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

Факультет інформатики та обчислювальної техніки
Кафедра автоматизованих систем обробки інформації та управління

"На правах рукопису"
УДК 004.023:004.052

До захисту допущено
В.о. завідувача кафедри

_____ Олександр ПАВЛОВ
)
“ ____ ” _____ 20 *20* р.

МАГІСТЕРСЬКА ДИСЕРТАЦІЯ

на здобуття ступеня магістра

за освітньо-професійною програмою

«*Інформаційні управляючі системи та технології*»

зі спеціальності 126 «*Інформаційні системи та технології*»

на тему:

«Система з підтримки процесу верифікації програмного продукту»

Виконала:
студентка VI курсу, групи **ІС-92мп**
Шишман Юлія Михайлівна _____

Керівник:
доцент, к.т.н., доцент,
Жданова Олена Григорівна _____

Консультант:
професор, д.т.н., доцент,
Жаріков Едуард В’ячеславович _____

Рецензент:
доцент, к.т.н., доцент,
Репнікова Наталія Борисівна _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ – 2020 року

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

Факультет інформатики та обчислювальної техніки
Кафедра автоматизованих систем обробки інформації та управління

Рівень вищої освіти – *другий (магістерський)*

Спеціальність – 126 «*Інформаційні системи та технології*»

Освітньо-професійна програма *«Інформаційні управляючі системи та технології»*

В.о.з авідувача кафедри

_____ Олександр ПАВЛОВ

«__» _____2020 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Шишман Юлія Михайлівна

1. Тема дисертації «**Система з підтримки процесу верифікації програмного продукту**», науковий керівник дисертації Жданова Олена Григорівна, доцент, к.т.н., доцент, затверджені наказом по університету від «26» жовтня 2020 р. № 3132-с

2. Строк подання студентом дисертації “ *2* ” *12* 20*20* р.

3. Об’єкт дослідження – процес тестування програмних продуктів.

4. Перелік завдань, які потрібно розробити
Проаналізувати існуючі методи дослідження автоматизації програмного забезпечення, провести аналіз відомих робіт з розв’язання поставленої в рамках роботи задачі, розробити методи побудови плану тестування в залежності від обраної стратегії, розробити метаевристичні алгоритми покриття множини, виконати програмну реалізацію розроблених алгоритмів, налагодити інтеграцію з

TeamCity, проаналізувати дані експериментів, провести порівняльний аналіз розроблених алгоритмів обраної стратегії.

5. Орієнтовний перелік графічного (ілюстративного) матеріалу

1. Схема структурна контекстної моделі системи. 2. Діграма діяльності побудови планів тестування. 3. Діаграма послідовності роботи розробленого фреймворку. 4. Схема структурна класів програмного забезпечення. 5. Блок-схема алгоритму генерації популяції. 6. Екранні форми. 7. Результати експериментів.

6. Орієнтовний перелік публікацій

Дві публікації: одна стаття на міжнародну науково-практичну конференцію та одні тези доповіді на науково-практичній конференції

7. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

8. Дата видачі завдання “ *1* ” *вересня* 20 *20* р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів магістерської дисертації	Примітка
1	*Систематизація результатів огляду літератури*	*14.09*	
2	*Аналіз існуючих методів розв’язання задачі*	*23.09*	
3	*Постановка та формалізація математичної моделі задачі*	*11.10*	
4	*Розробка метаевристичних алгоритмів розв’язання задачі*	*19.10*	
5	*Розробка інформаційного та програмного забезпечення*	*01.11*	
7	*Проведення експериментальних досліджень розроблених алгоритмів*	*10.11*	
8	*Оформлення документації*	*18.11*	
9	*Подання роботи на попередній захист*	*20.11*	
10	*Подання роботи на основний захист*	*02.12*	

Студент _____ *Юлія ШИШМАН*

Науковий керівник _____ *Олена ЖДАНОВА*

РЕФЕРАТ

Магістерська дисертація: 91 с., 23 рис., 7 табл., 37 джерел, 1 додаток.

Актуальність. Розробка якісного програмного продукту - це складний процес, який вимагає високого рівня підготовки команди розробників. Задля підвищення якості та роботоспроможності програмних продуктів потрібно велику увагу приділяти тестуванню.

Ручне тестування в деяких випадках займає більше часу для перевірки системи, в той час як автоматизоване тестування дозволяє значно заощадити витрати компанії-клієнта, зекономити ресурси та час на тестування та підтримку високоякісної продукції та зменшити ризик випуску неякісних продуктів або продуктів, які не відповідаюсь потребам замовника. Ось чому технологія автоматизації тестування дуже популярна серед компаній, що займаються інформаційними технологіями, робота яких пов’язана з розробкою програмного забезпечення.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалась на кафедрі автоматизованих систем обробки інформації та управління Національного технічного університету України «Київський політехнічний інститут ім. Ігоря Сікорського» в рамках теми «Ефективні методи розв'язання задач теорії розкладів» (№ ДР 0117U000919).

Мета дослідження - підвищення якості та надійності програмного продукту за рахунок автоматизації процесу тестуваннння, що дозволяє скоротити час та витрати на виконання процесів тестування.

Для досягнення мети необхідно виконати наступні **завдання:** –

проаналізувати існуючі методи автоматизації тестування програмного забезпечення;

– провести аналіз відомих робіт з розв’язання поставленої в рамках роботи задачі;

– розробити методи побудови плану тестування в залежності від обраної стратегії;

– розробити метаевристичні алгоритми покриття множини для вирішення задачі побудови плану тестування;

– виконати програмну реалізацію розроблених алгоритмів;

– налагодити інтеграцію з TeamCity;

– проаналізувати дані обчислювальних експериментів для визначення параметрів алгоритмів;

– провести порівняльний аналіз розроблених алгоритмів обраної стратегії.

Об’єкт дослідження – процес тестування програмних продуктів.

Предмет дослідження – методи автоматизації верифікації програмних продуктів на основі побудови планів тестування програмних продуктів.

Наукова новизна одержаних результатів полягає у впровадженні системи для підтримки автоматизованого тестування, яка потребує мінімального втручання людини для знаходження помилок та невідповідностей у роботі програмних продуктів.

Публікації. Матеріали роботи опубліковані у збірнику IX Міжнародної науково-практичної конференції “DYNAMICS OF THE DEVELOPMENT OF WORLD SCIENCE” 13-15 травня 2020 у Ванкувері та у тезах всеукраїнської науково-практичної конференції молодих вчених та студентів «Інформаційні системи та технології управління» НТУУ «КПІ ім. Ігоря Сікорського» 26-27 листопада 2020 р .

АВТОМАТИЗАЦІЯ, ТЕСТУВАННЯ ПРОГРАМНИХ ПРОДУКТІВ, ПОБУДОВА ПЛАНІВ ТЕСТУВАННЯ, ПОКРИТТЯ МНОЖИНИ, МЕТАЕВРИСТИЧНІ АЛГОРИТМИ, ГЕНЕТИЧНИЙ АЛГОРИТМ, АЛГОРИТМ МУРАШИННИХ КОЛОНІЙ

ABSTRACT

Master’s thesis: 91 p., 23 pic., 7 tab., 37 sources, 1 supplement.

Topicality. Developing a quality software product is a complex process that requires a high level of training from the development team. In order to improve the quality and performance of software products, great attention should be paid to testing.

Manual testing in some cases takes more time to test the system, while automated testing can significantly reduce the cost of customer companies, save resources and time used to test and maintain high product quality, and reduce the risk of marketing a substandard product. or a product that does not meet the needs of users. That is why testing automation technologies are quite popular in information technology companies, whose work is related to software development.

Relationship of work with scientific programs, plans, themes. The work was performed at the Department of Computer-Aided Management And Data Processing Systems of the National Technical University of Ukraine «Igor Sikorsky Kyiv Polytechnic Institute» within the topic «Effective methods for solving problems of schedule theory» (state registration number 0117U000919)

The purpose of the study is to improve the quality and reliability of the software product by automating the testing process, which reduces the time and cost of testing.

To achieve this goal you need to perform the following tasks:

– to analyze existing methods for software testing automation;

– to analyze the known work to solve the problem set in the work;

– to develop methods for building a testing plan depending on the chosen strategy;

– to develop metaheuristic algorithms for covering the set to solve the problem of constructing a test plan;

– to perform software implementation of the developed algorithms;

– to establish integration with TeamCity;

– to analyze the data of computational experiments to determine the parameters of algorithms;

– to carry out the comparative analysis of the developed algorithms of the chosen strategy.

Object of research is the process of software testing.

Subject of research is methods of automation of verification on the basis of construction of plans of testing of software products.

The scientific novelty of the obtained results is the introduction of a system to support automated testing, which requires minimal human intervention to find errors and inconsistencies in the operation of software products**.**

AUTOMATION, TESTING OF SOFTWARE PRODUCTS, CONSTRUCTION OF TESTING PLANS, COVER SET, METAHEURISTIC ALGORITHMS, ANT COLONY OPTIMIZATION, GENETIC ALGORITHM

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПП – програмний продукт

ПЗ – програмне забезпечення

ГА – генетичний алгоритм

ACO – мурашинний алгоритм

TC – TeamCity

ТЗ – технічне завдання

ПК – персональний комп’ютер

ІТ – інформаційні технології

ВСТУП

Тестування - це процес дослідження технології, який виконується на вимогу замовників, та призначений для розкриття інформації про якість продукту відносно середовища його використання. Цей процес передбачає виконання програми з метою знайдення невідповідностей та помилок [1]. Методи тестування ПЗ, що існують на сьогоднішній день, не дозволяють у повній мірі виявити всі невідповідності та встановити правильність функціонування аналізованої програми, всі існуючі методи тестування діють в рамках формального процесу перевірки досліджуваного ПЗ, що розробляється. Такий процес формальної перевірки може довести, що з погляду використовуваного методу дефекти відсутні. Іншими словами, враховуючи людські фактори, що існують на кожному етапі життєвого циклу програмного забезпечення, неможливо точно визначити або гарантувати відсутність дефектів програмного продукту. Існує багато підходів до рішення завдання тестування і перевірки програмного забезпечення, але ефективне тестування складних ПЗ — це дослідницький процес, такий, що не зводиться до проходження строгим і чітким процедурам або створенню таких.

За стандартом ISO 9126, якість програмних засобів визначається як сукупність характеристик досліджуваного ПЗ з урахуванням таких складових, як надійність, ефективність, практичність, супровід, функціональність, мобільність.

Перші програмні системи були розроблені в рамках програм наукових досліджень або програм, які задовольняють потреби міністерств оборони. Тестування таких програмних продуктів проводилося строго формалізовано із записом всіх тестових даних, тестових процедур, отриманих результатів. Тестування виділялося як окремий процес, який починався після завершення кодування, та, як правило, виконувалося тим же персоналом [2].

Вже у 1980-х роках тестування було розширено, включаючи такі поняття, як попередження дефектів. Проектування тестів – це найбільш ефективний з усіх відомих методів попередження помилок. В той же час стали

висловлюватися думки, що необхідна методологія тестування, зокрема, що тестування повинне включати перевірки протягом циклу розробки, та це повинен бути контрольований процес. В ході тестування треба перевірити не тільки розроблену та зібрану програму, але й вимоги, архітектуру, код, самі тести. «Традиційне» тестування, яке існувало до початку 1980-х років, відносилося тільки до скомпільованої, готової системи (наразі це зазвичай називається системне тестування), але надалі тестери стали залучатися до всіх аспектів життєвого циклу розробки. Це дозволяло раніше знаходити проблеми у вимогах і архітектурі і тим самим скорочувати терміни і бюджет розробки. В середині 1980-х років з'явилися перші інструменти для автоматизованого тестування. Передбачалося, що комп'ютер зможе проводити більше тестів, ніж людина, і зробить це більш надійнішим. Спочатку ці інструменти були надзвичайно простими, без можливості написання сценаріїв на скриптових мовах.

На початку 1990-х років в поняття «тестування» почали включати проектування, планування, створення, підтримку і виконання тестів, та це означало перехід від звичайного тестування до забезпечення якості, яке охоплює важливий цикл розробки ПЗ. В той час починають з'являтися різні програмні засоби та інструменти для підтримки процесу тестування - просунуті середовища для автоматизації з можливістю створення скриптів і генерації звітів, системи управління тестами, ПЗ для проведення тестування навантаження [2]. В середині 1990-х років з розвитком Інтернету та розробкою великої кількості web-додатків популярність стало отримувати «гнучке тестування» (по аналогії з гнучкими методологіями програмування).

У 2000-х роках з'явилося ще ширше визначення терміну тестування, в нього було додано поняття «Оптимізація бізнес-технологій», що направляє розвиток IT відповідно до цілей бізнесу. Основний метод полягає в оцінці та максимізації важливості кожного етапу життєвого циклу розробки ПЗ для досягнення необхідного рівня доступності, якості, продуктивності.

Сучасна наука забезпечує багато способів вирішення завдань тестування та верифікації ПЗ, проте ефективне тестування нових та складних програмних продуктів – це завжди креативий процес, який не обмежується чіткими та одноманітними процедурами. Він вимагає від тестувальника розгляду кожного випадку в індивідуальному порядку.

Для розробки ПЗ витрачається достатньо велика кількість коштів, адже кожен замовник хоче отримати якісний та надійний результат, покриття певної кількості функціональності, тому впровадження тестування як однієї з основних частин розробки ПП є особливо важливою та існує саме для забезпечення якості та ефективності програмного продукту, підтримки робочого стану системи. Тому завдання забезпечення якості розроблюємого ПЗ є одним із найактуальніших в індустрії ІТ.

1 ПРОЕКТНІ РІШЕННЯ З РОЗРОБКИ СИСТЕМИ З ПІДТРИМКИ ПРОЦЕСУ ВЕРИФІКАЦІЇ ПРОГРАМНОГО ПРОДУКТУ

1.1 Загальні положення

Тестування ПЗ (*Software Testing*) — це процес перевірки відповідності заявлених вимог та реально реалізованої функціональності, який виконується шляхом спостереження за його роботою в штучно створених ситуаціях та на обмеженому наборі даних[3]. Техніка тестування також включає не тільки процес пошуку дефектів та помилок, а й випробування програмних складових системи з метою оцінки. Може оцінюватись:

- виконання функцій за прийнятний час
- правильна відповідь для усіх можливих вхідних даних;
- практичність;
- відповідність вимогам, якими керувалися проектувальники та розробники;
- сумісність з програмним забезпеченням та операційними системами;
- відповідність задачам замовника.

Розглянемо детальніше основні види тестування, які найбільш широко застовуються у наш час [4]:

- module testing;
- integration testing;
- unit testing;
- system testing;
- acceptance testing.

Модульне тестування (Module testing) – оцінка програмного забезпечення щодо деталізації проектування, дозволяє перевірити коректність окремих модулів початкового коду програми.

Відноситься до тестів, які перевіряють функціональність певного розділу коду, зазвичай на функціональному рівні. В об'єктно-орієнтованому

середовищі - це тестування на рівні класу, а мінімальні модульні тести включають в себе конструктори та деструктори.

Такі тести зазвичай пишуться розробниками під час роботи над кодом (принцип «білої скриньки») для того, щоб впевнитись, що дана функція працює так, як очікується. Кілька тестів можуть перевіряти одну функцію, щоб переглянути всі випадки використання коду. Саме по собі модульне тестування не може перевірити функціонування частини ПЗ, воно використовується, щоб довести та гарантувати, що основні блоки ПЗ працюють незалежно один від одного.

Модульне тестування — це процес розробки ПЗ, який включає в себе синхронізовані застосування широкого спектру для запобігання невідповідностей, дефектів та для виявлення стратегій з метою зменшення ризиків розробки ПЗ, витрат та часу. Воно виконується інженером або розробником ПЗ під час будівельної фази життєвого циклу ПЗ. Даний вид тестування спрямований на усунення дефектів проектування. Ця стратегія спрямована на підвищення та забезпечення якості одержуваного ПЗ до такого рівня, як того вимагає процес контролю якості. В залежності від очікуваної організації розробки програмного забезпечення, модульне тестування може включати статичний аналіз коду, аналіз покриття коду та інші методи перевірки ПЗ, аналіз потоку даних аналізу метрик та експертні оцінки коду [4].

При цьому в ході процесу модульного тестування вирішуються чотири основні завдання.

Підтримка розробки та рефакторинга низькорівневої архітектури системи і міжмодульної взаємодії - це завдання більш властиве "легким" методологіям типу XP, де використовується принцип тестування перед розробкою (test-driven development), де основним джерелом вимог для програмного модуля є тест, написаний до самого модуля. Проте, навіть при класичній схемі тестування модульне тестування може виявити проблеми в дизайні системи та заплутані або нелогічні механізми роботи з модулем.

Пошук і документування невідповідностей вимогам - це класичне завдання тестування, яке включає як розробку тестового оточення та тестових прикладів, так і виконання тестів, складання звітів про проблеми, протоколювання результатів виконання.

Підтримка усунення дефектів і налагодження - це завдання, яке отримують розробники від тестувальників у вигляді звітів про проблеми (баг-репортів). Детальні звіти про проблеми, що складені на етапі модульного тестування, дозволяють усунути та локалізувати багато дефектів в ПП на ранніх стадіях його розробки або розробки нової функціональності.

Підтримка рефакторинга модулів - це завдання, що пов'язане з підтримкою процесу зміни системи, адже доволі часто в ході розробки необхідно проводити рефакторінг модулів - оптимізацію або повну переробку програмного коду задля підвищення його швидкості роботи, надійності та супроводження. Модульні тести при цьому є потужним інструментом, який може перевірити, чи функціональність системи з новою версією коду працює в точності так само, як й при старій [4].

Юніт тестування (Unit testing) – оцінка програмного забезпечення щодо виконання. Це тестування є тестуванням “найнижчого” рівня.

Юніт-тестування використовується для оцінки продукту на виконання поетапно. У декількох випадках, наприклад при розробці універсальних модулів бібліотек, для юніт-тестування не потрібне знання інкапсулюючого додатку програмного забезпечення. Більшість компаній, що займаються розробкою програмного забезпечення, роблять юніт-тести відповідальністю програміста. Один з кращих прикладів відмінностей між юніт-тестуванням та системним тестуванням можна проілюструвати в контексті ганебного дефекту Pentium. У 1994, Intel ввів свій мікропроцесор Pentium, і декількома місяцями пізніше, Thomas Nicely, математик в Коледжі Lynchburg у Вірджинії, виявив, що чіп надає некоректні відповіді з розподілених обчислень з плаваючою крапкою. Головна мета тестування - досягти результатів і ефективного юніт-тестування. Ефективність юніт-тестування в значній мірі визначається при при

тестуванні отриманих даних. Використання юніт-тестів в цій стадії є визначеним планом юніт-тестів, тестуванням пакетів, і тестуванням інструментів підтримки, які були зроблені на ранній стадії. Результати юніт-теста і проблеми потрібно зберегти і зробити звіт для наступної обробки. Проектувальники і розробники, чиї обов'язки стають легшими у цей момент, повинні бути доступними, щоб допомогти тестерам.

Один з найбільш ефективних підходів до компонентного тестування - це підготовка автоматизованих тестів до початку основної розробки ПЗ. Це називається розробка від тестування (test-driven development) або підхід тестування спочатку (test first approach). За допомогою цього підходу створюються методи коду, для яких запускаються тести, що були написані до початку кодування. Розробка ПЗ ведеться до тих пір, поки всі написані тести не будуть успішними.

Різниця між компонентним і модульним тестуванням. По суті ці рівні тестування представляють одне і теж саме, різниця тільки в тому, що в компонентному тестуванні в якості параметрів функцій використовують реальні драйвери та об’єкти, а в модульному тестуванні - конкретні значення.

Інтеграційне тестування (Integration testing) – оцінка програмного забезпечення щодо проектування підсистем. Тестування програмного забезпечення, при якому окремі програмні модулі об'єднуються і тестуються у групі. Інтеграційне тестування є видом тестування ПЗ, яке прагне перевірити інтерфейси між компонентами системи. Програмні компоненти можуть бути інтегровані як всі разом, так і в рамках ітеративного підходу [4].

За допомогою інтеграційного тестування виявляються дефекти та невідповідності у інтерфейсах та взаємодії інтегрованих компонентів (модулів). Інтеграційне тестування проводиться до тих пір, поки великі групи тестованих компонентів програмного забезпечення, які відповідають потрібній архітектурі, починають працювати як єдина система.

Для автоматизації процесу інтеграційного тестування застосовуються системи безперервної інтеграції (Continious Integration System, CIS). Принцип дії таких систем полягає в наступному:

– при зміні початкових кодів в репозиторії проводиться оновлення локального сховища;

– CIS проводить моніторинг системи контролю версій;

– виконуються необхідні перевірки і модульні тести;

– початкові коди компілюються в готові виконувані модулі;

– виконуються тести інтеграційного рівня;

– генерується звіт про тестування.

Таким чином, автоматичні інтеграційні тести виконуються відразу ж після внесення змін, що дозволяє виявляти і усувати помилки в короткі терміни.

Розшлянемо рівні інтеграційного тестування [4]:

– *системний інтеграційний рівень* (System Integration Testing) - перевіряється взаємодія між різними системами після проведення системного тестування.

– *компонентний інтеграційний рівень* (Component Integration testing) - перевіряється взаємодія між компонентами системи після проведення компонентного тестування;

Розглянемо основні підходи до інтеграційного тестування.

Знизу вгору (Bottom Up Integration). При даному підході усі низькорівневі модулі, функції або процедури збираються разом та потім тестуються. Потім збирається наступний рівень модулів для інтеграційного тестування. Цей підхід вважається корисним, якщо всі або майже всі модулі розроблюваного рівня готові. Даний підхід також допомагає визначити рівень готовності додатків за результатами тестування.

Зверху вниз (Top Down Integration). При даному підході в першу чергу тестуються компоненти верхнього рівня об'єктів за допомогою заглушок замість компонентів більш низького рівня.

Великий вибух («Big Bang» Integration). При даному підході всі або практично всі розроблені модулі збираються воєдино у вигляді готової системи або її основної частини й вже тільки потім проводиться інтеграційне тестування. Даний підхід дуже хороший для заощадження часу але, якщо тест кейси та їхні результати записані некоректно, то сам процес інтеграції дуже ускладниться, що стане перешкодою для команди тестувальників при досягненні головної мети інтеграційного тестуваннях [4].

Системне тестування (System testing) – оцінка програмного забезпечення щодо проектування архітектури, тестування програмного забезпечення, яке виконується на інтегрованій, повній системі, з метою перевірки відповідності системи початковим вимогам. Крім того, системне тестування програмного забезпечення повинно гарантувати, що програма функціонує так, як очікувалося, а також, що не можна знищити програму або пошкодити її робоче середовище, адже це викличе процеси в цьому середовищі, які переведуть систему в неробочий стан. Також, системне інтеграційне тестування перевіряє, чи інтегрується система в будь-яку зовнішню систему відповідно до системних вимог [4].

Альфа-тестування — це моделювання реальної роботи системи штатними розробниками або реальна робота з системою потенційними користувачами/замовником. Альфа-тестування зазвичай проводиться на ранніх стадіях розробки ПП, але в деяких випадках може застосовуватися для розробленого продукту як внутрішнє приймальне тестування. Іноді альфа-тестування виконується з використанням середовища або під відлагоджувачем, яке допомагає швидко виявити помилки. Знайдені помилки можуть бути передані тестувальникам для подальшого дослідження у середовищі, подібному тому, де буде використовуватися програма.

Бета-тестування — це інтенсивне використання майже готового ПЗ для виявлення максимальної кількості помилок та дефектів в його роботі задля подальшого усунення перед остаточним релізом. Іноді бета-тестування

виконується, щоб отримати зворотній зв'язок про ПП від його майбутніх користувачів [4].

Доволі часто для відкритого ПЗ стадія альфа-тестування характеризує функціональне наповнення коду, в той час як бета-тестування — стадію виправлення помилок. При цьому, на кожному етапі розробки ПЗ проміжні результати роботи доступні також кінцевим користувачам.

Оскільки системне тестування - це процес, що вимагає значних ресурсів для його проведення, окремо виділяють команду тестувальників, а часто системне тестування виконується організацією, яка взагалі не пов'язана з командою розробників та тестувальників, що виконували роботи на попередніх етапах тестування. Також необхідно відзначити, що при розробці деяких типів ПЗ (наприклад, бортового, авійційного), вимога незалежного тестування на всіх етапах розробки є обов'язковою. Даний вид тестування проводиться в кілька фаз, на кожній фазі перевіряється один з аспектів поведінки системи, тобто проводиться один з типів системного тестування [3]. Всі ці фази можуть відбуватися одночасно або послідовно. Далі розглянемо особливості кожного з типів системного тестування.

Виділяють такі види системного тестування:

– тестування продуктивності;
– функціональне тестування;
– навантажувальний або стресове тестування;
– тестування надійності і відновлення після збоїв;
– тестування безпеки;
– тестування зручності використання;
– тестування конфігурації.

У ході системного тестування проводяться не всі з перерахованих видів тестування - їх набір залежить від тестованої системи.

Приймальне тестування (Acceptance testing) – формалізоване тестування, спрямоване на перевірку застосунків з точки зору кінцевого

користувача / замовника і винесення рішення про те, чи приймає замовник роботу чи ні.

Для того, щоб проводить таке тестування швидко, часто і головне ефективно, існують засоби для автоматизації функціонального тестування.

Формальний процес тестування перевіряє відповідність системи вимогам та проводиться з метою:

– визначити чи задовольняє система приймальним критеріями;

– прийняття рішення замовником або іншою уповноваженою особою приймається застосування чи ні.

Приймальне тестування проводиться на основі набору типових тестових випадків і сценаріїв, розроблених відповідно до вимог даного додатку.

Рішення щодо проведення приймального тестування приймається, коли:

– замовник ознайомлений з планом приймальних робіт (product acceptance plan) або іншим документом, де описано набір дій, пов'язаних з проведенням приймального тестування, дата проведення, відповідальні та інше;

– продукт досяг необхідного рівня якості.

Фаза приймального тестування триває поки замовник не виносить рішення про відправку програми на доопрацювання або видачі додатка.

1.2 Опис бізнес-процесів

У магістерській роботі розробляється система, що дозволяє створити плани тестування та згенерувати автотести на основі цих планів, розробити власний фреймворк тестування. Розроблювана система повинна наступні функції:

– побудова плану тестування згідно обраного критерію;

– побудова фреймворку з автотестами на основі плану тестування;

– можливість інтеграції з ТС та запуск тестів.

1.2.1 Структура бізнес-процесів

Ефективно представити та проаналізувати бізнес-процеси компаній та сервісів допомогає методологія IDEF0, яка набуває широкого використання завдяки тому, що вона має просту та зрозумілу графічну нотацію. Головне місце відводиться діаграмам, на яких відображені зв'язки між функціями та зовнішнім середовищем. За допомогою геометричних прямокутників відображаються функції системи, а зв'язки між ними - за допомогою стрілок. Схема структурна контекстної моделі представлена у графічному матеріалі.

1.2.2 Актори і функції

В системі наявний один актор і це Тестувальник, який має можливість конструювати та редагувати графи станів та переходів, будувати плани тестування згідно обраної стратегії, запускати та моніторити результати запуску тестів в TC. На рисунку 1.1 наведено головні функції актора.

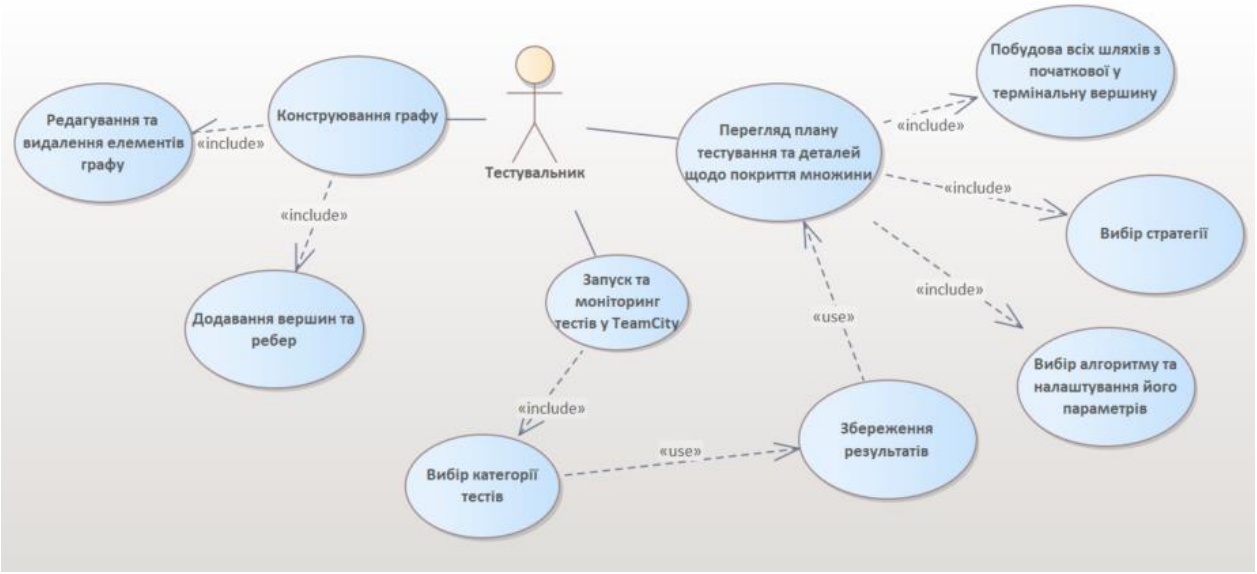


Рисунок 1.1 – Схема структурна варіантів використання

1.2.3 Опис процесу діяльності

Загальний процес діяльності починається з конструювання графу діаграми станів та переходів, після цього тестувальник повинен обрати стратегію, алгоритм та налаштувати параметри алгоритму для побудови плану тестування згідно обраної стратегії. Після чого, тестувальник має можливість перегляду та збереження результатів роботи. На основі плану тестування

формується фреймворк тестування, який містить автотести, та тестувальник має змогу переглянути їх, запустити, виконати інтеграцію з ТС. Схема структурна діаграми діяльності підсистеми побудови планів тестування наведена у графічному матеріалі. На рисунку 1.2 відображено процес діяльності для підсистеми побудови та запуску тестів.

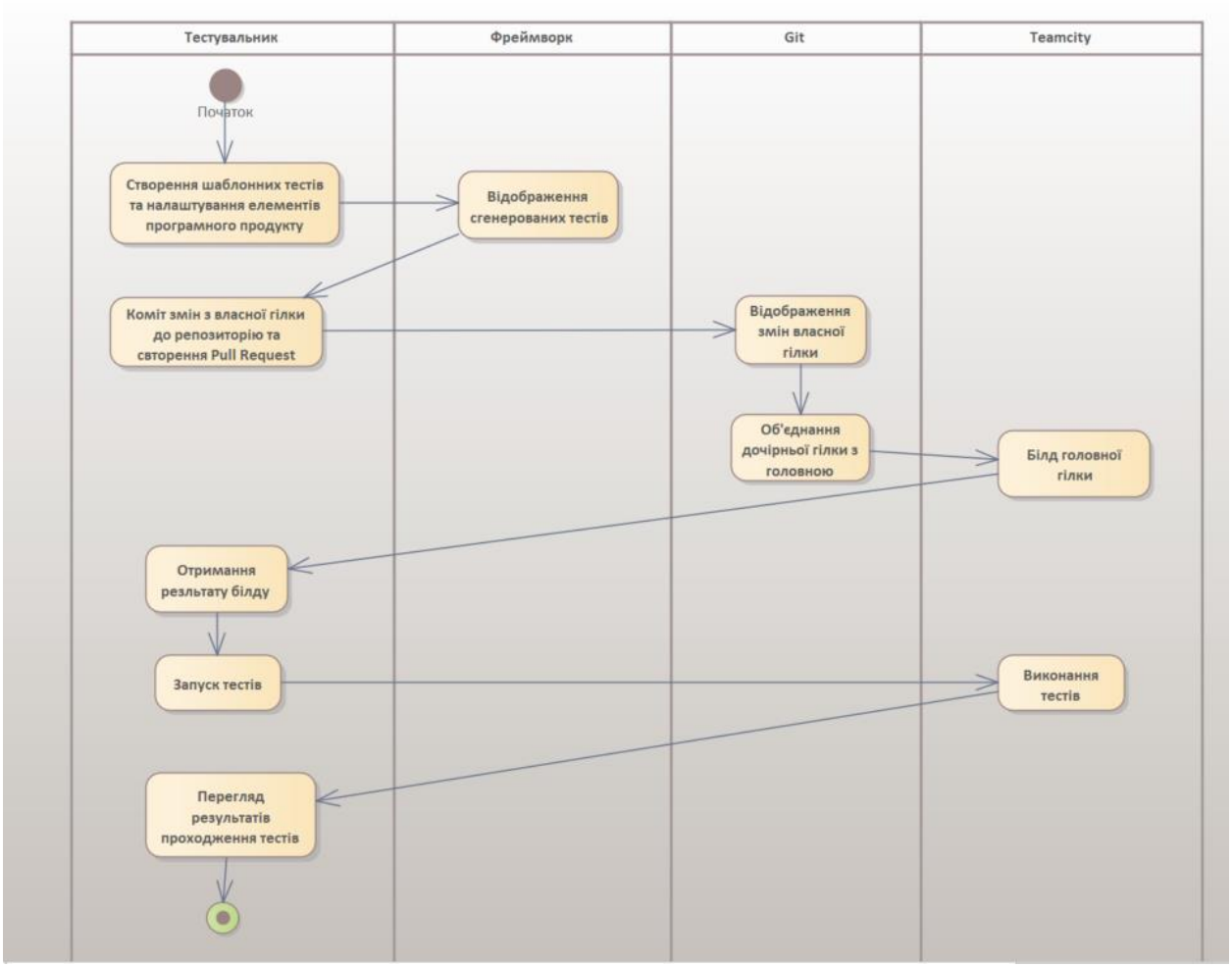


Рисунок 1.2 – Схема структурна діаграми діяльності побудови та запуску тестів

1.3 Опис постановки задачі

В системі з підтримки процесу верифікації програмного продукту реалізована функція приймального тестування, тобто тестування, яке дозволяє перевірити, чи задовольняє створене ПЗ початковому ТЗ, тому воно є ключовим етапом у життєвому циклі програмного продукту. Саме тому сучасні виробники програмного забезпечення в області автоматизації

тестування приділяють багато уваги покращенню засобів генерації планів тестування та засобів автоматизації тестування.

В даній роботі розглядається підхід до тестування програмної системи за відповідною їй діаграмою станів та переходів, яка створюється на етапі проектування. Така діаграма має один вхід (початкова точка програми) та один або декілька виходів, при чому можлива наявність циклів та петель. Поставимо у відповідність кожному стану діаграми вершину орієнтованого графу, а в якості ребер буде перехід від одного стану до іншого. Отримали орієнтований граф, де кожному автотесту у відповідність ставиться шлях, який починається у конкретній початковій вершині та завершується в одній з термінальних. На рисунку 1.3 вершина 1 виступає в якості початкової вершини, а вершина 8 – термінальної.

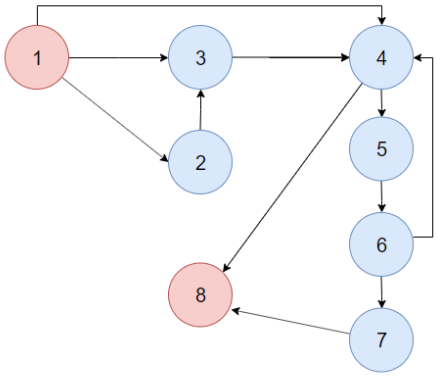


Рисунок 1.3 - Граф станів та переходів

Під планом тестування будемо розуміти сукупність автотестів, яка задовольняє певним умовам. Відповідно до цих умов можливі декілька підходів до побудови плану тестування. Поширеними є покриття вершин (Edge Coverage) та ребер (Branch Coverage), покриття шляхами (Path Coverage), яке має багато різних видів, одним з яких є Edge-Pair Coverage (покриття типу вхід-вихід) [6].

Таблиця 1.1 відображає покриття типу Edge-Pair Coverage для графа зображеного на рисунку 1.3, знаходження такої мінімальної множини шляхів, яка покриває усі комбінації пар дуг вхід-вихід для кожної вершини графу. Наприклад, для вершини 4 множина пар дуг вхід-вихід така: {[(1;4)(4;5)],

[(1;4)(4;8)], [(3;4)(4;5)], [(3;4)(4;8)], [(6;4)(4;5)],[(6;4)(4;8)]}. Надалі ці пари будемо називати тріадами [3]. Після цього кожній тріаді у відповідність ставиться підмножина шляхів, що її покривають. У таблиці 1.1 представлені: множина усіх тріад, множина усіх шляхів від початкової до термінальних, а також ознака входу/не входу тріади у шлях.

Таблиця 1.1 – Входження тріад у шляхи

Тріади / Шляхи	(1;2) (2;3)	(1;3) (3;4)	(1;4) (4;5)	(2;3) (3;4)	(1;4) (4;8)	(3;4) (4;5)	(3;4) (4;8)	(4;5) (5;6)	(5;6) (6;4)	(5;6) (6;7)	(6;4) (4;5)	(6;4) (4;8)	(6;7) (7;8)
[1,4,8]	0	0	0	0	1	0	0	0	0	0	0	0	0
[1,3,4,8]	0	1	0	0	0	0	1	0	0	0	0	0	0
[1,2,3,4,8]	1	0	0	1	0	0	1	0	0	0	0	0	0
[1,4,5,6,4,8]	0	0	1	0	0	0	0	1	1	0	0	1	0
[1,3,4,5,6,4,8]	0	1	0	0	0	1	0	1	1	0	0	1	0
[1,2,3,4,5,6,4,8]	1	0	0	1	0	1	0	1	1	0	0	1	0
[1,4,5,6,7,8]	0	0	1	0	0	0	0	1	0	1	0	0	1
[1,3,4,5,6,7,8]	0	1	0	0	0	1	0	1	0	1	0	0	1
[1,2,3,4,5,6,7,8]	1	0	0	1	0	0	0	1	0	1	0	0	1
[1,4,5,6,4,5,6,7,8]	0	0	1	0	0	0	0	1	0	1	1	0	1
[1,3,4,5,6,4,5,6,7,8]	0	1	0	0	0	1	0	1	0	1	1	0	1
[1,2,3,4,5,6,4,5,6,7,8]	1	0	0	1	0	1	0	1	0	1	1	0	1

Наступне важливе питання - визначення критеріїв якості множини тестів. Для тестера одним з важливих критеріїв є час виконання тесту, тому були виділені такі стратегії: «довга», під якою розуміється знаходження плану тестування, який включає в себе найбільш довгі тести та «коротка», яка вимагає знаходження плану тестування, який би складався з більш коротких тестів.

Побудувати допустимий план тестування – не дуже важка задача. Значно складніше – це побудувати план, який би у найбільшій мірі задовольняв би потреби менеджера команди тестувальників і при цьому будувався би за прийнятний час.

Метою дослідження є підвищення якості та надійності програмного продукту за рахунок автоматизації процесу тестуванння, що дозволяє скоротити час та витрати на виконання процесів тестування.

Для досягнення мети необхідно виконати наступні **завдання**:

– проаналізувати існуючі методи автоматизації тестування програмного забезпечення;

– провести аналіз відомих робіт з розв’язання поставленої в рамках роботи задачі;

– розробити методи побудови плану тестування в залежності від обраної стратегії;

– розробити метаевристичні алгоритми покриття множини для вирішення задачі побудови плану тестування;

– виконати програмну реалізацію розроблених алгоритмів;

– налагодити інтеграцію з TeamCity;

– проаналізувати дані обчислювальних експериментів для визначення параметрів алгоритмів;

– провести порівняльний аналіз розроблених алгоритмів обраної стратегії.

1.4 Рішення з інформаційного забезпечення

1.4.1 Вхідні дані

Вхідні дані:

– граф станів та переходів;

– тип стратегії (довга або коротка);

– параметри мурашинного алгоритму (кількість ітерацій, кількість мурашок, ступінь жадібності (α), вплив феромону(β), значення початкового феромону, коефіцієнт випаровування феромону);

– параметри генетичного алгоритму (розмір початкової популяції, кількість точок схрещування, максимальна кількість ітерацій).

1.4.2 Вихідні дані

Вихідними даними системи є:

– перелік усіх шляхів з початкової вершини у термінальні;

– сгенерований план тестування згідно обраної стратегії і алгоритму;

– результати покриття множини (кількість тестів, середня довжина шляхів, середнє значення модуля відхилення від середньої довжини шляхів покриття);

– сгенеровані автотести згідно плану тестування.

Висновки до розділу

У розділі розглянуто тестування програмних продуктів, як одна з основних частин розробки програми для забезпечення якості та підтримки робочого стану системи. Розглянуто основні види тестування та обгрунтовано актуальність розробки системи. Описано структуру бізнес процесів у вигляді контекстної діаграми та наведено схеми структурні варіантів використання та діаграми послідовності різних процесів. Визначені цілі та завдання магістерської дисертації. Розроблено рішення з інформаційного забезпечення системи.

2 МОДЕЛІ ТА МЕТОДИ ПОБУДОВИ ПЛАНІВ ТЕСТУВАННЯ

2.1 Змістовна постановка задачі

Дано орієнтований граф з однією початковою і декількома кінцевими вершинами. Отже є множина шляхів, які у сукупності покривають усі тріади. Після цього кожній тріаді у відповідність ставиться підмножина шляхів, що її покривають. Подальшим завданням є формування підмножини множини шляхів, яка є не надлишковою. При цьому обрана сукупність повинна відповідати заданій стратегії тестування.

Початковими даними для задачі є:

- множина шляхів;
- множина тріад;
- задана стратегія.

Таким чином необхідним є розв'язання задачі знаходження покриття.

2.2 Математична постановка задачі

Задача знаходження покриттів зводиться до класичної задачі мінімального покриття множини [7]. Початковими даними цієї задачі є скінчена множина T тріад і множина W шляхів (кожен з шляхів визначається множиною покритих тріад, тобто шлях можна розглядати як підмножину T). Покриттям $\overline{w} \subseteq w$ називають множину найменшої потужності (найменшої вартості) підмножин, об'єднанням яких є T [2].

Дана задача відноситься до класу NP-складних та може бути сформульована як задача булевого програмування. Нехай граф має m тріад (множина T) і в ньому можливі n шляхів від початкової до термінальних вершин (множина W). До кожного шляху відомі тріади, що входять до нього, тобто для $i = \overline{1, m}$ та $j = \overline{1, n}$ задані

$$a_{ij} = \begin{cases} 1, \text{ якщо тріада } i \text{ входить у шлях } j, \\ 0, \text{ якщо тріада } i \text{ не входить у шлях } j. \end{cases}$$

Необхідно знайти таку підмножину шляхів $\overline{w} \subseteq w$, щоб кожна тріада входила хоча б у один шлях і ця множина була б найкращою згідно обраної стратегії. За умови, що

$$x_j = \begin{cases} 1, \text{ якщо } j-й \text{ шлях входить до множ.} \overline{W}, \\ 0, \text{ якщо } j-й \text{ шлях не входить до множ.} \overline{W}, \end{cases}$$

вказане обмеження формулюється наступним чином:

$$\sum_j a_{ij} x_j \geq 1, \; i = \overline{1,m} \; .$$

Цільова функція

«Довга» стратегія призводить до мінімізації кількості шляхів, які будуть включені в план тестування, тобто

$$\sum_{j=1}^{n} x_j \to \min \; .$$

При цьому автоматично буде гарантуватися ненадлишковість знайденого покриття (покриття є ненадлишковим, коли виключення з нього хоча б одного шляху призведе до недопустимості розв'язку).

«Коротка» стратегія ж призводить до максимізації кількості шляхів. Тобто такий план буде включати в себе найбільш можливу кількість коротких шляхів (за відсутності надлишковості розв'язку). Цільова функція для цього випадку будеє:

$$\sum_{j=1}^{n} x_j \to \max \; .$$

2.3 Огляд методів розв'язання задачі

Комбінаторна постановка задачі про покриття множини полягає в наступному. Нехай дано множину $M = \{1, \dots, m\}$ і набір її підмножин $M_1, \dots, M_n$ таких, що $\bigcup_{j=1}^{n} M_j = M$. Сукупність підмножин, $M_j, j \in J \subseteq \{1, \dots, n\}$ називається *покриттям множини* M, якщо $\bigcup_{j=1}^{n} M_j = M$. Кожному M_j приписана вага $c_j > 0$. Потрібно знайти покриття мінімальної сумарної ваги. Задача називається *незваженою*, якщо всі підмножини M_j мають одиничну

вагу. Часто задача про покриття формулюється як задача цілочисельного програмування:

$$\sum_{j=1}^{n} c_j x_j \to \min \tag{2.1}$$

$$\sum_{j} a_{ij} x_j \geq 1,\ i = \overline{1,m}, \tag{2.2}$$

$$x \in \{0,1\}^n, \tag{2.3}$$

де $A = (a_{ij})$- матриця розміру $m \times n$ з елементами $a_{ij} = 1$, якщо $i \in M_j$ та $a_{ij} = 0$ у іншому випадку; $c = (c_1, \dots, c_n)$ - вектор ваг; $x = (x_1, \dots, x_n)$ - вектор змінних з компонентами $x_j = 1$, якщо M_j входить у покриття, та $x_j = 0$ в іншому випадку. Крім того, в літературі розглядаються узагальнені задачі про покриття, в яких в правій частині системи (2.2) замість одиничного вектору використовується вектор натуральних чисел [7, 8], і спеціальні класи задач [9, 10]. Значна увага приділяється задачам про покриття у графі, зокрема задачу про вершинне покриття, яка може бути сформульована наступним чином. Нехай $G = (V, E)$- граф з множиною вершин V і множиною ребер E. Для кожної вершини $v \in V$ визначена вага $w_v > 0$. Підмножина $C \subseteq V$ називається *вершинним покриттям*, якщо кожне ребро з E інцидентне хоча б одній вершині з C. Потрібно знайти вершинне покриття мінімальної ваги. Розглядаються також задачі про покриття графа його вершинами або ребрами (задача про домінуючу множину вершин і задача про реберне покриття).

Задача про покриття множини належить до *NP*-складних. Зокрема, *NP*-складною є навіть незважена задача про вершинне покриття у разі, коли G - планарний граф і ступені всіх його вершин рівні трьом. Цікавий результат Ю. І. Журавльова [11] про складність задачі про покриття множин. Він встановив, що в класі локальних алгоритмів кінцевого індексу задача про входження підмножини M_j в яке-небудь оптимальне покриття є нерозв'язною. До задачі про покриття зводяться багато відомих задач дискретної оптимізації: задача стандартизації, упаковки та розбиття множини [12], задача про найбільший клік [13], задача мінімізації полінома від булевих змінних та ін.

Відома також і зворотна зводимість задачі про покриття до цих задач [12, 13]. На практиці задачі про покриття виникають при розміщенні пунктів обслуговування [14], в системах інформаційного пошуку, при призначенні екіпажів на транспорті [15], проектуванні інтегральних схем і конвеєрних ліній тощо.

2.3.1 Властивості задачі про покриття

Розглянемо релаксаційні багатогранні множини

$$\Omega = \{x \in R^n | Ax \geq e, 0 \leq x \leq e\}$$

$$\Omega' = \{x \in R^n | Ax \geq e, e \geq 0\}$$

де e *−вектор з одиниць*

Легко бачити, що при будь-якому векторі $c \geq 0$ виконується рівність

$$\min_{x \in \Omega} cx = \max_{x \in \Omega'} cx$$

Багатогранник Ω є цілочисельним (тобто всі вершини мають цілочисельні координати) тоді і тільки тоді, коли цією властивістю володіє і багатогранник Ω'. Матриця A, для якої багатогранник Ω' є цілочисельним, називається ідеальною. Цілочисельні багатогранники для задачі про покриття становлять великий інтерес, оскільки задача в цьому випадку зводиться до вирішення ЛП-релаксації $min = \{cx | x \in \Omega'\}$ Отже, її можна вирішити за поліноміальний час. Вагу оптимального рішення ЛП-релаксації далі будемо позначати через $\bar{f}(c, A)$ [1].

2.3.2 Точні методи розв’язання задачі про покриття

Як було зазначено вище, точний розв’язок задачі про покриття може бути отримано за поліноміальний час, якщо матриця A ідеальна, тобто всі вершини багатогранника Ω' цілочисельні. Сюди відносяться задачі про вершинне покриття в дводольному графі і у дереві. Ефективні точні алгоритми відомі і для деяких задач про покриття, багатогранники яких, взагалі кажучи, не володіють властивістю цілочисельності. Зокрема, якщо кожен стовпець матриці A містить рівно дві одиниці, то завдання про покриття можна розглядати як завдання реберного покриття в графі, яка ефективно зводиться

до пошуку максимального паросполучення [14]. У [16] встановлено поліноміальна розв'язність так званих надщільних завдань про покриття.

Для вирішення загальної задачі про покриття запропонована велика кількість точних алгоритмів, заснованих на методах гілок та меж [8, 14, 17, 21], відсікання [18], перебору L -класів [19, 23] та ін. Верхні оцінки часової складності точних методів порівнянні зі складністю переборних алгоритмів [20]. У зв'язку з цим при вирішенні практичних завдань більшої актуальності набувають евристичні способи скорочення перебору. Значна увага приділяється розробці гібридних алгоритмів, в яких переборні схеми комбінуються з відсіканням і різними евристиками для отримання верхніх і нижніх оцінок оптимуму цільової функції на допоміжних підзадачах.

Роботи Е. Балаша і А. Ху [17, 35] - одні з перших, в якій схема гілок і меж для задачі про покриття була успішно суміщена з лагранжевою евристикою. Ця робота мала великий вплив на дослідження в даному напрямі. З використанням аналогічного підходу в [21] було запропоновано гібридний алгоритм гілок і меж, що включає в себе відсікання Гоморі, лагранжеву релаксацію та інші евристики. Також, гарні обчислювальні властивості в експерименті продемонстрував алгоритм з [22], також заснований на схемі гілок і меж. Відмінною особливістю цього алгоритму є динамічна субградієнтного процедура, використовувана для вирішення підзадач лагранжевої релаксації. За допомогою алгоритму з [22] успішно вирішувалися тестові завдання з випадковими даними до 4000 змінних, а також завдання призначення екіпажів з реальними даними при n, близьких до 10^4 [36].

У [23] загальний метод перебору L -класів був адаптований до вирішення завдання про покриття. Цей алгоритм заснований на лексикографічному впорядкування елементів L - розбиття релаксаційного багатогранника і послідовному їх виключенні за допомогою рішення лінійних підзадач. У гібридному алгоритмі з [19] перебір L - класів комбінується з генетичним алгоритмом і лагранжевої евристики. У розробленій тут схемі генетичний алгоритм використовується для знаходження початкового рішення, а з

допомогою лагранжевой евристики знаходяться нижні і верхні оцінки цільової функції на підзадачах, що виникають у процесі перебору L - класів. Експерименти показали перспективність гібридного підходу [36]. Час вирішення завдань гібридним алгоритмом у багатьох випадках є в кілька разів меншим в порівнянні з базовим алгоритмом [23].

2.3.3 Наближені методи розв'язання задачі про покриття

З огляду на те, що в загальному вигляді задача про покриття NP-складна, природно шукати наближені алгоритми поліноміальної складності, що знаходять розв'язки зі значеннями цільової функції, близькими до оптимуму. Прийнято говорити, що наближений алгоритм для задачі мінімізації має відносну похибку δ, якщо для довільної індивідуальної задачі отриманий ним розв'язок перевищує оптимум не більш ніж у δ разів [1].

Цікаві результати за складністю наближеного рішення NP-важких задач отримані з використанням імовірнісних методів і нової ймовірнісної характеризації класу NP. Встановлено [25], що існування поліноміального алгоритму для задачі про покриття, що має відносну похибку $\delta = a \ln m$, де $a > 0$ - деяка константа, виходить рівність P = NP. В [24] доведено, що якщо справедлива гіпотеза $NP \neq DTIME(n^{O(\log\log n)})$, то при будь-якому $\varepsilon > 0$ для задачі про покриття не існує поліноміального алгоритму з відносною похибкою $\varepsilon = (1 - \varepsilon)\ln m$. Відома зводимість задачі про покриття до задачі про домінуючу множину вершин у графі зберігає відношення сумарної ваги покриття до оптимуму. Таким чином, обидві задачі мають однаковий рівень складності з точки зору наближеного рішення із заданою точністю. Для задач про покриття розроблено значну кількість наближених алгоритмів з апріорними оцінками точності одержуваних розв'язків.

Одним з перших наближених алгоритмів, запропонованих для незваженної задачі про покриття, є жадібний алгоритм. У цьому алгоритмі на кожній ітерації в якості чергового елемента покриття вибирається підмножина, що покриває найбільше число елементів з M, не покритих на

попередніх ітераціях. У роботі [26] при $f^*(e,A)>1$ для жадібного алгоритму показано, що

$$\delta \le 1 + \frac{\ln m - \ln f^*(e,A)}{f^*(e,A)\ln\left(1+\frac{1}{f^*(e,A)-1}\right)} \le 1 + \ln m - \ln f^*(e,A).$$

Близький результат був незалежно отриманий в роботах Д. Джонсона і Л. Ловаса:

$$\delta \le 1 + \ln \max_{j=1,\dots,n} |M_j|.$$

У [24] було встановлено, що в незваженому випадку для жадібного алгоритму справедливо нерівність $\delta \le \ln m - \ln\ln m$, що близько до відомої нижньої оцінки з [24]. Позначимо через $f_{gr}(e,A)$ потужність покриття, знайденого жадібним алгоритмом. У [26] показано, що якщо в кожному рядку матриці A міститься не менше k одиниць, то $f_{gr}(e,A) \le 1 + \frac{n}{k}\left(1 + \ln\frac{mk}{n}\right)$. Очевидно, ця нерівність дає також верхню оцінку потужності оптимального покриття для зазначеного класу матриць. У ряді робіт отримані верхні оцінки потужності найменших покриттів для деяких класів незважених задач про покриття [10].

Для задачі про покриття з довільними вагами В. Хватал запропонував модифікацію жадібного алгоритму, де на кожній ітерації в покриття включається підмножина M_j з максимальним відношенням числа покритих елементів, не покритих на попередніх ітераціях, до ваги c_j (якщо $c_j = 0$, то відношення вважаємо рівним +∞).

У випадку, коли число одиниць в кожному рядку матриці A обмежено зверху константою Δ, відомий алгоритм [28] з тимчасовою складністю $O(mn)$ і відносною похибкою Δ.

2.3.4 Евристичні методи розв'язання задачі про покриття

Для пошуку наближених розв'язків задач великої розмірності широке поширення одержали методи лагранжевой релаксації [22], генетичні алгоритми [29], пошук із заборонами [30], алгоритми мурашиної колонії [31], нейронні мережі та інші евристичні алгоритми. Велика частина евристичних

методів заснована на частковому переборі покриттів з урахуванням накопичуваної при цьому статистичної інформації. Як правило, порівняння евристик здійснюється на основі обчислювального експерименту, так як апріорні оцінки точності на практиці часто виявляються далекі від експериментальних результатів, а теоретичний імовірнісний аналіз ускладнюється складністю обчислювальних схем. Методи лагранжевої релаксації вперше були запропоновані для наближеного рішення ЛП-релаксації задачі комівояжер і широко використовуються для отримання оцінок оптимального значення цільової функції і побудови наближеного роз'язку в різних задачах цілочисельного програмування. Відмітна особливість цих методів полягає в тому, що в багатьох випадках вони дозволяють отримувати рішення, досить близькі до оптимуму, при порівняно малих обчислювальних витратах [1].

2.3.5 Критерії оцінки покриття

Вище розглянуті методи показали, що результати, отримані для задачі про покриття множини, стосуються мінімізації критерію $\sum_{j=1}^{n} x_j \to min$ та немає результатів, які б стосувалися його максимізації. Для урахування специфіки задачі (побудова покриттів за заданими стратегіями) наведемо можливі критерії оцінки покриттів:

– середнє значення модуля відхилення від середньої довжини шляхів покриття (обираються шляхи приблизно однакової довжини, максимізація не має сенсу в даному випадку);

– середня довжина шляхів покриття (максимізація відповідає довгій стратегії, а мінімізація – короткій);

– потужність покриття (кількість шляхів, що входять в покриття).

Так як критерій потужності покриття не завжди може забезпечити виконання заданої стратегії, доцільним буде використання перших двох критеріїв, які забезпечують побудову покриття, що складається з шляхів прилизно однакової довжини.

2.4 Розробка методів розв’язання задачі

Як показано в п. 2.2, при зведені задачі до задачі покриття множини, кожній тріаді ставиться у відповідність рівняння. Кожне рівняння відповідає множині шляхів, яка покриває відповідну тріаду. Отже, вхідними даними для задачі покриття множини є система рівнянь.

Процес розв’язання задачі можна розбити на два кроки.

Крок 1. Оптимізація системи рівнянь.

Крок 2. Розв’язання задачі.

Залежно від розмірності задачі та наявних обчислювальних ресурсів для розв’язання задачі пропонується один з таких методів:

- отримання повної множини покрить методом бектрекінгу;
- розв’язання задачі із застосуванням алгоритму мурашиних колоній;
- розв’язання задачі із застосуванням жадібного алгоритму;
- розв’язання задачі із застосуванням генетичного алгоритму.

2.4.1 Оптимізація системи рівнянь

На цьому етапі для збільшення швидкості подальших обчислень виконується оптимізація системи рівнянь:

- видалення рівнянь, що містять тільки один шлях з подальшим включенням цього шляху до кожного знайденого розв’язку;
- видалення рівнянь, які містять усі шляхи;
- видалення рівнянь у випадку, якщо воно повністю включає в себе інше рівняння).

Цей процес дозволяє значно зменшити розмірність задачі.

2.4.2 Отримання повної множини покрить методом бектрекінгу

На першому етапі дослідження було застосовано метод, запропонований в [25], який заснований на використанні законів булевої алгебри. Але з ростом розмірності задач цей підхід себе не виправдав. Тому було розроблено алгоритм рекурсивного спуску (в основу якого покладена ідея алгоритму бектрекінга), яким знаходиться повна множина покриттів. Алгоритм полягає у

послідовному виклику для кожного (починаючи з першого) елементу рівнянь рекурсивної процедури, що додає до цього елементу елемент наступного рівняння і так, поки не буде досягнуто останнє рівняння, що свідчитиме про отримання деякого допустимого розв’язку.

Бектрекінг (англ. backtracking), пошук з поверненням — це загальний алгоритм для знаходження всіх розв'язків деякої обчислювальної задачі, який покроково будує кандидатів на розв'язок, і відкидає кожного неповного кандидата *K* («вертається») як тільки визначає, що *K* не може бути доповненим до вірного розв'язку. Бектрекінг може застосовуватися тільки до задач, що дозволяють поняття «неповних кандидатів на розв'язок» та швидку перевірку на можливість доповнення кандидата до вірного розв'язку. За можливості застосування, бектрекінг значно швидший ніж повний перебір всіх можливих кандидатів завдяки виключенню великої кількості кандидатів однією перевіркою. Алгоритм бектрекінгу перебирає множину неповних кандидатів що, в принципі, можуть бути доповнені кількома шляхами для отримання всіх можливих розв'язків даної задачі. Доповнення будується покроково, послідовністю кроків розширення кандидата [12].

Концептуально, неповні кандидати є вузлами потенційного дерева пошуку. Кожний неповний кандидат є батьком кандидатів, відмінних від нього на один крок розширення; листями дерева є кандидати, які не можуть бути розширені далі.

Алгоритм бектрекінгу обходить це дерево рекурсивно, від кореня донизу, пошуком в глибину. В кожному вузлі *K* алгоритм перевіряє, чи може *K* бути доповненим до вірного розв'язку. Якщо ні, ціле піддерево з корнем в *K* пропускається (обрізається). Інакше, алгоритм перевіряє чи є *K* вірним розв'язком та повідомляє про це користувача і рекурсивно обходить усі піддерева *K*. Обидві перевірки і процедура формування дочірніх вузлів визначаються розробником алгоритму.

Внаслідок цього, реальне дерево пошуку, яке обходить алгоритм, становить лише частину потенційного дерева. Загальна трудомісткість

алгоритму – це кількість вузлів реального дерева помножена на вартість отримання і обробки кожного вузла. Цей факт має бути врахованим коли обирається потенційне дерево і реалізується перевірка на відсікання.

2.4.3 Розв’язання задачі із застосуванням жадібного алгоритму

Доволі часто для реальних практичних NP-складних задач за прийнятний час знайти точний розв’язок не можливо. В такому випадку можна взяти кращий розв’язок, який вдалося знайти за певний встановлений ліміт часу роботи точного алгоритму, або використовувати наближені або евристичні алгоритми. Жадібний алгоритм є одним з найпоширеніших зразків евристичних алгоритмів. Перевага цього алгоритму полягає в тому, що час знаходження рішення поліноміально зростає з збільшенням розмірності. Для задачі знаходження покриття графу шляхами розроблено та реалізовано такий жадібний алгоритм. Маємо початкове покриття, яке не включає в себе жодного шляху. Поки це покриття не задовольняє умові усі комбінації пар вхід вихід додаємо новий шлях, який покриває найбільш ймовірну кількість ще не покритих тріад. Цілком можливо, що отриманий розв’язок може бути надлишковим. Вихідна система рівнянь будується на основі шляхів, що увійшли у знайдений розв’язок. Після оптимізації можливо отримати декілька розв’язків, що дуже схожі між собою [2].

З урахуванням сутності дій алгоритму побудовані розв’язки будуть в більшості випадків задовольняти вимогам довгої стратегії.

2.4.4 Розв’язання задачі із застосуванням алгоритму мурашинних колоній

Ідея мурашиного алгоритму - моделювання поведінки мурашок, здатних швидко знаходити найкоротший шлях від мурашника до джерела їжі і адаптуватися до тих умов, що змінюються, знаходячи новий найкоротший шлях. Коли мурашка рухається, вона мітить свій шлях феромоном, а інші мурашки використовують цю інформацію для вибору шляху. Це елементарне правило поведінки визначає здатність мурашок знаходити новий шлях, якщо старий стає вже недоступний.

Розглянемо ситуацію, коли на оптимальному до тих пір шляху виникає перешкода. В цьому випадку необхідно визначити новий оптимальний шлях. Коли вони дійдуть до перешкоди, мурашки з рівною імовірністю обходитимуть її зліва та справа, те саме відбуватиметься і на зворотному боці перешкоди. Однак, ті мурашки, які випадково вибрали найкоротший шлях, будуть проходити його швидше та за декілька пересувань він буде більш збагачений феромоном. Так як рух мурашок визначається концентрацією феромона, то наступні віддаватимуть перевагу саме цьому шляху і будуть продовжувати збагачувати його феромоном доти, поки цей шлях з якихось причини не стане недоступний.

Отже, моделювання поведінки мурашок пов'язане з розподілом феромона на стежці - ребрі графа. При цьому вірогідність включення ребра в множину шляхів окремої мурашки пропорційна кількості феромона на цьому ребрі, та кількість феромона, яка відкладається, пропорційна довжині шляху покриття. Чим коротше шлях, тим більше буде відкладено феромона на його ребрах, тому, більша кількість мурашок включатиме його в синтез власних шляхів. Очевидний позитивний зворотний зв'язок швидко приведе до того, що найкоротший шлях стане єдиним маршрутом для руху більшості мурашок. Моделювання випаровування феромона негативного зворотного зв'язку гарантує, що знайдене локальне оптимальне рішення не буде єдиним - мурашки шукатимуть ще й інші шляхи. Швидке випаровування феромону призводить до забування хороших рішень та втрати пам'яті колонії мурашок, з іншого боку, великий час випаровування може призвести до отримання стійкого локального оптимального рішення. При моделюванні такої поведінки на деякому графі, ребра якого є можливими шляхами пересування мурашок, в перебігу певного часу, найбільш збагачений феромоном шлях по ребрах графа і буде розв’язком задачі, отриманим за допомогою мурашиного алгоритму.

Узагальнений алгоритм

Будь-який мурашиний алгоритм, незалежно від модифікацій, може бути представлений в такому вигляді:

ПОКИ (умови виходу не виконані)

КРОК 1. Створення мурашок.

КРОК 2. Пошук розв’язку.

КРОК 3. Оновлення феромона.

КРОК 4. Додаткові дії.

Розглянемо кожен крок детальніше.

Створення мурашок

Стартовий пункт, куди поміщається мурашка, залежить від обмежень, накладених умовами задачі. Тому що для кожної задачі спосіб розміщення мурашок є визначаючим. Або всі вони поміщаються в один пункт, або в різні з повтореннями, або без повторень.

На цьому ж етапі задається початковий рівень феромона. Він ініціалізується невеликим додатним числом для того, щоб на початковому кроці вірогідності переходу в наступну вершину не були нульовими.

Пошук розв’язку

Ймовірність переходу з вершини i у вершину j у момент часу t визначається за такою формулою:

$$p_{ij} = \frac{\tau_{ij}^{\alpha}(t) \cdot \eta_{ij}^{\beta}}{\sum_{j \in J_i} \tau_{ij}^{\alpha}(t) \cdot \eta_{ij}^{\beta}}, \qquad (2.4)$$

де J_i - множина вершин, в які дозволено перехід з вершини i, $\tau_{ij}(t)$ – рівень феромона, η_{ij} – видимість вершини j з вершини i, α та β - константи.

При $\alpha = 0$ вибір найближчого місця більш ймовірний, тобто алгоритм стає жадібним, коли $\beta = 0$ вибір відбувається тільки на базі феромона, що може приводити до локальних оптимумів. Тому стає необхідним компроміс між цими двома величинами, що знаходиться експериментально.

Оновлення феромона

Рівень феромона оновлюється за формулою:

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \sum_{k \grave{o} M_{ij}} \frac{L_{min}}{L_k}, \qquad (2.5)$$

де M_{ij} - множина мурашок, що пройшли по ребру *(i,j)*, L_k - ціна поточного розв'язку (довжина маршруту) для k-ої мурашки, L_{min} – параметр, що має значення порядку ціни оптимального розв'язку (передбачувана ідеальна ціна розв'язку у випадку мінімізації цільової функції), $\frac{L_{min}}{L_k}$ – кількість феромона, що залишає k-та мурашка на ребрі (*i*,*j*). Таким чином, чим гірший розв'язок (чим більше відповідне L_k), в який увійшло ребро (*i*,*j*), тим менше феромона буде залишено мурашкою на цьому ребрі.

Щоб побудувати відповідний мурашиний алгоритм для вирішення задачі побудови плану тестування необхідно виконати наступні дії.

КРОК 1. Представити задачу у вигляді набору компонент (вершин) і переходів (ребер) або навпаки неорієнтованих зважених графів, на яких мурашки зможуть будувати розв'язок.

КРОК 2. Визначити евристику поведінки мурашки при побудові розв'язок (у основі лежить формула визначення вірогідності переходів 2.4).

КРОК 3. Визначити значення сліду феромона (аналог формули 2.5).

КРОК 4. Якщо можливо, то визначити процедуру ефективного локального пошуку.

КРОК 5. Підібрати параметри мурашинного алгоритму (α, β, ρ, L_{min}).

Також визначальними є:

- кількість мурашок;
- момент, коли оновлюється феромон;
- поєднання з жадібними евристиками або локальним пошуком.

Розробка мурашиного алгоритму для задачі знаходження мінімального покриття

Задача формулюється як задача пошуку мінімального покриття типу Path Coverage, тобто знаходження такої (мінімальної за потужністю) множини шляхів, яка покриває усі комбінації пар дуг вхід–вихід для кожної вершини

орієнтованого графу і відповідає заданій стратегії. Змістовно вершини графа є шляхами, які повинна пройти (зібрати) мурашка.

Згідно математичної постановки задачі (п. 2.1), граф має m тріад (множина T) і в ньому можливі n шляхів від початкової до термінальних вершин (множина W). Для кожного шляху відомі тріади, що входять до нього, тобто для $i = \overline{1,m}$ та $j = \overline{1,n}$ задані

$$a_{ij} = \begin{cases} 1, \text{ якщо тріада } i \text{ входить у шлях } j, \\ 0, \text{ якщо тріада } i \text{ не входить у шлях } j. \end{cases}$$

Необхідно знайти таку підмножину шляхів $\overline{W} \subseteq W$, де кожна тріада входила хоча б у один шлях і ця множина була б найкращою згідно обраної стратегії. За умови, що

$$x_j = \begin{cases} 1, \text{ якщо } j-\text{й шлях входить до множ. } \overline{W}, \\ 0, \text{ якщо } j-\text{й шлях не входить до множ. } \overline{W}, \end{cases}$$

вказане обмеження формулюється таким чином:

$$\sum_j a_{ij} x_j \geq 1, \; i = \overline{1,m}.$$

Для побудови покриття множини визначимо основні елементи мурашиного алгоритму.

Розміщення мурашок по шляхах

На початку першої ітерації створюється стільки мурашок, скільки шляхів має множина W (k-та мурашка – на k-ий шлях). Це робиться для того, щоб на початку роботи алгоритму кожен з шляхів попав хоча б в одне з покрить (що дозволить максимально розширити область пошуку). На наступних ітераціях створюється певна кількість мурашок (ця величина є параметром алгоритму), які випадково обирають один з шляхів множини W. Це зменшення кількості мурашок (у порівнянні з першою ітерацією) обґрунтовується тим, що при великому значенні кількості мурах значно зростає час роботи алгоритму.

Локальні правила поведінки мурашок

Тепер з урахуванням особливостей задачі про мінімальне покриття множини, опишемо локальні правила поведінки мурашок при виборі шляху.

На відміну від класичного змісту мурашиного алгоритму будемо вважати, що мурашки, при своєму руху «збирають» шляхи з множини W у свій кошик (до тих пір, поки збирана множина шляхів не стане повним покриттям).

Мурашки мають власну "пам'ять", оскільки кожен шлях можна пройти (взяти в кошик) тільки один раз, то в кожної мурашки наявний список вже пройдених шляхів - список заборон (W_k^+) – множина шляхів, яку вже набрала мурашка). Позначимо через W_k^- - список шляхів, які ще не відвідала мурашка k (в цю множину входять тільки ті шляхи, які містять не покриті тріади). Мурашки володіють "зором". Видимість - евристичне бажання пройти шлях j.

Мурашки володіють "нюхом", вони можуть уловлювати слід феромона, підтверджуючий бажання пройти шлях j із множини W_k^- на підставі досвіду інших мурашок. Кількість феромона, що відповідає шляху j у момент часу t позначимо через τ_j.

Правило, що визначає вірогідність вибору наступного шляху (переходу)

Правило визначення величини, від якої прямо пропорційно буде залежати вірогідність вибору наступного шляху k-ої мурашки залежить від обраної стратегії.

Нехай k-а мурашка набрала шляхи:

$$W_k^+ = \{i_1, i_2, ...\},$$

(на початку першої ітерації $i_1 = k$).

Ця множина однозначно визначає:

а) множину T^+ - множину покритих тріад

б) T^- - множина непокритих тріад

в) W_k^- - множина всіх шляхів, що відповідають непокритим тріадам.

Визначимо величину видимості шляху $j \in W_k^-$.

Коротка стратегія

Логічно, що видимість шляху $j \in W_k^-$ для короткої стратегії обернено пропорційна довжині l_j цього шляху:

$$\eta_j = \frac{1}{l_j}.$$

Довга стратегія

Очевидно, що видимість шляху $j \in W_k^-$ для довгої стратегії прямо пропорційна довжині l_j цього шляху ($\eta_j \sim l_j$):

$$\eta_j = \frac{l_j}{l_{max}},$$

де l_{max} - довжина найдовшого шляху.

2.4.5 Розв'язання задачі із застосуванням генетичного алгоритму

Схема роботи генетичного алгоритму наведена на рисунку 2.1.

КРОК 1. Генеруємо початкову популяцію з *n* хромосом.

КРОК 2. Обчислюємо для кожної хромосоми її придатність.

КРОК 3. Обираємо пару хромосом-батьків за допомогою одного із способів відбору.

КРОК 4. Проводимо кросинговер двох батьків з імовірністю p_c, виробляючи двох нащадків.

КРОК 5. Проводимо мутацію нащадків з імовірністю p_m.

КРОК 6. Повторюємо кроки 3-5 доки не буде згенеровано нове покоління популяції, що містить *n* хромосом.

КРОК 7. Повторюємо кроки 2-6 доки не буде досягнутий критерій закінчення процесу.

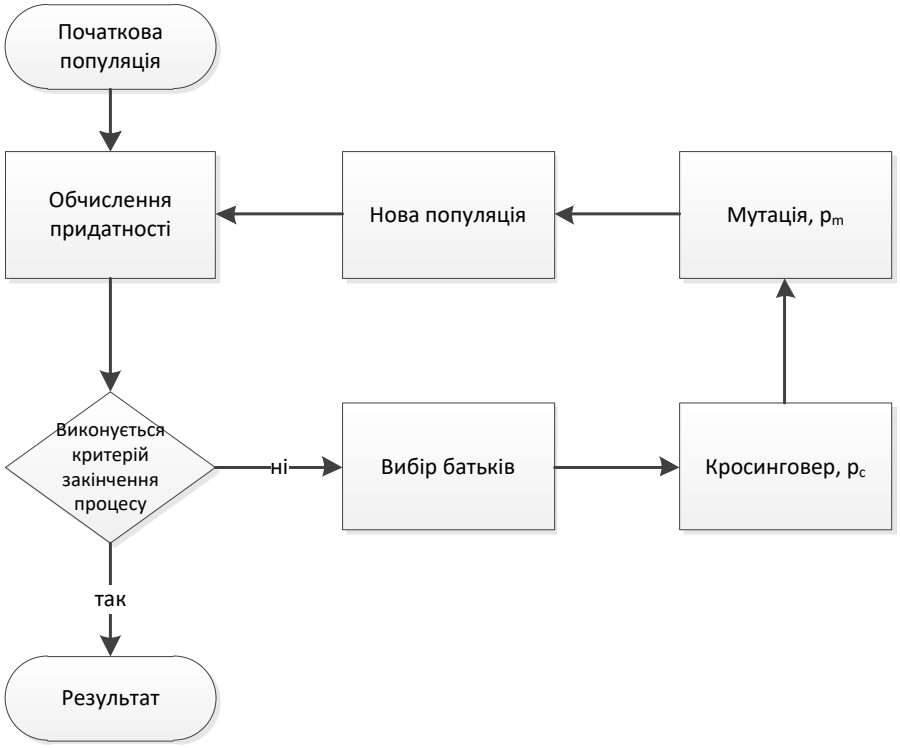


Рисунок 2.1 – Блок-схема роботи генетичного алгоритму

Оператори вибору батьків

Для генетичного алгоритму виділяють різні варіації селекції. Найбільш відомі з них - це *рулеточний* (пропорційний) та *турнірний* відбори.

При *турнірному відборі* (tournament selection) вибираються випадковим чином t особин з популяції, що складається із N особин, та найкраща особина записується в проміжний масив. Дана операція повторюється $M<N$ разів. Особини з результуючого проміжного масиву потім використовуються для схрещування, яке також відбувається випадковим чином. Часто розмір групи рядків, що відбираються для турніру, дорівнює 2. В такому випадку говорять про двійковий або парний турнір. Чисельність турніру визначається як t.

У *методі рулетки* (roulette-wheel selection) особини відбираються за допомогою N «запусків» рулетки, де N - розмір популяції. Колесо рулетки для кожного члена популяції містить по одному сектору. Розмір i - го сектору пропорційний ймовірності попадання в нову популяцію $P(i)$, яка обчислюється за формулою:

$$P(i) = \frac{f(i)}{\sum_{i=1}^{N} f(i)},$$

де $f(i)$ – природність i-ї особини. Число копій i-ї хромосоми, яке очікуємо після оператора рулетки визначають по формулі:

$$N_i = P(i)N.$$

При даному відборі члени популяції з більш високою пристосованістю з більшою ймовірністю будуть вибиратись, ніж особини з низькою пристосованістю.

Для вирішення задачі знаходження мінімального покриття за допомогою ГА обраний турнірний відбір, так як перевагою даного відбору є те, що він не вимагає додаткових обчислень.

Рекомбінація

Оператор рекомбінації використовують відразу після оператора відбору батьків для отримання нових особин-нащадків. Суть рекомбінації полягає в тому, що створені нащадки повинні успадковувати генну інформацію від обох батьків.

Для *багатоточкового кросинговеру* (Multi-point crossover), обираємо m точок розриву $k_i \in \{1,2,\dots,Nvar\},\ i = \overline{1,m},\ Nvar$ - кількість змінних (генів) у особині. Точки розриву обираються випадковим чином без повторень і сортуються в порядку зростання. Під час кросинговеру відбувається обмін ділянками хромосом, які обмежені точками розрізу, таким чином отримують двох нащадків. Ділянка особини з першим геном до першої точки розрізу не бере участь в обміні. Проаналізуємо дві особини по 11 двійковим генам.

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

Оберемо точки розриву кросинговеру:

Точка розриву (m=3)	2	6	10

Отримаємо таких нащадків:

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

Використання багатоточкового кросинговеру вимагає введення декількох змінних (точок розрізу) та для відтворення обираються особини з найбільшою пристосованістю.

Одноточковий і багатоточковий кросинговери визначають точки розрізу, які поділяють особини на частини. *Однорідний кросинговер* (Uniform crossover) створює маску особини, де в кожному локусі особини перебуває потенційна точка кросинговеру. Довжина маски кросинговеру така сама як перехресних особини. Створимо маску можна наступним чином: введемо деяку величину $0 < p_0 < 1$, і якщо випадкове число більше p_o, то на n-у позицію маски ставимо 0, інакше - 1. Отже, гени маски є випадковими двійковими числами (0 або 1). Виходячи з цих значень, геном нащадка стає перша (ген маски = 0) або друга (ген маски = 1) особина-батько.

Розглянемо особини:

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

Для кожного створеного потомка попередньо створюється маска з 11 випадково вибраних елементів із множини {0,1}:

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

Створимо нащадків наступним чином: якщо на i-му місці відповідної маски стоїть 1, то ген першого батька переходить до потомка, в іншому випадку успадковується ген другого батька. Отримуємо наступні особини:

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

Однорідний кросинговер дуже схожий на багатоточковий, але рядок випадкових бітових значень в ньому довший, тому для вирішення задачі знаходження мінімального покриття за допомогою ГА обраний багатоточковий кросинговер.

Мутація

Після процесу відтворення відбуваються мутації *(mutation).* Даний оператор необхідний для вибивання функції з локального екстремуму та запобіганню передчасної збіжності. Це досягається зміною випадково обраного гену в хромосомі:

$$x_1 x_2 \dots x_{n-1} x_n x_{n+1} \dots x_m \rightarrow x_1 x_2 \dots x_{n-1} \overline{x}_n x_{n+1} \dots x_m .$$

Так само як і кросинговер мутація може проводитися не тільки по одній випадково вибраній точці. Можна вибрати для зміни декілька точок в хромосомі, при цьому їх число також може бути випадковим. Використовують і мутації зі зміною відразу деякої групи точок, що йдуть поспіль.

Ймовірність мутації p_m(як правило, $p_m \ll 1$) може бути будь яке число на відрізку [0,1], або функцією від будь-якої характеристики задачі що розв'язується. Наприклад, можна вважати ймовірність мутації генів обернено пропорційною числу всіх генів в особині (розмірності).

Розробка генетичного алгоритму для задачі знаходження мінімального покриття

З урахуванням специфіки задачі розроблені основні оператори алгоритму.

Формування початкової популяції

Схема алгоритму генерації популяції представлена у графічному матеріалі. Параметрами алгоритму формування початкової популяції є:

- максимальний розмір популяції;
- максимальна кількість ітерацій.

Оператор вибору батьків

Реалізовано оператор вибору батьків, в основу якого покладений «турнірний» спосіб відбору. В результаті якого в якості першого батька обирається найкраща на поточний момент особина, а в якості другого – випадкова.

Оператор схрещування

Реалізовано багаточковий кросинговер. При цьому при схрещуванні забезпечується допустимість та не надлишковість розв’язку. Перше досягається за рахунок розташування точок схрещування таким чином, що відповідні ділянки генетичного коду кожного з батьків, відповідають одній і тій самій множині тріад.

Оператор мутації

Випадковим чином виключається певний шлях, а замість нього випадковим чином додаються шляхи (доти, поки не буде отриманий допустимий розв’язок).

Оператор додавання особини в популяцію

Оператор реалізований за рахунок видалення найгіршої особини наприкінці кожної ітерації.

2.5 Результати досліджень ефективності методу

Так як розроблені метаевристичні алгоритми мають ймовірнісну природу, то дуже важливим є питання визначення значень параметрів, які б давали гарні результати. Для отримання статистично стійких результатів кожен з алгоритмів був виконаний 20 разів з одним і тим самим набором параметрів на різних вхідних даних (графах) на різних комбінаціях параметрів.

Експерименти призначені для підбору значень параметрів алгоритмів, за яких алгоритм дозволяє стійко отримувати «гарні» результати та для порівняння ефективності розроблених алгоритмів.

Розмірність графа становить 106 шляхів / 4 цикли. Критерій – мінімізація середнього значення модуля відхилення від середньої довжини шляхів покриття.

2.5.1 Визначення параметрів ГА

Основними параметрами ГА є розмір початкової популяції та кількість точок схрещування. Проаналізуємо вплив цих параметрів на точність алгоритму. На рисунку 2.4 зображено графік впливу розміру початкової популяції на точність отриманого результату, а саме значення критерію - середнє значення модуля відхилення від середньої довжини тестів покриття.

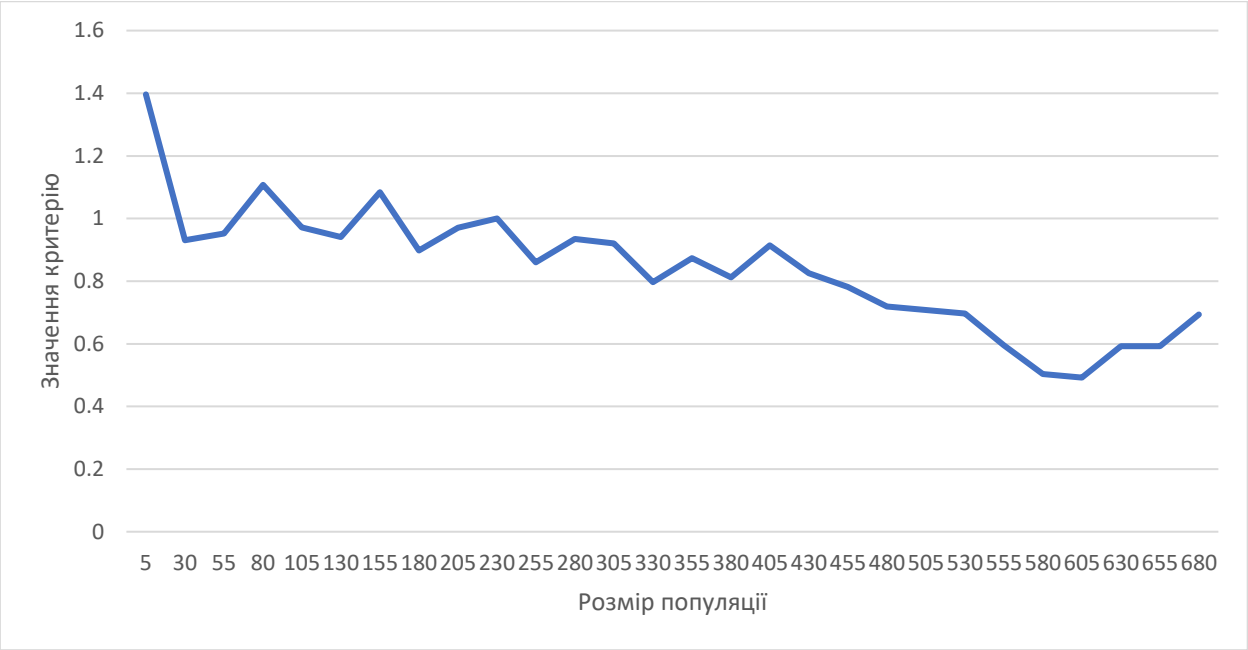


Рисунок 2.4 – Графік впливу розміру популяції на точність результату

На рисунку 2.5 зображено графік впливу кількості точок схрещування на точність алгоритму.

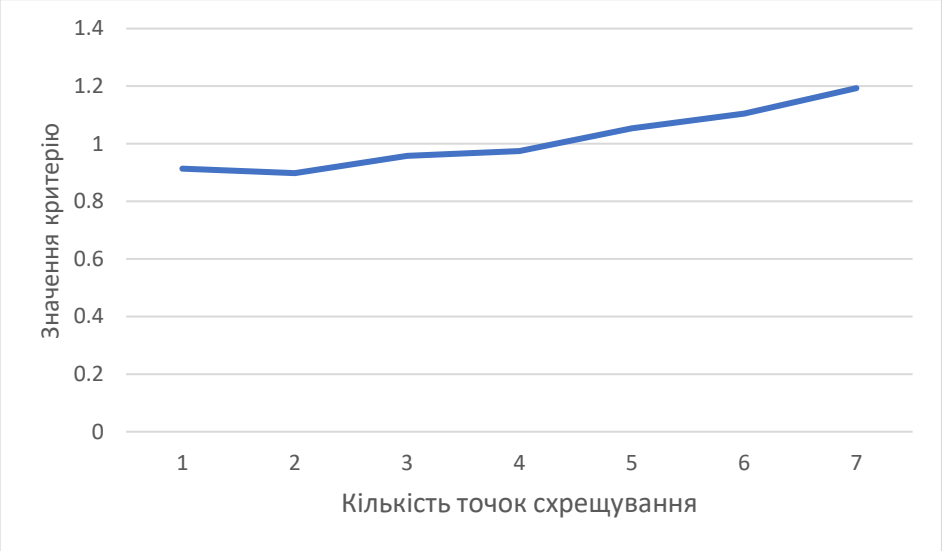


Рисунок 2.5 – Графік впливу кількості точок схрещування на точність алгоритму

Як видно з графіків, ГА чутливий до розміру популяції, проведені експерименти показали, що з ростом популяції значення «гарних» результатів було досягнуто при розмірі популяції ~ 550-700 особин. Також, що з збільшенням кількості точок схрещування значення середнього відхилення збільшується.

2.5.2 Визначення параметрів ACO

Есперименти проводилися для визначення коефіцієнту випаровування феромону та значення початкового феромону. На рисунку 2.6 зображено графік впливу початкового значення феромону на значення критерію мінімізації середнього значення модуля відхилення.

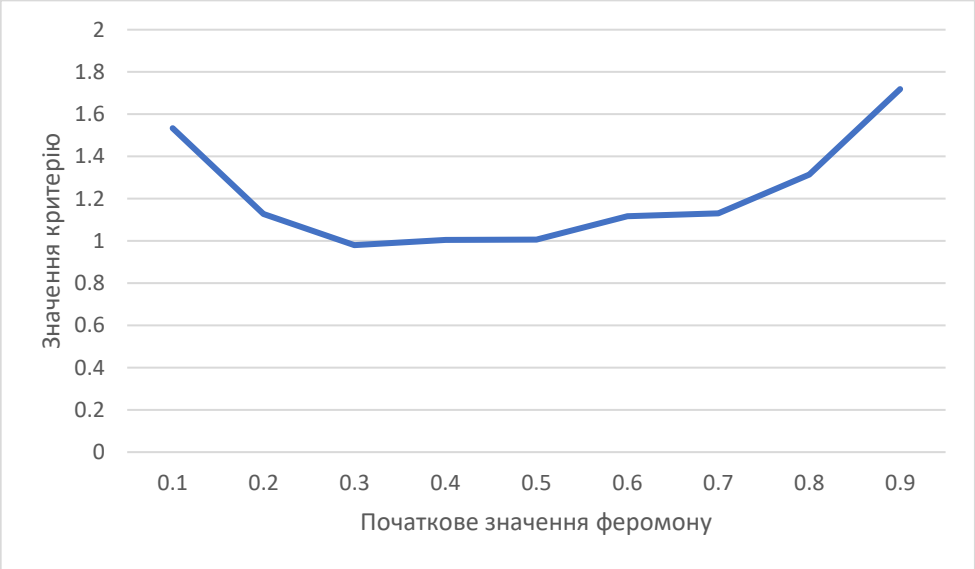


Рисунок 2.6 – Графік впливу початкового значення феромону на точність результату

На рисунку 2.7 представлено графік впливу значення коефіцієнту випаровування феромону на точність отриманого результату.

Рисунок 2.7 – Графік впливу значення коефіцієнту випаровування феромону на точність результату

В результаті експериментів визначили, що найбільш вдалим значенням параметру випаровування феромону є діапазон [0.15-0.35] та початкове значення феромону залишимо на відмітці 0.3.

2.5.3 Порівняльний аналіз алгоритмів

Для порівняльного аналізу алгоритмів та визначення ефективності в експериментах використовувались тестові задачі різної розмірності для різних типів стратегій. Результати експериментів для короткої стратегії подано у таблиці 2.1 та для довгої стратегії у таблиці 2.2.

Таблиця 2.1 – Резльтати експериментів для короткої стратегії

Розмірність графу	Алгоритм	Параметри	Час(мс)	Значення критерію
76/4	Жадібний		16	5,8
76/4	ГА	150 популяція, 350 ітерацій	2073	3,25
76/4	ACO	$\alpha = \beta = 1, \rho = 0.35$ 180 ітерацій, 75 мурашок	2893	3,5
827/9	Жадібний		1354	4,95
827/9	ГА	850 популяція, 150 ітерацій	29658	2,135
827/9	ACO	$\alpha = \beta = 1, \rho = 0.28$ 50 ітерацій, 95 мурашок	27532	2,04
2984/22	Жадібний		13997	3,467
2984/22	ГА	600 популяція, 200 ітерацій	89786	1,843
2984/22	ACO	$\alpha = \beta = 1, \rho = 0.3$ 50 ітерацій, 50 мурашок	145133	1,98

Таблиця 2.2 – Результати експериментів для довгої стратегії

Розмірність графу	Алгоритм	Параметри	Час(мс)	Значення критерію
76/4	Жадібний		21	2,8
76/4	ГА	150 популяція, 350 ітерацій	2174	3,17
76/4	ACO	$\alpha = \beta = 1, \rho = 0.35$ 180 ітерацій, 75 мурашок	2993	3,67
827/9	Жадібний		1251	3,05
827/9	ГА	850 популяція, 150 ітерацій	29898	2,315
827/9	ACO	$\alpha = \beta = 1, \rho = 0.28$ 50 ітерацій, 95 мурашок	26502	2,105
2984/22	Жадібний		12907	2,867
2984/22	ГА	600 популяція, 200 ітерацій	87783	1,703
2984/22	ACO	α=β=1, ρ=0.3 50 ітерацій, 50 мурашок	125143	1,83

Як видно з результатів кожен алгоритм має свої переваги та недоліки в залежності від умов використання. На рисунку 2.8 відображено графік для короткої стратегії, рисунок 2.9 відображає результати роботи при довгій стратегії.

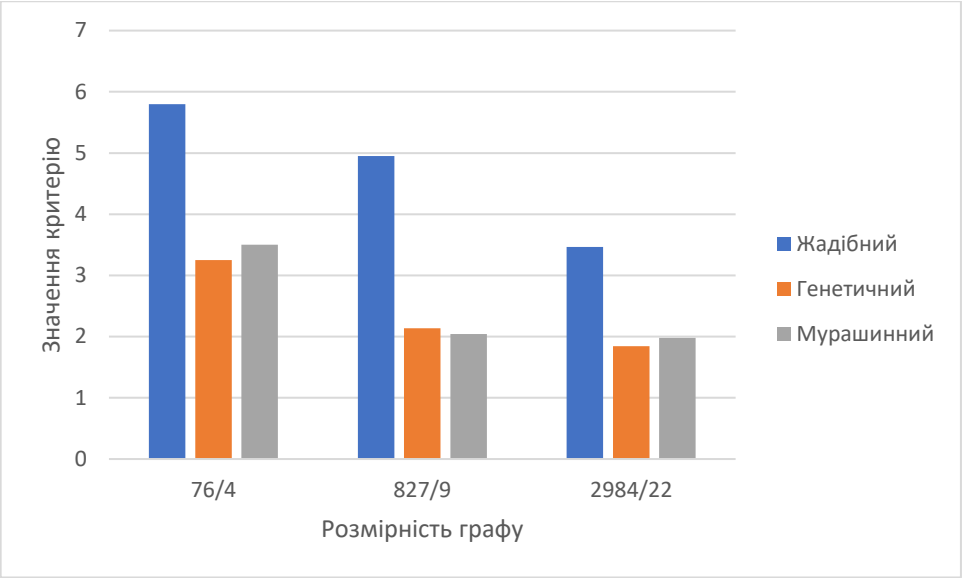


Рисунок 2.8 – Порівняльний аналіз алгоритмів для короткої стратегії

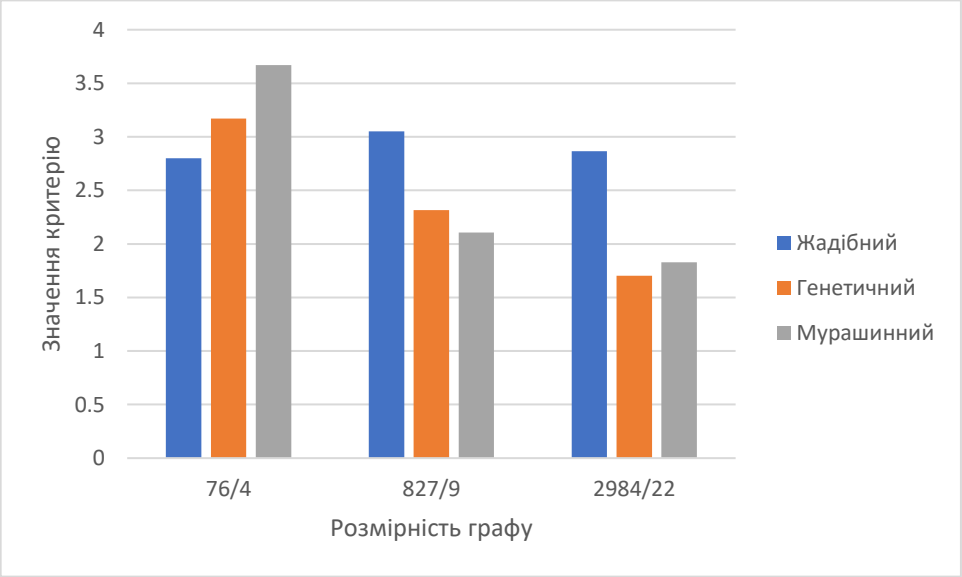


Рисунок 2.9 – Порівняльний аналіз алгоритмів для довгої стратегії

Жадібний алгоритм виправдовує своє використання на задачах невеликої розмірності але тільки для довгої стратегії, так як його принцип роботи базується на тому, що він при кожній ітерації додає шлях, який покриває найбільш можливу кількість ще не покритих тріад. На задачах середньої розмірності краще себе показав алгоритм мурашинних колоній для обох стратегій, який за однаковий час видає кращі результати чим жадібний та ГА. Проте на задачах великої розмірності він поступається ГА у швидкодії пошуку.

Висновки до розділу

У розділі наведено змістовну та математичну постановку задачі функції побудови планів тестування, яка зводиться до класичної задачі мінімального покриття множини. Проведено огляд існуючих методів розв’язання задачі, на основі цього виділені критерії оцінки покриттів.

В залежності від розмірності задачі та наявних обчислювальних ресурсів для розв’язання задачі запропоновані та розроблені метаевристичні та жадібний алгоритми. Описано схеми алгоритмів та проведено обчислювальні експерименти для визначення параметрів алгоритмів, за якими алгоритм видає кращі результати. Виконано дослідження ефективності розроблених алгоритмів для задач різної розмірності та проведено їх порівняльний аналіз.

3 ОПИС ПРОГРАМНОГО ТА ТЕХНІЧНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Засоби розробки

Основні засоби для розробки програмного забезпечення для створення системи з підтримки процесу верифікації програмного продукту наведено у таблиці 3.1.

Таблиця 3.1 – Засоби розробки

Операційна система	Windows 10
Мови програмування	C#
Фреймворки	.Net, Nunit
Бібліотека	Selenium
Засоби контролю версій та командної комунікації	Github
CI, CD сервер	TeamCity
Засоби проектування	Enterprise Architect

Код був написаний на сучасній об'єктно-орієнтованій та типубезпечній мові програмування C#, яка була спеціально створена для роботи з фреймворком .NET. Основними його перевагами є:

– підтримка декількох мов;
– потужна бібліотека класів;
– різноманітність технологій.

Для написання шаблонних тестів були використані Nunit та Selenium. Selenium - це набір інструментів, призначених для автоматизації веб - браузерів на різних платформах. Інструмент Selenium Web Driver здатний автоматизувати багато різних браузерів на різних платформах, використовуючи при цьому різні мови програмування та інтегруючись з різними тестовими фреймворками. На поточний момент Nunit є лідер серед фреймворків для тестування, навіть не дивлячись на той факт, що ставити його треба Nuget - пакетом. NUnit – це потужний засіб для тестування та має різні

варіанти для запуску. В даний час NUnit широко використовується в unit - тестуванні, інтеграційному і автоматизованому тестуваннях.

Для моніторингу та запуску тестів використовували TeamCity - систему управління побудовою застосунків і неперервної інтеграції, для організації ефективної колективної роботи над кодом та тестування програмного забезпечення. Серед функціоналу TeamCity можна відзначити:

- розумна взаємодія з продуктами розробки;
- отримання коду з VCS сховищ;
- збереження історії збірок, тестів, іншої статистики і логів змін;
- відстеження якості коду;
- створення, контроль стану інфраструктури для CI;
- інтеграція з Github, Atlassian Products, Puppit і іншими продуктами;
- можливість використання CLI, а також REST API з бібліотеками для різних мов;
- підтримка хмарних та віртуальних технологій Azure, Amazon EC2, Docker, VMWare vSphere.

3.2 Вимоги до технічного забезпечення

3.2.1 Вимоги до серверу

Для роботи системи з підтримки процесу верифікації програмного продукту необхідіні такі мінімальні параметри системи:

- ОС Windows 10;
- RAM – не менше 8 – 12 GB;
- Процесор з частотою не менше 2 GHz;
- 20-50 GB вільного місця.

3.2.2 Вимоги до клієнта

Так як сервер ТС знаходиться в Інтернеті, то у клієнта повинен бути доступ до мережі Інтернет. Версії веб-браузерів користувача повині співпадати з версію Selenium WebDriver, на даний момент це найактуальніша версія Chrome 87.0.4280.88.

3.3 Архітектура програмного забезпечення

3.3.1 Діаграма компонентів

Підсистема побудови та запуску тестів має 3 компоненти, які представляють структуру взаємодії фреймворку та систем, з якими проводиться інтеграція. Схема компонентів представлена на рисунку 3.1

Компонент Context представляє створення об’єктів та методів звернення до веб-елементів. Компонент Tests відповідає за запуск тестів, оперуючи методами компоненту Steps та вхідними даними. Компонент Steps зберігає методи комунікації з методами обробки елементів та надає доступ до їх використання.

Компонент Git – засіб контролю версій та командної комунікації. Відповідає за віддалене зберігання проекту. Дозволяє створювати запити на об’єднання відгалужених гілок з головною (master) та керувати процесом розробки програмного коду.

Компонент TC – виконується білд репозиторію та проекту, запуск тестів на віддаленому агенті, зв’язок репозиторію та агента здійснюється за рахунок його підключення до віддаленого репозиторію Git.

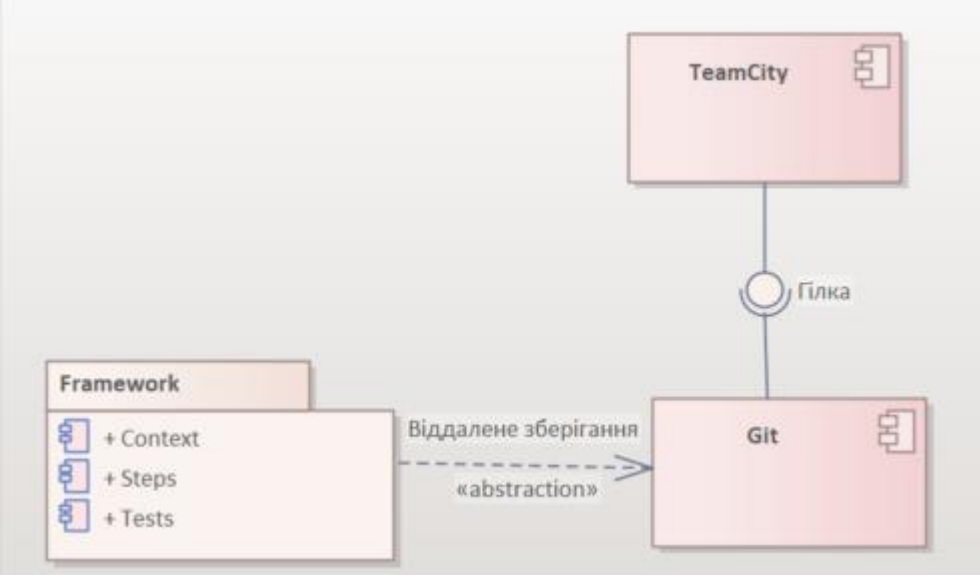


Рисунок 3.1 – Даграма компонентів

3.3.2 Діаграма послідовності

У додатку відображена діаграма послідовності для підсистеми побудови планів тестування, у верхній частині якої відображено перелік об’єктів, які у

процесі побудови плану взаємодіють між собою. Відповідно до черги пересилання між об’єктами повідомлення пронумеровані.

3.3.3 Діаграма класів

Для створення коду системи з підтримки процесу верифікації програмного продукту розроблена структурна схема класів, яка представлена у графічному матеріалі. За допомогою неї описується логічне представлення системи, створюється внутрішня структура системи, взаємне положення класів відносно один одного, оскільки класи є лише прототипами, на основі яких в майбутньому визначаються фізичні об’єкти, які слугують, щоб позначати множини об’єктів, які мають однакову структуру, поведінку та відносини з об’єктами інших класів. Розглянемо основні класи програмного продукту та їх методи, які представлені у таблиці 3.2.

Таблиця 3.2 – Класи та їх методи

Назва	**Примітка**
Main – клас для відображення головного вікна системи	
public void drawEdgeButton_Click()	Відображає ребро між двома обраними вершинами
public void drawVertexButton_Click()	Відображає вершину графа
public void sheet_MouseClick()	Виконує функції мишки
public void selectButton_Click()	Відповідає за інструменти редактора, дозволяє використовувати їх коли обраний інструмент курсор
public void showPaths_Click()	Відображає всі можливі шляхи з початкової у термінальні вершини
public void ParametresButton_Click()	Відображає форму для виставлення параметрів алгоритму
public void deleteButton_Click()	Видаляє обраний елемент з редактору
public void deleteALLButton_Click()	Видаляє всі елементи з редактору

Продовження таблиці 3.2

public void buildPaths_Click()	Відображає шляхи покриття згідно обраного алгоритму та стратегії
public void coverageDetailsbutton_Click()	Відображає форму з деталями та метрикою покриття
public void saveButton_Click()	Зберігає результати виконання у файл
DrawGraph – клас для відображення графу	
public void drawSelectedVertex()	Малює обрану вершину
public void drawVertex()	Малює ребро між двома вершинами
public void GetBitmap()	Ініціалізація секції для відображення графа
public void DrawGraph()	Відображає граф
CalculatePaths – клас для побудови всіх можливих шляхів з початкової у термінальні методом бектрекінгу	
public void calculateTriades()	Повертає список усіх тріад
public void calculatePaths ()	Повертає список ребер
public void buildPaths()	Повертає список усіх можливих шляхів
public void showPaths()	Відображає список шляхів
Genetic – клас, який реалізує пошук покриттів за допомогою генетичного алгоритму	
public void runGenetic()	Повертає список шляхів при довгій стратегії
public void runGeneticByShort()	Повертає список шляхів при короткій стратегії
public void howTriadsLeft()	Повертає число непокритих тріад
AntColony – клас, який реалізує пошук покриттів за допомогою мурашинного алгоритму	

Продовження таблиці 3.2

public void runAnts(bool shortType)	Повертає список шляхів при довгій або короткій стратегії в залежності від параметру методу
public void iWasHere()	Ознака того, що мурашка була на цьому шляхі
Greedy – клас, який реалізує пошук покриттів за допомогою жадібного алгоритму	
public void runGreedy(bool shortType)	Повертає список шляхів при довгій або короткій стратегії в залежності від параметру методу

3.4 Інструкція користувача

Для початку роботи з програмним продуктом, запустіть .exe файл, головна форма відображає редактор графів, випадаючі списки з вибором стратегії та алгоритму, кнопку з формою налаштування параметрів алгоритму, побудови планів тестування та кнопку з формою відображення результатів покриття. Головна форма показана на рисунку 3.2.

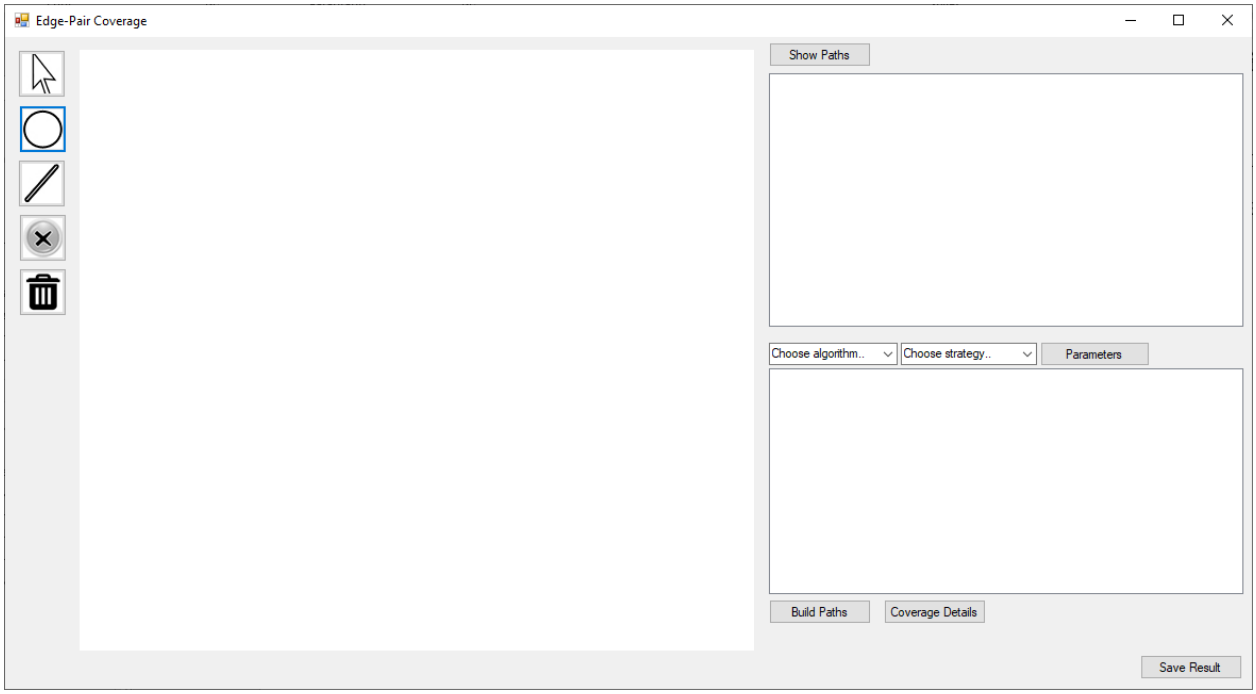


Рисунок 3.2 – Головна форма програмного продукту

Для побудови графу потрібно використати інструменти редактору у лівій частині форми. Наявні такі інструменти як додавання вершини, ребра, видалення вершини/ребра, видалення всього графу. В якості прикладу оберемо діаграму станів та переходів для переходів основної частини функціоналу інтернет-магазину https://pethouse.ua/, яка зображена на рисунку 3.3. Кожному стану буде відповідати вершина графу, а перехід між станами – ребру.

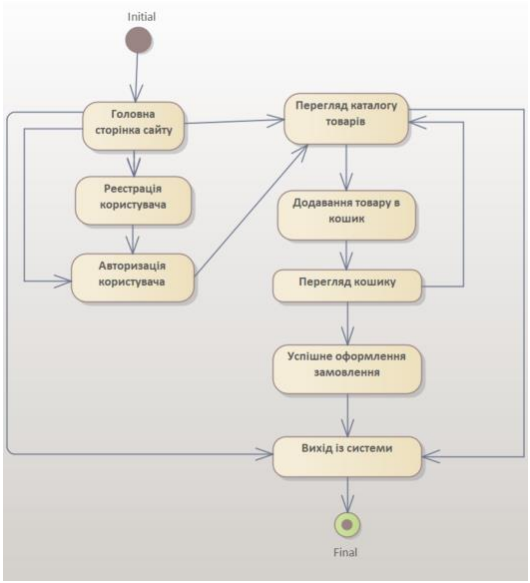


Рисунок 3.3 – Діаграма станів та переходів

На рисунку 3.4 відображений граф створений за допомогою редактору (його граф також представлений у п.1.3).

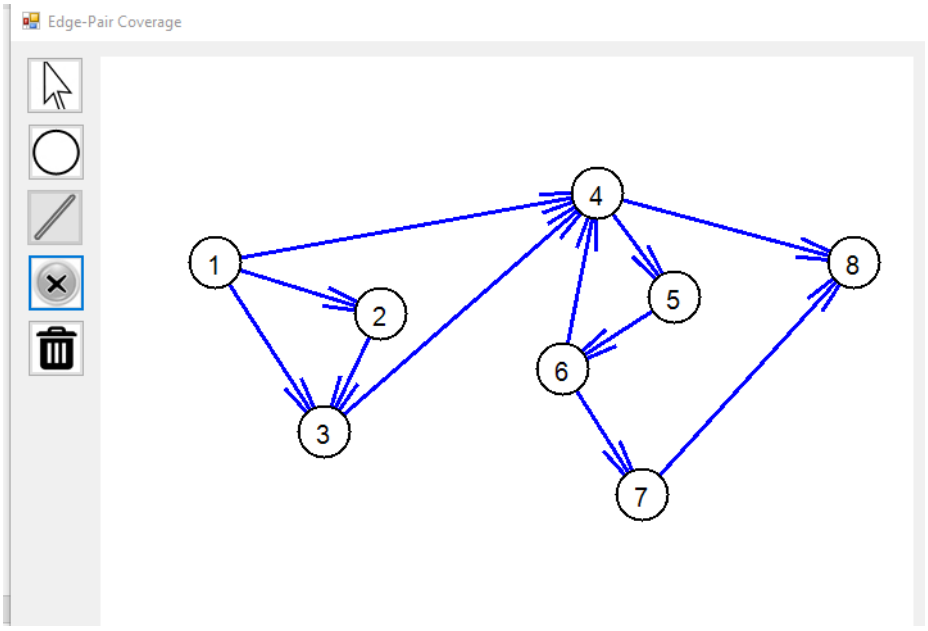


Рисунок 3.4 – Граф створений за допомогою редактору графів

На рисунку 3.5 відображено всі можливі шляхи графу. Для того, щоб переглянути всі шляхи з початкової вершини у термінальні(в даному випадку початковою вершиною вважається та, в яку не входить ні одна дуга, а терміінальними ті, з яких не виходить ні одна дуга) необхідно натиснути на кнопку Show Paths.

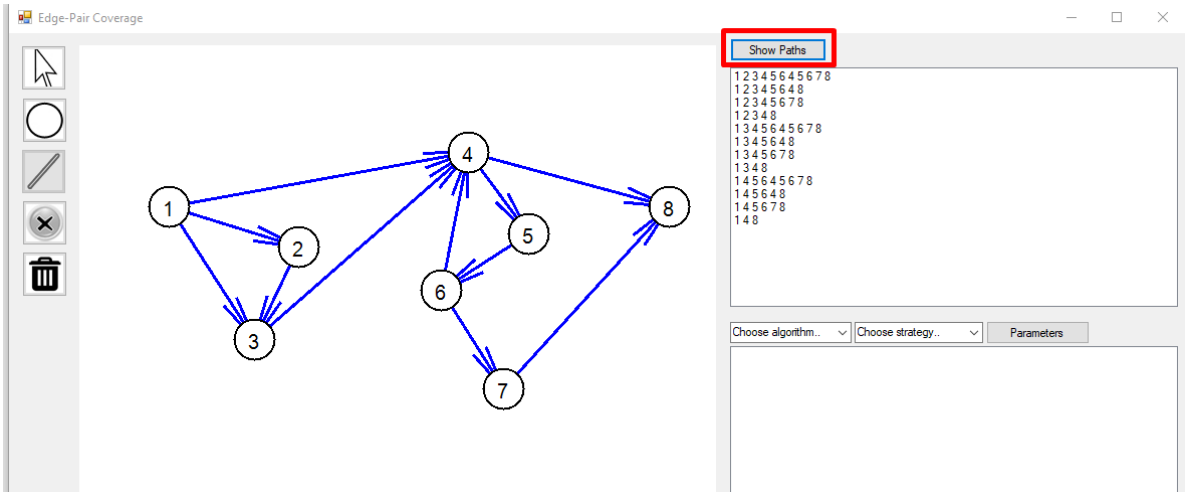


Рисунок 3.5 – Перелік всіх можливих шляхів графу

Для того, щоб налаштувати параметри алгоритму, необхідно у випадаючому списку обрати потрібний алгоритм та натиснути кнопку Parameters, після цього обравши стратегію та натиснувши кнопку Build Paths отримуємо перелік шляхів, які відповідають плану тестування. Параметри ACO алгоритму та результат роботи програми для короткої стратегії відображено на рисунку 3.6.

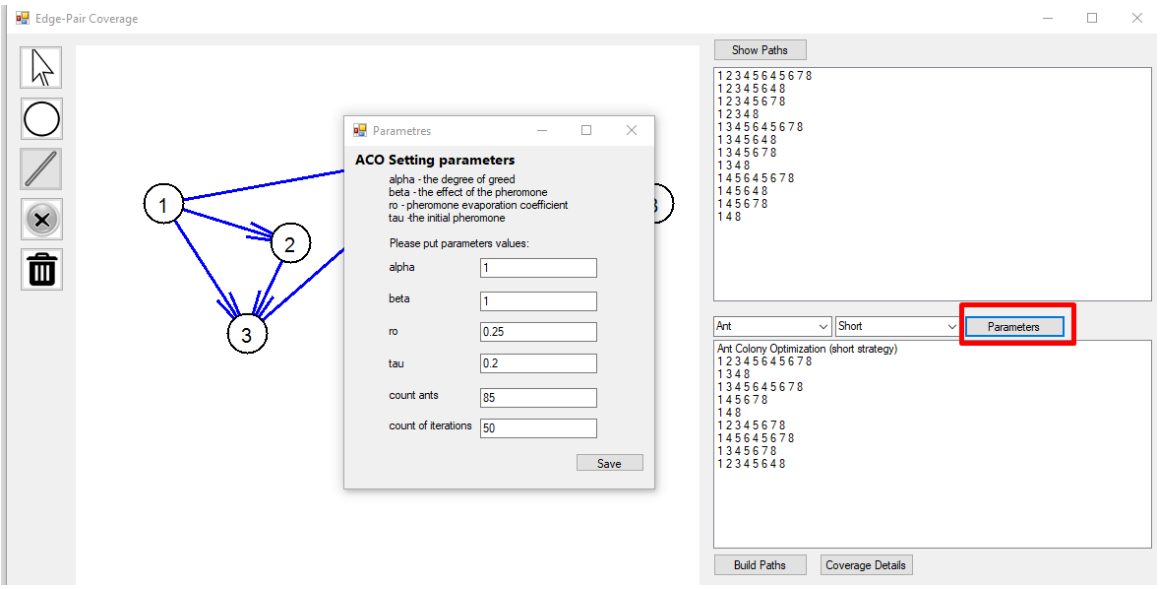


Рисунок 3.6 – Результат роботи ACO алгоритму

Після побудови плану тестування є можливість переглянути деталі покриття у формі Coverage Details, яка відображає кількіть тестів, середню довжину тестів та середнє відхилення. На рисунку 3.7 відображено деталі покриття для попередього випадку.

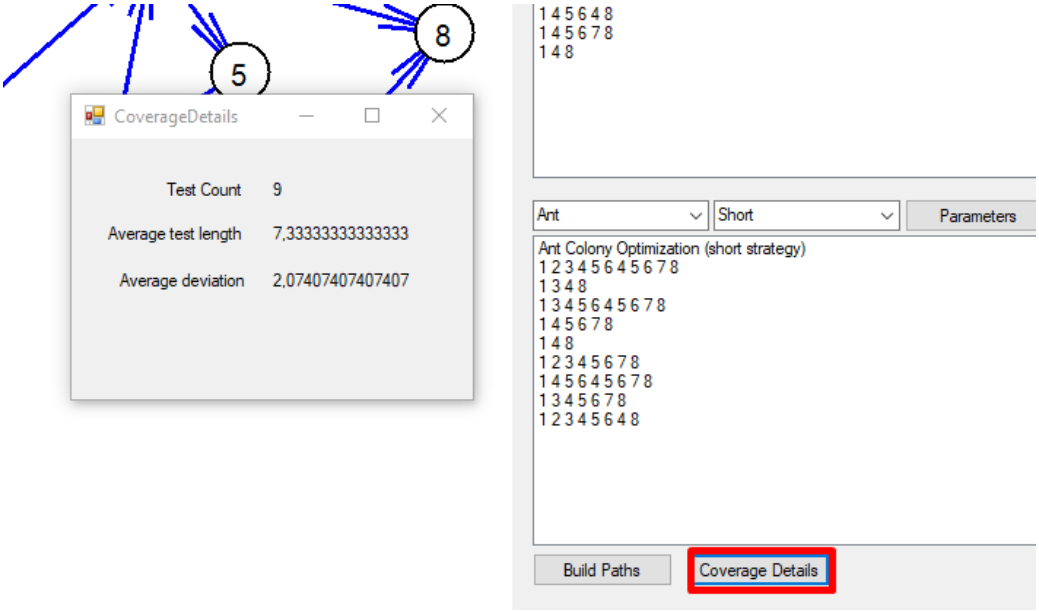


Рисунок 3.7 – Деталі покриття

Після побудови плану тестування користувая повинен зберегти результати у файл, а саме перелік шляхів за допомогою кнопки Save Results. На рисунку 3.8 відображено файл з збереженими результатами.

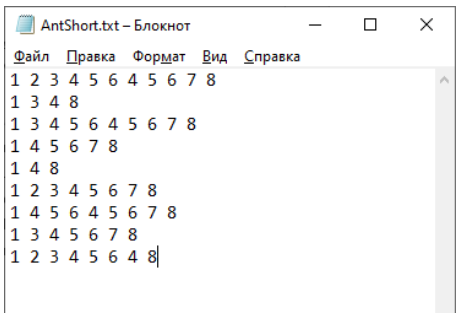


Рисунок 3.8 – Файл з збереженими результатами

Використовуючи цей файл, користувач генерує автотести у Visual Studio. На рисунку 3.9 відображено вікно Test Explorer з переліком тестів.

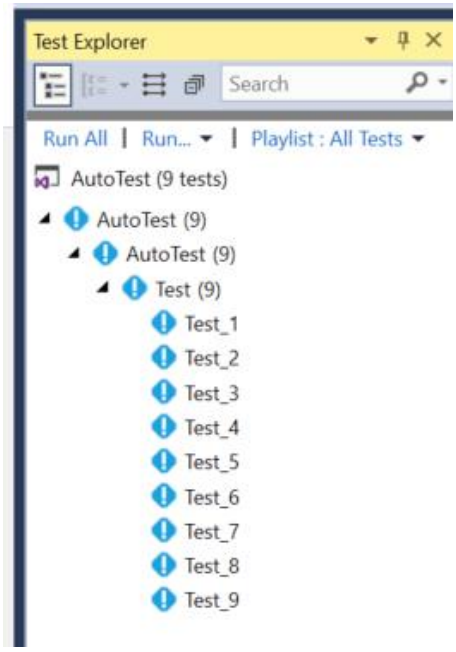


Рисунок 3.9 – Вікно Test Explorer фреймворку тестування

За допомогою GitHub розширення потрібно зберігаємо зміни у віддаленому репозиторію, щоб мати останню версію проекту для запуску тестів у TC. На сторінці TC для запуску тестів користувач повинен натиснути Run навпроти створеної конфігурації. На рисунку 3.9 відображено процес запуску тестів.

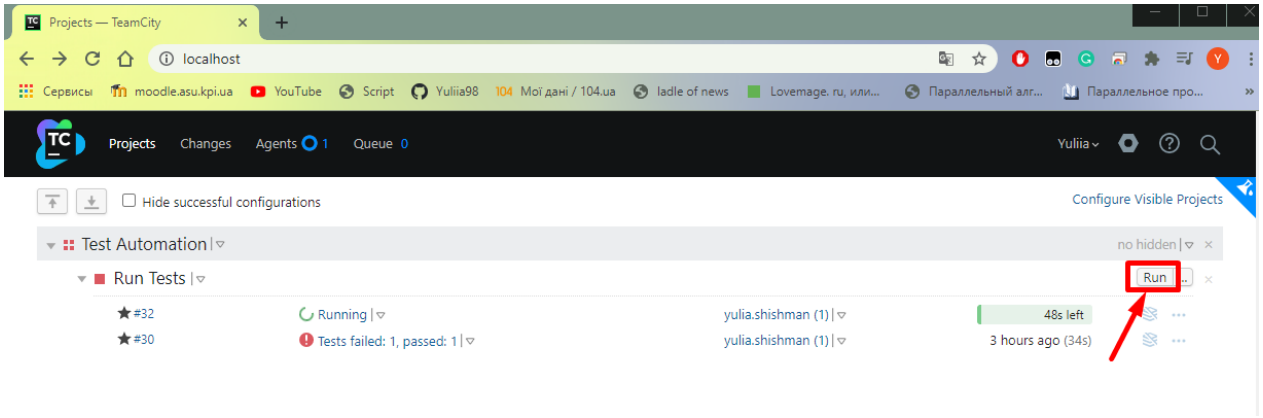


Рисунок 3.9 – Запуск тестів на TC

Після завершення роботи агента користувач може переглянути результати роботи. На рисунку 3.10 відображено статус виконання кожного тесту.

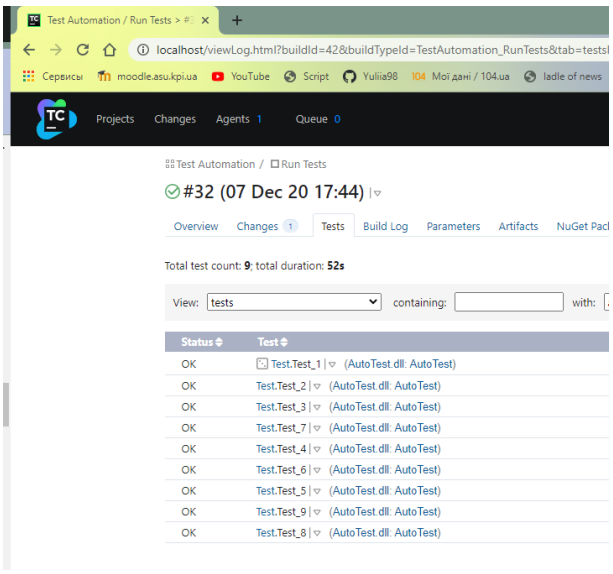


Рисунок 3.10 – Результати роботи TC агента

Висновки до розділу

У розділі було показано основні засоби розробки системи з підтримки процесу верифікації ПП. Наведені вимоги то технічного забезпечення серверу та користувачів. Представлено архітектуру програмного забезпечення за допомогою діаграм компонентів, класів та послідовності. Наведено інструкцію користувача.

4 РОЗРОБКА СТАРТАП-ПРОЕКТУ

4.1 Назва проекту

Мобільний застосунок «My Kitchen».

4.2 Короткий опис проекту

Мобільний застосунок - помічник на кухні, який фіксує запаси продуктів у вашому холодильнику та допомагає створювати списки покупок, які слід зробити. Ці покупки можна замовити додому з доставкою з обраного магазину або ж знайти магазин неподалік, у якому є всі (або майже всі) необхідні продукти. Також прямо з застосунку можна замовити вже готову їжу з ресторану. Помічник порадить вам страву, яку ви можете приготувати з наявних продуктів, або що треба докупити для того, щоб приготувати конкретну страву. Також, можна знайти людину неподалік, яка буде рада допомогти вам з приготуванням нової страви за плату або безкоштовно. Або навіть якщо така людина далеко – ви можете готувати через відеозв’язок.

4.3 Бізнес-модель

4.3.1 Цінний продукт

Цінними якостями системи «My Kitchen» відмінними від існуючих є (фактори створення цінності):

– зберігає список запасів продуктів;

– надає список рекомендованих до покупки продуктів;

– надає список страв, які користувач може приготувати з наявних продуктів;

– замовлення доставки продуктів з магазинів;

– здійснює пошук магазинів, де є всі необхідні продукти;

– замовлення доставки їжі з ресторану;

– пошук людей поблизу, які навчать вас готувати.

4.3.2 Сегмент споживачів

Представимо цільову аудиторію застосунку у вигляі таблиці 4.1

Таблиця 4.1 – Сегмент споживачів

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1	Економія часу за рахунок доставки їжі з супермаркету	Люди з щільним графіком, люди яким не цікаво ходити за покупками	Відсутні	Інтуітивна зрозумілість в користуванні, висока якість продуктів
2	Рекомендації щодо страв	Господині, які мають брак фантазії	Відсутні	Цікаві пропозиції, зручна інструкція
3	Списки покупок	Заклопотані люди	Відсутні	Актуальні списки
4	Доставка їжі з ресторанів	Офісні робітники, Самотні чоловіки, Іменинники	Відсутні	Зручний процес замовлення, швидка доставка, якісна їжа
5	Пошук людей, які допомогли б опанувати кулінарну майстерність та склали б компанію в приготуванні їжі	Самотні люди, які потребують соціальної взаємодії Студенти Люди, які хотіли б навчитися новим рецептам від інших	Відсутні	Велика мережа користувачів

В майбутньому система повинна мати змогу масштабування, для використання її за кордоном.

4.3.3 Канали збуту

Каналами збуту є:

- продажі «My Kitchen» користувачам безпосередньо від Play Market;
- продажі з офіційного сайту стартап-проекту.

4.3.4 Взаємодія з споживачами

1. Залучення.

Надання клієнтам різних бонусів, залучення нових користувачів за рахунок реклами у безкоштовній версії, % скидок для нових користувачів, яких приведе користувач застосування.

2. Підтримка користувачів здійснюється через:

- інтерактивний сайт «www.mykitchen.com.ua»;
- телеграм;
- коментарі до застосунку у Play Market

4.3.5 Дохід (монетизація)

Дохід отримуватиметься за допомогою прямих та посередницьких продажів застосунку з play market чи офіційного сайту, також при умові налаштування партнерства з ресторанами та супермаркетами – дохід від доставки.

Дохід стартапом буде отримуватись від каналів збуту систем відповідно до 4.3.3.

4.3.6 Ключові види діяльності

Головним видом діяльності стартапу буде здійснення проекту реалізації застосунку «My Kitchen». Для вирішення проблеми користувача щодо створення списку покупків, замовлення покупків була зроблена:

- розробницька діяльність - закінчення розробки сайту/застосунку;
- наукова діяльність - продовження дослідно-експериментальних робіт по покращенню інтерфейсу сайту/застосунку;
- маркетингова діяльність - збільшення каналів донесення інформації до потенційних користувачів.

4.3.7 Ключові ресурси

Ключовими ресурсами системи є :

- програмні - функціонал сайту/застосунку розроблятиметься кваліфікованими робітниками у орендованому стартапом офісі;
- матеріальні - ПК чи ноутбук, для мобільного застосунку – телефон;

– інтелектуальні ресурси - будуть використані власні технічні розробки, різні патенти, креслення.

4.3.8 Людські ресурси

Людськими ресурсами даної системи є:

– директор – маркетолог із вищою освітою та сертифікатом менеджера стартапу, менеджер управлінець-економіст, досвід роботи не менше 4 років;

– менеджер з продажу;

– маркетолог з досвідом роботи не менше 2 роки;

– IT розробники (IOS, Android, Front-End, Back-End, MS SQL).

Всього: 8 працівників.

4.3.9 Витрати

Виготовлення та реалізація даної системи:

– повна собівартість реалізованої продукції стартапу – 900 000 грн;

– прибуток стартапу – 36 000 000 грн;

– оптова ціна виробника – 900 000 грн;

Непрямі податки:

– податок на додану вартість – 180 000 грн;

– оптова ціна відпускна – 900 000 грн.

4.4 Споживчі властивості товару

– професійне спеціалізоване програмне забезпечення;

– надає дані в режимі реального часу;

– система підкаже людину неподалік, що з радістю допоможе вам на кухні та навчить готувати нову страву;

– база рецептів постійно поповнюється користувачами;

– система надає рекомендації страв, які можна приготувати з наявних продуктів.

4.5 Дослідження ринку

Існують подібні застосунки, що мають частковий функціонал (або рецепти, або списки продуктів, або доставка), повноцінного конкурента нема, принаймні в Україні.

4.6 Дослідження конкурентного оточення

Таблиця 4.2 – Аналіз конкурентів

№ п/п	Технічно-економічні характеристики ідеї	Товари/концепції конкурентів			
		Мій проект	**Zakaz**	**Афиша - Еда**	**Готовят все!**
1.	Доступність для користувачів	Безкоштовно / 55 грн	Загальнодоступний сайт	Безкоштовно	Безкоштовно / 83 грн ($3)
2.	Якість	Вища середнього	Висока	Середня	Середня
3.	Повнота функціоналу	Весь функціонал	Лише доставка з супер-маркетів	Тільки рецепти по інгредієнтам	Тільки рецепти
4.	Зручність інтерфейсу	Висока	Середня	Середня	Середня
5.	Витрати на розробку та впровадження	Дуже великі	Великі	Малі	Малі

4.7Маркетингова стратегія просування

Орієнтація на якісний застосунок з релевантними рекомендаціями. Концепцією рекламного звернення буде «Приготування їжі – це весело та легко».

4.8 Елементи фінансового плану

4.8.1 Опис бізнес-проекту

Розробка мобільного застосунку, де було б все в одному, що стосується продуктів та харчування.

4.8.2 Опис товару/послуги/технології

Мобільний застосунок - помічник на кухні, який фіксує запаси продуктів у вашому холодильнику та допомагає створювати списки покупок, які слід зробити. Ці покупки можна замовити додому з доставкою з обраного магазину або ж знайти магазин неподалік, у якому є всі (або майже всі) необхідні продукти. Також прямо з застосунку можна замовити вже готову їжу з

ресторану. Помічник порадить страву, яку ви можете приготувати з наявних продуктів, або що треба докупити для того, щоб приготувати конкретну страву. Також можна знайти людину неподалік, яка буде рада допомогти з приготуванням нової страви за плату або безкоштовно. Або навіть якщо така людина далеко – можете готувати через відеозв'язок.

4.8.3 Маркетинг та продаж

Дослідження ринку показало, що даний проект є першим на ринку. Поки не має системи яка б об'єднувала в собі всі функції, які були перечилені вище.

4.8.4 Фінансовий план

Витрати на організацію стартапу на період 12 місяців:

– організацію виробничої ділянки, виробничого приміщення, електропостачання, закупівля обладнання, організація складу матеріалів та інструментів, охорони, умов безпечної роботи– 1 500 000 грн;

– створення робочої команди стартапу, прийом на роботу 8-и робітників – 900 000 грн;

– закупівля ліцензійного забезпечення – 120 000;

– оренда приміщення – 210 000 грн;

– комунальні платежі – 60 000 грн;

– резерв, непередбачувані витрати - 45 000 грн.

4.8.5 Резюме

Система планується поширюватися за рахунок прямих продажів через Play Market та за допомогою відсотка від доставки. В Play Market та App Store буде випущена бета-версія, яка буде безкоштовною та надалі користувач може придбати платну версію, застосування буде орієнтоване на жителів міста Київ, якщо це стосується замовлення та доставки їжі з ресторанів.

4.9 Презентація проекту інвестору

4.9.1 Ідея (суть) проекту

Ідея - спрощення процесу приготування їжі та походу в магазин за рахунок створення універсального мобільного застосунку.

4.9.2 Опис проблеми або можливості

Сильні сторони: - зростання попиту на мобільні пристрої; - все в одному, що стосується продуктів та харчування; - рекомендації щодо покупок та страв «у кишені»; - можливість знайти друзів.	Слабкі сторони: - велика трудоємкість налагодження мережі партнерів; - великі затрати на розробку; - високі ризики неокупності.
Можливості: - широка мережа партнерів; - великий попит серед киян; - зростання попиту на доставку.	Загрози: - обмеженість зацікавлених користувачів; - не готовність користувачів платити за проект; - неготовність супермаркетів та ресторанів йти на співпрацю.

4.9.3 Рішення

В результаті використання “My Kitchen” за призначенням споживач має можливість:

– зберігати список запасів продуктів;

– переглянути список рекомендованих до покупки продуктів;

– отримати список страв, які користувач може приготувати з наявних продуктів;

– замовити доставку продуктів з магазинів;

– здійснювати пошук магазинів, де є всі необхідні продукти;

– замовити доставку їжі з ресторану;

– знайти друзів.

4.9.4 Конкуренти

Існують подібні застосунки, що мають частковий функціонал (або рецепти, або списки продуктів, або доставка), повноцінного конкурента нема, принаймні в Україні.

4.9.5 Ринок

Новий товар на старий ринок.

4.9.6 Маркетингова стратегія

Активне просування через сайт/Play Market, активне просування через прямі продажі.

4.9.7 Поточна ситуація

На даний момент розробляється мобільний застосунок та сайт.

4.9.8 Команда проекту

Попередні домовленості про участь 3-х потенційних робітників.

4.9.9 Фінансові показники

Доопрацьовуються. Попередньо на 12 місяців витрати складають 750 тис. гривень.

4.9.10 Пропозиція інвестору

Вкласти угоду на отримання 20% від продажу застосунку протягом 3-х місяців та з подальшими переговорами про продовження співпраці.

4.10 Подальші кроки в проекті

4.10.1 Наукова діяльність

Продовження дослідно-експериментальних робіт по покращенню інтерфейсу сайту/застосунку, розробка нових функціональних можливостей.

4.10.2 Організаційна діяльність

Розширення обсягів виробництва продуктів, виробничої бази, кількості робочих місць.

4.10.3 Маркетингова діяльність

Збільшення каналів донесення інформації до потенційних користувачів.

Висновки до розділу

У розділі розроблено стартап проект під назвою «My Kitchen». Майже кожна людина стикається з проблемою формування списку продуктів, з питанням, що приготувати та поїсти, де і що можна замовити. Даний стартап допомогає фіксувати запаси продуктів та на основі цього може порекомендувати страву, ресторан, допомогти створити список продуктів або замовити їжу з ресторану. Даний проект є новим в своїй галузі, що дає непогані шанси на успіх.

ВИСНОВКИ

У дисертації досліджено процес тестування програмних продуктів за допомогою побудови планів тестування на основі діаграми станів та переходів. Розглянуто основні види тестування, описано структуру бізнес процесів у вигляді контекстної діаграми та наведено схеми структурні варіантів використання та діаграми послідовності різних процесів.

Наведено змістовну та математичну постановку задачі функції побудови планів тестування, яка зводиться до класичної задачі мінімального покриття множини. Проведено огляд існуючих методів розв’язання задачі та на основі цього виділені критерії оцінки покриттів.

В залежності від розмірності задачі та наявних обчислювальних ресурсів для розв’язання задачі запропоновані та розроблені метаевристичні та жадібний алгоритми. Проведені обчислювальні експерименти для визначення параметрів алгоритмів, за якими алгоритм видає кращі результати. Результати показали, що ГА чутливий до розміру початкової популяції, та результат роботи алгоритму погіршується з збільшенням кількості точок схрещування. Виконано дослідження ефективності розроблених алгоритмів для задач різної розмірності та проведено їх порівняльний аналіз. Жадібний алгоритм є ефективним на задачах невеликої розмірності для довгої стратегії, на задачах середньої розмірності краще себе показав алгоритм мурашинних колоній для обох стратегій, який за однаковий час видає кращі результати чим жадібний та ГА. Проте на задачах великої розмірності він поступається ГА у швидкодії пошуку майже в 1.5 рази.

Наведено основні засоби розробки системи з підтримки процесу верифікації програмного продукту. Представлено архітектуру програмного забезпечення за допомогою діаграм компонентів, класів та послідовності. Наведено інструкцію користувача для системи з підтримки процесу верифікації програмного продукту. Результати виконання показали, що впровадження такої системи у процес розробки значно зекономить часові та фінансові ресурси.

Розроблено стартап - проект під назвою «My Kitchen» для спрощення процесу приготування їжі та походу в магазин за рахунок створення універсального мобільного застосунку, який допомогає фіксувати запаси продуктів та на основі цього може порекомендувати страву, ресторан, допомогти створити список продуктів або замовити їжу з ресторану.

ПЕРЕЛІК ПОСИЛАНЬ

1. Шишман Ю.М. Огляд методів розв’язання задачі для вирішення проблеми автоматизації верифікації програмного забезпечення / О. Г. Жданова, М.О. Сперкач, Ю.М. Шишман // Матеріали IX Міжнародної науково-практично конференції «DYNAMICS OF THE DEVELOPMENT OF WORLD SCIENCE» - м. Ванкувер, 13-15 травня 2020 р. – С. 488-504.

2. Шишман Ю.М. Система з підтримки процесу верифікації програмного продукту / О. Г. Жданова, Ю.М. Шишман // Матеріали V всеукраїнської науково-практичної конференції молодих вчених та студентів «Інформаційні системи та технології управління» (ІСТУ-2020) – м. Київ.: НТУУ «КПІ ім. Ігоря Сікорського», 26-27 листопада 2020 р. – С. 52-58.

3. Тестування програмного продукту [Електронний ресурс URL: https://uk-wiki.ru/wiki/%D0%A2%D0%B5%D1%81%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F

4. Тестування програмного забезпечення [Електронний ресурс URL: https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F

5. Оператори вибору батьків [Електронний ресурс URL: https://uk.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%B8_%D0%B2%D0%B8%D0%B1%D0%BE%D1%80%D1%83_%D0%B1%D0%B0%D1%82%D1%8C%D0%BA%D1%96%D0%B2

6. Жданов О. Г. Про проблему генерації планів тестування для автоматизації процесу верифікації програмного продукту / О. Г. Жданова, О.

Б. Падалко // Вісник НТУУ «КПІ». Інформатика, управління та обчислювальна техніка : збірник наукових праць. – 2009. – № 50. – С. 34–41.

7. Герман О. В., Ефремов О. В. Алгоритм решения обобщенной задачи о покрытии / / Экономика и мат. методы. 1998. Т. 34, вып. 4. С. 134-140.

8. Hochbaum D. S. Approximating covering and packing problems: set cover, vertex cover, independent set, and related problems / / Approximation algorithms for NP-hard problems. Boston: PWS Publ. Co., 1997. P. 94-143.

9. Кузюрин Н. Н. О сложности построения асимптотически оптимальных покрытий и упаковок / / Докл. РАН. 1998. Т. 363, № 1. С. 11-13.

10. Сапоженко А. А., Асратян А. С, Кузюрин Н. Н« Обзор некоторых результатов по задачам о покрытии / / Методы дискретного анализа в решении комбинаторных задач: Сб. науч. тр. Новосибирск: Ин-т математики СО АН СССР, 1977. Вып. 30. С. 46-75.

11. Журавлев Ю. И. Теоретико-множественные методы алгебры логики // Проблемы кибернетики. Вып. 8. М.: Физматгиз, 1962. С. 5-44.

12. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. М.: Мир, 1982.

13. Krarup J., Pruzan P. M. The simple plant location problem: survey and synthesis / / European J. Oper. Res. 1983. V. 12, N 1. P. 36-81.

14. Кристофидес Н. Теория графов. Алгоритмический подход. М.: Мир, 1978.

15. Caprara A., Fischetti M., Toth P. Algorithms for the set covering problem / DEIS — Operations Research Group. 1998. Technical Rep. No. OR- 98-3.

16. Karpinski M., Zelikovsky A. Approximating dense cases of covering problems / / Network design: connectivity and facilities location. Providence, RT: Amer. Math. Soc, 1998. P. 169-178. (DIMACS Ser. Discrete Math. Theoret Cornput. Sci.; V. 40.).

17. Balas E., Ho A. Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study / / Math. Program. Study . 1980. V. 12. P. 37-60.

18. Garfinkel R. S. Nemhauser G. L. Integer programming. New York: Wiley, 1972.

19. Eremeev A. V., Kolokolov A. A., Zaozerskaya L. A. A hybrid algorithm for set covering problem / / Proc. of Intern, workshop on discrete optimization methods in sheduling and computer-aided design. Minsk, 2000. P. 123-129.

20. Lifschitz V., Pittel B. The worst and the most probable performance of a class of set-covering algorithms / / SIAM J. Cornput. 1983. V. 12, N 2 . P. 329-346

21. Beasley J. E., J0nsten K. Enhancing an algorithm for set covering problems / / European J. Oper. Res. 1992. V. 58, N 2. P. 293-300.

22. Balas E., Carrera M. C. A dynamic subgradient-based branch and bound procedure for set covering / / Oper. Res. 1996. V. 44, N 6. P. 875-890.

23. Заозерская Л. А. Об одном алгоритме перебора L-классов для решения задачи о покрытии множества / / Тр. 11-й Байкальской междунар. школы-семинара «Методы оптимизации и их приложения». Иркутск, 1998. Т. 1. С. 139-142.

24. Feige U. A threshold of In *n* for approximating set cover / / Proc. of the 28th annual ACM symp. on theory of computing. New York: ACM Press, 1996 . P. 314-318.

25. Raz R., Safra S. A sub-constant error-probability low-degree test, and subconstant error-probability PCP characterization of NP / / Proc. of the 29th annual ACM symp. on theory of computing. New York: ACM Press, 1997 . P. 475-484.

26. Нечепуренко М. И., Попков В. К., Майнагашев С. М. и др. Алгоритмы и программы решения задач на графах и сетях. Новосибирск: Наука, 1990.

27. Slavik P. A tight analysis of the greedy algorithm for set cover / / J. Algorithms . 1997. V. 25, N 2. P. 237-254.

28. Bar-Yehuda R., Even S. A linear-time approximation algorithm for the weighted vertex cover problem / / J. Algorithms. 1981. V. 2, N 2. P. 198-203.

29. Back Th., Schiiltz M., Khuri S. A comparative study of a penalty function, a repair heuristic, and stochastic operators with the set-covering problem // Artificial

Evolution. Proc. Berlin: Springer, 1996. P. 3-20. (Lecture Notes in Comput. Sci.; V. 1063.).

30. Ramalhinho H., Pinto R., Portugal R. Metaheuristics for the bus-driver scheduling problem / Univ. Pompeu Fabra. Economic Working Papers Series . 1998. Technical Rep. No. 304.

31. Alexandrov D.; Kochetov Yu. Behavior of the ant colony algorithm for the set covering problem / / Operations Research Proceedings 1999 (Magdeburg, 1999). Berlin: Springer, 2000. P. 255-260.

32. Cliques, coloring, and satisfiability. Providence, RI: Amer. Math. Soc, 1996. (DIMACS Ser. in Discrete Math. Theoret. Comput. Sci.; V. 26.).

33. Balas E., Niehaus W. Optimized crossover-based genetic algorithms for the maximum cardinality and maximum weight clique problems //J . Heuristics. 1998. V. 4, N 2. P. 107-122.

34. Caprara A., Fischetti M., Toth P. Algorithms for the set covering problem / DEIS — Operations Research Group. 1998. Technical Rep. No. OR98-3.

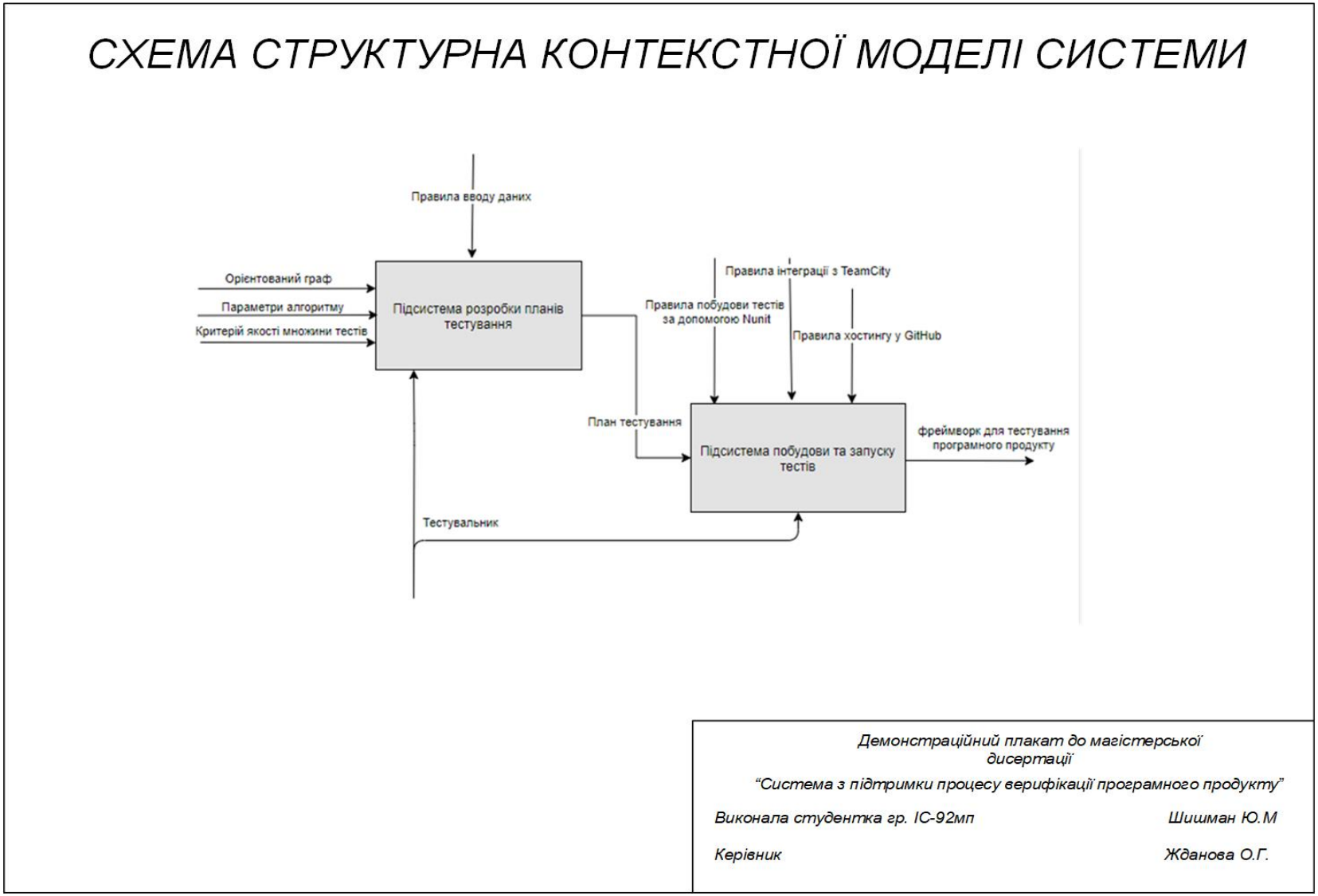
35. Balas E., Carrera M. C. A dynamic subgradient-based branch and bound procedure for set covering // Oper. Res. 1996. V. 44, N 6. P. 875-890.

36. Задача о покрытии множества: сложность, алгоритмы, экспериментальные исследования [Електронний ресурс URL: http://math.nsc.ru/publishing/DAOR/content/2000/06/22.pdf

37. Kalyanmoy D., Debayan D., Analyzing Mutation Schemes for Real-Parameter Genetic Algorithms// KanGAL Report Number 2012016.

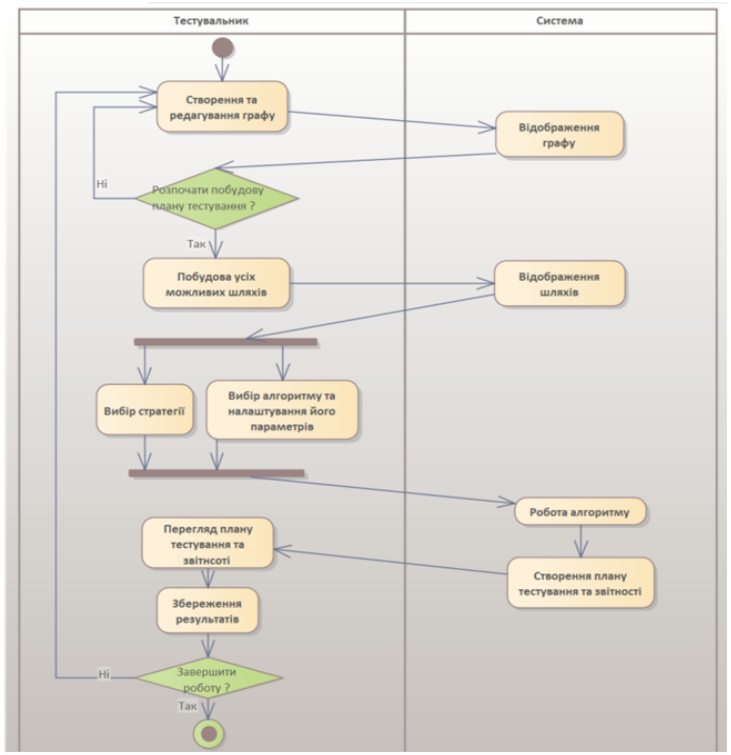
ДОДАТОК А Графічний матеріал

Схема структурна контекстної моделі системи



Діаграма діяльності побудови планів тестування

ДІГРАМА ДІЯЛЬНОСТІ ПОБУДОВИ ПЛАНІВ ТЕСТУВАННЯ



Демонстраційний плакат до магістерської дисертації

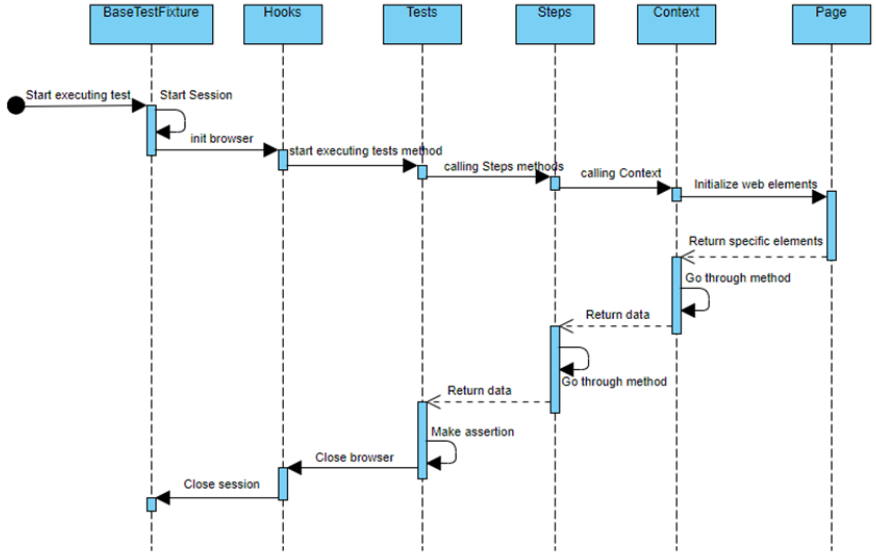
"Система з підтримки процесу верифікації програмного продукту"

Виконала студентка гр. ІС-92мп *Шишман Ю.М*

Керівник *Жданова О.Г.*

Діаграма послідовності роботи розробленого фреймворку

ДІАГРАМА ПОСЛІДОВНОСТІ РОБОТИ РОЗРОБЛЕНОГО ФРЕЙМВОРКУ



Демонстраційний плакат до магістерської дисертації

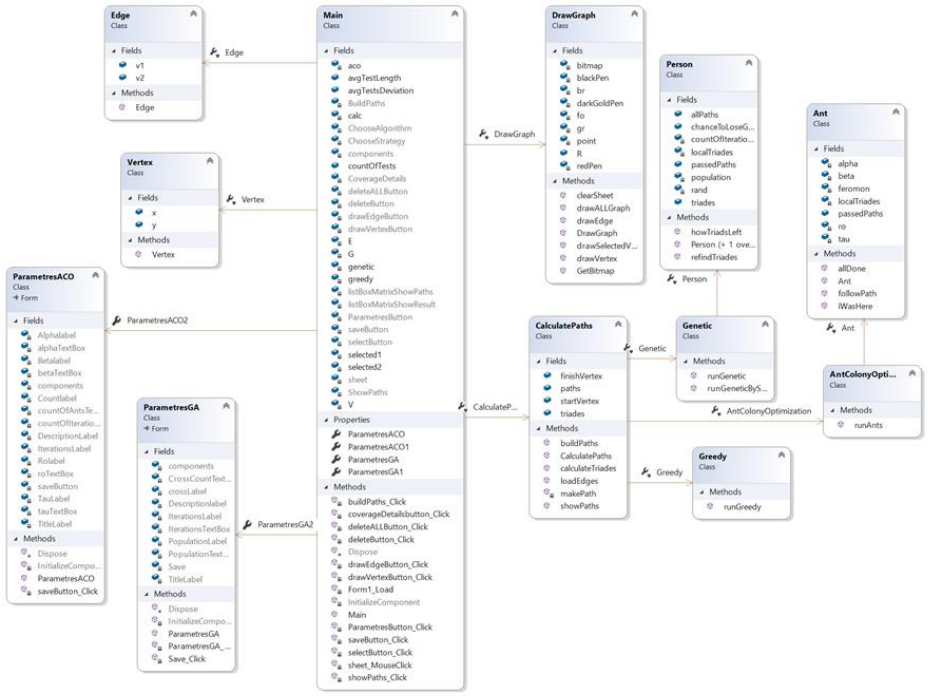
"Система з підтримки процесу верифікації програмного продукту"

Виконала студентка гр. ІС-92мп *Шишман Ю.М*

Керівник *Жданова О.Г.*

Схема структурна класів програмного забезпечення

СХЕМА СТРУКТУРНА КЛАСІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



Демонстраційний плакат до магістерської дисертації

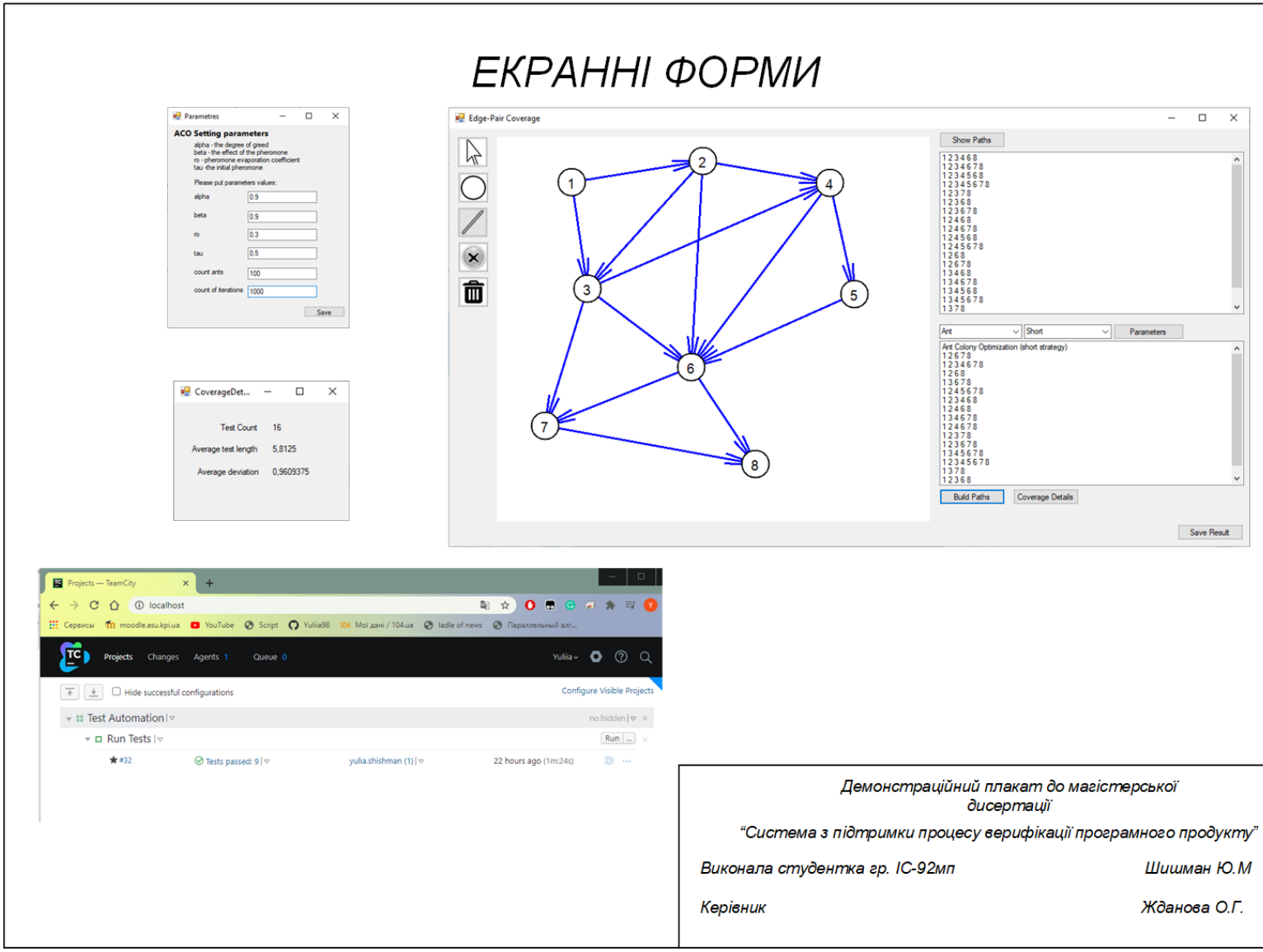
"Система з підтримки процесу верифікації програмного продукту"

Виконала студентка гр. ІС-92мп *Шишман Ю.М*

Керівник *Жданова О.Г.*

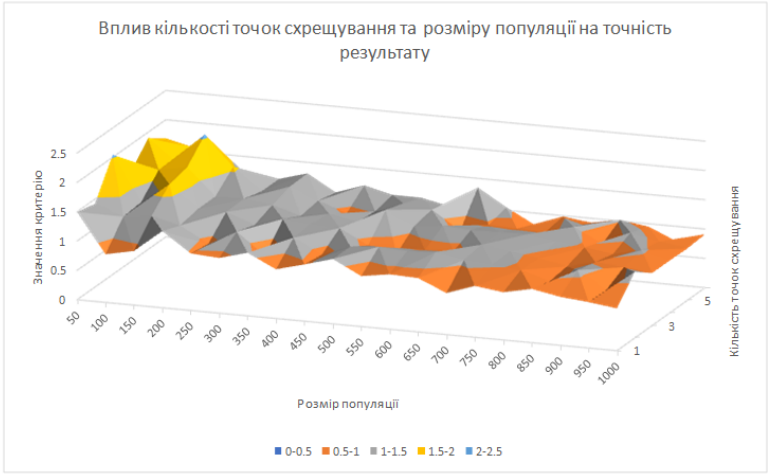
Блок-схема алгоритму генерації популяції

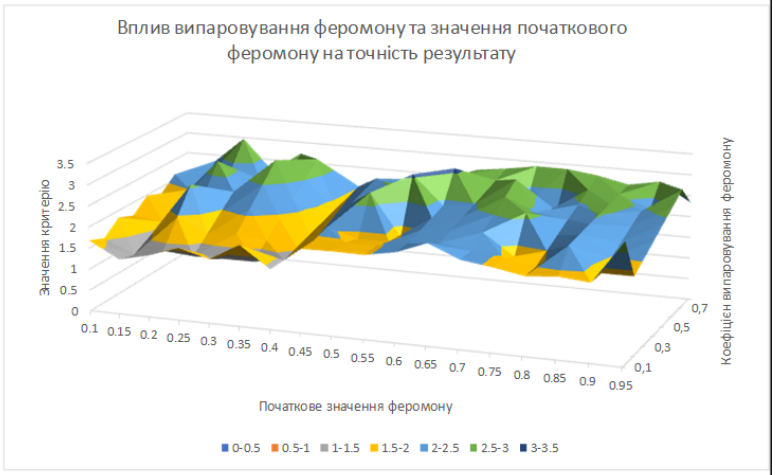
Екранні форми



Результати експериментів

РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТІВ





Демонстраційний плакат до магістерської дисертації

“Система з підтримки процесу верифікації програмного продукту”

Виконала студентка гр. ІС-92мп *Шишман Ю.М*

Керівник *Жданова О.Г.*