УДК 004.056

АВТОМАТИЗОВАНИЙ ПОШУК ВРАЗЛИВОСТЕЙ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ІЗ ЗАСТОСУВАННЯМ МЕТОДІВ ГЛИБИННОГО НАВЧАННЯ

Кирило Чернов[1], Єгор Єрьомін[1], Марія Попова[1], Олексій Шаповал[1], Євген Котух[2]

[1] Харківський національний університет імені В.Н. Каразіна, майдан Свободи 4, Харків, 61022, Україна
kirillfilippsky@gmail.com, suvenick2@gmail.com, mariia.popova26@gmail.com, alex.shapoval@protonmail.com

[2] Університет митної справи та фінансів, вул. Володимира Вернадського 2/4, Дніпро, 49000, Україна
yevgenkotukh@gmail.com

Reviewer: Олександр Оксіюк, д.т.н., проф., Київський національний університет імені Т. Шевченка, вул. М. Ломоносова 81, Київ, 03189, Україна.
o.oksiuk@gmail.com

Надійшло: Листопад 2018.

Анотація: *Наведено теоретичну інформацію про тестування програмного забезпечення методом фаззінгу. Розглянуто технології навчання з підкріпленням та використання інтелектуального фаззінгу в процесі тестування програмного забезпечення. Описано алгоритм, за допомогою якого реалізуються зазначені методи та технології. Запропоновані статистичні результати досліджень, які були проведені під час тестування деяких програм та утиліт, призначених для повсякденного використання, а також програми, розробленої студентами.*

Ключові слова: *фаззінг; тестування; навчання з підкріпленням; Q-learning.*

1 Аналіз літератури та постановка задачі дослідження

Фаззінг – це метод тестування програмного забезпечення та відкритого коду на наявність вразливостей безпеки шляхом повторного тестування з подачею вхідних даних, що мутовані [1]. Повторне тестування проводиться методом випадкових мутації та майже завжди час, за який проводилось тестування, не є оптимальним. В цій роботі розглянуто проблему інтелектуального фаззінгу та методи її вирішення [7]. Основною метою авторів є розробка технології, що може орієнтуватися та приймати відповідні рішення, посилаючись на досвід, отриманий безпосередньо під час проведення тестування. Для рішення цього питання використано технологію машинного навчання, а саме навчання з підкріпленням (Reinforcement learning) за алгоритмом deep Q-learning [6], який реалізує максимально можливу винагороду, визначену в процесі розробки, використовуючи аналіз вихідних даних програми та доступні винагороди. Це дозволяє застосовувати оптимальні мутації вхідних даних. Таким чином, агент отримує можливість навчитися формувати оптимальну політику дій для отримання максимальної винагороди. В межах даної роботи пропонується алгоритм і відповідна комп'ютерна модель процесу фаззінгу із застосуванням глибинного навчання, та проводяться дослідження ефективності автоматизованого пошуку вразливостей у порівняні із тестуванням методом випадкової мутації. При проведенні тестування застосовується метод «чорного ящика», тобто інформація, яку ми маємо під час тестування, представляє собою лише результат роботи програми та вхідні дані, які їй необхідні для виконання [1].

2 Алгоритм інтелектуального фаззінгу

При проведенні дослідження ми прийшли до формування такої проблеми процесу фаззінгу: при випадковій генерації вхідних даних використовуваний час не є оптимальним, адже виконується процедура звичайного перебору варіантів. Їх може бути дуже багато, в результаті чого на вхід подається величезна кількість мутацій, які не приносять користі для процесу тестування. Оскільки процес фаззінгу представляє собою виконання циклу задач в визначеній середі (програмі), де на її вхід подається послідовність, яка пройшла деяку операцію му-

тації, ідеальним варіантом для вирішення подібної проблеми є технологія машинного навчання – навчання з підкріпленням (*Reinforcement learning*). Кращим прикладом використання такого алгоритму є програма AlphaGO, що розроблена компанією Google DeepMind в 2015 році. Вона стала першою в світі програмою, яка виграла партію в гру "го" у професіонала вищого рангу Лі Седоля [2].

Поєднавши фаззінг та навчання з підкріпленням, в результаті ми отримали систему, що здатна формувати правила вибору визначеної мутації. Вона подає на вхід мутовані дані і в залежності від вихідних даних програми формує нагороду для того, щоб при подальшому тестуванні спиратися на власний досвід та вибирати оптимальні мутації для конкретного випадку. Таким чином кількість мутації, що не несуть внеску в процес тестування, значно зменшиться. Це в свою чергу, як очікується, прискорить тестування. Схема розробленої системи представлена на рис. 1.

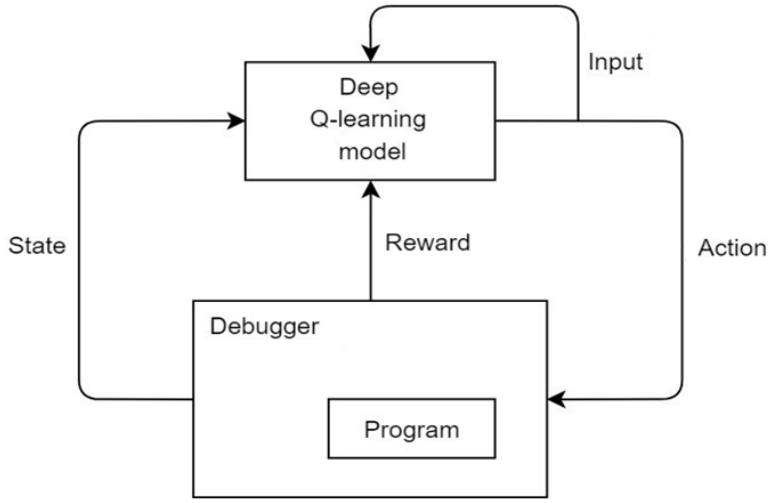


Рис. 1 – Схема алгоритму інтелектуального фаззінгу

Процес тестування починається з визначення початкових не мутованих вхідних даних. Формат залежить від типу фаззінгу. Сформовані пакети подаються на вхід програми, а за допомогою спеціальних засобів дебаггінгу визначається реакція програми. З отриманих даних (*це може бути час виконання програми, покриття коду, код завершення програми та ін*.) формується стан системи State, який, в разі необхідності, підлягає попередній обробці та подається в обробленому вигляді на вхід Deep Q-learning model, що в свою чергу приймає рішення про дію Action, котру слід прийняти наступною. В цей самий час, залежно від вибраної дії та отриманого стану програми, формується нагорода Reward для алгоритму. За допомогою винагороди алгоритм розуміє поставлене йому задачу і в процесі навчання визначає оптимальну поведінку для її виконання. Також алгоритм запам'ятовує дії які принесли йому максимальну нагороду для досягнення поставленої мети *(знаходження помилки, вивід програми з ладу, та ін*.), та в подальших раундах тестування, на власному досвіді, вирішує яку дію слід вчинити. При розрахунку наступної дії попередні вхідні дані підлягають мутації відповідно до дії, яку було вибрано. На вхід програми подаються вже "нові" мутовані дані. Ця процедура повторюється до дих пір, поки алгоритм не досягне поставленої йому цілі. Запропонована модель використовує Марківський процес прийняття рішень, а якщо більш точніше, то – deep Q-learning [3].

3 Навчання з підкріпленням (*Reinforcement learning*)

Навчання з підкріпленням – це обчислювальний підхід розуміння та автоматизації цілеспрямованого навчання і прийняття рішень. Він відрізняється від інших відомих алгоритмів машинного навчання тим, що агент навчається безпосередньо при взаємодії з середовищем, не посилаючись на зразкові приклади [3]. Цей алгоритм, перш за все, спрямований на вирі-

шення труднощів, які виникають при взаємодії з середовищем для досягнення довгострокових дій. Він використовує формальну структуру Марківського процесу прийняття рішень [3], визначення взаємодії між агентом і середовищем з точки зору станів, дій та нагород. Зазначені особливості включаються до себе розуміння причин та наслідків, а також наявність чітких цілей. При цьому, поняття цінності та функції цінності є основними ознаками методів навчання з підкріпленням.

Як було зазначено раніше, взаємодія агента з середовищем може бути розглянута, як Марківський процес прийняття рішень $M = (S, A, P)$, де S – набір станів системи, A – набір дій, P – множина перехідних ймовірностей. Для кожної пари стан – дія $(s, a) \in S \times A$, множина P це набір ймовірностей $P(s' \vee s, a)$, де s' це наступний стан системи. Тоді агент, розглядаючи можливі стани системи при вибраній дії, де кожному переходу відноситься своя нагорода $r(s, a)$, вивчає оптимальну поведінку, яка максимізує нагороду.

Під час процесу навчання основною метою алгоритму є максимізація кінцевої суми нагород:

$$R = \sum_{t=0}^{\infty} \gamma^t r_{t+1},$$

де $\gamma \in (0,1)$ – коефіцієнт знижки, що визначає пріоритет винагороди з плином часу. Вибір дії a_t при стані s_t визначається політикою дії $a_{t\pi}(\vee s_t)$. Політика π прикріпляє розглянуті можливі стани до дій, що в свою чергу визначає поведінку агенту.

Нехай очікувана кумулятивна нагорода для агенту, який слідкує політиці π визначається як:

$$Q^{\pi}(s, a) = E\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \vee s_0 = s, a_0 = a\right].$$

Тоді проблему знаходження оптимального значення $Q_{\pi}(s, a)$ можна звести до процедури апроксимації функції. Для досягання цього необхідно лише оновлювати $Q_{\pi}(s, a)$ після кожної ітерації отримання нагороди [4]. Це визначається як

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha,$$

де α – швидкість навчання (*learning rate*).

Всю процедуру можна записати в такій послідовності: агент отримує стан s_t, приймає дію $a_t = \arg\max(Q(s_t, a))$, що визначає нагороду r_t, та спричиняє перехід системи в стан $s_t + 1$. Отримуючи нагороду r_t та стан $s_t + 1$, агент визначає кращу можливу дію $a_t + 1 = \arg\max(Q(s, a))$. Далі він оновлює значення $Q(s_t, a_t)$.

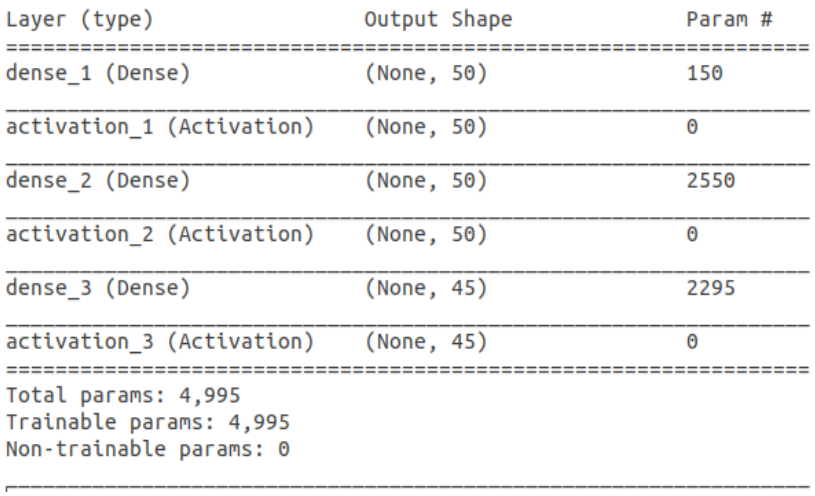
Для апроксимації функції $Q(s_t, a_t)$ використовуються глибокі нейронні мережі (*чим і визначається назва deep Q-learning*), де в свою чергу, метою є мінімізація функції втрати:

$$L = (r + \gamma \cdot \max(Q(s_t + 1, a_t)) - Q(s_t, a_t)^2.$$

4 Результати моделювання

Було змодельоване процес запуску програмного забезпечення, що піддається тестуванню, і розроблено спеціальні логічні тести, в яких програма повертала код помилки при визначених вхідних даних для порівняння роботи фаззінгу з допомогою штучного інтелекту та без нього. Можливі стани системи представляються у вигляді даних, що формуються при завершенні програми. Далі ці дані передаються нейронній мережі, яка складається з одного вхідного шару, двох прихованих шарів по 50 нейронів кожний та функції активації *(Rectified Linear Unit)*: $f(x) = \max(0, x)$. Вихід нейронної мережі має 45 елементів, що представляють собою кількість можливих мутацій.

Повна схема нейронної мережі для апроксимації функції $Q(s_t, a_t)$, представлена на рис. 2.



Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 50)	150
activation_1 (Activation)	(None, 50)	0
dense_2 (Dense)	(None, 50)	2550
activation_2 (Activation)	(None, 50)	0
dense_3 (Dense)	(None, 45)	2295
activation_3 (Activation)	(None, 45)	0

Total params: 4,995
Trainable params: 4,995
Non-trainable params: 0

Рис. 2 – Схема нейронної мережі

Навчання мережі здійснено за допомогою алгоритму градієнтного спуску Adam [4].

Нехай $f(\theta)$ – шумна цільова функція: стохастична скалярна функція, що є диференційованою відносно параметру θ. Ми зацікавлені в мінімізації очікуваної вартості цієї функції, $E\left[f(\theta)\right]$ відносно параметру θ. За допомогою $f_1(\theta),\ldots,f_T(\theta)$ позначаємо реалізацію стохастичної функції в наступних кроках $1,\ldots,T$. Стохастичність може виходити від оцінки на випадкових підмножинах (міні-група) точок даних або від шуму функції. При $g_t = \nabla_\theta f_t(\theta)$ ми позначаємо градієнт, тобто вектор часткових похідних f_t відносно θ, який оцінюється за часом t. Алгоритм оновлює експоненціальні рухливі середні значення градієнту (m_t) та квадрату градієнту (v_t), де гіперпараметри $\beta_1, \beta_2 \in [0,1]$ контролюють експоненціальні швидкості розкладання цих рухливих середніх. Самі рухливі середні – оцінки першого моменту (*середнє значення*) і другого моменту (*не центрована дисперсія*) градієнта. Проте ці рухливі середні ініціюються як вектори нулів, що призводить до оцінки моментів, котрі зміщуються у напрямку до нуля, особливо в початкових часових кроках, та коли показники розкладання невеликі (наприклад, значення β_s близьке до 1). Корисна якість полягає в тому, що цієї упередженості ініціалізації можна легко запобігти, отримуючи виправлені помилки m_{bt} та v_{bt}.

Структура алгоритму, що розглядається, представлена нижче.

Необхідні вхідні дані:

- α : Швидкість навчання
- $\beta_1, \beta_2 \in [0,1)$: Експоненціальні показники розпаду для оцінок моменту (стандартні налаштування $\beta_1 = 0.9, \beta_2 = 0.999$)
- $f(\theta)$: Стохастична функція з параметром θ
- θ_0 : Вектор початкових параметрів

Алгоритм:

$m_0 \leftarrow 0$ (*Ініціалізувати перший вектор моменту*)

$v_0 \leftarrow 0$ (*Ініціалізувати другий вектор моменту*)

$t \leftarrow 0$ (*Ініціалізувати час*)

while θ_t не зійшлась:

$t \leftarrow t+1$

$g_0 \leftarrow \nabla\theta f_t(\theta_t - 1)$ (*Взяти градієнт відносно стохастичної функції під час* t)

$m_t \leftarrow \beta_1 \bullet m_t - 1 + (1 - \beta_1) \bullet g_t$ (*Оновити відхилену оцінку першого моменту*)

$v_t \leftarrow \beta_2 \bullet v_t - 1 + (1 - \beta_2) \bullet g_{2t}$ (*Оновити відхилену оцінку другого моменту*)

$m_b t \leftarrow \frac{m_t}{1 - \beta_{t_1}}$ (*Обчислити виправлену оцінку першого моменту*)

$yb_t \leftarrow \frac{y_t}{1 - \beta_{t_2}}$ (*Обчислити виправлену оцінку другого моменту*)

$\theta_t \leftarrow \theta_t - 1 - \alpha \bullet \frac{mb_t}{\sqrt{vb_t}}$ (*Оновити параметри*)

end while

return θ_t (*Розраховані параметри*)

Мутації вхідних даних були вибрані на основі стандартного списку: збільшення та зменшення довжини рядка, цілочисленні вставки, додавання спеціальних символів (наприклад, "%s", який також може викликати помилки). Було створено 45 функцій і поміщено до словнику для їх подальшого використання.

Нагороду система отримувала, якщо час виконання системи був більшим за попередній та при виникненні помилок під час тестування. При знаходженні помилки алгоритм закінчує роботу. При формуванні такого типу нагороди ми зіткнулися з проблемою, коли алгоритм вже знайшов одну помилку, почав її викликати кожного разу за для отримання максимальної нагороди. Щоб уникнути цієї проблеми ми встановили дві константи: $\gamma \in (0,1)$ – коефіцієнт знижки та $\varepsilon \in (0,1)$ – швидкість розвідки. Про першу константу вже було зазначено раніше. Друга константа визначає наскільки алгоритм буде здатен до відкриття нових рішень, тобто з вірогідністю ε буде вибрана випадкова дія, а з вірогідністю $1 - \varepsilon$ буде вибрана максимально вигідна дія.

Гіпотеза – наукове припущення, що висувається для пояснення будь-якого явища і потребує перевірки на досліді та теоретичного обґрунтування для того, щоб стати достовірною науковою теорією [8].

Статистична гіпотеза – будь-яке твердження (припущення), яке стосується вигляду чи параметрів розподілу деякої ознаки досліджуваних об'єктів [8].

Тестування гіпотез проводиться в такій послідовності дій:

1. Здійснюється обчислення певної статистики, розподіл якої відомий.
2. Знаходиться *P-value* для обчислених результатів.
3. Робляться відповідні висновки в залежності від критерію значущості та значенні *P-value*.

Для проведення тестування ми розробили спеціальний тест, в якому була визначена помилка. Гіпотеза нашого експерименту полягає в тому, чи є тестування за допомогою нашого алгоритму швидшим, ніж випадковий вибір дій. Коефіцієнт знижки встановлений 0.9, а швидкість розвідки 0.5. Остання зменшується в 0.99 разів після кожної епохи. Для перевірки гіпотези було використано критерій Ст'юдента [5].

Критерій Ст'юдента – загальна назва класу методів статистичної перевірки гіпотез (*статистичних критеріїв*), заснованих на порівнянні з розподілом Ст'юдента. Найчастіші випадки застосування цього критерію пов'язані з перевіркою рівності середніх значень у двох вибірках [5]. Щоб використовувати цей критерій треба задовольнити деяким умовам: нормальний розподіл початкових даних та рівність дисперсії.

Першою групою були результати тестування за допомогою запропонованого алгоритму, а в іншій групі були результати випадкового вибору мутації. Тестування проводилось в наступній послідовності: генерується 15 експериментів, наприкінці кожного записується кількість

мутацій, яка знадобилася для знаходження помилки. Відповідні результати експериментів представлені в Таблиці 1.

Таблиця 1 – Результати тестування

№	Deep Q-learning model	Випадковий вибір мутації
1	3671	3686
2	1191	1897
3	1879	3164
4	1640	3233
5	1966	10446
6	1585	5358
7	1135	1134
8	4877	752
9	2465	2157
10	3266	3684
11	1895	2026
12	2093	2993
13	1150	295
14	1181	3381
15	1153	358

Результати розрахунку критерію Ст'юдента: $t = -12.40$.

Число степенів свободи дорівнює: $v = 2n - 2 = 2 \times 15 - 2 = 28$.

При критерії значущості $\alpha = 0.01$ та $P - value = 2.763$, оскільки $t < P - value$, наша гіпотеза приймається. Результат є статистично значущим при заданому критерії, якщо, за умови вірності нульової гіпотези, ймовірність випадкового виникнення такого ж або більш екстремального результату менша від заданого рівня (0.01).

5 Висновки

Як свідчать результати експериментів час тестування за допомогою розробленого алгоритму кращий ніж час тестування з використанням випадкових мутацій, навіть за умови, що алгоритм не навчався перед початком проведення експерименту.

Запропонований авторами роботи алгоритм, в його порівнянні з алгоритмом з випадковим тестуванням, знаходить помилку за значно меншу кількість мутацій (2076, проти 2832), та, в середньому, знаходить помилку на 30% швидше.

Посилання

[1] Sutton M., Greene A., Amini P. Fuzzing: Brute Force Vulnerability Discovery. Boston, MA, USA: Addison-Wesley Professional, 2007. URL: https://www.oreilly.com/library/view/fuzzing-brute-force/9780321446114/
[2] AlphaGo Games – English. URL: https://deepmind.com/research/alphago/match-archive/alphago-games-english/
[3] Sutton R.S., Barto A.G. Reinforcement learning: An introduction. MIT press Cambridge, 1998. URL: http://incompleteideas.net/book/bookdraft2017nov5.pdf
[4] Kingma D. P., Ba J. Adam: A Method for Stochastic Optimization. URL: https://arxiv.org/pdf/1412.6980.pdf
[5] t-kryterii Stiudenta. URL: http://fpo.bsmu.edu.ua/static/t-kryteriy-styudenta
[6] Li Yu. Deep Reinforcement Learning: An Overview. URL:https://arxiv.org/pdf/1810.06339.pdf
[7] Böttinger K., Godefroid P., Singh R. Deep Reinforcement Fuzzing. URL: https://arxiv.org/pdf/1801.04589.pdf
[8] Deviniak O. Statystychni hipotezy ta yikh perevirka. 2014. URL: http://stat.org.ua/statclasses/hypotheses-testing/ Deviniak Statystychni hipotezy ta yikh perevirka

Reviewer: Oleksandr Oksiiuk, Doctor of Sciences (Engineering), Full Professor, Taras Shevchenko National University of Kiev 81 Lomonosova St., Kyiv, 03189, Ukraine. E-mail: o.oksiuk@gmail.com

Received on November 2018.

Authors:
Kyrylo Chernov, Student, V. N. Karazin National University, Kharkov, Ukraine.
E-mail: kirillfilippsky@gmail.com
Yehor Yeromin, Student, V. N. Karazin National University, Kharkov, Ukraine.
E-mail: suvenick2@gmail.com
Popova Mariia, Student, V. N. Karazin National University, Kharkov, Ukraine.
E-mail: mariia.popova26@gmail.com
Shapoval Oleksiy, Student, V. N. Karazin National University, Kharkov, Ukraine.
E-mail: alex.shapoval@protonmail.com
Yevgen Kotukh, Ph.D., Associative professor of the Department Cybersecurity of the University of Customs and Finance, Dnipro, Ukraine. E-mail: yevgenkotukh@gmail.com

Automated software vulnerability testing using in-depth training methods.

Abstract. Theoretical information about testing of software using fuzzing method. The technologies of reinforcement training and intellectual fuzzing in the software testing process. An algorithm is described with the help of which the indicated methods and technologies are realized. Statistical results of studies that were conducted during the testing of some programs and utilities intended for everyday use, as well as the program developed by the students are offered.

Keywords: fuzzing; testing; reinforcement learning; Q-learning.

Рецензент: Александр Оксиюк, д.т.н., проф., Киевский национальный университет имени Т. Шевченко, ул. Ломоносова 81, Киев, 03189 Украина. E-mail: o.oksiuk@gmail.com

Поступила: Ноябрь 2018.

Авторы:
Кирилл Чернов, студент, Харьковский национальный университет имени В. Н. Каразина, Харьков, Украина.
E-mail: kirillfilippsky@gmail.com
Егор Ерёмин, студент,Харьковский национальный университет имени В. Н. Каразина, Харьков, Украина.
E-mail: suvenick2@gmail.com
Мария Попова, студент, Харьковский национальный университет имени В. Н. Каразина, Харьков, Украина.
E-mail: mariia.popova26@gmail.com
Алексей Шаповал, студент, Харьковский национальный университет имени В. Н. Каразина, Харьков, Украина.
E-mail: alex.shapoval@protonmail.com
Евгений Котух, к.т.н., доцент кафедры кибербезопасности, Университет таможенного дела и финансов, Днепр, Украина.
E-mail: yevgenkotukh@gmail.com

Автоматизированный поиск уязвимостей программного обеспечения с применением методов глубинного обучения.

Аннотация. Приведена теоретическая информация о тестировании программного обеспечения методом фаззинга. Рассмотрены технологии обучения с подкреплением и интеллектуального фаззинга в процессе тестирования программного обеспечения. Описан алгоритм, с помощью которого реализуются указанные методы и технологии. Предложены статистические результаты исследований, которые были проведены во время тестирования некоторых программ и утилит, предназначенных для повседневного использования, а также программы разработанной студентами.

Ключевые слова: фаззинг; тестирование; обучение с подкреплением; Q-learning.