УДК 519. 688

ЗАСТОСУВАННЯ МЕТОДУ НАВЧАННЯ З ПІДКРІПЛЕННЯМ ДЛЯ КЕРУВАННЯ ВИВІЛЬНЕННЯМ ОПЕРАТИВНОЇ ПАМ'ЯТІ

Є. М. Юдін[1, a]

[1] *Національний Технічний Університет України «Київський Політехнічний Інститут»*

Анотація

У роботі розглянуто можливість застосування алгоритму навчання з підкріпленням до задачі автоматичного керування вивільненням оперативної пам'яті під час виконання прикладних програм для зменшення частоти виклику прибиральника. В роботі описується реалізований автором маркуючий-зачищаючий прибиральник сміття з алгоритмом навчанням з підкріпленням Actor-Crtitic, що працює з моделлю оперативної пам'яті комп'ютера. Продуктивність програми було порівняно з роботою звичайних алгоритмів, що не використовують машинне навчання і в результаті порівнянь алгоритм показав більшу ефективність у випадку небажаності виконання екстреного прибирання. Також було підтверджено адаптивність розробленого алгоритму до зміни зовнішніх умов.

Ключові слова: навчання з підкріпленням, машинне навчання, прибиральник сміття

Вступ

Задача керування вивільненням пам'яті, яка також називається задачею збиранням сміття, є однією з форм задач автоматичного керування оперативною пам'яттю під час виконання програм. Таке керування полягає у своєчасному вивільнені пам'яті від об'єктів, які не будуть використовуватися програмою у подальшому. Відповідний процес, що вирішує цю задачу, називається прибиральником сміття.

Навчання з підкріпленням – це розділ штучного інтелекту, що дозволяє реалізувати заздалегідь не визначені конкретно алгоритми, використовуючи формалізм взаємодії агенту і оточуючого середовища. На відміну від підходу навчання з вчителем, процес навчання реалізується не через прямі вказівки щодо дій, а за допомогою системи нагород і штрафів. Такий підхід може бути застосований до задач пошуку інформації, автоматизованого контролю, медичної діагностики, теорії ігор та інших, у тому числі для керування оперативною пам'яттю [1]. Незважаючи на те, що навчання з підкріпленням розвивається досить тривалий час, у більшості подібних задач, зазвичай, використовують класичні алгоритми, хоча нові підходи можуть виявитися більш ефективними.

У даній роботі розглянуто адаптивні процеси прийняття рішень, що застосовується для задачі збирання сміття. Основною інформацією, що використовується для прийняття рішення, є характер розподілу пам'яті виконуваними програмами, а навчання виконується через метод проб та помилок з метою виконання у подальшому більш ефективних дій у початково невідомому середовищі.

[a] yehor.yudin1@gmail.com

1. Постановка проблеми

Комп'ютерні програми працюють у динамічних умовах, використовуючи заздалегідь невідому кількість обчислювальних ресурсів. У випадку з прибиральником сміття зовнішньою змінною є програма, яку він обслуговує. Очевидно, що стратегія вивільнення пам'яті, яка виявилась ефективною для однієї програми, може не бути такою для іншої. Одним з дієвих способів запобігти розробки прибиральника сміття для кожної програми окремо є створення універсального прибиральника, що не залежить від початкових умов, і може виробляти оптимальний алгоритм вивільнення пам'яті на основі навчання з підкріпленням. Однією з найбільш популярних мов, що використовує збирання сміття, є Java. У сучасній віртуальній машині HotSpot/JRockit проблему вибору стратегії збирання сміття вирішено завдяки наявності декількох різних збиральників сміття, з різними алгоритмами роботи [2]. Вибір між ними виконується вручну, а потім адаптується під конкретну програму, шляхом налаштування набору параметрів. Проте, жоден зі збиральників сміття не використовує навчання з підкріпленням. Таким чином, мета дослідження – запропонувати алгоритм керування прибиранням сміття, що використовує навчання з підкріпленням для вибору дій, зв'язаних з вивільненням пам'яті, що зменшить частоту викликів прибиральника та дозволить адаптуватися до поведінки обслуговуваних прикладних програм.

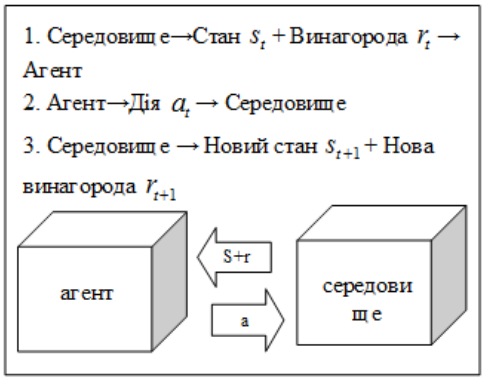

Рис. 1. Ілюстрація до моделі навчання з підкріпленням: взаємодія агента з середовищем

2. Методіка дослідження

Основними математичними поняттями, на яких базується навчання з підкріпленням є стан та множина станів, дія та множина дій, стратегія та множина стратегій, функція користі та функція користі пари стан-дія. Роботу методів навчання з підкріпленням теоретично обґрунтовано для тих випадків, коли модель системи, що навчається можливо подати у вигляді Марковського процесу прийняття рішень [3]. Для цього потрібне виконання марковскьої властивості відносно еволюції системи, тобто її наступний стан повинен залежати лише від поточного стану та обраної дії, а не від її історії. Це виражається формулою:

$$Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, \ldots, s_1, a_1, r_1\} = Pr\{s_{t+1} = s, r_{t+1} = r | s_t, a_t, r_t\}$$

де s – стан системи, r – винагорода, a – дія. Для досягнення цієї умови необхідно правильно визначити як описувати стан системи. Також для гарантованої збіжності алгоритмів необхідно повне знання системи агентом, хоча для більшості реальних випадків без повного знання працює і вибірковий метод проб та помилок.

Як вже було зазначено, у формалізмі навчання з підкріпленням система, яка називається агентом, взаємодіє з середовищем, виконуючи деякі дії і отримуючи деяку винагороду. У загальному випадку процес навчання можна описати, як на рис. 1 [4]. Спершу середовище, що знаходиться у певному стані s з множини станів S, передає деяку винагороду r агенту, що разом з минулими винагородами впливає на вибір стратегії поведінки π. У відповідь агент виконує деяку дію a з множини дій A, що змінює стан середовища на новий s'. Середовище посилає нову винагороду r' і починається новий крок.

Стратегія – це відображення $\pi : S \times A \to [0; 1]$, тобто визначена ймовірність виконати агентом певну дію a, коли середовище знаходиться у певному стані s. Результатом повинна бути максимізація певної сумарної винагороди. Необхідно враховувати, що не завжди вибір дії, що дасть більшу винагороду на наступному кроці, призведе до максимуму усієї винагороди. Зазвичай у якості сумарної винагороди вводять функцію вигляду $R = r_t + \gamma\, r_t t + 1 + \gamma^2 r_{t+2} + \ldots$, де $\gamma \in [0; 1]$ – коефіцієнт, що визначає важливість наступних винагород у порівнянні з тільки що отриманими.

3. Опис моделі та алгоритму

У результаті роботи було побудовано формалізовану модель автоматичного керування вивільненням пам'яті. Також було запропоновано та реалізовано алгоритм вивільнення пам'яті з навчанням з підкріпленням. Робота цього алгоритму була промодельована, а її результати – описані та порівняні з роботою класичних алгоритмів.

Створений алгоритм задіяє три сутності, що взаємодіють між собою – оперативна пам'ять, що виступає як середа навчання з підкріпленням, мутатор та прибиральник сміття, що є агентом навчання.

У представленій моделі пам'ять – це масив байтів довжиною L, що можуть бути зайнятими чи ні. В пам'яті розташовані об'єкти – програмні сутності, що мають адресу першого байту у пам'яті Ad_i, розмір та час життя. Кожен з них представляє собою неперервну послідовність зайнятих байтів пам'яті. Пам'ять може створювати у собі нові об'єкти, змінюючи свій стан, у відповідь на виклик команди мутатора та повертати йому адресу та нового об'єкта або повідомлення про помилку у разі провалу операції. Також вона може надавати інформацію про свій стан прибиральнику та інформацію про успішність або невдалість виконання прибирання, тобто формально – винагороду або штраф.

У якості вектору стану середовища, тобто набору чисельник характеристик, обрано вектор $\mathbf{s} = (s_1, s_2)$, перша компонента s_1 якого – кількість зайнятих байтів, а друга s_2 – збільшення зайнятої пам'яті з минулого кроку. Розіб'ємо шкали значень компонент на підінтервали, у нашому випадку $Nint_i = 10$, і в результаті отримаємо 100 станів, кожен з яких є декартовим добутком підінтервалів шкал. Таким чином, оскільки шкали значень обмеженні і всі стани відомі, отримуємо опис середовища як повністю спотережуваного Марковського ланцюга.

Мутатор – програмна сутність, що виконує запити на створення об'єктів у пам'яті і передає їй характеристики об'єкта – розмір та час життя. Він створює запити на створення як деякий пуассонівський потік з дискретним часом. Тобто кількість створених на кожному такті об'єктів – це випадкова величина з розподілом Пуассона $Pois(\lambda)$, де λ – середня кількість об'єктів, що виникла на одному такті, або $Pr\{N_{i\}=k} = \frac{e^{-\lambda}\lambda k}{k!}$. Кожен об'єкт O_i має випадково згенерований розмір l_i та час життя t_i, що є випадково розподіленими величними на деяких інтервалах значень $[l_{min}, l_{max}]$, $[t_{min}, t_{max}]$.

У нашому випадку агент – це програма, що реалізує прибиральник сміття. Для нього визначено дві дії – або виконання повного прибирання a_0, або очікуван-

ня a_1, тобто множина дій $A = \{a_0, a_1\}$ двоелементна. На кожному такті прибиральник за функцією користі стану $Q = Q(a_i, s_j)$, у якому знаходится пам'ять, визначає ймовірність виконати збирання і після виконання однієї з дій отримує штраф. У нашому випадку ймовірність виконання дії визначається однозначно за формулою

$$\pi(s_t, a_1) = sigm(Q(a_1, s_t)) = \frac{1}{1 - exp(-Q(a_1, s_t))} \quad (1)$$

тобто за допомогою сигмоідальної логістичної функції, що відображає усю числову вісь у відрізок $[0; 1]$, $sigm(x) : \Re \to [0; 1]$. Завдяки такому визначенню ймовірності виконання дій на будь-якому кроці завжди існує ненульова ймовірність виконати дію з меншою вигодою – з ціллю перевірити чи не з'явиться у неї в наслідок зміни зовнішніх умов якась нова користь, проте більшу ймовірність виконання мають гарантовано більш корисні для агента дії. У якості сигналу від середовища обрано саме штрафи, тобто від'ємнозначні дійсні числа, що отримує агент після дії, оскільки ми маємо задачу мінімізації і кожне збирання нами оцінюється негативно. У нашому випадку це $r_{(1)} = -1$ умовна одиниця за виконання збирання та $r_2 = -3$ умовних одиниць за нестачу пам'яті і виконання так званого екстреного збирання (ці штрафи - лише числельні вирази що використовуються в обрахунках, наважливіше в них – це їх співвідношення) Виходячи з отриманих штрафів прибиральник оновлює функцію корисності за принципом Actor-Critic. Тобто спершу формується TD-похибка, що дорівнює

$$\delta_t = r_{t+1} + \gamma\, V(s_{t+1}) - V(s_t)$$

Далі поновлюється функція користі стану

$$V'(s_{t+1}) = V(s_t) + \alpha\delta_t$$

та функція користі пари стан-дія

$$Q'(a_1, s_t) = Q(a_1, s_t) + \beta\delta_t$$

Що і використовується для визначення стратегії.

Тут α та β – це коефіцієнти навчання, що відображають довіру до нової інформації, а γ – коефіцієнт дисконтування, що відображає планування майбутніх винагород. За рахунок властивостей пуассоновського потоку у моделі у такій системі з'являється марковська властивість, тобто відсутність пам'яті. Цей алгоритм не сильно залежить від початкових функцій користі, тому ініціюємо їх як тотожно рівну нулю.

Відповідно на кожному такті відбувається генерування запиту мутатором, розміщення об'єктів у пам'яті, дії та навчання прибиральника – виконується випадково одна з випадкових дій з ймовірністю (3) та виконується оновлення функцій корисності, а також відбувається зменшення строку життя вже створених об'єктів. Якщо він падає до нуля – це значить що мутатору цей об'єкт вже не знадобиться і його можна буде видалити у майбутньому. Проте це відомо мутатору, а не пам'яті, тобто для неї цей об'єкт ще займає певне місце.

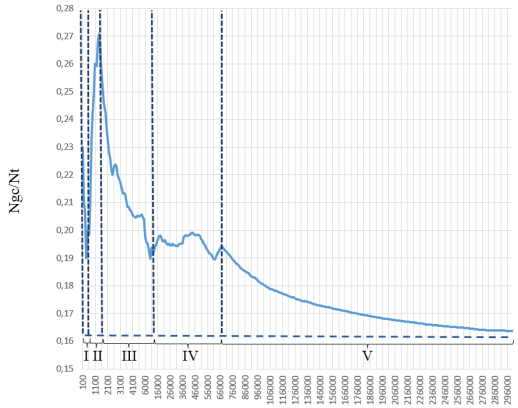

Рис. 2. Залежність частоти викликів прибиральника від часу у загальному випадку

4. Моделювання та аналіз отриманих результатів

У якості основного параметру ефективності взято співвідношення кількості виконаних збирань до кількості тактів на певний момент, оскільки цей показник також демонструє і швидкість навчання.

Було перевірено, що для великої кількості тактів середній час, що був витрачений на збирання, або частота викликів прибиральника, збігається до деякої оптимальної величини. Також було виявлено і продемоснстровано на рис. 2, що у загальному випадку процес навчання виконується у п'ять етапів: на першому етапі після ініціалізації функції користі відбувається початкове навчання і параметр спадає (зазвичай така функція недостатньо враховує майбутні винагороди і виявляється неоптимальною, тому цей етап дуже швидко закінчується); далі після початкового встановлення функції користі параметр швидко росте через велику долю неправильних дій (але цей етап також швидко закінчується); на третьому етапі формується первинна функція користі і параметр спадає; потім відбувається досить тривалий період спроб та помилок із налаштуванням функції користі (що відображається як випадкова зміна параметра у певному діапазоні); на останньому, п'ятому, етапі відбувається тонке налаштування функції і параметр збігається до деякого значення.

Також для порівняння ефективності алгоритму, що використає навчання з підкріпленням, було реалізовано класичний серійний маркуючий-зачищаючий прибиральник сміття. Він використовує ту ж саму модель пам'яті та мутатора, проте сам прибиральник алгоритм полягає у обранні дії прибирання тільки у тому випадку, коли у пам'яті виникає помилка зв'язана з виділенням пам'яті для нового об'єкта. Така ситуація не завжди є рівноцінною зі звичайним прибиранням, оскільки таке екстрене збирання може займати більше часу або потенційно викликати інші помилки. Тому доцільно також було дослідити кількість екстрених прибирань та його відношення до кількості тактів.

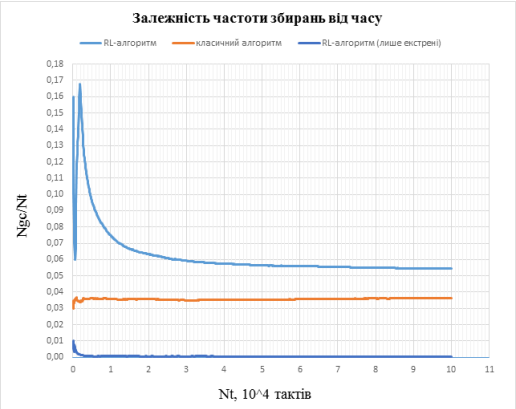

Рис. 3. Графік часототи збирань та екстрених збирань для нашого алгоритма з навчанням, частота для класичного алгоритма

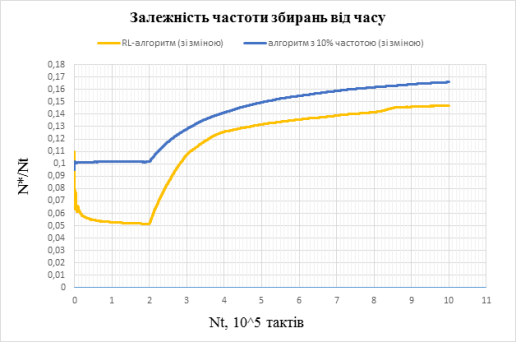

Рис. 4. Реакція на зміну поведінки мутатора класичного алгоритма та такого, що навчається

У такому випадку частота збирань, до якої збігається алгоритм з навчанням з підкріпленням, більша за частоту алгоритму, що чекає переповнення пам'яті, у приблизно 1.5 рази, адже така частота мінімальна з усіх можливих. Проте доля екстрених прибирань, тобто таких, що виконуються після помилки відділення пам'яті, для алгоритму з навчанням швидко падає до 1% від всіх збирань, що склало відповідно близько 1.5% від результату класичного алгоритму, що видно з рис. 3. Якщо виразити цінність кількості збирань пропорційно зазначеним у моделі штрафам за них, тобто екстрене збирання вважати у $w = \frac{r_3}{r_1} = 3$ рази менш бажаним, ніж звичайне збирання, то алгоритм з навчанням виграє приблизно у 2.5 рази.

Іншим звичайним рішенням для класичного алгоритму є встановлення користувачем деякої частоти викликів прибиральника. У даному випадку очікувана і встановлена частотою виклика прибиральника була 0.1. Алгоритм з навчанням не потребує встановлення частоти чи інших параметрів, у наслідок навчання частота без вказівок з боку користувача збігається до знайденого оптимального значення, у даному випадку – приблизно 0.055.

Також було досліджено поведінку алгоритмів у випадку зміни поведінки мутатора. На рис. 4 зображено графік реакції на подвоєння інтенсивності виникнення об'єктів для мутатора після 200000 ітерації алгоритму. Ріст частоти викликів для класичного алгоритму з заданою частотою зумовлений лише збільшенням частоти екстрених викликів прибиральника. Алгоритм з навчанням після зміни інтенсивності без інформації від користувача реагує на зміни, що відбулися, і починає перерахунок функцій цінності. У результаті частота викликів для нього збігається до величини у приблизно 1.15 меншої, аніж для класичного алгоритму. Характер перерахунку зв'язаний з загальною для алгоритмів з навчанням властивістю – після зміни характеру поведінки середовища фактично виникає нова задача навчання і функції користі, отримані на минулій задачі, можуть бути невигідними у якості ініційованих функцій для нової задачі.

Висновки

У роботі розглянуто можливість застосування навчання з підкріпленням до задачі збирання сміття під час виконання прикладних програм. Побудовано модель системи, що навчається, визначено основні її основні сутності та параметри та створено алгоритм вивільнення пам'яті, за яким вона працює.

Була написана прикладна програма, що втілює створену модель та алгоритм. Програма демонструє працездатність алгоритму збирання на основі навчання з підкріпленням – для великої кількості тактів частота викликів прибиральника збігається до деякої оптимальної з точки зору моделі навчання величини.

У ході моделювання було підтверджено адаптивність алгоритму до змін поведінки мутатора. Запропонований алгоритм дозволив майже повністю уникнути екстрених викликів прибиральника (частота екстрених викликів складає приблизно $1,5\%$ від частоти для класичного алгоритма), проте для даної моделі керування пам'яттю він не виявився кращим за простий алгоритм, оскільки простий алгоритм за теорією для даного випадку є оптимальним. У рамках запропонованої ситуації, коли екстрене збирання у декілька разів менш бажане за звичайне, алгоритм з навчанням дозволив знизити частоту викликів з відповідною вагою $w = 3$ приблизно у 1.5 рази.

Література

1. R. S. Sutton, A. G. Barto Reinforcement Learning: An Introduction — MIT Press, Cambridge, MA, 1998 A Bradford Book

2. Java Documentation Java Platform, Standard Edition HotSpot Virtual Machine Garbage Collection Tuning Guide

3. D.Andre Programmable Reinforcement Learning Agents. — Stanford University, 1994

4. E.Andreasson Reinforcement Learning for a Dynamic Java™ Virtual Machine — Royal Institute of Technology Stockholm, 2002