Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження та застосування генетичного алгоритму ES-HyperNEAT для вирішення задачі комівояжера
(тема)

Виконав:
студент 2 курсу, групи СШІм-19-1
Шумак А.Д.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту
(повна назва спеціалізації)

Керівник проф. Філатов В. О.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ В.О. Філатов
(підпис) (прізвище, ініціали)

2020 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту (СШІ)
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові Шумаку Альберту Джозефовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження та застосування генетичного алгоритму ES-HyperNEAT для вирішення задачі комівояжера

затверджена наказом університету від 30 жовтня 20 20 р. № 1497ст
2. Термін подання студентом роботи до екзаменаційної комісії 16 грудня 20 20 р.
3. Вихідні дані до роботи Науково-технічні публікації, дані відомих наукових проектів щодо розробки систем логістики, дані статей, результати експериментальних досліджень.

4. Перелік питань, що потрібно опрацювати в роботі 1. Аналіз предметної галузі та формалізована постановка задачі, 2. Нейроеволюція, 3. Нейроеволюція наростаючої топології, 4. Аналіз інструментальних засобів, 5. Імітаційне моделювання

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри)
Рисунок 1 – Порогова функція, Рисунок 2 – Лінійно–порогова функція, Рисунок 3 – Сигмоїдальна функція, Рисунок 4 – Найпростіша штучна мережа, Рисунок 5 – Двошарова нейронна мережа з ваговими коефіцієнтами, Рисунок 6 – Симетрична задача для чотирьох міст, Рисунок 7 – Порівняння геномів для різних мережевих топологій, Рисунок 8 – Кодування CPPN, Рисунок 9 – Інтерпретація шаблону геометричних зв'язків на основі гіперкуба, Рисунок 10 – Приклад вилучення інформації з дерева квадрантів, Рисунок 11 – Алгоритм ES-HyperNEAT, Рисунок 12 – Розділ і ініціалізація, Рисунок 13 – Випадкова модель графу з містами, Рисунок 14 – Залежність між довжиною шляху і поколінням для 48 міст, Рисунок 15 – Графік залежності значення функції пристосованості останнього покоління від кількості міст

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Основна частина	проф. Філатов В. О.		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на дипломну роботу	01.11.2020	виконано
2	Аналіз предметної галузі і постановка завдання	10.11.2020 – 14.11.2020	виконано
3	Дослідження методів та технологій	15.11.2020 – 17.11.2020	виконано
4	Аналіз існуючих реалізацій	18.11.2020 – 19.11.2020	виконано
5	Розробка методу рішення поставленої проблеми	20.11.2020 – 22.11.2020	виконано
6	Створення імітаційної моделі	23.11.2020 – 30.11.2020	виконано
7	Тестування і опрацювання імітаційної моделі	01.12.2020 – 02.12.2020	виконано
8	Оформлення пояснювальної записки	03.12.2020 – 10.12.2020	виконано
9	Оформлення графічних матеріалів	11.12.2020 – 12.12.2020	виконано
10	Попередній захист	14.12.2020	виконано
11	Захист перед ЕК	16.12.2020	

Дата видачі завдання 01 листопада 20 20 р.

Студент _____
(підпис)

Керівник роботи _____ _____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Записка пояснювальна: 85 с., 25 рис., 1 табл., 2 дод., 41 джерел.

ГЛИБИННЕ НАВЧАННЯ, ЕВОЛЮЦІЙНІ АЛГОРИТМИ, НЕЙРОЕВОЛЮЦІЯ, CPPN, ES-HYPERNEAT, INDIRECT ENCODING, JAVA, KERAS, PYTHON, REINFORCMENT LEARNING

У цій атестаційній роботі розглядається рішення задачі комівояжера за допомогою технологій штучного інтелекту, а саме нейронних мереж і еволюційних алгоритмів. Поставлена задача полягає в тому, щоб розробити систему, яка буде показувати найвигідніші маршрути між обранами містами з урахуванням заданих витратат, використовуючи найвищі значення функції пристосованості при обранні ваг зв’язків між нейронами.

Для вирішення поставленої задачі було проаналізовано математичну модель задачі, аналогічні алгоритми для вирішення подібних задач, переваги і недоліки генетичних алгоритмів і пристосування штучних нейроних мереж для знаходження оптимального розв’язку для цієї задачі. В якості інструменту використовуються мережі, що створюють композиційні шаблони, і еволюційний алгоритм ES-HyperNEAT.

Крім того, в цій роботі описується процес імітаційного моделювання, який включає в себе створення, навчання і використання таких нейроних мереж. Розглядаються архітектура мережі, її переваги та недоліки, а також сфери задач, де вона користується особливим попитом. Проводиться порівняльний аналіз точності і швидкості навчання моделей нейроних мереж для поставленої задачі.

У результаті було показано, що генетичні алгоритми, як метаевристичні алгоритми, дають досить швидкий оптимальний розв’язок задачі у порівнянні з іншими алгоритмами, досягаючи переваги на 30% у швидкості навчання і 15% більш оптимальних розв’язків.

РЕФЕРАТ

Пояснительная записка: 85 с., 25 рис., 1 табл., 2 прил., 41 источников.

ГЛУБОКОЕ ОБУЧЕНИЕ, НЕЙРОЭВОЛЮЦИЯ, ЭВОЛЮЦИОННЫЕ АЛГОРИТМЫ, CPPN, ES-HYPERNEAT, INDIRECT ENCODING, JAVA, KERAS, PYTHON, REINFORCMENT LEARNING

В этой аттестационной работе рассматривается решение задачи коммивояжера с помощью технологий искусственного интеллекта, а именно нейронных сетей и эволюционных алгоритмов. Поставленная задача заключается в том, чтобы разработать систему, которая будет показывать самые выгодные маршруты между выбранными городами с учетом заданных стоимостей, используя наилучшие значения функции приспособленности при выборе весов связей между нейронами.

Для решения поставленной задачи были проанализированы математическая модель задачи, аналогичные алгоритмы для решения подобных задач, преимущества и недостатки генетических алгоритмов и приспособления искусственных нейронных сетей для нахождения оптимального решения для этой задачи. В качестве инструмента используются сети, создающие композиционные шаблоны, и эволюционный алгоритм ES-HyperNEAT.

Кроме того, в этой работе описывается процесс имитационного моделирования, который включает в себя создание, обучение и использование таких нейронных сетей.

В результате было показано, что генетические алгоритмы, как метаэвристические алгоритмы, дают достаточно быстрое оптимальное решение задачи по сравнению с другими алгоритмами, достигая преимущества в 30% по скорости обучения и 15% более оптимальных решений.

ABSTRACT

Explanatory note: 85 pages, 25 figures, 1 table, 2 appendixes, 41 sources.

CPPN, DEEP LEARNING, ES-HYPERNEAT, EVOLUTIONARY ALGORITHMS, INDIRECT ENCODING, JAVA, KERAS, NEUROEVOLUTION, PYTHON, REINFORCMENT LEARNING

This work is about the travelling salesman's problem with the help of artificial intelligence technologies, namely neural networks and evolutionary algorithms. The task is to develop a system that will show the most advantageous routes between selected cities at a given cost, using the highest values of the fitness function when selecting the weights of connections between neurons.

To solve this problem, the mathematical model of the problem, another algorithms for solving similar problems, the advantages and disadvantages of genetic algorithms and the adaptation of ANN to find the optimal solution for this problem were analyzed. As a tool, compositional pattern-producing networks and the evolutionary algorithm ES-HyperNEAT are used.

In addition, this paper describes the process of simulation, which includes the creation, training and use of such neural networks. The architecture of the network, its advantages and disadvantages, as well as areas of task where it is in great demand are considered. The comparative analysis of accuracy and speed of training of models of neural networks for the set task is done.

As a result, it was shown that genetic algorithms, as metaheuristic algorithms, give a fairly fast optimal solution of the problem compared to other algorithms, achieving a 30% advantage in learning speed and 15% more optimal solutions.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів 8
Вступ 9
1 Аналіз предметної галузі та формалізована постановка задачі 11
1.1 Штучна нейронна мережа 11
1.1.1 Поняття штучної нейронної мережі 12
1.1.2 Складові штучної нейронної мережі 13
1.1.3 Навчання штучних нейронних мереж 17
1.2 Навчання з підкріпленням 20
1.3 Еволюційні алгоритми 21
1.3.1 Опис і види алгоритмів 22
1.3.2 Генетичні алгоритми і генетичне кодування 23
1.4 Застосування еволюційних алгоритмів для створення штучної нейронної мережі 25
1.5 Задача комівояжера 27
1.6 Постановка завдання дослідження 30
2 Нейроеволюція 30
2.1 Нейроеволюційні системи фіксованої топології 32
2.2 TWEANNs 32
2.2.1 Непряме кодування 33
2.3 Початкові популяції та топологічні нововведення 35
3 Нейроеволюція наростаючої топології 37
3.1 Алгоритм NEAT 37
3.2 Генеративні і розвивальні системи 42
3.2.1 Мережі, що створюють композиційні шаблони 42
3.2.2 HyperNEAT 44
3.3 Вибір зв’язків для кодування 47
3.4 Вилучення інформації з дерева квадрантів 50
3.5 Алгоритм ES-HyperNEAT 53

4 Аналіз інструментальних засобів 56
4.1 Фреймворк Keras 56
4.2 Бібліотека MultiNEAT 58
5 Імітаційне моделювання 62
5.1 Створення і налаштування моделей 62
5.2 Навчання і візуалізація результатів 68
5.3 Порівняння результатів 70
Висновки 72
Перелік джерел посилань 73
Додаток А Вихідний код програми для імітаційного моделювання 77
Додаток Б Відомість атестаційної роботи 84

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ГА – генетичний алгоритм;

МПВ – марківський процес вирішування;

НЕ – нейроеволюція;

ШНМ – штучна нейронна мережа;

AE – artificial embryogeny – штучний ембріогенез;

ANN – artificial neural network – штучна нейронна мережа;

CPPN – compositional pattern-producing networks – мережі, що створюють композиційні щаблони;

ES-HyperNEAT – evolvable-substrate hypercube-based neuroevolution of augmenting topologies – еволюційний субстрат нейроеволюції наростаючої топології на основі гіперкубу;

GDS – generative and developmental systems – генеративні і розвивальні системи;

NEAT – neuroevolution of augmenting topologies – нейроеволюція наростаючої топології;

ReLU – rectified linear unit – випрямлена лінійна одиниця;

TWEANNs – topology and weight evolving artificial neural networks – штучні нейронні мережі з топологією і вагами, що розвиваються.

ВСТУП

Однією з актуальних проблем сучасності ще досі є задача пошуку найбільш вигідного маршруту з обраного переліку міст. З розвитком технологічного прогресу і потужностей, людина приходить до того, що більшість рутинних справ слід автоматизувати через більш продуктивне виконання і забезпечення безпеки. Задача логістики традиційно була проблемою для вирішення з часів появи перших бродячих торговців. Нажаль, майже увесь цей час єдиним доступним алгоритмом рішення був повний перебір, що не дозволяє шукати вигідні маршрути для більше, ніж кілька міст. Проблема автоматизації і удосконалення подібної діяльності стоїть майже увесь час становлення торгівлі між народами. На протязі багатьох десятиліть був розвиток теоретичної бази комп’ютерних методів навчання моделей для вирішення таких задач, зокрема штучні нейроні мережі, методи оптимізації, генетичні алгоритми, мурашиний алгоритм та інші. За ці роки підіймалися важливі питання щодо методів та технологій, що підтримують автоматичний і ефективний пошук необхідних маршрутів, особливо в режимі реального часу. Задача комівояжера може застосовуватись для широкого спектра задач, в основі яких лежить проходження певного об’єкту через множину пунктів так, щоб закінчення шляху збігалося з початком. На сьогодні, найбільше потребується це для планування робіт для кур'єра з доставки товарів та планування робіт для однієї машини, яка займається перевезенням товарів на великі відстані. Обрання найвигіднішого маршруту призводить до економії пального і часу, а слідом і зниження вартості таких перевезень, зменшення кількості заторів на дорогах, більшої безпеки і більш ефективного планування і керування даної сфери діяльності. Через збільшення кількості перевезень у світі, саме ці питання висвітлюються у розділах інтелектуального аналізу даних і машинного навчання і зараз мають активну розробку по всьому світу.

Інтелект у природі є продуктом живого мозку, який сам є продуктом

природного розвитку. Незважаючи на те, що дослідники в галузі НЕ намагаються повторити цей процес, штучні нейронні мережі, дотепер розвинені за допомогою алгоритмів НЕ, не відповідають особливим можливостям біологічного мозку. Нещодавно представлений підхід HyperNEAT, заснований на гіперкубі, зменшив цей пробіл, продемонструвавши, що шаблон вагових коефіцієнтів зв'язків ANN може створюватися як функція його геометрії, тим самим дозволяючи розвивати великі ANN для багатовимірних задач. Проте положення і кількість нейронів, пов'язаних за допомогою цього підходу, користувач повинен визначати апріорі і, на відміну від живого мозку, не можуть змінюватися під час еволюції. Еволюційний субстрат HyperNEAT (ES-HyperNEAT) вирішує це обмеження шляхом автоматичного виведення геометрії вузла на основі неявної інформації в шаблоні ваг, кодованих HyperNEAT, тим самим уникаючи необхідності кодувати явне розміщення. Цей підхід може не тільки змінити розташування кожного нейрона в мережі, але також може повторювати представлення області різної щільності, а це означає, що роздільна здатність може цілісно зростати в процесі еволюції. ES-HyperNEAT демонструється за допомогою задачі комівояжера, виявляючи, що ANN, створені цим новим підходом, набувають природних властивостей, таких як нейронна топографія та геометрична регулярність. Також важливо відзначити, що компактне непряме кодування ES-HyperNEAT може бути розпочато з упередженням до бажаного класу топографій ANN, що полегшує еволюційний пошук. То ж можемо говорити про те, що ES-HyperNEAT значно розширює сферу нейронних структур, які може виявити еволюція.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ФОРМАЛІЗОВАНА ПОСТАНОВКА ЗАДАЧІ

1.1 Штучна нейронна мережа

Штучні нейронні мережі (ШНМ), що являють собою один із напрямів обчислювального інтелекту – це потужний інструмент для вирішення широкого кола завдань інтелектуального аналізу даних. З плином часу ця методологія активно і успішно використовувалася дослідниками в різних сферах діяльності людини [1], [12].

Важливою особливістю нейронних мереж є їх здатність до навчання чи самонавчання. Під процесом навчання розуміють адаптацію параметрів та архітектури мережі для рішення поставленої задачі шляхом оптимізації обраного критерію якості [8]. Зазвичай до параметрів, що адаптуються відносять синаптичні вагові коефіцієнти нейронів, рідше – архітектуру мережі та параметри активаційної функції.

Всі алгоритми навчання можна поділити на дві групи: детерміновані та стохастичні. Для алгоритмів першої групи характерною особливістю є наявність жорстких правил дій при адаптації параметрів, а для другої притаманні дії, що підкоряються деякому випадковому процесу [15]. На поточний час відомо три парадигми навчання нейронних мереж, в основу яких покладено особливості машинного навчання: навчання з вчителем (supervised learning), навчання без вчителя (unsupervised learning) та навчання з підкріпленням (reinforcement learning).

На сьогоднішній день розроблено безліч типів штучних нейронів, архітектур ШНМ з різною топологією, алгоритмів навчання, що дозволяють робити налаштування синаптичних ваг за короткі проміжки часу і обробляти великі об’єми даних в потоковому режимі, постійно «донавчатися» на даних, що надходять, при цьому традиційні нейронні мережі в загальному випадку є системою типу «чорний ящик».

1.1.1 Поняття штучної нейронної мережі

Людський мозок можна описати як біологічну нейронну мережу – взаємопов’язану мережу нейронів, що передає складні схеми електричних сигналів. Дендрити отримують вхідні сигнали і, ґрунтуючись на цих входах, подають вихідний сигнал через аксон.

Штучна нейронна мережа – це парадигма обробки інформації, яка базується на біологічних системах, таких як мозок, інформація про процес. Ключовим елементом цієї парадигми є нова структура системи обробки інформації. Вона складається з великої кількості взаємопов’язаних елементів (нейронів), що працюють для вирішення конкретних завдань. Штучна нейронна мережа, як і люди, вчиться на прикладах. Вона налаштовується для конкретного завдання, такі як кластерування або класифікація даних, за допомогою навчального процесу. Навчання в такого роду системах включає в себе коригування синаптичних зв’язків, що існують між нейронами.

Нейронні мережі мають здатність витягувати релевантну інформацію зі складних або неточних даних і можуть використовуватися для отримання шаблонів і виявлення тенденцій, які занадто складні, щоб їх помітили люди або інші машинні методи. Навчену нейронну мережу можна розглядати як «експерта» в області, яка піддавалася аналізу. Потім отримані знання можуть бути використані для надання прогнозів з урахуванням нових ситуацій і відповідей на питання «що, якщо».

На ряду з цим є й інші переваги, такі як:

- адаптивне навчання: здатність навчитися виконувати завдання на основі даних, наданих для навчання або початкового досвіду;

- самоорганізація: мережа може створити свою власну організацію або подання інформації, яку вона отримує під час навчання;

- операції в режимі реального часу: обчислення мережі можуть виконуватися паралельно, розробляються і виробляються спеціальні

апаратні пристрої, які використовують цю можливість;

- стійкість до відмов за допомогою надлишкового інформаційного кодування.

Часткове руйнування мережі призводить до відповідного зниження продуктивності. Однак деякі мережеві можливості можуть зберігатися навіть при великому мережевому пошкодженні.

1.1.2 Складові штучної нейронної мережі

Базовим складовим штучної нейронної мережі є нейрон, головна функція якого, формувати вихідний сигнал, в залежності від сигналів, що надходять на його входи. В найпоширенішій конфігурації штучного нейрона вхідні сигнали множаться на певні ваги і обробляються адаптивним суматором, потім вихідний сигнал суматора надходить в перетворювач, де перетворюється активаційною функцією, і результат подається на вихід (в точку розгалуження).

Активаційна функція, яку також називають характеристичною – це нелінійна функція, що обчислює вихідний сигнал формального нейрона.

Часто використовувані активаційні функції:

- порогова передавальна функція;
- лінійний поріг;
- сигмоїдальна функція;
- випрямляч (ReLU).

На рисунках 1.1–1.4 наведені приклади активаційних функцій [5].

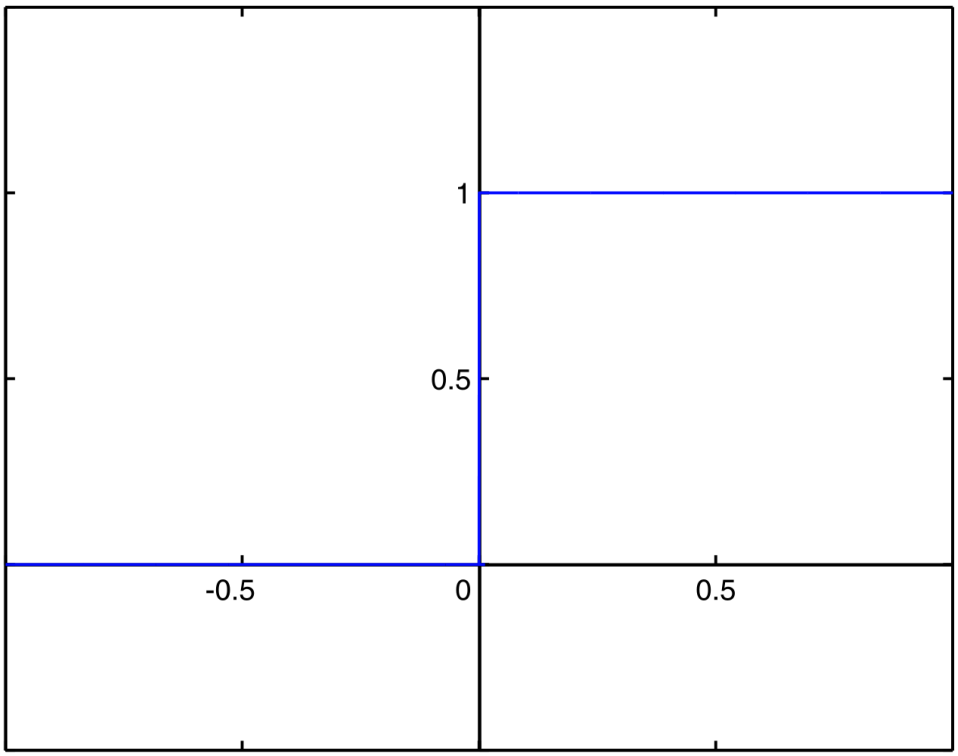

Рисунок 1.1 – Порогова функція

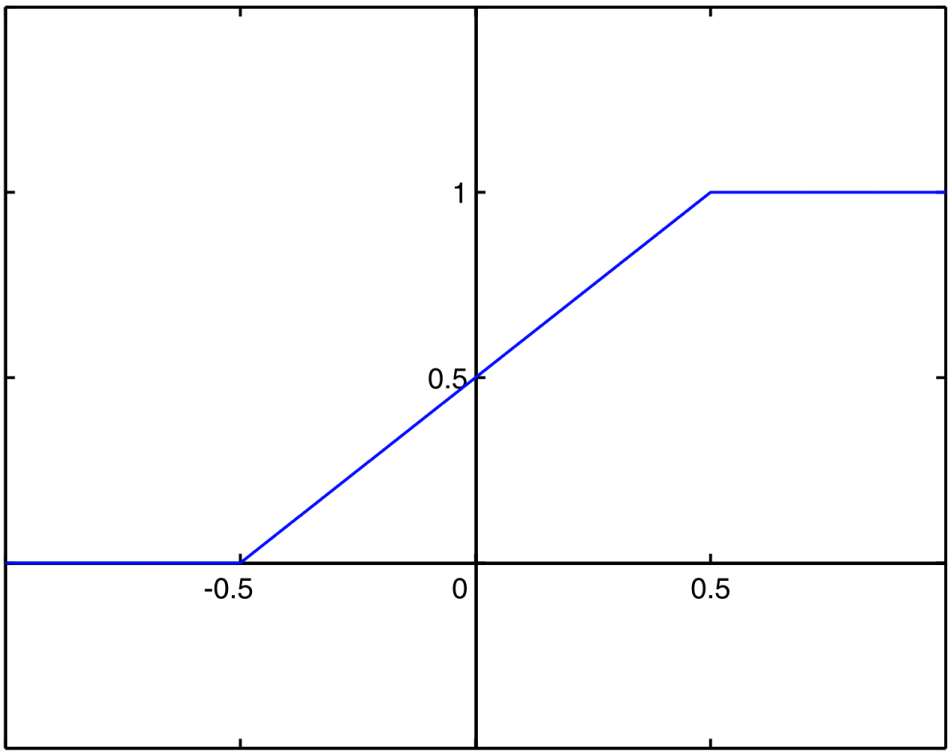

Рисунок 1.2 – Лінійно–порогова функція

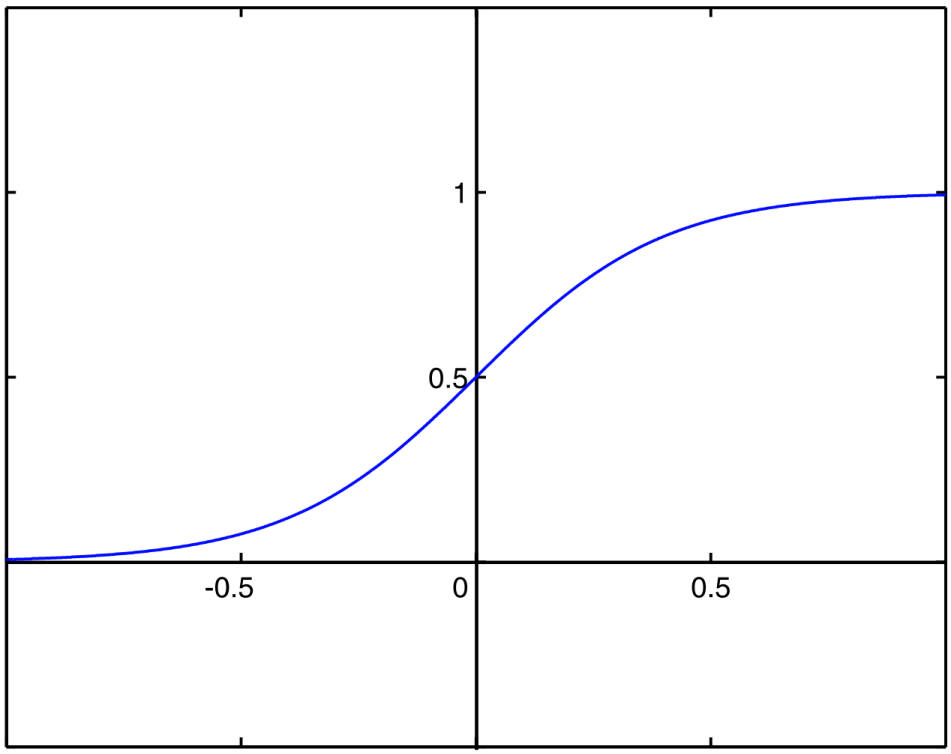

Рисунок 1.3 – Сигмоїдальна функція

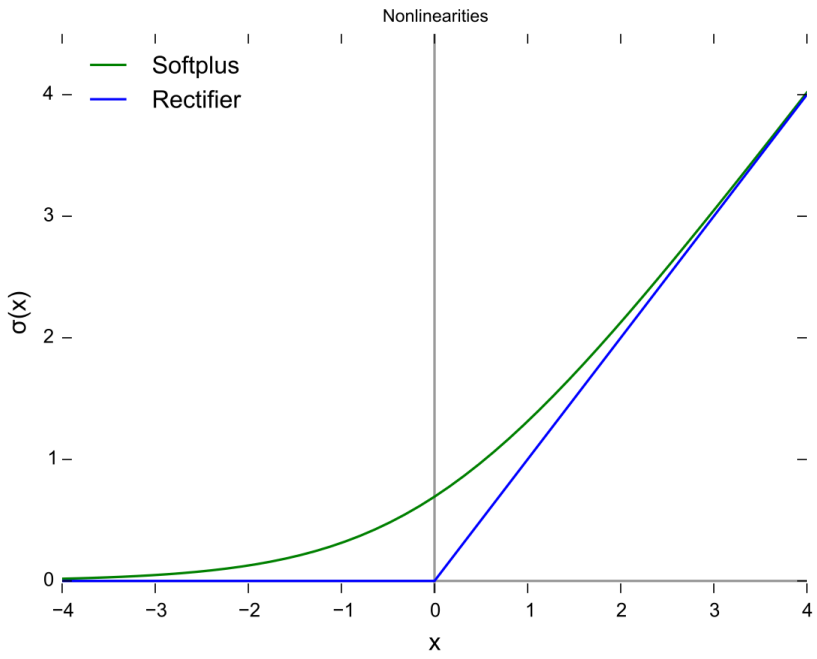

Рисунок 1.4 – Функції випрямляч і пом’якшення

Вибір активаційної функції визначається специфікою поставленого завдання або обмеженнями, що накладаються деякими алгоритмами навчання.

На рисунку 1.5 наведена загальна схема нейрона.

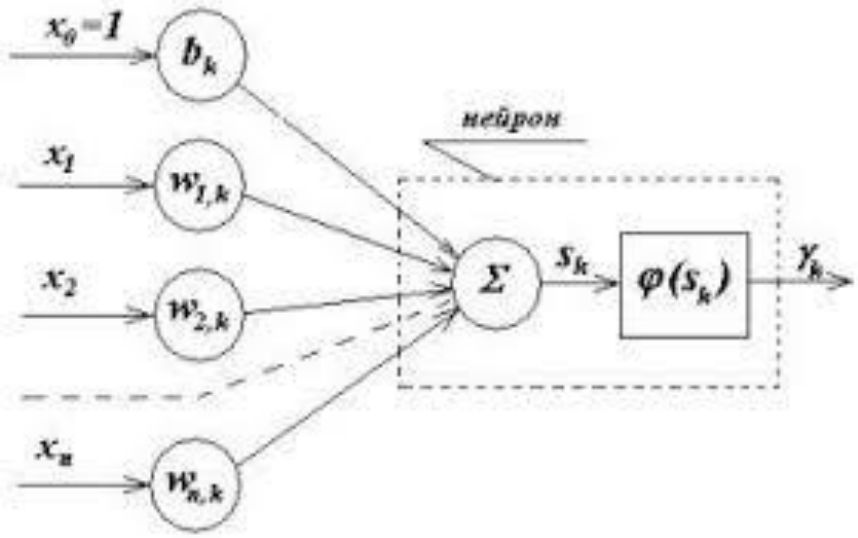

Рисунок 1.5 – Схема нейрона

Групи нейронів в свою чергу формують шари нейронної мережі. Найбільш поширений тип штучної нейронної мережі складається з трьох шарів: шар «вхідних» нейронів прив’язаний до шару «прихованих» нейронів, який підключений до шару «вихідних» нейронів.

Вхідний шар являє собою вихідну інформацію, яка подається в мережу. Кожен нейрон прихованого шару визначається діями блоків введення і вагами на з’єднаннях між вхідним і прихованим шарами. Поведінка вихідного шару залежить від активності прихованих шарів і ваг між прихованими і вихідними шарами.

На рисунку 1.6 представлена найпростіша штучна нейронна мережа.

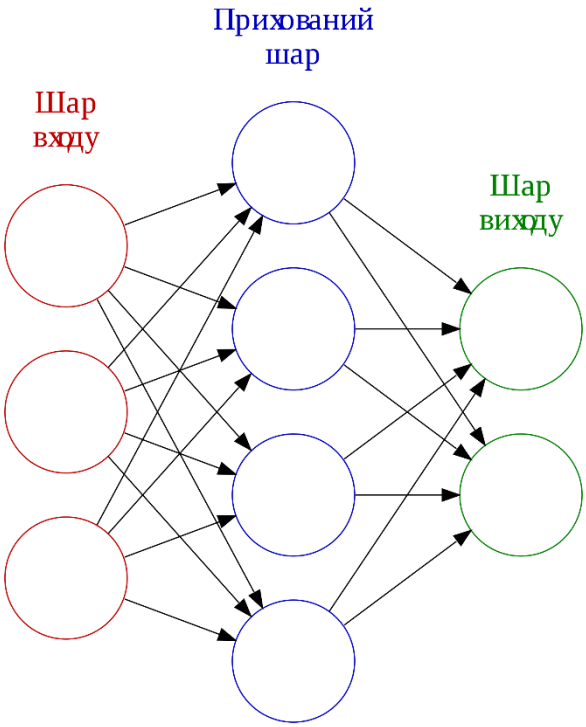

Рисунок 1.6 – Найпростіша штучна мережа

1.1.3 Навчання штучних нейронних мереж

Однією з основних особливостей інтелектуальних систем є їх здатність проводити узагальнення, ґрунтуючись на вже наявних і нових даних, або – іншими словами – їх здатність до навчання.

Навчання – це процес, в якому вільні параметри нейронної мережі налаштовуються за допомогою моделювання середовища, в яку ця мережа вбудована. Тип навчання визначається способом підлаштування цих параметрів [6]. Якщо ж говорити з точки зору математики, то навчання нейронних мереж можна визначити як параметричну задачу нелінійної оптимізації.

Дотепер не існує такого універсального алгоритму навчання, який підходив би для всіх нейронних мереж. Існують три парадигми навчання, на підставі яких проводиться налаштування параметрів нейронних мереж

будь-яких розмірів і архітектури [6], [10]:

- навчання з учителем (supervised learning). При застосуванні цієї парадигми вважається, що для кожного вектора $x(k)$ з заданої навчальної вибірки вже відомий бажаний результат $y(k)$. Таким чином, наприклад, навчається перцептрон. Налаштування параметрів мережі здійснюється з використанням навчального вектора і сигналу помилки. Одним з недоліків цієї парадигми є те, що для навчання мережі не завжди є достатня кількість образів з відповідями;

- навчання без учителя (unsupervised learning). При застосуванні цієї парадигми вважається, що бажані результати роботи нейронної мережі не відомі, а алгоритм навчання налаштовує ваги самостійно. Важливо відзначити те, що, на відміну від навчання «з учителем», в даному випадку вхідні образи, за якими проводиться навчання нейронної мережі, не марковані, тобто для них не заданий бажаний вихід системи. При цьому підході здійснюється пошук закономірностей в навчальній вибірці і проводиться групування схожих вхідних векторів за неявними ознаками. Відмінною особливістю цієї парадигми є наявність міри якості представлення, якому повинна навчитися система, а параметри нейронної мережі повинні оптимізуватися по відношенню до цієї міри;

- навчання «з підкріпленням» (навчання методом критики, reinforcement learning). При застосуванні цієї парадигми вважається, що є можливість оцінювати коректність роботи мережі і вказувати бажаний напрям навчання, тобто формувати сигнал підкріплення. Цей підхід можна позначити як проміжний між навчанням «з учителем» і навчанням «без вчителя».

Архітектура нейронних мереж пов’язана з алгоритмами, які використовуються для її навчання. Відповідно до класифікації, даної Хайкіним [14], виділяють 3 основні класи архітектур нейронних мереж:

- одношарові мережі прямого поширення. Вони містять єдиний шар нейронів. Інформація при цьому передається від вхідного шару на вихідний;

- багатошарові мережі прямого поширення. Вони містять один або кілька прихованих шарів. Зазвичай нейрони кожного з шарів мережі в якості вхідних сигналів використовують тільки вихідні сигнали нейронів попереднього шару;

- рекурентні мережі. Їх відмітною особливістю є наявність однієї або декількох зворотних зв'язків.

Також виділяють повнозв'язні нейронні мережі (коли всі вузли одного шару з'єднані з усіма вузлами суміжних шарів) і неповнозв'язні (коли деякі синаптичні зв'язки відсутні).

Згідно до Хайкіна [14] існує 5 основних моделей навчання:

- навчання на основі корекції помилок (реалізація методу оптимальної фільтрації);

- навчання з використанням пам'яті (з явним використанням навчальних даних);

- Хеббовське навчання (ця інформація базується на нейробіологічних принципах);

- конкурентне навчання (також засноване на нейробіологічних принципах);

- навчання за методом Больцмана (ця інформація базується на ідеях статистичної механіки).

Одним з найбільш поширених є навчання, засноване на корекції помилок. Воно використовує спеціальний механізм управління: до синаптичних ваг нейрона застосовується послідовність коригувань з метою наблизити вихідний сигнал системи до бажаного.

Коригування залежить від отриманого сигналу помилки, що видно з формули 1.1

$$e(k) = y(k) - \bar{y}(k), \tag{1.1}$$

де k – поточний дискретний час або порядковий номер спостереження з навчальної вибірки;

$y(k)$ – бажаний вихід нейрона;

$\bar{y}(k)$ – фактичний вихідний сигнал нейрона.

Ще однією часто використовуваною моделлю навчання є конкурентне навчання. Як випливає з назви, в даному випадку в нейронній мережі вихідні нейрони конкурують між собою за «право» бути активізованими. На відміну від нейронної мережі, заснованої на навчанні Хебба, в кожен момент часу в конкурентній мережі тільки один нейрон може бути активним. Його називають нейроном–переможцем, а принцип конкурентного навчання зазвичай формулюється як «переможець отримує все » (winner takes all) [7].

1.2 Навчання з підкріпленням

Як зазначалося вище, навчання з підкріпленням є парадигма навчання штучної нейронної мережі, за допомогою якої налаштовуються параметри. В машинному навчанні середовище зазвичай формулюється як марковський процес вирішування (МПВ), оскільки багато алгоритмів навчання з підкріпленням для цього контексту використовують методики динамічного програмування. Основна відмінність між класичними методиками й алгоритмами навчання з підкріпленням полягає в тому, що останні не потребують знання про МПВ, і вони орієнтовані на великі МПВ, в яких точні методи стають нездійсненними. Навчання з підкріпленням відрізняється від стандартного навчання з учителем тим, що пари правильних входів/виходів ніколи не представляються, а недостатньо оптимальні дії явно не виправляються. Крім того, є акцент на інтерактивній продуктивності, який включає знаходження балансу між дослідженням (незвіданої території) та використанням (поточного знання). Компроміс між дослідженням та використанням у навчанні з підкріпленням найретельніше вивчався через задачу багаторукого бандита та скінченні МПВ.

Базова модель навчання з підкріпленням складається з:

1) множини станів середовища S;

2) множини дій A;

3) правил переходу між станами;

4) правил, які визначають скалярну безпосередню винагороду (scalar immediate reward) переходу;

5) правил, які описують, що спостерігає агент.

Агент навчання з підкріпленням взаємодіє зі своїм середовищем у дискретні моменти часу. В кожен момент часу t агент отримує спостереження o_t, яке зазвичай включає винагороду r_t. Потім він обирає дію a_t з множини доступних дій, яка відтак відправляється до середовища. Середовище переходить до нового стану s_{t+1}, і визначається винагорода r_{t+1}, пов'язана з «переходом» (transition) (s_t, a_t, s_{t+1}). Метою агента навчання з підкріпленням є збирати якомога більше винагороди. Агент може обирати будь-яку дію як функцію історії, і може навіть робити свій вибір дії випадковим.

Потужним навчання з підкріпленням роблять дві складові: використання зразків для оптимізації продуктивності, та застосування наближень функцій, щоби мати справу з великими середовищами. Завдяки цим двом складовим навчання з підкріпленням можливо застосовувати у великих середовищах в будь-яких із наступних ситуацій:

- модель середовища є відомою, але аналітичний розв'язок відсутній;

- задано лише імітаційну модель середовища (предмет оптимізації на основі імітації);

- єдиним способом збирання інформації про середовище є взаємодія з ним.

1.3 Еволюційні алгоритми

Еволюційні алгоритми є напрямом в штучному інтелекті (розділ

еволюційного моделювання), що використовує і моделює біологічну еволюцію. Розрізняють різні алгоритми: генетичні алгоритми, еволюційне програмування, еволюційні стратегії, системи класифікаторів, генетичне програмування тощо. Всі вони моделюють базові положення в теорії біологічної еволюції - процеси відбору, мутації і відтворення. Поведінка агентів визначається довкіллям. Множину агентів прийнято називати популяцією. Така популяція еволюціонує відповідно до правил відбору відповідно до цільової функції, що задається довкіллям. Таким чином, кожному агентові (індивідуумові) популяції призначається значення його пристосовуваності в довкіллі. Розмножуються лише найпридатніші види. Рекомбінація і мутація дозволяють агентам змінюватись і пристосовуватися до середовища. Такі алгоритми відносяться до адаптивних пошукових механізмів.

1.3.1 Опис і види алгоритмів

Моделювання еволюції можна розділити на дві категорії:

1) системи, які використовують лише еволюційні принципи. Вони успішно використовувалися для завдань виду функціональної оптимізації і можуть легко бути описані на математичній мові. До них відносяться еволюційні алгоритми, такі як еволюційне програмування, генетичні алгоритми, еволюційні стратегії;

2) системи, які є біологічно реалістичніші, але які не виявилися корисними в прикладному сенсі. Вони більше схожі на біологічні системи і менш направлені на вирішення технічних завдань. Вони володіють складною і цікавою поведінкою, і, мабуть, незабаром отримають практичне вживання. До цих систем відносять так зване штучне життя.

Еволюційні алгоритми можна поділити на три групи:

- еволюційне програмування: фокусується більше на адаптації індивідів, аніж на еволюції генетичної інформації. Зазвичай, еволюційне

програмування застосовує безстатеве розмноження та мутації, тобто, внесення невеликих змін в поточний розв’язок та методи селекції основані на прямій конкуренції;

- еволюційні стратегії (ЕС): Важливою особливістю еволюційних стратегій є використання само-адаптивних механізмів для контролю процесу мутації. Ці механізми зосереджені не лише на еволюції шуканих розв’язків, а й на еволюції параметрів мутації;

- генетичні алгоритми (ГА): Основною особливістю генетичних алгоритмів є використання оператора рекомбінації як основного механізму пошуку. Це ґрунтується на припущенні, що частини оптимального розв’язку можуть бути знайдені незалежно та рекомбіновані для кращого розв’язку.

1.3.2 Генетичні алгоритми і генетичне кодування

Генетичні алгоритми (ГА) є класом алгоритмів обчислювального пошуку, натхнених еволюцією в природі. Завдання ГА в пошуку в просторі параметрів набору параметрів, що оптимізують деякий критерій пристосовуваності. ГА заснований на природному відборі, де більш підходящі особини відбираються для продовження роду. В ГА зберігається сукупність рішень, де кожне рішення представляє точку в просторі пошуку. Зазвичай простір пошуку розуміється як скінченновимірний простір параметрів. Ці параметри можуть бути двійковими, дискретними, реальними або будь-який іншими доступними для пошуку схемою кодування. Рядок параметрів називається генотипом. Генотип перетворюється в фенотип через процедуру генезису. Фенотип потім оцінюється за деякими критеріями придатності. Фенотип з більш високою пристосованістю можуть схрещувати свої геноми, щоб створити потомство в надії, що комбінація двох добрих наборів параметрів справить ще кращу параметризацію. Мутації, що виникають при розмноженні, викликають випадкові зміни параметрів. Мутації гарантують, що пошук охоплює нові

області простору пошуку.

Важливо відзначити, що загальна тема ГА може істотно відрізнятися. Спосіб кодування генотипу може суттєво вплинути на кількість оцінок, необхідних для пошуку рішення, і може навіть зробити неможливим пошук певних рішень. Одна з областей, де алгоритм NEAT відрізняється від стандартного ГА, полягає в тому, що NEAT може змінювати простір, в якому він шукає, додаючи або віднімаючи вимірювання з цього простору. Іншими словами, NEAT може змінювати кількість параметрів, які він намагається оптимізувати. Таким чином, принципи стандартного ГА, які працюють в кінцевих просторах, не обов'язково застосовуються до форми пошуку NEAT, і NEAT може знаходити як малі, так і великі рішення, ніж алгоритми, які шукають простір фіксованого числа вимірювань.

ГА особливо корисний в задачах з невеликим знанням предметної області, тому що вони не вимагають апріорного аналізу простору гіпотез або предметної області. Наприклад, вони можуть шукати нейронні мережі в завданнях з розрідженим підкріпленням, навіть коли внесок будь-якого інформації в досягнення мети невідомий. ГА також корисний для задач з локальними мінімумами на поверхні помилок, які можуть захоплювати методи градієнтного спуску. Оскільки зразок ГА має кілька точок на поверхні помилок, вони можуть одночасно шукати різні частини простору параметрів.

У NEAT ваги зв'язків кодуються в геномі як дійсні числа. Голланд [9] показав за допомогою теореми схем, що генетичні алгоритми мають тенденцію сходитися до кращих рішень. Теорема схеми була заснована на двійковому кодуванні, але Райт [34] розширив її, щоб охопити кодування дійсних чисел. Райт показав, що реальні кодування чисел краще для деяких завдань безперервної оптимізації. Еррера і Лозано [35] представили великий огляд генетичних операторів над реальними кодованими хромосомами, включаючи різні типи схрещування. Дійсні числа є відповідною і добре зрозумілою формою представлення ГА в безперервних областях.

1.4 Застосування еволюційних алгоритмів для створення штучної нейронної мережі

Задача навчання нейронних мереж зводиться до налаштування вагових коефіцієнтів і тут найбільш прийнятним можна вважати генетичний алгоритм.

Генетичний алгоритм є ітераційним, і дає лише наближений розв'язок, що компенсується областю його застосування і швидкістю обчислень при обмежених обчислювальних ресурсах. Він може застосовуватися для підстроювання ваг схованих і вихідних шарів при фіксованому наборі зв'язків і широко використовується в задачах оптимізації і навчання нейронних мереж. У плані математичної моделі – це алгоритм знаходження глобального екстремуму багатоекстремальної функції, що полягає в паралельній обробці множини альтернативних розв'язків. При цьому пошук концентрується на найбільш перспективних із них. Алгоритм використовує наступні визначення:

- ген – ваговий коефіцієнт нейронної мережі;
- хромосома – набір генів (тобто вагових коефіцієнтів нейронної мережі, зчитуваних у певному порядку зверху вниз, зправа наліво); кожна хромосома є можливим розв'язком (тобто таким набором вагових коефіцієнтів, які краще підходять для поведінки агента у задачі);
- популяція – безліч хромосом, варіантів наборів вагових коефіцієнтів;
- епоха – ітерація, відповідна до створення нового покоління хромосом.

Хромосоми є основними сутностями, над якими в певному порядку в межах однієї епохи проводяться наступні операції:

- схрещування – створення з певним ступенем імовірності (p_c) нової хромосоми з генів двох інших і додавання її в популяцію;
- мутація – зміна з певним ступенем імовірності (p_m) значення

довільного гена будь-якої хромосоми і додавання її в популяцію;

- пристосування – видалення з популяції хромосом (тобто наборів вагових коефіцієнтів), що показали гірший результат [15].

Мутація вирішує одну із проблем, яка присутня у навчанні багатошарової нейронної мережі методом зворотного поширення помилки. Вона необхідна для виходу популяції з локального екстремума і сприяє захисту від передчасної збіжності. На рисунку 1.7 показана нейронна мережа, вагові коефіцієнти якої (роблячи обхід зверху вниз, зправа наліво) заповнюються хромосоми. Таким чином, хромосома являє собою набір генів – вагових коефіцієнтів [12].

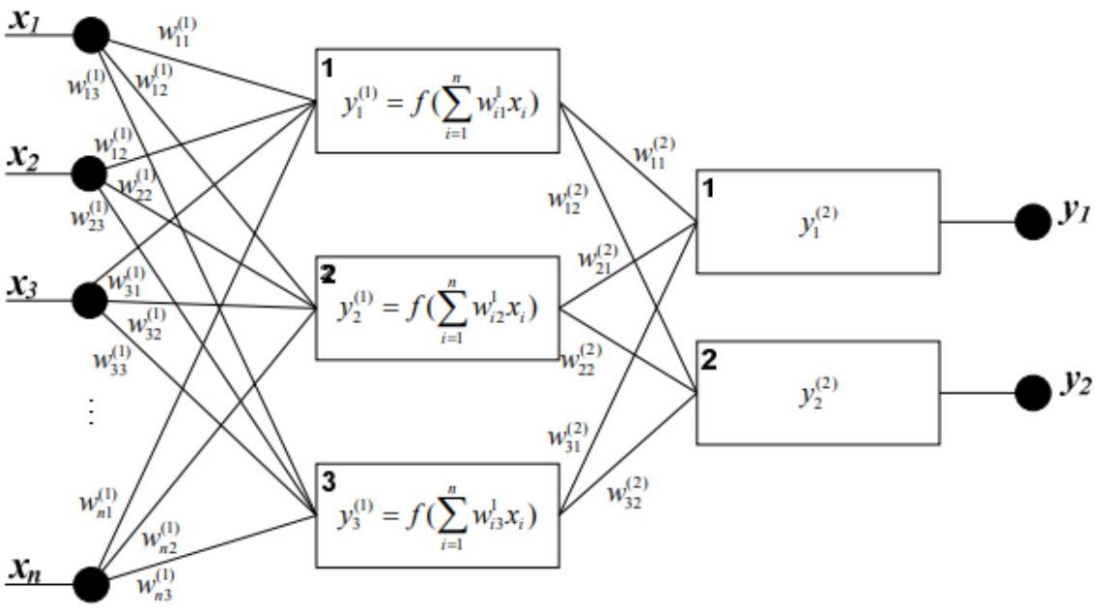

Рисунок 1.7 – Двошарова нейронна мережа з ваговими коефіцієнтами – генами хромосом

Алгоритм навчання ШНМ генетичним алгоритмом є наступним:

1) створити набір хромосом (популяцію), що складається із генів, рівних у кожній хромосомі числу вагових коефіцієнтів нейронної мережі;

2) заповнити кожну хромосому популяції генами з випадковими значеннями, що не суперечать умові задачі;

3) перевірити чи операції зроблені для всіх хромосом популяції і перейти до кроку 8 у позитивному випадку;

4) якщо ні, то вибрати з популяції пари хромосом з імовірністю p_c і зробити схрещування, і результуючу нову хромосому додати в популяцію;

5) вибрати з популяції хромосому з імовірністю p_m і зробити мутацію генів, і результуючу нову хромосому додати в популяцію;

6) перевірити кожну хромосому на відповідність розв'язку задачі і розташувати хромосому в порядку убування відповідності;

7) вилучити з популяції хромосому, порядковий номер якої перевищує первісне число хромосом у популяції і перейти до кроку 3;

8) перевірити чи відповідає хромосома 1 у популяції розв'язку задачі і зупинити розв'язок у позитивному випадку.

Популяція хромосом еволюціонує: при схрещуванні народжуються нащадки хромосом, які відрізняються від своїх батьків. Зміна генома походить від батька до нащадка. Кожен ген w_i нащадка задається як випадкова величина x, рівномірно розподілена на інтервалі $[w_{p1}, w_{p2}]$, де w_{p1} – відповідний ген батька 1, і w_{p2} – ген батька 2. Геном агента S складається з хромосом $S = (w_b, w_l)$. Перша хромосома w_b містить ваги синапсів нейронної мережі блоку поведінки w_{ij}. Друга хромосома w_l містить ваги синапсів нейронної мережі блоку оцінки дій. При схрещуванні змінюються тільки хромосоми блоку поведінки, так як мається на увазі, що система цінностей кожного агента незмінна.

1.5 Задача комівояжера

Задача комівояжера полягає у знаходженні найвигіднішого маршруту, що проходить через вказані міста хоча б по одному разу. В умовах завдання вказуються критерій вигідності маршруту (найкоротший, найдешевший, сукупний критерій тощо) і відповідні матриці відстаней, вартості тощо. Зазвичай задано, що маршрут повинен проходити через кожне місто тільки

один раз, в такому випадку розв’язок знаходиться серед гамільтонових циклів. Існує маса різновидів узагальненої постановки задачі, зокрема геометрична задача комівояжера (коли матриця відстаней відображає відстані між точками на площині), трикутна задача комівояжера (коли на матриці вартостей виконується нерівність трикутника), симетрична та асиметрична задачі комівояжера. Задача комівояжера є NP-повною. Всі ефективні (такі, що скорочують повний перебір) методи розв’язання задачі комівояжера – евристичні. У більшості евристичних методів знаходиться не найефективніший маршрут, а наближений розв’язок.

Для можливості застосування математичного апарату для розв’язання проблеми, її слід представити у вигляді математичної моделі. Проблему комівояжера можна представити у вигляді моделі на графі, тобто, використовуючи вершини та ребра між ними. Таким чином, вершини графу відповідають містам, а ребра (i, j) між вершинами i та j сполучення між цими містами. У відповідність кожному ребру (i, j) можна зіставити вагу $c_{ij} \geq 0$, яку можна розуміти як, наприклад, відстань між містами, час або вартість подорожі. Маршрутом (також гамільтоновим маршрутом) називається маршрут на цьому графі до якого входить по одному разу кожна вершина графа. Задача полягає у відшуканні найкоротшого маршруту.

З метою спрощення задачі та гарантії існування маршруту, зазвичай вважається, що модельний граф задачі є повним, тобто, що між довільною парою вершин існує ребро. Це можна досягти тим, що в тих випадках, коли між окремими містами не існує сполучення, вводити ребра з максимальною вагою (довжиною, вартістю тощо). Через велику довжину таке ребро ніколи не потрапить до оптимального маршруту, якщо він існує. На рисунку 1.8 зображено приклад симетричної задачі для описаної вище математичної моделі.

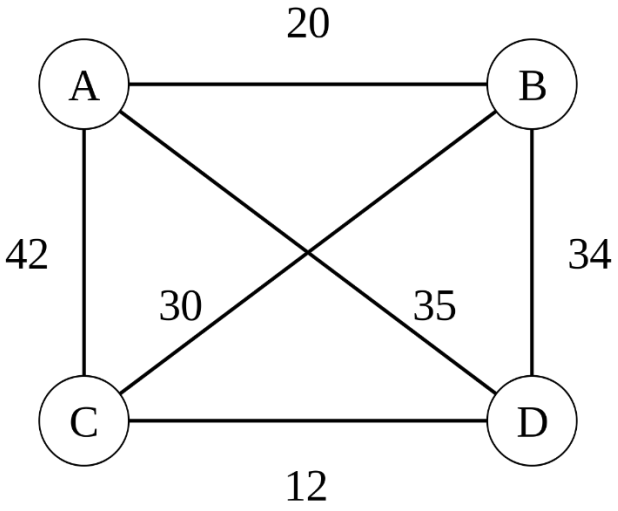

Рисунок 1.8 – Симетрична задача для чотирьох міст

Оскільки комівояжер в кожному з міст постає перед вибором наступного міста з тих, що він ще не відвідав, існує $(n-1)!$ маршрутів для асиметричної та $(n-1)!/2$ маршрутів для симетричної задачі комівояжера. Таким чином, розмір простору пошуку залежить над-експоненційно від кількості міст. Різні варіанти задачі комівояжера (метричний, симетричний та асиметричний) є NP-еквівалентними. Відповідно до поширеної але недоведеної гіпотези про нерівність класів складності P та NP не існує детермінованої машини Тюринга, здатної знаходити розв’язки екземплярів задачі за поліноміальний час в залежності від кількості міст.

Відомі методи розв’язання поділяють на дві групи, що можна комбінувати. Точні методи знаходять, маючи достатньо часу, гарантовано оптимальний шлях. Евристичні методи знаходять, часто за коротший час, гарні розв’язки, що, в загальному випадку, можуть бути гіршими за оптимальні.

Можна знайти точний розв’язок задачі комівояжера, тобто, обчислити довжини всіх можливих маршрутів та обрати маршрут з найменшою довжиною. Однак, навіть для невеликої кількості міст в такий спосіб задача

практично нерозв’язна. Для простого варіанта, симетричної задачі з n містами, існує $(n-1)!/2$ можливих маршрутів, тобто, для 15 міст існує 43 мільярдів маршрутів та для 18 міст вже 177 більйонів. Те, як стрімко зростає тривалість обчислень можна показати в наступному прикладі. Якщо існував би пристрій, що знаходив би розв’язок для 30 міст за годину, то для двох додаткових міст в тисячу раз більше часу; тобто, більш ніж 40 діб. Методи дискретної оптимізації, зокрема гілок і меж дозволяють знаходити оптимальні або приблизні розв’язки для достатньо великих задач.

Для пришвидшення пошуку прийнятних маршрутів можна використовувати евристики, що, в загальному випадку, не гарантують точності знайдених розв’язків. В залежності від того, чи обчислює евристика новий маршрут, чи намагається покращити вже існуючий, евристики поділяють на конструктивні та ітеративні евристики, на останні з яких слід звернути увагу. Ітеративні евристики (також після оптимізаційні евристики) намагаються шляхом деяких змін скоротити вже обчислений маршрут. Прикладом такої евристики є k-opt-евристики, що систематично видаляють з маршруту групи з k ребер і замінюють їх іншими k ребрами отримуючи новий маршрут. Оскільки повний перебір можливих варіантів дорівнює кількості можливих маршрутів, на практиці обмежують k щонайбільше до 5. При цьому, випробовуються всі варіанти заміни двох та трьох ребер, уникаючи перебору з більшою кількістю ребер через значні витрати часу.

1.6 Постановка завдання дослідження

Ґрунтуючись на вищевикладеному, формально завдання дослідження зводиться до створення штучної нейронної мережі, аналізу методів навчання мереж та існуючих видів і архітектур, проведення її навчання та оптимізації, реалізації еволюційного алгоритму (зокрема генетичного алгоритму з використанням інбридингу), а саме ES-HyperNEAT,

дослідження і використання навчання з підкріпленням. Прикладом застосування для цієї системи є розробка системи пошуку найвигіднішого шляху для заданих міст з заданою матрицею витрат. Потрібно дослідити обрану штучну нейрону мережу для знаходження кількості вхідних нейронів, створити системи для імітації задачі пошуку найвигіднішого маршруту для користувача на основі попередньо введених даних про міста і матриці затрат, реалізувати генетичний алгоритм у парі зі штучною нейронною мережею, навчити цю мережу на основі навчання з підкріпленням та проходу багатьох поколінь з використанням інбридингу та природного відбору, протестувати отриману модель в різних умовах, проаналізувати отримані результати та порівняти з існуючими реалізаціями.

Для досягнення поставленої мети необхідно розглянути усі наступні питання:

- провести аналіз існуючих методів еволюційних алгоритмів для вирішення задачі навчання та побудови архітектури штучної нейронної мережі;

- розробити метод і нейронну мережу для створення системи пошуку найвигіднішого маршруту між містами;

- провести імітаційне моделювання та порівняльний аналіз розробленої штучної нейронної системи з існуючими на даний момент іншими системами для описаної задачі комівояжера.

Як результат, ми зможемо впевнитися, що даний підхід показує гарні результати у порівнянні з існуючими методами і реалізаціями, і може бути придатний до подальшого розвитку і дослідження для пристосування для вирішення різних реальних проблем світу, які мають жорсткі вимоги до тривалості роботи алгоритму, складності отриманої мережі і можливості подальшого навчання з використанням попередньої історії вагових коєфіцієнтів синапсів нейронної мережі.

2 НЕЙРОЕВОЛЮЦІЯ

Нейроеволюція (Neuroevolution, NE) є комбінацією нейронних мереж і генетичних алгоритмів, де нейромережі є фенотипом, що оцінюється. Генотип це компактне представлення, яке може бути переведено в штучну нейронну мережу. NE здійснює пошук нейронних мереж, які оптимізують деяку діяльність. NE може шукати практично будь-яку нейронну мережу, будь то прості, прямі, повторювані або навіть адаптивні мережі.

Хромосоми в NE можуть представляти будь-який параметр нейронних мереж, від вагових зв’язків і топології до функцій активації. Оскільки вибір кодування впливає на пошуковий простір рішень, він є ключовим аспектом проектування системи NE.

Хоча деякі методи NE розвивають тільки вагові зв’язки мережі, Topology and Weight Evolving Neural Networks (TWEANNs) розвивають як вагові коефіцієнти, так і мережеві топології [23]. Методи фіксованої топології вимагають, щоб людина обирала правильну топологію для проблеми. На відміну від цього, TWEANNs можуть самостійно виявити потрібну топологію. Крім того, еволюція топології може бути використана для підвищення ефективності шляхом збереження розмірів мереж якомога менше, що є стратегією, що застосовується NEAT. Таким чином, TWEANNs є важливим класом методів NE, і вони стикаються з кількома специфічними проблемами і труднощами, які не можуть вирішити методи NE з фіксованою топологією.

2.1 TWEANNs

Зараз, розглядаючи даний тип мереж, потрібно дати деяку інформацію про кодування TWEANNs, а потім переглянути декілька прототипних методів, які були розроблені для одночасного розвитку топології та ваг ANN.

Перед розглядом методів TWEANNs важливо відзначити, як пряме кодування та непряме кодування в TWEANNs відрізняються. Геноми в TWEANNs кодують як топологію, так і ваги з'єднання мережі. Розробники TWEANNs повинні вирішити, чи слід кодувати довільні топології прямим або непрямим чином. Прямі схеми кодування вказують в геномі кожне з'єднання і вузол, який з'явиться в фенотипі [27]. З іншого боку, непрямі кодування зазвичай тільки визначають правила побудови фенотипу [28]. Ці правила можуть бути специфікаціями шарів або правилами зростання через поділ клітин. Основна ідея непрямого кодування полягає в тому, що кожне з'єднання і вузол не вказані в геномі, хоча вони можуть бути отримані з нього. TWEANNs, які використовують непряме кодування, включають методи штучного ембріогенезу (artificial embryogeny , AE), які розвивають фенотипи, що починають з малої ембріональної стартової структури.

2.1.1 Непряме кодування

Метод клітинного кодування (Cellular Encoding, CE) Груа [16] є системою AE, яка використовує непряме кодування. У CE геноми є програмами, написаними спеціалізованою мовою трансформації графа, який називається деревом граматики. Перетворення мотивовані природою в тому, що вони вказують поділ клітин. Різні види з'єднань можуть бути результатом поділу, тому можливі кілька видів поділу клітин. CE має одну велику перевагу: його генетичні представлення компактні. Гени в CE можуть бути повторно використані кілька разів під час розробки мережі, кожен раз запитуючи поділ клітини в іншому місці. Повторне використання генів заощаджує простір в геномі, оскільки кожне з'єднання і вузол в мережі не повинні бути явно вказані в геномі. CE є знаковою системою, яка показує, що правила розвитку можуть кодувати мережі з однієї клітини, так як організми в природі починають з єдиної клітини, яка розрізняється, оскільки вона розпадається на більше клітин, які в свою чергу розщеплюються на ще

більше клітин.

Рисунок 2.1 зображує, як граматичне дерево перетворюється в розроблену мережу. Дерево граматики містить інструкції розвитку на кожному вузлі, які розділяють одну клітинку на дві клітинки, змінюють значення зв'язків між клітинами і видаляють існуючі зв'язки між клітинами. Розвиток клітини в CE починається з різних частин дерева граматики одночасно. Голова читання для кожної клітинки вказує з якої частини граматичного дерева він читає. Коли комірка зустрічається з кінцевою командою, її стан завершується, і він припиняє зчитування. CE використовує чергу клітин First In First Out (FIFO) для того, щоб відслідковувати, які клітини виконуються в даний час інструкціями, і в якому порядку вони повинні виконуватися.

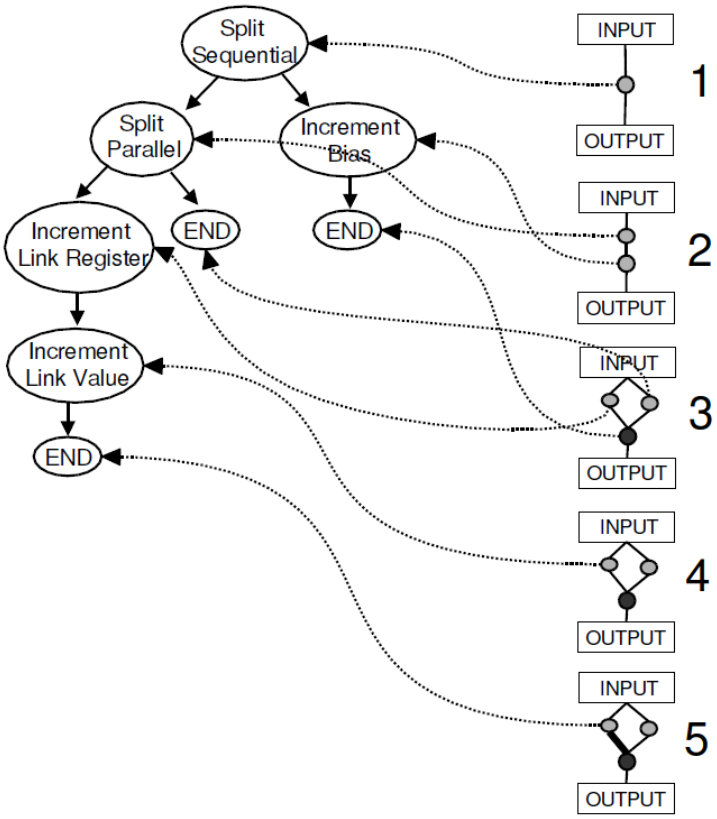

Рисунок 2.1 – Приклад клітинного кодування (CE)

Клітина у передній частині черги виконує інструкцію, на яку вказує її читання, переміщує голову до наступної команди у дереві граматики, а потім переходить до кінця черги. Іноді клітини зустрічаються з інструкціями для поділу (існує безліч методів поділу клітин, таких як паралельний і послідовний поділ), і в цьому випадку початкова клітина переміщує голову читання вниз по лівому піддереві, і нова клітина переміщує свою голову вниз до правого піддеревом. Таким чином, клітинні ділення дозволяють різним лініям клітин стежити за різними шляхами розвитку.

2.2 Початкові популяції та топологічні нововведення

Багато методів TWEANNs слідують філософії, що початкова популяція повинна починатися з випадкових топологій. Випадкова початкова популяція гарантує, що топології різноманітні з самого початку. Однак випадкові початкові популяції викликають кілька проблем для TWEANNs, і тому можуть бути не кращим способом почати еволюцію. Рандомізація початкових топологій може призвести до створення нездійсненних мереж. Однак є більш тонка проблема з запуском випадковим чином, що є більш серйозною. Бажано знайти мінімальні рішення, так як вони зменшують кількість шуканих параметрів. Запуск з випадковими топологіями не призводить до мінімальних рішень, так як населення вже має багато непотрібних вузлів і з’єднань. Жоден з цих вузлів або сполук не повинен був витримувати єдиної оцінки, тобто їх конфігурація не виправдана. Зусилля з мінімізації мереж повинні бути витрачені на позбавлення від апарату, якого не повинно було бути в першу чергу, і рекомбінація різних топологій не підштовхує до мінімізації сама по собі. Оскільки у створенні великих мереж немає витрат на пристосованості, вони будуть домінувати до тих пір, поки у них є висока пристосованість.

Можна включити розмір мережі в функцію пристосованості, і деякі TWEANNs дійсно роблять це [5]. Однак, якби населення починало без

прихованих вузлів і зростало тільки в міру того, як воно вигравало від рішення, не було б необхідності в спеціальній адаптації для мінімізації мереж. Насправді, якщо б населення починалося з мінімальної структури, а не з випадковою, було б безглуздо додавати штрафи до функції придатності для додаткової структури. Починати з мінімальною населеністю і зростаючою структурою важливий принцип конструкції в цілому. Ніби для того, щоб підтвердити, що, починаючи з випадкових топологій, призводить до надмірно великим мережам, Pujol і Poli [17] додали алгоритм обрізки в свою систему PDGP «щоб знайти рішення з мінімальною складністю». PDGP запускає цей абсолютно окремий алгоритм обрізки після того, як знаходить рішення. Очевидно, PDGP поодинці не розвиває мінімальні мережі. Чому так важливо розвивати мінімальні мережі? Дослідники TWEANNs зазвичай слідують аргументу Чжана і Муленбейна [19], які стверджують: «Пошук мінімальної мережі для конкретних додатків важливий, тому що швидкість і точність навчання залежать від складності мережі». Хоча в цьому твердженні і є частка правди, воно злегка вводить в оману. Зрештою, оптимізація ваги виконується до того часу, коли рішення виводиться з TWEANNs, так що яке навчання залишилося зробити? Деяка локальна оптимізація у ваговому просторі дійсно може бути простішою (можливо, за допомогою backpropagation) з мінімальною архітектурою, але ця перевага здається завищеною, враховуючи, що більшість TWEANNs представлені як альтернатива backpropagation.

Що дійсно важливо в топології, так це не остаточна топологія, а швидше проміжні топології, які система повинна шукати, щоб дістатися до остаточного рішення. Кількість зв’язків, зазначених хромосомою, вимірює розмірність досліджуваного в даний час простору пошуку. Пошук в просторах більш високих вимірів займає більше часу, ніж у просторах більш низьких вимірів. Якщо пошук може бути збережений до мінімальної розмірності протягом всього прогону, метод буде шукати в найпростіших можливих просторах.

3 НЕЙРОЕВОЛЮЦІЯ НАРОСТАЮЧОЇ ТОПОЛОГІЇ

Метод HyperNEAT, що дозволяє вчитися з геометрії, є продовженням оригінального алгоритму NEAT, який розвиває ANN за допомогою прямого кодування. Для подальшого розуміння почнемо з суті алгоритму NEAT.

3.1 Алгоритм NEAT

Спочатку метод NEAT був розроблений для еволюції ANN для вирішення складного контролю та послідовних завдань прийняття рішень і виявився успішним у широкому діапазоні сфер [1], [6], [10], [13]. Еволюціоновані ANN контролюють агентів, які обирають дії на основі своїх сенсорних входів. NEAT відмінний від багатьох попередніх методів, що розвивали нейронні мережі, тобто методів нейроеволюції, які традиційно розвивають або мережі з фіксованою топологією [10], [18], або довільні мережі з випадковою топологією [2], [3], [6]. Натомість NEAT починає еволюцію із сукупності невеликих, простих мереж та ускладнює топологію мережі на різноманітні види протягом поколінь, що призводить до зростання слкадності поведінки. Подібний процес поступового додавання нових генів був підтверджений в природній еволюції [4], [6] і показав, що він покращує адаптацію в кількох попередніх еволюційних [64] і нейроеволюційних [25] підходах. Однак ключова особливість, яка відрізняє NEAT від попередніх робіт в області комплексіфікації – це її унікальний підхід до збереження здорового різноманіття комплексифікації структури одночасно, як розглядається в цьому розділі.

Метод NEAT базується на трьох ключових ідеях. По-перше, щоб дозволити мережевим структурам ускладнюватись протягом поколінь, потрібен метод, щоб відстежувати, який ген є яким. В іншому випадку в наступних поколіннях ми не бачимо те, яка особина сумісна з якою в популяції різноманітних структур, або як їх гени слід поєднувати для

отримання потомства. NEAT вирішує цю проблему, присвоюючи унікальне історичне позначення кожному новому фрагменту мережевої структури, що з’являється через структурну мутацію. Історичне маркування – це число, присвоєне кожному гену, що відповідає порядку його появи протягом еволюції. Ці цифри успадковуються під час кросинговеру без змін і дозволяють NEAT виконувати кросовер між різними топологіями без необхідності коштовного топологічного аналізу.

По-друге, історичні маркування дозволяють системі розділити популяцію на види на основі того, як подібні мережі топологічно. Таким чином, особи змагаються насамперед у власних «нішах», а не серед населення в цілому. Оскільки додавання нової структури часто спочатку невигідне, то поділ означає, що унікальні топологічні інновації захищаються і, отже, встигають оптимізувати свою структуру перед тим, як конкурувати з іншими «нішами» населення.

По-третє, багато систем, що розвивають мережеві топології та ваги, починають еволюцію із сукупності випадкових топологій [6], [22]. На відміну від цього, NEAT починає з рівномірної популяції простих мереж без прихованих вузлів, що різняться лише в їх початкових випадкових вагах. Через видоутворення нові топології поступово накопичуються протягом еволюції, що дозволяє представляти різноманітні та складні моделі фенотипу. Ніяких обмежень щодо розміру, до якого можуть зростати топології. Нова структура вводиться поступово, коли відбуваються структурні мутації, і виживають лише ті структури, які виявляються корисними за допомогою оцінок точності і потужності. Таким чином, NEAT шукає компактну, відповідну топологію шляхом поступового збільшення складності існуючої структури.

Приклад відображення і структурних мутацій відображено на рисунках 3.1 і 3.2.

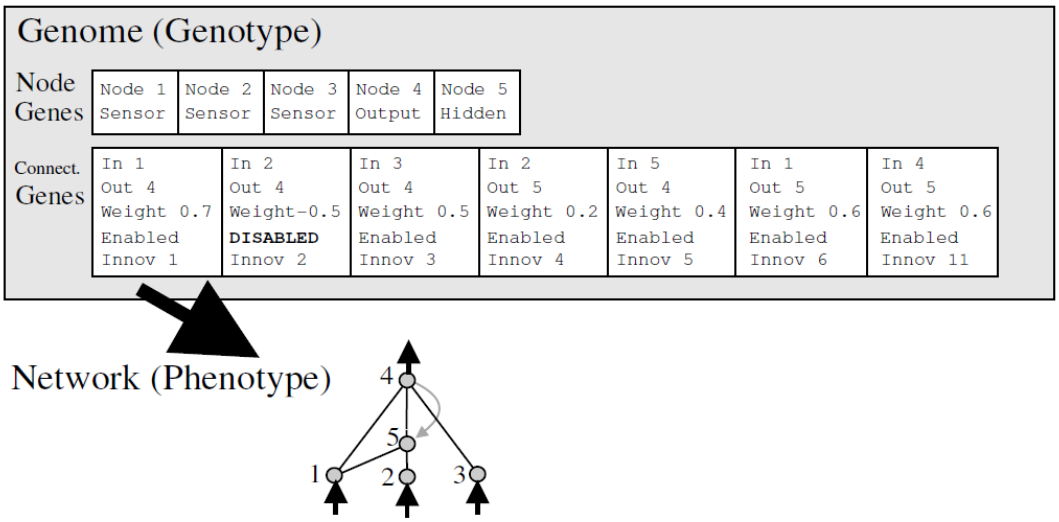

Рисунок 3.1 – Приклад відображення NEAT генотипу на фенотип

Мутація в NEAT може змінити як вагу з'єднання, так і мережеві структури. Ваги з'єднань мутують, як і в будь-якій системі NE; кожна вага з'єднання обирається з фіксованою ймовірністю шляхом додавання числа з плаваючою комою, вибраного з рівномірного розподілу додатніх і від'ємних значень. Структурні мутації, що становлять основу комплексифікації, відбуваються двома шляхами. Кожна мутація збільшує розмір генома, додаючи гени. В мутації «add connection» додається один новий ген з'єднання, що з'єднує два раніше незв'язаних вузла. В мутації «add node» існуюче з'єднання розділяється, а новий вузол поміщається туди, де раніше було старе з'єднання. Старе з'єднання розірване, і в геном додаються два нових з'єднання. З'єднанню між першим вузлом в ланцюзі і новим вузлом присвоюється вага одиниці, а з'єднанню між новим вузлом і останнім вузлом в ланцюзі присвоюється ту ж вагу, що і поділюваному з'єднанню. Поділ з'єднання таким чином вводить нелінійність (тобто сигмовидна функція), де раніше її не було. Оскільки новий вузол відразу ж увійшов в мережу, його вплив на функцію пристосовуваності можна оцінити відразу. Існуюча мережева структура не руйнується і виконує ту ж функцію, в той час як нова структура дає можливість зупинитися на початковій поведінці.

Завдяки мутації геноми в NEAT поступово стануть більше. Результатом будуть геноми різних розмірів, іноді з різними зв'язками в одних і тих же позиціях. Кросинговер повинен бути здатний рекомбінувати мережі з різними топологіями, що може бути складно [3].

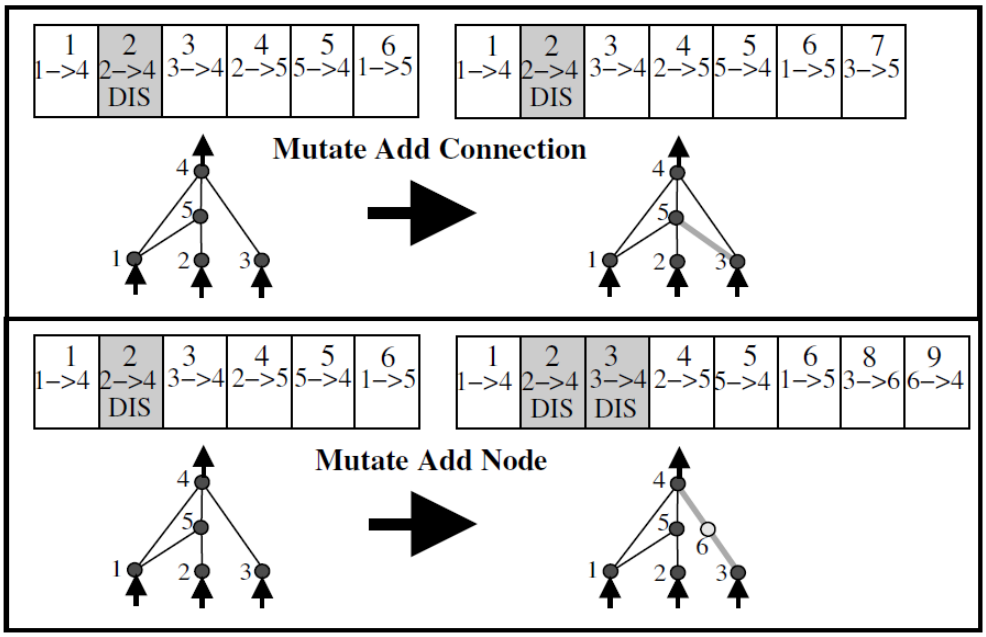

Рисунок 3.2 – Два типи структурних мутацій в NEAT

Одна з можливих проблем полягає в тому, що одна і та ж структурна інновація отримає різні номери інновацій в одному і тому ж поколінні, якщо вона відбудеться випадково більше одного разу. Однак, зберігаючи список інновацій, які відбулися у поточному поколінні, можна гарантувати, що, коли одна і та ж структура виникає більше одного разу через незалежні мутації в тому ж поколінні, кожної ідентичної мутації присвоюється один і той же номер інновації. Багато експериментів [15], [19] показали, що для запобігання вибуху числа інновацій досить скидати список кожне покоління, а не зберігати зростаючий список мутацій протягом всієї еволюції. Завдяки інноваціям отриманих номерів, тепер система точно знає, які гени збігаються з якими (рисунок 3.3). Гени, які не мають того ж

історичного маркування, що і будь-який інший ген в іншому батьківському геномі, не зустрічаються. Гени, які з’явилися у одного з батьків пізніше, ніж у будь-якого іншого, є надлишковими. При схрещуванні гени з однаковими номерами інновацій шикуються в лінію і схрещуються одним з двох способів. У першому методі гени, що збігаються, обираються випадковим чином для генома потомства. В якості альтернативи можна усереднити ваги зв’язків відповідних генів [2].

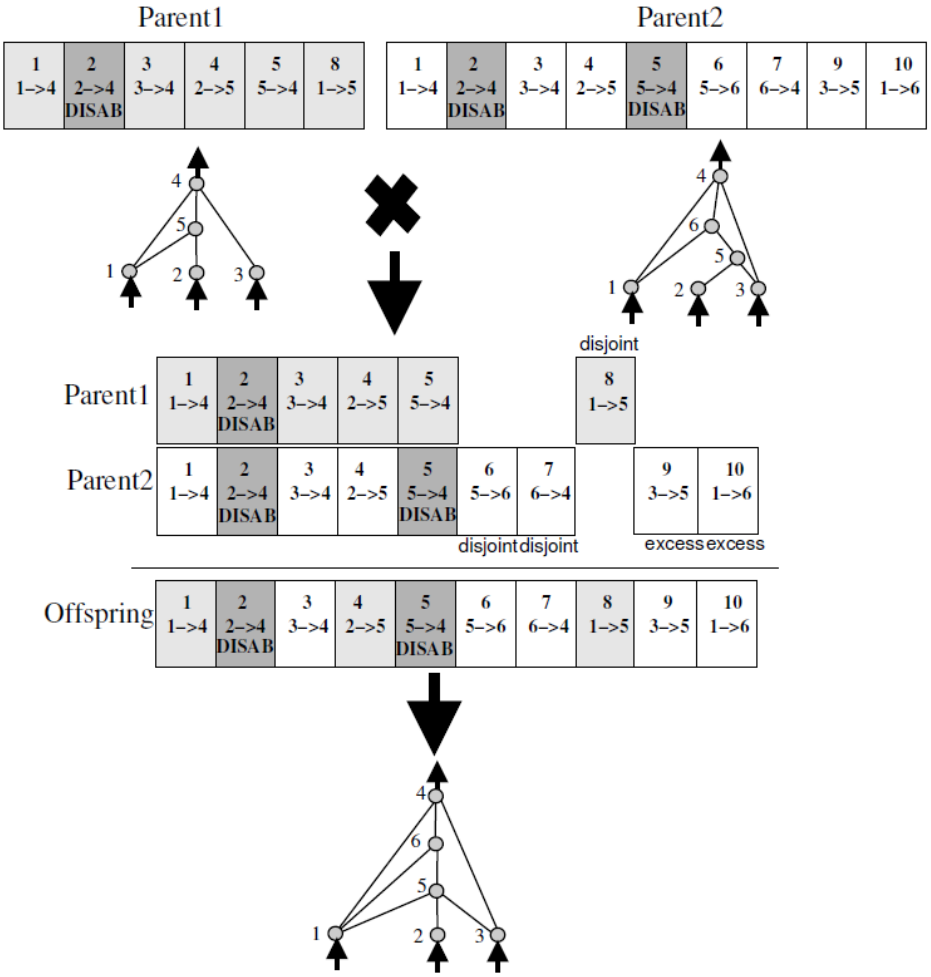

Рисунок 3.3 – Порівняння геномів для різних мережевих топологій з використанням номерів інновацій

3.2 Генеративні і розвивальні системи

У прямих кодуваннях, таких як NEAT, кожна частина, що представляє рішення, відображається в єдиному фрагменті структури в кінцевому [6], [17]. Суттєвим недоліком цього підходу є те, що навіть коли різні частини рішення подібні, вони повинні кодуватися і, отже, використовуватися окремо. Для вирішення цього слід використовувати непряме кодування, що означає, що опис рішення стискається таким чином, що інформація може бути використана повторно, що дозволяє остаточному рішенню містити більше компонентів, ніж самого опису. Непряме кодування, яке є основним фокусом області генеративних та розвивальних систем (GDS), є потужним, оскільки дозволяє представляти рішення у вигляді шаблону параметрів, а не вимагати, щоб кожен параметр був представлений окремо [19], [24], [26].

3.2.1 Мережі, що створюють композиційні шаблони

Нещодавно NEAT було розширено з метою еволюції абстракції розвитку на високому рівні, яка називається мережами, що створюють композиційні шаблони (CPPN) [25]. Ідея CPPN полягає в тому, що закономірності в природі можуть бути описані на високому рівні як композиція функцій, де кожна функція в композиції являє собою етап розвитку. CPPN подібні до ANN, але вони покладаються на більше ніж одну функцію активації (кожна з яких представляє загальну закономірність). Цікаво, що, оскільки CPPN – це також зв’язані графи, їх можна розвивати за допомогою NEAT, як і ANN. Таким чином, кодування CPPN не вимагає нового еволюційного алгоритму для еволюції.

Непряме кодування CPPN може компактно кодувати шаблони з такими закономірностями, як симетрія, повторення та повторення з варіацією [10], [12], [39]. Наприклад, просто додавши гауссову функцію, яка є симетричною, вихідний шаблон може стати симетричним. Періодична

функція, така як синус, створює сегментацію шляхом повторення. Найголовніше, повторення з варіацією (наприклад, такі як пальці людської руки) легко виявити, поєднуючи правильні координатні каркаси (наприклад, синусоїдальні та гауссові) з неправильними (наприклад, асиметричну вісь x). Наприклад, функція, яка приймає на вхідні дані суму симетричної функції з асиметричною, виводить шаблон із недосконалою симетрією. Таким чином, CPPN створюють регулярні шаблони з незначними варіаціями. Потенціал CPPN у представленні шаблонів з особливостями, що нагадують шаблони у природних організмах, було продемонстровано в ряді досліджень [19], [30], [32].

Зокрема, CPPN створюють фенотип, який є функцією від n вимірів, де n – кількість вимірів у фізичному просторі. Для кожної координати в цьому просторі рівень її вираження є результатом функції, що кодує фенотип. На рисунку 3.4 показано, як двовимірний фенотип може бути сформований функцією від двох параметрів, яка представлена мережею складених функцій. Рисунок 3.4(a) показує, як функція f приймає аргументи x та y, які є координатами у двовимірному просторі. Коли всі координати накреслені з інтенсивністю, що відповідає виходу f, результатом є просторова картина, яку можна розглядати як фенотип, генотип якого f.

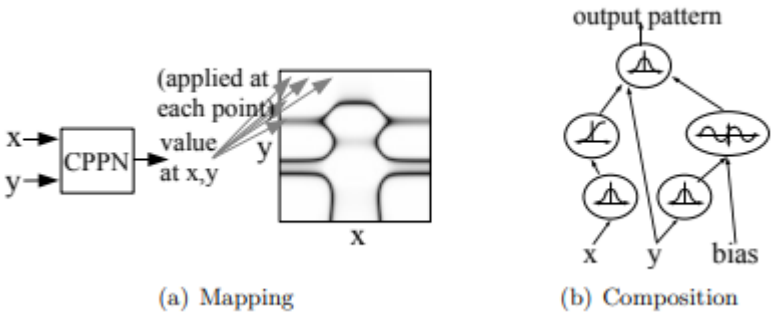

Рисунок 3.4 – Кодування CPPN

Рисунок 3.4(b) зображує, що CPPN – це графік, який визначає, які функції пов’язані. Зв’язки зважуються таким чином, що вихід функції помножується на вагу вихідного з’єднання.

Оскільки CPPN – це набір традиційних ANN, які можуть апроксимувати будь-яку функцію [10], CPPN також є універсальними апроксиматорами функцій. Таким чином, CPPN може кодувати будь-який шаблон у межах свого n-мірного простору.

3.2.2 HyperNEAT

HyperNEAT є непрямим розширенням кодування NEAT, яке доведено в ряді складних сфер застосування, що вимагають виявлення закономірностей [7], [12], [18], [19].

Основною ідеєю HyperNEAT є розширення CPPN, які кодують просторові шаблони, щоб також представляти моделі зв’язку [7], [18], [19]. Таким чином, NEAT може розвивати CPPN, які представляють широкомасштабні ANN зі своїми симетріями та закономірностями. Ключове розуміння полягає в тому, що двовимірні просторові структури є ізоморфними шаблонами зв’язку в n вимірах, тобто в яких координати кожної кінцевої точки задаються n параметрами. Розглянемо CPPN, яка приймає чотири входи, позначені $x1$, $y1$, $x2$ та $y2$; ця точка в чотиривимірному просторі також позначає зв’язок між двовимірними точками $(x1, y1)$ і $(x2, y2)$, а вихід CPPN для цього входу тим самим представляє вагу цього з’єднання (рисунок 3.5). Перевірючи таким чином всі можливі зв’язки між набором точок, CPPN може створити нейронну мережу, де кожна опитувана точка є позицією нейрона. Простір, в якому розташовані ці нейрони, називається субстратом. Оскільки зв’язки створюються за допомогою функції їх кінцевих точок, то кінцева структура створюється із знанням її геометрії. Фактично, CPPN малює шаблон на внутрішній стороні чотиривимірного гіперкуба, що інтерпретується як

ізоморфний шаблон зв’язку, що пояснює походження назви NEAT на основі Hypercube (HyperNEAT). Шаблони зв’язку, створені CPPN таким чином, називаються субстратами, щоб їх можна було словесно відрізнити від самого CPPN, який має свою внутрішню топологію.

Набору вузлів, що називається субстратом, присвоюються координати, які варіюються від -1 до 1 у всіх вимірах. На першому кроці кожне потенційне з’єднання у субстраті опитується для визначення його присутності та ваги; темні спрямовані лінії у субстраті, зображені на рисунку, представляють зразок з’єднань, які опитуються. На другому кроці зображено CPPN (який еволюціонував) зсередини, що є графіком, який визначає, які функції активації пов’язані. Як і в ANN, з’єднання зважуються таким чином, що вихід функції помножується на вагу вихідного з’єднання. Для кожного запиту CPPN приймає в якості вхідних даних позиції двох кінцевих точок і на третьому кроці виводить вагу зв’язку між ними. Таким чином, CPPN можуть створювати правильні моделі зв’язків у просторі.

Кожна опитувана точка у субстраті є вузлом у ANN. У традиційних реалізаціях HyperNEAT експериментатор визначає як розташування, так і роль (тобто приховану, вхідну чи вихідну) кожного такого вузла. Як правило, вузли розміщують у субстраті для відображення геометрії задачі [7], [12], [18]. Таким чином, зв’язаність субстрату є функцією структури задачі.

Наприклад, датчики автономного робота можуть бути розміщені зліва направо у субстраті в тому самому порядку, в якому вони існують на роботі. Виходи для переміщення вліво або вправо також можна розміщувати в однаковому порядку, дозволяючи HyperNEAT зрозуміти з самого початку співвідношення датчиків до ефекторів. Таким чином, знання про геометрію проблеми можуть бути використані у пошуку, а HyperNEAT може використовувати закономірності (наприклад, суміжність або симетрію) проблеми, невидимі для традиційних кодувань.

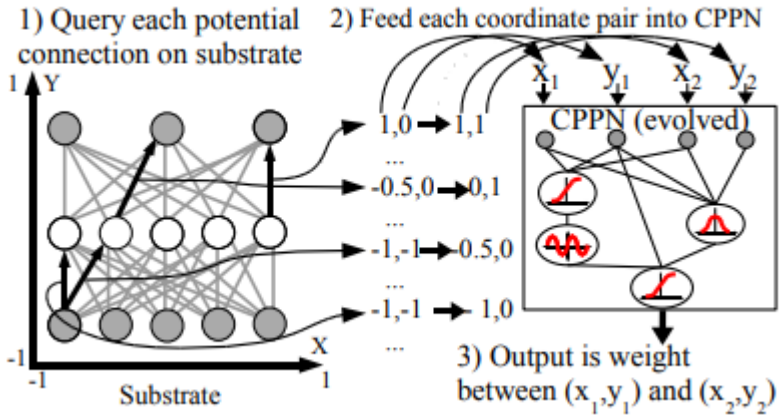

Рисунок 3.5 – Інтерпретація шаблону геометричних зв’язків на основі гіперкуба

Звичайний метод управління зв’язуванням в HyperNEAT – це поріг, який обмежує діапазон значень, що виводяться CPPN, який може бути виражений як ваговий коефіцієнт. Поріг – це параметр, вказаний при ініціалізації, який рівномірно застосовується до всіх запитуваних з’єднань. Коли величина виходу CPPN нижче цього порогу, з’єднання не записується.

Однак, Verbancsics і Stanley [4] ввели альтернативу традиційному рівномірному порогу, який називається вихідним виразом зв’язку (HyperNEAT-LEO), що дозволило HyperNEAT еволюціонувати шаблон ваг незалежно від шаблону вираження зв’язку. LEO представляється як додатковий вивід до CPPN, який вказує, чи має бути виражене з’єднання чи ні. Якщо вихідний коефіцієнт LEO перевищує нуль, тоді створюється відповідне з’єднання і його вага встановлюється як вихідне значення вихідної ваги CPPN. Оскільки HyperNEAT розвиває такі закономірності, як функції геометрії, важливі загальні топографічні принципи організації зв’язку можуть бути закладені в початкову сукупність [12]. Наприклад, HyperNEAT можна посіяти з упередженням до локальних зв’язків, створеного через LEO, в якому локальність виражається за допомогою функції Гауса. Оскільки функція Гауса досягає максимуму, коли її вхід

дорівнює 0.0, вводячи різницю між координатами, напр. x, досягає найвищого значення, коли координати однакові. Таким чином, таке насіння забезпечує концепцію локальності, оскільки чим локальніший зв’язок (тобто коли x наближається до 0.0), тим більший результат виходить з функції Гауса.

Оскільки зараз для HyperNEAT існують два різні порогові методи, новий підхід, що буде розглядатися в цій роботі, порівнюється з обома. Однак, незалежно від підходу до встановлення порогів, проблема, яка виникла з HyperNEAT, полягає в тому, що експериментатору залишається вирішити, скільки прихованих вузлів має бути і де їх також розмістити. Тобто, хоча CPPN визначає, як зв’язувати вузли в геометричному просторі, він не вказує, де мають бути вузли, що особливо неоднозначно для прихованих вузлів.

Відповідаючи на цю проблему, слід розглянути розширення для HyperNEAT, в якому розміщення та щільність прихованих вузлів не потрібно встановлювати апріорі, коли вони насправді повністю визначаються неявною інформацією в самому CPPN.

3.3 Вибір зв’язків для кодування

Розміщення вузлів в оригінальному HyperNEAT вирішує користувач. І все ж, хоча часто можна визначити, як датчики та ефектори співвідносяться з геометрією задачі, користувачеві важко апріорі визначити найкраще розміщення та кількість необхідних прихованих вузлів. Наприклад, розташування прихованих вузлів у субстраті на рисунку 3.5 повинен був визначити користувач. Таким чином, HyperNEAT створює дивну ситуацію, коли він може вирішити, з якою вагою слід з’єднати будь-які два вузли в просторі, але він не може нічого сказати нам про те, де повинні бути вузли. Чи існує представлення, яке може змінити розміщення та щільність вузлів, які потенційно можуть охоплювати мережі від

декількох десятків вузлів до декількох мільярдів? Для відповіді на питання слід звернутися до неявної інформації в гіперкубі

Новим розумінням ES-HyperNEAT є те, що представлення, що кодує шаблон зв'язків через мережу, автоматично містить неявну інформацію, яка може бути корисною для вирішення, де слід розміщувати вузли. У HyperNEAT модель зв' язку описується CPPN, де кожна точка в чотиривимірному просторі позначає потенційний зв' язок між двома двовимірними точками (точка в чотиривимірному гіперкубі насправді є вагою з' єднання і не вузол). Оскільки CPPN приймає $x1$, $y1$, x2, і $y2$ як вхід, це функція безперервного континууму можливих координат для цих точок. Іншими словами, CPPN кодує потенційно нескінченну кількість ваг з'єднання в межах гіперкуба ваг. Таким чином, один цікавий спосіб думати про гіперкуб як теоретично нескінченний зразок можливих зв'язків, які можуть бути включені в субстрат нейронної мережі. Якщо з'єднання обрано для включення, то за необхідності вузли, які він з'єднує, також повинні бути включені в основу. Таким чином, запитуючи, які зв'язки включити з безлічі, ми також запитуємо, які вузли (а отже, і їх позиції) включити.

Розглядаючи питання про те, що включати в субстрат з вузлів на з'єднання, випливає два важливих уявлення. По-перше, чим більше таких з'єднань включено, тим більше вузлів також буде додано до субстрату. Таким чином, щільність вузлів зростає із збільшенням кількості з'єднань. По-друге, для будь-якого з наведених шаблонів з роздільною здатністю кінцевих значень є певна щільність вибірки, вище якої збільшення щільності надалі не дає переваг. Наприклад, якщо гіперкуб є рівномірним градієнтом максимальних ваг з'єднання (тобто всі ваги мають однакову константу), тоді він кодує субстрат, який обчислює ту саму функцію на кожному вузлі. Таким чином, додавання більше таких з'єднань та вузлів не додає нової інформації. З іншого боку, якщо через гіперкуб проходить смуга різних ваг, але в іншому випадку скрізь є рівномірні максимальні зв'язки, то ця смуга містить інформацію, яка могла б сприяти різній функції від її надмірно

рівномірних сусідів.

Таким чином, головне розуміння полягає в тому, що не завжди є гарною ідеєю додавати більше з’єднань, оскільки для будь-якого даного кінцевого шаблону при деякій роздільній здатності немає більше інформації, а додавання більшої ваги при такій високій роздільній здатності буде зайвим і непотрібним. Ця максимальна корисна роздільна здатність варіюється для різних областей гіперкуба залежно від складності основної схеми ваги в цих областях. Таким чином, відповідь на питання про те, які з’єднання слід включати в ES-HyperNEAT, полягає в тому, що з’єднання повинні бути включені з достатньо високою роздільною здатністю, щоб зафіксувати деталі (тобто інформацію) в гіперкубі. Більше, ніж це, було б зайвим. Отже, потрібен алгоритм, який може вибрати багато точок для вираження в областях високої дисперсії та меншу кількість точок для вираження в областях відносної однорідності. Кожна така точка є вагою з’єднання в субстраті, відповідні вузли якої також будуть виражені. Головний принцип простий: щільність слідує за інформацією. Таким чином, розміщення вузлів у топографічному слої ANN в кінцевому підсумку означає, де інформація зберігається в межах ваг.

Для виконання завдання вибору точок (тобто ваг) для вираження необхідна структура даних, яка дозволяє представляти простір на змінних рівнях деталізації. Одним із таких методів визначення є дерево квадрантів [16], яке традиційно описує двовимірні області. Він успішно застосовується в областях, починаючи від розпізнавання образів і закінчуючи кодуванням зображення [7], [9] і заснований на рекурсивному розділенні двовимірної області на чотири підобласті. Таким чином, розкладання області на чотири нові області може бути представлене як піддерево, батьківським є вихідна область з одним нащадком для кожної розкладеної області. Рекурсивне розбиття областей можна повторювати до досягнення бажаної роздільної здатності або поки не потрібний подальший підрозділ, оскільки додаткова роздільна здатність більше не розкриває нової інформації.

3.4 Вилучення інформації з дерева квадрантів

Замість того, щоб шукати безпосередньо у чотиривимірному просторі гіперкуба (для представлення двовимірної схеми зв'язку потрібні чотири виміри), ES-HyperNEAT ітеративно розкриває зв'язки ANN, починаючи з входів та виходів ANN (які є попередньо визначені користувачем). Цей підхід зосереджує пошук у гіперкубі на двовимірних перерізах гіперкуба.

Основою алгоритму ES-HyperNEAT є процедура вилучення інформації з дерева квадрантів, яка отримує двовимірну позицію як вхід, а потім аналізує вихідний шаблон зв'язку від цього одного нейрона (якщо він є входом), або вхідний зв'язок шаблону (якщо це вихідний результат). Наприклад, з урахуванням вхідного нейрона в точці $(a;\ b)$, алгоритм вибору зв' язку для дерева квадрантів застосовується лише до двовимірних вихідних шаблонів зв' язку, описаних функцією CPPN $(a;\ b;\ x2;\ y2)$, де діапазон $x2$ та y2 - від -1 до 1. Алгоритм працює у дві основні фази (рисунок 3.6): На фазі поділу та ініціалізації (рисунок 3.6 зверху) дерево квадрантів створюється шляхом рекурсивного розподілу початкового квадрата, який охоплює простір від $(-1,\ -1)$ до $(1,\ 1)$, доки не буде досягнуто бажаної початкової роздільної здатності r (наприклад, 4x4, що відповідає глибині 3 дерева квадрантів). Для кожного квадрата дерева квадрантів з центром $(x;\ y)$ CPPN взаємодіє з аргументами $(a;\ b;\ x;\ y)$ і зберігає отримане значення ваги з' єднання w.

Враховуючи значення $(w1;\ w2;\ \dots\ wk)$ k-листкових вузлів у піддереві дерева квадрантів вузла p та середню вагу w, дисперсію вузла p у квадраті можна розрахувати як

$$\sigma_p^2 = \frac{1}{k}\sum_1^k(\overline{w} - w_i)^2. \qquad (3.1)$$

Ця дисперсія є евристичним показником неоднорідності (тобто наявності

інформації) області. Якщо дисперсія батьківського елемента дерева квадрантів все ще перевищує заданий поріг поділу dt, тоді фазу поділу можна повторно застосувати для відповідного листа дерева, дозволяючи все більші щільності. Подібно до того, як роздільна здатність ініціалізації гарантує, що виконується якийсь мінімальний рівень вибірки (тобто так, що, швидше за все, буде виявлена основна форма закодованого шаблону), також можна встановити максимальний рівень роздільної здатності rm, щоб встановити верхню межу на кількість можливих нейрони за бажанням. Однак теоретично цікаво, що в принципі цей алгоритм може давати довільно високу щільність, а це означає, що можна представити дуже великі ANN.

Представлення дерева квадрантів, створене на фазі ініціалізації, служить показником евристичної дисперсії для прийняття рішення про зв'язки (і, отже, розміщення та щільність нейронів) для кодування. Оскільки в областях з більшою дисперсією слід виразити більше зв'язків, наступним чином виконується фази обрізання та видобутку (рисунок 3.6 снизу), в якому дерево квадрантів проходить спочатку з глибини, поки дисперсія поточного вузла не буде меншою, ніж поріг дисперсії $t2$ або поки вузол не має дочірніх елементів (що означає, що дисперсія дорівнює нулю). Згодом для кожного відповідного вузла з центром $(x; y)$ створюється зв' язок $(a; b; x; y)$. Результатом є вища роздільна здатність у зонах із більшими варіаціями.

На рисунку 3.7 показаний приклад вихідних з'єднань з вихідного нейрона $(0; 1)$ (зображених лише їх цільовим розташуванням для зрозумілості), обраних на цьому етапі алгоритму. Дисперсія велика на кордонах кіл, що призводить до високої щільності виражених точок поблизу цих місць. Однак з метою ідентифікації зв'язків для включення в нейронну топографію вихідний зразок, що виводиться алгоритмом дерева квадрантів, може бути додатково вдосконалений. Якщо ми міркуємо про шаблон, що виводиться CPPN, як про своєрідну мову для вказівки розташування

виражених з’єднань, то має сенс додатково обрізати точки навколо кордонів, щоб CPPN легко було остаточно кодувати точки в межах одної області.

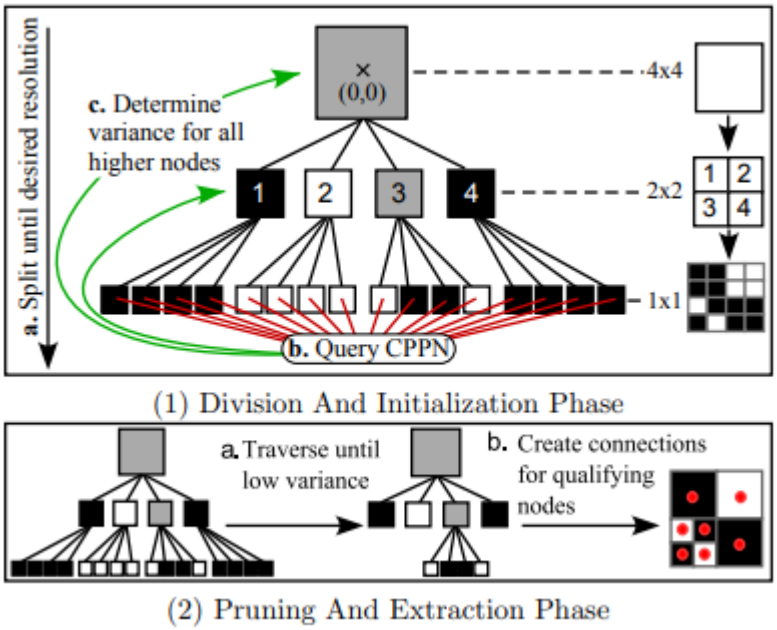

Рисунок 3.6 – Приклад вилучення інформації з дерева квадрантів

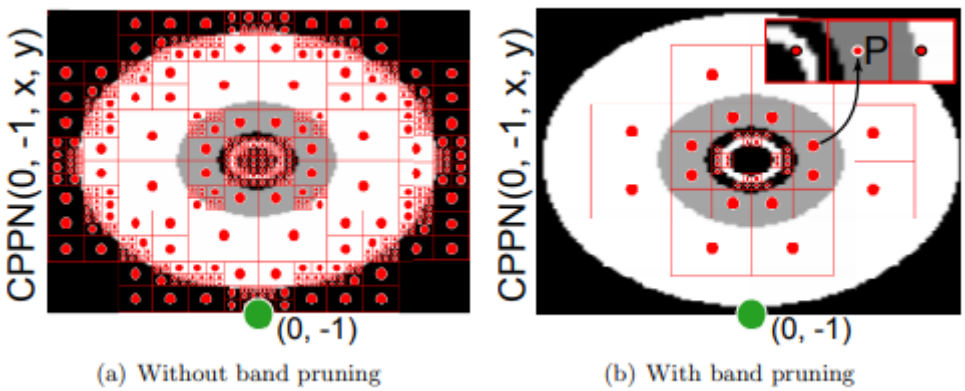

Рисунок 3.7 – Приклад обрання зв’язків

Таким чином, більш ощадлива «мова» для опису моделей щільності буде ігнорувати краї і зосереджуватись на внутрішній області смуг, які є точками, які укладені принаймні двома сусідами з протилежних сторін

(наприклад, зліва та справа) з різними рівнями активації CPPN. Крім того, більш вузькі смуги можна інтерпретувати як запити на більшу точкову щільність, надаючи CPPN чіткий механізм впливу щільності.

Таким чином, для полегшення пов'язування додається етап обрізання, який видаляє точки, які не знаходяться в смузі. Членство в групі для квадрата з центром $(x; y)$ і шириною w визначається рівнем діапазону

$$\beta = \max(\min(d_{top}, d_{bottom}), \min(d_{left}, d_{right})), \quad (3.2)$$

де d_{left} – різниця рівнів активації CPPN між з'єднанням $(a; b; x; y)$ та його лівим сусідом $(a; b; x - w; y)$;

d_{right}, d_{bottom} і d_{top} обчислюються відповідно.

Якщо рівень смуги нижче заданого порогу t, відповідне з'єднання не кодується. Цей підхід також, природно, дозволяє CPPN збільшувати щільність обраних точок, створюючи більше смуг або роблячи їх тоншими. Таким чином, для кодування розміщення вузлів та зв'язків не потрібно ніякої нової інформації та жодної нової репрезентативної структури, окрім CPPN, яка вже використовується в HyperNEAT.

3.5 Алгоритм ES-HyperNEAT

Повний алгоритм ES-HyperNEAT зображений на рисунку 3.8. Зв'язки, що беруть початок з $(0; 1)$, обираються на основі підходу до вибору зв'язків, що описані в попередньому розділі. Відповідний прихований вузол створюється, якщо він ще не існує. Цей підхід можна ітеративно застосовувати до виявлених прихованих вузлів, поки не буде досягнуто встановлений користувачем максимальний рівень ітерації або більше не буде виявлено необхідної інформації в гіперкубі. Щоб прив'язати мережу до виходів, підхід потім вибирає зв'язки на основі вхідних моделей зв'язку кожного виходу.

Як тільки всі приховані нейрони виявляються, зберігаються лише ті, які мають шлях до вхідного та вихідного нейронів. Цей ітераційний підхід допомагає зменшити обчислювальні витрати, фокусуючи пошук на послідовності двовимірних перерізів гіперкуба, замість того, щоб шукати інформацію безпосередньо в повному чотиривимірному гіперпросторі.

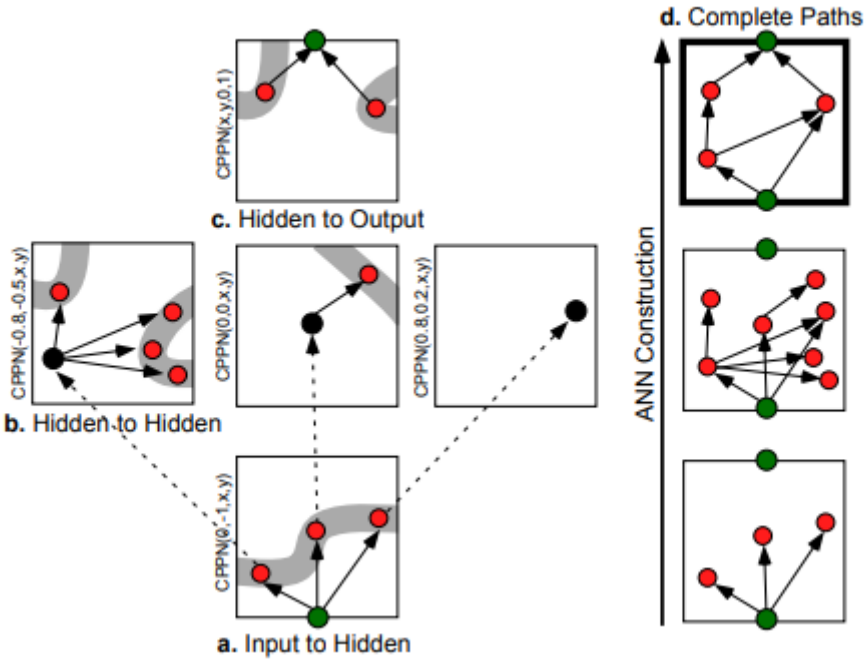

Рисунок 3.8 – Алгоритм ES-HyperNEAT

ES-HyperNEAT в кінцевому підсумку об’єднує набір алгоритмічних переваг, що тягнуться до NEAT, кожен абстрагований від важливої грані природної еволюції, що сприяє його здатності розвивати складність. По-перше, розвинення складності вимагає механізму збільшення інформаційного вмісту в геномі протягом поколінь [23], [35]. По-друге, геометрія відіграє важливу роль у природному нейронному зв’язку; при нейроеволюції наділення нейронів геометричними координатами означає, що геном може впливати на закономірності зв’язку через нейронну геометрію, забезпечуючи тим самим своєрідне сканування для розміщення когнітивних структур [8], [19], [28]. По-третє, розміщення та щільність

нейронів по всій геометрії мережі повинні відображати складність базової функціональності відповідних частин [35].

Оскільки ES-HyperNEAT може автоматично визначати геометрію та щільність вузла з CPPN замість того, щоб вимагати апріорного розміщення (як у оригінальному HyperNEAT), це значно розширює сферу нейронних структур, які може виявити еволюція. Цей підхід не тільки змінює розташування кожного нейрона в мережі, але також може представляти області різної щільності, а це означає, що роздільна здатність може цілісно зростати в процесі еволюції. Основне розуміння полягає в тому, що зв’язок та розміщення прихованих вузлів можуть бути автоматично визначені за інформацією, вже притаманною шаблону, кодованому CPPN. Таким чином, щільність вузлів визначається автоматично і фактично необмежена. Отже, субстрати необмеженої щільності можуть бути розвинені та визначені без будь-якого додаткового представлення поза вихідної CPPN в HyperNEAT.

4 АНАЛІЗ ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ

4.1 Фреймворк Keras

Keras це відкрита нейромережева бібліотека, написана мовою Python. Вона здатна працювати поверх Deeplearning4j, TensorFlow та Theano. Спроектовану для уможливлення швидких експериментів з мережами глибинного навчання, її зосереджено на тому, щоби вона була мінімальною, модульною та розширюваною.

Ця бібліотека містить численні реалізації широко вживаних будівельних блоків нейронних мереж, таких як шари, цільові та передавальні функції, оптимізатори, та безліч інструментів для спрощення роботи із зображеннями та текстом. Основною перевагою є відкрита безкоштовна ліцензія для використання у будь-яких цілях.

Keras є високорівневим шаром абстракціїї, що дозволяє нам легко створювати архітектуру бажаної нейронної мережі. У лістингу 4.1 наведено код для створення декілька слоїв мережі у вигляді послідовності (Sequental).

Лістинг 4.1 – Створення архітектури нейронної мережі

```
from keras.layers import Sequential
model = Sequential()
model.add(Dense(512, input_shape=(max_words,)))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

В наведеному прикладі ми маємо змогу легко створити наступні шари:

- dense. Простий «щільний» шар, де усі нейрони з'єднані з попередніми;

- activation. Функція активації, яку ми задаємо у вигляді випрямлювача;

- dropout. Шар, який випадковим чином відкидує частину вхідної інформації з попереднього слою (розмір відкиду ми має змогу обирати самі).

Як можна побачити, заданими командами ми створили просту нейронну мережу всього за декілька рядків. Додатково, для більш зручного сприйняття, фреймворк надає нам інструменті для візуалізації створеної архітектури (лістинг 4.2).

Лістинг 4.2 – Візуалізація нейронної мережі

```
from keras.utils import plot_model
plot_model(model,to_file='model.png',
show_shapes=True)
```

Лістинг 4.3 – Підготовка моделі

```
model.compile(loss='categorical_crossentropy',
optimizer='adam',
metrics=['accuracy'])
```

Отже, ми сформували нашу модель. Тепер потрібно підготувати її до роботи. Що означають параметри функції compile? loss – це функція помилки, в нашому випадку – це перехресна ентропія, саме для неї ми готували наші мітки у вигляді матриць; optimizer – використовуваний оптимізатор, тут міг би бути звичайний стохастичний градієнтний спуск, але Adam показує кращу збіжність на цій задачі; metrics – метрики, за яким вважається якість моделі, в нашому випадку – це точність (accuracy), тобто частка вірно вгаданих відповідей (лістинг 4.3).

Врешті, нам потрібно навчити нашу модель на заданих параметрах. Метод fit виконує навчання. Він приймає на вхід навчальну вибірку разом з мітками – x_train і y_train, розміром батча batch_size, який обмежує кількість прикладів, поданих за раз, кількістю епох для навчання epochs (одна епоха – це один раз повністю пройдена моделлю навчальна вибірка), а також тим, яку частку навчальної вибірки віддати під валідацію validation_split.

Повертає цей метод history – це історія помилок на кожному кроці навчання. І нарешті, тестування. Метод evaluate отримує на вхід тестову вибірку разом з мітками для неї. Метрика була задана ще при підготовці до роботи, так що більше нічого не потрібно. Лістинг 4.4 ілюструє цей процес у програмному коді.

Лістинг 4.4 – Навчання і тестування моделі

```
history = model.fit(x_train, y_train,
batch_size=batch_size,
epochs=epochs,
verbose=1,
validation_split=0.1)
score = model.evaluate(x_test, y_test,
batch_size=batch_size)
```

4.2 Бібліотека MultiNEAT

MultiNEAT – це портативна бібліотека програмного забезпечення для виконання нейроеволюції, форми машинного навчання, яка тренує нейронні мережі з генетичним алгоритмом. Він заснований на NEAT, вдосконаленому методі розвитку нейронних мереж шляхом комплексифікації. Нейронні мережі в NEAT починають еволюцію з дуже простих геномів, які ростуть протягом наступних поколінь. Особи в популяції, що розвивається, групуються за подібністю за видами, і кожна з них може конкурувати лише з особинами одного виду.

Комбінований ефект видоутворення, починаючи з найпростішої початкової структури та правильного узгодження геномів через маркування генів з історичними маркуваннями, дає алгоритм, який, як було доведено, є дуже ефективним у багатьох сферах та порівняльних показниках проти інших методів. Вона представляє собою абстракцію штучних нейронних мереж і генетичного програмування, надаючи нам найбільш зручні можливості до створення нейронних мереж та навчання і зміни архітектури

за допомогою NEAT, використовуючи невелику кількість програмного коду.

У поточній реалізації NEAT-Python підтримується популяція окремих геномів. Кожен геном містить два набори генів, які описують, як побудувати штучну нейронну мережу:

- вузлові гени, кожен з яких визначає окремий нейрон;
- гени зв'язку, кожен з яких визначає єдиний зв'язок між нейронами.

Щоб розробити рішення проблеми, користувач повинен надати функцію придатності, яка обчислює єдине дійсне число, яке вказує якість окремого геному: краща здатність вирішити проблему означає більш високий бал. Алгоритм прогресує через певний користувачем число поколінь, з кожним поколінням, виробленим відтворенням (або зі спорідненням, або без) і мутацією найбільш пристосованих особин попереднього покоління.

Операції розмноження і мутації можуть додавати вузли та/або з'єднання до геному. Так як алгоритм продовжує працювати, геноми (і нейронні мережі, які вони створюють) можуть ставати все більш і більш складними. Коли задане число поколінь досягнуто або коли принаймні одна особина (для функції критерію придатності max; інші налаштовуються) перевищує заданий користувачем поріг придатності, алгоритм завершується.

Одна з труднощів у цьому налаштуваннні пов'язана з реалізацією кросовера – як зробити кросовер між двома мережами різної структури? NEAT обробляє це, відстежуючи походження вузлів з ідентифікуючим номером (нові, більш високі номери генеруються для кожного додаткового вузла). Ті, що отримані від спільного батька (які є гомологічними), зіставляються для кросовера, і з'єднання зіставляються, якщо вузли, які вони з'єднують, мають спільне походження (існують відмінності в тому, як саме це робиться в залежності від реалізації NEAT).

Після того, як реалізована функція пристосованості (лістинг 4.5), нам

в основному просто потрібен додатковий шаблонний код, який виконує наступні кроки:

- створює об’єкт neat.config.Config конфігурації з файлу конфігурації;
- створює об’єкт neat.population.Population заповнення за допомогою об’єкта конфігурації, створеного вище;
- викликає метод run для об’єкта Population, надавши йому функцію придатності і (необов’язково) максимальну кількість поколінь, які ми хочемо запустити.

Лістинг 4.5 – Створення функції обчислення нових геномів

```
def eval_genomes(genomes, config):
    for genome_id, genome in genomes:
        genome.fitness = 4.0
        net                                    =
neat.nn.FeedForwardNetwork.create(genome, config)
        for xi, xo in zip(xor_inputs, xor_outputs):
            output = net.activate(xi)
        genome.fitness -= (output[0] - xo[0])** 2
```

Після повернення виклику методу run об’єкта population можна запросити елемент статистики population (neat.statistics.StatisticsReporter), щоб отримати кращий геном(и), що був під час запуску (лістинг 4.6). У цьому прикладі ми беремо геном «переможця», який повертається методом pop.statistics.best_genome(). Додатково, у модулі візуалізації доступні функції для побудови графіків найкращої і середньої придатності порівняно з поколінням, побудови графіка зміни видів у порівнянні з поколінням і відображення структури мережі, описуваної геномом (лістинг 4.7).

Лістинг 4.6 – Створення основних об’єктів для запуску Neat і показу результатів

```
    config = neat.Config(neat.DefaultGenome,
neat.DefaultReproduction,
neat.DefaultSpeciesSet, neat.DefaultStagnation,
config_file)

        # Create the population, which is the top-level
object for a NEAT run.
        p = neat.Population(config)

        # Add a stdout reporter to show progress in the
terminal.
        p.add_reporter(neat.StdOutReporter(True))
        stats = neat.StatisticsReporter()
        p.add_reporter(stats)
        p.add_reporter(neat.Checkpointer(5))
        # Run for up to 300 generations.
        winner = p.run(eval_genomes, 300)
        # Display the winning genome.
        print('\nBest genome:\n{!s}'.format(winner))
        # Show output of the most fit genome against
training data.
        print('\nOutput:')
        winner_net =
neat.nn.FeedForwardNetwork.create(winner, config)
        for xi, xo in zip(xor_inputs, xor_outputs):
            output = winner_net.activate(xi)
        print("input {!r}, expected output {!r}, got
{!r}".format(xi, xo, output))
    p = neat.Checkpointer.restore_checkpoint('neat-
checkpoint-4')
        p.run(eval_genomes, 10)
```

Лістинг 4.7 – Візуалізація результатів

```
    node_names = {-1:'A', -2: 'B', 0:'A XOR B'}
        visualize.draw_net(config, winner, True,
node_names=node_names)
        visualize.plot_stats(stats, ylog=False,
view=True)
        visualize.plot_species(stats, view=True)
```

5 ІМІТАЦІЙНЕ МОДЕЛЮВАННЯ

5.1 Створення і налаштування моделі

Перед моделюванням задачі нам слід налаштувати моделі еволюції за допомогою біблотеки ES-HyperNEAT. Для початку еволюції нам потрібно створити з неї насіннєвий геном та популяцію. Однак перед усім ми створюємо об’єкт, який містить усі параметри, використовувані NEAT. У лістингу 5.1 представлена ініціалізація з розміром популяції 100. Зазвичай це точка, де встановлюються всі спеціальні значення для параметрів. Тут ми встановлюємо розмір популяції 100 особин (значення за замовчуванням – 300). Тепер ми створюємо геном з 3 входами і 2 виходами (лістинг 5.2).

Звернемо увагу, що ми встановлюємо більше властивостей геному, ніж просто кількість входів / виходів. Крім того, якщо кількість входів, які ми збираємося використовувати у своєму проекті, дорівнює 2, нам потрібно написати 3 у конструкторі. Завжди додаємо один додатковий вхід. Останній вхід завжди використовується як зміщення, а також, коли ми активуємо мережу, завжди встановлюємо для останнього входу значення 1,0 (або будь-яке інше постійне ненульове значення). Також встановлюється тип функції активації виходів та прихованих нейронів. Приховані нейрони необов’язкові. Після створення геному ми створюємо популяцію так, як показано у лістингу 5.3.

Лістинг 5.1 – Ініціалізація

```
import MultiNEAT as NEAT
params = NEAT.Parameters()
params.PopulationSize = 100
```

Лістинг 5.2 – Створення геному

```
genome = NEAT.Genome(0, 3, 0, 2, False, NEAT.ActivationFunction.UNSIGNED_SIGMOID, NEAT.ActivationFunction.UNSIGNED_SIGMOID, 0, params, 0)
```

Лістинг 5.3 – Створення популяції

```
pop = NEAT.Population(genome, params, True, 1.0, 0) # the 0 is the RNG seed
```

Останні два параметри визначають, чи слід розподіляти популяцію і скільки. Оскільки ми починаємо з нового геному, а не з того, який раніше було збережено, ми рандомізуємо початкову популяцію. Зараз можна починати еволюцію. Для цього нам потрібна функція оцінки пристосованості. Вона приймає геном як параметр і повертає число з плаваючею комою, яке відповідає фенотипу геному. Сам метод представлений у лістингу 5.4-5.5.

Лістинг 5.4 – Метод запуску еволюції

```
def evaluate(genome):
    net = NEAT.NeuralNetwork()
    genome.BuildPhenotype(net)
    # let's input just one pattern to the net, activate it once and get the output
    net.Input( [ 1.0, 0.0, 1.0 ] )
    net.Activate()
```

Лістинг 5.5 – Метод запуску еволюції (продовження)

```
output = net.Output()

        # the output can be used as any other Python
iterable. For the purposes of the tutorial,
        # we will consider the fitness of the
individual to be the neural network that outputs
constantly
        # 0.0 from the first output (the second output
is ignored)

        fitness = 1.0 - output[0]
        return fitness
```

Отже, оскільки ми вже маємо свою функцію оцінки, ми можемо ввести основний цикл еволюції поколінь, як показано у лістингу 5.6.

Лістинг 5.6 – Цикл запуску і відбору поколінь

```
for generation in range(100): # run for 100
generations

        genome_list = NEAT.GetGenomeList(pop)
        # apply the evaluation function to all genomes
        for genome in genome_list:
            fitness = evaluate(genome)
            genome.SetFitness(fitness)

        # advance to the next generation
        pop.Epoch()
```

Використання алгоритму ES-HyperNEAT поділяється на 3 етапи: розділ і ініціалізація, обрізка і вилучення та сам алгоритм еволюції. Розглянемо псевдокод для кожного етапу.

Перший етап зображено на рисунку 5.1. На вході ми отримуємо координати вихідного або цільового вузла у точці (a, b). На виході ми отримуємо дерево квадрантів, в якому кожен вузол у точці (x, y) зберігає рівень активації CPPN для своєї позиції. Ініціалізоване дерево квадрантів використовується на етапі обрізки і вилучення для створення фактичних з'єднань ANN.

```
1  begin
2      root ← QuadPoint(0, 0, 1, 1) ; // x, y, width, level
3      q ← Queue() ;
4      q.enqueue (root) ;
5      while q is not empty do
6          p ← q.dequeue();
7          // Divide into sub-regions and assign children to parent
8          p.cs[0] ← QuadPoint(p.x - p.width/2, p.y - p.width/2, p.width/2, p.lev + 1) ;
9          p.cs[1] ← QuadPoint(p.x - p.width/2, p.y + p.width/2, p.width/2, p.lev + 1) ;
10         p.cs[2] ← QuadPoint(p.x + p.width/2, p.y + p.width/2, p.width/2, p.lev + 1) ;
11         p.cs[3] ← QuadPoint(p.x + p.width/2, p.y - p.width/2, p.width/2, p.lev + 1) ;
12         foreach c ∈ p.cs do
13             if outgoing then       // Querying connection from input or hidden node
14                 c.w ← CPPN (a, b, c.x, c.y);          // Outgoing connectivity pattern
15             else // Querying connection to output node
16                 c.w ← CPPN (c.x, c.y, a, b);          // Incoming connectivity pattern
17             end
18         end
19         // Divide until initial resolution or if variance is still high
20         if (p.level < initialDepth) | (p.level < maxDepth & variance (p) > divThr) then
21             foreach child ∈ p.cs do
22                 q.enqueue (child);
23             end
24         end
25     end
26     return root;
27 end
```

Рисунок 5.1 – Розділ і ініціалізація

Другий етап зображено на рисунку 5.2. На вході ми отримуємо координати вихідного або цільового вузла у точці (a, b) і сгенероване дерево квадрантів p. На виході ми додаємо з'єднання, які знаходяться в смугах двовимірного перерізу гіперкуба, що містить вихідний або цільовий вузол, до списку з'єднань.

```
1  begin
2  |  // Traverse quadtree depth-first
3  |  foreach c ∈ p.cs do
4  |  |  if variance (c) ≥ varianceThreshold then
5  |  |  |  PruningAndExtraction (a, b, connections, c, outgoing);
6  |  |  else
7  |  |  |  // Determine if point is in a band by checking neighbor CPPN values
8  |  |  |  if outgoing then
9  |  |  |  |  d_left ← | c.value - CPPN (a, b, c.x-p.width, c.y) | ;
10 |  |  |  |  d_right ← | c.value - CPPN (a, b, c.x+p.width, c.y) | ;
11 |  |  |  |  d_top ← | c.value - CPPN (a, b, c.x, c.y − p.width) | ;
12 |  |  |  |  d_bottom ← | c.value - CPPN (a, b, c.x, c.y + p.width) | ;
13 |  |  |  else // Querying connection to output node
14 |  |  |  |  d_left ← | c.value - CPPN (c.x-p.width, c.y, a, b) | ;
15 |  |  |  |  d_right ← | c.value - CPPN (c.x+p.width, c.y, a, b) | ;
16 |  |  |  |  d_top ← | c.value - CPPN (c.x, c.y − p.width, a, b) | ;
17 |  |  |  |  d_bottom ← | c.value - CPPN (c.x, c.y + p.width, a, b) | ;
18 |  |  |  end
19 |  |  |  if max(min(d_top, d_bottom), min(d_left, d_right)) > bandThreshold then
20 |  |  |  |  // Create new connection specified by (x1, y1, x2, y2, weight)
21 |  |  |  |  // and scale weight based on weight_range (e.g. [-3.0, 3.0])
22 |  |  |  |  if outgoing then
23 |  |  |  |  |  con ← Connection(a, b, c.x, c.y, c.w * weight_range);
24 |  |  |  |  else // Querying connection to output node
25 |  |  |  |  |  con ← Connection(c.x, c.y, a, b, c.w * weight_range);
26 |  |  |  |  if con ∉ connections then connections ← connections ∪ con;
27 |  |  |  end
28 |  |  end
29 |  end
30 end
```

Рисунок 5.2 – Обрізка і вилучення

Третій етап зображено на рисунку 5.3. На вході ми отримуємо сгенеровану CPPN, вхідні і вихідні позиції. На виході отримуємо зв'язки і приховані нейрони. Саме цей етап, по суті, і є алгоритмом ES-HyperNEAT.

```
2  begin
3      foreach input ∈ InputPositions do          // Input to hidden node connections
4          // Analyze outgoing connectivity pattern from this input
5          root ← DivisionAndInitialization (input.x, input.y, true);
6          // Traverse quadtree and add connections to list
7          PruningAndExtraction (input.x, input.y, connections1, root, true);
8          foreach c ∈ connections1 do
9              node ← Node(c.x2,c.y2);
10             if node ∉ HiddenNodes then HiddenNodes ← HiddenNodes ∪ node;
11         end
12     end
13     // Hidden to hidden node connections
14     UnexploredHiddenNodes ← HiddenNodes;
15     for i = 1 to iterationLevel do
16         foreach hidden ∈ UnexploredHiddenNodes do
17             root ← DivisionAndInitialization (hidden.x, hidden.y, true);
18             PruningAndExtraction (hidden.x, hidden.y, connections2, root, true);
19             foreach c ∈ connections2 do
20                 node ← Node(c.x2,c.y2);
21                 if node ∉ HiddenNodes then  HiddenNodes ← HiddenNodes ∪ node;
22             end
23         end
24         // Remove the just explored nodes
25         UnexploredHiddenNodes ← HiddenNodes - UnexploredHiddenNodes;
26     end
27     foreach output ∈ OutputPositions do          // Hidden to output connections
28         // Analyze incoming connectivity pattern to this output
29         root ← DivisionAndInitialization (output.x, output.y, false);
30         PruningAndExtraction (output.x, output.y, connections3, root, false);
31         /* Nodes not created here because all the hidden nodes that are
              connected to an input/hidden node are already expressed.          */
32     end
33     connections ← connections1 ∪ connections2 ∪ connections3;
34     Remove all neurons and their connections that do not have a path to an input and an
       output neuron;
35 end
```

Рисунок 5.3 – Алгоритм отримання зв’язків

Для імітування системи, перш за все, слід створити якийсь набір міст з випадковими матрицями витрат. Для ліпшого порівняння результатів ми будемо 10 разів створювати різні конфігурації набору міст, починаючи з 48 міст і закінчуючи 1522 міст. Приклад розташування міст показано на рисунку 5.4. Умовою задачі є початок і закінчення у місті 1. Програмний код для вищеописаних етапів алгоритму наведено у додатку А.

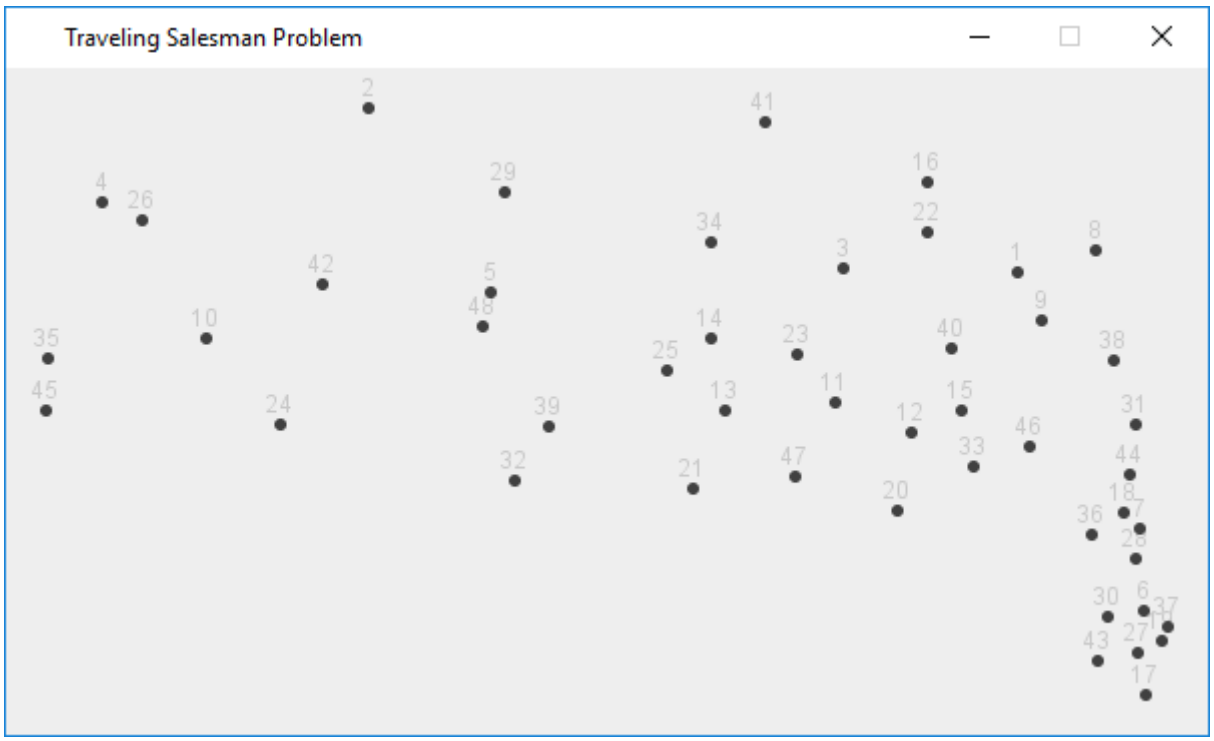

Рисунок 5.4 – Випадкова модель графу з містами

5.2 Навчання і візуалізація результатів

У результаті дослідів, імітаційна модель навчалася приблизно150 поколінь для 50 міст і 450 для 1500 міст, доки не було отримано найбільш оптимальний розв’язок. При цьому, рівень кросинговеру був 90%, а мутацій – 0,04%. Середня дистанція при проходженні усіх міст з поверненням у перше дорівнює 157496 абсолютних одиниць для першого покоління і 36442 для останнього. Отримані дані показують, що з часом оптимальність моделі зросла у приблизно 4 рази. Однак, збільшуючи кількість міст, для оптимального рішення час навчання моделі збільшувався експоненційно, а оптимальність зменшуваллася лінійно, що є більш кращим результатом у порівнянні з іншими алгоритмами.

На рисунках 5.5-5.6 представлено графіки залежності довжини шляху від покоління для 48 міст і 500 міст.

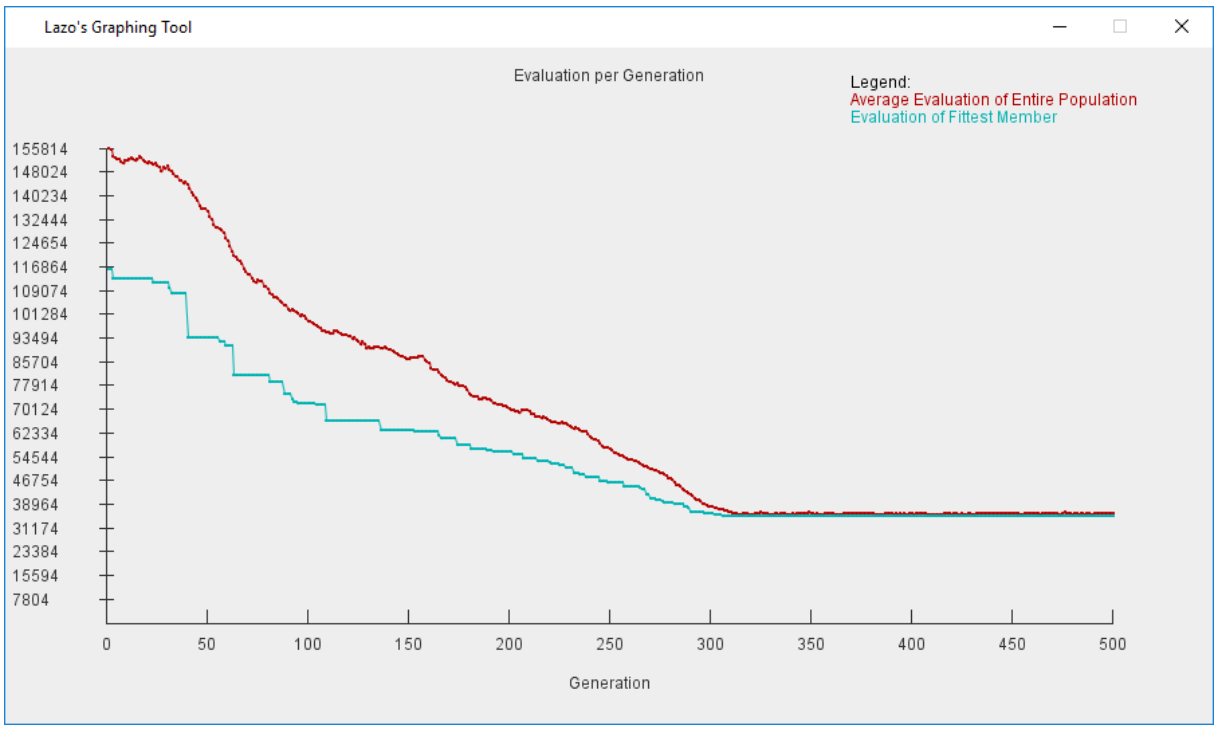

Рисунок 5.5 – Залежність між довжиною шляху і поколінням для 48 міст

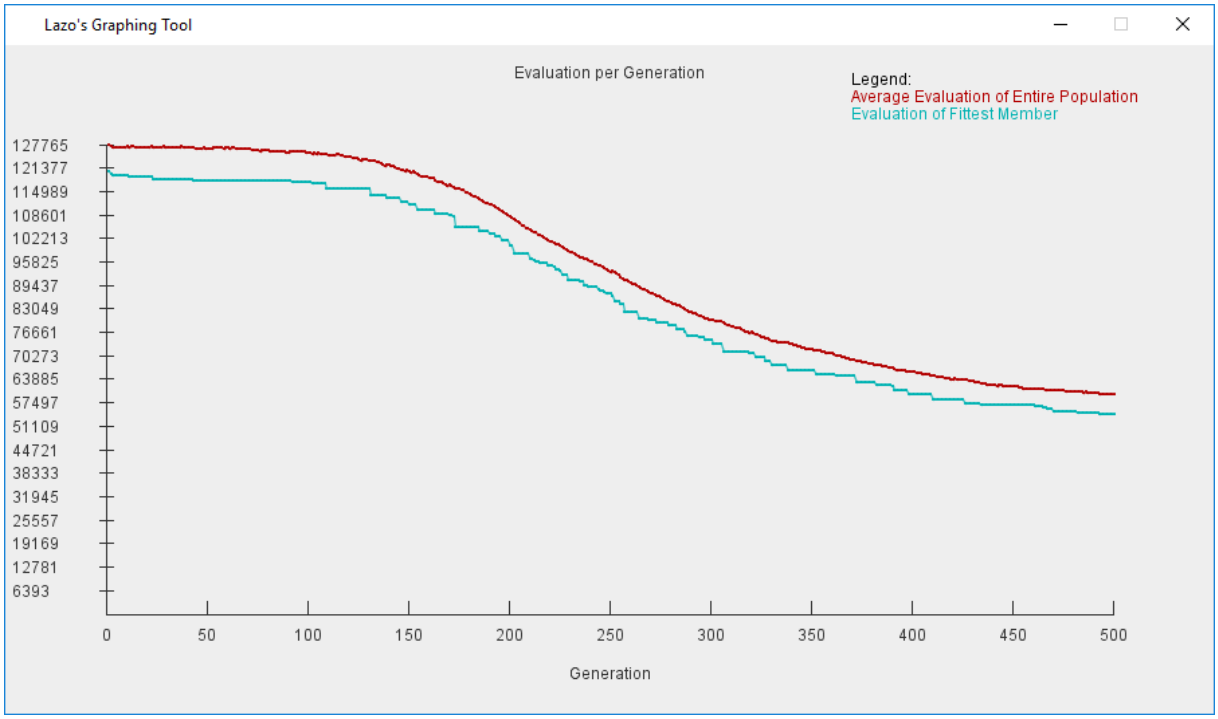

Рисунок 5.6 – Залежність між довжиною шляху і поколінням для 500 міст

Як можна побачити, приблизно на 300-му поколінні з’являється збіжність і більш оптимального розв’язку задачі ми не отримаємо. Отриманий розв’язок для першої умови задачі зображено на рисунку 5.7.

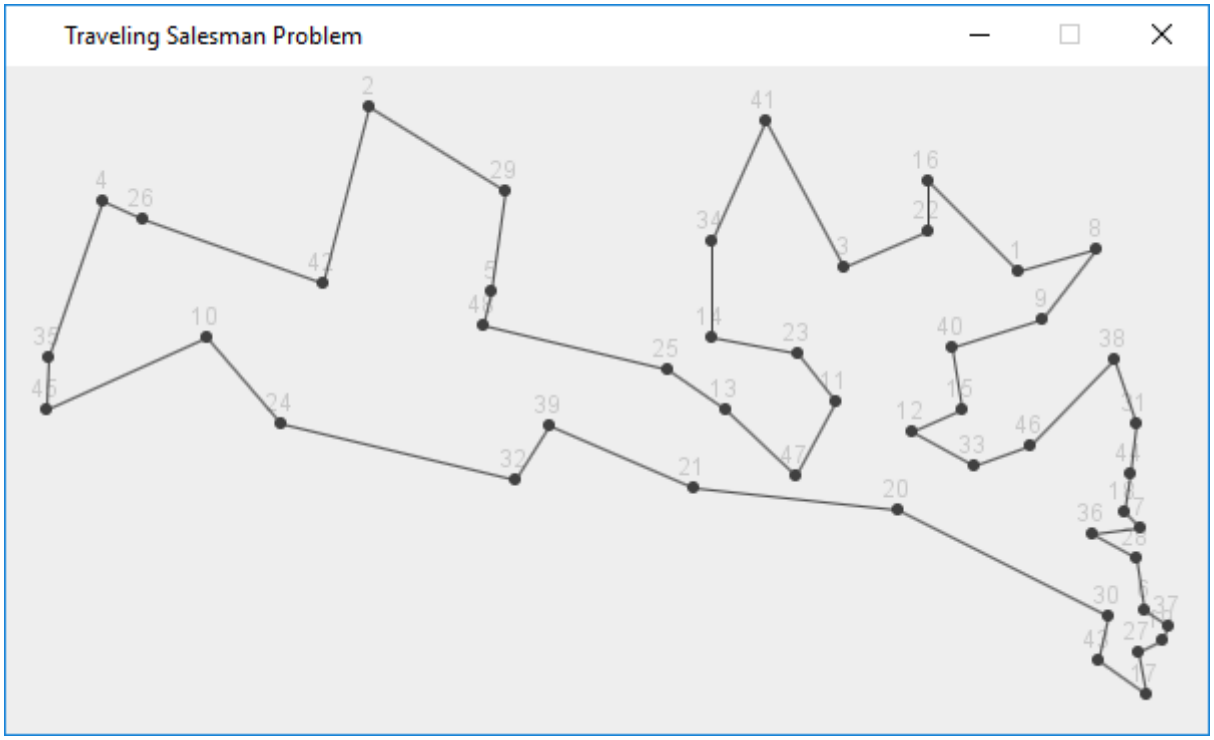

Рисунок 5.7 – Розв’язок задачі комівояжера для 48 міст

5.3 Порівняння результатів

У ході випробувань також були порівняні різні ітеративні і метаевристичні алгоритми та їх придатність до вирішення поставленої задачі. Результати наведені у таблиці 5.1.

Оскільки мурашиний і бджолиний алгоритми мають досить схожий алгоритм роботи, то їх результати не є занадто гіршими за ES-HyperNEAT. Однак метод k-opt-swap виявися дещо гіршим як з точки зору розв’язку, так і часу виконання. Але він все ще залишається досить точним методом для знаходження приблизного розв’язку за прийнятий час, при цьому залишаюсь дуже легким у реалізаціїї методом.

Таблиця 5.1 – Порівняння різних алгоритмів для задачі комівояжера

Метод	Середній шлях (48 міст)	Середній шлях (500 міст)	Час навчання (абсолютна величина)
ES-HyperNEAT	36442	60066	1723
Мурашиний алгоритм	38753	65152	2202
Бджолиний алгоритм	37955	61156	2163
k-opt-swap	62153	115789	5262

Для порівняння загального розв’язку задачі комівояжера різними алгоритмами за функцією придатності, на рисунку 5.8 представлений графік залежності функції пристосованості останнього покоління від кількості міст (чим менше, тим краще).

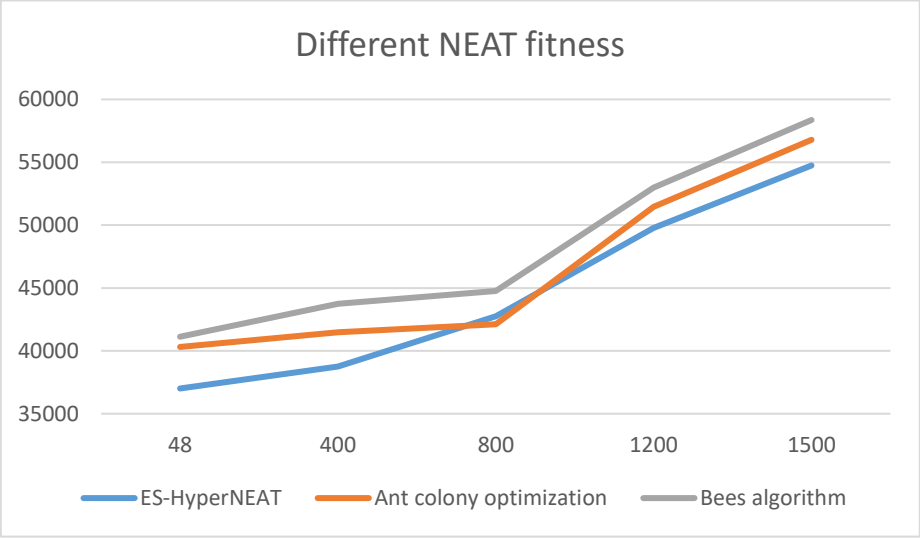

Рисунок 5.8 – Графік залежності значення функції пристосованості останнього покоління від кількості міст

ВИСНОВКИ

У цій роботі представлений новий підхід до автоматичного виведення геометрії вузлів на основі неявної інформації за схемою ваг з нескінченною роздільною здатністю. Еволюційний субстрат HyperNEAT не тільки змінює розташування кожного нейрона в мережі, але також може представляти області різної щільності, а це означає, що роздільна здатність може цілісно зростати в процесі еволюції. Щоб продемонструвати цей підхід, ES-HyperNEAT створив і еволюціонував контролер для вирішення задачі комівояжера. Аналіз результатів та еволюціонованих ANN показав, що покращена продуктивність зумовлена здатністю ES-HyperNEAT еволюціонувати ANN з частковим та цільовим зв'язком, розробляти існуючу структуру ANN та компенсувати рух інформації в межах базового гіперкуба. Крім того, ES-HyperNEAT може легше розвивати модульні ANN, якщо упереджений до локальності та певних канонічних топографій ANN. Таким чином, головний висновок полягає в тому, що ES-HyperNEAT – це перспективний новий підхід, який може створювати складні, регулярні та модульні ANN як функцію геометрії нейронної мережі.

NEAT вносить свій значний внесок, оскільки NEAT – це алгоритм, який втілює філософію інновацій. Загальний принцип полягає в тому, що молодим повинен бути наданий достатній шанс реалізувати свій потенціал, навіть якщо вони іноді зазнають невдачі на цьому шляху; точно так само в інтересах суспільства нові ідеї повинні бути ретельно перевірені та вдосконалені, перш ніж вони будуть прийняті або відкинуті. В чистому вигляді, відокремлюючи нову топологію мережі від іншої частини населення, так що він може конкурувати у своїй власній «ніші», захищає інновації і залишається вірним цієї філософії. Більше того, коли вся популяція поступово ускладняється шляхом додавання нових структур і створення нових видів, стає можливим знаходити нові рішення. Таким чином, NEAT є справжнім алгоритмом інновацій.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Kenneth O. Stanley, Risto Miikkulainen. Evolving Neural Networks Through Augmenting Topologies. Evolutionary Computation, 2002. 99–127 p.

2. Kenneth O. Stanley, Risto Miikkulainen. Efficient Reinforcement Learning Through Evolving Neural Network Topologies. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002), 2002.

3. Kenneth O. Stanley; Bobby D. Bryant, Risto Miikkulainen. Evolving Adaptive Neural Networks with and without Adaptive Synapses. Proceedings of the 2003 IEEE Congress on Evolutionary Computation (CEC-2003). 2003.

4. Colin Green. Phased Searching with NEAT: Alternating Between Complexification And Simplification, 2004.

5. T. Aaltonen et al. Measurement of the top quark mass with dilepton events selected using neuroevolution at CDF. Physical Review Letters, 2009.

6. Risi, Sebastian; Stanley, Kenneth O. An Enhanced Hypercube-Based Encoding for Evolving the Placement, Density and Connectivity of Neurons. Artificial Life. 2012. 331–363 p.

7. P. J. Angeline Morphogenic evolutionary computations: Introduction, issues and examples. In J. R. McDonnell, R. G. Reynolds, and D. B. Fogel, editors, Evolutionary Programming IV: The Fourth Annual Conference on Evolutionary Programming, pages 387–401. MIT Press, 1995.

8. P. J. Angeline, G. M. Saunders, and J. B. Pollack. An evolutionary algorithm that constructs recurrent neural networks. Neural Networks, IEEE Transactions on, 5(1):54–65, 1994.

9. Holland, John. Adaptation in Natural and Artificial Systems. Cambridge, MA: MIT Press. 1992. ISBN 978-0262581110.

10. J. S. Astor and C. Adami. A developmental model for the evolution of artificial neural networks. Artificial Life, 6(3):189–218, 2000.

11. P. J. Bentley and S. Kumar. The ways to grow designs: A comparison of embryogenies for an evolutionary design problem. In Proceedings of the

Genetic and Evolutionary Computation Conference (GECCO-1999), pages 35–43, San Francisco, 1999. Kaufmann.

12. Бодянский Е. В. Искусственные нейронные сети: архитектуры, обучение, применения. Харьков ТЕЛЕТЕХ, 2004. 372 с.

13. J. Bongard. Evolving modular genetic regulatory networks. In Proceedings of the 2002 Congress on Evolutionary Computation (CEC-2002), volume 2, pages 1872 –1877, 2002.

14. J. Clune, B. E. Beckmann, C. Ofria, and R. T. Pennock. Evolving coordinated quadruped gaits with the HyperNEAT generative encoding. In Proceedings of the IEEE Congress on Evolutionary Computation (CEC-2009) Special Section on Evolutionary Robotics, Piscataway, NJ, USA, 2009. IEEE Press.

15. J. Clune, B. E. Beckmann, P. McKinley, and C. Ofria. Investigating whether hyperneat produces modular neural networks. In GECCO ‘10: Proceedings of the Genetic and Evolutionary Computation Conference, pages 635–642, New York, NY, USA, 2010. ACM.

16. J. Clune, K. Stanley, R. Pennock, and C. Ofria. On the performance of indirect encoding across the continuum of regularity. Evolutionary Computation, IEEE Transactions on, 15(3): 346–367, 2011.

17. G. Cybenko. Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals, and Systems, 2(4):303–314, 1989.

18. D. D’Ambrosio and K. O. Stanley. A novel generative encoding for exploiting neural network sensor and output geometry. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2007), New York, NY, 2007. ACM Press.

19. J. Drchal, J. Koutnık, and M. Snorek. HyperNEAT controlled robots learn to drive on roads in simulated environment. In Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2009). IEEE Press, 2009.

20. P. Durr, C. Mattiussi, and D. Floreano. Genetic representation and evolvability of modular neural controllers. IEEE Computational Intelligence

Magazine, 2010.

21. P. Eggenberger. Evolving Morphologies of Simulated 3d Organisms Based on Differential Gene Expression. Fourth European Conference on Artificial Life, 1997.

22. R. Finkel and J. Bentley. Quad trees: A data structure for retrieval on composite keys. Acta informatica, 4(1):1–9, 1974.

23. D. Floreano, P. D¨urr, and C. Mattiussi. Neuroevolution: from architectures to learning. Evolutionary Intelligence, 1(1):47–62, 2008.

24. Moriarty, D. E., and Miikkulainen, R. Forming neural networks through efficient and adaptive co-evolution. Evolutionary Computation, 1997.

25. J. Gauci and K. O. Stanley. Autonomous evolution of topographic regularities in artificial neural networks. Neural Computation., 22:1860–1898, 2010.

26. C. Green. SharpNEAT homepage. http://sharpneat.sourceforge.net/, 2003–2006.

27. S. Harding, J. F. Miller, and W. Banzhaf. Developments in cartesian genetic programming: self-modifying cgp. Genetic Programming and Evolvable Machines, 11(3-4):397–439, 2010.

28. G. S. Hornby and J. B. Pollack. Creating high-level components with a generative representation for body-brain evolution. Artificial Life, 8(3), 2002.

29. E. R. Kandel, J. H. Schwartz, and T. M. Jessell, editors. Principles of Neural Science. Elsevier, Amsterdam, third edition, 1991.

30. Angeline, P. J., Saunders, G. M., and Pollack, J. B. An evolutionary algorithm that constructs recurrent neural networks. IEEE Transactions on Neural Networks, 5:54–65, 1993.

31. G. M. Khan, J. F. Miller, and D. M. Halliday. Evolution of Cartesian Genetic Programs for Development of Learning Neural Architecture. Evolutionary Computation, 19(3):469–523, 2011.

32. J. Lehman and K. O. Stanley. Improving evolvability through novelty search and selfadaptation. pages 2693–2700, Piscataway, NJ, 2011. IEEE Press.

33. Hancock, P. J. B. Genetic algorithms and permutation problems: A comparison of recombination operators for neural net structure specification. IEEE Computer Society Press, 1992.

34. H. Lipson. Principles of modularity, regularity, and hierarchy for scalable systems. Journal of Biological Physics and Chemistry, 7(4):125, 2007. ISSN 1512-0856.

35. A. P. Martin. Increasing genomic complexity by gene duplication and the origin of vertebrates. The American Naturalist, 154(2):111–128, 1999.

36. E. Mjolsness, D. H. Sharp, and J. Reinitz. A connectionist model of development. Journal of Theoretical Biology, 152:429–453, 1991.

37. M. Pigliucci. Is evolvability evolvable? Nature Reviews Genetics, 9(1):75–82, 2008.

38. T. Reil and P. Husbands. Evolution of central pattern generators for bipedal walking in a real-time physics environment. IEEE Trans. Evolutionary Computation, 6(2):159–168, 2002.

39. J. Reisinger, K. O. Stanley, and R. Miikkulainen. Towards an empirical measure of evolvability. In Genetic and Evolutionary Computation Conference (GECCO2005) Workshop Program, pages 257–264, Washington, D.C., 2005. ACM Press.

40. S. Risi and K. O. Stanley. Indirectly encoding neural plasticity as a pattern of local rules. In S. Doncieux, B. Girard, A. Guillot, J. Hallam, J.-A. Meyer, and J.-B. Mouret, editors, From Animals to Animats 11, volume 6226 of Lecture Notes in Computer Science, pages 533–543. Springer Berlin / Heidelberg, 2010.

41. S. Risi, J. Lehman, and K. O. Stanley. Evolving the placement and density of neurons in the hyperneat substrate. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2010), pages 563–570, 2010.