УДК 004.89

Р. В. Вовнянка, О. В. Оборська
Національний університет “Львівська політехніка”,
кафедра інформаційних систем та мереж

МОДЕЛЮВАННЯ ПОВЕДІНКИ ІНТЕЛЕКТУАЛЬНОГО АГЕНТА НА ОСНОВІ СТИМУЛЮЮЧОГО НАВЧАННЯ

Розглянуто моделювання поведінки інтелектуального агента на основі стимулюючого навчання. Наведено відповідні алгоритми, які використовують для моделювання поведінки таких агентів.

Ключові слова: інтелектуальний агент, модель кінцевого горизонту, індекси розподілу Гіттінса, динамічне програмування, марковський процес.

The article deals with modeling the rational agent behavior based on incentive learning. Matching algorithms that are used to model the behavior of agents are introduced.

Key words: rational agent, model finite horizon, Gittins allocation indices, dynamic programming, markov process.

Вступ. Постановка проблеми у загальному вигляді

Інтелектуальним агентом називається агент, який виконує “правильні дії”. Що розуміють під виразом “виконання правильних дій”? У першому наближенні можна сказати, що правильною дією є така дія, яка забезпечує найуспішніше функціонування агента. Тому потрібний певний спосіб вимірювання успіху. Критерії успіху, разом з описом середовища, а також давачів і виконавчих механізмів агента, надають повну специфікацію завдання, з яким стикається агент. Маючи ці компоненти, ми можемо визначити точніше, що розуміти під словом "інтелектуальний". Одним зі способів вимірювання такого успіху є використання методів, які ґрунтуються на стимульованому навчанні. Під час стимульованого навчання агент, що навчається, взаємодіє з навколишнім середовищем, виконуючи якісь дії (вибираючи їх, звичайно, з фіксованого набору). Навколишнє середовище його якось заохочує за ці дії, а агент продовжує їх виконувати. Єдиний спосіб для агента зрозуміти, що він чинить правильно, – стежити за заохоченнями від навколишнього середовища. Формально задача має такий вигляд. Нехай на кожному кроці агент перебуває у стані s з деякої множини станів S. На кожному кроці він вибирає з наявного набору дій A деяку дію a. У відповідь на це навколишнє середовище повідомляє агенту, яку винагороду він отримав і в якому стані. У загальному випадку – агент повинен досліджувати навколишнє середовище і вибирати оптимальну поведінку. Виникає задача: як порівнювати і оцінювати алгоритми стимульованого навчання під час діяльності інтелектуальних агентів?

Аналіз останніх досліджень та публікацій

Проаналізуємо алгоритми, що навчаються отримувати винагороду від навколишнього середовища. Такі алгоритми «мають добру поведінку», тобто отримують велику винагороду. Але що таке «добре»? На це запитання є декілька відповідей, вибір між якими залежить від того, на який термін життя агента вони розраховані й на що необхідно звернути увагу у своїх оцінках.

Перша, проста модель – це так звана модель кінцевого горизонту (finite horizon model). У ній якість поведінки агента вимірюється тільки щодо h кроків, а також здатністю максимізувати потрібне математичне сподівання $E\left[\sum_{t=0}^{h} r_t\right]$, де r_t – виграш [1]. Ця модель відповідає ситуації, коли в агента обмежений термін життя. Наприклад: ми знаємо, що у нас лише десять спроб, і тому нас не цікавить, чи успішно ми навчимося виконувати одинадцяту.

Природно, хотілося б врахувати всі можливі кроки в майбутньому, тому що час життя агента може бути не визначений. Але в більшості реальних задач що раніше ми отримаємо нагороду від навколишнього середовища, то краще. Наприклад, гривня сьогодні – це набагато краще, ніж гривня через рік, адже її за цей час можна розумно вкласти і примножити.

Як це врахувати в математичній моделі? Для цього досить надати швидшому прибутку більші ваги, а віддаленішому в часі – менші. За рахунок уведеного коефіцієнта γ та ще й у разі підсумовування за нескінченною довжиною життя агента ряд почне збігатися (вважають, що виплати r_t обмежені зверху – врешті-решт, кількість різних станів і різних виплат скінченна). Отже, математичне очікування суми ряду має вигляд:

$$E\left[\sum_{t=0}^{\infty}\gamma^t r_t\right],$$

де γ – деяка константа (звана в англомовних текстах *discount factor*).

Така модель називається моделлю нескінченного горизонту (*infinite horizon model*). У нашому подальшому викладі користуватимемося саме нею [2].

Третя модель, яка все рідше трапляється, – модель середньої винагороди (*average-reward model*). У ній максимізації підлягає границя очікування середньої винагороди:

$$\lim_{h\to\infty} E\left[\frac{1}{h}\sum_{t=0}^{h} r_t\right].$$

Ця модель не дає швидкого виграшу, тому на практиці нечасто є кращою за модель нескінченного горизонту [1]. Всі вищеописані моделі різні, й нескладно побудувати граф станів, на якому всі три моделі приводитимуть до різних оптимальних стратегій. Окрім поведінки вже готового алгоритму, треба також навчитися оцінювати і якість його навчання – те, наскільки якісно він прагне до своєї границі.

Щонайперша (і зазвичай її не надто складно довести) властивість алгоритмів стимульованого навчання – збіжність до оптимального алгоритму (для конкретної моделі, звичайно). Це корисна властивість, але, на жаль, вона нічого не говорить ні про швидкість збіжності, ні про те, з якими втратами ми підійдемо до оптимального алгоритму. Щодо оцінки швидкості збіжності, то тут можливі два альтернативні підходи: оцінка швидкості збіжності до якоїсь фіксованої частини оптимальності (власне оптимального алгоритму агент, що навчається, зазвичай досягає лише на нескінченності) й оцінка якості роботи алгоритму через деякий фіксований час. На жаль, в обох випадках виникають конкретні запитання: яка потрібна частина оптимальності? через який час треба порівнювати якість алгоритмів? Однозначну відповідь на ці запитання знайти важко.

Третій можливий підхід – мінімізувати ціну (regret) вибору тієї чи іншої стратегії навчання. Будь-яка стратегія не може бути кращою за оптимальну стратегію із самого початку. Інакше кажучи, задача оптимального навчання – це задача мінімізації різниці між загальною сумою виграшу при навчанні порівняно із застосуванням оптимальної стратегії із самого початку. Якщо вдається підрахувати цю різницю, то можна дуже просто порівнювати алгоритми навчання – у якого різниця менша, той і переміг. Ця міра, власне, й відображає справжню кінцеву мету того, що відбувається. Але, на жаль, вона погано піддається аналізу: розумні результати щодо цієї міри отримати дуже складно. Кожний алгоритм стимульованого навчання повинен вивчати навколишнє середовище і користуватися своїми знаннями, щоб максимізувати виграш. Виникає запитання – як досягти оптимального співвідношення? Саме підхід до розв'язування такої задачі описано у цій статті.

Формулювання мети

Розробити метод та алгоритм стимулюючого навчання інтелектуальних агентів, який би давав змогу ефективно використовувати знання агента для максимізації виграшу.

Аналіз отриманих наукових результатів

Функціонування інтелектуальних агентів

У будь-який конкретний момент часу оцінка дій агента залежить від чотирьох таких чинників:

- показники продуктивності, які визначають критерії успіху;
- знання агента про середовище, набуті раніше;
- дії, які може виконати агент;
- послідовність актів сприйняття агента, які відбулися дотепер.

З урахуванням цих чинників можна сформулювати таке визначення інтелектуального агента. Для кожної можливої послідовності актів сприйняття інтелектуальний агент повинен вибрати дію, що, як очікується, максимізувала б його показники продуктивності, з урахуванням фактів, наданих певною послідовністю актів сприйняття і всіх вбудованих знань, якими володіє агент [2], тобто здійснювати раціональні дії.

Раціональність не можна розглядати як синонім досконалості. Раціональність – це максимізація очікуваної продуктивності, а досконалість – максимізація фактичної продуктивності. Відмовляючись від прагнення до досконалості, ми не тільки застосовуємо до агентів справедливі критерії, але і враховуємо реальність. Річ у тому, що якщо від агента вимагають, щоб він виконував дії, які виявляються якнайкращими після їх здійснення, то завдання проектування агента, що відповідає цій специфікації, стає нездійсненним.

Раціональний вибір залежить тільки від послідовності актів сприйняття, сформованих до заданого моменту. Необхідно також стежити за тим, щоб ми ненавмисно не дозволили агентові брати участь в діях, які, безумовно, не є інтелектуальними.

Наше визначення вимагає, щоб інтелектуальний агент не тільки збирав інформацію, але й також навчався максимально можливо на тих даних, які він сприймає. Початкова конфігурація агента може відображати деякі попередні знання про середовище, але, у міру набуття агентом досвіду, ці знання можуть модифікуватися і поповнюватися. Існують крайні випадки, коли середовище повністю відоме наперед. У такому разі агентові не потрібно сприймати інформацію або навчатися; він просто відразу діє правильно.

У агентів, що успішно діють, завдання обчислення функції агента ділять на три окремі періоди: під час проектування агента деякі обчислення здійснюють його проектувальники; додаткові обчислення агент здійснює, вибираючи одну зі своїх чергових дій; а у міру того, як агент вчиться на підставі досвіду, він здійснює інші допоміжні обчислення, щоб вирішити, як модифікувати свою поведінку.

Якщо ступінь, в якому агент покладається на апріорні знання свого проектувальника, а не на свої сприйняття, дуже високий, то вважають, що у агента недостатня автономність. Інтелектуальний агент має бути автономним – він повинен навчатися всьому, що може засвоїти, для компенсації неповних або неправильних апріорних знань. На практиці агентові рідко ставлять вимогу, щоб він був повністю автономним з самого початку: якщо агент має мало досвіду або взагалі не має досвіду, то вимушений діяти довільно, якщо проектувальник не надав йому певної допомоги. Тому в штучному інтелекті агентові надають деякі початкові знання, а не тільки наділяють його здатністю навчатися. Після достатнього досвіду існування в своєму середовищі поведінка інтелектуального агента може за суттю стати незалежною від його апріорних знань. Тому введення у проект здібності до навчання дає змогу проектувати простих інтелектуальних агентів, які можуть діяти успішно у різноманітних варіантах середовища.

Формалізація функціонування інтелектуальних агентів

Рухатимемося від простого до складного. Розглянемо поведінку інтелектуальних агентів, що мають лише один стан ($|S|=1$). У такого агента є фіксований набір дій A і можливість вибору дії з цього набору. У штучному інтелекті загальноприйнята витончена і доступна для розуміння модель поведінки такого агента. Нехай агент перебуває в кімнаті з декількома ігровими автоматами. У кожного автомата є своє, невідоме агентові очікування виграшу. Агент повинен за обмежену

кількість спроб отримати максимальний виграш. Зазначимо, що така (здавалося б, сильно спрощена) модель насправді реалізує всі можливі ситуації з одним станом. Справді, за кожну виконану дію середовище дає винагороду (можливо, випадково) і повертає агента в початковий стан (результат наступного підходу до автомата не залежить від попередніх).

Розглянемо загальні принципи, якими повинен керуватися агент у такій ситуації. Зрозуміло, хочеться застосувати в якомусь сенсі «жадібну» стратегію, тобто завжди вибирати стратегію, яка максимізує виграш. Проте абсолютно очевидно, що агент в кімнаті не знає із самого початку, що йому робити, і не може вибрати найкращу стратегію. Понад це, навіть якщо він вже смикнув по одному разу за ручку кожного автомата і лише один раз переміг, це зовсім не означає, що треба тепер весь час вибирати автомат, на якому він переміг. Для цього в літературі використовується евристика – оптимізм за невизначеності [1, 3–5]. Інакше кажучи, вибирати стратегію поведінки треба "жадібну", але бути вельми оптимістичним щодо очікуваного виграшу. Необхідно отримати серйозні негативні наслідки, перш ніж ту чи іншу стратегію відхилити.

Розглянемо методи, які гарантовано знаходять оптимальний розв'язок або збігаються до нього. Одним із таких методів є динамічне програмування. Динамічне програмування широко використовується для різних задач. Не стало винятком і стимульоване навчання. Динамічне програмування тут можна застосувати, якщо заздалегідь відомий термін життя агента: допустимо, агент діє упродовж h кроків. Тоді для визначення оптимальної стратегії можна використати нескладний байєсівський підхід. Треба обчислити відображення зі всіх можливих станів досвіду (*belief states*) агента в множину дій, задавши його стратегію поведінки [6].

Стан досвіду агента виражається як $S=\{n_1, w_1, ..., n_k, w_k\}$, що означає, що кожний автомат i запустили n_i разів, отримавши при цьому виграш w_i разів (тут і далі вважаємо, що результат бінарний – або виграв, або програв). Позначимо через $V^*(S)$ очікуваний виграш. Базою для рекурентних співвідношень стане випадок, коли $\sum_{i=1}^{k} n_i = h$. У такому разі агентові більше нічого робити, і $V^*(S)=0$. Якщо ж ми знаємо V^* для всіх станів, коли у агента ще залишилося t спроб, зможемо перерахувати V^* і для станів з t+1 спробою, що залишилася:

$$V^*=\{n_1, w_1, ..., n_k, w_k\} = \max_i \left(p_i \left(1+V^*(...., n_i+1, w_i+1, ...)\right) + (1-p_i) V^*(...., n_i+1, w_i+1, ...)\right),$$

де p_i – апостеріорна ймовірність того, що дія i виправдається за даних досвіду (n_i, w_i). У випадку з випробуваннями Бернуллі:

$$p_i = \frac{w_i+1}{n_i+2}.$$

Отриманий алгоритм наведено на рис. 1. Цей алгоритм вийшов не дуже ефективним: ціна побудови такої таблиці лінійна до добутку кількості станів досвіду на кількість дій $|A|$, що експоненціально залежить від h. Інакше кажучи, динамічне програмування можна використовувати тільки у випадках, коли потрібно планувати не дуже далеко. Зате такий підхід гарантовано приводить до оптимальної стратегії. V^* абсолютно не залежить від конкретної ситуації, а від конкретних очікувань виграшу автоматів. Метод динамічного програмування побудує шлях по цій таблиці, який вибере оптимальна стратегія. Нехай ми вже n разів зіграли з автоматом й отримали w одиниць виграшу. Існують таблиці індексів розподілу Гіттінса (Gittins allocation indices) $I(n,w)$, які враховують як очікуваний виграш, так і кількість нової інформації, яку ми отримаємо, виконавши цю дію ще раз. Отже, оптимальна стратегія проста: вибирати автомат, для якого значення $I(n,w)$ максимальне [7]. Індекси розподілу Гіттінса працюють відмінно, але, на жаль, ніяк не узагальнюються – їх аналоги для декількох станів агента невідомі.

1. Ініціалізувати таблицю $\left\{V^* = \left(n_1, w_1, ..., n_k, w_k\right)\right\}_{n_i, w_i = 1}^{h}$.

2. Для всіх елементів, для яких $\sum_{i=1}^{n} n_i \geq h$, присвоїти $V^* = \left\{n_1, w_1, ..., n_k, w_k\right\} := 0$.

3. Для всіх *t* від *h* до 0:

для всіх елементів, для яких $\sum_{i=1}^{n} n_i = t$, присвоїти

$$V^* = \left\{n_1, w_1, ..., n_k, w_k\right\} :=$$

$$= \max_i \left\{ \frac{w_i + 1}{n_i + 2} \left(1 + V^*\left(...., n_i + 1, w_i + 1, ...\right)\right) + \left(1 - \frac{w_i + 1}{n_i + 2}\right) V^*\left(...., n_i + 1, w_i, ...\right) \right\}.$$

4. На кожному кроці із заданим фіксованим $\left\{n_1, w_1, ..., n_k, w_k\right\}$, вибираємо дію *i*, для якої

$$\max_i \left\{ \frac{w_i + 1}{n_i + 2} \left(1 + V^*\left(...., n_i + 1, w_i + 1, ...\right)\right) + \left(1 - \frac{w_i + 1}{n_i + 2}\right) V^*\left(...., n_i + 1, w_i, ...\right) \right\}.$$

Рис. 1. Динамічне програмування для стимульованого навчання

Індекс Гіттінса є мірою винагороди, якої можна досягти за допомогою розвиваючого процесу від його теперішнього стану і далі з ймовірністю, що вона буде припинена в майбутньому. Це справжній скаляр, пов'язаний зі станом стохастичного процесу з функцією винагороди і з ймовірністю припинення. Для послідовності доходів індекс Гіттінса показує найбільший з можливих, середній переоцінений дохід за одну одиницю переоціненого часу, що дорівнює базовій стадії, для фрагментів послідовності, є частиною основної послідовності доходів і починається з першої стадії. Індексне правило полягає в тому, що оптимальна стратегія на кожному кроці вибирає стадію і проект з найбільшим індексом Гіттінса.

Третій прийом серед тих, аналіз яких виконати досить просто, пов'язаний з «тренуванням» оптимальної стратегії. Справді, в моделі з кімнатою з автоматами будь-який алгоритм можна подати як набір імовірності, з якою він вибирає ту чи іншу дію (все решта не важливо для кінцевого результату). І цю ймовірність можна тренувати приблизно як нейронну мережу. Алгоритм лінійної винагороди–бездіяльності (*linear reward-inaction algorithm*, рис. 2) лінійно збільшує ймовірність дії a_i, якщо вона привела до успіху:

$$p_i := p_i + \alpha\left(1 - p_i\right),$$
$$p_j := p_j - \alpha p_j, j \neq i.$$

Якщо вона безуспішна, то ймовірність зберігається. У цих формулах α – константа, задає швидкість навчання.

Алгоритм лінійної винагороди-бездіяльності з імовірністю 1 збігається до вектора з однієї одиниці та решти нулів. Він не завжди збігається до оптимальної стратегії; цілком ймовірно, що не дуже характерні перші декілька запусків «навчать» алгоритм настільки, що він вже ніколи не повернеться до оптимального варіанта, а зведе до одиниці ймовірність субоптимального вибору. Однак ймовірність помилитися можна зробити як завгодно малою, зменшуючи α.

1. Випадково ініціалізувати ймовірності p_i.

2. Поки агент продовжує роботу:

а) вибрати дію *i* з ймовірністю p_i;

б) якщо дія привела до успіху:

$$p_i := p_i + \alpha\left(1 - p_i\right),$$
$$p_j := p_j - \alpha p_j, j \neq i.$$

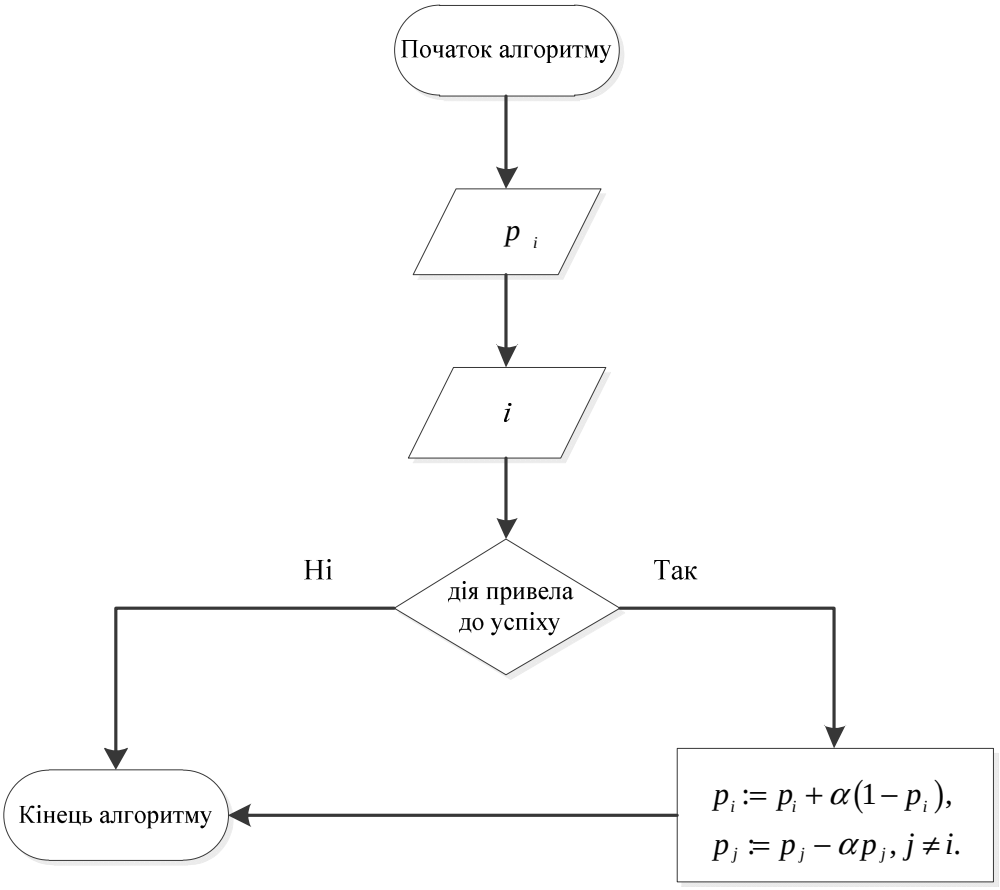

Рис. 2. Алгоритм лінійної винагороди-бездіяльності

Крім алгоритмів, які доказово збігаються до оптимального розв'язку, бувають також випадки, в яких оптимальність абсолютно не гарантована, однак на практиці виявляється, що метод працює добре.

Розглянемо детальніше такі випадки. Перший з них – випадкові стратегії. Проста випадкова стратегія виглядає так: вибрати дію з якнайкращим очікуваним виграшем з імовірністю p, а з імовірністю $1-p$ вибрати випадкову дію. Щоб імовірність не збіглася до субоптимальної дії, зазвичай починають з маленьких значень p, а потім їх збільшують [8].

Основний недолік цього алгоритму очевидний: він виділяє лідера, але не відрізняє хорошу йому альтернативу від поганої. Щоб врахувати відмінності між всіма стратегіями, треба «розмазати» ймовірність по всіх діях, беручи до уваги їх очікуваний виграш, але не забуваючи про додаткові дослідження. Одним з відповідних методів є дослідження за Больцманом (*Boltzmann exploration*):

$$p(a) = \frac{e^{ER(a)/T}}{\sum_{a'} e^{ER(a')/T}},$$

де ER – очікуваний виграш, T – фіксована константа. Ця константа визначає, наскільки велика різниця між ймовірністю вибору хороших і поганих стратегій; очевидно, що чим вища константа, тим ближче ймовірності одна до одної, а чим нижча, тим більша ймовірність вибору оптимальної та близької до неї дії. Зазвичай така константа починається з достатньо високого значення, а потім з часом знижується.

Друга евристична стратегія належить до класу «оптимістично жадібних» алгоритмів. Припустимо, що нам потрібно вибрати стратегії доволі оптимістично, але все ж таки врахувати наявний негативний досвід. Теорія ймовірності тут пропонує вельми розумний вихід – довірчі інтервали!

Для кожної дії треба зберігати статистику n і w, і перед ухваленням рішення обчислювати довірчий інтервал для ймовірності успіху (з деякою наперед заданою межею $1-\alpha$), а для вибору дії, що реалізовується, максимізувати верхню межу цього інтервалу.

Алгоритм стимулюючого навчання методом довірчих інтервалів наведено на рис. 3.

1. Ініціалізація довірчих інтервалів $I_k = (I_{1k}, I_{2k}) := (0,1)$.

2. Поки агент продовжує роботу:

а) вибрати дію $j = \arg\max_{i=1..k} I_k^2$.

б) Перерахувати довірчий інтервал I_j, залежно від результатів цієї дії.

Рис. 3. Алгоритм стимулюючого навчання методом довірчих інтервалів

Модель агентів з декількома станами

Вище розглянуто ситуацію – коли у агента рівно один стан, тобто впродовж всієї роботи алгоритму множина дій і їх результатів не змінюється. Проте в реальному житті алгоритмам, що навчаються, зазвичай треба переходити з одного стану в інший, перш ніж буде досягнутий якийсь певний результат. Отже, потрібно побудувати адекватну модель, що описує переходи між цими станами і таку, що задає максимізацію отриманого виграшу. Для цього пропонуємо використати марковські процеси.

Марковський процес прийняття рішень (*Markov decision process*) складається з:

- множини станів *S*;
- множини дій *A*;
- функції заохочення $R: S \times A \to R$;
- функції переходу між станами $T: S \times A \to \Pi(S)$, де $\Pi(S)$ – множина розподілів ймовірностей над *S*.

Отже, ймовірність потрапити зі стану *s* у стан *s'* після здійснення дії *a* позначається через $T(s,a,s')$. Зазначимо, що в нашій моделі переходи між станами ймовірнісні: після тієї чи іншої дії новий стан настає не з абсолютною необхідністю, а з деяким (заданим) розподілом ймовірності. А ось функцію винагороди для простоти вважаємо постійною. Переходи між станами не залежать від історії попередніх переходів, тобто, перебуваючи у певному стані, агент за однакові дії отримуватиме однакову винагороду, незалежно від того, як саме він потрапив у цей стан.

Задача полягає в максимізації виграшу. Зрозуміло, що в реальній ситуації на початку процесу агент абсолютно не знає – не відома реакція системи на жодні дії, зокрема й переходи між станами. Однак вважатимемо, що модель задачі відома. Саме розв'язок цієї задачі необхідний для розв'язування загальнішої. Тому спочатку розглянемо алгоритми, які знаходять оптимальну стратегію для відомої моделі, а потім перейдемо до складнішої задачі навчання.

Введемо таке поняття, як оптимальне значення стану. Оптимальне значення стану – очікуваний сумарний виграш, який отримає агент, якщо почне з цього стану і дотримуватиметься оптимальної стратегії:

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

Інакше кажучи, оптимальне значення стану – це та нагорода, яку отримуємо, якщо гратимемо найкраще. Це значення можна визначити як розв'язок рівнянь:

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

Якщо його знати, то вибір оптимальної стратегії здійснюємо згідно з формулою:

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

На рис. 4 та 5 наведено алгоритми, які розв'язують цю задачу ітераційним способом. Перший з них здійснює ітерації за значеннями, тобто намагається на кожному кроці підрахувати функцію $Q(s,a)$ – поточну версію очікування виграшу у випадку вибору дії a у стані s, а потім, коли значення цієї функції нас вже задовольняють, вибирає дію, яка максимізує це очікування. Алгоритм

зупиняється тоді, коли значення $Q(s,a)$ перестають змінюватися (тобто різниця між послідовними значеннями за модулем перестає перевищувати деяку заздалегідь фіксовану межу ε).

1. Ініціалізувати $V(s)$
2. Доки стратегія недостатньо добра:
для всіх $s \in S, a \in A$

$$Q(s,a) := R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V(s')$$

3. $V(s) := \max_a Q(s,a)$.

1. Ініціалізувати π.
2. Повторювати
   а) обчислити значення станів для стратегії π, розв'язавши систему лінійних рівнянь

$$V_\pi(s) = R(s,\pi(s)) + \gamma \sum_{s' \in S} T(s,\pi(s),s') V_\pi(s'),$$

   б) покращити стратегію на кожному стані:

$$\pi'(s) := \arg\max_a \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_\pi(s') \right).$$

3. Поки $\pi \neq \pi'$.

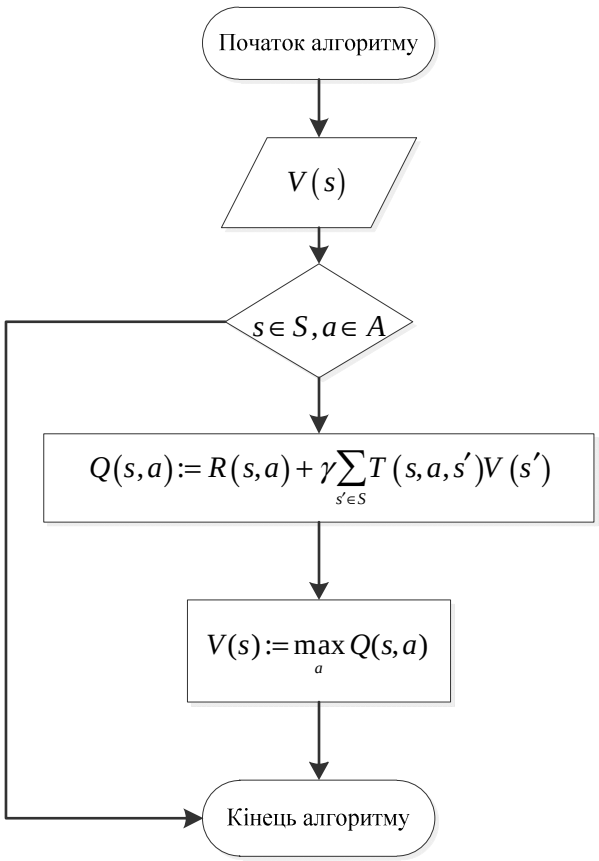

Рис. 4. Алгоритм пошуку оптимального значення стану ітераціями за значеннями

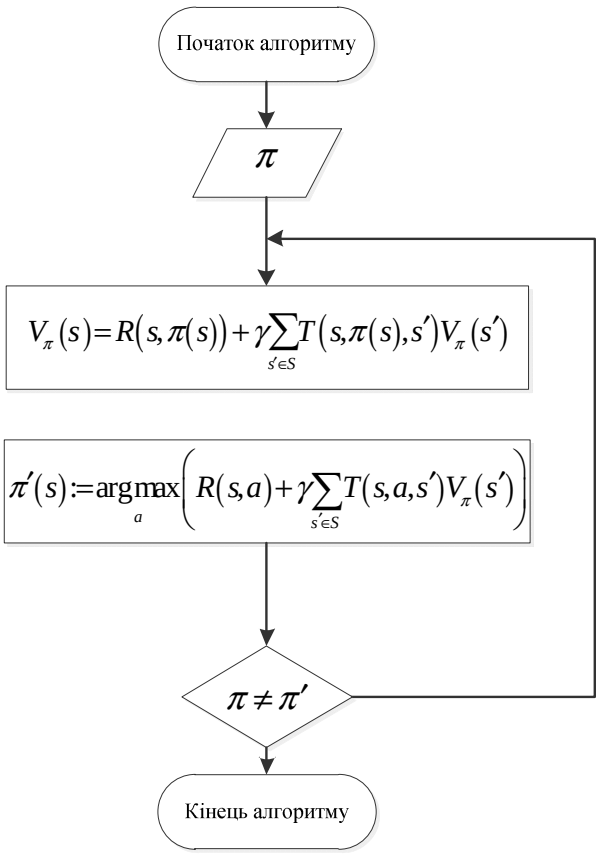

Рис. 5. Алгоритм пошуку оптимального значення стану ітераціями за стратегіями

Зазначимо, що перерахунок в цьому алгоритмі використовує інформацію від всіх станів-попередників. Але можна використати й інший, «стохастичний» варіант:

$$Q(s,a) := Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right).$$

Доведено, що він працює, якщо кожна пара (s,a) зустрічається нескінченну кількість разів, s' вибирають з розподілу $T(s,a,s')$, а r вибирають із середнім $R(s,a)$ і обмеженою варіацією.

Алгоритм ітерації за стратегіями (рис. 5) використовує ту саму ідею, але ітерації відбуваються не за очікуваними значеннями, а власне за стратегіями π, яких може дотримуватися агент. Алгоритм має більшу складність, ніж алгоритм ітерації за значеннями, оскільки потрібно на кожному кроці розв'язувати системи лінійних рівнянь. Зате для ітерацій за стратегіями очевидна збіжність, оскільки на кожному кроці цільова функція строго покращується, а всього існує скінченна кількість $\left(|A|^{|S|}\right)$ стратегій. Правда, це міркування дає значну оцінку на час роботи алгоритму; на практиці він працює куди швидше, але теоретично не відомо, чи він поліномний.

На основі побудованого математичного забезпечення надалі планують досліджувати поведінку інтелектуальних агентів як тактичних підрозділів збройних сил [9], а також раціональне функціонування вищих навчальних закладів [10, 11].

Висновки і перспективи подальших наукових розвідок

Запропоновано для моделювання поведінки інтелектуальних агентів використовувати стимулююче навчання. Розроблено відповідне математичне забезпечення залежно від кількості станів, у яких може перебувати такий агент, яке ґрунтується на динамічному програмуванні, теорії ймовірності та марковських процесах. Наведено відповідні алгоритми, які використовують для моделювання поведінки інтелектуальних агентів.

1. Литвин В. В. Методи та засоби інженерії даних та знань / В. В. Литвин. – Львів : Магнолія-2006, 2012. – 241 с. 2. Рассел С. Искусственный интеллект / С. Рассел, П. Норвиг. – М., С.-П., К.: Вильямс, 2006. – 1408 с. 3. Литвин В. В. Моделювання плану поведінки інтелектуального агента на основі мереж Петрі та онтологічного підходу / В. В. Литвин // Вісн. Нац. ун-ту «Львівська політехніка». Серія : Інформаційні системи та мережі. – 2009. – № 653. – С. 170–175. 4. Литвин В. В. Бази знань інтелектуальних систем підтримки прийняття рішень / В. В. Литвин. – Львів: Видавництво Львівської політехніки, 2011. – 240 с. 5. Інтелектуальні системи, базовані на онтологіях // Д. Г. Досин, В. В. Литвин, Ю. В. Нікольський, В. В. Пасічник. – Львів: Цивілізація, 2009. – 414 с. 6. Domingos P. (2012). A few useful things to know about machine learning / P.Domingos // Communications of the ACM, 2012. – 55(10). – P. 78–87. 7. Wooldridge M. An Introduction to MultiAgent Systems / M. Wooldridge. – 2nd edition, John Wiley &Sons, 2009. – 412 p. 8. Damien Ernst. Tree-based batch mode reinforcement learning / Damien Ernst, Pierre Geurts, Louis Wehenkel // Journal of Machine Learning Research, 2005. – P. 503–556. 9. Литвин В. В. Система підтримки прийняття рішень як складова автоматизованої системи управління сухопутних військ Збройних сил України / В. В. Литвин, Е. В. Лучук, П. П. Ткачук // Військово-науковий вісник. – 2013. – Вип. 2(9). – С. 43–46. 10. Литвин В. В. Метод моделювання процесу підтримки прийняття рішень у конкурентному середовищі / В. В. Литвин, О. В. Оборська, Р. В. Вовнянка // Математичні машини й системи : наук. журн. – Київ, 2014. – № 1. – С. 50–57. 11. Литвин В. В. Метод використання онтологій в петлі OODA на прикладі функціонування вищих навчальних закладів / В. В. Литвин, Р. В. Вовнянка Складні системи і процеси: наук. журн. / Запорізький інститут державного та муніципального управління. – 2012. – № 2(22). – С. 14–19.