УДК 004.4’24, 004.896 https://doi.org/10.15407/pp2022.03-04.301

АВТОМАТИЗОВАНА ГЕНЕРАЦІЯ ПРОГРАМ ДЛЯ ОДНОГО КЛАСУ ПАРАМЕТРИЧНИХ АЛГОРИТМІВ НЕЙРОЕВОЛЮЦІЇ

Анатолій Дорошенко, Ілля Ашур

Застосовано апарат алгебри гіперсхем для автоматизованої генерації алгоритмів нейроеволюції на прикладі задачі оцінювання для двійкового мультиплексора, яка входить до складу системи SharpNEAT. Згадана система є фреймворком з відкритим програмним кодом мовою C#, що реалізує генетичний алгоритм нейроеволюції для платформи .NET. У статті показано, як за допомогою апарату алгебр алгоритмів та гіперсхем можна автоматизувати генерацію програм оцінювання для задач нейроеволюції. Гіперсхема є високорівневою параметризованою специфікацією алгоритму для розв’язання певного класу задач. Установка значень параметрів і подальша інтерпретація гіперсхеми дозволяє отримати алгоритми, адаптовані до конкретних умов застосування. Автоматизоване конструювання гіперсхем та генерація алгоритмів на їх основі виконується в розробленому інтегрованому інструментарії проєктування й синтезу програм. Проведено експеримент із виконання згенерованої програми для задачі оцінювання двійкового мультиплексора на розподіленій хмарній платформі.

Ключові слова: автоматизоване проєктування програм, алгебра алгоритмів, гіперсхема, нейроеволюція, нейромережа, паралельні та розподілені обчислення, хмарні обчислення.

The facilities of algebra of hyperschemes are applied for automated generation of neuroevolution algorithms on an example of a binary multiplexer evaluation problem, which is a part of the SharpNEAT system. SharpNEAT is an open-source framework developed in C# programming language, which implements a genetic neuroevolution algorithm for the .NET platform. Neuroevolution is a form of artificial intelligence, which uses evolution algorithms for creating neural networks, parameters, topology, and rules. Evolution algorithms apply mutation, recombination, and selection mechanisms for finding neural networks with behavior that satisfies to conditions of some formally defined problem. In this paper, we demonstrate the use of algebra of algorithms and hyperschemes for the automated generation of evaluation programs for neuroevolution problems. Hyperscheme is a high-level parameterized specification of an algorithm for solving some class of problems. Setting the values of the hyperscheme parameters and further interpretation of a hyperscheme allows obtaining algorithms adapted to specific conditions of their use. Automated construction of hyperschemes and generation of algorithms based on them is implemented in the developed integrated toolkit for design and synthesis of programs. The design of algorithms is based on Glushkov systems of algorithmic algebra. The schemes are built using a dialogue constructor of syntactically correct programs, which consists in descending design of algorithms by detailing the constructions of algorithmic language. The design is represented as an algorithm tree. Based on algorithm schemes, programs in a target programming language are generated. The results of the experiment consisting in executing the generated binary multiplexer evaluating program on a cloud platform are given.

Keywords: automated program design, algebra of algorithms, hyperscheme, neuroevolution, neural network, parallel and distributed computing, cloud computing.

Вступ

Нейроеволюція є перспективним підходом для розв’язання складних задач машинного навчання, створення штучних нейронних мереж, адаптивного керування, багатоагентних систем, еволюційної робототехніки [1]. Основна перевага нейроеволюції полягає в тому, що вона може використовуватися ширше, ніж алгоритми навчання з учителем, для яких необхідна програма правильних пар введення-виведення. Нейроеволюція вимагає лише оцінювання продуктивності мережі при виконанні завдання. Вона використовує еволюційні алгоритми для навчання нейромережі й належить до категорії методів навчання з підкріпленням. Всі еволюційні алгоритми розвивають набір (“популяцію”) рішень (“індивідуумів”). Індивіди представлені своїм генотипом, який виражається у формі фенотипу, з яким пов’язують якість, “пристосованість”. Існує велика кількість нейроеволюційних алгоритмів, що поділяються на дві групи. До першої належать алгоритми, які здійснюють еволюцію вагів при заданій топології мережі, до другої – алгоритми, що окрім еволюції вагів також здійснюють еволюцію топології мережі. Еволюційні алгоритми маніпулюють множиною генотипів, які є поданням нейромережі. У схемі з прямим кодуванням генотип еквівалентний фенотипу, нейрони і зв’язки безпосередньо вказані в генотипі. І навпаки, у схемі з непрямим кодуванням в генотипі вказані правила і структури для створення нейромережі.

Однією з реалізацій еволюційних алгоритмів є SharpNEAT [2] – фреймворк з відкритим програмним кодом, розроблений мовою C#, що є імплементацією генетичного алгоритму нейроеволюції NEAT (NeuroEvolution of Augmenting Topologies) для платформи .NET. Еволюційний алгоритм використовує еволюційні механізми мутації, рекомбінації та селекції для пошуку нейромереж з поведінкою, яка відповідає умовам певної формально визначеної задачі. Прикладами таких задач є керування рухами кінцівок робота, польоту ракети або знаходження нейромережі, що реалізує певну цифрову логіку (наприклад, мультиплексор). Особливість NEAT і SharpNEAT полягає в тому, що вони виконують пошук як структури нейромережі (вузлів та зв’язків), так і вагових параметрів зв’язків між вузлами.

Попри сильні сторони методу нейроеволюції наростаючої топології, як-от можливість його застосування в завданнях, де важко обрати функцію витрат та топологію нейронної мережі, однією з проблем нейро-

еволюції та, зокрема, методу нейроеволюції наростаючої топології, є повільна конвергенція до оптимальних результатів, особливо у випадку роботи з комплексними та складними середовищами. Запропонована у попередній роботі [3] розподілена реалізація методу нейроеволюції наростаючої топології дозволяє радикально збільшити швидкість знаходження оптимальних конфігурацій нейронних мереж за наявності достатніх обчислювальних ресурсів.

У даній статті продемонстровано, як за допомогою апарату алгебр алгоритмів та гіперсхем [4–6] можна автоматизувати генерацію програм оцінювання для задач нейроеволюції на прикладі задачі двійкового мультиплексора, яка входить до складу SharpNEAT. Гіперсхеми є параметризованими алгоритмами для розв'язання певного класу задач. Установка конкретних значень параметрів і наступна інтерпретація гіперсхем дозволяє одержати алгоритми, адаптовані до конкретних умов застосування. Застосовано підхід до розробки алгоритмів на основі алгебр гіперсхем та параметрично-керованого генератора схем алгоритмів, запропонованих у [6]. Генератор алгоритмів є одним з компонентів розробленого інтегрованого інструментарію проєктування й синтезу програм. Розроблювані в системі специфікації алгоритмів оформляються за допомогою схем програм, поданих в системах алгоритмічних алгебр Глушкова (САА). Проведено експерименти з виконання згенерованого програмного коду мультиплексора в рамках SharpNEAT для багатопотокового та розподіленого варіантів процедури оцінки покоління.

1. Алгебри алгоритмів і гіперсхем та інструментальні засоби генерації алгоритмів і програм

Даний розділ присвячений розгляду систем алгоритмічних алгебр Глушкова та гіперсхем, покладених в основу використовуваного алгебро-алгоритмічного підходу до проєктування програм, а також інструментальні засоби синтезу алгоритмів та програм.

САА Глушкова орієнтовані на аналітичну форму подання алгоритмів й формалізовану трансформацію цих подань, зокрема, з метою оптимізації алгоритмів за заданими критеріями [4, 5]. САА є двоосновною алгеброю $GA = <\{Pr, Op\};\ \Omega_{GA}>$, де основи Pr й Op – множини логічних умов й операторів, визначені на інформаційній множині IS; Ω_{GA} – сигнатура операцій. Оператори є відображеннями (можливо, частковими) інформаційної множини в себе, логічні умови – предикати на множині IS, які приймають значення тризначної логіки $E_3 = \{0, 1, \mu\}$, де 0 – "хибність", 1 – "істина", μ – "невизначеність". Сигнатура $\Omega_{GA} = \Omega_1 \cup \Omega_2$ складається з системи Ω_1 логічних операцій (кон'юнкції, диз'юнкції, заперечення, прогнозування), що приймають значення в множині Pr, і системи Ω_2 операторних операцій, що приймають значення в множині операторів Op (композиція, альтернатива, цикл та ін.), які будуть розглянуті далі.

Засоби САА покладені в основу алгоритмічної мови САА/1, призначеної для багаторівневого структурного проєктування й документування послідовних та паралельних алгоритмів і програм. Перевагою її використання є можливість опису алгоритмів у природно-лінгвістичній формі. Оператори, подані мовою САА/1, називаються САА-схемами. Ідентифікатори предикатів записуються в одинарних лапках, операторів – у подвійних. Предикати та оператори в САА/1 можуть бути базисними або складеними. Базисні елементи є елементарними атомарними абстракціями в САА-схемах. Складені умови та оператори будуються з базисних із використанням операцій сигнатури САА.

Основними операторними операціями САА є такі (подані в природно-лінгвістичній формі):

- композиція (послідовне виконання операторів): "*operator* 1"; "*operator* 2";
- альтернатива (умовний оператор): IF '*condition*' THEN "*operator* 1" ELSE "*operator* 2" END IF;
- цикл: WHILE '*condition*' "*operator*" END OF LOOP;
- цикл з лічильником: FOR (*counter* FROM *start* TO *fin*) "*operator*" END OF LOOP;
- паралельне виконання n операторів: PARALLEL($i = 1,...,n$)("*operator i*");
- паралельна обробка елементів списку: PARALLEL FOR EACH (*elem* IN *list*)("*operator*(*elem*)").

Однією з важливих проблем в рамках алгебро-алгоритмічного підходу є підвищення адаптивності програм до конкретних умов їхнього використання. Зокрема, вона може бути вирішена завдяки застосування параметрично-керованої генерації специфікацій алгоритмів на основі схем вищого рівня, що називаються гіперсхемами [4–7].

В основу алгебраїчного апарату генерації схем покладені САА та абстрактно-автоматна модель параметрично-керованого генератора текстів [7]. Подібно до моделі ЕОМ, модель параметрично керованого генератора схем функціонує за принципом зворотного зв'язку. Як керуючий автомат використовується магазинний автомат Ψ, а як операційний – автомат $\overline{\Phi}$ зі стрічкою $\tilde{L}$ (Рис. 1). Стрічка $\tilde{L}$ призначена для запису тексту САА-схеми, що генерується. Множина $\overline{M}$ станів автомата автомата $\overline{\Phi}$ асоціюється з параметрами, що керують генерацією схем. Елементи інформаційної множини $\overline{P} = \overline{M} \times \overline{L}$ (де $\overline{L}$ – множина станів стрічки $\tilde{L}$) називаються станами операційної структури. На кожному кроці роботи з операційного в керуючий автомат надходить набір значень логічних умов $\{u_k \mid k \in I\}$, визначених на множині P, залежно від яких, а також від вмісту магазину $\tilde{M}$, керуючий автомат ініціює виконання певного оператора. Множина операторів $Op = \{A_j \mid j \in I\}$ поділяється на дві неперетинні множини – R_T (термінальні оператори) і R_N (нетермінальні оператори).

Виконання термінального оператора з множини R_T полягає в зміні поточного стану операційної структури, що, зокрема, може виражатися в запису на стрічку $\tilde{L}$ певного тексту. Виконання оператора $A \in R_N$ при поточному стані $p \in \overline{P}$ полягає в запису в магазин $\tilde{M}$ певного терму $F_N(A, p)$ і його подальшій інтерпретації

керуючим автоматом. Терм $F_N(A, p)$ є аналогом понять макровизначення, процедури, підпрограми та ін. Магазин $\tilde{M}$ використовується при обробці вкладених і рекурсивних термів. Згенерований текст є вмістом стрічки $\tilde{L}$ у заключному стані операційної структури.

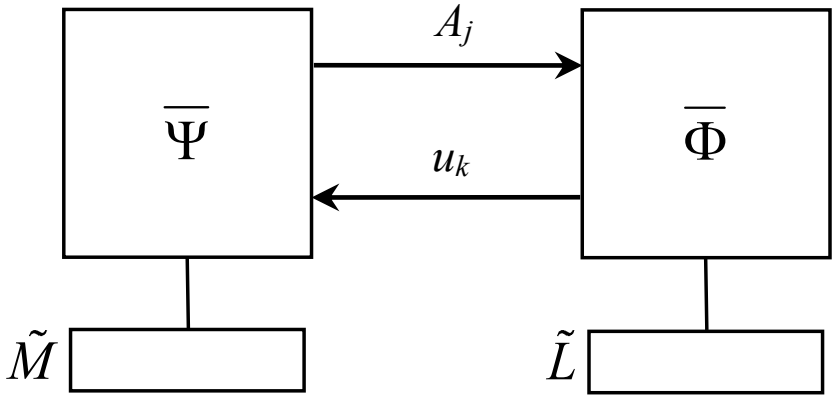

Рис. 1. Абстрактно-автоматна модель параметрично-керованого генератора текстів

Розглянутій абстрактно-автоматній моделі в [7] поставлена у відповідність алгебра гіперсхем (АГС) – апарат для формалізації алгоритмів параметрично керованої генерації схем, поданих в САА. АГС є двоосновною алгеброю $AHS = <\{Pr, Op\}; \Omega_{AHS}>$, де Pr – множина предикатів, що визначені на інформаційній множині $\overline{P}$ й приймають значення чотиризначної логіки $E_4 = \{0,\ 1,\ \mu,\ \eta\}$, де 0 – "хибність", 1 – "істина", μ – "невизначеність", η – "не обчислено"; Op – множина операторів, що визначені на інформаційній множині $\overline{P}$ й приймають значення у цій же множині; Ω_{AHS} – сигнатура операцій.

Множина предикатів асоційована з параметрами, які керують процесом генерації САА-схем. Операції сигнатури Ω_{AHS} подібні до операцій САА. Відмінність від САА полягає в тому, що предикати з множини Pr відображають елементи інформаційної множини $\overline{P}$ в множину $E_4 = \{0,\ 1,\ \mu,\ \eta\}$, де додаткове значення η означає "не обчислено". Елемент η використовується для вказання того, що значення предиката не може бути обчислене через нестачу інформації про значення параметрів гіперсхеми.

Застосування оператора $A \in Op$ до стану $p \in P$, приводить до переходу операційної структури $\overline{\Phi}$ в новий стан $A(p) \in P$ і запису на стрічку $\tilde{L}$ певного (можливо, пустого) фрагменту $F(A, p)$ схеми, яка генерується. Тут $F(A, p)$ – функція, яка специфікує метод генерації для всіх операцій сигнатури АГС і детально визначена в [7].

Операторні подання алгоритмів в АГС називаються гіперсхемами. Кожна гіперсхема A, застосована до стану $p \in \overline{P}$, породжує САА-схему $F(A, p)$. Гіперсхема A задає клас $L(A)$ схем в САА: $L(A) = \{F(A, p) \mid p \in \overline{P}\}$.

Функція $F(A, p)$ для операції "*operator* 1"; "*operator* 2" породжує оператор композиції без змін.

Операція альтернативи породжує оператор

$$F(C, p) = \begin{cases} \text{"}operator\ 1\text{"}, \text{ якщо '}condition\text{'} = 1; \\ \text{"}operator\ 2\text{"}, \text{ якщо '}condition\text{'} = 0; \\ \text{IF '}condition\text{' THEN "}operator\ 1\text{" ELSE "}operator\ 2\text{" END IF, якщо '}condition\text{'} = \eta; \\ e, \text{ якщо '}condition\text{'} = \mu, \end{cases}$$

де e – пусте слово. Результатом інтерпретації цієї операції є текст оператора "*operator* 1" при істинному значенні умови '*condition*'. Текст оператора "*operator* 2" буде згенерований при хибному значенні умови. Текст операції альтернативи без змін буде згенерований при не обчисленому значенні умови. Пустий текст буде результатом у випадку, якщо виникла помилка в процесі інтерпретації.

Обробка базисних умов та операторів гіперсхем у процесі генерації полягає в обчисленні виразів з параметрами гіперсхеми та їх підстановці в текст цих базисних елементів.

Гіперсхеми належать до методів трансформаційного синтезу і є суміжними до засобів переписування термів [8] та макрогенерації.

Розглянутий підхід до генерації схем програм реалізований в інструментарії, що дозволяє інтерпретувати базисні оператори й умови гіперсхем і входить до складу системи проєктування й синтезу програм [4–6]. Гіперсхеми та САА-схеми проєктуються в діалоговому режимі, що забезпечує їх синтаксичну правильність. Схеми будуються із використанням діалогового конструктора синтаксично правильних програм, основна ідея якого полягає у порівневому конструюванні схем програм зверху вниз за допомогою деталізації конструкцій мови САА, що подане у вигляді дерева (Рис. 2).

На кожному кроці побудови схеми система надає користувачу на вибір лише ті конструкції мови, підставляння яких у текст, що генерується, не порушує синтаксичну правильність САА- або гіперсхеми. За сконструйованою таким чином гіперсхемою виконується генерація САА-схеми, а за САА-схемою – синтез програми обраною цільовою мовою програмування (Рис. 3).

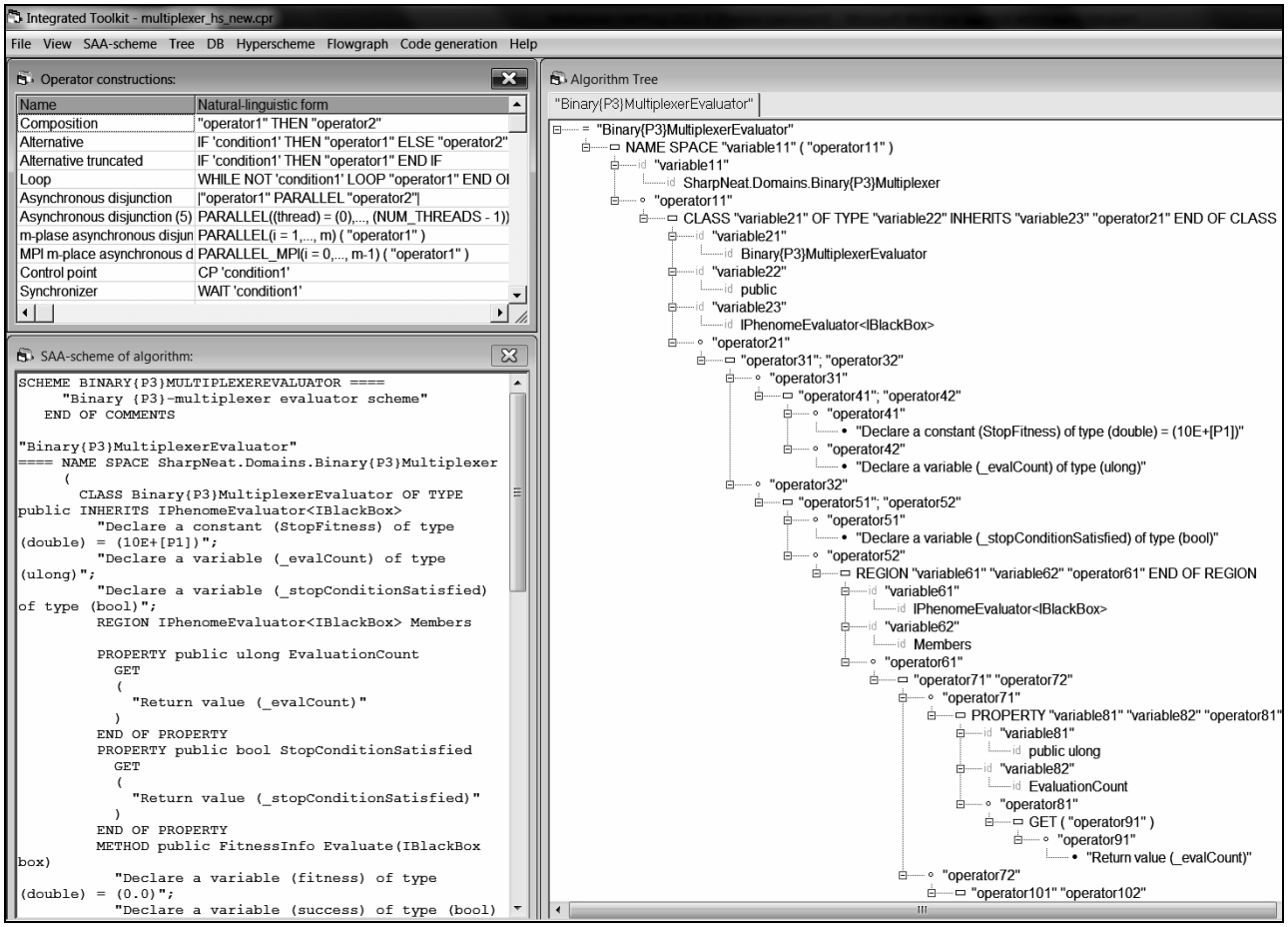

Рис. 2. Основне вікно інтегрованого інструментарію

Рис. 3. Послідовність генерації алгоритмів та програм в інтегрованому інструментарії

САА-схеми та гіперсхеми мають схожий синтаксис, що дозволяє гнучко використовувати параметри керування генерацією схем, що задаються на рівні базисних операторів та умов. Для полегшення обробки параметри позначаються в тексті базисних та інших елементів гіперсхеми у вигляді P*i*, де *i* – номер параметра. Вирази з параметрами гіперсхем вказуються у квадратних або фігурних дужках.

Приклад. Проілюструємо застосування апарату гіперсхем на прикладі фрагменту гібридного алгоритму сортування [9]. В наведеній далі схемі виконується вибір одного з алгоритмів сортування (вставками, послідовного або паралельного сортування злиттям) залежно від значення параметру **P1** – розміру масиву.

```
"Hybrid sort (array)"
==== IF '[P1 <= 200]'
       THEN
         "Insertion sort (array)"
       ELSE
         IF '[P1 <= 1000]'
         THEN
           "Sequential merge sort (array)"
```

```
    ELSE
      "Parallel merge sort (array)"
    END IF
  END IF
```

Нехай заздалегідь відомо, що заданий схемою алгоритм буде застосовуватися в умовах, коли довжина вхідного масиву **P1** ≥ 500. Тоді наведена схема стає надлишковою. Розглядаючи її як гіперсхему, можна вважати, що на етапі генерації САА-схеми умова '[**P1** <= 200]' набуває значення "хибність", тоді як '[**P1** <= 1000]' - значення "не обчислено". В результаті генерації тексту за гіперсхемою отримаємо скорочену САА-схему:

```
"Hybrid sort (array)"
==== IF '[P1 <= 1000]'
     THEN
       "Sequential merge sort (array)"
     ELSE
       "Parallel merge sort (array)"
     END IF
```

2. Розробка гіперсхеми для генерації алгоритмів оцінки двійкового мультиплексора

В даному розділі апарат алгебри гіперсхем застосований для генерації класів САА-схем оцінювання для двійкового мультиплексора (BinaryMultiplexerEvaluator), з подальшою реалізацією схем мовою C# для фреймворку SharpNEAT [2].

Мультиплексор – пристрій, що має декілька входів даних x_i ($i = 0,...,n-1$), адресні входи s_j ($j = 0,...,m-1$), та один вихід y. Пристрій передає сигнал з одного із входів даних на вихід; водночас вибір потрібного входу здійснюється шляхом подачі відповідної комбінації керуючих сигналів на адресних входах. Кількість входів даних n та кількість адресних входів m зв'язані співвідношенням: $n = 2^m$. Умовна схема мультиплексора з 11-ма входами зображена на Рис. 4.

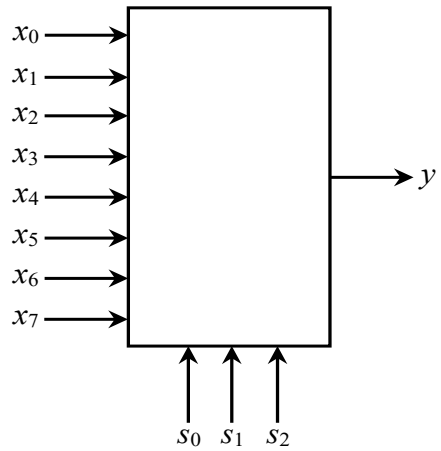

Рис. 4. Умовна схема мультиплексора на 11 входів

Побудована за допомогою інтегрованого інструментарію гіперсхема наведена далі. Параметрами гіперсхеми є такі: **P1** – кількість адресних входів мультиплексора; **P2** – кількість інформаційних входів; **P3 = P1 + P2 –** загальна кількість входів. Всі входи приймають двійкові значення (0 або 1). Двійкова адреса подається на адресні входи, що репрезентує вибір одного зі значень входів для даних. Оцінка складається з вичерпної перевірки нейронної мережі на кожній з 2^{P3} можливих комбінацій входів. Вихідне значення нейронної мережі має збігатися зі значенням одного з входів даних, представленого двійковою адресою з адресних входів. Вихідне значення, менше за 0,5, вважають двійковим нулем, вихідне значення, більше або рівне 0,5, – двійковою одиницею. Значення оцінки (придатності) адитивно розраховують у результаті вичерпної перевірки.

```
SCHEME BINARY{P3}MULTIPLEXEREVALUATOR ====
    "Binary {P3}-multiplexer evaluator scheme"
  END OF COMMENTS

"Binary{P3}MultiplexerEvaluator"
==== NAME SPACE SharpNeat.Domains.Binary{P3}Multiplexer
      (
```

```
CLASS Binary{P3}MultiplexerEvaluator OF TYPE public INHERITS
        IPhenomeEvaluator<IBlackBox>
  "Declare a constant (StopFitness) of type (double) = (10E+[P1])";
  "Declare a variable (_evalCount) of type (ulong)";
  "Declare a variable (_stopConditionSatisfied) of type (bool)";

  REGION IPhenomeEvaluator<IBlackBox> Members

    PROPERTY public ulong EvaluationCount
      GET
      (
        "Return value (_evalCount)"
      )
    END OF PROPERTY

    PROPERTY public bool StopConditionSatisfied
      GET
      (
        "Return value (_stopConditionSatisfied)"
      )
    END OF PROPERTY

    METHOD public FitnessInfo Evaluate(IBlackBox box)
      "Declare a variable (fitness) of type (double) = (0.0)";
      "Declare a variable (success) of type (bool) = (true)";
      "Declare a variable (output) of type (double)";
      "Declare a variable (inputArr) of type (ISignalArray) = (box.InputSignalArray)";
      "Declare a variable (outputArr) of type (ISignalArray) = (box.OutputSignalArray)";
      "Increase (_evalCount) by (1)";
      FOR (i FROM 0 TO [Pow(2,P3)-1])
      LOOP
        "Declare a variable (tmp) of type (int) = (i)";
      FOR (j FROM 0 TO [P3-1])
      LOOP
        (inputArr[j] := tmp&0x1);
        (tmp := tmp >> 1)
      END OF LOOP;
      "Activate the black box (box)";
      "Read output signal (output)(outputArr)";
      IF (((1<<([P1]+(i&0x[P2-1])))&i) != 0)
      THEN (fitness := fitness + 1.0 - ((1.0 - output) * (1.0 - output)));
        IF (output < 0.5)
        THEN (success := false)
        END IF
      ELSE (fitness := fitness + 1.0 - (output * output));
        IF (output >= 0.5)
        THEN (success := false)
        END IF
      END IF;
      "Reset black box state ready for next test case (box)"
    END OF LOOP;
    IF success
    THEN (fitness := fitness + 10E+[P1])
    END IF;
    IF (fitness >= StopFitness)
    THEN (_stopConditionSatisfied := true)
     END IF;
     "Return value (new FitnessInfo(fitness, fitness))"
  END OF METHOD

  METHOD public void Reset()
    "Empty operator"
  END OF METHOD
```

```
        END OF REGION
      END OF CLASS
    )

END OF SCHEME BINARY{P3}MULTIPLEXEREVALUATOR
```

Залежно від встановлених значень параметрів, на основі гіперсхеми інтегрований інструментарій виконує генерацію САА-схеми із зазначеного класу, наприклад:

1) мультиплексор з 3-ма входами – значення параметрів: **P1** = 1, **P2** = 2, **P3** = 3;

2) мультиплексор з 6-ма входами – значення параметрів: **P1** = 2, **P2** = 4, **P3** = 6;

3) мультиплексор з 11-ма входами – значення параметрів: **P1** = 3, **P2** = 8, **P3** = 11.

У гіперсхемі у фігурних дужках **{P3}** вказано параметр, який потрібно замінити на відповідне число, записане словами, тобто при значенні **P3** = 11 буде вставлено текст “Eleven”. В квадратних дужках (наприклад, **[P1]** або **[P3-1]**) вказуються параметри або арифметичні вирази з ними, які потрібно замінити на відповідне число. Так, для циклу FOR (i FROM 0 TO **[Pow(2,P3)-1]**) при значенні параметра **P3** = 11 буде згенеровано текст FOR (i FROM 0 TO 2047).

На основі гіперсхеми в інтегрованому інструментарії були згенеровані САА-схеми оцінки мультиплексорів з 3-ма, 6-ма та 11-ма входами. Далі за схемами був згенерований програмний код мовою C# для системи SharpNEAT [2].

Схема паралельної багатопотокової процедури оцінки чергового покоління, реалізованої в SharpNEAT, має такий вигляд:

```
METHOD private void Evaluate_Caching(IList<TGenome> genomeList)
  PARALLEL FOR EACH (genome IN genomeList)
  (
    “Get (phenome) for (genome)”;
    IF (phenome = null)
    THEN “Decode the (phenome) and store a reference against the (genome)”
    END IF;
    IF (phenome = null)
    THEN
       “Set (genome) fitness to (0.0)”;
       “Set (genome) auxiliary fitness info to (null)”
    ELSE
      “Evaluate (phenome) and get fitness (fitnessInfo)”;
      “Set (genome) fitness to (fitnessInfo._fitness)”;
      “Set (genome) auxiliary fitness info to (fitnessInfo._auxFitnessArr)”
    END IF
  )
END OF METHOD
```

В [3] розроблено розподілений варіант цієї процедури для виконання на платформі хмарних обчислень.

3. Результати експериментів

У даному розділі наведено результати експериментів з багатопотоковою та розподіленою реалізаціями нейроеволюції наростаючої топології. Задачею обрано мультиплексор з 11-ма входами.

На Рис. 5 подано графік залежності швидкості оцінки для багатопотокової реалізації від максимального рівня паралелізму на процесорі AMD Ryzen 5 3550H (2.1 ГГц), 8 ядер. Максимальна середня кількість оцінювань досягнута при рівні паралелізму, що дорівнював 8. Середня кількість оцінювань обчислювалась для перших 200 оцінювань, що в сукупності тривали приблизно 4 хв. Прискорення порівняно з кількістю оцінювань, отриманому при рівні паралелізму 1, становить близько 3.1.

За інші середовища для виконання однопроцесної та розподіленої реалізації були обрані такі конфігурації:

1) локальне середовище, Intel Core i9-9900K CPU (3.60 ГГц – 5.00 ГГц), 8 ядер, 16 логічних процесорів, 32.0 ГБ RAM, один процес, 16 потоків;

2) локальне середовище, Intel Core i9-9900K CPU (3.60 ГГц – 5.00 ГГц), 8 ядер, 16 логічних процесорів, 32.0 GB RAM, розподілена реалізація, 16 локальних клієнтів-виконувачів;

3) хмарне середовище, 3rd Gen AMD EPYC Amazon EC2 C6a.large, 3.60 ГГц, 2 ядра, 4.0 ГБ RAM, до 12.5 Гбіт/с мережевої пропускної здатності та до 6600 Мбіт/с пропускної здатності сховища. Розподілена реалізація, 16 хмарних клієнтів-виконувачів;

4) те ж саме хмарне середовище, але 32 хмарних клієнтів-виконувачів;

5) те ж саме хмарне середовище, але 64 хмарних клієнтів-виконувачів.

На Рис. 6 зображено графік залежності швидкості оцінювання (кількості оцінювань в секунду) від номеру покоління для локальних конфігурацій середовища. Як видно з графіку, розподілена реалі-

зація очікувано демонструє гірші результати у порівнянні з однопроцесною реалізацією у зв'язку з наявністю накладних витрат на взаємодію між процесами. З ростом складності задачі оцінювання (ростом розміру згенерованої нейромережі) ефективність однопроцесної та локальної розподіленої реалізації вирівнюється, оскільки накладні витрати обчислювальних ресурсів стають непомірно меншими за витрати на оцінювання.

Рис. 5. Графік залежності швидкості оцінки для багатопотокової реалізації від максимального рівня паралелізму

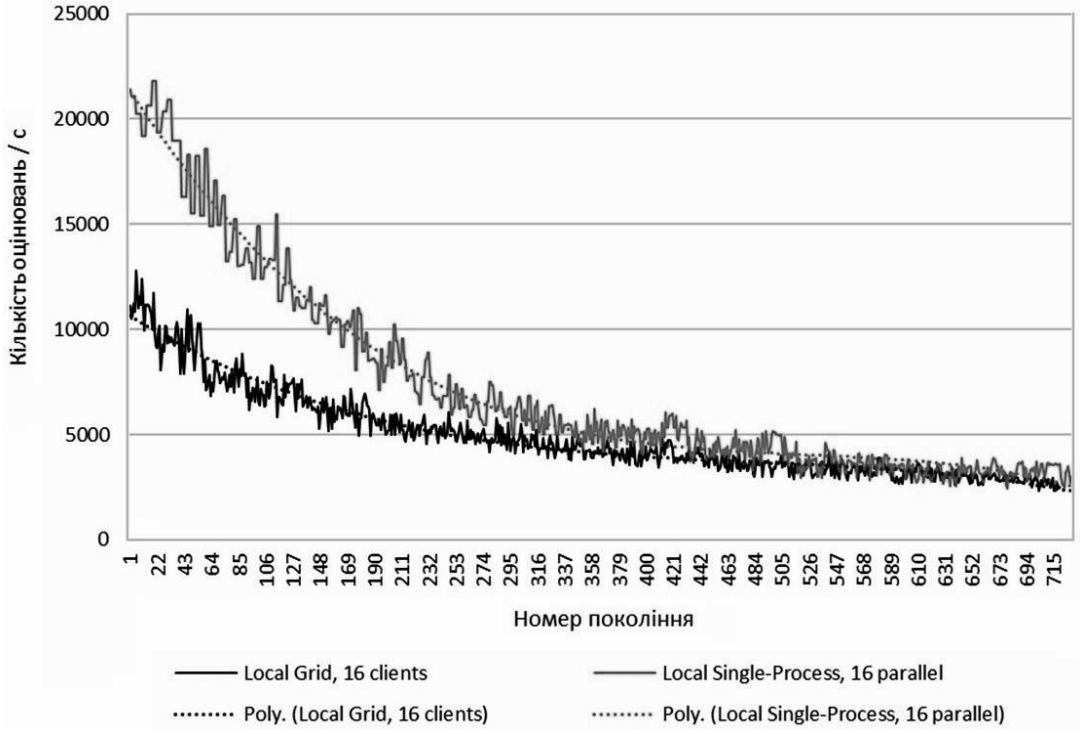

Рис. 6. Графік залежності швидкості оцінювання від номера покоління для локальних конфігурацій середовища

На Рис. 7 зображено графік залежності швидкості оцінювання від номера покоління для хмарних конфігурацій середовища. Як видно з графіка, розподілена хмарна реалізація очікувано демонструє гірші результати (за тієї ж кількості клієнтів-виконувачів) у порівнянні з однопроцесною та локальною розподіленою реалізацією у зв'язку з наявністю накладних витрат на взаємодію між процесорами багатьох комп'ютерів, клієнтів-виконувачів. Проте, з ростом кількості клієнтів-виконувачів, ми можемо нехтувати сталим значенням накладних витрат і отримувати лінійний ріст ефективності розподіленої системи.

Даний експеримент продемонстрував здатність розподіленої системи проводити оцінювання на 64 хмарних клієнтах-виконувачах і отримувати приріст у 60–100 % від максимальних можливостей однопроцесорної локальної реалізації.

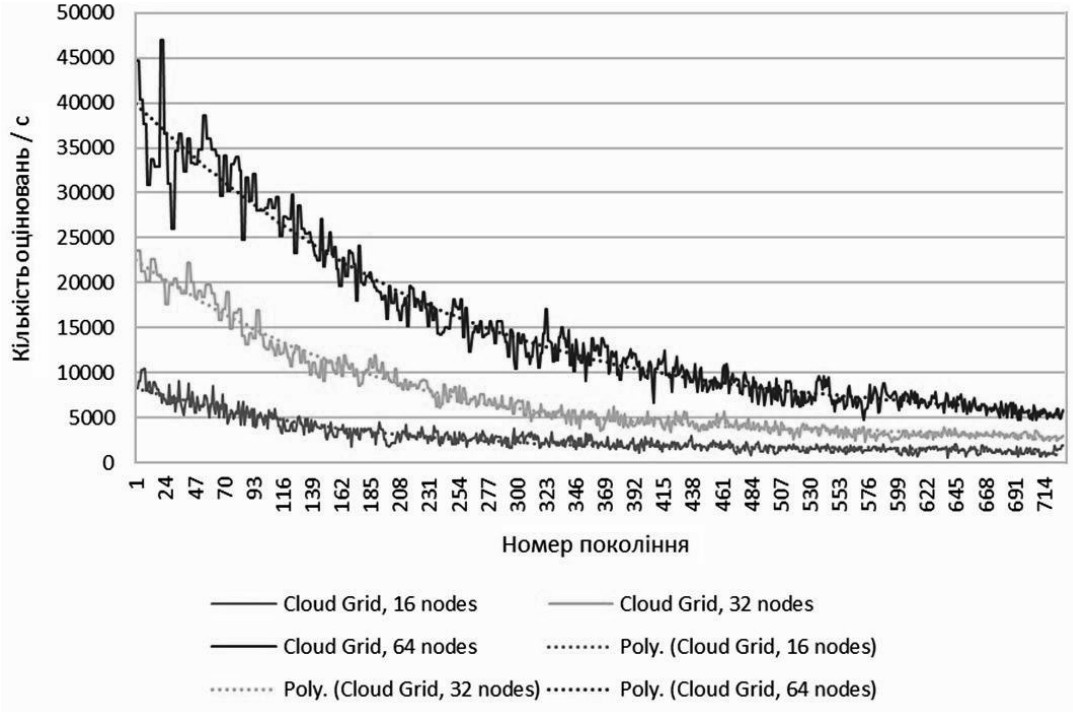

Рис. 7. Графік залежності швидкості оцінювання від номеру покоління для хмарних конфігурацій середовища

Висновки

Застосовано апарат алгебри гіперсхем для автоматизованої генерації параметричних алгоритмів нейроеволюції на прикладі задачі оцінювання для двійкового мультиплексора для системи SharpNEAT. Гіперсхема є високорівневим параметризованим алгоритмом для розв'язання певного класу задач. Установка значень параметрів й подальша інтерпретація гіперсхеми дозволяє отримати схеми алгоритмів, адаптованих до конкретних умов застосування. Засоби гіперсхем реалізовані в розробленому інтегрованому інструментарії автоматизованого проєктування й синтезу програм. На основі схем алгоритмів в системі виконується генерація програм у цільовій мові програмування. Перевагою системи є можливість опису схем алгоритмів у природно-лінгвістичній формі. Проведено експеримент із виконання згенерованої програми для задачі оцінювання двійкового мультиплексора на розподіленій хмарній платформі, що продемонстрував можливість розробленої розподіленої системи проводити оцінювання на 64 хмарних клієнтах-виконувачах і отримувати приріст у 60–100 % від максимальних можливостей однопроцесорної локальної реалізації.

Література

1. Stanley K.O., Clune J., Lehman J., Miikkulainen R. Designing neural networks through neuroevolution. Nature Machine Intelligence. 2019. Vol. 1. P. 24–35.
2. SharpNEAT – Evolution of Neural Networks : веб-сайт. URL: https://github.com/colgreen/sharpneat (дата звернення: 12.08.2022).
3. Ашур І.З., Дорошенко А.Ю. Розподілена реалізація методу нейроеволюції наростаючої топології. Проблеми програмування. 2021. № 3. С. 3–15. URL: http://pp.isofts.kiev.ua/ojs1/article/view/467 (дата звернення: 12.08.2022).
4. Doroshenko A., Yatsenko O. Formal and adaptive methods for automation of parallel programs construction: emerging research and opportunities. Hershey: IGI Global, 2021. 279 p.
5. Algebra-algorithmic models and methods of parallel programming / Andon P.I. et al. Kyiv: Akademperiodyka, 2018. 192 p.
6. Yatsenko O. On parameter-driven generation of algorithm schemes. Proc. Int. Workshop "Concurrency, Specification, and Programming", CS&P'2012, Berlin, Germany (26–28 September 2012). Berlin: Humboldt University, 2012. P. 428–438. URL: http://ceur-ws.org/Vol-928/0428.pdf (дата звернення: 12.08.2022).
7. Ющенко Е.Л., Цейтлин Г.Е., Галушка А.В. Алгебро-грамматические спецификации и синтез структурированных схем программ. Кибернетика. 1989. № 6. С. 5–16.
8. Doroshenko A., Shevchenko R. A rewriting framework for rule-based programming dynamic applications. Fundamenta Informaticae. 2006. Vol. 72, No. 1–3. P. 95–108. URL: https://www.researchgate.net/publication/250731334 (дата звернення: 12.08.2022).
9. A mixed method of parallel software auto-tuning using statistical modeling and machine learning / Doroshenko A. et al. Communications in Computer and Information Science. Information and Communication Technologies in Education, Research, and Industrial Applications. 2019. Vol. 1007. P. 102–123.

References

1. STANLEY, K. O., CLUNE, J., LEHMAN, J. & MIIKKULAINEN, R. (2019) Designing neural networks through neuroevolution. *Nature Machine Intelligence*. 1. p. 24-35.
2. SharpNEAT – Evolution of Neural Networks. [Online] Available from: https://github.com/colgreen/sharpneat [Accessed 12/08/ 2022]
3. ACHOUR, I. Z. & DOROSHENKO, A. YU. (2021) Distributed implementation of neuroevolution of augmenting topologies method. Problems in programming. [Online] (3). p. 3-15. (in Ukrainian). Available from: http://pp.isofts.kiev.ua/ojs1/article/view/467 [Accessed 12/08/2022]

4. DOROSHENKO, A. & YATSENKO, O. (2021) Formal and adaptive methods for automation of parallel programs construction: emerging research and opportunities. Hershey: IGI Global.
5. ANDON, P. I. et al. (2018) Algebra-Algorithmic Models and Methods of Parallel Programming. Kyiv: Akademperiodyka.
6. YATSENKO, O. On parameter-driven generation of algorithm schemes. (2012) Proc. Int. Workshop "Concurrency, Specification, and Programming", CS&P'2012, Berlin, Germany (26–28 September 2012). [Online] Berlin: Humboldt University. p. 428-438. Available from: http://ceur-ws.org/Vol-928/0428.pdf [Accessed 12/08/2022]
7. YUSHCHENKO, K. L., TSEITLIN, G. O. & GALUSHKA, A. V. (1989) Algebra-algorithmic specifications and synthesis of structured schemes of programs. Cybernetics. (6). p. 5-16. (in Russian).
8. DOROSHENKO, A. & SHEVCHENKO, R. (2006) A rewriting framework for rule-based programming dynamic applications. *Fundamenta Informaticae*. [Online] 72 (1-3). p. 95-108. Available from: https://www.researchgate.net/publication/250731334 [Accessed 12/08/2022]
9. DOROSHENKO, A. et al. (2019) A mixed method of parallel software auto-tuning using statistical modeling and machine learning. *Communications in Computer and Information Science. Information and Communication Technologies in Education, Research, and Industrial Applications*. 1007. p. 102-123.

Одержано 09.08.2022

***Про авторів*:**

Дорошенко Анатолій Юхимович,
доктор фізико-математичних наук, професор, завідувач відділу.
Кількість публікацій в українських виданнях – понад 180.
Кількість зарубіжних публікацій – понад 60.
Індекс Хірша – 6.
http://orcid.org/0000-0002-8435-1451,

Ашур Ілля Зін-Еддінович,
аспірант НТУУ "КПІ імені Ігоря Сікорського".
Кількість публікацій в українських виданнях – 3.
Кількість наукових публікацій в іноземних виданнях – 1.
https://orcid.org/0000-0003-2348-8777,

Місце роботи авторів:

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»,
проспект Перемоги 37
та Інститут програмних систем НАН України, 03187,
м. Київ-187, проспект Академіка Глушкова, 40.
Тел.: (38)(044) 526-35-59.
E-mail: doroshenkoanatoliy2@gmail.com,
ilyaachour@gmail.com,

Прізвища та ім'я авторів і назва доповіді українською мовою:
Дорошенко А. Ю., Ашур І. З.-Е.
Автоматизована генерація програм для одного класу параметричних
алгоритмів нейроеволюції

Прізвища та ім'я авторів і назва доповіді англійською мовою:
Doroshenko A. Yu., Achour I. Z.-E.
Automated generation of programs for
a class of parametric neuroevolution algorithms

Контакт для редактора: Ашур Ілля Зін-Еддінович,
аспірант НТУУ «КПІ імені Ігоря
Сікорського», e-mail: ilyaachour@gmail.com,
тел.: +(38)(044) 526-35-59