які варто розглянути детальніше. Порівняльна характеристика наведена в таблиці 3.1.

Таблиця 3.2 –Переваги і недоліки нечіткої логіки

Переваги		Недоліки	
Параметр	Опис	Параметр	Опис
Гнучкість	Дозволяє враховувати нечіткість та суб'єктивність, що є важливим у реальних сценаріях	Налаштування	Складність у налаштуванні: Визначення функцій приналежності та правил може вимагати значних зусиль і досвіду
Інтуїтивність	Використання лінгвістичних змінних робить систему більш зрозумілою для користувачів, які не мають глибоких математичних знань	Строгі критерії	Відсутність строгих критеріїв, через свою природу нечітка логіка може не забезпечувати абсолютної точності у висновках
Можливість обробки комплексних систем	Може ефективно моделювати системи з багатьма взаємопов'язаними параметрами		

3.3.1. Переваги нечіткої логіки. По-перше, нечітка логіка дозволяє працювати з невизначеністю та нечіткістю даних [13]. Вона особливо корисна в ситуаціях, де класичні підходи можуть не дати точних результатів через недостатню чіткість або точність вхідних даних. Наприклад, коли мова йде про вибір оптимальних параметрів для велосипедистів, важливо врахувати різноманітні змінні, такі як погода, дорожні умови та фізичні характеристики. Нечітка логіка дозволяє формулювати правила, які можуть мати розмиті значення, що допомагає системі адаптуватися до реальних умов.

По-друге, нечітка логіка здатна інтегрувати експертні знання, що робить її дуже корисною в складних завданнях, де потрібно врахувати багато факторів. За допомогою нечітких множин і правил можна моделювати неформальні знання експертів у певній галузі. Це може бути особливо корисно в контексті спортивних технологій, де варіації в поведінці велосипедистів і характеристиках трас вимагають гнучкості у прийнятті рішень.

По-третє, системи на основі нечіткої логіки зазвичай прості у використанні. Вони можуть бути представлені у вигляді графіків, діаграм та зрозумілих правил, що робить їх доступними для широкого кола користувачів, навіть якщо вони не мають глибоких технічних знань. Це дозволяє розширити коло потенційних користувачів, включаючи аматорів, які можуть бути менш обізнані в складних математичних моделях.

3.3.2 Обмеження та недоліки нечіткої логіки. Однак, незважаючи на всі ці переваги, нечітка логіка має свої обмеження. По-перше, для розробки ефективної системи необхідно створити чітку базу знань, що включає в себе правила і параметри, які повинні бути точно визначені. Це може вимагати значних зусиль для збору та аналізу експертних даних, а також часу на обробку інформації. Наприклад, щоб побудувати ефективну модель для вибору оптимального передатного числа на велосипеді, потрібно врахувати численні фактори, що впливають на ефективність їзди.

По-друге, системи на основі нечіткої логіки вимагають регулярного оновлення даних, щоб враховувати нові фактори і зміни в умовах. Це може включати нові види дорожніх покриттів, зміни в техніці їзди або нові дослідження фізичних параметрів велосипедистів. Невиконання цього вимоги може призвести до застарілих рекомендацій і зниження точності моделі.

Крім того, нечітка логіка може стикатися з труднощами у визначенні меж нечітких множин, що може призвести до нестабільності в результатах. Якщо правила не будуть чітко сформульовані, то система може давати суперечливі або неточні рекомендації, що може зашкодити безпеці та ефективності.

Врешті-решт, незважаючи на обмеження, використання нечіткої логіки у вирішенні складних задач, таких як розрахунок оптимального передатного числа для велосипедистів, може суттєво покращити якість їзди. Ця технологія дозволяє забезпечити більш адаптивне, безпечне і ефективне використання велосипедів в різних умовах, підвищуючи комфорт і продуктивність велосипедистів, навіть в умовах значної невизначеності.

Висновок: застосування нечіткої логіки, як методу моделювання складних систем, дозволяє встановлювати правила, які враховують необхідні численні параметри, що не можуть бути постійними і вираженими точними величинами. Маючи свої переваги та недоліки, загалом нечітка логіка є перспективним рішенням для розробки застосунку для розрахунку оптимального передатного числа.

РОЗДІЛ 4. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

Нечітка логіка знайшла застосування в різноманітних сферах, таких як автоматизація, системи контролю, робототехніка, медичні діагностики та фінансовий аналіз. Її здатність адаптуватися до змінних умов робить її цінним інструментом у вирішенні складних завдань.

Surplace Fantastic Fixed Gear Calculator — це спеціалізований онлайн-інструмент, призначений для велосипедистів, які користуються фіксованими передачами. Цей калькулятор дозволяє користувачам вводити параметри свого велосипеда, такі як кількість зубців на передніх та задніх зірках, а також діаметр колеса. Калькулятор виконує розрахунки, щоб визначити оптимальне передатне число та швидкість, яку може досягти велосипедист з обраними налаштуваннями. Користувачі можуть експериментувати з різними комбінаціями зірок, щоб знайти найбільш підходящі для своїх умов їзди, але цей калькулятор видає лише “сиру” інформацію, яку не досвідченний велосипедист не може правильно інтерпретувати й йому доведеться йти шляхом проб та помилок.

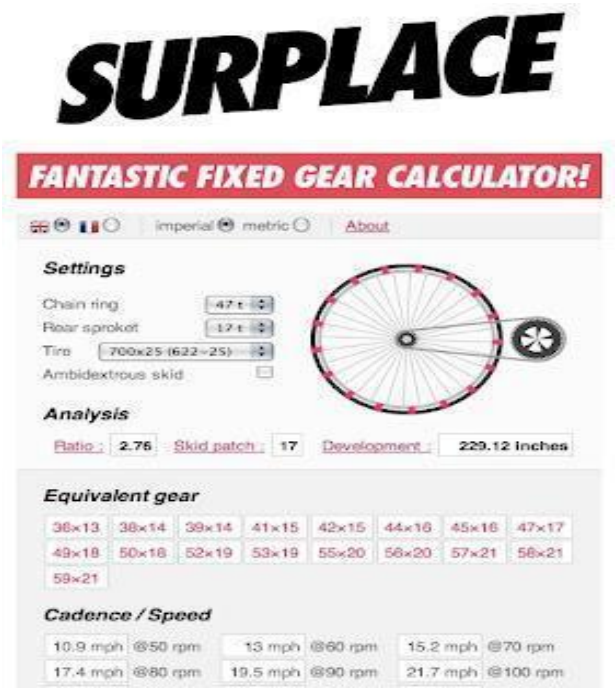

Рисунок 4.1 – Приклад роботи калькулятора Surplace Fantastic Fixed Gear Calculator

Переваги:

- простота використання: інтуїтивно зрозумілий інтерфейс дозволяє навіть новачкам легко знайти потрібні дані;

- спеціалізація: калькулятор розроблений виключно для фіксованих передач, що дозволяє отримати точні результати для цього типу велосипедів;

- експериментування: можливість налаштовувати параметри зірок дає змогу користувачам знаходити оптимальні комбінації для своїх потреб.

Недоліки:

- Обмеженість функціоналу: Калькулятор не враховує специфіку різних типів покриття та умов їзди, що може призвести до невідповідності між розрахунковими даними та реальними відчуттями.

- Прості формули: Хоча калькулятор ефективно виконує розрахунки, він не надає користувачам глибокого розуміння того, як ці цифри впливають на їзду в реальних умовах.

Bicycle Gear Calculator — це універсальний онлайн-інструмент, призначений для розрахунку передач на багатошвидкісних велосипедах. Користувачі можуть вводити параметри, такі як кількість зубців на передній та задній зірках, діаметр колеса, тип передачі (гірський, шосейний тощо) і отримувати дані про оптимальну швидкість, оберти педалей та ефективність передач. Цей калькулятор дозволяє користувачам порівнювати різні комбінації передач, що допомагає знаходити найбільш підходящі налаштування для їх стилю їзди. Однак важливо відзначити, що **Bicycle Gear Calculator** більше орієнтований на багатошвидкісні велосипеди і не має окремого режиму для одношвидкісних моделей, таких як fixed-gear. Це може створити певні труднощі для користувачів, які шукають специфічні дані для своїх фіксованих велосипедів.

Переваги:

- Універсальність: Калькулятор дозволяє розраховувати передачі для різних типів велосипедів, що робить його корисним для широкого кола

користувачів.

- Порівняння комбінацій: Можливість експериментувати з різними передавальними співвідношеннями дозволяє велосипедистам знайти найбільш підходящі варіанти.

Максимальное передаточное число		3.45
Минимальное соотношение		0.72

		26	38
11	18%	2.36	3.45
13	15%	2.00	2.92
15	13%	1.73	2.53
17	12%	1.53	2.24
19	11%	1.37	2.00
21	14%	1.24	1.81
24	17%	1.08	1.58
28	14%	0.93	1.36
32	13%	0.81	1.19
36		0.72	1.06

Скорость (км/ч)				
Каденция:	60	80	100	120
Высшая передача	26.0	34.6	43.3	51.9
Самая низкая передача	5.4	7.2	9.0	10.8

Рисунок 4.2 – Приклад роботи калькулятора Bicycle Gear Calculator

Недоліки:

- Відсутність режиму для фіксованих передач: Користувачі, які їдуть на фіксованих велосипедах, можуть не знайти необхідних функцій, що обмежує застосування цього калькулятора для них.

- Прості математичні розрахунки: Як і в випадку з Surplace, даний калькулятор базується на простих формулах, що не дають користувачам чіткого уявлення про те, як результати впливають на практику їзди, не враховуючи відчуття та досвід.

Обоє калькуляторів — Surplace Fantastic Fixed Gear Calculator та Bicycle Gear Calculator — є корисними інструментами для велосипедистів, але їх обмеження в контексті специфіки використання можуть вплинути на ефективність у реальних умовах. Вони здатні надати базові дані, однак користувачі повинні бути готові до того, що ці дані потребують додаткового контексту для практичного застосування.

Висновок: існуючі калькулятори не забезпечують необхідного розрахунку передатних чисел з врахуванням необхідних параметрів, є корисними інструментами, але з обмеженими можливостями.

РОЗДІЛ 5.
ПОСТАНОВКА ЗАВДАННЯ

Розробка системи для визначення оптимального передатного числа для велосипедів є складною задачею, що потребує комплексного підходу до аналізу різних факторів, що впливають на комфорт та ефективність їзди. Створення такої системи має враховувати різні аспекти велосипедної їзди — від фізичних характеристик користувача до профілю маршруту. Використання сучасних інструментів обробки даних та моделей на основі нечіткої логіки дозволить створити точну і надійну систему, що забезпечить високий рівень адаптивності до різних умов і вимог велосипедистів.

5.1 Мета і завдання проєкту

Основною метою даного проєкту є створення додатку, який на основі аналізу різноманітних даних, таких як рельєф траси, тип дорожнього покриття, кліматичні умови, фізичні параметри велосипедиста та інші ключові показники, буде надавати рекомендації щодо вибору оптимального передатного числа. Це передатне число дозволить оптимізувати витрати енергії велосипедиста, зробить їзду комфортнішою, та підвищить загальну ефективність пересування, незалежно від складності траси чи погодних умов.

Завдання проєкту можна розділити на кілька основних етапів, кожен із яких має свою конкретну мету і вимагає різного підходу до реалізації:

1. Аналіз та огляд літератури — дослідити існуючі підходи і методики у виборі передатного числа, проаналізувати їх переваги та недоліки для визначення того, які технології та методи можна адаптувати для розроблюваної системи.

2. Побудова математичної моделі — створення математичних моделей, що зможуть коректно обробляти комплексні параметри для вибору передатного числа, зокрема на основі нечіткої логіки.

3. Розробка бази знань і формування правил нечіткої логіки — визначення ключових параметрів, таких як тип поверхні, ухил траси, маса велосипедиста, вітрові умови тощо, а також формування правил, за допомогою яких система обиратиме передатне число.

4. Інтеграція з іншими системами — розробка механізмів для зчитування даних з зовнішніх систем, таких як GPS-додатки або файли у форматі GPX, для отримання та обробки інформації про маршрут.

5. Розробка інтерфейсу користувача — створення інтуїтивно зрозумілого та функціонального інтерфейсу, який дозволить користувачам легко вводити необхідні параметри і отримувати результат у зручному форматі.

6. Тестування і налагодження — проведення серії тестів для перевірки точності і надійності системи, включаючи тестування на різних типах маршрутів і під різними умовами.

5.2 Вимоги до системи

Система повинна задовольняти кілька ключових вимог для того, щоб вона могла виконувати поставлені завдання з максимальною точністю і ефективністю:

- гнучкість та адаптивність: система повинна бути налаштована на врахування широкого спектру параметрів, які можуть відрізнятися залежно від маршруту, типу їзди або індивідуальних потреб користувача;
- масштабованість**:** структура системи повинна дозволяти додавання нових параметрів або змінних, щоб адаптуватися до нових вимог чи розвитку технологій у велосипедній сфері;
- висока точність розрахунків: система повинна забезпечувати високий рівень точності при розрахунку оптимального передатного числа,

використовуючи повний набір параметрів, які мають значення для конкретної ситуації;

- можливість інтеграції з іншими додатками: забезпечення сумісності з популярними GPS-трекерами та додатками для обробки даних, такими як Strava чи Garmin, для автоматичного збору інформації;
- зручність у використанні: інтерфейс користувача повинен бути зрозумілим, а також забезпечувати швидке введення параметрів та перегляд результатів, щоб процес налаштування і використання системи був легким і доступним навіть для новачків.

5.3 Обмеження та припущення

Система для вибору передатного числа, хоч і має великий потенціал, стикається з рядом обмежень і припущень, які варто врахувати при розробці:

1. Залежність від точності даних: точність рекомендацій залежить від якості введених даних, таких як інформація про рельєф, дорожні покриття і параметри велосипедиста. Помилки в цих даних можуть призвести до неефективних або навіть небезпечних рекомендацій.
2. Необхідність у регулярному оновленні: для забезпечення актуальності даних і правил нечіткої логіки система потребує періодичного оновлення, що дозволить врахувати нові типи поверхонь або зміни у вимогах велосипедистів.
3. Обмеження математичних моделей: нечітка логіка добре підходить для обробки даних із великою кількістю варіантів, проте її точність може знижуватися при надмірному ускладненні умов чи при великій кількості змінних, що можуть бути важко відстежуванні.

5.4 Структура та компоненти системи

Запропонована система включатиме кілька ключових компонентів, кожен з яких відіграє свою роль у процесі вибору оптимального передатного числа. Нижче наведені основні елементи структури системи:

1. Модуль збору та обробки даних: цей модуль відповідає за отримання інформації від користувача, включаючи параметри, пов’язані з фізичними характеристиками, даними про маршрут (наприклад, рельєф, дорожні покриття, довжина траси), та умовами (наприклад, вітрові навантаження). Крім того, він може інтегруватися з іншими джерелами, такими як GPS, для автоматичного збору даних.

2. База знань**:** у базі знань зберігаються всі правила нечіткої логіки, що допомагають системі приймати рішення. Кожне правило визначає, які умови мають бути виконані для вибору конкретного передатного числа. Наприклад, правила можуть враховувати тип траси, рельєф та інші фактори, щоб надавати відповідні рекомендації. Цей модуль також може оновлюватися новими правилами у випадку, якщо з'являються нові вимоги.

3. Модуль нечіткої логіки: цей модуль містить всі необхідні алгоритми для обробки отриманих параметрів і застосування нечітких множин до конкретних значень. На основі введених значень і правил із бази знань цей модуль генерує рекомендації щодо передатного числа.

4. Інтерфейс користувача**:** забезпечує зручний спосіб введення даних та отримання результатів. Інтерфейс повинен бути інтуїтивним, з можливістю швидкого введення та перегляду результатів. Користувач повинен мати змогу ввести свої параметри та отримати рекомендації за кілька кліків.

5. Модуль аналізу результатів**:** після розрахунку оптимального передатного числа система може надати короткий аналіз, який пояснює користувачу, чому обране саме це передатне число, та як воно вплине на його їзду. Цей модуль також може включати функцію порівняння кількох передатних чисел для надання більш гнучких варіантів.

5.5 Очікувані результати та їх значення

Основними очікуваними результатами розробки є створення системи, здатної надавати індивідуалізовані рекомендації щодо передатного числа, базуючись на обраних параметрах, що підвищить комфорт і безпеку їзди в різних умовах. Ось основні переваги та значення очікуваних результатів:

1. Підвищення ефективності їзди: оптимізація передатного числа допоможе велосипедистам знизити витрати енергії та збільшити швидкість, особливо на довгих і складних маршрутах. Це особливо актуально для спортивних змагань, де точний вибір передачі може значно вплинути на результати.

2. Адаптивність до різних умов: завдяки можливості врахування параметрів рельєфу, покриття, клімату тощо, система зможе надати рекомендації, які адаптуються під конкретний маршрут та індивідуальні потреби користувача.

3. Зниження ризику травм**:** оптимізація передатного числа з урахуванням фізичних характеристик велосипедиста може зменшити навантаження на м’язи та суглоби, знижуючи ризик травм при їзді на високих передачах на важких ділянках маршруту.

4. Покращення досвіду користувача: система допоможе новачкам швидше зорієнтуватися у виборі передачі, а досвідченим велосипедистам надасть точні рекомендації, що дозволить зосередитися на техніці їзди, а не на підборі передачі.

5.6 Вимірювання та оцінка успішності

Оцінка ефективності розробленої системи буде базуватися на таких показниках [14]:

1. Точність рекомендацій: аналізуючи рівень задоволеності

користувачів на основі їхніх відгуків та результатів їзди, система буде адаптувати правила та покращувати рекомендації.

2. Адаптивність і масштабованість: можливість додавання нових параметрів і факторів буде показником гнучкості системи і здатності до подальшого розвитку.

3. Задоволеність користувачів: відгуки та досвід користувачів будуть ключовим критерієм оцінки, особливо для визначення, наскільки інтуїтивним і корисним є інтерфейс системи.

4. Зниження навантаження на велосипедиста**:** за допомогою аналізу показників продуктивності та зниження енергетичних витрат велосипедистів, можна оцінити, наскільки система справляється з покращенням комфорту та безпеки.

Висновок: проведено аналіз та огляд існуючих підходів і методик у виборі передатного числа. Визначено ключові параметри та вимоги до системи. Розроблено структуру системи з розумінням очікуваних результатів та оцінкою їх ефективності.

РОЗДІЛ 6. ПРОЕКТНІ РІШЕННЯ

Проектування системи вибору оптимального передатного числа для велосипеда на основі нечіткої логіки є складним і багатокомпонентним процесом. Він включає в себе не лише технічні аспекти, а й інтерфейсні рішення, розробку алгоритмів, а також інтеграцію з різноманітними джерелами даних для створення ефективної та надійної системи. Створення такої системи потребує детального аналізу вимог, ретельного вибору технологій, визначення архітектури і алгоритмів роботи з даними, що дозволяють реалізувати необхідні функції з максимальною ефективністю. У цьому розділі будуть детально розглянуті основні проектні рішення, які лежать в основі побудови цієї системи, а також обґрунтовані вибори компонентів, які забезпечують точність і надійність визначення оптимального передатного числа з урахуванням різноманітних умов їзди та індивідуальних параметрів користувача [15].

6.1 Загальна архітектура системи

Загальна архітектура системи є основою, на якій будується застосунок для обчислення оптимального передатного числа для велосипедів. Архітектура повинна бути гнучкою і масштабованою, щоб задовольняти різноманітні вимоги користувачів, враховуючи не тільки фізичні параметри велосипедиста, але й умови місцевості, тип покриття дороги, а також інші фактори, що можуть впливати на вибір оптимальної передачі. Основними складовими архітектури є модулі для збору і обробки даних, система нечіткої логіки для аналізу цих даних і прийняття рішень, а також інтерфейс користувача, який має бути простим та інтуїтивно зрозумілим. Всі компоненти системи повинні працювати злагоджено і швидко, забезпечуючи високий рівень продуктивності та точності розрахунків.

6.1.1 Компоненти архітектури. Архітектура системи складається з наступних основних компонентів, які забезпечують її функціональність та зручність використання:

Розглянемо користувацький інтерфейс (UI).

UI повинен бути інтуїтивно зрозумілим та зручним для користувача, забезпечуючи простоту введення даних, налаштувань та отримання результатів. Інтерфейс повинен підтримувати відображення графіків, таблиць і рекомендацій у зрозумілому форматі. Важливо враховувати, що дизайн повинен відповідати потребам різних категорій користувачів, від новачків до досвідчених велосипедистів.

Необхідно впровадити адаптивний дизайн, щоб система могла зручно працювати на різних пристроях, включаючи комп'ютери, планшети та мобільні телефони. Користувачі повинні мати можливість налаштовувати інтерфейс відповідно до своїх вподобань, наприклад, змінювати кольорову гаму або розташування елементів.

Опишемо модуль обробки даних.

Цей модуль відповідає за імпорт даних з GPS-пристроїв у форматі GPX та їхню валідацію. Він також повинен обробляти дані про висоти, координати, та типи покриття дороги, забезпечуючи їхню підготовку для подальшого аналізу.

Модуль повинен мати можливість обробляти великі обсяги даних і забезпечувати швидку обробку без втрати точності. Також доцільно реалізувати механізми для фільтрації та нормалізації даних, щоб уникнути помилок, пов'язаних із неточними чи неповними даними.

1. Модуль аналізу**:**

Основною функцією цього модуля є аналіз вхідних даних для визначення оптимальних параметрів їзди. Він повинен використовувати нечітку логіку для обчислення оптимального передатного числа на основі змінних, таких як швидкість, градієнти та типи покриття. Важливо, щоб модуль мав можливість враховувати зміни в умовах їзди.

Модуль повинен бути гнучким і мати можливість налаштовувати алгоритми в залежності від конкретних потреб користувача. Для досягнення цієї мети доцільно реалізувати систему самонавчання, яка б могла адаптуватися до індивідуальних стилів їзди та вподобань користувача.

2. Інтеграція з зовнішніми джерелами:

Система повинна бути здатною інтегруватися з зовнішніми сервісами, такими як OpenStreetMap, для отримання додаткової інформації про покриття дороги та профілі висот. Це розширить можливості аналізу та підвищить точність рекомендацій. Зокрема, інтеграція з метеорологічними службами дозволить враховувати погодні умови під час планування маршрутів.

Інтеграція повинна бути здійснена через API, що дозволить системі отримувати актуальні дані в реальному часі. Крім того, доцільно передбачити механізми обробки помилок, щоб уникнути збоїв у системі під час отримання зовнішніх даних.

6.1.2 Взаємозв'язки компонентів. Компоненти архітектури повинні бути взаємопов'язані, забезпечуючи безперебійну передачу даних та інформації. Користувач взаємодіє з UI, який передає введені дані в модуль обробки даних. Після обробки даних, результати направляються в модуль аналізу, який генерує рекомендації на основі заданих параметрів. Інформація з бази даних використовується для покращення точності та адаптації алгоритмів.

Крім того, важливо реалізувати механізми моніторингу та управління, які дозволять системі автоматично коригувати свої алгоритми на основі зворотного зв'язку від користувачів. Це може включати опитування користувачів про задоволеність отриманими рекомендаціями, що дозволить проводити аналітику та вдосконалювати систему.

Загальна архітектура системи повинна бути модульною, що дозволить легко додавати нові функції та можливості, а також забезпечить гнучкість у вдосконаленні системи в майбутньому. Проектуючи архітектуру, необхідно

також врахувати питання масштабованості, оскільки зростання кількості користувачів та даних вимагатиме адаптації системи до нових умов.

6.2 Алгоритм роботи системи

Алгоритм роботи системи визначає послідовність кроків і логічних операцій, які необхідні для обробки вхідних даних, аналізу умов, і видачі рекомендацій для оптимального передатного числа. Основна мета цього алгоритму полягає в забезпеченні високої точності результатів та ефективної обробки даних, що враховують індивідуальні параметри користувача та умови маршруту. Детальний опис етапів алгоритму допоможе зрозуміти, як система обробляє інформацію та формує рішення.

6.2.1 Перший етап: Отримання вхідних даних. На початку роботи система отримує вхідні дані від користувача, а також доступні дані про маршрут. Ці дані можуть включати:

- фізичні параметри велосипедиста**:** зріст, вага, рівень підготовки, а також особливості техніки їзди;
- характеристики маршруту: дані про нахил траси, довжину ділянок, а також тип покриття (асфальт, гравій тощо), отримані через API або з наявної бази;
- зовнішні умови: температура, вологість повітря, швидкість вітру, що можуть впливати на опір руху.

Для збору даних використовуються запити до бази даних і спеціальних API (наприклад, API для маршруту). На цьому етапі важливо забезпечити коректність введених користувачем значень.

6.2.2 Другий етап: Попередня обробка даних. Після отримання вхідних даних система здійснює попередню обробку для стандартизації та фільтрації

інформації. Цей етап включає:

- нормалізацію даних: усі параметри приводяться до єдиних одиниць вимірювання для коректності подальших обчислень;
- фільтрацію: видалення або корекція некоректних значень (наприклад, дуже малих чи великих значень), що можуть спотворити результати;
- стандартизацію вхідних величин**:** для того, щоб уникнути відмінностей у форматах даних з різних джерел, система конвертує дані в стандартизовані формати (наприклад, градуси для кутів нахилу, км/год для швидкості);

Ця обробка підвищує точність аналізу та зменшує кількість можливих помилок у наступних етапах обчислень.

6.2.3 Третій етап: Аналіз даних і розрахунок показників. На основі оброблених даних система починає основний етап аналізу. Використовуючи алгоритми машинного навчання та нечіткої логіки, система обчислює оптимальні параметри для руху. Цей етап складається з декількох основних кроків:

1. Розрахунок фізичних показників**:** включає обчислення показників енерговитрат, зусилля та швидкості. Відповідно до введених даних, система визначає оптимальний рівень опору, що повинен бути врахований у розрахунках.
2. Аналіз характеристик траси**:** нахил траси, тип покриття та інші параметри аналізуються для визначення умов, за яких буде проходити рух. Це допомагає системі виділити ділянки, що вимагають додаткового зусилля або навпаки, полегшують рух.
3. Визначення оптимального передатного числа**:** обчислення коефіцієнтів на основі нечітких змінних, що відповідають фізичним і дорожнім характеристикам.

На кожному з кроків обчислення враховуються умови, що найбільше впливають на швидкість, маневреність і комфорт їзди для велосипедиста.

6.2.4 Четвертий етап: Оптимізація на основі постійного аналізу даних.

На цьому етапі система здійснює додатковий аналіз зібраних даних для покращення точності рекомендацій та налаштувань. Вона використовує результати попередніх розрахунків та зворотний зв’язок від користувача для оптимізації алгоритмів і параметрів. Основні кроки включають:

1. Аналіз історичних даних**:** система зберігає інформацію про попередні розрахунки, що допомагає виявити закономірності та тенденції в різних умовах.
2. Коригування налаштувань на основі зворотного зв’язку**:** при повторному використанні система бере до уваги нові дані про ефективність попередніх рекомендацій і коригує параметри для кращої адаптації.
3. Покращення логічних правил**:** алгоритм оновлює правила для розрахунків передаточного числа та інших параметрів на основі змін у параметрах траси та ефективності попередніх рекомендацій.

Цей етап дозволяє системі адаптуватися до змінних умов та потреб користувачів, використовуючи накопичені дані для точніших рекомендацій без застосування методів машинного навчання [16].

6.2.5 П'ятий етап: Зворотній зв’язок і адаптація системи. Щоб підвищити точність майбутніх прогнозів, система використовує дані зворотного зв’язку, які збираються під час реального використання. Збір цих даних дозволяє алгоритмам системи:

1. Вносити корективи в базу знань**:** додавання нових правил на основі отриманих даних дозволяє системі покращити ефективність у наступних розрахунках.
2. Здійснювати самоадаптацію**:** система автоматично налаштовується на різні типи траси та характеристики користувача, що підвищує ефективність та надійність роботи системи.

3. Оновлення даних маршруту**:** інформація про часті маршрути та типові умови траси допомагає автоматично підлаштовуватися до цих умов у подальших аналізах.

6.3 Використання нечітких множин для обчислення передатного числа

Нечіткі множини виступають важливим компонентом для обчислення оптимального передатного числа у системах, де важливо враховувати широкий спектр факторів, які мають розмиті значення, що постійно змінюються. На відміну від традиційних методів, які потребують точних вхідних даних, нечітка логіка дозволяє обробляти нечіткі, приблизні або розмиті значення [17], враховуючи різні комбінації умов дороги, фізичні характеристики велосипедиста, тощо.

Проектування ефективної системи на основі нечіткої логіки для розрахунку оптимального передатного числа є важливим кроком, який потребує ретельного підходу до вибору параметрів і формулювання правил. Параметри та правила визначають, як система аналізуватиме вхідні дані та даватиме рекомендації для кожного конкретного випадку. Правильно налаштовані параметри забезпечують адекватне відображення факторів, що впливають на їзду, тоді як правила визначають, як ці фактори перетворюються на рішення щодо вибору передатного числа [22].

Таблиця 6.1 – Параметри класичних та нечітких множин

Класичний комплект	Теорія нечітких множин
Класи об'єктів з різкими межами	Класи об'єктів не мають різких меж

Продовження таблиці 6.1

Класичний комплект	Теорія нечітких множин
Класичний набір визначається чіткими межами, тобто існує ясність щодо розташування меж набору.	Нечітка множина завжди має неоднозначні межі, тобто може існувати невизначеність щодо розташування меж множини
Широко використовується в проектуванні цифрових систем	Використовується тільки в нечітких контролерах.

6.3.1 Визначення основних параметрів нечіткої системи. Першим етапом у створенні системи є вибір параметрів, що суттєво впливають на їзду велосипедом. В основі системи, побудованої на нечіткій логіці, лежить набір параметрів, кожен з яких впливає на обчислення передатного числа. Вони включають:

- нахил траси (slope): кут нахилу поверхні, що безпосередньо впливає на необхідне зусилля для педалювання. Один із ключових факторів, оскільки нахил впливає на силу, необхідну для просування велосипеда вперед. Нахил визначається на основі даних висотних точок на маршруті;
- швидкість руху (speed**)**: залежить від фізичних можливостей велосипедиста, типу велосипеда і профілю дороги. Поточна або бажана швидкість велосипедиста впливає на вибір передатного числа. Для підтримки високої швидкості може знадобитися інше передатне число, ніж для повільної їзди, особливо на складних ділянках маршруту;
- тип дорожнього покриття (surface**)**: важливий фактор, який визначає ефективність педалювання. Стан дорожнього покриття прямо впливає на вибір передачі. Наприклад, на асфальті ефективність вища, ніж на ґрунті чи гравію на асфальті велосипедист може зберігати високу швидкість з меншими зусиллями, тоді як на грунтових або кам’янистих дорогах виникає більший опір;

- фізичні характеристики велосипедиста та велосипеда: зокрема, вага велосипедиста, тип велосипеда та передбачуваний стиль їзди (агресивний або спокійний. Особливості фізичної підготовки, такі як сила і витривалість, можуть визначати здатність велосипедиста працювати на певних передачах за різних умов.

6.3.2 Формування нечітких множин для параметрів. Кожен параметр в системі необхідно представити у вигляді нечітких множин, що описують його характеристики у вигляді функцій належності [23].

Це дозволяє створювати діапазони значень, які характеризують можливі стани параметрів у нечіткій логіці [18]. Це забезпечує гнучкість та адаптивність аналізу, враховуючи невизначеність і варіативність умов. Наприклад:

- нахил траси: для нього можуть бути визначені такі нечіткі множини, як *пологий, середній нахил* та *крутий нахил* з урахуванням різних значень градусів. Кожна множина може бути визначена на основі даних про висоту маршруту з різними градаціями, що дозволяє гнучко відображати фізичну реальність нахилу. Це дозволяє моделювати траси з різним рівнем складності;
- швидкість**:** значення швидкості можуть бути поділені на множини *низька швидкість*, *середня швидкість* і *висока швидкість*. Кожна категорія має діапазон значень, що дозволяє враховувати різні режими руху. Усі функції належності розробляються таким чином, щоб плавно змінюватися, надаючи систему гнучкість у прийнятті рішень;
- дорожнє покриття: можна визначити нечіткі множини, які відображають різні типи покриттів, наприклад, *гладке покриття, гравійне покриття* або *грунтове покриття.* Для представлення типу дорожнього покриття використовуються відповідні нечіткі множини, що описують гладкість поверхні. Множини "асфальт", "грунт" і "камені" можуть мати градації належності в залежності від рівня впливу покриття на їзду. Значення належності в цьому випадку можуть коригуватися на основі оцінок велосипедистів або зовнішніх даних про маршрути;

- пульс: для пульсу можуть бути визначені нечіткі множини, такі як *низький пульс*, *середній пульс* та *високий пульс*. Це дозволяє врахувати різні фізіологічні стани велосипедиста, що допомагає коригувати інтенсивність поїздки в залежності від пульсу;
- потужність: значення потужності можуть бути розподілені на множини *низька потужність*, *середня потужність* та *висока потужність*. Це дозволяє оптимізувати вибір передачі в залежності від поточної енергії велосипедиста;
- каденс: для каденсу визначаються нечіткі множини *низький каденс*, *середній каденс* та *високий каденс*, що дозволяє адаптувати рекомендації залежно від стилю їзди — наприклад, для підйомів або швидких спусків;
- сумарний набір висоти: для цього параметра можна створити множини *малий набір висоти*, *середній набір висоти* та *великий набір висоти*, що допомагає врахувати висоту, яку велосипедист піднімався протягом маршруту.
- для рівнів підготовки система може мати значення для параметру, як "початківець", "просунутий" і "експерт", що впливатиме на підбір передач залежно від особистих фізичних можливостей;
- Параметр погодних умов може включати такі множини, як "сприятливі", "помірно несприятливі" і "несприятливі" погодні умови, що впливають на вибір передатного числа. Всі ці множини відображають вплив погодних факторів на оптимальний режим їзди, враховуючи комфорт та безпеку велосипедиста.

Кожна з цих множин описується функцією належності, яка показує, наскільки певне значення належить до тієї чи іншої множини. Наприклад, пульс 140 ударів на хвилину може частково належати до множини *середній пульс* (з належністю 0.7) і частково до *високого пульсу* (з належністю 0.3). Це дозволяє використовувати наближені значення для визначення оптимального передатного числа, що відповідає поточному фізичному стану велосипедиста.

Такий підхід дозволяє комбінувати ці параметри для визначення найбільш ефективного передатного числа, враховуючи не тільки фізичні показники, а й зовнішні умови маршруту, що дозволяє отримати точніші та індивідуалізовані рекомендації для кожного велосипедиста.

6.3.3 Побудова правил нечіткої логіки. Основним завданням системи є використання нечіткої логіки для розробки правил, які визначають вплив кожного з параметрів на кінцевий результат [19]. Після визначення нечітких множин для кожного параметра наступним кроком є формулювання правил, що описують взаємозв'язок між цими параметрами для вибору оптимального передатного числа [24]. Ці правила формуються у вигляді логічних умов, які можна інтерпретувати як:

1. Використання правил типу "Якщо-Тоді"**:**

- якщо нахил низький **і** покриття гладке, **то** вибираємо високе передатне число;
- якщо нахил високий **і** швидкість низька, **то** вибираємо низьке передатне число для полегшення педалювання;
- якщо покриття гравійне **і** швидкість середня, **то** вибираємо середнє передатне число.

Це правило забезпечує належну рекомендацію для велосипедиста в конкретних умовах.

У системі зазвичай створюється багато таких правил для всіх можливих комбінацій параметрів.

2. Приклади складних правил**:**

Складніші правила можуть включати кілька умов і враховувати взаємозалежність параметрів. Наприклад: "Якщо швидкість висока і нахил середній, але покриття грунтове, то передатне число середнє". Це правило є прикладом складнішого логічного зв’язку між параметрами.

Такі правила формуються на основі аналізу реальних сценаріїв і дозволяють системі адаптуватися до комбінованих умов на маршруті.

3. Аналіз конфліктних ситуацій у правилах**:**

У деяких випадках може виникати конфлікт між правилами, коли одне правило рекомендує одне передатне число, а інше — інше. У таких випадках система може використовувати методи агрегації чи пріоритетності для вирішення конфліктів.

Наприклад, якщо правила для "крутого нахилу" і "високої швидкості" конфліктують, система може дати пріоритет передатному числу, яке забезпечує безпечнішу їзду.

4. Оптимізація правил на основі зворотного зв’язку**:**

Система може вдосконалювати правила на основі аналізу даних минулих поїздок. Якщо певні комбінації параметрів повторюються часто, це може вказувати на необхідність удосконалення певного правила для підвищення точності.

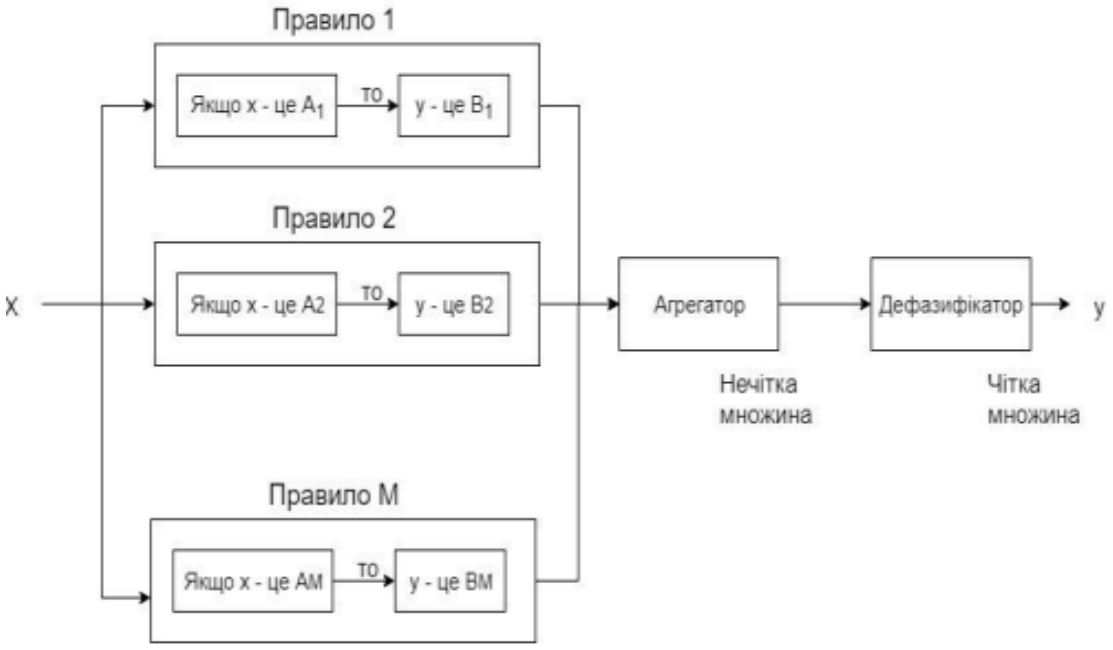

Рисунок 6.1 – Система нечіткого логічного висновку

Ці правила дозволяють алгоритму адаптуватися до змін у середовищі в реальному часі та надавати рекомендації, що враховують актуальні умови.

Таким чином, вибір параметрів і формулювання правил є важливими складовими проектування системи. Завдяки грамотному підходу до цього етапу можна створити адаптивну та ефективну систему, здатну надавати оптимальні рекомендації навіть у складних умовах маршруту.

6.3.4 Моделювання функцій належності. Для кожної нечіткої множини визначаються функції належності, які відображають ступінь належності конкретного значення до нечіткої множини [21]. Наприклад, функції належності для швидкості можуть виглядати як трикутні або трапецієподібні графіки, що визначають ступінь належності значень до множин *низька*, *середня* та *висока* швидкість. Ці функції дозволяють враховувати поступові зміни параметрів,

уникаючи різких переходів, які можуть знижувати ефективність системи.

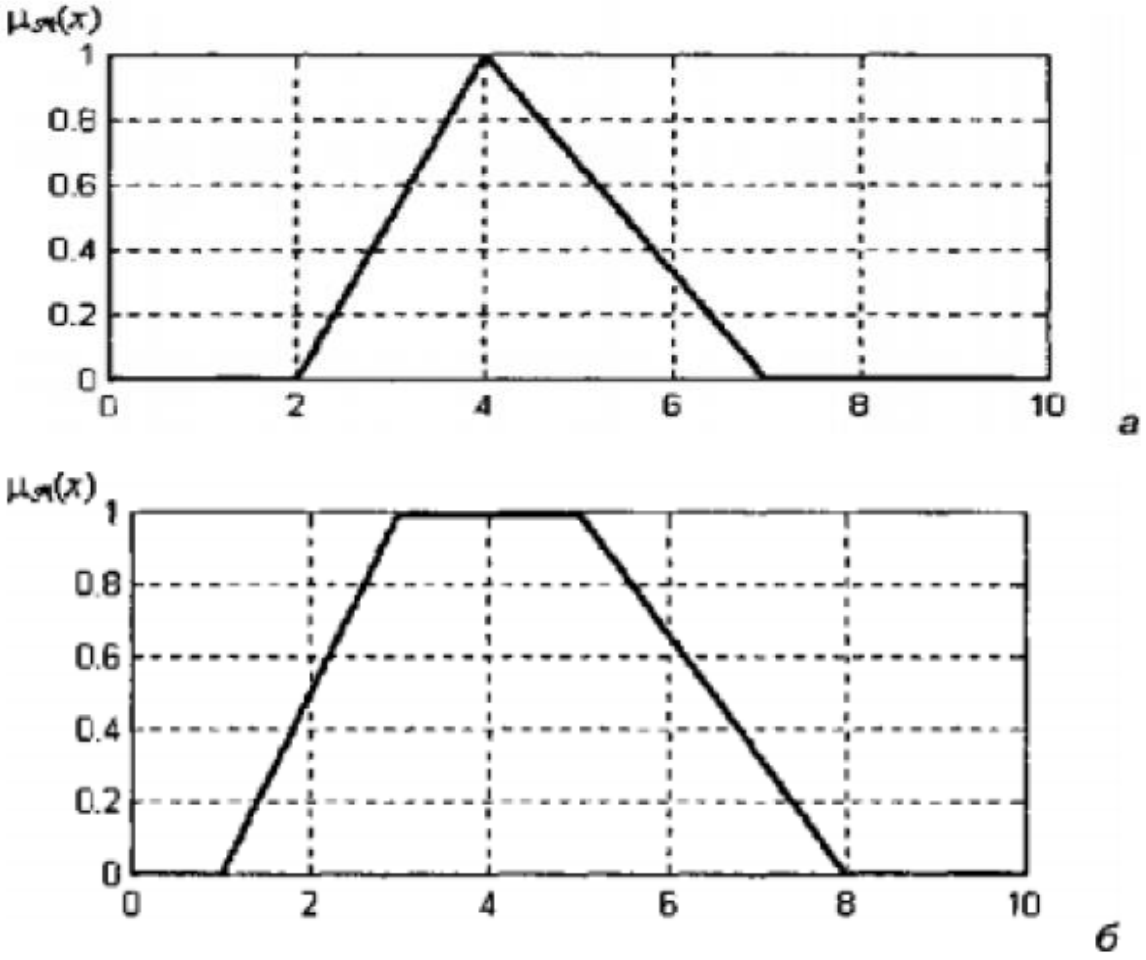

Рисунок 6.2 – Графіки функцій приналежності трикутної (а) та трапецевидної (б) форми

Загалом є різні види функцій приналежності. Особливістю трапецеїдальної форми є те, що існує певна кількість значень “1” та два значення “0”. Трикутна форма є окремим випадком трапецеїдальної, але має лише одне значення - “1”. Проте ці функції мають два проміжки: на одному функція зростає, а іншому садає. Z-подібна функція може бути спадаючою або зростаючою. Зростаюча S-функція є дзеркальною до Z. Ці функції приналежності фактично представляють собою значення, які може приймати фізична величина.

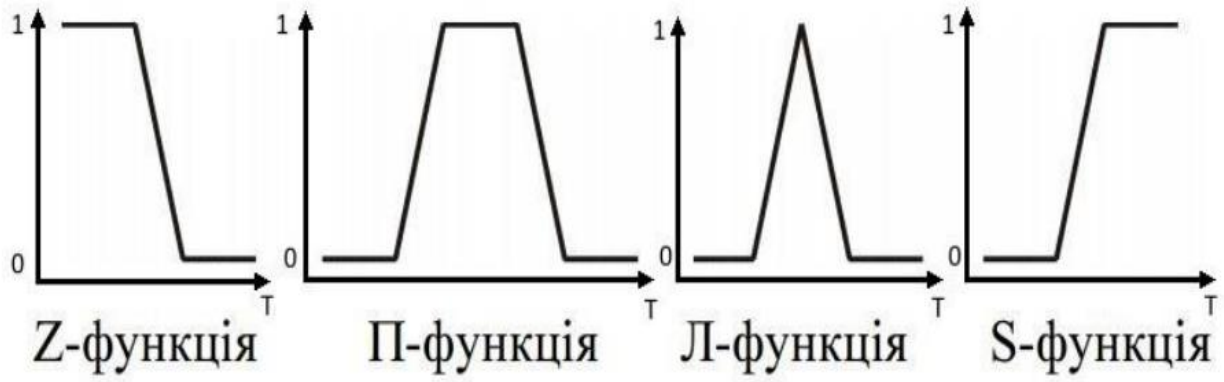

Рисунок 6.3 – Види функції приналежності

6.3.5 Дефазифікація та отримання результату. Після застосування правил нечіткої логіки система отримує множину нечітких значень для передатного числа, які повинні бути перетворені у конкретне числове значення. Цей процес називається дефазифікацією.

- метод центру тяжіння**:** визначає середньозважене значення для всіх активованих функцій належності, що забезпечує баланс між усіма параметрами.

$$\frac{z * \int \mu(z)dz}{\int z \cdot \mu(z)dz}$$

де: z — потенційні значення для передатного числа,

μ(z) — значення функції належності для кожного значення z;

- метод середнього максимуму: використовується для дефазифікації в нечітких системах, коли потрібно отримати чітке значення на основі функцій належності нечіткої множини. Зокрема, цей метод визначає середнє значення всіх точок, у яких функція належності досягає свого максимального значення.

Формально це можна записати так:

$$\frac{z * \sum z \in Zmaxz}{Zmax \vee}$$

де: z∗ — дефазифіковане значення, отримане методом середнього максимуму,

Zmax — множина всіх значень z, де функція належності μ(z) досягає свого максимуму,

|Zmax| — кількість елементів у множині Zmax.

Обраний метод дефазифікації забезпечує точність кінцевого результату, роблячи систему адаптивною та гнучкою.

6.3.6 Переваги використання нечітких множин у системі вибору передатного числа. Використання нечітких множин має низку переваг, зокрема:

- адаптивність**:** система легко пристосовується до змінюваних умов дороги і параметрів користувача, забезпечуючи індивідуальний підхід до кожного випадку;
- гнучкість у налаштуваннях: правила та функції належності можуть бути змінені відповідно до конкретних вимог або нових умов, що розширює можливості системи;
- ефективне опрацювання нечітких даних: система здатна обробляти нечіткі значення, що робить її ефективною у випадках, коли точні дані важко отримати або вони мають велику похибку.

Використання нечітких множин дозволяє створити адаптивну систему, яка здатна забезпечити оптимальний підбір передатного числа для різних умов їзди, враховуючи індивідуальні потреби та параметри кожного велосипедиста.

6.4 Технологічні рішення для розробки системи

Для реалізації системи вибору оптимального передатного числа на основі нечіткої логіки необхідно вибрати відповідний стек технологій.

Python: основна мова для реалізації нечіткої логіки, алгоритмів обробки даних та математичних обчислень завдяки великій кількості бібліотек (NumPy, SciPy, Fuzzy Logic Toolkit) [20]. Мультипарадигменна мова програмування, стандартна бібліотека включає великий набір корисних функцій, що переносяться, починаючи з можливостей для роботи з текстом і до засобів для написання мережевих додатків.

OSM API — це REST-інтерфейс для взаємодії з базою даних OpenStreetMap. Він дозволяє здійснювати читання та запис даних за допомогою стандартних HTTP-методів та простих URL-адрес. Комунікація відбувається через протокол HTTP із використанням XML-формату для передачі даних.

Основні характеристики:

- HTTP-методи:
- GET — для отримання даних.
- POST — для створення нових даних.
- PUT — для оновлення існуючих даних.
- DELETE — для видалення даних.
- URL-структура: Кожен ресурс (точка, лінія або область) доступний через унікальну URL-адресу, що спрощує роботу з даними.
- Доступ через Інтернет: API дозволяє здійснювати підключення до бази даних OSM через мережу Інтернет, забезпечуючи глобальний доступ до картографічних даних.

База даних: основна база даних – стрижневий компонент OpenStreetMap, у базі даних OSM зберігаються всі географічні об'єкти у вигляді:

- Точок (nodes).
- Ліній (ways).
- Областей (relations).

База даних має таблиці для кожного типу елементів даних (точок, ліній, зв'язків). Насправді кожний тип даних має кілька таблиць для збереження інформації: таблиця з поточними даними, таблиця з історичними даними, таблиця поточних

теґів, таблиця з історичними теґами. Крім того, існують таблиці для зберігання даних наборів змін, файлів gpx, профілів користувачів, записів щоденника, поточних сеансів, OAuth тощо.

Доступ до бази даних для редагування даних надається засобами API. Цей API використовується іншими компонентами OSM для роботи з картографічними даними.

Таблиця 6.2 – Технології для реалізації системи

Назва	Тип	Мета використання	Призначення
Python	Мова програмування	Реалізація нечіткої логіки, алгоритмів обробки даних та математичних обчислень	Створення системи
Jquery та jQuery UI	Бібліотека	Створення інтерфейсу користувача	Допоміжні інструменти для створення зручного інтерфейсу користувача
JavaScript	Мова програмування	Створення інтерфейсу користувача	Обробка подій в реальному часі
Flask	Фреймворк	Розробаи бекенд-архітектури та API	Отримання та обробка даних з GPS-трекерів та інших зовнішніх джерел
NumPy	Інструмент для обробки даних	Аналіз та обробка великих масивів даних	Координати GPS, профіль висот та інформація про покриття
Fuzzy Logic Toolkit	Бібліотека	Реалізація системи нечіткої логіки	Створення нечітких множин, правил та функції належності для обчислень
OpenStreet Map API	Сервіс для інтеграції з зовнішніми	Отримання інформації про тип дорожнього покриття, профіль	Використання даних для розрахунку

	джерелами	місцевості та інші географічні дані	

Крім того в даному завдані застосовано підхід, в якому виключено просторове індексування і замість цього обробляються запити простим групуванням координат в пакети і використанням паралельних запитів.

Така пакетна обробка з паралельним виконанням для отримання геопросторових даних можна назвати евристичною оптимізацією, хоча вона не є строгою оптимізацією за алгоритмом, так як не використовуються складні структури індексування, а розрахунки покладаються на просту пакетну обробку та паралельне виконання, роблячи розумний компроміс між складністю та продуктивністю.

Загалом цей метод базується на таких пов'язаних концепціях:

1. Пакетна обробка (Batch processing): це процес обробки даних у пакетах, при якому координати групуються в шматки перед запитом до API Overpass. Пакетна обробка є відомою оптимізацією, особливо при роботі з великими наборами даних, оскільки вона зменшує накладні витрати на обробку кожного елемента окремо.

2. Просторові запити без індексації: хоча просторові індекси (як R-tree або Quad-tree) є типовою технікою оптимізації для геопросторових даних, застосований метод представляє собою запити без використання таких індексів, що можна описати як наївне просторове запитування. Цей підхід може бути ефективним для менших наборів даних, але може зіткнутися з проблемами продуктивності при збільшенні їх обсягу.

3. Конкурентна або паралельна обробка: використання ThreadPoolExecutor і паралельних запитів – це форма павралельної обробки задач, зокрема конкурентна обробка для задач, пов'язаних з введенням / виведенням (I / O bound tasks). Це корисно коли програма чекає на зовнішні запити і дозволяє виконувати кілька запитів одночасно, покращуючи продуктивність без блокуванняосновного потоку.

Висновок: детально розглянуто можливість використання нечітких множин для вибору оптимального передатного числа приїзді на одношвидкісному велосипеді. Визначено ключові параметри, сформульовано правила на їх основі. Розроблено архітектуру системи, взаємозв'язки її компонентів. Визначено відповідний стек технологій, що забезпечить створення, інтеграцію з зовнішніми джерелами, коректну роботу адаптивної систему, яка здатна забезпечити оптимальний підбір передатного числа для різних умов їзди, враховуючи індивідуальні потреби та параметри кожного велосипедиста системи.

ВИСНОВОКИ

Було проведено детальний аналіз проблеми вибору оптимального передатного числа для одношвидкісних та фіксованих велосипедів, що базується на використанні нечіткої логіки та даних з GPS та GPX. Актуальність дослідження обумовлена зростаючою популярністю одношвидкісних та фіксованих велосипедів серед велосипедистів, та відповідним запитом від починаючих велосипедистів без власного досвіду на визначення найкращого співвідношення під їх потреби.

У процесі роботи було визначено ключові фактори, що впливають на вибір передатного числа, такі як фізичні параметри велосипедиста, особливості траси, стан дорожнього покриття та бажана швидкість. Це дозволило створити комплексний підхід до аналізу, що включає врахування багатьох змінних факторів.

Запропонована система на основі нечіткої логіки має значні переваги, оскільки вона дозволяє адаптуватися до різноманітних умов та індивідуальних потреб користувачів, створюючи алгоритми, які можуть надавати рекомендації щодо оптимального передатного числа на основі реальних даних.

У підсумку, робота показала, що існуючі рішення не завжди враховують специфіку використання одношвидкісних велосипедів, а також не надають користувачам глибокого розуміння впливу вибору передачі на їхню їзду. Розроблена система має потенціал для подальшого розвитку та вдосконалення, вона може стати основою для створення інтуїтивно зрозумілого інструменту, який допоможе велосипедистам приймати обґрунтовані рішення в виборі співвідношення для велосипеду, підвищуючи якість їхнього досвіду на дорозі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. A Fixed-gear bicycle. URL: https://en.wikipedia.org/wiki/Fixed-gear_bicycle (дата звернення: 20.10.2024).

2. The fixed gear - history, culture and philosophy. URL: https://locabikes.com/blog/blog/the-fixed-gear-history-culture-and-philosophy (дата звернення: 20.10.2024).

3. Graeme Obree. Flying Scotsman : Cycling to Triumph Through My Darkest Hours., 2005. 120-200с.

4. Single Speed Gearing. URL: https://www.peterverdone.com/wiki/index.php?title=Single_Speed_Gearing (дата звернення: 22.10.2024).

5. ДБН В.2.3-4:2015 "Автомобільні дороги. Частина І. Проектування Частина ІІ. Будівництво". [Чинний від 01.04.2016]. Вид. офіц. Київ: Міністерство регіонального розвитку, будівництва та житлово-комунального господарства України, 2015.

6. George J. Klir. Fuzzy Sets And Fuzzy Logic. Pearson College Div, 1995.283с.

7. James K. Peckol, Introduction to fuzzy logic. John Wiley & Sons Ltd., 2021. 306с.

8. Fuzzy Logic - Introduction. URL: https://www. geeksforgeeks. org/ fuzzy-logic-introduction/ (дата звернення: 27.10.2024).

9. Bellman R.E., Zadeh L.A. Decision-making in fuzzy environment // Management Science. – 1970. –. Vol. 17. – № 4. – PP. 141-160

10. Бідюк П.І., Коршевнюк Л.О.. Проектування комп'ютерних інформаційних систем підтримки прийняття рішень: посібник. Київ: ННК «ІПСА» НТУУ, «КПІ», 2-10, 340с

11. Mamdani, E.H., Assillan, S.: An experiment in linguistic synthesis with a fuzzy logic controller.Int. J. Man-Mach. Stud. 7(1), 1–13, 197

12. Нечіткі методи в економічних розрахунках. URL: https://pzs.dstu.dp.ua/logic/tasks.html

13. Олізаренко С. А., Перепелиця А. В., Капранов В. А. Нечіткі логічні системи інтервального типу 2. Архітектура і механізм виводу // Системи обробки інформації. 2011. №5 (95). С. 156-164

14. Lee, C.C.: Fuzzy logic in control systems: Fuzzy logic controller - part I and II. IEEE Trans.Syst., Man Cybern. 20(2), 404–435, 1990

15. Jang, J.-S.R., Sun, C.-T.: Neuro-fuzzy modelling and control. Proc. IEEE 83, 378–406, 1995.

16. David J.M., Krivine J.P., Simmons R. Second generation expert systems: a step forward in knowledge engineering // Second Generation Expert Systems.Springer-Verlag, 1993. P. 3–25.

17. Mouzouris, G.C., Mendel, M.J.: Dynamic non-singleton fuzzy logic systems for nonlinear modeling. IEEE Trans. Fuzzy Syst. 5(2), 199–208, 199.

18. Preece A.D., Shinghal R., Batarekh A. Verifying expert systems: a logical framework and a practical tool // Expert systems with applications. 1992. Vol. 5. P. 421–436.

19. Митюшкін Ю.І., Мокін Б.І., Ротштейн А.П. Soft Computing: ідентифікація закономірностей нечіткими базами знань. – Вінниця: УНІВЕРСУМ-Вінниця, 2002. – 145 с.

20. Методичні вказівки до занять з курсу “Нечіткі моделі та методи” / Редактор О. С. Самініна; м. Харків: НТУ ХПІ, 2012, 48с.

21. Fuzzy Inference System implementation in Python. URL: https://towardsdatascience.com/fuzzy-inference-system-implementation-in-python-8af88d1f0a6e (дата звернення: 27.10.2024).

22. Ротштейн О. П. Інтелектуальні технології ідентифікації: нечіткі множини, генетичні алгоритми, нейронні мережі. Вінниця : Універсум-Вінниця, 1999. 320 с.

23. Камінський В.В., Мокін Б.І. Вступ до теорії слабких множин: монографія. Вінниця, ВНТУ, 2012, 128С.

24. Wang L. X., Mendel J. M. Generating Fuzzy Rules from Numerical Data, with Applications // Signal and Image Processing Institute, University of Southern California, Department of Electrical Engineering-Systems, 1991.63 p.

ДОДАТОК А: ЛІСТИНГ КОДУ

```
<!DOCTYPE html>

<html lang="en">
<head>
<title>Ratio Master</title>
<meta name="viewport" content="width=465; user-scalable=no">
<link type="text/css" href="{{ url_for('static', filename='css/styles.css') }}" rel="stylesheet" media="screen">
<link type="text/css" href="{{ url_for('static', filename='css/jquery-ui.min.css') }}" rel="stylesheet">
<link rel="icon" href="{{ url_for('static', filename='images/favicon.ico') }}" type="image/x-icon">
<script type="text/javascript" src="{{ url_for('static', filename='js/jquery-3.7.1.min.js') }}"></script>
<script type="text/javascript" src="{{ url_for('static', filename='js/jquery-ui.min.js') }}"></script>
<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
<script type="text/javascript" src="{{ url_for('static', filename='js/values.js') }}"></script>
<script type="text/javascript" src="{{ url_for('static', filename='js/index.js') }}"></script>
</head>
<body>
<div id="loading-overlay"><div class="spinner"></div><span data-translate="loading">Loading, please wait, it may take few minutes...</span></div>
<div id="results-overlay" title="Analysis Results">
<div class="results-content">
<p class="results-message">Data is loaded and analyzed!</p>
<p class="results-message">Recommended gear ratio is: <span id="recommended-gear-ratio" class="highlighted-text"></span></p>
<p class="results-message success-message">Gear ratio is set to recommended!</p>
<canvas id="resultsChart"></canvas>
</div>
</div>
<header>
<span>
<a href="/" title="Ratio Master"><img src="{{ url_for('static', filename='images/logo.png') }}" alt="RATIO MASTER"></a>
</span>
<span id="settings">
<label for="language-selector">
<select id="language-selector" name="lang">
<option value="en" selected="selected">  English</option>
<option value="ua">  Ukrainian</option>
</select>
</label>
<span class="separator"></span>
<span class="clickable">About</span>
</span>
</header>
<main>
<div id="calc" class="center-column">
<div id="form">
<div class="center-column">
```

```
<h3 style="margin-bottom: 0px;" data-translate="measurement-system">Measurement system</h6>
<div style="padding: 0.66rem">
<label for="u2"><span class="multi" data-translate="metric">metric</span><input id="u2" type="radio" name="unit"
value="m" checked="checked"></label>
<label for="u1"><span class="multi" data-translate="imperial">imperial</span><input id="u1" type="radio"
name="unit" value="i"></label>
</div>
</div>
<div class="values">
<label for="chainring" class="multi" data-translate="chainring">Chainring</label>
<select id="chainring"></select>
<label for="sprocket" class="multi" data-translate="sprocket">Rear sprocket</label>
<select id="sprocket"></select>
<label for="tire" class="multi" data-translate="tire">Tire</label>
<select id="tire"></select>
</div>
</div>
<div id="upload-container">
<h3 style="margin-top: 6px;" data-translate="upload-gpx">Upload GPX files</h3>
<p data-translate="upload-gpx-desc">Upload your GPX files to find gear ratio that suits your rides!</p>
<span>
<input type="file" id="gpxFiles" multiple accept=".gpx" />
<label for="gpxFiles" class="upload-label" data-translate="chose-gpx">Choose GPX files</label>
</span>
<div id="fileList" class="file-list"></div>

<button id="submitGpx" type="button" class="submit-button" data-translate="upload">Submit</button>
</div>

<h3 class="multi" data-translate="analysis">Analysis</h3>
<div>
<span>
<span data-translate="ratio">Ratio: </span><span id="ratio" style="font-weight: bold;"></span>
</span>
<span class="separator"></span>
<span>
<span data-translate="gear-inches">Gear inches: </span><span id="gear-inches" style="font-weight: bold;"></span>
</span>
<span class="separator"></span>
<span>
<span data-translate="d-per-pedal">Distance per pedals rotation: </span><span id="development" style="font-
weight: bold;"></span>
</span>
</div>
</div>
<div id="more">
<table id="speeds-table">
<thead>
```

```
<tr>
<th data-translate="riding-situation">Riding Situation</th>
<th data-translate="cadence-range">Cadence Range (RPM)</th>
<th data-translate="average-speed">Average Speed <span id="avg-speed-unit">(km/h)</span></th>
</tr>
</thead>
<tbody id="speeds-table-body">
<!-- Rows will be populated dynamically -->
</tbody>
</table>
<h3 class="multi" data-translate="simmilar-gears">Simmilar gears</h3>
<div>
<table id="simmilar-ratios-table">
<tbody>
<!-- Rows will be populated dynamically -->
</tbody>
</table>
</div>
</div>
</main>
</body>
</html>
```

```
let translations = []

let resultsChart;

$(document).ready(function() {
$('#loading-overlay').fadeOut();
calculate();
loadTranslations('en');

$('#language-selector').change( function(val) { loadTranslations(val.currentTarget.value); });
$('#chainring').change( function() { calculate(); });
$('#sprocket').change( function() { calculate(); });
$('#tire').change( function() { calculate(); });
$('#ambidextrous').change( function() { calculate(); });
$('[name=unit]').change( function(val) {
calculate();
$('#avg-speed-unit').html(val.currentTarget.value == 'm' ? '(km/h)' : '(mph)');
});

$("#results-overlay").dialog({
width: 600,
height: 600,
modal: true,
autoOpen: false,
draggable: false,
resizable: false,
```

```
position: { my: "center", at: "center", of: window },
open: function() {
$(this).parent().css({
position: "fixed",
top: "50%",
left: "50%",
transform: "translate(-50%, -50%)"
});

$(".ui-dialog-titlebar-close").html("&#10005;");
}
});
$("#results-overlay").dialog("close");

$("#close-dialog").click(function() {
$("#results-overlay").dialog("close");
});

$('#gpxFiles').on('change', function () {
const files = $(this).prop('files');
const fileList = $('#fileList');
fileList.empty();

if (files.length) {
$.each(files, function (index, file) {
fileList.append(`<div>${file.name}</div>`);
});
} else {
fileList.append('<div>No files selected.</div>');
}
});

$('#submitGpx').on('click', function() {
const formData = new FormData();
const files = $('#gpxFiles').prop('files');

if (files.length === 0 && stravaLinks.length === 0) {
alert('Please add GPX data to upload.');
return;
}

$('#loading-overlay').fadeIn();
$.each(files, function(index, file) {
formData.append('files', file);
});
formData.append('wheel_circumference', $('#tire').val());
$.ajax({
url: '/upload_gpx',
type: 'POST',
data: formData,
```

```
processData: false,
contentType: false,
success: function(response) {
$('#loading-overlay').fadeOut();
$("#results-overlay").dialog("open");
updateResultsChart(response.data);
$("#recommended-gear-ratio").text((response.optimal_gear_ratio[0]/response.optimal_gear_ratio[1]).toFixed(2));
$('#chainring').val(response.optimal_gear_ratio[0]);
$('#sprocket').val(response.optimal_gear_ratio[1]);
calculate();
},
error: function(xhr, status, error) {
alert('Error uploading files: ' + error);
$('#loading-overlay').fadeOut();
}
});
});
});

function calculateGCD(num1, num2) {
while (num2 !== 0) {
const remainder = num1 % num2;
num1 = num2;
num2 = remainder;
}
return num1;
}

function calculate() {
let chainringT = parseInt($('#chainring').val());
let sprocketT = parseInt($('#sprocket').val());
let unit = $("input[name='unit']:checked").val();
const ratio = Math.round(chainringT/sprocketT*100)/100;
const tire = Number($('#tire').val());
const tire_inches = tire / (Math.PI * 25.4);

$('#ratio').html(ratio);
$('#gear-inches').html(Math.ceil(ratio * tire_inches));

let thisFactor = 1;
let thisUnit = '';
if ( unit == "m" ) {
thisUnit = 'meters';
} else {
thisFactor = 0.0254;
thisUnit = 'inches';
}

let development = ratio * (tire/1000);
```

```
$('#development').html(Math.round(development*100/thisFactor)/100 + ' ' + thisUnit);

const situations = [
{ name: "Top speed", range: [120, 140] },
{ name: "Fast Riding", range: [100, 120] },
{ name: "Moderate Riding", range: [80, 110] },
{ name: "Casual Riding", range: [70, 90] },
{ name: "Hill Climbing", range: [40, 70] },
{ name: "Steep hill climbing ", range: [30, 50] },
{ name: "Long Distance Riding", range: [75, 85] },
{ name: "Off-Road Riding", range: [65, 85] },
];

$('#speeds-table-body').empty();
situations.forEach(situation => {
const [minCadence, maxCadence] = situation.range;
const averageCadence = (minCadence + maxCadence) / 2;

const speedKMH = (averageCadence * development / 1000) * 60;
const speedMPH = speedKMH * 0.621371;

const newRow = `
<tr>
<td>${situation.name}</td>
<td>${minCadence} - ${maxCadence}</td>
<td>${unit == "m" ? speedKMH.toFixed(2) : speedMPH.toFixed(2)}</td>
</tr>
`;
$('#speeds-table-body').append(newRow);
});

populateTable(ratio);
}

function generateGearCombinations(minRing, maxRing, minSprocket, maxSprocket) {
const combinations = [];
for (let ring = minRing; ring <= maxRing; ring++) {
for (let sprocket = minSprocket; sprocket <= maxSprocket; sprocket++) {
combinations.push({ ring, sprocket, ratio: (ring / sprocket).toFixed(2) });
}
}
return combinations;
}

function populateTable(currentRatio, tolerance = 0.05) {
const tableBody = $("#simmilar-ratios-table");
tableBody.empty();

const allCombinations = generateGearCombinations(28, 100, 11, 28);
```

```
const filteredCombinations = allCombinations.filter(({ ratio }) => {
return Math.abs(ratio - currentRatio) <= tolerance;
});

let row = $("<tr></tr>");
filteredCombinations.forEach(({ ring, sprocket, ratio }, index) => {
const cell = $(`<td title="${ratio}" onclick="changeGearRatio(${ring}, ${sprocket}, ${ratio})">${ring}x${sprocket}</td>`);
row.append(cell);

if ((index + 1) % 8 === 0) {
tableBody.append(row);
row = $("<tr></tr>");
}
});

if (row.children().length > 0) {
tableBody.append(row);
}
}

function changeGearRatio(ring, sprocket, ratio) {
$('#chainring').val(ring);
$('#sprocket').val(sprocket);
$('#ratio').text(ratio);
calculate();
}

function loadTranslations(lang) {
$.getJSON('static/locales/' + lang + '.json', function(translations) {
translations = translations;
$('[data-translate]').each(function() {
var key = $(this).data('translate');
$(this).text(translations[key]);
});
});
}

function updateResultsChart(data) {
const labels = ["Cadence", "Heart Rate", "Estimated Power", "Speed", "Surface Quality", "Elevation Gain"];
const dataValues = [
normalizeValue(data.avg_cadence, 0, 140),
normalizeValue(data.avg_heart_rate, 40, 200),
normalizeValue(data.avg_power, 0, data.avg_power > 500 ? data.avg_power : 500),
normalizeValue(data.avg_speed, 0, data.avg_speed_threshold),
normalizeValue(data.avg_surface, 0, 1),
normalizeValue(data.elevation_gain, 0, data.elevation_threshold)
];
const filteredLabels = labels.filter((_, index) => dataValues[index] !== 0);
const filteredData = dataValues.filter(value => value !== 0);
```

```
if (resultsChart) {
resultsChart.data.labels = filteredLabels;
resultsChart.data.datasets[0].data = filteredData;
resultsChart.update();
} else {
const ctx = document.getElementById("resultsChart").getContext("2d");
resultsChart = new Chart(ctx, {
type: "radar",
data: {
labels: filteredLabels,
datasets: [{
label: "Ride Metrics",
data: filteredData,
backgroundColor: "rgba(54, 162, 235, 0.2)",
borderColor: "rgba(54, 162, 235, 1)",
borderWidth: 1,
pointRadius: 0
}]
},
responsive: true,
maintainAspectRatio: true,
options: {
scales: {
r: {
min: 0,
max: 1,
ticks:{
display: false
}
}
},
scale: {
ticks: {
display: false,
beginAtZero: true
},
grid: {
display: false
},
angleLines: {
display: false
},
pointLabels: {
display: false
}
}
}
```

```
});
}
}

function normalizeValue(value, min, max) {
let returnValue = value ? (value - min) / (max - min) : 0;
if (returnValue > 1) returnValue = 1;
return value > 1 ? returnValue : value;
}

$(document).ready(() => {

let wheelSizes = [
{ label: "700x23 (622-23)", value: 2105 },
{ label: "700x25 (622-25)", value: 2110 },
{ label: "700x28 (622-28)", value: 2136 },
{ label: "700x32 (622-32)", value: 2155 },
{ label: "700x35 (622-35)", value: 2168 },
{ label: "700x38 (622-38)", value: 2180 },
{ label: "700x40 (622-40)", value: 2200 },
{ label: "700x42 (622-42)", value: 2224 },
{ label: "700x45 (622-45)", value: 2242 },
{ label: "700x50 (622-50)", value: 2270 },
{ label: "26x1.0 (559-25)", value: 1913 },
{ label: "26x1.5 (559-40)", value: 1948 },
{ label: "26x1.75 (559-47)", value: 2026 },
{ label: "26x2.0 (559-50)", value: 2050 },
{ label: "26x2.1 (559-54)", value: 2068 },
{ label: "26x2.2 (559-56)", value: 2075 },
{ label: "27.5x1.5 (584-40)", value: 2079 },
{ label: "27.5x2.0 (584-50)", value: 2120 },
{ label: "27.5x2.1 (584-54)", value: 2148 },
{ label: "27.5x2.2 (584-56)", value: 2170 },
{ label: "29x1.9 (622-48)", value: 2250 },
{ label: "29x2.0 (622-50)", value: 2272 },
{ label: "29x2.1 (622-54)", value: 2288 },
{ label: "29x2.2 (622-56)", value: 2298 },
{ label: "29x2.25 (622-57)", value: 2305 },
{ label: "29x2.3 (622-58)", value: 2326 },
{ label: "29x2.4 (622-60)", value: 2345 },
{ label: "29x2.5 (622-62)", value: 2360 },
{ label: "29x2.6 (622-65)", value: 2380 },
{ label: "650x23B (571-23)", value: 1948 },
{ label: "650x25B (571-25)", value: 1952 },
{ label: "650x28B (571-28)", value: 1968 },
{ label: "650x42B (584-42)", value: 2100 },
{ label: "24x1.75 (507-47)", value: 1890 },
{ label: "24x2.0 (507-50)", value: 1925 },
```

```
{ label: "24x2.1 (507-54)", value: 1940 },
{ label: "20x1.75 (406-47)", value: 1590 },
{ label: "20x2.0 (406-50)", value: 1620 },
{ label: "20x2.1 (406-54)", value: 1640 },
{ label: "16x1.75 (305-47)", value: 1255 },
{ label: "16x2.0 (305-50)", value: 1275 },
{ label: "16x2.125 (305-54)", value: 1290 }
];

$.each(wheelSizes, (index, size) => {
let option = $('<option></option>').val(size.value).text(size.label);
$('#tire').append(option);
});

$('#sprocket').empty();
for (i = 11 ; i <= 28 ; i++) {
let option = $('<option></option>').val(i).text(`${i} t`);
$('#sprocket').append(option);
}
$('#sprocket').val("17").change();

$('#chainring').empty();
for (i = 28 ; i <= 100 ; i++) {
let option = $('<option></option>').val(i).text(`${i} t`);
$('#chainring').append(option);
}
$('#chainring').val("49").change();

});
```

```
@app.route('/', methods=['GET'])
def render_index():
try:
return render_template("index.html")
except Exception as e:
return "ERROR: {}".format(str(e))

@app.route('/upload_gpx', methods=['POST'])
async def upload_gpx_files():
try:
if 'files' not in request.files and 'links' not in request.form:
raise ValueError("No selected files or links!")

files = request.files.getlist('files')
links = request.form.getlist('links')
wheel_circumference = request.form.get('wheel_circumference', type=int, default=2111)

return jsonify(await analyze_data(
{
```

```
"files": files,
"links": links,
"wheel_circumference": wheel_circumference
})), 200

except Exception as e:
return abort(400, str(e))

def load_gpx(file):

return gpxpy.parse(file)

def calculate_distance(coord1, coord2):
R = 6371000
lat1, lon1 = np.radians(coord1)
lat2, lon2 = np.radians(coord2)

dlat = lat2 - lat1
dlon = lon2 - lon1

a = np.sin(dlat / 2) ** 2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon / 2) ** 2
c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1 - a))

return R * c

def calculate_air_density(altitude):
rho0 = 1.225
scale_height = 8500

air_density = rho0 * np.exp(-altitude / scale_height)

return air_density

def calculate_power(speed, slope, altitude, total_weight=75, drag_coefficient=0.88, frontal_area=0.5,
rolling_resistance_coefficient=0.004):
air_density = calculate_air_density(altitude)
gravity = 9.81
slope_radians = np.arctan(slope / 100.0)
power_gravity = total_weight * gravity * np.sin(slope_radians) * speed
power_aero = 0.5 * air_density * drag_coefficient * frontal_area * (speed ** 3)
power_rolling = total_weight * gravity * rolling_resistance_coefficient * speed
total_power = power_gravity + power_aero + power_rolling

return total_power

def filter_elevations(elevations):
if not elevations:
return []

changes = np.diff(elevations)
dynamic_tolerance = np.std(changes) * 3.33
```

```
filtered = [elevations[0]]

for i in range(1, len(elevations)):
if abs(elevations[i] - filtered[-1]) > dynamic_tolerance:
filtered.append(elevations[i])

return filtered

def calculate_elevation_gain(elevations):
total_gain = 0
previous_elevation = elevations[0]

for current_elevation in elevations[1:]:
if current_elevation > previous_elevation:
total_gain += current_elevation - previous_elevation

previous_elevation = current_elevation

return total_gain


def filter_close_coordinates(coordinates, threshold_distance=25):
if not coordinates:
return []

filtered_coords = [coordinates[0]] # Keep the first coordinate

for coord in coordinates[1:]:
last_coord = filtered_coords[-1]
if calculate_distance(last_coord, coord) > threshold_distance:
filtered_coords.append(coord)

return filtered_coords

def map_surface_value(surface):
if surface == 'asphalt':
return 1
elif surface == 'concrete':
return 1
elif surface == 'chipseal':
return 1
elif surface == 'concrete:plates':
return 0.9
elif surface == 'sett':
return 0.9
elif surface == 'paving_stones':
return 0.9
elif surface == 'bricks':
return 0.85
elif surface == 'paved':
return 1
```

```
elif surface == 'compacted':
return 0.65
elif surface == 'unpaved':
return 0.6
elif surface == 'fine_gravel':
return 0.66
elif surface == 'gravel':
return 0.5
elif surface == 'pebblestone':
return 0.3
elif surface == 'rock':
return 0.01
elif surface == 'ground':
return 0.3
elif surface == 'grass':
return 0.2
elif surface == 'unhewn_cobblestone':
return 0.13
elif surface == 'cobblestone':
return 0.15
elif surface == 'dirt':
return 0.1
elif surface == 'mud':
return 0.1
elif surface == 'sand':
return 0.1
else:
return 1

def map_highway_value(highway):
if highway == 'motorway':
return 1.0
elif highway == 'trunk':
return 0.9
elif highway == 'primary':
return 1
elif highway == 'secondary':
return 0.9
elif highway == 'tertiary':
return 0.75
elif highway == 'residential':
return 0.95
elif highway == 'service':
return 0.6
elif highway == 'track':
return 0.5
elif highway == 'path':
return 0.3
```

```
elif highway == 'cycleway':
return 1
elif highway == 'footway':
return 0.3
elif highway == 'unclassified':
return 0.5
else:
return 1

def map_tracktype_value(tracktype):
if tracktype == 'grade1':
return 0.9
elif tracktype == 'grade2':
return 0.666
elif tracktype == 'grade3':
return 0.333
elif tracktype == 'grade4':
return 0.25
elif tracktype == 'grade5':
return 0
else:
return 0


def process_element(element, coordinates, max_distance):
element_coords = None
if 'center' in element:
element_coords = (element['center']['lat'], element['center']['lon'])
elif 'geometry' in element:
element_coords = (element['geometry'][0]['lat'], element['geometry'][0]['lon'])

if not element_coords:
return None

is_near = any(calculate_distance(coord, element_coords) <= max_distance for coord in coordinates)

if not is_near:
return None

score = 0
tag_count = 0

if 'tags' in element:
tags = element['tags']

if 'tracktype' in tags:
tracktype_value = map_tracktype_value(tags['tracktype'])
score += tracktype_value
tag_count += 1

if 'surface' in tags:
```

```
surface_value = map_surface_value(tags['surface'])
score += surface_value
tag_count += 1

if tag_count > 0:
score /= tag_count
return score

return None


def get_surface_types(coordinates, max_distance=100, max_workers=4):
if not coordinates:
return []

latitudes = [coord[0] for coord in coordinates]
longitudes = [coord[1] for coord in coordinates]

min_lat = min(latitudes)
max_lat = max(latitudes)
min_lon = min(longitudes)
max_lon = max(longitudes)

query = f"""
[out:json];
(
way({min_lat},{min_lon},{max_lat},{max_lon})["surface"];
);
out body geom;
"""

response = requests.get("http://overpass-api.de/api/interpreter", params={'data': query})

if response.status_code != 200:
raise Exception("Error fetching data from Overpass API")

data = response.json()
elements = data.get('elements', [])

results = []

with ThreadPoolExecutor(max_workers=max_workers) as executor:
future_to_element = {
executor.submit(process_element, element, coordinates, max_distance): element
for element in elements
}

for future in as_completed(future_to_element):
score = future.result()
if score is not None:
results.append(score)
```

```
return results

def parse_gpx_data(gpx_data):
elevations = []
distances = []
slopes = []
speeds = []
heart_rates = []
cadences = []
powers = []
estimated_powers = []
coordinates = []

previous_point = None

for track in gpx_data.tracks:
for segment in track.segments:
for point in segment.points:
if point.elevation is not None:
elevations.append(point.elevation)
if point.latitude is not None and point.longitude is not None:
coordinates.append((point.latitude, point.longitude))

if hasattr(point, 'speed') and point.speed is not None:
speeds.append(point.speed)

if hasattr(point, 'heart_rate') and point.heart_rate is not None:
heart_rates.append(point.heart_rate)

if hasattr(point, 'cadence') and point.cadence is not None:
cadences.append(point.cadence)

if hasattr(point, 'power') and point.power is not None:
powers.append(point.power)

if previous_point is not None:
if previous_point.latitude is not None and previous_point.longitude is not None and \
point.latitude is not None and point.longitude is not None:
distance = calculate_distance(
(previous_point.latitude, previous_point.longitude),
(point.latitude, point.longitude)
)
distances.append(distance)

if previous_point.elevation is not None and point.elevation is not None:
elevation_change = point.elevation - previous_point.elevation
if distance > 0:
slope = (elevation_change / distance) * 100
slopes.append(slope)
```

```
if hasattr(point, 'time') and point.time is not None and \
hasattr(previous_point, 'time') and previous_point.time is not None:
time_difference = (point.time - previous_point.time).total_seconds()

if time_difference > 0:
speed_mps = distance / time_difference
speed_kmh = speed_mps * 3.6
speeds.append(speed_kmh)

previous_point = point

if coordinates:
surfaces = get_surface_types(filter_close_coordinates(coordinates))
else:
surfaces = []

total_distance = round(sum(distances) if distances else 0)
elevation_flat_threshold = total_distance * 0.001
elevation_low_threshold = total_distance * 0.005
elevation_medium_threshold = total_distance * 0.01
elevation_high_threshold = total_distance * 0.015
elevation_step_threshold = total_distance * 0.05

return {
"elevations": elevations,
"distances": distances,
"slopes": slopes,
"speeds": speeds,
"heart_rates": heart_rates,
"cadences": cadences,
"powers": powers,
"surfaces": surfaces,
"total_distance": total_distance,
"elevation_flat_threshold": elevation_flat_threshold,
"elevation_low_threshold": elevation_low_threshold,
"elevation_medium_threshold": elevation_medium_threshold,
"elevation_high_threshold": elevation_high_threshold,
"elevation_step_threshold": elevation_step_threshold,
"elevation_gain": calculate_elevation_gain(filter_elevations(elevations)) if elevations else 0,
"avg_speed": np.mean(speeds) if speeds else 25,
"avg_heart_rate": np.mean(heart_rates) if heart_rates else 0,
"avg_cadence": np.mean(cadences) if cadences else 0,
"avg_power": np.mean(powers) if powers else 0,
"avg_surface": np.mean(surfaces) if surfaces else 1,
}


def calculate_optimal_gear_ratio(data):

speed = ctrl.Antecedent(np.arange(0, 75, 1), 'speed')
```

```
slope = ctrl.Antecedent(np.arange(-35, 35, 1), 'slope')
surface = ctrl.Antecedent(np.arange(0, 1, 0.01), 'surface')
cadence = ctrl.Antecedent(np.arange(0, 140, 1), 'cadence')
heart_rate = ctrl.Antecedent(np.arange(0, 200, 1), 'heart_rate')
power = ctrl.Antecedent(np.arange(0, 1000, 1), 'power')
gear_ratio = ctrl.Consequent(np.arange(1, 5, 0.05), 'gear_ratio')

speed['low'] = fuzz.trimf(speed.universe, [0, 0, 20])
speed['medium'] = fuzz.trimf(speed.universe, [15, 25, 35])
speed['high'] = fuzz.trimf(speed.universe, [30, 40, 60])

cadence['low'] = fuzz.trimf(cadence.universe, [0, 0, 60])
cadence['medium'] = fuzz.trimf(cadence.universe, [50, 80, 110])
cadence['high'] = fuzz.trimf(cadence.universe, [100, 130, 140])

heart_rate['low'] = fuzz.trimf(heart_rate.universe, [0, 0, 120])
heart_rate['medium'] = fuzz.trimf(heart_rate.universe, [100, 140, 180])
heart_rate['high'] = fuzz.trimf(heart_rate.universe, [160, 200, 200])

power['low'] = fuzz.trimf(power.universe, [0, 0, 150])
power['medium'] = fuzz.trimf(power.universe, [100, 200, 300])
power['high'] = fuzz.trimf(power.universe, [250, 400, 500])

surface['bad'] = fuzz.trimf(surface.universe, [0, 0, 0.66])
surface['medium'] = fuzz.trimf(surface.universe, [0.44, 0.77, 0.9])
surface['good'] = fuzz.trimf(surface.universe, [0.82, 0.96, 1])
surface['perfect'] = fuzz.trimf(surface.universe, [0.95, 1, 1])

flat_threshold = data['elevation_flat_threshold']
low_threshold = data['elevation_low_threshold']
medium_threshold = data['elevation_medium_threshold']
high_threshold = data['elevation_high_threshold']
step_threshold = data['elevation_step_threshold']

elevation_gain = ctrl.Antecedent(np.arange(0, high_threshold + 50, 1), 'elevation_gain')
elevation_gain['flat'] = fuzz.trimf(elevation_gain.universe, [0, 0, flat_threshold])
elevation_gain['low'] = fuzz.trimf(elevation_gain.universe, [flat_threshold, low_threshold, low_threshold +
((medium_threshold-low_threshold)/2)])
elevation_gain['medium'] = fuzz.trimf(elevation_gain.universe,
[low_threshold, medium_threshold, medium_threshold + ((high_threshold-medium_threshold)/2)])
elevation_gain['high'] = fuzz.trimf(elevation_gain.universe,
[medium_threshold, high_threshold, high_threshold + ((step_threshold-high_threshold)/2)])
elevation_gain['step'] = fuzz.trimf(elevation_gain.universe,
[high_threshold, step_threshold, step_threshold])

gear_ratio['low'] = fuzz.trimf(gear_ratio.universe, [1, 2, 2.5])
gear_ratio['medium'] = fuzz.trimf(gear_ratio.universe, [2.3, 2.7, 3.2])
gear_ratio['high'] = fuzz.trimf(gear_ratio.universe, [3, 3.2, 3.5])
gear_ratio['track'] = fuzz.trimf(gear_ratio.universe, [3.4, 3.6, 4.2])
```

```
slope['negative'] = fuzz.trimf(slope.universe, [-35, -10, -5])
slope['flat'] = fuzz.trimf(slope.universe, [-10, 0, 10])
slope['positive'] = fuzz.trimf(slope.universe, [5, 10, 35])

rules = [
ctrl.Rule(elevation_gain['flat'], gear_ratio['track']),
ctrl.Rule(elevation_gain['low'], gear_ratio['high']),
ctrl.Rule(elevation_gain['medium'], gear_ratio['medium']),
ctrl.Rule(elevation_gain['high'], gear_ratio['low']),
ctrl.Rule(elevation_gain['step'], gear_ratio['low']),

ctrl.Rule(speed['low'], gear_ratio['low']),
ctrl.Rule(speed['medium'], gear_ratio['medium']),
ctrl.Rule(speed['high'], gear_ratio['high']),
]

avg_slope = np.mean(data["slopes"])
slope_rules = [
ctrl.Rule(slope['negative'], gear_ratio['high']),
ctrl.Rule(slope['flat'], gear_ratio['medium']),
ctrl.Rule(slope['positive'], gear_ratio['low']),
]
if avg_slope > 1 or avg_slope < -1:
rules+=slope_rules

surface_rules = [
ctrl.Rule(surface['bad'], gear_ratio['low']),
ctrl.Rule(surface['medium'], gear_ratio['medium']),
ctrl.Rule(surface['good'], gear_ratio['high']),
ctrl.Rule(surface['perfect'], gear_ratio['track']),
]
rules += surface_rules

track_surface_rules = [
ctrl.Rule(surface['perfect'] & elevation_gain['flat'], gear_ratio['track']),
]
rules += track_surface_rules

cadence_rules = [
ctrl.Rule(cadence['low'], gear_ratio['low']),
ctrl.Rule(cadence['medium'], gear_ratio['medium']),
ctrl.Rule(cadence['high'], gear_ratio['high']),
]

if data['avg_cadence'] > 0:
rules += cadence_rules

heart_rate_rules = [
ctrl.Rule(heart_rate['low'], gear_ratio['low']),
ctrl.Rule(heart_rate['medium'], gear_ratio['medium']),
```

```
ctrl.Rule(heart_rate['high'], gear_ratio['high']),
]

if data['avg_heart_rate'] > 0:
rules += heart_rate_rules

power_rules = [
ctrl.Rule(power['low'], gear_ratio['low']),
ctrl.Rule(power['medium'], gear_ratio['medium']),
ctrl.Rule(power['high'], gear_ratio['high']),
]

if data['avg_power'] > 0:
rules += power_rules

gear_ratio_ctrl = ctrl.ControlSystem(rules)
gear_ratio_simulation = ctrl.ControlSystemSimulation(gear_ratio_ctrl)

gear_ratio_simulation.input['speed'] = data['avg_speed']
gear_ratio_simulation.input['elevation_gain'] = data['elevation_gain']
gear_ratio_simulation.input['surface'] = data["avg_surface"]
if avg_slope > 1 or avg_slope < -1:
gear_ratio_simulation.input['slope'] = avg_slope
if data['avg_cadence'] > 0:
gear_ratio_simulation.input['cadence'] = data['avg_cadence']
if data['avg_heart_rate'] > 0:
gear_ratio_simulation.input['heart_rate'] = data['avg_heart_rate']
if data['avg_power'] > 0:
gear_ratio_simulation.input['power'] = data['avg_power']

gear_ratio_simulation.compute()

return gear_ratio_simulation.output['gear_ratio']


def estimate_speed_threshold(data):
surface = ctrl.Antecedent(np.arange(0, 1, 0.01), 'surface')

surface['bad'] = fuzz.trimf(surface.universe, [0, 0, 0.66])
surface['medium'] = fuzz.trimf(surface.universe, [0.44, 0.77, 0.9])
surface['good'] = fuzz.trimf(surface.universe, [0.82, 0.96, 1])
surface['perfect'] = fuzz.trimf(surface.universe, [0.95, 1, 1])

flat_threshold = data['elevation_flat_threshold']
low_threshold = data['elevation_low_threshold']
medium_threshold = data['elevation_medium_threshold']
high_threshold = data['elevation_high_threshold']
step_threshold = data['elevation_step_threshold']

elevation_gain = ctrl.Antecedent(np.arange(0, high_threshold + 50, 1), 'elevation_gain')
elevation_gain['flat'] = fuzz.trimf(elevation_gain.universe, [0, 0, flat_threshold])
```

```
elevation_gain['low'] = fuzz.trimf(elevation_gain.universe, [flat_threshold, low_threshold, low_threshold + (
(medium_threshold - low_threshold) / 2)])
elevation_gain['medium'] = fuzz.trimf(elevation_gain.universe,
[low_threshold, medium_threshold,
medium_threshold + ((high_threshold - medium_threshold) / 2)])
elevation_gain['high'] = fuzz.trimf(elevation_gain.universe,
[medium_threshold, high_threshold,
high_threshold + ((step_threshold - high_threshold) / 2)])
elevation_gain['step'] = fuzz.trimf(elevation_gain.universe,
[high_threshold, step_threshold, step_threshold])

speed_threshold = ctrl.Consequent(np.arange(0, 75, 1), 'speed_threshold')

speed_threshold['low'] = fuzz.trimf(speed_threshold.universe, [0, 0, 20])
speed_threshold['medium'] = fuzz.trimf(speed_threshold.universe, [15, 25, 35])
speed_threshold['high'] = fuzz.trimf(speed_threshold.universe, [30, 35, 45])
speed_threshold['track'] = fuzz.trimf(speed_threshold.universe, [40, 50, 70])

rules = [
ctrl.Rule(surface['bad'] & elevation_gain['flat'], speed_threshold['high']),
ctrl.Rule(surface['bad'] & elevation_gain['low'], speed_threshold['medium']),
ctrl.Rule(surface['bad'] & elevation_gain['medium'], speed_threshold['medium']),
ctrl.Rule(surface['bad'] & elevation_gain['high'], speed_threshold['low']),
ctrl.Rule(surface['bad'] & elevation_gain['step'], speed_threshold['low']),

ctrl.Rule(surface['medium'] & elevation_gain['flat'], speed_threshold['high']),
ctrl.Rule(surface['medium'] & elevation_gain['low'], speed_threshold['high']),
ctrl.Rule(surface['medium'] & elevation_gain['medium'], speed_threshold['medium']),
ctrl.Rule(surface['medium'] & elevation_gain['high'], speed_threshold['low']),
ctrl.Rule(surface['medium'] & elevation_gain['step'], speed_threshold['low']),

ctrl.Rule(surface['good'] & elevation_gain['flat'], speed_threshold['track']),
ctrl.Rule(surface['good'] & elevation_gain['low'], speed_threshold['high']),
ctrl.Rule(surface['good'] & elevation_gain['medium'], speed_threshold['medium']),
ctrl.Rule(surface['good'] & elevation_gain['high'], speed_threshold['low']),
ctrl.Rule(surface['good'] & elevation_gain['step'], speed_threshold['low']),

ctrl.Rule(surface['perfect'] & elevation_gain['flat'], speed_threshold['track']),
ctrl.Rule(surface['perfect'] & elevation_gain['low'], speed_threshold['track']),
ctrl.Rule(surface['perfect'] & elevation_gain['medium'], speed_threshold['high']),
ctrl.Rule(surface['perfect'] & elevation_gain['high'], speed_threshold['medium']),
ctrl.Rule(surface['perfect'] & elevation_gain['step'], speed_threshold['low']),
]

speed_control = ctrl.ControlSystem(rules)
speed_simulation = ctrl.ControlSystemSimulation(speed_control)

speed_simulation.input['elevation_gain'] = data['elevation_gain']
speed_simulation.input['surface'] = data["avg_surface"]
```

```
speed_simulation.compute()

return speed_simulation.output.get('speed_threshold', 70)

def estimate_average_power(data):
speed = ctrl.Antecedent(np.arange(0, 75, 1), 'speed')
surface = ctrl.Antecedent(np.arange(0, 1, 0.01), 'surface')
heart_rate = ctrl.Antecedent(np.arange(0, 200, 1), 'heart_rate')
average_power = ctrl.Consequent(np.arange(0, 750, 1), 'average_power')

speed['low'] = fuzz.trimf(speed.universe, [0, 0, 20])
speed['medium'] = fuzz.trimf(speed.universe, [15, 25, 35])
speed['high'] = fuzz.trimf(speed.universe, [30, 35, 45])
speed['track'] = fuzz.trimf(speed.universe, [40, 50, 70])

surface['bad'] = fuzz.trimf(surface.universe, [0, 0, 0.66])
surface['medium'] = fuzz.trimf(surface.universe, [0.44, 0.77, 0.9])
surface['good'] = fuzz.trimf(surface.universe, [0.82, 0.96, 1])
surface['perfect'] = fuzz.trimf(surface.universe, [0.95, 1, 1])

flat_threshold = data['elevation_flat_threshold']
low_threshold = data['elevation_low_threshold']
medium_threshold = data['elevation_medium_threshold']
high_threshold = data['elevation_high_threshold']
step_threshold = data['elevation_step_threshold']

elevation_gain = ctrl.Antecedent(np.arange(0, high_threshold + 50, 1), 'elevation_gain')
elevation_gain['flat'] = fuzz.trimf(elevation_gain.universe, [0, 0, flat_threshold])
elevation_gain['low'] = fuzz.trimf(elevation_gain.universe, [flat_threshold, low_threshold, low_threshold + (
(medium_threshold - low_threshold) / 2)])
elevation_gain['medium'] = fuzz.trimf(elevation_gain.universe,
[low_threshold, medium_threshold,
medium_threshold + ((high_threshold - medium_threshold) / 2)])
elevation_gain['high'] = fuzz.trimf(elevation_gain.universe,
[medium_threshold, high_threshold,
high_threshold + ((step_threshold - high_threshold) / 2)])
elevation_gain['step'] = fuzz.trimf(elevation_gain.universe,
[high_threshold, step_threshold, step_threshold])

heart_rate['low'] = fuzz.trimf(heart_rate.universe, [0, 0, 90])
heart_rate['medium'] = fuzz.trimf(heart_rate.universe, [70, 120, 150])
heart_rate['high'] = fuzz.trimf(heart_rate.universe, [130, 200, 200])

average_power['low'] = fuzz.trimf(average_power.universe, [0, 0, 100])
average_power['medium'] = fuzz.trimf(average_power.universe, [50, 150, 300])
average_power['high'] = fuzz.trimf(average_power.universe, [250, 400, 750])

rules = [
ctrl.Rule(speed['low'] & surface['bad'], average_power['low']),
ctrl.Rule(speed['low'] & surface['medium'], average_power['low']),
```

```
ctrl.Rule(speed['low'] & surface['good'], average_power['medium']),
ctrl.Rule(speed['low'] & surface['perfect'], average_power['medium']),

ctrl.Rule(speed['medium'] & surface['bad'], average_power['low']),
ctrl.Rule(speed['medium'] & surface['medium'], average_power['medium']),
ctrl.Rule(speed['medium'] & surface['good'], average_power['medium']),
ctrl.Rule(speed['medium'] & surface['perfect'], average_power['high']),

ctrl.Rule(speed['high'] & surface['bad'], average_power['medium']),
ctrl.Rule(speed['high'] & surface['medium'], average_power['high']),
ctrl.Rule(speed['high'] & surface['good'], average_power['high']),
ctrl.Rule(speed['high'] & surface['perfect'], average_power['high']),

ctrl.Rule(speed['track'] & surface['bad'], average_power['medium']),
ctrl.Rule(speed['track'] & surface['medium'], average_power['high']),
ctrl.Rule(speed['track'] & surface['good'], average_power['high']),
ctrl.Rule(speed['track'] & surface['perfect'], average_power['high']),

ctrl.Rule(speed['low'] & elevation_gain['flat'], average_power['low']),
ctrl.Rule(speed['low'] & elevation_gain['low'], average_power['low']),
ctrl.Rule(speed['low'] & elevation_gain['medium'], average_power['medium']),
ctrl.Rule(speed['low'] & elevation_gain['high'], average_power['medium']),
ctrl.Rule(speed['low'] & elevation_gain['step'], average_power['medium']),

ctrl.Rule(speed['medium'] & elevation_gain['flat'], average_power['medium']),
ctrl.Rule(speed['medium'] & elevation_gain['low'], average_power['medium']),
ctrl.Rule(speed['medium'] & elevation_gain['medium'], average_power['high']),
ctrl.Rule(speed['medium'] & elevation_gain['high'], average_power['high']),
ctrl.Rule(speed['medium'] & elevation_gain['step'], average_power['high']),

ctrl.Rule(speed['high'] & elevation_gain['flat'], average_power['high']),
ctrl.Rule(speed['high'] & elevation_gain['low'], average_power['high']),
ctrl.Rule(speed['high'] & elevation_gain['medium'], average_power['high']),
ctrl.Rule(speed['high'] & elevation_gain['high'], average_power['high']),
ctrl.Rule(speed['high'] & elevation_gain['step'], average_power['high']),

ctrl.Rule(speed['track'] & elevation_gain['flat'], average_power['high']),
ctrl.Rule(speed['track'] & elevation_gain['low'], average_power['high']),
ctrl.Rule(speed['track'] & elevation_gain['medium'], average_power['high']),
ctrl.Rule(speed['track'] & elevation_gain['high'], average_power['high']),
ctrl.Rule(speed['track'] & elevation_gain['step'], average_power['high']),
]

heart_rate_rules = [
ctrl.Rule(heart_rate['low'], average_power['low']),
ctrl.Rule(heart_rate['medium'], average_power['medium']),
ctrl.Rule(heart_rate['high'], average_power['high'])
]
if data['avg_heart_rate'] > 0:
rules += heart_rate_rules
```

```
power_control = ctrl.ControlSystem(rules)
power_simulation = ctrl.ControlSystemSimulation(power_control)

power_simulation.input['speed'] = data['avg_speed']
power_simulation.input['surface'] = data['avg_surface']
power_simulation.input['elevation_gain'] = data['elevation_gain']

if data['avg_heart_rate'] > 0:
power_simulation.input['heart_rate'] = data['avg_heart_rate']

power_simulation.compute()

return power_simulation.output.get('average_power', 122)
```

```
load_dotenv()

STRAVA_API_KEY = os.getenv('STRAVA_API_KEY')


def find_gear_combination(target_ratio, max_chainring=60, max_sprocket=28, threshold=0.01):
combinations = []

for chainring in range(28, max_chainring + 1):
for sprocket in range(9, max_sprocket + 1):
ratio = chainring / sprocket

if abs(ratio - target_ratio) < threshold:
combinations.append((chainring, sprocket))

return combinations[-1] if len(combinations) else find_gear_combination(target_ratio, threshold=threshold+0.01)


def calculate_gear_ratio_with_adjustments(current_gear_ratio, wheel_circumference=2111, crank_length=170):
standard_wheel_circumference = 2111 # in mm
standard_crank_length = 170 # in mm, typical crank length
distance_per_pedal_revolution = current_gear_ratio * standard_wheel_circumference
adjusted_distance_per_pedal_revolution = distance_per_pedal_revolution * (crank_length / standard_crank_length)

new_gear_ratio = adjusted_distance_per_pedal_revolution / wheel_circumference

return new_gear_ratio


def extract_activity_id(link):
match = re.search(r"activities/(\d+)", link)
if match:
return match.group(1)
raise ValueError("Invalid Strava link format")


async def analyze_data(input_data):
strava_gpx_api = strava2gpx(client_id=138004, client_secret='d994095b0ef52c46f328460ee4d321396729b335',
refresh_token='d5c5809884a46f6fcad1a7ddb422bf79ec9eedf5')
```

```
await strava_gpx_api.connect()

def process_file(file):
if file.filename.endswith('.gpx'):
data = parse_gpx_data(load_gpx(file))
data["speed_threshold"] = estimate_speed_threshold(data)
data["avg_estimated_power"] = estimate_average_power(data)
return {
"data": data,
"gear_ratio": calculate_gear_ratio_with_adjustments(calculate_optimal_gear_ratio(data),
input_data["wheel_circumference"])
}
return None

async def fetch_gpx_from_strava(link):
activity_id = extract_activity_id(link)
output_filename = f"{activity_id}.gpx"
await strava_gpx_api.write_to_gpx(activity_id, output=output_filename)

async with aiofiles.open(output_filename, 'r') as f:
gpx_data = await f.read()

return parse_gpx_data(gpx_data)

data = []

with ThreadPoolExecutor() as executor:
futures = [executor.submit(process_file, file) for file in input_data["files"]]
for future in as_completed(futures):
result = future.result()
if result:
data.append(result)

for link in input_data["links"]:
try:
gpx_data = await fetch_gpx_from_strava(link)
result = calculate_gear_ratio_with_adjustments(calculate_optimal_gear_ratio(gpx_data),
input_data["wheel_circumference"])
data.append(result)
except Exception as e:
print(f"Error processing link {link}: {e}")

avg_gear_ratio = np.mean(np.array([obj["gear_ratio"] for obj in data]))
avg_speed = np.array([obj["data"]["avg_speed"] for obj in data])
avg_speed_thresholds = np.array([obj["data"]["speed_threshold"] for obj in data])
avg_heart_rates = np.array([obj["data"]["avg_heart_rate"] for obj in data])
avg_cadences = np.array([obj["data"]["avg_cadence"] for obj in data])
avg_powers = np.array([obj["data"]["avg_power"] for obj in data])
avg_estimated_powers = np.array([obj["data"]["avg_estimated_power"] for obj in data])
avg_surfaces = np.array([obj["data"]["avg_surface"] for obj in data])
```

```
elevation_gains = []
elevation_thresholds = []
for obj in data:
elevation_gains.append(obj["data"]["elevation_gain"])
obj_elevation_threshold = (
obj["data"]["elevation_high_threshold"]
if obj["data"]["elevation_gain"] < obj["data"]["elevation_high_threshold"]
else obj["data"]["elevation_step_threshold"]
)
elevation_thresholds.append(obj_elevation_threshold)

avg_power = np.mean(avg_powers)
if avg_power == 0:
avg_power = np.mean(avg_estimated_powers)

avg_data = {
"avg_speed": np.mean(avg_speed),
"avg_speed_threshold": np.mean(avg_speed_thresholds),
"avg_heart_rate": np.mean(avg_heart_rates),
"avg_cadence": np.mean(avg_cadences),
"avg_power": avg_power,
"avg_surface": np.mean(avg_surfaces),
"elevation_gain": np.mean(elevation_gains),
"elevation_threshold": np.mean(elevation_thresholds),
}

return {
"optimal_gear_ratio": find_gear_combination(avg_gear_ratio),
"data": avg_data
```

ДОДОТОК Б: ТЕЗИ КОНФЕРЕНЦІЇ

Development of an Application for Selecting Gear Ratios for Single-Speed Bicycles Using Fuzzy Logic and Artificial Intelligence

T.V. Ptashchenko[a], O.A. Dvirna[a]

Poltava National Technical University, Pervotravnevy prospect, 24, Poltava, 36011, Ukraine

Abstract

This project presents a gear ratio calculator application tailored for single-speed and fixed-gear bicycles. The application features a standard gear ratio calculator, along with AI-based analysis of tracking data (e.g., average speed, heart rate, weight, terrain type, average slope) to select optimal gear ratios. The app supports GPX and TCX file parsing to enable detailed performance analysis for cyclists.

Keywords 23

Gear ratio calculator, GPX analysis, TCX analysis, single-speed bicycles, AI in cycling, fixed-gear optimization

Introduction

The application aims to assist single-speed and fixed-gear cyclists in selecting optimal gear ratios based on a variety of cycling data inputs. With the increasing popularity of single-speed and fixed-gear bicycles in urban areas [6] (**Figure 5** demonstrates increasing popularity of single-speed and fixed-gear bicycles), the app's focus is on optimizing gear ratio selection for various riding conditions by analyzing data obtained from GPX and TCX files [9] and integrating with popular tracking platforms. The app leverages fuzzy logic and AI models to adjust gear ratio recommendations dynamically according to factors like terrain, speed, and surface type.

Methodology

Data Parsing

To determine the most suitable gear ratios for any given route, the application first relies on GPX or TCX files [9], which contain a wealth of geo and physical data essential for cycling analysis. The file is parsed to extract critical data points, such as GPS coordinates and elevation values, speed, cadence, hear rate, etc. which allow for the mapping and profiling of the ride. These coordinates provide the precise route, while elevation data highlights altitude changes, allowing for accurate slope calculations.

---

[23]DEIT 2024: Digital Economy and IT: Trends and Perspectives 2024, November 28–29, 2024, Poltava, Ukraine
EMAIL: timotiaztec@gmail.com (A. 1);
ORCID: 0009-0003-9717-601X (A. 1);

Also application estimates physical abilities of a rider based on available info. To further refine the model's understanding of the terrain, the app integrates with the Overpass API [8], a tool within the OpenMaps platform. By querying the API, the app retrieves information on the road surface type for each segment of the route. Surface type data includes details on asphalt, gravel, dirt trails, and other surface variations, each of which can significantly impact a cyclist's performance and optimal gearing. This integration ensures the app considers not only elevation changes but also surface textures, enabling it to suggest gear ratios that better match the physical demands of the entire route.

Fuzzy Logic and AI Integration

For a cyclist, the best gear ratio changes based on ride conditions such as slope, speed, cadence, and the type of surface. Rather than relying on rigid, rule-based calculations, this application uses fuzzy logic principles [1]. Fuzzy logic allows the system to interpret continuous values like incline or speed in a way that mimics human reasoning - categorizing each value into qualitative categories (such as "steep," "moderate," or "flat" for incline) and generating recommendations that feel intuitive to the cyclist [2]. These fuzzy logic rules form a flexible framework that adjusts the suggested gear ratio by weighing variables such as average speed, surface roughness, and incline. Additionally, OpenAI's API is incorporated to further refine the model's insights, providing a layer of predictive analytics that tailors suggestions to the user's unique tendencies and long-term riding data.

Application Design

User Interface and Interaction

The application's interface is designed to be intuitive, allowing users to upload multiple files effortlessly. The UI is styled with interactive components styled using jQuery UI for a smooth, polished experience. Key metrics such as elevation gain, power output, cadence, speed, hear rate and surface type are displayed prominently, providing the cyclist with a quick overview of the ride profile. Feedback on recommended gear ratios is delivered in an easy-to-read format, example **Figure 5:** .

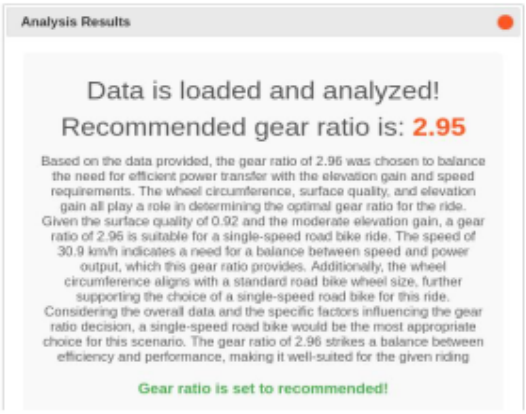

Figure 5: Example of AI based easy-to-read feedback.

Backend and Processing

The application's backend is developed in Python, leveraging libraries such as NumPy and Gpxpy for data processing and Skfuzzy for fuzzy logic integration. Flask serves as the core web framework, handling file uploads and API requests. Upon receiving a GPX/TCX file, the app initiates a series of processing steps that parse the file's data, query the Overpass API for surface types, and compute elevation and slope for each segment. These metrics are then fed into the fuzzy logic and AI systems,

which process them to generate optimal gear ratio recommendations. The backend efficiently manages data, processing multiple GPX/TCX files simultaneously while maintaining accuracy and speed. This design ensures that the app can handle diverse input from multiple rides or users.

Testing and Validation

The app is tested in real-world cycling environments. Cyclists participate in field tests where they use the app during actual rides, and their feedback is collected on the accuracy and practicality of the gear ratio recommendations. Data from these tests allows for fine-tuning of both the fuzzy logic rules and AI model, ensuring that the recommendations are not only theoretically sound but also practical and intuitive. Field testing provides critical insights that help refine the app to cater to a broader range of cycling preferences and terrain types.

Results and Future Development

Current Performance

Initial testing results indicate that the app delivers accurate and user-specific gear ratio recommendations, adapting well to various terrain types and surface conditions. The integration of fuzzy logic and AI enables the app to handle complex scenarios, such as mixed-surface routes or fluctuating terrain, with a high degree of accuracy. Cyclists have reported that the app's suggestions feel intuitive, reflecting both real-time conditions and their personal riding style, which supports the efficacy of the combined fuzzy logic and AI approach.

Future Enhancements

Future development will focus on expanding the app's functionality and refining its performance. Planned enhancements include increasing the range of data inputs to consider additional factors. Further AI integration will also enable the app to make even more personalized recommendations based on a broader history of user data. Additional UI improvements are planned to enhance the app's interactivity.

Acknowledgements

This project was made possible through the support and insights from several individuals and communities whose expertise and resources significantly contributed to the development of this work.

We would like to extend our sincere gratitude to Andy Fisher, a bicycle engineer from Poltava and an experienced singlespeed mountain bike rider, for his invaluable technical guidance and expertise were essential in refining the calculations and application mechanics. Andy's deep understanding of cycling mechanics and real-world insights provided a practical foundation for many calculations in this application. Also tanks to the "GEMU" Lviv track cycling community, whose members offered continuous feedback during the beta-testing phase. Their experience of track cycling conditions and performance metrics played a key role in shaping the app's adaptive gear recommendations and improving its overall usability.

Lastly, we acknowledge the resources provided by the open-source communities behind OpenMaps, gpxpy, and skfuzzy, which enabled seamless integration of data parsing, machine learning, and fuzzy logic into this project. This work would not have been possible without their dedication to open-source innovation and accessibility.

References

[1] George J. Klir, Fuzzy Sets And Fuzzy Logic, 1995.
[2] James K. Peckol, John Wiley & Sons Ltd., Introduction to fuzzy logic., 2021.