DOI: 10.32347/2412-9933.2024.59.155-164
УДК 005.8:005.94

Левицький Володимир Володимирович
Аспірант кафедри інформаційних технологій,
https://orcid.org/0000-0003-1829-488X
Київський національний університет будівництва і архітектури, Київ
Лопуга Олексій Миколайович
Аспірант кафедри інформаційних технологій,
https://orcid.org/0000-0001-6397-2710
Київський національний університет будівництва і архітектури, Київ

ГЕНЕРАЦІЯ ТЕСТОВИХ ДАНИХ ЗА ДОПОМОГОЮ ГЛИБОКОГО НАВЧАННЯ З ПІДКРІПЛЕННЯМ

Анотація. *Процес створення тестових даних для програмного забезпечення є одним із найскладніших і найтрудомісткіших етапів у циклі розроблення програмного забезпечення. Він вимагає значних ресурсів і зусиль, особливо коли йдеться про досягнення високого рівня покриття тестів. Тестування програмного забезпечення на основі пошуку (ТПЗОП) є підходом, що уможливлює автоматизувати цей процес, використовуючи метаевристичні алгоритми для генерування тестових даних. Метаевристичні алгоритми, такі як генетичні алгоритми або алгоритми імітації відпалу, працюють за принципом систематичного перебору можливих варіантів і вибору найбільш ефективних рішень на основі зворотного зв'язку від функції пристосованості. Цей процес нагадує навчання з підкріпленням, де агент отримує винагороду за правильні дії та покарання за неправильні, з метою оптимізації загальної продуктивності. Запропоновано новий підхід, який досліджує можливість застосування навчання з підкріпленням у ТПЗОП, з метою заміни традиційних, створених людиною, метаевристичних алгоритмів. Для цього розроблено платформу GunPowder, яка трансформує процес тестування програмного забезпечення (ТПЗ) в навчальне середовище. У цьому середовищі агент використовує метод Double Deep Q-Networks (DDQN), який включає в себе глибокі нейронні мережі для навчання оптимальної стратегії взаємодії з програмним забезпеченням, що тестується. Пропоноване дослідження включало серію емпіричних експериментів для оцінки ефективності запропонованого підходу. Отримані результати вказують на те, що агенти, навчені з використанням GunPowder, можуть ефективно вивчати і застосовувати метаевристичні алгоритми, досягаючи високого рівня покриття гілок у навчальних функціях. Зокрема, наші агенти досягли 100% покриття гілок у випадку навчальних функцій, що є свідченням їх здатності адаптуватися до різних тестових сценаріїв. Ці висновки підкреслюють потенціал використання глибоких нейронних мереж і навчання з підкріпленням у ТПЗОП, що може значно покращити якість і ефективність процесу тестування програмного забезпечення в майбутньому. Отже, робота відкриває нові можливості для інтеграції сучасних методів машинного навчання в автоматизоване тестування програмного забезпечення, що може привести до зменшення витрат часу і ресурсів на тестування, а також підвищення якості кінцевих продуктів.*

Ключові слова: ТПЗОП; DDQN; GunPowder; НП; Qtip

Вступ

Тестування програмного забезпечення на основі пошуку (ТПЗОП) показало свою ефективність в автоматичному генеруванні тестових даних, зокрема для структурного покриття. Для генерування тестових даних у програмному забезпеченні використовувалися різні метаевристичні техніки, включаючи алгоритми підйому на пагорб, імітаційного відпалу та еволюційні алгоритми [1].

Метаевристичні алгоритми вирішують проблеми по суті методом спроб і помилок. Алгоритми ітеративно оцінюють кандидатні рішення і генерують кращі рішення на основі зворотного зв'язку від функції придатності. Цей ітеративний процес можна розглядати як серію рішень, що приймаються на основі винагород (тобто покращення значень придатності). Якщо рішення покращує значення придатності, це можна вважати отриманням винагороди. Отже, ми можемо розглядати метаевристичний алгоритм як агентів, що слідують певній політиці для прийняття рішень.

У цьому контексті, проблемну ситуацію метаевристичного алгоритму можна описати як середовище для агента.

Навчання з підкріпленням (НП) є технікою машинного навчання, що прагне навчити оптимальній політиці керування для агентів, які взаємодіють з невідомим середовищем. У НП агенти намагаються і оцінюють дію, а потім приймають наступне рішення на основі спостережень за зворотним зв'язком від середовища. Гра у відеоігри є одним з найбільш відомих прикладів середовища для прийняття рішень з точки зору НП, і останні досягнення в глибокому навчанні та НП показали здатність навчати агентів на рівні людини для серії ігор Atari 2600 [2].

Аналогія між метаевристичним алгоритмом та НП спонукає нас до цікавого питання: чи можемо ми навчити агента вирішувати проблему ТПЗОП? Було запропоновано ідею реформулювання ТПЗОП як гри. Вважаючи ТПЗОП грою, ми можемо навчити політику керування, тобто новий метаевристичний алгоритм, для ТПЗОП. Більшість існуючих метаевристичних алгоритмів розробляються вручну: НП може дозволити нам автоматизувати розробку нового алгоритму через навчання НП агентів. Щоб відповісти на питання про доцільність використання НП для ТПЗОП, ми формулюємо проблему генерування тестових даних на основі пошуку як середовище НП, а потім навчаємо та тестуємо Double Deep Q-Networks (DDQN) на різних гілкових предикатах, що приймають числові вхідні дані. Зазначимо, що, наскільки нам відомо, це перша спроба використання НП агента в контексті ТПЗОП [3].

Мета статті

Завдання цієї роботи є такими:

1. Ознайомити із загальною, відкритою платформою, GunPowder, яка інструментує програмне забезпечення під тестуванням та обчислює значення придатності для даних критеріїв структурного тестування. Вона забезпечує платформу для застосування різних методів пошуку до ТПЗОП, включаючи НП алгоритми. GunPowder сумісний з широко використовуваною платформою НП OpenAI Gym.

2. Реформулювати ТПЗОП як середовище навчання з підкріпленням, а потім навчаємо та оцінюємо НП агента, який може замінити метаевристичні алгоритми, розроблені людиною. Ми представляємо невелике емпіричне дослідження, що оцінює ефективність нашого підходу.

Хоча результати не є одразу практичними, ми сподіваємось, що наше дослідження проллє світло на майбутнє використання навчання з підкріпленням та глибоких нейронних мереж у ТПЗОП. Почнемо з базових знань про навчання з підкріпленням.

Предметна область

Навчання з підкріпленням

Мета НП полягає в тому, щоб навчити оптимальних керуючих правил для агентів, які взаємодіють з невідомим середовищем, Е. Завдання агента полягає в тому, щоб вибрати послідовність дій, спостерігаючи за Е, яка максимізує накопичену винагороду за всі кроки часу.

Цей процес формально представлений як Марковський процес прийняття рішень (МППР), визначений кортежем (S, A, P, R), де S – це набір станів, а A – це набір дій. На кожному часовому кроці t агент виконує дію $a_t \in A$ на основі спостереження стану $s_t \in S$ і переходить до наступного стану $s_t + +1 \sim P(s_t, a_t)$, отримуючи зворотний зв'язок у вигляді скалярної винагороди, $r_t = R(s_t, a_t)$. $P(s_t, a_t)$ позначає ймовірність переходу від s_t до $s_t + +1$ через дію a_t . $R(s_t, a_t)$ є негайною винагородою після виконання дії a_t у стані s_t . Повернення на часовому кроці t визначається як $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$, де T – це часовий крок, коли "гра" закінчується. Майбутні винагороди дисконтуються за допомогою коефіцієнта $\gamma \in [0, 1]$. Мета НП – навчитися правилу π, який відображає стани в ймовірнісний розподіл дій. Правило визначає поведінку агента, а оптимальна політика π^* максимізує очікуване повернення з початку R_0.

Функція цінності дій Q – це очікуване повернення, починаючи зі стану s_t, після виконання дії a_t та подальшого слідування правилу π:

$$Q^{\pi}(s_t, a_t) = E[R_t | s_t, a_t].$$

Рівняння Беллмана виражає функцію цінності дій у рекурсивній формі:

$$Q^{\pi}(s_t, a_t) = E[r(s_t, a_t) + \gamma Q^{\pi}(s_{t+1}, \pi(s_{t+1}))].$$

Q-навчання – це алгоритм, що не використовує модель і не базується на правилах, які широко застосовуються. Він використовує апроксиматор функції для оцінки функції цінності дій, $Q(s, a; \theta) \approx Q(s, a)$. Глибокі Q-мережі (DQN) використовують велику нейронну мережу як апроксиматор функції, яка називається Q-мережею. Q-мережа навчається шляхом мінімізації втрат, що визначаються таким чином, де $y_t = r(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a; \theta)$:

$$L(\theta) = E[(y_t - Q(s_t, a_t, \theta))^2].$$

Використання апроксимації Q-функції може призвести до нестабільної поведінки, оскільки та сама мережа, яка генерує цільові Q-значення, також використовується для оновлення своїх Q-значень. Іншою проблемою є те, що послідовні вибірки даних сильно корельовані, що призводить до розбіжності Q-мереж під час навчання з них. Щоб полегшити ці

проблеми, Глибокі Q-мережі вводять дві основні зміни: повторний перегляд досвіду та окрему цільову мережу для обчислення y_t.

По-перше, DQN зберігає досвід агента e_t на кожному часовому кроці $e_t = (s_t, a_t, r_t, s_{t+1})$ у наборі даних $D_t = \{e_1, \dots, e_t\}$, який називається пам'яттю повторного перегляду. Потім Q-мережа оновлюється випадковими вибірками з D, що називається технікою повторного перегляду досвіду. Ця техніка допомагає уникнути сильного корелювання між послідовними вибірками даних.

По-друге, DQN використовує окрему цільову мережу для генерації цільових значень y_i у функції втрат. Ця цільова мережа оновлюється через фіксовані інтервали шляхом клонування параметрів Q-мережі, що приводить до більш стабільного навчання. Завдяки цим змінам DQN показав, що стабільне навчання за допомогою нейронної мережі є можливим. Велика перевага використання нейронної мережі полягає в тому, що вона дає змогу витягувати високорівневі ознаки з необроблених даних, які можуть бути безпосередньо використані для НП.

У цій статті ми використовуємо Double DQN (DDQN), яка є покращеною версією DQN. Проблема переоцінки полягає в навчанні нереалістично високих значень дій. Відомо, що Q-навчання має цю проблему в деяких випадках. DDQN вирішує цю проблему, використовуючи різні мережі для вибору та оцінки дії. Ціль, яку використовує DQN, визначається так:

$$y_t = r(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a;\ \theta).$$

Вона використовує ті самі ваги \(\theta\) для вибору дії та оцінки вибраної дії. Замість цього ціль DDQN визначається так:

$$y_t = r(s_t, a_t) + \gamma \max_a Q(s_{t+1}, \arg\max_a Q(s_{t+1}, a; \theta'); \theta),$$

де θ – це ваги цільової мережі; θ' – це ваги онлайн-мережі. Як і в DQN, цільова мережа періодично копіюється з онлайн-мережі. Зауважте, що ваги θ' використовуються для вибору дії, тоді як θ оцінює цінність цієї політики. У результаті DDQN уникає переоцінки в Q-навчанні [4].

GunPowder: Загальна структура для ТПЗОП

Генерація тестових даних на основі пошуку є динамічною технікою, що включає інструментування програм. Ми узагальнюємо загальні вимоги до інструментування та аналізу для генерування тестових даних таким чином:

1. Інструментування тестованого програмного забезпечення (ТПЗ) для вимірювання структурних критеріїв тестування.

2. Виконання ТПЗ з кандидатним вводом, згенерованим алгоритмами пошуку.

3. Обчислення значення функції пристосованості.

Загальна структура, що задовольняє ці вимоги, уможливить ефективно і результативно порівнювати різні методи пошуку, які можуть бути реалізовані на основі цієї структури. Хоча IGUANA надавала загальну структуру для мови C, вона більше не підтримується активно. Ми представляємо GunPowder, загальну структуру для генерування тестових даних на основі пошуку для структурного тестування для мови C, написану на Python [5].

Структура GunPowder складається з трьох частин: інструментування, виконання та оцінювання пристосованості. По-перше, GunPowder інструментує вихідний код ТПЗ для обчислення значень функцій пристосованості для певного критерію адекватності тесту: наразі GunPowder підтримує програми на C та покриття гілок. Інструментування виконується на рівні вихідного коду за допомогою Clang фронтенду з LLVM фрейморком. Порівняно з javacc, використовуваним в IGUANA, Clang забезпечує більш надійний аналіз та маніпулювання кодом на C.

По-друге, GunPowder виконує ТПЗ із заданим вводом. У випадку програм на C, GunPowder компілює ТПЗ як спільну бібліотеку, щоб уникнути необхідності створення додаткового драйверного коду. Ця бібліотека викликається безпосередньо через інтерфейс іноземних функцій Python; GunPowder реалізовано на Python. Коли інструментоване ТПЗ виконується, воно повертає трасу виконання. Траса передається до функції пристосованості, яка може бути будь-якою користувацькою функцією на Python, обчисленою з трасування виконання та структури керування ТПЗ.

GunPowder розроблено як розширюваний інструмент. Завдяки наданню аналізу структури керування та інструментування для нових мов програмування, ми можемо розширити GunPowder для мов, відмінних від C. Крім того, нові алгоритми пошуку для ТПЗОП можуть бути реалізовані та оцінені за допомогою інтерфейсу Python [6].

Функція Пристосованості

Наразі GunPowder підтримує стандартну функцію пристосованості для покриття гілок, яка складається з рівня наближення, A, та відстані до гілки, Δ. Значення пристосованості отримується за допомогою трасування виконання кандидатного рішення та інформації про залежність керування. Коли вузол ni у інструментованому ТПЗ виконується, запис трасування $t_i = (i, c_i(X), \Delta(X,\ c_i), \Delta(X,\ -c_i))$ зберігається у трасі $T(X) = \{t_i \mid n_i \in P\}$, де P – список виконаних вузлів; c_i – предикат для вузла n_i.

При інструментуванні ТПЗ GunPowder також аналізує його структури керування. На основі цього аналізу ми можемо отримати бажаний шлях виконання P_d для будь-якої цільової гілки. Порівнюючи трасу виконання T і бажаний шлях P_d, ми підраховуємо, скільки вузлів у P_d не входять до фактичного шляху виконання P. Кількість цих неперетнутих вузлів призначається рівню наближення A для вводу X. Одночасно, розбіжний вузол nc може бути ідентифікований порівнянням T і P_d . Значення $\Delta(\mathrm{X}, c_c)$ у записі трасування t_c призначається до відстані до гілки Δ для вводу X. Поєднання A та Δ (повертається як значення функції пристосованості.)

Виклад основного матеріалу

Генерація тестових даних за допомогою Q-навчання

Ми представляємо Qtip, техніку генерації тестових даних на основі Q-навчання. Qtip складається із середовища Qtip, яке перетворює ТПЗ на середовище навчання з підкріпленням, і агента Qtip, навченого у середовищі Qtip за допомогою Q-навчання. У цій статті ми використовуємо DDQN для навчання агента Qtip.

Формулювання ТПЗОП як задачі навчання з підкріпленням

ТПЗОП розглядає генерацію тестових даних як задачу оптимізації і використовує метаевристичні алгоритми для її розв'язання. Через ітеративні проби та помилки алгоритми намагаються знайти відповідне рішення, слідуючи зворотному зв'язку від функції пристосованості. Це можна розглядати як безперервний процес прийняття рішень. Алгоритм, який співвідноситься з агентом, робить дію, створюючи нове кандидатне рішення. Після цього він отримує винагороду, виражену як покращення значення функції пристосованості.

У цій структурі генерація тестових даних залишається задачею оптимізації, але метаевристичний алгоритм замінюється агентом, навченим за допомогою НП. Відповідно, задача ТПЗОП співвідноситься із середовищем і представляється як приклад MDP, який є формальним представленням задачі НП, визначеної кортежем (S, A, P, R).

Залежно від того, як ми визначаємо $s_t \in \mathrm{S}, a_t \in \mathrm{A}$) та $r_t \sim \mathrm{R}(s_t, a_t)$, продуктивність агента змінюватиметься. У цій статті пропонуємо такі формулювання.

Стан та дія

Більшість генерацій тестових даних на основі пошуку залежить від концепції рівня наближення A та відстані до гілки Δ. Функція пристосованості визначається так:

$$F(X) = A(X) + N(\Delta(X, \mathrm{c_c})),$$

де $\mathrm{N}: \mathrm{R} \to [0,1)$ є функцією нормалізації.

Інформація, що використовується метаевристичним алгоритмом, включає поточний рівень наближення та відстань до гілки. Аналогічно формулюємо стан у часовому кроці t у середовищі НП так:

$$s_t = \{\mathrm{A}(X_t), \mathrm{N}(\Delta(X_t, c_c))\},$$

де X_t – вектор вводу в часовому кроці t. Це формулювання означає, що агент передає вектор вводу до середовища ТПЗОП, після чого середовище оцінює це введення і повертає рівень наближення A та відстань до гілки Δ. Агент спостерігає ці відповіді і розглядає їх як поточний стан середовища.

На кожному часовому кроці t агент повинен надати кандидатне рішення, яке є вектором вводу для ТПЗ. Це відповідає дії з точки зору процесу прийняття рішень. Тому дія має привести до нового кандидатного рішення. У випадку підйому на пагорб нове рішення генерується шляхом модифікації поточного рішення, процесу, який описується як перехід до сусіднього рішення у ландшафті пошуку. Аналогічно визначаємо простір дій як набір можливих маніпуляцій для поточного вектору вводу X таким чином:

$$\mathrm{A} = \{a_i \,|\mathrm{i} \in [1, 2 \cdot |X|]\}$$

$$a_i = \begin{cases} x_i \leftarrow x_i + 1 & \text{якщо } i \text{ парне} \\ x_i \leftarrow x_i - 1 & \text{якщо } i \text{ непарне,} \end{cases}$$

де x_i є і-тим елементом вектора вводу X.

Оскільки ми визначили дві можливі маніпуляції для кожного елемента вектора вводу, розмір простору дій дорівнює подвоєній довжині вектора вводу. Як дослідження здійсненності обмежимо наше дослідження функціями лише з числовими вводами. Проте ми вважаємо, що формулювання на основі дій може бути згодом розширено для інших типів вводу. Наприклад, генерація динамічних структур даних і вказівників для програм на C була викликом у ТПЗОП, для якого запропоновано різні підходи. Формулювання на основі дій допомагає визначати різні дії для різних типів даних, включаючи виділення пам'яті або навіть виклики інших евристик пошуку [7].

Часткова спостережуваність

Одним з внутрішніх обмежень у наведеному вище формулюванні (тобто на основі пристосованості поточного кандидатного вводу) є те, що агент має лише часткове спостереження за траєкторією пошуку. Розглянемо алгоритм підйому на пагорб: алгоритм оснащений (простим) механізмом пам'яті, тобто змінною, яка зберігає значення пристосованості попереднього кандидатного рішення, для спрямування пошуку. Порівняно з цим у нашому формулюванні процесу прийняття рішень агент не має інформації про попередній стан. Хоча може бути можливим

навчитися мінімізувати функцію пристосованості в абсолютному (тобто поточна пристосованість має бути якомога меншою), а не відносному (тобто поточна пристосованість має бути меншою, ніж попередня) сенсі, ми припускаємо, що інформація про траєкторію пошуку може покращити продуктивність нашого агента НП.

Для оцінки цієї гіпотези ми представляємо альтернативне формулювання, яке включає попередні спостереження значень пристосованості як частину стану. Формально, спостереження $o^{(t)}$ у часовому кроці t стає

$$o^{(t)} = \left(a_t, \mathrm{A}(X_t), \Delta(X_t, c_c)\right), s_t = \left\{o^{(t)}, \ldots, o^{(\mathrm{t-m+1})}\right\},$$

де m – це розмір пам'яті, який називається розміром вікна; $a_t \in A$ – дія, здійснена на часовому кроці t. Ми навчаємо агентів з обома представленнями стану: одне з розміром вікна 200, а інше без вікна.

Винагорода

Останнім компонентом процесу прийняття рішень є винагорода. Метою НП є максимізація сукупної винагороди з початкового часового кроку:

$$R_t = \sum_{t=0}^{T} \gamma^t \; r_t,$$

де T – це часовий крок, коли гра завершується.

Винагороди мають бути спроєктовані таким чином, щоб змусити агента поводитися так, як очікується.

У ТПЗОП стандартною метою оптимізації для структурного тестування є мінімізація значення пристосованості, тобто $\mathrm{A} + \mathrm{N}(\Delta)$. Аналогічно, очікувана поведінка для нашого агента НП – це постійне зниження значення пристосованості: ми повинні винагороджувати агента, коли він знижує значення пристосованості. Відповідно, винагорода rtr_trt на кроці ttt визначається як величина зниження значення пристосованості:

$$r_t = \mathrm{F}(X_{t-1}) - \mathrm{F}(X_t),$$

де X_t – це вектор вводу, а $\mathrm{F(X)} = \mathrm{A(X)} + {+\mathrm{N}\left(\Delta(\mathrm{X}, c_c)\right)}$. Отже, агент отримує позитивну винагороду, коли знижує значення пристосованості.

Сумісність GunPowder з OpenAI Gym

OpenAI Gym [8] – це бібліотека з відкритим кодом, написана на Python, яка спрямована на підтримку розробки та порівняння різних алгоритмів НП, надаючи стандартизований набір середовищ для НП. Ми розробили GunPowder, щоб він був сумісний з OpenAI Gym, щоб відкрити ТПЗОП як майбутню тему досліджень для різних алгоритмів НП.

Клас Env в OpenAI Gym визначає стандартний інтерфейс для середовища НП. Наведемо необхідні методи Env:

- reset(self): Скидає стан середовища. Повертає спостереження.
- step(self, action): Виконує середовище на один часовий крок і повертає спостереження, винагороду, завершення, інформацію.
- render(self, mode='human', close=False): Візуально відображає один кадр середовища, якщо це необхідно.

Щоб створити нове середовище OpenAI Gym, ці методи повинні бути реалізовані. Оскільки ТПЗОП не має візуального представлення, ми реалізуємо лише reset і step.

У середовищі Qtip метод step виконує маніпуляцію, що відповідає заданій дії для поточного вектора вводу X. Після цього він виконує ТПЗ з модифікованим вводом і обчислює значення пристосованості, яке агент спостерігає і використовує як винагороду. У кожному епізоді агент може виконати певну кількість дій, що відповідає бюджету оцінки пристосованості в метаевристичних алгоритмах. Якщо агент використовує весь бюджет або покриває цільову гілку, епізод завершується.

І GunPowder, і наше середовище Qtip працюють на рівні функцій: вони інструментують цільову функцію ТПЗ та оцінюють пристосованість для конкретної гілки в цій функції. Отже, одна функція складає один екземпляр середовища Qtip. Коли починається новий епізод, середовище встановлює нову цільову гілку в цільовій функції, що реалізовано в методі reset.

Експеременталні установки

Питання перед дослідженням

Ми прагнемо відповісти на наступні дослідницькі питання за допомогою емпіричного дослідження.

Питання 1. Ефективність: Який середній рівень структурного покриття може досягти агент Qtip?

Питання 2. Невидимі структури: Чи може агент Qtip покрити функції, які не були видимі під час навчання?

Питання 3. Невидимі діапазони вхідних даних: Чи може агент Qtip покрити вхідні дані в діапазонах, які не були видимі під час навчання?

Мета пропонованого підходу – навчитися керуючій політиці, яка досягає структурного покриття. Ми відповідаємо на Питання 1, вимірюючи покриття гілок, досягнуте агентом Qtip. Через стохастичну природу алгоритму НП повторюємо 30 спроб і повідомляємо середнє покриття.

Щоб замінити метаевристичні алгоритми, агенти мають бути загалом ефективними, тобто вони мають бути здатні досягати покриття для невидимих довільних гілок. Крім того, агенти мають бути здатні навчитися політиці незалежно від розміру пошукового простору. Питання 2 розглядає проблему узагальнюваності, вимірюючи досягнуте покриття гілок для функцій, які не були видимі під час навчання. Питання 3 зосереджується на впливі діапазону вхідних даних на продуктивність.

Ми повторюємо 30 запусків для обох Питання 2 та Питання 3 для кожної гілки.

Навчання

Ми навчаємо Qtip за допомогою алгоритму DDQN, щоб покрити набір основних бінарних реляційних операторів: ==, !=, <, <=, > і >=, а також унарне логічне заперечення !. Наша навчальна програма містить серію «if-операторів», кожен з яких містить один з основних реляційних операторів. Усі гілки не вкладені, тому рівень підходу A завжди дорівнює нулю. Цільова функція приймає два цілі числа як аргументи, тобто |X| = 2 і |A| = 4. Діапазон значень обох вхідних змінних за замовчуванням встановлено від [-128, 128].

У кожному епізоді випадкова гілка вибирається як цільова, щоб уникнути перенавчання на конкретний тип гілки. Вхідні значення ініціалізуються випадковими числами, які не покривають цільову гілку. Агенту надається бюджет у 2,000 дій, і епізод закінчується, коли агент або покриває цільову умову, або вичерпує бюджет. Винагорода дисконтується з коефіцієнтом $\gamma = 0.99$ за кожен крок, а розмір буфера повторного перегляду встановлено на 5,000. Як зазначено в розділі «Формулювання ТПЗОП як задачі навчання з підкріпленням», ми використовуємо вікно станів розміром 200 для зберігання інформації про попередні стани.

Архітектура нейронної мережі

Рис. 1 ілюструє архітектуру нашої Q-мережі. Для апроксимації функції Q ми використовуємо три повнозв'язані шари з 16 вузлами. Окрім шару activation_4, усі шари активації використовують функцію активації Rectified Linear Unit (ReLU) [9].

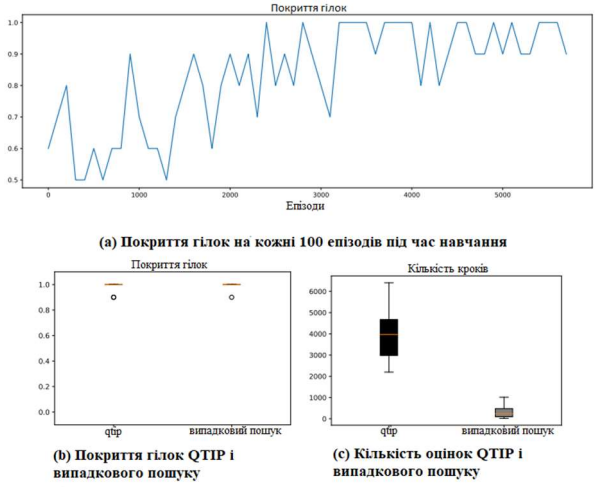


Рисунок 1 – Архітектура Q-мережі: a – агент Qtip навчається покривати більше гілок у функції тренування; b – зліва – Qtip, справа – випадковий пошук (обидва методи покривають всі гілки в більшості випадків); c – у порівнянні з випадковим пошуком, Qtip потребує більше бюджетів на оцінювання

Архітектура мережі має значний вплив на продуктивність агента. Наприклад, DQN використовує згорткові шари для витягання високорівневих ознак зі знімків екрана ігор Atari 2600 [10]. Однак, як початкове дослідження здійсненності, ми тримаємо архітектуру мережі якомога простішою: оптимізація архітектури мережі залишена для майбутніх досліджень.

Слідом за Тімоті та ін., ми оновлюємо цільову мережу з певною затримкою: $\theta' \to \tau\theta + (1 - |\tau)\theta'$ з $\tau \ll 1$, що називається "м'якими" оновленнями цілі. Однак онлайн Q-мережа оновлюється на кожному кроці. Ми використовуємо алгоритм оптимізації Adam [11] з розміром міні-пакету 32. Швидкість навчання lr, визначається спадом так:

$$\mathrm{lr}_i = \mathrm{lr}_{i-1} \times \frac{1}{(1+D\times i)},$$

де lr_0 дорівнює 10^{-7}, а коефіцієнт спаду D дорівнює 10^{-6}. Спад швидкості навчання допомагає швидкому зближенню оптимізації [11].

Суб'єкти

Як зазначено у цьому розділі, наша тренувальна функція містить гілки з базовими предикатами без будь-якої вкладеної структури. Після навчання ми використовували три невеликі функції для тестування продуктивності Qtip: ми застосовуємо навченого агента до цих невидимих тестових функцій і вимірюємо покриття гілок. Рис. 2 представляє вивчені функції. Усі функції приймають два цілі числа на вході: GCD і EXP є відомими алгоритмічними прикладами, тоді як функція Remainder була вибрана з бенчмарку IGUANA.

Наразі GunPowder підтримує лише програми на мові C для повністю автоматизованої інструментації. Однак ми реалізували всі тренувальні і тестувальні функції на Python: це через те, що накладні витрати міжпроцесної комунікації стали надзвичайно великими, оскільки Qtip потребує значно більшої кількості оцінок придатності, навіть у порівнянні з випадковим пошуком. У результаті ми вручну інструментували Python-реалізації тренувальної функції, а також тих, що згадані в рис. 2, і написали невеликий драйвер на Python між GunPowder і Qtip для уникнення накладних витрат. Незважаючи на цей обхідний шлях, будь ласка, зверніть увагу на таке: 1) GunPowder все ще повністю сумісний з OpenAI Gym для цільових програм на C; 2) GunPowder може легко розширюватися для інструментування цільових функцій на Python.

Результати

Ефективність

Поширеним способом моніторингу навчання НП є спостереження за зміною загальних винагород, отриманих у кожному епізоді. Однак у випадку з Qtip головною турботою є те, чи покривається цільова гілка, а не загальна винагорода в кожному епізоді.

Відповідно, для моніторингу покращень під час навчання ми вимірюємо кількість гілок, покритих агентом, кожні 100 епізодів. На рис. 3, *a* показано, що Qtip дійсно вивчає політику, яка призводить до покриття більшої кількості гілок. Навчання потребувало загалом 5700 епізодів і зайняло 5 год та 44 хв.

Таблиця 1 – ***Короткий опис архітектури мережі Q***

Шар (тип)	Вихідна форма	Кількість параметрів
вирівнювання_1 (вирівнювання)	600	0
щільний_1 (щільний)	16	9,616
активація_1 (активація)	16	0
щільний_2 (щільний)	16	272
активація_2 (активація)	16	0
щільний_3 (щільний)	16	272
активація_3 (активація)	16	0
щільний_4 (щільний)	4	68
активація_4 (активація)	4	0
Загальна кількість параметрів: 10,228		
Треновані параметри: 10,228		
Нетреновані параметри: 0		

Таблиця 2 – ***Короткий опис функцій предмета***

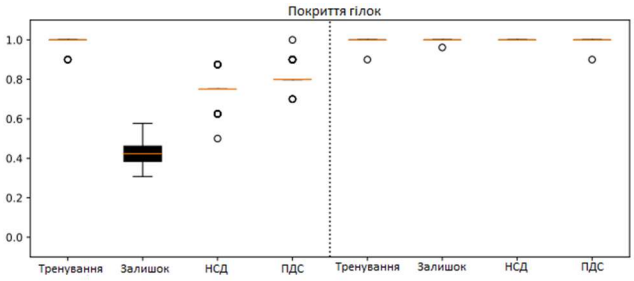
Назва	Кількість гілок	Мета
Тренування	10	тренування
Найбільший спільний дільник	8	тренування
Піднесення до ступеня	10	тренування
Залишок	26	тренування

Щоб виміряти ефективність нашого підходу, ми використали випадковий пошук як базовий рівень. Рис. 1, *b* показує покриття гілок функції тренування, досягнуте Qtip та випадковим пошуком. У більшості випадків обидва методи успішно покривають всі гілки у функції.

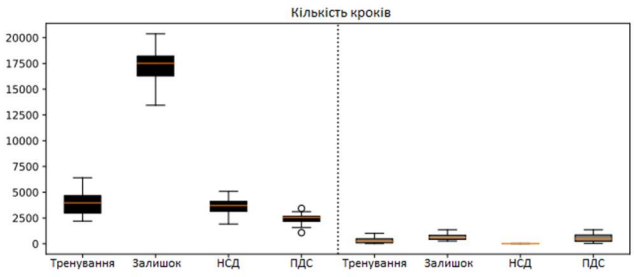
Однак для досягнення того ж рівня покриття гілок Qtip потребує більше бюджетів на оцінювання порівняно з випадковим пошуком, що видно на рис. 1, *c*. Ми підозрюємо, що як випадковий характер агента під час тренування, так і те, що агент змінює вектор вводу лише на одиницю, сприяють вищим витратам у Qtip.

Невидимі структури

Рис. 2, *a* показує коробкову діаграму покриття гілок, досягнутого навченим агентом Qtip на трьох тестових функціях, на основі 30 повторених запусків. Хоча Qtip ніколи не зустрічав ці функції під час тренування, він досягає в середньому 66.03% покриття гілок. Це свідчить про те, що Qtip засвоїв метаевристичну поведінку, яка працює для випадкових проблем. Рис. 3 підсумовує результати.



(a) Покриття гілок Qtip (зліва) і випадковий пошук (справа)



(b) Кількість оцінок Qtip (зліва) і випадковий пошук (справа)

Рисунок 2 – Моніторинг навчання Qtip: a – ліва частина показує покриття гілок Qtip, а права частина – покриття гілок випадкового пошуку (хоча Qtip може покривати деякі випадкові гілки, випадковий пошук є ефективнішим); b – ліва частина показує кількість оцінювань Qtip, а права частина – кількість оцінювань випадкового пошуку (як зазначено у Розділі «Ефективність», Qtip потребує більше бюджетів на оцінювання, ніж випадковий пошук)

У випадку функції "Залишок" Qtip не змогла покрити половину гілок. Ручна перевірка засвідчила, що всі ці непокриті гілки мають контрольну залежність від інших гілок (тобто вони вкладені). Оскільки ми навчаємо агентів Qtip на гілках без глибини, рівень підходу A залишається нульовим під час тренування. Як результат, Qtip не навчається використовувати інформацію про рівень підходу, яка розроблена для допомоги у вирішенні вкладених структур у ТЗП. Ми очікуємо, що більш ефективне тренування агентів Qtip у майбутньому, з більш різноманітним набором структур гілок, може дати кращі результати для цих гілок.

Таблиця 3 – ***Середнє покриття μ гілок та стандартне відхилення σ для Qtip та випадкового пошуку на основі 30 запусків: найвищий рівень покриття для кожної функції виділений жирним шрифтом***

Функція	Qtip (μ)	Qtip (σ)	Випадкові дані (μ)	Випадкові дані (σ)
Тренування	99.00	0.03	99.67	0.02
Залишок	42.69	0.06	99.87	0.01
Найбільший спільний дільник	73.75	0.09	100.00	0.00
Піднесення до ступеня	81.67	0.06	99.67	0.02
Загалом	74.28	0.21	99.80	0.01

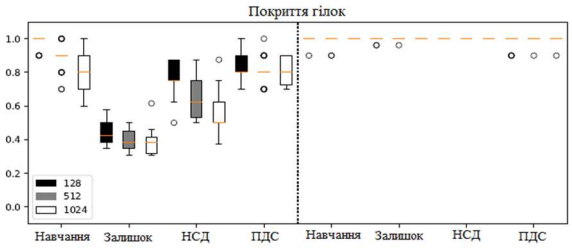
Невидимі діапазони вхідних даних

Qtip тренується в діапазоні вхідних даних від [- 128, 128]. Ми тестуємо Qtip з різними діапазонами вхідних даних, щоб перевірити, чи дійсно Qtip навчається загальній метаевристичній поведінці, а не перенавчається на меншому діапазоні вхідних даних. Табл. 4 містить підсумок розмірів альтернативних тестових просторових вхідних даних та відповідну кількість дозволених кроків.

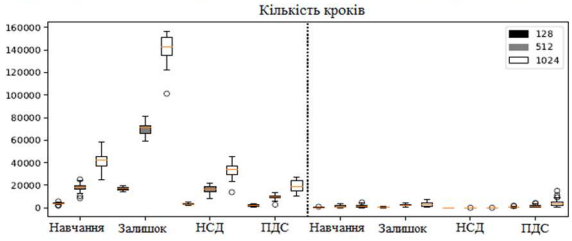
Таблиця 4 – ***Підсумок розмірів простору вхідних даних та відповідних бюджетів на оцінювання***

Діапазон вхідних даних	Кількість оцінок
[-128, 128]	1000
[-512, 512]	4000
[-1024, 1024]	8000

На рис. 3, *а* показано зміну досягнутого покриття гілок щодо різних розмірів простору вхідних даних. Незалежно від діапазону вхідних даних, середнє покриття гілок залишається стабільним.



(а) Покриття гілок Qtip (зліва) та випадковий пошук (справа)



(б) Кількість оцінок Qtip (зліва) та випадковий пошук (справа)

Рисунок 3 – Ілюстрація підсумку результатів: а – ліва половина – це покриття гілок Qtip, а права половина – покриття гілок випадкового пошуку (зі збільшенням розміру пошукового простору Qtip не вдається покрити деякі гілки); b – ліва половина – це кількість оцінок Qtip, а права половина – кількість оцінок випадкового пошуку

Для діапазону вхідних даних від [-512, 512] і [-1024, 1024] Qtip досягає покриття гілок на рівні 68.12% і 64.15% відповідно. Це означає, що ефективність Qtip не залежить від розміру простору вхідних даних. У табл. 5 підсумовано результати.

Вікно станів

На рис. 4 відображено зміну покриття, виміряного кожні 100 епізодів. Коли як стан враховуються лише поточний рівень підходу, A, і відстань до гілки, Δ, агент мало навчається. Однак, якщо стан включає попередні рівні підходу і відстані до гілок з вікном розміром 200, агент покриває більше гілок у процесі навчання. Без вікна ми спостерігали, що агент має тенденцію повторювати тільки одну дію для будь-якого стану. Це вказує на можливість того, що Q-мережа відхиляється через недостатню зворотну інформацію про винагороду. На основі цього спостереження припускаємо, що наше припущення про часткову спостережуваність є дійсним.

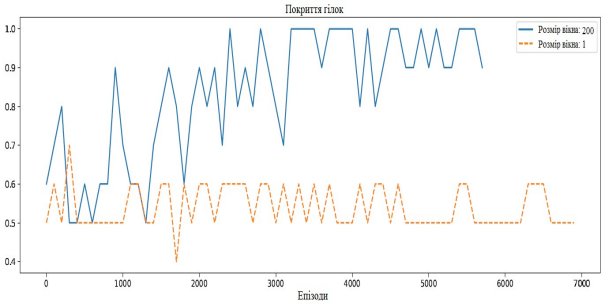


Рисунок 4 – Графіки зміни покриття: суцільна лінія – це Qtip з розміром вікна 200, а пунктирна лінія – з розміром вікна 1. Покриття гілок вимірюється окжні 100 епізодів. Хоча суцільна лінія зростає з продовженням навчання, пунктирна лінія не показує жодного покращення

Загрози валідності

Загрози внутрішній валідності включають правильність інструментів, що використовуються. Ми тренували Qtip за допомогою keras-НП, що є активно підтримуваною бібліотекою НП з відкритим вихідним кодом. GunPowder базується на Clang, який є фронт-ендом для мови C у широко використовуваному фреймворку LLVM.

Обидва інструменти пройшли обширну громадську перевірку. Сам GunPowder також є з відкритим вихідним кодом для подальшої громадської інспекції.

Загрози зовнішній валідності включають фактори, які можуть вплинути на те, наскільки добре узагальнюється висновок, такі як розмір емпіричного дослідження. Хоча наші висновки можуть бути обмежені розміром і вибором досліджуваних функцій, ми вважаємо, що це надає достатньо доказів для здійсненності формулювання SBST як задачі НП. Більш ґрунтовний аналіз витрат і вигод вимагатиме більшого емпіричного дослідження з більшою кількістю цільових функцій, а також застосування різних алгоритмів навчання.

Таблиця 5 – ***Середнє покриття гілок μ та стандартне відхилення σ від Qtip та випадкового пошуку за 30 запусків: найвище покриття для кожної функції виділено жирним шрифтом***

Діапазон вхідних даних	128				512				1024			
Функція	Qtip		Випадкове		Qtip		Випадкове		Qtip		Випадкове	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
Навчання	98.33	0.04	99.67	0.02	88.67	0.08	99.33	0.02	81.00	0.12	100.00	0.00
Залишок	43.97	0.06	99.74	0.01	40.13	0.06	99.87	0.01	37.95	0.07	100.00	0.00
НСД	77.50	0.01	100.00	0.00	63.33	0.10	100.00	0.00	56.67	0.12	100.00	0.00
ПДС	84.00	0.09	98.67	0.03	80.33	0.08	99.67	0.02	81.00	0.08	99.67	0.02
Всього	75.95	0.21	99.52	0.02	68.12	0.20	99.72	0.02	64.15	0.21	99.92	0.01

Висновок

У роботі було сформульовано генерацію тестових даних на основі пошуку як процес прийняття рішень для застосування навчання з підкріпленням. Була також представлена GunPowder, загальна структура для SBST, яка сумісна зі стандартним середовищем навчання з підкріпленням, OpenAI Gym. Наукова новизна полягає у використанні GunPowder. Проведено дослідження здійсненності генерації тестових даних на основі НП за допомогою невеликого емпіричного дослідження. Наша техніка (Qtip) досягає 100% покриття гілок для тренувальної функції та 60.06% покриття гілок для невидимих довільних функцій. Результати свідчать про те, що навчання поведінки метаевристичних алгоритмів є здійсненним.

Застосування пропонованого алгоритму буде корисним у сфері тестування різноманітного програмного забезпечення.

Список літератури / References

1. Harman, M., McMinn, P., de Souza, J. T., Yoo, S. (2012). Search Based Software Engineering: Techniques, Taxonomy, Tutorial. In: Meyer, B., Nordio, M. (eds) Empirical Software Engineering and Verification. LASER LASER LASER 2010 2009 2008. Lecture Notes in Computer Science, vol 7007. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-25231-0_1.
2. Silver, D., Huang, A., Maddison, C. *et al.* (2016). Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 484–489. https://doi.org/10.1038/nature16961.
3. Levytskyi V., Kruk P., Lopuha O., Sereda D., Sapaiev V., Matsiievskyi O. (2024). “Use of Deep Learning Methodologies in Combination with Reinforcement Techniques within Autonomous Mobile Cyber-physical Systems”, 2024 IEEE. (прийнято до друку).
4. Hado Van Hasselt, Arthur Guez, and David Silver. (2016). Deep Reinforcement Learning with Double Q-Learning. In AAAI. 2094–2100. https://doi.org/10.1609/aaai.v30i1.10295.
5. McMinn P. (2007). IGUANA: Input Generation Using Automated Novel Algorithms. A Plug and Play Research Tool. Technical Report CS-07-14. Department of Computer Science, University of Sheffield. https://doi.org/10.1145/3194718.3194720.
6. Lattner C. and Adve V. (2004). LLVM: A compilation framework for lifelong program analysis & transformation. In Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization. IEEE Computer Society, 75. https://doi.org/10.1109/CGO.2004.1281665.
7. Kim, J., You, B., Kwon, M., McMinn, P., Yoo, S. (2017). Evaluating CAVM: A New Search-Based Test Data Generation Tool for C . In: Menzies, T., Petke, J. (eds) Search Based Software Engineering. SSBSE 2017. Lecture Notes in Computer Science, vol. 10452. Springer, Cham. https://doi.org/10.1007/978-3-319-66299-2_12.
8. Brockman G., Cheung V., Pettersson L., Schneider J., Schulman J., Tang J., and Zaremba W. (2016). OpenAI Gym. (2016). https://doi.org/10.48550/arXiv.1606.01540.
9. Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep Sparse Rectifier Neural Networks. International Conference on Artificial Intelligence and Statistics 2011. *Fort Lauderdale, United States*. pp. 315–323.
10. Lecun Y., Bottou L., Bengio Y. and Haffner P. (1998). "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324. https://doi.org/10.1109/5.726791.
11. Kingma D. and Ba J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Стаття надійшла до редколегії 17.08.2024

Levytskyi Volodymyr
Postgraduate of the department of information technologies,
https://orcid.org/0000-0003-1829-488X
Kyiv National University of Construction and Architecture, Kyiv
Lopuha Oleksii
Postgraduate of the department of information technologies,
https://orcid.org/0000-0001-6397-2710
Kyiv National University of Construction and Architecture, Kyiv

TEST DATA GENERATION USING DEEP REINFORCEMENT LEARNING

Abstract. *The process of creating test data for software is one of the most complex and labor-intensive stages in the software development cycle. It requires significant resources and effort, especially when it comes to achieving high test coverage. Search-Based Software Testing (SBST) is an approach that automates this process by using metaheuristic algorithms to generate test data. Metaheuristic algorithms, such as genetic algorithms or simulated annealing, operate on the principle of systematically exploring possible options and selecting the most effective solutions based on feedback from a fitness function. This process is similar to reinforcement learning, where an agent receives rewards for correct actions and penalties for incorrect ones, with the goal of optimizing overall performance. We have proposed a new approach that explores the feasibility of applying reinforcement learning in SBST, with the aim of replacing traditional, human-designed metaheuristic algorithms. To this end, we developed the GunPowder platform, which transforms the software testing process (STP) into a learning environment. In this environment, the agent uses the Double Deep Q-Networks (DDQN) method, which incorporates deep neural networks to learn the optimal strategy for interacting with the software under test. Our study involved a series of empirical experiments to evaluate the effectiveness of the proposed approach. The results indicate that agents trained using GunPowder can effectively learn and apply metaheuristic algorithms, achieving high branch coverage in training functions. In particular, our agents achieved 100% branch coverage in the case of training functions, demonstrating their ability to adapt to different test scenarios. These findings highlight the potential of using deep neural networks and reinforcement learning in SBST, which could significantly improve the quality and efficiency of the software testing process in the future. Thus, our work opens up new opportunities for integrating modern machine learning methods into automated software testing, potentially reducing the time and resources required for testing and improving the quality of final products.*

Keywords: SBST; DDQN; GunPowder; STP; Qtip

Посилання на публікацію

APA Levytskyi, V. & Lopuha, O. (2024). Test data generation using deep reinforcement learning. *Management of Development of Complex Systems*, 59, 155–164, dx.doi.org\10.32347/2412-9933.2024.59.155-164.

ДСТУ Левицький В. В., Лопуга О. М. Генерація тестових даних за допомогою глибокого навчання з підкріпленням. *Управління розвитком складних систем.* Київ, 2024. № 59. С. 155 – 164, dx.doi.org\10.32347/2412-9933.2024.59.155-164.