УДК 004.4’24, 004.89, 004.942

І.П. Сініцин, А.Ю. Дорошенко, Т.А. Мамедов, О.А. Яценко

МЕТОД АВТОМАТИЗОВАНОГО ПРОЄКТУВАННЯ НЕЙРОЕВОЛЮЦІЙНИХ АЛГОРИТМІВ З ВИКОРИСТАННЯМ АЛГЕБРИ АЛГОРИТМІВ ГЛУШКОВА

Сініцин Ігор Петрович

Інститут програмних систем НАН України, м. Київ,
https://orcid.org/0000-0002-4120-0784

ips2014@ukr.net

Дорошенко Анатолій Юхимович

Інститут програмних систем НАН України, м. Київ,
https://orcid.org/0000-0002-8435-1451

doroshenkoanatoliy2@gmail.com

Мамедов Турал Алірзайович

Інститут програмних систем НАН України, м. Київ,
https://orcid.org/0000-0003-3029-5834

tural.mamedov1@gmail.com

Яценко Олена Анатоліївна

Інститут програмних систем НАН України, м. Київ,
https://orcid.org/0000-0002-4700-6704

oayat@ukr.net

Академік В.М. Глушков був зачинателем багатьох напрямків наукових досліджень, зокрема і напрямку автоматизації проєктування алгоритмів, а розроблена ним концепція алгебри алгоритмів стала основою для багатьох розробок інструментальних засобів у цій галузі. Автори пропонують налаштування раніше створеного алгебро-алгоритмічного інструментарію на автоматизоване проєктування та синтез програм, що використовують нейроеволюційні алгоритми. Нейроеволюція є сукупністю методів машинного навчання, що застосовують еволюційні алгоритми для полегшення вирішення складних завдань, що імітують процес природного відбору. Метод нейроеволюції наростаючих топологій (NEAT) призначений для зменшення розмірності простору пошуку параметрів нейромережі у вигляді поступового розвитку її структури у процесі еволюції. Процес починається з популяції дрібних, найпростіших геномів і поступово збільшує їхню складність з кожним новим поколінням. Кінцевим результатом нейроеволюції є оптимальна топологія мережі, яка робить модель більш енергоефективною та зручною для аналізу. Розроблений інструментарій проєктування програм забезпечує автоматизацію побудови високорівневих специфікацій алгоритмів, поданих у системах алгоритмічних алгебр Глушкова, та синтез відповідних програм на основі шаблонів реалізацій цільовою мовою програмування. Налаштування інструментарію на проєктування нейроеволюційних алгоритмів полягає у внесенні в базу даних системи описів відповідних елементарних операторів і предикатів та їх програмних реалізацій. Використання інструментарію проілюстроване на прикладі проєктування та генерації програми для задачі балансування зі зворотним маятником, яка застосовує нейроеволюційний алгоритм бібліотеки NEAT-Python. Задача

полягає в тому, щоб керувати модельованим візком, який може рухатися лише в двох напрямках за допомогою жердини, прикріпленої шарніром до його верхньої частини. Наведено результати експерименту з виконання програми, згенерованої за допомогою алгебро-алгоритмічного інструментарію.

Ключові слова: автоматизоване проєктування, алгебра алгоритмів, генерація програм, задача балансування, нейроеволюція наростаючих топологій, нейронна мережа.

Вступ

Штучні нейронні мережі використовуються в різних напрямках застосувань обчислювального інтелекту. Нейромережі, що мають невелику кількість нейронів, через свої обмежені апроксимаційні здатності не дозволяють вирішувати реальні практичні завдання [1], а вибір надлишкової кількості нейронів у мережі призводить до проблеми перенавчання та втрати апроксимаційних властивостей, збільшення кількості необхідних апаратних ресурсів для її реалізації та збільшення часових затримок на обробку інформації та навчання нейронних мереж.

Традиційно структура нейромереж визначається вручну експертом-розробником обчислювальних систем, що використовують ці мережі. Оптимізація структури нейронної мережі проводиться видаленням ряду малозначних вагових коефіцієнтів, використовуючи алгоритми послідовного зменшення або збільшення структури та методи еволюційної оптимізації (нейроеволюція). Останні використовуються для вирішення різних завдань синтезу нейромереж, а саме: відбору ознак, настроювання ваг, вибору оптимальної архітектури мережі, адаптації навчальних правил та ініціалізації значень вагових коефіцієнтів.

Метод нейроеволюції став дуже поширеним в останні роки [2–4]. Відомі лабораторії штучного інтелекту та дослідники експериментують з ним, низка нових успіхів підсилює ентузіазм, і з'являються нові можливості впливу на глибоке навчання, що було і залишається популярним в останнє десятиліття. Метод нейроеволюції — це метод автоматизації створення нейромереж. На відміну від ручних методів, він у процесі проєктування дозволяє змінювати одночасно і вагові коефіцієнти зв'язків між вузлами і топологію вузлів. Одною з найвідоміших реалізацій нейроеволюційних алгоритмів є нейроеволюція наростаючої топології [4–6] (NeuroEvolution of Augmenting Topologies — NEAT). Нейроеволюція наростаючої топології становить собою форму машинного навчання з підкріпленням, яка використовує еволюційні (генетичні) алгоритми для генерації штучних нейронних мереж, їхніх параметрів, топологій і правил прийняття рішень. Головна перевага нейроеволюції полягає в можливості її ширшого застосування порівняно з навчанням з учителем, що вимагає розмічених коректних пар вхідних та вихідних даних для тренування. На відміну від цього, нейроеволюція потребує лише можливостей оцінити ефективність згенерованої мережі на будь-якому етапі навчання.

У даній роботі запропоновані засоби автоматизації проєктування та генерації програм, що використовують нейроеволюційні алгоритми. Засоби ґрунтуються на високорівневих специфікаціях програм, в основу яких покладено системи алгоритмічних алгебр Глушкова (САА) [7, 8]. На основі схем, поданих в САА, здійснюється автоматизована генерація коду мовою програмування. Підхід продемонстровано на задачі балансування візка зі зворотним маятником. Виконані проєктування та генерація програми, що використовує відкриту бібліотеку реалізації нейроеволюційних алгоритмів NEAT-Python [9].

1. Нейроеволюція наростаючих топологій

Нейроеволюція — це сімейство методів машинного навчання, які використовують еволюційні алгоритми для полегшення вирішення складних завдань, таких

як ігри, робототехніка та моделювання природних процесів [6]. Нейроеволюційні алгоритми імітують процес природного відбору. Дуже прості штучні нейронні мережі можуть стати занадто складними. Кінцевим результатом нейроеволюції є оптимальна топологія мережі, яка робить модель більш енергоефективною та зручною для аналізу. Метод NEAT призначений для зменшення розмірності простору пошуку параметрів у вигляді поступового розвитку структури нейромережі у процесі еволюції. Еволюційний процес починається з популяції дрібних, найпростіших геномів і поступово ускладнює їх з кожним новим поколінням.

Розглянемо суть підходу NEAT. На початку можемо мати дуже просту топологію, що складається тільки з вхідних $(x_0, x_1, \ldots)$ та вихідних вузлів (a), а також (опціонально) прихованого шару вузлів (h), якого може і не бути. Далі працює генетичний алгоритм, який шляхом схрещування та мутацій автоматично продукує інші топології, що оцінюються на основі функції придатності (fitness), з яких відбираються найкращі для продовження генерації поколінь. При цьому якість розпізнавання покращується за рахунок нарощування кількості прихованих шарів нейромережі. Якість можна контролювати, спостерігаючи за графіком залежності функції придатності від кількості створених поколінь. Як показує досвід, в результаті створюється найкраща за якістю нейромережа, яка до того ж має переваги компактності та швидкодії.

Однією з популярних тестових задач, яка ефективно вирішується за допомогою NEAT, є балансуванням жердини [2, 6, 10]. Задача полягає в тому, щоб керувати симульованим візком, який може рухатися лише в двох напрямках за допомогою жердини, прикріпленої до його вершини шарніром (зворотний маятник). Чим довше візок (керований нейронною мережею) може утримувати жердину в повітрі, тим вища його придатність. Ця задача дуже схожа на спробу збалансувати олівець на долоні — вона вимагає чіткої координації та швидкої реакції.

Програмні реалізації алгоритму нейроеволюції представлені низкою бібліотек, зокрема: NEAT-Python [9], SharpNEAT [11], PyTorch-NEAT [12], MultiNEAT [13], NEAT Java [14]. У даній роботі проведено експеримент з автоматизації розробки алгоритму нейроеволюції, реалізований із використанням бібліотеки NEAT-Python [9].

NEAT-Python є реалізацією алгоритму NEAT мовою програмування Python. Бібліотека NEAT-Python забезпечує реалізацію стандартних методів NEAT для моделювання генетичної еволюції геномів організмів в популяції. Вона містить утиліти для перетворення генотипу організму в його фенотип (штучну нейромережу) і надає зручні методи для завантаження та збереження конфігурації геному разом з параметрами NEAT. Крім того, пропонує дослідникам корисні підпрограми, що допомагають зібрати статистику про хід еволюційного процесу і зберегти/завантажити проміжні контрольні точки. Контрольні точки дозволяють періодично зберігати стан еволюційного процесу і згодом поновлювати виконання процесу зі збережених контрольних точок. До переваг бібліотеки NEAT-Python відносяться:

- стабільна реалізація, всебічна документація;
- доступність для легкої установки за допомогою менеджера пакетів PIP;
- наявність вбудованих інструментів збирання статистики та підтримки збереження контрольних точок виконання, а також відновлення виконання із заданої контрольної точки;
- наявність кількох типів функцій активації, підтримка фенотипів рекурентних нейромереж неперервного часу;
- легка розширюваність для підтримки різних модифікацій NEAT.

Бібліотека NEAT-Python використовує набір гіперпараметрів, що впливають на виконання та точність алгоритмів NEAT (файл конфігурації зберігається у форматі, аналогічному файлам .ini Windows). Параметри включають, зокрема:

• fitness_criterion — функцію, яка обчислює критерій завершення з набору значень придатності всіх геномів у популяції;

• fitness_threshold — граничне значення, яке порівнюється з придатністю, обчисленою за допомогою функції fitness_criterion для перевірки необхідності завершення еволюції;

• pop_size — кількість окремих організмів у кожному поколінні;

• початкову конфігурацію мережі за кількістю прихованих num_hidden, вхідних num_inputs та вихідних вузлів num_outputs.

2. Засоби автоматизованого проєктування алгоритмів та програм

Для автоматизації розробки програм, що використовують нейроеволюційні алгоритми, у даній роботі мають місце мова САА/1 та алгебро-алгоритмічний інструментарій проєктування алгоритмів та програм [7, 8, 15, 16]. Дана мова призначена для багаторівневого структурного проєктування й документування послідовних і паралельних алгоритмів та програм. Її математичним базисом є САА Глушкова.

Основними операторними конструкціями мови САА/1, що використовуються в даній роботі, є такі:

• композиція — послідовне виконання операторів: «operator 1»; «operator 2»;

• альтернатива (умовний оператор): IF ‘condition’ THEN «operator 1» ELSE «operator 2»;

• цикл: FOR EACH («variable 1» IN «expression 1») LOOP «operator 1» END OF LOOP.

Процес автоматизованої розробки алгоритмів та програм в інструментарії зображено на рис. 1. Він включає такі компоненти:

• конструктор високорівневих схем алгоритмів, представлених в САА (САА-схем);

• базу даних, що містить опис конструкцій САА та параметризовані шаблони реалізації цих конструкцій цільовими мовами програмування (C++, Java, Python);

• генератор текстів програм на основі схем алгоритмів із використанням описів з бази даних.

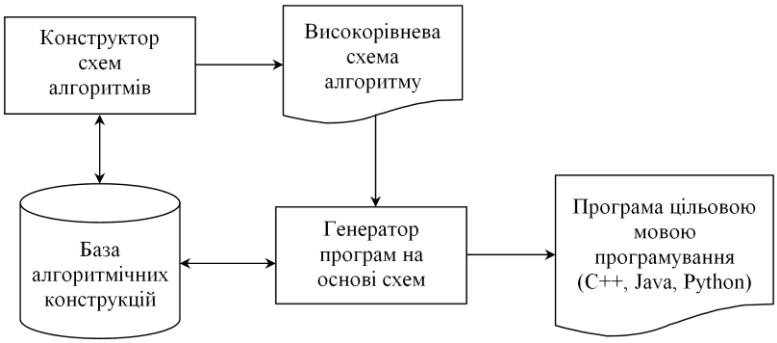

Рис. 1

У процесі роботи з конструктором користувач може редагувати опис операторних і предикатних мовних конструкцій, а також базисних операторів та умов, які зберігаються в базі даних інструментарію. Опис елемента бази даних (операції САА або базисного поняття) включає його подання в аналітичній та природно-лінгвістичній формі, а також реалізацію мовою програмування.

Налаштування інструментарію на проєктування нейроеволюційних алгоритмів, що є метою даної роботи, полягає у внесенні в базу даних описів відповідних елементарних операторів та предикатів. У табл. 1 подані приклади опису базисних елементів для алгоритмів нейроеволюції. Ідентифікатори операторів обрамлені лапками, предикатів — апострофами. Реалізація мовою програмування використовує методи бібліотеки NEAT-Python.

Таблиця

№	**Природно-лінгвістична форма**	**Реалізація мовою Python з використанням методів бібліотеки NEAT-Python**
1	«Activate the NET (net)(input)»	%1.activate(%2)
2	«Load configuration (config) from file (file)»	%1 = neat.Config(neat.DefaultGenome, neat.DefaultReproduction, neat.DefaultSpeciesSet, neat.DefaultStagnation, %2)
3	«Create the population (p), which is the top-level object for a NEAT run, for configuration (config)»	%1 = neat.Population(%2)
4	«Run neuroevolution for up to (n) generations and display the best genome (best_genome) among generations»	%2 = p.run(eval_genomes, %1=%1_generations) print('\nBest genome:\n{!s}'.format(%2))
5	«Set (genome) fitness to (value)»	%1.fitness = %2
6	«Adjust (fitness) based on (additional_num_runs) and (success_runs)»	%1 = 1.0 - (%2 - %3) / %2
7	«Create feedforward neural network (net) for (genome) and configuration (config)»	%1 = neat.nn.FeedForwardNetwork. create(%2, %3)
8	«Evaluate fitness of the genome that was used to generate net (net)»	fitness = cart.eval_fitness(%1)
9	'Genome (fitness) is greater or equal to fitness threshold (fitness_threshold) for configuration (config)'	%1 >= %3.%2
10	'Genome (fitness) is less than fitness threshold (fitness_threshold) for configuration (config)'	%1 < %3.%2

Оператори включають:

• активацію нейромережі net на основі масиву вхідних даних input;

• завантаження конфігурації експерименту з файлу;

• створення популяції p на основі конфігурації;

• запуск процесу нейроеволюції для n популяцій та відображення найкращого геному;

• установку значення придатності геному (fitness);

• корегування значення придатності на основі кількості додаткових запусків моделювання additional_num_runs та успішних запусків success_runs;

• створення нейромережі net для вказаного геному та конфігурації;

• оцінку придатності геному, що використовувався для генерації мережі.

Предикати (номери 9, 10) призначені для перевірки значення придатності з пороговим значенням, вказаним у файлі конфігурації.

Природно-лінгвістична форма опису базисного елемента містить імена формальних параметрів, вказані у дужках. Фактичні параметри, які задаються в САА-схемах, заміняють при синтезі програми відповідні формальні параметри в тексті реалізації базисного поняття мовою програмування. Параметри в реалізації мовою програмування позначені номерами %1, %2 і т.д.

Приклад проєктування програми, що використовує бібліотеку NEAT-Python, розглядається нижче.

3. Застосування інтегрованого інструментарію для відомої прикладної задачі з використанням відкритого програмного забезпечення NEAT-Python

Для ілюстрації розглянемо використання інтегрованого інструментарію для відомої прикладної задачі балансування візка зі зворотним маятником [6, 10]. Згадана задача є класичним прикладом експерименту з навчання з підкріпленням, який також є загальноприйнятим еталоном для тестування різних реалізацій стратегій управління. У даному експерименті алгоритм NEAT застосовується для реалізації контролера, що керує фізичним пристроєм (візком). Розглянуто застосування апарату САА та інструментарію проєктування та генерації програм для реалізації симулятора пристрою з візком та алгоритму керування, що ґрунтується на нейроеволюції, для стабілізації зворотного маятника протягом заданого проміжку часу. Для реалізації нейроеволюційного алгоритму використовується бібліотека NEAT-Python [9].

Постановка задачі балансування зворотного маятника та математична модель (рівняння руху) детально розглянуті в [6]. Контролер балансування маятника приймає масштабовані вхідні значення і видає вихідний сигнал, що є двійковим значенням і визначає дію, яка має застосовуватися в певний момент часу.

Початкова конфігурація нейромережі контролера представлена на рис. 2. Вона включає п’ять вхідних вузлів: для горизонтального положення візка (x1) і його швидкості (x2), для вертикального кута маятника (x3) і його кутової швидкості (x4) і додатковий вхідний вузол для зміщення (x0) (яке може бути необов’язковим залежно від конкретної використовуваної бібліотеки NEAT). Вихідний вузол (a) є двійковим вузлом, що видає керуючий сигнал (0 або 1). Прихований вузол (h) є необов’язковим і може бути пропущений.

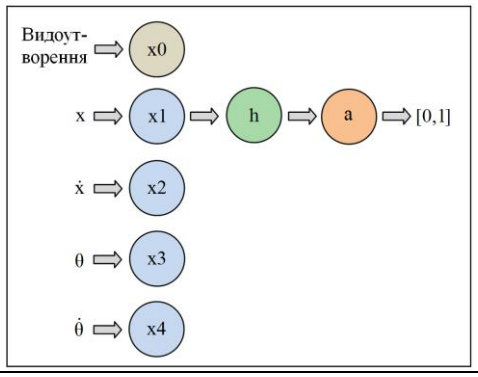

Рис. 2

Текст САА-схеми CART POLE, що містить опис рівнянь руху та функцію для оцінки придатності балансувальника одиночного маятника, наведено далі. Схема сконструйована за допомогою інтегрованого інструментарію. Вона містить чотири складені оператори: «Cart pole», «Perform the one step of simulation», «Run cart-pole simulation» та «Evaluate fitness of the genome that was used to generate net», що відповідають підпрограмам (функціям) у реалізації мовою програмування. Деталізація складених операторів наведена після ланцюжків символів «====», у дужках — наведені параметри базисних та складених операторів.

```
SCHEME CART POLE ====


«Cart pole»

==== «Import library (math)»; «Import library (random)»;
      «Declare a constant (GRAVITY) = (9.8)»;
      «Declare a constant (MASSCART) = (1.0)»;
```

```
        «Declare a constant (MASSPOLE) = (0.5)»;
        «Declare a constant (TOTAL_MASS) = ((MASSPOLE + MASSCART))»;
        «Declare a constant (LENGTH) = (0.5)»;
        «Declare a constant (POLEMASS_LENGTH) = ((MASSPOLE * LENGTH))»;
        «Declare a constant (FORCE_MAG) = (10.0)»;
        «Declare a constant (FOURTHIRDS) = (4.0/3.0)»;
        «Declare a constant (TAU) = (0.02)»;
        «Set random seed (42)»;

«Perform the one step of simulation (action, x, x_dot, theta, theta_dot)»
==== «Comment (Find the force direction)»;
        IF (action <= 0)
        THEN «(force) := (-FORCE_MAG)»
        ELSE «(force) := (FORCE_MAG)»
        END IF;
        «Comment (Pre-calcuate cosine and sine to optimize performance)»;
        «(cos_theta) := (math.cos(theta))»; «(sin_theta) := (math.sin(theta))»;
        «(temp) := ((force + POLEMASS_LENGTH * theta_dot * theta_dot *
                    sin_theta) / TOTAL_MASS)»;
        «Comment (The angular acceleration of the pole)»;
        «(theta_acc) := ((GRAVITY * sin_theta - cos_theta * temp) /
          (LENGTH * (FOURTHIRDS - MASSPOLE * cos_theta * cos_theta /
         TOTAL_MASS)))»;
       «Comment (The linear acceleration of the cart)»;
       «(x_acc) := (temp - POLEMASS_LENGTH * theta_acc * cos_theta /
       TOTAL_MASS)»;
      «Comment (Update the four state variables, using Euler's method)»;
      «(x_ret) := (x + TAU * x_dot)»;
      «(x_dot_ret) := (x_dot + TAU * x_acc)»;
      «(theta_ret ) := (theta + TAU * theta_dot)»;
      «(theta_dot_ret) := (theta_dot + TAU * theta_acc)»;
      «Return value (x_ret, x_dot_ret, theta_ret, theta_dot_ret)»

«Run cart-pole simulation (net, max_bal_steps, random_start=True)»
==== «Comment (Set random initial state if appropriate)»;
        «(x, x_dot, theta, theta_dot) := (0.0, 0.0, 0.0, 0.0)»;
        IF random_start
        THEN «(x) := ((random.random() * 4.8 - 2.4) / 2.0)»;
                «(x_dot) := ((random.random() * 3 - 1.5) / 4.0)»;
                «(theta) := ((random.random() * 0.42 - 0.21) / 2.0)»;
                «(theta_dot) := ((random.random() * 4 - 2) / 4.0)»
        END IF;
        «Comment (Run simulation for specified number of steps while
                  cart-pole system stays within contstraints)»;
        «(input) := ([None] * 4)»;
        FOR EACH (steps IN range(max_bal_steps))
        LOOP
          «Comment (Load scaled inputs)»;
          «(input[0]) := ((x + 2.4) / 4.8)»;  «(input[1]) := ((x_dot + 1.5) / 3)»;
          «(input[2]) := ((theta + 0.21) / .42)»; «(input[3]) := ((theta_dot + 2.0) / 4.0)»;
          (output := «Activate the NET (net)(input)»);
          IF (output[0] < 0.5)
         THEN «(action) := (0)»
         ELSE «(action) := (1)»
         END IF;
         (x, x_dot, theta, theta_dot := «Perform the one step of simulation
         (action = action, x = x, x_dot = x_dot, theta = theta,
            theta_dot = theta_dot)»);
         «Comment (Check for failure due constraints violation. If so, return
          number of steps.)»;
         IF (x < -2.4) OR (x > 2.4) OR (theta < -0.21) OR (theta > 0.21)
         THEN «Return value (steps)»
         END IF
         END OF LOOP;
         «Return value (max_bal_steps)»

«Evaluate fitness of the genome that was used to generate net (net,
  max_bal_steps=500000)»
==== (steps := «Run cart-pole simulation (net, max_bal_steps)»);
        IF (steps = max_bal_steps)
```

```
THEN «Return value (1.0)»
ELSE IF (steps = 0)
        THEN «Return value (0.0)»
        ELSE
           (log_steps := log(steps));
           (log_max_steps := log(max_bal_steps));
           «(error) := ((log_max_steps - log_steps) / log_max_steps)»;
           «Return value (1.0 - error)»
        END IF
 END IF

END OF SCHEME CART POLE
```

Схема починається з визначення використовуваних бібліотек та констант, що описують фізику системи з візком. Ці визначення подані у вигляді складеного оператора «Cart pole». Далі реалізуються рівняння руху з використанням вказаних констант у складеному операторі «Perform the one step of simulation (action, x, x_dot, theta, theta_dot)». Як вхідні дані використовується поточний стан системи (x, x_dot, theta, theta_dot) разом з керуючою дією. Даний оператор застосовує рівняння руху для оновлення стану системи перед наступним кроком. Оновлений стан системи повертається, щоб оновити симулятор і перевірити порушення обмежень. У цілому отримуємо цикл моделювання, що розглядається далі.

У циклі моделювання використовується нейромережа контролера, щоб оцінити поточний стан системи та вибрати відповідну дію (зусилля, яке буде додане до візка) для наступного кроку. Згадана раніше нейромережа створюється для кожного геному популяції на певному поколінні еволюції, що дозволяє оцінювати ефективність усіх геномів.

Повний цикл моделювання складається з таких кроків:

• ініціалізація змінних початкового стану (x, x_dot, theta, theta_dot) випадковими значеннями в рамках обмежень;

• цикл моделювання через певну кількість кроків, які задаються параметром max_bal_steps;

• змінні стану масштабуються таким чином, щоб відповідати діапазону [0, 1], перш ніж завантажувати їх як вхідні дані в нейромережу контролера. Результат масштабування записується в комірки масиву input[0], …, input[3];

• масштабовані вхідні дані можна використовувати для активації нейромережі, а вихідні дані нейромережі — для отримання дискретного значення дії;

• отримавши значення дії і поточні значення змінних стану, можна запустити один крок моделювання візка з маятником. Після кроку моделювання змінні стану, що були повернуті, перевіряються на відповідність обмеженням, щоб перевірити, чи знаходиться стан системи в допустимих межах.

Якщо нейромережа контролера змогла підтримувати стабільний стан балансування системи візок–маятник для всіх кроків моделювання, функція «Run cart-pole simulation» повертає значення з максимальною кількістю кроків моделювання.

Оцінка придатності геному виконується у функції «Evaluate fitness of the genome that was used to generate net».

САА-схема, що реалізує експеримент, наведена нижче. Вона починається з визначення використовуваних бібліотек та констант. Ці визначення подані у вигляді складеного оператора «GlobalData», є також складений оператор main. Функція оцінки придатності всіх геномів у популяції міститься в складеному операторі «Evaluate best net». Вона отримує список усіх геномів у популяції і параметри конфігурації NEAT і для кожного генома створює фенотип нейромережі, використовуючи його як контролера для запуску моделювання.

```
SCHEME SINGLE POLE EXPERIMENT ====

«GlobalData»
==== «Import library (os)»; «Import library (neat)»;
       «Import library (visualize)»; «Import library (cart_pole as cart)»;
       «Import library (utils)»;
       «Set current working directory (local_dir)»;
       «Set the directory to store outputs (out_dir)(local_dir)(out)»;
       «Set the directory to store outputs (out_dir)(out_dir)(single_pole)»;
       (additional_num_runs := 100);
       (additional_steps := 200)

«main»
==== «Determine path (config_path) to configuration file (local_dir)
        (single_pole_config.ini)»;
       «Clean results of previous run if any or init the ouput directory (out_dir)»;
       «Run experiment (config_path)»;

«Run experiment (config_file, n_generations=100)»
==== «Load configuration (config) from file (config_file)»;
       «Create the population (p), which is the top-level object for a NEAT run,
        for configuration (config)»;
       «Add a stdout reporter to show progress in the terminal  for population (p)»;
       «Run neuroevolution for up to (n) generations and display the best
        genome (best_genome) among generations»;
       «Create feedforward neural network (net) for (best_genome) and
        configuration (config)»;
       «Output the message (\n\nEvaluating the best genome in random runs)»;
       (success_runs := «Evaluate best net (net, config, additional_num_runs)»);
       «Output successful (success_runs) and expected runs
        (additional_num_runs) and check if the best genome is a winner»;
       «Visualize the experiment results for configuration (config), best genome
        (best_genome) from directory (out_dir)»;

«Evaluate genomes (genomes, config)»
==== FOR EACH (genome_id, genome IN genomes)
       LOOP
         «Set (genome) fitness to (0.0)»;
         «Create feedforward neural network (net) for (genome) and
          configuration (config)»;
         «Evaluate fitness of the genome that was used to generate net (net)»;
         IF 'Genome (fitness) is greater or equal to fitness threshold (fitness_threshold)
                for configuration (config)'
         THEN
           (success_runs := «Evaluate best net (net, config, additional_num_runs)»);
           «Adjust (fitness) based on (additional_num_runs) and (success_runs)»
         END IF;
         «Set (genome) fitness to (fitness)»
       END OF LOOP;

«Evaluate best net (net, config, num_runs)»
==== FOR EACH (run IN range(num_runs))
       LOOP
         «Evaluate fitness of the genome that was used to generate net (net,
          max_bal_steps=additional_steps)»
       END OF LOOP;
       IF 'Genome (fitness) is less than fitness threshold (fitness_threshold) for
            configuration (config)'
       THEN «Return value (run)»
       END IF;
       «Return value (num_runs)»

END OF SCHEME SINGLE POLE EXPERIMENT
```

Функція виконання експерименту знаходиться в складеному операторі «Run experiment». Вона починається із завантаження гіперпараметрів з файлу конфігурації і породжує початкову популяцію, використовуючи завантажену конфігурацію. Після цього функція налаштовує репо́ртери для збирання статистики, що стосується виконання еволюційного процесу. Також додаються вихідні репо́ртери

для виведення результатів виконання на консоль у режимі реального часу. Процес еволюції виконується протягом вказаної кількості поколінь, а результати зберігаються у вихідному каталозі. Після того як в ході еволюційного процесу знайдено найкращий геном, перевіряється, чи дійсно він відповідає критеріям порогу придатності, встановленому у файлі конфігурації. Програма повертає геном з формально найкращою відповідністю.

4. Результати експерименту

На основі сконструйованих САА-схем за допомогою інтегрованого інструментарію виконано генерацію програмного коду мовою Python. Далі наведено результати експерименту з виконання згенерованої програми. У файлі конфігурації NEAT-Python визначено популяцію з 150 окремих організмів та встановлено поріг придатності зі значенням 1.0 як критерій завершення. Значення початкових параметрів нейромережі є такими: num_hidden = 0, num_inputs = 4, num_outputs = 1.

При виконанні програми виводяться дані для кожного покоління еволюції (рис. 3). Найкращий геном, який є переможцем еволюції, кодує фенотип нейромережі, який складається лише з одного нелінійного вузла (вихід) і трьох з'єднань із вхідних вузлів (size: (1, 3)). Граф нейромережі контролера-переможця балансування одиночного зворотного маятника наведений на рис. 4.

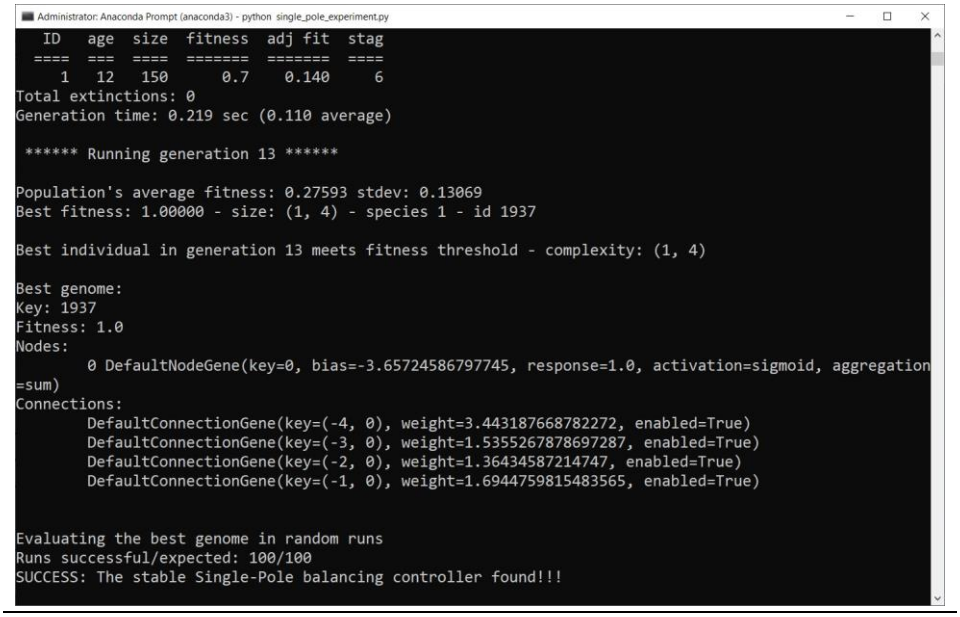

Рис. 3

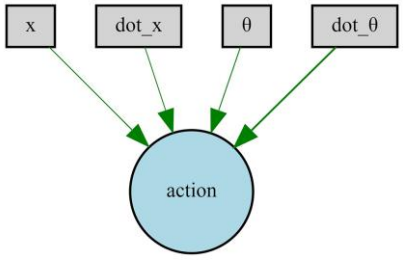

Рис. 4

Графік зміни значень придатності упродовж еволюції показано на рис. 5. Середня придатність популяції у всіх поколіннях була низькою, але з самого початку виникла корисна мутація, що породила певну лінію організмів. З покоління в по-

коління обдаровані особини з цієї лінії змогли не лише зберегти свої корисні ознаки, а й покращити їх, що зрештою виявило переможця еволюції.

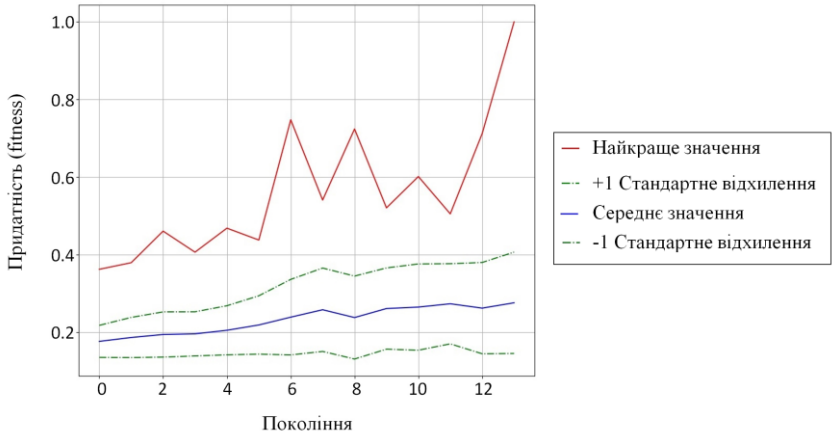

Рис. 5

Висновок

У роботі запропоновано метод автоматизованого проєктування нейроеволюційних алгоритмів на основі розширення та налаштування алгебро-алгоритмічного інструментарію. В основу інструментарію покладено конструювання високорівневих специфікацій алгоритмів, поданих у системах алгоритмічних алгебр Глушкова, та генерацію відповідних програм на основі шаблонів реалізацій цільовою мовою програмування. Підхід проілюстровано на прикладі проєктування програми для задачі балансування зі зворотним маятником, яка використовує нейроеволюційний алгоритм бібліотеки NEAT-Python.

I. Sinitsyn, A. Doroshenko, T. Mamedov, O. Yatsenko

A METHOD OF AUTOMATED DESIGN OF NEUROEVOLUTION ALGORITHMS BASED ON GLUSHKOV ALGEBRA OF ALGORITHMS

Igor Sinitsyn

Institute of Software Systems of the National Academy of Sciences of Ukraine, Kyiv,

ips2014@ukr.net

Anatoliy Doroshenko

Institute of Software Systems of the National Academy of Sciences of Ukraine, Kyiv,

doroshenkoanatoliy2@gmail.com

Tural Mamedov

Institute of Software Systems of the National Academy of Sciences of Ukraine, Kyiv,

tural.mamedov1@gmail.com

Olena Yatsenko

Institute of Software Systems of the National Academy of Sciences of Ukraine, Kyiv,

oayat@ukr.net

Academician V.M. Glushkov was the initiator of many directions of scientific research in his time, in particular, the direction of algorithm design automation, and the concept of algebra of algorithms proposed by him became the basis for many tools developed in this field. In this paper, authors propose the adjustment of the previously developed algebra-algorithmic toolkit towards the automated design and synthesis

 ISSN 2786-6491

of programs using neuroevolutionary algorithms. Neuroevolution is a set of machine learning techniques that apply evolutionary algorithms to facilitate the solving of complex tasks that imitate the process of natural selection. The method of neuroevolution of augmenting topologies (NEAT) is intended to reduce the dimensionality of the space for searching for neural network parameters in the form of gradual development of the neural network structure in the process of evolution. The evolutionary process begins with a population of small, the simplest genomes and gradually increases their complexity with each new generation. The final result of neuroevolution is the optimal topology of the network, which makes the model more energy efficient and convenient for analysis. The developed program design toolkit provides automated construction of high-level algorithm specifications represented in Glushkov's system of algorithmic algebra and synthesis of corresponding programs based on implementation templates in a target programming language. The adjustment of the toolkit for designing neuroevolutionary algorithms consists in entering the descriptions and software implementations of the relevant elementary operators and predicates into a database of the toolkit. The use of the toolkit is illustrated by an example of designing and generating a program for the single pole balancing problem, which uses the neuroevolutionary algorithm of the NEAT-Python library. The challenge is to control a simulated cart that can only move in two directions using a pole hinged to its top. The results of the experiment on the execution of the program generated using the algebraic-algorithmic toolkit are presented.

Keywords: automated design, algebra of algorithms, program generation, pole-balancing problem, neuroevolution of augmenting topologies, neural network.

ПОСИЛАННЯ

1. Subbotin S.O., Oliinyk A.O., Oliinyk O.O. Non-iterative, evolutionary and multi-agent methods for synthesis of fuzzy and neural network models. Zaporizhzhia : ZNTU, 2009. 375 p. (in Ukrainian).
2. Stanley K.O., Clune J., Lehman J., Miikkulainen R. Designing neural networks through neuroevolution. *Nature Machine Intelligence*. 2019. Vol. 1. P. 24–35. http://doi.org/10.1038/s42256-018-0006-z.
3. Stanley K.O. Neuroevolution: a different kind of deep learning. 2017. http://www.oreilly.com/radar/neuroevolution-a-different-kind-of-deep-learning.
4. Doroshenko A.Y., Achour I.Z., Yatsenko O.A. Parameter-driven generation of evaluation program for a neuroevolution algorithm on a binary multiplexer example. *Radio Electronics, Computer Science, Control.* 2023. N 1. P. 80–88. http://doi.org/10.15588/1607-3274-2023-1-8.
5. Stanley K.O., Miikkulainen R. Evolving neural networks through augmenting topologies. *Evolutionary Computation*. 2002. Vol. 10, N 2. P. 99–127. http://doi.org/10.1162/106365602320169811.
6. Omelianenko I. Hands-on neuroevolution with Python. Birmingham : Packt, 2019. 368 p.
7. Doroshenko A., Yatsenko O. Formal and adaptive methods for automation of parallel programs construction: emerging research and opportunities. Hershey : IGI Global, 2021. 279 p. http://doi.org/10.4018/978-1-5225-9384-3.
8. Andon P.I., Doroshenko A.Yu., Zhereb K.A., Yatsenko O.A. Algebra-algorithmic models and methods of parallel programming. Kyiv : Akademperiodyka, 2018. 192 p. http://doi.org/10.15407/akademperiodyka.367.192.
9. NEAT-Python. URL: http://github.com/CodeReclaimers/neat-python.
10. Chang O., Kwiatkowski R., Chen S., Lipson H. Agent embeddings: a latent representation for pole-balancing networks. *Proc. International Conference on Autonomous Agents and Multiagent Systems (AAMAS'19) (13–17 May 2019, Montreal, Canada).* Richland : International Foundation for Autonomous Agents and Multiagent Systems, 2019. P. 656–664. http://doi.org/10.48550/arXiv.1811.04516.
11. SharpNEAT. Evolution of Neural Networks. http://sharpneat.sourceforge.io.
12. PyTorch-NEAT. URL: http://github.com/uber-research/PyTorch-NEAT.
13. MultiNEAT: Portable NeuroEvolution Library. http://github.com/peter-ch/MultiNEAT.
14. NEAT Java (JNEAT). http://nn.cs.utexas.edu/soft-view.php?SoftID=5.
15. Doroshenko A., Shymkovych V., Yatsenko O., Mamedov T. Automated software design for FPGAs on an example of developing a genetic algorithm. *Proc. 17th International Conference «ICT in Education, Research and Industrial Applications. Integration, Harmonization and Knowledge Transfer» (ICTERI 2021) (28 September – 2 October 2021, Kherson, Ukraine).* Cham : Springer, 2021. P. 74–85. https://ceur-ws.org/Vol-3013/20210074.pdf.
16. Doroshenko A., Zhereb K., Yatsenko O. Using algebra-algorithmic and term rewriting tools for developing efficient parallel programs. *Proc. 9th International Conference «ICT in Education, Research and Industrial Applications: Integration, Harmonization and Knowledge Transfer» (ICTERI 2013) (19–22 June 2013, Kherson, Ukraine).* Cham : Springer, 2013. P. 38–46. https://ceur-ws.org/Vol-1000/ICTERI-2013-p-038-046.pdf.

Отримано 25.05.2023