Одеський національний університет імені І. І. Мечникова
Факультет математики, фізики та інформаційних технологій
Кафедра оптимального керування та економічної кібернетики

Кваліфікаційна робота

на здобуття ступеня вищої освіти «магістр»

«Використання глибинного навчання у розв'язуванні NP-складних задач»

«Usage of Deep Learning for solving NP-hard problems»

Виконав: здобувач денної форми навчання
спеціальності 113 Прикладна математика
Освітня програма «Прикладна математика»
Пасенченко Томас Олексійович

Керівник: канд. фіз.-мат. наук, доц. Страхов Є. М. _____
Рецензент: канд. техн. наук, проф. Мороз В. В.

Рекомендовано до захисту:

Протокол засідання кафедри

№ ___ від _____ 2022 р.

Завідувач кафедри

_____ _____

Захищено на засіданні ЕК № _____

Протокол № ___ від _____ 2022 р.

Оцінка _____ / _____ / _____

Голова ЕК

_____ _____

Одеса — 2022 р.

ЗМІСТ

Вступ 4

1 Складність задач та основні NP-складні задачі 5
1.1 Основні класи обчислювальної складності 5
1.2 Задачі комбінаторної оптимізації 6
1.3 NP-повні задачі комбінаторної оптимізації на графах 7

2 Огляд методів навчання з підкріпленням 8
2.1 Основні поняття 8
2.1.1 Марковський процес вирішування 8
2.1.2 Рівняння оптимальності Беллмана 9
2.2 Класифікація методів навчання з підкріпленням 12
2.2.1 Методи без моделі 12
2.2.2 Методи, засновані на моделі середовища 14

3 Розв'язання NP-складних задач на графах за допомогою навчання з підкріпленням 17
3.1 Загальний підхід 18
3.1.1 Зведення задачі до MDP 18
3.1.2 Вибір графової нейронної мережі 19
3.1.3 Застосування алгоритму навчання з підкріпленням . 21
3.2 Методи 21
3.2.1 S2V-DQN 21
3.2.2 CompOpt Zero 23

4 Розв'язання задачі о мінімальном вершинном покритті за допомогою методу CombOpt Zero 27
4.0.1 Постановка задачі 27
4.0.2 Тестові дані 27
4.0.3 Реалізація методу 28
4.0.4 Процес тренування 29
4.0.5 Результати 30

Висновки 31

Список літератури 32

ВСТУП

Розв'язання NP-складних задач є ключовим етапом в оптимізації багатьох процесів у різних галузях науки та техніки. Наприклад, розв'язання задачі о знаходженні маршруту транспортного засобу може значно прискорити логістику, розв'язок задачі про мінімальне вершинне покриття - допомогти вирішити задачі генетики [27].

Свої застосування теорія графів та задачі з неї знаходять у хімії, фармації, фізиці, біології, урбаністиці та проектуванні життєво важливих інфраструктурних мереж [20]. Тому розробка методів ефективного розв'язання таких задач зараз є актуальним напрямком наукових досліджень.

Для NP-складних задач з теорії графів було розроблено багато різних методів розв'язання - починаючи з класичних апроксимаційних та еврістичних алгоритмів, завершуючи комерційними розв'язувачами.

З розвиненням машинного навчання, задачі на графах почали вирішувати його методами. Але вирішувалися лише окремі задачі, та загального підходу до розв'язання такого роду задач не існувало.

У 2017 році був запропонований загальний підхід до розв'язання класу NP-складних задач комбінаторної оптимізації на графах [7]. У 2019 році він був покращений та розширений ще на декілька задач [9]. Наразі, дослідження за даною темою продовжуються.

Метою цієї роботи є ознайомлення із сучасними методами розв'язання найбільш поширених NP-складних задач на графах та застосування одного з цих методів для розв'язання задачі про мінімальне вершинне покриття.

РОЗДІЛ 1

СКЛАДНІСТЬ ЗАДАЧ ТА ОСНОВНІ NP-СКЛАДНІ ЗАДАЧІ

1.1 Основні класи обчислювальної складності

Поняття складності алгоритму є базовим у теорії обчислюваності. Воно відображає те, скільки часу (чи пам'яті) знадобиться для розв'язання тієї чи іншої задачі. Усі алгоритмічні задачі можна віднести до певного класу складності. Класи складності визначаються через поняття *проблеми вибору*.

Проблема вибору - це задача на довільній множині вхідних даних, розв'язком якої є булеве значення «Так» чи «Ні».

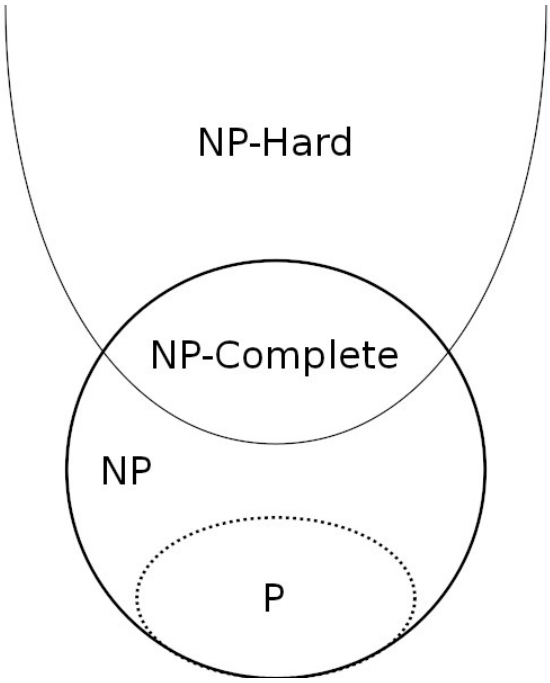

Рис. 1.1. Взаємозв'язок класів складності при припущенні, що $P \neq NP$

Основними класами складності є:

- **P** - множина проблем вибору, які можуть бути розв'язаними за поліноміальний час, тобто за час $O(n^k)$, де n - обсяг вхідних даних, k - деяка константа.

- **NP** - множина проблем вибору, які мають наступну властивість: Якщо відповідь - «Так», то існує доведення цього факту, яке може бути перевірене за поліноміальний час. Будь-яка проблема з класу P належить класу NP ($P \subseteq NP$). А чи виконується це включення в іншу сторону ($NP \subseteq P$), поки ще невідомо.
- **NP-повні (NPC)** - множина проблем вибору, які належать до класу NP та будь-яка проблема з NP може бути зведена до даної за поліноміальний час. Взагалі, NP-повні проблеми є найбільш складними у класі NP.
- **NP-складні** - множина проблем вибору, які не обов'язково належать до класу NP та будь-яка проблема з NP може бути зведена до даної за поліноміальний час. Будь-яка NP-повна проблема є NP-складною. NP-складна проблема, взагалі, може бути набагато складнішою, ніж проблеми з класу NP.

Для NP-складних задач досі не знайдено поліноміального алгоритму. Це означає, що час їх розв'язання може дуже сильно зростати при збільшенні обсягу вхідних даних.

1.2 Задачі комбінаторної оптимізації

Нехай задані множина елементів довільної природи V та функція $f : V \to \mathbb{R}$. Функцію f зазвичай називають функцією вартості.

Задача комбінаторної оптимізації полягає у знаходженні оптимального (наприклад, максимального чи мінімального) значення функції вартості f та відповідних елементів із множини V.

Уперше список NP-повних задач склав Річард Карп [4]. За допомогою теореми Кука [15] він довів, що NP-повна задача о виконуваності булевої формули (SAT) може бути зведена до однієї з 21 задач комбінаторної оптимізації (які, таким чином, теж є NP-повними).

Список Карпа включає декілька важливих задач на графах, зокрема задачу о пошуку Гамільтонова цикла, задачі про кліку та покриття кліки, задачі про вершинне покриття та максимальний розріз.

1.3 NP-повні задачі комбінаторної оптиміза-ції на графах

- Задача пошуку Гамільтонова шляха (шляха, який проходить через усі вершини графа рівно один раз)
- Задача пошуку Гамільтонова цикла. Є зуженням задачі пошуку Гамільтонова шляха.
- Задача комівояжера (TSP) - полягає у знаходженні найкоротшого замкненому маршруту між містами. Зводиться до пошуку Гамільтонова цикла с найменшою сумою ваг ребер.
- Задача о знаходженні маршруту транспортного засобу (VRP). Є узагальненням задачі комівояжера. Найчастіше розглядається з обмеженням на об'єм (CVRP) чи з обмеженням на об'єм та час (CVRP-TW).
- Задача пошуку кліки (множини суміжних вершин). Найчастіше розглядається задача про пошук кліки з максимальною кількістю вершин
- Задача пошуку максимальної незалежної множини (множини попарно несуміжних вершин). Цю задачу можна звести до задачи пошуку максимальної кліки.
- Задача пошуку максимального розрізу (множини ребер, видалення яких ділить граф на два ізольованих подграфа).
- Задача пошуку мінімального вершинного покриття (множини вершин, інцидентних усім ребрам графа).

РОЗДІЛ 2

ОГЛЯД МЕТОДІВ НАВЧАННЯ З ПІДКРІПЛЕННЯМ

Навчання з підкріпленням є різновидом машинного навчання. Ключовою його відмінністю є набуття «знань» через взаємодію із середовищем.

2.1 Основні поняття

2.1.1 Марковський процес вирішування

Для розв'язання задачі за допомогою методів навчання з підкріпленням, її потрібно представити у вигляді марковського процесу вирішування (MDP).

Марковський процес вирішування - це випадковий процес керування з дискретним часом, який визначається як четвірка (S, A, T, R):

- S - множина станів, у яких може знаходитися середовище
- A - множина дій, які може зробити агент.
- $T(s, a, s')$ - функція переходу зі стану s до стану s' за допомогою дії a. У задачах комбінаторної оптимізації ця функція є заданою.
- $R(s, a)$ - функція негайної винагороди за виконання дії a у стані s

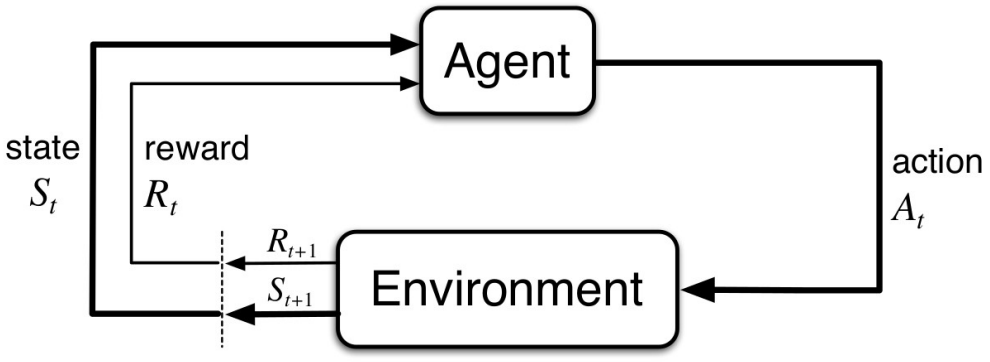

Рис. 2.1. Діаграма взаємодії агента із середовищем у форматі марковського процесу вирішування [2]

Часто до цих елементів додають фактор дисконту γ ($0 < \gamma \leq 1$), який мотивує агента орієнтуватися на винагороду у короткостроковій перспективі.

Нехай маємо вихідний стан середовища s_0. Послідовно обираючи дії $a \in A_s$ (де $A_s \subseteq A$ - множина можливих дій зі стану s) та застосовуючи функцію переходу $T(s, a)$, отримуємо послідовність станів та дій, яку називають траєкторією:

$$[s_0, a_0, s_1, a_1, \ldots, a_{H-1}, s_H]$$

Будемо розглядати лише скінченні траєкторії (якщо траєкторія є нескінченною, то дуже важко підрахувати кумулятивну винагороду). Тоді скінченне число H називається довжиною траєкторії чи горизонтом. s_H у такому випадку є термінальним (чи кінцевим) станом. Слід зазначити, що таких станів може бути декілька.

2.1.2 Рівняння оптимальності Беллмана

Поведінка агента, тобто відповідна дія у кожному стані, задається **функцією політики** $\pi(s) : S \to A$. Частіше за все вона є стохастичною, тобто представляє собою вектор ймовірностей для кожної дії у певному стані.

Головною задачею агента є максимізація кумулятивної (суммарної) винагороди. Цього можна досягнути лише вибором найбільш «вигідної» послідовності дій. Та оскільки вибір дій здійснюється за допомогою політики, то нам потрібно знайти оптимальну політику π^*:

$$\pi^* = \underset{\pi}{argmax}\, \mathbb{E}[\sum_{t=0}^{H} \gamma^t R(s_t, a_t)],$$

яка максимізує кумулятивну дисконтну винагороду.

При розробці багатьох алгоритмів буває корисним знати, настільки добре для агента знаходитися у певному стані s чи здійснювати дію a зі стану s. Для цього використовують функції стану-значення та дії-значення.

Функція стану-значення $V_\pi(s)$ відображає очікувану кумулятивну

винагороду, якщо агент починає зі стану s та після цього дотримується політики π:

$$V_\pi(s) = \mathop{\mathbb{E}}_{\pi}[\sum_{k=0}^{H} \gamma^k R_{t+k+1} \big| s_t = s], \forall s \in S$$

Функція дії-значення $Q_\pi(s, a)$ визначається як очікувана кумулятивна винагорода при умові, що агент, знаходячись у початковому стані s, виконає дію a, а потім буде дотримуватись політики π:

$$Q_\pi(s, a) = \mathop{\mathbb{E}}_{\pi}[\sum_{k=0}^{H} \gamma^k R_{t+k+1} \big| s_t = s, a_t = a]$$

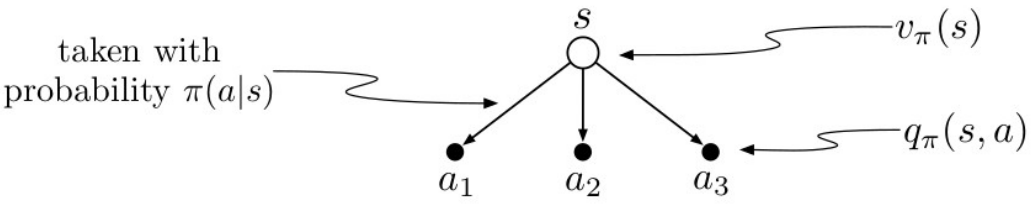

Рис. 2.2. Функції $V_\pi(s)$ та $Q_\pi(s, a)$ [2]

Нехай $\pi(a|s)$ - ймовірність того, що політика π обере наступною дію a, коли агент знаходиться у стані s. Очевидно, що $\sum_a \pi(a|s) = 1$, тобто із поточного (нетермінального) стану s завжди можна перейти у наступний.

Тоді можемо функцію стану-значення представити як суму добутків функцій дії-значення усіх можливих дій $a \in A_s$ у стані s на відповідні ймовірності обрання таких дій згідно політиці π у поточному стані:

$$V_\pi(s) = \sum_{a \in A_s} \pi(a|s) \times Q_\pi(s, a)$$

Оскільки функція стану-значення визначається через кумулятивну суму дисконтованих *майбутніх* винагород, можна представити її як суму негайної винагороди та дисконтованої функції стану-значення у наступних станах. Таке подання є рівнянням Беллмана для $V_\pi(s)$:

$$V_\pi(s) = \sum_{a \in A_s} \pi(a|s) \sum_{s'} P(s'|s,a))[R(s,a) + \gamma V_\pi(s')],\ \forall s \in S,$$

де s' - наступний стан, $R(s,a)$ - негайна винагорода при переході зі стану s у стан s' за допомогою дії a, $P(s'|s,a)$ - ймовірність переходу за допомогою дії a у стан s' зі стану s.

Враховуючи взаємозв'язок функцій $V_\pi(s)$ та $Q_\pi(s,a)$, можемо записати рівняння Беллмана для функції дії-значення:

$$Q_\pi(s,a) = \sum_{s'} P(s'|s,a)\left[R(s,a) + \gamma V_\pi(s')\right]$$

Для оптимальної політики π^* маємо відповідні оптимальні функції стану-значення $V^*(s)$ та дії-значення $Q^*(s,a)$:

$$V^*(s) = \max_\pi V_\pi(s),\ \forall s \in S$$
$$Q^*(s,a) = \max_\pi Q_\pi(s,a),\ \forall s \in S, \forall a \in A$$

Функцію $Q^*(s,a)$ можна записати за допомогою $V^*(s)$:

$$Q^*(s,a) = \mathbb{E}\left[r_{t+1} + \gamma V^*(s_{t+1}) \middle| s_t = s, a_t = a\right]$$

Враховуючи, що

$$V_\pi(s) = \max_a Q_\pi(s,a)$$

та

$$V^*(s) = \max_a Q^*(s,a),$$

можемо отримати **рівняння оптимальності Беллмана** для функцій $V^*(s)$ та $Q^*(s,a)$:

$$V^*(s) = \max_{a \in A_s} \sum_{s'} P(s'|s,a)[R(s,a) + \gamma V^*(s')]$$

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)[R(s,a) + \gamma \max_{a' \in A_{s'}} Q^*(s',a')]$$

2.2 Класифікація методів навчання з підкріпленням

Оскільки розв'язання задачі методами навчання з підкріпленням є результатом певної послідовності взаємодій агента із середовищем, то виникає питання моделювання цього середовища. Маючи модель середовища, можна побудувати більш точну політику та збільшити кумулятивну винагороду. Але такий підхід спрацьовує не у всіх випадках. Дійсно, якщо середовище є дуже складним та слабкопрогнозованим, то побудова його моделі може виявитися надто складною та взагалі непотрібною. Тому у навчанні з підкріпленням сформувалися два загальних класа методів - методи, засновані на моделі середовища, та методи без моделі.

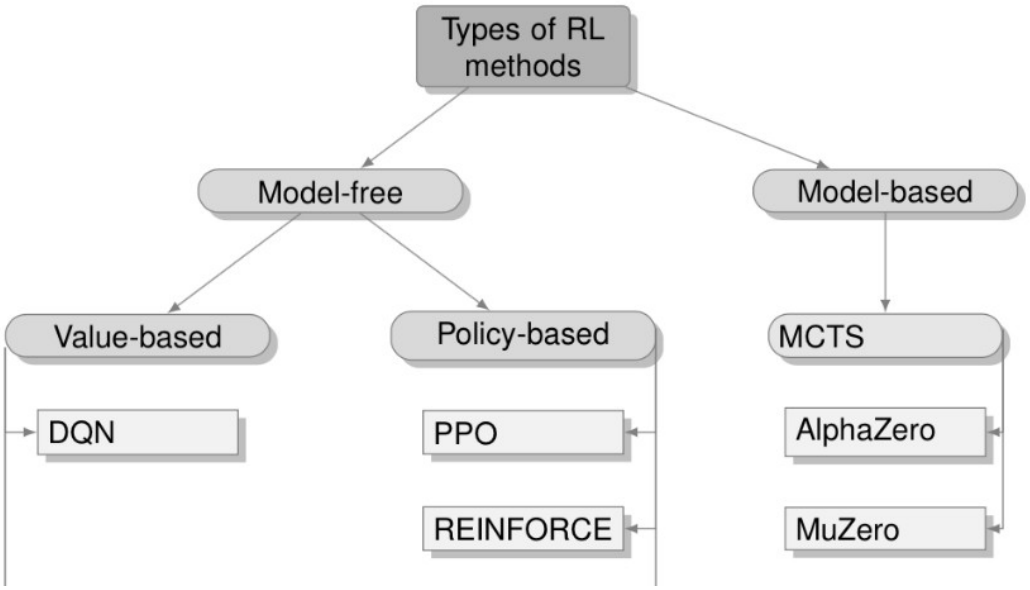

Рис. 2.3. Схема класифікації методів навчання з підкріпленням [16]

2.2.1 Методи без моделі

Методи без моделі будують розв'язок (тобто оптимальну політику), спираючись лише на досвід взаємодії агенту із середовищем. Вони навіть не

намагаються спрогнозувати чи смоделювати саме середовище. Такий підхід є дуже ефективним у складних чи недостатньо досліджених середовищах, але не може будувати досить довгострокові передбачення.

Такі методи можуть бути заснованими на наближенні значення (value-based) чи на наближенні політики (policy-based).

Методи, засновані на значенні

Ці методи засновані на наближенні функції дії-значення $Q_\pi(s, a)$ до оптимальної функції дії-значення $Q^*(s, a)$.

У методі Q-навчання [3] функція $Q(s, a)$ розраховується ітеративно за допомогою рівняння оптимальності Беллмана.

Функція $Q(s, a)$ представляється за допомогою Q-таблиці, де на перетині рядків (які відповідають станам середовища) та стовпчиків (які відповідають діям) знаходяться чисельні оцінки функції дії-значення для певних стану та дії.

Після кожної дії a_t, виконаної агентом у стані s_t, відбувається уточнення значення відповідної клітинки Q-таблиці за формулою:

$$Q^{\text{new}}(s_t, a_t) = Q^{\text{old}}(s_t, a_t) + \alpha \left[R_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right],$$

де R_t - негайна винагорода, отримана при переході зі стану s_t до стану s_{t+1} за допомогою дії a_t.

Метод Q-навчання є дуже простим та наочним, окрім цього є добрі теоретичні доведення сходимості $Q \to Q^*$ [3]. Але його застосування обмежено простими задачами з малою кількістю можливих станів та дій.

Щоб усунути обмеженість Q-навчання для складних задач, була розроблена нейронна мережа DQN [6] (Deep Q-network), яка аппроксимує функцію оптимального значення Q^*. DQN складається зі згорткових та повнозв'язних шарів. При її розробці було зроблено декілька оптимізацій, щоб усунути кореляцію між спостереженнями та покращити сходимість.

Методи, засновані на політиці

Щоб не наближати спочатку функцію оптимального значення Q^*, а потім оптимальну політику π^*, ми можемо одразу наближати оптимальну політику π^* за допомогою теореми о градієнті політики [2].

Також існує метод REINFORCE, заснований на модифікації цієї теореми, який використовує методи Монте-Карло для покращення сходимості.

2.2.2 Методи, засновані на моделі середовища

Якщо методи без моделі орієнтуються лише на досвід взаємодії із середовищем, методи, засновані на моделі середовища, використовують певну інформацію о середовищі. Це може бути наперед задана інформація (наприклад, функція переходу T(s, a, s')), а також інформація, отримана у процесі навчання (наприклад, ваги нейронної мережі, яка апроксимує середовище).

Побудована модель середовища дозволяє **планувати** дії у довгостроковій перспективі, що знайшло застосування у вирішенні багатокрокових ігор. Маючи модель середовища, можна обрахувати можливі наступні стани, подивитись, який виграш можна отримати, та побудувати послідовність кроків відповідним чином.

AlphaGo Zero

AlphaGo Zero [23] - це нейронна мережа, яка розроблена для гри в Go. Вона досягла надлюдської майстерності у цій грі без використання даних людських ігор.

Нехай s - стан ігрової дошки. Нейронна мережа f_θ приймає на вхід цей стан, та на виході має два значення: p - вектор ймовірностей для кожного кроку та $v \in [-1, 1]$ - значення поточного стану (чим ближче до 1, тим більша ймовірність виграти гру).

$$(p, v) = f_\theta(s)$$

Головна ідея алгоритму складається у тренуванні нейронних мереж за допомогою гри із самим собою. Нейронна мережа f_θ тренується таким чином, що p імітує політику π, покращену за допомогою пошуку по дереву Монте-Карло (MCTS), а v імітує фактичну винагороду z ($z = 1$, якщо гравець виграв і $z = -1$, якщо гравець програв). Іншими словами, мінімізується функція втрат:

$$L = (z - v)^2 + \text{CrossEntropy}(p, \pi) + c_{\text{reg}}\|\theta\|_2^2,$$

де c_{reg} - невід'ємна константа.

Пошук по дереву Монте-Карло (MCTS) - це еврістичний метод пошуку по великим деревоподібним масивам даних.

У AlphaGo Zero це дерево являє собою частину дерева гри Go, де вершини відповідають станам гри $s \in S$, а ребра - відповідним крокам (діям) $a \in A$. Кожне ребро (s, a) відповідає кроку a у стані s та додатково зберігає четвірку значень

- $N(s, a)$ - кількість відвідувань
- $W(s, a)$ - загальне значення дії-значення
- $Q(s, a)$ - середнє значення дії-значення
- $P(s, a)$ - попередня ймовірність

Ітерація алгоритму MCTS в AlphaGo Zero складається із трьох етапів:

1) *Вибір (select).* Починаючи від кореневого вузла, ми обираєм дію a, яка максимізує верхню межу довіри (UCB):

$$\text{UCB} = Q(s, a) + c_{\text{puct}} P(s, a) \frac{\sqrt{\sum_{a'} N(s, a')}}{1 + N(s, a)} \to \max,$$

де c_{puct} - невід'ємна константа.

2) *Розширення (expand).* Коли пошук досягає недослідженої вершини s, четвірка значень у ребрі ініціалізується за допомогою передбачення мережі $(p, v) = f_\theta(s)$.
3) *Зворотнє поширення (backup).* Після розширення нової вершини, кожне відвідане ребро обходиться у зворотньому порядку, а його значення оновлюються таким чином, щоб підтримувати актуальне значення

Q:

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{s'|s,a\to s'} v_{s'},$$

де підсумовування проводиться по станам s', досягнутим зі стану s після виконання дії a.

Також оновлюються значення $N(s,a)$ та $W(s,a)$.

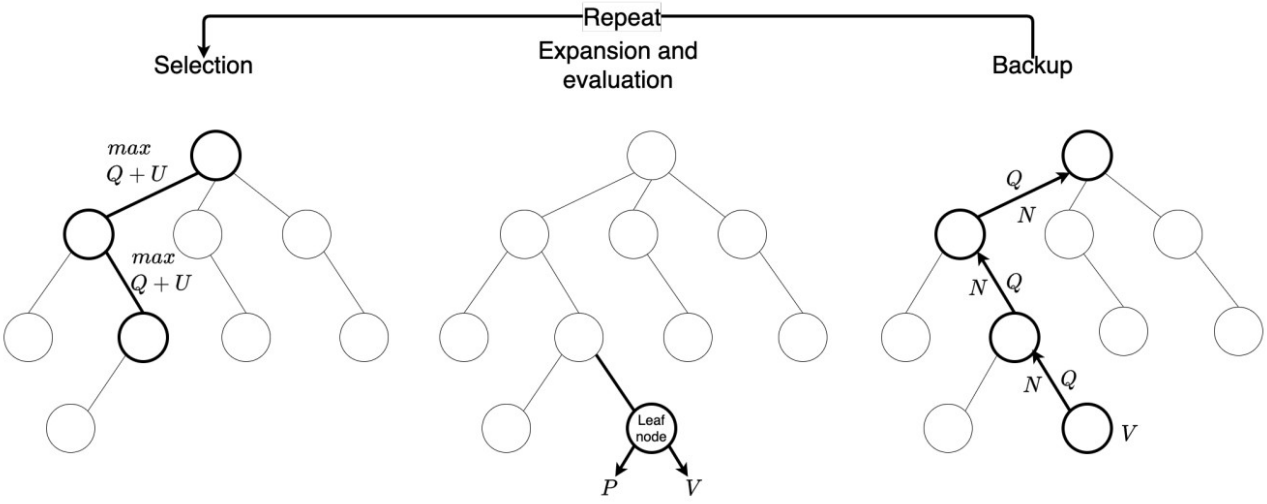

Рис. 2.4. Три етапи алгоритму MCTS [16]

Після декількох ітерацій, для $\forall a \in A_{s_0}$ обчислюється вектор ймовірностей π:

$$\pi_a = \frac{N(s_0,a)^{1/\tau}}{\sum_b N(s_0,b)^{1/\tau}},$$

де τ - параметр температури.

Алгоритм AlphaGo Zero пізніше був розширений до AlphaZero [24], який окрім Go підтримує шахи та сьоґі (японські шахи). У 2019 році був запропонований алгоритм MuZero [25], який не тільки здатен грати в настільні та комп'ютерні ігри, але й грає в ігри, не знаючи їхніх правил. Удосконалена версія MuZero - EfficientZero [26] побачила світ наприкінці 2021-го року.

РОЗДІЛ 3

РОЗВ'ЯЗАННЯ NP-СКЛАДНИХ ЗАДАЧ НА ГРАФАХ ЗА ДОПОМОГОЮ НАВЧАННЯ З ПІДКРІПЛЕННЯМ

Оскільки частіше за все буває необхідним розв'язувати лише одну певну задачу, то більшість розроблених методів та підходів зорієнтовані саме на це. Наприклад, у недавніх дослідженнях пропонується розв'язувати задачу коммівояжера (TSP) за допомогою графових нейронних мереж з покажчиками (GPN) [5] чи за допомогою структури encoding-decoding [8].

У 2017 році був запропонований принципово новий підхід - S2V-DQN [7], який дозволив вирішувати задачі на графах у загальному вигляді, не спираючись на специфіку розв'язання конкретних задач та не використовуючі жодних алгоритмів на графах. Головне - це коректно подати задачу у вигляді MDP.

Згодом, у 2019 році цей підхід був покращений. Фреймворк CombOpt Zero усунув недоліки S2V-DQN за допомогою використання алгоритму MCTS у варіації AlphaGo Zero, а також застосування різних графових нейронних мереж. Також CombOpt Zero був протестований на більшій кількості задач.

S2V-DQN та CombOpt Zero мають дві суттєві переваги:

- Навчання відбувається без жодних тренувальних даних. Алгоритми самостійно генерують тренувальні дані та навчаються на них. Це означає, що не потрібно збирати та балансувати датасети, а можна увесь час витратити на тренування та підбір гіперпараметрів.
- Навчання відбувається без інформації о конкретній вирішуваній задачі. Це позбавляє алгоритм від «ухилу» чи «упередженості» до конкретної задачі. Завдяки цьому можна легко адаптувати його до нової задачі (точніше, не сам алгоритм тренування, а його обгортку), а також працювати над самим алгоритмом у загальному вигляді.

3.1 Загальний підхід

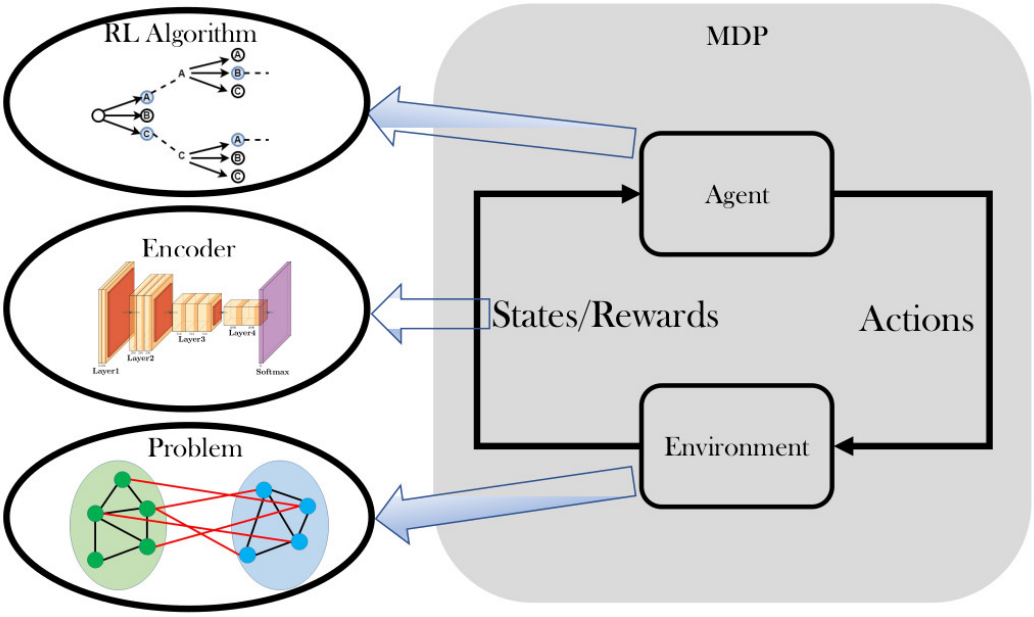

Рис. 3.1. Схема розв'язання задачі на графах за допомогою навчання з підкріпленням [16]

Алгоритм розв'язання задачі на графах за допомогою навчання з підкріпленням у загальному вигляді складається з наступних етапів:

3.1.1 Зведення задачі до MDP

Оскільки агент навчається у середовищі, побудованому за законами марковського процесу вирішування, то розв'язувану задачу необхідно звести до нього.

Це можна зробити, вказавши стандартні для будь-якого MDP множини станів S, термінальних станів S_{end} та дій A_s, функцію $T(s, a, s')$ переходу з попереднього стану s до наступного s' дією a, та $R(s, a)$ - функцію негайної винагороди.

Окрім цього, потрібно вказати параметри, притаманні задачам на графах, а саме:

- Init - функція, яка відображає вихідний граф G_0 у початковий стан s_0.

- $d : V \to L$ - функція розмітки графу, яка кожній вершині ставить у відповідність певну мітку з довільного простіру L. Ця функція може бути використана, наприклад, у задачі пошуку максимального розрізу.
- L - простір міток.

Ціллю розв'язання задачі, таким чином, є пошук траєкторії (чи, можливо, траєкторій) $[s_0, a_0, s_1, a_1, \ldots, a_{H-1}, s_H \in S_{\text{end}}]$, які забезпечать максимальну суму негайних винагород $R(s_i, a_i)$, $i = \overline{0, H-1}$, якщо $s_0 = \text{Init}(G_0)$:

$$r^*(\text{Init}(G_0)) = \max \sum_{i=0}^{H-1} R(s_i, a_i)$$

Наприклад, для задачі пошуку мінімального вершинного покриття можна обрати необхідні компоненти MDP наступним чином:

- S - стани відповідають графам
- S_{end} - множина пустих графів ($|V| = |E| = 0$)
- $A_s = V$ - діями є вибір однієї вершини
- T - перехід до наступного стану (тобто отримання графу, відповідного до наступного стану) відбувається за допомогою видалення певної вершини ($v = a$), ребер, інцидентних їй, та усіх ізольованих вершин
- $R(s, a) \equiv -1$ - оскільки ми хочемо отримати траєкторію з мінімальною довжиною
- Init - відповідає початковому графу
- $d(s) \equiv 1$, $L = \mathbb{R}$ - у цій задачі нам не потрібно розмічати граф

Тоді множина дій $a_0, a_1, \ldots, a_{H-1}$, максимізуюча r^*, буде відповідати множині вершин графа, що утворює мінімальне вершинне покриття цього графа.

3.1.2 Вибір графової нейронної мережі

Графова нейронна мережа (GNN) - це нейронна мережа, яка на вході приймає граф. Аналогічно тому, як в обробці зображень зручно використовувати згорткові нейронні мережі, GNN стають у нагоді при роботі з графами. Вони допомагають перетворити інформацію о структурі графу

на багатовимірний масив. Кожній вершині графу ставиться у відповідність вектор, який відображає її зв'язок із сусідніми вершинами. Також можливе опрацювання довільних додаткових властивостей вершини (наприклад, її «коліру» чи значення функції розмітки d).

S2V

Мережа S2V (structure2vec) [21] перетворює вихідних граф у масив репрезентацій.

Нехай маємо властивість вершини на вході $H^{(0)} \in \mathbb{R}^{n \times C_0}$, L - кількість шарів.

Властивості вершин поширюються за формулою

$$H_v^{(l+1)} = \mathrm{ReLU}(\theta_1 H_v^{(0)} + \theta_2 \sum_{u \in \mathrm{Neighbors}(v)} H_u^{(l)}),$$

де $\theta_1 \in \mathbb{R}^{p \times C_0}$, $\theta_2 \in \mathbb{R}^{p \times p}$, p - фіксоване ціле число.

В останньому шарі властивості усіх вершин поєднуються, щоб закодувати увесь граф:

$$y_v = \theta_3^T \times ReLU\left(\left[\theta_4 \sum_{u \in V} H_u^{(L)}, \theta_5 H_v^{(L)}\right]\right),$$

де $\theta_3 \in \mathbb{R}^{C_{\mathrm{out}} \times 2p}$, $\theta_4, \theta_5 \in \mathbb{R}^{p \times p}$, $[\cdot, \cdot]$ - оператор конкатенації.

GCN

Мережа GCN [22] за структурою схожа на згорткову нейронну мережу. Структура її шарів нагадує спектральні згортки на графах.

Нехай маємо властивість вершини на вході $H^{(0)} \in \mathbb{R}^{n \times C_0}$, L - кількість шарів.

Застосовується наступне правило поширення:

$$H^{(l+1)} = \sigma(\tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2} H^{(l)} \theta^{(l)}),$$

де $\tilde{A} = A + I_n$ - матриця суміжності графу A з доданими до кожної вершини петлями (I_n - одинична матриця розмірності $n \times n$), $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, $\theta^{(l)}$ - треновані ваги l-го шару, $\tilde{D}^{-1/2}$ - множники для нормалізації $\tilde{A}$.

3.1.3 Застосування алгоритму навчання з підкріпленням

До репрезентацій, отриманих за допомогою GNN, застосовується певний алгоритм навчання з підкріпленням. Це може бути модифікований алгоритм Q-навчання, як у методі S2V-DQN, чи адаптований варіант пошуку по дереву Монте-Карло, як в CombOpt Zero. Далі наведено більш детальніший опис кожного з цих методів.

3.2 Методи

3.2.1 S2V-DQN

S2V-DQN [7] - це метод для розв'язання широкого класу оптимізаційних задач на графах. Зокрема, в оригінальній публікації автори розглядають чотири задачі - про мінімальне вершинне покриття, максимальний розріз, покриття множини та задачу комівояжера.

Для кожної з вершин графа $v \in V$ за допомогою мережі S2V розраховуються p-вимірні вектори властивостей μ_v. При цьому також враховується поточне часткове рішення S (множина обраних на поточний момент вершин графа).

Вектори властивостей $\mu_v^{(t+1)}$ розраховуються таким чином, щоб упростити подальшу параметризацію функції $\hat{Q}(h(S), v; \Theta)$ ($h(S)$ - допоміжна функція, яка залежить від задачі), а саме:

$$\mu_v^{(t+1)} = \mathrm{ReLU}\left(\theta_1 x_v + \theta_2 \sum_{u \in \mathrm{Neighbors(v)}} \mu_u^{(t)} + \theta_3 \sum_{u \in \mathrm{Neighbors(v)}} ReLU\left(\theta_4 w(v,u)\right)\right),$$

де $\theta_1, \theta_4 \in \mathbb{R}^p$, $\theta_2, \theta_3 \in \mathbb{R}^{p \times p}$ - параметри моделі.

Після того, як вектори властивостей були підраховані за допомогою T ітерацій (зазвичай, для графів беруть мале T, наприклад $T = 4$), їх використовуються для визначення функції $\hat{Q}(h(S), v; \Theta)$:

$$\hat{Q}(h(S), v; \Theta) = \theta_5^T \mathrm{ReLU}\left(\left[\theta_6 \sum_{u \in V} \mu_u^{(T)}, \theta_7 \mu_v^{(T)}\right]\right),$$

де $\theta_5 \in \mathbb{R}^{2p}$, $\theta_6, \theta_7 \in \mathbb{R}^{p \times p}$ - параметри, $[\cdot, \cdot]$ - оператор конкатенації.

Функція $\hat{Q}(h(S), v; \Theta)$, таким чином, залежить від набору 7 параметрів $\Theta = \{\theta_i\}_{i=1}^7$. Для її навчання використовується комбінація n-крокового Q-навчання та підігнаної Q-ітерації.

n-крокове Q-навчання допомагає бачити винагороди у більш довгостроковій перспективі, що дозволяє будувати більш оптимальні траєкторії. Відбувається мінімізація квадратичної функції втрат:

$$L = \left(y - \hat{Q}(h(s_t), v_t; \Theta)\right)^2 \to \min,$$

де

$$y = \sum_{i=0}^{n-1} R(s_{t+i}, v_{t+i}) + \gamma \max_{v'} \hat{Q}(h(s_{t+n}), v'; \Theta)$$

Підігнана Q-ітерація оновлює Q-функцію за допомогою батчей з датасету E, який складається із даних з попередніх епізодів (послідовностей дій, які складають повний розв'язок задачі) тренування. Замість одиничних градієнтних кроків, оновлення здійснюється за допомогою стохастичного градієнтного спуску (SGD). Ця техніка має назву *повторення досвіду*. Вона дозволяє покращити сходимість навчання.

Функція $\hat{Q}$ є наближенням оптимальної функції Q^*. На основі $\hat{Q}$ визначається детерміністична жадібна політика:

$$\pi(v|S) = \underset{v' \in \overline{S}}{argmax}\, \hat{Q}(h(S), v')$$

3.2.2 CompOpt Zero

Метод CompOpt Zero [9] був розроблений на базі S2V-DQN. Його автори виявили, що S2V-DQN демонструє погані результати на деяких видах графів, відмінних від випадкових, зокрема на синтетичних графах та на графах з реальних задач. На їхню думку, така поведінка була спричинена обмеженістю простіру дослідження у методі Q-навчання.

Q-навчання було замінено на нову стратегію навчання, яка отримала назву CombOpt Zero. У експериментах вона показала значно кращу здатність до узагальнення на різноманітних графах.

Особливості адаптації алгоритму AlphaGo Zero для задач на графах

При адаптації алгоритму AlphaGo Zero для розв'язання задач комбінаторної оптимізації на графах, були враховані дві особливості:

- *Різні розмірності станів.* У грі Go стани представлені матрицею фіксованого розміру 19×19. У задачах на графах стани представлені графами довільного розміру. Ця особливість легко вирішується застосуванням графової нейронної мережі замість згорткових мереж у AlphaGo Zero.
- *Різні можливі значення станів.* Оскільки результат гри в Go може бути лише «виграв» чи «програв», значення стану легко змоделювати як «ймовірність виграшу» із діапазоном можливих значень $[-1, 1]$. Але відповідь на задачу комбінаторної оптимізації може приймати значення з довільного діапазону. Вона може зростати необмежено, в залежності від розміру графу. Для вирішення цієї проблеми була запропонована *техніка нормалізації винагороди.*

Техніка нормалізації винагороди

Техніка нормалізації винагороди призводить до наступних змін, порівняно з AlphaGo Zero:

- У прогнозах нейронної мережі $(p, v) = f_\theta$ замість скаляру $v \in [-1, 1]$

введено вектор v, у якому v_a прогнозує *нормалізовану винагороду* за виконання дії a зі стану s.

Нормалізована винагорода визначається за формулою:

$$v_a = \frac{R(s,a) + r^*(T(s,a)) - \mu_s}{\sigma_s},$$

де μ_s та σ_s - середнє значення та стандартне відхилення кумулятивної винагороди, отриманої за допомогою випадкових ігор зі стану s.

- При оцінці значення стану s, ми виконуємо дію a, яка максимізує v_a та відновлюємо вихідне ненормалізоване значення винагороди, використовуючи μ_s та σ_s:

$$r_{\text{estim}}(s) = \begin{cases} 0, & \text{якщо } s \in S_{\text{end}} \\ \mu_s + \sigma_s \times \max\limits_{a \in A_s} v_a, & \text{інакше} \end{cases}$$

- За аналогією, в ітераціях MCTS значення $W(s,a)$ та $Q(s,a)$ містять суму та середнє *нормалізованих оцінок* дії-значення.

  Відповідно, відбувається зміна етапу зворотнього поширення (backup) у MCTS: ми обчислюємо кумулятивну винагороду на батьківській вершині як і раніше: $r = r + R(s,a)$, але після цього ми знаходимо нормалізовану винагороду $r' = (r - \mu_s)/\sigma_s$ та додаємо до $W(s,a)$ вже цю нормалізовану винагороду $W(s,a) = W(s,a) + r'$. Середнє значення $Q(s,a)$ рахується вже за нормалізованим $W(s,a)$: $Q(s,a) = W(s,a)/N(s,a)$, а тому теж містить нормалізоване значення.

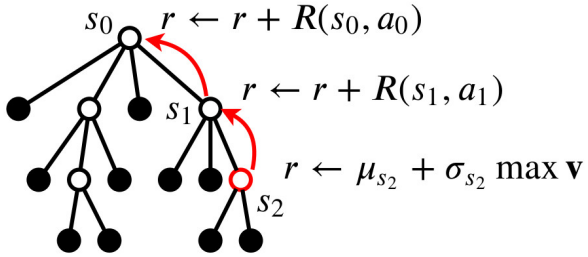

Рис. 3.2. Зміни в оновленні винагород на етапі backup MCTS [9]

- На кожному ребрі дерева MCTS окрім четвірки

$$N(s,a), W(s,a), Q(s,a), P(s,a)$$

зберігається пара значень (μ_s, σ_s) - середнє та стандартне відхилення результатів випадкових ігор.

Слід зазначити, що техніка нормалізації винагород звільняє алгоритм від специфіки задачи, тобто можна вирішувати задачі із різними критеріями, не змінюючи алгоритм тренування MCTS.

Тренування моделей

Подібно до AlphaGo Zero, тренування CombOpt Zero складається із трьох ролей:

- *Генератори даних (data generators)* безперервно генерують записи ігор з самими собою на випадкових графах, використовуючи MCTS з найкращою на даний момент моделлю.
  Записи ігор є послідовностями (s, a, π, z') (дія a була зроблена зі стану s згідно з покращеною за допомогою MCTS політикою π, та $z' = (z - \mu_R(s))/\sigma_R(s)$, де z - кумулятивна винагорода зі стану s до термінального стану).
- *Учні (learners)* вибирають випадкові міні-батчі з даних, створених генератором, та оновлюють параметри найкращої моделі так, щоб мінімізувати функцію втрат

$$L = (z' - v_a)^2 + CrossEntropy\,(p, \pi) + c_{\text{reg}}\|\theta\|_2^2 \to \min\,,$$

  де v_a - елемент вектору v, який відповідає дії a, z' - нормалізована кумулятивна винагорода.
- *Оцінювачі моделей (model evaluators)* на згенерованих тестових випадкових графах порівнюють середні результати оновленої учнем моделі та найкращої з наявних моделей. Якщо оновлена модель виявилася кращою за найкращу відому, то оновлена модель записується замість найкращої. Якщо ні, то нічого не відбувається.

Слід зазначити, що найкраща з наявних моделей використовується одночасно усіма ролями: генераторами - для створення нових записів ігор через MCTS, учнями - для спроб покращити цю модель, оцінювачами - для порівняння нової моделі із найкращою.

Також представників кожної з ролей може бути декілька (тому ролі й названі у множині). Це реалізується за допомогою декількох процесорів чи відеокарт, що дозволяє розпаралелити та прискорити обчислення.

Застосування

Даний метод був адаптований для задач про мінімальне вершинне покриття, максимальний розріз, максимальну кліку, максимальну незалежну множину та розрізаюча цикли множина вершин.

Були апробовані різні графові нейронні мережі, такі як S2V, GCN, GIN та 2-IGN+. Також були проведені різні порівняльні експерименти на випадкових та реальних графах.

У більшості, отримані результати значно випередили головного конкурента - метод S2V-DQN.

РОЗДІЛ 4

РОЗВ'ЯЗАННЯ ЗАДАЧІ О МІНІМАЛЬНОМ ВЕРШИННОМ ПОКРИТТІ ЗА ДОПОМОГОЮ МЕТОДУ COMBORT ZERO

4.0.1 Постановка задачі

Нехай маємо заданий неорієнтований незважений граф $G = (V, E)$, де V - множина вершин, E - множина ребер.

Вершинним покриттям $V' \subseteq V$ називається така підмножина множини ребер, що усі ребра графа є інцидентними хоча б одній вершині з V'. Тобто для усіх ребер $\forall(x, y) \in E$ виконується хоча б один з виразів $x \in V'$, $y \in V'$.

Задача о мінімального вершинного покриття полягає у пошуку вершинного покриття з мінімальною кількістю вершин $|V'| \to \min$.

4.0.2 Тестові дані

У якості тестових даних було обрано 4 графа з наукового репозиторію мережевих даних Network Data Repository [20]:

- **email-enron-only** - взаємозв'язок email-ів вищого керівництва корпорації Enron. Вершини відображать менеджерів, а ребра - їх контакт за допомогою email.
- **inf-USAir97** - зв'язок аеропортів США, між якими компанією US Air було здійснено авіарейси у 1997 році. Вершини відображають аеропорти, ребра - рейси.
- **road-chesapeake** - зв'язок доріг у місті Чесапік (США). Дороги відображаються ребрами, а перехрестя - вершинами.
- **road-euroroad** - зв'язок європейських міст за допомогою автомагістралей. Вершини графа - міста, ребра - магістралі.

У таблиці 4.1 наведені основні характеристики тестових графів.

Назва графу	$\|V\|$	$\|E\|$	Середній ступінь вершини
email-enron-only	143	623	8
inf-USAir97	332	2126	12
road-chesapeake	39	170	8
road-euroroad	1174	1417	2

Табл. 4.1. Основні характеристики тестових графів

4.0.3 Реалізація методу

Метод CombOpt Zero був реалізований за допомогою фреймворку для автоматичного дифференціювання PyTorch [17]. Для роботи із графами був використаний пакет NetworkX [18].

У якості тренувальних та тестових даних використовувалися випадкові графи Ердеша - Реньї [19] $G(n, p)$ (граф з числом вершин n, де кожне ребро додається з ймовірністю p незалежно від інших ребер). Параметри $n_{\min} \leq n \leq n_{\max}$ та p вказані у таблиці 4.2.

	Набір 1	Набір 2
$n_{\min}$	15	20
$n_{\max}$	20	30
p	0.5	0.6
GNN	S2V	GCN

Табл. 4.2. Набори гіперпараметрів

Розмір батча всюди використовувався 16.

З першим набором гіперпараметрів використовувалася графова нейронна мережа S2V з параметрами $p = 64$, $L = 5$.

З другим набором була використана більш нова мережа GCN з параметрами $L = 5$, hidden_dim $= 32$.

Інші гіперпараметри були встановлені згідно з рекомендаціями із

оригінальної наукової публікації [9].

4.0.4 Процес тренування

Тренування на обох наборах гіперпараметрів проводилося упродовж однієї години.

Вибір найкращої моделі здійснювався за допомогою порівняння їхніх результатів на наборі випадкових графів з такими параметрами, які використовувалися при тренуванні. Оскільки у кожному з наборів відрізняються параметри $n_{\max}$, $n_{\min}$ та p, то масштаб графіків також дещо відрізняється. Але обидва графіки демонструють поступове зменшення сумарного розміру вершинних покриттів. Це свідчить про те, що тренування є ефективним.

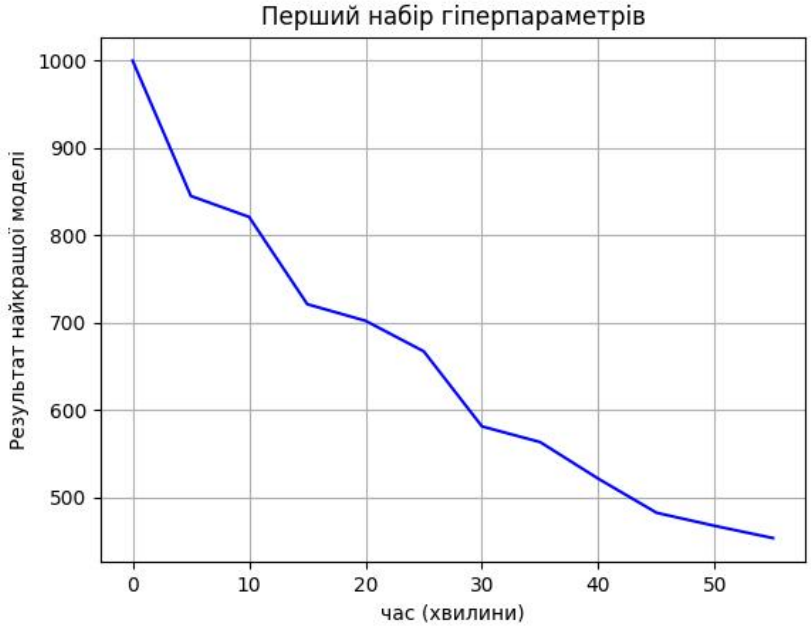

Рис. 4.1. Процес тренування на першому наборі гіперпараметрів

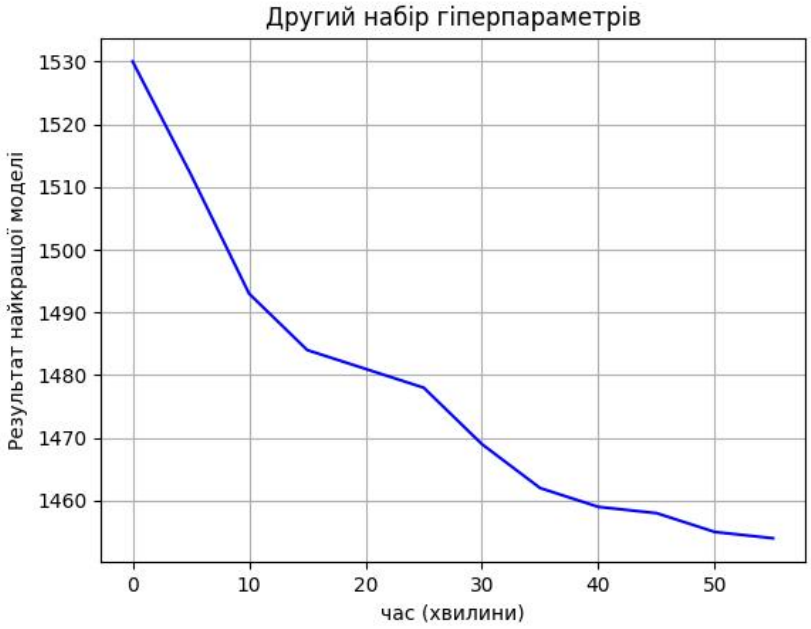

Рис. 4.2. Процес тренування на другому наборі гіперпараметрів

4.0.5 Результати

Після тренування, моделі були протестовані на кожному з графів. Результати (розміри отриманих вершинних покриттів V') наведені у таблиці 4.3.

Назва графу	Набір гіперпараметрів 1	Набір гіперпараметрів 2
email-enron-only	99	86
inf-USAir97	149	149
road-chesapeake	23	22
road-euroroad	273	272

Табл. 4.3. Результати на чотирьох графах

З отриманих результатів можна побачити, що тренування на першому наборі гіперпараметрів виявилося менш ефективним, ніж тренування на другому. Причиною тому може бути слабкість S2V у порівнянні з GCN, а також розмір тренувальних випадкових даних - чим більше за розміром тренувальний граф, тим краще мережа вчиться розпізнавати взаємозв'язки та будувати кращі траєкторії.

ВИСНОВКИ

У даній роботі були розглянуті основні класи обчислювальної складності задач та наведений перелік поширених NP-складних задач на графах.

Розглянуто основні поняття навчання з підкріпленням та наведена базова класифікація його методів.

Були описані та порівняні сучасні методи та підходи для розв'язання широкого класу NP-повних оптимізаційних задач на графах.

В останьому розділі цієї роботи за допомогою новітнього методу CombOpt Zero [9] на прикладі чотирьох графів з наукового репозиторію мережевих даних Network Data Repository [20] була розв'язана задача про мінімальне вершинне покриття. Отримані результати були проаналізовані.

СПИСОК ЛІТЕРАТУРИ

1. T. H. Cormen, C. E. Leiserson et al. Introduction to Algorithms. MIT Press, 2009.
2. R. S. Sutton and A. G. Barto. Reinforcement Learning, second edition: An Introduction. MIT Press, 2008.
3. C. Watkins. Q-Learning. Kluwer Academic Publishers, Boston, Machine Learning, 8, pp. 279-292, 1992.
4. R. M. Karp. Reducibility Among Combinatorial Problems. In R. E. Miller; J. W. Thatcher; J.D. Bohlinger (eds.). Complexity of Computer Computations. New York: Plenum. pp. 85–103, 1972.
5. Q. Ma et al. Combinatorial Optimization by Graph Pointer Networks and Hierarchical Reinforcement Learning. arXiv preprint 1911.04936, 2019.
6. V. Mnih et al. Human-level control through deep reinforcement learning. In Nature 518, 529–533, 2015. https://doi.org/10.1038/nature14236
7. Hanjun Dai et al. Learning Combinatorial Optimization Algorithms over Graphs. NIPS proceeding, 2017.
8. Christian Lowens Solving the Traveling Salesperson Problem with Precedence Constraints by Deep Reinforcement Learning. arXiv preprint 2207.01443, 2022.
9. K. Abe, Z. Xu et al. Solving NP-Hard Problems on Graphs with Extended AlphaGo Zero. arXiv preprint 1905.11623v2, 2019.
10. N. Mingshuo, C. Dongming and W. Dongqi. Reinforcement learning on graphs: A survey. arXiv preprint arXiv:2204.06127, 2022.
11. T. D. Barrett et al. Learning to Solve Combinatorial Graph Partitioning Problems via Efficient Exploration. arXiv preprint arXiv:2205.14105, 2022.
12. H. Khadilkar. Solving the capacitated vehicle routing problem with timing windows using rollouts and MAX-SAT. arXiv preprint arXiv:2206.06618, 2022.
13. A. I. Garmendia et al. Neural Combinatorial Optimization: a New Player in the Field. arXiv preprint arXiv:2205.01356, 2022.
14. L. Ardon. Reinforcement Learning to Solve NP-hard Problems: an Appli-

cation to the CVRP. arXiv preprint arXiv:2201.05393, 2022.

15. S. Cook. The complexity of theorem-proving procedures. Proceedings of the third annual ACM symposium on Theory of computing (pp. 151-158), 1971.
16. N. Mazyavkina et al. Reinforcement Learning for Combinatorial Optimization: A Survey. arXiv preprint arXiv:2003.03600v3, 2020.
17. A. Paszke et al. Pytorch: An imperative style, high-performance deep learning library. arXiv preprint arXiv:1912.01703, 2019.
18. A. Hagberg et al. Exploring network structure, dynamics, and function using NetworkX. Proceedings of the 7th Python in Science Conference (SciPy2008), pp. 11–15, 2008.
19. P. Erdos. On random graphs. Publicationes mathematicae, 6:290–297, 1959.
20. R. Rossi and N. Ahmed. The Network Data Repository with Interactive Graph Analytics and Visualization. AAAI, 2015.
21. H. Dai et al. Discriminative embeddings of latent variable models for structured data. In International Conference on Machine Learning, 2016.
22. T. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In International Conference on Learning Representations, 2017.
23. D. Silver et al. Mastering the game of go without human knowledge. Nature, 550(7676):354, 2017.
24. D. Silver et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. arXiv preprint arXiv:1712.01815, 2017.
25. J. Schrittwieser et al. Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. arXiv preprint arXiv:1911.08265, 2019.
26. W. Ye et al. Mastering Atari Games with Limited Data. arXiv preprint arXiv:2111.00210, 2021.
27. G. Lancia et al. SNPs Problems, Complexity and Algorithms. ESA 2002, LNCS 2161, pp. 182-193, 2001.