КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 3 (31), 2025
ISSN 2663 - 4023
DOI 10.28925/2663-4023.2025.31.1005
УДК 004.77

Романенко Сергій Олексійович
старший викладач кафедри комп'ютерних наук та інтелектуальних технологій
Військовий інститут телекомунікацій та інформатизації імені Героїв Крут, Київ, Україна
ORCID: 0009-0004-0240-0777
serhii.romanenko@viti.edu.ua

Редзюк Євгеній Володимирович
кандидат технічних наук, доцент,
начальник кафедри комп'ютерних наук та інтелектуальних технологій
Військовий інститут телекомунікацій та інформатизації імені Героїв Крут, Київ, Україна
ORCID: 0000-0001-5592-5121
yevhenii.redziuk@viti.edu.ua

Голуб Олександр Олександрович
старший викладач кафедри комп'ютерних наук та інтелектуальних технологій
Військовий інститут телекомунікацій та інформатизації імені Героїв Крут, Київ, Україна
ORCID: 0009-0006-1122-7667
oleksandr.golub@viti.edu.ua

МОДЕЛЮВАННЯ ВЕБЗАСТОСУНКІВ У ФОРМАТІ ГРАФІВ ЗНАНЬ

Анотація. Традиційні методи вебсканування та парсингу, які ґрунтуються на вилучення та рекурсивне відстеження гіперпосиланнями для створення карти вебзастосунку, дедалі більше стають неефективними для сучасних, динамічних вебзастосунків. Даний метод створює граф, де вузли є лише фрагментами неструктурованих даних, а ребра представляють прості переходи між сторінками. Він фундаментально не здатний охопити багату інтерактивну поведінку та зміни станів, що є визначальними для сучасних застосунків. У відповідь пропонується новий підхід, який моделює вебзастосунок не як набір посилань, а як систему структурованих станів. Згідно з цією парадигмою, кожен вузол представляє точний, організований опис поточного стану застосунку, тоді як ребра суворо позначають дії, ініційовані користувачем, або цілеспрямовані переходи. Цей фундаментальний перехід від неструктурованого контенту до структурованого представлення забезпечує набагато повніше та функціональніше розуміння справжньої поведінки застосунку. Ця вдосконалена модель гарантує значну корисність для складних подальших завдань, зокрема, в автоматизованому тестуванні та комплексному аналізі поведінки.

Ключові слова: вебзастосунки; графи знань; парсинг вебзастосунків; видобуток даних; моделювання вебзастосунків.

ВСТУП

Вебзастосунки потребують представлення великого обсягу даних для таких завдань, як автоматизоване тестування, аналіз поведінки користувачів та функціональна перевірка. Традиційні вебпарсери працюють за структурованим, але спрощеним алгоритмом (рис. 1). Хоча даний підхід ефективно працює для статичних вебзастосунків, він не здатний обробляти динамічні додатки, де значна частина контенту недоступна через прості переходи за гіперпосиланнями. Сучасні вебзастосунки часто мають структуровані користувацькі потоки, які включають взаємодії за межами гіперпосилань. Наприклад, на типовому сайті перехід на сторінку оформлення замовлення може

вимагати кількох дій: пошук товару, додавання його до кошика, введення адреси доставки та лише потім перехід до оформлення.

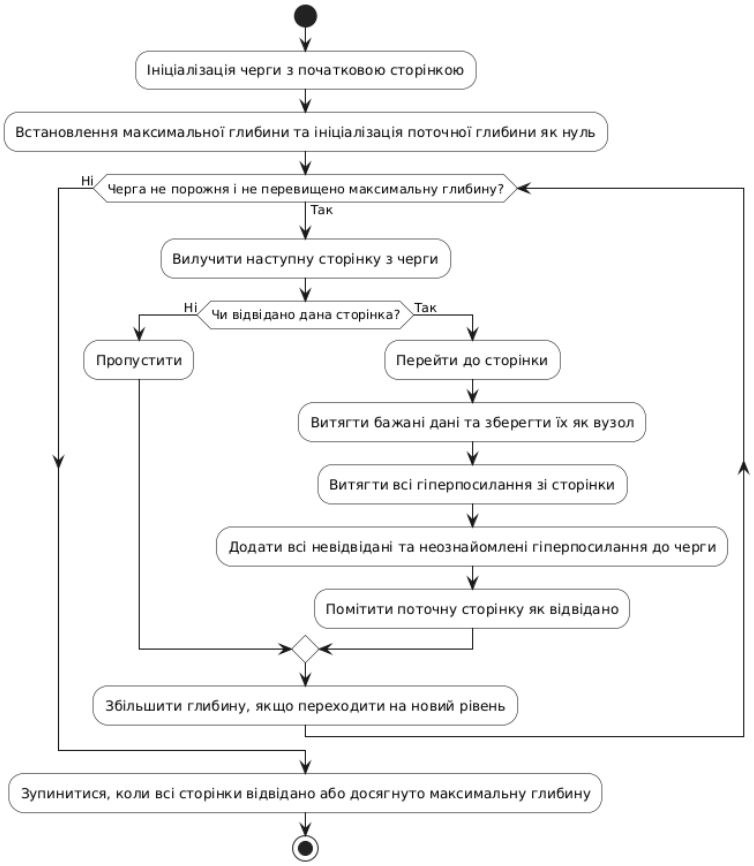

Рис. 1. Структурна UML-діаграма алгоритму роботи традиційного вебпарсера

Крім того, багато вебзастосунків демонструють варіативність на одному й тому ж кінцевому пристрої залежно від контексту користувача. Наприклад, сторінка оформлення замовлення може відображати "Готово до покупки" для одного користувача та "Товар не може бути доставлений за вашою адресою" для іншого, залежно від введеної адреси доставки.

У даній статті запропоноване рішення вирішить ці проблеми, представляючи кожен унікальний стан вебзастосунку як вузол, а ребра визначаються конкретними діями, виконаними в додатку. Даний метод захоплює повну складність користувацьких потоків, що дозволяє отримати більш точне та інтерпретоване представлення знань про вебзастосунки.

Аналіз останніх досліджень і публікацій. Ранні вебкраулери, такі як WebCrawler [1], були розроблені для вимірювання масштабів глобальної мережі шляхом збору базового HTML зі статичних вебресурсів [2]. По мірі розширення інтернету з’явилися інструменти, такі як World Wide Web Worm, які стали першими пошуковими системами, що використовували вебкраулери для індексації вебконтенту [3]. Водночас ранні системи мали суттєві обмеження, оскільки працювали виключно зі статичним контентом. Динамічні вебсторінки, побудовані на технологіях JavaScript та AJAX, у той період ще не отримали широкого розповсюдження, що значно звужувало функціональні можливості таких інструментів.

Поява динамічного контенту значно ускладнила процес вебскрапінгу для традиційних парсерів. Були розроблені фреймворки, такі як Beautiful Soup (2004), для полегшення добування структурованих даних зі складних вебсторінок. Хоча ці інструменти ефективні для аналізу статичного HTML-контенту, вони обмежені у можливостях обробки динамічних елементів, що функціонують за допомогою JavaScript, а також взаємодії з подіями, ініційованими користувачем. У міру того, як сучасні вебзастосунки почали активно використовувати динамічне завантаження контенту та клієнтські інтеракції, виникла потреба у більш просунутих методах для точного захоплення даної поведінки.

Для вирішення даних обмежень розроблено візуальні інструменти вебскрапінгу, такі як Apify Crawler [5], які пропонують зручний інтерфейс для автоматизації витягнення даних зі статичних та динамічних вебсайтів. Дані інструменти імітують поведінку користувача, таку як кліки та відправлення форм, для захоплення даних. Однак інструменти, подібні до Apify Crawler, не мають можливостей самостійного дослідження та не можуть автономно навігувати складними вебзастосунками. В результаті вони не можуть захопити повний спектр переходів станів та інтеракцій користувача, що є важливим для моделювання сучасних динамічних вебзастосунків.

Більш сучасні підходи зосереджуються на поєднанні динамічного аналізу з техніками краулінгу, заснованими на подіях. Наприклад, Common Crawl [6] використовує динамічний аналіз для підключення до JavaScript API та виявлення мережевих подій, динамічно згенерованих URL-адрес та відправлення форм користувачами. Використовуючи граф навігації, Common Crawl може досліджувати на 86% більше поверхні вебзастосунку порівняно з традиційними підходами. Аналогічно, CETR [7] використовує абстракцію стану для генерації графу потоків станів для AJAX-додатків, що може бути використано для автоматизації тестування динамічних користувацьких потоків.

Системи, засновані на графах знань, такі як Scrapy Distributed System [8-11], були запропоновані для краулінгу семантично орієнтованих вебресурсів та представлення даних у структурованих форматах. Однак дані підходи зазвичай обмежені вебданими, поданими у форматі каркасу опису ресурсів (Resource Description Framework), і не враховують динамічних, користувацько-ініційованих інтеракцій, що є характерною ознакою сучасних вебзастосунків.

Запропоноване рішення базується на основі даних існуючих інструментів, представляючи вебзастосунки у вигляді графу знань. У даному підході кожен вузол представляє унікальний стан додатку, а ребра представляють дії користувача, що призводять до переходів між станами. Таке структуроване представлення дозволяє ефективніше захоплювати динамічні робочі процеси та переходи станів, що робить систему ідеальною для таких завдань, як автоматизоване тестування та обробка помилок у складних вебсередовищах.

Метою статті є розроблення концептуального підходу до моделювання вебзастосунків у форматі графів знань, який забезпечує формалізоване представлення динамічної поведінки, станів і переходів сучасних вебсистем. Запропонований підхід спрямований на подолання обмежень традиційних методів вебсканування та парсингу шляхом переходу від неструктурованого збору контенту до структурованого відображення логіки взаємодій користувача з інтерфейсом. Така модель дає змогу більш точно описувати функціональні зв’язки між станами вебзастосунку, підвищувати ефективність автоматизованого тестування, виявлення помилок і поведінкового аналізу складних інтерактивних вебсередовищ.

ВИКЛАД ОСНОВНОГО МАТЕРІАЛУ ДОСЛІДЖЕННЯ

Основні компоненти системи розроблені для захоплення динамічної поведінки вебзастосунків, включаючи взаємодії користувачів та переходи станів. Система складається з трьох основних компонентів: модуля інтерпретації функціональності (Functionality Inferring Module), виконавця дій (Action Executor) та моделі заохочення/штрафи (Reward/Penalty Model). Кожен компонент взаємодіє з іншими для побудови інтерпретованого, дієвого графу вебзастосунку (рис. 2).

Поняття стану. У даній системі стан означає конфігурацію вебзастосунку в певний момент часу, характеризовану його візуальними та структурними властивостями. Кожен стан відображає результати взаємодій користувачів та змін у вебзастосунку, надаючи детальний знімок як його користувацького інтерфейсу, так і базової функціональності [12].

Стан визначається наступними ключовими компонентами:

- знімок екрана (візуальне зображення інтерфейсу вебзастосунку в певний момент, яке служить посиланням для графічного представлення, як його бачить користувач);
- вихідний код сторінки (HTML та структура Document Object Model (DOM), що складають вебсторінку. Це включає критичні елементи, такі як форми, кнопки та інтерактивні компоненти, які визначають макет та доступні функції);
- метадані (додаткові дані, пов'язані з поточною вебсесією, включаючи HTTP-заголовки, куки та змінні, специфічні для сесії. Ці метадані надають додатковий контекст щодо стану додатку, відображаючи умови, такі як автентифікація користувача, збереження сесії або динамічні коригування контенту).

Переходи станів відбуваються, коли користувачі взаємодіють з додатком, наприклад, через навігацію, відправлення форм або клацання кнопок. Ці переходи, захоплені як ребра в графі, формують зв'язки між станами та керують представленням знань [13].

У сучасних вебзастосунках кілька станів можуть відповідати одній URL-адресі, але можуть відрізнятися через фактори, специфічні для сесії, або динамічно рендерний контент. Наприклад, сторінка оформлення замовлення може представляти різні стани в залежності від того, чи були товари додані до кошика або чи доступні варіанти доставки для місцезнаходження користувача. Ці варіації захоплюються через комбінацію структурних даних та метаданих.

Поняття дії. У контексті даної системи Дія означає операцію або подію, ініційовану користувачем, яка переводить вебзастосунок з одного стану в інший. Дії представляють точки взаємодії між користувачем та вебзастосунком, такі як клацання кнопки, відправлення форми, перехід на нову сторінку або запуск AJAX-запиту. Ці дії є фундаментальними для здатності системи досліджувати та виводити функціональності всередині вебзастосунку [14].

Дії захоплюються та представляються як ребра в графі знань, де кожне ребро з'єднує два вузли (стани) та позначає перехід, викликаний певною взаємодією. Мета системи полягає не лише у фіксації дії, що зумовлюють переходи між станами, але й у визначенні їхньої важливості для роботи вебзастосунку та впорядкуванні за рівнем значущості.

Щоб зрозуміти роль дії у процесі моделювання поведінки вебзастосунку, варто виділити її основні характеристики:

- тип дії. Дії можуть сильно відрізнятися, від простої навігації (переходу за гіперпосиланням) до складних взаємодій (заповнення та відправлення форми). Ці дії

розрізняються за типами взаємодії – такими як клацання, відправлення форм, введення з клавіатури або динамічні тригери подій;

• контекст дії. Кожна дія пов'язана з певним елементом у структурі DOM, таким як кнопка, посилання або поле форми. Контекст включає метадані, такі як атрибути елемента (наприклад, ID, клас) та його розташування в ієрархії сторінки. Цей контекст допомагає системі зрозуміти, як дія пов'язана зі структурою вебзастосунку;

• вплив на стан. Дії значущі лише тоді, коли вони призводять до зміни стану, тобто переводять вебзастосунок з одного стану в інший. Модуль виведення функціональності аналізує вплив кожної дії на стан, забезпечуючи захоплення лише значущих переходів. Наприклад, відправлення форми може перевести користувача з сторінки входу на панель управління, тоді як клацання на неінтерактивний елемент не призведе до зміни стану;

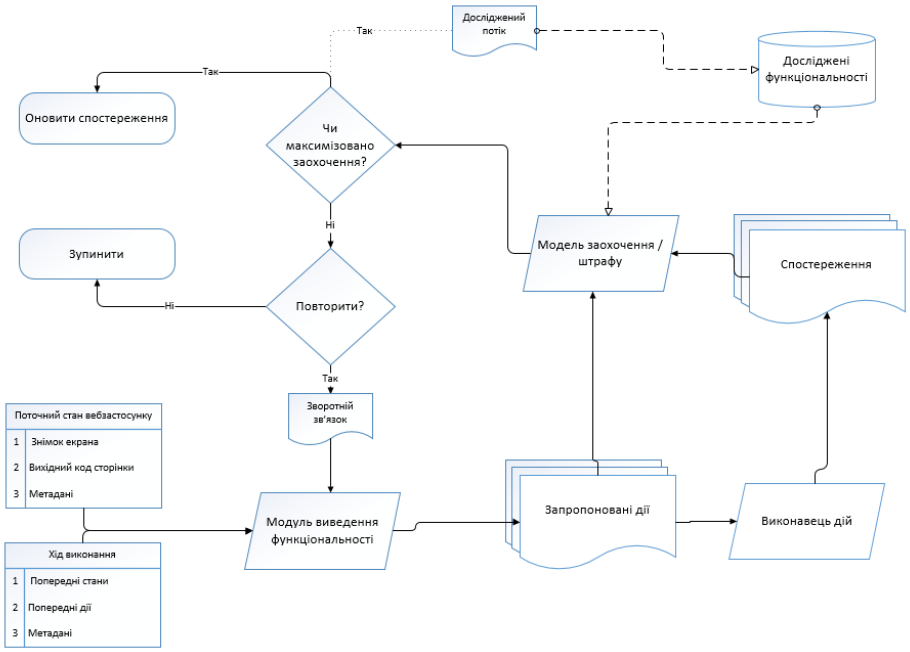

Рис. 2. Архітектура системи побудови графа знань вебзастосунку

• пріоритет дії. Не всі дії рівнозначно сприяють дослідженню функціональності додатку. Система надає перевагу діям, які ведуть до нових або недосліджених станів. Дії, які призводять до тривіальних або надмірних переходів (наприклад, клацання правою кнопкою миші або наведення на елемент без значущої зміни), знижуються в пріоритеті Модулем переранжування, забезпечуючи ефективність процесу дослідження.

Модуль виведення функціональності відповідає за аналіз поточного стану вебзастосунку та прогнозування потенційних дій, здатних перевести систему у нові стани. У процесі роботи цей модуль інтегрує дані, отримані внаслідок поточного спостереження, історії взаємодій за схемою "стан-дія", а також інформацію про вже досліджений функціонал системи. На основі цього масиву даних здійснюється формування та впорядкування (ранжування) переліку потенційних дій для подальшого випробування або аналізу [15-16].

Модуль виведення функціональності складається з чотирьох ключових компонентів (рис. 3):

• агент міркування. Даний агент обробляє поточну спостереження – вихідний код сторінки, знімок екрана та метадані – разом із записом раніше дослідженого

функціоналу. Його основна задача – синтезувати низку запитів до бази даних, щоб точно визначити, які саме можливості системи вже були випробувані, та які дії є потенційно можливими, виходячи з поточного стану та історії минулих взаємодій.

Як результат, агент міркування генерує перелік можливих дій, які можна виконати. Це забезпечує ретельне та систематичне дослідження всіх аспектів функціоналу вебзастосунку на основі як поточного, так і попередніх станів системи.

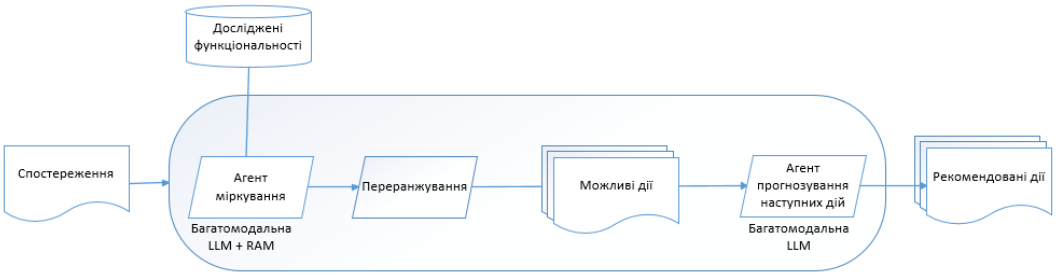

Рис. 3. Схема представлення модуля виведення функціональності

- модуль переранжування. Після того, як агент міркування згенерує список можливих дій, модуль переранжування оцінює дані дії та сортує їх за метриками, такими як ентропія та очікувана винагорода. Мета – визначити пріоритети діям, які найімовірніше розкриють нові функціональності або призведуть до значущих переходів станів, зменшити пріоритети тривіальній або надмірній взаємодії (наприклад, нефункціональні дії, такі як клацання правою кнопкою миші на елемент). Цей динамічний процес переранжування гарантує, що дослідження додатку залишається зосередженим на виявленні значущих користувацьких потоків та взаємодій, в кінцевому підсумку максимізуючи ефективність та ефективність системи в навігації складними вебзастосунками;
- агент прогнозування наступних дій використовує налаштовану мультимодальну LLM, яка уточнює список дій. Даний агент вибирає дії з найвищим пріоритетом зі списку можливих дій та прогнозує найкращі наступні кроки. Цей агент поєднує цінні відомості як з переранжованого списку, так і з розуміння системи вебзастосунку, щоб вибрати дії, які максимізують загальну цінність системи.

Модуль виведення функціональності працює через зворотний зв'язок. Після виконання кожної дії модуль оновлює своє розуміння поведінки вебзастосунку, включаючи нові спостережувані стани та дії. Даний зворотний зв'язок гарантує, що система постійно покращує свої прогнози та фокусується на виявленні найважливіших функціональностей.

Виконавець дій відповідає за виконання дій або послідовностей дій у вебзастосунку на основі даних від модуля переранжування та агента прогнозування наступних дій. Він виконує операції взаємодії, такі як клацання, відправлення форм та складні багатоступеневі операції [17].

- виконання дій. Виконавець застосовує вибрані дії, які можуть включати окремі взаємодії або багатоступеневі. Він обробляє різні типи дій, включаючи взаємодії з інтерфейсом користувача, навігаційні переходи та тригери подій, таких як JavaScript;
- перевірка стану. Після виконання дій виконавець перевіряє, чи призвела дія до значущої зміни стану, захоплюючи оновлений вихідний код сторінки, знімки екрана та метадані. Ця перевірка критична для оновлення графу знань;

ISSN 2663 - 4023

- обробка помилок та відновлення стану. У разі невдалих дій через неправильні входи або необроблені випадки виконавець реєструє помилку та повторює спробу або виконує дії відновлення, щоб повернути додаток до стабільного стану.

Модель заохочення/штрафи кількісно оцінює прогрес системи у дослідженні значущих функціональностей у вебзастосунку. Після того, як виконавець дій виконує дію та повертає новий стан, ця модель оцінює результат, призначаючи бал від -1 до +1, що вказує на цінність дії. Позитивні бали відображають значний прогрес, тоді як негативні бали вказують на тривіальні або надмірні дії.

- заохочення за значний прогрес. Дії, які призводять до нових переходів станів або відкриття недосліджених функціональностей, отримують позитивні заохочення (ближче до +1). Наприклад, навігація від головної сторінки до списку продуктів або доступ до процесу оформлення замовлення є діями, які приносять високу винагороду, оскільки вони розкривають критичні поведінки додатку;
- штраф за надмірні дії. Коли дії призводять до тривіальних переходів (наприклад, досягнення листового вузла, де неможливо виконати подальші значущі дії), модель призначає штрафи (ближче до -1). Це запобігає системі застрягти в непродуктивних станах, таких як сторінка "Дякуємо" після завершення покупки);
- зупинка дослідження. Якщо жодні дії не приносять позитивної винагороди, система припиняє дослідження за цим шляхом. Це гарантує, що ресурси не витрачаються на тупикові шляхи, а дослідження зосереджене на виявленні цінних переходів;

Для оцінки ефективності запропонованого підходу проведено експерименти на реальному вебсайті електронної комерції Rozetka.ua, який спеціалізується на продажу різних товарів та послуг. Даний вебсайт включає широкий спектр динамічних та інтерактивних компонентів, таких як пошук продуктів, фільтрація, управління кошиком та автентифікація користувачів, що робить його найкращим вибором для тестування обмежень традиційних парсерів та можливостей нашого рішення.

Для порівняння реалізовано традиційний парсер за допомогою фреймворку Scrapy, який часто використовується для вебсканування. Запропонований парсер на основі Scrapy слідує базовому підходу пошуку в глибину для сканування вебсторінок, видобутку гіперпосилань та збору статичного контенту сторінок. Зокрема, парсер налаштовано з наступними параметрами:

- максимальна глибина сканування – 3 рівні;
- переслідування перенаправлень – увімкнено;
- одночасні запити – 8;
- чергування User-agent – реалізовано для імітації різних браузерів.

Для подолання обмежень традиційних парсерів побудовано орієнтований граф, який моделює вебзастосунок Rozetka.ua. Граф був побудований з використанням наступних гіперпараметрів:

- мінімальна винагорода – 0. Мінімальний поріг для переходів на основі заохочення, використовуваний для усунення низькопріоритетних або надлишкових дій;
- максимальна кількість гілок листя – 999. Максимальна кількість гілок, які може мати листовий вузол перед видаленням;
- максимальна кількість послідовних дій – 5. Максимальна кількість послідовних дій, дозволених в межах одного стану перед примусовим переходом;
- максимальна кількість повторних спроб – 3. Максимальна кількість спроб для кожної дії перед класифікацією її як невдалої взаємодії.

Також порівняно продуктивність традиційного парсера на основі Scrapy та запропонованого рішення за допомогою наступних ключових метрик:

- кількість станів: кількість унікальних станів, що були виявлені та зафіксовані в процесі роботи парсера. Для традиційних парсерів унікальний стан зазвичай ідентифікується унікальним URL у межах домену. Вище покриття станів краще, оскільки відображає більш повне дослідження вебзастосунку, включаючи всі ключові функціональності та динамічні стани;
- кількість ребер: загальна кількість ребер (взаємодій) між станами. В оптимальному випадку, це значення близьке до n − 1, де n — загальна кількість станів. Це вказує на мінімальні відволікання між потоками, без непов'язаних або надмірних переходів, що свідчить про те, що кожен перехід значуще сприяє досліджуваній функціональності;
- відновлення після збоїв: відношення дій, які не відбулися при першій спробі, але успішно виконані в межах максимально дозволених повторних спроб. Вища величина вказує на більшу стійкість, оскільки система може відновлюватися після збоїв та досліджувати альтернативні шляхи або повторювати дії успішно;
- час виконання: загальний час, витрачений кожним методом на завершення сканування. У даній метриці менші значення є перевагою, оскільки швидше завершення означає більш ефективне дослідження;
- щільність графу: дана метрика вимірює відношення фактичної кількості ребер до загальної можливої кількості ребер у графі. Нижча щільність є перевагою, оскільки це вказує на те, що граф не перевантажений несуттєвими (або беззмістовними) зв'язками, що свідчить про більш структуровану та чітку взаємодію між станами;
- довжина найкоротшого шляху: середня довжина найкоротшого шляху між будь-якими двома вузлами в графі, яка вимірює загальну сполученість. Більша довжина найкоротшого шляху може вказувати на більшу кількість унікальних станів та глибше дослідження вебзастосунку. Для традиційних парсерів довжина шляху може бути коротшою через меншу кількість виявлених унікальних станів, тоді як у нашому підході вона, ймовірно, довша через більш широке покриття станів та складні взаємодії;

посередницька центральність: дана метрика вимірює важливість вузлів у з'єднанні різних частин графу. Вища величина свідчить про те, що певні стани (вузли) слугують ключовими перехрестями (або хабами) у потоках вебзастосунку. Це може бути корисним для визначення критичних сторінок, таких як сторінки входу або оформлення замовлення, які відіграють значну роль у навігації застосунком. Вища центральність посередництва зазвичай бажана для визначення ключових точок взаємодії в користувацькому шляху.

РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

Проведено експерименти на вебзастосунку Rozetka.ua, порівнюючи традиційний парсер на основі Scrapy з запропонованим рішенням. Традиційний парсер захопив базові статичні стани та ребра, але не зміг впоратися з динамічним контентом та не захопив поведінки користувачів, ініційованих формами або завантаженням контенту на основі AJAX. На відміну від цього, запропоноване рішення ефективно моделювало динамічні переходи станів, захопивши значно більше станів та ребер, хоча і з довшим часом виконання (таблиця 1).

ISSN 2663 - 4023

Таблиця 1

Порівняння традиційного парсера та запропонованого рішення за ключовими метриками

Метрика	Традиційний парсер	Запропоноване рішення
Складність станів (кількість станів)	24	**95**
Складність зв'язків (кількість ребер)	86	**94**
Рівень відновлення після збоїв	–	**0.72**
Час виконання (секунди)	**300**	5500
Щільність графу	0.72	**0.15**
Довжина найкоротшого шляху	2.1	**6.4**
Середня посередницька центральність	**0.59**	0.02

Запропоноване рішення досягло значно кращого покриття станів та складності ребер, зафіксувавши значно більшу кількість взаємодій в порівнянні з традиційними парсерами (рис. 4). Воно також проявило себе краще у виявленні динамічних поведінок із надійним відновленням після збоїв, чого традиційний парсер не міг забезпечити. Зокрема, середня посередницька центральність у запропонованому підході нижча, ніж у традиційних парсерів. Це пояснюється тим, що запропонована модель ідентифікує більшу кількіть унікальних станів, кожен з яких представляє окремі переходи станів, ініційовані взаємодіями користувачів та динамічним контентом. Ймовірність того, що ці унікальні стани слугуватимуть проміжними вузлами в різних потоках, менша, що й зменшує їхню загальну центральність у графі. На відміну від цього, традиційні парсери часто базуються на гіперпосиланнях, що призводить до більшої кількості спільних або повторно використовуваних станів у різних потоках. Це збільшує ймовірність того, що ці стани слугуватимуть мостами (ключовими перехрестями). У результаті, хоча традиційні парсери створюють більш централізовані графи, запропонований метод призводить до більш різноманітної та децентралізованої структури з меншою кількістю критичних перехресть.

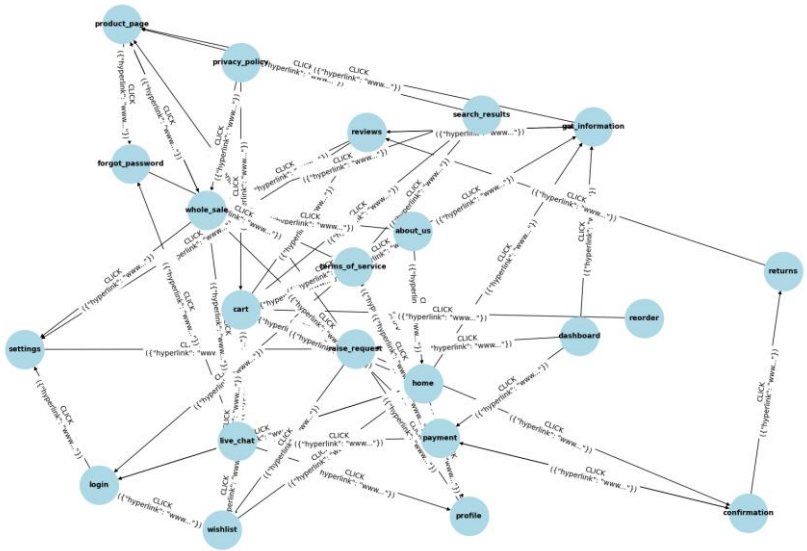

Рис. 4. Спрощений та неповний граф, згенерований традиційними парсерами, який демонструє обмежену взаємодію з користувачем та лише базові переходи станів.

ВИСНОВКИ ТА ПЕРСПЕКТИВИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

Одним з значних застосувань деталізованого графового представлення, створеного даним рішенням, є автоматизоване тестування. Кожен шлях від кореня до листа у графі відповідає унікальному користувацькому потоку або послідовності взаємодій у вебзастосунку (рис. 5). Оскільки кожна функціональність у застосунку відображена як окремий шлях, вдалося процедурно генерувати тестові випадки, проходячи ці шляхи від кореня до листа.

В цілому, запропоноване рішення згенерувало 45 унікальних тестових випадків, охоплюючи широкий спектр користувацьких взаємодій, таких як вхід у систему, пошук продуктів, додавання товарів до кошика та завершення процесу оформлення замовлення. Автоматизуючи обхід даних шляхів, тестові випадки можуть бути програмно згенеровані, забезпечуючи повне покриття користувацьких взаємодій, включаючи крайні випадки та складні робочі процеси, які можуть бути важкими для ручного тестування.

Дана процедурна генерація тестів гарантує, що вся функціональність застосунку буде протестована. Динамічні стани та поведінки, охоплені графом, надають детальну інформацію про кожну можливу користувацьку взаємодію. Як результат, система може легко виявляти потенційні помилки, проблеми зручності або проблеми продуктивності, систематично досліджуючи кожен унікальний шлях у застосунку.

Перспективи подальших досліджень полягають у розширенні представленої моделі шляхом інтеграції алгоритмів машинного навчання для автоматичного виявлення аномалій у поведінкових шаблонах користувачів та прогнозування можливих збоїв у функціонуванні вебзастосунків. Доцільним напрямом розвитку також є побудова онтологічних моделей для опису семантичних зв'язків між елементами інтерфейсу, що дозволить підвищити точність навігації графом знань. Окрім того, подальші дослідження можуть бути спрямовані на створення інтерактивних інструментів візуалізації таких графів.

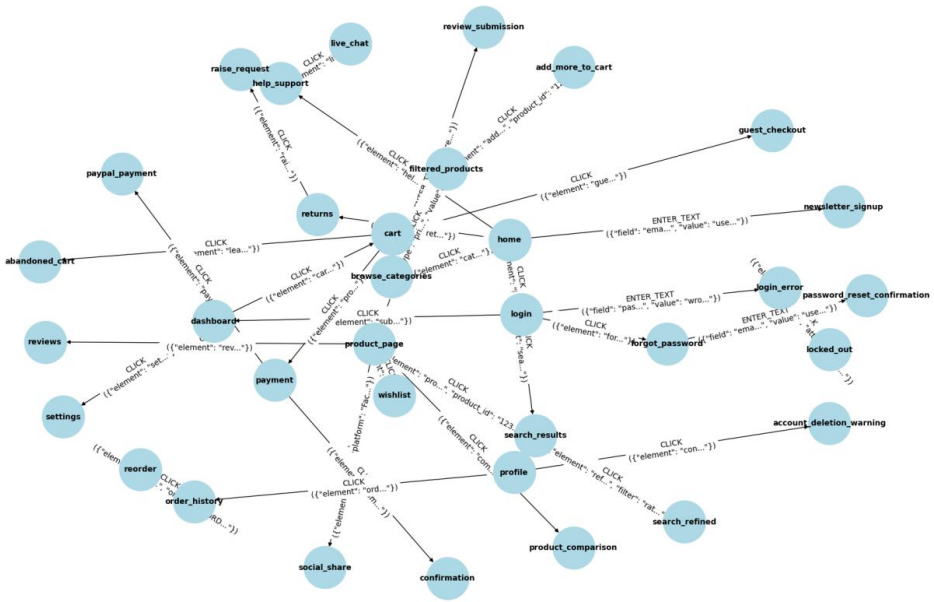

Рис. 5. Детальний граф, згенерований запропонованим рішенням, який ілюструє більш детальне представлення взаємодій користувачів із динамічними переходами станів, складними користувацькими потоками та ребрами, що базуються на діях, які пропонують більш глибоке розуміння поведінки вебзастосунку

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Mesbah, A., van Deursen, A., & Lenselink, S. (2012). Crawling AJAX-based Web Applications through Dynamic Analysis of User Interface State Changes. *ACM Transactions on the Web*, *6*(1), 1–30.
2. Pellegrino, G., Tschürtz, C., Bodden, E., & Rossow, C. (2015). jÄk: Using Dynamic Analysis to Crawl and Test Modern Web Applications. У *Proceedings of the 18th International Symposium on Research in Attacks, Intrusions and Defenses (RAID'15)* (pp. 295–316). Kyoto, Japan.
3. van Deursen, A., & Mesbah, A. (2015). Crawl-based analysis of web applications: Prospects and challenges. *Information & Software Technology*, *57*, 123–136.
4. Najork, M. (2009). *Web Crawler Architecture* (Microsoft Research Technical Report).
5. Heydon, A., & Najork, M. (1999). Mercator: A Scalable, Extensible Web Crawler. У *Proceedings of the 8th International World Wide Web Conference (WWW'99)*. Toronto, Canada.
6. Abu Kausar, M., Dhaka, V. S., & Singh, S. K. (2013). Web Crawler: A Review. *International Journal of Computer Applications*, *63*(2), 31–34.
7. Olston, C., & Najork, M. (2010). *Web Crawling Contents: A Survey* (Stanford University Technical Report).
8. Furche, T., Gottlob, G., Grasso, G., Guo, X., Orsi, G., & Schallhart, C. (2012). The Ontological Key: Automatically Understanding and Integrating Forms to Access the Deep Web. У *Proceedings of the VLDB Workshop on Deep Web*.
9. Kosala, R., & Blockeel, H. (2000). Web Mining Research: A Survey. *ACM SIGKDD Explorations*, *2*(1), 1–15.
10. Mukhopadhyay, D., Mukherjee, S., Ghosh, S., Kar, S., & Kim, Y.-C. (2011). *Architecture of a Scalable Dynamic Parallel WebCrawler with High-Speed Downloadable Capability for a Web Search Engine* (arXiv preprint).
11. Hassanshahi, B., Lee, H., Krishnan, P., & Güß, J. (2020). *Gelato: Feedback-driven and Guided Security Analysis of Client-side Web Applications* (arXiv preprint).
12. Cazzaro, L. (2025). Less is More: Boosting Coverage of Web Crawling through Adaptive Strategies. У *Proceedings of the International Conference on Dependable Systems and Networks (DSN'25)*.
13. Stafeev, A. (2024). *Evaluating the Effectiveness of Crawling Algorithms for Web Applications* (USENIX Summer School Report).
14. Mirtaheri, S. M., Dinçtürk, M. E., Hooshmand, S., & Bochmann, G. V. (2013). A Brief History of Web Crawlers. У *Proceedings of the 22nd International World Wide Web Conference (WWW'13)* (pp. 1–10).
15. Kolias, V., Anagnostopoulos, I., & Kayafas, E. (2014). *Exploratory Analysis of a Terabyte-Scale Web Corpus* (arXiv preprint).
16. Olston, C., & Najork, M. (2010). *Web Crawling Contents: Techniques and Challenges* (Stanford University Technical Report).
17. Kausar, M. A., Dhaka, V. S., & Singh, S. K. (2013). Web Crawler: Survey and Comparison. *International Journal of Computer Applications*, *63*(2), 35–40.

КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 3 (31), 2025
ISSN 2663 - 4023

Serhii Romanenko
Senior Lecturer, Department of Computer Science and Intelligent Technologies
Military Institute of Telecommunications and Information Technologies named after the Heroes of Kruty, Kyiv, Ukraine
ORCID: 0009-0004-0240-0777
serhii.romanenko@viti.edu.ua

Yevhenii Redziuk
Ph.D., Associate Professor, Head of the Department of Computer Science and Intelligent Technologies
Military Institute of Telecommunications and Information Technologies named after the Heroes of Kruty, Kyiv, Ukraine
ORCID: 0000-0001-5592-5121
yevhenii.redziuk@viti.edu.ua

Oleksandr Holub
Senior Lecturer, Department of Computer Science and Intelligent Technologies
Military Institute of Telecommunications and Information Technologies named after the Heroes of Kruty, Kyiv, Ukraine
ORCID: 0009-0006-1122-7667
oleksandr.golub@viti.edu.ua

KNOWLEDGE GRAPH-BASED MODELING OF WEB APPLICATIONS

Abstract. Traditional web crawling and parsing techniques, which rely on extracting and recursively following hyperlinks to map a web application, are increasingly inadequate for modern, dynamic web applications. This conventional method essentially creates a graph where nodes are merely chunks of unstructured data (the web page content) and edges represent simple page transitions. It fundamentally fails to capture the rich interactive behaviors and state changes that define contemporary applications. In response, a novel approach is proposed that models the web application not as a collection of links, but as a system of structured states. Under this paradigm, each node represents a precise, organized description of the application's current condition, while the edges strictly denote user-initiated actions or purposeful transitions. This fundamental shift from unstructured content to structured representation yields a far more complete and functional understanding of the application's true behavior. This enhanced model promises significant utility for advanced downstream tasks, particularly in automated testing and sophisticated behavior analysis.

Keywords: web applications; knowledge graphs; web application parsing; data extraction; web application modeling.

REFERENCES

1. Mesbah, A., van Deursen, A., & Lenselink, S. (2012). Crawling AJAX-based Web Applications through Dynamic Analysis of User Interface State Changes. ACM Transactions on the Web, 6(1), 1–30.
2. Pellegrino, G., Tschürtz, C., Bodden, E., & Rossow, C. (2015). jÄk: Using Dynamic Analysis to Crawl and Test Modern Web Applications. У Proceedings of the 18th International Symposium on Research in Attacks, Intrusions and Defenses (RAID'15) (pp. 295–316). Kyoto, Japan.
3. van Deursen, A., & Mesbah, A. (2015). Crawl-based analysis of web applications: Prospects and challenges. Information & Software Technology, 57, 123–136.
4. Najork, M. (2009). Web Crawler Architecture (Microsoft Research Technical Report).
5. Heydon, A., & Najork, M. (1999). Mercator: A Scalable, Extensible Web Crawler. У Proceedings of the 8th International World Wide Web Conference (WWW'99). Toronto, Canada.
6. Abu Kausar, M., Dhaka, V. S., & Singh, S. K. (2013). Web Crawler: A Review. International Journal of Computer Applications, 63(2), 31–34.
7. Olston, C., & Najork, M. (2010). Web Crawling Contents: A Survey (Stanford University Technical Report).

8. Furche, T., Gottlob, G., Grasso, G., Guo, X., Orsi, G., & Schallhart, C. (2012). The Ontological Key: Automatically Understanding and Integrating Forms to Access the Deep Web. У Proceedings of the VLDB Workshop on Deep Web.
9. Kosala, R., & Blockeel, H. (2000). Web Mining Research: A Survey. ACM SIGKDD Explorations, 2(1), 1–15.
10. Mukhopadhyay, D., Mukherjee, S., Ghosh, S., Kar, S., & Kim, Y.-C. (2011). Architecture of a Scalable Dynamic Parallel WebCrawler with High-Speed Downloadable Capability for a Web Search Engine (arXiv preprint).
11. Hassanshahi, B., Lee, H., Krishnan, P., & Güß, J. (2020). Gelato: Feedback-driven and Guided Security Analysis of Client-side Web Applications (arXiv preprint).
12. Cazzaro, L. (2025). Less is More: Boosting Coverage of Web Crawling through Adaptive Strategies. У Proceedings of the International Conference on Dependable Systems and Networks (DSN'25).
13. Stafeev, A. (2024). Evaluating the Effectiveness of Crawling Algorithms for Web Applications (USENIX Summer School Report).
14. Mirtaheri, S. M., Dinçtürk, M. E., Hooshmand, S., & Bochmann, G. V. (2013). A Brief History of Web Crawlers. У Proceedings of the 22nd International World Wide Web Conference (WWW'13) (pp. 1–10).
15. Kolias, V., Anagnostopoulos, I., & Kayafas, E. (2014). Exploratory Analysis of a Terabyte-Scale Web Corpus (arXiv preprint).
16. Olston, C., & Najork, M. (2010). Web Crawling Contents: Techniques and Challenges (Stanford University Technical Report).
17. Kausar, M. A., Dhaka, V. S., & Singh, S. K. (2013). Web Crawler: Survey and Comparison. International Journal of Computer Applications, 63(2), 35–40.