МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Бочкарьов О. Ю.

Навчання з підкріпленням в автономних інтелектуальних системах

Навчальний посібник

Львів
Видавництво Львівської політехніки
2022

УДК 004.8
Б

Рецензенти:

Рак Т. Є., доктор технічних наук, доцент, професор кафедри інформаційних технологій, проректор з науково-педагогічної роботи ПЗВО “ІТ СТЕП Університет”;

Ткачук Р. Л., доктор технічних наук, професор, начальник кафедри управління інформаційною безпекою, Львівський державний університет безпеки життєдіяльності;

Глухов В. С., доктор технічних наук, професор, професор кафедри електронних обчислювальних машин, Національний університет “Львівська політехніка”

Рекомендувала Науково-методична рада
Національного університету “Львівська політехніка”
як навчальний посібник для студентів спеціальності 123 “Комп’ютерна інженерія”
(протокол № 65 від 20 жовтня 2022 р.)

Бочкарьов О. Ю.

Б Навчання з підкріпленням в автономних інтелектуальних системах : навч. посібник / О. Ю. Бочкарьов. – Львів: Видавництво Львівської політехніки, 2022. – 122 с. Режим доступу: http://eom.lp.edu.ua/textbooks/np-npais.pdf

ISBN xxx-xxx-xxx-xxx-x

В навчальному посібнику розглянуто теоретичні принципи навчання з підкріпленням в автономних інтелектуальних системах. Основна увага приділена моделям та методам навчання з підкріпленням. Розглянуто питання практичного застосування методів навчання з підкріпленням для побудови автономних інтелектуальних систем.

Навчальний посібник призначений для студентів спеціальності 123 “Комп’ютерна інженерія” галузі знань 12 “Інформаційні технології”.

УДК 004.8

ЗМІСТ

Вступ

Одним з головних напрямків розвитку ідей і технологій штучного інтелекту є створення автономних інтелектуальних систем, що здатні самостійно без участі людини розв’язувати поставлені перед ними складні завдання [1-5]. Прикладами автономних інтелектуальних систем можуть бути автономні роботи, безпілотні літальні апарати, автономні підводні апарати, самокеровані автомобілі, інтелектуальні програмні агенти, тощо. Сучасні досягнення в області комп'ютерних технологій, технологій бездротового зв'язку, мобільних обчислень та в інших областях дають можливість реалізовувати все більш складні архітектури автономних інтелектуальних систем і нарощувати їх функціональність [6,7]. Разом з тим масштаби застосування автономних інтелектуальних систем у різних областях також стрімко зростають. Це звільняє людей від виконання рутинних операцій та роботи у небезпечних умовах, а також відкриває нові цікаві можливості, наприклад, дозволяє розв’язувати складні задачі в недоступних для людини середовищах або з недосяжною для людського мозку швидкістю прийняття рішень. Відтак питання створення та застосування автономних інтелектуальних систем є надзвичайно актуальним. Серед найбільш перспективних напрямків у дослідженні та проектуванні автономних інтелектуальних систем – використання методів машинного навчання, зокрема методів навчання з підкріпленням. Вони дозволяють створювати автономні інтелектуальні системи спроможні вивчати поведінку та особливості об’єкту управління та свого оточення з метою пристосування до непередбачуваних з точки зору розробника змін та самостійного пошуку послідовності цілеспрямованих ефективних дій на шляху до поставленої перед системою мети.

В навчальному посібнику розглянуто теоретичні принципи навчання з підкріпленням в автономних інтелектуальних системах. Оглянуто підходи до застосування методів штучного інтелекту для побудови інтелектуальних систем. Розглянуто узагальнену функціональну схему автономної інтелектуальної системи та проблему прийняття рішень в умовах невизначеності та нестачі інформації. Наведено принципи моделювання простих форм цілеспрямованої поведінки у стаціонарному випадковому середовищі та випадковому середовищі з перемиканням станів. Розглянуто принципи побудови та конструкції цифрових автоматів, що навчаються (Learning Automata). Основна увага в навчальному посібнику приділена моделям та методам навчання з підкріпленням. Наведено класифікацію задач навчання з підкріпленням. Розглянуто принципи навчання з підкріпленням у випадковому середовищі Multi-armed Bandit (MAB) та відповідні методи навчання з підкріпленням, зокрема методи зваженої оцінки дій (action-value methods), метод експоненціального усереднення, метод нормованої експоненціальної функції, метод на основі стохастичного градієнтного підйому та метод верхньої довірчої межі. Розглянуто задачу навчання з підкріпленням у випадковому середовищі MAB з контекстною залежністю. Розглянуто проблему навчання з підкріпленням для випадку марківського процесу прийняття рішень (Markov Decision Process, MDP), зокрема оглянуто пошук оптимальної стратегії для відомої моделі MDP та відповідні методи навчання з підкріпленням, в тому числі метод адаптивної евристичної оцінки (Adaptive Heuristic Critic), методи навчання з підкріпленням на основі часових різниць (temporal difference learning), методи навчання з підкріпленням SARSA та Q-learning. Розглянуто проблему навчання з підкріпленням в багатоагентних системах (multi-agent systems). Наведено моделі та методи навчання з підкріпленням в багатоагентних системах, зокрема метод навчання розподілу обов'язків між агентами. Розглянуто питання практичного застосування моделей та методів навчання з підкріпленням для побудови автономних інтелектуальних систем.

Метою навчального посібнику є виробити у студентів систематизоване уявлення про основні принципи побудови автономних інтелектуальних систем, надати студентам базові знання про моделі та методи навчання з підкріпленням, а також надати навички практичного застосування методів навчання з підкріпленням для побудови автономних інтелектуальних систем. Використання навчального посібника допоможе студенту досягнути наступних результатів навчання: знати принципи побудови, архітектуру та призначення автономних

інтелектуальних систем; розуміти концептуальні основи та загальні принципи функціонування автономних інтелектуальних систем; знати основні моделі та методи навчання з підкріпленням, а також способи їх використання у складі автономних інтелектуальних систем; мати практичні навички роботи з моделями та методами навчання з підкріпленням; вміти створювати, конфігурувати та налагоджувати автономні інтелектуальні системи з використанням методів навчання з підкріпленням.

Навчальний посібник призначений для студентів спеціальності 123 «Комп'ютерна інженерія» галузі знань 12 «Інформаційні технології», зокрема для студентів другого (магістерського) рівня вищої освіти спеціалізацій 123.01 «Комп'ютерні системи та мережі», 123.02 «Системне програмування» та 123.04 «Кіберфізичні системи». В основу посібника покладено навчальні матеріали, розроблені та підготовлені автором для викладання навчальних дисциплін «Теорія інтелектуальних систем» і «Технології машинного навчання в кіберфізичних системах» на кафедрі електронних обчислювальних машин Національного університету «Львівська політехніка».

1. Автономна інтелектуальна система

1.1. Використання методів штучного інтелекту для побудови інтелектуальних систем

1.1.1. Поняття штучного інтелекту

Штучний інтелект (ШІ) в широкому розумінні це комп'ютерна система наділена розумом (інтелектом), який в деяких аспектах (або в цілому) по своїх можливостях еквівалентний розуму людини чи переважає його. Іншими словами в рамках технологій штучного інтелекту комп'ютерні системи використовуються для того, щоб імітувати здатність людського розуму розв'язувати проблеми та приймати рішення [1-4]. За останні кілька десятиліть з'явилася низка визначень штучного інтелекту. Наприклад, у своїй статті 2004 року видатний американський вчений в цій галузі Джон Маккарті (John McCarthy, 1927- 2011) пропонує таке визначення: «Це наука та техніка створення інтелектуальних машин, особливо інтелектуальних комп'ютерних програм. Це пов'язано з аналогічним завданням використання комп'ютерів для розуміння людського інтелекту, але ШІ не повинен обмежуватись лише методами, які мають аналоги в біологічних системах». Сам термін «Artificial Intelligence» був запропонований Джоном Маккарті зі співавторами у 1956 році (Dartmouth conference, Summer 1956).

Однак за десятиліття до наведеного визначення розмова про штучний інтелект почалася з роботи Алана Тюрінга (Alan Turing, 1912-1954) «Обчислювальна техніка та інтелект» (Computing Machinery and Intelligence) 1950 року [5], в якій він запропонував концепцію штучного інтелекту, експеримент по виявленню здатності машини демонструвати інтелектуальну поведінку та концепцію машинного навчання (machine learning). У цій статті Тюрінг, якого часто називають «батьком інформатики», ставить таке запитання: «Чи можуть машини мислити?» Як варіант відповіді, він пропонує експеримент по виявленню здатності машини демонструвати інтелектуальну поведінку, який тепер відомий як «тест Тюрінга». В цьому експерименті людина проводить опитування і намагається розрізнити з ким вона спілкується з ШІ чи з іншою людиною. За думкою Тюрінга, якщо різницю неможливо побачити, то це каже про наявність у комп'ютерної системи інтелекту. Ще раніше чеський письменник Карел Чапек (Karel Čapek, 1890–1938) у своїй відомій п'єсі «R.U.R.» (Rossumovi univerzální roboti, 1920) висловив ідею робота – штучно створеної автономної інтелектуальної системи подібної до людини. Згодом термін «робот» у цьому значенні став популярним і увійшов до багатьох мов світу.

Автори одного з найкращих підручників з ШІ виділяють чотири підходи до визначення штучного інтелекту з огляду на мету його використання [1]: 1) з точки зору подібності ШІ до людини вони розрізняють: 1.1) комп'ютерні системи, що думають як людина, та 1.2) комп'ютерні системи, що діють як людина, і 2) з точки зору абстрактної ідеї розуму вони розрізняють: 2.1) комп'ютерні системи, що мислять раціонально, 2.2) комп'ютерні системи, що діють раціонально. Відтак визначення ШІ Алана Тюрінга відноситься до підходу «комп'ютерні системи, що діють як людина».

Також з точки зору типів ШІ розрізняють слабкий та сильний ШІ. Слабкий штучний інтелект (weak AI) призначений для виконання окремих прикладних завдань. До слабкого ШІ відноситься більшість систем ШІ, які оточують нас сьогодні. «Спеціалізований» може бути більш точною назвою для цього типу ШІ, оскільки він зовсім не слабкий з точки зору своїх можливостей. Наприклад, він реалізований у персональних асистентах Siri (Apple) або Alexa (Amazon), в потужній діалоговій системі штучного інтелекту IBM Watson, в сучасних автономних транспортних системах, тощо.

Сильний штучний інтелект (strong AI) включає два поняття: загальний штучний інтелект (Artificial General Intelligence, AGI) та штучний суперінтелект (Artificial Super Intelligence, ASI). Загальний ШІ (AGI) – це теоретична форма ШІ, який за своїми можливостями рівний

людському, зокрема має свідомість, здатний вирішувати складні проблеми, вміє вчитися та планувати майбутнє. Штучний суперінтелект (ASI) – це ШІ, який перевершує інтелект людини, в тому числі з огляду на біологічні обмеження людського мозку. Незважаючи на те, що сильний ШІ все ще є предметом лише теоретичних досліджень і не реалізований в сучасних комп'ютерних системах, дослідники технологій ШІ ведуть активну роботу в цьому напрямку. Тим часом цікаві приклади сильного ШІ, можна зустріти у художніх творах наукової фантастики, наприклад, бортовий суперкомп'ютер HAL 9000 у фільмі «Космічна одіссея 2001 року» (2001: A Space Odyssey, 1968) або комп'ютерний суперінтелект Wintermute у романі Вільяма Ґібсона «Нейромант» (Neuromancer, 1984).

З точки зору дослідження та розроблення технологій ШІ розрізняють два основних підходи:

1) Низхідний («семіотичний») підхід: створення складних комп'ютерних систем ШІ, які демонструють високорівневу розумну поведінку.

2) Висхідний («біологічний») підхід: створення простих «цеглинок» (наприклад, штучних нейронів), з яких можна утворити інтелект еквівалентний по своїм можливостям інтелекту людини або такий, що переважає його.

1.1.2. Основні напрямки досліджень в області штучного інтелекту

Різні напрямки досліджень в області ШІ зосереджені навколо вирішення окремих складних проблем та інструментів, що використовуються для їх вирішення. Найбільш базові напрямки досліджень в області штучного інтелекту включають системи логічного виводу (logical reasoning), способи представлення знань, методи пошуку та планування, машинне навчання, обробку природної мови (natural language processing), комп'ютерний зір та системи робототехніки. Окремо стоїть напрямок дослідження сильного ШІ, зокрема загального ШІ, здатного розв'язувати довільну проблему за аналогією до універсальності людського інтелекту. В рамках цих напрямків дослідники штучного інтелекту адаптували та інтегрували широкий спектр інструментів і методик розв'язання задач, включаючи математичні методи пошуку та оптимізації, формальну логіку, штучні нейронні мережі, методи математичної статистики та теорії ймовірностей, моделі і методи теорії ігор, тощо. Дослідження в області ШІ також спираються на наукові та прикладні результати в галузі комп'ютерних наук, психології, лінгвістики, філософії та багатьох інших галузях.

Основні напрямки досліджень в області штучного інтелекту можна згрупувати наступним чином [1]:

1) Методи пошуку (Searching), розв'язок задач задоволення обмежень (Constraint Satisfaction Problems);

2) Знання (Knowledge), логічний вивід (reasoning) та планування (planning), в тому числі використання логіки першого порядку(First-Order Logic), представлення знань (Knowledge Representation) та автоматизоване планування (Automated Planning);

3) Знання з елементами невизначеності (Uncertain knowledge) та ймовірнісний логічний вивід (Probabilistic reasoning), включаючи методи прийняття рішень в багатоагентних системах (Multiagent Decision Making);

4) Машинне навчання (Machine Learning), в тому числі навчання з вчителем (Supervised Learning), навчання з використанням ймовірнісних моделей (Learning Probabilistic Models), глибинне навчання (Deep Learning) та навчання з підкріпленням (Reinforcement Learning);

5) Обробка природної мови (Natural Language Processing), комп'ютерний зір (Computer Vision), робототехніка (Robotics);

6) Філософія, етика та безпека ШІ (Philosophy, Ethics, and Safety of AI).

До окремих напрямків досліджень в області штучного інтелекту можна також віднести:

- Символьний ШІ, в тому числі розробка та використання мови програмування Lisp,
- Логічне програмування, зокрема використання мови програмування PROLOG,
- Нечітка логіка (fuzzy logic),

- Ігрові програми (штучні гравці в шахи і т.п.),
- Експертні системи (бази знань),
- Семантичні мережі та графи знань (knowledge graphs),
- Розпізнавання зображень в системах комп'ютерного зору,
- Методи класифікації, в тому числі методи кластеризації,
- Системи розпізнавання людської мови,
- Машинний переклад та «розуміння» людської мови і текстів,
- Нейрокібернетика і штучні нейронні мережі,
- Використання методів пошуку та оптимізації в задачах з невизначеністю,
- Методи прогнозування та планування подальших дій,
- Ймовірнісні методи планування дій та прийняття рішень,
- Системи ШІ з контекстною залежністю (context awareness),
- Рекомендаційні системи (recommender system) з елементами ШІ,
- Інтегральний підхід на основі інтелектуальних агентів та багатоагентних систем.

1.1.3. Основні концепції штучного інтелекту

Серед основних концепцій, які лежать в основі технологій штучного інтелекту, можна виділити: концепцію автоматизації, подібність до людини, прагматичний підхід, автономність, машинне навчання, багатоагентні системи та самоорганізацію.

Концепція автоматизації передбачає пошук способів замінити людину при розв'язанні тих чи інших задач технікою, в тому числі задля вивільнення людини від тяжкої праці чи вирішення таких задач, які не під силу людині. Завдяки розвитку комп'ютерної техніки та систем автоматичного управління, рутинні, відносно прості операції, що піддаються повній формалізації, порівняно легко автоматизуються, наприклад, з використанням технології Robotic process automation (RAP). Однак зі зростанням складності задач виникає низка проблем (неможливість повної формалізації, надвелика розмірність задачі, нестача інформації про умови розв'язання задачі, тощо), вирішення яких стає можливим лише за рахунок використання методів ШІ.

В рамках концепції подібності до людини, ставиться питання розроблення комп'ютерної системи чи програми, яка була б здатна розв'язувати складні завдання, що потребують людського інтелекту, тобто людина розглядається як своєрідний еталон інтелектуальності. Наприклад, в тесті Тюрінга перевіряється здатність інтелектуальної системи спілкуватись людською мовою, а в області розроблення ігрових програм перевіряється здатність штучних гравців протистояти гравцям-людям. Серед найбільш відомих здобутків ШІ в цьому напрямі: перемога суперкомп'ютера Deep Blue (IBM) над тодішнім чемпіоном світу Гаррі Каспаровим в шаховому матчі (1997), перемога системи штучного інтелекту IBM Watson над чемпіонами популярної телевікторини Jeopardy! (2011), здатність суперкомп'ютера Minwa (Baidu) ідентифікувати та класифікувати зображення з більшою точністю, ніж людина (2015), перемога програми AlphaGo (DeepMind) над чемпіоном світу з гри в го Лі Седолем (Lee Sedol) у матчі з п'яти партій. З точки зору прикладного застосування технологій ШІ, концепція подібності до людини лежить в основі розроблення та використання, так званих, інтелектуальних персональних асистентів (Intelligent Virtual Assistants), зокрема Siri (Apple), Alexa (Amazon), Google Assistant, Cortana (Microsoft), Mycroft (open-source) та ін.

Прагматичний підхід в галузі ШІ виник в результаті критичного аналізу класичних методів прийняття рішення, в тому числі із застосування логічного виводу (reasoning) та інших подібних методів. Зокрема американський вчений-робототехнік Родні Брукс (Rodney Brooks) у своїй класичній роботі «Elephants Don't Play Chess» (Robotics and Autonomous Systems, 6, 1990, pp. 3-15) висловив ідею, що інтелектуальна поведінка це, в першу чергу, ефективна поведінка, тобто така поведінка, яка дозволяє автономній інтелектуальній системі досягти поставленої перед нею мети. Згідно наведеній вище класифікації підходів до визначення ШІ – концепція прагматичного підходу перекликається з визначенням ШІ, як «комп'ютерної

системи, що діє раціонально». Прикладом прагматичного підходу в ШІ є, так звана поведінкова робототехніка (Behavior-based robotics).

Концепція автономності багато в чому перекликається з концепцією автоматизації і передбачає, що інтелектуальна комп'ютерна система спроможна розв'язувати складну задачу самостійно без вказівок або втручання людини. В даному випадку за допомогою методів ШІ створюється своєрідний інтелектуальний «інструмент» (інтелектуальний агент), якому людина делегує певну частину своїх повноважень по прийняттю рішень в процесі вирішення агентом поставленого перед ним завдання. В рамках концепції машинного навчання перед розробниками систем ШІ стоїть завдання забезпечити здатність інтелектуального агенту навчатись чомусь новому, в тому числі на основі прикладів ефективної поведінки, а також шляхом спроб та помилок. Концепція багатоагентних систем (Multi-agent systems) передбачає об'єднання інтелектуальних агентів в єдину систему з метою вирішення деякої складної задачі спільними зусиллями. Принципи колективної поведінки інтелектуальних агентів досліджуються в рамках розподіленого ШІ (Distributed Artificial intelligence), технологій багатоагентних систем та розподіленої робототехніки (Distributed Robotics).

Згідно концепції самоорганізації досліджуються принципи створення окремих інтелектуальних агентів або багатоагентних систем здатних перебудовувати свою структуру задля пристосування до нових умов вирішення поставленої задачі, підвищення ефективності своєї роботи або з метою пошуку розв'язку нової задачі. Процес самоорганізації – об'єднання та взаємодоповнення елементів системи забезпечує функціональну емерджентність (emergence), тобто виникнення у системи якісно нових функцій більш високого рівня, які не зводяться до простої суми функцій елементів. Розробка автономних інтелектуальних систем на основі принципів самоорганізації є одним з найбільш актуальних напрямків розвитку технологій ШІ.

2.1. Автономна інтелектуальна система: основні ідеї та визначення

1.2.1. Поняття автономної інтелектуальної системи

В найбільш широкому розумінні інтелектуальна система (ІС) – це комп'ютерна система с елементами ШІ. В прикладному розумінні – це комп'ютерна система, яка розв'язує деяке складне завдання, яке потребує застосування методів ШІ [1]. Автономна інтелектуальна система розв'язує поставлене перед нею завдання самостійно, без участі людини, і, як правило, в процесі розв'язку завдання взаємодіє з деяким середовищем (об'єктом управління) (рис.1.1). Згідно сучасній термінології в галузі ШІ автономні інтелектуальні системи називаються інтелектуальними агентами (intelligent agents) або просто агентами (agents). Застосування автономних інтелектуальних систем, як правило, передбачає делегування їм людиною частини своїх повноважень по прийняттю рішень.

Використання автономних інтелектуальних систем є доцільним в складних задачах, в яких присутня невизначеність, наприклад у вигляді нестачі інформації про наслідки вибраного управління (в який стан перейде об'єкт управління внаслідок реалізації даного управління?). При цьому розрізняють два основних типи невизначеності: 1) невизначеність щодо наступного стану об'єкту управління («байдужий» об'єкт управління); 2) невизначеність цілеспрямованої протидії («зацікавлений» об'єкт управління, який має протилежні цілі).

Базові характеристики автономних інтелектуальних систем включають:

- автономність: здатність діяти самостійно, без втручання людини та здатність діяти за власною ініціативою (в межах повноважень, які делеговані системі людиною);
- цілеспрямованість: здатність досягати поставленої мети, здатність оцінювати поточну відстань до мети та приймати і реалізовувати рішення, які наближають до мети;
- адаптивність: здатність адаптуватись до змін у оточуючому середовищі, здатність до самонавчання (здатність накопичувати та враховувати попередній досвід), здатність цілеспрямовано вивчати (досліджувати) своє оточення, тобто долати нестачу інформації про своє оточення), здатність змінювати себе та створювати подібні системи;
- здатність взаємодіяти (діяти спільно та узгоджено) з іншими подібними системами та людиною.

Рис. 1.1. Схема взаємодії автономної інтелектуальної системи з середовищем.

1.2.2. Узагальнена функціональна схема автономної інтелектуальної системи

Узагальнена функціональна схема автономної інтелектуальної системи вміщує наступні елементи (рис.1.2):

1. $P(E,a)$ – сенсорна під-система; відповідає за визначення стану середовища (об'єкту управління) у вигляді значень набору параметрів, що вимірюються відповідними сенсорами.

2. $R(s,a)$ – блок оцінки; формує сигнал програшу або виграшу (відгук середовища) в залежності від визначеного стану середовища, інформації про попередні стани та інформації, отриманої з блоку інформаційної взаємодії.

3. $U(a)$ – блок прийняття рішень; забезпечує самостійний вибір рішень в умовах нестачі інформації (невизначеності) заданого типу (наприклад, змодельованої у вигляді стаціонарного випадкового середовища).

4. $D(a)$ – виконавча система; відповідає за виконання прийнятих рішень (наприклад, переміщення у просторі).

5. $C(a)$ – блок інформаційної взаємодії з іншими інтелектуальними системами; забезпечує обмін інформацією заданого змісту та формату з іншими інтелектуальними системами.

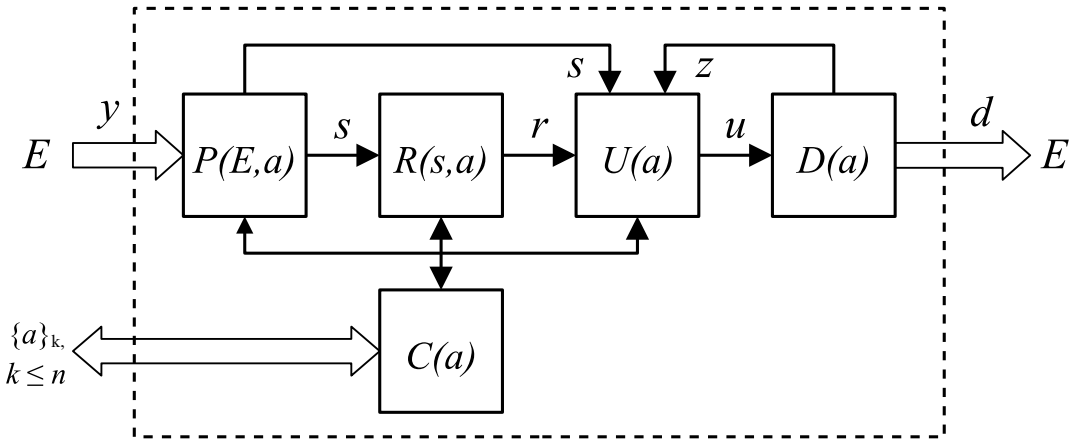


Рис.1.2. Узагальнена функціональна схема автономної інтелектуальної системи.

На узагальненій функціональній схемі автономної інтелектуальної системи (рис.1.2) використано наступні позначення: E – середовище (об'єкт управління); a – агент; y – значення параметрів середовища; s – стан середовища; r – оцінка успішності дій агенту на кроці t; u – рішення прийняте агентом на кроці t (сигнал управління); d – дія агенту згідно прийнятого рішення (вплив на середовище); z – інформація про стан виконання обраної дії.

Цільова функція агенту f(a) обумовлює вибір його функціональних компонент:

$$f(a) \rightarrow [\ P(E,a),\ R(s,a),\ C(a),\ U(a),\ D(a)\ ].$$

З точки зору підходів до реалізації блоку оцінки та блоку прийняття рішень, розрізняють наступні рівні складності автономної інтелектуальної системи (агенту) [1] (в порядку зростання складності):

1) Простий рефлексивний агент (simple reflex agent),

2) Рефлексивний агент з моделлю середовища (model-based reflex agent) - рефлексивний агент, що використовує модель середовища (об'єкта управління),

3) Агент орієнтований на досягнення мети (goal-based agent) - агент, який намагається досягнути поставлену перед ним мету,

4) Агент орієнтований на максимізацію цільової функції (utility-based agent) - агент, який намагається максимізувати значення заданої цільової функції (функції корисності).

Простий рефлексивний агент (simple reflex agent)вирізняється тим, що він

- обирає дії на основі поточної інформації про стан середовища s(t) (current percept), не зважаючи на інформацію про попередні стани;
- використовує для прийняття рішень правила «умова-дія» (condition-action rules) вигляду: If s(t)=S2 then u=D9.

Рефлексивний агент з моделлю середовища (model-based reflex agent) вирізняється тим, що він

- зберігає у пам'яті деякий «внутрішній стан» (internal state), який залежить від інформації про попередні стани середовища (percept history);
- здатний відслідковувати частину оточення, про яку він не має інформації (яку він не «бачить») в даний момент часу;
- використовує модель середовища, яка відображає: 1) зміни у середовищі, які відбуваються незалежно від дій агенту; 2) результати впливу дій агенту на середовище.

Агент орієнтований на досягнення мети (goal-based agent) вирізняється тим, що він

- використовує інформацію про «бажані»/вигідні ситуації або стани середовища, тобто інформацію про поставлену мету (goal information);
- зіставляє інформацію про мету з моделлю середовища для вибору дій, які наближають його до мети чи відразу призводять до її досягнення (може обиратись одна дія або послідовність дій);
- використовує методи пошуку (search) та планування (planning), тобто методи ШІ для знаходження деякої послідовності дій для досягнення поставленої мети (наприклад, у вигляді шляху у графі);
- при прийнятті рішення бере до уваги майбутнє (шлях до мети), що відрізняє його від рефлексивного агенту.

Агент орієнтований на максимізацію цільової функції (utility-based agent), тобто агент, який намагається максимізувати значення заданої цільової функції (функції корисності), вирізняється тим, що він

- використовує корисність (utility) - більш загальну інформація про успішність дій агентів ніж інформація про мету (яку в найпростішому випадку можна розглядати як «бінарну» інформацію: мета досягнута - так/ні);
- використовує цільову функцію (utility function), яка кожній послідовності дій і відповідним станам середовища ставить у відповідність величину, за якою можна розрізняти їх успішність (наприклад, можна вирізняти більш та менш вигідні шляхи до поставленої мети);
- обирає дії в такий спосіб, щоб максимізувати очікувану корисність послідовності обраних дій (очікуване значення цільової функції).

1.2.3. Застосування автономних інтелектуальних систем

Дослідження, розробка та впровадження автономних інтелектуальних систем є одним з основних способів подолання кризи у сфері розробки сучасних надскладних та віддалених комп'ютерних систем, зокрема автономних розподілених систем різного призначення. Криза у сфері розробки надскладних та віддалених комп'ютерних систем має наступні основні аспекти.

1) Подальше зростання складності комп'ютерних систем та мереж призводить до того, що процеси в них стають подібними до природних явищ, тобто системи стають настільки складними, що

1.1) потребують значних витрат на підтримку їх у робочому стані (в багатьох випадках поточні витрати на підтримку складної системи у робочому стані перевищують «одноразові» витрати на створення такої системи);

1.2) подальше ускладнення системи стає принципово неможливим;

1.3) різко знижується надійність та живучість системи.

2) Зростає число ситуацій та режимів роботи, коли людина не може приймати безпосередню участь в управлінні системою. Це зокрема

1.1) небезпечні середовища (зони радіоактивного або хімічного забруднення, зони надвисоких або наднизьких температур),

1.2) недоступні середовища, тобто середовища до яких людині важко або взагалі неможливо потрапити (глибини Світового Океану, космічний простір, інші планети, мікропростори),

1.3) надшвидкі середовища, в яких швидкість змін набагато перевищує природні можливості людини («обчислювальні» середовища: операційні середовища комп'ютерних систем, процеси в комп'ютерних мережах, віртуальні середовища, технологічні процеси, наукові експерименти, тощо).

Серед прикладів областей застосування інтелектуальних систем відзначимо:

- автономні системи моніторингу та управління комп'ютерними системами та мережами (Autonomic Computing, IBM);
- штучні інтелектуальні персональні асистенти та системи управління персональною інформацією;
- інтелектуальні агенти в комп'ютерних іграх (штучні гравці, боти та ін.);
- інтелектуальні системи біржових торгів (Algorithmic trading);
- робототехнічні системи: промислові роботи, андроїди, «спортивні» змагання роботів, поліморфні робототехнічні системи (polymorphic robotics), системи розподіленої робототехніки (колективи роботів) і т.п.;
- автономні транспортні системи: intelligent transportation system, self-driving car (autonomous car), vehicular ad hoc network (VANET);
- системи автономних розподілених досліджень, наприклад, Autonomous Exploration for Gathering Increased Science System (AEGIS), NASA; The integrated Arctic Ocean Observing System (iAOOS);
- автономні системи озброєнь.

Контрольні питання

1. Що таке штучний інтелект?
2. Чим дослідження в області штучного інтелекту відрізняються від інших областей комп’ютерної техніки та інформаційних технологій?
3. Для чого використовується тест Тюрінга?
4. Які напрямки наукових досліджень відносяться до області штучного інтелекту?
5. Які базові характеристики відрізняють автономні інтелектуальні системи від інших систем?
6. Які основні блоки входять до узагальненої функціональної схеми автономної інтелектуальної системи?
7. Яка властивість автономної інтелектуальної системи забезпечує її здатність діяти самостійно та за власною ініціативою?
8. Яка властивість автономної інтелектуальної системи забезпечує її здатність пристосовуватись до змін у оточуючому середовищі чи об’єкті управління?
9. В який спосіб обирає наступну дію простий рефлексивний агент (simple reflex agent)?
10. Чим рефлексивний агент з використанням моделі середовища (model-based reflex agent) відрізняється від простого рефлексивного агенту (simple reflex agent)?
11. Чим агент орієнтований на досягнення мети (goal-based agent) відрізняється від агенту орієнтованого на максимізацію цільової функції (utility-based agent)?
12. Який з перерахованих типів агентів (simple reflex agent, model-based reflex agent, goal-based agent, utility-based agent) є найбільш простим в реалізації?
13. В чому полягає специфіка областей застосування автономних інтелектуальних систем?
14. В яких основних областях застосовуються автономні інтелектуальні системи?
15. В яких областях практичного використання автономних інтелектуальних систем застосовуються програмні агенти (software agents)?

2. Автомати, що навчаються (learning automata)

2.1. Моделювання простих форм цілеспрямованої поведінки

2.1.1. Стаціонарне випадкове середовище

Стаціонарне випадкове середовище (binary multi-armed bandit, binary MAB) задається у вигляді:

$$E = \{p^+(d_1), p^+(d_2), \dots, p^+(d_n)\},$$

де

$D = \{d_1, d_2, ..., d_n\}$ – множина дій, які агент може здійснювати у середовищі E (n - загальна кількість дій),

$p^+(d_i)$ – ймовірність того, що агент отримає виграш, якщо він здійснить дію d_i.

Агент взаємодіє з стаціонарним випадковим середовищем (СВС) за наступною схемою:

1) на кожному кроці взаємодії з середовищем агент обирає і здійснює дію d(t) (одну з n дій);

2) у відповідь на це середовище повертає агенту сигнал про виграш/програш (відгук середовища) r(t).

В найбільш простому випадку усі можливі наслідки здійснення агентом дій у середовищі можна віднести до одного з двох класів: виграшні (позитивні) та програшні (негативні). Сигнал про виграш (далі просто виграш) на кроці взаємодії t позначимо як $r(t) = r^+ = 1$, і відповідно програш позначимо як $r(t) = r^- = 0$. Відмітимо, що в деяких джерелах розглядається інший варіант значень виграшу та програшу: $r(t) = r^+ = +1$, $(t) = r^- = -1$. Оскільки функція виграшу (reward function) R:d→r може приймати лише два значення (r^+, r^-), ймовірності отримання виграшу $p^+(d)$ та програшу $p^-(d)$ за здійснення дії d в сумі дорівнюють одиниці: $p^+(d) + p^-(d) = 1$.

Таким чином у стаціонарному випадковому середовищі результат r(t), який отримає агент у відповідь на здійснення дії d задається дискретним розподілом ймовірностей

$$P(d) = \{p^+(d); p^-(d)\}.$$

При тому для кожної дії d визначено свій, відмінній від інших дій розподіл ймовірностей.

Слово “випадкове” у назві СВС відображає той факт, що середовище повертає виграш або програш у відповідь на дію агента з деякою ймовірністю. Слово “стаціонарне” у назві СВС відображає той факт, що ймовірності $\{p^+(d); p^-(d)\}$ не змінюються у часі, тобто залишаються тими самими на всіх кроках взаємодії агента з середовищем. З точки зору концепції станів стаціонарне випадкове середовище завжди знаходиться в одному незмінному стані.

Нижче наведено приклади двох СВС на три та чотири дії відповідно:

1) Приклад 1: $E = \{0.2; 0.7; 0.4\}$, n=3

$d_1 \to r^+$: $p^+(d_1) = \mathbf{0.2}$; $d_1 \to r^-$: $p^-(d_1) = 1 - p^+(d_1) = \mathbf{0.8}$;
$d_2 \to r^+$: $p^+(d_2) = \mathbf{0.7}$; $d_2 \to r^-$: $p^-(d_2) = 1 - p^+(d_2) = \mathbf{0.3}$;
$d_3 \to r^+$: $p^+(d_3) = \mathbf{0.4}$; $d_3 \to r^-$: $p^-(d_3) = 1 - p^+(d_3) = \mathbf{0.6}$;

2) Приклад 2: $E = \{0.8; 0.3; 0.2; 0.6\}$, n=4

$d_1 \to r^+$: $p^+(d_1) = \mathbf{0.8}$; $d_1 \to r^-$: $p^-(d_1) = 1 - p^+(d_1) = \mathbf{0.2}$;
$d_2 \to r^+$: $p^+(d_2) = \mathbf{0.3}$; $d_2 \to r^-$: $p^-(d_2) = 1 - p^+(d_2) = \mathbf{0.7}$;
$d_3 \to r^+$: $p^+(d_3) = \mathbf{0.2}$; $d_3 \to r^-$: $p^-(d_3) = 1 - p^+(d_3) = \mathbf{0.8}$;
$d_4 \to r^+$: $p^+(d_4) = \mathbf{0.6}$; $d_4 \to r^-$: $p^-(d_4) = 1 - p^+(d_4) = \mathbf{0.4}$.

Розглянемо також приклад опису параметрів СВС для випадку, коли використовується схема виграшів r(t)={-1;+1}. Нехай це стаціонарне випадкове середовище E_1 на дві дії: $D = \{d_1,$

$d_2\}$, в якому дія d_1 з ймовірністю $q_1=0.8$ приносить виграш та з ймовірністю $p_1=0.2$ приносить програш, і тоді $a_1=q_1–p_1=0.8–0.2=0.6$ – це середній виграш за дію d_1, а дія d_2 з ймовірністю $q_2=0.4$ приносить виграш та з ймовірністю $p_2=0.6$ приносить програш, і тоді $a_2=q_2–p_2=0.4–0.6=–0.2$ – це середній виграш за дію d_2. Відтак середовище E_1 можна задати двома числами: $E_1=\{a_1;a_2\}=\{0.6;–0.2\}$.

Розглянемо наступні два приклади змістовної інтерпретації стаціонарного випадкового середовища.

1) Рекомендаційна система (recommendation system), в якій середовище (об'єкт управління) – це користувач; дія агенту d – це інформаційний об'єкт (новина, пісня, відеокліп, тощо), який рекомендується користувачу для перегляду; виграш/програш – це оцінка, яку виставляє користувач рекомендованому інформаційному об'єкту; $p^+(d)$ – це ймовірність, з якою користувачу сподобається інформаційний об'єкт d.

2) Автономна транспортна система (autonomous transportation system), в якій середовище – це дорожня мережа; дія агенту d – це маршрут (один з декількох можливих), який обирає автономний транспортний засіб; виграш (програш) нараховується, якщо час подолання маршруту менший (більший) заданого порогового значення; $p^+(d)$ – це ймовірність, з якою автономний транспортний засіб потрапить у дорожній затор, якщо він обере маршрут d.

Приклад програмної реалізації стаціонарного випадкового середовища для проведення обчислювальних експериментів по дослідженню методів навчання з підкріпленням наведено в додатку Д.1.

2.1.2. Модель цілеспрямованої поведінки

Розглянемо просту модель цілеспрямованої поведінки, в якій агент A розміщується в деякому середовищі E. Його дії d(t) викликають у відповідь реакції середовища r(t), які в свою чергу поступають на вхід агенту A. При цьому агент використовує їх для вибору рішення про свої наступні дії. В найпростішому випадку будемо вважати, що всі можливі реакції (відгуки) середовища, які сприймає агент, він відносить до одного з двох класів – сприятливих або несприятливих реакцій. Ці класи будемо називати виграшами (r^+) та програшами ($r^–$). Будемо вважати, що всередині цих двох класів реакції середовища агент не розрізняє. Будемо також вважати, що цілеспрямованість поведінки агенту полягає у збільшені кількості сприятливих і зменшенні кількості несприятливих реакцій середовища.

В якості моделі об'єкта управління (середовища) будемо розглядати стаціонарне випадкове середовище. В даному випадку невизначеність (нестача інформації про середовище) обумовлена тим, що агенту невідомі характеристики середовища, тобто розподіли ймовірностей $P(d) = \{p^+(d); p^–(d)\}$ для дій $D=\{d\}$. Перед агентом стоїть завдання самостійно знайти рішення у вигляді дії d*, здійснення якої найчастіше призводить до сприятливих реакцій середовища (виграшів). Відтак виникає питання: як оцінити загальну ефективність усіх минулих дій агента на момент часу t?

Для цього будемо використовувати цільову функцію (utility function), як спосіб оцінки ефективності дій агента (загального виграшу на усіх попередніх кроках) на момент часу t. Таким чином мета агента – обирати дії в такий спосіб, щоб максимізувати значення цільової функції. Варіанти цільової функції включають:

1) значення відгуку r(t) на останньому кроці взаємодії T;

2) найбільше значення відгуку r(t) у проміжку часу [0;T];

3) сума $k < T$ останніх відгуків у проміжку часу [0;T];

4) сума усіх відгуків r(t) у проміжку часу [0;T];

5) середнє значення відгуку, яке припадає на один крок взаємодії.

Найбільш часто, особливо при дослідженні методів навчання з підкріпленням, використовуються наступні варіанти цільової функції:

1) сумарний виграш $W_1(t) = \Sigma r(t)$, $t=0,...,T$,

2) середній виграш $W_2(t) = 1/t * \Sigma r(t)$, $t=0,...,T$.

Для визначення того, що поведінка агенту є цілеспрямованою, її порівнюють:

1) з поведінкою агенту $\mathbf{A_R}$, який здійснює рівноймовірний вибір дій, тобто обирає дію d з ймовірністю 1/n, де n - кількість дій в множині доступних для вибору дій *D*,

2) з поведінкою агенту $\mathbf{A_P}$, який від початку обирає найкращу дію d* з множини дій *D*, тобто дію з найбільшою ймовірністю виграшу серед усіх інших дій: $d^*=\text{argmax}\{p^+(d)\}$, $d \in D$.

Розглянемо приклад розрахунку значення цільової функції для агентів $\mathbf{A_R}$ та $\mathbf{A_P}$ у стаціонарному випадковому середовищі $E = \{0.8; 0.3; 0.2; 0.6\}$, n=4:

$\mathbf{A_R}$, $t\to\infty$: $W2(t) = 0.25\cdot0.8 + 0.25\cdot0.3 + 0.25\cdot0.2 + 0.25\cdot0.6 =$
$= 0.2 + 0.075 + 0.05 + 0.15 =$ **0.475**

$\mathbf{A_P}$, $t\to\infty$: $W2(t) = 1\cdot0.8 + 0\cdot0.3 + 0\cdot0.2 + 0\cdot0.6 =$ **0.8**.

Цілеспрямованою будемо вважати таку поведінку, яка в довільному стаціонарному випадковому середовищі є кращою за поведінку агента $\mathbf{A_R}$. Разом з тим цілеспрямована поведінка є тим кращою, чим ближче вона до поведінки агента $\mathbf{A_P}$. Відтак для розв'язання проблеми розробки агента з цілеспрямованою поведінкою у стаціонарному випадковому середовищі потрібно відповісти на питання: в який спосіб відображати історію взаємодії агента з середовищем $\{d(t)\to r(t)\}$ на попередніх кроках у наступну дію d(t+1)?

Ще раз відзначимо, що невизначеність (нестача інформації) в даному випадку полягає в тому, що агенту невідомі наперед ймовірнісні характеристики середовища, в якому він розміщений (тобто характеристики об'єкту управління, з яким він взаємодіє).

Розглянемо найпростіший випадок моделі цілеспрямованої поведінки з СВС на дві дії $D=\{d_1,d_2\}$,n=2 та двома типами відгуків середовища (виграш/програш) $R=\{r^+,r^-\}$, $r^+=1,r^-=0$. Зауважимо, що чим ближчі за значенням ймовірності виграшу для різних дій $\{p^+(d_1); p^+(d_2)\}$, тим складнішою є задача вибору найкращої дії. Для наведеного найпростішого випадку моделі можна розглядати чотири типи стаціонарного випадкового середовища з точки зору конфігурації його параметрів:

1) $A = \{p^+(d_1)<0.5, p^+(d_2)<0.5\}$,
2) $B = \{p^+(d_1)>0.5, p^+(d_2)>0.5\}$,
3) $C = \{p^+(d_1)<0.5, p^+(d_2)>0.5\}$,
4) $D = \{p^+(d_1)>0.5, p^+(d_2)<0.5\}$.

Зауважимо, що середовища типу C і D є легшими з точки зору задачі вибору дії ніж середовища типу A і B.

Приклад програмної реалізації розглянутої моделі цілеспрямованої поведінки для проведення обчислювальних експериментів по дослідженню методів навчання з підкріпленням наведено в додатку Д.3.

2.1.3. Конструкції автоматів, що навчаються

Наукові дослідження в області автоматів, що навчаються (learning automata) розпочав у 60-ті роки 20-го століття видатний вчений-кібернетик Михайло Львович Цетлін [8]. В основу цих досліджень покладено ідею М.Л. Цетліна про простоту (в рамках висхідного підходу):

1) найпростіша "цеглинка" – це цифровий автомат, який демонструє цілеспрямовану поведінку в стаціонарному випадковому середовищі;

2) з таких "цеглинок" можна побудувати систему будь-якої потрібної складності, яка також буде демонструвати цілеспрямовану поведінку.

З точки зору узагальненої функціональної схеми автономної інтелектуальної системи (рис.1.2), автомат, що навчається, (Learning Automaton, LA), або деяка комбінація таких автоматів реалізують блок прийняття рішень $U(a)$ агенту орієнтованого на максимізацію цільової функції (utility-based agent).

Найпростіший варіант автомату, що навчається, – це цифровий автомат, що демонструє цілеспрямовану поведінку в стаціонарному випадковому середовищі [8-10]. На вхід LA подається один з двох сигналів r(t):

$$R=\{r^+,r^-\},$$

на виході LA - дія d з множини *D* (алфавіту доступних LA дій). Розглянемо основні конструкції автоматів, що навчаються, які демонструють цілеспрямовану поведінку у стаціонарному випадковому середовищі.

Автомат з лінійною тактикою (автомат М.Л. Цетліна) вирізняється тим, що:

- кожній дії d відповідає своя «гілка» станів автомату;
- кількість станів у одній «гілці» називається глибиною пам'яті автомату (m);
- при надходженні сигналу виграшу $r(t) = r^+$ відбувається перехід зі стану φ_i у стан φ_{i+1}; якщо $\varphi_i = m$ (останній стан в «гілці»), то автомат залишається в цьому стані (рис.2.1);
- при надходженні сигналу програшу $r(t) = r^-$ відбувається перехід зі стану φ_i у стан φ_{i-1}; якщо $\varphi_i=1$ (перший стан в «гілці»), то автомат переходить в стан 1 «гілки» іншої дії (наступної за порядком), тобто перемикається на виконання іншої дії (рис.2.1);
- автомат виконує ту дію d, в «гілці» якої він знаходиться на даний момент (рис.2.2).

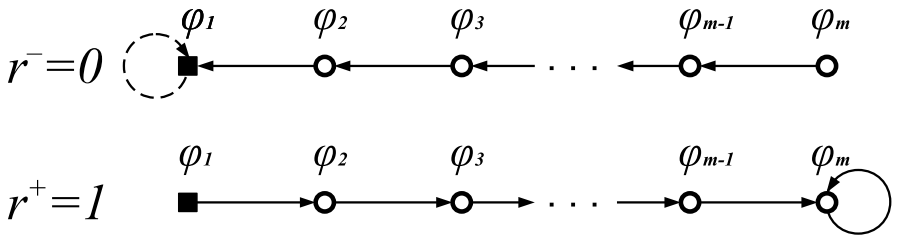


Рис. 2.1. Граф зміни станів LA з лінійною тактикою.

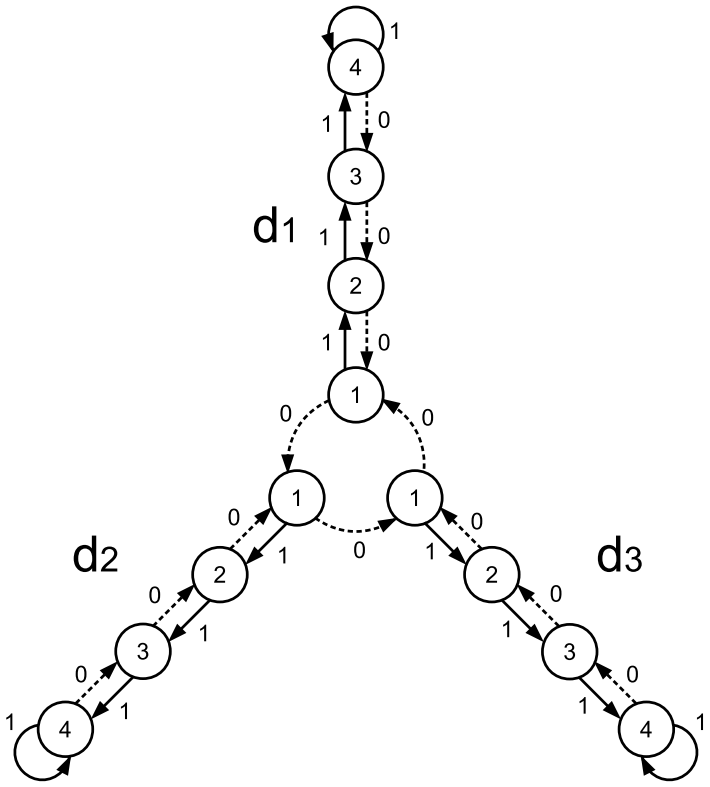


Рис. 2.2. Приклад LA з лінійною тактикою на три дії:
d_1, d_2, d_3 з глибиною пам'яті m=4.

Автомат В.І. Крінського («довірливий» автомат) вирізняється тим, що:

- при надходженні сигналу виграшу $r(t) = r^+$ відбувається перехід зі стану φ_i відразу в останній стан у «гілці» (тому цей автомат називають «довірливим»);

- при надходженні сигналу програшу $r(t) = r^-$ переходи між станами відбуваються так само як в LA з лінійною тактикою;

- граф зміни станів LA В.І. Крінського (рис.2.3.) відрізняється від графу станів LA з лінійною тактикою в частині переходів між станами під впливом сигналів виграшу.

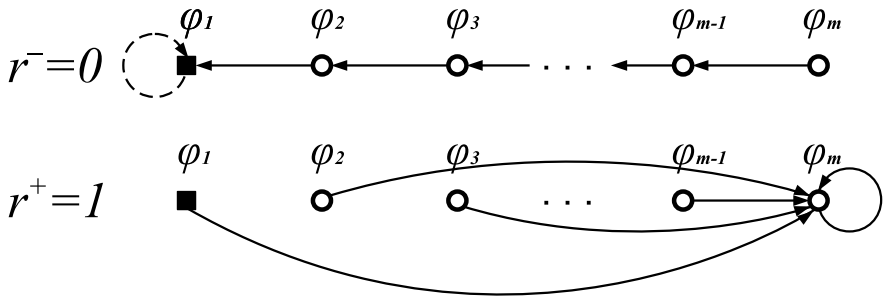


Рис. 2.3. Граф зміни станів LA В.І. Крінського.

Автомат Г. Робінса («інерційний» автомат) вирізняється тим, що:

- при надходженні сигналу виграшу $r(t) = r^+$ відбувається перехід зі стану φ_i відразу в останній стан у «гілці» (так само, як в LA В.І. Крінського);

- при надходженні сигналу програшу $r(t) = r^-$ відбувається перехід зі стану φ_i у стан φ_{i-1}; якщо $\varphi_i=1$ (перший стан в «гілці»), то автомат переходить в останній стан «гілки» іншої дії (тому цей автомат називають «інерційним»);

- граф зміни станів LA Г. Робінса (рис.2.4) в частині переходів між станами під впливом сигналів виграшу виглядає так само як граф зміни станів LA В.І. Крінського.

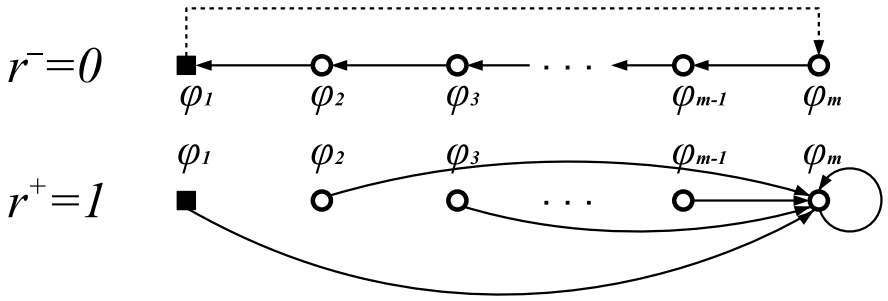


Рис. 2.4. Граф зміни станів LA Г. Робінса.

Автомат В.Ю. Крилова («обережний» автомат) вирізняється тим, що:

- при надходженні сигналу виграшу $r(t) = r^+$ переходи між станами відбуваються так само як в LA з лінійною тактикою;

- при надходженні сигналу програшу $r(t) = r^-$ з ймовірністю 1/2 відбувається перехід зі стану φ_i у стан φ_{i-1} або у стан φ_{i+1} (тому цей автомат називають «обережним»); якщо $\varphi_i=1$ (перший стан в «гілці»), то автомат з ймовірністю 1/2 переходить в стан 1 «гілки» іншої дії або в стан 2 даної «гілки»;

- граф зміни станів LA В.Ю. Крилова (рис.2.5) містить ймовірнісні переходи між станами.

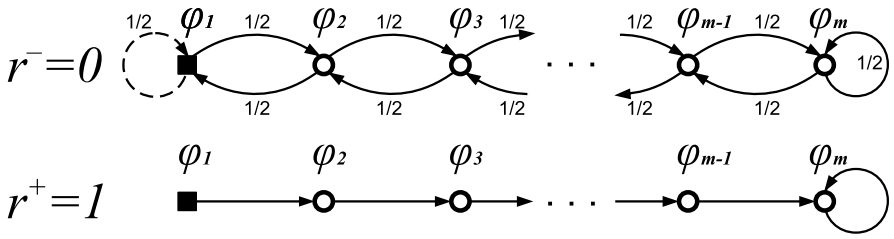
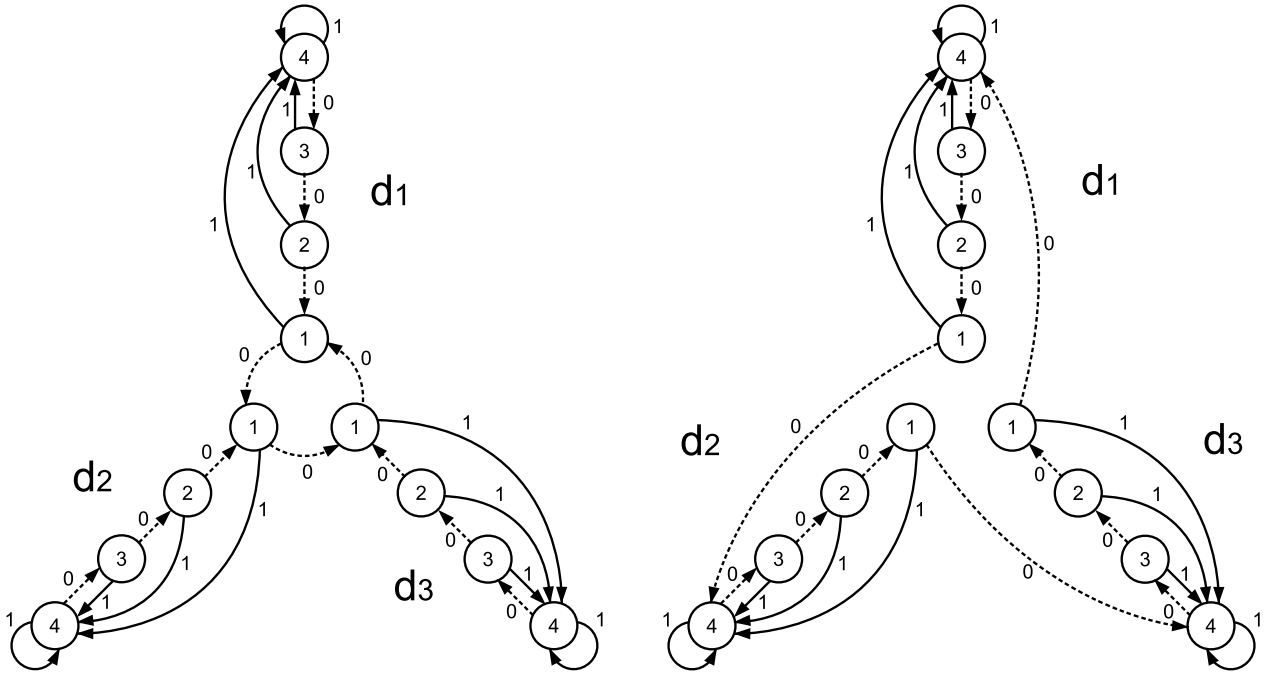


Рис. 2.5. Граф зміни станів LA В. Ю. Крилова.



а) б)

Рис. 2.6. Приклад схеми переходів: а) LA В.І. Крінського; б) LA Г. Робінса на три дії: d_1,d_2,d_3, з глибиною пам'яті m=4.

2.1.4. Асимптотично-оптимальні послідовності симетричних автоматів, що навчаються

Для оцінки «складності» автомату, що навчається, використовується поняття глибини пам'яті автомату. Глибина пам'яті автомату, що навчається, – основний параметр його конструкції, який визначає кількість станів в окремій «гілці» автомату (кожна гілка відповідає дії, яку може обрати автомат). Встановлено, що для випадку стаціонарного випадкового середовища: чим більше глибина пам'яті m, тим ближче поведінка автомату, що навчається, до оптимальної.

Усі розглянуті вище конструкції автоматів, що навчаються демонструють цілеспрямовану поведінку в стаціонарному випадковому середовищі. Тобто розглянуті конструкції гарантують, що LA методом спроб та помилок з часом знайде найкращу дію d* з максимальним значенням ймовірності виграшу $p^+(d)$ серед усіх інших дій.

Під симетричними LA розуміють LA з однаковою кількістю станів у «гілках» усіх дій. Асимптотично-оптимальна послідовність симетричних LA – це послідовність LA однакової конструкції із зростаючим значенням глибини пам'яті m = 1,2,3,... LA демонструє тим кращу поведінку в стаціонарному випадковому середовищі, чим більше його глибина пам'яті m. Якщо глибина пам'яті прямує до нескінченності ($m \to \infty$), то поведінка LA наближається до поведінки агента $\mathbf{A_P}$, тобто відповідна послідовність симетричних автоматів є асимптотично оптимальною.

Число станів LA (глибина пам’яті m) виступає кількісною оцінкою його «складності». Чим більше глибина пам’яті m, тим ближче поведінка LA до оптимальної, тобто, чим «складніше» LA, тим більш цілеспрямованою є його поведінка в стаціонарному випадковому середовищі. На основі цього можна сформулювати наступний базовий принцип побудови автономних інтелектуальних систем: для досягнення більшої ефективності цілеспрямованої поведінки потрібно збільшувати пам’ять та/або складність інтелектуальної системи, тобто пам’ять є необхідною умовою наявності інтелекту. З точки зору апаратної реалізації конструкцій LA, один з найпростіших варіантів – це використання регістра зі зсувом. Приклади програмної реалізації розглянутих конструкцій LA наведено в додатку Д.4.

М.Л. Цетлін сформулював цікаве питання [8]: яку частку від усіх можливих конструкцій автоматів (при деяких розумних обмеженнях) складають конструкції автоматів, що навчаються, тобто такі які демонструють цілеспрямовану поведінку у стаціонарному випадковому середовищі? Це питання на даний час лишається відкритим.

Ще одним цікавим питанням є порівняння поведінки людини та LA у СВС. Досліди показали, що людина поводить себе в СВС гірше ніж LA, особливо, коли ймовірності виграшу для різних дій мають близькі значення. В цьому плані людину можна прирівняти до LA з невеликою глибиною пам’яті. Особливість людської поведінки у СВС полягає в тому, що вже після процесу навчання (тобто після визначення того, яка дія є найкращою) людина час від часу перевіряє дії з меншою ймовірністю виграшу. Щурі в умовах досліду у Т-лабіринті (T-maze), який є аналогом СВС з двома діями (n=2), діють зі значно більшою глибиною пам’яті і в цьому відношенні переважають людину. В СВС з близькими ймовірностями виграшів для різних дій (середовища типу А і В у розглянутому вище прикладі) найкращі результати демонструють «беземоційні» LA.

2.2. Поведінка автоматів, що навчаються, у випадкових середовищах з перемиканням станів

2.2.1. Випадкове середовище з перемиканням станів

Випадкове середовище з перемиканням станів (ВСПС) – це набір стаціонарних випадкових середовищ (кожне з яких відповідає окремому стану), перемикання між якими відбувається випадково:

$$E = (\{E_1, E_2, \ldots, E_k\}, T(s,s')),$$

де

$E_i = \{p^+(s_i,d_1), p^+(s_i,d_2), ..., p^+(s_i,d_n)\}$ – i-те стаціонарне випадкове середовище, яке відповідає стану середовища s_i;

$D = \{d_1, d_2,..., d_n\}$ – множина дій, які агент може здійснювати у середовищі E (n – загальна кількість дій),

$p^+(s_i,d_j)$ – ймовірність того, що агент отримає виграш, якщо він здійснить дію d_j в той час як середовище знаходиться у стані s_i.

T(s,s') – випадкова функція перемикання станів середовища (переходів між станами), яка для кожної пари станів визначає ймовірність перемикання (переходу): p(s,s').

Агент взаємодіє з випадковим середовищем з перемиканням станів за наступною схемою:

1) на кожному кроці взаємодії з середовищем агент обирає і здійснює дію d(t) (одну з n дій); при цьому середовище знаходиться в стані s(t), «номер» якого агенту невідомий;

2) у відповідь на це середовище повертає агенту сигнал про виграш/програш (відгук середовища) r(t), який визначається згідно дискретного розподілу ймовірностей $P(s,d) = \{p^+(s,d); p^-(s,d)\}$ (різного для різних станів);

3) на кожному кроці середовище переходить зі стану s(t) у деякий інший стан s(t+1) згідно дискретного розподілу ймовірностей: P(s)={p(s,s')}, який задається функцією T(s,s').

Зазначимо, що ВСПС у порівнянні з СВС є більш складною моделлю об'єкту управління, оскільки в ньому присутній додатковий фактор невизначеності – випадкове перемикання станів середовища, тобто СВС з різними функціями виграшу.

Розглянемо два приклади випадкового середовища з перемиканням станів.

Приклад 1. ВСПС на дві дії: $D = \{d_1, d_2\}$, n=2, яке складається з трьох стаціонарних випадкових середовищ

$$E_{2,3} = \{E_1, E_2, E_3, \Delta\}, k=3,$$

де

$E_1 = \{0.3, 0.9\}$, тобто $p^+(s_1,d_1)=0.3$, $p^+(s_1,d_2)=0.9$,

$E_2 = \{0.4, 0.7\}$, тобто $p^+(s_2,d_1)=0.4$, $p^+(s_2,d_2)=0.7$,

$E_3 = \{0.8, 0.3\}$, тобто $p^+(s_3,d_1)=0.8$, $p^+(s_3,d_2)=0.3$,

Δ – матриця ймовірностей переходу p(s,s') з одного стану в інший (3x3), яка задає функцію перемикання станів середовища:

$$T(s,s'): \Delta = \{\{0.8, 0.1, 0.1\},\{0.5, 0.3, 0.2\},\{0.1, 0.3, 0.6\}\},$$

де $p(s_i,s_1) + p(s_i,s_2) + p(s_i,s_3) = 1$.

Приклад 2. Найбільш простий варіант ВСПС, яке складається з двох СВС на дві дії $D=\{d_1, d_2\}$, n=2, можна представити у параметризованій формі:

$$E_{2,2} = \{E_1, E_2, \Delta\}, k=2,$$

де

$E_1 = \{x; 1-x\}$, $x\in[0;1]$, тобто $p^+(s_1,d_1)=x$, $p^+(s_1,d_2)=1-x$,

$E_2 = \{1-x; x\}$, $x\in[0;1]$, тобто $p^+(s_2,d_1)=1-x$, $p^+(s_2,d_2)=x$,

наприклад, для x=0.8 отримаємо таку конфігурацію двох СВС у складі ВСПС:

E_1: $p^+(s_1,d_1)=0.8$, $p^+(s_1,d_2)=0.2$

E_2: $p^+(s_2,d_1)=0.2$, $p^+(s_2,d_2)=0.8$,

Δ – матриця ймовірностей переходу p(s,s') з одного стану в інший (2x2), яка задає функцію перемикання станів середовища:

$$T(s,s'): \Delta = \{ \{1-\delta; \delta\}, \{\delta; 1-\delta\} \}, \delta \square [0;0.5],$$

наприклад, для $\delta = 0.3$ отримаємо

$$T(s,s'): \Delta = \{ \{0.7; 0.3\}, \{0.3; 0.7\} \}.$$

В цьому прикладі ВСПС:

1) функції виграшу у станах s_1 (E_1) і s_2 (E_2) – «протилежні» («віддзеркалені»), тобто дія вигідна у одному стані одночасно є невигідною у іншому стані;

2) функція переходів між станами T(s,s') симетрична, тобто ймовірність залишитись у стані $p(s_i,s_i)$ та ймовірність перейти в інший стан $p(s_i,s_j)$ однакові для обидвох станів.

Параметр δ задає «частоту» («швидкість») перемикання між E_1 та E_2, тобто визначає наскільки динамічно змінюється середовище $E_{2,2}$. Наприклад, у випадку коли $\delta=0$ – перемикання між станами відсутні, а у випадку коли $\delta=0.5$ – перемикання відбуваються з найбільшою можливою «частотою» («швидкістю»). Тобто, чим ближче значення δ до 0, тим повільніше відбуваються зміни у середовищі $E_{2,2}$, і, чим ближче значення δ до 0.5, тим зміни у середовищі $E_{2,2}$ відбуваються швидше.

Параметр $\rho = |p^+(s,d_1) - p^+(s,d_2)| = |x-(1-x)| = |2x-1|$, $\rho \in [0;1]$ задає масштаб переваги одної дії над іншою з точки зору ймовірності виграшу за реалізацію дії. Наприклад, якщо $\rho=0$, то $x=0.5$, $p^+(s,d_1) = p^+(s,d_2) = 0.5$ – жодна з дій не має переваги над іншою дією, а якщо $\rho=1$, то $x=1$, $p^+(s_1,d_1)=1$, $p^+(s_1,d_2)=0$ (або $p^+(s_2,d_1)=0$, $p^+(s_2,d_2)=1$) – одна дія повністю переважає іншу. Тобто параметр ρ визначає наскільки вигідною є здатність до навчання в окремому стаціонарному випадковому середовищі (E_1, E_2). Чим ближче значення ρ до 0, тим менший ефект дає навчання у середовищі $E_{2,2}$, і, чим ближче значення ρ до 1, тим більший ефект дає навчання у середовищі $E_{2,2}$. Відтак змінюючи значення параметру ρ від 0 до 1, можна змінювати «коефіцієнт корисної дії» навчання у окремому стані середовища $E_{2,2}$.

Обираючи різні значення параметрів (ρ, δ) можна отримувати різні за характеристиками реалізації ВСПС $E_{2,2}$ з метою дослідження того, як залежать результати поведінки LA у ВСПС від масштабу переваги одної дії над іншою (ρ) та динаміки змін у середовищі (δ).

Приклад програмної реалізації випадкового середовища з перемиканням станів для проведення обчислювальних експериментів по дослідженню методів навчання з підкріпленням наведено в додатку Д.2.

Відмінність ВСПС від СВС полягає в тому, що функція виграшу змінюється у часі внаслідок переходів між станами середовища. В кожному стані «діє» своя – відмінна від інших – функція виграшу. Відтак те, чому агент навчився в одному стані, не підходить для іншого стану середовища.

Поведінка LA з лінійною тактикою (та усіх інших конструкцій LA, розглянутих вище) у ВСПС характеризується тим, що цілеспрямовану поведінку демонструють лише LA з оптимальною для цього середовища глибиною пам’яті m*. При цьому оптимальна глибина пам’яті m* залежить від того, з якою «швидкістю» відбувається перемикання між станами у ВСПС, за принципом – чим більше швидкість, тим меншою є оптимальна глибина пам’яті і навпаки (рис.2.7, рис.2.8). Зокрема у ВСПС $E_{2,2} = \{E_1, E_2, \Delta\}$ (приклад 2) оптимальна глибина пам’яті m* тим менше, чим ближче значення параметру δ наближається до 0.5 (рис.2.7). В ситуації, коли $\delta=0.5$ (максимальна «швидкість» перемикання станів середовища), m*=1. Відтак, чим швидше зміни у середовищі, тим менша глибина пам’яті потрібна LA, і навпаки, чим зміни відбуваються повільніше, тим більша глибина пам’яті потрібна LA.

В ситуації, коли швидкість змін у ВСПС (зокрема функція перемикання станів T(s,s')) попередньо невідома, потрібна така конструкція LA, яка б вміла самостійно знаходити оптимальну глибину пам’яті m*.

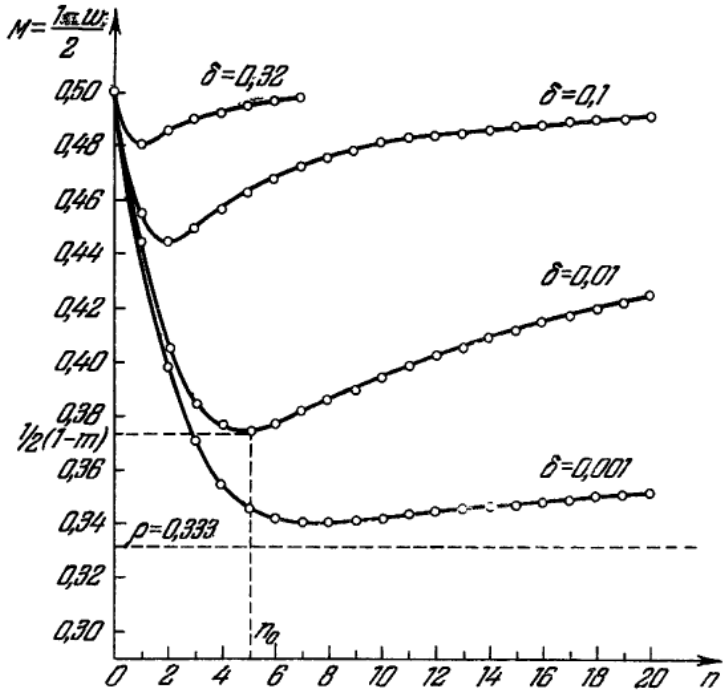


Рис. 2.7. Приклад визначення оптимальної глибини пам’яті LA з лінійною тактикою для $E_{2,2} = \{E_1, E_2, \Delta\}$, $\rho = 1/3 = 0.3(3)$, M=(1-W)/2, де W - середній виграш LA.

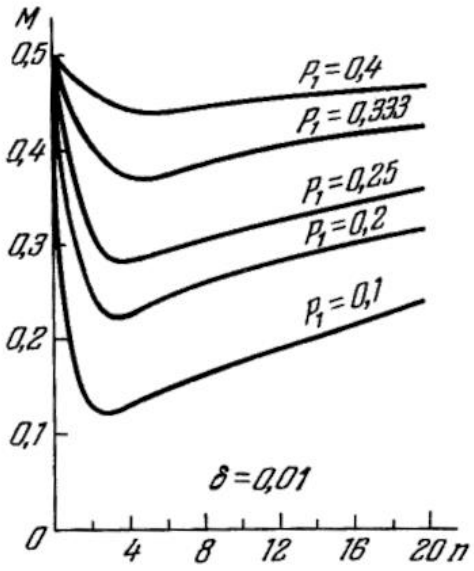


Рис. 2.8. Приклад визначення оптимальної глибини пам’яті LA з лінійною тактикою для $E_{2,2} = \{E_1, E_2, \Delta\}$, $\delta = 0.01$, M=(1-W)/2, де W - середній виграш LA.

2.2.2. Каскад двох автоматів з лінійною тактикою

Каскад двох автоматів з лінійною тактикою (каскадна конструкція LA) – це композиція з двох автоматів A_1 і A_2, наприклад, двох LA з лінійною тактикою: $A_1 = L_{m,n}$; $A_2 = L_{w,k}$ (рис.2.9). Автомат A_1 функціонує у випадковому середовищі з перемиканням станів $\boldsymbol{E}$, має n дій і m внутрішніх станів на дію, тобто глибину пам'яті m. Глибина пам'яті автомату A_1 визначається вихідним сигналом (дією) $m(t)$ автомату A_2, який в свою чергу має k дій і w внутрішніх станів на дію, тобто глибину пам'яті w.

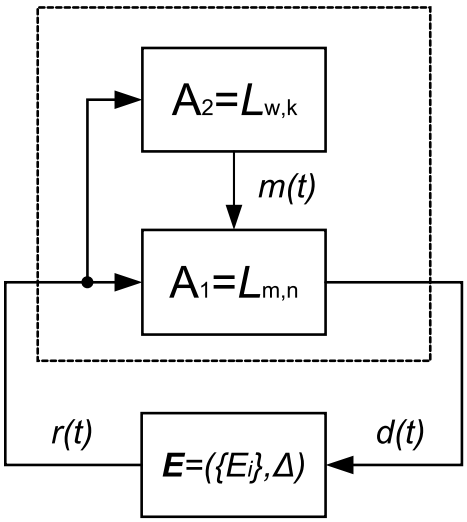


Рис. 2.9. Каскад двох автоматів з лінійною тактикою.

На вхід обох автоматів поступає відгук середовища *r(t)*, тобто сигнал виграшу $r(t) = r^+$ або сигнал програшу $r(t) = r^-$. При досить великій глибині пам'яті автомату A_2, тобто при досить великих часах усереднення, він буде формувати оптимальну глибину пам'яті автомату A_1, тобто вся конструкція буде демонструвати оптимальну поведінку в випадковому середовищі з перемиканням станів.

Зміна дії автоматом A_2 може привести до невизначеності в роботі автомату A_1, так як останній може виявитися в стані, що знаходиться поза ємністю його пам'яті. Це, наприклад, ситуація, коли A_1 знаходиться в останньому стані гілки дії d (з номером стану m=20), і в цей же час автомат A_2 здійснює дію *m(t)*=18, зменшуючи глибину пам'яті A_1 з 20 до 18. Виходить, що A_1 знаходиться в стані (*m*=20), який тепер відсутній в його конструкції. Щоб уникнути цього можна прийняти, що в разі зміни дії автомат A_2 змінює свою дію $m(t)$ на дію $m(t) + 1$ або $m(t) - 1$ з ймовірностями 0,5. А також дії $m(t) = 1$ і $m(t) = k$ змінюються на дії *2* і $k - 1$ відповідно. З огляду на те, що зміна дії відбувається тільки в разі програшу ($r(t) = r^-$), то очевидно, що в цьому випадку ніяких непорозумінь виникнути не може.

Обидва автомати (A_1 і A_2) отримують сигнал програшу одночасно. Оскільки автомат A_1 – це LA з лінійною тактикою, то у разі отримання сигналу програшу, він або зменшує номер свого стану на одиницю, або переходить в перший стан іншої дії. В цей момент збільшення його глибини пам'яті на 1 (*m(t)*+1) ніяк не вплине на його роботу. Так само зменшення його глибини пам'яті на 1 (*m(t)*–1) навіть в «найгіршому» випадку, коли він знаходився в останньому стані гілки і спустився у попередній стан (який тепер став останнім) ніяк не вплине на його роботу.

Експериментальне дослідження підтверджує висловлені міркування, однак розглянутий автомат має дуже великі часи встановлення стаціонарного режиму, оскільки для виходу на оптимальне значення потрібно дуже велика ємність пам'яті автомата A_2.

Принцип, закладений в каскадну конструкцію LA, можна узагальнити для побудови архітектур автономних інтелектуальних систем, в яких «верхній рівень» цілеспрямовано змінює структуру «нижнього рівня», здійснюючи пошук найбільш ефективних структур та їх оптимізацію. Подібний принцип діє в людському мозку. Як зазначає експериментальний психолог Брюс Худ (Bruce Hood) у своїй роботі «The Self Illusion» (2012): «... більш високі рівні мозкової діяльності, пов'язані зі складними уявленнями (зокрема, з такими як «Я» і «особистість»), можуть управляти нижчими структурами базової обробки інформації, яка надходить у мозок».

2.2.3. Стохастичні автомати зі змінною структурою

Іншою більш ефективною конструкцією автомату, що навчається, який вирішує задачу пошуку оптимальної глибини пам’яті m* у ВСПС є стохастичний автомат зі змінною структурою. На початку своєї роботи стохастичний автомат знаходиться в «байдужому» стані, коли ймовірності всіх переходів між станами для нього однакові. Автомат задається двома матрицями: $П^+$ і $П^-$. Відповідно матриця $П^+$ задає ймовірності переходу зі стану в стан після отримання сигналу про виграш, а матриця $П^-$ задає ймовірності переходу зі стану в стан після отримання сигналу про програш. Це квадратні матриці $(N \cdot M) \times (N \cdot M)$, де N – кількість дій автомата, а M – глибина пам'яті автомата.

В даному випадку досліджується задача знаходження стохастичного автомату зі змінною структурою, що володіє цілеспрямованою поведінкою в випадкових середовищах, як завдання знаходження алгоритмів зміни перехідних ймовірностей в $П^+$ і $П^-$, що забезпечують збільшення математичного сподівання виграшу. Способи зміни перехідних ймовірностей, які ми будемо розглядати, ґрунтуються на таких міркуваннях. Якщо автомат перейшов зі стану з номером i в стан з номером j, і після цього був оштрафований, то ймовірність переходу p_{ij} повинна бути зменшена. При цьому для збереження умови нормування відповідно повинні бути збільшені всі інші перехідні ймовірності в цьому рядку. Якщо ж після переходу автомат виграв, то ймовірність повинна бути збільшена і відповідно повинні бути зменшені всі інші ймовірності цього рядка.

Розглянемо наступний спосіб зміни перехідних ймовірностей. Якщо в момент часу t автомат, перебуваючи в стані i, перейшов в стан j і $S_t = +1$, $S_{t+1} = +1$, то в матриці $П^+$

$$p^+_{ij,t+1} = p^+_{ij,t} + \Delta$$
$$p^+_{ik,t+1} = p^+_{ik,t} - \Delta/(N \cdot M - 1),\ k \neq j.$$

Якщо ж $S_t = +1$, $S_{t+1} = -1$, то в матриці $П^+$

$$p^+_{ij,t+1} = p^+_{ij,t} - \Delta$$
$$p^+_{ik,t+1} = p^+_{ik,t} + \Delta/(N \cdot M - 1),\ k \neq j.$$

При цьому елементи матриці $П^-$ не змінюються. Відповідно якщо в момент часу t автомат, перебуваючи в стані i, перейшов в стан j і $S_t = -1$, $S_{t+1} = +1$, то в матриці $П^-$

$$p^-_{ij,t+1} = p^-_{ij,t} + \Delta$$
$$p^-_{ik,t+1} = p^-_{ik,t} - \Delta/(N \cdot M - 1),\ k \neq j.$$

Якщо ж $S_t = -1$, $S_{t+1} = -1$, то в матриці $П^-$

$$p^-_{ij,t+1} = p^-_{ij,t} - \Delta$$
$$p^-_{ik,t+1} = p^-_{ik,t} + \Delta/(N \cdot M - 1),\ k \neq j.$$

При цьому елементи матриці $П^+$ не змінюються. Δ – величина приросту (зменшення) ймовірностей переходу ($0 \le \Delta \le 1$). Перехід в наступний стан здійснюється за допомогою матриці, яка відповідає значенням останнього сигналу (відгуку) середовища: якщо це сигнал виграшу, то перехід здійснюється по матриці $П^+$, а якщо це сигнал програшу, то по матриці $П^-$.

Алгоритм роботи агента з блоком прийняття рішень у вигляді стохастичного автомату зі змінною структурою наступний:

1. отримати відгук середовища;
2. перерахувати ймовірності переходу на підставі отриманого і попереднього відгуків середовища;

3. запам'ятати отриманий відгук середовища як попередній;

4. перейти в наступний стан по матриці переходів, яка відповідає отриманому відгуку середовища;

5. запам'ятати попередній стан, перейти до п.1.

Дослідження показали, що лінійні закони зміни перехідних ймовірностей (Δ=const) не завжди призводять до оптимальних конструкцій автоматів. Але якщо ввести нелінійну зміну елементів вказаних матриць, то вихідні «розмазані» матриці Π^+ і Π^- з однаковими значеннями p_{ij} збігаються до матриць з нулів і одиниць, що відповідають автоматам, які найкращим чином поводять себе в відповідному стаціонарному випадковому середовищі. У випадковому середовищі з перемиканням станів з плином часу функціонування стохастичного автомата зі змінною структурою необмежено наближається до функціонування автомата з лінійною тактикою, який володіє оптимальною глибиною пам'яті для даного середовища (рис.2.10, рис.2.11). Іншими словами, автомат зі змінною структурою сам знаходить цю оптимальну глибину пам'яті.

Приклад нелінійної зміни ймовірностей переходу. Якщо в момент часу t автомат, перебуваючи в стані i, перейшов в стан j і $S_t = +1$, $S_{t+1} = +1$, то в матриці Π^+

$$p^+_{ij,t+1} = p^+_{ij,t} + \alpha \cdot p^+_{ij,t} \cdot (1 - p^+_{ij,t})$$
$$p^+_{ik,t+1} = p^+_{ik,t} - \alpha \cdot p^+_{ik,t} \cdot p^+_{ij,t}, \quad k \neq j.$$

Якщо ж $S_t = +1$, $S_{t+1} = -1$, то в матриці Π^+

$$p^+_{ij,t+1} = p^+_{ij,t} - \alpha \cdot p^+_{ij,t} \cdot (1 - p^+_{ij,t})$$
$$p^+_{ik,t+1} = p^+_{ik,t} + \alpha \cdot p^+_{ik,t} \cdot p^+_{ij,t}, \quad k \neq j.$$

При цьому елементи матриці Π^- не змінюються. Відповідно якщо в момент часу t автомат, перебуваючи в стані i, перейшов в стан j і $S_t = -1$, $S_{t+1} = +1$, то в матриці Π^-

$$p^-_{ij,t+1} = p^-_{ij,t} + \alpha \cdot p^-_{ij,t} \cdot (1 - p^-_{ij,t})$$
$$p^-_{ik,t+1} = p^-_{ik,t} - \alpha \cdot p^-_{ik,t} \cdot p^-_{ij,t}, \quad k \neq j.$$

Якщо ж $S_t = -1$, $S_{t+1} = -1$, то в матриці Π^-

$$p^-_{ij,t+1} = p^-_{ij,t} - \alpha \cdot p^-_{ij,t} \cdot (1 - p^-_{ij,t})$$
$$p^-_{ik,t+1} = p^-_{ik,t} + \alpha \cdot p^-_{ik,t} \cdot p^-_{ij,t}, \quad k \neq j.$$

При цьому елементи матриці Π^+ не змінюються. Величина α повинна задовольняти нерівності $0 \le \alpha \le 1$. Тільки в цьому випадку стохастичність матриць буде зберігатись.

Стохастичний автомат зі змінною структурою з часом наближається до поведінки LA з оптимальною глибиною пам’яті. При цьому «конструкції» (схеми переходів між станами), які утворює стохастичний автомат, є дуже подібними до відповідних конструкцій LA. Процес формування структури автомату у випадковому середовищі може служити прикладом автоматичного синтезу автомату по заданому критерію якості його роботи [8]. Згідно проведених експериментів людина в середньому краще вирішує задачу в більш складному ВСПС ніж в СВС, і у своїх результатах майже наближається до LA з оптимальною глибиною пам’яті.

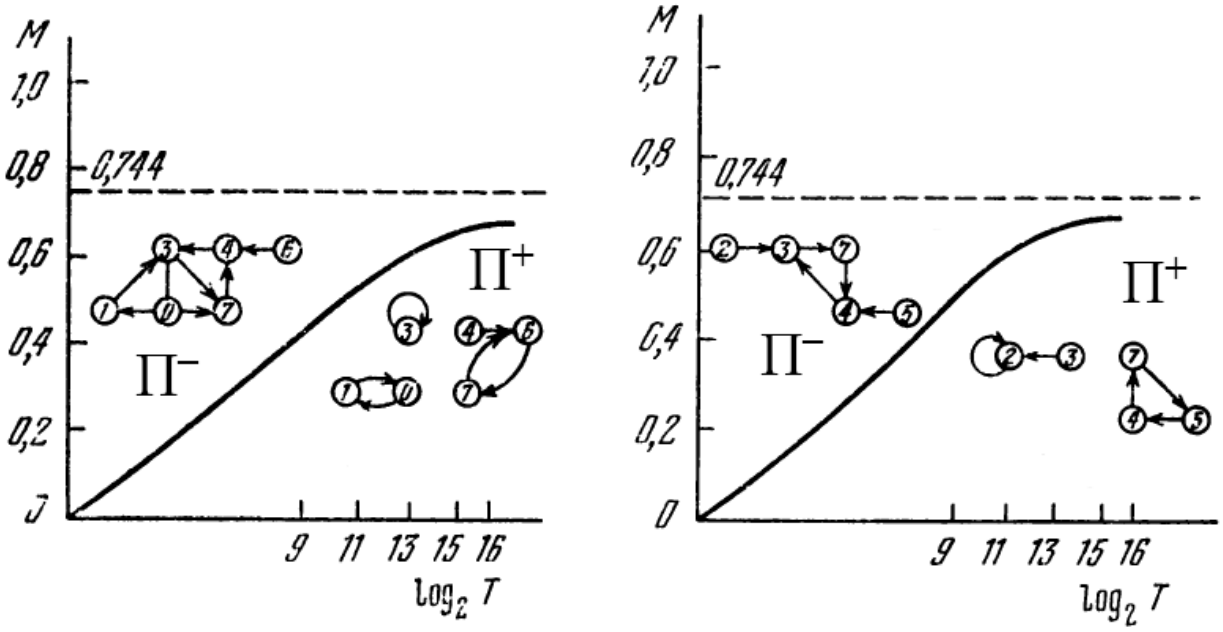


Рис. 2.10. Приклади формування стохастичним LA структур з оптимальною глибиною пам’яті m*, які схожі за конструкцією на LA з лінійною тактикою та інші LA.

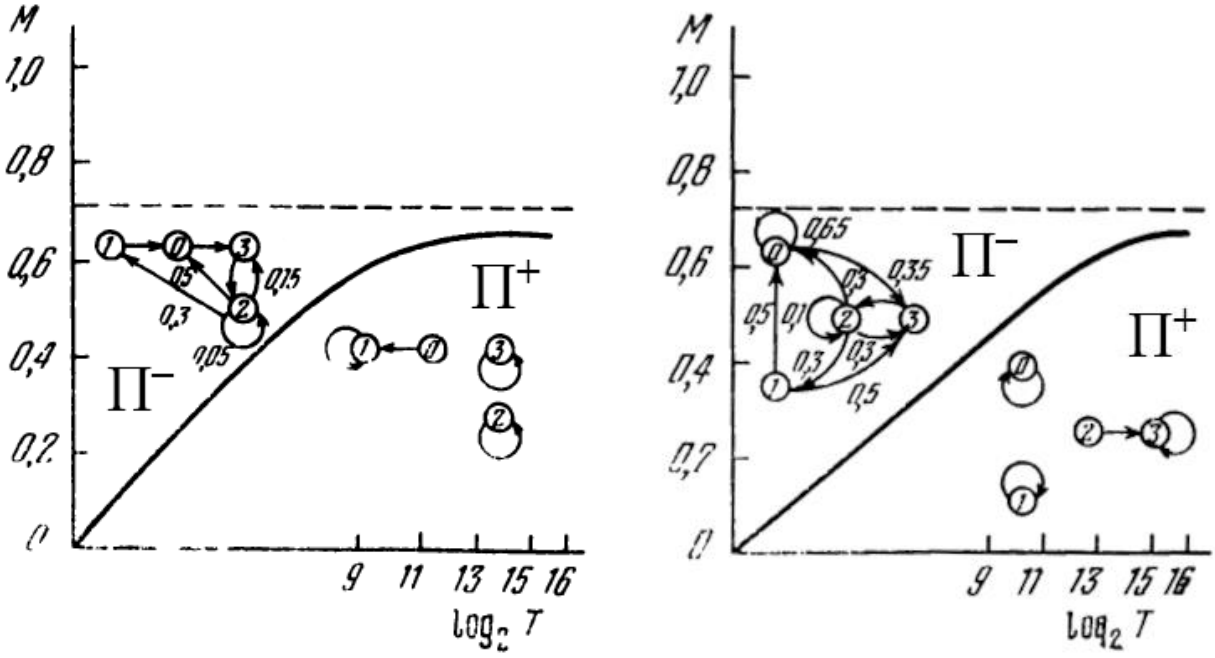


Рис. 2.11. Приклади формування стохастичним LA структур з оптимальною глибиною пам’яті m*, які схожі за конструкцією на LA з ймовірнісними переходами між станами, наприклад на автомат В.Ю. Крилова.

Контрольні питання

1. Що представляє собою модель цілеспрямованої поведінки?
2. В який спосіб задаються параметри стаціонарного випадкового середовища?
3. За якою схемою взаємодіє з випадковим середовищем автомат, що навчається?
4. Яким буде середній по часу виграш агента, який обирає дії з однаковою ймовірністю 1/n, де n - кількість дій, у стаціонарному випадковому середовищі (r(d)={0;1}) з наступними параметрами: E = {0.8; 0.3; 0.2; 0.6}, n=4?
5. Яким буде середній по часу виграш агента, який від початку обирає найкращу дію d* з множини дій D, у стаціонарному випадковому середовищі (r(d)={0;1}) з наступними параметрами: E = {0.4; 0.1; 0.3}, n=3?
6. Чому відповідає кількість гілок станів в конструкції автомату, що навчається, в стаціонарному випадковому середовищі?
7. Який параметр конструкції автомату, що навчається, є основним?
8. Як залежить поведінка автомату, що навчається, в стаціонарному випадковому середовищі від глибини пам'яті автомату?
9. В якій конструкції автомату, що навчається, у разі програшу автомат переходить на один стан нижче, а у разі виграшу на один стан вище у гілці станів відповідної дії?
10. В якій конструкції автомату, що навчається, у разі програшу автомат переходить на один стан нижче, а у разі виграшу відразу в останній стан у гілці станів відповідної дії?
11. В якій конструкції автомату, що навчається, у разі програшу автомат з першого стану гілки станів поточної дії переходить в останній стан гілки станів іншої дії?
12. Чим випадкове середовище з перемиканням станів відрізняється від стаціонарного випадкового середовища?
13. В який спосіб задаються параметри випадкового середовища з перемиканням станів?
14. Які дії виконує автомат другого рівня у каскадній конструкції двох автоматів з лінійною тактикою?
15. В чому полягає принцип роботи стохастичного автомату зі змінною структурою?

3. Навчання з підкріпленням (Reinforcement Learning)

3.1. Проблема навчання з підкріпленням

3.1.1. Машинне навчання (machine learning)

Машинне навчання (machine learning) – це галузь штучного інтелекту, в якій досліджуються різні підходи до забезпечення здатності інтелектуальних комп'ютерних систем навчатись по аналогії до здатності навчатись, яка властива тваринам і людям. Машинне навчання дозволяє створювати інтелектуальні комп'ютерні системи здатні само-вдосконалюватися та самостійно знаходити розв'язок складних задач без явного програмування цих систем [11-13]. В області машинного навчання досліджуються методи розв'язку задач та алгоритми, які здатні само-вдосконалюватися завдяки використанню здобутого досвіду. Алгоритми машинного навчання будують модель на основі вхідних даних, відомих як «навчальні дані» (training data). В подальшому ця модель використовуються для прогнозування станів об'єкту управління або прийняття рішень про дії, які впливають на об'єкт управління. Завдяки використанню у методах машинного навчання здобутого досвіду, з'являється можливість вирішувати задачі, які принципово не можуть бути вирішені шляхом явного програмування послідовностей та варіантів необхідних дій.

У машинному навчанні для прогнозування значень деяких параметрів застосовуються алгоритми, що дозволяють інтелектуальній системі вивчати та використовувати закономірності у вхідних даних. При використанні достатньої кількості таких даних система здатна встановити зв'язок між вхідними змінними та прогнозованими параметрами. За рахунок цього інтелектуальна система може прогнозувати нові значення параметрів з урахуванням зміни вхідних змінних та відповідним чином змінювати свою поведінку. Цей підхід відрізняється від звичайного програмування, коли комп'ютерна система використовує точні вказівки (програми), заздалегідь підготовлені для неї людиною. Незважаючи на те, що фундаментальні концепції машинного навчання запропоновані вже досить давно, останнім часом ця сфера розвивається прискореними темпами завдяки різкому зростанню обчислювальної потужності сучасних комп'ютерних систем та великій кількості доступних даних, які є ключем до досягнення точних прогнозів.

За способом організації процесу навчання розрізняють [11]:

1) навчання під керуванням або навчання з вчителем (supervised learning), в основі якого лежить принцип наслідування (imitation) та навчання на прикладі;

2) навчання з підкріпленням (reinforcement learning), в якому в ролі «вчителя» виступає середовище (об'єкт управління);

3) навчання без керування або навчання без вчителя (unsupervised learning), яке полягає у дослідженні причинно-наслідкових зв'язків (закономірностей) та способів їх використання для досягнення поставленої мети.

За способом збереження та використання здобутого досвіду розрізняють навчання:

1) без моделювання середовища (learning without model, model-free);

2) з моделюванням середовища (learning with model, model-based).

За способом набуття досвіду (отримання інформації про середовище) розрізняють:

1) пасивне навчання (passive learning);

2) активне навчання (active learning).

Крім цього розрізняють два основних сценарії «навчання - використання результатів навчання»:

1) після етапу навчання (здобуття досвіду) йде етап використання результатів навчання (здобутого досвіду); це типова схема для навчання під керуванням;

2) обидва етапи суміщені у часі, тобто навчання і використання результатів навчання для вирішення задачі відбуваються одночасно; це типова схема для навчання з підкріпленням.

Серед найбільш перспективних напрямків досліджень в галузі машинного навчання можна відмітити:

- методи перенесення результатів навчання (Transfer learning), які призначені для збереження досвіду і знань, отриманих інтелектуальною системою при вирішенні однієї проблеми, та застосуванні їх для вирішення іншої пов'язаної чи подібної за змістом проблеми;
- глибинне навчання (Deep learning) [14], в якому використовуються багатошарові штучні нейронні мережі та вкладені ієрархічні моделі на їх основі для обробки та аналізу даних, в тому числі для розпізнавання зображень та обробки природної мови;
- автоматизоване машинне навчання (Automated machine learning, AutoML), в якому розробляються методи та засоби автоматизації застосування машинного навчання для вирішення складних задач, зокрема шляхом організації процесів по роботі з наборами вхідних даних та моделями машинного навчання в автономному режимі без участі людини.

Приклади методів машинного навчання:

- методи навчання з використанням штучних нейронних мереж,
- байесовський метод навчання,
- навчання на основі дерева прийняття рішень (decision tree learning),
- навчання на основі формування та перевірки гіпотез,
- Q-навчання (один з методів навчання з підкріпленням) та ін.

3.1.2. Навчання з підкріпленням (reinforcement learning)

Навчання з підкріпленням (reinforcement learning, RL) – це галузь машинного навчання, яка спирається на методи прийняття рішень в умовах невизначеності та методи математичної статистки. В даному випадку інтелектуальна система навчається оптимальній поведінці у деякому середовищі з метою отримання максимальної винагороди згідно заданої цільової функції (utility function) [15]. Ця оптимальна поведінка є результатом взаємодії автономної інтелектуальної системи з середовищем (об’єктом управління), в ході якої вона спостерігає за наслідками своїх дій (підкріпленнями з боку середовища), збираючи в такий спосіб досвід, що допомагає їй досягти поставленої мети [16-21]. На відміну від навчання під керуванням (supervised learning) в задачах навчання з підкріпленням автономна інтелектуальна система повинна самостійно виявити послідовність дій, що максимізує цільову функцію. В основі цього процесу лежить пошук методом спроб та помилок, з врахуванням того, що успішність окремих дій обумовлюється не лише негайною винагородою, яку вони приносять, а й відкладеною у часі винагородою, яку вони можуть принести в майбутньому.

Розглянемо узагальнену схему навчання з підкріпленням (рис.3.1) у вигляді багатокрокової взаємодії агента з попередньо невідомим динамічним середовищем. В процесі цієї взаємодії агент самостійно навчається найкращій (оптимальній з деякої точки зору) поведінці шляхом спроб та помилок. При цьому розробник вказує агенту що вважати найкращою поведінкою і не вказує як саме її досягнути. На кожному кроці взаємодії агент обирає та реалізує дію з множини усіх доступних на цьому кроці дій і отримує відгук (реакцію) середовища на цю дію. Відгук (реакція) середовища називається підкріпленням (reinforcement), яке може бути позитивним (виграш, нагорода) чи негативним (програш, покарання). Зміст навчання з підкріпленням полягає у одночасному дослідженні характеристик середовища та використанні результатів цього дослідження для вибору дій в наступних кроках взаємодії (набуття та використання досвіду).

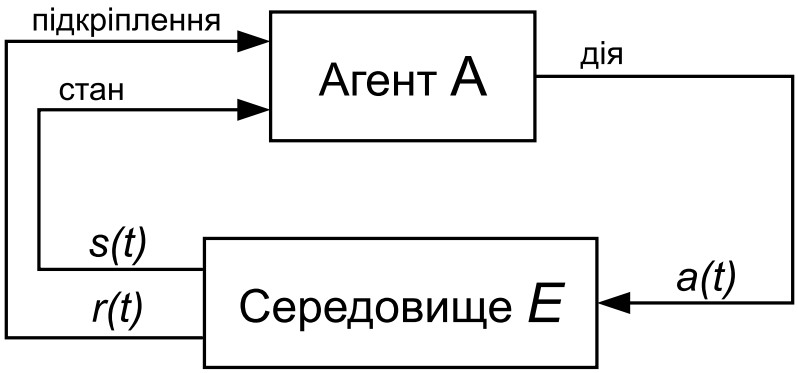


Рис. 3.1. Схема взаємодії агента з середовищем
в задачах навчання з підкріпленням.

Розглянемо основні елементи навчання з підкріпленням:

1) A – множина дій агента, S – множина станів середовища, {r} – множина відгуків середовища (підкріплень або виграшів за здійсненні дії);

2) стратегія (policy) агента π:s→a – спосіб відображення агентом стану середовища у дію (тобто спосіб прийняття рішення про вибір наступної дії);

3) функція оцінки або функція виграшу (reward function): «миттєва» оцінка ефективності дій на поточному кроці взаємодії агента з середовищем (поточний виграш);

4) цільова функція (value function, utility function): «довготермінова» оцінка ефективності дій агента шляхом акумулювання окремих біжучих виграшів згідно обраної моделі оптимальної поведінки (інтегральний виграш);

5) модель середовища (environment model): спосіб представлення невизначеності, з якою має справу агент.

Приклади випадкових середовищ (моделей об’єкту управління):

- стаціонарне випадкове середовище (binary Multi-armed Bandit),
- випадкове середовище с перемиканням станів (restless bandit problem),
- випадкове середовище Multi-armed Bandit (MAB),
- марківський процес прийняття рішень (Markov decision process, MDP).

3.1.3. Випадкове середовище Multi-armed Bandit (MAB)

Випадкове середовище Multi-armed Bandit (MAB) – це однофакторне випадкове середовище з єдиним фактором невизначеності у вигляді розподілу ймовірностей, з якими агент отримує підкріплення (позитивні чи негативні) за свої дії у цьому середовищі. До основних елементів задачі навчання з підкріпленням у випадковому середовищі MAB відносяться:

1) $A = \{a_1, a_2, ..., a_n\}$ – множина доступних агенту дій (n – кількість доступних агенту дій) та стан середовища $S=\{s_0\}$, який у MAB є незмінним;

2) $B = \{R_1, R_2, ..., R_n\}$ – множина розподілів ймовірностей (рис.3.2), де R_i – це розподіл ймовірностей отримання агентом виграшу $r \in \mathbf{R}$ за здійснення дії a_i;

3) $(\mu_1,\mu_2,\mu_3,\ldots,\mu_n)$ – середні значення розподілів ймовірностей $\{R_i\}$;

4) нехай в момент часу t агент здійснює дію a_t і отримує від середовища відгук у вигляді виграшу r_t, тоді будемо вважати, що $q^*(a_i)=\mathbf{E}[r_t|a_t=a_i]$ – це очікуваний виграш внаслідок обрання агентом дії a_i.

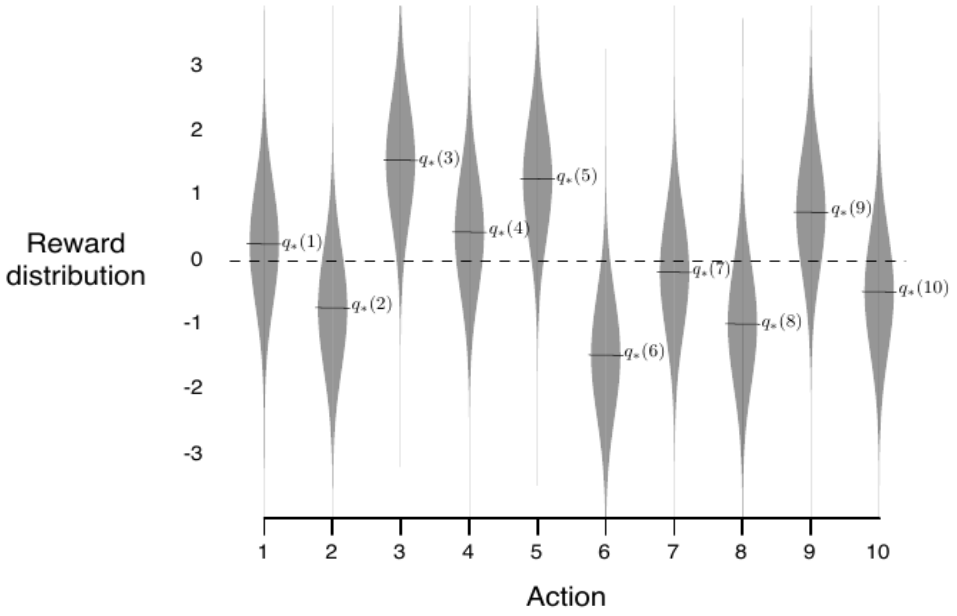


Рис. 3.2. Приклад MAB для n = 10.

Основні способи оцінки ефективності навчання в задачі навчання з підкріпленням у випадковому середовищі MAB це:

1) кінцева збіжність до оптимальної поведінки (eventual convergence to optimal);

2) швидкість збіжності до оптимальної поведінки (speed of convergence to optimal);

3) втрати в порівнянні з оптимальною від самого початку поведінкою (regret).

Розглянемо порядок визначення втрат (regret) в порівнянні з оптимальною від самого початку поведінкою в MAB:

1) нехай ($\mu_1,\mu_2,\mu_3,\ldots,\mu_k$) – множина усіх можливих середніх значень виграшів;

2) тоді максимально можливий виграш: $\mu^*=\max(\mu_i)$, i=1,…,k;

3) відтак значення втрат (regret) в момент часу T визначається як:

$$\rho(T) = T\mu^* - \sum_{t=1}^{T} r_t.$$

Відзначимо, що стратегія без втрат (zero regret policy) це випадок, коли, якщо $T \to \infty$, то $\rho(T) \to 0$.

Основна проблема навчання з підкріпленням, зокрема навчання з підкріпленням у випадковому середовищі MAB полягає у організації розподілу ресурсів (в першу чергу часового ресурсу) між дослідженням (exploring) середовища та використанням (exploiting) здобутого досвіду для досягнення поставленої мети (максимізації цільової функції).

Зауважимо, що можна провести широку аналогію між задачами навчання з підкріпленням та деякими експериментами і парадигмами в області психології, серед яких

1) процес формування умовних рефлексів у тварин (собака Павлова);

2) формування поведінки (shaping) та оперантне обумовлення (operant conditioning) (B. F. Skinner);

3) вивчена безпорадність (learned helplessness) (Martin Seligman) та ін.

Розглянемо приклади змістовної інтерпретації задачі навчання з підкріпленням у випадковому середовищі MAB.

1) Адаптивна маршрутизація в комп'ютерних мережах та автономних транспортних системах. Дія агента – це вибір одного з n доступних маршрутів. Функція оцінки приймає значення r = 0 якщо час проходження обраного маршруту більше заданого порогового значення, інакше функція оцінки приймає значення r = 1.

2) Адаптивне апаратне забезпечення (adaptive hardware). Дія агента – це вибір варіанту конфігурації, яка завантажується у ПЛІС. Функція оцінки відображає забезпечену швидкість обчислень, кількість спожитої енергії та ін.

3) Само-адаптивне програмне забезпечення (self-adaptive software). Дія агента – це вибір варіанту конфігурації всієї програми або її окремого модуля. Функція оцінки відображає час відгуку програми на зовнішні впливи, кількість спожитого програмою процесорного часу, частку оперативної пам'яті, яку займає програма та ін.

4) Структурна адаптація вимірювально-обчислювальних процесів в бездротових сенсорних мережах (wireless sensor networks). Дія агента – це вибір режиму роботи сенсорної підсистеми окремого сенсорного вузла. Функцією оцінки може бути показник кількості зібраної інформації, ймовірність виявлення цілі, час автономної роботи сенсорної мережі без поповнення запасу енергії та ін.

5) Тестування версій веб-сторінок або веб-додатків (як альтернатива A/B-тестуванню). Дія агента – це вибір версії веб-сторінки, яка буде показана наступному користувачу. Функція оцінки – це час перегляду сторінки користувачем або величина показника конверсії (conversion rate).

3.1.4. Класифікація задач навчання з підкріпленням

Задачі навчання з підкріпленням можна класифікувати з використанням наступних основних класифікаційних ознак:

1) кількість факторів невизначеності, які присутні в задачі (позначення: E1, E2, ...);

2) змістовний аспект фактору невизначеності, зокрема

2.1) з якою ймовірністю чи за яким законом розподілу ймовірностей нараховується виграш за дію (позначення: R),

2.2) з якими ймовірностями відбувається перехід середовища з одного стану в інший (в тому числі внаслідок дії агента) (позначення: T),

2.3) в якому стані знаходиться середовище в даний момент часу (позначення: S).

3) залежність фактору невизначеності від дій агента (позначення: […] – якщо залежить; (…) – якщо не залежить).

4) способі отримання відомостей, що зменшують вплив фактору невизначеності, тобто як і в якому вигляді агент отримує інформацію, яка дозволяє йому зменшувати відповідну невизначеність.

Для четвертої класифікаційної ознаки будемо використовувати наступні позначення: «0» – агент взагалі не отримує відомостей, , що зменшують вплив фактору невизначеності; «+» – агент отримує інформацію в «точному вигляді», наприклад у вигляді деякої числової величини; «–» – агент отримує інформацію в «розмитому вигляді», наприклад, у вигляді якоїсь числової величини з випадковим шумом; «T+» – агент отримує інформацію на всіх кроках взаємодії з середовищем, або «T–» – лише на підмножині кроків взаємодії (крім цього можна розрізняти варіанти, коли агент може управляти вибором кроків, на яких він отримує інформацію, і коли він не може цього робити).

Наведемо приклади кодування задач навчання з підкріпленням у різних випадкових середовищах згідно наведеним класифікаційним ознакам і відповідним позначенням:

1) E1[R+] – задача навчання з підкріпленням у CBC або MAB,

2) E2[R+](T0) – задача навчання з підкріпленням у ВСПС,

3) E2[R+][T+] – задача навчання з підкріпленням у MDP,

4) E2[R+][T–] – задача навчання з підкріпленням у POMDP.

Зауважимо, що за аналогією до «розмиття» інформації про поточний стан середовища в POMDP можна розглядати додаткове «розмиття» інформації про виграш за реалізовану дію у відповідних задачах навчання з підкріпленням E2[R–][T+] або E2[R–][T–], враховуючи той факт, що отримання виграшів з деякими ймовірностями вже є свого роду «розмиттям». В даному випадку можна провести аналогію з переходами між станами середовища, які відбуваються також з деякими ймовірностями (як, наприклад, в MDP). При ускладненні моделі середовища до POMDP, додатково вводять ймовірності, з якими агенту повідомляються істинний стан, в який перейшло середовище. В такий же спосіб можна поступати з виграшами: виграш нараховується з одною ймовірністю, а потім ще з однією додатковою ймовірністю

агенту повідомляється істинний виграш. В такому разі цілеспрямованість дій агента додатково обумовлюється його здатністю уточнювати (розпізнавати) справжню величину отриманого виграшу.

Також зауважимо, що деяким аналогом задачі навчання з підкріпленням в середовищі MAB – E1[R+] є задача навчання з підкріпленням E1[T+], в якій виграш не залежить від дій агента, в той час як від його дій залежать переходи між станами випадкового середовища. В простому випадку, в кожному стані випадкового середовища є фіксований однаковий для всіх дій виграш. В такій моделі випадкового середовища фактор невизначеності – це ймовірності переходу між станами в залежності від дій агента. Більше того в даному випадку агенту попередньо може бути відомий розподіл незмінних виграшів по станах випадкового середовища.

3.2. Принципи навчання з підкріпленням в MAB

3.2.1. Методи зваженої оцінки дій (action-value methods)

Базовими методами навчання з підкріпленням в MAB є методи зваженої оцінки дій (action-value methods) [15]. В узагальненій формі методи зваженої оцінки дій також використовуються для вирішення більш складних задач навчання з підкріпленням. При розгляді цих методів будемо використовувати наступні поняття та позначення:

- a – будь-яка дія з множини дій A={a_j}, j = 1,...,n, яку може обрати і виконати агент на кожному кроці взаємодії з випадковим середовищем MAB;
- r_i – значення поточного виграшу (підкріплення) за і-ту реалізацію дії a;
- k_a – поточна кількість реалізацій дії a, тобто лічильник того, скільки разів дія a була обрана і виконана агентом на попередніх кроках взаємодії з випадковим середовищем MAB;
- (r_1, r_2, ..., r_{ka}) – послідовність виграшів, отриманих в результаті реалізацій дії a;
- $Q_t(a)$ – оціночна вага дії (estimated action value) на кроці взаємодії t; на початку взаємодії агента з випадковим середовищем, якщо $k_a=0$, то $Q_0(a)=0$;
- Q*(a) – дійсна вага дії (true (actual) value of action), яка попередньо невідома агенту.

Розрахунок оціночної ваги дії $Q_t(a)$ в методах зваженої оцінки дій здійснюється за формулою:

$$Q_t(a) = \frac{r_1+r_2+\cdots+r_{k_a}}{k_a}.$$

Згідно закону великих чисел, якщо $k_a \to \infty$ отримаємо $Q_t(a) \to Q^*(a)$. Тобто зі зростанням кількості кроків взаємодії агента з випадковим середовищем MAB значення оціночної ваги дії $Q_t(a)$ буде наближатись до її попередньо невідомої агенту дійсної ваги Q*(a). Це дозволить агенту визначитись з тим, яка серед усіх дій є кращою за інші дії.

Розглянемо приклад розрахунку оціночної ваги дії в задачі навчання з підкріпленням в MAB на три дії D={a,b,c} з двома можливими підкріпленнями r∈{0;1}. Нехай агент на протязі 15 кроків взаємодії з випадковим середовищем (t=15) виконав послідовність дій: (aaababbaccabacb) та отримав відповідну послідовність підкріплень: (010110111010 11). Тоді лічильники для кожної дії мають наступні значення: k_a = 7, k_b = 5, k_c = 3. Відповідно можна розрахувати оціночну вагу для кожної дії:

$Q_{15}(a)$ = (0 + 1 + 0 + 1 + 1 + 1 + 1) / 7 = 5/7 = 0.714,

$Q_{15}(b)$ = (1 + 0 + 1 + 1 + 1) / 5 = 4/5 = 0.8,

$Q_{15}(c)$ = (1 + 0 + 1) / 3 = 2/3 = 0.6(6).

З точки зору способу вибору наступної дії розрізняють два основних методи зваженої оцінки дій:

1) жадібний метод навчання з підкріпленням (greedy RL) – на наступному кроці взаємодії з випадковим середовищем обирається дія, оціночна вага якої найбільша серед усіх дій:

$$a^* = arg\max_A\big(Q_t(a)\big),$$

2) ε-жадібний метод навчання з підкріпленням (ε-greedy RL) – на наступному кроці взаємодії з випадковим середовищем з ймовірністю ε дія обирається випадково з ймовірністю 1/n (тобто кожна з дій може бути обрана з однаковою ймовірністю 1/n), і з ймовірністю (1 - ε) обирається дія, оціночна вага якої найбільша серед усіх дій; в цьому методі значення ε є незмінним ε = const, 0 ≤ ε ≤ 1 і, як правило, обирається в діапазоні від 0.01 до 0.1.

Жадібний метод навчання з підкріпленням (greedy RL) можна реалізувати наступним алгоритмом:

1. першу дію обрати рівноймовірно, тобто з ймовірністю 1/n;
2. здійснити обрану дію a;
3. отримати відгук середовища (підкріплення) r;

4. перерахувати значення оціночної ваги $Q_t(a)$ дії a;

5. обрати наступну дію a*, для якої $Q_t(a^*) = \max_A\{Q_t(a)\}$, перейти до п.2.

ε-жадібний метод навчання з підкріпленням (ε-greedy RL) можна реалізувати наступним алгоритмом:

1. першу дію обрати рівноймовірно, тобто з ймовірністю 1/n;

2. здійснити обрану дію a;

3. отримати відгук середовища (підкріплення) r;

4. перерахувати значення оціночної ваги $Q_t(a)$ дії a;

5. з ймовірністю (1 – ε) обрати наступну дію a*, для якої $Q_t(a^*) = \max_A\{Q_t(a)\}$ і перейти до п.2;

6. з ймовірністю ε обрати наступну дію a* рівноймовірно (тобто з ймовірністю 1/n) і перейти до п.2.

Якщо порівняти жадібний та ε-жадібний методи навчання з підкріпленням (рис.3.3), то можна побачити, що жадібний метод виконує лише пасивне навчання, а ε-жадібний метод з ймовірністю ε виконує активне навчання. Внаслідок цього ε-жадібний метод гарантує, що із збільшенням кількості кроків взаємодії ($t \to \infty$), будуть збільшуватись кількості реалізацій усіх дій ($k_a \to \infty$), а відтак в будь-якому випадку буде знайдена дійсна вага Q*(a) усіх дій. При цьому, чим більше ε, тим швидше відбувається процес дослідження характеристик середовища (знаходження дійсної ваги дій), але тим більше частка втрат в інтегральному виграші внаслідок випадкової складової в поведінці агента.

Для порівняння ефективності жадібного та ε-жадібного методів навчання з підкріпленням в [15] був запропонований обчислювальний експеримент, в якому використано K=2000 різних MAB, згенерованих випадково; кількість дій агента n=10; кількість кроків взаємодії агента з MAB T=1000; функція виграшу R (розподіл ймовірностей отримання агентом виграшу) взята у вигляді нормального закону розподілу ймовірностей з середнім Q*(a) та дисперсією 1. Конфігурація {Q*(a)} для кожної репліки експерименту генерується також згідно нормального закону розподілу з середнім 0 та дисперсією 1. Результати обчислювального експерименту (рис.3.3) підтверджують, що ε-жадібний метод навчання з підкріпленням переважає жадібний метод, який внаслідок використання локального пошуку не завжди знаходить найкращу дію серед усіх інших дій.

Для зменшення втрат внаслідок випадкової складової в поведінці агента, ε-жадібний метод навчання з підкріпленням можна модифікувати в такий спосіб, щоб у міру того, як збирається все більше досвіду, зменшувалася ймовірність ε, з якою виконується дослідження випадкового середовища. Наприклад, в адаптивному ε-жадібному методі навчання з підкріпленням [22] реалізовано дослідження на основі різниці значень Q(a) у часі (value difference based exploration), зокрема чим більші «коливання» (дисперсія) оціночної ваги дії Q(a), тим більшою є ймовірність вибору «дослідницьких» дій ε. Для цього визначається величина:

$$f(a) = \left| \frac{e^{\frac{Q_t(a)}{\sigma}} - e^{\frac{Q_{t+1}(a)}{\sigma}}}{e^{\frac{Q_t(a)}{\sigma}} + e^{\frac{Q_{t+1}(a)}{\sigma}}} \right|,$$

яка є тим більшою, чим більше різниця між поточним та попереднім значенням оціночної ваги дії Q(a). Ця величина використовується у перерахунку ймовірності вибору «дослідницьких» дій ε:

$$\varepsilon_{t+1} = \varepsilon_t + \delta(f(a_t) - \varepsilon_t),$$

де δ = 1/n, n – кількість дій. Параметр σ – це зворотна чутливість (inverse sensitivity), за допомогою якого можна змінювати силу залежності *f(a)* від Q(a). Якщо $\sigma \to 0$, то стратегія стає чисто «дослідницькою». Початкове значення ε дорівнює одиниці: $\varepsilon_1 = 1$, тобто робота алгоритму починається з 100%-го дослідження (exploration), частка якого поступово і адаптивно зменшується.

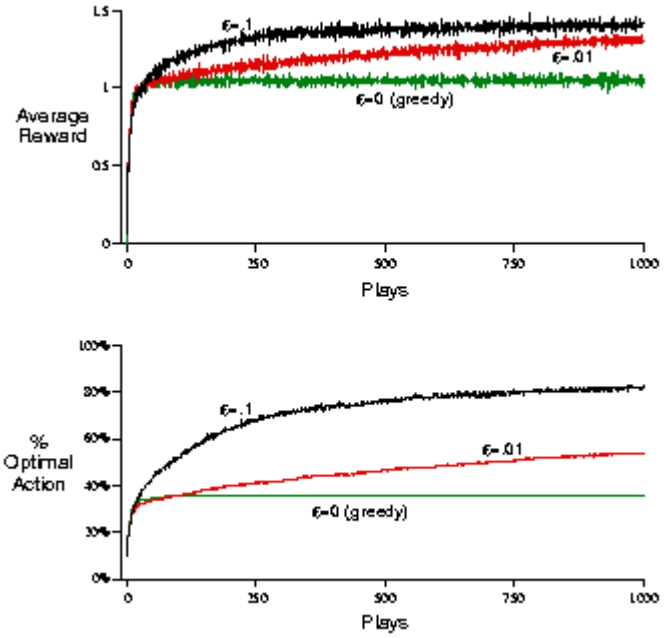


Рис. 3.3. Результати моделювання роботи базових методів зваженої оцінки дій.

3.2.2. Узагальнена форма методів зваженої оцінки дій

Методи зваженої оцінки дій у розглянутому вище вигляді мають деякі обмеження з точки зору реалізації, оскільки із збільшенням кількості кроків взаємодії з середовищем збільшується об'єм необхідної для обчислень {k(a), Q(a)} пам'яті (зокрема суми виграшів у чисельнику) та обсяг самих обчислень.

Для вирішення цієї проблеми доцільно модифікувати рівняння розрахунку оціночної ваги дії в такий спосіб, щоб зробити його рекурентним, зокрема

$$Q_{k+1} = \frac{1}{k+1}\sum_{i=1}^{k+1} r_i = \frac{1}{k+1}\left[r_{k+1} + \sum_{i=1}^{k} r_i\right] = \frac{1}{k+1}[r_{k+1} + kQ_k + Q_k - Q_k] =$$
$$= \frac{1}{k+1}[r_{k+1} + (k+1)Q_k - Q_k] = Q_k + \frac{1}{k+1}[r_{k+1} - Q_k].$$

В результаті модифікації отримано рекурентне рівняння, для реалізації якого потрібна пам'ять лише для оціночної ваги дії Q_k і лічильника k, та яка потребує малого обсягу обчислень. Від цього рівняння можна перейти до узагальненої форми рівняння розрахунку оціночної ваги дії, яке ще називають правилом навчання з підкріпленням, і яке виглядає наступним чином:

<нова оцінка> = <стара оцінка> + <розмір кроку> [<мета> – <стара оцінка>],

Q(a) = Q(a) + α [r – Q(a)].

В цьому рівнянні величину [<мета> – <стара оцінка>] можна розглядати як похибку визначення оціночної ваги, яка відображає наскільки стара оцінка відхиляється від мети на поточному кроці навчання з підкріпленням, а все рівняння можна розглядати як спробу відкоригувати значення оцінки в сторону наближення до мети.

3.2.3. Метод експоненціального усереднення

Якщо характеристики випадкового середовища змінюється у часі (тобто середовище є нестаціонарним), то розумним рішенням є збільшення ваги останніх по часу виграшів («останніх новин»), тобто врахування давнини отриманих виграшів при формуванні значення оціночної ваги дії. З цією метою замість змінного у часі розміру кроку 1/(k+1) використовують фіксовану величину α, таку що $0 < \alpha < 1$, α = const. Тоді рівняння розрахунку оціночної ваги дії буде мати наступний вигляд:

$$Q_{k+1} = Q_k + \alpha(r_{k+1} - Q_k).$$

Для того, щоб побачити з якою вагою береться кожний поточний виграш r_i, розглянемо залежність поточного значення оціночної ваги Q_k від початкового значення Q_0 та отриманих за реалізацію даної дії виграшів (r_1, r_2, ..., $r_{k(a)}$):

$$Q_k = Q_{k-1} + \alpha(r_k - Q_{k-1}) = \alpha r_k + (1-\alpha)Q_{k-1} = \alpha r_k + (1-\alpha)r_{k-1} + (1-\alpha)^2 Q_{k-2} =$$

$$= \alpha r_k + (1-\alpha)r_{k-1} + (1-\alpha)^2 r_{k-2} + \cdots + (1-\alpha)^{k-1}\alpha r_1 + (1-\alpha)^k Q_0 =$$

$$= (1-\alpha)^k Q_0 + \sum_{i=1}^{k} \alpha(1-\alpha)^{k-i} r_i.$$

В даному випадку має місце зважене усереднення, оскільки сума вагових коефіцієнтів дорівнює одиниці:

$$(1-\alpha)^k + \sum_{i=1}^{k} \alpha(1-\alpha)^{k-i} = 1.$$

Ваговий коефіцієнт $\alpha(1-\alpha)^{(k-i)}$, який ставиться у відповідність виграшу r_i, зменшує його значення в залежності від давнини цього виграшу. Оскільки величина $(1-\alpha)<1$ і ступінь $(k-i)$ зростає з давниною виграшу, то $(1-\alpha)^{(k-i)}$ зменшується з давниною. Таким чином вага виграшу r_i зменшується відповідно до того, як давно він був отриманий, тобто чим менше його номер i у порівняння з поточною кількістю кроків k.

Чим більше значення α (тобто чим ближче воно до одиниці), тим «вужче» часове вікно, в якому виграші мають достатньо великі вагові коефіцієнти і тим коротша пам'ять про минулі виграші. Чим менше значення α (тобто чим ближче воно до нуля), тим «ширше» часове вікно і тим довша пам'ять про минулі виграші. В даному випадку можна провести пряму аналогію з глибиною пам'яті автомату, що навчається (Learning Automaton) в задачі цілеспрямованої поведінки у випадковому середовищі з перемиканням станів. Відповідно для нестаціонарних середовищ з великою швидкістю змін вигідно обирати великі значення α, тоді як у випадку нестаціонарних середовищ з низькою швидкістю змін більше підходять малі значення α.

Розглянута модифікація методу зваженої оцінки дій отримала назву: експоненціальне або зважене по давнині усереднення (exponential, recency-weighted average). На відміну від цього випадок, коли α = var і $\alpha_k = 1/k$ називається вибірковим усередненням (sample average), тобто усереднення по набору вибірок (samples). Під вибіркою в даному випадку розуміється значення виграшу на і-му кроці взаємодії агента з випадковим середовищем.

Згідно теорії стохастичної апроксимації для гарантованої збіжності $Q_k(a) \to Q^*(a)$ при $k_a \to \infty$ потрібно, щоб виконувалися дві вимоги щодо розміру кроку навчання α:

$$1) \sum_{k=1}^{\infty} \alpha_k(a) = \infty, \qquad 2) \sum_{k=1}^{\infty} \alpha_k^2(a) < \infty.$$

Згідно першої вимоги розмір кроку має бути таким, щоб врешті решт гарантувати досягнення мети, тобто не бути «занадто малим». Згідно другої вимоги розмір кроку має виключати можливість «перестрибування» через мету, тобто не бути «занадто великим».

Зауважимо, що для випадку вибіркового усереднення (α = var і $\alpha_k = 1/k$) обидві вимоги справджуються, тобто використання вибіркового усереднення гарантує збіжність, тоді як для випадку зваженого по давнині усереднення справджується лише перша вимога, тобто використання цього типу усереднення не гарантує збіжності. Разом з тим в багатьох випадках практичного застосування це зауваження щодо зваженого по давнині усереднення ігнорується задля можливості відслідковувати зміни характеристик середовища.

3.3. Методи навчання з підкріпленням в MAB

3.3.1. Метод нормованої експоненціальної функції

Свого роду недоліком ε-жадібного методу навчання з підкріпленням (ε-greedy RL) є рівноцінний вибір дій на кроці «дослідження» випадкового середовища, коли дії обираються рівноймовірно, тобто кожна дія може бути обрана з ймовірністю 1/n, де n – кількість дій. Відтак на кроці «дослідження» ε-жадібного методу дії з більшою оціночною вагою Q(a) обираються з тою ж ймовірністю, що і дії з меншою оціночною вагою Q(a). В той же час доцільніше було б виділяти більше дослідницьких зусиль (більше «випробувань») на ті дії, про які вже відомо, що вони краще за інші.

Рішення цієї проблеми полягає у зміні ймовірностей вибору дій p(a) в залежності від значень оціночної ваги Q(a), яка відповідає цим діям, за принципом: чим більшу оціночну вагу має дія, тим більше ймовірність її вибору. При цьому згідно правила нормування сума ймовірностей вибору всіх дій завжди дорівнює 1. Таким чином метод нормованої експоненціальної функції (softmax action selection або Boltzmann exploration) полягає у тому, що наступна дія завжди обирається випадково, але за законом розподілу ймовірностей, який гарантує, що дія з більшою оціночною вагою має більшу ймовірність бути обраною.

Для відображення множини значень оціночних ваг дій {Q(a)} у множину значень ймовірностей їх вибору {p(a)} використовується нормована експоненціальна функція (softmax function або normalized exponential function):

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}},$$

де $\sigma(z)_i$ – значення в інтервалі [0,1]; z_i – значення з деякої вхідної множини, якому відповідає значення $\sigma(z)_i$; z_k – k-те значення з вхідної множини. Ця функція дозволяє перетворити значення вхідної множини у пропорційні їм значення вихідної множини в інтервалі [0,1], які в сумі дають 1.

Відповідно для вибору дій використовуються ймовірності, розраховані за наступною формулою (розподіл ймовірностей Больцмана, Boltzmann distribution):

$$p_t(a) = \frac{e^{\frac{Q_t(a)}{T}}}{\sum_{b \in A} e^{\frac{Q_t(b)}{T}}}.$$

де T – значення «температури» (T > 0, T = const); *a*, *b* – дії з множини усіх дій агента A, $Q_t(a)$ – оціночна вага дії *a* на кроці *t*.

Чим більше значення параметру T, тим ближче закон розподілу до рівномірного. Чим менше значення T, тим більша різниця між ймовірностями вибору дій {p(a)} (в залежності від відповідних значень оціночної ваги {Q(a)}). Відтак, якщо значення $T \to \infty$, то поведінка агента наближається до поведінки агента з рівноймовірним вибором дій ($\mathbf{A_R}$). І чим ближче значення T до 0 ($T \to 0$), тим більше метод нормованої експоненціальної функції наближається до жадібного методу навчання з підкріпленням (greedy RL). Відповідно в методі нормованої експоненціальної функції параметр T є аналогом параметру ε в ε-жадібному методі навчання з підкріпленням (ε-greedy RL).

3.3.2. Метод на основі стохастичного градієнтного підйому

В методах на основі стохастичного градієнтного підйому (Gradient Bandit Algorithms) [15] використовується значення переваги (preference) $H_t(a)$ дії a над іншими діями (на противагу оціночній вазі дії Q(a), яка в цих методах не використовується). Ідея використання $H_t(a)$ полягає в том, що чим більше перевага дії a, тим частіше вона обирається агентом. Але величина $H_t(a)$ на відміну від Q(a) не визначається напряму в термінах виграшу (reward) і відображає лише відносну перевагу одних дій над іншими.

В базовому варіанті методу на основі стохастичного градієнтного підйому [15] для кожної дії визначається ймовірність її вибору $p_t(a)$ на кроці t за наступною формулою (розподіл ймовірностей Больцмана, Boltzmann distribution):

$$p_t(a) = \frac{e^{H_t(a)}}{\sum_{b \in A} e^{H_t(b)}},$$

де $H_t(a)$ – перевага дії a; a, b – дії з множини усіх дій агента A. Згідно цієї формули, чим більше перевага дії $H_t(a)$ над іншими діями, тим більше ймовірність її вибору.

На першому кроці роботи методу переваги всіх дій $\{H_t(a)\}$ мають однакові значення (наприклад, нульові), і відповідно всі дії мають однакову ймовірність бути обраними. Уточнення значень $\{H_t(a)\}$ відбувається в ході взаємодії агента з середовищем і базується на ідеї стохастичного градієнтного підйому (stochastic gradient ascent). На кожному кроці після здійснення дії a_t і отримання виграшу (підкріплення) r_t, значення переваг $\{H_t(a)\}$ змінюються в наступний спосіб:

$$H_{t+1}(a_t) = H_t(a_t) + \alpha(r_t - \bar{r}_t)\big(1 - p_t(a_t)\big),$$
$$H_{t+1}(b) = H_t(b) + \alpha(r_t - \bar{r}_t)p_t(b), \forall b \neq a_t$$

де $\alpha > 0$ – розмір кроку навчання (наприклад, α=0.1); a, b – дії з множини усіх дій агента A; $\bar{r}_t$ – середнє значення всіх виграшів, яке розраховується інкрементно і слугує основою, з якою порівнюється кожний отримуваний виграш.

Якщо виграш r_t за дію a_t був більшим за середній: $r_t > \bar{r}_t$, то перевага дії $H_t(a_t)$ над іншими діями збільшується, і, якщо виграш r_t за дію a_t був меншим за середній: $r_t < \bar{r}_t$, то перевага дії $H_t(a_t)$ зменшується. При цьому переваги інших дій $\{H_t(b)\}$ «рухаються» в протилежному напрямку (відповідно зменшуються або збільшуються).

3.3.3. Метод верхньої довірчої межі

Метод верхньої довірчої межі (Upper-Confidence-Bound, UCB) [15, 23] заснований на ідеї, що перевагу при виборі треба віддавати діям з більшою 1) оціночною вагою Q(a) (value) та 2) невизначеністю (uncertainty) щодо точності оцінки цієї ваги. При цьому величина невизначеності оцінюється у вигляді довірчого інтервалу оціночної ваги Q(a), як випадкової величини.

Довірчий інтервал (confidence interval) – термін, який використовується в математичній статистиці при інтервальній оцінці статистичних параметрів. Довірчим називають інтервал, який покриває діапазон значень невідомого параметру із заданою надійністю (confidence level), тобто інтервал, у межах якого з заданою довірчою ймовірністю можна чекати значення оцінюваної (шуканої) випадкової величини. Відповідно верхня межа (границя) довірчого інтервалу – це верхня довірча межа (upper confidence bound), яка дала назву методу.

В методі UCB на перших кроках взаємодії з середовищем обираються дії, для яких $N_t(a)=0$, тобто які ще жодного разу не були випробувані [15]. На наступних кроках, після того, як всі дії були обрані хоча б один раз, обирається дія, для якої

$$a^* = arg\max_A \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right],$$

де t – поточний номер кроку взаємодії агента з середовищем; $N_t(a)$ – кількість кроків, на яких була обрана і здійснена дія a (знаменник у формулі визначення оціночної ваги дії); $Q_t(a)$ – оціночна вага дії (розраховується так само, як в методах зваженої оцінки дій); $\ln t$ – натуральний логарифм від t; $c > 0$ – параметр, який визначає ступінь дослідження (degree of exploration).

Другий доданок в квадратних дужках (вираз з квадратним коренем) – це один з варіантів визначення верхньої довірчої межі (upper confidence bound), як міри невизначеності щодо точності оцінки $Q_t(a)$ [15, 23]. При цьому величина c визначає надійність (confidence level) відповідного довірчого інтервалу. В роботі [23], де був запропонований метод UCB, цей варіант названо найпростішим (UCB1) та наведені більш складні варіанти (UCB2, UCB1-TUNED).

Логіка використання другого доданку полягає в наступному. Якщо дія a була обрана, то $N_t(a)$ збільшується на одиницю і другий доданок зменшується (оскільки $N_t(a)$ в знаменнику). Це відображає факт зменшення невизначеності щодо точності оцінки $Q_t(a)$ в результаті чергового «випробування» дії a. Якщо ж дія a не була обрана, то була обрана якась інша дія, відповідно зросло значення t і другий доданок збільшується (оскільки t в чисельнику). Це відображає факт збільшення невизначеності щодо точності оцінки $Q_t(a)$, як наслідок не «випробування» дії a в порівнянні з іншими діями. Використання натурального логарифму ($\ln t$) зменшує приріст чисельника зі зростанням величини часового кроку, але не обмежує його зростання (рис.3.4). Відтак зрештою всі дії будуть «випробувані», але з плином часу затримки між «випробуваннями» будуть ставати все більшими (тобто частота вибору все меншою) для тих дій, оціночна вага яких менше або які вже були «випробувані» багато разів.

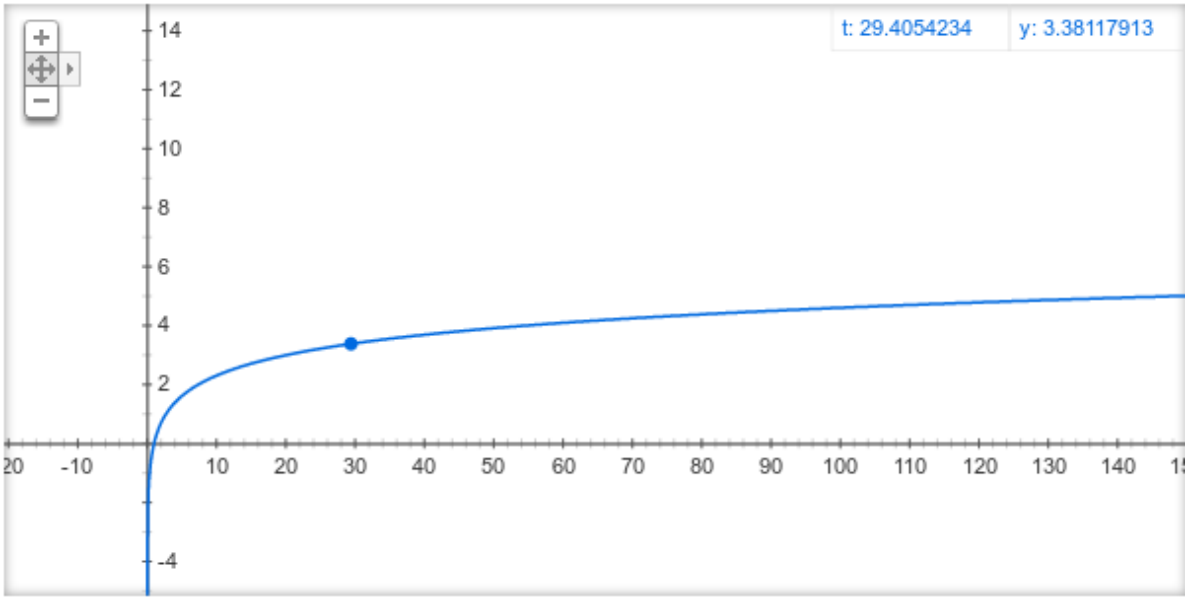


Рис. 3.4. Графік функції ln(t).

Метод UCB, як правило, дає одні з найкращих результатів у випадковому середовищі MAB (multi-armed bandit) (рис.3.5), але, в той же час, спроби узагальнити його для використання в більш складних задачах навчання з підкріпленням (наприклад, для навчання з підкріпленням в MDP) пов'язані з великими труднощами [15].

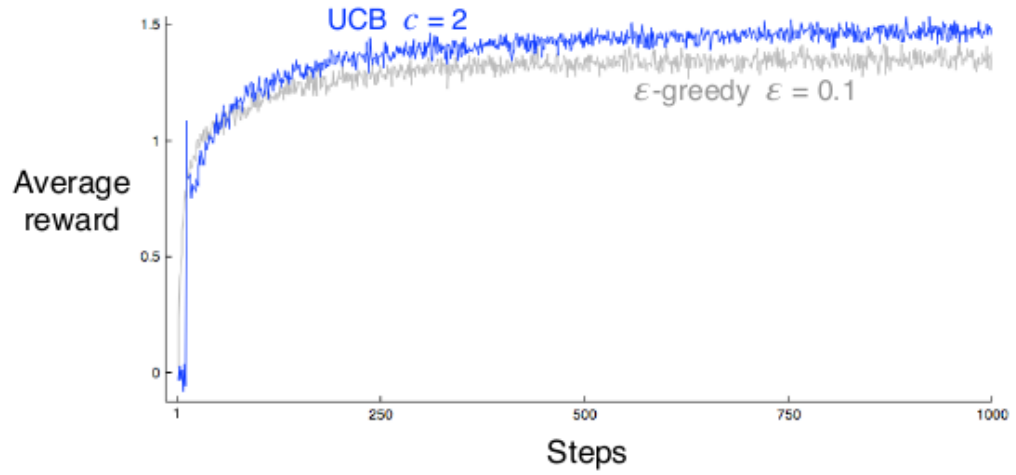


Рис. 3.5. Порівняння ефективності двох методів навчання з підкріпленням: UCB (c=2) та ε-greedy (ε=0.1).

3.3.4. Порівняння ефективності методів навчання з підкріпленням в MAB

Порівняння розглянутих методів з точки зору визначення найкращого з них є достатньо складною задачею. По-перше, розглянуті методи демонструють різні результати у порівнянні один з одним в різних конфігураціях випадкового середовища MAB (multi-armed problem). Це стосується основних параметрів конфігурації MAB: кількості дій агента та характеристик функції нарахування виграшів. По-друге, результати роботи розглянутих методів сильно залежать від вибору значень їх параметрів (початкові значення $\{Q_0(a)\}$, ε, α, T, c). В різних конфігураціях MAB метод з однаковими значеннями параметрів може демонструвати значну різницю у ефективності роботи.

В [15] запропоновано підхід до експериментального порівняння розглянутих методів з використанням значення середнього виграшу після 1000 кроків взаємодії агента з середовищем для методів з різними варіантами налаштувань параметрів (рис.3.6).

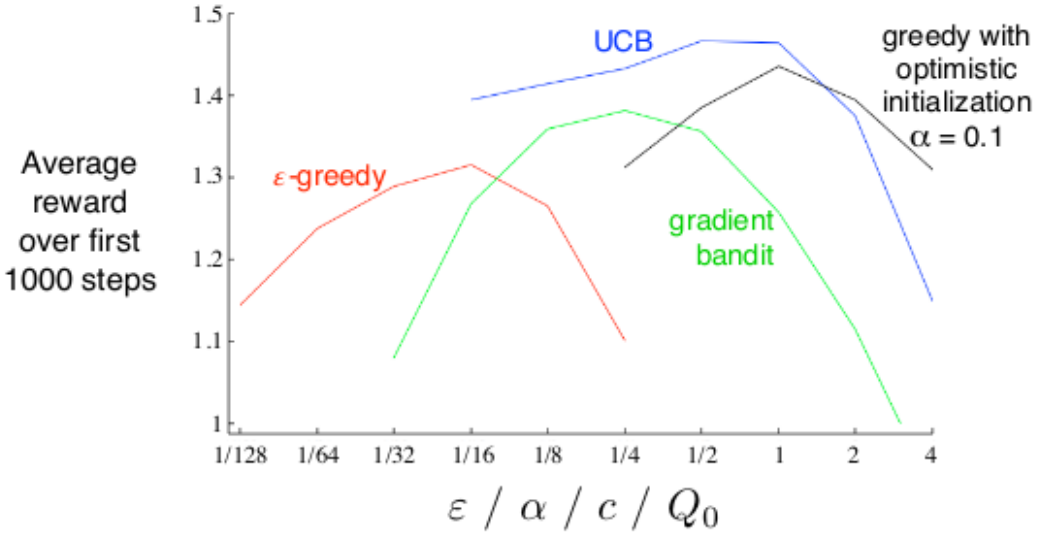


Рис. 3.6. Порівняння ефективності методів навчання з підкріпленням в MAB (кількість дій n=10).

Ще одним аспектом порівняння розглянутих методів є можливість їх узагальнення на більш складні варіанти задач навчання з підкріпленням (в тому числі з точки зору складності математичного опису та аналізу). В цьому плані найбільш зручними є методи зваженої оцінки дій (action-value methods) та метод нормованої експоненціальної функції (softmax action selection) [15].

Приклади програмної реалізації методів навчання з підкріпленням в MAB наведено в додатку Д.5.

3.3.5. MAB з контекстною залежністю

У випадковому середовищі MAB з контекстною залежністю (Contextual bandit) використовується поняття контексту (context) у вигляді вектору деяких параметрів {x}, які на кожному кроці взаємодії агента з середовищем набувають певних значень. При цьому, різним варіантам значень параметрів {x} (різним контекстам) відповідають різні функції нарахування виграшів за дії агента, що суттєво ускладнює цю задачу в порівнянні з базовим варіантом MAB. В даній задачі навчання з підкріпленням передбачається, що агенту на кожному кроці взаємодії з випадковим середовищем відомі значення параметрів {x}, тобто агент може розрізняти відповідні стани, в яких знаходиться середовище (на відміну від ВСПС, де така можливість відсутня).

В такий спосіб моделюється сценарій, коли агент в ході взаємодії з середовищем (об’єктом управління) потрапляє у різні «ситуації» (контексти), і повинен вивчити в яких «ситуаціях» які дії є найкращими. Окремо слід зауважити, що в MAB з контекстною залежністю (Contextual bandit), так само як і у ВСПС, дії агента не впливають на те, в яку наступну «ситуацію» потрапить агент, а лише визначають його виграш в поточній «ситуації».

Розглянемо варіанти задачі навчання з підкріпленням в MAB з контекстною залежністю. В базовому варіанті постановки задачі випадкове середовище складається з множини MAB, які перемикаються випадково (по аналогії з ВСПС) [15]. В більш складних варіантах задачі залежність функції виграшу від контексту задається в інший спосіб, наприклад, в одному з варіантів постановки задачі навчання з підкріпленням в MAB з контекстною залежністю для кожної дії розглядається свій окремий контекст. У іншому варіанті задачі – Constrained Contextual bandit – з виконанням кожної дії пов’язується відповідна величина витрати деякого ресурсу, загальний об’єм якого обмежений.

Основний підхід до розв'язку базового варіанту задачі полягає у використанні різних примірників алгоритму навчання з підкріпленням для різних контекстів, тобто агент розпізнає контекст, як свого роду стан середовища, і застосовує відповідний примірник алгоритму навчання з вже накопиченим досвідом взаємодії з середовищем в цьому стані. Недоліком цього підходу є його непридатність для випадків великої кількості різних варіантів контексту (кількості станів середовища). В таких ситуаціях застосовуються методи, які дають апроксимоване рішення задачі, зокрема це методи: LinUCB, UCBogram algorithm, NeuralBandit algorithm, KernelUCB algorithm та Bandit Forest algorithm.

Серед перспективних областей практичного застосування методів навчання з підкріпленням в MAB з контекстною залежністю можна відмітити розроблення контекстно-залежного програмного забезпечення (Context-Aware software), в тому числі з використанням технологій мобільних обчислень, та створення контекстно-залежних рекомендаційних систем (Context-Aware Recommender Systems).

Контрольні питання

1. В чому полягають основні принципи машинного навчання?
2. Які основні типи машинного навчання можна виділити з точки зору способу організації процесу навчання?
3. Чим навчання під керуванням (supervised learning) відрізняється від навчання з підкріпленням (reinforcement learning)?
4. В чому полягає основна проблематика навчання з підкріпленням?
5. За якими основними ознаками класифікуються задачі навчання з підкріпленням?
6. В який спосіб задаються параметри випадкового середовища Multi-armed bandit?
7. Чим випадкове середовище Multi-armed bandit відрізняється від стаціонарного випадкового середовища?
8. Що таке оціночна вага дії і як вона використовується в методах зваженої оцінки дій (action-value methods)?
9. Якими будуть значення оціночних ваг {Q(a), Q(b), Q(c)} дій (a, b, c), якщо результати взаємодії агента з середовищем MAB наступні: {[a~0], [a~0], [a~1], [b~1], [a~1], [b~0], [c~0], [c~1], [b~0], [a~1], [b~1], [b~0], [a~0], [c~0], s[c~1]}?
10. Який основний недолік жадібного методу навчання з підкріпленням (greedy RL)?
11. В чому полягає різниця між жадібним методом навчання з підкріпленням (greedy RL) та ε-жадібним методом навчання з підкріпленням (ε-greedy RL)?
12. В чому полягає адаптивний ε-жадібний метод навчання з підкріпленням?
13. Як розраховується оціночна вага дії в узагальненій формі методів зваженої оцінки дій (action-value methods)?
14. На що впливає параметр α в методі експоненціального (зваженого по давнині) усереднення?
15. Якщо оціночна вага дій (a,b), n=2 дорівнює Q(a)=0.4, Q(b)=0.3, то з якими ймовірностями будуть обиратись ці дії за методом нормованої експоненціальної функції (softmax action selection), за умови що параметр T=0.1?
16. На що впливає параметр T в методі нормованої експоненціальної функції (softmax action selection)?
17. Якщо оціночна вага дій (a, b), n=2 дорівнює Q(a)=0.4, Q(b)=0.3, а кількість застосувань цих дій дорівнює k(a)=10, k(b)=5, то яка з них буде обрана для виконання за методом верхньої довірчої межі (UCB), при умові що параметр c=2?
18. В якому методі навчання з підкріпленням в MAB використовується значення переваги (preference) дії H(a) над іншими діями?
19. Від чого залежить функція виграшу R(a) в випадковому середовищі Multi-armed bandit з контекстною залежністю (Contextual bandit)?

4. Марківський процес прийняття рішень (Markov Decision Process)

4.1. Навчання з підкріпленням в MDP

4.1.1. Задача навчання з підкріпленням в MDP

У випадку марківського процесу прийняття рішень (Markov Decision Process, MDP) проблема навчання з підкріпленням ускладнюється тим, що від обраної агентом дії залежить не тільки його поточний виграш *r*, але і наступний стан випадкового середовища *s*, в який воно перейде внаслідок цієї дії. Скажімо, можлива ситуація, коли дія, що може принести агенту великий виграш, в той же час «перемикає» середовище в дуже невигідний для нього стан (наприклад, в такий стан, в якому будь-яка дія агента призводить до великого програшу).

Таким чином агент має одночасно враховувати можливий поточний виграш та послідовність виграшів, які очікують на нього в майбутньому в тих станах середовища, в які воно потенційно може перейти під впливом його дій. Для інтегральної оцінки таких майбутніх виграшів використовується поняття відкладеного у часі підкріплення (delayed reinforcement). З цим поняттям зокрема пов'язана проблема визначення того, яка з виконаних агентом дій, яку частку сумарного виграшу принесла (temporal credit assignment problem).

Зауважимо, що поняття відкладеного у часі підкріплення 1) розкриває той факт, що поточний виграш агента у MDP залежить не лише від його дії на даному кроці взаємодії з середовищем, але й від послідовності усіх попередніх дій (при погляді в минуле); та 2) визначає оптимальну поведінку агента як таку, що максимізує суму усіх виграшів, які він потенційно може отримати у скільки завгодно далекому майбутньому (при погляді в майбутнє). Відтак модель випадкового середовища MDP висвітлює проблематику навчання з відкладеним у часі підкріпленням.

Розглянемо складові компоненти моделі випадкового середовища MDP. Марківський процес прийняття рішень (MDP) задається четвіркою <S,A,R,T>, де

1) S={s} – множина станів середовища, в загальному випадку нескінченна; проте в більшості варіантів задачі навчання з підкріпленням кількість станів середовища S скінченна, тобто розглядається скінченний марківський процес прийняття рішень (finite MDP);

2) A={a} – множина доступних агенту дій; в загальному випадку в різних станах середовища агенту можуть бути доступні різні підмножини дій;

3) R – функція відображення пари (s,a) у виграш (функція виграшу, функція підкріплення, reinforcement function): $R:S\times A\to\Re$, яка визначає схему (розподіл) виграшів для кожної пари (стан середовища, дія агента);

4) T – функція переходів між станами середовища внаслідок дій агента (функція ймовірностей переходів, transition probability function): $T:S\times A\to\Pi(S)$, яка визначає розподіл ймовірностей переходу середовища з одного стану в інший під впливом дії агента.

Випадкове середовище є марківським процесом прийняття рішень, якщо розподіл ймовірностей переходу (функція переходів T) не залежить від усіх попередніх станів середовища та попередніх дій агента. В задачі навчання з підкріпленням в MDP агенту попередньо невідомий вигляд двох функцій: 1) функції підкріплення R(s,a) та 2) функції переходів між станами T(s,a,s'). Відтак у порівнянні з випадковим середовищем MAB замість пошуку одної найкращої дії агенту потрібно знайти найкращу послідовність дій з прив'язкою до станів середовища.

Для вирішення задачі навчання з підкріпленням в MPD потрібно розробити методи знаходження агентом оптимальної поведінки (policy) $\pi:S\to A$, яка відображає стани середовища S={s} у дії агента A={a} і максимізує значення цільової функції (utility function) згідно обраної моделі оптимальної поведінки. При цьому на кожному кроці взаємодії з середовищем, що знаходиться в стані *s*, агент обирає і реалізує деяку дію *a*. Після цього він отримує на вхід: 1) виграш *r*, який принесла дія *a* в стані середовища *s*, та 2) наступний стан

середовища s', в який воно перейшло під впливом дії a (рис.3.1). Ця інформація зберігається в пам'яті агента у вигляді четвірки значень (s,a,r,s') для кожного кроку взаємодії агента з середовищем.

Відзначимо дві основні відмінності між розглянутим вище випадковим середовищем з перемиканням станів (ВСПС) та випадковим середовищем MDP:

1) у ВСПС на відміну від MDP агент не розрізняє станів середовища (тобто отримує на вхід лише виграш r, отриманий за реалізацію дії a),

2) у ВСПС функція ймовірностей переходів між станами середовища T(s,s') на відміну від MDP не залежить від дій агента (з цієї точки зору ВСПС можна рахувати більш простою моделлю середовища ніж MDP).

Розглянемо основні варіанти моделі оптимальної поведінки у MDP, зокрема модель з обмеженим інтервалом та модель з відступаючим інтервалом.

В моделі оптимальної поведінки з обмеженим (скінченним) інтервалом (finite-horizon model) цільова функція агента визначається як сума виграшів на інтервалі часу [t,T], де t – поточний крок взаємодії агента з середовищем:

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_T,$$

тобто цільова функція має вигляд:

$$W = E\left(\sum_{t=0}^{h} r_t\right).$$

В моделі оптимальної поведінки з відступаючим інтервалом (receding-horizon model) цільова функція агента визначається як зважена сума виграшів на всіх кроках взаємодії з середовищем, починаючи з поточного кроку t:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1},$$

тобто цільова функція має вигляд:

$$W = E\left(\sum_{t=0}^{\infty} \gamma^t r_t\right),$$

де γ - коефіцієнт послаблення майбутніх виграшів (discount rate, discount factor), такий що $0 \le \gamma < 1$.

Величина коефіцієнту послаблення γ обирається в діапазоні від 0 до 1. Чим більше значення параметру γ (тобто, чим ближче воно до 1), тим «ширше» вікно прогнозування (тим агент більш «далекоглядний») і навпаки, чим менше значення параметру γ (тобто, чим ближче воно до 0), тим «вужче» вікно прогнозування. Якщо коефіцієнт послаблення γ прямує до 1, то ми отримуємо граничний варіант моделі оптимальної поведінки з відступаючим інтервалом – модель середнього виграшу (average-reward model):

$$\lim_{h\to\infty} E\left(\sum_{t=0}^{h} r_t\right).$$

Зауважимо, що в більшості задач навчання з підкріпленням в MPD та відповідних методах навчання з підкріпленням в якості основної моделі оптимальної поведінки розглядається модель з відступаючим інтервалом (receding-horizon model).

Приклад програмної реалізації марківського процесу прийняття рішень наведено в додатку Д.6.

4.1.2. Пошук оптимальної стратегії для відомої моделі MDP

У випадку випадкового середовища MDP стратегія (policy) агента π(s)=a – це «програма» його поведінки, яка відповідає на питання: яку дію a обрати в поточному стані s. Перед тим, яка перейти до розгляду методів навчання з підкріпленням в MDP, розглянемо проблему пошуку та розрахунку оптимальної стратегії π* для відомої моделі MDP, тобто для випадку, коли функція підкріплення R(s,a) та функції переходів між станами T(s,a,s') відомі. Оптимальною будемо вважати таку стратегію π*, яка максимізує значення цільової функції (utility function) в рамках розглянутої вище моделі оптимальної поведінки з відступаючим інтервалом (receding-horizon model).

Для пошуку оптимальної стратегії для відомої моделі MDP та в методах навчання з підкріпленням в MDP використовується поняття оптимальної оціночної ваги стану s (optimal value of a state):

$$V^*(s) = \max_{\pi} E\left(\sum_{t=0}^{\infty} \gamma^t r_t\right),$$

яке за аналогією до дійсної ваги дії Q*(a) в методах зваженої оцінки дій визначає наскільки вигідним є той чи інший стан середовища з точки зору максимізації цільової функції.

Для пошуку та розрахунку оптимальної стратегії за умов відомої моделі MDP використовується система двох рівнянь:

1) рівняння для визначення оптимальної оціночної ваги стану V*(s), як максимуму суми виграшу за дію R(s,a) та «послабленої» величини оптимальної оціночної ваги наступного стану V*(s'), в який перейде MDP, внаслідок здійснення цієї дії:

$$V^*(s) = \max_{a}\left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V^*(s')\right), \forall s \in S,$$

2) рівняння для визначення оптимальної стратегії π*(s) у вигляді вибору такої дії, для якої є максимальною сума виграшу за дію R(s,a) та «послабленої» величини оптимальної оціночної ваги наступного стану V*(s'), в який перейде MDP, внаслідок здійснення цієї дії:

$$\pi^*(s) = arg\max_{a}\left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V^*(s')\right),$$

Розглянемо два основних підходи до організації розрахунків для пошуку оптимальної стратегії π*(s) за допомогою ітераційної процедури, зокрема ітерацію по оціночній вазі стану та ітерацію по стратегіях.

1. Ітерація по оціночній вазі стану (value iteration) складається з двох кроків:

1) виконувати в циклі обрахунок оціночної ваги V(s) для всіх станів (рекурсивні ітерації), наприклад, за наступною формулою, яка є варіантом рівняння для визначення оптимальної оціночної ваги стану:

$$V_{k+1}(s) = \max_{a} \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma V_k(s')],$$

2) зупинитись, коли різниця попереднього та поточного значення V(s) буде меншою заданої величини похибки розрахунку для всіх s.

Приклад алгоритму розрахунків для випадку ітерації по оціночній вазі стану наведено на рис.4.1.

2. Ітерація по стратегіях (policy iteration) складається з трьох кроків:

1) для всіх станів s виконати один раз розрахунок згідно рівняння для визначення оптимальної стратегії, тобто отримати стратегію π(s),

2) розв'язати систему лінійних рівнянь, складену із застосуванням рівняння для визначення оптимальної оціночної ваги для кожного стану, або виконувати розрахунки за допомогою цього рівняння ітеративно до моменту збіжності, тобто знайти {V(s)} для заданої стратегії π(s)),

3) якщо перерахунок рівняння для визначення оптимальної стратегії не змінив стратегію π(s), то зупинитись на знайденій стратегії, яка є оптимальною: π*(s) = π(s), інакше перейти до п.1.

Приклад алгоритму розрахунків для випадку ітерації по стратегіях наведено на рис.4.2.

01: Встановити довільні початкові значення {V(s)}
02: цикл до моменту досягнення потрібної точності розрахунку {V(s)}
03: цикл по станах s∈S
04: цикл по діях a∈A
05: $Q(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V(s')$
06: $V(s) = \max_a Q(s,a)$
07: кінець циклу
08: кінець циклу

Рис. 4.1. Приклад алгоритму розрахунків для випадку ітерації по оціночній вазі стану (value iteration).

01: Обрати довільну початкову стратегію π'
02: цикл
03: π := π'
04: розрахувати значення $\{V_\pi(s)\}$ для стратегії π:
05: розв'язати систему лінійних рівнянь:
06: $V_\pi(s) = R(s,\pi(s)) + \gamma \sum_{s' \in S} T(s,\pi(s),s')V_\pi(s')$
07: визначити нову покращену стратегію, тобто для кожного стану s∈S:
08: $\pi'(s) := \max_a (R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V_\pi(s'))$
09: зупинитись, якщо π = π', інакше продовжити цикл

Рис. 4.2. Приклад алгоритму розрахунків для випадку ітерації по стратегіях (policy iteration).

4.2. Методи навчання з підкріпленням в MDP

4.2.1. Метод адаптивної евристичної оцінки

Нагадаємо, що основною проблемою навчання з підкріпленням в MDP є те, що функція розподілу ймовірностей переходів між станами середовища T(s,a,s') попередньо невідома агенту. Таким чином агенту необхідно досліджувати не тільки схему виграшів для кожного стану *s* (функцію підкріплення R(s,a)), але і вигляд функції переходів T(s,a,s'). При цьому агент вирішує проблему розподілення отриманого виграшу у часі (temporal credit assignment), тобто оцінює яка з дій a є найбільш вдалою у стані *s*, при умові що вона має далекосяжні наслідки, а також визначає яка з обраних в минулому дій призвела до поточного виграшу.

Відмітимо, що методи навчання з підкріпленням у MDP розрізняються за

1) способом організації отримання досвіду взаємодії агента з середовищем,

2) способом використання (аналізу) отриманого досвіду для покращення стратегії поведінки агента $\pi(s)=a$.

У методі адаптивної евристичної оцінки (Adaptive Heuristic Critic, AHC) (рис.4.3) кожному стану середовища ставиться у відповідність оціночна вага стану (estimated value of state) V(s), $s \in S$. На кожному кроці взаємодії агента з середовищем ця величина модифікується (уточнюється) в такий спосіб, щоб з часом гарантувати її збіжність до попередньо невідомої оптимальної оціночної ваги стану або іншими словами дійсної ваги стану V*(s).

Значення V(s) модифікується з використанням поточного виграшу r та передбачуваної ваги стану s', в який перейшло середовище внаслідок реалізації обраної дії a [16]:

$$V(s) = V(s) + \alpha\big(r + \gamma V(s') - V(s)\big),$$

де

V(s) – оціночна вага стану *s*,

V(s′) – оціночна вага стану s', в який перейшло середовище внаслідок дії агента a,

r – поточний виграш агента за дію a у стані *s*,

α – крок навчання (learning rate), такий що $0 \le \alpha < 1$,

γ – коефіцієнт послаблення (discount rate, discount factor), такий що $0 \le \gamma < 1$.

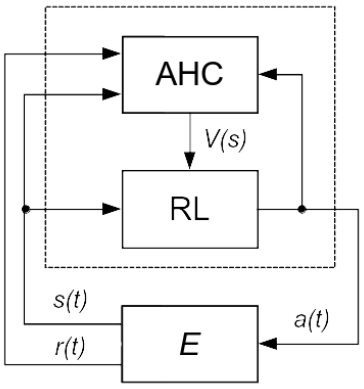


Рис. 4.3. Схема роботи методу адаптивної евристичної оцінки (Adaptive Heuristic Critic).

Розглянемо схему роботи методу адаптивної евристичної оцінки (рис.4.3). Блок верхнього рівня AHC навчається правильно відображати стани середовища {s} у оціночну вагу цих станів {V(s)} для поточної стратегії $\pi(s)$, яку реалізує блок нижнього рівня RL. У блоці RL виконується метод навчання з підкріпленням в MAB, модифікований для роботи з кількома станами середовища, тобто для кожного стану використовується свій незалежний

примірник методу навчання. Це, наприклад, може бути жадібний метод навчання з підкріпленням (greedy RL), ε-жадібний метод навчання з підкріпленням (ε-greedy RL) або метод нормованої експоненціальної функції (softmax action selection). В якості підкріплення блок RL отримує вихідну величину блоку АНС, тобто значення оціночної ваги поточного стану V(s), тоді як «справжнє» підкріплення *r* поступає на вхід блоку АНС.

4.2.2. Навчання з підкріпленням на основі часових різниць

В методах навчання з підкріпленням на основі часових різниць (Temporal difference learning, TD-learning) використовується поняття «часової різниці» (temporal difference), як різниці між

1) поточним значенням оцінюваної величини («прогнозом»), отриманим внаслідок накопичення досвіду взаємодії агента з середовищем на минулих кроках взаємодії, і

2) безпосереднім результатом взаємодії агента з середовищем в даний момент («реальністю»), який визначається в деякому масштабі часу (наприклад, як один перехід між поточним та наступним станом).

Оцінюваною величиною в методах навчання з підкріпленням на основі часових різниць може бути:

1) оціночна вага стану: V(s) (базовий варіант),

2) оціночна вага пари (стан, дія): Q(s,a).

В найбільш простому варіанті методу навчання з підкріпленням на основі часових різниць (TD(0) або one-step TD) береться до уваги лише один крок взаємодії агента з середовищем, зокрема значення оціночної ваги стану V(s) модифікуються (уточнюється) за формулою [15]:

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

де

$V(s_t)$ – оціночна вага стану $s = s_t$,

$V(s_{t+1})$ – оціночна вага стану $s' = s_{t+1}$, в який перейшло середовище внаслідок дії агента a,

r_{t+1} – поточний виграш агента за дію a у стані s_t,

α – крок навчання, такий що $0 \le \alpha < 1$,

γ - коефіцієнт послаблення, такий що $0 \le \gamma < 1$.

Алгоритм роботи методу TD(0) в частині уточнення оціночної ваги кожного стану V(s) для заданої стратегії наведено на рис. 4.4.

Вхідні дані: стратегія ***π***, ефективність якої оцінюється
01: Встановити довільні початкові значення {V(s)}, наприклад V(s)=0
02: цикл по епізодах
03: визначити початковий стан епізоду ***s***
04: цикл по кроках епізоду:
05: обрати дію ***a*** згідно стратегії ***π*** для поточного стану ***s***
06: реалізувати дію ***a***, отримати виграш ***r***, визначити наступний стан ***s′***
07: $V(s) := V(s) + \alpha\big(r + \gamma V(s') - V(s)\big)$
08: ***s*** := ***s′***
09: зупинитись, якщо стан ***s*** - кінцевий, інакше продовжити цикл
10: кінець циклу

Рис. 4.4. Алгоритм роботи методу TD(0) в частині уточнення оціночної ваги кожного стану V(s).

В більш складних варіантах методу навчання з підкріпленням на основі часових різниць додатково враховуються оціночна вага станів в більшому масштабі часу: два кроки, три кроки і т.д. (рис.4.5). Відповідно для кожного масштабу часу можна розрахувати значення цільової функції в рамках моделі оптимальної поведінки з відступаючим інтервалом (receding-horizon model), зокрема

1) для одного кроку (one-step return):

$$R_{t:t+1} = r_{t+1} + \gamma V_t(s_{t+1}),$$

2) для двох кроків (two-step return):

$$R_{t:t+2} = r_{t+1} + \gamma r_{t+2} + \gamma^2 V_{t+1}(s_{t+2}),$$

. . .

n) для n кроків (n-step return):

$$R_{t:t+n} = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V_{t+n-1}(s_{t+n}).$$

В загальному варіанті методу навчання з підкріпленням на основі часових різниць TD(λ) (n-step TD) значення оціночної ваги модифікуються (уточнюються) для всіх станів, для яких e(s)≠0 за формулою:

$$V(s) = V(s) + \alpha_t\big(r_{t+1} + \gamma V(s_{t+1}) - V(s)\big)e(s),$$

$$\forall s \in S: e(s) \neq 0,$$

де e(s) – це показник прийнятності (eligibility) стану *s*, який тим більше, чим більш нещодавно був відвіданий стан *s*.

Для поточного стану величина e(s) збільшується на 1:

$$e(s_t) = e(s_t) + 1.$$

Для решти станів величина e(s) з кожним кроком зменшується з коефіцієнтом затухання $\lambda \in [0, 1]$:

$$e(s) = \gamma\lambda e(s), \forall s \in S: e(s) \neq 0.$$

В методі навчання з підкріпленням TD(λ) використовується кортеж досвіду (experience tuple): < s, a, r, s′ >. Параметр $\lambda \in [0, 1]$ задає масштаб часу, в межах якого визначається безпосередній результат взаємодії агента з середовищем на кроці *t*. Якщо λ=0 (TD(0)), то враховується лише перехід з поточного стану у наступний стан. Якщо λ=1, то враховуються переходи між усіма станами.

Пошук оптимальної стратегії вибору дій агентом відбувається в рамках узагальненої процедури ітерації по стратегіях (generalized policy iteration, GPI) [15], в тому чи іншому варіанті її конкретної реалізації. Базовий варіант цієї процедури складається з двох етапів, які повторюються (рис.4.6):

1) на протязі T кроків (episode) для фіксованої стратегії π(s) визначається оціночна вага станів {V(s)} методом TD(λ);

2) нова стратегія π(s) обирається в такий спосіб, щоб максимізувати очікуване значення цільової функції з врахуванням знайдених {V(s)}; перехід до п.1.

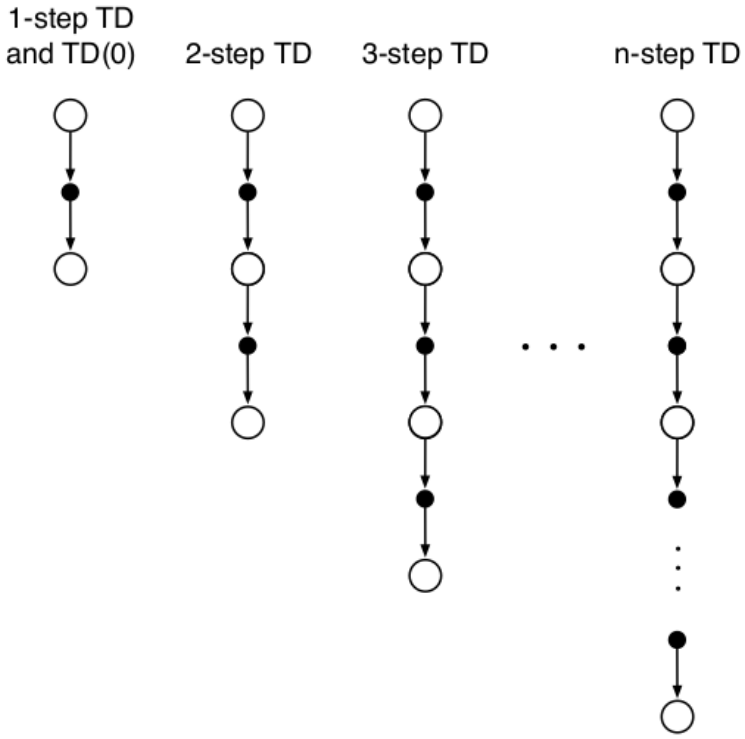


Рис. 4.5. Схема роботи методів TD(λ), в яких враховуються «сліди прийнятності» – ланцюжки нещодавно відвіданих станів середовища.

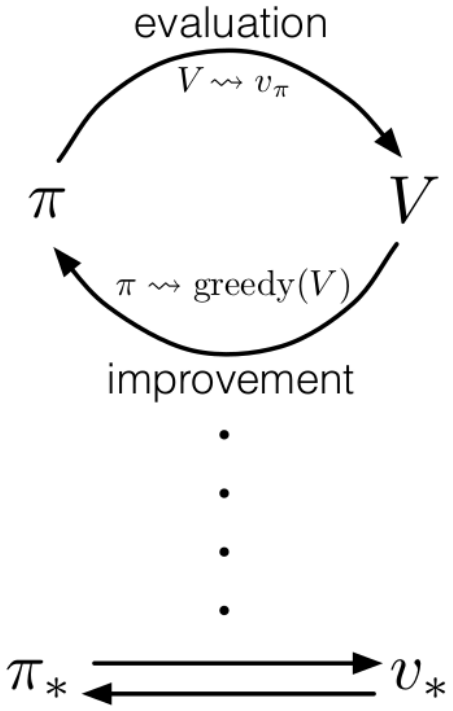


Рис. 4.6. Схема роботи процедури ітерації по стратегіях (generalized policy iteration, GPI).

4.2.3. Метод навчання з підкріпленням SARSA

Метод навчання з підкріпленням SARSA є одним з варіантів реалізації узагальненої процедури ітерації по стратегіях (generalized policy iteration, GPI) з використанням навчання на основі часових різниць (TD-learning) для прогнозування значень оціночної ваги пар (стан, дія) {Q(s,a)}, які використовуються в цьому методі замість значень оціночної ваги станів {V(s)} [15].

В методі SARSA реалізовано слідування поточній стратегії вибору агентом дій π(s) (on-policy TD control) на протязі деякого проміжку часу (епізоду) з подальшим покращенням стратегії π(s) в наступних епізодах, використовуючи знайдені значення оціночних ваг {Q(s,a)}.

Для цього використовується, так званий, кортеж досвіду (experience tuple): <s, a, r, s′, a′> або в іншому представленні:

$$(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}),$$

де S_t – поточний стан, A_t – дія реалізована в цьому стані, R_{t+1} – виграш отриманий за дію A_t, S_{t+1} – наступний стан, в який перейшло середовище, і A_{t+1} – дія реалізована в наступному стані.

Один епізод (episode) складається з T кроків взаємодії агента з середовищем. В різних епізодах реалізуються різні ланцюжки переходів між станами середовища (рис.4.7).

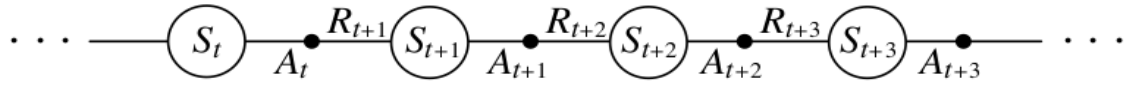


Рис. 4.7. Схема переходів між станами в межах одного епізоду навчання з підкріпленням за методом SARSA.

В методі SARSA замість переходу від стану до стану ($s \rightarrow s'$) аналізується перехід від одної пари (стан, дія) до іншої: $(s,a) \rightarrow (s',a')$. Протягом чергового епізоду покроково уточнюються значення оціночних ваг {Q(s,a)} за формулою:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)].$$

Алгоритм роботи методу SARSA наведено на рис. 4.8.

```
01: Встановити довільні початкові значення {Q(s,a)}
02: цикл по епізодах
03:     визначити початковий стан епізоду s
04:     обрати дію a в стані s згідно значень {Q(s,a)} за методом ε-greedy
05:     цикл по кроках епізоду:
06:         реалізувати дію a, отримати виграш r, визначити наступний стан s'
07:         обрати дію a' в стані s' за методом ε-greedy
08:         Q(s,a) = Q(s,a) + α[r + γQ(s',a') − Q(s,a)].
09:         s := s' , a := a'
10:     зупинитись, якщо стан s - кінцевий, інакше продовжити цикл
11: кінець циклу
```

Рис. 4.8. Алгоритм роботи методу SARSA.

4.2.4. Метод навчання з підкріпленням Q-learning

В методі Q-learning реалізовано безпосереднє оцінювання значень оціночних ваг {Q(s,a)} без слідування заданій стратегії вибору агентом дій π(s) (off-policy TD control), замість чого відбувається покрокове покращення стратегії π(s) по мірі уточнення значень оціночних ваг {Q(s,a)} [15]. Для цього використовується кортеж досвіду (experience tuple): < s, a, r, s′ >.

В результаті покрокового уточнення значень оціночної ваги всіх пар (стан, дія) {Q(s,a)}, вони з часом збігаються до дійсних (оптимальних) значень оціночної ваги: Q(s,a) → Q*(s,a). Уточнення значення оціночної ваги виконується за формулою:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right],$$

де

$Q(S_t,A_t)$ – оціночна вага пари (стан, дія),

$Q(S_{t+1},a)$ – оціночна вага пари (наступний стан, дія),

R_{t+1} – поточний виграш агента за дію A_t ,здійснену в стані S_t,

α – крок навчання, такий що $0 \le \alpha < 1$,

γ - коефіцієнт послаблення, такий що $0 \le \gamma < 1$.

Алгоритм роботи методу Q-learning для випадку, коли взаємодія агента з випадковим середовищем MDP складається з послідовності епізодів, наведено на рис. 4.9.

Приклади програмної реалізації методів навчання з підкріпленням в MDP наведені в додатку Д.7.

```
01: Встановити довільні початкові значення {Q(s,a)}
02: цикл по епізодах
03:     визначити початковий стан епізоду s
04:     цикл по кроках епізоду:
06:         обрати дію a в стані s згідно значень {Q(s,a)} за методом ε-greedy
06:         реалізувати дію a, отримати виграш r, визначити наступний стан s'
07:         Q(s,a) = Q(s,a) + α [r + γ max_a Q(s',a) − Q(s,a)]
08:         s := s'
09:     зупинитись, якщо стан s - кінцевий, інакше продовжити цикл
10: кінець циклу
```

Рис. 4.9. Алгоритм роботи методу Q-learning для випадку, коли взаємодія агента з випадковим середовищем MDP складається з послідовності епізодів.

Контрольні питання

1. В який спосіб задаються параметри випадкового середовища MDP?
2. В чому полягає основна проблема навчання з підкріпленням в випадковому середовищі MDP?
3. Чим випадкове середовище MDP відрізняється від випадкового середовища Multi-armed bandit?
4. В чому полягає різниця між випадковим середовищем MDP та випадковим середовищем з перемиканням станів?
5. Чим відрізняються модель оптимальної поведінки з обмеженим інтервалом (finite-horizon model)) та модель оптимальної поведінки з відступаючим інтервалом (receding-horizon model)?
6. Який параметр використовується в моделі оптимальної поведінки з відступаючим інтервалом (receding-horizon model)?
7. Які основні підходи використовуються до організації розрахунків для пошуку оптимальної стратегії агента для відомої моделі MDP?
8. Як відбувається навчання з підкріпленням за методом адаптивної евристичної оцінки (Adaptive Heuristic Critic)?
9. Що отримує в якості підкріплення блок RL в методі адаптивної евристичної оцінки (Adaptive Heuristic Critic)?
10. За яким принципом відбувається навчання з підкріпленням на основі часових різниць (Temporal difference learning)?
11. В який спосіб уточнюється значення оціночної ваги стану V(s) в методах навчання з підкріпленням на основі часових різниць (TD-learning)?
12. Для чого в методах навчання з підкріпленням на основі часових різниць (TD-learning) використовується показник прийнятності стану середовища?
13. Який кортеж досвіду (experience tuple) використовується в методі навчання з підкріпленням SARSA?
14. За яким принципом уточнюється значення оціночної ваги Q(s,a) в методі навчання з підкріпленням SARSA?
15. Як розраховується оціночна вага Q(s,a) у методі навчання з підкріпленням Q-learning?
16. Чим метод навчання з підкріпленням Q-learning відрізняється від методу SARSA?

5. Навчання з підкріпленням в багатоагентних системах

5.1. Багатоагентні системи (multi-agent systems)

5.1.1. Поняття багатоагентної системи

Багатоагентна система (БАС, колектив автономних агентів) – це слабозв'язаний набір автономних інтелектуальних агентів, об’єднаних частково або повністю спільною метою в рамках вирішення деякої складної задачі шляхом колективних взаємодоповнюючих зусиль [24-26]. Взаємодія багатоагентної системи з середовищем (об’єктом управління) обумовлюється тим завданням, яке ставить перед БАС розробник (рис.5.1). Характерними особливостями багатоагентної системи є

1) повна або часткова незалежність агентів у прийнятті рішень (автономність агентів);

2) обмежена інформаційна взаємодія агентів у складі БАС та обмежений (локальний) доступ агентів до середовища (об’єкта управління);

3) повна або часткова відсутність єдиного центру управління у складі БАС, що потребує дослідження та використання ресурсу децентралізованого управління.

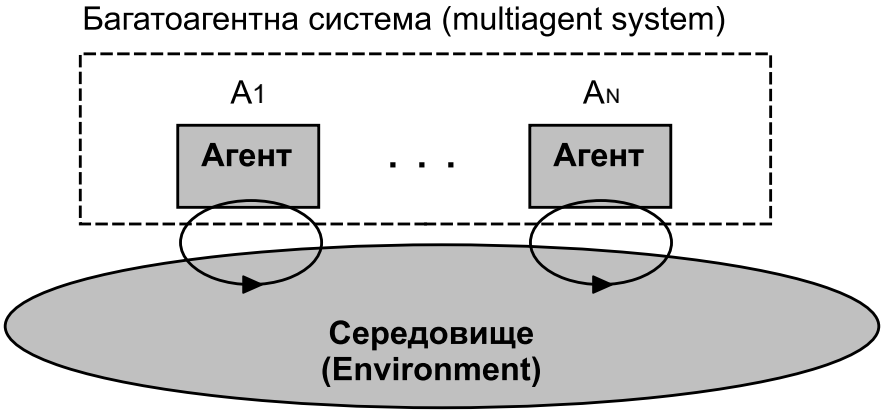


Рис. 5.1. Взаємодія багатоагентної системи з середовищем (об’єктом управління).

В рамках багатоагентної системи реалізується колективна поведінка агентів, які входять до її складу. Колективна поведінка - поведінка деякого числа автономних сутностей, об'єднаних спільними, частково спільними або протилежними цілями, в результаті чого їх дії повністю або частково взаємозумовлені.

Розглянемо різні варіанти схем взаємодії БАС з середовищем (об’єктом управління), використовуючи наступні позначення: множина агентів $\{A_i\}_n = A_1, A_2, \ldots, A_n$, де n – кількість агентів у складі БАС; множина середовищ $\{E_j\}_m = E_0, E_1, E_2, \ldots, E_m$, де m – кількість різних середовищ, з якими взаємодіють агенти; середовище $E_0=\varnothing$ – спеціальний випадок відсутності середовища. Усі можливі варіанти схем взаємодії БАС з середовищем задаються виразом:

$$TE(n,m,u) = \{A_i\}_n \times \{0,1\} \times \{E_j\}_m$$

Наприклад, схема «всі агенти розміщені в одному середовищі» задається як:

$$TE(n,1,1) = \{A_i\}_n \times \{1\} \times \{E_1\}_1,$$

а схема «всі агенти взаємодіють лише між собою за відсутності середовища задається як:

$$TE(n,0,1) = \{A_i\}_n \times \{1\} \times \{E_0\}_0.$$

Найбільш розповсюдженими в області дослідження та практичної реалізації БАС є наступні варіанти схем:

1) багато агентів – одне середовище;

2) багато агентів – декілька розділених середовищ;

3) багато агентів – середовище $E_0=\varnothing$ (тільки міжагентна взаємодія).

З точки зору організації процесів прийняття рішень в багатоагентній системі можуть використовуватись різні структурні принципи побудови розподілених систем. Серед цих принципів виділяють два основних: централізоване та децентралізоване управління. В області дослідження та розробки багатоагентних систем основну увагу приділяють децентралізованому управлінню. Питання дослідження та використання ресурсу децентралізованого управління є одним з ключових питань при побудові багатоагентних систем.

Централізоване управління – структурний принцип побудови розподілених систем, згідно якого всі рішення приймає єдиний центр управління. Центр збирає інформацію від усіх інших компонент системи та передає їм вказівки згідно прийнятого ним рішення. Централізоване управління в розподілених системах характеризується 1) потенційно високою точністю за рахунок наявності у центрі усієї доступної інформації про розподілений об'єкт управління, 2) низькою оперативністю внаслідок витрат часу на передавання даних між компонентами системи і центром, а також необхідності обробки великих об'ємів даних, що надходять від компонент системи у центр. З точки зору живучості всієї розподіленої системи центр є слабким місцем. В разі його відмови зупиняється робота всієї системи.

Децентралізоване управління – структурний принцип побудови розподілених систем, згідно якого всі рішення приймаються локально розподіленими компонентами системи за умов відсутності єдиного центру управління. Компоненти системи використовують для прийняття рішення локальну обмежену інформацію про об'єкт управління. Децентралізоване управління в розподілених системах характеризується 1) потенційно низькою точністю внаслідок відсутності у окремої компоненти системи усієї доступної інформації про розподілений об'єкт управління в момент прийняття рішення, 2) високою оперативністю за рахунок прийняття рішень компонентами, які безпосередньо взаємодіють з об'єктом управління, та обробкою у окремій компоненті системи порівняно невеликого об'єму даних. Відсутність єдиного центру управління різко підвищує живучість розподіленої системи. Така система продовжує виконувати свою функцію (можливо з втратами у продуктивності) доти, доки в робочому стані лишається хоча б одна компонента системи.

Основні задачі, які досліджуються та вирішуються в області розробки багатоагентних систем:

1) Проблеми синхронізації, спільної роботи (cooperation), координації та самоорганізації колективної поведінки агентів.

2) Співвідношення децентралізованого і централізованого управління багатоагентною системою.

3) Співвідношення однорідності та неоднорідності агентів у складі багатоагентної системи.

4) Співвідношення співпраці та суперництва агентів у складі багатоагентної системи.

5) Дослідження колективних моделей реальності (колективне знання).

6) Питання колективного прийняття рішень.

7) Питання інформаційної взаємодії агентів у складі багатоагентної системи, в тому числі мови «спілкування» агентів.

В рамках схеми взаємодії БАС з середовищем множина автономних агентів A, об'єднаних деякою спільною метою, утворює колектив автономних агентів (рис.5.2). Колективна дія – це сукупність індивідуальних дій окремих автономних агентів. Послідовність колективних дій в ході міжагентної взаємодії та взаємодії агентів з середовищем E формує колективну поведінку БАС. Таким чином колективна поведінка є результатом частково або повністю незалежних процесів прийняття рішень окремими автономними агентами, які об'єднані у складі БАС. Цільова функція всього колективу F(A) – це агрегатна функція (aggregate function), яка об'єднує цільові функції окремих агентів $\{f(a)\}$ і визначає характер та спрямування колективної поведінки. Відтак, у разі заданої цільової функції колективу постає проблема її відображення у множину цільових функцій окремих агентів:

$$F(A) \rightarrow \{f(a)\}.$$

В загальному випадку припускається, що цільові функції автономних агентів {f(a)} можуть бути різними. В більш специфічному (з точки зору особливостей синтезу) випадку додатково вимагається, щоб усі цільові функції {f(a)} були однаковими. В такий спосіб досягається максимальна живучість БАС внаслідок повної взаємозамінності окремих агентів та спрощується практична реалізація БАС за рахунок повної уніфікації конструкції агентів.

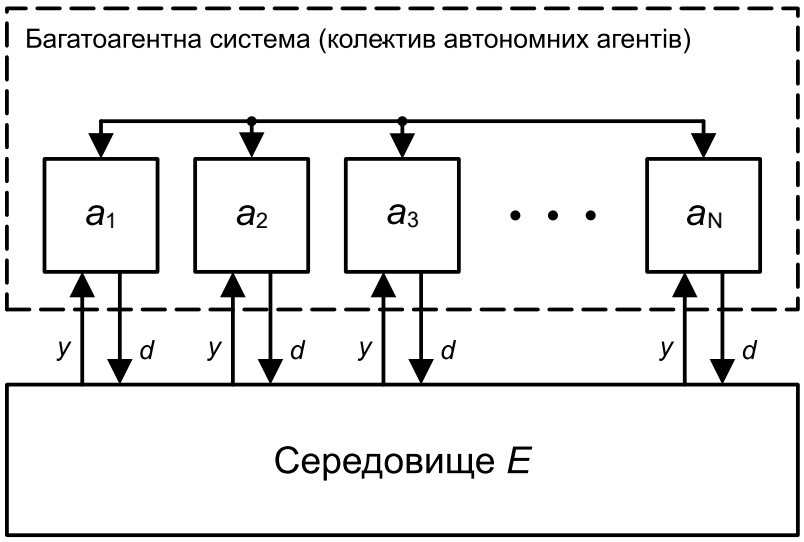


Рис. 5.2. Схема взаємодії БАС (колективу автономних агентів) з середовищем.

Після знаходження цільових функцій {f(a)} постає проблема їх відображення у характеристики сенсорної системи, функції оцінки, спосіб інформаційної взаємодії агентів, алгоритми прийняття рішень та можливості щодо їх реалізації:

$$\{\mathrm{f}(a)\} \to \{P(E,a),R(s,a),C(a),U(a),D(a)\}.$$

Як правило, цю задачу синтезу доцільно розбити на дві підзадачі:

1) вибір і забезпечення сенсорних, комунікаційних та виконавчих можливостей автономних агентів $\{P(E,a),C(a),D(a)\}$;

2) розробка алгоритмів колективної поведінки $\{R(s,a),U(a)\}$, ґрунтуючись на заданих можливостях автономних агентів.

Разом з тим розглядаються також інші питання, пов’язані з витратами енергії, забезпеченням необхідної обчислювальної потужності та ін.

Основні вимоги, які висуваються до алгоритмів колективної поведінки, наступні:

1) робота в реальному масштабі часу: вибір рішення окремим автономним агентом має займати деякий незмінний проміжок часу, який не перевищує заданої величини затримки;

2) локальність поведінки: алгоритм колективної поведінки має бути поданий у вигляді сукупності локальних алгоритмів індивідуальної поведінки автономних агентів;

3) локальність взаємодії: алгоритм колективної поведінки має коректно працювати в умовах обмеженої інформаційної взаємодії агентів (наприклад, в умовах обмеженого радіусу видимості засобів зв’язку агентів);

4) уніфікованість («однаковість»): всі автономні агенти мають виконувати один і той самий локальний алгоритм (для випадку однакових цільових функцій {f(a)});

5) незалежність роботи локальних алгоритмів від поточної кількості автономних агентів: алгоритм колективної поведінки має продовжувати працювати коректно, не дивлячись на зміни чисельності колективу (наприклад, внаслідок виходу деяких автономних агентів з ладу або приєднання до колективу нових автономних агентів).

Відмітимо, що останні дві вимоги в багатьох випадках розглядаються як занадто жорсткі і не приймаються до уваги.

Процес створення БАС складається з трьох основних кроків:

1) визначення цільової функції колективу автономних агентів F(A);

2) визначення цільових функцій окремих автономних агентів $\{f(a)\}$, виходячи з F(A);

3) забезпечення необхідних можливостей автономних агентів $\{P(E,a),C(a),D(a)\}$ та розробка алгоритмів колективної поведінки $\{R(s,a),U(a)\}$, виходячи з $\{f(a)\}$.

Основна проблематика створення БАС з точки зору способів об'єднання агентів у складі одної системи може бути розділена на три рівні складності (в порядку зростання):

1) забезпечення співпраці агентів (cooperation);

2) координація спільних узгоджених дій агентів (coordination);

3) самоорганізація колективу агентів (self-organization, emergent collective behavior).

Основні наукові напрямки дослідження принципів колективної поведінки автономних агентів:

1) колективна поведінка цілеспрямованих автоматів (автоматів, що навчаються),

2) distributed artificial intelligence (DAI),

3) multi-agent systems,

4) swarm intelligence,

5) distributed (collective) robotics (в тому числі swarm robotics),

6) artificial life,

7) collective intelligence.

Одним з напрямків досліджень, який в багатьох питаннях перекликається з дослідженнями в області багатоагентних систем є, так званий, ройовий інтелект (swarm intelligence). Ройовий інтелект – це концепція в області штучного інтелекту, згідно якої колективна поведінка простих елементів деякої децентралізованої системи за певних умов призводить до її самоорганізації, тобто виникнення функціональних можливостей вищого рівня, які не зводяться до простої суми функціональних можливостей окремих елементів. Системи ройового інтелекту, як правило, складаються з сукупності простих агентів, що локально взаємодіють між собою та своїм оточенням. Основним джерелом ідей для побудови таких систем, як правило, слугують різні біологічні системи і, в першу чергу, колонії суспільних комах (мурахи, бджоли, оси, терміти, деякі види павуків). Агенти в таких системах використовують невеликий набір простих правил, і за відсутності центру управління, в умовах локальної взаємодії з елементами випадковості, забезпечують виникнення інтелектуальної поведінки на рівні всієї системи. Серед прикладів ройового інтелекту в природних системах – колонії мурах, зграї птахів, популяції бактерій та ін. Одним з прикладів застосування цієї концепції є ройова робототехніка (swarm robotics). З практичної точки зору, концепція ройового інтелекту знайшла найбільше використання у вирішенні задач пошуку та оптимізації, зокрема задач комбінаторної оптимізації.

Наукові напрямки, ідеї та результати яких використовуються у дослідженні принципів колективної поведінки автономних агентів:

1) теорія ігор;

2) адаптивне управління, зокрема дуальне управління;

3) децентралізоване управління;

4) машинне навчання, зокрема навчання з підкріпленням;

5) синергетика, зокрема принципи самоорганізації;

6) клітинні автомати (cellular automata);

7) паралельні та розподілені обчислення;

8) одноранговi мережні технології (p2p).

Приклади областей практичного застосування багатоагентних технологій:

1) автоматизація збору наукових даних, в тому числі розв'язок задач автономних розподілених досліджень (приклад: проект ARGO — система автономних океанологічних досліджень на основі автономних дрифтерів);

2) автоматизація наукових досліджень (приклад: управління науковими завданнями в міжнародній розподіленій мережі телескопів, в тому числі планування та диспетчеризація наукових експериментів та виділення засобів для їх проведення);

3) колективна поведінка інформаційних агентів (приклад: пошук інформації в мережі Інтернет колективом пошукових агентів в рамках проекту InfoSpiders).

4) розподілена (колективна) робототехніка, зокрема поліморфна робототехніка (приклад: колективи автономних колісних роботів для картографування невідомої місцевості (mapping), дослідження плану невідомого приміщення (exploration), пошуку заданого об'єкту (search), відслідковування переміщень деякого об'єкту (tracking), патрулювання заданого периметру охорони і т.д.);

5) автономні транспортні системи (приклад: автоматизована складська система (automated storage and retrieval system) компанії Amazon, розроблена її підрозділом Amazon Robotics (колишня Kiva Systems); всього на складах компанії Amazon по всьому світу працює більше 100 тис. роботів);

6) моделювання та візуалізація колективної поведінки людей (приклад: система моделювання та візуалізації просторової динаміки великих колективів MASSIVE (Multiple Agent Simulation System in Virtual Environments), яка використовувалась у створенні фільмів «The Lord of the Rings», «Avatar», «I,Robot», «WALL-E» та ін.).

5.1.2. Концептуальна модель багатоагентної системи

Для опису загальної концептуальної моделі багатоагентної системи (колективу автономних агентів) зручно розглянути окремо дві групи моделей-компонент:

1) моделі функціонування колективу автономних агентів, які відображають різні аспекти побудови та режиму роботи колективу;

2) моделі об'єднання автономних агентів у колектив, які відображають різні аспекти (само-)організації колективної поведінки.

До моделей функціонування колективу автономних агентів можна віднести наступні моделі:

1) $\mathbf{M_a}$ – модель автономного агента, як складового елементу колективу. Ця модель визначає можливості окремого автономного агента по сприйняттю станів середовища і впливу на нього, а також «когнітивні» можливості агента з точки зору процесу прийняття рішень, тобто рівень складності його архітектури.

2) $\mathbf{M_c}$ – модель інформаційної взаємодії автономних агентів. Ця модель визначає:

2.1) присутність або відсутність можливості інформаційної взаємодії між агентами (самовиявлений та несамовиявлений колектив);

2.2) спосіб ідентифікації одним автономним агентом інших агентів колективу;

2.3) спосіб та порядок обміну інформацією між автономними агентами (в тому числі якісні показники швидкості обміну та відповідного об'єму інформації, яким можуть обмінятись автономні агенти за одиницю часу).

3) $\mathbf{M_n}$ - модель чисельного складу колективу автономних агентів. Ця модель визначає кількісні характеристики колективу та характер їхніх змін у часі. При цьому порядок кількості агентів у колективі, який визначається цією моделлю (наприклад, 10, 100, 1000,…), суттєво впливає на роботу колективу, так як із зростанням кількості агентів, як правило, зростає «хаотична складова» в колективній поведінці [8]. На окрему увагу також заслуговують способи моделювання процесів зростання та скорочення чисельності агентів, наприклад, на основі механізмів популяційної динаміки.

Розглянемо більш детально способи ідентифікації одним автономним агентом інших агентів колективу. З точки зору можливостей координації спільних дій автономних агентів можна запропонувати наступні варіанти способів ідентифікації (у порядку зростання відповідних можливостей координації):

1) даний агент не розрізняє виявлених «сусідніх» агентів між собою і не може встановити їхню кількість (результат ідентифікації: факт наявності або відсутності «сусідніх» агентів);

2) даний агент не розрізняє виявлених «сусідніх» агентів, але може встановити їхню

кількість (додатковий результат ідентифікації: кількість «сусідніх» агентів (можливо не точна кількість, а деяке відповідне порогове значення));

3) даний агент розрізняє виявлених «сусідніх» агентів між собою, може визначити їхню кількість, але агенти не мають унікальних ідентифікаторів в межах всього колективу (додатковий результат ідентифікації: масив «сусідів», в якому ідентифікатор «сусіднього» агента дорівнює індексу в масиві);

4) даний агент розрізняє виявлених «сусідніх» агентів між собою, може визначити їхню кількість, і всі агенти мають унікальні ідентифікатори в межах всього колективу (додатковий результат ідентифікації: попередня історія «сусідства», яка може бути використана для підвищення ефективності координації спільних дій автономних агентів).

Крім цього на процедуру ідентифікації інших агентів можуть накладатись певні обмеження, наприклад, у вигляді заданої максимальної кількості «сусідніх» агентів (обмеження зверху), які можуть бути виявлені даним агентом в один момент часу (як умова гарантованої швидкодії роботи процедури ідентифікації або як наслідок обмеженої пам'яті автономного агента).

До моделей об'єднання автономних агентів у колектив можна віднести наступні моделі:

1) $\mathbf{M_{dc}}$ – модель співвідношення децентралізованого та централізованого управління процесами спільного прийняття рішень автономними агентами. Ця модель визначає ступінь незалежності окремого агента від інших агентів колективу з точки зору механізму прийняття рішень в рамках деякої структури взаємного підпорядкування агентів один одному. В рамках цієї моделі виконання одним агентом наказів іншого розглядається як елемент централізованого управління, в той час, як самостійне прийняття рішення агентом розглядається як елемент децентралізованого управління. При цьому можна висунути гіпотезу, що максимальна децентралізація управління (відсутність будь яких елементів централізації) в повному обсязі розкриває всі переваги ідеї самоорганізації.

2) $\mathbf{M_{hg}}$ – модель співвідношення однорідності та неоднорідності автономних агентів колективу. Ця модель визначає 2.1) який зміст вкладається в поняття однорідності (неоднорідності) агентів (наприклад, можна розрізняти «однаковість» агентів з точки зору їх функціональних можливостей (сенсорних, комунікаційних, виконавчих) та «однаковість» агентів з точки зору їх «когнітивних» можливостей) та 2.2) яке співвідношення «однакових» та «різних» автономних агентів у складі колективу реалізовано (наприклад, в повністю однорідному колективі – усі автономні агенти «однакові» в тому, чи іншому розумінні). В рамках цієї моделі розкриваються поняття функціональної спеціалізації автономних агентів та взаємодоповнення різних функціональних ролей, які обирають для себе автономні агенти.

3) $\mathbf{M_{cc}}$ – модель співвідношення співпраці та суперництва між автономними агентами колективу. Ця модель визначає ступінь неспівпадіння індивідуальних інтересів окремих автономних агентів і загального колективного інтересу, тобто силу об'єднання автономних агентів спільним колективним інтересом (або ступінь інтегрованості автономних агентів у склад колективу). В рамках цієї моделі узгоджене сумісне виконання дій агентами розглядається як елемент співпраці, а виконання суперечливих дій, які призводять до конфліктів між агентами, розглядається як елемент суперництва (наприклад, в ситуаціях конкуренції за одиниці спільного ресурсу).

Таким чином на основі запропонованих моделей функціонування та моделей об'єднання можна побудувати наступну концептуальну модель колективу автономних агентів:

$$M(k) = \{(\mathbf{M_a}, \mathbf{M_c}, \mathbf{M_n})\} \times \{(\mathbf{M_{dc}}, \mathbf{M_{hg}}, \mathbf{M_{cc}})\},$$

різні реалізації якої k=1,2,… задаються вибором вигляду відповідних моделей-компонент та визначенням їхніх параметрів.

5.1.1. Алгоритмічне забезпечення багатоагентних систем

Виходячи з ідеї самоорганізації колективу агентів, а також спираючись на

1) принцип ієрархії («вкладеності») процесів управління та прийняття рішень, застосування якого дозволяє абстрагуватись від специфіки способу організації переміщення та інших аспектів роботи окремого агента, та

2) принцип функціональної декомпозиції (розбиття задачі на підзадачі), можна запропонувати наступний набір базових службових алгоритмів колективної поведінки агентів, який складається з двох частин

- організаційної (А1):

А1-1. Самовиявлення колективу агентів,

А1-2. Самоіменування колективу агентів,

А1-3. Самоузгодження колективу агентів,

- функціональної (А2):

А2-1. Самоорганізація колективу агентів в просторі,

А2-2. Самоорганізація колективу агентів в часі,

А2-3. Самоорганізація колективу агентів за параметром.

Алгоритми, які входять до організаційної частини набору А1, відповідають за підтримку організаційної інфраструктури колективу агентів. Під самовиявленням колективу розуміється отримання кожним агентом інформації про інших агентів (які, наприклад, знаходяться в зоні видимості засобів детектування або зв'язку даного агента) з метою сформувати «зв'язний» колектив. Формат та зміст цієї інформації визначається розробником в рамках відповідної моделі інформаційної взаємодії агентів. Під самоіменуванням розуміється процес породження множини унікальних в межах колективу імен (ідентифікаторів) агентів, тобто перехід від набору можливо однакових величин-ідентифікаторів агентів до набору гарантовано різних величин-ідентифікаторів, в тому числі в умовах коливання чисельності колективу (вибуття одних агентів і входження в колектив інших агентів). Під самоузгодженням колективу агентів розуміється процес погодження усіма агентами, що входять у колектив, деякої однакової для всіх величини, тобто перехід від набору можливо різних величин до набору однакових для всіх агентів величин (задачі цього класу також називають задачами на пошук консенсусу).

Алгоритми, які входять до функціональної частини набору А2, забезпечують базову функціональність колективу агентів. Під самоорганізацією колективу агентів в просторі розуміється здатність колективу цілеспрямовано управляти розміщенням та переміщенням своїх представників у просторі, виходячи з поставлених перед ним задач більш високого рівня. До алгоритмів просторової самоорганізації зокрема відносяться: 1) впорядковане розміщення агентів у просторі (розгортання колективу): рівномірне заповнення обмеженого деякими границями простору; формування правильної решітки з заданим кроком; формування гнучкої решітки, форма якої залежить від характеристик оточуючого середовища; 2) «формування» геометричних фігур: лінії, кола, квадрату, тощо із заданим кроком між агентами; 3) впорядковане переміщення агентів у просторі (узгоджене групове переміщення, переміщення за схемою лідер-наслідувач, слідування заданій траєкторії руху, тощо); 4) уникнення зіткнень з іншими агентами в процесі переміщення; 5) колективне подолання (обминання) перешкод та ін.

Під самоорганізацією колективу агентів в часі розуміється процес синхронізації дій агентів у фізичному часі або логічному часі (з урахуванням лише послідовності подій) за умов відсутності «зовнішнього годинника» (тобто єдиного центра, який забезпечує «примусову» синхронізацію), локальної обмеженої взаємодії між агентами та змінних невідомих наперед затримок при обміні синхронізуючими сигналами. Під самоорганізацією колективу агентів за параметром розуміється здатність агентів узгоджувати свої дії (наприклад, переміщення) на основі показів своїх сенсорних підсистем. До алгоритмів параметричної самоорганізації зокрема відносяться: 1) рух колективу агентів вздовж лінії рівня деякого параметру оточуючого середовища (наприклад, температури); 2) виявлення та оточення зони збурень

оточуючого середовища (наприклад, виявлення та оточення нафтової плями); 3) рівномірний розподіл агентів в межах виявленої зони збурень; 4) уникання агентами зони збурень; 5) супроводження агентами зони збурень, що переміщується в просторі та ін.

Основні вимоги, які висуваються до базових алгоритмів колективної поведінки з набору {A1,A2} наступні: 1) робота в реальному масштабі часу: вибір рішення окремим агентом має займати деякий незмінний проміжок часу, який не перевищує заданої величини затримки; 2) локальність поведінки: алгоритм колективної поведінки має бути поданий у вигляді алгоритмів індивідуальної поведінки агентів; 3) локальність взаємодії: алгоритм колективної поведінки має коректно працювати в умовах обмеженої інформаційної взаємодії агентів (наприклад, в умовах обмеженого радіусу видимості засобів зв'язку агентів); 4) уніфікованість («однаковість»): всі агенти мають виконувати один і той самий локальний алгоритм (для випадку повністю однорідного колективу агентів); 5) незалежність від кількості агентів: алгоритм колективної поведінки має продовжувати працювати коректно, не дивлячись на зміни чисельності колективу (наприклад, внаслідок виходу деяких агентів з ладу). Основними показниками ефективності роботи алгоритмів {A1,A2} є час виконання алгоритму (мета - мінімізація кількості часових кроків) та використання енергетичних, обчислювальних та комунікаційних ресурсів (мета - мінімізація витрат ресурсу). Окремо може розглядатись питання «функціональної повноти» запропонованого набору службових алгоритмів {A1,A2}.

Таким чином згідно узагальненого підходу пошук рішень проблеми організації узгоджених колективних дій агентів може вестись одночасно на двох рівнях: 1) інфраструктурному рівні (розробка службових алгоритмів в рамках набору {A1,A2}) та 2) прикладному рівні (розробка складних процедур колективної поведінки з використанням службових алгоритмів {A1,A2} в якості «готових» складових елементів). В свою чергу складні процедури колективної поведінки прикладного рівня, які можуть бути достатньо формалізовані (до них, наприклад, відносяться процедури пошуку, патрулювання, моніторингу та ін.) також можуть бути виділені в окремий проміжний рівень і використовуватись в подальшому в якості «готових» рішень.

Багаторівнева схема організації програмно-алгоритмічного забезпечення багатоагентної системи містить наступні основні модулі (рис.5.3):

1) Базові службові алгоритми:
   - 1.1) утворення колективу (team formation control),
   - 1.2) самоузгодження (consensus dynamics control),
   - 1.3) самоіменування (symmetry breaking control);
   - 1.4) локальна навігація в просторі (local navigation);

2) Базові алгоритми самоорганізації:
   - 2.1) самоорганізація в просторі (formation control),
   - 2.2) самоорганізація в часі (self-synchronization),
   - 2.3) параметрична самоорганізація (parametric self-organization);

3) Алгоритми колективної поведінки:
   - 3.1) багатоагентна координація (multi-agent coordination),
   - 3.2) колективний пошук (cooperative search),
   - 3.3) колективне планування (cooperative planning),
   - 3.4) колективне навчання (cooperative learning).

Наведена схема побудована за принципом багаторівневості, згідно якого модулі кожного наступного рівня схеми спираються на функціональність, реалізовану у модулях нижчого рівня. Інший принцип, закладений в схему, – це функціональна повнота, тобто реалізація всіх базових функціональних компонент для забезпечення можливості організувати на вищому рівні колективну поведінку будь якої складності для вирішення задач колективної поведінки широкого спектру.

Приклади задач колективної поведінки автономних мобільних агентів:

1. Задача впорядкованого розташування у заданій області простору.
2. Задача рівномірного розподілу обмеженої області простору.
3. Задача колективного пошуку одного або декількох об'єктів.

4. Задача пошуку та розподілення об’єктів для моніторингу.

5. Задача подолання лабіринту.

6. Задача патрулювання, детектування порушників та відслідковування траєкторій їх руху.

7. Задача виявлення та оточення зони збурення.

8. Задача колективного розвідування оточуючого середовища (cooperative environment exploration).

9. Задача автономних розподілених досліджень (рис.5.4).

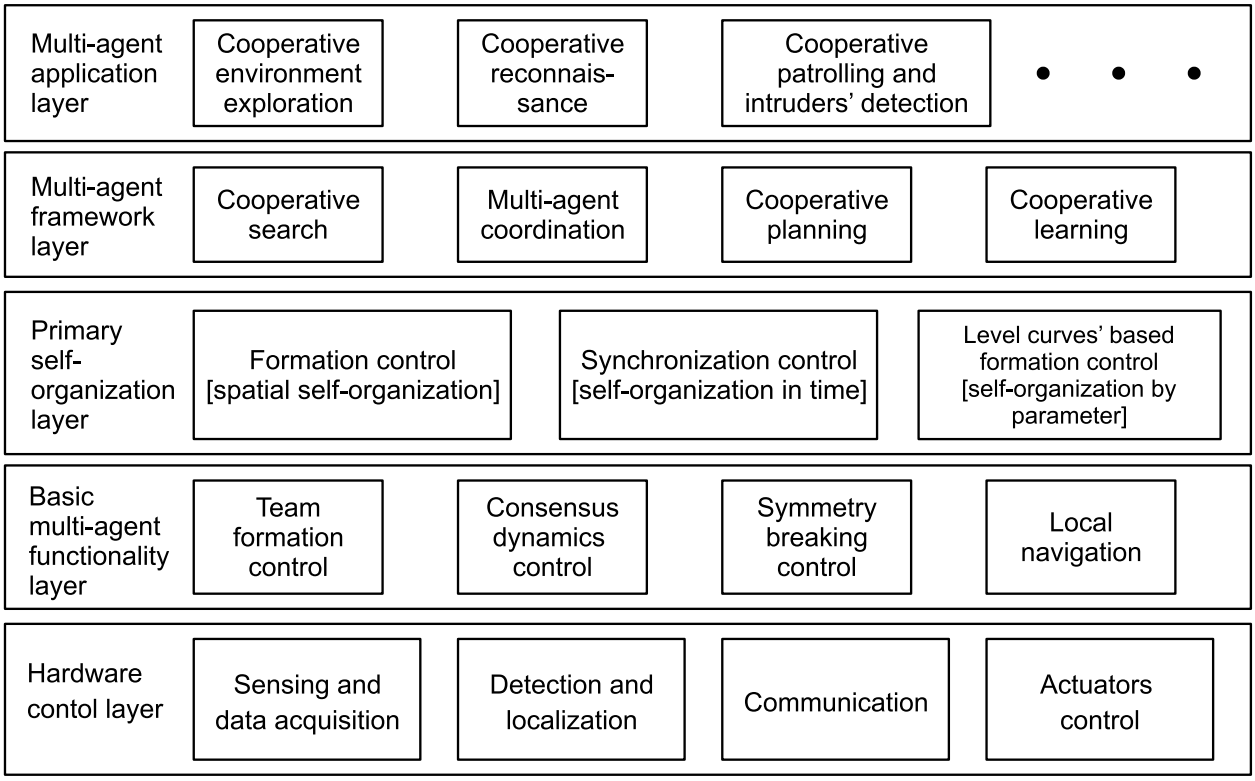


Рис. 5.3. Багаторівнева схема організації програмно-алгоритмічного забезпечення багатоагентної системи.

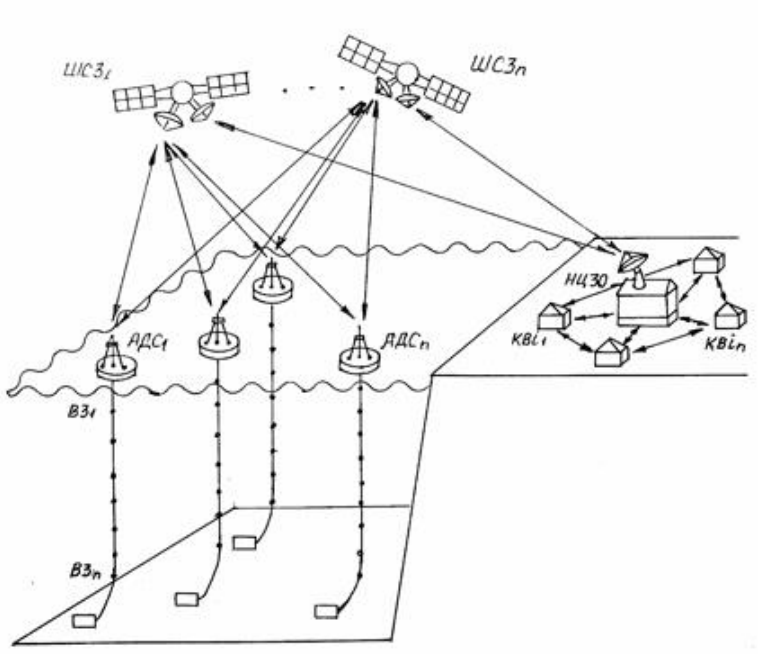


Рис. 5.4. Задача автономних розподілених досліджень Світового Океану.

5.1.2. Механізми координації колективної поведінки інтелектуальних агентів

Координація колективної поведінки – це процес інформаційної взаємодії агентів, який дозволяє організувати їхні спільні узгоджені дії по вирішенню поставленої задачі, тобто забезпечує цілеспрямовану колективну поведінку за умов відсутності єдиного центру управління. Механізм (спосіб) координації – це набір правил, яким підпорядковуються агенти у своїй поведінці. Ці правила спрацьовують в результаті обміну між агентами координаційними повідомленнями.

Ігровий механізм координації передбачає використання положень теорії ігор для побудови набору правил та способу обміну координаційними повідомленнями. Ідея ігрових механізмів координації полягає у використанні рівноваги Неша (розв'язку фіктивної координаційної гри) в якості мети, на досягнення якої скеровуються дії агентів. Відтак пошук рішення координаційної гри реалізує процес координації спільних дій агентів. Для організації такого пошуку, можуть бути використані автомати, що навчаються, та методи навчання з підкріпленням.

Гра з нульовою сумою (zero-sum game) – термін теорії ігор, який означає гру, в якій інтереси гравців прямо протилежні. Відтак сума виграшів гравців в такій грі дорівнює нулю (рис. 5.5).

		B	
		B_1	B_2
A	A_1	+1;-1	-1;+1
	A_2	-1;+1	+1;-1

Рис. 5.5. Приклад гри з нульовою сумою (zero-sum game).

Гра з ненульовою сумою (general sum game) – термін теорії ігор, який означає гру, в якій інтереси гравців частково протилежні і частково співпадають. Відтак сума виграшів гравців в такій грі більше нуля (рис.5.6).

		B	
		B_1	B_2
A	A_1	2;1	-1;-1
	A_2	-1;-1	1;2

Рис. 5.6. Приклад гри з ненульовою сумою (general sum game):
гра «сімейна суперечка» (battle of the sexes).

Рівновага Неша (Nash equilibrium) – такий набір стратегій гравців у грі, в якому жоден учасник не може збільшити свій виграш, змінивши свою стратегію, якщо інші учасники своїх стратегій не змінюють. Це рішення гри названо на честь Джона Неша (John Nash), який довів існування рівноваги в змішаних стратегіях в будь-якій скінченій грі.

Координаційна гра (coordination game) – поняття з теорії ігор, що означає спеціальну гру, в якій перед гравцями стоїть дилема щодо вибору між двома чи більше рішеннями гри, кожне з яких в тому чи іншому аспекті задовольняє гравців і в той же час є результатом лише їх спільного (скоординованого) вибору. Відтак в координаційній грі до проблеми вибору стратегії додається проблема вибору способу реагування на поведінку інших гравців.

Координаційні ігри широко застосовуються для моделювання поведінки агентів у складі багатоагентних систем, в тому числі з метою розробки механізмів координації колективної поведінки. Одною з найбільш відомих та досліджених координаційних ігор є, так звана, дилема в'язня (prisoner's dilemma).

Приклад координаційної гри. Дилема в'язня (prisoner's dilemma) – координаційна гра двох гравців з ненульовою сумою, в якій кожний з гравців обирає між двома стратегіями: співпрацювати чи протидіяти іншому гравцю (рис.5.7-5.9). Рішенням гри з точки зору рівноваги Неша (тобто стійкої рівноваги) є <протидія, протидія>. Тобто з позицій раціональної поведінки обидва гравці мають обрати це рішення. В той же час рішення гри <співпраця, співпраця> за умовами гри дає обом гравцям більший виграш ніж рівновага Неша. Відтак поводячись окремо раціонально, разом учасники приходять до нераціонального рішення. Аналіз дилеми в'язня ускладнюється, коли гра стає ітераційною, тобто коли розіграш гри повторюється у часі, і гравці пам’ятають результати попередніх партій.

		B	
		C	D
A	C	R_1, R_2	S_1, T_2
	D	T_1, S_2	P_1, P_2

Рис. 5.7. Матриця гри «дилема в'язня» в узагальненій формі.

Кожний з гравців має дві можливості: співпрацювати (C – cooperate) чи протидіяти (D – defect). Ідея гри полягає в тому, що, якщо обидва гравці співпрацюють, то вони виграють (R), але, якщо один з них починає протидіяти, то він виграє більше (T > R). Якщо ж обидва гравці протидіють, то вони програють (P), але не так багато як програє «ошуканий» гравець в попередньому випадку (S < P). Дилема ув’язненого має зміст, коли T>R>P>S, де R>(S+T)/2.

		B	
		C	D
A	C	3;3	0;5
	D	5;0	1;1

Рис. 5.8. Приклад гри «дилема в'язня»: T = 5, R = 3, P = 1, S = 0.

		B	
		C	D
A	C	0.9;0.9	0;1
	D	1;0	0.1;0.1

Рис. 5.9. Приклад гри «дилема в'язня»: T = 1, R = 0.9, P = 0.1, S = 0.

Ітераційна дилема в'язня (iterated prisoner’s dilemma, IPD) – це гра з багатокроковим повторенням, в якій кожний з опонентів пам’ятає деяку кількість результатів минулих розіграшів (в межах своєї глибини пам’яті). Згідно теорії ігор оптимальною (стійкою за Нешем) стратегією в однокроковій IPD є протидія (D). В той же час при збільшені кількості кроків виникає задача максимізації середнього виграшу окремого гравця, яка не має тривіального рішення. Це означає, що постійна протидія не гарантує максимального середнього виграшу.

При дослідженні ітераційної дилеми ув’язненого розглядають наступні стратегії поведінки гравця (IPD-агента):

1) ALLC: завжди співпрацювати;

2) ALLD: завжди протидіяти;

3) TFT (Tit for tat, «зуб за зуб»): на першому кроці співпрацювати, на всіх наступних кроках застосовувати ту стратегію, яка була застосована суперником на попередньому кроці (спочатку співпрацювати, потім «віддзеркалювати» дії опонента); варіанти: Suspicious Tit For Tat (STFT), Tit For Two Tats (TF2T), Generous Tit For Tat (GTFT);

4) Grim («невблаганний»): співпрацювати до першої протидії, потім завжди протидіяти;

5) Stochastic: обирати співпрацю та протидію випадково з фіксованими ймовірностями (стаціонарне випадкове середовище) та ін.

Гру «дилема в'язня» можна узагальнити на N гравців. Розглянемо наступний варіант такого узагальнення. У гру грають N гравців, кожний з яких в кожній партії гри може обрати одну з двох стратегій: співпрацю (C) чи суперництво (D). Далі будемо використовувати наступні позначення: m - деяка фіксована «гранична» кількість гравців (така, що $1 \le m \le N-1$, m = const); k(C) - кількість гравців, які обрали співпрацю; k(D) - кількість гравців, які обрали суперництво, $k(C) + k(D) = N$, $0 \le k(C) \le N$, $0 \le k(D) \le N$. При цьому якщо k(C)=k, то k(D)= N-k, $0 \le k \le N$. Тоді матриця виграшів для i-го гравця (A_i), що визначає його виграш в залежності від вибору стратегій усіма іншими гравцями $\{A_j\}, j \ne i$, буде мати наступний вигляд (рис.5.10). В найбільш загальному випадку виграші $T_i(k)$, $R_i(k)$, $S_i(k)$, $P_i(k)$ – це функції від k (тобто від кількості гравців, які в даній партії гри обрали співпрацю). В більш простому випадку можна вважати що виграші – це деякі константи {T,R,P,S}.

		$\{A_j\}, j \ne i$	
		$k(C) \ge m$	$k(D) > (N-m)$
A_i	C	$R_i(k)$	$S_i(k)$
	D	$T_i(k)$	$P_i(k)$

Рис. 5.10. Матриця гри «дилема в'язня» у формі узагальненій на N гравців.

Окрім гри «дилема в'язня» для побудови механізмів координації колективної поведінки можуть також застосовуватись інші координаційні ігри (табл.5.1).

Таблиця 5.1. Приклади координаційних ігор.

Співвідношення виграшів	Гра для 2 гравців	Гра для N гравців
(D,C)>(C,C)>(D,D)>(C,D) T > R > P > S T = 5, R = 3, P = 1, S = 0	дилема ув’язненого (prisoner’s dilemma)	дилема запрошених на обід (diner’s dilemma)
(C,C)>(D,C)>(D,D)>(C,D) R > T > P > S R = 5, T = 3, P = 1, S = 0	охота на оленя (stag hunt)	гра у бунт (mutiny scenario)
(D,C)>(C,C)>(C,D)>(D,D) T > R > S > P T = 5, R = 3, S = 1, P = 0	гра у сміливішого (game of chicken)	дилема волонтерів (volunteer's dilemma)

5.2. Моделі та методи навчання з підкріпленням в багатоагентних системах

5.2.1. Задача навчання з підкріпленням в багатоагентній системі

В задачі багатоагентного навчання з підкріпленням агенти колективу мають повністю або частково спільну мету навчання, що відповідає тим задачам, які поставив перед колективом агентів розробник. В такому випадку зміст колективного навчання з підкріпленням полягає у:

1) здобутті та подальшому використанні досвіду колективних дій (наприклад, за рахунок неявної взаємодії між агентами шляхом здійснення дій в середовищі та аналізу їх наслідків),

2) використанні одними агентами досвіду набутого іншими агентами (цей досвід, наприклад, може передаватися в явному вигляді у формі відповідних інформаційних повідомлень).

Основний момент полягає в тому, що кожний окремий агент отримує відгук середовища не на власну індивідуальну дію, а на сумісні дії всього колективу (тобто так звану колективну дію). В більшості випадків це різко ускладнює процес навчання з підкріпленням, оскільки з'являється додаткова невизначеність щодо того, дії яких агентів призвели до отриманого усім колективом результату (позитивного або негативного). З іншого боку можна зробити припущення, що за рахунок колективних зусиль навчання може відбуватися значно швидше в порівнянні з індивідуальним навчанням, наприклад, за рахунок використання кожним окремим агентом досвіду інших агентів колективу.

Розглянемо приклад модифікації задачі навчання з підкріпленням в MDP для випадку багатоагентної системи (табл.5.2). В даному випадку використано наступні позначення: N – кількість агентів в колективі, A_i – множина доступних і-му агенту дій, S – множина станів середовища, R – функція виграшу (функція підкріплення), T – функція переходів. Замість стохастичних функцій виграшу та переходу з двома аргументами (Individual RL) розглядаються функції виграшу та переходу з N незалежними аргументами (по кількості агентів). В даному випадку можна зробити висновок про різке ускладнення задачі, по-перше за рахунок зростання її розмірності, і, по-друге, за рахунок виникнення нових факторів невизначеності.

Таблиця 5.2. Задача навчання з підкріпленням для одного агента та багатоагентної системи.

	Правило нарахування виграшів (reward rule) та правило переходу (transition rule)	Функція виграшу (reward function) та функція переходу (transition function)
Індивідуальне навчання з підкріпленням (Individual RL)	$R: S\times A\rightarrow \Re$ $T: S\times A \rightarrow S$	$R(s,a) \rightarrow r_t$, $r_t\in\Re$ $T(s,a) \rightarrow s'$, $s,s'\in S$
Багатоагентне навчання з підкріпленням (Multiagent RL)	$R: S\times A_1\times\ldots\times A_N\rightarrow\Re$ $T: S\times A_1\times\ldots\times A_N \rightarrow S$	$R(s,a_1,a_2,\ldots,a_N) \rightarrow \{r_{i,t}\}$ $T(s,a_1,a_2,\ldots,a_N) \rightarrow s'$

Стосовно задачі навчання з підкріпленням в багатоагентній системі можна зробити наступні зауваження.

1) Кожен з агентів може брати участь одночасно у кількох колективних та можливо індивідуальних процесах навчання. Виникає проблема координації (диспетчеризації) цих процесів.

2) Можна розглядати різні моделі колективної поведінки з різними сценаріями процесу навчання. Наприклад, можна розглядати ситуацію, коли в кожен момент часу дозволяється реалізувати дію лише одному агенту. Відповідно в кожному такті взаємодії з середовищем колектив має обирати найкращого з певних міркувань агента (наприклад, такого, який найчастіше за всіх інших потрапляв у подібні ситуації в минулому).

3) Спосіб колективного навчання напряму залежить від моделі інформаційної зв'язності, яка реалізована в колективі: різні за змістом і складністю методи навчання вимагають різних за пропускною здатністю та структурою зв'язків системи інформаційної взаємодії агентів колективу.

Ключовою проблемою багатоагентного навчання з підкріпленням є розпізнавання стану та інтерпретації відгуку середовища [24,27-29]. В загальному випадку проблема інтерпретації відгуку середовища (credit-assignment problem, CAP) полягає у визначенні того, яка дія або послідовність дій системи (окремого агента або колективу) призвели до отриманого виграшу (програшу) на даному кроці взаємодії з середовищем. Для багатоагентних систем загальну CAP зручно розбити на дві підпроблеми:

- внутрішньо-агентна CAP (intra-agent CAP): яка дія або який елемент рішення вплинули на отриманий окремим агентом виграш (програш) в біжучому такті взаємодії з середовищем (тобто, в який спосіб перераховувати вагові коефіцієнти дій та станів середовища в методі навчання з підкріпленням),

- між-агентна CAP (inter-agent CAP): які дії, яких агентів, в якому відношенні призвели до отримання колективом виграшу (програшу) в біжучому такті взаємодії з середовищем (тобто, в який спосіб розподілити колективний виграш між агентами).

На практиці ці дві проблеми вирішуються одночасно, тобто в більшості методів колективного навчання не має чіткого поділу між цими проблемами. Розглянемо приклад між-агентної проблеми інтерпретації відгуку середовища (рис.5.11). Колектив з шістьох агентів (N=6) отримує виграш у розмірі 29 одиниць за колективну дію (a_{11}, a_{21}, a_{34}, a_{42}, a_{53}, a_{63}), де a_{ij} – j-та дія і-го агента. В даному прикладі цей виграш розподілився між агентами в наступний спосіб: перший агент отримав 3 одиниці виграшу, другий – 7, третій – 2, четвертий – 5, п'ятий – 9 і шостий – 3. В загальному випадку механізм розподілу спільного виграшу є інтегральною частиною алгоритму колективного навчання.

Приклад програмної реалізації обчислювального експерименту для дослідження методів навчання з підкріпленням в багатоагентних системах наведено в додатку Д.8.

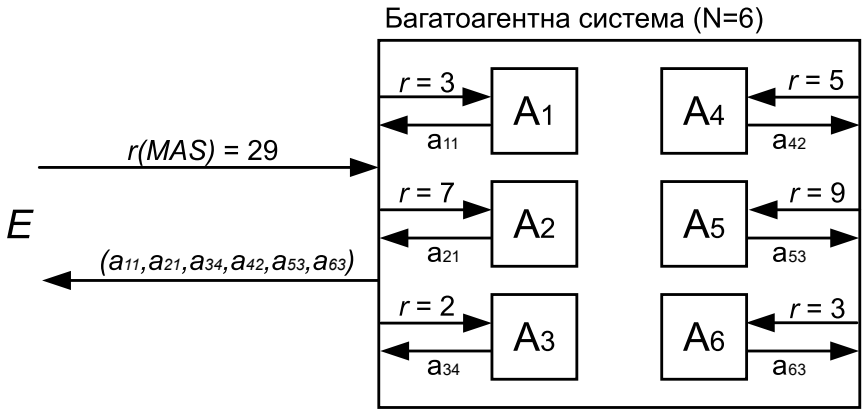


Рис. 5.11. Приклад між-агентної проблеми інтерпретації відгуку середовища.

5.2.2. Класифікація задач навчання з підкріпленням в багатоагентних системах

Задачі навчання з підкріпленням в багатоагентних системах можна класифікувати за багатьма різними ознаками та критеріями [29]. Серед основних класифікаційних ознак можна виділити: мету навчання, спосіб інформаційної взаємодії між агентами, тип колективних дій з точки зору їх наслідків, спосіб організації процесу навчання у часі та ін. Розглянемо більш детально кожну з наведених класифікаційних ознак.

З точки зору мети навчання, задачі багатоагентного навчання з підкріпленням можна поділити на дві основні категорії (табл.5.3):

1) задачі на пошук найкращої схеми взаємодії агентів, тобто найкращого способу організації колективних дій (наприклад, розподіл функціональних ролей між агентами БАС (спеціалізація), розподіл спільного ресурсу між агентами, регулювання рівня інформаційної зв'язності між агентами, тощо);

2) задачі на пошук найбільш ефективної колективної поведінки в середовищі, в якому розміщені агенти, тобто найкращого способу взаємодії колективу з середовищем згідно обраної моделі оптимальної колективної поведінки (наприклад, перевезення вантажів агентами автономної транспортної системи за якомога менший час, виявлення порушників заданого периметру агентами автономної системи патрулювання, розподіл обчислювального навантаження програмними агентами, тощо).

Таблиця 5.3. Класифікація задач колективного навчання з підкріпленням за метою навчання.

	Фіксована схема взаємодії агентів	**Змінна схема взаємодії агентів**
Фіксований спосіб взаємодії з середовищем	навчання відсутнє	навчання міжагентній взаємодії
Змінний спосіб взаємодії з середовищем	навчання взаємодії з середовищем	одночасне навчання міжагентній взаємодії та взаємодії з середовищем

За способом інформаційної взаємодії між агентами задачі багатоагентного навчання з підкріпленням можна поділити на:

1) задачі без інформаційної взаємодії між агентами (так зване, ізольоване навчання з підкріпленням),

2) задачі з інформаційною взаємодією агентів (так зване, інтерактивне навчання з підкріпленням), в тому числі 2.1) з обмеженою (локальною) інформаційною взаємодією агентів, або 2.2) з «необмеженою» інформаційною взаємодією агентів.

За типом колективних дій з точки зору їх наслідків задачі багатоагентного навчання з підкріпленням можуть відрізнятись за наступними ознаками:

1) сильнозв'язані чи слабозв'язані дії агентів,

2) рівноцінні чи нерівноцінні по впливу дії агентів,

3) кількісний чи синергетичний ефект спільних дій.

З точки зору способу організації багатоагентного навчання з підкріпленням у часі розрізняють:

1) одночасне (паралельне) навчання, коли всі агенти приймають участь у багатовалентному навчанні,

2) навчання зі зсувом у часі (staggered or lock-step learning), коли кожному агенту відводиться певний час, на протязі якого він один серед усіх інших агентів навчається, після чого він реалізує ту поведінку, якій навчився, а до навчання переходить інший агент.

5.2.3. Методи навчання з підкріпленням в багатоагентній системі

Розглянемо дві групи методів навчання з підкріпленням в багатоагентній системі, зокрема одночасне ізольоване навчання з підкріпленням та інтерактивне навчання з підкріпленням. У складі цих методів в якості компонент використовуються розглянуті вище методи навчання з підкріпленням, модифіковані в той чи інший спосіб для використання у багатоагентній системі. Зауважимо, що складність багатоагентного навчання з підкріпленням так само залежить від складності моделі об'єкту управління, з яким взаємодіє багатоагентна система.

Одночасне ізольоване навчання з підкріпленням вирізняється тим, що кожний агент виконує свій індивідуальний алгоритм навчання (при цьому, як правило, інші агенти в рамках виконання алгоритму навчання не моделюються). Тобто обмін інформацією між агентами відсутній. Мета багатоагентного навчання в даному випадку полягає у знаходженні найкращої колективної поведінки з огляду на вирішення поставленої перед багатоагентною системою завдання та відповідну цільову функцію.

Основна проблема організації одночасного ізольованого навчання полягає у формуванні таких індивідуальних функцій виграшу та оцінки, які б гарантували збіжність колективної поведінки до найкращої. Одночасне ізольоване навчання з підкріпленням набуває особливої ваги в ситуаціях, коли зв'язок між агентами або взагалі неможливий, або тимчасово недоступний, наприклад, внаслідок відмов каналів зв'язку, збою в роботі всієї системи зв'язку, високого рівня завад, тощо.

Серед методів одночасного ізольованого навчання з підкріпленням розрізняють два основних класи:

1) методи, в яких використовується глобальний відгук (глобальне підкріплення), коли всі агенти отримують один і той самий відгук середовища на одному кроці взаємодії; ці методи базуються на порівняно простих моделях колективної поведінки, наприклад, на моделі механічного врівноваження, коли колектив мобільних агентів намагається врівноважити деяку площину з однією точкою опори, на якій вони розташовані (в даному випадку глобальний відгук – це поточний кут нахилу площини),

2) методи, в яких використовується локальний відгук (локальне підкріплення), коли агенти отримують різні «індивідуальні» відгуки середовища на кожному кроці взаємодії; ці методи базуються на порівняно складних моделях колективної поведінки, наприклад, на різних варіантах лабіринтних задач, які вирішуються багатоагентною системою.

Інтерактивне навчання з підкріпленням вирізняється тим, що на кожному кроці взаємодії з середовищем агенти повідомляють один одному, які дії вони збираються реалізувати або повідомляють свій вибір на попередніх кроках взаємодії та його результати. При цьому кожний агент «моделює» поведінку інших агентів, що дає змогу оцінити, які дії агентів в межах всього колективу є найбільш успішними. Наприклад, якщо сам агент і двоє його «сусідів» реалізували першу дію і лише один «сусід» реалізував другу дію, і в наслідок цих колективних зусиль було отримано виграш, то ваговий коефіцієнт першої дії треба збільшити, оскільки саме вона швидше за все призвела до цього виграшу.

В даному випадку за рахунок обміну інформацією в рамках відповідної моделі інформаційної зв'язності з'являється можливість в явний спосіб обирати (організовувати) узгоджені колективні дії. При цьому використовуються різні схеми (методи, алгоритми) узгодження та обміну між агентами здобутим досвідом. Наприклад, кожний з агентів формує вагові коефіцієнти, що відповідають діям, які він може реалізувати, і повідомляє значення цих коефіцієнтів іншим агентам. Після кожного кроку взаємодії з середовищем значення всіх вагових коефіцієнтів модифікуються згідно заданого правила навчання з підкріпленням з

урахуванням значень вагових коефіцієнтів, що надійшли від інших агентів. Внаслідок цього з часом колектив знаходить найкращу поведінку у вигляді дійсних вагових коефіцієнтів дій.

Ще одним прикладом може бути багатоагентне навчання з підкріпленням у вигляді модифікованого методу зваженої оцінки дій, зміненого для випадку колективу агентів, коли оціночна вага дії перераховується не лише на основі власного досвіду агента, але і на основі досвіду, здобутого іншими агентами. В даному випадку інтуїтивно можна зробити висновок про пришвидшення процесу навчання (тобто збільшення швидкості, з якою значення оціночної ваги дій збігаються до дійсних значень) за рахунок збільшення кількості вибірок (випробувань різних дій агентами). Також в цьому прикладі, чим більша пропускна здатність системи міжагентного зв'язку, тим швидше відбувається процес багатоагентного навчання з підкріпленням.

Приклади програмної реалізації одночасного ізольованого навчання з підкріпленням та інтерактивного навчання з підкріпленням в багатоагентній системі наведено в додатку Д.9.

5.2.4. Навчання розподілу обов'язків в багатоагентній системі

Мета навчання розподілу обов'язків полягає у пошуку найбільш вдалого варіанту (схеми) розподілу функціональних ролей (режимів роботи), які виконують різні агенти у складі багатоагентної системи, з точки зору ефективності вирішення поставленої перед ними задачі. Основна ідея такого пошуку полягає в тому, що взаємодоповнення різних за змістом функціональних ролей (обов'язків, режимів роботи, тощо) якісно покращує колективну поведінку за рахунок синергетичного ефекту в рамках концепції функціональної емерджентності. В базовому варіанті набір різних функціональних ролей (обов'язків) агентів, що використовується для навчання розподілу обов'язків, формує розробник перед початком навчання. В більш складних варіантах задачі, формування цього набору покладається на багатоагентну систему, тобто колектив агентів самостійно формує набір функціональних ролей і навчається розподіляти їх між агентами.

Розглянемо приклад методу навчання розподілу обов'язків на основі навчання з підкріпленням [28]. В рамках цього методу перехід середовища з одного стану в інший (або ланцюжок переходів між станами заданої довжини) розглядається як окрема ситуація. При цьому кожний стан середовища характеризується набором можливих ситуацій. В кожній ситуації і-ий агент багатоагентної системи може вибирати поміж k_i різних функціональних ролей {r}. Далі будемо використовувати наступні позначення: s – деяка ситуація, в яку потрапив агент; r – деяка функціональна роль, яку обрав для себе агент в ситуації s.

В даному методі навчання розподілу обов'язків для всіх можливих пар (s,r) визначаються чотири значення:

1) корисність (utility) U_t – інтегральна оцінка виграшу, який отримає агент, якщо в поточній ситуації s він обере роль r і середовище при цьому перейде в бажаний (виграшний) стан (тобто яку винагороду отримає агент),

2) ймовірність (probability) P_t – ймовірність того, що внаслідок обрання агентом ролі r в ситуації s середовище перейде в бажаний стан (тобто з якою ймовірністю буде отримана винагорода),

3) ціна (cost) C_t – оцінка того, яку кількість деякого власного ресурсу (наприклад, енергетичного чи обчислювального) витратить агент, якщо в ситуації s він буде виконувати роль r (тобто якими будуть втрати на виконання функціонально ролі r в ситуації s),

4) потенціал (potential) L_t – оцінка користі, яку принесе колективу виконання агентом ролі r в ситуації s з точки зору дослідження загальних можливостей та обмежень для колективних дій (тобто наскільки важливим буде цей вибір з точки зору дослідження середовища та збирання досвіду взаємодії з ним).

Ці чотири значення об'єднуються за допомогою цільової функції

$$f(s,r) = f(U_t,P_t,C_t,L_t).$$

На основі значення цільової функції визначається ймовірність вибору ролі r в ситуації s як відношення:

$$\pi(s,r) = f(s,r)/\Sigma f(s,r'),$$

де r' - усі інші ролі, які може обрати агент в ситуації s. Вибір функціональних ролей за допомогою ймовірностей $\{\pi(s,r)\}$ здійснюється на протязі деякого періоду навчання. Після його завершення агент обирає роль з найбільшим серед усіх інших значенням цільової функції:

$$r = \arg \max_s\{f(U_t, P_t, C_t, L_t)\}.$$

Попередньо невідомі дійсні значення (U_t, P_t, L_t) для кожної пари (s,r) агент отримує за допомогою навчання з підкріпленням:

$$U_{t+1} = (1-\alpha)U_t + \alpha \cdot U_F,$$

де

U_F – оцінка виграшу, який отримав агент у фінальному стані F (тобто після завершення ситуації),

α – крок навчання, $0 \le \alpha \le 1$,

$$P_{t+1} = (1-\alpha)P_t + \alpha \cdot O(F),$$

де

O(F) = 0, якщо фінальний стан F – програшний (небажаний),

O(F) = 1, якщо фінальний стан F – виграшний (бажаний),

$$L_{t+1} = (1-\alpha)L_t + \alpha \cdot H(F),$$

де

H(F) = 1, якщо в процесі досягнення фінального стану F відбувся хоча б один конфлікт між агентами (за участю даного агента) і відбувся обмін інформацією для вирішення цього конфлікту (тобто відбулась «суперечка» між агентами за вибір функціональної ролі),

H(F) = 0, якщо в процесі досягнення фінального стану F не відбулося жодного конфлікту за участю даного агента,

Вибір оцінок витрат ресурсів $\{C_t\}$ і вигляду цільової функції f залежить від області застосування методу навчання розподілу обов'язків і покладається на розробника.

Контрольні питання

1. Що таке багатоагентна система (multi-agent system)?
2. Якому структурному принципу побудови розподілених систем приділяють основну увагу в області дослідження та розробки багатоагентних систем?
3. Для якого структурного принципу побудови розподілених систем є характерною потенційно висока оперативність прийняття рішень?
4. В чому полягає концепція ройового інтелекту (swarm intelligence)?
5. Які компоненти та групи компонент включає концептуальна модель багатоагентної системи?
6. Які основні алгоритми входить до набору базових службових алгоритмів колективної поведінки в багатоагентних системах?
7. Як вирішується проблема координування колективної поведінки в багатоагентних системах?
8. В чому полягає ідея ігрових механізмів координації колективної поведінки в багатоагентних системах?
9. Які основні проблеми вирішується в задачах навчання з підкріпленням в багатоагентних системах?
10. В чому полягає проблема інтерпретації відгуку середовища (credit-assignment problem) в багатоагентному навчанні з підкріпленням?
11. Які основні принципи покладено в основу багатоагентного навчання з підкріпленням?
12. За якими ознаками класифікуються задачі навчання з підкріпленням в багатоагентних системах?
13. Які основні моделі навчання з підкріпленням використовуються в багатоагентних системах?
14. Як організується робота методів одночасного ізольованого навчання з підкріпленням в багатоагентних системах?
15. В яких методах навчання з підкріпленням в багатоагентних системах агенти обмінюються здобутим досвідом?
16. Чим відрізняються одночасне ізольоване навчання з підкріпленням та інтерактивне навчання з підкріпленням в багатоагентних системах?
17. Яка основна ідея покладена в основу навчання розподілу обов'язків в багатоагентній системі?
18. В який спосіб оцінюється користь, яку приносить багатоагентній системі виконання агентом деякої функціональної ролі, в методі навчання розподілу обов’язків на основі навчання з підкріпленням?

Глосарій

Штучний інтелект (Artificial Intelligence, AI) – «розум» комп’ютерної системи, який в деяких аспектах (або в цілому) по своїх можливостях еквівалентний розуму людини або переважає його.

Інтелектуальна система – в широкому розумінні, будь-яка комп’ютерна система з елементами штучного інтелекту. В прикладному розумінні інтелектуальна система – це автономна комп’ютерна система, яка взаємодіє з деяким середовищем (об’єктом управління) і самостійно вирішує достатньо складну задачу, поставлену перед нею власником системи. В області штучного інтелекту інтелектуальну систему також називають інтелектуальним агентом (intelligent agent) або просто агентом (agent).

Агент (agent) – поняття в області штучного інтелекту, яке означає деяку автономну сутність, яка діє самостійно, проактивно та цілеспрямовано.

Інтелектуальний агент (intelligent agent) – комп’ютерна система яка взаємодіє з деяким середовищем та 1) здатна діяти в цьому середовищі самостійно без втручання людини (автономність), 2) здатна досягати поставлену перед нею власником мету (цілеспрямованість), 3) здатна адаптуватись до змін у середовищі та цілеспрямовано вивчати відповідні закономірності, 4) здатна взаємодіяти (діяти спільно та узгоджено) з іншими інтелектуальними агентами та людиною.

Алан Тюрінг (Alan Turing) (1912–1954) – англійській математик, спеціаліст в області криптографії, логіки, інформатики, математичної біології, один з засновників комп’ютерних наук та досліджень в області штучного інтелекту. Наукові ідеї Тюрінга істотно вплинули на розвиток комп'ютерної техніки та інформаційних технологій. В своїй роботі «Обчислювальні машини і розум» (Computing Machinery and Intelligence, 1950) Тюрінг вперше висловив ідею штучного інтелекту, як властивості деякої інтелектуальної системи, побудованої на основі комп'ютерних засобів. В цій же роботі Тюрінг вперше висловив ідею машинного навчання в його сучасному розумінні та запропонував методику визначення чи володіє комп’ютерна система інтелектом (тест Тюрінга). В роботі «Хімічні основи морфогенезу» (The Chemical Basis of Morphogenesis, 1952) Тюрінг вперше запропонував математичну модель процесу самоорганізації в біологічній системі.

Тест Тюрінга – емпіричний тест для визначення здатності комп'ютерної системи демонструвати інтелектуальну поведінку, яку в умовах тесту неможливо відрізнити від поведінки людини. Ідею тесту запропонував Алан Тюрінг у своїй роботі «Обчислювальні машини і розум» (Computing Machinery and Intelligence, 1950).

Китайська кімната – уявний експеримент в області філософії знання та філософії штучного інтелекту, мета якого полягає в спростуванні твердження про те, що обчислювальна машина, наділена «штучним інтелектом», здатна володіти свідомістю в тому ж розумінні, в якому нею володіє людина. Іншими словами, метою є спростування гіпотези так званого «сильного» штучного інтелекту і критика тесту Тюрінга. Цей уявний експеримент був запропонований філософом Джоном Серлем (John Searle) в його роботі «Розум, мізки і програми» (Minds, Brains, and Programs, 1980).

Об’єкт управління – термін кібернетики і теорії автоматичного управління, що означає об’єкт, пристрій або динамічний процес, управління поведінкою якого є метою створення системи автоматичного управління.

Середовище (environment) – поняття в області штучного інтелекту, яке означає проблемну ситуацію, в рамках якої перед агентом стоїть мета вирішити деяку поставлену перед ним задачу. Поняттю «середовище» з деякими застереженнями еквівалентне поняття «об'єкт управління» з теорії автоматичного управління.

Модель цілеспрямованої поведінки – математична модель, в рамках якої 1) визначається спосіб взаємодії агента з середовищем (об'єктом управління), 2) структура та характеристики цього середовища, та 3) способи оцінки загальної успішності дій агента в цьому середовищі (тобто способи задання цільової функції агента).

Стаціонарне випадкове середовище (СВС) – проста модель середовища з нестачею інформації, в якій у відповідь на кожну дію a, обрану агентом з деякої заданої множини дій A, середовище повертає сигнал виграшу або програшу r={0;1}. При цьому для кожної дії a задається незмінне у часі значення ймовірності {p(a)}, з якою у відповідь на цю дію буде отримано сигнал виграшу (функція нарахування виграшу).

Випадкове середовище з перемиканням станів (ВСПС) – модель середовища з нестачею інформації, яка складається з заданого набору стаціонарних випадкових середовищ, на множині яких визначена функція переходів (ланцюг Маркова) між відповідними станами ВСПС (ця функція не залежить від дій агента). Особливістю ВСПС є те, що до невизначеності стосовно функції нарахування виграшу додається невизначеність стосовно функції переходів між СВС (станами ВСПС) з різними характеристиками.

Автомат, що навчається (Learning Automaton) – цифровий автомат, який взаємодіє з деяким середовищем (об'єктом управління) по схемі дія-відгук, та здатний діяти цілеспрямовано шляхом навчання найкращій (згідно заданого критерію) дії або послідовності дій в цьому середовищі.

Глибина пам'яті автомату, що навчається – основний параметр його конструкції, який визначає кількість станів в окремій «гілці» автомату (кожна гілка відповідає дії, яку може обрати автомат). Для випадку стаціонарного випадкового середовища: чим більше глибина пам'яті n, тим ближче поведінка автомату, що навчається, до оптимальної.

Стохастичний автомат зі змінною структурою – автомат, що навчається, призначений для роботи у випадковому середовищі з перемиканням станів. Принцип роботи стохастичного автомату базується на покроковій нелінійній зміні ймовірностей переходу у матрицях перехідних ймовірностей $П^+$ (перехід між станами автомату під впливом сигналу виграшу) та $П^-$ (перехід між станами під впливом сигналу програшу).

Цетлін Михайло Львович (1924—1966) – математик, спеціаліст в області кібернетики, біокібернетики, систем управління, теорії ігор, математичних методів біології, один з засновників наукових досліджень в області цифрових автоматів, що навчаються (Learning Automata), засновник наукового напрямку «колективна поведінка автоматів». Цетлін М.Л. запропонував ідею ієрархічного принципу побудови складних систем.

Машинне навчання (machine learning), навчання обчислювальних машин – галузь комп'ютерних наук та штучного інтелекту, яка досліджує шляхи «надання комп'ютеру можливості навчатися на противагу безпосередньому програмуванню його дій» (Артур Семюель). В рамках навчання обчислювальних машин розглядаються алгоритми здатні до навчання та прогнозування з використанням попередньо накопичених даних. В багатьох складних задачах (особливо в задачах з нестачею інформації) такі алгоритми значно переважають звичайні «статичні» алгоритми, в яких виконується наперед задана незмінна послідовність дій.

Навчання з вчителем (supervised learning) - спосіб машинного навчання, в ході якого систему навчають відображати вхідні дані у вихідні на основі прикладів пар <вхід-вихід> (<стимул-реакція>). При цьому на етапі навчання система знаходиться під керуванням «вчителя» (людини чи іншої системи), який в той чи інший спосіб повідомляє системі про правильність чи неправильність відображення даних на кожному кроці навчання.

Навчання без вчителя (unsupervised learning) – спосіб машинного навчання, в ході якого система спонтанно навчається виконувати поставлене завдання, без втручання з боку «вчителя» (людини чи іншої системи). Як правило, метою навчання без вчителя є виявлення внутрішніх взаємозв'язків, залежностей, закономірностей, що існують між об'єктами, що входять у навчальну вибірку, з подальшим використанням цієї інформації для вирішення поставленої задачі.

Навчання з підкріпленням (Reinforcement Learning) – галузь машинного навчання, яка розглядає організацію процесу одночасного навчання інтелектуального агента та виконання поставленої перед ним задачі в ході взаємодії агента з середовищем (об’єктом управління). Основна проблематика навчання з підкріпленням стосується пошуку способів подолання нестачі інформації (невизначеності) про характеристики середовища шляхом різних варіантів методу спроб та помилок.

Multi-armed bandit (MAB) – модель випадкового середовища, в якій для кожної дії агента з множини доступних йому дій визначена функція виграшу у вигляді відповідного розподілу ймовірностей виграшів, який не змінюється у часі. Перед початком взаємодії агента з MAB йому ці функції не відомі. Для забезпечення цілеспрямованої поведінки агента в цих умовах застосовуються відповідні методи навчання з підкріпленням.

Оціночна вага дії (estimated action value) – величина, що відображає результативність вибору дії, і розраховується як відношення суми підкріплень, отриманих внаслідок виконання даної дії, до кількості кроків взаємодії агента з середовищем, в яких ця дія була виконана.

Оціночна вага стану (estimated state value) – величина, що відображає результативність поведінки агента внаслідок перебування середовища (об’єкту управління) в даному стані.

Методи зваженої оцінки дій (action-value methods) – методи навчання з підкріпленням, в яких для вибору наступної дії використовується оціночна вага дії (estimated action value), яка розраховується як відношення суми підкріплень, отриманих внаслідок виконання даної дії, до кількості кроків взаємодії агента з середовищем, в яких ця дія була виконана.

Жадібний метод навчання з підкріпленням (greedy RL) – метод зваженої оцінки дій, в якому для виконання в наступному кроці взаємодії агента з середовищем обирається дія з максимальним значенням оціночної ваги.

ε-жадібний метод навчання з підкріпленням (ε-greedy RL) – метод зваженої оцінки дій, в якому для виконання в наступному кроці взаємодії агента з середовищем з ймовірністю (1-ε) обирається дія з максимальним значенням оціночної ваги, і з ймовірністю ε дія обирається рівноймовірно (тобто кожна з дій може бути обрана з ймовірністю 1/n, де n - кількість дій).

Марківський процес прийняття рішень (Markov Decision Process, MDP) – модель середовища з нестачею інформації, яка складається з чотирьох елементів: <S,A,R,T>, де S={s} – множина станів середовища; A={a} – множина доступних агенту дій; $R:S\times A\rightarrow\Re$ – функція виграшу (підкріплення), яка відображає пару (s,a) у виграш; $T:S\times A\rightarrow\Pi(S)$ - функція переходів

між станами середовища (функція ймовірностей переходів, transition probability function). Особливістю MDP є те, що функція переходів між станами залежить від дій агента, що створює додаткові труднощі при розробці алгоритмів цілеспрямованої поведінки в цьому середовищі.

POMDP (Partially observable Markov decision process) – модель середовища з нестачею інформації на основі марківського процесу прийняття рішень (MDP), в якій додатково вноситься невизначеність щодо поточного стану, в якому перебуває середовище (об'єкт управління).

Архітектура інтелектуального агента – набір та спосіб поєднання різних компонент, які у сукупності реалізують автономну, проактивну та цілеспрямовану поведінку штучного агента, який здатний навчатись та адаптуватись до змін у середовищі (об'єкті управління та/або оточенні агента).

Простий рефлексивний агент (simple reflex agent) – агент, який обирає дії на основі поточної інформації про стан середовища (current percept), не зважаючи на інформацію про попередні стани, та використовує правила «умова-дія» (condition-action rules).

Рефлексивний агент з моделлю середовища (model-based reflex agent) – рефлексивний агент, що використовує модель середовища (об'єкта управління), зокрема 1) зберігає у пам'яті деякий «внутрішній стан» (internal state), який залежить від інформації про попередні стани середовища (percept history); 2) завдяки цьому здатний відслідковувати частину оточення, про яку він не має інформації в даний момент часу.

Агент орієнтований на досягнення мети (goal-based agent) – агент, який намагається досягнути поставлену перед ним мету, зокрема 1) використовує інформацію про «бажані»/вигідні ситуації або стани середовища - інформацію про поставлену мету (goal information); 2) зіставляє інформацію про мету з моделлю середовища для вибору дій, які наближають до мети чи відразу призводять до її досягнення.

Агент орієнтований на максимізацію цільової функції (utility-based agent) – агент, який намагається максимізувати значення заданої цільової функції (функції корисності, utility function), тобто обирає дії в такий спосіб, щоб максимізувати очікувану корисність послідовності обраних дій (очікуване значення цільової функції).

Логічно-символьна архітектура інтелектуального агента – архітектура на основі деякої моделі логічного мислення, в якій для прийняття рішення використовуються 1) база знань, що включає факти і логічні твердження, та 2) відповідні правила логічного виводу.

Поведінкова архітектура інтелектуального агента – архітектура на основі ідеї про спонтанне виникнення (emergence) складної цілеспрямованої поведінки в результаті взаємодії набору відносно простих правил вигляду «стимул-реакція» (або «якщо-то»).

Комбінована архітектура інтелектуального агента (Hybrid architecture) – архітектура, в рамках якої поєднуються два архітектурних підходи до побудови інтелектуальних агентів: підхід на основі моделей логічного мислення (логічно-символьні архітектури) та підхід на основі простих правил поведінки (поведінкові архітектури) з метою збереження переваг обох підходів та компенсації їх недоліків.

Багатоагентна система (multi-agent system) – слабозв'язаний набір автономних інтелектуальних агентів, об'єднаних частково чи повністю спільною метою.

Колективна поведінка – поведінка деякого числа автономних сутностей, об'єднаних спільними, частково спільними або протилежними цілями, в результаті чого їх дії повністю або частково взаємозумовлені.

Централізоване управління – структурний принцип побудови розподілених систем, згідно якого всі рішення приймає єдиний центр управління. Центр збирає інформацію від усіх інших компонент системи та передає їм вказівки згідно прийнятого ним рішення. Централізоване управління в розподілених системах характеризується 1) потенційно високою точністю за рахунок наявності у центрі усієї доступної інформації про розподілений об’єкт управління, 2) низькою оперативністю внаслідок витрат часу на передавання даних між компонентами системи і центром, а також необхідності обробки великих об’ємів даних, що надходять від компонент системи у центр. З точки зору живучості всієї розподіленої системи центр є слабким місцем. В разі його відмови зупиняється робота всієї системи.

Децентралізоване управління – структурний принцип побудови розподілених систем, згідно якого всі рішення приймаються локально розподіленими компонентами системи за умов відсутності єдиного центру управління. Компоненти системи використовують для прийняття рішення локальну обмежену інформацію про об’єкт управління. Децентралізоване управління в розподілених системах характеризується 1) потенційно низькою точністю внаслідок відсутності у окремої компоненти системи усієї доступної інформації про розподілений об’єкт управління в момент прийняття рішення, 2) високою оперативністю за рахунок прийняття рішень компонентами, які безпосередньо взаємодіють з об’єктом управління, та обробкою у окремій компоненті системи порівняно невеликого об’єму даних. Відсутність єдиного центру управління різко підвищує живучість розподіленої системи. Така система продовжує виконувати свою функцію (можливо з втратами у продуктивності) доти, доки в робочому стані лишається хоча б одна компонента системи.

Swarm intelligence (ройовий інтелект) – концепція в області штучного інтелекту, згідно якої колективна поведінка простих елементів деякої децентралізованої системи за певних умов призводить до її самоорганізації, тобто виникнення функціональних можливостей вищого рівня, які не зводяться до простої суми функціональних можливостей окремих елементів. Системи ройового інтелекту, як правило, складаються з сукупності простих агентів, що локально взаємодіють між собою та своїм оточенням. Основним джерелом ідей для побудови таких систем, як правило, слугують різні біологічні системи і, в першу чергу, колонії суспільних комах (мурахи, бджоли, оси, терміти, деякі види павуків). Агенти в таких системах використовують невеликий набір простих правил, і за відсутності центру управління, в умовах локальної взаємодії з елементами випадковості, забезпечують виникнення інтелектуальної поведінки на рівні всієї системи. Приклади ройового інтелекту в природних системах: колонії мурах, зграї птахів, популяції бактерій та ін. Одним з прикладів застосування цієї концепції є ройова робототехніка (swarm robotics). З практичної точки зору, концепція ройового інтелекту знайшла найбільше використання у вирішенні задач пошуку та оптимізації, зокрема задач комбінаторної оптимізації.

Artificial life (A-Life) (штучне життя) – область дослідження систем, що в тих чи інших аспектах імітують живі біологічні системи та процеси які в них розгортаються, в тому числі процеси еволюційного розвитку. Дослідження в області штучного життя, як правило, спираються на комп’ютерне моделювання. В окремих дослідницьких проектах ведеться розробка рішень в області робототехніки та біохімії. Одним з напрямків досліджень в області штучного життя є дослідження колективної поведінки штучних істот, наприклад, на основі популяційних моделей колективної поведінки.

Координаційна гра (coordination game) – поняття з теорії ігор, що означає спеціальну гру, в якій перед гравцями стоїть дилема щодо вибору між двома чи більше рішеннями гри, кожне з

яких в тому чи іншому аспекті задовольняє гравців і в той же час є результатом лише їх спільного (скоординованого) вибору. Відтак в координаційній грі до проблеми вибору стратегії додається проблема вибору способу реагування на поведінку інших гравців. Координаційні ігри широко застосовуються для моделювання поведінки агентів у складі багатоагентних систем, в тому числі з метою розробки механізмів координації колективної поведінки. Одною з найбільш відомих та досліджених координаційних ігор є, так звана, дилема в'язня (prisoner's dilemma).

Гра з нульовою сумою (zero-sum game) – термін теорії ігор, який означає гру, в якій інтереси гравців прямо протилежні. Відтак сума виграшів гравців в такій грі дорівнює нулю.

Гра з ненульовою сумою (general sum game) – термін теорії ігор, який означає гру, в якій інтереси гравців частково протилежні і частково співпадають. Відтак сума виграшів гравців в такій грі більше нуля.

Рівновага Неша (Nash equilibrium) – такий набір стратегій гравців у грі, в якому жоден учасник не може збільшити свій виграш, змінивши свою стратегію, якщо інші учасники своїх стратегій не змінюють. Це рішення гри названо на честь Джона Неша (John Nash), який довів існування рівноваги в змішаних стратегіях в будь-якій скінченій грі.

Дилема в'язня (prisoner's dilemma) – координаційна гра двох гравців з ненульовою сумою, в якій кожний з гравців обирає між двома стратегіями: співпрацювати чи протидіяти іншому гравцю. Рішенням гри з точки зору рівноваги Неша (тобто стійкої рівноваги) є (протидія, протидія). Тобто з позицій раціональної поведінки обидва гравці мають обрати це рішення. В той же час рішення гри (співпраця, співпраця) за умовами гри дає обом гравцям більший виграш ніж рівновага Неша. Відтак поводячись окремо раціонально, разом учасники приходять до нераціонального рішення. Аналіз дилеми в'язня ускладнюється, коли гра стає ітераційною, тобто коли розіграш гри повторюється у часі, і гравці пам’ятають результати попередніх партій.

Самоорганізація – процес виникнення деякої впорядкованої (у просторі і/або часі) структури чи схеми міжелементних взаємовідносин внаслідок локальної взаємодії елементів від початку невпорядкованої системи.

Структурна самоорганізація – автономний процес цілеспрямованої зміни (впорядкування) системою своєї структури з метою пошуку та забезпечення якісно нових функціональних можливостей, які не зводяться до простої суми функціональних можливостей елементів системи.

Автономна децентралізована система – комп’ютерна система, вузли якої розподілені у просторі та самостійно виконують деяку спільну задачу в умовах відсутності єдиного центру управління.

Гайнц фон Ферстер (Heinz von Förster) (1911-2002) – австрійський та американський фізик, математик, один з засновників кібернетики. Досліджував загальні кібернетичні основи біологічних і електронних систем. В роботі «On Self-Organizing Systems and Their Environments» (1959) запропонував модель процесу самоорганізації на основі інформаційної ентропії.

Вільям Ешбі (William Ross Ashby) (1903-1972) – англійській психіатр, спеціаліст з кібернетики, один з засновників наукового дослідження складних систем та принципів самоорганізації. В роботі «Principles of the Self-Organizing Dynamic System» (1947) вперше

запропонував термін «самоорганізація» (self-organization). В роботі «Principles of Self-Organizing Systems» (1962) представив свою концепцію самоорганізації в складних системах.

Ентропія - (грец. ἐντροπία – поворот, перетворення) – функція стану термодинамічної системи, що характеризує напрямок протікання самовільних процесів в цій системі і є мірою їх незворотності. У широкому сенсі ентропія означає міру складності, хаотичності або невизначеності системи: чим менше елементи системи підпорядковані якомусь порядку, тим вище ентропія.

Інформаційна ентропія (шенонівська ентропія) – середній рівень «несподіваності» («невизначеності»), з яким приймає свої значення деяка випадкова величина. Інформаційна ентропія є мірою невизначеності деякої системи, зокрема, непередбачуваності появи будь-якого символу первинного алфавіту джерела інформації. У останньому випадку при відсутності інформаційних втрат ентропія чисельно рівна об’єму інформації на один символ повідомлення, що передається.

Клод Шеннон (Claude Elwood Shannon) (1916-2001) – американський інженер та математик. Заклав основи теорії інформації: використовуючи теорію ймовірностей і математичну статистику, вирішив задачу знаходження оптимального способу передачі інформації (теорема Шеннона, 1948-49). Заклав основи теорії автоматів (1938).

Література

1. Stuart Russell, Peter Norvig, Artificial Intelligence: A Modern Approach, 4th edition, Pearson, 2020. - 1136 p.
2. David L. Poole, Alan K. Mackworth, Artificial Intelligence: Foundations of Computational Agents, Cambridge University Press, 2010. - 682 p.
3. Kevin Warwick, Artificial Intelligence: The Basics, Routledge, 2011. - 192 p.
4. Richard E. Neapolitan, Xia Jiang, Artificial Intelligence: with an Introduction to Machine Learning, Chapman and Hall, 2018. – 480 p.
5. Alan Turing, Computing Machinery and Intelligence, Mind, vol. LIX, no. 236, October 1950. – pp. 433–460
6. Adrian A. Hopgood, Intelligent Systems for Engineers and Scientists: A Practical Guide to Artificial Intelligence, 4th ed., CRC Press, 2021. – 515 p.
7. Geoff Hulten, Building Intelligent Systems: A Guide to Machine Learning Engineering, Apress, 2018. – 339 p.
8. M. L. Tsetlin, Automaton Theory and Modeling of Biological Systems, Academic Press, New York, 1973. – 288 p.
9. Narendra, K. and Thathachar, M. A. L., Learning Automata: An Introduction, 2nd ed., Dover Publications, 2013. – 496 p.
10. K. Najim, A.S. Poznyak, Learning Automata: Theory and Applications, Elsevier, 2014. – 236 p.
11. Mohri M., Rostamizadeh A., Talwalkar A., Foundations of Machine Learning, 2nd edition, The MIT Press, 2018. - 504 p.
12. Stephen Marsland, Machine Learning: An Algorithmic Perspective, 2nd Edition, Chapman & Hall, 2014. - 457 p.
13. Machine Learning for Cyber Physical Systems, Oliver Niggemann, Jürgen Beyerer (eds.), Springer Vieweg, 2016. - 124 p.
14. Goodfellow I., Bengio Y., Courville A., Deep Learning, The MIT Press, 2016. - 800 p.
15. Richard S. Sutton, Andrew G. Barto, Reinforcement Learning: An Introduction, 2nd edition, The MIT Press, 2018. - 552 p.
16. L.P. Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. Journal of AI Research, N4, 1996. - pp.237-285
17. L.P. Kaelbling, Learning in Embedded Systems, MIT Press, 1993. – 176 p.
18. Phil Winder, Reinforcement Learning: Industrial Applications of Intelligent Agents, O'Reilly Media, 2021. - 408 p.
19. Warren B. Powell, Reinforcement Learning and Stochastic Optimization, Wiley, 2022. – 1136 p.
20. Maxim Lapan, Deep Reinforcement Learning Hands-On, 2nd edition, Packt Publishing, 2020. - 798 p.
21. Reinforcement Learning: Theory and Applications, ed. by C. Weber, M. Elshaw, N. M. Mayer, I-Tech Education and Publishing, Vienna, 2008. - 424 p.
22. Michel Tokic, Adaptive ε-greedy exploration in reinforcement learning based on value differences, in Proceedings of the 33rd German conference on Advances in artificial intelligence. 2010. – pp. 203–210
23. Peter Auer, Nicolò Cesa-Bianchi, Paul Fischer, Finite-time Analysis of the Multiarmed Bandit Problem, Machine Learning, May 2002, Volume 47, Issue 2–3. – pp. 235–256
24. Multiagent Systems, by Gerhard Weiss (Editor), 2nd edition, The MIT Press, 2013. – 920 p.

25. Michael Wooldridge, An Introduction to MultiAgent Systems, 2nd edition, Wiley, 2009. – 484 p.
26. Yoav Shoham, Kevin Leyton-Brown, Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations, Cambridge University Press, 2008. – 504 p.
27. M. Tan, Multi-agent reinforcement learning: independent vs. cooperative agents, in Proceedings of the Tenth International Conference on Machine Learning, 1993. – pp. 330-337
28. M.V. Nagendra Prasad, V.R. Lesser, and S.E. Lander, Learning organizational roles in a heterogeneous multi-agent system, in Proceedings of the Second International Conference on Multiagent Systems, 1996. – pp. 291-298
29. Howard M. Schwartz, Multi-Agent Machine Learning: A Reinforcement Approach, Wiley, 2014. – 256 p.

Додатки

Д.1. Приклад програмної реалізації стаціонарного випадкового середовища

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

#define ENVTYPE         0
#define NACTIONS        2
#define NSTATES         2

#define REWARD          1
#define PENALTY         0

int nA = NACTIONS;
int nS = NSTATES;

int env = ENVTYPE;
float sePa[NACTIONS];

int   ceState;
float cePa[NSTATES][NACTIONS];
float cePs[NSTATES][NSTATES];

int action=0;                   // current action = {0, ... ,(nA-1)}
int response;                   // current response of the environment = {0;1}

// ------------------------------------------------------------------------
// uniform discrete probability distribution


int uRand (int x)
{
      int _rnum = (int) ((float)x * (float)rand() / (float)RAND_MAX);
      return _rnum;
}

// ------------------------------------------------------------------------
// discrete probability distribution specified by probabilities from <_array>


int dRand (float* _array, int size)
{
      int _rnum = size-1;
      float _left = 0;
      float _right = _array[0];
      float ftmp = (float)rand() / (float)RAND_MAX;

      for (int i=0; i < size-1; i++)
      {
            if ((ftmp >= _left) && (ftmp < _right)) {_rnum = i; break;}
            _left = _right;
            _right += _array[i+1];
      }

      return _rnum;
}
```

```
// ------------------------------------------------------------------------
// initialization of stationary environment


void seInit (void)
{
      for (int i=0; i < nA; i++)
            sePa[i] = (float)rand() / (float)RAND_MAX;

      //sePa[0] = 0.8f;
      //sePa[1] = 0.2f;
}

// ------------------------------------------------------------------------
// response of stationary environment


int seResponse (void)
{
      int _r;
      float rnum = (float)rand() / (float)RAND_MAX;

      if (rnum < sePa[action])      _r = REWARD;
      else                                  _r = PENALTY;

      return _r;
}
```

Д.2. Приклад програмної реалізації випадкового середовища з перемиканням станів

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

#define ENVTYPE         0
#define NACTIONS        2
#define NSTATES         2

#define REWARD          1
#define PENALTY         0

int nA = NACTIONS;
int nS = NSTATES;

int env = ENVTYPE;
float sePa[NACTIONS];

int   ceState;
float cePa[NSTATES][NACTIONS];
float cePs[NSTATES][NSTATES];

int action=0;                   // current action = {0, ... ,(nA-1)}
int response;                   // current response of the environment = {0;1}

// ------------------------------------------------------------------------
// initialization of commutative environment


void ceInit (void)
{
      int i,j;
      float _sum1, _sum2;

      // probabilities of rewards
      for (i=0; i < nS; i++)
            for (j=0; j < nA; j++)
                  cePa[i][j] = (float)rand() / (float)RAND_MAX;

      // probabilities of state transition
      for (i=0; i < nS; i++)
      {
            _sum1 = 0;
            _sum2 = 0;

            for (j=0; j < nS; j++)
            {
                  cePs[i][j] = (float)rand() / (float)RAND_MAX;
                  _sum1 += cePs[i][j];
            }

            for (j=0; j < nS-1; j++)
            {
                  cePs[i][j] = cePs[i][j] / _sum1;
                  _sum2 += cePs[i][j];
            }

            cePs[i][nS-1] = 1.0f - _sum2;
```

```
	}

	// initial state
	ceState = uRand(nS);
}

// ------------------------------------------------------------------------
// response of commutative environment


int ceResponse (void)
{
	int _r;

	// get response in current state
	float rnum = (float)rand() / (float)RAND_MAX;

	if (rnum < cePa[ceState][action])   _r = REWARD;
	else                                            _r = PENALTY;

	// commutate states
	ceState = dRand(cePs[ceState],nS);


	return _r;
}
```

Д.3. Приклад програмної реалізації обчислювального експерименту для дослідження методів навчання з підкріпленням

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

#define ENVTYPE         0
#define NACTIONS        2
#define NSTATES         2

#define NSTEPS          200
#define NREPLICAS       1000

#define REWARD          1
#define PENALTY         0

int t;
int T = NSTEPS;
int n = NREPLICAS;
int nA = NACTIONS;
int nS = NSTATES;

int env = ENVTYPE;

int agt;
int action=0;                 // current action = {0, ... ,(nA-1)}
int response;                 // current response of the environment = {0;1}

int paction;

float sumR;
float avrR;

// ---------------------------------------------------------------------------
// tabulated results

float _sumR[NSTEPS][NREPLICAS];
float _avrR[NSTEPS][NREPLICAS];

// ---------------------------------------------------------------------------
// final simulation results

float sumRm[NSTEPS];               // mean values of sumR(t)
float sumRv[NSTEPS];               // corresponding variances

float avrRm[NSTEPS];               // mean values of avrR(t)
float avrRv[NSTEPS];               // corresponding variances

// ---------------------------------------------------------------------------
// files for parameters and results

char * par_file_name = "./lab1.parameters.dat";
FILE * par_file;

char * RA_res_file_name = "./lab1.RA.results.dat";
FILE * RA_res_file;

char * PA_res_file_name = "./lab1.PA.results.dat";
```

```
FILE * PA_res_file;

// ------------------------------------------------------------------------
// environment


int environment (int _en)
{
      int _r = 0;

      switch (_en)
      {
            case 0: _r = seResponse(); break;
            case 1: _r = ceResponse(); break;
            default: printf("lab1 error: wrong env code specified\n");
      }

      return _r;
}

// ------------------------------------------------------------------------
// save parameters to file


void saveParameters (void)
{
      int i,j;

      if ((par_file = fopen(par_file_name,"w")) == NULL) {
            fprintf(stderr, "Cannot open file <%s> for parameters of
experiment.\n", par_file_name);
      }

      fprintf(par_file,"T = %d\n", T);
      fprintf(par_file,"n = %d\n", n);

      fprintf(par_file,"env = %d\n", env);
      fprintf(par_file,"nA = %d\n", nA);
      if (env) fprintf(par_file,"nS = %d\n", nS);
      fprintf(par_file,"====================\n");

      switch (env)
      {
            case 0: // se (stationary environment)

                  for (i=0; i < nA; i++)
                        fprintf(par_file,"p(a%d) = %f\n", i, sePa[i]);

                  break;

            case 1: // ce (commutative environment)

                  // probabilities of rewards
                  for (i=0; i < nS; i++)
                  {
                        for (j=0; j < nA; j++)
                              fprintf(par_file,"p(s%d,a%d) = %f\n", i, j,
cePa[i][j]);

                        if (i < nS-1) fprintf(par_file,"--------------------
\n");
```

```
                  }

                  fprintf(par_file,"\n====================\n");

                  // probabilities of state transition
                  for (i=0; i < nS; i++)
                  {
                        for (j=0; j < nS; j++)
                              fprintf(par_file,"p(s%d,s%d) = %f\n", i, j,
cePs[i][j]);

                        fprintf(par_file,"--------------------\n");
                  }

                  break;

            default: printf("lab1 error: wrong env model code specified\n");
      }

      fclose(par_file);
}

// ---------------------------------------------------------------------------
// save results of random agent


void saveResultsRA (void)
{
      int i;

      if ((RA_res_file = fopen(RA_res_file_name,"w")) == NULL)
            fprintf(stderr, "Cannot open file <%s> for experimental results.\n",
RA_res_file_name);

      for (i=0; i < T; i++)
            fprintf(RA_res_file,"%f,%f,%f,%f\n", sumRm[i], sumRv[i], avrRm[i],
avrRv[i]);

      fclose(RA_res_file);
}

// ---------------------------------------------------------------------------
// save results of perfect agent


void saveResultsPA (void)
{
      int i;

      if ((PA_res_file = fopen(PA_res_file_name,"w")) == NULL)
            fprintf(stderr, "Cannot open file <%s> for experimental results.\n",
PA_res_file_name);


      for (i=0; i < T; i++)
            fprintf(PA_res_file,"%f,%f,%f,%f\n", sumRm[i], sumRv[i], avrRm[i],
avrRv[i]);

      fclose(PA_res_file);
}
```

```
// ------------------------------------------------------------------------
// return index of maximal value in <_array>


int argmax(float* _array, int size)
{
      int _arg = uRand(size);
      float _max = _array[_arg];

      for (int i=0; i < size; i++)
            if (_array[i] > _max) {_max = _array[i]; _arg = i;}

      return _arg;
}

// ------------------------------------------------------------------------
// init agent


void initAgent (int _ag)
{
      switch (_ag)
      {
            case 0:
                  break;

            case 1:
                  break;

            default: printf("lab1 error: wrong agent code specified\n");
      }
}

// ------------------------------------------------------------------------
// random agent


int randomAgent (void)
{
      return uRand(nA);
}

// ------------------------------------------------------------------------
// perfect agent


int perfectAgent (void)
{
      if (env) paction = argmax(cePa[ceState],nA);
      else     paction = argmax(sePa,nA);

      return paction;
}

// ------------------------------------------------------------------------
// choose the type of an agent


int agent (int _ag)
{
      int _a = 0;
```

```
	switch (_ag)
	{
		case 0: _a = randomAgent(); break;
		case 1: _a = perfectAgent(); break;
		default: printf("lab1 error: wrong agent code specified\n");
	}

	return _a;
}

// ----------------------------------------------------------------------------
// simulation


void simulation (int _i)
{
	initAgent(agt);
	sumR = 0.0f;
	avrR = 0.0f;

	for (t=0; t < T; t++) {

		// get action of agent
		action = agent(agt);

		// get response of environment
		response = environment(env);

		// calculate cumulative results
		sumR = sumR + (float)response;
		avrR = sumR / ((float)t + 1);

		// save results
		_sumR[t][_i] = sumR;
		_avrR[t][_i] = avrR;
	}
}

// ----------------------------------------------------------------------------
// get mean values of simulation results


void getMeanValues (void)
{
	for (t=0; t < T; t++)
	{
		float tmps1 = 0.0f;
		float tmps2 = 0.0f;

		for (int i=0; i < n; i++)
		{
			tmps1 += _sumR[t][i];
			tmps2 += _avrR[t][i];
		}

		sumRm[t] = (float)tmps1 / (float)n;
		avrRm[t] = (float)tmps2 / (float)n;
	}
}
```

```
// ------------------------------------------------------------------------
// get variances of simulation results


void getVarianceValues (void)
{
	for (t=0; t < T; t++)
	{
		float tmps1 = 0.0f;
		float tmps2 = 0.0f;

		for (int i=0; i < n; i++)
		{
			tmps1 += (sumRm[t] - _sumR[t][i]) * (sumRm[t] - _sumR[t][i]);
			tmps2 += (avrRm[t] - _avrR[t][i]) * (avrRm[t] - _avrR[t][i]);
		}

		sumRv[t] = (float)tmps1 / (float)(n-1);
		avrRv[t] = (float)tmps2 / (float)(n-1);

		//sumRv[t] = (float)tmps1 / (float)n;
		//avrRv[t] = (float)tmps2 / (float)n;
	}
}

// ------------------------------------------------------------------------
// main


int main(int argc, char* argv[])
{
	int i;

	// init random-number generator
	srand((unsigned)time(NULL));

	// init environment
	if (env == 0)	seInit();
	else		ceInit();

	// save parameters of experiment
	saveParameters();

	// run experiment for random agent
	agt = 0;
	for (i=0; i < n; i++) simulation(i);
	getMeanValues();
	getVarianceValues();
	saveResultsRA();

	// run experiment for perfect agent
	agt = 1;
	for (i=0; i < n; i++) simulation(i);
	getMeanValues();
	getVarianceValues();
	saveResultsPA();

	return 0;
}
```

Д.4. Приклади програмної реалізації конструкцій автоматів, що навчаються

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <tchar.h>
#include <math.h>

#define ENVTYPE         0
#define NACTIONS  2

#define REWARD          1
#define PENALTY         0

#define LATYPE          2 //3 //4
#define LAMEMSIZE 8

int t;
int nA = NACTIONS;

int action=0;
int response;

int memSize = LAMEMSIZE;
int state;

// --------------------------------------------------------------------------
// learning automaton with linear tactics (Tsetlin's automaton)


int LLA (void)
{
      int _action = action;

      if (response > 0) // 1 -> reward
      {
            if (state < memSize) state++; // step up in current branch
      }
      else // 0 -> penalty
      {
            if (state == 1)
            {
                  // change action (change branch of automaton)
                  if (action == (nA-1))   _action = 0;
                  else                          _action = action + 1;
            }
            else state--; // step down in current branch
      }
/*
      // compact version
      (response>0)?
            ((state<memSize)?state++:state):
            ((state==1)?
                  ((action==(nA-1))?
                        (_action=0):
                        (_action++)):
                  (state--));

      // one line version
      int r=response, s=state, m=memSize, a=action;
```

```
	(r>0)?((s<m)?s++:s):((s==1)?((a==(nA-1))?(a=0):(a++)):(s--)); state=s;
return a;
*/
	return _action;
}

// ------------------------------------------------------------------------
// trustful learning automaton (Krinsky's automaton)


int TLA (void)
{
	int _action = action;

	if (response > 0) // 1 -> reward
	{
		if (state < memSize) state = memSize; // go to the deepest state in
current branch
	}
	else // 0 -> penalty
	{
		if (state == 1)
		{
			// change action (change branch of automaton)
			if (action == (nA-1))	_action = 0;
			else				_action = action + 1;
		}
		else state--; // step down in current branch
	}
/*
	// compact version
	(response>0)?
		((state<memSize)?state=memSize:state):
		((state==1)?
			((action==(nA-1))?
				(_action=0):
				(_action++)):
			(state--));

	// one line version (int r=response, s=state, m=memSize, a=action;)
	int r=response, s=state, m=memSize, a=action;
	(r>0)?((s<m)?s=m:s):((s==1)?((a==(nA-1))?(a=0):(a++)):(s--)); state=s;
return a;
*/
	return _action;
}

// ------------------------------------------------------------------------
// inertial learning automaton (Robinson's automaton)


int ILA (void)
{
	int _action = action;

	if (response > 0) // 1 -> reward
	{
		if (state < memSize) state = memSize; // go to the deepest state in
current branch
	}
	else // 0 -> penalty
```

```
	{
		if (state == 1)
		{
			// change action (change branch of automaton)
			if (action == (nA-1))	_action = 0;
			else				_action = action + 1;

			// go to the deepest state in new branch
			state = memSize;
		}
		else state--; // step down in current branch
	}
/*
	// compact version
	(response>0)?
		((state<memSize)?state=memSize:state):
		((state==1)?
			((action==(nA-1))?
				(_action=0):
				(state=memSize+_action++-_action)):
			(state--));

	// one line version (int r=response, s=state, m=memSize, a=action;)
	int r=response, s=state, m=memSize, a=action;
	(r>0)?((s<m)?s=m:s):((s==1)?((a==(nA-1))?(a=0):(s=m+a++-a):(s--));
state=s; return a;
*/
	return _action;
}
```

Д.5. Приклади програмної реалізації методів навчання з підкріпленням в MAB

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <tchar.h>
#include <math.h>

#define NACTIONS  10

#define REWARD          1
#define PENALTY   0

#define RLTYPE          3 //2 //4
#define RLEPSILON 0.1f
#define RLTAU           0.12f

int t;
int nA = NACTIONS;

int action=0;
int response;

float e = RLEPSILON;
float tau = RLTAU;

int k[NACTIONS];
int r[NACTIONS];

float Q[NACTIONS];
float p[NACTIONS];

// -----------------------------------------------------------------------------
// return index of maximal value in <_array>


int argmax(float* _array, int size)
{
      int _arg = uRand(size);
      float _max = _array[_arg];

      for (int i=0; i < size; i++)
            if (_array[i] > _max) {_max = _array[i]; _arg = i;}

      return _arg;
}

// -----------------------------------------------------------------------------
// init agent


void initAgent (int _ag)
{
      int i;

      switch (_ag)
      {
            case 0: break;
            case 1: break;
```

```
            case 2: for(i=0;i<nA;i++){k[i]=0; r[i]=0; Q[i]=1.0f;}; action =
uRand(nA); break;
            case 3: for(i=0;i<nA;i++){k[i]=0; r[i]=0; Q[i]=1.0f;}; action =
uRand(nA); break;
            case 4: for(i=0;i<nA;i++){k[i]=0; r[i]=0; Q[i]=0.0f; p[i]=0.0f;};
action = uRand(nA); break;
            default: printf("lab3 error: wrong agent code specified\n");
      }
}

// ---------------------------------------------------------------------------
// greedy RL


int greedy (void)
{
      int _action = action;

      // modify estimated action value
      r[action] += response;
      k[action]++;
      Q[action] = (float)r[action] / (float)k[action];

      // select next action
      _action = argmax(Q,nA);

      return _action;
}

// ---------------------------------------------------------------------------
// epsilon greedy RL


int epsilonGreedy (void)
{
      int _action = action;

      // modify estimated action value
      r[action] += response;
      k[action]++;
      Q[action] = (float)r[action] / (float)k[action];

      // select next action
      float rnum = (float)rand() / (float)RAND_MAX;
      if (rnum < e)     _action = uRand(nA);
      else              _action = argmax(Q,nA);

      return _action;
}

// ---------------------------------------------------------------------------
// softmax action selection


int softmax (void)
{
      int i;
      int _action = action;
      float tmp[NACTIONS], pSum = 0.0f;

      // modify estimated action value
```

```
	r[action] += response;
	k[action]++;
	Q[action] = (float)r[action] / (float)k[action];

	// modify values of selection probabilities
	for (i=0; i < nA; i++) {tmp[i] = expf(Q[i]/tau); pSum += tmp[i];}
	for (i=0; i < nA; i++) {p[i] = tmp[i]/pSum;}

	// select next action
	_action = dRand(p,nA);

	return _action;
}

// ------------------------------------------------------------------------
// agent


int agent (int _ag)
{
	int _a = 0;

	switch (_ag)
	{
		case 0: _a = randomAgent(); break;
		case 1: _a = perfectAgent(); break;
		case 2: _a = greedy(); break;
		case 3: _a = epsilonGreedy(); break;
		case 4: _a = softmax(); break;
		default: printf("lab3 error: wrong agent code specified\n");
	}

	return _a;
}
```

Д.6. Приклад програмної реалізації марківського процесу прийняття рішень

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <tchar.h>
#include <math.h>

#define ENVTYPE         1
#define NACTIONS        2
#define NSTATES         2

#define REWARD          10
#define PENALTY         0

int t;
int nA = NACTIONS;
int nS = NSTATES;

int env = ENVTYPE;

int mdpState;
int mdpR[NSTATES][NACTIONS];
float mdpT[NSTATES][NACTIONS][NSTATES];

int action=0;
int response;

int paction[NSTATES];          // actions of perfect agent (per state) (MDP)
float gammaVI = 0.1f;          // learning rate for value iteration (MDP)

float e = RLEPSILON;           // epsilon value (epsilon-greedy RL)
float tau = RLTAU;             // tau value (softmax action selection)

float k[NSTATES][NACTIONS];    // number of realizations for each action
float r[NSTATES][NACTIONS];    // total reward for each action

float Q[NSTATES][NACTIONS];    // estimated action value for each action;
float p[NACTIONS];             // selection probability for each action;

int mdpPrevState;              // previous state of MDP

// ------------------------------------------------------------------------
// initialization of mdp


void mdpInit (void)
{
      int i,j,v;
      int maxReward = REWARD;
      float _sum1, _sum2;

      // probabilities of rewards
      for (i=0; i < nS; i++)
            for (j=0; j < nA; j++)
                  mdpR[i][j] = uRand(maxReward);

      // probabilities of state transition
      for (i=0; i < nS; i++)
            for (j=0; j < nA; j++)
```

```
	{
		_sum1 = 0;
		_sum2 = 0;

		for (v=0; v < nS; v++)
		{
			mdpT[i][j][v] = (float)rand() / (float)RAND_MAX;
			_sum1 += mdpT[i][j][v];
		}

		for (v=0; v < nS-1; v++)
		{
			mdpT[i][j][v] = mdpT[i][j][v] / _sum1;
			_sum2 += mdpT[i][j][v];
		}

		mdpT[i][j][nS-1] = 1.0f - _sum2;
	}

	// initial state
	mdpState = uRand(nS);
}

// ------------------------------------------------------------------------
// response of mdp & state transition


int mdpResponse (void)
{
	int _r;

	// get response in current state
	_r = mdpR[mdpState][action];

	// commutate states
	mdpState = dRand(mdpT[ceState][action],nS);

	return _r;
}

// ------------------------------------------------------------------------
// return maximum value from <_array> of <size> elements


float max(float* _array, int size)
{
	int _arg = uRand(size);
	float _max = _array[_arg];

	for (int i=0; i < size; i++)
		if (_array[i] > _max) _max = _array[i];

	return _max;
}

// ------------------------------------------------------------------------
// return index of maximum value in <_array> of <size> elements


int argmax(float* _array, int size)
{
```

```
	int _arg = uRand(size);
	float _max = _array[_arg];

	for (int i=0; i < size; i++)
		if (_array[i] > _max) {_max = _array[i]; _arg = i;}

	return _arg;
}


// ---------------------------------------------------------------------------
// init perfect agent (for MDP)


void initPerfectAgent (void)
{
	int i,j,z;
	float sum=0.0f;

	// perform value iteration --> optimal value function V*(s)

	for (z=0; z < nS; z++) V[z] = 1.0f;

	for (t=0; t < T*30; t++)
	{
		for (i=0; i < nS; i++)
		{
			for (j=0; j < nA; j++)
			{
				sum = 0.0f;
				for (z=0; z < nS; z++) sum = sum + mdpT[i][j][z] * V[z];
				Qsa[i][j] = mdpR[i][j] + gammaVI * sum;
			}
			V[i] = max(Qsa[i],nA);
		}
	}

	// determine the optimal policy given the optimal value function
	for (i=0; i < nS; i++)
	{
		for (j=0; j < nA; j++)
			{
				sum = 0.0f;
				for (z=0; z < nS; z++) sum = sum + mdpT[i][j][z] * V[z];
				Qsa[i][j] = mdpR[i][j] + gammaVI * sum;
			}
		paction[i] = argmax(Qsa[i],nA);
	}
}

// ---------------------------------------------------------------------------
// perfect agent (for MDP)


int perfectAgent (void)
{
	return paction[mdpState];
}
```

Д.7. Приклади програмної реалізації методів навчання з підкріпленням в MDP

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <tchar.h>
#include <math.h>

#define ENVTYPE         1
#define NACTIONS        2
#define NSTATES         2

#define REWARD          10
#define PENALTY         0

#define RLTYPE          5
#define RLEPSILON       0.1f
#define RLTAU           0.12f

#define AHCALFA         0.3f
#define AHCGAMMA        0.1f

#define QLALFA          0.5f
#define QLGAMMA         0.1f

int t;
int nA = NACTIONS;
int nS = NSTATES;

int action=0;
int response;

int paction[NSTATES];         // actions of perfect agent (per state) (MDP)
float gammaVI = 0.1f;         // learning rate for value iteration (MDP)

float e = RLEPSILON;          // epsilon value (epsilon-greedy RL)
float tau = RLTAU;            // tau value (softmax action selection)

float k[NSTATES][NACTIONS];   // number of realizations for each action
float r[NSTATES][NACTIONS];   // total reward for each action

float Q[NSTATES][NACTIONS];   // estimated action value for each action;
float p[NACTIONS];            // selection probability for each action;

int mdpPrevState;             // previous state of MDP
float AHCresponse;            // current response of Adaptive Heuristic Critic

float alfaAHC = AHCALFA;      // learning rate (AHC)
float gammaAHC = AHCGAMMA;    // discount factor (AHC)

float alfaQL = QLALFA;        // learning rate (Q-learning)
float gammaQL = QLGAMMA;      // discount factor (Q-learning)

float V[NSTATES];             // estimated value of state (AHC)
float Qsa[NSTATES][NACTIONS]; // expected discounted reinforcement

// ----------------------------------------------------------------------
// init RL agent
```

```
void initRLAgent (void)
{
      int i,j,z;
      float sum=0.0f;

      // set initial values of estimated values of state (TD(0)))
      for (z=0; z < nS; z++) V[z] = 5.0f;

      // init parameters of RL sub-agent for TD(0)
      for (i=0; i < nS; i++)
            for (j=0; j < nA; j++)
                  {k[i][j]=0.0f; r[i][j]=0.0f; Q[i][j]=5.0f; p[i]=0.0f;}

      // set initial values of expected discounted reinforcements (Q-learning)
      for (i=0; i < nS; i++)
            for (j=0; j < nA; j++)
                        Qsa[i][j] = 5.0f;
}

// ------------------------------------------------------------------------
// greedy RL


int greedy (void)
{
      int ps = mdpPrevState;
      int _action = action;

      // modify estimated action value for previous state
      r[ps][action] += AHCresponse;
      k[ps][action]++;
      Q[ps][action] = r[ps][action] / k[ps][action];

      // select next action in current state
      _action = argmax(Q[mdpState],nA);

      return _action;
}

// ------------------------------------------------------------------------
// epsilon greedy RL


int epsilonGreedy (void)
{
      int ps = mdpPrevState;
      int _action = action;

      // modify estimated action value  for previous state
      r[ps][action] += AHCresponse;
      k[ps][action]++;
      Q[ps][action] = r[ps][action] / k[ps][action];

      // select next action in current state
      float rnum = (float)rand() / (float)RAND_MAX;
      if (rnum < e)     _action = uRand(nA);
      else              _action = argmax(Q[mdpState],nA);

      return _action;
}
```

```
// ------------------------------------------------------------------------
// softmax action selection


int softmax (void)
{
      int i;
      int ps = mdpPrevState;
      int _action = action;
      float tmp[NACTIONS], pSum = 0.0f;

      // modify estimated action value for previous state
      r[ps][action] += AHCresponse;
      k[ps][action]++;
      Q[ps][action] = r[ps][action] / k[ps][action];

      // modify values of actions' selection probabilities
      for (i=0; i < nA; i++) {tmp[i] = expf(Q[mdpState][i]/tau); pSum +=
tmp[i];}
      for (i=0; i < nA; i++) {p[i] = tmp[i]/pSum;}

      // select next action in current state
      _action = dRand(p,nA);

      return _action;
}

// ------------------------------------------------------------------------
// Adaptive Heuristic Critic -> AHC


int AHC (void)
{
      int ps, cs;
      int _action;

      // select first action randomly
      if (t == 0) { mdpPrevState = mdpState; return uRand(nA); }

      // adaptive heuristic critic
      ps = mdpPrevState; // previous state of MDP
      cs = mdpState; // current state of MDP
      V[ps] = V[ps] + alfaAHC * (((float)response + (gammaAHC * V[cs])) -
V[ps]);
      AHCresponse = V[ps];

      // call RL sub-agent

      _action = greedy();
      //_action = epsilonGreedy();
      //_action = softmax();

      mdpPrevState = mdpState;
      return _action;
}

// ------------------------------------------------------------------------
// Q-learning


int qLearning (void)
```

```
{
	int ps, cs;
	int a = action;
	int _action;
	float maxQ;

	// select first action randomly
	if (t == 0) { mdpPrevState = mdpState; return uRand(nA); }

	// modify estimated action value Q(s,a) (= expected discounted
reinforcement)
	ps = mdpPrevState; // previous state of MDP
	cs = mdpState; // current state of MDP
	maxQ = max(Qsa[cs],nA);
	Qsa[ps][a] = Qsa[ps][a] + alfaQL * (((float)response + (gammaQL * maxQ)) -
Qsa[ps][a]);

	// select next action
	_action = argmax(Qsa[cs],nA);

	mdpPrevState = mdpState;
	return _action;
}

// ------------------------------------------------------------------------
// agent


int agent (int _ag)
{
	int _a = 0;

	switch (_ag)
	{
		case 0: _a = randomAgent(); break;
		case 1: _a = perfectAgent(); break;
		case 5: _a = AHC(); break;
		case 6: _a = qLearning(); break;
		default: printf("lab3 error: wrong agent code specified\n");
	}

	return _a;
}
```

Д.8. Приклад програмної реалізації обчислювального експерименту для дослідження методів навчання з підкріпленням в багатоагентних системах

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <tchar.h>
#include <math.h>

#define ENVTYPE         1
#define NACTIONS        2
#define NSTATES         2

#define NAGENTS         10

#define REWARD          1
#define PENALTY         0

#define RLTYPE          5
#define RLEPSILON       0.1f
#define RLTAU           0.12f

int t;
int T = NSTEPS;
int n = NREPLICAS;
int nA = NACTIONS;
int nS = NSTATES;

int k;                  // index for agents' numeration
int nK = NAGENTS;       // number of agents

int env = ENVTYPE;

int agt;                // type of agent
int mas;                // type of multi-agent system

int action[NAGENTS];    // current action = {0, ... ,(nA-1)}
int response[NAGENTS];  // current response of environment = {0;1}/{-1;+1}

int ka[NAGENTS][NACTIONS];
int ra[NAGENTS][NACTIONS];

float Q[NAGENTS][NACTIONS];
float p[NAGENTS][NACTIONS];

// ---------------------------------------------------------------------------
// results for current replica


float sumR[NAGENTS];    // total reward over time: sumR(t)
float avrR[NAGENTS];    // average reward over time: avrR(t) = sumR(t)/t

float sumRmas;          // total reward over time for multi-agent system
float avrRmas;          // average reward over time   for multi-agent system

// ---------------------------------------------------------------------------
// tabulated results


float _sumR[NSTEPS][NREPLICAS];
```

```
float _avrR[NSTEPS][NREPLICAS];

// ------------------------------------------------------------------------
// final simulation results


float sumRm[NSTEPS];                // mean values of sumR(t)
float sumRv[NSTEPS];                // corresponding variances

float avrRm[NSTEPS];                // mean values of avrR(t)
float avrRv[NSTEPS];                // corresponding variances

// ------------------------------------------------------------------------
// files for parameters and results

char * par_file_name = "d:\\temp2\\lab4.parameters.dat";
FILE * par_file;

char * RA_res_file_name = "d:\\temp2\\lab4.MAS-RA.results.dat";
FILE * RA_res_file;

char * PA_res_file_name = "d:\\temp2\\lab4.MAS-PA.results.dat";
FILE * PA_res_file;

char * RL1_res_file_name = "d:\\temp2\\lab4.MAS-RL-ISO.results.dat";
FILE * RL1_res_file;

char * RL2_res_file_name = "d:\\temp2\\lab4.MAS-RL-INT.results.dat";
FILE * RL2_res_file;


// ------------------------------------------------------------------------

void saveParameters (void)
{
      int i,j;

      if ((par_file = fopen(par_file_name,"w")) == NULL) {
            fprintf(stderr, "Cannot open file <%s> for parameters of
experiment.\n", par_file_name);
      }

      fprintf(par_file,"T = %d\n", T);
      fprintf(par_file,"n = %d\n", n);

      fprintf(par_file,"env = %d\n", env);
      fprintf(par_file,"nA = %d\n", nA);
      fprintf(par_file,"nK = %d\n", nK);
      if (env) fprintf(par_file,"nS = %d\n", nS);

      fprintf(par_file,"RL-agent type = %d\n", RLTYPE);
      if (agt==3) fprintf(par_file,"epsilon = %f\n", e);
      if (agt==4) fprintf(par_file,"tau = %f\n", tau);

      fprintf(par_file,"====================\n");

      switch (env)
      {
            case 0: // se (stationary environment)

                  for (i=0; i < nA; i++)
```

```
                        fprintf(par_file,"p(a%d) = %f\n", i, sePa[i]);

                  break;

            case 1: // ce (commutative environment)

                  // probabilities of rewards
                  for (i=0; i < nS; i++)
                  {
                        for (j=0; j < nA; j++)
                              fprintf(par_file,"p(s%d,a%d) = %f\n", i, j,
cePa[i][j]);

                        if (i < nS-1) fprintf(par_file,"--------------------
\n");
                  }

                  fprintf(par_file,"\n====================\n");

                  // probabilities of state transition
                  for (i=0; i < nS; i++)
                  {
                        for (j=0; j < nS; j++)
                              fprintf(par_file,"p(s%d,s%d) = %f\n", i, j,
cePs[i][j]);

                        fprintf(par_file,"--------------------\n");
                  }

                  break;

            default: printf("lab6 error: wrong env model code specified\n");
      }

      fclose(par_file);
}

// ---------------------------------------------------------------------------
// save results of random agent


void saveResultsRA (void)
{
      int i;

      if ((RA_res_file = fopen(RA_res_file_name,"w")) == NULL)
            fprintf(stderr, "Cannot open file <%s> for experimental results.\n",
RA_res_file_name);

      for (i=0; i < T; i++)
            fprintf(RA_res_file,"%f,%f,%f,%f\n", sumRm[i], sumRv[i], avrRm[i],
avrRv[i]);

      fclose(RA_res_file);
}

// ---------------------------------------------------------------------------
// save results of perfect agent


void saveResultsPA (void)
```

```
{
      int i;

      if ((PA_res_file = fopen(PA_res_file_name,"w")) == NULL)
            fprintf(stderr, "Cannot open file <%s> for experimental results.\n",
PA_res_file_name);


      for (i=0; i < T; i++)
            fprintf(PA_res_file,"%f,%f,%f,%f\n", sumRm[i], sumRv[i], avrRm[i],
avrRv[i]);

      fclose(PA_res_file);
}

// ------------------------------------------------------------------------
// save results of RL-agent


void saveResultsRL_ISO (void)
{
      int i;

      if ((RL1_res_file = fopen(RL1_res_file_name,"w")) == NULL)
            fprintf(stderr, "Cannot open file <%s> for experimental results.\n",
RL1_res_file_name);


      for (i=0; i < T; i++)
            fprintf(RL1_res_file,"%f,%f,%f,%f\n", sumRm[i], sumRv[i], avrRm[i],
avrRv[i]);

      fclose(RL1_res_file);
}

// ------------------------------------------------------------------------
// save results of RL-agent


void saveResultsRL_INT (void)
{
      int i;

      if ((RL2_res_file = fopen(RL2_res_file_name,"w")) == NULL)
            fprintf(stderr, "Cannot open file <%s> for experimental results.\n",
RL2_res_file_name);


      for (i=0; i < T; i++)
            fprintf(RL2_res_file,"%f,%f,%f,%f\n", sumRm[i], sumRv[i], avrRm[i],
avrRv[i]);

      fclose(RL2_res_file);
}

// ------------------------------------------------------------------------
// init agent


void initAgent (int _ag, int id)
{
```

```
      int i;

      switch (_ag)
      {
            case 0: break;
            case 1: break;
            case 2: for(i=0;i<nA;i++){ka[id][i]=0; ra[id][i]=0; Q[id][i]=1.0f;};
action[id] = uRand(nA); break;
            case 3: for(i=0;i<nA;i++){ka[id][i]=0; ra[id][i]=0; Q[id][i]=1.0f;};
action[id] = uRand(nA); break;
            case 4: for(i=0;i<nA;i++){ka[id][i]=0; ra[id][i]=0; Q[id][i]=0.0f;
p[id][i]=0.0f;}; action[id] = uRand(nA); break;
            default: printf("lab4 error: wrong agent code specified\n");
      }
}

// ---------------------------------------------------------------------------
// simulation


void simulation (int _i)
{
      float tmpS = 0.0f;
      float tmpA = 0.0f;

      initMAS(mas);
      sumRmas = 0.0f;
      avrRmas = 0.0f;

      for (t=0; t < T; t++) {

            // get action of each agent
            for (k=0; k < nK; k++) action[k] = agent(agt,k);

            // get response of environment for each agent
            for (k=0; k < nK; k++) response[k] = environment(env,k);

            // calculate cumulative results for each agent
            tmpS = 0.0f;
            tmpA = 0.0f;
            for (k=0; k < nK; k++)
            {
                  sumR[k] = sumR[k] + (float)response[k];         tmpS = tmpS +
sumR[k];
                  avrR[k] = sumR[k] / ((float)t + 1);             tmpA = tmpA +
avrR[k];
            }

            // calculate cumulative results for malti-agent system
            sumRmas = tmpS / (float)nK;
            avrRmas = tmpA / (float)nK;

            // save results
            _sumR[t][_i] = sumRmas;
            _avrR[t][_i] = avrRmas;
      }
}

// ---------------------------------------------------------------------------
// get mean values of simulation results
```

```
void getMeanValues (void)
{
	for (t=0; t < T; t++)
	{
		float tmps1 = 0.0f;
		float tmps2 = 0.0f;

		for (int i=0; i < n; i++)
		{
			tmps1 += _sumR[t][i];
			tmps2 += _avrR[t][i];
		}

		sumRm[t] = (float)tmps1 / (float)n;
		avrRm[t] = (float)tmps2 / (float)n;
	}
}

// ------------------------------------------------------------------------
// get variances of simulation results


void getVarianceValues (void)
{
	for (t=0; t < T; t++)
	{
		float tmps1 = 0.0f;
		float tmps2 = 0.0f;

		for (int i=0; i < n; i++)
		{
			tmps1 += (sumRm[t] - _sumR[t][i]) * (sumRm[t] - _sumR[t][i]);
			tmps2 += (avrRm[t] - _avrR[t][i]) * (avrRm[t] - _avrR[t][i]);
		}

		sumRv[t] = (float)tmps1 / (float)(n-1);
		avrRv[t] = (float)tmps2 / (float)(n-1);

		//sumRv[t] = (float)tmps1 / (float)n;
		//avrRv[t] = (float)tmps2 / (float)n;
	}
}

// ------------------------------------------------------------------------
// main


int main(int argc, char* argv[])
{
	int i;

	// init random-number generator
	srand((unsigned)time(NULL));

	// init environment
	if (env == 0)	seInit();
	else		ceInit();

	// save parameters of experiment
	saveParameters();
```

```
        // run experiment for MAS with random agents
        mas = 0; agt = 0;
        for (i=0; i < n; i++) simulation(i);
        getMeanValues();
        getVarianceValues();
        saveResultsRA();

        // run experiment for MAS with perfect agents
        mas = 1; agt = 1;
        for (i=0; i < n; i++) simulation(i);
        getMeanValues();
        getVarianceValues();
        saveResultsPA();

        // run experiment for MAS with RL-agent + Concurrent Isolated RL
        mas = 2; agt = RLTYPE;
        for (i=0; i < n; i++) simulation(i);
        getMeanValues();
        getVarianceValues();
        saveResultsRL_ISO();

        // run experiment for MAS with RL-agent + Interactive RL
        mas = 3; agt = 10 + RLTYPE;
        for (i=0; i < n; i++) simulation(i);
        getMeanValues();
        getVarianceValues();
        saveResultsRL_INT();

        return 0;
}
```

Д.9. Приклади програмної реалізації методів навчання з підкріпленням в багатоагентній системі

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <tchar.h>
#include <math.h>

#define ENVTYPE         1
#define NACTIONS        2
#define NSTATES         2

#define NAGENTS         10

#define REWARD          1
#define PENALTY         0

#define RLTYPE          5
#define RLEPSILON       0.1f
#define RLTAU           0.12f

int t;
int T = NSTEPS;
int n = NREPLICAS;
int nA = NACTIONS;
int nS = NSTATES;

int k;                  // index for agents' numeration
int nK = NAGENTS;       // number of agents

int env = ENVTYPE;

int agt;                // type of agent
int mas;                // type of multi-agent system

int action[NAGENTS];    // current action = {0, ... ,(nA-1)}
int response[NAGENTS];  // current response of environment = {0;1}/{-1;+1}

int ka[NAGENTS][NACTIONS];
int ra[NAGENTS][NACTIONS];

float Q[NAGENTS][NACTIONS];
float p[NAGENTS][NACTIONS];

// ------------------------------------------------------------------------
// greedy RL


int greedy (int id)
{
      int _action = action[id];

      // modify estimated action value
      ra[id][action[id]] += response[id];
      ka[id][action[id]]++;
      Q[id][action[id]] = (float)ra[id][action[id]] / (float)ka[id][action[id]];

      // select next action
      _action = argmax(Q[id],nA);
```

```
      return _action;
}

// ------------------------------------------------------------------------
// epsilon greedy RL


int epsilonGreedy (int id)
{
      int _action = action[id];

      // modify estimated action value
      ra[id][action[id]] += response[id];
      ka[id][action[id]]++;
      Q[id][action[id]] = (float)ra[id][action[id]] / (float)ka[id][action[id]];

      // select next action
      float rnum = (float)rand() / (float)RAND_MAX;
      if (rnum < e)     _action = uRand(nA);
      else              _action = argmax(Q[id],nA);

      return _action;
}

// ------------------------------------------------------------------------
// softmax action selection


int softmax (int id)
{
      int i;
      int _action = action[id];
      float tmp[NACTIONS], pSum = 0.0f;

      // modify estimated action value
      ra[id][action[id]] += response[id];
      ka[id][action[id]]++;
      Q[id][action[id]] = (float)ra[id][action[id]] / (float)ka[id][action[id]];

      // modify values of selection probabilities
      for (i=0; i < nA; i++) {tmp[i] = expf(Q[id][i]/tau); pSum += tmp[i];}
      for (i=0; i < nA; i++) {p[id][i] = tmp[i]/pSum;}

      // select next action
      _action = dRand(p[id],nA);

      return _action;
}

// ------------------------------------------------------------------------
// interactive greedy RL


int greedyINT (int id)
{
      int _action = action[id];

      // modify estimated action value
      ra[id][action[id]] += response[id];
      ka[id][action[id]]++;
```

```
	Q[id][action[id]] = (float)ra[id][action[id]] / (float)ka[id][action[id]];

	// get action values from neighbours -> average out
	int left = (id==0)?(nK-1):(id-1);
	int right = (id==(nK-1))?0:(id+1);
	for (int i=0; i<nA; i++) Q[id][i] = (Q[left][i] + Q[id][i] + Q[right][i])
/ 3.0f;

	// select next action
	_action = argmax(Q[id],nA);

	return _action;
}

// ------------------------------------------------------------------------
// interactive epsilon greedy RL


int epsilonGreedyINT (int id)
{
	int _action = action[id];

	// modify estimated action value
	ra[id][action[id]] += response[id];
	ka[id][action[id]]++;
	Q[id][action[id]] = (float)ra[id][action[id]] / (float)ka[id][action[id]];

	// get action values from neighbours -> average out
	int left = (id==0)?(nK-1):(id-1);
	int right = (id==(nK-1))?0:(id+1);
	for (int i=0; i<nA; i++) Q[id][i] = (Q[left][i] + Q[id][i] + Q[right][i])
/ 3.0f;

	// select next action
	float rnum = (float)rand() / (float)RAND_MAX;
	if (rnum < e)	_action = uRand(nA);
	else		_action = argmax(Q[id],nA);

	return _action;
}

// ------------------------------------------------------------------------
// interactive softmax action selection


int softmaxINT (int id)
{
	int i;
	int _action = action[id];
	float tmp[NACTIONS], pSum = 0.0f;

	// modify estimated action value
	ra[id][action[id]] += response[id];
	ka[id][action[id]]++;
	Q[id][action[id]] = (float)ra[id][action[id]] / (float)ka[id][action[id]];

	// get action values from neighbours -> average out
	int left = (id==0)?(nK-1):(id-1);
	int right = (id==(nK-1))?0:(id+1);
	for (int i=0; i<nA; i++) Q[id][i] = (Q[left][i] + Q[id][i] + Q[right][i])
/ 3.0f;
```

```
	// modify values of selection probabilities
	for (i=0; i < nA; i++) {tmp[i] = expf(Q[id][i]/tau); pSum += tmp[i];}
	for (i=0; i < nA; i++) {p[id][i] = tmp[i]/pSum;}

	// select next action
	_action = dRand(p[id],nA);

	return _action;
}

// ------------------------------------------------------------------------
// agent


int agent (int _ag, int id)
{
	int _a = 0;

	switch (_ag)
	{
		case 0: _a = randomAgent(); break;
		case 1: _a = perfectAgent(); break;
		case 2: _a = greedy(id); break;
		case 3: _a = epsilonGreedy(id); break;
		case 4: _a = softmax(id); break;
		case 12: _a = greedyINT(id); break;
		case 13: _a = epsilonGreedyINT(id); break;
		case 14: _a = softmaxINT(id); break;

		default: printf("lab4 error: wrong agent code specified\n");
	}

	return _a;
}

// ------------------------------------------------------------------------
// init mas


void initMAS (int _mas)
{
	int i;

	switch (_mas)
	{
		case 0: for(i=0;i<nK;i++){initAgent(agt,i); sumR[i]=0.0f;
avrR[i]=0.0f;} break;
		case 1: for(i=0;i<nK;i++){initAgent(agt,i); sumR[i]=0.0f;
avrR[i]=0.0f;} break;
		case 2: for(i=0;i<nK;i++){initAgent(agt,i); sumR[i]=0.0f;
avrR[i]=0.0f;} break;
		case 3: for(i=0;i<nK;i++){initAgent(agt-10,i); sumR[i]=0.0f;
avrR[i]=0.0f;} break;
		default: printf("lab4 error: wrong mas code specified\n");
	}
}
```

НАВЧАЛЬНЕ ВИДАННЯ

НАВЧАННЯ З ПІДКРІПЛЕННЯМ В АВТОНОМНИХ ІНТЕЛЕКТУАЛЬНИХ СИСТЕМАХ

Навчальний посібник

Редактор
Технічний редактор
Комп'ютерне верстання
Художник-дизайнер

Режим доступу:
http://eom.lp.edu.ua/textbooks/np-npais.pdf

Видавець і виготівник: Видавництво Львівської політехніки
Свідоцтво суб'єкта видавничої справи ДК №4459 від 27.12.2012 р.

вул. Ф. Колесси, 4, Львів, 79013
тел. +380 32 2584103, факс +380 32 2584101
vlp.com.ua, ел. пошта: vmr@vlp.com.ua