НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

До захисту допущено:

Завідувач кафедри

_____ Оксана ТИМОЩУК

«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системи і методи штучного інтелекту»

спеціальності 122 «Комп'ютерні науки»

на тему: «Логістична система залізничних вантажних перевезень»

Виконав:
студент IV курсу, групи КА-75
Загній Єгор Васильович _____

Керівник:
Доцент, к.ф.-м.н.,
Стусь Олександр Вікторович _____

Консультант з економічного розділу:
Доцент, к.е.н., Рощина Надія Василівна _____

Консультант з нормконтролю:
Доцент, к.т.н., Коваленко Анатолій Єпіфанович _____

Рецензент:
Доцент кафедри СП, к.т.н., Безносик Олександр Юрійович _____

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

Київ – 2021 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки»

Освітня програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Оксана ТИМОЩУК

«26» травня 2021 р.

ЗАВДАННЯ

на дипломну роботу студенту

Загнію Єгору Васильовичу

1. Тема роботи «Логістична система залізничних вантажних перевезень», керівник роботи Стусь Олександр Вікторович, доцент, к.ф.-м.н., затверджені наказом по університету від «26» травня 2021 р. № 1344-с

2. Термін подання студентом роботи 8.06.2021.

3. Вихідні дані до роботи: Файли Stations.xml, Locomotives.xml, Cargo.xml, в яких зберігаються дані про станції, локомотиви і вагони відповідно.

4. Зміст роботи: теоретичні основи логістики залізничних вантажних перевезень, опис алгоритмів рішення поставленої задачі, програмна реалізація, функціонально-вартісний аналіз.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): схеми залізничних шляхів, інтерфейс програмної реалізації, результати роботи алгоритмів, презентація до виступу.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Н.В., доцент		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Формування теми БДР	03.11.2020	
2.	Огляд літератури за тематикою роботи та її опрацювання	03.11.2020-6.03.2021	
3.	Написання першого розділу БДР	7.03.2021-18.04.2021	
4.	Узгодження теми БДР з науковим керівником	19.04.2021	
5.	Програмна реалізація продукту	20.04.2021-30.04.2021	
6.	Написання другого розділу БДР	03.05.2021-18.05.2021	
7.	Написання третього розділу БДР	18.05.2021-25.05.2021	
8.	Написання четвертого розділу БДР	25.05.2021-1.06.2021	

Студент Єгор Загній

Керівник Олександр Стусь

РЕФЕРАТ

Дипломна робота: 89 с., 8 рис., 2 додатки, 6 табл., 14 джерел.

ЛОГІСТИКА, ЗАЛІЗНИЦЯ, ТЕОРІЯ ГРАФІВ, МАШИННЕ НАВЧАННЯ, НЕЙРОМЕРЕЖІ.

В цій роботі досліджується логістика вантажних перевезень. Предметом дослідження роботи є система залізничних вантажних перевезень.

Метою роботи є розробка алгоритму, який буде вирішувати задачу по складанню розкладу для вантажних залізничних поїздів, при цьому буде видавати оптимальне рішення, мінімізуючи час доставки товарів та витрати палива локомотивів.

У цій роботі розглядаються як аналітичні рішення так і рішення за допомогою машинного навчання. Створено програмну реалізацію запропонованих аналітичних рішень.

ABSTRACT

The work consists of: 89 pages., 8 images., 6 tables, 14 sources.

LOGISTICS, RAILWAYS, GRAPH THEORY, MACHINE LEARNING, NEURAL NETWORKS.

This paper investigates the logistics of freight transportation. The subject of the study is the system of rail freight.

The aim of the work is to develop an algorithm that will solve the problem of scheduling for freight railway trains, while issuing the optimal solution, minimizing the time of delivery of goods and fuel consumption of locomotives.

This paper considers both analytical solutions and machine learning solutions. The software implementation of the offered analytical decisions is created.

ЗМІСТ

ВСТУП

У наш час складних логістичних систем задача знаходження оптимального шляху набуває все більшої актуальності. Цю задачу вирішують у багатьох сферах, від алгоритму у грі до дорожнього навігатора.

З огляду на практичність цієї задачі та опираючись на свої вподобання до залізничної тематики, я вирішив створити програму що буде складати маршрути для вантажних поїздів. Для прикладу в цій роботі буде використана схема залізничних шляхів Придніпровської залізниці.

Метою роботи є розробка програмного продукту який буде вирішувати дану задачу. Планується створити два рішення цієї задачі. Одне за допомогою алгоритму. Друге за допомогою машинного навчання. Даний програмний продукт буде створено за допомоги мови програмування C#. Частину з машинним навчанням за допомоги Python. При виконанні роботи застосовується інтегроване середовище розробки програмного забезпечення Microsoft Visual Studio 2017.

РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ ЛОГІСТИКИ ЗАЛІЗНИЧНИХ ВАНТАЖНИХ ПЕРЕВЕЗЕНЬ

1.1 Поняття логістичної системи

У наш час слово «логістика» має великий спектр вжитку. Ним можуть називати будь-які процеси, які пов'язані з транспортуванням, зберіганням та обробкою будь-яких вантажів, товарів. У цій роботі нас буде цікавити саме процес транспортування. Саме слово «логістика» походить з грецької мови, де означає облік.

Поняття логістики виникло дуже давно з появою перших цивілізацій у бронзовий вік, коли активно почала розвиватися торгівля і купцям потрібно було прокладати найбільш оптимальний та безпечний маршрут.

У 1928 р плуг сирійського землероба натрапив на камінь, під яким виявився склеп із стародавньою керамікою. Навряд чи той орач усвідомлював цінність свого відкриття. Дізнавшись про випадкову знахідку, на те місце в наступному році приїхали французькі археологи. Незабаром було знайдено напис, що дозволила археологам визначити, руїни якого міста знаходяться під землею. Це був Угарит, один з найважливіших стародавніх міст Близького Сходу. А самі документи датовані XIV ст. до н. е.

Процвітання Угарита було пов'язано не тільки з його вдалим географічним розташуванням, а й з розвитком науки. Про це свідчать знайдені серед руїн Угарита тисячі глиняних табличок. У їх числі господарські, дипломатичні, юридичні та економічні документи, записані на восьми мовах з використанням п'яти видів письма. Сучасна розшифровка, зокрема економічних документів, ясно свідчить про існування в той далекий час логістики як науки про переміщення товарів і безлічі товарів всередині самого міста і між містами суміжних держав по суші і морськими шляхами [3].

У давній історії постачання запасів їжі, фуражу та спорядження часто виступає як логістична основа для військових походів. Деякі з цих походів

знайомі багатьом школярам - довгий похід Олександра Македонського з Македонії до Інду, сага "Десять тисяч" Ксенофонта, походи Ганнібала в Італію. Більші військові компанії давніх часів - як персидське вторгнення в Грецію в 480 р. до н. е. - здавалося, забезпечувались складами та ринками за маршрутом маршу. Римський легіон поєднав усі три способи постачання в дивовижно гнучку систему. Здатність легіону швидко і далеко ходити багато в чому зумовлена чудовими дорогами та ефективно організованою системою постачання, до складу якого входили пересувні ремонтні майстерні та загони інженерів, майстрів, зброєносці та інші техніки. Запаси витребувались у місцевих органів влади та зберігались в укріплених складах; могли задіюватися робітники та тварини. За потреби легіон міг проносити у своєму обозі та на спинах своїх солдатів запас продовольства до 30 днів. У Першій пунічній війні проти Карфагена (264–241 рр. до н. е.) Римська армія проходила в середньому 26 кілометрів на день протягом чотирьох тижнів.

Однією з давніх найефективніших логістичних систем, коли-небудь відомих, була система монгольських кавалерійських армій XIII століття. Її основою були строгість, дисциплінованість, ретельне планування та організація. Під час походів монгольські армії ділилися на кілька корпусів і широко розповсюджувалися по країні, супроводжуючись обозами з візків, в'ючних тварин та стадами худоби. Маршрути та місця під табори обиралися так щоб був доступ до гарного випасу та продовольчих культур; їжа та корм зберігались заздалегідь за маршрутами маршу. При в'їзді в ворожу країну армія відходила від свого обозу, розділившись на широко відокремлені колони, і з великою швидкістю нападала на непідготовленого ворога з кількох напрямків. Одним із таких підходів монгольська армія за три дні подолала 290 миль. Служби ремонту та транспорту були ретельно організовані. Жорсткий і досвідчений монгольський воїн міг майже нескінченно харчуватися сушеним м'ясом і сирком, доповненими випадковою дичиною; перебуваючи в протоці, він може злити трохи крові з вени на шиї свого коня. Кожен чоловік мав низку поні; багаж був мінімальним, а обладнання було стандартизованим та легким.

На початку 17 століття шведський король Густав II Адольф та принц Моріс Нассауський, військовий герой Нідерландів, ненадовго відновили до європейської війни міру мобільності, яку не спостерігали з часів римського легіону. У цей період помітно збільшився чисельність армій; Густав та його супротивники збирали сили до 100 000, а Людовик XIV з Франції наприкінці століття ще більше. Армії такого розміру повинні були продовжувати рухатися, щоб уникнути голоду; доки вони це робили, у родючій країні вони, як правило, могли утримуватися без підстав, навіть із своїм звичним величезним небойовим "хвостом". Логістична організація покращилася. Під час Тридцятилітньої війни (1618–48) стратегія, як правило, стала придатком логістики, оскільки армії, де це було можливо, пересувалися і забезпечувались вздовж річок, експлуатуючи економіку водного транспорту, і діяли в багатих регіонах виробництва продуктів харчування [14].

Після Тридцятилітньої війни європейська війна стала більш млявою та формалізованою, з обмеженими цілями та складною логістикою, що жертвувала як дальністю, так і мобільністю. Нова наука про фортифікацію зробила міста майже неприступними, одночасно посилюючи їхню стратегічну цінність, роблячи війни 18 століття скоріше справою облог, ніж битв. Логістичними нововведенням стало стратегічно розміщене депо, що як правило створене для підтримки армії, яка проводила облогу. Безпечні лінії зв'язку стали життєво важливими, і для їх захисту були розгорнуті цілі армії. Збільшення чисельності армій та артилерійських і продовольчих обозів ускладнювало транспорт. Крім того, відраза проти зневаги та нелюдськості релігійних воєн 17 століття призвела до обмеження грабунку та спалення, а також до регламентованих реквізицій чи придбання провіанту у місцевих органів влади. Через високу вартість найманого солдата командири, як правило, уникали боїв, а кампанії, як правило, ставали млявими маневрами, спрямованими на погрозу або захист баз і ліній зв'язку. "Шедевр успішного полководця, - зауважив Фрідріх Великий, - полягає в тому, щоб голодувати свого ворога".

Епоха Французької революції та наполеонівського панування в Європі (1789–1815) повернула як мобільність, так і діапазон пересування до європейської війни, а також величезне подальше збільшення чисельності армій. Покинувши облогову війну 18 століття, наполеонівська стратегія наголосила на швидких наступальних операціях, спрямованих на розгром головної сили ворога в кількох вирішальних битвах. Логістична система, успадкована від Старого режиму, виявилася напрочуд пристосованою до нових масштабів і темпів операцій. Організацію зробили більш ефективною, обози багажу знизили, а частину їх вантажу переклали на спину солдата, і значну частину не бойового хвоста було ліквідовано. Тяжко обтяжений регулярний солдат рушив швидше і далі, ніж його найманий попередник. У густонаселених і родючих регіонах армії, що рухались, продовжували існувати шляхом купівлі та реквізиції в сільській місцевості, через яку вони йшли, розподіляючись по паралельних дорогах.

Наполеон зробив порівняно мало логістичних нововведень. Він мілітаризував деякі послуги, які раніше виконували підрядники та цивільний персонал, але служба постачання (інтендант) залишалася цивільною, хоча і під військовим контролем. Суттєвою зміною стало створення в 1807 р. Повністю мілітаризованого лінійного сполучення, яке розділено на секції, кожна з яких обслуговувалась комплектом човникових возів - передвіщаючи поетапну систему поповнення запасів 20 століття. 600-мільовий просунок Великої армії Наполеона з 600 000 чоловік до Росії в 1812 р. Передбачав безпрецедентні масштаби логістичної підготовки. Незважаючи на великі саботажі російського селянства, система привела армію-переможницю до Москви.

З середини 19-го до середини 20-го століть умови та методи логістики були перетворені в результаті фундаментальної зміни інструментів та способів ведення війни - можливо, найбільш фундаментальної зміни з початку організованої війни. Революція мала чотири аспекти: мобілізація масових армій; революція в технології озброєння, що передбачає феноменальне збільшення вогневої сили; економічна революція, яка забезпечила засоби для

харчування, озброєння та транспортування масових армій та революція в техніках управління та організації, яка дозволила країнам експлуатувати свої військові установи ефективніше, ніж будь-коли раніше.

Ці взаємопов’язані події відбулися не відразу. Армії безпрецедентного розміру з'явилися в пізніші роки наполеонівських війн. Але протягом майже століття після 1815 року світ не бачив порівнянної мобілізації робочої сили, окрім громадянської війни в Америці. Тим часом приріст населення (в Європі зі 180 млн. У 1800 р. До 490 млн. У 1914 р.) Створював величезний резерв робочої сили. До кінця XIX століття більшість держав будували великі постійні армії за підтримки ще більших частково навчених резервів. У світові війни 20 століття великі держави мобілізували збройні сили чисельністю мільйонів.

Революція в зброї розпочалася раніше, але прискорилася приблизно в 1830 р. До 1850-х і 60-х років нарізний ударний мушкет, нарізна артилерія із зарядною зброєю, великокаліберні боєприпаси та броньовані бойові кораблі з паровим рухом знаходилися в загальному користуванні.

До середини XIX століття Промислова революція вже дала Великобританії, Франції та США можливість виробляти боєприпаси, продовольство, транспорт та багато інших предметів у кількості, про яку жоден комісар чи інтендант ніколи не мріяв. Але за винятком північних штатів під час громадянської війни в Америці, війни 19 століття майже не подряпали поверхню існуючого воєнного потенціалу. Характер міжнародного суперництва того періоду, як правило, обмежував цілі війни та мобілізацію прихованої військової сили. Лише в часи Першої світової війни, ціною колосальних промахів і марних зусиль, країни почали вивчати техніки "тотальної" війни. Однак задовго до 1914 року з'явилися нові інструменти та методи логістики.

Залізниця, пароплав і телеграф глибоко вплинули на логістичний метод протягом останньої половини 19 століття. Починаючи з Кримської війни (1854–56), телеграфний зв’язок став незамінним інструментом командування,

розвідки та оперативної координації, особливо при контролі залізничного руху. У 20 столітті він поступився більш ефективним формам електронного зв'язку - телефону, радіо, радіолокації, телебаченню, телефотографії та високошвидкісному комп'ютеру [14].

1.2 Логістика залізничних вантажних перевезень

У 60-90-х роках XVIII століття спочатку в Англії, а потім і в інших країнах почався промисловий підйом. Замість ручної праці з'явилося машинне виробництво, замість ремісничих майстерень і мануфактур - великі промислові підприємства.

Англійський винахідник Томас Ньюкомен в 1712 році продемонстрував свій «атмосферний двигун». Це був вдосконалений паровий двигун Севері, в якому Ньюкомен істотно знизив робочий тиск пара. Першим застосуванням двигуна Ньюкомена була відкачка води з глибокої шахти.

У 1763 році російський інженер І. І. Ползунов представив проект парового двигуна для подачі повітря в плавильні печі. Парова машина Ползунова мала потужність 40 кінських сил.

У 1773 році Дж. Ватт будує свою першу діючу парову машину. А в 1774 році, спільно з промисловцем Меттью Болтоном, Ватт відкриває компанію з виробництва парових машин.

Справжню революцію в промисловості справила перша універсальна парова машина, створена інженером Джеймсом Ваттом в 1784 році. З цього моменту парова машина перестає бути прив'язана до вугільним шахтам. Її починають застосовувати на заводах, встановлювати на пароплави, створювати поїзда.

Паровий двигун дав потужний поштовх розвитку транспорту. У 1769 році французький артилерійський офіцер Жозеф Кюньо винайшов перший паровий візок для пересування важких знарядь. Вона виявилася громіздкою і під час

випробувань на вулицях Парижа пробила стіну будинку. Цей візок знайшов своє місце в Паризькому музеї мистецтв і ремесел.

У 1802 році англійський конструктор Річард Тревітік зробив паровий автомобіль. Екіпаж рухався з гуркотом і чадом, лякаючи пішоходів. Його швидкість досягла 10 км / год. Щоб отримати таку швидкість руху, Тревітік зробив величезні провідні колеса, які були гарною підмогою на поганих дорогах.

Одним з попередників рейкового шляху був давньогрецький Діолк - кам'яна дорога-волок для перевезення кораблів через Коринфський перешийок. Як напрямні служили глибокі жолоби, в які поміщали полози, змащені жиром.

У XVI столітті на шахтах Німеччини і сусідніх регіонів використовувалися дерев'яні рейкові шляхи і вагонетки, колеса яких були забезпечені ребордами. У деяких регіонах Англії дерев'яні рейкові дороги для вагонеток були відомі під час правління королеви Єлизавети I (друга половина XVI століття), в XVII столітті вони набули широкого поширення в гірничодобувних районах Англії, а в XVIII столітті їх поступово витіснили залізні рейкові дороги [1].

Першою наземної рейкової дорогою вважається дерев'яна «Воллатонська вагонна дорога» (англ. Wollaton Wagonway) [2]. Ця рейкова дорога довжиною приблизно три кілометри була побудована між 1603 і 1604 роками для перевезення вугілля на кінній тязі між селищами Стрелла (англ. Strelley) і Воллатон (англ. Wollaton) поруч з Ноттінгемом. Точний час закриття дороги невідомо, але шахти Стрелла були закриті в 1620 році. Швидше за все, тоді ж припинила своє існування і рейкова дорога.

У 1755 році для перевезення породи на рудниках Алтаю вже був побудований вузькоколійний шлях з дерев'яними рейками, по яких рухалися дерев'яні ж вагонетки. Уздовж шляху була натягнута тросова петля. Для приведення її в рух використовувалися коні, що обертали шків. На кожній вагонетці було по два затиску, які можна було по черзі чіплялися до однієї або

іншої сторони петлі ведучого троса. Завдяки цьому була можливість зупиняти вагонетки або змінювати напрямок їх руху при безперервному русі провідного троса.

У 1788 році в Петрозаводську з'являється «Чавунний колесопровід» - перша в Росії залізниця. Залізниця була побудована на Олександрівському заводі для потреб підприємства. (Нині ділянки першої російської залізниці зберігаються в Петрозаводську біля будівлі музею ОТЗ і в Губернаторському саду; крім того, в Губернаторському саду збережені колеса від вагонетки).

Довгий час залізничні колії споруджувалися тільки на рудниках, але потім набули поширення пасажирські дороги з кінною тягою. Перша така рейкова дорога була влаштована в 1801 році в Англії між Вондсвортом і Кройдоном.

Першою залізницею, на якій були організовані регулярні пасажирські перевезення, стала в 1807 році Залізниця Свонсі і Мамблза в Уельсі. Так як працездатних паровозів в той час ще не було, як тягової сили використовувалися коні.

Перша залізниця континентальної Європи була побудована в Бельгії між Мехеленом і Брюсселем щодо проекту інженерів П'єра Сімонса і Гюстава Де Ріддер. Вона відкрилася 6 травня 1835 року.

Перша залізниця в Німеччині була відкрита між баварськими містами Нюрнбергом і Фюртом в 1835 році.

Перша в світі залізниця загального користування з паровою тягою була побудована в Англії Джорджем Стефенсоном в 1825 році - між Стоктоном і Дарлінгтоном, і була протяжністю 40 кілометрів (26 миль). Перша залізниця між відносно великими містами була відкрита в 1830 році і з'єднала промисловий центр Манчестер з портовим містом Ліверпуль (56 км). На лінії також використовувалися паровози Стефенсона [4]. До 1840 року протяжність залізниць у Великобританії склала 2390 км [5].

У другій половині 1880-х років XIX століття був досягнутий найвищий рівень приросту світової залізничної мережі в історії. За десять років, з 1880 по 1890 р, залізнична мережа зросла на 245 тис. км., Досягнувши 617,3 тис. км.

Залізничні капіталовкладення в світову мережу за цей п'ятирічний період склали близько 2 млрд фунтів стерлінгів, досягнувши загальної суми 7 млрд фунтів стерлінгів. За темпами і абсолютним приростом залізниць попереду йшли США, де гігантський розмах залізничного будівництва стимулював інтенсивний ріст промислового виробництва засобів виробництва.

На сьогодні у світі побудовано 1 123 000 км залізниці, що в 4 рази більше за відстань до Місяця.

1.3 Постановка задачі

Відома система залізничних шляхів, яка представлена у вигляді графу, де вершини - це станції, а вага зв'язку це довжина шляху. На кожній станції є множина вагонів, кожний з яких має вагу і кінцеву станцію. Також є множина локомотивів, кожен з яких має характеристики потужності та швидкості. Необхідно побудувати маршрути для кожного локомотива з найменшими витратами палива та часу, так щоб кожен з вагонів опинився на станції кінцевого призначення.

Потрібно запропонувати рішення для цієї задачі та розробити програмну реалізацію запропонованих рішень.

1.4 Висновки до розділу 1

В цьому розділі було розглянуто поняття логістики, історію її розвитку. Розглянуто історію виникнення і становлення залізниці, розвиток мережі залізничних шляхів.

Також було сформульовану постановку задачі, сенс якої полягає у створенні програмного забезпечення, яке автоматизує і оптимізує процес складання розкладу для вантажних залізничних поїздів. Було описано вхідні дані до задачі.

РОЗДІЛ 2 ОПИС АЛГОРИТМІВ РІШЕННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

2.1 Аналітичне рішення

Дану задачу неможливо розглядати як звичайну транспортну задачу чи задачу комівояжера, якогось оптимального рішення не існує. Кількість можливих розв’язків є дуже великою, тому звичайний перебір тут не допоможе. Складності тут додає факт, що локомотивів більше одного, що значно збільшує кількість комбінацій дій.

У даній роботі є спроба створити алгоритм, який якщо не завжди буде видавати найоптимальніший варіант розв’язання поставленої задачі, то хоча наближене до оптимального розв’язання задачі.

2.1.1 Алгоритм Дейкстри

В розроблених аналітичних рішеннях одну з головних ролей буде виконувати алгоритм Дейкстри. Алгоритм Дейкстри - це алгоритм пошуку найкоротшого шляху між вузлами у графі, який може, наприклад, системою залізних шляхів. Він був задуманий інформатиком Едсгером В. Дейкстрою в 1956 році і опублікований через три роки.

Алгоритм існує у багатьох варіантах. Оригінальний алгоритм Дейкстри знаходить найкоротший шлях між двома заданими вузлами, але більш поширений варіант фіксує один вузол як вузол "початок" і знаходить найкоротші шляхи від джерела до всіх інших вузлів у графі, створюючи дерево найкоротших шляхів.

Для початкового вузла на графіку алгоритм знаходить найкоротший шлях між цим вузлом та кожним іншим. Він також може бути використаний для пошуку найкоротших шляхів від початкового вузла до одного цільового вузла, тоді він зупиниться, як тільки визначить найкоротший шлях до цільового

вузла. Наприклад, якщо вузли графіка представляють міста, а витрати на край дороги представляють відстань проїзду між парами міст, пов’язаних прямою дорогою, можна використовувати алгоритм Дейкстри, знайти найкоротший маршрут між одним містом та усіма іншими містами. Широко використовують алгоритм мережеві протоколи маршрутизації, зокрема IS-IS та OSPF. Він також використовується як підпрограма в інших алгоритмах, таких як Джонсон.

Алгоритм Дейкстри використовує мітки, які є натуральними або дійсними числами.

Алгоритм Дейкстри використовує структуру даних для зберігання часткових рішень, відсортованих за відстанню від початку, і працює в часі $O((|V| + |E|) \log|V|)$ (де |V| - кількість вузлів і |E| - кількість ребер).

Отже сам алгоритм виглядає так. Нехай вузол, з якого ми починаємо, буде називатися початковим вузлом. Нехай відстань вузла Y - це відстань від початкового вузла до Y. Алгоритм Дейкстри призначить деякі початкові значення відстані і спробує покращити їх поетапно.

1. Позначаємо всі вузли невідвіданими. Створюємо множину усіх невідвіданих вузлів, яку називаємо невідвіданою множиною.

2. Призначаємо кожному вузлу орієнтовне значення відстані: встановлюємо його на нуль для нашого початкового вузла та на нескінченність для всіх інших вузлів. Встановлюємо початковий вузол як поточний.

3. Для поточного вузла розглядаємо всіх його невідвіданих сусідів та обчислюємо їх орієнтовні відстані через поточний вузол. Порівнюємо нещодавно обчислену орієнтовну відстань із поточним призначеним значенням і призначаємо меншу. Наприклад, якщо поточний вузол A позначений відстанню 6, а край, що з'єднує його із сусідом B, має довжину 2, тоді відстань до B через A буде 6 + 2 = 8. Якщо B раніше був позначений відстань більше 8, потім змініть її на 8. В іншому випадку поточне значення буде збережено.

4. Коли ми закінчимо розглядати всіх невідвіданих сусідів поточного вузла, позначаємо поточний вузол як відвіданий та видаляємо його з невідвіданої множини. Відвіданий вузол ніколи більше не перевірятиметься.

5. Якщо цільовий вузол був позначений відвіданим (при плануванні маршруту між двома конкретними вузлами) або якщо найменша орієнтовна відстань серед вузлів у невідвіданому наборі - нескінченність (при плануванні повного обходу; виникає, коли немає зв'язку між початковим вузлом і залишилися невідвідані вузли), зупиняємося. Алгоритм закінчений.

6. В іншому випадку виберіть невідвіданий вузол, позначений найменшою попередньою відстанню, встановіть його як новий "поточний вузол" і поверніться до кроку 3.

При плануванні маршруту насправді не потрібно чекати, поки цільовий вузол буде "відвіданий", як зазначено вище: алгоритм може зупинитися, коли цільовий вузол має найменшу орієнтовну відстань серед усіх "невідвіданих" вузлів (і, отже, його можна вибрати наступний "поточний").

Припустимо, ви хотіли б знайти найкоротший шлях між двома перехрестями на карті міста: початковою точкою та пунктом призначення. Алгоритм Дейкстри спочатку позначає відстань (від початкової точки) до кожного іншого перетину на карті нескінченністю. Це робиться не для того, щоб означати, що існує нескінченна відстань, а для того, щоб зазначити, що ці перехрестя ще не відвідували. Деякі варіанти цього методу залишають відстань перетину без позначень. Тепер виберіть поточний перетин на кожній ітерації. Для першої ітерації поточний перетин буде вихідною точкою, а відстань до нього (мітка перетину) буде дорівнювати нулю. Для наступних ітерацій (після першої) поточне перехрестя буде найближчим невідвіданим перехрестям до вихідної точки (це буде легко знайти).

Від поточного перехрестя оновіть відстань до кожного невідвіданого перехрестя, яке безпосередньо з ним пов’язане. Це робиться шляхом визначення суми відстані між невідвіданим перехрестям і значенням поточного перетину, а потім перемаркіруванням невидимого перехрестя з цим

значенням (сумою), якщо воно менше поточного значення невідвіданого перехрестя. По суті, перехрестя перемаркірується, якщо шлях до нього через поточне перехрестя коротший, ніж раніше відомі шляхи. Щоб полегшити ідентифікацію найкоротшого шляху, олівцем познаємо дорогу стрілкою, що вказує на перемаркіроване перехрестя, якщо перемаркірумо, і видаляємо усі, що вказують на неї. Оновивши відстані до кожного сусіднього перехрестя, познаємо поточне перехрестя як відвідане та вибираємо невідвідане перехрестя з мінімальною відстанню (від початкової точки) - або найнижчу мітку - як поточне перехрестя. Перехрестя, позначені як відвідані, позначені найкоротшим шляхом від вихідної точки до нього і не будуть переглянуті або повернуті.

Продовжуємо цей процес оновлення сусідніх перехресть з найкоротшими відстанями, позначення поточного перехрестя як відвіданого та переміщення до найближчого невикористаного перехрестя, поки ви не позначите пункт призначення як відвіданий. Після того, як позначили пункт призначення як відвіданий (як у випадку з будь-яким відвіданим перехрестям), визначили найкоротший шлях до нього з початкової точки і можемо простежити шлях назад, рухаючись стрілками в зворотному напрямку. У реалізаціях алгоритму це зазвичай робиться (після того, як алгоритм дійшов до вузла призначення), стежачи за батьками вузлів від вузла призначення до початкового вузла; ось чому ми також відстежуємо батьківські дані кожного вузла.

Цей алгоритм не робить спроб безпосереднього "дослідження" до місця призначення, як можна було б очікувати. Швидше, єдиним фактором, що враховує визначення наступного «поточного» перетину, є його відстань від вихідної точки. Таким чином, цей алгоритм розширюється назовні від початкової точки, інтерактивно розглядаючи кожен вузол, який знаходиться ближче з точки зору найкоротшої відстані шляху, поки не дійде до пункту призначення. Якщо зрозуміти це таким чином, стає зрозумілим, як алгоритм обов’язково знаходить найкоротший шлях. Однак це може також виявити одну зі слабких сторін алгоритму: відносну повільність у деяких топологіях [9].

2.1.2 Опис аналітичного рішення

Перший розроблений алгоритм вирішення задачі починається з проходження по множині локомотивів. Аналізуються усі вагони на станції на якій знаходиться локомотив. Спочатку за допомогою алгоритму Дейкстри знаходяться найкоротші відстані від даної станції до усіх інших. Потім для кожного вагону будується маршрут до його станції призначення. Далі вагони збираються у состави по напрямках від даної станції. Локомотив зчіплюється з найбільшим составом, якщо в нього вистачає потужності щоб зрушити його з місця, і відправляється до наступної станції. При цьому, знаючи відстань і швидкість, розрахується час прибуття поїзда до наступної станції. Після обробки локомотива увесь масив локомотивів сортується за часом прибуття до наступної станції, і алгоритм завжди буде оброблювати перший локомотив у списку, таким чином усі поїзди оброблюється у порядку їх приїзду на станцію, і вони не зможуть бачити вагони які знаходяться в дорозі. Якщо на станції немає вагонів поїзд поїде до найближчої станції де вони є.

Цей алгоритм вже почав видає доволі логічний результат. Проте він не завжди буде найоптимальнішим. Також в цьому рішенні немає взаємодії між локомотивами.

Наступна ідея була запозичена мною з навчання з підкріплення. Наприклад алгоритм Q-learning намагається визначити вигоду кожної дії для кожного стану [5]. Я також вирішив створити функцію, яка буде визначати ступінь вирішення задачі в кожний момент. Функція дорівнює сумі всіх відстаней від вагонів до станцій їх кінцевих станцій та відстаней від вагонів до найближчих до них локомотивів.

$$\sum_{i=1}^{N}\left(l_s(w_i) + k l_t(w_i)\right), \tag{2.1}$$

де w_i – це і-ий вагон;

N – кількість вагонів;

$l_s(w_i)$ – відстань і-ого від вагона до його кінцевої станції призначення;

$l_t(w_i)$ – відстань від і-ого вагона до найближчого до нього локомотива;

k – коефіцієнт важливості відстані до локомотива.

Коли ми будемо планувати маршрут, то будемо вибирати такий варіант що найбільше мінімізує дану функцію. Тобто для локомотива буде вигідніше поїхати в сторону де невелика кількість вагонів, але немає локомотива, ніж де велика кількість вагонів але біля них вже є локомотив. Таким чином створено взаємодію між локомотива, якої не було в першому варіанті, і алгоритм буде працювати ефективніше.

Вибирати найкращий варіант маршруту локомотива будемо доволі простим методом. Будемо по черзі нібито переміщувати локомотив на кожну з сусідніх станцій, кожного разу перераховуючи нашу функцію при цьому запам’ятовуючи який внесок у неї робить кожний вагон. Далі визначаємо наскільки змінить переміщення кожного вагона на дану станцію значення функції:

$$l_{s1}(w_i) + kl_t(w_i) - l_{s2}(w_i), \tag{2.2}$$

де $l_{s1}(w_i)$ – відстань від станції на якій ми перебуваємо до кінцевої станції вагона;

$l_{s2}(w_i)$ – відстань від станції-кандидата на переміщення до кінцевої станції вагона.

Сортуємо вагони на станції за рівнем цього внеску і вибираємо вагони, які зменшують цю функцію, стільки скільки зможемо увезти цим локомотивом. Цю операцію робимо з кожною станцією та вибираємо найвигіднішу з них.

2.2 Рішення за допомогою машинного навчання

2.2.1 Нейронні мережі

Штучні нейронні мережі є галуззю машинного навчання і лежать в основі алгоритмів глибокого навчання. Їх ім’я та структура натхнені людським мозком, імітуючи спосіб сигналізації біологічних нейронів один одному.

Штучні нейронні мережі складаються з шарів вузлів, що містять вхідний шар, один або кілька прихованих шарів та вихідний шар. Кожен вузол, або штучний нейрон, з'єднується з іншим і має відповідну вагу та поріг. Якщо вихід будь-якого окремого вузла перевищує вказане порогове значення, цей вузол активується, надсилаючи дані на наступний рівень мережі. В іншому випадку дані не передаються на наступний рівень мережі.

Для навчання та підвищення точності нейронним мережам потрібні дані навчання. Однак, як тільки ці алгоритми навчання точно налаштовані, вони є потужними інструментами в галузі інформатики та штучного інтелекту, що дозволяє класифікувати та кластеризувати дані з великою швидкістю. Завдання з розпізнавання мови чи розпізнавання зображень можуть зайняти хвилини проти годин у порівнянні з ідентифікацією людей, проведеною вручну. Однією з найбільш відомих нейронних мереж є алгоритм пошуку Google.

Нейронні мережі навчаються, обробляючи приклади, кожен з яких містить відомі "вхідні дані" та "точно правилі результати", з яких нейромережі утворюють зважені за ймовірністю асоціації між даними які зберігаються в структурі даних самої мережі. Навчання нейронної мережі з поданого прикладу зазвичай проводиться шляхом визначення різниці між оброблюваним виходом мережі (часто передбачуваним) та цільовим результатом. У цьому полягає помилка. Потім мережа коригує свої зважені асоціації відповідно до правила навчання та використовуючи це значення помилки. Послідовне коригування призведе до того, що нейронна мережа буде

виробляти результат, який все більше нагадує цільовий результат. Після достатньої кількості цих коригувань навчання може бути припинено на основі певних критеріїв. Це відоме як навчання з учителем.

Такі системи "вчяться" виконувати завдання, розглядаючи приклади, як правило, не запрограмовані на конкретні завдання. Наприклад, при розпізнаванні зображень вони можуть навчитися ідентифікувати зображення, що містять котів, аналізуючи приклади зображень, які вручну позначені як "кіт" або "не кіт", і використовуючи результати для ідентифікації котів на інших зображеннях. Вони роблять це без попереднього знання котів, наприклад, що у них є шерсть, хвости, вуса та котячі обличчя. Натомість вони автоматично генерують ідентифікаційні характеристики на прикладах, які вони обробляють.

Штучні нейронні мережі почалися як спроба використати архітектуру людського мозку для виконання завдань, з якими звичайні алгоритми мали небагато успіху. Незабаром вони переорієнтувались на покращення емпіричних результатів, здебільшого відмовляючись від спроб залишатися вірними своїм біологічним попередникам. Нейрони з'єднані між собою різними схемами, щоб вихід деяких нейронів став входом інших. Мережа утворює спрямований, зважений графік.

Штучна нейронна мережа складається з колекції змодельованих нейронів. Кожен нейрон є вузлом, який з'єднаний з іншими вузлами за допомогою зв'язків, що відповідають біологічним зв'язкам аксон-синапс-дендрит. Кожна ланка має вагу, яка визначає силу впливу одного вузла на інший.

Нейронні мережі складаються зі штучних нейронів, які концептуально походять від біологічних нейронів. Кожен штучний нейрон має вхідні дані і виробляє єдиний вихідний сигнал, який може бути надісланий кільком іншим нейронам. Вхідними даними можуть бути значення характеристик вибірки зовнішніх даних, таких як зображення або документи, або вони можуть бути виходами інших нейронів. Виходи кінцевих вихідних нейронів нейронної мережі виконують таке завдання, як розпізнавання об’єкта на зображенні.

Щоб знайти вихід нейрону, спочатку беремо зважену суму всіх входів, зважену вагами зв’язків від входів до нейрона. До цієї суми додаємо термін упередженості. Цю зважену суму іноді називають активацією. Потім ця зважена сума передається через (як правило, нелінійну) функцію активації для отримання вихідних даних. Початкові дані - це зовнішні дані, такі як зображення та документи. Кінцеві результати виконують завдання, наприклад, розпізнавання об’єкта на зображенні.

Мережа складається з синапсів, кожний синапс забезпечує вихід одного нейрона як вхід до іншого нейрона. Кожному синапсу присвоюється вага, яка відображає його відносну важливість. Даний нейрон може мати кілька вхідних і вихідних синапсів. Схему шарів нейромережі зображено на рисунку 2.1 [11].

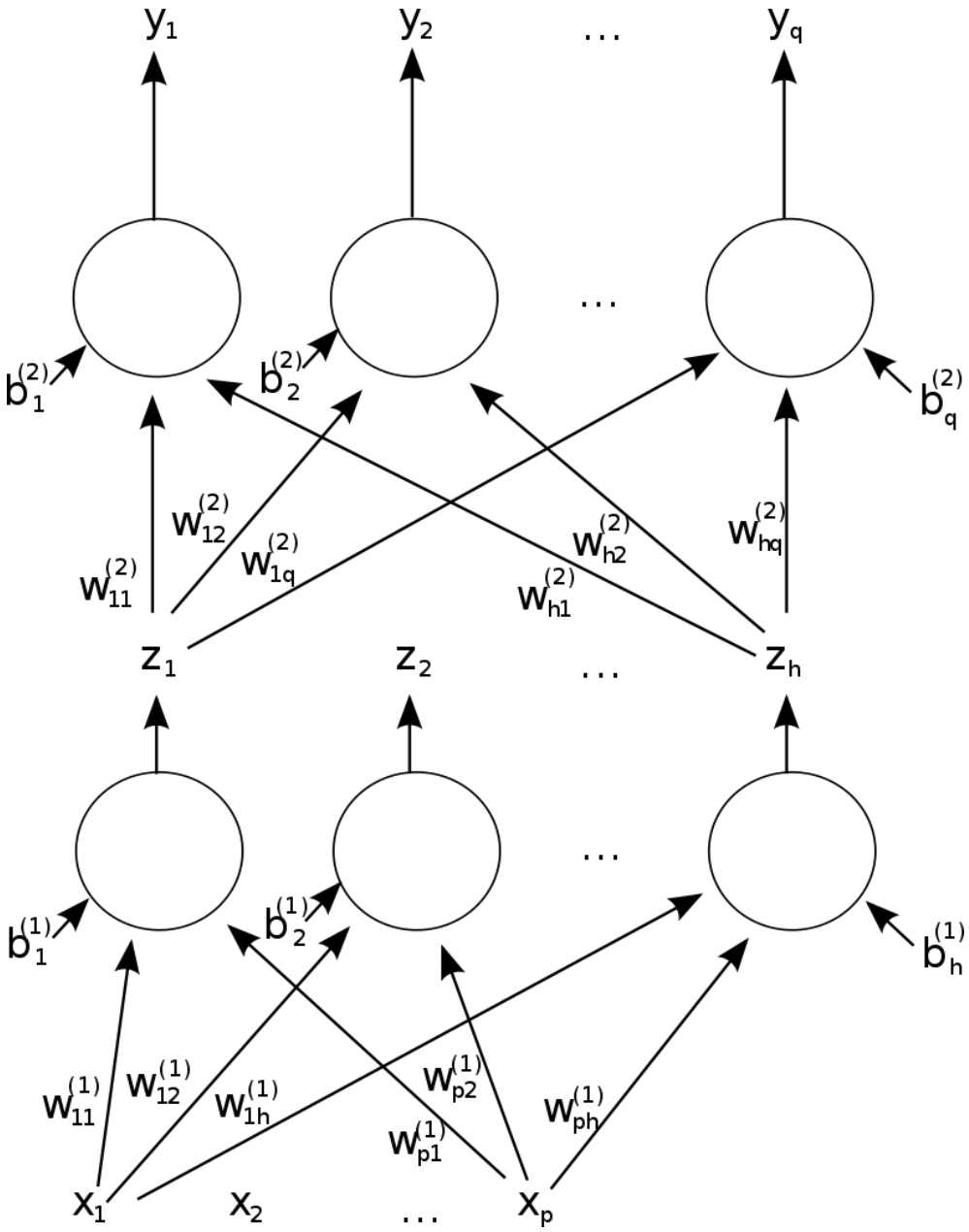


Рисунок 2.1 – Схема шарів нейронної мережі

Функція поширення обчислює вхід до нейрона з виходів нейронів-попередників та їх з'єднань у вигляді зваженої суми. До результату розповсюдження можна додати значення упередженості.

Нейрони зазвичай організовані в кілька шарів, особливо при глибокому навчанні. Нейрони одного шару з'єднуються лише з нейронами безпосередньо попереднього і безпосередньо наступного шарів. Шар, який приймає зовнішні дані, є вхідним шаром. Шар, який дає кінцевий результат, є вихідним шаром. Між ними знаходиться нуль або більше прихованих шарів. Також

використовуються одношарові та безшарові мережі. Між двома шарами можливі декілька шаблонів з'єднання. Вони можуть бути повністю зв’язані, при цьому кожен нейрон в одному шарі з’єднується з кожним нейроном у наступному шарі. Вони можуть об'єднуватися, коли група нейронів в одному шарі з'єднується з одним нейроном у наступному шарі, тим самим зменшуючи кількість нейронів у цьому шарі. Нейрони, що мають лише такі зв’язки, утворюють спрямований ациклічний граф і відомі як мережі прямого пересилання. Крім того, мережі, що дозволяють зв’язок між нейронами в тому ж або попередньому шарах, відомі як рекурентні мережі.

Гіперпараметр - це постійний параметр, значення якого встановлюється до початку навчального процесу. Значення параметрів отримуються шляхом навчання. Приклади гіперпараметрів включають швидкість навчання, кількість прихованих шарів та розмір партії. Значення деяких гіперпараметрів можуть залежати від значень інших гіперпараметрів. Наприклад, розмір деяких шарів може залежати від загальної кількості шарів.

Навчання - це адаптація мережі для кращого вирішення завдання, враховуючи вибіркові спостереження. Навчання передбачає регулювання ваг (і необов’язкових порогів) мережі для підвищення точності результату. Це робиться шляхом мінімізації спостережуваних помилок. Навчання завершено, коли вивчення додаткових спостережень не корисно зменшує рівень помилок. Навіть після навчання частота помилок зазвичай не досягає 0. Якщо після навчання рівень помилок занадто високий, мережа, як правило, повинна бути перенавчена. Практично це робиться шляхом визначення функції витрат, яка періодично оцінюється під час навчання. Поки обсяги виробництва продовжують знижуватися, навчання продовжується. Вартість часто визначається як статистика, значення якої може бути лише наближеною. Результати насправді є цифрами, тому, коли помилка низька, різниця між результатом (майже напевно кішка) і правильною відповіддю (кішка) невелика. Навчальні спроби зменшити загальну різницю між

спостереженнями. Більшість моделей навчання можна розглядати як пряме застосування теорії оптимізації та статистичного оцінювання.

Швидкість навчання визначає розмір коригувальних кроків, які модель робить, щоб коригувати помилки в кожному спостереженні. Високий рівень навчання скорочує час навчання, але з нижчою граничною точністю, тоді як менший рівень навчання займає більше часу, але з потенціалом для більшої точності. Такі оптимізації, як Quickprop, в основному спрямовані на пришвидшення мінімізації помилок, тоді як інші вдосконалення в основному намагаються підвищити надійність. Щоб уникнути коливань усередині мережі, таких як змінні ваги з'єднань, і поліпшити швидкість збіжності, уточнення використовують адаптивну швидкість навчання, яка відповідно збільшується або зменшується. Концепція імпульсу дозволяє зважувати баланс між градієнтом та попередньою зміною таким чином, що коригування ваги певною мірою залежить від попередньої зміни. Імпульс, близький до 0, підкреслює градієнт, тоді як значення, близьке до 1, підкреслює останню зміну.

Хоча можна визначити функцію витрат спеціально, часто вибір визначається бажаними властивостями функції (наприклад, опуклістю) або тому, що вона випливає з моделі.

Зворотне поширення - це метод, що використовується для регулювання ваги з'єднання для компенсації кожної помилки, виявленої під час навчання. Сума помилок ефективно ділиться між з'єднаннями. Технічно, зворотне поширення обчислює градієнт (похідну) функції витрат, пов'язаної з даним станом, відносно вагових коефіцієнтів. Оновлення ваги можна здійснювати за допомогою стохастичного градієнтного спуску або іншими методами [11].

2.2.2 Навчання з підкріпленням

Навчання з підкріпленням - це навчання моделей машинного навчання для прийняття послідовності рішень. Агент вчиться досягати мети в невизначеному, потенційно складному середовищі. Під час навчання з підкріпленням штучний інтелект стикається з ігровою ситуацією. Комп’ютер використовує методи спроб і помилок, щоб знайти рішення проблеми. Щоб машина зробила те, що хоче програміст, штучний інтелект отримує або винагороду, або покарання за дії, які він виконує. Його мета - максимізувати загальну винагороду.

Незважаючи на те, що дизайнер встановлює політику винагороди - тобто правила гри - він не дає моделі жодних підказок чи пропозицій щодо вирішення гри. Від моделі залежить, як виконати завдання, щоб максимізувати винагороду, починаючи від абсолютно випадкових випробувань і закінчуючи вишуканою тактикою та надлюдськими навичками. Використовуючи потужність пошуку та багато випробувань, навчання з підкріплення в даний час є найефективнішим способом демонстрації креативності машини. На відміну від людей, штучний інтелект може накопичувати досвід з тисяч паралельних навчань, якщо алгоритм навчання працює на достатньо потужній комп’ютерній інфраструктурі.

Основна проблема в навчанні з підкріплення полягає у підготовці середовища моделювання, яке сильно залежить від завдання, яке потрібно виконати. Коли модель повинна перевершити людину в іграх «Шахи», «Go» або «Atari», підготовка середовища моделювання є відносно простою. Коли мова заходить про створення моделі, здатної керувати автономним автомобілем, створення реалістичного тренажера має вирішальне значення перед тим, як дозволити їздити на вулиці. Модель повинна з'ясувати, як загальмувати або уникнути зіткнення в безпечному середовищі, де жертвування навіть тисячею автомобілів відбувається за мінімальну ціну. Перенесення моделі з тренувального середовища у реальний світ - це де справа стає складним.

Масштабування та налаштування нейронної мережі, що контролює агент, є ще однією проблемою. Існує не один спосіб спілкування з мережею, за винятком системи винагород та покарань. Це, зокрема, може призвести до катастрофічного забуття, коли отримання нових знань спричиняє стирання деяких старих з мережі.

Ще однією проблемою є досягнення локального оптимуму - тобто агент виконує завдання як є, але не оптимальним або необхідним способом. Чудовим прикладом є "стрибун", що стрибає, як кенгуру, замість того, щоб робити те, чого очікували від нього – гуляти.

Нарешті, є агенти, які оптимізують приз, не виконуючи завдання, для якого він був розроблений.

2.2.3 Опис рішення за допомогою машинного навчання

Для реалізації рішення задачі за допомогою машинного навчання є проблема в тому як представити вхідні дані, оскільки вони в першу чергу представлені у вигляді графу нефіксованого розміру, тому не очевидно як застосовувати звичайні нейромережі. Також дані неможна представити у вигляді потокової інформації, щоб застосовувати рекурентні нейронні мережі. Але зображення, які обробляють нейромережі, також можуть бути різного розміру, але перед аналізом їх розтягують або стискають до потрібного розміру. В певному розумінні таку ж операцію можна зробити і з графом. Наприклад коли ви знаходитись в Дніпрі, дивитись на мапу і шукаєте шлях до Лісабона, то ви скоріше за все не зможете утримувати в пам’яті одночасно всі дороги Європи. Спочатку ви будете оцінювати дороги біля Дніпра, а Лісабон буде для вас знаходитись просто десь далеко на заході. Потім ви вже будете аналізувати дороги у Польщі, Німеччині, Франції, Іспанії і тільки тоді, коли ви дійдете до Португалії, Лісабон з’явиться у вашому «полі зору». Аналогічно і тут, нехай нейронна мережа аналізує тільки десять найближчих вершин до

вершини на якій знаходиться локомотив для якого ми створюємо шлях. Будемо вважати що усі інші вершини є найближчим до них вершинами, які увійшли до множини, яку ми розглядаємо. Таким чином, ідея полягає в тому, що ми будемо йти вздовж списку локомотивів відсортованого за часом прибуття на наступну станції і запускати для кожного локомотива нейронну мережу, щоб визначити його подальший маршрут.

Тоді на кожній такій ітерації нейромережа буде приймати множину станцій, локомотивів та вагонів. Щоб представити множину станцій можна взяти N найближчих до локомотива залізних станцій та побудувати матрицю відстаней між ними. Діагональ цієї матриці нам не потрібна, одна з половинок також. Отже отримуємо трикутник з катетами по (N-1) клітинок, тобто всього (N*(N-1)/2) значень.

Множину вагонів можна представити як матрицю розміром в кількість станцій, які ми розглядаємо N*N, де кожна клітинка (i,j) це сума мас усіх вагонів на і-ій станції з кінцевою j-ою стацією. При цьому будемо вважати, що всі станції, які не увійшли у список, який ми розглядаємо, стають найближчими до них станціями з цього списку.

Множину локомотивів можна представити як вектор довжиною в кількість станцій, яку ми розглядаємо. Де кожне значення це кількість локомотивів на цій станції.

Отже нейромережа буде приймати вектор довжиною в N*(N-1)/2 + N*N + N значень, та повертати вектор довжиною 2N: перша половина – це наступна станція (найбільше значення), друга половина – вагони які потрібно взяти з собою (якщо і-ий елемент вектора більше за 0,5 беремо усі вагони, які прямують на цю і-ту станцію).

І так поки всі вагони не опиняться на станціях призначення. Тренувати нейронну мережу будемо за допомоги навчанням з підкріпленням. Спочатку тренувати будемо на невеликій системі 3-4 станції. З досягненням адекватних результатів будемо збільшувати розмір системи.

Але тут ще залишаються проблеми. Навчання з підкріпленням ще не дуже сильно розвинулося. Дана задача є складною через велику кількість можливих комбінацій початкової ситуації та ще більшу кількість варіантів рішення задачі. Для того щоб створити більш менш нормальну модель потрібно багато ресурсів, яких у мене немає. Та і якщо б були, то все одно ймовірність того, що вдасться створити модель більш ефективну за аналітичне рішення, не є великою. Тому, на мою думку, потрібно, ще почекати, поки в навчанні з підкріплення винайдуть нові технології, які значно збільшать ефективність систем, як це наприклад відбулося в сфері розпізнавання образів з винайденням згорткових нейронних мереж [6].

2.3 Висновки до розділу 2

В другому розділі було запропоновано можливі варіанти аналітичного рішення. Детально описано алгоритми цих рішень. При створенні алгоритмів була спроба організації співпраці локомотивів.

Також було розглянуто можливі варіанти рішення поставленої задачі за допомогою машинного навчання. Запропоновано рішення проблеми представлення вхідних даних даної задачі для нейронної мережі.

РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Опис програмної реалізації

Програмну реалізацію створено в Microsoft Visual Studio за допомогою технології Windows Forms мовою програмування C#. Дану середу розробки було вибрано через те, що в ній доволі просто створювати інтерфейс до невеликих програм і в автора вже був досвід користування нею.

Діаграму класів створеної програми можна побачити на рис. 3.1.

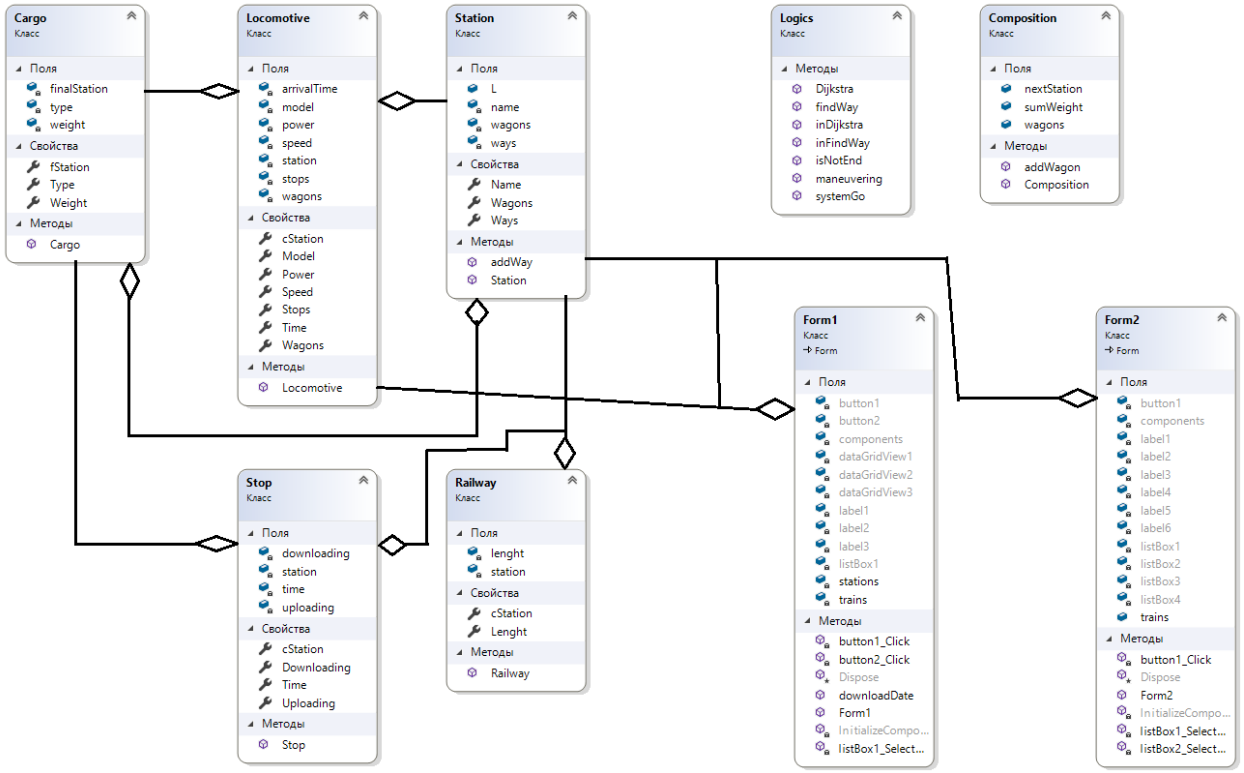


Рисунок 3.1 – Діаграма класів

Для запуску програми потрібно запустити файл RailwayLogistics.exe. У файлах Stations.xml, Locomotives.xml, Cargo.xml зберігаються вхідні дані для програми. Ці файли повинні бути в тому ж каталозі, що і програма.

3.2 Демонстрація інтерфейсу

При запуску програми відкриється форма для перегляду вхідних даних, при натисканні на кнопку «Зчитати дані» відбудеться зчитування даних з xml файлів, і в списку в лівій частині форми виведуться усі існуючі станції. При натисканні на станції у списку в інших трьох таблицях виведеться інформація про станції з якими вона сполучена та відстані до них, а також інформація про те, які вагони та локомотиви присутні на даній станції. При натисканні на кнопку «Запуск» запуститься алгоритм пошуку оптимального шляху та відкриється форма з результатами роботи алгоритму.

Інтерфейс на якому відображаються вхідні дані програми можна побачити на рис. 3.2

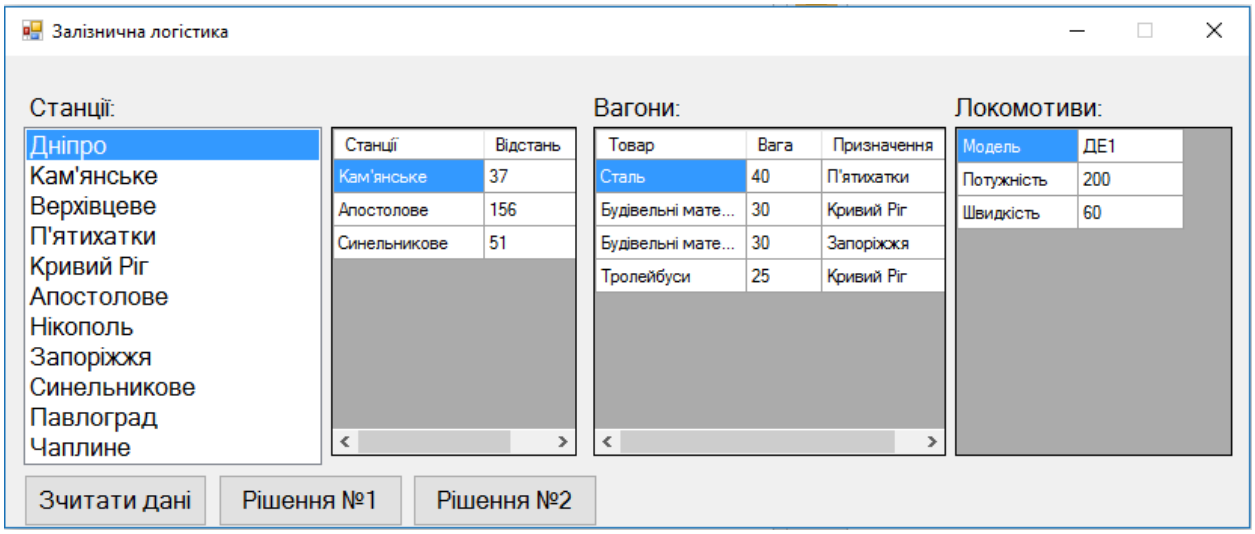


Рисунок 3.2 – Інтерфейс вхідних даних

На рисунку 3.3 зображено інтерфейс вхідних даних, в якому вибрана станція П’ятихатки.

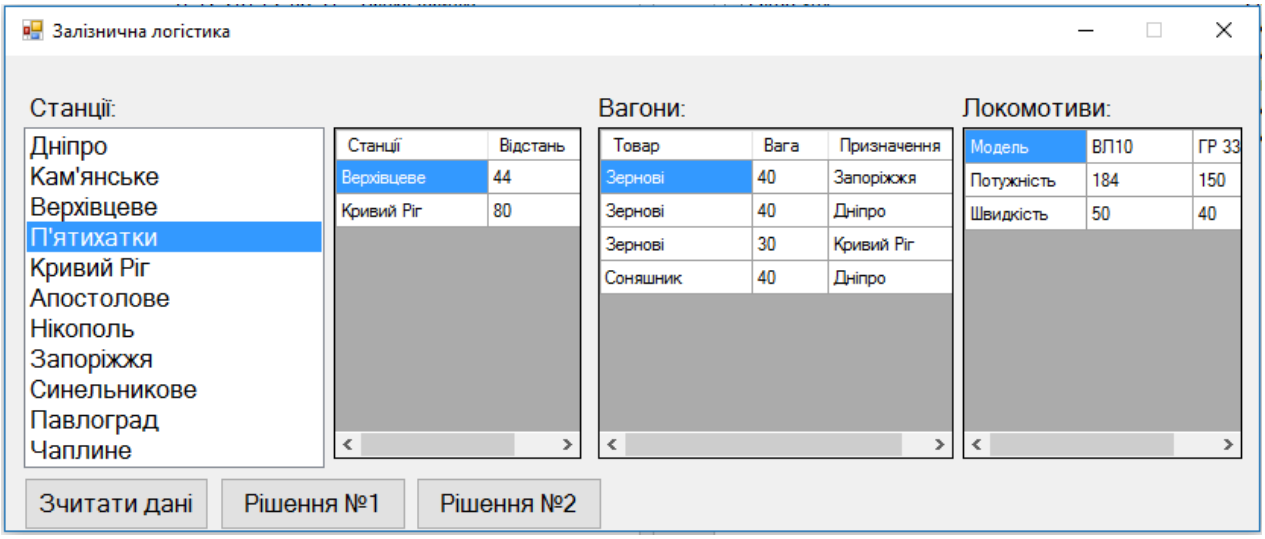


Рисунок 3.3 – Інтерфейс вхідних даних

Систему залізничних шляхів, яка використовувалася як вхідні дані програми в даному прикладі можна побачити на рисунку 3.4

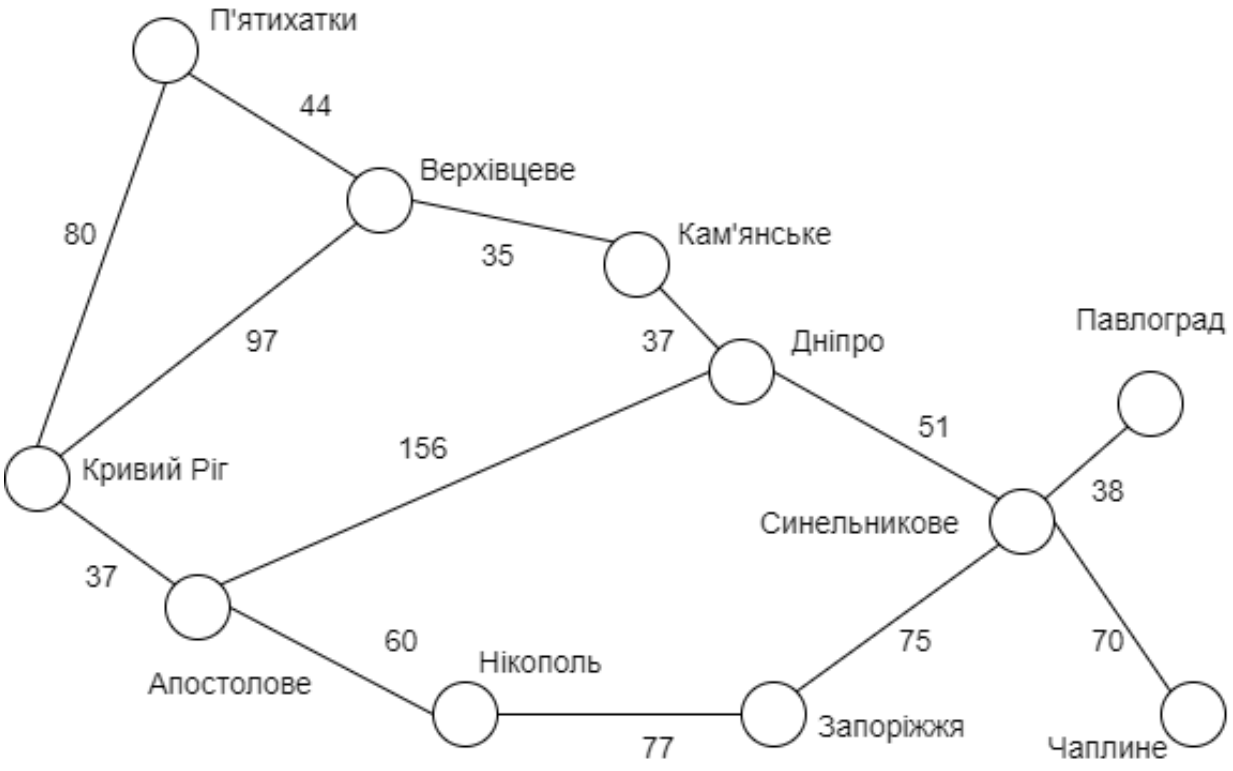


Рисунок 3.4 – Система залізниці, яка подана як вхідні дані для програми

Інтерфейс програми для виведення результатів першого аналітичного рішення зображено на рисиску 3.5.

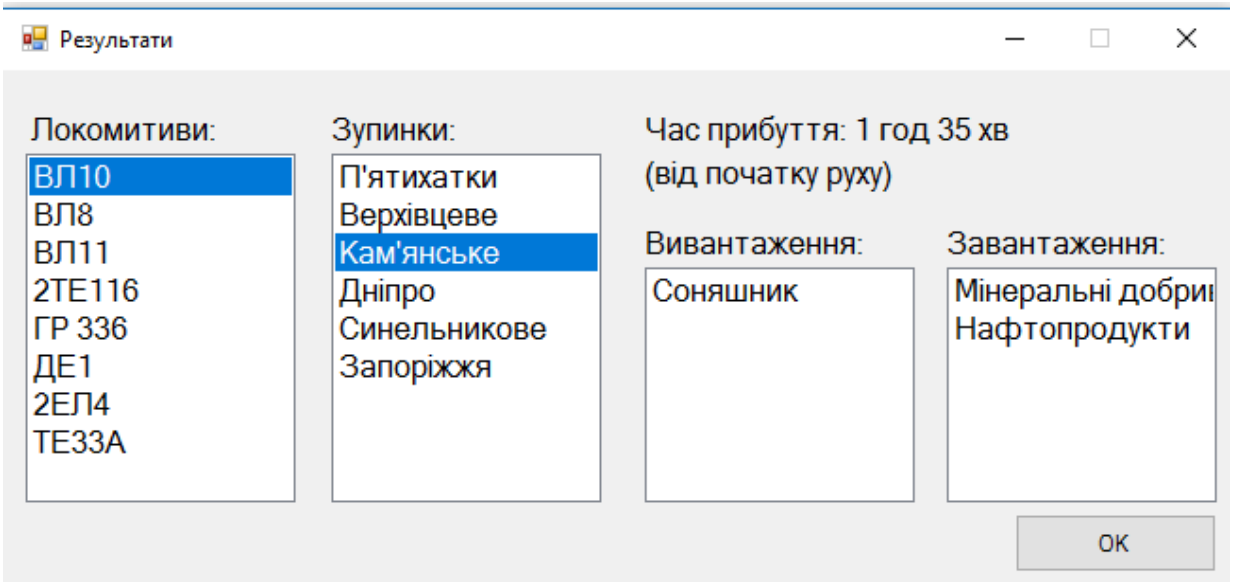


Рисунок 3.5 – Результати першого аналітичного рішення

На рисиску 3.6 зображено результати другого аналітичного рішення

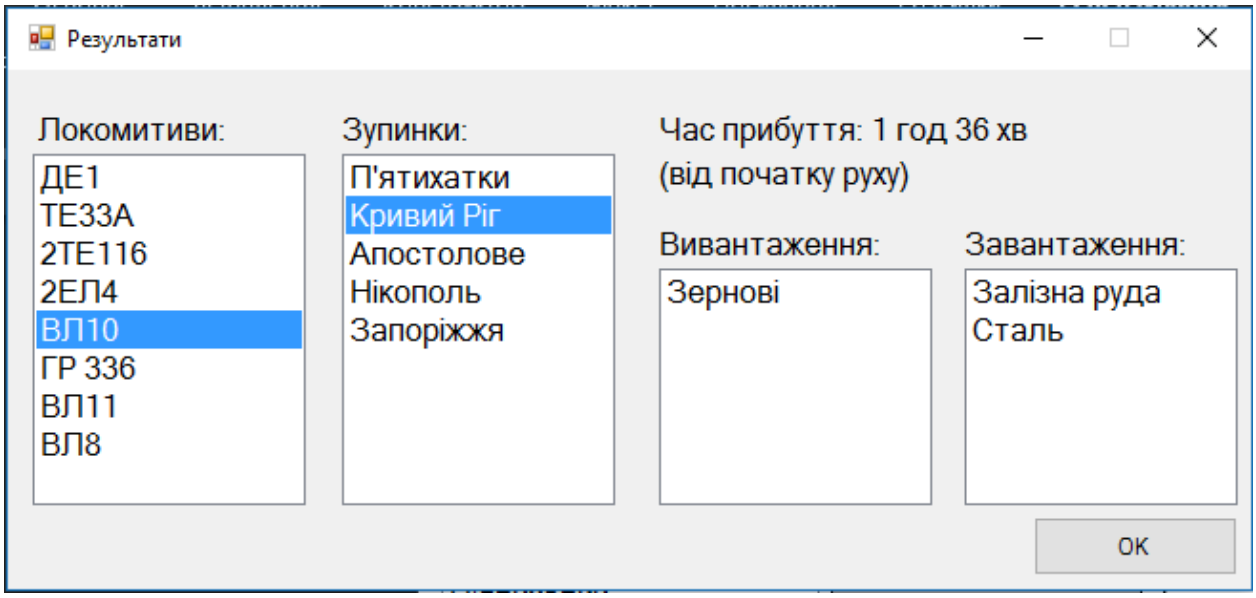


Рисунок 3.6 – Результати другого аналітичного рішення

При натисканні на локомотив у першому списку, у другому списку буде виводитися усі станції які проїжджає локомотив на своєму шляху. І при натисканні на зупинки у цьому списку, у правій частині форми буде виводитися детальна інформація про цю зупинку (час прибуття, списки вивантажених та завантажених вагонів з цього поїзда).

2.3 Висновки до розділу 3

В третьому розділі було створено програмну реалізацію запропонованих аналітичних рішень мовою програмування C# технологією Windows Forms. Презентовано архітектуру розробленого програмного забезпечення. Було описано інтерфейс програми та надано інструкцію з її користуванням. Також було продемонстровано роботу розроблених алгоритмів на прикладі системи залізничних шляхів Дніпропетровської області.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ.

4.1 Постановка задачі проектування

Проектований програмний продукт є додатком, в якому демонструється робота алгоритмів вирішення посталеної задачі складання розкладу для вантажних залізничних поїздів. Продукт може використовуватись операційній системі Windows.

4.2 Обґрунтування функцій та параметрів програмного продукту

Головна функція F_0 – розробка програмного продукту, який вирішує поставлену задачу складання розкладу для вантажних залізних поїздів. Беручи за основу цю функцію, можна виділити наступні:

F_1 – вибір мови програмування;

F_2 – вибір інтерфейсу програмного додатку;

F_3 – вибір середовища розробки.

Кожна з цих функцій має декілька варіантів реалізації:

Функція F_1:

а) C#

б) Python

Функція F_2:

а) Windows Forms;

б) Web-додаток

Функція F_3:

а) Jupyter Notebook;

б) Visual Studio.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1).

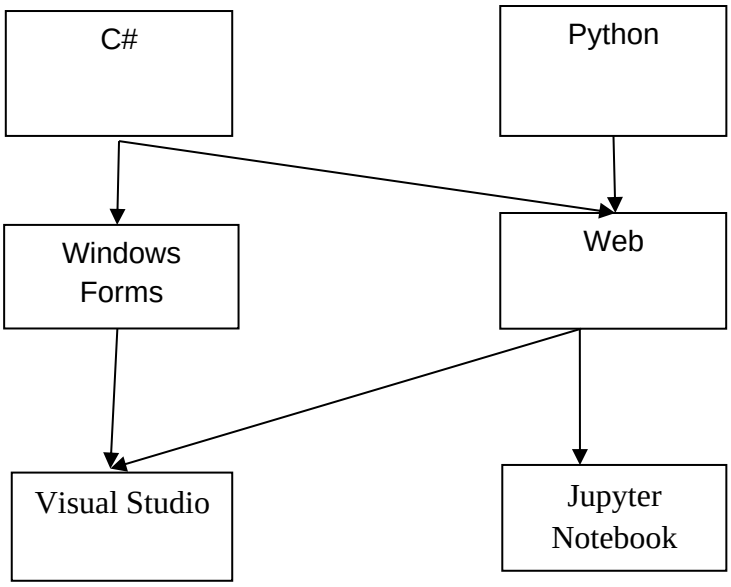


Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіанти основних функцій.

Таблиця 4.1 - Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	*А*	Просто розробити програмний інтрерфейс	Відсутня кросплатформність
	Б	Швидка розробка програми, доступність бібліотек, кросплатформеність	Низька швидкість роботи, особливо, якщо потрібно обробляти велику кількість даних
F_2	*А*	Дуже простий	Відсутня кросплатформність
	Б	Набагато складніше реалізувати	кросплатформність

F_3	*А*	Багато інструментів, безпечна	Підтримує одночасно лише одну мову програмування
	Б	Підтримується багатьма мовами програмування, легко запускається на будь-якому сервері	Відсутня можливість роботи без інтернету

На основі цієї карти будуємо позитивно-негативну матрицю варіантів основних функцій (Таб.4.1). Робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F_1:

Перевагу віддаємо простоті створення програмного інтерфейсу.

Функція F_2:

Обидва варіанти можна використовувати в розробці.

Функція F_3:

Віддаємо перевагу варіанту А в разі вибору мови програмування C#.

Таким чином, будемо розглядати такий варіанти реалізації ПП:

F_1а – F_2а – F_3а

F_1а – F_2б – F_3а

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

4.3 Обґрунтування системи параметрів ПП

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

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

- *X1* – швидкодія мови програмування;
- *X2* – об’єм пам’яті для обчислень та збереження даних;
- *X3* – час навчання даних;
- *X4* – потенційний об’єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у таблиці 5.4.

Таблиця 4.2 - Основні параметри ПП

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	оп/мс	10000	14000	19000
Об’єм пам’яті	X2	Мб	420	128	64
Час попередньої обробки даних	X3	мс	4	3	2
Потенційний об’єм програмного коду	X4	кількість рядків коду	4000	2500	1000

За даними таблиці 4.2 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

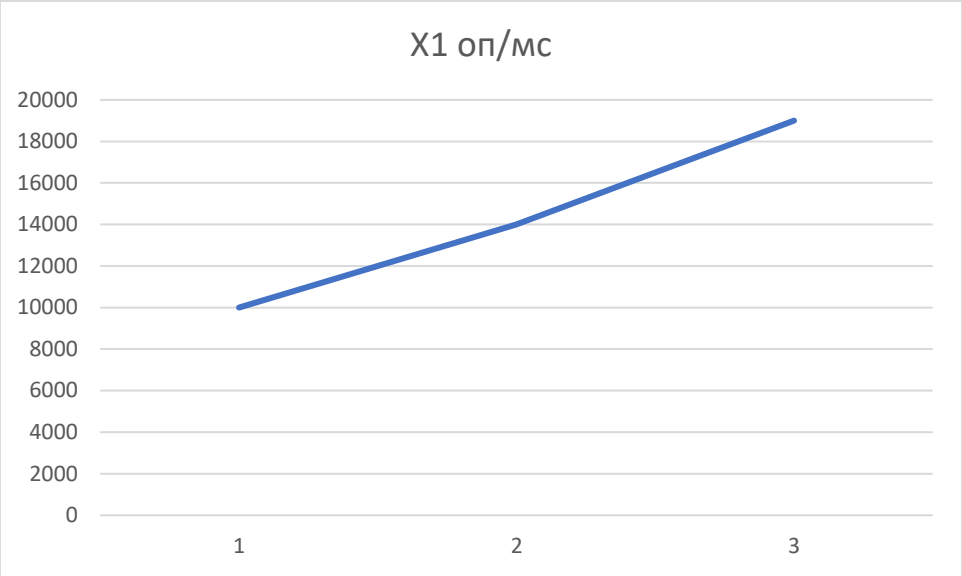


Рисунок 4.2 – X1, швидкодія мови програмування

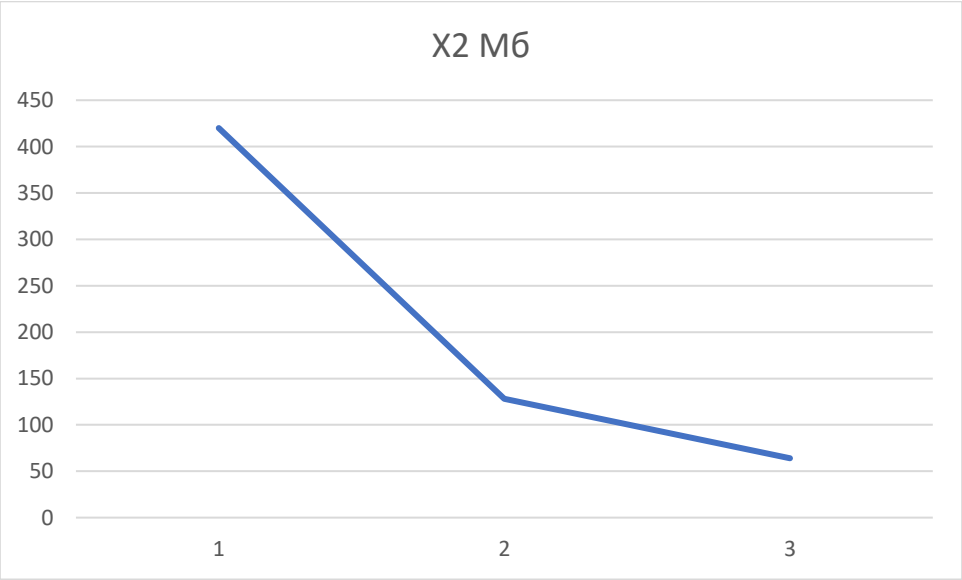


Рисунок 4.3 – X2, об’єм пам’яті

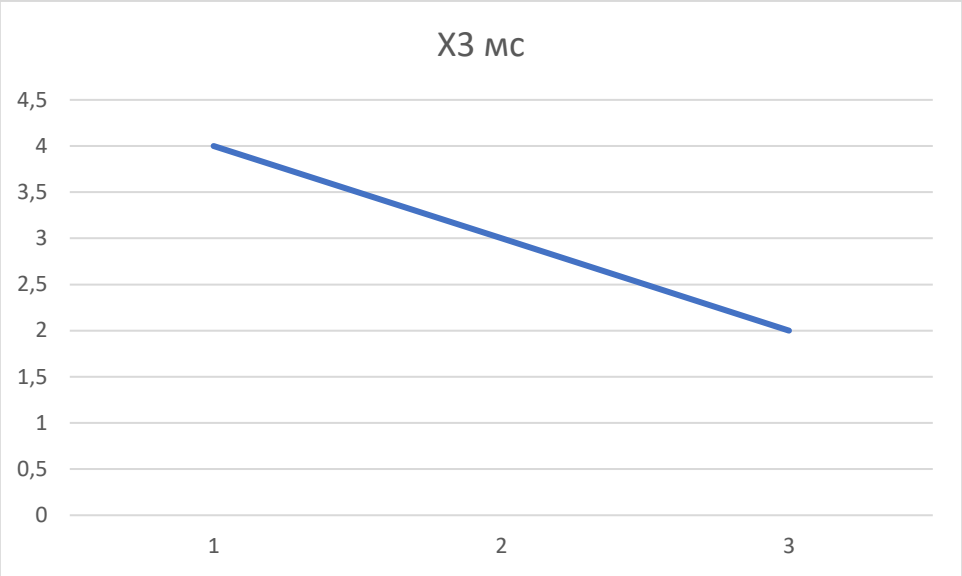


Рисунок 4.4 – X3, час попередньої обробки даних

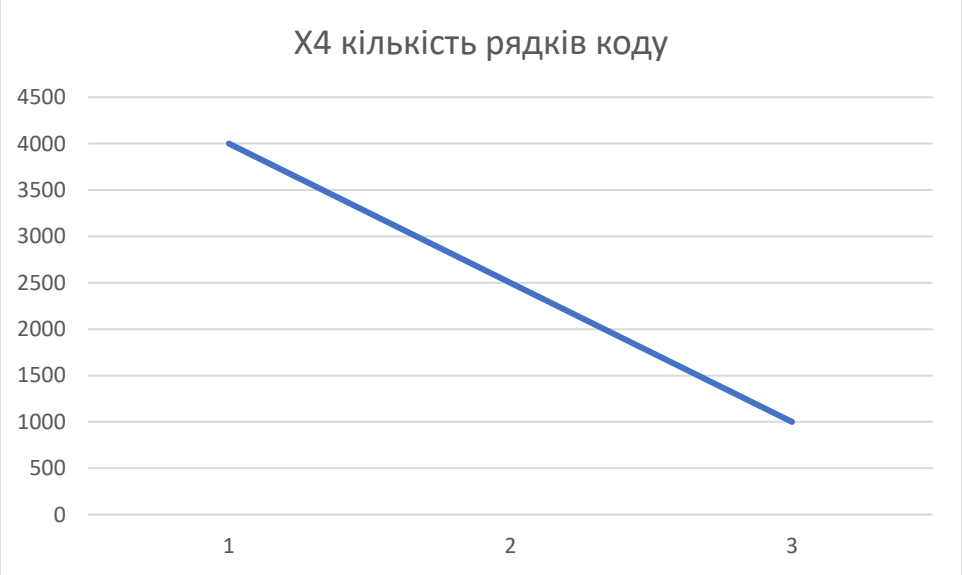


Рисунок 4.5 – X4, потенційний об’єм програмного коду

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні

параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

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

– визначення рівня значимості параметра шляхом присвоєння різних рангів;

– перевірку придатності експертних оцінок для подальшого використання;

– визначення оцінки попарного пріоритету параметрів;

– обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 - Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	Оп/мс	4	5	2	5	3	4	5	28	3,5	12,25
X2	Об’єм пам’яті	Mb	2	1	3	1	2	1	2	12	-12,5	156,25
X3	Час попередньої обробки даних	мс	5	3	5	5	4	5	3	30	5,5	30,25

X4	Потенційний об’єм програмного коду	Кількість рядків коду	3	5	4	3	5	4	4	28	3,5	12,25
	Разом		14	14	14	14	14	14	14	98	0	211

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

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

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 98, \quad (4.1)$$

де N – число експертів,

n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 24{,}5 \quad (4.2)$$

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

$$\Delta_i = R_i - T. \quad (4.3)$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 211. \quad (4.4)$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 211}{7^2(4^3 - 4)} = 0{,}86 > W_k = 0{,}67. \quad (4.5)$$

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

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Таблиця 4.4 - Попарне порівняння параметрів.

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

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

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}. \quad (4.6)$$

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

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

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i} \tag{4.7}$$

$$b_i = \sum_{i=1}^{N} a_{ij} \tag{4.8}$$

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

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \tag{4.9}$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j \tag{4.10}$$

.Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 - Розрахунок вагомості параметрів

Параметриx_i	Параметриx_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X1	1,0	1,5	0,5	1,5	4,5	0,36	17,75	0,25	73,38	0,25
X2	0,5	1,0	1,5	0,5	3,5	0,28	15,75	0,22	65,63	0,23
X3	1,5	1,5	1,0	1,5	5,5	0,44	22,75	0,33	93,6	0,32
X4	0,5	1,5	0,5	1,0	3,5	0,28	13,75	0,2	57,63	0,2
Всього:					12,5	1	70	1	290,25	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2* (Об'єм пам'яті), *X3* (час попередньої обробки даних) та *X4* (потенційний об'єм програмного коду) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X1* (швидкість роботи мови програмування) обрано не найгіршим.

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

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j}\,, \qquad (4.11)$$

де n – кількість параметрів;

$K_{вi}$– коефіцієнт вагомості i–го параметра;

B_i – оцінка i–го параметра в балах.

Таблиця 4.6 - Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	А	X1	10000	6	0,25	1,5
F2	А	X2	64	5	0,23	1,15
	Б	X2	128	3	0,32	0,96
F3	А	X3	1000	8	0,2	1,6

За даними з таблиці 4.6 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}], \qquad (4.12)$$

визначаємо рівень якості кожного з варіантів:

$$К_{К1} = 1,5 + 1,15 + 1,6 = 4,25,$$

$$К_{К2} = 1,5 + 0,96 + 1,6 = 4,06.$$

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

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка програмної оболонки;

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

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

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

Загальна трудомісткість обчислюється як

$$Т_О = Т_Р \cdot К_П \cdot К_{СК} \cdot К_М \cdot К_{СТ} \cdot К_{СТ.М}, \quad (4.13)$$

де $Т_Р$ – трудомісткість розробки ПП;

$К_П$ – поправочний коефіцієнт;

$К_{СК}$ – коефіцієнт на складність вхідної інформації;

$К_{М}$ – коефіцієнт рівня мови програмування;

$К_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$К_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

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

$$Т_1 = 90 \cdot 1.7 \cdot 0.8 = 122.4 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $Т_{Р}$ =27 людино-днів, $К_{П}$ =0.9,$К_{СК}$ = 1,$К_{СТ}$ =0.8:

$$Т_2 = 27 \cdot 0.9 \cdot 0.8 = 19.44 \text{ людино-днів.}$$

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

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

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

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 12000 грн., один аналітик в області даних з окладом 15000. Визначимо середню зарплату за годину за формулою:

$$С_ч = \frac{М}{T_m \cdot t} \text{грн.,} \quad (4.14)$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$С_ч = \frac{12000 + 12000 + 15000}{3 \cdot 21 \cdot 8} = 77{,}38 \text{ грн.} \quad (4.15)$$

Тоді, розрахуємо заробітну плату за формулою:

$$С_{ЗП} = С_ч \cdot Т_i \cdot К_Д\,, \quad (4.16)$$

де $С_ч$– величина погодинної оплати праці програміста;

$Т_i$ – трудомісткість відповідного завдання;

$К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 77{,}38 \cdot 1328{,}64 \cdot 1{,}2 = 123372{,}20$ грн.

II. $С_{ЗП} = 77{,}38 \cdot 1345{,}52 \cdot 1{,}2 = 124939{,}61$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0,22 = 123372,20 \cdot 0.22 = 27141,88$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0,22 = 124939,61 \cdot 0.22 = 27486,71$ грн.

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

Так як одна ЕОМ обслуговує одного програміста з окладом 12000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 12000 \cdot 0,2 = 28800 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 28800 \cdot (1 + 0.2) = 34560 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 34560 \cdot 0,22 = 7603,2 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 25000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.15 \cdot 0.25 \cdot 25000 = 7187,5 \text{ грн.,}$$

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$– річна норма амортизації;

$Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.15 \cdot 25000 \cdot 0.05 = 1437,5 \text{ грн.},$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 12 – 16) \cdot 8 \cdot 0.9 =$$

$$= 1677.6 \text{ годин},$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1677,6 \cdot 0,3 \cdot 0,92 \cdot 3,51 = 1625,19 \text{ грн.},$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$– коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 25000 \cdot 0,67 = 16750 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н, \quad (4.17)$$

$$С_{ЕКС} = 34560 + 603,2 + 7187,5 + 1437,5 + 1625,19 + 16750 = 69163,39 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 69163,39 / 1677.6 = 41,23 \text{ грн/год.}$$

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

$$С_М = С_{М-Г} \cdot Т, \quad (4.18)$$

I. $С_М = 41,23 \cdot 1328,64 = 54779,83 \text{ грн.}$

II. $С_М = 41,23 \cdot 1345.52 = 55475,79 \text{ грн.}$

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0,67, \quad (4.19)$$

I. $С_Н = 123372,20 \cdot 0,67 = 82659,37 \text{ грн.}$

II. $С_Н = 124939,61 \cdot 0,67 = 83709,54$ грн.

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н, \quad (4.20)$$

I. $С_{ПП} = 123372,20 + 27141,88 + 54779,83 + 82659,37 = 287953,28$ грн.

II. $С_{ПП} = 124939,61 + 27486,71 + 55475,79 + 83709,54 = 291611,65$ грн.

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР j} = К_{Кj} / С_{Фj}, \quad (4.21)$$

$$К_{ТЕР1} = 4,25/ 287953,28 = 1,48 \cdot 10^{-5},$$

$$К_{ТЕР2} = 4,06/ 291611,65 = 1,39 \cdot 10^{-5}.$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 1,48 \cdot 10^{-5}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У

нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 1,48 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

– мова програмування – C#;
– Використання Windows Forms
– Використання Visual Studio

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

4.8 Висновки до розділу 4

Проведено повний функціонально-вартісний аналіз програмного продукту. Визначено та проведено оцінку основних функцій програмного продукту. Визначено параметри, які характеризують програмний продукт. Проведено експертне оцінювання параметрів та аналіз якості варіантів реалізації функцій.

Проведено економічний аналіз варіантів розробки – трудомісткість, витрати на заробітну плату та інші витрати.

На основі аналізу вибрано варіант реалізації програмного продукту.

ВИСНОВКИ ПО РОБОТІ ТА РЕКОМЕНДАЦІЇ ЩОДО ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

У роботі було запропоновану декілька аналітичних рішень для розв’язання поставленої задачі залізничної логістики по створенню розкладу для вантажних поїздів. Також було надані міркування для рішення задачі за допомогою машинного навчання. Алгоритми видають адекватні рішення, проте вони не є повністю ідеальним, тому інколи вони можуть видавати неоптимальний маршрут, але через складність задачі важко перевірити на оптимальність результат.

Надалі планується вдосконалення програми: покращення роботи алгоритму, реалізація рішення даної задачі за допомогою машинного навчання, а також зміни у інтерфейсі.

У процесі підготовки даної роботи, були надбані практичні навички роботи у програмному середовищі Visual Studio 2017, на мові програмування C# . Робота виконана на основі об’єктно орієнтованого підходу.

ПЕРЕЛІК ПОСИЛАНЬ

1. Introduction: Origin and Impact of Railways. URL: https://archive.org/details/pictorialencyclo0000elli/page/n7/mode/2up

2. The Wollaton Wagonway of 1604. URL: https://www.island-publishing.co.uk/WRC_mirror/woll_wag_leaflet_a4.pdf

3. Історія виникнення і становлення логістики. URL: http://logisticstime.com/istoya/istoriya-logistiki/

4. Liverpool and Manchester. URL: http://www.spartacus.schoolnet.co.uk/RAliverpool.html

5. Промислова революція. URL: https://www.krugosvet.ru/enc/gumanitarnye_nauki/ekonomika_i_pravo/PROMISHLENNAYA_REVOLYUTSIYA.html

6. Richard S. Sutton, Andrew G. Barto. Reinforcement Learning. The MIT Press. – 165 p.

7. Что не так с обучением с подкреплением (Reinforcement Learning)? URL: https://habr.com/ru/post/437020/

8. Нейросеть. URL: https://basegroup.ru/deductor/function/algorithm/neuronet

9. Dijkstra's algorithm. URL: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

10. Нейронные сети — математический аппарат. URL: https://basegroup.ru/community/articles/math

11. Artificial neural network. URL: https://en.wikipedia.org/wiki/Artificial_neural_network

12. Залізничний транспорт. URL: https://uk.wikipedia.org/wiki/Залізничний транспорт

13. What is reinforcement learning? The complete guide. URL: https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/

14. New technology. URL: https://www.britannica.com/topic/logistics-military/New-technology

ДОДАТОК А КОД ПРОДУКТУ

Form1.cs

```
using RailwayLogistics.algorithm;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Xml;

namespace RailwayLogistics
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         dataGridView1.ColumnCount = 2;
      }

      List<Station> stations;
      List<Locomotive> trains;

      bool dataIsRead = false;

      private void button1_Click(object sender, EventArgs e)
      {
         listBox1.Items.Clear();
         downloadDate();
         dataIsRead = true;
      }

      private void listBox1_SelectedIndexChanged(object sender, EventArgs
e)
      {
         dataGridView1.ColumnCount = 2;
```

```
                dataGridView1.RowCount =
stations[listBox1.SelectedIndex].Ways.Count;
                dataGridView1.Columns[0].Name = "Станції";
                dataGridView1.Columns[1].Name = "Відстань";
                for(int i = 0; i<stations[listBox1.SelectedIndex].Ways.Count; i++)
                {
                   dataGridView1.Rows[i].Cells[0].Value =
stations[listBox1.SelectedIndex].Ways[i].cStation.Name;
                   dataGridView1.Rows[i].Cells[1].Value =
stations[listBox1.SelectedIndex].Ways[i].Lenght;
                }
                dataGridView2.ColumnCount = 3;
                if (stations[listBox1.SelectedIndex].Wagons.Count > 0)
                   dataGridView2.RowCount =
stations[listBox1.SelectedIndex].Wagons.Count;
                else
                {
                   dataGridView2.RowCount = 1;
                   dataGridView2.Rows[0].Cells[0].Value = "";
                   dataGridView2.Rows[0].Cells[1].Value = "";
                   dataGridView2.Rows[0].Cells[2].Value = "";
                }
                dataGridView2.Columns[0].Name = "Товар";
                dataGridView2.Columns[1].Name = "Вага";
                dataGridView2.Columns[1].Width = 50;
                dataGridView2.Columns[2].Name = "Призначення";
                for (int i = 0; i < stations[listBox1.SelectedIndex].Wagons.Count;
i++)
                {
                   dataGridView2.Rows[i].Cells[0].Value =
stations[listBox1.SelectedIndex].Wagons[i].Type;
                   dataGridView2.Rows[i].Cells[1].Value =
stations[listBox1.SelectedIndex].Wagons[i].Weight;
                   dataGridView2.Rows[i].Cells[2].Value =
stations[listBox1.SelectedIndex].Wagons[i].fStation.Name;
                }
                int c = 0;
                foreach(var loc in trains)
                {
                   if (loc.cStation.Name == stations[listBox1.SelectedIndex].Name)
c++;
                }
                dataGridView3.ColumnCount = c + 1;
                dataGridView3.RowCount = 3;
                dataGridView3.Rows[0].Cells[0].Value = "Модель";
```

```
            dataGridView3.Rows[1].Cells[0].Value = "Потужність";
            dataGridView3.Rows[2].Cells[0].Value = "Швидкість";
            dataGridView3.Columns[0].Width = 80;
            c = 1;
            foreach (var loc in trains)
            {
                if (loc.cStation.Name == stations[listBox1.SelectedIndex].Name)
                {
                    dataGridView3.Rows[0].Cells[c].Value = loc.Model;
                    dataGridView3.Rows[1].Cells[c].Value = loc.Power;
                    dataGridView3.Rows[2].Cells[c].Value = loc.Speed;
                    dataGridView3.Columns[c].Width = 70;
                    c++;
                }
            }
        }
        public void downloadDate()
        {
            XmlDocument xDoc = new XmlDocument();
            xDoc.Load("Stations.xml");
            XmlElement xRoot = xDoc.DocumentElement;
            stations = new List<Station>();
            foreach (XmlNode xnode in xRoot)
            {
                if (xnode.Attributes.Count > 0)
                {
                    XmlNode attr = xnode.Attributes.GetNamedItem("name");
                    if (attr != null)
                    {
                        stations.Add(new Station(attr.Value));
                        listBox1.Items.Add(attr.Value);
                    }
                }

            }
            int k = 0;
            foreach (XmlNode xnode in xRoot)
            {
                for (int i = 0; i < xnode.ChildNodes.Count; i += 2/* XmlNode
childnode in xnode.ChildNodes*/)
                {

                    for (int j = 0; j < stations.Count; j++)
                    {
                        if (stations[j].Name == xnode.ChildNodes[i].InnerText)
```

```
                    {
                        stations[k].Ways.Add(new Railway(stations[j],
int.Parse(xnode.ChildNodes[i + 1].InnerText)));
                    }
                }
            }
            k++;
        }
        xDoc.Load("Locomotives.xml");
        trains = new List<Locomotive>();
        xRoot = xDoc.DocumentElement;
        foreach (XmlNode xnode in xRoot)
        {
            XmlNode attr = xnode.Attributes.GetNamedItem("name");
            for (int i = 0; i < stations.Count; i++)
            {
                if (stations[i].Name == xnode.ChildNodes[0].InnerText)
                {
                    trains.Add(new Locomotive(attr.Value,
int.Parse(xnode.ChildNodes[2].InnerText),
int.Parse(xnode.ChildNodes[1].InnerText), stations[i]));
                }
            }
        }
        xDoc.Load("Cargo.xml");
        xRoot = xDoc.DocumentElement;
        foreach (XmlNode xnode in xRoot)
        {
            for (int i = 0; i < stations.Count; i++)
            {
                if (stations[i].Name == xnode.ChildNodes[2].InnerText)
                {
                    for (int j = 0; j < stations.Count; j++)
                    {
                        if (stations[j].Name == xnode.ChildNodes[3].InnerText)
                        {
                            stations[i].Wagons.Add(new
Cargo(xnode.ChildNodes[0].InnerText, int.Parse(xnode.ChildNodes[1].InnerText),
stations[j]));
                        }
                    }
                }
            }
        }
    }
```

```
        private void button2_Click(object sender, EventArgs e)
        {
            if (dataIsRead)
            {
                Logics.SystemGo(stations, trains);
                Form2 frm2 = new Form2(trains);
                //dataIsRead = false;
                frm2.Show();
                listBox1.Items.Clear();
                downloadDate();
            }
            else
            {
                MessageBox.Show("Немає даних");
            }
        }

        private void button3_Click(object sender, EventArgs e)
        {
            if (dataIsRead)
            {
                Logics2.Run(stations, trains);
                Form2 frm2 = new Form2(trains);
                //dataIsRead = false;
                frm2.Show();
                listBox1.Items.Clear();
                downloadDate();
            }
            else
            {
                MessageBox.Show("Немає даних");
            }
        }
    }
}
```

Form2.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace RailwayLogistics
{
   public partial class Form2 : Form
   {
      public List<Locomotive> trains;

      public Form2(List<Locomotive> trains)
      {
         InitializeComponent();
         this.trains = trains;
         foreach (var tr in trains)
            listBox1.Items.Add(tr.Model);
      }

      private void button1_Click(object sender, EventArgs e)
      {
         foreach(var tr in trains)
         {
            tr.Stops = new List<Stop>();
         }
         Close();
      }

      private void listBox1_SelectedIndexChanged(object sender, EventArgs
e)
      {
         listBox2.Items.Clear();
         foreach (var st in trains[listBox1.SelectedIndex].Stops)
            listBox2.Items.Add(st.cStation.Name);
         listBox2.SelectedIndex = 0;
      }

      private void listBox2_SelectedIndexChanged(object sender, EventArgs
e)
      {
         listBox3.Items.Clear();
         listBox4.Items.Clear();

if(trains[listBox1.SelectedIndex].Stops[listBox2.SelectedIndex].Uploading != null)
            foreach (var car in
trains[listBox1.SelectedIndex].Stops[listBox2.SelectedIndex].Uploading)
```

```
                    listBox3.Items.Add(car.Type);
                if
(trains[listBox1.SelectedIndex].Stops[listBox2.SelectedIndex].Downloading !=
null)
                    foreach (var car in
trains[listBox1.SelectedIndex].Stops[listBox2.SelectedIndex].Downloading)
                    listBox4.Items.Add(car.Type);
                double time =
trains[listBox1.SelectedIndex].Stops[listBox2.SelectedIndex].Time;
                label3.Text = "Час прибуття: " + (time - time % 1) + " год " +
Math.Round(time % 1 * 60) + " хв";
            }
        }
    }
    Cargo.cs

    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;
    using System.Threading.Tasks;

    namespace RailwayLogistics
    {
        public class Cargo
        {
            private string type;
            private int weight;
            private Station finalStation;

            public int L;

            public string Type { get { return type; } }
            public int Weight { get { return weight; } }
            public Station fStation { get { return finalStation; } }

            public Cargo(string type, int weight, Station finalStation)
            {
                this.type = type;
                this.weight = weight;
                this.finalStation = finalStation;
            }
        }
    }
```

Station.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace RailwayLogistics
{
   public class Station
   {
      private string name;
      private List<Railway> ways;
      private List<Cargo> wagons;

      public int L;

      public string Name { get { return name; } }
      public List<Railway> Ways { get { return ways; } }
      public List<Cargo> Wagons { get { return wagons; } set { wagons =
value; } }

      public Station(string name)
      {
         this.name = name;
         ways = new List<Railway>();
         wagons = new List<Cargo>();
      }
      public void addWay(Station station, int lenght)
      {
         ways.Add(new Railway(station, lenght));
      }
      public static bool operator ==(Station s1, Station s2) { return s1.Name
== s2.Name; }
      public static bool operator !=(Station s1, Station s2) { return s1.Name
!= s2.Name; }

   }

   public class Railway
   {
      private Station station;
      private int lenght;
```

```
            public Station cStation { get { return station; } }
            public int Lenght { get { return lenght; } }

            public Railway(Station station, int lenght)
            {
               this.station = station;
               this.lenght = lenght;
            }
         }
      }
```

Locomotive.cs

```
      using System;
      using System.Collections.Generic;
      using System.Linq;
      using System.Text;
      using System.Threading.Tasks;

      namespace RailwayLogistics
      {
         public class Locomotive
         {
            private string model;
            private int power;
            private int speed;
            private Station station;

            private double arrivalTime;
            private List<Cargo> wagons;
            private List<Stop> stops;

            public string Model { get { return model; } }
            public int Power { get { return power; } }
            public int Speed { get { return speed; } }
            public Station cStation { get { return station; } set { station = value; } }

            public double Time { get { return arrivalTime; } set { arrivalTime =
value; } }
            public List<Cargo> Wagons { get { return wagons; } set { wagons =
value; } }
            public List<Stop> Stops { get { return stops; } set { stops = value; } }

            public Locomotive(string model, int power, int speed, Station station)
```

```
            {
                this.model = model;
                this.power = power;
                this.speed = speed;
                this.station = station;;
                arrivalTime = 0;
                stops = new List<Stop>();
            }
        }

        public class Stop
        {
            private Station station;
            private double time;
            private List<Cargo> uploading;
            private List<Cargo> downloading;

            public Station cStation { get { return station; } }
            public double Time { get { return time; } }
            public List<Cargo> Uploading { get { return uploading; } set {
uploading = value; } }
            public List<Cargo> Downloading { get { return downloading; } set {
downloading = value; } }

            public Stop(Station station, double time)
            {
                this.station = station;
                this.time = time;
            }
        }
    }
```

Logics.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace RailwayLogistics
{
    public class Logics
    {
        //Алгоритм Дейкстри
```

```
        public static void Dijkstra(List<Station> stations, Station beginStation)
        {
            foreach(var st in stations)
            {
                st.L = int.MaxValue;
            }
            beginStation.L = 0;
            inDijkstra(beginStation);
        }
        private static void inDijkstra(Station station)
        {
            foreach(var way in station.Ways)
                if(way.cStation.L > station.L + way.Lenght)
                    way.cStation.L = station.L + way.Lenght;
            foreach (var way in station.Ways)
                if (way.cStation.L == station.L + way.Lenght)
                    inDijkstra(way.cStation);
        }

        //Знаходження найкоротшого шляху для вагону
        public static List<Station> findWay(Station end)
        {
            List<Station> list = new List<Station>();
            list.Add(end);
            inFindWay(end, list);
            list.Reverse();
            return list;

        }
        private static void inFindWay(Station station, List<Station> list)
        {
            foreach(var st in station.Ways)
            {
                if(st.cStation.L + st.Lenght == station.L)
                {
                    list.Add(st.cStation);
                    inFindWay(st.cStation, list);
                    break;
                }
            }
        }
        public static void maneuvering(Locomotive train, List<Station>
stations)
        {
            Stop stop = new Stop(train.cStation, train.Time);
```

```
                if (train.Wagons != null)
                {
                    train.cStation.Wagons.AddRange(train.Wagons);
                    //train.Wagons.RemoveAll();
                    foreach (var w in train.Wagons)
                        if (w.fStation == train.cStation)
                            train.cStation.Wagons.Remove(w);
                    stop.Uploading = new List<Cargo>(train.Wagons);
                    train.Wagons = null;
                }
                Dijkstra(stations, train.cStation);
                if (train.cStation.Wagons.Count > 0)
                {
                    //Сортування вагонів у состави за напрямком
                    List<Station>[] ways = new
List<Station>[train.cStation.Wagons.Count];
                    for (int i = 0; i < ways.Length; i++)
                    {
                        ways[i] = findWay(train.cStation.Wagons[i].fStation);
                    }
                    bool[] mas = new bool[ways.Length];
                    List<Composition> comp = new List<Composition>();
                    for (int i = 0, l = 0; i < ways.Length; i++)
                    {
                        if (!mas[i])
                        {
                            comp.Add(new Composition(train.cStation.Wagons[i],
ways[i][1]));
                            mas[i] = true;
                            for (int j = i + 1; j < ways.Length; j++)
                            {
                                if (ways[i][1] == ways[j][1] && comp[l].sumWeight +
train.cStation.Wagons[j].Weight < train.Power)
                                {
                                    comp[l].addWagon(train.cStation.Wagons[j]);
                                    mas[j] = true;
                                }
                            }
                            l++;
                        }
                    }
                    //Відправлення поїзда зі станції з найбільшим составом
                    Composition maxComposition = comp[0];
                    for (int i = 1; i < comp.Count; i++)
                    {
```

```
                    if (comp[i].sumWeight > maxComposition.sumWeight)
maxComposition = comp[i];
                }
                foreach (var w in maxComposition.wagons)
                    train.cStation.Wagons.Remove(w);
                train.Wagons = maxComposition.wagons;
                stop.Downloading = train.Wagons;
                foreach (var st in train.cStation.Ways)
                    if (st.cStation == maxComposition.nextStation)
                    {
                        train.Time += (double)st.Lenght / train.Speed;
                        train.cStation = st.cStation;
                        break;
                    }
            }
            else
            {
                int minL = int.MaxValue;
                Station s = stations[0];
                foreach (var st in stations)
                    if (st.Wagons.Count > 0 && minL > st.L)
                    {
                        minL = st.L;
                        s = st;
                    }
                if(minL != int.MaxValue)
                {
                    s = findWay(s)[1];
                    foreach (var st in train.cStation.Ways)
                        if (st.cStation == s)
                        {
                            train.Time += (double)st.Lenght / train.Speed;
                            train.cStation = st.cStation;
                            break;
                        }
                }
                else
                {
                    train.Time++;
                }
            }
            train.Stops.Add(stop);
        }
        public static void SystemGo(List<Station> stations, List<Locomotive>
trains)
```

```
        {
            while (IsNotEnd(stations, trains))
            {
                maneuvering(trains[0], stations);
                for (int i = 0; i < trains.Count - 1; i++)
                {
                    if (trains[i].Time > trains[i + 1].Time)
                    {
                        var t = trains[i];
                        trains[i] = trains[i + 1];
                        trains[i + 1] = t;
                    }
                    else break;
                }
            }
            foreach(var tr in trains)
            {
                for (int i = tr.Stops.Count - 1; i >= 0; i--)
                {
                    if (tr.Stops[i].Uploading == null && tr.Stops[i].Downloading
== null)
                    {
                        tr.Stops.RemoveAt(i);
                    }
                    else break;
                }
                foreach (var st in tr.Stops)
                {
                    if (st.Uploading != null && st.Downloading != null)
                        for (int i = st.Downloading.Count - 1; i >= 0; i--)
                            for (int j = st.Uploading.Count - 1; j >= 0; j--)
                                if (st.Downloading[i] == st.Uploading[j])
                                {
                                    //Console.WriteLine(st.Downloading[i].Type +
st.Uploading[j].Type);
                                    st.Downloading.RemoveAt(i);
                                    st.Uploading.RemoveAt(j);
                                    break;
                                }
                }
            }
        }
        public static bool IsNotEnd(List<Station> stations, List<Locomotive>
trains)
        {
```

```
            foreach (var st in stations)
                if (st.Wagons.Count > 0) return true;
            foreach (var tr in trains)
                if (tr.Wagons != null) return true;
            return false;
        }
    }

    public class Composition
    {
        public List<Cargo> wagons;
        public int sumWeight;
        public Station nextStation;

        public Composition(Cargo cargo, Station station)
        {
            wagons = new List<Cargo>();
            wagons.Add(cargo);
            sumWeight = cargo.Weight;
            nextStation = station;
        }
        public void addWagon(Cargo cargo)
        {
            wagons.Add(cargo);
            sumWeight += cargo.Weight;
        }
    }
}
```

Logics2.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace RailwayLogistics.algorithm
{
    public class Logics2
    {
        //"функція корисності"  сума відстаней вагонів до їх кінцевих станцій та до найближчих локомотивів
        public static int FunctionOfBenefits(List<Station> stations, List<Locomotive> trains)
```

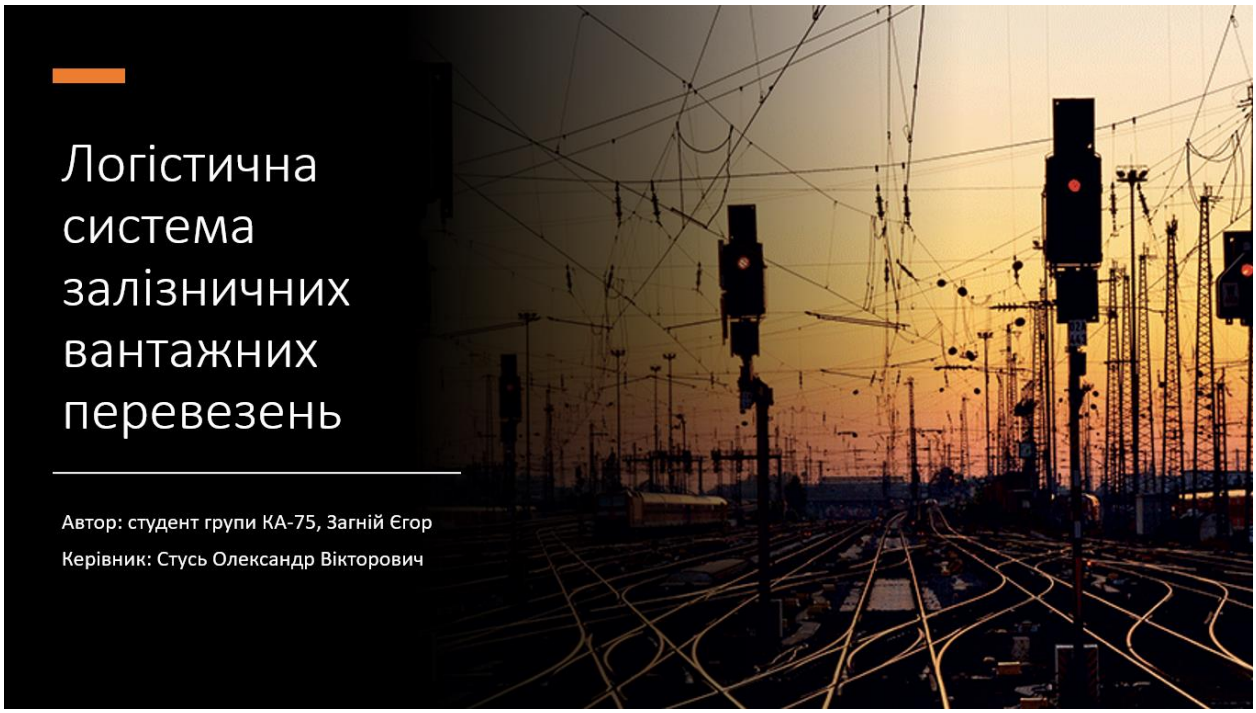
```
        {
            int benefit = 0;
            foreach(var st in stations)
            {
                Logics.Dijkstra(stations, st);

                int distToTrain = int.MaxValue;
                foreach(var tr in trains)
                {
                    if (tr.cStation.L < distToTrain) distToTrain = tr.cStation.L;
                }

                foreach(var wagon in st.Wagons)
                {
                    wagon.L = wagon.fStation.L + distToTrain;
                    benefit += wagon.L;
                }
            }
            return benefit;
        }
        public static int BenefitOfWagon(Cargo w)
        {
            return w.L - w.fStation.L;
        }
        public static void SortByBenefit(Station ourStation)
        {
            for (int i = 0; i < ourStation.Wagons.Count - 1; i++)
            {
                for (int j = i; j < ourStation.Wagons.Count - 1; j++)
                {
                    if (BenefitOfWagon(ourStation.Wagons[j]) >
BenefitOfWagon(ourStation.Wagons[j + 1]))
                    {
                        var t = ourStation.Wagons[j];
                        ourStation.Wagons[j] = ourStation.Wagons[j + 1];
                        ourStation.Wagons[j + 1] = t;
                    }
                }
            }
        }

        public static void FindNextStation(Locomotive train, List<Station>
stations, List<Locomotive> trains)
        {
            int benefit1 = FunctionOfBenefits(stations, trains);
```

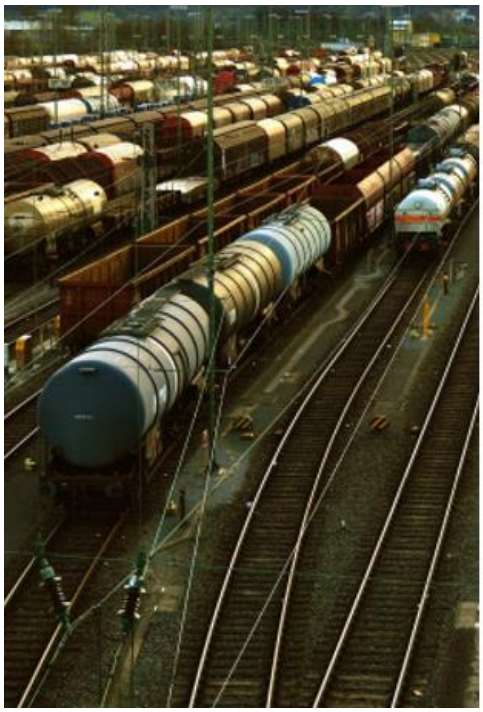
```
                Stop stop = new Stop(train.cStation, train.Time);
                if (train.Wagons != null)
                {
                    train.cStation.Wagons.AddRange(train.Wagons);
                    foreach (var w in train.Wagons)
                        if (w.fStation == train.cStation)
                            train.cStation.Wagons.Remove(w);
                    stop.Uploading = new List<Cargo>(train.Wagons);
                    train.Wagons = null;
                }
                var ways = train.cStation.Ways;
                int min = int.MaxValue;
                Railway iMax = null;
                int[] benefits = new int[ways.Count];
                var ourStation = train.cStation;
                foreach (var st in ways)
                {
                    train.cStation = st.cStation;
                    int count = FunctionOfBenefits(stations, trains);
                    Logics.Dijkstra(stations, st.cStation);
                    SortByBenefit(ourStation);
                    for (int i = 0; i < ourStation.Wagons.Count; i++)
                    {
                        if (BenefitOfWagon(ourStation.Wagons[i]) > 0)
                            count -= BenefitOfWagon(ourStation.Wagons[i]);
                    }
                    if (count < min)
                    {
                        min = count;
                        iMax = st;
                    }
                }

                train.cStation = iMax.cStation;
                int benefit = FunctionOfBenefits(stations, trains);
                Logics.Dijkstra(stations, iMax.cStation);
                SortByBenefit(ourStation);
                List<Cargo> toTrain = new List<Cargo>(), stayOnStation = new
List<Cargo>();
                foreach (var w in ourStation.Wagons)
                {
                    if (BenefitOfWagon(w) > 0) toTrain.Add(w);
                    else stayOnStation.Add(w);
                }
                train.Wagons = toTrain;
```

```
            ourStation.Wagons = stayOnStation;
            stop.Downloading = train.Wagons;
            train.Time += (double)iMax.Lenght / train.Speed;
            train.Stops.Add(stop);
        }
        public static void Run(List<Station> stations, List<Locomotive> trains)
        {
            while (IsNotEnd(stations, trains))
            {
                FindNextStation(trains[0], stations, trains);
                for (int i = 0; i < trains.Count - 1; i++)
                {
                    if (trains[i].Time > trains[i + 1].Time)
                    {
                        var t = trains[i];
                        trains[i] = trains[i + 1];
                        trains[i + 1] = t;
                    }
                    else break;
                }
            }
            foreach (var tr in trains)
            {
                for (int i = tr.Stops.Count - 1; i >= 0; i--)
                {
                    if (tr.Stops[i].Uploading == null && tr.Stops[i].Downloading
== null)
                    {
                        tr.Stops.RemoveAt(i);
                    }
                    else break;
                }
                foreach (var st in tr.Stops)
                {
                    if (st.Uploading != null && st.Downloading != null)
                        for (int i = st.Downloading.Count - 1; i >= 0; i--)
                            for (int j = st.Uploading.Count - 1; j >= 0; j--)
                                if (st.Downloading[i] == st.Uploading[j])
                                {
                                    //Console.WriteLine(st.Downloading[i].Type +
st.Uploading[j].Type);
                                    st.Downloading.RemoveAt(i);
                                    st.Uploading.RemoveAt(j);
                                    break;
                                }
```

```
                }
                for (int i = tr.Stops.Count - 1; i >= 0; i--)
                {
                    if (tr.Stops[i].Uploading.Count == 0) tr.Stops.RemoveAt(i);
                    else break;
                }
            }
        }
        public static bool IsNotEnd(List<Station> stations, List<Locomotive>
trains)
        {
            foreach (var st in stations)
                if (st.Wagons.Count > 0) return true;
            foreach (var tr in trains)
                if (tr.Wagons.Count > 0) return true;
            return false;
        }
    }
}
```

ДОДАТОК Б ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ





Постановка задачі

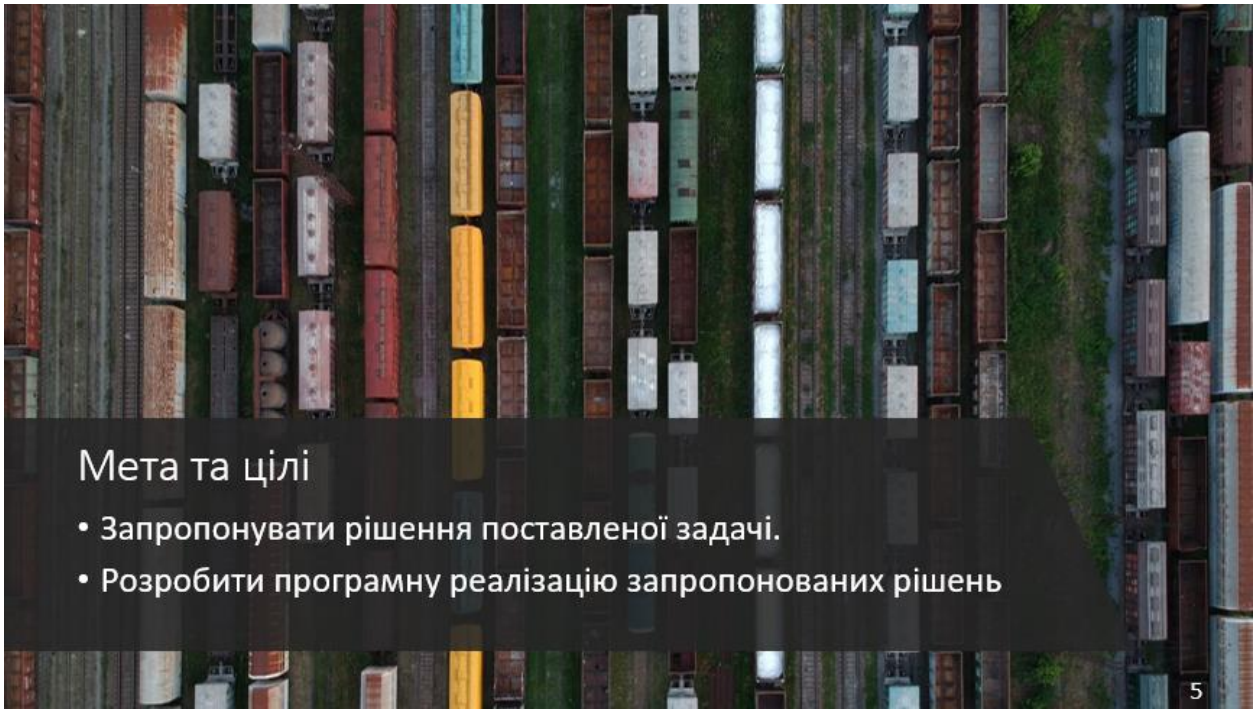
Відома система залізничних шляхів, яка представлена у вигляді графу, де вершини - це станції, а вага зв'язку це довжина шляху. На кожній станції є множина вагонів, кожний з яких має вагу і кінцеву станцію. Також є множина локомотивів, кожен з яких має характеристики потужності та швидкості. Необхідно побудувати маршрути для кожного локомотива з найменшими витратами палива та часу, так щоб кожен з вагонів опинився на станції кінцевого призначення.

2

- Об’єкт дослідження: логістика вантажних перевезень.
- Предмет дослідження: логістична система вантажних залізничних перевезень.

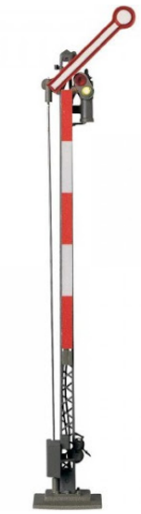
Актуальність

- Дана робота є спробою автоматизації й оптимізації процесу складання розкладу для вантажних залізничних поїздів, що економить людські та майнові ресурси і скорочує час доставки товарів.
- Запропоновані рішення можна використовувати в будь-якій іншій сфері від доставки пошти до морських вантажних перевезень.



Аналітичне рішення №1

- Вибираємо локомотив з найменшим часом прибуття на наступну станцію.
- Алгоритмом Дейкстри визначаємо відстані від станції, на якій знаходиться локомотив, до усіх інших.
- Знаходимо найкоротші шляхи для кожного з вагонів на даній станції.
- Групуємо вагони на групи в залежності від наступної станції в їх шляху.
- Відправляємо локомотив з найбільшою групою на призначену станцію. Визначаємо час прибуття на наступну станцію.



6

Аналітичне рішення №2

Оцінка ситуації:

$$\sum_{i=1}^{N}(l_s(w_i) + kl_t(w_i))$$

де w_i - це і-ий вагон, N - кількість вагонів,

$l_s(w_i)$ - відстань від і-ого вагона до його кінцевої станції призначення,

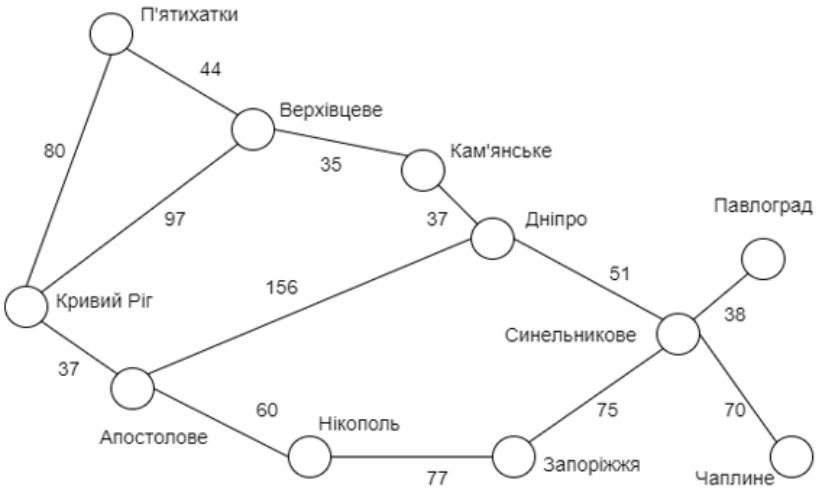
$l_t(w_i)$ - відстань від і-ого вагона до найближчого до нього локомотива,

k - коефіцієнт важливості відстані до локомотива.

Аналітичне рішення №2

- Вибираємо локомотив з найменшим часом прибуття на наступну станцію.
- Почергово розміщуємо локомотив на станції з множини наступних станцій-кандитатів.
- Рахуємо оцінку даної ситуації, при цьому запам’ятовуючи, який внесок в неї робить кожний вагон.
- Перебираємо усі вагони на станції, на якій насправді знаходиться локомотив, визначаємо їх внесок у оцінку ситуації, якщо їх перемістити на санцію-кандидата.
- Визначаємо різницю внесків кожного вагону, у випадку переміщення та непереміщення. Сумуємо усі позитивні різниці.
- Визначаємо станцію, в котрій була найбільша сума різниць внесків вагонів у оцінку ситуації. Вона і буде наступною станцією для локомотива.
- Причеплюємо до локомотива вагони з позитивнею різницею і визначаємо час прибуття на наступну станцію.

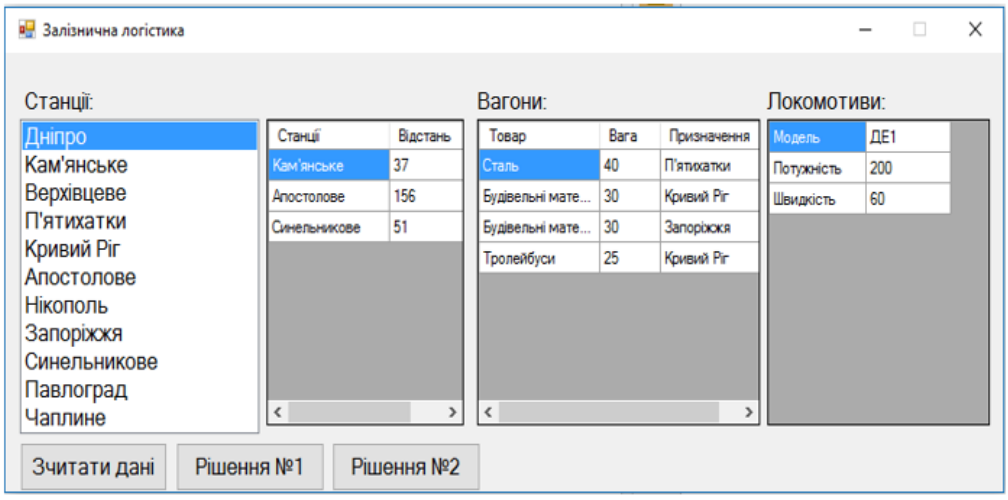
Приклад



На цій системі залізниць будемо тестувати алгоритм

9

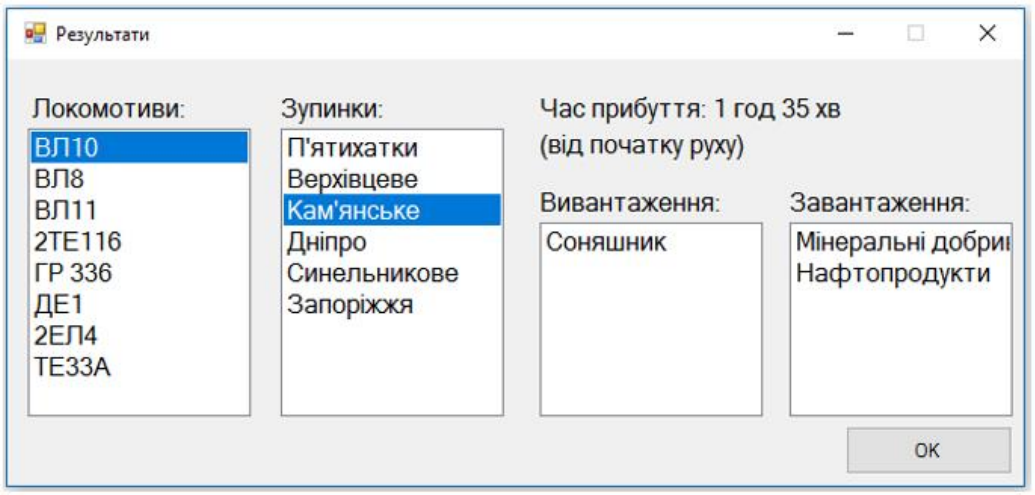
Програмна реалізація



Інтерфейс вхідних даних

10

Програмна реалізація



Результати аналітичного рішення

11

Рішення за допомогою машинного навчання

Вхідні дані

Система доріг і станцій:

W_{11}	W_{12}	W_{13}	W_{14}	W_{15}
W_{21}	W_{22}	W_{23}	W_{24}	W_{25}
W_{31}	W_{32}	W_{33}	W_{34}	W_{35}
W_{41}	W_{42}	W_{43}	W_{44}	W_{45}
W_{51}	W_{52}	W_{53}	W_{54}	W_{55}

W_{ij} – відстань між і-ою та j-ою станціями

Розмір: N(N-1)/2

12

Рішення за допомогою машинного навчання

Вхідні дані

Вагони:

C_{11}	C_{12}	C_{13}	C_{14}	C_{15}
C_{21}	C_{22}	C_{23}	C_{24}	C_{25}
C_{31}	C_{32}	C_{33}	C_{34}	C_{35}
C_{41}	C_{42}	C_{43}	C_{44}	C_{45}
C_{51}	C_{52}	C_{53}	C_{54}	C_{55}

C_{ij} – вага всіх вагонів на і-ій станції з j-ою кінцевою станцією.

Розмір: N*N

13

Рішення за допомогою машинного навчання

Вхідні дані

Локомотиви:

t_1	t_2	t_3	t_4	t_5

t_i – к-сть локомотивів на і-ій станції кінцевою станцією.

Розмір: N

14

Рішення за допомогою машинного навчання

Вихідні дані

Наступна станція:

S_1	S_2	S_3	S_4	S_5

i, s_i = max(S) – номер наступної станції

Розмір: N

Вагони:

C_1	C_2	C_3	C_4	C_5

$\bar{C} = \{i, c_i > 0{,}5\}$ – зчеплюємо з локомотивом всі вагони з і-ою кінцевою станцією

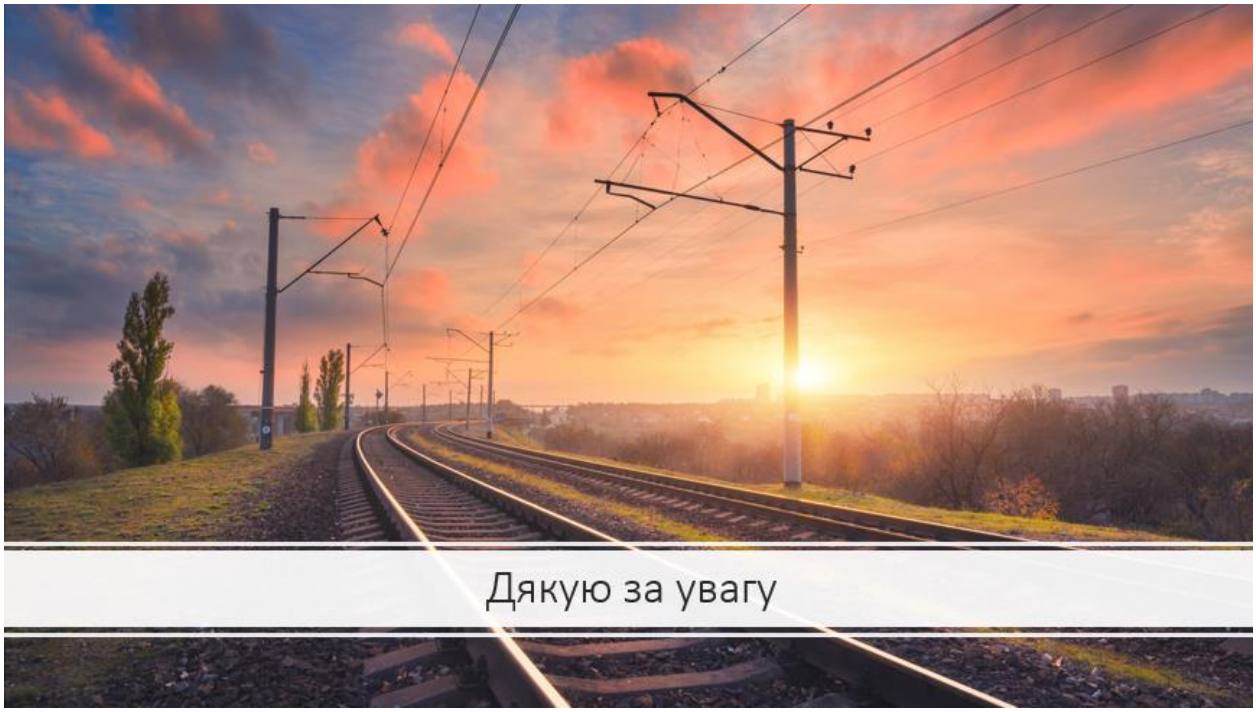
Розмір: N

15

Висновки

- Для поставленої задачі були запропоновані аналітичні рішення та надані роздуми над рішенням за допомогою машинного навчання.
- Було створено програмний продукт в якому було реалізовано аналітичне рішення задачі.
- Розроблений алгоритм видає раціональні рішення задачі. Із-за складності задачі їх складно перевірити на оптимальність. Спроектувати універсальний алгоритм, який завжди буде видавати оптимальний результат, скоріше за все є неможливим.

16


Дякую за увагу