УДК 681.3 http://doi.org/10.15407/pp2024.04.003

І. Гришанова, Ю. Рогушина

ТЕХНОЛОГІЯ ВИКОРИСТАННЯ МАШИННОГО НАВЧАННЯ ДЛЯ ПОБУДОВИ КОМПОЗИЦІЙНОГО ВЕБ-СЕРВІСУ

Проаналізовано алгоритми динамічного програмування та машинного навчання (на прикладі Q-learning), що використовуються для автоматичної адаптивної композиції веб-сервісів на основі оцінок якості сервісів, їхні вхідні параметри та особливості роботи. Для порівняння параметрів роботи цих алгоритмів на наборах сервісів різного обсягу розроблено програмну реалізацію.
Визначено, що розглянуті методи дозволяють знаходити оптимальний набір сервісів тільки для композиції з попередньо визначеним маршрутом фіксованої довжини. Тому виникає потреба узагальнити постановку задачі для довільного набору класів у маршруті композиції сервісів. На основі виконаного аналізу розроблено алгоритм розв'язання цієї задачі – побудови композитного сервісу з маршрутом довільної довжини методом Q-Learning, що має найкращі сумарні оцінки якості. Розроблено програмну реалізацію як цього алгоритму, так і інших алгоритмів розв'язання цієї задачі (генетичний алгоритм, жадібний пошук, динамічне програмування, SARSA), яка дозволила порівняти швидкість їх роботи та оцінки результуючого композитного сервісу на наборах даних різного обсягу.
Ключові слова: веб-сервіс, композиція сервісів, машинне навчання.

I. Grishanova, J. Rogushina

THE TECHNOLOGY OF MACHINE LEARNING FOR OF A COMPOSITE WEB SERVICE DEVELOPMENT

We analyze dynamic programming and machine learning algorithms (on example of Q-learning) used for automatic adaptive composition of web services based on service quality assessments, their input parameters and work specifics. Software implementation of these algorithms on sets of services of different volumes is developed for comparison their performance parameters.
We determine that the considered methods allow finding the optimal set of services only for composition with a predefined fixed-length route. This restriction causes a need to generalize the problem formulation for an arbitrary set of service classes in the composition route. On base of the performed analysis, we developed an algorithm that solves this problem of building a composite service with a route of arbitrary length (using the Q-Learning method), that has the best overall quality ratings. A software implementation of both this algorithm and other algorithms for solving this problem (genetic algorithm, greedy search, dynamic programming, SARSA, etc.) are developed to compare the speed of their work and the evaluation of the resulting composite service on data sets of different volumes.
Keywords: web service, composition of services, machine learning.

Вступ

У сучасному світі інформаційних технологій веб-сервіси стали важливим інструментом для надання функціональності через мережу. Веб-сервіси дозволяють розробникам об'єднувати окремі програмні компоненти для створення складних, функціональних та інтегрованих рішень. Композитні веб-сервіси відіграють ключову роль у створенні таких рішень, дозволяючи використовувати кілька сервісів у межах єдиного робочого процесу для досягнення поставлених цілей.

Композиція сервісів є активною сферою досліджень, яка застосовує розробки зі сфери машинного навчання та семантичних технологій [1], для перетворення веб-середовища на розподілену обчислювальну платформу. Ця діяльність потребує створення та дослідження відповідних концепцій, моделей, алгоритмів, платформ та інструментів.

Процес *композиції* веб-сервісів є досить складним, особливо в умовах динамічного середовища, де сервіси можуть бути тимчасово недоступними або змінювати свої характеристики, зокрема, параметри *якості обслуговування* (QoS). Крім того, потрібно швидко адаптувати систему до нових умов, обирати оптимальні сервіси, а також оптимізувати композитний сервіс за допомогою параметрів QoS.

Один із перспективних підходів до автоматичної адаптивної композиції веб-сервісів базується на використані машинного навчання, зокрема, навчання з підкріпленням, у якому агент навчається на основі взаємодії із середовищем через отримання винагороди за свої дії.

Використання одного з основних алгоритмів навчання з підкріпленням – *Q-learning* – дозволяє агенту поступово вивчати оптимальну стратегію вибору веб-сервісів для побудови композитних сервісів: він дозволяє агенту коригувати свої рішення на основі зворотного зв'язку від середовища, не потребуючи повної моделі цього середовища.

Машинне навчання і Reinforcement Learning

Машинне навчання – це підгалузь штучного інтелекту, яка полягає у використанні алгоритмів і моделей, що дозволяють комп'ютерам здобувати знання з даних та знаходити рішення без чіткої програми. Один із важливих класів машинного навчання – *навчання з підкріпленням* (reinforcement learning, RL)**.**

Навчання з підкріпленням вирізняється тим, що агент взаємодіє із середовищем у режимі реального часу, отримуючи зворотний зв'язок у вигляді винагороди або штрафу залежно від того, наскільки успішно його дія привела до досягнення поставленої мети.

Однією з найбільш популярних технік RL є група алгоритмів Q-learning, де агент будує таблицю Q-значень для різних станів і дій, поступово знаходячи оптимальну стратегію (policy) через спроби і помилки. На відміну від інших алгоритмів RL, Q-learning використовує прості Q-функції. Існує велика кількість підтипів Q-learning – з одним або кількома агентами, з різними типами взаємодії [2].

Завдяки тому, що агент постійно оновлює свої знання на основі отриманих винагород, він здатний адаптуватися до змін у середовищі, що робить його ефективним інструментом для задач, де середовище є динамічним або частково відомим.

У контексті веб-сервісів навчання з підкріпленням стає особливо важливим, оскільки середовище веб-сервісів є динамічним та складним, веб-сервіси можуть змінювати свої характеристики якості (Quality of service – QoS) [3], або ставати недоступними. Тому агент, що навчається, повинен вміти адаптуватися і швидко підбирати нові рішення для композиції.

Для побудови композиції сервісів потрібно спочатку визначити множину можливих маршрутів Flow (автоматизовано або вручну, як робить більшість дослідників), що визначає послідовність виконання сервісів, а потім для таких маршрутів визначити оптимальний набір сервісів.

Навчання з підкріпленням дозволяє агенту самостійно знаходити оптимальні комбінації веб-сервісів на основі їхніх параметрів QoS, причому значення цих параметрів можуть змінюватися динамічно в процесі виконання сервісів. Підхід Q-learning надає можливість вирішувати ці завдання через поступове покращення стратегії складання композитного сервісу, який буде найбільш оптимальним за заданими параметрами.

Задача композиції веб-сервісів

Розглянемо задачу створення композитного сервісу, взявши за основу приклад, наданий у роботі [4], поступово змінюючи вимоги для досягнення більш адаптивних можливостей в композиції веб-сервісів.

У контексті композиції веб-сервісів, *композитний сервіс* (КС) складається з набору окремих веб-сервісів, що виконують певні завдання, які об'єднуються для досягнення конкретної мети або виконання складної операції в певній послідовності.

Основними компонентами побудови композитного сервісу є:

1. **Атомарні вебсервіси**: це окремі веб-сервіси, які можуть виконувати певні операції або надавати функціональність, наприклад, отримання даних про погоду, конвертацію одиниць виміру тощо. Кожен такий атомарний веб-сервіс має атрибути якості (QoS), такі як: *Доступність (Availability):* Частота успішних викликів; *Час виконання (Execution Time):* Час, необхідний для виконання запиту; *Пропускна здатність (Throughput):* Кількість запитів, які можуть бути оброблені за одиницю часу.
2. **Класи веб-сервісів**: веб-сервіси можуть бути розподілені на *класи*, залежно від їхньої функціональності. Наприклад, один клас може містити веб-сервіси для отримання даних про погоду, інший – для конвертації одиниць виміру температури. КС складається з вибору веб-сервісів із кожного класу, які виконуються у певній послідовності.
3. **Маршрут (Flow)**: КС має впорядковану послідовність виконання атомарних веб-сервісів, де кожен із сервісів виконує свою задачу, а вихід одного сервісу може бути вхідним для іншого. Зокрема, перший сервіс отримує дані про температуру, а другий конвертує отриману температуру в іншу одиницю виміру (наприклад, з Фаренгейта в Цельсій).
4. **Оптимізація за параметрами якості**: КС формується таким чином, щоб мінімізувати або максимізувати певні показники якості сервісу, як-от, мінімальний час виконання або максимальну доступність.

Постановка задачі

В роботі аналізуються існуючі рішення для побудови композитних сервісів, що дозволяють це робити для зафіксованого маршруту, який визначає послідовності виконання сервісів, і обрати оптимальний набір сервісів. Оптимальність оцінюється на основі характеристик якості атомарних сервісів, що входять до складу КС.

Проблема полягає у тому, що більшість дослідників цієї задачі використовують різні форми опису маршруту та вхідних даних сервісів, які значно обмежують сферу застосування методів композиції, звужуючи її до фіксованого плану виконання сервісів. Це зменшує потенційну гнучкість сервіс-орієнтованої архітектури та потребує дослідження більш узагальнених початкових умов. Такий підхід потребує також дослідження методів побудови маршрутів та критеріїв їх порівняння, але ці дослідження є поза рамками даної публікації.

Щоб визначити можливості та недоліки алгоритмів вибору атомарних сервісів для побудови композитного сервісу на основі маршруту, потрібно створити програмну реалізацію різних варіантів для алгоритмів композиції сервісів та порівняти швидкість і якість їх виконання та навчання. Це дозволить визначити, які з цих алгоритмів найбільш придатні для розв'язання задачі композиції у динамічному середовищі і перспективні для подальшого удосконалення та які параметри впливають на їхню ефективність.

Методи композиції веб-сервісів з використанням QoS

Зараз існує велика кількість методів композиції веб-сервісів на основі QoS, що різняться часом виконання, структурою вхідних даних та кількістю ітерацій. Всі вони використовують модель Марковського Процесу Прийняття Рішення - Markov Decision Process (MDP) для ухвалення рішення щодо оптимальності кожного з окремих сервісів, що розглядаються як кандидати на виконання.

Метод динамічного програмування

Метод динамічного програмування [5] використовує алгоритми Policy Iteration, Value Iteration та Iterative Policy Evaluation [6].

Програмна реалізація цих алгоритмів[1] показала, що алгоритм Policy Iteration найшвидше знаходить оптимальні стратегії композиції веб-сервісів (з мінімальною кількістю ітерацій), тоді як Value Iteration та Iterative Policy Evaluation показують задовільні результати, але потребують більше часу.

Алгоритм Q-learning з фіксованим маршрутом композитного сервісу

Алгоритм Q-learning з фіксованим маршрутом композитного сервісу і без динамічної зміни класів сервісів та їх кількості реалізує композицію веб-сервісів за допомогою навчання RL, а саме – використовуючи Q-learning. Цей підхід може використовувати ті ж самі вхідні дані, що й попередньо розглянуті динамічні методи, але з акцентом на дослідженні простору станів без попередньо відомої функції переходів.

У програмній реалізації[2] алгоритм Q-learning для вибору оптимальної послідовності веб-сервісів використовує такі параметри QoS, як доступність (availability), час виконання (execution time) та пропускна здатність (throughput). КС формується через навчання агента на основі нагород, що відповідають атрибутам QoS. Агент взаємодіє з середовищем і поступово вчиться вибирати послідовність дій (тобто, виклик веб-сервісів) таким чином, щоб максимізувати сумарну винагороду. Наприклад, якщо агент вибирає сервіс із високою доступністю, але з довгочасним виконанням, він отримає меншу винагороду, ніж якщо він обере сервіс із збалансованими QoS-показниками.

Тестування показало, що алгоритм навчання з підкріпленням потребує більше часу порівняно з динамічним програмуванням (Policy Iteration або Value Iteration), оскільки Q-learning досліджує простір станів через багаторазові ітерації. Q-таблиця після завершення навчання зберігає найкращі очікувані винагороди для кожної пари "стан-дія". Це дозволяє агенту обирати оптимальні дії на основі попереднього досвіду.

З отриманих результатів можна зробити висновки: динамічні алгоритми (Policy Iteration, Value Iteration): працюють швидше, оскільки не вимагають дослідження простору станів та вимагають точного знання функцій переходів між станами, тоді як Q-learning більше відповідає середовищам з невідомими функціями переходів та потребує більше ітерацій, оскільки досліджує простір станів експериментальним шляхом. Таким чином, для проблеми композиції веб-сервісів, якщо відомі функції переходів між станами, динамічні методи показують кращі результати як за часом, так і за ефективністю. Проте, Q-learning є надійним рішенням для випадків, де функції переходів невідомі або змінюються динамічно.

Отримані нами висновки співпадають з результатами інших дослідників, але їхні підходи, а саме структури даних, не дозволяють знаходити оптимальний набір сервісів для композиції з попередньо не визначеним маршрутом. Тому виникає потреба узагальнити постановку задачі для довільного набору класів у маршруті композиції сервісів.

Знаходженням оптимальної композиції сервісів для довільного маршруту

Розглянемо використання Q-learning для композиції сервісів і знаходженням оптимальної композиції із заданими наборами груп вебсервісів і набором Flow композитного сервісу для довільного маршруту, який не зафіксовано явно до початку побудови композитного сервісу.

У попередньому прикладі з використанням Q-learning для композиції вебсервісів не було явного опису композитного сервісу або Flow, але він був заданий алгоритмічно. КС Flow має на увазі послі-

---

1 https://github.com/TurtleIren/test-ML-methods/blob/3db8d92c050b5dc66a65914756a988b8aad51d1e/WSComposition-VI-PI-IPE-compare.py

2 https://github.com/TurtleIren/test-ML-methods/blob/55315be65031ab0144ae6beec8c55ba01abb59ea/WSCompositionQ-learn-fixed-1.py

довність кількох окремих веб-сервісів, кожен з яких виконує частину загального завдання. У термінах коду Flow має бути представлене як послідовний вибір кількох дій (веб-сервісів) з різних класів, які спільно реалізують КС.

Ціллю алгоритму є знаходження оптимального КС із набору існуючих Flow, використовуючи алгоритм Q-learning. Після тренування алгоритм повинен обирати оптимальний Flow на основі максимальної суми винагород для кожного композитного сервісу. Такий алгоритм має розширений набір вхідних даних:

- *Класи веб-сервісів* **(**service_classes**)**: кожен клас, як і у розглянутих вище алгоритмах, містить набір веб-сервісів, згрупованих за функціональними характеристиками, із явно заданими параметрами QoS. Класи явно визначені, наприклад, клас 1 містить "сервіс_1", "сервіс_2";
- *Flow:* до вхідних даних додається опис множини маршрутів Flow, який визначає припустимі послідовності класів веб-сервісів довільної скінченої довжини, які можуть бути використані для створення композитного сервісу. Наприклад, flows = [[1, 2], [1, 3, 4]] означає, що перший Flow складається з веб-сервісу з класу 1 і веб-сервісу з класу 2, а другий Flow – з послідовності веб-сервісів з класів 1, 3 і 4;
- *Параметри QoS*: значення параметрів якості веб-сервісу (такі як доступність, час виконання та пропускна здатність).

Для реалізації поставленої задачі пропонується алгоритм композиції веб-сервісів із використанням Q-learning**,** який може адаптуватися до різної кількості веб-сервісів у класах і Flow, і забезпечити гнучкість у визначенні композиції (Flow) веб-сервісів.

Відповідно до моделі MDP для адаптивної композиції веб-сервісів, цей алгоритм описується через наступні терміни**:**

- *Стан* (state): кожен стан представляє часткову композицію веб-сервісів.
- *Дії* (actions): вибір веб-сервісу з певного класу для додавання в поточний Flow.
- *Нагорода* (reward): враховує атрибути QoS (доступність**,** час виконання, пропускна здатність).
- *Q-learning***:** алгоритм для навчання.

Цей алгоритм дозволяє обробляти адаптивні дані, а саме**:**

- кількість класів веб-сервісів може змінюватись;
- Flow композитного сервісу може містити різну кількість веб-сервісів на різних етапах;
- дані можуть генеруватись випадково для тестування різних сценаріїв.

Програмна реалізація цього методу[3] містить явно задані дані про веб-сервіси. У масиві service_data кожен веб-сервіс має фіксовані значення для атрибутів QoS: availability, exec_time, та throughput. Однак у процесі експлуатації ці значення можуть змінюватися, і відповідно до цих змін буде змінюватися значення винагороди і відповідно оптимальний Flow. Винагорода розраховується на основі значень QoS для обраного веб-сервісу. Для кожного Flow генерується КС. Метод find_optimal_flow() знаходить оптимальний Flow на основі суми винагород. Завдяки новій структурі з явним заданням Flow можна мати різну кількість веб-сервісів на різних етапах композиції. Крім того, можна змінювати кількість епізодів для навчання та інші параметри.

Після навчання для кожного Flow виводиться набір сервісів, які були обрані

- 3 https://github.com/TurtleIren/test-ML-methods/blob/068c4f1fdc0970702b448398e7e13dad4988955e/WSCompositionQ-learn-adaptive.py

як оптимальні саме для цього маршруту. Наприклад:

```
Згенерований композитний сервіс для Flow 1: ['сервіс_1', 'сервіс_3'] із загальною винагородою: 41.233
Згенерований композитний сервіс для Flow 2: ['сервіс_1', 'сервіс_6', 'сервіс_8'] , із загальною винагородою: 48.924
```

Потім алгоритм обирає найкращий Flow, ґрунтуючись на сукупній винагороді за всі сервіси, що увійшли до отриманого Flow. Таким чином, програма генерує композитний веб-сервіс із найкращою сумарною оцінкою якості атомарних сервісів.

```
Найкращий Flow: 2, із загальною винагородою: 48.924
```

Програма призначена для вирішення задачі композиції веб-сервісів із метою визначення оптимального композитного сервісу на основі наданих класів веб-сервісів і можливих Flow. КС формується на основі вибору сервісів з різних класів за допомогою підходу Q-learning, який належить до методів навчання з підкріпленням (reinforcement learning).

Програма здійснює навчання на основі винагород (reward) для вибору веб-сервісів із різних класів так, щоб максимізувати загальну якість сервісів (QoS). Основні модулі програми:

1. *Ініціалізація (конструктор класу):* Програма встановлює список станів та можливих дій для кожного Flow. Станами є різні етапи композитного Flow, а діями – вибраний веб-сервіс з відповідного класу.
2. *Генерація станів*: Метод generate_states() створює список можливих станів для кожного Flow.
3. *Генерація дій:* Метод generate_actions() генерує всі можливі дії для кожного стану, де кожна дія – це вибір одного веб-сервісу з певного класу.
4. *Обчислення винагороди (QoS):* Метод get_reward() обчислює винагороду за вибір конкретного веб-сервісу на основі його параметрів якості. Формула для винагороди враховує такі параметри, як доступність, час виконання і пропускну здатність:
   $$reward=(availability\times K1)-(executiontime\times K2)+(throughput\times K3)$$
   Значення коефіцієнтів важливості параметрів K1, K2 та K3 у формулі винагороди визначають вагу кожного параметра якості обслуговування (QoS): доступність (availability), час виконання (execution time) та пропускну здатність (throughput). Ці значення обираються емпірично або за вимогами користувача (наприклад, в даній програмній реалізації обрано $K1=0.4$, $K2=0.4$ та $K3=0.2$). Таким чином, веб-сервіси з високою доступністю та пропускною здатністю і мінімальним часом виконання отримують вищу винагороду.
5. *Вибір дій (epsilon-greedy):* Для вибору дій програма використовує епсілон-жадібну (*epsilon-greedy*) стратегію, яка дозволяє знаходити баланс між дослідженням нових сервісів та використанням уже знайдених оптимальних рішень: з імовірністю ϵ програма обирає випадковий веб-сервіс для дослідження (exploration); з імовірністю $1-\epsilon$ програма обирає дію з найвищим Q-значенням на основі поточної Q-таблиці.
6. *Оновлення Q-таблиці:* Метод update_q_table() використовує стандартну формулу Q-learning для оновлення значень Q у таблиці на основі поточної винагороди та очікуваної максимальної винагороди для наступного стану.
7. *Навчання*: Метод train() виконує навчання протягом заданої кількості епізодів (прикладів навчання). Під час навчання система проходить через кожен Flow, вибираючи сервіси, обчислюючи винагороди та оновлюючи Q-таблицю.
8. *Генерація композитного сервісу:* Метод generate_composite_service()

генерує КС для кожного Flow на основі навченої Q-таблиці, обираючи сервіси з найвищим значенням Q у кожному стані.

9. *Пошук оптимального Flow:* Метод find_optimal_flow() обчислює загальну винагороду для кожного Flow та визначає найкращий КС на основі максимальної суми винагород.

Алгоритм адаптується до різних Flow, які представляють різні варіанти складання композитних сервісів із класів веб-сервісів. Підхід із використанням навчання з підкріпленням дозволяє знайти оптимальний КС, орієнтуючись на параметри якості сервісів (QoS), такі як доступність, час виконання та пропускна здатність.

Визначення коефіцієнтів важливості винагороди

Визначення коефіцієнтів важливості винагороди може здійснюватися кількома різними способами:

Емпіричний підхід: Значення можуть бути підібрані емпірично на основі попереднього досвіду або аналізу системи. Вони можуть бути скориговані після серії експериментів або тестувань для забезпечення правильного балансу між різними параметрами QoS.

Вимоги системи або користувача: у різних системах або для різних сервісів деякі параметри можуть бути важливішими за інші. Наприклад, у критичних системах доступність може мати найбільший пріоритет, оскільки від неї залежить надійність сервісу. Інші системи можуть вимагати мінімізації **часу** виконання або оптимізації пропускної здатності.

Збалансований підхід: коефіцієнти також можуть бути підібрані для збалансованого впливу на загальну винагороду. Зокрема, використання однакових коефіцієнтів для доступності та часу виконання може означати, що система хоче забезпечити стабільну роботу сервісів з хорошою доступністю і водночас підтримувати короткий час відповіді. Менша вага для пропускної здатності свідчить про те, що її важливість у цьому сценарії менша.

Для визначення цих значень, необхідно здійснити наступне:

- *Аналіз потреб системи:* Якщо деякі параметри QoS є важливішими для вашої системи, ці значення можуть бути змінені відповідно до вимог. Наприклад, якщо в задачі важливий час виконання, можна збільшити коефіцієнт для execution time.
- *Тестування і оптимізація:* В процесі навчання з підкріпленням здійснюються серії тестів з різними значеннями коефіцієнтів для визначення найкращих налаштувань, які забезпечують оптимальну роботу системи.
- *Моделювання користувацького досвіду:* У деяких випадках користувачі можуть задати пріоритетність параметрів, виходячи з їхніх потреб. Наприклад, у хмарних сервісах для певних користувачів важливіша доступність, аніж час виконання, тоді як для інших навпаки.

Порівняння алгоритмів композиції сервісів

Крім Q-Learning, що розглянутий вище, для композиції сервісів можуть використовуватися інші алгоритми. Розглянемо найбільш поширені з них.

Алгоритм жадібного пошуку (Greedy algorithm): на кожному кроці обирається той сервіс, який має максимальну винагороду на поточному етапі, без урахування майбутніх винагород. Такий підхід не враховує довгострокові наслідки вибору на кожному етапі. Хоча він швидкий, але не гарантує глобально оптимальних результатів, оскільки може пропустити вигідніші дії у довгостроковій перспективі.

Еволюційні алгоритми (Genetic Algorithms): можуть використовуватися для пошуку оптимальних рішень у просторі можливих Flow, але вони потребують значного обчислювального часу та надто залежні від налаштування параметрів;

Динамічне програмування: складне завдання розбивається на підзадачі і вирі-

шує їх шляхом обчислення оптимальних результатів для менших підзадач. Алгоритм зберігає проміжні рішення для уникнення повторних обчислень. Він використовує таблицю для запам'ятовування оптимальних рішень для кожного стану. Як і жадібний алгоритм, динамічне програмування шукає оптимальні рішення, але на основі глобального аналізу. Цей підхід може бути використаний для знаходження оптимальних рішень, але він також вимагає попереднього знання всіх можливих станів і дій**.**

*SARSA***:** метод навчання з підкріпленням, яке, на відміну від Q-learning, оновлює значення Q-функції на основі виконаної дії та отриманої наступної дії (Q-Learning використовує максимальне Q-значення для наступної дії). SARSA [7] розглядає поточний стан, дію, винагороду, наступний стан і наступну дію, що робить його менш агресивним, але стабільнішим у змінних середовищах. Отож, SARSA є більш консервативним методом, оскільки оновлює Q-функцію відповідно до реальних дій, а не теоретично максимальних.

Щоб порівняти Q-Learning із цими методами, проведемо їх тестування на однакових наборах даних. Для проведення такого тестування ми змінюємо механізм отримання даних, а саме дані будуть братися із згенерованих csv-файлів.

Згенеруємо масив текстових даних для сценарію, де максимальна кількість сервісів у класі становить 100, максимальна довжина Flow – 100 сервісів[4]. Кожен клас міститиме довільну кількість сервісів, а Flow складатимуться з довільних класів і сервісів. Крім того, для кожного сервісу згенеруємо параметри QoS. Ми використовуємо наступні домовленості:

- Класи *вебсервісів:* У кожного класу випадкова кількість сервісів від 1 до 100. Для кожного сервісу генеруються параметри якості (QoS), такі як доступність, час виконання та пропускна здатність.
- *Flow:* Випадковим чином генеруються Flow з різних класів. Максимальна кількість класів у Flow – 100.

Файл services.csv[5] містить інформацію про всі сервіси та їхні параметри QoS, а flows.csv[6] містить послідовності Flow для тестування.

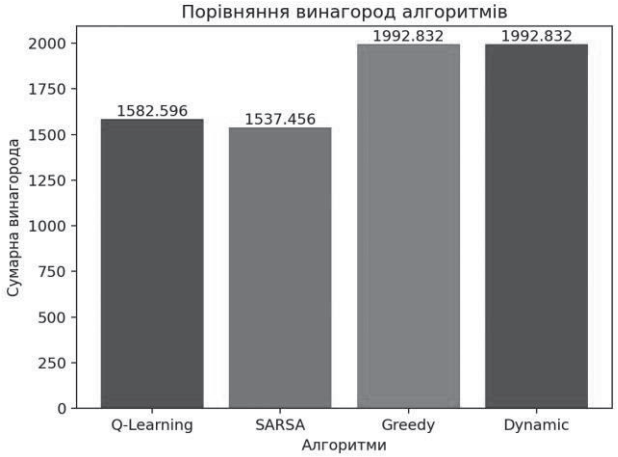

Рис. 1. Порівняння винагород знайденого композитного сервісу різними алгоритмами

Рис. 2. Порівняння часу навчання побудови композитного сервісу.

4 https://github.com/TurtleIren/test-ML-methods/blob/ed6f17d668e9e98382424924664579dfbc6a8860/generarte_100_services_100_flows.py

5 https://github.com/TurtleIren/test-ML-methods/blob/2bb687e16be812e74c613e2b4b919e136ca44860/services_100.csv

6 https://github.com/TurtleIren/test-ML-methods/blob/2bb687e16be812e74c613e2b4b919e136ca44860/flows_100.csv

Результати порівняльного тестування[7] показали наступне (Рис.1, Рис.2):

1. *Q-Learning:*

Переваги: Добре підходить для середовищ, де система потребує багато дослідження і поступового покращення на основі довгострокових вигод. Однак час виконання задовгий (50.57 секунд), а результат є субоптимальним (винагорода 1552.072), порівняно з іншими методами.

Недоліки: Час виконання значно більший порівняно з жадібним алгоритмом і динамічним програмуванням.

2. *SARSA:*

Переваги: Стабільніший у середовищах з динамічними змінами, оскільки враховує реальні дії. Час виконання (49.99 секунд) дещо менший, ніж у Q-Learning, а результат (винагорода 1574.544) дещо кращий.

Недоліки: Час виконання також доволі високий, і може не завжди знаходити глобально оптимальні рішення**.**

3. *Жадібний алгоритм:*

Переваги: Найшвидший алгоритм (0.07192 секунд), оскільки він просто обирає найкращу дію на кожному кроці без глибоких розрахунків. Він досягає найвищої винагороди (1992.832), що робить його ефективним для композиції веб-сервісів.

Недоліки: Може пропустити оптимальніші глобальні рішення через короткозорість, якщо деякі рішення вимагають короткострокових втрат.

4. *Динамічне програмування:*

Переваги: Показує другий найкращий результат після жадібного алгоритму, але також знаходить оптимальний Flow з найвищою винагородою (1992.832) і виконується дуже швидко (0.14223 секунд).

Недоліки: Використовує більше пам'яті, оскільки зберігає всі проміжні результати, але це виправдано швидкістю та результатами.

Нами також була спроба виконати поставлену задачу *методом генетичного алгоритму.* Цей метод працював довше на невеликих наборах даних, однак на тестовому прикладі він не зміг виконати поставлену задачу. Генетичний алгоритм часто потребує багатьох ітерацій, щоб зійтися до оптимального або навіть близького до оптимального рішення. Це призводить до повільного виконання порівняно з іншими методами. Генетичний алгоритм не завжди гарантує глобальне оптимальне рішення. Через випадкові процеси він може потрапити у локальні оптимуми, і без належного налаштування може зупинитися на субоптимальних рішеннях. Генетичний алгоритм потребує налаштування багатьох гіперпараметрів: розмір популяції, ймовірність мутації, ймовірність кросовера, умови зупинки тощо. Неправильне налаштування цих параметрів може значно вплинути на продуктивність і точність алгоритму. Випадкові операції мутації та кросовера можуть призвести до менш ефективного рішення або навіть віддалити від оптимального. В свою чергу це може призвести до нестабільних результатів між різними запусканнями алгоритму. Отже, генетичний алгоритм має потенціал для знаходження глобальних оптимальних рішень, але для задач композиції веб-сервісів на основі QoS з великими наборами даних він менш ефективний через високу обчислювальну складність і тривалість конвергенції.

Таким чином, можна зробити висновок, що для швидкої композиції вебсервісів на основі QoS найкраще використовувати жадібний алгоритм або динамічне програмування, оскільки вони демонструють відмінну швидкість виконання і знаходять глобально оптимальні рішення. Q-Learning та SARSA краще підходять для середовищ, де важливіша адаптація до динамічних змін і дослідження можливих рішень у довгостроковій перспективі, але вони працюють повільніше і можуть знаходити субоптимальні рішення.

Висновки

В роботі розглянуто застосування Q-learning для знаходження оптимального композитного сервісу з урахуванням даних QoS. Запропонована архітектура та структура даних є більш гнучкими та дозволя-

7 https://github.com/TurtleIren/test-ML-methods/blob/30b8fd6ed80a44bc776693ac6a35e6137ca3881c/WSComposition-compare.py

ють підтримувати більше класів або типів сервісів і містять потенціал для подальшого удосконалення адаптивності процесу автоматизації побудови КС. Структура станів є складнішою, де кожен стан може відображати наявні вхідні та вихідні дані веб-сервісів, що дає змогу використовувати більший простір станів і більше можливих шляхів для композиції. Використання однієї спільної Q-таблиця для всіх станів і дій дозволяє гнучкіше масштабувати та обробляти складніші комбінації. Запропонована програмна реалізація дозволяє продемонструвати переваги цього рішення. Відповідно до концепції відкритої науки щодо відтворюваності результатів, створений програмний код є у відкритому доступі, і його можна отримати за посиланнями, наведеними у статті.

Здійснене тестування рішення задачі композиції веб-сервісів різними методами, з якого можна зробити висновок, що для швидкої композиції веб-сервісів на основі QoS найкраще використовувати жадібний алгоритм або динамічне програмування, оскільки вони демонструють відмінну швидкість виконання і знаходять глобально оптимальні рішення. Але ці методи не придатні для великих або невідомих просторів станів і потребують повного знання середовища. Методи Q-Learning та SARSA краще придатні для середовищ, де важливіше значення мають адаптація до динамічних змін і дослідження можливих рішень у довгостроковій перспективі, але ці методи працюють повільніше і можуть знаходити субоптимальні рішення.

Тестування програми, що базується на використанні Q-learning, демонструє його ефективність у знаходженні оптимальних композитних сервісів на різних тестових наборах даних. Найкращий композитний сервіс обирається на основі максимізації суми винагород за параметрами якості сервісів QoS. Q-learning є більш гнучким порівняно з альтернативними підходами (такими як жадібний пошук та еволюційні алгоритми), оскільки він адаптується до змін у середовищі та дозволяє оцінювати довгострокові наслідки вибору на кожному етапі.

Наступним кроком у подальших дослідженнях може бути автоматизація складання маршрутів Flows для веб-сервісів з урахуванням змінних параметрів середовища. Автоматизація дозволить не тільки динамічно адаптувати сервіси під нові умови, а й покращити якість і швидкість складання оптимальних маршрутів на основі QoS. Крім того, доцільно буде дослідити можливості аналізу семантики складових сервісів та доповнити розглянуті вище підходи елементами інтелектуалізації описів сервісів на основі онтологій [8].

References

1. A. L. Lemos, F. Daniel, B. Benatallah, Web service composition: a survey of techniques and tools. *ACM Computing Surveys (CSUR*), 2015, 48(3), 1-41.
2. B. Jang, M. Kim, G. Harerimana, J. W. Kim, Q-learning algorithms: A comprehensive classification and applications. *IEEE access*, 2019б 7, 133653-133667.
3. M. Karakus, A. Durresi, Quality of service (QoS) in software defined networking (SDN): A survey. *Journal of Network and Computer Applications*, 2017, 80, 200-218.
4. V. Uc-Cetina, F. Moo-Mena, R. Hernandez-Ucan, Composition of web services using Markov decision processes and dynamic programming. *The Scientific World Journal,* 2015, (1), 545308.
5. S. Kalasapur, M. Kumar, B. A. Shirazi, Dynamic service composition in pervasive computing. *IEEE Transactions on parallel and distributed systems,* 2007, 18(7), 907-918.
6. A. Alla, M. Falcone, D. Kalise, An efficient policy iteration algorithm for dynamic programming equations. *SIAM Journal on Scientific Computing,* 2015, 37(1), A181-A200.
7. D. Zhao, H. Wang, K. Shao, Y. Zhu, (, December). Deep reinforcement learning with experience replay based on SARSA. In: 2016 IEEE symposium series on computational intelligence (SSCI) 2016, pp. 1-6.
8. J.V. Rogushina, A.Y. Gladun, V.V. Osadchiy, S.M. Pryima Ontological analysis into the Web Melitopol: Bogdan Hmelnitsky MDPU, 2015, 407 p.[in Ukrainian],

Одержано: 25.10.2024
Внутрішня рецензія отримана: 02.11.2024
Зовнішня рецензія отримана: 05.11.2024

Про авторів:

Гришанова Ірина Юріївна,
науковий співробітник,
ORCID
http://orcid.org/0000-0003-4999-6294.

Рогушина Юлія Віталіївна,
к.ф.-м.н., доцент,
старший науковий співробітник,
ORCID
http://orcid.org/0000-0001-7958-2557.

Місце роботи авторів:

Інститут програмних систем
НАН України,
тел. (+38) 066 5501999,
e-mail:
ladamandraka2010@gmail.com