МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ

Навчальний посібник

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня магістра
за освітньою програмою «Системне програмування та спеціалізовані комп‘ютерні системи»
спеціальності 123 Комп‘ютерна інженерія

Укладачі: Ю.Є.Боярінова, О.О.Кучмій

Електронне мережне навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2022

Рецензент Хіцко Я.В., кандидат технічних наук, старший викладач кафедри ПЗКС ФПМ КПІ ім. Ігоря Сікорського.

Відповідальний редактор *Тарасенко-Клятченко О.В.*, кандидат технічних наук, доцент кафедри СПСКС ФПМ КПІ ім. Ігоря Сікорського.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол №1 від 02.09.2022 р.)
за поданням Вченої ради факультету прикладної математики
(протокол № 1 від 01.09.2022 р.)

Навчальний посібник розроблено для ознайомлення студентів з дисципліною «Системи штучного інтелекту», яка навчається за спеціальністю 123 «Комп`ютерна інженерія» освітньо-наукової та освітньо -професійної програми «Системне програмування та спеціалізовані комп`ютерні системи» кафедри системного програмування і спеціалізованих комп`ютерних систем факультету прикладної математики КПІ ім. Ігоря Сікорського.

Посібник «Системи штучного інтелекту» містить відомості про метаевристичні алгоритми, алгоритми кластеризації, відомості про нейромережі

Обсяг 7,3 авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Перемоги, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів
і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

ЗМІСТ

ПЕРЕДМОВА 5
ВСТУП 6
1.МЕТАЕВРИСТИЧНІ МЕТОДИ 8
1.1. Метод рою часток 9
1.2.Метод «кажанів» 24
1.2.1. Рух віртуальних кажанів 26
1.2.2. Гучність і випромінення 27
1.2.3. Модифікований метод кажанів 28
1.3. Метод зозулі 29
1.3.1. Модифікований метод зозулі 31
1.4. Метод світлячків 34
1.4.1. Модифікований метод світлячків 40
1.5. Метод мурах 42
1.5.1. Постановка задачі 43
1.5.2. Поведінка мурах 45
1.5.3. Вдосконалення мурашиного алгоритму 52
1.5.4. Мурашиний алгоритм для розв'язання задач безперервної оптимізації 55
1.6. Метод бджолиної колонії 60
1.6.1. Модифікована версія МБК 66
2. МЕТОДИ КЛАСТЕРІЗАЦІЇ 69
2.1. Кероване та некероване навчання 70
2.2. Види кластерізації 72
2.3. Базовий алгоритм k-means 73
2.3.1. Поділ k-means 77
2.3.2. k-means++ 78
2.4. Агломеративні методи AGNES (Agglomerative) 80
2.5. Однолінійна кластерізація 81
2.6. Повнозв'язана кластерізація 81
2.7. Метод найближчого сусіда або метод одиночного зв'язку 82
2.8. Метод найбільш віддалених сусідів або метод повного зв'язку 82
2.9. Метод зваженого попарного середнього(WPGMA) 83
2.10.Метод невиваженого попарного середнього(UPGMA) 83
3. ШТУЧНІ НЕЙРОННІ МЕРЕЖІ 84
3.1. Історія нейронних мереж 84
3.2. Застосування нейромереж для вирішення практичних задач 86
3.3. Задачі штучних нейронних мереж 98

3.4. Проблема класифікації та категоризації 99
3.5. Архітектура штучних нейронних мереж 101
3.6. Нейронна мережа Кохонена 105
3.7. Нейронні мережі зустрічного поширення 107
3.8. Рекурентні структури 108
3.9. Нейронна мережа Хопфілда 109
Асоціативна пам'ять 113
Двонаправлена асоціативна пам'ять 114
4. ГЕНЕТИЧНІ АЛГОРИТМИ 117
4.1. Генерація популяції 117
4.2. Схрещення 118
4.3. Мутації 129
4.4. Селекція 130
4.5. Обмін 133
4.6. Припинення 133
ДОДАТОК А 134
Чисельне наближення переміщення Леві (Levy Flight) 134
Алгоритм Мантегни 137
Спрощена версія алгоритму 138
Спрощена версія алгоритму з P-best 138
ДОДАТОК Б 139
Метод Box-Muller: 139
ДОДАТОК В 141
Покрокове пояснення ABC 141
ДОДАТОК Г 145
Приклад ітерації 145
ДОДАТОК Д 148
Метод найбільш віддалених сусідів або метод повного зв'язку 149
Метод зваженого попарного середнього (WPGMA) 151
Метод незваженого попарного середнього (UPGMA) 154
ЛІТЕРАТУРНІ ДЖЕРЕЛА 158

ПЕРЕДМОВА

Навчальне видання призначене для студентів, які вивчають дисципліну «Системи штучного інтелекту» з циклу професійної підготовки спеціальності 123 «Комп‘ютерна інженерія» за освітньою програмою Системне програмування та спеціалізовані комп’ютерні системи», що спрямована на використання нових інформаційних технологій, моделювання складних систем, систем штучного інтелекту і для вирішення інших задач професійної діяльності.

Навчальне видання є опорним конспектом лекцій для дисципліни «Системи штучного інтелекту».

Метою викладання дисципліни є оволодіння основними поняттями і методами, що необхідні для формування світогляду на створення та моделювання систем штучного інтелекту з використанням нейронних мереж та метаевристичних і генетичних алгоритмів

Навчальне видання містить викладання метаевристичних алгоритмів, алгоритмів кластеризації та відомості про нейронні мережі.

Навчальне видання має стати в нагоді під час опрацювання матеріалів лекцій, підготовки до екзамену, до виконання лабораторних робіт.

ВСТУП

Штучний інтелект – один з найперспективніших напрямків інформаційних технологій, який вивчає методи розв'язання задач, для яких не існує звичайних способів вирішення. Системи штучного інтелекту можуть оперувати даними та самонавчатися. Сфери застосування таких систем є різноманітними. Значна кількість сучасних інформаційних систем уже містить елементи штучного інтелекту, які переконливо доводять свою ефективність при розв’язанні складних задач сьогодення. Разом із тим, можливості інтелектуальних технологій є набагато більшими, їх створення відкриватиме нові перспективи розвитку суспільства.

Розвиток сучасних систем штучного інтелекту розпочався з 50-х років XX століття. Роботи того періоду поклали початок першого етапу досліджень в галузі штучного інтелекту, пов’язаного з розробкою програм, які розв’язують задачі на основі використання різноманітних евристичних методів.

Сьогодні розвиток фундаментальних досліджень в галузі штучного інтелекту передбачає вирішення зокрема таких проблем: автоматизоване створення програмних продуктів, автоматизований переклад, інформаційний пошук, генерація документів, обробка та сприйняття природної мови та тексту, системи технічного зору та розпізнавання образів, створення баз знань, створення експертних систем.

Штучний інтелект отримав широке розповсюдження у всіх сферах людської діяльності. Можна виділити наступні напрями їх застосування: доказ теорем; розпізнавання зображень; машинний переклад і розуміння людської мови; ігрові програми; експертні системи та інші.

Також можна виділити проблеми, які пов'язані подальшим розвитком та застосуванням систем штучного інтелект: проблема визначення штучного інтелекту, проблема визначення завдань штучного інтелекту, проблема безпеки.

Отже, можна сказати, що системи штучного інтелекту на сьогодні відіграють велику роль в розвитку науки та техніки. Вони охоплюють безліч сфер людської діяльності, виконуючи різні завдання і мають велике підґрунтя для подальшого розвитку.

1. МЕТАЕВРИСТИЧНІ МЕТОДИ

Задачі комбінаторної оптимізації інтригують, оскільки можна легко описати постановку задачі, але дуже важко їх вирішити. Багато проблем, що виникають у програмах, є NP-складними задачами, тобто вважається, що вони не можуть бути оптимально вирішені в межах поліноміально обмеженого часу. Отже, щоб практично вирішувати такі випадки, часто доводиться використовувати приблизні методи, які повертають розв'язки, близькі до оптимальних за відносно короткий час. Методи цього типу називають евристичними[1]. Вони часто використовують певні специфічні знання, щоб знаходити або вдосконалювати рішення.

Останнім часом багато дослідників зосереджують свою увагу на новому класі методів, що називаються метаевристичними [2,3]. Метаевристика – це набір алгоритмічних понять, які можуть бути використані для визначення евристичних методів, що застосовуються для широкого кола різних завдань. Використання метаевристики значно підвищує можливість знаходження якісного розв'язку для складних, актуальних комбінаторних задач оптимізації за розумний час.

Інакше кажучи, метаевристичний метод може розглядатися як загальноприйнятий евристичний метод, призначений для вирішення конкретної евристичної задачі (наприклад, алгоритму локального пошуку) і може бути використаний для інших умов (наприклад, дослідження перспективних областей пошукового простору, що містять якісні розв'язки). Отже, метаевристика є загальною алгоритмічною обгорткою, яка може бути застосована до різних задач оптимізації з відносно невеликою кількістю модифікацій, щоб зробити її адаптованою до конкретної проблеми [4,5].

Особливо успішним є метаевристичні методи, натхненні поведінкою мурах, світлячків, зозулі та ін.

1.1. Метод рою часток

Метод рою часток(МРЧ) — метод чисельної оптимізації, для використання якого не потрібно знати точного градієнта оптимізованої функції. МРЧ був доведений Кеннеді, Еберхартом і Ші[6] і спочатку призначався для імітації соціальної поведінки. Алгоритм був спрощений, і було зауважено, що він придатний для виконання оптимізації[6-20].

Опишемо математичну структуру. Нехай $A \subset R^n$ – простір пошуку, а $f: A \to Y \subset R$ – цільова функція. Для максимально простого опису, ми припускаємо, що A є також можливим розв'язком функції, тобто не існує жодних явних обмежень на потенційні розв'язки. Зауважимо, що немає необхідності в додаткових припущеннях щодо форми цільової функції та простору пошуку. МРЧ – це метод на основі популяції, тобто він експлуатує популяцію потенційних розв'язків для одночасного точкового пошуку в просторі. Популяція називається *роєм*, його особи – *частками*, а нотації використовується як і для подібних моделей у соціальних науках та фізиці часток. Рій визначається як множина $S = \{x_1, x_2, \dots, x_N\}$ з N часток (потенційні розв'язки), кожна з яких має вигляд $x_i = \left(x_{i_1}, x_{i_2}, \dots, x_{i_n}\right)^T \subset A, i = 1, 2, \dots, N$.

Індекси довільно визначаються для часток, а N – визначений користувачем параметр алгоритму. Цільова функція $f(x)$ вважається доступною для всіх точок A. Отже, кожна частка має унікальну функціональну величину $f_i = f(x_i) \subset Y$.

Передбачається, що частки будуть рухатися в пошуковому просторі A ітераційно. Це досягається шляхом регулювання їх положення за допомогою

правильного зсуву розміщення, що називається швидкістю, і позначається як $v_i = \left(v_{i_1}, v_{i_2}, \dots, v_{i_n}\right)^T \subset A,\ i = 1, 2, \dots, N.$

Швидкість також адаптується ітераційно, щоб надати часткам здатності відвідувати будь-яку область в A. Якщо t – номер ітерації, то поточне положення частки i та її швидкість буде надалі позначатися як $x_i(t)$ і $v_i(t)$ відповідно.

Швидкість оновлюється на основі інформації, отриманої на попередніх кроках алгоритму. Це реалізується в пам'яті, де кожна частка може зберігати найкращі координати, в яких вона коли-небудь була за час пошуку. Для цього, крім рою S, який містить поточні розміщення часток, МРЧ містить також набір пам'яті $P = \{p_1, p_2, \dots, p_N\}$, який містить найкращі координати $p_i = \left(p_{i_1}, p_{i_2}, \dots, p_{i_n}\right)^T \subset A,\ i = 1, 2, \dots, N$, коли-небудь відвідані кожною часткою. Ці розміщення позначаються як: $p_i(t) = f_i(t)$, де t – номер ітерації.

МРЧ оснований на імітації моделі соціальної поведінки. Таким чином, існує механізм обміну інформацією, щоб частки могли взаємно повідомляти свій досвід. Алгоритм апроксимує глобальний мінімум за найкращими координатами, які коли-небудь відвідали всі частки. Тому це розумний вибір для обміну даною ключовою інформацією. Нехай g – індекс найкращих координат з найменшим значенням функції в P на ітерації t, тобто $p_i(t) = argmin\left(f_i(t)\right)$.

Тоді базову версію МРЧ можна описати наступними рівняннями:

$$\begin{aligned} v_{ij}(t+1) &= v_{ij}(t) + c_1 R_1 \left(p_{ij}(t) - x_{ij}(t)\right) + c_2 R_2 \left(p_{gj}(t) - x_{ij}(t)\right) \\ x_{ij}(t+1) &= x_{ij}(t) + v_{ij}(t+1) \end{aligned} \qquad (1.1)$$

Псевдокод алгоритму МРЧ:

На вході: кількість часток (N); рій (S); найкращі координати (P)

Крок 1. Встановити $t \leftarrow 0$

Крок 2. Ініціалізувати S і встановити $P = S$

Крок 3. Оцінити S та P, і визначити індекс g найкращих координат

Крок 4. While (критерій зупинки не виконується)

Крок 5. Оновити S за допомогою рівнянь (1) та (2)

Крок 6. Обчислити S

Крок 7. Оновити P та перевизначити індекс g

Крок 8. Встановити $t \leftarrow t + 1$

Крок 9. End While

Крок 10. Визначити найкращі координати

Тут $i = 1, 2, \ldots, N$, $j = 1, 2, \ldots, n$, t – номер ітерації; R_1 і R_2 – випадкові величини, рівномірно розподілені в межах $[0, 1]$; c_1, c_2 є ваговими факторами, які також називаються когнітивним та соціальним параметрами відповідно.

На кожній ітерації після оновлення та оцінки часток оновлюються найкращі розміщення (пам'ять). Таким чином, нове найкраще розміщення x_i на ітерації $t + 1$ визначається наступним чином

$$p_i(t+1) = \begin{cases} x_i(t+1), \text{якщо } f\big(x_i(t+1)\big) \le f\big(p_i(t)\big) \\ p_i(t), \text{інакше} \end{cases}. \qquad (1.2)$$

Нове визначення показника g для оновлених координат завершує ітерацію МРЧ. Частки, як правило, ініціалізуються довільним чином, згідно рівномірного розподілу в пошуковому просторі A. Цей варіант виконує однаковий пошук у кожній з областей A; тому в основному це краще в тих випадках, коли немає інформації про форму пошукового простору або цільову функцію, що потребує іншої схеми ініціалізації. Крім того, він реалізується досить легко, оскільки всі сучасні комп'ютерні системи можуть бути оснащені єдиним генератором випадкових чисел.

Попереднє значення швидкості $v_{ij}(t)$ у правій частині рівняння (1), забезпечує засіб інерційного руху до частки, беручи до уваги своє попереднє

положення. Ця властивість може запобігти тому, що частка «захопилася» б пошуком найкращої позиції і могла б потрапити до локального мінімуму.

Крім того, попереднє значення швидкості служить збудженням для найкращої частки x_g. Дійсно, якщо частка x_i виявляє нову позицію з нижчим значенням функції, ніж найкраще значення, то вона стає найкращою у рої (тобто $g \leftarrow i$), а її найкраща позиція p_i, буде збігатися з p_g та x_i на наступній ітерації. Таким чином, дві стохастичні частини в рівнянні (1) зникнуть. Якщо в рівнянні (1) не було попереднього значення швидкості, то вищезгадана частка залишатиметься в одному і тому ж положенні протягом кількох ітерацій, допоки нове найкраще положення не буде виявлено іншою часткою. На відміну від цього, доданок швидкості дозволяє цій частині продовжувати пошук в напрямі його колишнього зсуву розміщення.

Значення c_1 і c_2 можуть впливати на пошукову спроможність МРЧ шляхом зміщення напряму вибору нових позицій частки x_i, відповідно до кращих позицій x_i і p_g.

Очевидно що величина пошуку значно відрізняється у двох випадках. Якщо потрібне краще глобальне дослідження, тоді більші значення c_1 і c_2 можуть забезпечити нові точки у відносно далеких областях пошукового простору. З іншого боку, більш детальний локальний пошук за найкращими позиціями, досягнутими до теперішнього часу, потребує вибору менших значень для двох параметрів. Крім того, вибравши, $c_1 > c_2$, буде відбуватися рух часток у напряму p_i, тоді як у протилежному випадку, $c_1 < c_2$ буде направляти рух часток у напрямку p_g. Цей ефект може бути корисним у випадках, коли існує інформація щодо форми цільової функції. Наприклад, в опуклих унімодальних цільових функціях вибір, який сприяє руху часток у напрямку p_g, очікується,

буде більш ефективним, якщо його поєднувати з правильною пошуковою величиною.

Слід зауважити, що МРЧ діє на кожному напрямку координат самостійно. На цьому етапі ми згадаємо типову помилку, зроблену кількома дослідниками, особливо, коли в їхній векторній формі розглядаються рівняння МРЧ:

$$v_i(t+1) = v_i(t) + c_1 R_1 \left(p_i(t) - x_i(t)\right) + c_2 R_2 \left(p_g(t) - x_i(t)\right), \quad (1.3)$$

$$x_i(t+1) = x_i(t) + v_i(t+1), \quad (1.4)$$

де $i = 1, 2, \ldots, N$, де всі векторні операції виконуються покординатно. У цьому випадку R_1 та R_2 слід розглядати як випадкові n-вимірні вектори з їх компонентами, рівномірно розподіленими в межах [0,1]. Замість цього R_1 і R_2 часто розглядаються як випадкові одновимірні значення, подібно до рівняння (1.1), в результаті чого виходить схема, яка використовує ту ж довільну кількість для всіх компонентів напрямку відповідного різницевого вектора в рівнянні (1.3).

У більшості додатків для оптимізації бажано або неминуче розглянути лише частинки, що лежать в межах пошукового простору. Для цього границі накладаються на положення кожної частки x_i, щоб обмежити її в пошуковому просторі A. Якщо частка, що виконує надто великий крок з пошукового простору після застосування рівняння (1.2), то вона негайно повертається на границю. У найпростішому випадку, де простір пошуку можна визначити як

$$A = [a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_n, b_n], \quad (1.5)$$

З $a_i, b_i \in R$, $i = 1,2, \ldots, n$ частки мають обмеження

$$x_{ij}(t+1) = \begin{cases} a_j, \text{якщо } x_{ij}(t+1) < a_j \\ b_j, \text{якщо } x_{ij}(t+1) > b_j \end{cases},$$

де $i = 1,2, \ldots, N$, $j = 1,2, \ldots, n$.

В іншому випадку, розглядається рух зі «стрибанням» з-за границі назад до пошукового простору, подібно до м'яча, що відбивається від стіни. Популярність цього підходу обмежена, оскільки він вимагає моделювання руху часток з комплексними фізичними рівняннями. У тих випадках, коли A не можна визначити як простір кубічного вигляду, для обмеження часток можуть знадобитися спеціальні умови, що залежать від функції для оптимізації.

Покращення МРЧ

Представлений варіант МРЧ задовільний для простих задач оптимізації. Однак їх суттєві дефекти виявляються, якщо їх застосувати для складніших функцій на більших просторах пошуку і з великою кількістю локальних мінімумів.

1.1.1. «Вибух рою» і обмеження швидкості

Першою значною проблемою, підтвердженою кількома дослідниками, був ефект «вибуху рою» (*swarm explosion*). Він відноситься до неконтрольованого збільшення швидкостей, що призводить до різкого розходження часток у різні сторони. Цей дефект пов'язаний з відсутністю механізму скорочення швидкостей в попередніх варіантах МРЧ, і він легко вирішується, використовуючи суворі границі для обмеження швидкості на вибраних етапах, запобігаючи надмірним крокам від їх поточної позиції.

Користувачем визначається порогова гранична швидкість, $v_{max} > 0$. Після визначення нової швидкості кожної частки за допомогою (1.1) для оновлення позиції за допомогою (2) застосовуються наступні обмеження:

$\left|v_{ij}(t+1)\right| \leq v_{max}, i = 1,2,\dots,N, j = 1,2,\dots,n.$

У випадку виходу за границю, значення встановлюється на цій границі:

$$v_{ij}(t+1) = \begin{cases} v_{max}, \text{якщо } v_{ij}(t+1) > v_{max} \\ -v_{max}, \text{якщо } v_{ij}(t+1) < -v_{max} \end{cases}. \quad (1.6)$$

При необхідності можуть бути використані різні границі для різних напрямків. Значення v_{max} зазвичай вибирається як частина пошукового простору для кожного напрямку. Таким чином, якщо пошуковий простір визначається за допомогою (5), то загальна максимальна швидкість для всіх компонентів напрямків може бути визначена як

$$v_{max} = \frac{\min_i\{b_i - a_i\}}{k}.$$

Іншим способом задання лімітів максимальних швидкостей покомпонентно є формула:

$$v_{max,i} = \frac{b_i - a_i}{k},$$

де $i = 1,2, \dots, n$.

При $k = 2$ формула має загальний випадок. Якщо функція вимагає менших кроків часток, то слід використовувати більш високі значення k. Наприклад, якщо в пошуковому просторі є безліч локальних мінімумів з вузькими областями притягання одне до одного, тоді k має прийняти розумні великі значення, щоб запобігти «перестрибування» частками даних областей. З іншого боку, значення k не повинно бути дуже малим чи великим, і не повинно перешкоджати задовільному процесу пошуку.

Обмеження швидкості являє собою просте, але ефективне вирішення проблеми «вибуху рою». Однак воно не вирішує проблему конвергенції. Частки тепер можуть виконувати пошук навколо своїх кращих позицій, але вони не можуть ні досягти конвергенції для перспективної позиції, ні виконувати покращений пошук навколо неї. Ця проблема була вирішена шляхом введення нового параметра в оригінальну модель МРЧ, як це описано в наступній частині.

1.1.2. Концепція інерції швидкості

Хоча використання порогу максимальної швидкості покращило ефективність ранніх варіантів МРЧ, цього недостатньо, щоб зробити алгоритм ефективним для складних задач оптимізації. Незважаючи на полегшення з «вибухом рою», рій не може зосередити свої частки навколо найбільш перспективних рішень на останній фазі процедури оптимізації. Таким чином, навіть якщо перспективний район пошукового простору був випадково виявлений, жодної подальшого доопрацювання не буде, оскільки частки замість цього коливаються на широких траєкторіях навколо своїх найкращих позицій.

Причиною цього дефіциту є неможливість контролювати швидкість. Детальний пошук в перспективних регіонах, тобто навколо кращих позицій, вимагає максимального залучення часток в них та невеликих переміщень, які забороняють рухатися за межами околиці. Це досягається шляхом зменшення збуджень, що переміщують частки від кращих позицій; ефект, пов'язаний з попереднім значенням швидкості у рівнянні (1.1). Отже, вплив попереднього значення швидкості на поточний буде зникати для кожної частки. З цією метою у рівнянні (1) вводиться новий параметр w, що називається інертною вагою, в результаті чого з'являється новий варіант МРЧ

$$v_{ij}(t+1) = wv_{ij}(t) + c_1R_1\left(p_{ij}(t) - x_{ij}(t)\right) + c_2R_2\left(p_{gj}(t) - x_{ij}(t)\right), \qquad (1.7)$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1), \qquad (1.8)$$

$i = 1,2,\dots,N, j = 1,2,\dots,n.$

Решта параметрів залишається такою самою, як і для попередніх варіантів рівнянь (1) та (2). Вага інерції повинна вибиратися таким чином, щоб вплив $v_{ij}(t)$ зникав під час виконання алгоритму. Таким чином, бажано зменшувати

значення w з часом. Дуже поширеним способом є ініціалізація w значенням трохи більше 1 (наприклад 1,2) для сприяння розвідуванню на ранніх етапах оптимізації та лінійне зменшення до нуля, щоб усунути коливальну поведінку на пізніх стадіях пошуку. Зазвичай, строго додатна нижня межа w (наприклад 0,1) використовується для запобігання зникнення попереднього значення швидкості у формулі. Загалом, лінійне зменшення значення w може бути математично описане наступним чином

$$w(t) = w_{up} - \left(w_{up} - w_{low}\right)\frac{t}{T_{max}}, \qquad (1.9)$$

де t – номер ітерації; w_{low} та w_{up} – необхідні нижні та верхні значення w; і T_{max} – загальна кількість ітерацій. Рівняння (1.9) визначає інертну вагу, що лінійно зменшується за часом з початковим значенням w_{up} на ітерації $t = 0$ і кінцеве значення w_{low} на останній ітерації $t = T_{max}$.

Крім того, ми спостерігаємо, що при використанні інерції ваги спостерігається збільшення різноманітності переміщень рою протягом перших майже 100 ітерацій. Цей ефект можна пояснюється початковим значенням $w_{up} = 1{,}2$. Оскільки це значення більше 1, то попереднє значення швидкості має більший вплив у рівнянні (1.7), ніж у рівнянні (1.1). Це призводить до тимчасового «вибуху рою», що покращує можливості розвідки навіть у випадку поганої ініціалізації. Після майже 90 ітерацій вага інерції набуває значень менших за 1, і різноманітність починає знижуватися до нуля, тим самим сприяючи більш ретельному пошуку. Як обговорюється в наступному розділі, різні схеми адаптації ваги інерції можуть бути використані для отримання різних поведінок алгоритму.

Неймовірне покращення продуктивності, отримане за допомогою даного варіанту МРЧ, зробило його найпопулярнішим серед версій алгоритму протягом

багатьох років. Проте, хоч частки змогли уникнути «вибуху рою» та наблизитись до найкращих позицій, вони все ще легко потрапляють до локальних мінімумів, особливо для складних функцій. Цей недолік вирішено шляхом впровадження більш складної схеми обміну інформацією між частками, як це описано в наступній частині.

1.1.3. Концепція околиць

Використання інерційної ваги надало МРЧ можливостей конвергенції, як описано в попередньому розділі. Однак цього не достатньо для підвищення його ефективності до задовільних значень у складних, мультимодальних середовищах. Може статися, що після ряду ітерацій, рій «згортається» через повну втрату різноманіття координат. Це означає, що подальше дослідження неможливе, і частинки можуть виконувати лише локальний пошук навколо свого розміщення, що, найімовірніше, знаходиться в околиці загальної найкращої позиції. Хоча ефект швидкої конвергенції може бути незначним для простих функцій для оптимізації, особливо в унімодальних та опуклих функціях. Ефект стає значним у високорозмірних, складних середовищах.

Цей дефект можна віднести до глобальної схеми обміну інформацією, яка дозволяє кожній частці миттєво дізнатися загальне найкраще місце на кожній ітерації. Використовуючи цю схему, всі частинки отримують нові позиції в регіонах, пов’язаних із тим самим загальним найкращим положенням, зменшуючи розвідувальні можливості рою.

Вищезгадана проблема вирішена шляхом введення поняття околиць. Головною ідеєю є зменшення глобальної схеми обміну інформацією до локальної, де інформація розповсюджується лише на невеликих ділянках рою на кожній ітерації. Точніше, кожна частка вважає сукупність інших часток

сусідніми і на кожної ітерації повідомляє найкраще розташування лише для цих частинок, а не для всього рою. Таким чином, інформація про загальну найкращу позицію спочатку повідомляється лише в околиці кращої частки, а потім вже до решти через своїх сусідів і сусідів сусідів.

Для формального опису, нехай x_i – i-та частка рою $S = \{x_1, x_2, \dots, x_N\}$. Тоді окремо x_i визначається як множина $NB_i = \{x_{n_1}, x_{n_2}, \dots, x_{n_S}\}$, де $\{n_1, n_2, \dots, n_S\} \subseteq \{1,2, \dots, N\}$ – це множина індексів власних сусідів. Потужність $|NB_i|$ цієї множини називається *розміром околиці (області, району)*. Якщо g_i позначає індекс найкращої частки в NB_i, тобто $p_{g_i} = \underset{j:\, x_j \in NB_i}{\operatorname{argmin}} f(p_j)$, то рівняння (7) і (8) набувають вигляду:

$$v_{ij}(t+1) = wv_{ij}(t) + c_1 R_1 \left(p_{ij}(t) - x_{ij}(t)\right) + c_2 R_2 \left(p_{g_{ij}}(t) - x_{ij}(t)\right), \qquad (1.10)$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1), \qquad (1.11)$$

$i = 1,2, \dots, N, j = 1,2, \dots, n.$

Єдина відмінність між рівняннями (1.7) та (1.10) – це індекс найкращого положення у другому доданку. Відповідно до рівняння (1.10), частка буде рухатися у напрямку до свого кращого положення, а також найкращого положення своїх сусідів, а не загальної найкращої позиції в рівнянні (1.7).

Схема визначення сусідів кожної частки називається *топологією околиці*. Загальною схемою, що спадає на думку, є формування околиць на основі фактичних відстаней часток у пошуковому просторі. Відповідно до цього, для кожної частки буде призначено околицю, що складається з s частинок, що знаходяться найближче до його поточної позиції. Хоч це й здається простим, дана схема вимагає обчислення $N(N+1)/2$ відстаней між частками на кожній ітерації. Обсяги обчислень для даного підходу можуть стати неприйнятними, для великої кількість часток. Більше того, підхід демонструє загальну тенденцію

утворення кластерів часток, які можуть легко потрапити до локальних мінімумів. З цих причин специфічна топологія околиць не є перспективним рішенням.

Ідея формування околиць на основі довільних критеріїв пропонується з метою спрощення ефектів від кластеризації часток, що отримуються для топологій околиць за відстанями. Найпростішою і найбільш використовуваною альтернативою є утворення околиць на основі індексів частинок. Відповідно до цього, i-та частка вважає за сусідів частки із сусідніми індексами. Таким чином, околиця x_i може бути визначена як $NB_i = \{x_{i-r}, x_{i-r+1}, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_{i+r-1}, x_{i+r}\}$, так ніби частки розміщенні на колі, і кожна з них пов'язана лише зі своїми найближчими сусідами. Ця схема показана на рисунку 1а, і вона називається кільцевою топологією, тоді як параметр r, який визначає розмір околиці, називається радіусом кола. Очевидно, що в цій топології індекси є циклічними, тобто індекс $i = 1$ знаходиться відразу за індексом $i = N$ на колі.

Варіант МРЧ, що використовує загальну найкращу позицію рою, можна розглядати як особливий випадок вищезгаданої кільцевої схеми, де кожна околиця являє собою весь рій, тобто $NB_i = S$, для всіх $i = 1,2, \dots, N$. Для того, щоб розрізняти два підходи, схема, яка використовує загальну найкращу позицію, називається глобальним варіантом МРЧ (часто позначається як gbest – global best), тоді як той, що має суто менші околиці, називається локальним варіантом МРЧ (позначається, відповідно, lbest – local best). Схема gbest також називають топологією зірки, і її графічне представлення зображено на рисунку 1б, і всі частки обмінюються інформацією з найкращою.

а б

Рисунок 1. Топології МРЧ (а – топологія «коло» б – топологія «зірка»)

Використання околиць значно покращило ефективність МРЧ, та додало натхнення для досліджень у напрямку більш складних варіантів, які включають все описане вище. У наступній частині ми представляємо найактуальніші варіанти МРЧ, які широко використовуються в програмах і є найсучаснішими для нашого часу.

1.1.4. Сучасний варіант МРЧ

Аналіз Клерка та Кеннеді (Clerc and Kennedy) представив тверде теоретичне обґрунтування алгоритму, і він встановили одну з досліджених моделей як сучасний варіант МРЧ. Ця модель визначається наступними рівняннями

$$v_{ij}(t+1) = \chi\left[v_{ij}(t) + c_1 R_1\left(p_{ij}(t) - x_{ij}(t)\right) + c_2 R_2\left(p_{gj}(t) - x_{ij}(t)\right)\right], \quad (1.11)$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1), \quad (1.12)$$

$i = 1,2,\dots,N, j = 1,2,\dots,n.$

де χ – параметр, що називається коефіцієнтом стиснення, а решта параметрів залишаються такими ж, як і для описаних вище моделей МРЧ. Очевидно, цей варіант МРЧ алгебраїчно еквівалентний варіанту інерційної ваги, визначеної рівняннями (7) та (8). Однак він вирізняється в літературі завдяки

своїм теоретичним властивостям, що передбачає наступне явне задання його параметрів (Clerc & Kennedy, 2002):

$$\chi = \frac{2}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|},$$

де $\varphi = c_1 + c_2$ і $\varphi > 4$. Виходячи з даного рівняння значення параметрів $\chi = 0{,}729$ та $c_1 = c_2 = 2{,}05$ на даний момент вважаються значеннями за замовчуванням.

Оновлення швидкості рівняння (11) відповідає моделі gbest. Природно, що концепцію околиць можна також використати, замінивши $p_{gj}(t)$ в другому доданку рівняння (11) на відповідне локальну найкращу позицію. Таким чином рівняння (11) набуває вигляду

$$v_{ij}(t+1) = \chi\left[v_{ij}(t) + c_1 R_1\left(p_{ij}(t) - x_{ij}(t)\right) + c_2 R_2\left(p_{g_{ij}}(t) - x_{ij}(t)\right)\right], \quad (1.13)$$

де g_i позначає індекс найкращої частки в околиці x_i. Використання рівняння (13) замість рівняння (11) має такий же вплив на властивості розвідки як і раніше описаний метод інерційної ваги.

Стохастичні параметри R_1 і R_2 вважаються рівномірно розподіленими в діапазоні $[0{,}1]$. Таким чином, перспективні нові позиції частки розподіляються в прямокутнику. Також R_1 і R_2 можуть бути рівномірно розподілені всередині сфер, центрами яких є розв'язки, з радіусом, який дорівнює 1. Інший метод стверджує, що R_1 і R_2 розподіляються за розподілом Гауса $N(\mu, \sigma^2)$. У цьому випадку розподіл нових перспективних позицій може суттєво відрізнятися від попередніх двох випадків, значною мірою залежно від значень μ та σ, які, у свою чергу, зазвичай залежать від параметрів c_1 і c_2.

З іншого боку, метод розподілу Гауса представляє абсолютно іншу перевагу алгоритму. Залежно від параметрів розподілу, частка може рухатися навіть у напрямку, протилежному до найкращої позиції, що демонструє зовсім

іншу динаміку, ніж стандартні моделі МРЧ. Ефективність цієї моделі завжди залежить від функції. Якщо МРЧ виявив найбільш перспективні регіони пошукового простору, то розподіли Гауса можуть сповільнити зближення. З іншого боку, якщо існує безліч локальних мінімумів з вузькими привабливими областями, розташованих на глобальному рівні, то даний підхід може позитивно впливати на алгоритм. Тим не менше, вибір розподілу має очевидний вплив на продуктивність; таким чином, необхідно ретельно вибирати. У даній задачі експериментальні докази у вирішенні аналогічних проблем можуть бути важливим критерієм для такого вибору.

Псевдокод розподілу:

function CVT (**integer** k) **returns** множину з k CVT генераторів
Вбрати k довільних точок $\{g_i\}_{i=1}^{k}$ як початкові генератори
do {
    Вибрати q довільних точок
    for i = 1 to k {
        Зібрати до G_i підмножину з точок q, таких що
        ближчі до g_i ніж до інших генераторів
        Обчислити середні координати a_i точок G_i
        Перемістити g_i на певний відсоток відстані ближче до a_i
    }
} until досягнуто критерію виходу
return $\{g_i\}_{i=1}^{k}$

1.2. Метод «кажанів»

Один з метаевристичних алгоритмів, що моделює поведінку кажанів. Цей алгоритм є підвидом ройового інтелекту. Алгоритм розроблено в 2010 році Янгом[21]. Одна з переваг алгоритму – його швидкість.

Кажани є абсолютно унікальними комахоїдними тваринами. Більшість кажанів володіє досконало ехолокацією, котрі вони використовують для пошуку здобичі, пересування в абсолютній темряві. Кажани користуються ехолокацією для того, щоб полювати, уникати перешкод та знайти ночівлю в темряві. Вони видають гучний звуковий імпульс і слухають відлуння, що відбивається від оточуючих предметів.

Якщо ми ідеалізуємо деякі ехолокаційні характеристики кажанів, ми можемо розвивати різні методи, в основі яких знаходиться їх поведінка.

Для простоти, ми будемо використовувати наступні ідеалізовані правила:

1. Всі кажани використовують ехолокацію для вимірювання дистанції і вони також «знають» різницю між їжею/здобиччю та фоновими бар'єрами в якийсь магічний спосіб;
2. Кажани літають випадково зі швидкістю v_i, положенням x_i з фіксованою частотою f_{min} та змінними довжиною хвилі λ і гучністю A_0 для пошуку здобичі. Вони можуть автоматично коригувати довжину хвилі (або частоту) своїх імпульсивних випромінювань та регулювати частоту імпульсного випромінювання $r \in [0,1]$ залежно від відстані до цілі;
3. Незважаючи на те, що гучність може змінюватись багатьма способами, ми припускаємо, що вона змінюється від великого (додатного) A_0 до мінімального постійного значення A_{min}.

Для нашої задачі ми можемо використовувати будь-яку довжину хвилі для зручності. У фактичній реалізації ми можемо налаштувати діапазон шляхом

регулювання довжин хвиль (або частоти), а також діапазон, який можна виявити (або найбільшу довжину хвилі) такий, що на початку має величину як і розмір області пошуку, а потім зменшується до менших значень. Крім того, ми не обов'язково повинні використовувати довжини хвиль, замість цього ми можемо змінювати частоту при фіксованій довжині хвилі λ. Це можливо завдяки тому, що λ та f пов'язані залежністю, і значення λf буде сталим. Ми будемо використовувати цей підхід в нашій реалізації.

Для простоти можна вважати, що $f \in [0, f_{max}]$. Ми знаємо, що вищі частоти мають коротші довжини хвиль і сягають коротших відстаней. Для кажанів типові відстані – це кілька метрів. Величина імпульсу може бути в діапазоні $[0, 1]$, де 0 означає відсутність імпульсів, а 1 – максимальна величина їх випромінення.

На основі цих наближень та ідеалізацій, основні кроки методу кажанів (МК) можна узагальнити наступним псевдокодом:

Цільова функція – $f(x)$, $x = (x_1, x_2, \dots, x_d)^T$
Виконати ініціалізацію популяції кажанів x_i $(i = 1, 2, \dots, N)$ та v_i
Обчислити частоту f_i для x_i
Виконати ініціалізацію випромінення r_i та гучності A_i
while (t < Максимальна кількість ітерацій)
    Згенерувати нові розв'язки шляхом регулювання частоти,
    та оновлення швидкостей і координат/розв'язків згідно (1)-(3)
    if ($rand > r_i$)
    Вибрати розв'язок серед найкращих розв'язків
        Згенерувати локальний розв'язок в околі вибраного найкращого
        розв'язку
    end if
    Згенерувати нове рішення за допомогою довільного перельоту (*)
    if ($rand < A_i$ & $f(x_i) < f(x_*)$)
    Прийняти новий розв'язок
    Збільшити r_i та зменшити A_i
    end if
    Виконати рангування кажанів та визначити найкращого на даний момент x_*
end while

1.2.1. Рух віртуальних кажанів

Для моделювання ми використовуємо віртуальних кажанів. Ми повинні визначити правила оновлення їх координат x_i та швидкостей v_i в d-вимірному просторі. Нові координати x_i^t та швидкості v_i^t в момент часу t обчислюються як

$$f_i = f_{min} + (f_{max} - f_{min})\beta, \quad (1.14)$$

$$v_i^t = v_i^{t-1} + (x_i^t - x_*)f_i, \quad (1.15)$$

$$x_i^t = x_i^{t-1} + v_i^t, \quad (1.16)$$

де $\beta \in [0,1]$ – довільний вектор за рівномірним розподілом. Тут x_* – поточні найкращі глобальні координати (розв'язок), які отримуються в результаті порівнянь всіх розв'язків серед n кажанів.

Оскільки результатом виразу $\lambda_i f_i$ є приріст швидкості, ми можемо використовувати або f_i, або λ_i, щоб відрегулювати зміну швидкості при фіксуванні іншого параметра λ_i або f_i відповідно, залежно від типу функції. У нашій реалізації ми будемо використовувати $f_{min} = 0$ і $f_{max} = 100$. Спочатку кожен кажан ініціалізується довільною частотою, за допомогою рівномірного розподілу з $[f_{min}, f_{max}]$.

Для локальної частини пошуку, коли один з варіантів рішення є одним із кращих рішень, нове рішення для кожного кажана генерується локально за допомогою довільного переміщення (random walk)

$$x_{new} = x_{old} + \epsilon A^t, \quad (1.17)$$

де $\epsilon \in [-1,1]$ – довільне число, а $A^t = \langle A_i^t \rangle$ – середня гучність усіх кажанів в час t.

Оновлення швидкостей і позицій кажанів має деяку схожість з процедурою в стандартній версії методу рою часток, оскільки f_i по суті контролює темп і діапазон руху часток рою. До певної міри, ВА може розглядатися як

збалансоване поєднання стандартного МРЧ та інтенсивного локального пошуку, керованого гучністю та частотою пульсу.

1.2.2. Гучність і випромінення

Гучність A_i та величина r_i імпульсу повинні оновлюватись на кожній ітерації. Так як гучність зазвичай зменшується після знаходження кажаном здобичі, а швидкість імпульсу збільшується, гучність для зручності можна вибрати як будь-яке значення. Наприклад, ми можемо використовувати $A_0 = 100$ і $A_{min} = 1$. Для простоти, ми також можемо використовувати $A_0 = 1$ і $A_{min} = 0$. Припущення, що $A_{min} = 0$ означає, що кажан щойно знайшов здобич і тимчасово припинив генерувати будь-який звук. Тоді подане вище можна описати як:

$$A_i^{t+1} = \alpha A_i^t, r_i^{t+1} = r_i^0[1 - e^{-\gamma t}], \quad (1.18)$$

де α і γ – константи. Для будь-яких $0 < \alpha < 1$ і $\gamma > 0$ ми маємо

$$A_i^t \to 0, r_i^t \to r_i^0, \text{при } t \to \infty. \quad (1.19)$$

Для спрощеного випадку ми можемо використовувати $\alpha = \gamma$, і в нашому випадку ми використовуємо $\alpha = \gamma = 0{,}9$. Вибір параметрів завжди вимагає проведення експериментів. Спочатку кожен кажан повинен мати різні значення гучності і частоти випромінення, що досягається шляхом рандомізації. Початкова гучність A_i^0, як правило, знаходиться в діапазоні $[1{,}2]$, тоді як початкова частота випромінення r_i^0 може мати значення близьке до нуля або будь-якого значення $r_i^0 \in [0{,}1]$, якщо використовується рівняння (1.18). Гучність та частота випромінення будуть оновлюватися лише за умови покращення нових розв’язків, що означає, що кажани рухаються до оптимального рішення.

1.2.3. Модифікований метод кажанів

Метод глобальної оптимізації має відповідати, як мінімум, двом вимогам: він повинен бути спроможним відвідувати найбільш віддалені частини простору пошуку і, з іншого боку, ретельно обстежувати найбільш перспективні області[22]. Пункт «Згенерувати нове рішення за допомогою довільного перельоту» (***Generate a new solution by flying randomly)*** псевдокоду стандартного алгоритму не відповідає обом вимогам.

Задовільнити ці вимоги можна скориставшись так званим Levy Flight. Levy Flight є переміщенням, в якому довжина кроку є випадковою величиною, що підпорядковується сталому розподілу ймовірностей. У просторі розмірністю більше одиниці крок переміщення здійснюється у випадкових ізотропних напрямках. Як відомо, довжина кроку залежить від параметру λ. Чим менше значення λ, тим більший крок.

Щоб задовільнити обидві вимоги, скористуємося Levy Flight з керованим кроком. Кажани відсортовані в порядку зменшення значення їх функції пристосованості, тобто «гірші» кажани мають більший порядковий номер. Щоб задовільнити першу вимогу, будемо вважати, що «гірші» кажани переміщуються з більшим кроком Levy Flight (меншим значенням λ). «Кращі» ж кажани мають більш детально досліджувати окіл свого місцезнаходження (друга вимога) з метою покращення значень попередніх ітерацій. Це означає, що розмах їх переміщення має бути меншим і, відповідно, значення λ – більшим. Цього можна досягти якщо λ належатиме до інтервалу $[\lambda_{min};\ \lambda_{max}]$, а залежність значення λ від рангу кажана обчислювати як

$$\lambda = \lambda_{max} - \frac{i(\lambda_{max} - \lambda_{min})}{m-1}, \qquad (1.20)$$

де m – розмір колонії, i – ранг i-го кажана.

Друга модифікація стосується способу виконання локального пошуку (20) шляхом його гібридизації, яка полягає у врахуванні найкращого на даний момент розв'язку, знайденого усіма кажанами. Нехай bat_b – кажан, розв'язок якого є найкращим серед усіх. Будемо вважати, що інші кажани тим «гірші», чим далі вони розташовані від bat_b і, отже, буде логічним збільшити крок (тобто зменшити λ), з яким вони виконують Levy Fight для поширення зони пошуку. Тоді будемо обчислювати λ для i-го кажана за формулою

$$\lambda = Te^{\frac{-d_{i,bat\_t}}{D}Y}$$

де d_{i,bat_t} – відстань від -го кажана до найкращого bat_b, D – максимальна відстань в межах задачі, T і Y – параметри алгоритму. Такий спосіб організації Levy Fight забезпечує більш екстенсивні переміщення кажанів у просторі пошуку.

1.3. Метод зозулі

Метод зозулі являє собою оптимізований алгоритм, розроблений англ. Xin-She Yang та англ. Suash Deb у 2009 році[23]. Натхненням для його створення послужив гніздовий паразитизм деяких видів зозуль, що підкладають свої яйця до гнізд інших птахів (інших видів птахів). Деякі з власників гнізд можуть вступити у прямий конфлікт із зозулями, що вдираються до них. Наприклад, якщо власник гнізда виявить, що яйця не його, то він або викине ці чужі яйця або просто покине гніздо і збудує нове десь в іншому місці. Деякі види зозуль еволюціонувала таким чином, що самки дуже часто спеціалізуються на імітуванні кольорів і структури яєць обраних видів птахів-господарів. У алгоритмі зозулі такий спосіб поведінки був ідеалізований і таким чином може бути пристосованим для розв'язування різноманітних задач оптимізації. Можливо, він зможе перевершити інші метаевристичні алгоритми у прикладних

програмах. Іншою сферою застосування, здавалось би зовсім не пов'язаною з алгоритмом, є алгоритм хешування (розміщення з використанням функції розстановки), що був розроблений Расмусом Пагом та Флемінгом Фреш Родлером у 2001 році.

Метод зозулі базується на природному паразитизмі деяких видів зозулі шляхом закладання яєць у гнізда птахів-господарів. Ці зозулі-паразити жіночого роду можуть імітувати кольори та візерунок яєць птахів-господарів. Для простоти, передбачається, що в гнізді існує лише одне яйце. Наявне яйце в гнізді являє собою початкове рішення. Яйце, закладене зозулею, являє собою нове рішення, породжене алгоритмом. Алгоритм працює на основі трьох припущень:

1. Зозуля вибирає гніздо, щоб закласти яйце випадковим чином, і закладає лише одне яйце за раз;
2. Найкращі гнізда з якісними яйцями (рішеннями) переносяться на наступні покоління;
3. Загальна кількість наявних гнізд для господарських птахів є фіксованою, і птах-господар може виявити яйце зозулі з ймовірністю $P_\alpha \subset [0; 1]$. У цьому випадку птах-господар або знищує яйце, або повністю знищує гніздо, і збирає нове гніздо в іншому місці.

Третє припущення можна наблизити до частини P_α усіх n гнізд, які замінюються новими гніздами, що мають новий випадковий розв'язок. Коли створюється нове рішення $X^{(t+1)}$ для, скажімо, зозулі i, Levy Flight виконується як $X_i^{(t+1)} = X_i^{(t)} + \alpha \oplus Levy(\lambda)$, де $\alpha > 0$ – розмір кроку, який повинен бути пов'язаний із масштабами задачі. У більшості випадків використовується $\alpha = 1$. Дане рівняння є стохастичним рівнянням довільного переміщення. Це довільне переміщення є ланцюгом Маркова, коли наступне розташування залежить лише від поточного розташування та ймовірності переходу. $\oplus$ означає попарний

добуток елементів. Levy Flight по суті забезпечує довільне переміщення, тоді як довільна довжина кроку складається з розподілу $Levy{\sim}u = t^{-\lambda}, (1 < \lambda \leq 3)$, що має нескінченну дисперсію з нескінченним середнім значенням. Тут кроки по суті утворюють процес довільного переміщення з розподілом довжини кроку з хвостом, що повільно зменшується. Алгоритм також може бути розширений на більш складні випадки, коли кожне гніздо містить кілька яєць (набір рішень). Алгоритм може бути підсумований відповідно до наступного псевдокоду:

BEGIN

Цільова функція $f(X), X = (x_1, x_2, \dots, x_D)$

Згенерувати початкову популяцію з n векторів розв'язків $x_i, i = 1,2, \dots, n$

WHILE ($t <$ ⟨максимальна кількість ітерацій⟩) і (умову завершення не досягнуто)

Створити новий вектор розв'язків X_{new} за допомогою Levy Flight і обчислити значення його фітнес функції F_{new}

Випадково вибрати вектор X_j з поточної популяції, і порівняти значення функцій $f(X_j)$ і $f(X_{new})$

Якщо $f(X_{new}) < f(X_j)$, замінити X_j на X_{new}

Покинути частину P_α найгірших гнізд і згенерувати нові гнізда

Зберегти якісні розв'язки та знайти поточний найкращий вектор

END WHILE

Фінальна обробка та результати

END

1.3.1. Модифікований метод зозулі

Стандартний метод зозулі базується на трьох ідеалізованих правилах:

- Зозуля вибирає собі гніздо, в яке кладе яйце, випадковим чином.
- Кращі гнізда з високою якістю яйця (розв'язку) переходять до наступних поколінь.
- Загальна кількість доступних гнізд птахів-господарів фіксована й птах-господар може виявити яйце зозулі з ймовірністю $P_\alpha \subset [0; 1]$. В цьому

випадку птах-господар або знищує яйце, або повністю руйнує гніздо і будує нове гніздо деінде.

Зозуля i, поточним розв'язком якої є $X_i^{(t)}$ генерує новий $X_i^{(t+1)}$ розв'язок за допомогою Levy Flight

$$X_i^{(t+1)} = X_i^{(t)} + \alpha_t \oplus Levy(\lambda), \quad (21)$$

де $\oplus$ – сума по кожному виміру, $\alpha_t > 0$ – множник, значення якого в стандартному алгоритмі зозулі дорівнює 1. Levy Flight імітує стохастичні переміщення зозулі, розмір кроку яких береться з розподілу Леві

$$Levy \sim u = t^{-\lambda}, (1 < \lambda \leq 3),$$

який має нескінчену дисперсію і нескінчене середнє значення.

Тут застосовано ідею Levy Flight з керованим кроком. Тобто будемо виходити з того, що зозулі, які мають більше значення функції пристосованості, роблять кроки меншого розміру (з більшим λ), ніж зозулі з меншим значенням функції пристосованості (з меншим λ). Таким чином, кращим зозулям надається шанс більш ретельно обстежити окіл їх поточного розв'язку, а гіршим – покинути малоперспективну область. Для цього відсортуємо множину зозуль за неспаданням значень їх функції пристосованості й будемо визначати параметр λ в залежності від позиції зозулі в списку. Будемо вважати, що $\lambda \in [\lambda_{min}, \lambda_{max}]$, а його значення для зозулі i обчислюється як

$$\alpha_t = \alpha_0 \delta^t, \quad (1.22)$$

де $\alpha_0 = \alpha_{max}$, t – номер ітерації, $0 < \delta < 1$ розраховується за формулою

$$\delta = \sqrt[k]{\frac{\alpha_{min}}{\alpha_{max}}}.$$

Тоді псевдокод модифікованого алгоритму матиме вигляд

Цільова функція $f(x)$, $x = (x_1, \dots, x_d)$
Згенерувати початкову популяцію з n птахів-господарів
WHILE (t < ⟨кількість ітерацій⟩) **OR** (критерій завершення)
Обчислити λ згідно (2)
Обчислити α_t згідно (3)
Обрати зозулю, виконати переміщення Леві згідно (1) та обчислити значення її фітнес-функції F_i
Вибрати довільне гніздо j з n
IF ($F_i > F_j$)
замінити j новим розв'язком
Покинути частину p_α найгірших гнізд та замінити їх новими
Обчислити ранг розв'язків та вибрати найкращий
END WHILE

1.4. Метод світлячків

Метод світлячків – це метаевристичний метод, розроблений доктором Синь-Ши Яном[24]. Цей метод базується на природній поведінці світлячків, яка базується на явищі біолюмінесценції. Світлячки в природі здатні виробляти світло завдяки спеціальним фотогенним органам, розташованим дуже близько до поверхні тіла за оболонкою напівпрозорої кутикули. Це світло допомагає їм спілкуватися одне з одним і приваблювати інших світлячків.

Процес біолюмінесценції відповідає за випромінення світла світлячками. Є багато теорій, що стосуються причини та важливості миготливого світла в життєвому циклі світлячка, але більшість з них пояснюють це фазою спарювання. Основне завдання миготливого світла полягає в тому, щоб привабити партнера. Шаблони цих ритмічних спалахів унікальні на основі швидкості миготіння та часу, протягом якого спостерігаються спалахи. Цей шаблон приваблює як особин чоловічого роду, так і жіночого одне до одного. Згідно зі зворотним квадратичним законом, інтенсивність світла I зменшується при зростанні відстані r. Повітря також поглинає світло, і воно стає слабшим із збільшенням відстані. Поєднання цих двох чинників зменшує видимість світлячків на обмежену відстань, яка, як правило, становить кількасот метрів уночі, проте є достатньою для того, щоб світлячки могли спілкуватися один з одним.

Метод світлячків використовує дану біологічну інформацію, сформовану в три ідеалізовані правила:

1. Усі світлячки є безстатевими. Тобто один світлячок взаємодіє з іншими незалежно від статі.
2. Привабливість світлячка пропорційна його яскравості. Таким чином, у випадку двох світлячків, менш яскравий рухається в напрямку більш

яскравого. Оскільки привабливість пропорційна яскравості, вона зменшується зі збільшенням відстані між ними.

3. Значення функції, що оптимізується обернено пропорційне яскравості світлячка.

Для задачі максимізації вважається, що яскравість світлячків має бути пропорційною фітнес функції. Інші форми яскравості можна визначити так само як функцію фітнесу на зразок генетичних алгоритмів.

Інтенсивність світла та привабливість

Алгоритм світлячків ґрунтується на двох важливих речах: варіації інтенсивності світла та формулюванні поняття привабливості. Для простоти передбачається, що привабливість світлячка визначається її яскравістю, яка пов'язана з цільовою функцією.

У конкретному розташуванні x, яскравість I світлячка може бути обрана як $I(x)$ чи f(x) для задачі максимізації.

Привабливість β є відносною, що означає, що вона має оцінюватися іншими світляками, тому вона буде відрізнятися від відстані r_{ij} між світлячком i та світлячком j. Як вже було зазначено, інтенсивність світла зменшується з відстанню від джерела, а світло також поглинається повітрям, тому привабливість повинна дозволяти змінюватися з різним ступенем поглинання. Отже, в найпростішому вигляді інтенсивність світла $I(r)$ змінюється в залежності від зворотного закону квадратів:

$$I(r) = \frac{I_S}{r^2}, \tag{1.23}$$

де I_S – яскравість світла джерела випромінювання.

Формулу (23) з урахуванням коефіцієнта поглинання можна подати у Гаусовій формі

$$I(r) = I_0 e^{-\gamma r}, \tag{1.24}$$

де I_0 – яскравість світла джерела випромінювання, γ – фіксований коефіцієнта поглинання світла (даний параметр можна подавати як залежний від відстані). Щоб уникнути сингулярності при $r = 0$ у виразі I_S/r^2, сукупний ефект як зворотного квадратного закону, так і поглинання можна апроксимувати як наступну Гаусову форму

$$I(r) = I_0 e^{-\gamma r^2}. \tag{1.25}$$

Як згадано раніше, привабливість світлячків β залежно від відстані між ними пропорційна яскравості світла I і описується аналогічною, до (1.25) формулою

$$\beta(r) = \beta_0 e^{-\gamma r^2}, \tag{1.26}$$

де β_0 – яскравість світла джерела випромінювання, γ – фіксований коефіцієнта поглинання світла.

Оскільки часто буває швидше обчислити $1/(1 + r^2)$, ніж експоненціальну функцію, приведену вище, то (1.26) можна апроксимувати як:

$$\beta(r) = \frac{\beta_0}{(1 + \gamma r^2)}. \tag{1.27}$$

Обидва рівняння (1.26) і (1.27) визначають характерну відстань $1/\gamma$, на якій привабливість істотно змінюється від β_0 до ${\beta_0}^{-1}$ для (5) або $\beta_0/2$ для (1.27). У реалізації для реального часу, $\beta(r)$ – це функція привабливості, яка може бути будь-якою функцією, що монотонно зменшується, наприклад:

$$\beta(r) = \beta_0 e^{-\gamma r^m} \ (m \geq 1). \tag{1.28}$$

При фіксації, характеристична довжина набуває вигляду:

$$\Gamma = \gamma^{-\frac{1}{m}} \to 1, m \to \infty. \tag{1.29}$$

І навпаки, параметр γ може бути використаний як типове початкове значення для певної довжини шкали Г задачі оптимізації. Наприклад:

$$\gamma = \frac{1}{\Gamma^m}. \tag{1.30}$$

Відстань $r_{i,j}$ між світлячками i та j обчислюється за формулою евклідової відстані

$$r_{i,j} = \|x_i - x_j\| = \sqrt{\sum_{k=1}^{d} \left(x_{i,k} - x_{j,k}\right)^2}, \tag{1.31}$$

де d – вимірність простору.

Для двовимірного простору формула (10) набуває вигляду

$$r_{i,j} = \sqrt{\left(x_i - x_j\right)^2 - \left(y_i - y_j\right)^2}, \tag{1.32}$$

З використанням формул (24) і (26) отримуємо формулу переміщення світлячка i до світлячка j

$$x_{i,k}^{t+1} = x_{i,k}^{t} + \beta_0 e^{-\gamma r_{i,j}^2}\left(x_{j,k}^{t} - x_{i,k}^{t}\right) + \alpha €_i, \tag{1.33}$$

де k – номер координати, $x_{i,k}^{t}$ – значення координати до переміщення, α_t – довільний коефіцієнт, $€_i$ – i-й елемент вектора довільних чисел.

Другий компонент використовується для привабливості, а третій – для рандомізації, причому α є параметром рандомізації, а $€_i$ – вектор випадкових чисел, який отримується з Гаусового або рівномірного розподілу. Наприклад, найпростіший форма – коли $€_i$ можна замінити на $(rand - 0{,}5)$, де $rand$ – генератор випадкових чисел, розподілених у рівномірному діапазоні [0, 1]. Найважливішим параметром у алгоритмі світлячків є γ, він відіграє дуже важливу роль у визначенні швидкості збіжності та поведінки алгоритму FA. Теоретично $\gamma \in [0, \infty)$, але на практиці $\gamma = O(1)$ визначається

характеристичною довжиною Г оптимізованої системи. Так що для майже всіх програм він змінюється від 0,1 до 10.

Якщо $\beta_0 = 0$, це стає простим випадковим переміщенням. З іншого боку, якщо $\gamma = 0$, алгоритм зводиться до варіанту МРЧ. Крім того, рандомізація $€_i$ може бути легко поширена на інші розподіли, такі як польоти Леві.

Оскільки α_t істотно контролює випадковість (або, в якійсь мірі, різноманітність рішень), ми можемо налаштувати цей параметр під час ітерацій так, щоб він міг змінюватися залежно від лічильника ітерацій t. Таким чином, α_t можна представити як

$$\alpha_t = \alpha_0 \delta^t, 0 < \delta < 1,$$

де α_0 – початковий коефіцієнт масштабування, який є випадковим, і є по суті фактором сповільнення. Для більшості програм ми можемо використовувати δ від 0,95 до 0,97.

Що стосується початкового α_0, то моделювання показують, що FA буде більш ефективним, якщо цей параметр пов'язаний з масштабуванням конструктивних змінних. Нехай L – середня значення досліджуваної функції. Тоді ми можемо встановити $\alpha_0 = 0{,}01L$. Множник 0,01 випливає з того факту, що для випадкових переміщень потрібно декілька кроків для досягнення мети, одночасно врівноважуючи локальний пошук, не перестрибуючи більше ніж на кілька кроків.

Параметр β керує привабливістю, а параметричні дослідження показують, що $\beta_0 = 1$ може використовуватися для більшості програм. Однак, γ має також залежати від масштабування L. Загалом, ми можемо встановити $\gamma = 1/\sqrt{L}$. Якщо варіанти масштабування не є значними, то ми можемо встановити $\gamma = O(1)$.

Алгоритм світлячків можна подати у вигляді наступного псевдокоду:

Цільова функція $f(x)$, $x = (x_1, x_2, \dots, x_d)^T$
Виконати ініціалізацію популяції світлячків x_i, $i = 1,2, \dots, n$
Задати коефіцієнт поглинання світла γ
WHILE ($t <$ ⟨максимальна кількість ітерацій⟩)
    FOR i=1: n (всі світлячки)
    FOR j=1: i
        Інтенсивність світла I_i в x_i визначається згідно $f(x_i)$
        IF ($I_i > I_j$)
            Перемістити світлячка i до світлячка j по всім координатам
        ELSE
            Перемістити світлячка i довільно
        END IF
        Обчислити нове значення цільової функції світлячка i
    END FOR
    END FOR
    Обчислити ранг світлячків згідно інтенсивності світла та визначити найкращого світлячка
END WHILE

Налаштування евристичних параметрів

Окрім параметрів α та γ та впливу випадкових чисел, описаних вище, конвергенція алгоритму пропорційна кількості світлячків. Тобто більшій кількості світлячків, для збіжності потрібно більше ітерацій. Найкраща швидкість конвергенції, тобто найкращий компроміс між розвідкою та експлуатацією, сильно залежить від багатьох речей, таких як кількість локальних оптимумів, оптимізована функція, відстань від глобального рішення, позиція глобального рішення в пошуковій області, розмір домену тощо. Майже неможливо знайти набір унікальних параметрів для алгоритму, який буде працювати у всіх випадках, але, на основі вищезгаданих міркувань, наведено наступну емпіричну процедуру, яка працює на практиці.

Збіжність

Алгоритм запускається кілька разів, доки він не збігатиметься з заданим набором конвергентних параметрів. Якщо в більшості випадків отримуються різні результати, то коефіцієнт конвергенції занадто високий; алгоритм передчасно збігається до не оптимальних точок. Якщо той самий результат отримується знову і знову, і за час великої кількості ітерацій кращих точок не зустрічається, швидкість конвергенції занадто низька і світлячки не фокусують пошук на перспективних рішеннях, то треба вибрати більш швидкий конвергентний набір параметрів і меншу кількість світлячків. Якщо отримані послідовні результати та коефіцієнт конвергенції є гарними і встановлено один і той же параметр, то для вирішення подібних проблем слід використати розмір домену.

Оптимізаційні експерименти

Алгоритм світлячків був використаний для оптимізації чотирьох контрольних функцій. В результаті було визначено три набори параметрів: ($\alpha = 0{,}25, \gamma = 0{,}1$), ($\alpha = 0{,}5, \gamma = 1$) і ($\alpha = 0{,}1, \gamma = 0{,}01$).

1.4.1. Модифікований метод світлячків

В якості $€_i$ в формулі (1.33) найчастіше застосовують вектор, згенерований за допомогою так званого Levy Flight (Додаток А), який визначає стохастичні переміщення за законом стабільного розподілу ймовірностей. Тоді переміщення світлячка визначається формулою

$$x_{i,k}^{t+1} = x_{i,k}^{t} + \beta_0 e^{-\gamma r_{i,j}^2}\left(x_{j,k}^{t} - x_{i,k}^{t}\right) + \alpha_t \oplus Levy(\lambda), \qquad (1.34)$$

де $\oplus$ – покомпоентне множення.

У стандартному алгоритмі світлячків, довжина кроку переміщення α_t світлячка є фіксованою величиною. Причому всі світлячки переміщуються з

фіксованою довжиною кроку на всіх ітераціях. Завдяки цьому алгоритм може втратити пошукові можливості й потрапити в один з локальних екстремумів.

Як відомо, розмах переміщення за допомогою Levy Flight залежить від значення λ – чим менше λ, тим більше розмах.

Відсортуємо світлячків в порядку спадання значення їх фітнес-функції. Для підвищення пошукових можливостей алгоритму, припустимо, що гірші світлячки мають виконувати Levy Flight з більшим розмахом (з меншим значенням λ), що забезпечить пошук у віддалених точках простору пошуку. З іншого боку, кращі світлячки повинні виконувати більш ретельний пошук у перспективних точках, знайдених на попередніх ітераціях. Тобто їх розмах Levy Flight має бути малим, або значення λ має бути більшим. Цього можна досягнути, якщо λ належить інтервалу $[\lambda_{min}, \lambda_{max}]$ і залежність λ від рангу світлячка обчислюється як

$$\lambda = \lambda_{max} - i\frac{\lambda_{max} - \lambda_{min}}{m - 1}, \tag{1.35}$$

де i – розмір колонії, i – ранг i-го світлячка.

Крім того, коли в процесі роботи алгоритму знайдено перспективні точки, бажаним є більш ретельне дослідження їх околиць. Досягти цього можна за рахунок зменшення значення α. Як було встановлено, найкращі результати можна отримати якщо $\alpha \in [\alpha_{min}, \alpha_{max}]$, де $\alpha_{min} = 0{,}0001$ і $\alpha_{max} = 0{,}01$. Тобто α повинно пробігати значення від α_{max} до α_{min} по мірі виконання алгоритму. Цього можна досягти, якщо

$$\alpha = \alpha_0 \delta^{iter}, \tag{1.36}$$

де $iter$ – номер ітерації, а δ обчислюється за формулою

$$\delta = \sqrt[k]{\frac{\alpha_{min}}{\alpha_{max}}},$$

де $k = \max \_iter$.

Таким чином псевдокод алгоритму має вигляд:

Цільова функція $f(x)$, $x = (x_1, x_2, \dots, x_d)^T$
Виконати ініціалізацію популяції світлячків x_i, $i = 1,2, \dots, n$
Задати коефіцієнт поглинання світла γ
WHILE ($t <$ ⟨максимальна кількість ітерацій⟩)
FOR i=1: n (всі світлячки)
FOR j=1: i
Інтенсивність світла I_i в x_i визначається згідно $f(x_i)$
IF ($I_i > I_j$)
Обчислити λ згідно (14)
Перемістити світлячка i до світлячка j по всім координатам згідно (13)
ELSE
Перемістити світлячка i довільно
END IF
Обчислити нове значення цільової функції світлячка i
END FOR
END FOR
Обчислити ранг світлячків згідно інтенсивності світла та визначити найкращого світлячка
END WHILE

1.5. Метод мурах

Особливо успішним є метаевристичний алгоритм, натхненний поведінкою мурах[26]. Починаючи з системи мурах, було розроблено та застосовано ряд алгоритмічних підходів, що базуються на ідеях, які мають значний успіх у

вирішенні різноманітних проблем комбінаторної оптимізації як для академічних, так і реальних застосувань. Мурашиний алгоритм оптимізації (МАО) – метаевристична структура, яка охоплює алгоритмічний підхід, згаданий вище. Метаевристика МАО була запропонована як загальна основа для існуючих програм та алгоритмічних варіантів мурашиних алгоритмів.

Метаевристика – це набір алгоритмічних понять, які можуть бути використані для визначення евристичних методів, що застосовуються для широкого спектра задач. Інакше кажучи, метаевристичний метод може розглядатися як загальноприйнятий евристичний метод, призначений для вирішення конкретної евристичної задачі (наприклад, алгоритму локального пошуку) і може бути використаний для інших умов (наприклад, дослідження перспективних областей пошукового простору, що містять якісні розв’язки). Отже, метаевристика є загальною алгоритмічною обгорткою, яка може бути застосована до різних задач оптимізації з відносно невеликою кількістю модифікацій, щоб зробити її адаптованою до конкретної проблеми.

1.5.1. Постановка задачі

Мураха в МАО – це стохастична конструктивна процедура, яка поступово знаходить розв’язок шляхом додавання короткочасно визначених компонентів розв’язку до часткового розв’язку, що обчислюється. Тому метаевристика МАО може застосовуватися до будь-якої комбінаторної задачі оптимізації, для якої може бути визначена конструктивна евристика. Хоча це означає, що метаевристика МАО може бути застосована до будь-яких комбінаторних задач оптимізації, проблема полягає в тому, як використати розглянуту проблему до форми, яка може використовуватися мурахами для побудови рішень.

Проблема комбінаторної оптимізації $(S; f; \Omega)$ відображається на задачі, яка може характеризуватися наступним списком елементів:

- Скінченна множина $C = \{c_1, c_2, \dots, c_{NC}\}$, де NC – її потужність;
- Стани задач визначаються в термінах послідовностей $x = \langle c_i, c_j, \dots, c_h, \dots \rangle$ скінченних розмірностей для елементів C. Набір всіх можливих станів позначається як x. Кількість елементів послідовності x позначається як $|x|$. Максимальний розмір послідовності обмежений додатною константою $n < +\infty$;
- Набір потенційних розв'язків S є підмножиною χ (тобто $S \subseteq \chi$);
- Набір допустимих станів $\tilde{\chi}$, $\tilde{\chi} \subseteq \chi$, визначається за допомогою тесту для конкретної функції, що перевіряє неможливість перетворення послідовності $x \in \tilde{\chi}$ в розв'язок, що задовольняє умовам (обмеженням) Ω. Варто відмітити, що за цим визначенням допустимість значень $x \in \tilde{\chi}$ слід сприймати в слабкому сенсі. Фактично, це не гарантує, що послідовність s перетворюється в x таким чином, що $s \in \tilde{\chi}$;
- Непорожньої множини s^* оптимальних розв'язків таких, що $s^* \subseteq \tilde{\chi}$ і $s^* \subseteq s$.
- Значення якості (значення цільової функції) $g(s,t)$ відповідає кожному потенційному розв'язку $s \in A$. У більшості випадків $g(s,t) \equiv f(s,t)$, $\forall s \in \tilde{S}$, де $S \subseteq \tilde{S}$ – це набір можливих потенційних розв'язків, отриманих з S з обмеженнями $\Omega(t)$.
- У деяких випадках якість (або оцінка якості $J(x,t)$) можуть бути пов'язані з іншими станами, ніж потенційні рішення. Якщо x_j можна, отримана шляхом додавання компонентів рішення до стану x_i, то $J(x_i, t) < J(x_j, t)$. Варто зазначити, що $J(x,t) \equiv g(s,t)$.

З огляду на це формулювання, мурашки створюють рішення, виконуючи довільне переміщення по повністю зв'язаному графу $G_C = (C, L)$, вузли якого є елементами C, а множина L повністю з'єднує елементи C. Граф G_C називається графом побудови, елементи L називаються ребрами (дугами). Обмеження $\Omega(t)$ реалізуються в правилах, яких дотримуються мурашки, як це викладено в наступній частині. Вибір реалізації обмежень при заданні мурах надає певну гнучкість. В залежності від комбінаторної задачі, що вирішується, може бути більш доцільним описати жорсткі обмеження, що дозволить мурахам отримувати лише прийнятні розв'язки або м'які обмеження, і тоді мурашки зможуть будувати недопустимі розв'язки (тобто потенційні розв'язки у $S \backslash \tilde{S}$), але будуть покарані в залежності від ступеня їх недопустимості.

1.5.2. Поведінка мурах

Як ми вже говорили, в алгоритмах МАО мурашки є стохастичними конструктивними процедурами, які будують розв'язок, переміщаючись по графу $G_C = (C, L)$, де множина L повністю з'єднує елементи C. Обмеження задачі $\Omega(t)$ вбудовані в евристики мурах. У більшості випадків мурашки отримують допустимі розв'язки. Проте іноді може бути корисно також дозволити їм будувати недопустимі рішення. Вершини $c_i \in C$ і ребра $l_{ij} \in L$ мають відповідні *сліди феромонів (pheromone trail)* τ (сліди τ_i пов'язані з вершинами, слід τ_{ij} пов'язані з ребрами), а також *евристичне значення* η (η_i і η_{ij} відповідно). Слід феромонів заносить в довготривалу пам'ять інформацію про весь процес пошуку мурашки та оновлюється самими мурахами. Інакше кажучи, евристичне значення, яке часто називають евристичною інформацією, являє собою апріорну інформацію про поточну задачу або інформацію, що отримується в момент виконання, що надається джерелом, відмінним від мурашок. У багатьох

випадках η – це вартість або оцінка вартості додавання елементів до розробленого рішення. Ці значення використовуються евристичними правилами мурах, щоб зробити ймовірнісні висновки про те, як переміщатись по графу.

Точніше, кожна мурашка колонії має наступні властивості:

- Використовує граф $G_C = (C, L)$ для пошуку оптимальних розв'язків $s^* \in S^*$;
- Має пам'ять M^k, яку може використовувати для зберігання інформації про шлях, яким вона пересувалася раніше. Пам'ять може бути використана для (1) побудови допустимих розв'язків (тобто реалізації обмежень Ω); (2) обчислень евристичних значень; (3) оцінки знайдених розв'язків; (4) відслідковування зворотного шляху;
- Має початковий стан x_k^s і одну або більше *умов зупинки* e^k. Зазвичай початковий стан виражається або як порожня послідовність, або як послідовність одиничної довжини, тобто послідовність з одним елементом;
- Коли мураха знаходиться в стані $x_r = \langle x_{r-1}, i \rangle$, якщо умова зупинки не виконується, вона рухається до сусідньої вершини j, тобто до стану $\langle x_r, j \rangle \in \chi$. Якщо принаймні одна із умов зупинки e^k виконується, то мурашка зупиняється. Коли мураха обчислює потенційний розв'язок, переміщення в недопустимий стан забороняється в більшості програм, як за допомогою пам'яті мурашки, так і за допомогою відповідних евристичних значень η;
- Виконує переміщення застосовуючи правило ймовірнісного вибору. Правило ймовірнісного вибору є: (1) функцією локально доступних феромонів і евристичних значень (тобто феромонів і евристичних значень, пов'язаних з вершинами та ребрами в околиці поточного розташування

мурашки на графі G_C); (2) приватною пам'яттю мурашки, що зберігає її поточний стан; (3) обмеженнями задачі;

- При додаванні вершин c_j до поточного стану, вона може оновити слід феромонів, пов'язаний з нею або з відповідним ребрами;
- Після того, як мураха знайшла розв'язки, вона може пройти той самий шлях у зворотному напрямку і оновити сліди феромонів для вершин.

Важливо зазначити, що мурашки діють всі разом і незалежно одна від одної одночасно, і хоча кожна мураха є достатньо складною, щоб знайти (хоч і не дуже якісний) розв'язок поставленої задачі, якісні розв'язки можуть бути отримані тільки в результаті колективної взаємодії між мурахами. Це досягається через непряму комунікацію, опосередковану інформаційними мурахами, які читають чи записують дані до змінних, що зберігають значення сліду феромонів. У певному сенсі це – розподілений процес навчання, в якому окремі агенти, мурашки, не тільки є адаптивними, а, навпаки, адаптують спосіб, в який представлена здача та сприймається іншими мурахами.

Формально алгоритм МАО можна представити як взаємодію трьох процедур: ConstructAntsSolutions, UpdatePheromones та DaemonActions.

ConstructAntsSolutions керує колонією мурах, яка одночасно асинхронно відвідує сусідні стани розглянутої проблеми, переміщаючись через сусідні вершини графа G_C задачі. Вони рухаються, застосовуючи стохастичну політику прийняття локальних рішень, яка використовує слід феромонів та евристичну інформацію. Таким чином, мурашки поступово будують оптимізаційний розв'язок. Після того, як мураха створює розв'язок, або в час його побудови, мураха оцінює (частковий) розв'язок, який буде використовуватися процедурою UpdatePheromones, щоб вирішити, скільки феромонів залишати на шляху.

UpdatePheromones – це процес, за допомогою якого модифікуються сліди феромонів. Значення слідів феромонів може збільшуватися, оскільки феромони розміщуються на вершинах або ребрах, які використовуються мурахами, або зменшуються, завдяки випаровуванню феромонів. З практичної точки зору депонування нового феромона збільшує ймовірність того, що ці вершини/ребра, які використовувались багатьма мурахами або які використовувались щонайменше одною мурашкою і через які проходить якісний розв'язок, знову будуть використані мурахами в майбутньому. Інакше, випаровування феромонів реалізує форму забуття: це дозволяє уникнути надто швидкої конвергенції алгоритму до субоптимальної області, і тому сприяє вивченню нових областей пошукового простору.

Нарешті, процедура DaemonActions використовується для здійснення централізованих дій, які не можуть виконувати окремі мурашки. Приклади дій процедури – це активація локальної процедури оптимізації або збір глобальної інформації, яка може бути використана для визначення того, чи є корисним не розміщувати додаткові феромони для модифікації процесу пошуку з глобального погляду. Наприклад, процедура може спостерігати за шляхами, отриманими кожною з мурах у колонії, і в результаті виділити один або кілька мурашок (наприклад, ті, які побудували найкращі розв'язки на ітерації алгоритму), які в майбутньому дозволяють розміщувати додаткові феромони на вершинах/ребрах, які використовувалися.

```
procedure MAOMetaheuristic
    ScheduleActivities
        ConstructAntsSolutions
        UpdatePheromones
        DaemonActions // не обов'язково
    end-ScheduleActivities
end-procedure
```

Основна процедура метаевристики МАО керує плануванням трьох вищезазначених компонентів алгоритмів МАО за допомогою блоку ScheduleActivities:

1) керування діяльністю мурах,

2) оновлення феромонів,

3) прийняття централізованих рішень.

Конструкція ScheduleActivities не описує планування та синхронізацію цих трьох дій. Іншими словами, вона не визначає, чи повинні вони виконуватися повністю паралельно та незалежно, або чи потрібна певна синхронізація. Дизайнер, таким чином, може вказувати спосіб взаємодії цих трьох процедур з урахуванням характеристик поставленої задачі.

Тепер ми представляємо деталі алгоритму S-MAO (або S-ACO, Ant Colony Optimization) , який адаптує поведінку штучних мурашок до вирішення задач мінімального за витратами шляху на графах. Для кожного ребра (i, j) графа $G = \{N, A\}$ ми маємо змінну τ_{ij}, яка називається слідом феромонів. Сліди феромонів зчитуються та змінюються мурахами. Величина (інтенсивність) сліду феромонів пропорційна корисності, яка розглядається мурахами, для побудови якісних розв’язків.

Поведінка мурах при пошуку шляхів

Кожна мураха стартує з початкової вершини та, застосовуючи покрокове прийняття рішень, отримує розв’язок. На кожній вершині мурашка зчитує локальну інформацію, що зберігається у власне вершині та на суміжних ребрах, і застосовує стохастичний спосіб, щоб визначити, в яку вершину виконати

переміщення на наступній ітерації. На початку процесу пошуку для всіх дуг встановлюється певна кількість феромонів (наприклад, $\tau_{ij} = 1, \forall (i,j) \in A$). При розташуванні у вершині i, мураха k використовує сліди феромонів τ_{ij} для обчислення ймовірності вибору j наступної вершини за формулою:

$$p_{ij}^k = \begin{cases} \frac{\tau_{ij}^\alpha}{\sum_{l \epsilon N_i^k} \tau_{ij}^\alpha}, \text{якщо } j \epsilon N_i^k \\ 0, \text{інакше} \end{cases}, \quad (1.37)$$

де N_i^k – околиця мурашки k, що знаходиться у вершині i. Для S-MAO околиця вершини містить всі вершини, суміжні з вершиною графа $G = \{N, A\}$, за винятком попередника вершини i (тобто останньої вершини, яку відвідала мураха перед переходом до i). Таким чином, мурашки уникають повернень до тих же вершин, які вони відвідали безпосередньо перед поточною вершиною. До N_i^k включається попередник вузла i тільки у тому випадку, коли N_i^k не містить вершин, що відповідає тупиковій частині графа. Проте така політика прийняття рішень може легко призвести до створення циклів на отриманих шляхах. Мураха неодноразово переходить від вершини до вершини, використовуючи цю політику прийняття рішень, допоки вона не досягне кінцевої вершини. Через відмінності між шляхами мурах, час, за який вони досягають цільової вершини (кількість ітерацій), можуть відрізнятись від мурашки до мурашки (мурашки, що подорожують по коротших шляхах, швидше досягнуть цільової вершини).

Відновлення шляху та оновлення феромонів

При досягненні цільової вершини мураха перемикається на зворотний режим, а потім поступово відтворює той самий шлях назад до початкової вершини. Під час його повернення до початкової вершини мураха k розміщує певну кількість феромонів $\Delta\tau^k$ на ребрах, які вона відвідала. Зокрема, якщо мураха k знаходиться у зворотному режимі, і вона проходить через ребро (i,j), вона змінює значення феромонів наступним чином:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau^k. \qquad (1.38)$$

За цим правилом мураха, що використовує ребро, що з'єднує вершини i та j, збільшує ймовірність того, що в майбутньому інші мурашки використають цю саму дугу. Важливим аспектом є вибір значення $\Delta\tau^k$.

Випаровування сліду ферамонів

Випаровування слідів феромонів можна розглядати як механізм дослідження, що дозволяє уникнути швидкого отримання всіма мурахами субоптимальних шляхів. Дійсно, зниження інтенсивності феромонів сприяє вивченню різних шляхів протягом всього процесу пошуку. В реальному світі, у колоніях мурашок феромони також випаровуються, але випаровування не відіграє для них важливу роль у пошуку найкоротшого шляху. Той факт, що випаровування феромонів є важливим для штучних мурахах, пов'язаний з тим, що проблеми оптимізації, що вирішуються штучними мурахами, набагато складніші, за проблеми біологічних мурах. Такий механізм як випаровування, який допомагає забувати помилки поганого вибору в минулому, дозволяє постійно вдосконалювати «вивчену» структуру проблеми. Крім того, штучне випаровування феромонів також відіграє важливу функцію обмеження максимального значення кількості феромонів для одного ребра.

Випаровування зменшує кількість феромонів для слідів з експоненціальною швидкістю. Після того, як кожна мураха k переміщується до наступної вершини відповідно до описаної раніше поведінки, сліди феромонів випаровуються, застосовуючи наступну формулу для всіх ребер:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij}, \forall (i,j) \in A,$$

де параметр $\rho \in [0;1]$. Після того, як випаровування феромонів застосовано до всіх ребер, кількість феромонів $\Delta\tau^k$ додається до ребер. Ітерація

S-MAO є повним циклом, що включає рух мурашок, випаровування феромонів та розміщення феромонів.

Під час переміщення у зворотному напрямку мураха k відкладає на кожному з ребер кількість феромонів, що дорівнює $\Delta\tau^k$. Зокрема, якщо мураха k проходить ребро (i, j), величина феромонів τ_{ij} набуває значення за (38).

Завдяки цьому збільшується ймовірність того, що в майбутньому інші мурашки виберуть ребро (i, j). Величина $\Delta\tau_{ij}^k$ визначається в наступний спосіб:

$$\Delta\tau_{ij}^k = \frac{Q}{L_k}, \qquad (1.39)$$

де Q – деяка константа, L_k – вага шляху, пройденого мурашкою k.

1.5.3. Вдосконалення мурашиного алгоритму

Елітізм системи мурах

Перше вдосконалення базового алгоритму МАО називається елітарною стратегією для системи мурах (ECM, Elitist strategy for Ant System – EAS). Ідея полягає в тому, щоб забезпечити значне додаткове покращення дуг, що належать до найкращого шляху, знайденого за час роботи алгоритму; цей шлях позначається як T^{bs} (best-so-far tour). Варто зазначити, що це додаткове покращення для найкращого шляху (яке можна розглядати як додатковий феромон, що розміщується додатковою мурашкою, яка називається прихованою мурашкою) є ще одним прикладом дії процедури DaemonActions.

Додаткове покращення найкращого шляху T^{bs} досягається шляхом додавання кількості e/C^{bs} до його дуг, де e – параметр, який визначає вагу даного шляху, C^{bs} – його довжина. Тоді формулу оновлення феромонів можна перевизначити як:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^k + e\Delta\tau_{ij}^{bs}, \qquad (1.40)$$

де $\Delta\tau_{ij}^{k}$ і $\Delta\tau_{ij}^{bs}$ визначаються як

$$\Delta\tau_{ij}^{bs} = \begin{cases} 1/C^{bs}, \text{якщо ребро } (i,j) \text{ належить до } T^{bs} \\ 0, \text{інакше} \end{cases},$$

$$\Delta\tau_{ij}^{k} = \begin{cases} 1/C^{k}, \text{якщо ребро } (i,j) \text{ належить до } T^{k} \\ 0, \text{інакше} \end{cases}.$$

В ECM, як і у всіх інших алгоритмах, випаровування феромонів здійснюється так само, як і в СМ(система мурах). Результати обчислень дозволяють припустити, що використання елітарної стратегії з відповідним значенням для параметра e дозволяє СМ знайти кращі шляхи за меншу кількість ітерацій.

Система мурах з рангом

Ще одне покращення МАО – версія з рангами – AS_{rank}. Для AS_{rank} кожна мураха розміщує кількість феромонів, що зменшується зі зменшенням рангу. Окрім того, як і в ECM, найкраща на даний момент мураха розміщує найбільшу кількість феромонів на кожній ітерації.

Перш ніж оновлювати сліди феромонів, мурашки сортуються за збільшенням довжини шляху, а кількість феромонів мурашки визначається відповідно до рангу r мурашки. Дублі можна опрацювати випадковим чином (у нашій реалізації це вирішується шляхом лексикографічного впорядкування за іменем мурашки k). На кожній ітерації тільки $(w-1)$, найкращим за рангом мурахам, і мурахам, що отримали найкращий на даний момент шлях (ця мураха не обов'язково належить до набору $(w-1)$ найкращих мурах поточної ітерації), дозволяється наносити феромони. Найкращий на даний момент шлях має найвищий вплив із вагою w (тобто його внесок $1/C^{bs}$ множиться на w); а r-а найкраща мураха поточної ітерації вносить вклад в оновлення феромона зі

значенням $1/C^{bs}$, помножений на вагу $max\{0; w-r\}$. Таким чином, правило оновлення феромонів AS_{rank} має вигляд

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{r=1}^{w-1}(w-r)\Delta\tau_{ij}^{r} + w\Delta\tau_{ij}^{bs}, \qquad (1.41)$$

де $\Delta\tau_{ij}^{r} = 1/C^{r}$ і $\Delta\tau_{ij}^{bs} = 1/C^{bs}$. Результати експериментальних досліджень Булгеймера (Bullnheimer) вказують, що AS_{rank} має трохи кращі показники за ECM і значно кращі за CM.

Система мурах "max – min"

Одним з покращень алгоритму SA є так звана max-min система мурах (MM CM). Основна ідея MM CM полягає в тому, що більш ретельно досліджується найкращий знайдений шлях. Це може бути або кращий шлях на даній ітерації, або найкращий із усіх знайдених. Очевидно, що такий підхід досить швидко приводить до збіжності до деякого локального екстремуму. Щоб перешкодити цьому вводиться діапазон $[\tau_{min}, \tau_{max}]$, у межах якого може змінюватися рівень феромонів для даного ребра. Цим же цілям служить постійна реініціалізація матриці $\{\tau_{ij}\}$ значно більшими додатними значеннями.

Відновлення рівня феромонів

Після побудови розв'язку виконується відновлення рівня феромонів

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^{best}, \Delta\tau_{ij}^{r} = 1/C^{best}, \qquad (1.42)$$

де C^{best} – вага найкращого знайденого шляху. У загальному випадку в якості C^{best} циклічно вибираються найкращий взагалі і кращий за ітерацію шляхи.

Межі рівня зміни феромонів

Діапазон $[\tau_{min}, \tau_{max}]$, у межах якого може змінюватися рівень феромонів для даного ребра визначається в такий спосіб:

$$\tau_{max} = \frac{1}{(1-\rho)} \frac{1}{C^{best}}, \tag{1.43}$$

$$\tau_{min} = \frac{\tau_{max}}{a},$$

де a – константа, визначена експериментальним шляхом.

Ініціалізація та реініціалізація слідів феромонів

На початку алгоритму початкові сліди феромонів встановлюються на приблизну верхню границю допустимих значень. Цей спосіб ініціалізації феромонів, в поєднанні з низьким значення параметра випаровування феромонів, призводить до повільного збільшення відносної різниці в рівнях феромонів, так що початкова фаза MMAS призводить до дослідження всіх частин графа.

Як ще один засіб для покращення вивчення шляхів, що мають невелику ймовірність бути обраними, сліди феромонів MMAS іноді повторно ініціалізуються. Повторна ініціалізація сліду феромонів як правило спрацьовує, коли алгоритм наближається до поведінки стагнації (вимірюється деякими статистичними даними щодо слідів феромонів), або якщо за задану кількість ітерацій алгоритму не знайдено покращеного шляху.

1.5.4. Мурашиний алгоритм для розв’язання задач безперервної оптимізації

У безперервному МАО сукупність розв’язків зберігається в архіві розв’язків T. Для кожного розв’язку n-вимірної функції МАО зберігає в T

значення n її змінних і значення цільової функції $f(s_i)$. i-а змінна l-го розв'язку позначається s_l^i. Структура архіву розв'язків T представлена на рисунку 2.

Компоненти розв'язків використовуються для динамічної генерації функції щільності ймовірності (PDF). Для цього визначений спосіб генерації PDF на основі набору збережених розв'язків. Гаусове ядро PDF параметризоване трьома векторами: ω, μ^i і σ^i (кожний розмірності k). Розв'язки в архіві використовуються для розрахунків значень цих параметрів і, отже, формування Гаусового ядра PDF, яке використовується для керування мурахами в процесі пошуку розв'язку.

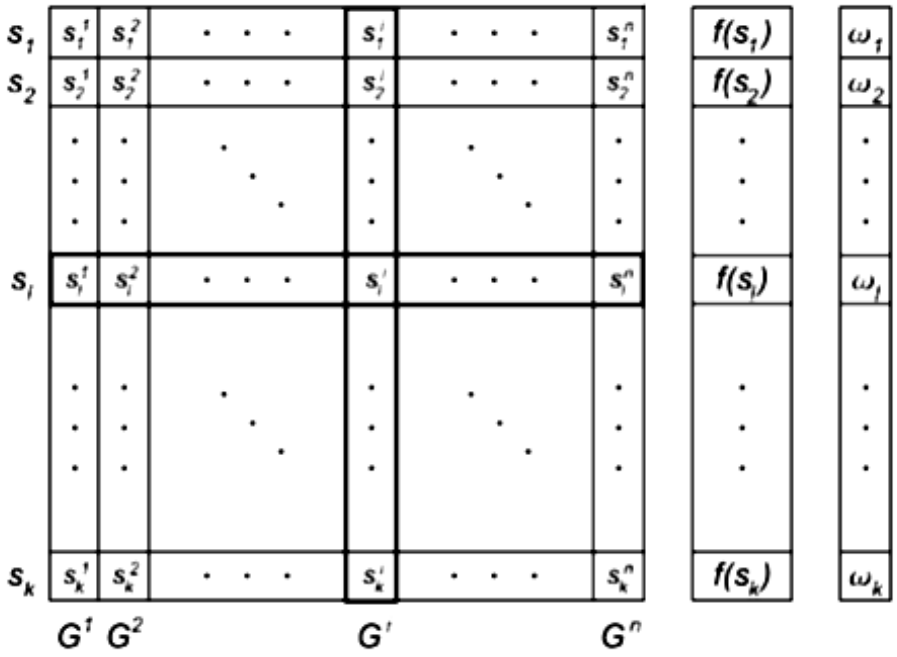


Рисунок 1.2. Структура архіву

Число розв'язків в архіві дорівнює k і цей параметр визначає складність PDF, тобто є k окремих функцій Гауса, що становлять Гаусове ядро (рис. 3)

Розв'язки впорядковані в архіві відповідно до їхньої якості, тобто, для задачі мінімізації: $f(s_1) \le f(s_2) \le f(s_3) \le \cdots \le f(s_k)$. Кожен розв'язок має відповідну вагу ω пропорційну якості розв'язку. PDF G^i побудована з використанням тільки i-ї координати всіх k розв'язків архіву. Для кожної розмірності $i = 1,2,\dots,n$ необхідно визначити власне Гаусове ядро G^i. Для кожного такого G^i значення i-ї змінної всіх розв'язків в архіві є елементами вектора μ^i:

$$\mu^i = \{\mu_1^i, \dots, \mu_k^i\} = \{s_1^i, \dots, s_k^i\}.$$

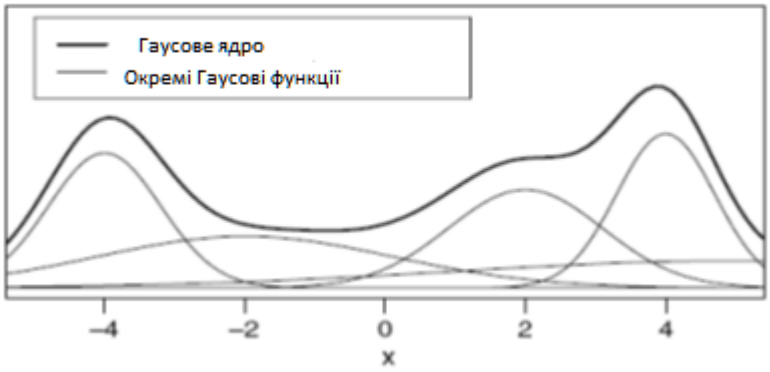


Рисунок 1.3 - Приклад Гаусових функції та їх суперпозицій

Вектор ваг створюється в наступний спосіб. Кожний розв'язок, який додається в архів T, оцінюється (обчислюється функція пристосованості). Розв'язки в архіві сортуються відповідно до їхнього рангу, тобто, розв'язок s_l має ранг l. Вагу розв'язку розраховують за наступною формулою:

$$\omega_l = \frac{1}{qk\sqrt{2\pi}} e^{-\frac{(l-1)^2}{2q^2k^2}}, \qquad (1.44)$$

яка по суті визначає вагу як значення функції Гауса з аргументом l, середнім значенням 1 і стандартним відхиленням qk, де q – параметр алгоритму. Коли значення q невелике, найкращі розв'язки вибираються з найбільшою ймовірністю, інакше ймовірність стає більш однорідною. Вплив цього параметра

на МАО подібний регулюванню балансу між iteration-best і best-so-far схем відновлення феромонів в МАО.

Для того щоб знайти остаточну форму кожного Гаусового ядра G^i, повинен бути визначений вектор стандартних відхилень σ^i. Розташовуючи набір змінних X_i, $i = 1,2,\dots,n$, мураха будує розв'язок за n ітерацій. На кожній ітерації i мураха вибирає значення для змінної X_i. Як згадувалося раніше, Гаусове ядро у форматі PDF складається з ряду звичайних Гаусових функцій. Кількість використовуваних функцій дорівнює розмірності k архіву T. На ітерації i, використовується тільки інформація про i-ту розмірність розв'язку. Таким чином, на кожному етапі i маємо інше Гауссово ядро G^i.

Як випливає з рівності

$$G^i(x) = \sum_{l=1}^{k} \omega_l g_l^i(x) = \sum_{l=1}^{k} \omega_l \frac{1}{\sigma_l^i \sqrt{2\pi}} e^{-\frac{(x-\mu_l^i)^2}{2(\sigma_l^i)^2}}. \qquad (1.45)$$

Для визначення PDF G^i, повинні бути визначені вектори ω, μ^i і σ^i. Визначення векторів μ^i і ω описано вище. Більш складним є обчислення вектора стандартних відхилень σ^i. Розглянемо спочатку спосіб практичної реалізації рівняння (1.45).

На практиці процес відбору G^i здійснюється в наступний спосіб. Насамперед, обчислюються елементи вектора вагових коефіцієнтів ω відповідно до (1.44). Потім відбір здійснюється у два етапи. Перший етап полягає у виборі однієї з Гаусових функцій, що утворюють Гаусове ядро. Імовірність p_l вибору l-ї функції Гауса визначається формулою

$$p_l = \frac{\omega_l}{\sum_{r=1}^{k} \omega_r}. \qquad (1.46)$$

Друга фаза полягає в моделюванні обраної функції Гауса (тобто на кроці i – функції g_l^i). Це можна зробити або використовуючи генератор випадкових

чисел, який може генерувати випадкові числа відповідно до параметризованого нормального розподілу, або використовуючи генератор з рівномірним розподілом у комбінації з методом Box-Muller. Цей двофазний відбір еквівалентний відбору Гаусового ядра G^i відповідно до (1.45).

Зрозуміло, що на кроці i необхідно знати стандартне відхилення тільки для однієї функції Гауса $g_l^i(x)$, обраної на першій фазі. Отже, ми обчислюємо не весь вектор σ^i, а тільки його елемент σ_l^i. Вибір l-ї функції Гауса виконується для кожної мурахи на кожній ітерації. Це значить, що мураха використовує функцію Гауса відповідну до обраного розв'язку s_l. Таким чином, використовуються функції $g_l^i,\ i = 1, \dots, n$ для побудови повного розв'язку на даній ітерації. Зрозуміло, що обчислена функція Гауса відрізняється на кожному кроці i побудови розв'язку, при цьому $\mu^i = s_l^i$, а σ_l^i обчислюється динамічно в наступний спосіб. Для обчислення значення стандартного відхилення σ_l^i на кроці i, обчислюється середня відстань від обраного розв'язку s_l до іншого розв'язку в архіві помноженого на величину ξ, яка є параметром алгоритму

$$\sigma_l^i = \xi \sum_{e=1}^{k} \frac{\left| s_e^i - s_l^i \right|}{k-1}. \qquad (1.47)$$

Параметр $\xi > 0$, який є однаковим для всіх координат, має той же ефект, що й коефіцієнт випаровування в стандартному МАО. Чим більше значення ξ, тим менша швидкість збіжності алгоритму.

Відновлення рівня феромонів. Оскільки інформація про рівень феромонів зберігається в архіві, очевидно, що процедура його модифікації повинна зводитися до змін в архіві.

Розмір k архіву T є одним з параметрів алгоритму. Перед початком роботи розв'язки в архіві T ініціалізуються випадковими значеннями. Відновлення рівня

феромонів здійснюється додаванням набору знову згенерованих розв'язків до розв'язків з архіву з наступним видаленням такого ж числа гірших розв'язків. У такий спосіб розмір архіву залишається незмінним. Ця процедура гарантує, що в архіві зберігаються кращі розв'язки, дозволяючи, тим самим, ефективно направляти мураху в просторі пошуку.

Приклад в Додатку Б.

1.6. Метод бджолиної колонії

Багато складних задач оптимізації функції багатьох змінних не можуть бути вирішені до кінця в межах поліноміально обмеженого часу. Це викликає великий інтерес до алгоритмів пошуку, які дозволяють знайти оптимальні рішення за розумні проміжки часу[28,29].

В біології (і реальному світі), бджолина колонія може простягатися на великі відстані (більше 10 км) і в кількох напрямках одночасно використовувати велику кількість джерел нектару. Колонія процвітає, посилаючи робочих бджіл на продуктивні поля. Логічно, що, квіткові ділянки з великою кількістю нектару та пилку, які можна зібрати з меншими зусиллями, повинні відвідуватися більшою кількістю бджіл, тоді як ділянки з меншою їх кількістю повинні відвідувати менше бджіл [29].

Процес збору нектару починається з колонії, коли бджоли-розвідники відправляються на пошук перспективних квіткових ділянок. Бджоли-розвідники рухаються випадково з однієї ділянки на іншу. Під час збору врожаю колонія продовжує вести розвідку, зберігаючи відсоток населення як бджіл-розвідників.

Коли вони повертаються до вулика, то бджоли-розвідники, які знайшли ділянку, яка оцінюється вище певного порогу якості (вимірюється як комбінація деяких складових, таких як вміст цукру), залишають нектар та пилок з нього, і

йдуть на «танцювальний майданчик» виконати танець, відомий як «танець з похитуванням».

Цей загадковий танець має важливе значення для спілкування в колонії, і містить три частини інформації про квіткову ділянку: напрямок, в якому він знаходиться, його відстань від вулика та його рейтинг якості (чи фітнес-функція). Ця інформація допомагає колонії відправити своїх бджіл прямо до квіткових ділянок, не використовуючи довідників чи карт. Знання кожної бджоли про навколишнє середовище отримується виключно з танцю. Цей танець дає змогу колонії оцінити відносний внесок різних ділянок за якістю нектару, яку вони надають, та обсягом енергії, необхідної для її збирання. Після виконання даного танцю, розвідник повертається до ділянки з квітами з бджолами-спостерігачами, які очікували у вулику. Чим більш перспективна ділянка, тим більше спостерігачів буде відправлено на неї. Це дозволяє колонії збирати їжу швидко і ефективно.

Під час збирання з ділянки бджоли контролюють рівень нектару. Це необхідно для прийняття рішення під час наступного танцю, коли вони повернуться до вулика. Якщо ділянка все-таки має гарний показник як джерело нектару, тоді вона буде рекламуватися в танці, і більше бджіл будуть залучені до цього джерела.

У методі бджолиної колонії (МБК) колонія складається з трьох типів бджіл: робочі бджоли, спостерігачі та розвідники. Бджола, яка чекає біля танцювального майданчика для прийняття рішення про вибір джерела нектару, називається спостерігачем. Бджола, що переміщується до джерела нектару, що було відвідане нею раніше, називається робочою бджолою. Бджола, яка проводить випадковий пошук, називається розвідником. У МБК перша половина колонії складається з робочих бджіл, а друга половина є спостерігачами. Для кожного джерела нектару є тільки одна робоча бджола. Іншими словами,

кількість робочих бджіл дорівнює кількості джерел нектару навколо вулика. Робоча бджола, чиє джерело нектару виснажується робочими-бджолами (власне нею) та спостерігачами, стає розвідником. Основні кроки алгоритму такі:

1. Ініціалізація
2. Повторювати
    a. Розмістити робочих бджіл на джерелах нектару
    b. Розмістити бджіл-спостерігачів на джерелах нектару
    c. Відправити розвідників в пошукову область, для знаходження нових джерел нектару
3. Допоки (вимоги не виконані).

В алгоритмі МБК кожен цикл пошуку складається з трьох етапів: відправлення робочих бджіл до джерел нектару, а потім вимірювання їх кількості нектару; вибір джерел нектару спостерігачами після обміну інформацією про робочих бджіл та визначення кількості нектару в нектару; визначення розвідників, а потім їх відправлення на потенційні джерела нектару. На стадії ініціалізації бджоли довільно вибирають джерело харчування та визначають кількість нектару. Потім бджоли повертаються до вулика і діляться інформацією щодо нектару в джерелах з бджолами, що чекають на танцювальному майданчику (спостерігачами). На другому етапі, після обміну інформацією, кожна робоча бджола переходить до області джерела нектару, яку вона відвідувала на попередній ітерації, допоки цей ресурс харчування існує в її пам’яті, а потім вибирає нове джерело нектару серед сусідніх ділянок. На третьому етапі спостерігач визначає найкраще джерело продуктів харчування залежно від інформації про нектар, що демонструється робочими бджолами в танцювальній зоні.

Чим більша кількість нектару в джерелі нектару (ділянці квітів), тим вища ймовірність вибору цього джерела нектару спостерігачем. Отже, танець робочих

бджіл, що приносять якісніший нектар, приваблює спостерігачів до їх ділянок. Після прибуття на обрану ділянку, спостерігач обирає нове джерело нектару в околиці пам'яті в залежності від візуальної інформації. Візуальна інформація базується на порівнянні координати джерела нектару. Коли джерело нектару відкидається бджолами, нове джерело нектару випадковим чином визначається бджолою-розвідником і бджоли тепер працюють з ним. У нашій моделі, на кожній ітерації, один розвідник займається пошуком нового джерела нектару, а кількість робочих-бджіл дорівнює кількості бджіл-спостерігачів.

В МБК координати джерела нектару є потенційним розв'язком задачі оптимізації, а кількість нектару джерела відповідає якості (фітнес-функції) відповідного розв'язку. Кількість робочих бджіл та бджіл-спостерігачів дорівнює кількості потенційних координат.

На першому етапі МБК генерує довільно розподілену початкову популяцію для SN рішень (позиції джерел нектару), де SN позначає – колонії. Кожен розв'язок x_i $(i = 1, 2, \ldots, SN)$ є D-вимірним вектором. Після ініціалізації популяція модифікується на ітераціях $C = 1, 2, \ldots, MCN$, під час пошукових процесів робочих бджіл, бджіл-спостерігачів та бджіл-розвідників. Робочі бджоли модифікують координати в пам'яті в залежності від локальної інформації (візуальної інформації) та перевіряють кількість нектару (фітнес-функція) нового джерела (нові координати). За умови, що кількість нектару в нових координатах є вищою, ніж на попередніх, бджола запам'ятовує нові координати і забуває попередні. Інакше вона зберігає попередні координати в своїй пам'яті. Після того як усі робочі бджоли завершують процес пошуку, вони поширюють інформацію про нектар в джерелах нектару та їх місцезнаходження з бджолами-спостерігачами на танцювальному майданчику. Бджола-спостерігач оцінює інформацію про нектар, отриману від усіх робочих бджіл та обирає джерело нектару з ймовірністю, залежною від кількості нектару. Аналогічно до

робочої бджоли, бджола-спостерігач модифікує в своїй пам'яті інформацію щодо джерела нектару та перевіряє кількість нектару в джерелі кандидата. Якщо в новому джерелі нектару більше, ніж у попередньому, то бджола запам'ятовує нові координати і забуває попередні.

Спостерігач обирає джерело нектару в залежності від значення ймовірності для цього джерела p_i, що обчислюється як

$$p_i = \frac{fit_i}{\sum_{n-1}^{SN} fit_n} \tag{1.48}$$

де fit_i – значення фітнес-функції i, пропорційне кількості нектару в джерелі нектару в координатах i, а SN – кількість джерел нектару, що дорівнює кількості робочих бджіл (BN). Для того, щоб визначити координати потенційного джерела нектару, ABC використовує наступний вираз:

$$v_{ij} = x_{ij} + \varphi_{ij}\left(x_{ij} - x_{kj}\right), \tag{1.49}$$

де $k \in \{1,2,\dots,SN\}$ і $j \in \{1,2,\dots,D\}$ – довільно вибрані індекси. Незважаючи на те, що k визначається довільним чином, його значення має бути відмінним від значення i. φ_{ij} – довільне число з діапазону $[-1,1]$. Це число контролює вибір серед сусідніх джерел нектару навколо x_{ij} і являє собою візуальне порівняння двох джерел бджолою. Як видно з (49), при зменшенні різниці між параметрами x_{ij} та x_{kj}, x_{ij} також зменшується. Таким чином, з наближенням бджоли до оптимального рішення в пошуковому просторі, ширина кроку зменшується.

Якщо значення v_{ij}, виходить за допустимі границі, параметр встановлюється значенням границі, яку було порушено. Джерело нектару, яке було покинуте бджолами, замінюється новими джерелами від розвідників. У ABC, якщо поточне джерело не може бути покращеним за певну кількість ітерації, бджоли його залишають. Значення заздалегідь визначеної кількості

ітерацій для спроб покращення є важливим параметром контролю алгоритму ABC, який називається «лімітом» для відмови від джерела. Припустимо, що покинутим джерелом є x_i та $j \in \{1,2,\dots,D\}$, тоді розвідник знаходить нове джерело нектару, яке замінює x_i. Ця операція може бути визначена як

$$x_i^j = x_{min}^j + rand(0,1)\left(x_{max}^j - x_{min}^j\right) \quad (1.50)$$

Після того, як обчислено координати потенційного джерела v_{ij}, його фітнес порівнюється з фітнесом старого джерела. Якщо нове джерело має таку ж кількість нектару, чи більшу, ніж старе джерело, воно замінюється старим в пам'яті. В іншому випадку старе зберігається в пам'яті. Іншими словами, використовується «жадібний» механізм як функція вибору між старим і потенційним новим.

З наведеного вище пояснення видно, що в МБК є чотири параметри керування: кількість джерел нектару, яка дорівнює кількості робочих бджіл або спостерігачів (SN), значення лімітів та максимальне число ітерацій (MCN). Нижче наведено детальний псевдокод МБК:

1. Ініціалізувати популяцію координат x_{ij}, $i, j \in \{1,2,\dots,SN\}$, $j \in \{1,2,\dots,D\}$
2. Виконати обчислення для популяції
3. $iter = 1$
4. Повторювати
5. Згенерувати нові координати v_{ij} для робочих бджіл за допомогою (2) та обчислити їх (значення фітнес-функції)
6. Застосувати жадібний процес вибору
7. Розрахувати значення ймовірності p_{ij} для координат x_{ij} за допомогою (1)
8. Знайти нові координати v_{ij} для спостерігачів з координат x_{ij}, які вибираються в залежності від p_{ij} та обчислити для них значення фітнес-функцій

9. Застосувати жадібний процес вибору
10. Визначити покинуті координати для розвідника, якщо такі існують, і замінити їх новими випадковими координатами x_{ij} за допомогою (3)
11. Запам’ятати найкращі отримані координати
12. $iter = iter + 1$
13. Допоки $iter < MCN$.

Приклади МБК в Додатку В.

1.6.1. Модифікована версія МБК

На першому кроці алгоритму в точки, що описуються випадковими координатами, відправляється кілька бджіл-розвідників (нехай буде S бджіл, від слова scout). Залежно від значення цільової функції, яке визначається координатами бджоли, виділяються два види ділянок на поверхні функції, що їх цікавлять, поблизу яких може бути розташований глобальний максимум. А саме:

- Вибирається n кращих ділянок, де значення цільової функції є найбільшим;
- Вибирається m так званих перспективних ділянок, де значення цільової функції менше, ніж на найкращих ділянках, але ці ділянки все одно є непоганими з точки зору значення цільової функції.

Необхідно звернути увагу на одну особливість, яка може бути важливою при реалізації алгоритму. Кілька бджіл можуть потрапити на одну і ту ж ділянку (близькість ділянок, розмір ділянки або можлива близькість бджіл задається окремим параметром). Тому можна виділити два варіанти поведінки:

1. Вважати, що ці дві бджоли знайшли дві різних ділянки, що перетинаються, і обидві ці ділянки відзначити як найкращі або перспективні.
2. Вважати, що це одна ділянка, центр якої знаходиться в точці, яка відповідає бджолі з більшим значенням цільової функції.

У реалізації, яка буде описана нижче, використовується другий варіант поведінки, так як він виявися менш схильним до застрягання в локальних екстремумах.

В околицю n найкращих ділянок надсилаються N бджіл, а в околиці m перспективних ділянок – M бджіл, причому на кожну з найкращих ділянок має припадати більше бджіл, ніж на кожну з перспективних ділянок. Можна зробити так, що чим більше значення цільової функції, тим більша кількість бджіл відправлятиметься на відповідну ділянку, інакше можна N і M зробити фіксованими величинами.

Зверніть увагу, що бджоли надсилаються не в точності на те місце, де бджоли-розвідники знайшли перспективні і найкращі місця, а в їх околиці, більш точні координати визначаються випадковим чином. Крім того, околицю, що визначає область, в яку може бути надіслана бджола, можна зменшувати в міру збільшення номера ітерації, щоб поступово розв’язки збігалися до самого «кінця» екстремуму. Але якщо область зменшувати занадто швидко, то бджола може потрапити до локального екстремума.

Після того як бджоли були відправлені до найкращих і перспективних ділянок, можна відправити бджіл-розвідників на інші випадкові точки.

Після всіх цих операцій знову знаходяться n найкращих і m перспективних ділянок, на цей раз серед усіх бджіл з рою, а не тільки серед розвідників, і запам’ятовується найкраще значення функції (та його координати), значення більше якого поки не було знайдено, це і буде проміжним рішенням.

Потім алгоритм повторюється до тих пір поки не спрацює який-небудь з критеріїв зупинки. Цих критеріїв може бути кілька. Наприклад, якщо ми знаємо значення цільової функції в глобальному екстремумі, то можемо повторювати алгоритм до тих пір, поки функція не досягне деякого значення, близького до бажаного. Якщо значення функції в екстремумі невідомо, то можемо повторювати кроки алгоритму до тих пір, поки протягом якоїсь досить великої кількості ітерацій знайдений розв’язок не буде покращуватися (див. Додаток Г).

2. МЕТОДИ КЛАСТЕРІЗАЦІЇ

Останнім часом різке зростання використання Інтернету та поліпшення комунікацій в цілому перетворили наше суспільство на таке, що сильно залежить від інформації. Величезна кількість даних, що генерується цими процесами зв'язку, містить важливу інформацію, що щоденно накопичується в базах даних і її нелегко отримати. Область data mining розробляється як засіб для отримання інформації та знань з баз даних для виявлення шаблонів або концепцій, які не є очевидними.

Машинне навчання забезпечує технічну основу виведення даних шляхом отримання інформації з вихідних даних в базах даних. Процес зазвичай складається з наступного: перетворення даних у відповідний формат, очищення його та отримання висновку щодо даних.

- У найзагальнішому випадку розрізняють два типу машинного навчання: навчання по прецедентах, або індуктивне навчання, і дедуктивне навчання.
- Навчання по прецедентах, в свою чергу, поділяють на три основних типи: контрольоване навчання, або навчання з учителем (supervised learning), неконтрольоване навчання (unsupervised learning), або навчання без учителя, і навчання з підкріпленням (reinforcement learning)

В рамках неконтрольованого навчання одним з основних інструментів є кластеризація. В цьому посібнику розглядаються основні алгоритми, що використовуються для кластеризації, з коротким і простим описом кожного з них.

2.1. Кероване та некероване навчання

При керованому навчанні (навчанням з учителем) алгоритм забезпечується як наборами для навчання, так і мітками (те, що треба вивчити), що представляє собою концепцію, яку слід застосовувати для кожного випадку. Мета полягає в тому, щоб навчити систему так, щоб коли на вхід буде подано нове невідоме досі значення, система могла його правильно класифікувати. Під цією парадигмою існує можливість «обману» шляхом запам'ятовування всіх міток для кожного випадку, а не вивчення загальних прогностичних зв'язків між значеннями входів та виходів. Для уникнення цього, керовані алгоритми намагаються досягти балансу між навчальними даними та узагальненнями, що називається дилемою упередження/варіації.

Результати цього класу алгоритмів, як правило, оцінюються на відмінних між собою тренувальних наборах, які також називаються тестовим наборами. Методи варіюються від традиційних статистичних підходів до нейронних мереж і, останнім часом, векторних машин. З іншого боку, при некерованому навчанні алгоритм забезпечується лише вхідними наборами даних та відсутністю міток, завдання полягає в тому, щоб знайти відповідну класифікацію розподілених даних. Одним із основних підходів до некерованого навчання є кластеризація даних.

Інколи контрольоване та неконтрольоване навчання поєднуються в тому, що називається підпорядкованим навчанням. Неконтрольована частина зазвичай спочатку застосовується до даних, щоб виконати деякі дії щодо розподілу даних, а потім ці дії посилюються, використовуючи контрольований підхід.

Навчання з підкріпленням є окремим випадком контрольованого навчання, але вчителем в даному випадку є «середовище». Машина (її в цій ситуації часто називають «агент») не має попередньої інформацією про середовище, але має

можливість здійснювати в ній будь-які дії. Середа реагує на ці дії і тим самим надає агенту дані, які дозволяють йому реагувати на них і вчитися. Фактично агент і середовище утворюють систему зі зворотним зв'язком.

Для машинного навчання використовують різні технології та алгоритми. Зокрема, можуть застосовуватися дискримінантний аналіз, байєсовські класифікатори та багато інших математичних методів. Але в кінці XX століття все більше уваги почали приділяти штучним нейронним мережам (ШНМ).

ШНМ є системою з'єднаних і взаємодіючих між собою штучних нейронів, виконаних на основі порівняно простих процесорів. Кожен процесор ШНМ періодично отримує сигнали від одних процесорів (або від сенсорів, або від інших джерел сигналів) і періодично посилає сигнали іншим процесорам. Всі разом ці прості процесори, з'єднані в мережу, здатні вирішувати досить складні завдання.

Існує багато визначень задачі кластеризації. Найпростіше визначення виділяють серед усіх і воно містить одне фундаментальне поняття: об’єднання подібних об’єктів даних у кластери. Просте, формальне математичне визначення кластеризації полягає в наступному: нехай набір елементів даних $X \in R^{m \times n}$, що представляють множину m точок x_i в R^n. Мета полягає в тому, щоб розділити X на K груп C_K, так, щоб дані, які належать до однієї групи, були більш «схожі» на дані своєї групи, ніж дані в інших групах. Кожна з K груп називається кластером. Результатом алгоритму є пряме відображення $X \to C$ елементів даних x_i в кластери C_K.

Характер проблеми кластеризації такий, що ідеальний підхід еквівалентний глобальному вирішенню задачі нелінійної оптимізації. Це, як правило, є складним завданням. Насправді, це НП-складна задача, яку неможливо вирішити за поліноміальний час.

2.2. Види кластерізації

1. Ієрархічна (вкладена)
2. Часткова (не вкладена)

Найбільша відмінність між типами кластеризації: чи є набір кластерів вкладеним або не вкладеним. Часткова кластеризація – це розподіл набору об'єктів даних на підмножини, що не перетинаються. Якщо ми дозволяємо кластерами мати підкластери, то отримуємо ієрархічну кластеризацію, організовану як дерево. Кожен вузол (кластер) в дереві (крім листків) є об'єднанням його підкласів, а корінь дерева являє собою кластер, що містить всі об'єкти.

Ексклюзивна кластеризація – коли кожен об'єкт призначений для одного кластера. Кластеризація з перетинами – об'єкт може одночасно належати до кількох кластерів. При нечіткій кластеризації кожен об'єкт належить до кожного кластера з певною вагою, що знаходиться в межах від 0 (зовсім не належить) до 1 (повністю належить). Іншими словами, тут кластери – це нечіткі множини.

Формальне математичне визначення кластеризації полягає в наступному: нехай набір елементів даних $X \in R^{m \times n}$, що представляють множину m точок x_i в R^n.

Мета полягає в тому, щоб розділити X на K груп C_K, так, щоб дані, які належать до однієї групи, були більш «схожі» на дані своєї групи, ніж дані в інших групах.

Кожна з K груп називається кластером. Результатом алгоритму є пряме відображення $X \to C$ елементів даних x_i в кластери C_K.

Позначення:

x	об’єкт (точка)
C_i	i-й кластер
c_i	центроїд i-го кластера
c	центроїд усіх точок
m_i	кількість точок в i-му кластері
m	кількість точок у наборі даних
K	кількість кластерів

Мета кластеризації, як правило, виражається об’єктивною функцією, яка визначає залежність близькості точок одна до одної або до центроїдів кластерів; наприклад, мінімізувати квадратну відстань кожної точки до найближчого центроїда.

2.3. Базовий алгоритм k-means

Він розбиває безліч елементів векторного простору заздалегідь відоме число кластерів k. Дія алгоритму така, що він прагне мінімізувати середньоквадратичне відхилення на точках кожного кластера. Основна ідея полягає в тому, що на кожній ітерації перераховується центр мас для кожного кластера, отриманого на попередньому кроці, потім вектори розбиваються на кластери знову відповідно до того, який з нових центрів виявився ближчим за обраною метрикою. Алгоритм завершується, коли якоїсь ітерації немає зміни кластерів.

Проблеми алгоритму k-means: необхідно заздалегідь знати кількість кластерів. Алгоритм дуже чутливий до вибору початкових центрів кластерів. Класичний варіант має на увазі випадковий вибір кластерів, що дуже часто було джерелом похибки. Як варіант рішення необхідно проводити дослідження об'єкта для більш точного визначення центрів початкових кластерів.

Вибрати K точок за початкові центроїди

do {

Сформувати K кластерів, призначивши кожну точку найближчому центроїду

Обчислити центроїд кожного кластера

} while (центроїди змінюються)

Як признак завершення часто використовують наступну умову: менше 1% точок змінили кластери.

Щоб призначити точку до найближчого центроїда, необхідно використовувати міру близькості. Евклідова відстань (L_2) часто використовується для точок в евклідовому просторі, Манхеттенська (L_1) відстань може також бути використана для евклідових даних.

В евклідовому просторі ми використовуємо суму квадратів відхилень (SSE – sum of the squared error). Враховуючи два різних набори кластерів, отримані в результаті двох запусків K-means, ми надаємо перевагу тому, який має менший SSE.

SSE обчислюється наступним чином:

$$SSE = \sum_{i=1}^{k} \sum_{x \epsilon C_i} dist(x, c_i)^2,$$

де $dist$ – евклідова відстань.

Центроїд описується як

$$c_i = \frac{1}{m_i} \sum_{x \epsilon C_i} x.$$

Наприклад, якщо кластер містить три точки (1,1), (2,3), (6,2) його центроїдом буде ((1 + 2 + 6) / 3, (1 + 3 + 2) / 3) = (3,2).

Проте базовий алгоритм гарантує лише локальний мінімум SSE, оскільки він базується на оптимізації SSE для конкретного набору центроїдів та кластерів.

P-норма визначається наступним чином

$$\|x\| := \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p}.$$

Слід відзначити, що при $p = 1$ ми отримуємо Манхеттенську норму, при $p = 2$ ми отримуємо евклідову норму, а при наближенні p до ∞, p-норма то наближається до нескінченної норми або *максимальної норми*.

Це визначення все ще має певний сенс при $0 < p < 1$, але отримана функція не визначає норму, оскільки вона порушує нерівність трикутника.

Вибір початкових центроїдів

Коли використовується випадкова ініціалізація центроїдів, різні запуски алгоритму k-means зазвичай забезпечують різні значення SSE. Одним з ефективних підходів є взяття вихідних точок та їх кластеризація за допомогою ієрархічної кластеризації. K кластерів отримуються в результаті ієрархічної кластеризації, і їх центроїди використовуються як початковий. Цей підхід використовується, коли

- вибірка мала,

- значення K відносно невелике у порівнянні з розміром вибірки.

Інший підхід – вибрати першу точку випадковим чином або взяти центроїд з усіх точок. Потім для кожного наступного центроїду виділяти точку, розташовану далі від будь-якого з вже обраних початкових центроїдів. Таким чином ми отримуємо добре відокремлені випадкові центроїди. На жаль, цей

підхід дозволяє вибирати виходи, а не точки в щільних регіонах. Також обчислення набору початкових центроїдів є складним для обчислення.

Складність часу і простору

Вимоги до пам'яті для k-means є простими, оскільки зберігаються лише точки даних та центроїди – $O\big((m+k)n\big)$, де n – кількість елементів. Необхідний час – $O(I \cdot K \cdot m \cdot n)$, де I – кількість ітерацій.

Обробка порожніх кластерів

Одна з проблем полягає в тому, що пусті кластери можуть бути отримані, якщо протягом певної кількості ітерацій, жодна з точок не класифікується до цього кластера. Тому необхідна стратегія для вибору центроїда на заміну. Один з підходів – вибрати точку, розташовану далеко від будь-якого поточного центроїду. Інший підхід полягає в тому, щоб вибрати центроїд на заміну з кластера, який має найвищий SSE. Це, як правило, розбиває кластер і зменшує загальну SSE. Якщо є кілька порожніх кластерів, цей процес повторюється кілька разів.

Винятки

Коли використовується критерій SSE, винятки можуть вплинути на знайдені кластери. Зокрема, отримані центроїди можуть бути не такі репрезентативні, як вони були б без них. Через це корисно виявити винятки та заздалегідь їх усунути. В іншому випадку винятки можуть бути ідентифіковані на етапі фінальної обробки. Наприклад, ми можемо стежити за впливом кожної з точок на SSE та усунути точки з надзвичайно високим впливом.

Зменшення SSE на етапі фінальної обробки

Очевидним способом зменшення SSE є використання більшої кількості кластерів. Різні методи використовуються для «виправлення» результуючих кластерів та отримання кластерів з нижчим SSE.

Поступове оновлення центроїдів

Центроїди можна оновлювати поступово після кожної класифікації точки до кластера. Для цього на кожному кроці вимагається два або нуль оновлень центроїдів кластерів, оскільки точка переміщується до нового кластера (два оновлення) або залишається в поточному кластері (нуль оновлень). Використання цієї стратегії гарантує, що не створюються порожні кластери, оскільки всі кластери на початку мають лише одну точку, то ця точно кваліфікується до того самого кластера.

З іншого боку, оновлення центроїдів поступово створює залежності – отриманий кластер може залежати від порядку обробки точок. Можна використати довільний порядок обробки точок. Також поступове оновлення є дещо складнішим, але K-means збігається досить швидко, а кількість точок, що змінюють кластери швидко стає порівняно невеликою.

2.3.1. Поділ k-means

Це прямолінійне розширення базового k-means на основі простої ідеї: щоб отримати K кластерів, необхідно розподілити набір всіх точок на два кластери, вибрати один з цих кластерів для поділу на інші два і т.д., Поки не буде створено K кластерів.

```
Ініціалізувати список кластерів одним кластером, що містить усі точки
do {
        видалити кластер зі списку кластерів
        // виконати кілька «пробних» поділів вибраного кластера
        for (i = 1; i <<кількість спроб>; i ++)
                Розбити вибраний кластер за допомогою базового K-means
        Виділіть два кластери з поділу з найменшим SSE
        Додати ці два кластери до списку кластерів
}
while (список містить K кластерів)
```

Існує кілька способів вибору кластера для поділу. На кожному кроці ми можемо вибрати найбільший кластер, або вибрати той, у якого найбільше значення SSE, або використати критерій на основі розміру та SSE.

K-means з поділом менш чутливий до проблеми ініціалізації.

2.3.2. k-means++

Щоб розподілити k початкових центроїдів кластерів подалі один від одного, перший центроїд вибирається рівномірно випадковим чином з точок даних для кластеризації, після чого кожен наступний кластерний центр вибирається з залишкових точок даних з ймовірністю, пропорційною квадрату її відстані до найближчого центроїда.

Точний алгоритм виглядає наступним чином:

1. Серед існуючих точок, що необхідно розподілити на кластери вибрати центроїд довільним чином;
2. Для кожної точки, що не є центроїдом, обчислити значення квадрату відстані до найближчого центроїда серед вже вибраних, а також підраховувати суму квадратів відстаней кожної точки до найближчого центроїда $\sum r_x^2$;

3. Вибрати серед точок, що залишились наступний центроїд таким чином, щоб ймовірність вибору кожної точки залежала від її значення квадрату відстані до найближчого центроїда r_x^2. Для цього необхідно спочатку визначити довільне число з інтервалу $[0, \sum r^2]$, а потім перерахувати суму відстаней кожної точки до найближчого центроїда у тому ж порядку, що і у пункті 2. Як тільки поточне значення суми буде, з визначеним відхиленням, дорівнювати попередньо вибраному довільному числу, вибрати дану точку в якості центроїда;
4. Повторювати пункти 2-3 допоки не буде вибрано необхідну кількість центроїдів;
5. Тепер, коли є центроїди, кластеризація виконується за допомогою k-means.

Цей метод дає значні покращення в остаточній похибці k-means. Хоча початковий вибір центроїдів займає додатковий час, основна частина k-means сама по собі збігається дуже швидко, і таким чином алгоритм фактично знижує час обчислення. Автори перевіряли свій метод з реальними та штучними наборами даних і отримали, 2-кратні покращення швидкості, а для деяких наборів даних – майже 1000-кратне покращення помилки. Їх випробування показали, що новий спосіб є як мінімум таким же по якості, як базовий k-means, як за швидкістю, так і за SSE.

Крім того, автори обчислюють коефіцієнт наближення для свого алгоритму. Алгоритм k-means++ гарантує коефіцієнт наближення $O(\log k)$, де k – кількість кластерів.

При ієрархічній кластеризації виконується послідовне об’єднання менших кластерів в більші або поділ більших кластерів на менші.

2.4. Агломеративні методи AGNES (Agglomerative)

Алгоритм k-means дає нам те, що іноді називається простим результатом, тому що він просто дає нам єдиний набір кластерів, у яких немає конкретної організації або структури. Але може виявитись так, що деякі кластери можуть бути дуже схожими на одні кластери і менш схожими на інші (якщо ми збираємо медичні записи пацієнтів, то ми хочемо, щоб «скарги на респіраторне захворювання» виглядали як батьківський кластер для кластерів «пневмонія», «грип», «SARS»).

Отже, іноді необхідно отримати ієрархічну кластеризацію, яка подається у вигляді дерева або дендрограми.

Існує два підходи до ієрархічної кластеризації: ми можемо йти «знизу вгору», об'єднуючи невеликі кластери у великі, або «зверху вниз», розбиваючи великі кластери на малі. Вони називаються агломераційною і кластеризацією, що роз'єднує відповідно. Базовий алгоритм агломераційної кластеризації дуже простий:

1. Розпочати з кожної точки у власному кластері
2. Допоки не залишиться лише один кластер:
    a. Знайти найближчу пару кластерів
    b. Об'єднати їх.

Будь-яка така процедура є «жадібною», як і наш алгоритм вибору функцій, та детерміністичною (без випадкових початкових умов, на відміну від k-means). Він повертає послідовність вкладених рівнів, де кожен рівень об'єднує два підлеглі рівні.

Щоб перетворити кластеризацію на визначену процедуру, ми повинні мати можливість оцінювати, наскільки подібні два кластери. Це не збігається з тим, наскільки близькі дві точки даних, або наскільки близькі два рівні. Існує кілька основних варіантів визначення цього.

2.5. Однолінійна кластерізація

Алгоритм однолінійної кластеризації є прикладом агломеративного ієрархічного методу кластеризації. Кожен об’єкт розміщується в окремому кластері, і на кожному кроці ми об’єднуємо найближчу пару кластерів, допоки не будуть виконані певні умови завершення. Це вимагає визначення поняття подібності кластерів.

Для однолінійної кластеризації подібність двох кластерів визначається як мінімальна відстань між будь-якими двома точками цих кластерів.

Однолінійна кластеризація визначає відстань між двома кластерами як мінімальну відстань між їх елементами:

$$d(A,B) = \min_{\vec{x}\in A,\vec{y}\in B} \|\vec{x} - \vec{y}\|.$$

Дана кластеризація називається однолінійною, оскільки вона вказує на те, що кластери подібні, якщо вони мають хоча б одну пару подібних точок, єдину «лінію». Це допомагає розв’язувати задачі зі складними формами кластерів. Проте, цей алгоритм лише займається розподілом, і не піклується про компактність або рівномірність.

2.6. Повнозв’язана кластерізація

Кластери повного зв’язку – це кластери, де відстань між кластерами є максимальною відстанню між їх членами.

$$d(A,B) = \max_{\vec{x}\in A,\vec{y}\in B} \|\vec{x} - \vec{y}\|.$$

Знову ж таки, є ситуації, коли цей підхід дає гарні результати, а інколи – погані. Основні слабкі сторони ієрархічних методів кластеризації включають в себе те, що вони не добре масштабуються: складність часу становить щонайменше $O(N^2)$, де N – кількість об’єктів, а також те що вони ніколи не зможуть скасувати результати попередніх ітерацій. Це і є яскравим прикладом жадібного підходу.

2.7. Метод найближчого сусіда або метод одиночного зв‘язку

Безліч методів ієрархічного кластерного аналізу розрізняються не тільки використовуваними мірами подібності та відмінності, а й алгоритмами класифікації. З них найбільш поширеними є *метод найближчого сусіда*. Цей метод відомий також під назвою метода одиничного зв’язку.

Нехай необхідно провести класифікацію заданої множини об’єктів методом найближчого сусіда. Відстань між двома класами визначається як відстань між найближчими їх представниками.

Перед початком роботи алгоритму розраховується матриця відстаней між об’єктами. На кожному кроці в матриці відстаней знаходиться мінімальне значення, що відповідає відстані між двома найближчими кластерами. Знайдені кластери об’єднуються, утворюючи новий кластер. Ця процедура повторюється до тих пір, поки не будуть об’єднані всі кластери.

При використанні методу найближчого сусіда особливу увагу слід приділяти вибору міри відстані між об’єктами. На основі неї формується початкова матриця відстаней, яка і визначає весь подальший процес класифікації.

2.8. Метод найбільш віддалених сусідів або метод повного зв‘язку

Перед початком роботи алгоритму розраховується матриця відстаней між об’єктами. На кожному кроці в матриці відстаней знаходиться мінімальне значення, що відповідає відстані між двома найближчими кластерами. Знайдені кластери об’єднуються, утворюючи новий кластер. Ця процедура повторюється до тих пір, поки не будуть об’єднані всі кластери.

2.9. Метод зваженого попарного середнього(WPGMA)

Метод зваженого попарного середнього – Weighted Pair-Group Method Using Arithmetic Averages або скорочено WPGMA.

Нехай необхідно виконати класифікацію заданої множини об’єктів методом зваженого попарного середнього.

Перед початком роботи алгоритму розраховується матриця відстаней між об’єктами. На кожному кроці в матриці відстаней знаходиться мінімальне значення, що дорівнює відстані між двома найближчими кластерами. Знайдені кластери u і v об’єднуються, утворюючи новий кластер. Рядки і стовпці, що відповідають кластерам u і v, видаляються з матриці відстаней, і додається новий рядок та новий стовпець, що відповідає кластеру k. В результаті матриця зменшується на один рядок і один стовпець. Ця процедура повторюється до тих пір, поки не будуть об’єднані всі кластери.

2.10. Метод невиваженого попарного середнього(UPGMA)

Метод невиваженого попарного середнього – Unweighted Pair-Group Method Using Arithmetic Averages або скорочено UPGMA.

Перед початком роботи алгоритму розраховується матриця відстаней між об’єктами. На кожному кроці в матриці відстаней знаходиться мінімальне значення, що відповідає відстані між двома найближчими кластерами. Знайдені кластери u і v об’єднуються, утворюючи новий кластер k. Рядки і стовпці, що відповідають кластерам u і v, видаляються з матриці відстаней, і додається новий рядок та новий стовпець, що відповідає кластеру k. В результаті матриця зменшується на один рядок і один стовпець. Ця процедура повторюється до тих пір, поки не будуть об’єднані всі кластери.

3. ШТУЧНІ НЕЙРОННІ МЕРЕЖІ

3.1. Історія нейронних мереж

Вивченню людського мозку тисячі років. З появою сучасної електроніки почалися спроби апаратного відтворення процесів мислення. Перший крок було зроблено в 1943 році. З виходом статті нейрофізіолога Уоррена Маккалоха (Warren McCulloch) та математика Уолтера Питтаса (Walter Pitts) про роботу штучних нейронів та представлення моделі нейронної мережі на електричних схемах.

• 1949 р. - опублікована книга Дональда Хебба (Donald Hebb) "Організація поведінки", де досліджується проблематика налаштування синаптичних зв'язків між нейронами.

• 1950-ті рр. - з'являються програмні моделі штучних нейромереж. Перші роботи провів Натаніаль Рочестер (Nathaniel Rochester) з дослідницької лабораторії IBM. І хоча подальші реалізації були успішними, ця модель була невдалою, оскільки бурхливий розвиток традиційних обчислень залишило в тіні нейронні дослідження.

• 1956 р. - Дартмутський дослідницький інститут штучного інтелекту забезпечив підйом штучного інтелекту, зокрема, нейронних мереж. Дослідження штучного інтелекту розподіляється на два напрямки: промислові застосування систем штучного інтелекту (експертні системи) та моделювання мозку.

• 1958 р. - Джон фон Нейман (John von Neumann) запропонував імітацію простых функцій нейронів з використанням вакуумних трубок.

• 1959 р. - Бернард Видров (Bernard Widrow) і Марсіан Хофф (Marcian Hoff) розробили моделі ADALINE і MADALINE (Множинні Адаптивні Лінійні Елементи). MADALINE діяла, як адаптивний фільтр для видалення відлуння на

телефонних лініях. Ця нейромережа до сьогодняшнього дня в комерційному використанні.

• Нейробіолог Френк Розенблатт (Frank Rosenblatt) почав роботу над перцептоном. Одношаровий перцептрон був побудований апаратно і вважається класичною нейромережею. Тоді перцептрон використовувався для класифікації вхідних сигналів в один із двох класів. На жаль, одношаровий перцептрон був обмеженим і його було розкритиковано в 1969 році в книзі Марвіна Мінського (Марвін Мінський) та Сеймура Пейперта (Seymour Papert) "Перцептрони".

Ранні успіхи, сприяли підвищенню потенціалу нейронних мереж, зокрема в світлі обмеженої на той час електроніки. Надмірне очікування, що процвітає в академічному та технічному світі, заразило загальну літературу цього часу. Впевненість, що ефект "мислячої машини" відобразиться на людині, весь час підігрівалася письменникам, зокрема, серія книг Азімова про роботів показала наслідки на моральних цінностях людини у разі можливості інтелектуальних роботів виконувати функції людини.

Ці загрози, об'єднані з невиконаними обіцянками, викликали безліч розчарувань фахівців, що критикували дослідження нейронних мереж. Результатом стало припинення фінансування. Період спаду продовжився до 80-х років.

• 1982 р. - до відродження интересу привело кілька подій. Джон Хопфілд (John Hopfield) представив статтю в національну Академію Наук США. Підхід Хопфілда показав можливості моделювання нейронних мереж на принципі нової архітектури.

• У той же час у Кіото (Японія) відбулася Об'єднана американо-японська конференція з нейронних мереж, які оголосили досягненням п'ятої генерації. Американські періодичні видання підняли цю історію, акцентуючи, що США

можуть залишитися позаду, що привело до зростання фінансування в області нейромереж.

• З 1985 р. Американський Інститут Фізики започаткував щорічні зустрічі - "Нейронні мережі для обчислення".

• 1989 р. - на зустрічі "Нейронні мережі для оборони" Бернард Видров повідомив аудиторію про початок четвертої світової війни, де полем бою є світові ринки та виробництво.

• 1990 г. - Департамент програм інноваційних досліджень захисту малого бізнесу назвав 16 основних та 13 додаткових тем, де можливе використання нейронних мереж.

Сьогодні обговорення нейронних мереж відбувається всюди. Перспектива їх використання видається досить яскравою у світлі рішення нетрадиційних проблем і є ключем до цілої технології. На даний момент більшість розробок нейронних мереж принципово працюють, але можуть існувати процесорні обмеження. Дослідження спрямовані на програмні й апаратні реализації нейромереж. Компанії працюють над створенням трьох типів нейрочіпів: цифрових, аналогових та оптичних, які обіцяють бути хвилею близького майбутнього.

3.2. Застосування нейромереж для вирішення практичних задач

Класифікация образов. Задача полягає у визначенні належності вхідного образу (наприклад, мовного сигналу або рукописного символу), поданого вектором ознак, до одного або декількох попередньо визначених класів. До відомих додатків належать розпізнавання букв, розпізнавання мови, класифікація сигналу електрокардіограми, класифікація клітин крові.

Кластерізация / категорізация. При вирішенні задач кластеризації, навчальний набір не має міток класів. Алгоритм кластерізації ґрунтується на подібності образів і вміщує схожі образи в один кластер. Відомі випадки застосування кластеризації для добування знань, стискування даних і дослідження властивостей даних.

Апроксимація функцій. Припустимо, що є навчальна вибірка ((x_1,y_1), (x_2,y_2)..., (x_n,y_n)) (пари даних вхід-вихід), яка генерується невідомою функцією F, спотвореною шумом. Задача апроксимації полягає в знаходженні невідомої функції F. Апроксимація функцій необхідна при розв'язуванні численних інженерних і наукових задач моделювання.

Передбачення/прогноз. Нехай задано n дискретних відліків {в(t_1), в(t_2) ..., в(t_n)} у послідовні моменти часу t_1, t_2..., t_n . Задача полягає в передбаченні значення в(t_n+1) у наступний момент часу t_n+1. Передбачення/прогноз мають велике значення для прийняття рішень в бізнесі, науці й техніці (передбачення цін на фондовій біржі, прогноз погоди).

Оптимізація. Численні проблеми в математиці, статистиці, техніці, науці, медицині і економіці можуть розглядатися як проблеми оптимізації. Задачею алгоритму оптимізації є знаходження такого рішення, що задовольняє системі обмежень і максимізує або мінімізує цільову функцію.

Пам'ять, що адресується за змістом. У традиційних комп'ютерах звернення до пам'яті доступно лише за допомогою адреси, не залежної від вмісту пам'яті. Більше того, якщо допущено помилку в обчисленні адреси, то може бути знайдена зовсім інша інформація. Асоціативна пам'ять або пам'ять, що адресується за змістом, доступна за вказівкою заданого вмісту. Вміст пам'яті може бути викликано навіть за частковим входом або пошкодженим вмістом. Асоціативна пам'ять може бути використана в мультімедійніх інформаційних базах даних.

Управління. Розглянемо динамічну систему, задану сукупністю {u(t), y(t)}, де u(t) - вхідний керуючий вплив, а y(t) - вихід системи в момент часу t. У системах управління з еталонною моделлю метою управління є розрахунок такого вхідного впливу u(t), при якому система діє за бажаною траєкторією, заданою еталонною моделлю. Прикладом є оптимальне управління двигуном.

Але, незважаючи на переваги нейронних мереж в окремих галузях над традиційними обчисленнями, існуючі нейромережі не є досконалими рішеннями. Вони навчаються і можуть робити "помилки". Крім того, не можна гарантувати, що розроблена мережа буде оптимальною мережею. Застосування нейромереж вимагає від розробника виконання ряду умов:

- множину даних, які містять інформацію, що характеризує проблему;
- відповідно встановлен за розмірами множина даних для навчання і тестування мережі;
- розуміння базової природи проблеми, що розв’язується;
- вибір функції суматора, передавальної функції і методів навчання;
- розуміння інструментальних засобів розробника;
- відповідна потужність обробки.

Нові можливості обчислень вимагають від розробника умінь поза межами традиційних обчислень. Спочатку, обчислення були лише апаратними і керували їх роботою інженери. Потім були фахівці з програмного забезпечення: програмісти, системні інженери, фахівці по базах даних і проектувальники. Тепер з’явилися нейронні архітектори. Новий професіонал повинен мати кваліфікацію, вище ніж у попередників. Наприклад, він має знати статистику для вибору і оцінювання навчальних і тестових множин.

При створенні ефективних нейромереж, важливим для сучасних інженерів програмного забезпечення є логічне мислення, емпіричне вміння та інтуїція.

Хоча було висунуте допущення, що в мозку людини кількість різних типів нейронів в мозку людини лежить між 100 та 1000, переважно, це лише спеціалізовані типи, що базуються та походять від звичайного базового нейрона.

Так само, як є певний базовий біологічний нейрон, існує базовий штучний нейрон.

Кожен нейрон має певну кількість входів, кожен з яких має певну вагу, що йому присвоєна.

Ці ваги просто показують, наскільки “важливим” вхідний сигнал на цьому вході є. Net value нейрона розраховується на основі ваг — фактично, це лише зважена сума, сума всіх значень входів, помножена на їх вагу. Кожен нейрон має своє власне порогове значення, і коли зважена сума нейрона більше порогу, нейрон видає на вихід значення 1. В зворотньому випадку, якщо порогове значення не досягнуто, нейрон видає на вхід 0. Вихід нейрона передається на вхід на входи всіх нейронів, до яких він під’єднаний.

Незважаючи на істотні відмінності, окремі типи НМ мають декілька загальних рис.

По-перше, основу кожної НМ складають відносно прості, переважно однотипні, елементи (комірки), що імітують роботу нейронів мозку. Основними компонентами нейромережі є нейрони /neurons/ (елементи, вузли), які з’єднані зв’язками. Сигнали передаються по зваженим зв’язкам (connection), з кожним з яких пов’язаний ваговий коефіцієнт (weighting coefficient) або вага.

Далі під нейроном буде матися на увазі штучний нейрон, тобто комірка НМ. Кожен нейрон характеризується своїм поточним станом за аналогією з нервовими клітинами головного мозку, які можуть бути збуджені або загальмовані. Він має групу синапсів - односпрямованих вхідних зв’язків, з’єднаних з виходами інших нейронів, а також має аксон - вихідний зв’язок

даного нейрона, з якого сигнал (збудження або гальмування) надходить на синапси наступних нейронів. Загальний вигляд нейрона наведено на рисунку 3.1.

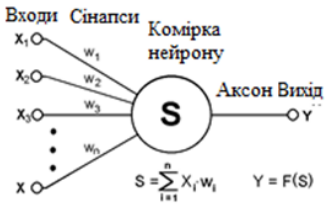


Рисунок 3.1 – Штучний нейрон

Кожен синапс характеризується величиною синаптичного зв'язку або її вагою w_i, що за фізичним змістом еквівалентна електричної провідності.

Поточний стан нейрона визначається, як зважена сума його входів:

$$s = \sum_{i=1}^{n} x_i w_i \qquad (3.1)$$

Вихід нейрона є функцією його стану:

$$y = f(s) \qquad (3.2)$$

Штучний нейрон імітує в першому наближенні властивості біологічного нейрона. На вхід штучного нейрона поступає множина сигналів, які є виходами інших нейронів. Кожен вхід множиться на відповідну вагу, аналогічну його синаптичній силі, і всі виходи підсумовуються, визначаючи рівень активації нейрона. Хоча мережеві парадигми досить різноманітні, в основі майже всіх їх лежить ця конфігурація. Тут множина вхідних сигналів, позначених x_1, x_2,…, x_n, поступає на штучний нейрон. Ці вхідні сигнали, в сукупності позначаються вектором X, відповідають сигналам, що приходять в синапси біологічного нейрона. Кожен сигнал множиться на відповідну вагу w_1, w_2,…, wn, і поступає на сумуючий блок, позначений Σ. Кожна вага відповідає «силі» одного

біологічного синаптичного зв'язку (множина ваг в сукупності позначається вектором W). Блок сумування, який відповідає тілу біологічного нейрона, складає зважені входи алгебраїчно, створюючи вихід.

У сучасній літературі зустрічається велика кількість парадигм штучних нейронних мереж, елементи яких реалізують різні активаційні функції. Найбільш поширеною є сигмоїдальна функція, що може бути представлена в дискретному та аналоговому варіанті.

Нелінійна функція f називається активаційною і може набувати різного вигляду, як показано на рисунку 3.2. Однією з найпоширеніших є нелінійна функція з насиченням, так звана логістична функція, або сигмоїда (тобто функция S-подібного вигляду) :

$$f(x)=\frac{1}{1+e^{-\alpha x}} \qquad (3.3)$$

Зі зменшенням α сигмоїда стає більш пологою, в межі при α=0 вироджуючись в горизонтальну лінію на рівні 0.5, зі збільшенням α сигмоїда наближається за зовнішнім виглядом до функції одиничного стрибка з порогом T в точці x=0. З виразу для сигмоїди очевидно, що вихідне значення нейрона лежить в діапазоні [0,1]. Одна з цінних властивостей сігмоідной функції - простий вираз для її похідної

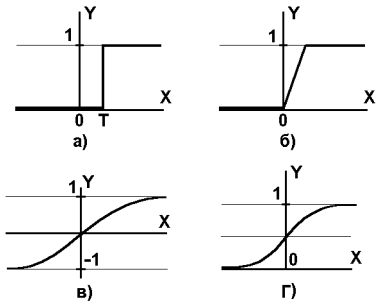


Рисунок 3.2 – Активаційні функції

а) функція одиничного стрибка; б) лінійний порог (гістерезис); в) сигмоїда – гіперболічний тангенс; г) сигмоїда – формула (3.3)

Слід зазначити, що сигмоїдна функція диференційована на всій осі абсцис, що використовується в деяких алгоритмах навчання. Крім того вона має властивість підсилювати слабкі сигнали краще, ніж великі, і запобігає насиченню від великих сигналів, оскільки вони відповідають областям аргументів, де сигмоїда має пологий нахил.

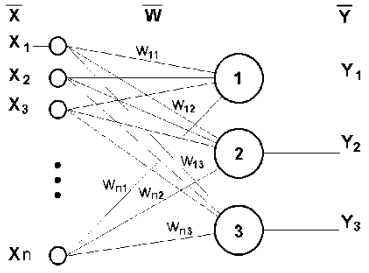


Рисунок 3.3 Одношаровий перцептрон

Повертаючись до загальних рис, властивих всім НМ, зазначимо, по-друге, принцип паралельної обробки сигналів, що досягається шляхом об'єднання великої кількості нейронів у так звані шари і з'єднання певним чином нейронів різних шарів, а також, у деяких конфігураціях, і нейронів одного шару між собою, причому обробка взаємодії всіх нейронів ведеться пошарово.

Як приклад найпростішої НМ розглянемо тринейронний перцептрон (рис.3), тобто таку мережу, нейрони якої мають активаційну функцію у вигляді одиничного стрибка. На n входів надходять деякі сигнали, що проходять по синапсах на три нейрони, що утворюють єдиний шар цієї НМ та видають три вихідних сигнали:

$$y_j = f\left[\sum_{i=1}^{n} x_i \cdot w_{ij}\right], \quad j=1...3 \qquad (3.4)$$

Очевидно, що всі вагові коефіцієнти синапсів одного шару нейронів можна звести в матрицю W, в якій кожен елемент w_{ij} задає величину і-го синаптичного зв'язку j-го нейрона. Таким чином, процес, що відбувається в НМ, може бути записано в матричній формі:

$$Y=F(XW) \qquad (3.5)$$

де X и Y – відповідно вхідний і вихідний сигнальні вектори, F(V) – активаційна функція, що застосовується поелементно до компонентів вектора V.

Теоретично кількість шарів і кількість нейронів у кожному шарі може бути довільною, однак фактично вона обмежена ресурсами комп'ютера або спеціалізованої мікросхеми, на яких зазвичай реалізується НМ. Що складнішою є НМ, то масштабнішими є задачі, підвладні їй.

Вибір структури НМ здійснюється відповідно до особливостей і складністю задачі. Для розв'язування деяких окремих типів задач вже існують оптимальні, на сьогоднішній день, конфігурації.

Якщо ж задача не може бути зведеною до жодного з відомих типів, розробнику доводиться розв'язувати складну проблему синтезу нової конфігурації.

Важливим фактором ефективності мережі є встановлення оптимальної кількості нейронів та типів зв'язків між ними. Під час опису нейромереж використовують кілька усталених термінів, які в різних джерелах можуть мати різне трактування, зокрема:

- структура нейромережі – спосіб зв'язків нейронів у нейромережі;
- архітектура нейромережі – структура нейромережі та типи нейронів;
- парадигма нейромережі – спосіб навчання та використання; іноді вміщує й поняття архітектури.

На основі однієї архітектури можуть бути реалізовані різні парадигми нейромережі і навпаки.

Аналізуючи найвідоміші на сьогодні розробки нейромереж, слід зазначити, що найпоширенішим варіантом архітектури є багатошарові мережі. Нейрони у цьому випадку об'єднуються у прошарки з єдиним вектором сигналів входів. Зовнішній вхідний вектор подається на вхідний прошарок нейронної мережі (рецептори). Виходами нейронної мережі є вихідні сигнали останнього прошарку (ефектори).

При цьому він керується кількома основними принципами: можливості мережі зростають зі збільшенням числа комірок мережі, щільності зв'язків між ними і числом виділених шарів, введення зворотних зв'язків поряд зі збільшенням можливостей мережі піднімає питання про динамічну стійкість мережі; складність алгоритмів функціонування мережі (в тому числі, наприклад, введення декількох типів синапсів - збуджуючих, гальмівних і ін.) також сприяє посиленню потужності НМ. Питання про необхідні і достатні властивості мережі для розв'язування того чи іншого роду задач є цілий напрям нейрокомп'ютерної науки. Оскільки проблема синтезу НМ дуже залежить від задачі, що розв'язується, надати загальні докладні рекомендації важко. У більшості випадків оптимальний варіант виходить на основі інтуїтивного добору.

Деякі типові архітектури нейронних мереж.

Мережі прямого розповсюдження:

- перцептрони;
- мережа зворотного (Back Propagation) розповсюдження;
- мережа зустрічного (Counter Propagation) розповсюдження;
- карта Кохонена.

Рекурентні мережі:

- мережа Хопфілда;
- мережа адаптивної резонансної теорії;
- двоспрямована асоціативна пам'ять.

Очевидно, що процес функціонування НМ, тобто сутність процесів, які вона здатна виконувати, залежить від величин синаптичних зв'язків, тому, задавшись певною структурою НМ, що відповідає будь-якій задачі, розробник мережі має знайти оптимальні значення всіх змінних вагових коефіцієнтів (деякі синаптичні зв'язки можуть бути постійними).

Цей етап називається навчанням НМ, і від того, наскільки якісно його буде виконано, залежить здатність мережі розв'язувати поставлені перед нею проблеми під час експлуатації. На етапі навчання крім параметра якості добору ваг важливу роль відіграє час навчання. Як правило, ці два параметри пов'язані зворотною залежністю і їх доводиться вибирати на основі компромісу.

Навчання НМ може відбуватися з вчителем або без нього. У першому випадку мережі висуваються значення як вхідних, так і бажаних вихідних сигналів, і вона за деяким внутрішнім алгоритмом підлаштовує ваги своїх синаптичних зв'язків. У другому випадку виходи НМ формуються самостійно, а ваги змінюються за алгоритмом, що враховує тільки вхідні і похідні від них сигнали.

Існує безліч різних алгоритмів навчання, які, однак, діляться на два великі класи: детерміновані і стохастичні. У першому з них підстроювання ваг являє собою жорстку послідовність дій, у другому - вона виробляється на основі дій, що підлягають деякому випадковому процесу.

Розвиваючи далі питання щодо можливої класифікації НМ, важливо відзначити існування бінарних і аналогових мереж. Перші з них оперують з двійковими сигналами, і вихід кожного нейрона може приймати тільки два значення: логічний нуль ("загальмований" стан) і логічна одиниця ("збуджений" стан). До цього класу мереж належить і розглянутий вище перцептрон, так як виходи його нейронів, що формуються функцією одиничного стрибка, дорівнюють або 0, або 1. В аналогових мережах вихідні значення нейронів здатні

приймати безперервні значення, що могло б мати місце після заміни активаційної функції нейронів перцептрону на сигмоїду.

Ще одна класифікація ділить НМ на синхронні й асинхронні. У першому випадку в кожен момент часу свій стан змінює лише один нейрон. У другому - стан змінюється відразу в цілої групи нейронів, як правило, у всього шару. Алгоритмічно хід часу в НМ задається ітераційним виконанням однотипних дій над нейронами. Далі будуть розглядатися тільки синхронні НМ.

Мережі також можна класифікувати за кількістю шарів. На рисунку 3.4 подано двошаровий перцептрон, отриманий з перцептрону з рисунку 3.3 шляхом додавання другого шару, що складається з двох нейронів.

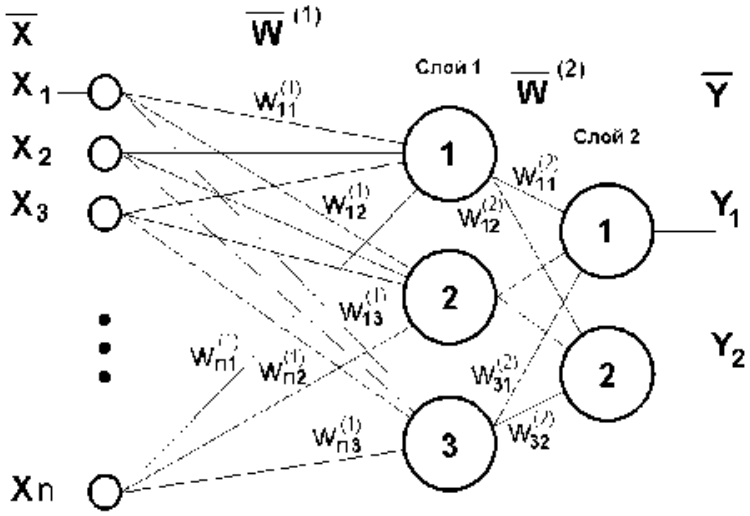


Рисунок 3.4 - Двошаровий перцептрон

Тут доречно відзначити важливу роль нелінійності активаційгої функції, так як, якщо б вона не володіла даною властивістю або не входила в алгоритм роботи кожного нейрона, результат функціонування будь-якої р-шарової НМ з ваговими матрицями $\mathbf{W}^{(i)}$, i=1,2,...p для кожного шару і зводився б до перемножування вхідного вектора сигналів X на матрицю

$$W^{(0)}=W^{(1)}xW^{(2)}\,x...xW^{(p)} \qquad (3.6)$$

тобто фактично така p-шарова НМ еквівалентна одношаровій НМ з ваговою матрицею єдиного шару $W^{(0)}$:

$$Y=XW \qquad (3.7)$$

Продовжуючи розмову про нелінійність, можна зазначити, що вона іноді вводиться і в синаптичні зв'язки. Більшість відомих на сьогодні НМ використовують для знаходження зваженої суми входів нейрона формулу (3.1), але для деяких застосуваннях НМ корисно ввести інший запис, наприклад:

$$s = \sum_{i=1}^{n} x_i^2 \cdot w_i \qquad (3.8)$$

або навіть

$$s = \sum_{i=1}^{n} x_i \cdot x_{((i+1)\mathrm{mod}n)} \cdot w_i \qquad (3.9)$$

Питання в тому, щоб розробник НМ чітко розумів, для чого він це робить, якими цінними властивостями він тим самим додатково наділяє нейрон, і яких позбавляє. Введення такого роду нелінійності, взагалі кажучи, збільшує обчислювальну потужність мережі, тобто дозволяє з меншого числа нейронів з "нелінійними" синапсами сконструювати НС, що виконує роботу звичайної НМ з великим числом стандартних нейронів і більш складної конфігурації.

Різноманіття існуючих структур НМ дає змогу відшукати й інші критерії для їх класифікації

Тепер розглянемо один нюанс, навмисно опущений раніше. З рисунка (3.2а)функції одиничного стрибка видно, що граничне значення T, в загальному випадку, може приймати довільне значення. Більш того, воно має приймати якесь довільне, невідоме заздалегідь значення, яке добирається на стадії навчання разом з ваговими коефіцієнтами. Те ж саме стосується й центральної

точки сигмоїдної залежності, яка може зсуватися праворуч або ліворуч по осі X, а також і всіх інших активаційних функцій. Це, однак, не відображено у формулі (3.1), яка мала була б виглядати так:

$$s = \sum_{i=1}^{n} x_i \cdot w_i - T \qquad (3.10)$$

Справа в тому, що таке зміщення зазвичай вводиться шляхом додавання до шару нейронів ще одного входу, що порушує додатковий синапс кожного з нейронів, значення якого завжди дорівнює 1. Надамо цьому входу номер 0. Тоді

$$s = \sum_{i=0}^{n} x_i \cdot w_i \qquad (3.11)$$

де $w_0 = -T$, $x_0 = 1$.

Очевидно, що відмінність формул (3.1) і (3.11) полягає лише у способі нумерації входів.

З усіх активаційних функцій, зображених на рисунку 3.2, одна виділяється особливо. Це гіперболічний тангенс, залежність якого симетрична відносно осі X і лежить в діапазоні [-1,1]. Забігаючи наперед, скажемо, що вибір області можливих значень виходів нейронів багато в чому залежить від конкретного типу НМ і є питанням реалізації, так як маніпуляції з нею впливають на різні показники ефективності мережі, часто не змінюючи загальної логіки її роботи.

3.3. Задачі штучних нейронних мереж

Задачі, які вирішують ШНМ, зводяться до апроксимації багатовимірнихфункцій, тобтопобудовивідображення $\boldsymbol{F}: \boldsymbol{x} \rightarrow \boldsymbol{y}$. В залежності

від вигляду активаційної функції формального нейрона дане відображення описує один з типів задач. У випадку застосування порогових активаційних функцій вихідні сигнали ШНМ мають дискретний характер, а задачі, що при цьому можуть бути розв'язні, називають задачами ***класифікації***. При застосуванні формальних нейронівіз сигмоїдальною або іншою неперервною активаційною функцією на ШНМ доцільно розв'язувати задачі ***регресії***. Такий поділ є до деякої міри умовним, оскільки задачі класифікації можуть використовувати також неперервні функції активації, значення яких трактуються як імовірності приналежності до відповідних класів.

3.4. Проблема класифікації та категоризації

Використання дискретної активаційної функції призводить до поділу простору вихідних сигналів на деяку множину класів. Завданням нейронної структури у цьому випадку єоднозначне встановлення відповідності між векторами вхідних сигналів та данними класів. Тип задач по такій відповідності називають задачами кластерізації або категоризації.

Поняття класу не завжди чітко задане і визначається з контексту розв'язуваних проблем. Але у більшості випадків під терміном ***клас*** розуміють сукупність об'єктів, які згруповані за деякими ознаками та правилами. Власне кажучи, завдання класифікації полягає в застосуванні до векторів вхідних сигналів встановлених правил виявлення сукупності ознак та прийняття рішення щодо віднесення кожного конкретного вектора до певного класудопустимий набір векторів вхідних сигналів бути однозначно розподіленим між заданими класами. Відповіддю на це питанняє вирішення проблеми лінійної роздільності.

У випадку, коли допускається динамічна зміна кількості класів та їх ознак, одержуємо задачу категоризації. Категорія, подібно до класу, є сукупність

об’єктів, що згруповані за деякими ознаками. Різниця тільки у тому, що категорія, у порівнянні з класом, не має статично визначеної сукупності ознак. Ці ознаки, як і кількість самих категорій, можуть змінюватися в процесі вирішення задачі категоризації. Формування ознак відбувається шляхом проб і помилок із подальшим відкиданням неістотних параметрів і підсиленням тих, які трапляються частіше.

Якщо задача категоризації дає можливість динамічної зміни множини ознак, то правила формування цих ознак залишаються незмінними і визначаються моделлю нейрона. Тому актуальним є питання, чи може бути та чи інша нейронна структура універсальним класифікатором. Іншими словами, чи може весь допустимий набір векторів вхідних сигналів бути однозначно розподіленим між заданими класами. Відповіддю на це питання є вирішення проблеми лінійної роздільності.

Задача класифікації зводиться до розподілу векторів вхідних сигналів, що належать до *n*-вимірного гіперпростору, на деяку кількість класів. Це відбувається шляхом поділу гіперпростору відповідною кількістю гіперплощин. У випадку *n*-входового персептрона

$$\sum_{i=1}^{n} w_{ik} x_i = T_k \qquad (3.12)$$

при $k = 1,2,...,m,$

Поняття класу не завжди чітко задане і визначається з контексту розв’язуваних проблем. Але у більшості випадків під терміном ***клас*** розуміють сукупність об’єктів, які згруповані за деякими ознаками та правилами. Власне кажучи, завдання класифікації полягає в застосуванні до векторів вхідних сигналів встановлених правил виявлення сукупності ознак та прийняття рішення щодо віднесення кожного конкретного вектора до певного класу.

У випадку, коли допускається динамічна зміна кількості класів та їх ознак, одержуємо задачу категоризації. Категорія, подібно до класу, є сукупність

об’єктів, що згруповані за деякими ознаками. Різниця тільки у тому, що категорія, у порівнянні з класом, не має статично визначеної сукупності ознак. Ці ознаки, як і кількість самих категорій, можуть змінюватися в процесі вирішення задачі категоризації. Формування ознак відбувається шляхом проб і помилок із подальшим відкиданням неістотних параметрів і підсиленням тих, які трапляються частіше.

Якщо задача категоризації дає можливість динамічної зміни множини ознак, то правила формування цих ознак залишаються незмінними і визначаються моделлю нейрона. Тому актуальним є питання, чи може бути та чи інша нейронна структура універсальним класифікатором. Іншими словами, чи може весь

де n — число входів, а m— число виходів персептрона.

3.5. Архітектура штучних нейронних мереж

На жаль, сьогодні не існує такого загальновизнаного означення, яке б задовольняло всіх. Причиною є той факт, що проблемою нейронних мереж займаються спеціалісти в різних галузях науки, і взаємному розумінню заважають методологічні та термінологічні бар’єри. Якщо розглядати штучну нейронну мережу як деяке середовище для обробки інформації, тоді її можна задати шляхом визначення елементів даного середовища та правил їх взаємодії. В цьому випадку говорять, що штучна нейронна мережа є структурою, яка складається з великої кількості процесорних елементів, кожен з яких має локальну пам’ять і може взаємодіяти з іншими процесорними елементами за допомогою комунікаційних каналів з метою передачі даних, що можуть бути інтерпретовані довільним чином. Процесорні елементи незалежно в часі обробляють локальні дані, що поступають до них через вхідні канали. Зміна параметрів алгоритмів такої обробки залежить тільки від характеристик даних.

Іншими словами, штучні нейронні мережі – це обчислювальні парадигми, які реалізують спрощені моделі біологічних нейронних мереж (БНМ). Під БНМ будемо розуміти локальні ансамблі нейронів, які об’єднані синаптичними зв’язками. Сукупність таких ансамблів формує мозок із його різноманітними функціональних можливостями. Сьогодні відома велика кількість нейронних структур та їх модифікацій, що орієнтовані на вирішення конкретного типу задач. Найбільш відомі типи таких структур показані на рисунку 3.5.

Розглянемо детальніше основні властивості ШНМ:

– локальна обробка інформації в штучному нейроні, який є базовою структурною одиницею мережі; – паралелізм, результатом якого є вирішення глобальної задачі шляхом представлення її у вигляді множини локальних задач, що тісно взаємодіють між собою;

– здатність до навчання, яке підвищує ефективність роботи мережі;

– здатність до розподіленого зберігання знань, які були одержані в ході навчання. ШНМ задають у вигляді направлених графів, вершинами яких є нейрони, а ребрами позначені міжнейронні зв’язки.

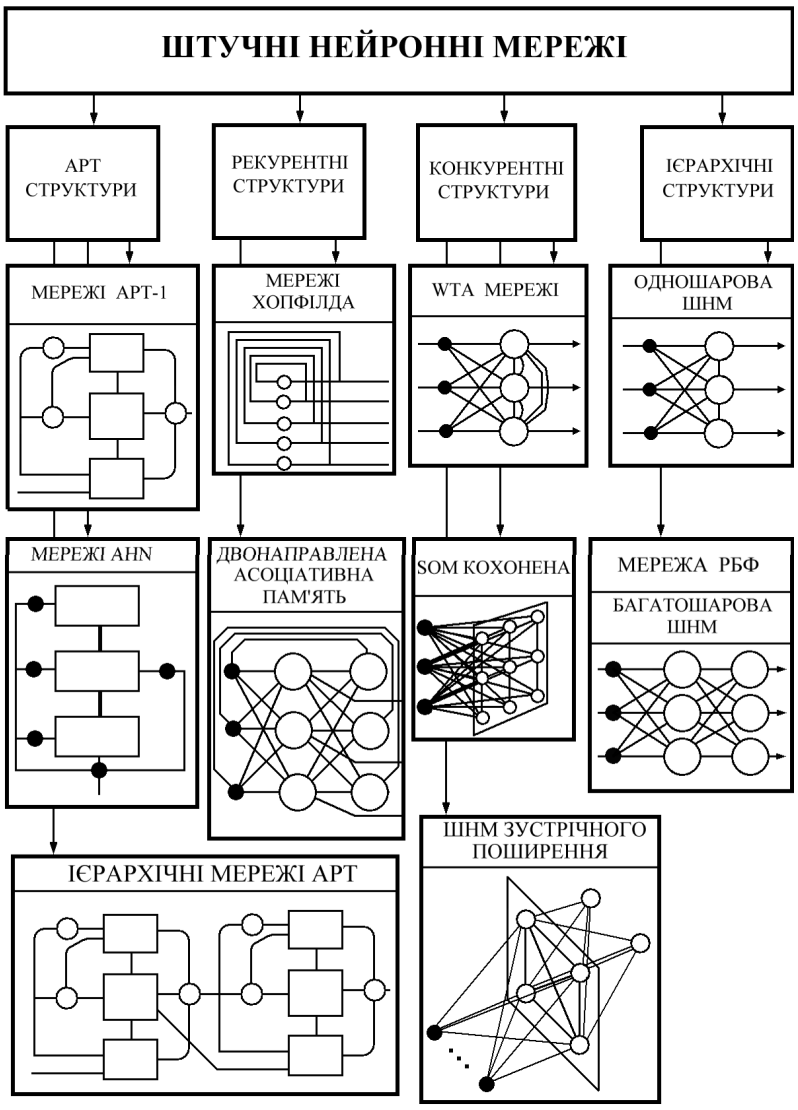


Рисунок 3.5 - Нейронні структури

Архітектури сучасних нейронних мереж найчастіше поділяють на три категорії

– мережі з повним набором міжнейронних зв’язків;

– мережі з фіксованим індексом оточення;

– мережі з пошаровою структурою. У ШНМ із повним набором міжнейронних зв’язків забезпечується можливість взаємодії кожного нейрона мережі з будь-яким іншим.

Першою штучною нейронною мережею, яку за сучасною термінологією відносять до одношарових, був персептрон Розенблатта. Оскільки персептрон розглядався автором як модель роботи мозку, то елементи його структури відповідали елементам простої рефлекторної мережі. S-елементи моделюють роботу сенсорних клітин, які збирають інформацію про навколишнє середовище та передають її до А-елементів, які насправді і є формальними нейронами з пороговою активаційною функцією. Для формування реакції персептрона призначені R-елементи, що мають фіксовані вхідні ваги для визначення внеску кожного А-елемента. Сучасним прототипом персептрона Розенблатта є одношарова нейронна мережа прямого поширення

Спроби застосувати одношарові нейронні мережі для розв'язування широкого кола задач наштовхнулися на ряд труднощів, пов'язаних з проблемою лінійної роздільності. Природним вирішенням цієї проблеми стало застосування багатошарових ШНМ, що нагадують багатошарові структури мозку. У нейронних мережах прямого поширення синаптичні зв'язки організовані таким чином, що кожний нейрон даного рівня ієрархії сприймає інформацію тільки від деякої непустої множини нейронів, які розташовані на більш низькому рівні. Назва мереж вказує на те, що у них існує виділений напрям поширення сигналів, які рухаються, починаючи з входу, через один або декілька прихованих шарів до вихідного шару. Легко помітити, що багатошарова нейронна мережа може бути одержана шляхом каскадного об'єднання одношарових мереж. У випадку лінійності активаційних функцій багатошарова нейронна мережа може бути зведена до еквівалентної одношарової.

Штучні нейронні мережі, які використовують радіальні базисні функції, називають РБФ-мережами. Вони є окремим випадком двошарових нейронних мереж прямого поширення, в яких прихований шар нейронів використовує радіальні базисні функції типу гаусової як активаційні. В просторі вхідних

векторів вибирають вектор, який називають центром, і відповідно до нього задають вагові коефіцієнти прихованого шару.

3.6. Нейронна мережа Кохонена

Модель нейронної мережі Кохонена характеризується структурою розподіленої пам'яті. Така структура дозволяє уникнути катастрофічної деградації у випадку відмови одного з нейронів.

Ефект підвищеної живучості досягається саме завдяки розподіленій пам'яті, дія якої проявляється за рахунок того, що за класифікацію вхідного вектора відповідає не один нейрон, а кластер нейронів.

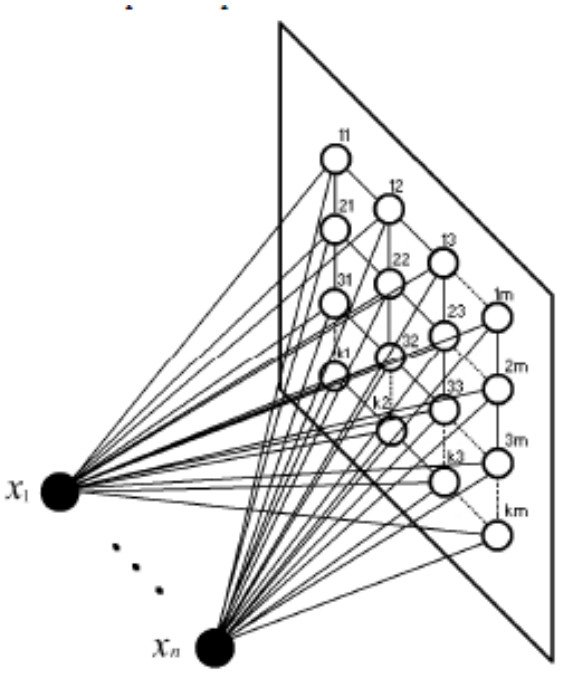


Рисунок 3.6 - нейронна мережа Кохонена

На рисунку 3.6 показана структура нейронної мережі Кохонена. У цій структурі кожний елемент вхідного вектора сигналів

$X = (x_1, x_2, \dots, x_n)$ поступає на вхід кожного елемента двовимірної матриці нейронів. Тому множина вагових коефіцієнтів структурована у вигляді матриці

$$W = \begin{pmatrix} W^{11} & W^{12} & \dots & W^{1m} \\ W^{21} & W^{22} & \dots & W^{2m} \\ . & . & \dots & . \\ W^{k1} & W^{k2} & \dots & W^{km} \end{pmatrix}, \qquad (3.13)$$

елементами якої є вектори вагових коефіцієнтів $w^{ij} = (w_1^{ij}, w_2^{ij}, \dots, w_n^{ij})$, що масштабують відповідні елементи вхідного вектора. Всім ваговим коефіцієнтам на початку роботи мережі задають випадкові значення. Наступний етап полягає в обчисленні відстані

$$d_{ij} = \sum_{p=1}^{n} (x_p(t) - w_p^{ij}(t))^2 \qquad (3.14)$$

між вектором вхідного сигналу та кожним з нейронів мережі.

Нейрон із координатами (i^*, j^*), для якого ця відстань є мінімальною, вибирається центром кластера. Відносно нього проводиться підстроювання вагових коефіцієнтів всіх нейронів, які входять до початкового кластера, за формулою:

$$W^{ij}(t+1) = W^{ij}(t) + \eta r(l^{ij})(X(t) - W^{ij}(t)) \qquad (3.15)$$

де 0 <η < 1 — коефіцієнт навчання, який змінюється обернено пропорційно до часу навчання;

$r(l^{ij}) = \exp\left(\frac{-(l^{ij})^2}{\sigma^2}\right)$ одна з можливих функцій сусідства, що задає зміну вагових коефіцієнтів у залежності від відстані

$$l^{ij} = \left|\sqrt{(j^* - j)^2 + (i^* - i)^2}\right| \qquad (3.16)$$

нейрона (i,j) до центра кластера.

Конфігурацію початкового кластера найчастіше формують як просту геометричну фігуру (квадрат, прямокутник, круг), яка займає від 0,5 до 0,75 розміру нейронної мережі.

3.7. Нейронні мережі зустрічного поширення

Нейронні мережі зустрічного поширення — це ще одна відома парадигма з використанням механізму WTA. Узагальнена структура такої мережі показана на рис.3.7

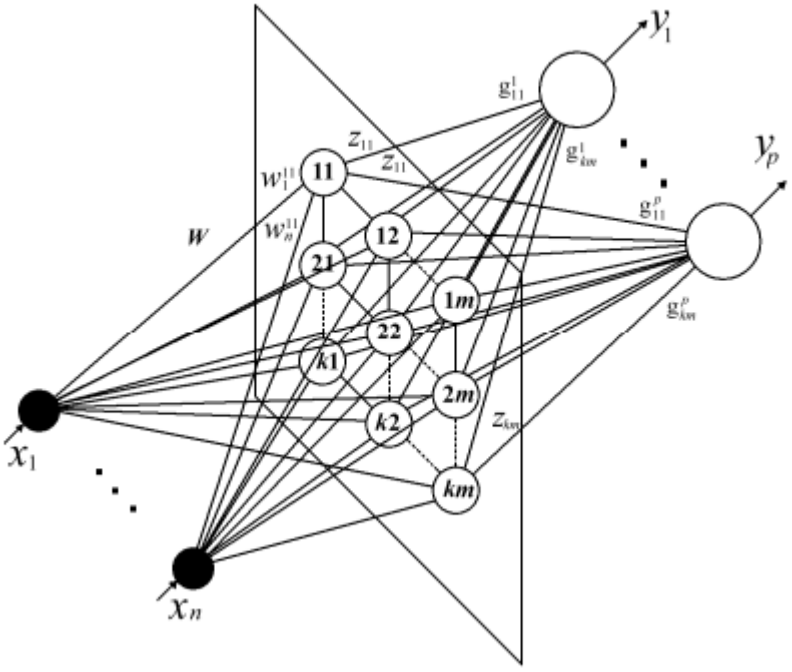


Рисунок 3.7 - Нейронна мережа зустрічного поширення.

Вона складається з двох шарів нейронів різного типу. Перший шар є двовимірною матрицею Кохонена, а другий шар називають шаром Гроссберга.

Об’єднання цих двох шарів надає структурі додаткові властивості, які відсутні у кожного з них окремо. Вхідний вектор сигналів X поступає на кожний з нейронів шару Кохонена. Масштабування відбувається за допомогою матриці вагових коефіцієнтів, що містить вектори вагових коефіцієнтів. Відомі два методи функціонування шару Кохонена, що отримали назви “метод акредитації” та “метод інтерполяції”.

3.8. Рекурентні структури

Структури мереж прямого поширення не мають зворотних зв’язків, тобто сигнали в них поширюються від шару до шару тільки в одному напрямку, формуючи статичний стан кожного нейрона мережі. Дещо складнішим є принцип функціонування конкурентних структур. В мережах такого типу затухаючий ітераційний процес відбувається в межах одного шару нейронів, а ітераційна формула завжди має властивість зниження рівня вихідного сигналу, що викликає затухання слабких вихідних сигналів до рівня, нижчого від порога чутливості. Таким чином реалізується стратегія “переможець забирає все”, що зупиняє ітераційний процес у випадку перемоги одного нейрона або кластера нейронів. Принципово новою є ситуація, коли структура нейронної мережі допускає зворотні зв’язки, тобто, коли обчислення в нейроні даного шару відбуваються з урахуванням попереднього стану цього ж шару. Мережі такого типу називають рекурентними та говорять про динамічний характер їх функціонування. Динаміка зміни станів сукупності взаємопов’язаних об’єктів традиційно вимагає визначення правила взаємної координації дій цих об’єктів у часі та просторі. Коли встановлюється послідовність спрацьовування нейронів одного шару, то таку нейронну мережу називають синхронною. У випадку, коли час спрацьовування кожного з нейронів не регламентується, мережу називають асинхронною. Синхронні мережі можуть бути паралельними, послідовними та

паралельно-послідовними. Для паралельних мереж визначальною є властивість одночасного спрацьовування всіх нейронів одного шару, для послідовних мереж задають послідовність спрацьовування нейронів одного шару, а для паралельно-послідовних мереж визначається послідовність спрацьовування кластерів нейронів. Про асинхронні мережі говорять, що вони є паралельними, маючи на увазі те, що такі мережі допускають одночасне функціонування всіх нейронів.

3.9. Нейронна мережа Хопфілда

Структура рекурентної нейронної мережі, яка показана на рис.3.8. Вона складається з одного шару нейронів, виходи яких через спеціальні пристрої з'єднані зі входами всіх нейронів цього ж шару, крім тих зв'язків, що з'єднують вихід нейрона з його власним входом.

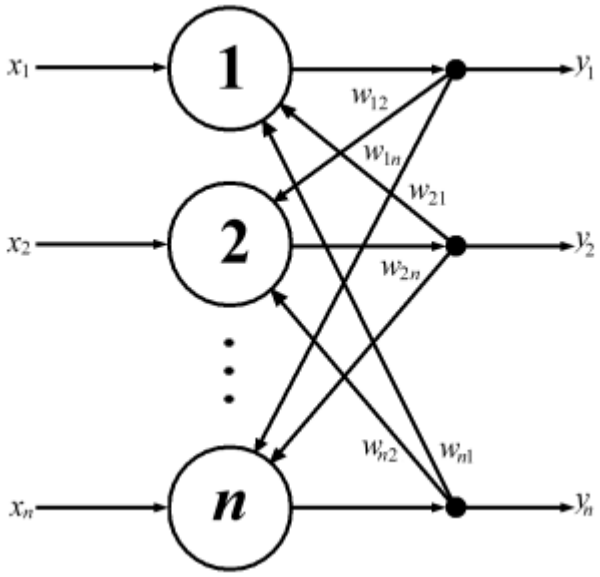


Рисунок 3.8 - Нейронна мережа Хопфілда

Формування аргументу активаційної функції v_j довільного нейрона j відбувається за формулою

$$v_j = \sum_{\substack{i=1 \\ i \neq j}}^{n} w_{ij} y_i + x_j$$

В залежності від вигляду активаційної функції розрізняють дискретну та аналогову моделі мереж Хопфілда.

Бінарна модель базується на використанні порогової активаційної функції, яку задають виразом:

$$y_j = f(v_j) = \{ \begin{matrix} 1 \text{ якщо } v_j > T_j \\ 0 \text{якщо } v_j < T_j \\ y_j \text{якщо } v_j = T_j \end{matrix} \quad (3.17)$$

Такий вигляд активаційної функції був запропонований Хопфілдом. У сучасній літературі частіше зустрічається порогова активаційна функція зі зміною знака

$$y_j = f(v_j) = \{ \begin{matrix} 1 \text{ якщо } v_j > T_j \\ -1 \text{якщо } v_j < T_j \\ y_j \text{якщо } v_j = T_j \end{matrix}$$

через T_j позначається величина порога чутливості довільного нейрона *j* мережі Хопфілда.

Якщо для виходу j у кожного нейрона поставити у відповідність двійковий розряд, то поточний стан мережі Хопфілда може бути виражений числом у двійковій системі числення, а множина допустимих сусідніх станів є множиною чисел з одиничною відстанню Хеммінга. Якщо стани нейронної мережі позначити точками з відповідними координатами у просторі та з'єднати дугами ті точки, між якими допустимий перехід, одержимо граф переходів мережі. Для мережі, що складається з трьох нейронів, такий граф має вигляд куба. Мережі з

кількістю нейронів більше трьох утворюють гіперкуби вищих порядків, що не мають графічної інтерпретації. У загальному випадку множина сусідніх вершин N для довільної вершини a визначається з виразу:

$$N(a) = \{a\ XOR\ 2^{i-1}\}\quad {}_{i=1}^{n}, \tag{3.18}$$

де $a = 0,1,,\ldots, 2^n - 1$

Функціонування мережі Хопфілда полягає в пересуванні вздовж ребер переходів, поки мережа не досягне стійкого стану. Мережа Хопфілда має стійкий стан у випадку, коли матриця вагових коефіцієнтів $\boldsymbol{W}$ симетрична і має нулі на головній діагоналі, тобто

$$w_{ij} = w_{ji}, w_{ii} = 0 \tag{3.19}$$

Для пояснення факту стійкості мережі за згаданих умов введемо функцію енергії довільної пари нейронів:

$$e_{ij} = -w_{ij}y_iy_j - x_jy_j + T_jy_j \tag{3.20}$$

Виходячи з умови симетричності (3.19) матриці вагових коефіцієнтів, загальну енергію E НМ визначають за функцією Ляпунова

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}y_iy_j - \sum_{j=1}^{n} x_jy_j + \sum_{j=1}^{n} T_jy_j \tag{3.21}$$

Стан мережі перед спрацьовуванням нейрона k:

$$\mathrm{C} = -\frac{1}{2}\sum_{\substack{i\neq k\\ j\neq k}} w_{ij}y_iy_j + \sum_{j\neq k}^{n} x_jy_j - \sum_{j\neq k}^{n} T_jy_j \tag{3.22}$$

Тоді загальна енергія мережі

$$E = \mathrm{C} - \frac{1}{2}\sum_{i=1}^{n} w_{ik} y_i y_k - \frac{1}{2}\sum_{i=1}^{n} w_{ki} y_i y_k + x_k y_k - T_k y_k \tag{3.23}$$

Відповідно до (3.19)

$$E = \mathrm{C} - \sum_{i=1}^{n} w_{ik} y_i y_k + x_k y_k - T_k y_k \tag{3.24}$$

Загальна енеогія перед черговим спрацьовуванням k-нейрона $E = \mathrm{C} - y_k(v_k - T_k)$, а після $E' = \mathrm{C} - y'_k(v_k - T_k)$.

Зміна енергії:

$$\Delta E = E' - E = -\Delta y_k (v_k - T_k) \tag{3.25}$$

Можливі варіанти:

1) $v_k > T_k$

$$\Delta y_k = y'_k - y_k = \begin{cases} 1, \text{ якщо } y'_k = 1\,, y_k = 0 \\ 0 \text{ якщо } y'_k = 1\,, y_k = 1 \end{cases}$$

$$\Delta E \le 0$$

2) $v_k < T_k$

$$\Delta y_k = y'_k - y_k = \begin{cases} -1, \text{ якщо } y'_k = 0\,, y_k = 1 \\ 0 \text{ якщо } y'_k = 0\,, y_k = 0 \end{cases}$$

$$\Delta E \le 0$$

3) $v_k = T_k$

$$\Delta E = 0$$

Отже, загальна енергія мережі Хопфілда за умов (3.19) завжди або зменшується, або залишається незмінною. Стійкий стан мережі відповідає мінімуму її енергії.

Асоціативна пам'ять

Поверхня енергії в просторі станів мережі має складну форму, яка характеризується великою кількістю локальних мінімумів. У випадку застосування мережі Хопфілда в ролі запам'ятовуючого середовища стійкі стани, що відповідають локальним мінімумам енергії, інтерпретують як образи пам'яті. Під час переходу мережі від одного стану до такого, що характеризується меншим рівнем енергії, завжди відбувається пошук сусіднього стану за формулою (3.18). Тому мережа завжди знаходить локальний енергетичний мінімум, який розміщується на мінімальній відстані Хеммінга від початкового стану. Нехай вектор вхідних сигналів X відповідає ідеальному образу пам'яті, тобто локальному мінімуму енергії мережі. Тоді надходження на вхід мережі вектора Xi приводить до еволюції мережі у напрямку локального мінімуму, що відповідає відновленню ідеального образу по його неповній копії. Така пам'ять відрізняється від тієї, що використовується в традиційних обчислювальних структурах. Вона більше нагадує людську пам'ять, для якої характерний асоціативний, а не адресний принцип запам'ятовування інформації. Асоціативна пам'ять є значно стійкішою щодо різного роду помилок та шумів за рахунок того, що має здатність відновлення ушкодженого образу, на відміну від адресної пам'яті, у якій помилка адресації призводить до неправильного зчитування значних фрагментів. Недоліком асоціативної пам'яті, яка реалізується на мережі Хопфілда, є незначний її об'єм, пропорційний кількості нейронів. У цьому випадку об'єм адресної пам'яті пропорційний 2^n , де n — кількість двійкових розрядів.

Двонаправлена асоціативна пам’ять

Основною властивістю асоціативної пам’яті, що реалізована на мережі Хопфілда, є здатність відтворювати образи шляхом їх асоціації з частково помилковим запитом. Таку мережу ще називають автоасоціативною. Запропоновано більш досконалу версію асоціативної пам’яті — двонаправлену асоціативну пам’ять (ДАП), що отримала назву гетероасоціативної. Ця структура дозволяє додатково встановлювати асоціативні зв’язки між різними образами. На рис.3.9 показана структурна схема такої мережі.

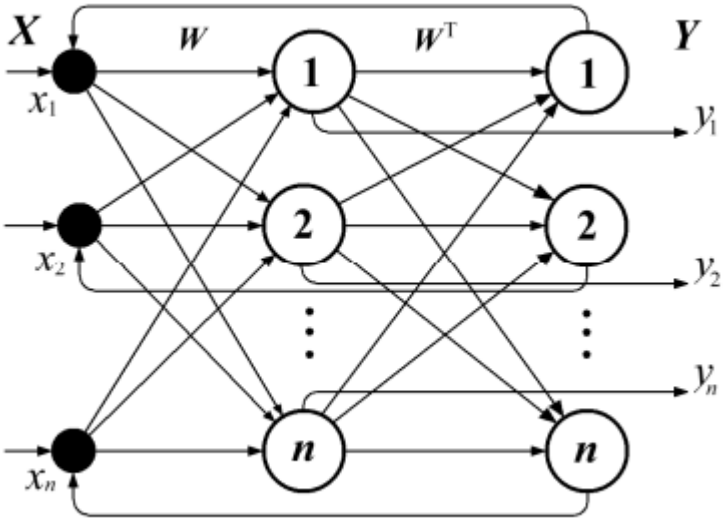


Рисунок 3.9 - Двонаправлена асоціативна пам’ять.

Вектор вхідних сигналів X, відповідно промасштабований елементами матриці вагових коефіцієнтів W, надходить на вхід першого шару нейронів. Результатом спрацьовування цього шару є вихідний вектор сигналів Y, який надходить на вхід другого шару з транспонованою матрицею вагових коефіцієнтів W^T . До моменту його спрацьовування зовнішній вхідний вектор X знімається і заміщується тим, який є результатом роботи другого шару. Далі починається новий цикл рекурентних обчислень, і він продовжується до того

часу, поки мережа не досягне стійкого стану, в якому вектори X та Y залишаються незмінними. Активаційна функція першого шару нейронів визначається за формулою :

$$y_j = f(v_j) = f(\sum_{i=1}^{n} w_{ij} x_i)$$

Нейрони другого шару формують вхідний вектор сигналів за формулою

$$x_i = f(v_i^t) = f(\sum_{j=1}^{n} w_{ji}^t y_j)$$

Де w_{ji}^t – ваговий коефіцієнт, що є елементом транспонованої матриці W^T

Найчастіше в ролі активаційної функції використовують традиційну сигмоїдальну функцію з великими значеннями коефіцієнта α , що наближає її до порогової функції. За принципами роботи розрізняють синхронні та асинхронні ДАП. У синхронному режимі відбувається одночасне спрацьовування всіх нейронів одного шару під дією зовнішнього синхронізуючого сигналу. Найчастіше у цьому випадку використовують порогову активаційну функцію. Для досягнення більшої подібності з біологічним прототипом застосовують асинхронний режим функціонування ДАП з сигмоїдальною активаційною функцією. В цьому режимі кожний нейрон обробляє неперервні сигнали і може бути змодельований на операційному підсилювачі з нелінійним зворотним зв’язком. ДАП — більш стійкі структури, ніж мережі Хопфілда. Для них притаманним є прагнення до пошуку локального енергетичного мінімуму у випадку, коли матриця вагових коефіцієнтів W має довільну форму. Спільним недоліком для цих мереж є незначна ємність пам’яті -кількість образів, що

можуть запам'ятовуватись мережами такого типу, пропорційна $\left(\frac{n}{2log_2 n}\right)$, де n - кількість нейронів. Існує ряд підходів, які дозволяють покращити розпізнавальні властивості ДАП і, відповідно, збільшити кількість образів, що можуть викликати адекватні асоціації. До таких підходів слід віднести застосування нейронів з різними порогами спрацьовування активаційних функцій. Мережі, що мають такі властивості, називають адаптивними ДАП. Активаційні функції нейронів адаптивної ДАП визначаються за формулою (3.17) у випадку синхронного режиму функціонування. При цьому для кожного нейрона j встановлюється свій поріг спрацьовування T_j . Асинхронний режим функціонування використовує неперервну сигмоїдальну активаційну функцію, амплітуда якої визначається параметром β_j окремо для кожного нейрона j.

$$y_j = f_j(v_j) = \frac{\beta_j}{1 + e^{-\alpha v_j}}$$

Алгоритм настройки порогів підвищує чутливість мережі у вибраному діапазоні вхідних сигналів. Ще одним підходом такого типу є застосування конкурентних зв'язків на кожному шарі нейронів, що приводить до формування нової матриці вагових коефіцієнтів. Як і в попередньому випадку, підвищення чутливості відбувається за рахунок зниження допустимого діапазону вхідних образів.

4. ГЕНЕТИЧНІ АЛГОРИТМИ

Останні революції в молекулярній генетиці показали, що модульна організація генів дуже важлива для еволюційної складності. Еволюційна концепція генетичного алгоритму була введена Джоном Холландом у 1975 році у Мічиганському університеті. Першим етапом генетичного алгоритму є початкова генерація популяції випадковим чином, заснована на логіці ймовірності, і кожен індивід у випадково генерованій популяції представлений хромосомами у біології. Метод випадкової генерації може відрізнятися від дослідники до дослідника. Хромосоми мають багато сегментів, а кількість n рядків залежить від складності даної задачі [1]. Ці хромосоми і використовуються для схрещення. Таким чином, породжені діти матимуть властивості обох батьків. Оператор мутації може застосовуватися до породженої хромосоми для запобігання стану стагнації. І після буде розраховано придатність створеного індивіду, який надалі буде використовуватися для породження ще кращих індивідів. Цей процес необхідно продовжувати до досягнення оптимального рішення.

4.1. Генерація популяції

Першим кроком Генетичного Алгоритму є ініціалізація популяції з n кількістю рядків згенерованих рандомним чином. В генетичних Алгоритмах, сукупність рядків називається хромосомами або генотипом геному, який кодує рішення-кандидати, які називаються індивідами, істотами або фенотипами для задачі оптимізації, що буде розвиватися до кращих рішень як показано на рис. 4.1.

Традиційно, рішення презентується у вигляді бінарного рядка з 0 і 1, але інші кодування також можливе. Десяткові цифри, абетка, арифметичні значення,

ASCII значення також можуть використовувати. Розмір популяції залежить від природи проблеми[2], але зазвичай містить сотню чи тисячу можливих індивідів. Традиційно, популяція генерується рандомно.

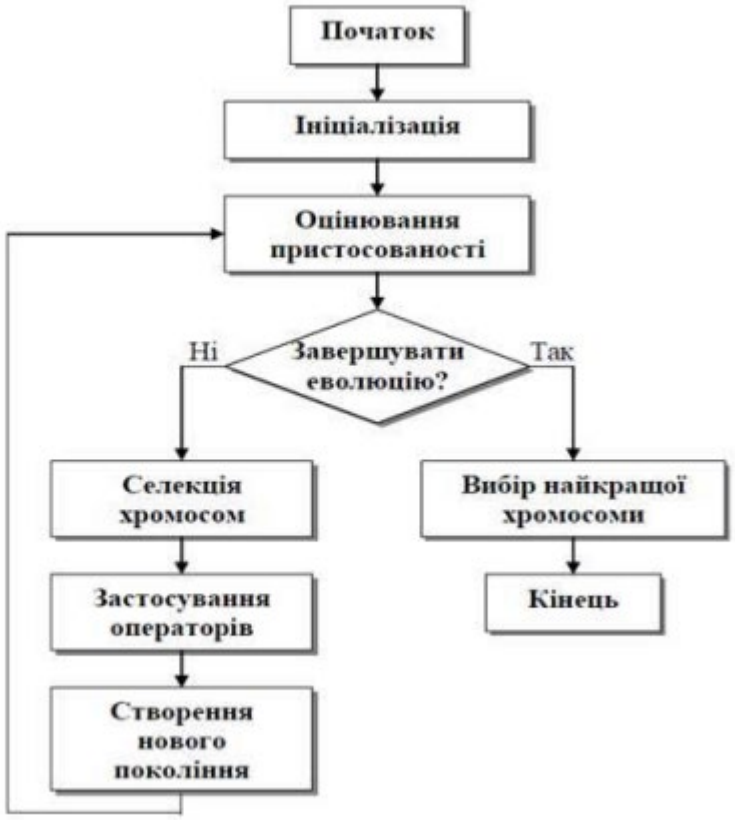


Рисунок 1 – Загальна схема генетичного алгоритму

4.2. Схрещення

Репродукція або схрещення є другим кроком Генетичного Алгоритму, де властивості батьків змішуються для формування кращого рішення. Кількість схрещень залежить від кількості батьків. Відсоток схрещення може змінюватися від додатку до додатку. Багато технік схрещень існує для організмів, які використовують різну структуру хромосом передати власні особливості.

Варіації технік схрещення, які зазвичай використовуються:

- One Point Crossover
- Two Point Crossover
- Multi Point Crossover
- Uniform Crossover
- Half Uniform Crossover
- Aritmetic Crossover

1-Point Crossover

Це один із простих методів кросовера, який використовується для випадкових GA. Цей кросовер використовує одноточкову фрагментацію батьків, а потім об'єднує батьків у точці перетину, щоб створити потомство. Метод спершу вибирає двох батьків, що будуть використовуватися для схрещення і потім випадковим чином обирається будь-яка точка кросоверу p_i ($i = 0$ to $n - 1$). Двоє нащадків утворюються шляхом об'єднання батьків у точці схрещування. Нижче наведено приклад, який виконує одноточковий кросовер і створює двох нащадків. На рисунку 4.2, точка між 4 і 5 геном вибрана як точка схрещування

Parent 1:	1 0 1 0 | 1 0 0 1 0
Parent 2:	1 0 1 1 | 1 0 1 1 0
Offspring 1:	1 0 1 0 | 1 0 1 1 0
Offspring 2:	1 0 1 1 | 1 0 0 1 0

Рисунок 2 – 1-Point crossover

K-Point Crossover

Використовується випадкова точка перетину, щоб об’єднати двох батьків так само, як і для 1-точкового кросоверу. Щоб забезпечити чудове поєднання батьків, він вибирає більше однієї точки схрещування для створення потомства. K-Point Crossover спершу обирає двох батьків, які будуть використовуватися для схрещення, а потім випадковим чином вибирає K точок перетину від P_1 і до P_{k-1}. Обидва нащадки утворюються шляхом об’єднання батьків у точці схрещування. На рисунку 4.3 точки між 2 і 3 , 4 і 5 та 7 і 8 геном обираються як точки схрещування

Parent 1:	1 0 | 1 0 | 1 0 0 | 1 0
Parent 2:	1 1 | 0 0 | 1 0 1 | 1 0
Offspring 1:	1 0 | 0 0 | 1 0 0 | 1 0
Offspring 2:	1 1 | 1 0 | 1 0 1 | 1 0

Рисунок 3 – K-Point crossover

Shuffle Crossover

Shuffle Crossover допомогає у створені нащадків, які мають незалежність від точки схрещування батьків. Він використовує 1-точковий кросовер як додаток до перемішування Shuffle Crossover вибирає двох батьків для схрещування. Спочатку випадковим чином перемішуються гени обох батьків, але однаково. Потім застосовується техніка 1-точкового схрещення, знову випадковим чином обираючи точку схрещення. А потім схрещуємо батьків для створення нащадків. Після виконання 1-точкового схрещення гени потомства

потім перемішуються так само. Приклад показано на рисунку 4.4

Select Shuffle Points
Parent 1: 1 1 1 0 1 0 0 1 0
Parent 2: 1 0 0 0 1 0 1 1 0
Shuffle genes as Shuffle Points
Parent 1: 0 1 0 1 1 0 1 1 0
Parent 2: 0 0 1 1 1 0 0 1 0

Select 1- Point Crossover Point
Parent 1: 0 1 0 1 | 1 0 1 1 0
Parent 2: 0 0 1 1 | 1 0 0 1 0
Perform 1-Point Crossover Point
Offspring 1: 0 1 0 1 | 1 0 0 1 0
Offspring 2: 0 0 1 1 | 1 0 1 1 0
Select unshuffled points same as shuffled points
Offspring 1: 0 1 0 1 1 0 0 1 0
Offspring 2: 0 0 1 1 1 0 1 1 0
Unshuffled the genes in Offspring
Offspring 1: 1 1 0 0 1 0 0 1 0
Offspring 2: 1 0 1 0 1 0 1 1 0

Рисунок 4 – Shuffle crossover

Reduced Surrogate Crossover

Зменшений сурогатний кросовер мінімізує небажані операції схрещування у випадку, якщо батьки мають однакові гени. У цих випадках зменшений сурогатний кросовер спочатку перевіряє наявність окремих генів у батьків. Він створює список усіх можливих точок кросинговеру, де гени обох батьків відрізняються. Зменшений сурогатний кросовер мінімізує небажані операції схрещування у випадку, якщо батьки мають однакові гени. У цих випадках

зменшений сурогатний кросовер спочатку перевіряє наявність окремих генів у батьків. Він створює список усіх можливих точок кросинговеру, де гени обох батьків відрізняються.

Uniform Crossover

Уніфікований кросовер забезпечує рівномірність поєднання бітів обох батьків. Він виконує цю операцію заміни бітів у батьків для включення в потомство, вибираючи однорідне випадкове дійсне число u (від 0 до 1). Уніфікований кросовер вибирає двох батьків для кросовера. Він створює двох дітей-нащадків з n генів, відібраних у обох батьків рівномірно. Випадкове дійсне число визначає, чи вибере перша дитина i-і гени від першого чи другого батька [10].

Parent 1: 1 1 1 0 1 0 0 1 0
Parent 2: 1 0 0 0 1 0 1 1 0
Offspring 1: 1 1 0 0 1 0 1 1 0
Offspring 2: 1 0 1 0 1 0 0 1 0

Рисунок 5 – Uniform crossover

Average Crossover (AX)

Середній кросовер – це метод схоещування на основі вартості. Він використовує двох батьків для виконання кросовера і створює лише одне потомство [4]. Середній кросовер створює одне потомство, взявши середнє з двох батьків. Він вибирає двох батьків як X і Y і створює дитину Z наступним чином: кожен ген у дитини береться шляхом усереднення генів обох батьків.

Parent 1: 5 3 3 2 3 9 7 6 5
Parent 2: 5 4 7 6 5 2 6 1 3
Offspring 1: 5 3 5 4 4 5 6 3 4

Рисунок 6 – Average crossover

Discrete Crossover (DC)

Дискретний кросовер використовує випадкове дійсне число для створення однієї дитини з двох батьків. На відміну від однорідного кросовера, у дискретному кросовері генерується лише один дочірній елемент. Він вибирає двох батьків як X і Y і створює дитину Z таким чином, що вибирає гени обох батьків рівномірно. Випадкове дійсне число визначає, у якого з батьків взяти гени для дитини[4].

Parent 1: 1 1 1 0 1 0 0 1 0
Parent 2: 1 0 0 0 1 0 1 1 0
Offspring 1: 1 1 0 0 1 0 1 1 0

Рисунок 7 – Discrete crossover

Flat Crossover (FC)

Flat Crossover використовує випадкове дійсне число, щоб створити одну дитину з двох батьків.

Так само, як і при дискретному кросинговері, він вибирає гени від батьків на основі однорідного випадкового дійсного числа. Але вибране випадкове дійсне число має бути підмножиною набору, що має мінімум і максимум генів обох батьків. Він вибирає двох батьків як X і Y і створює дитину Z так, що вибирає випадкове дійсне число, яке є або мінімальним, або максимальним з генів обох батьків, а потім призначає це реальне число в дочірньому гені.

Heuristic Crossover/Intermediate Crossover (HC/IC)

Евристичний кросовер створює одну дитину від двох батьків. Він використовує $\alpha\, \varepsilon\, <\, 0,1\, >$ за умови, що $x_i <= y_i$. Для кожного гена у дитини підбирають рівномірне випадкове дійсне число α. А дочірній ген розраховується з вищенаведеного рівняння

$$x_i(t+1) = x_i(t) + \alpha(y_i(t) - x_i(t))$$

Statistics-Based Adaptive Non-Uniform Crossover (SANUX)

Він заснований на концепціях внутрішнього атрибута та зовнішньої тенденції оцінки алеля для локусу гена. В оптимальному розв'язанні (закодованому у двійковому рядку) даної задачі для локусу гена, якщо його алель дорівнює 1, він називається 1-внутрішнім, якщо його алель дорівнює 0, він називається 0-внутрішнім, інакше, якщо його алель 0 або 1 його називають нейтральним. Під час роботи GA для локусу гена, якщо частота 1 в його алелях більше, популяція має тенденцію збільшуватися з часом (покоління), це називається 1-схилим; якщо частота 1 має тенденцію до зменшення, її називають 0-похилою; інакше його називають ненахиленим[5].

Зазвичай і сподіваємося, що в міру розвитку GA локуси генів, які є внутрішніми 1, виявляться 1-схилими. SANUX використовує цю інформацію про конвергенцію як інформацію зворотного зв'язку, щоб спрямувати кросовер, регулюючи ймовірність заміни кожного локусу.

Тепер під час еволюції GA, після генерації нової популяції, розраховується розподіл $f_1(i,t)$ від кожного локусу по сукупності. Потім виконується операція SANUX. Після застосування SANUX маска генерується покроково шляхом підкидання монети зі зміщенням, щоб генерувати «1» з ймовірністю $p_s(i,t)$. Нарешті, згенерована маска використовується для наведення кросовера так само, як він направляє традиційний однорідний кросовер.

1's Frq. in loci:	0.4	0.2	0.6	0.9	0.9	0.2
Calculating:	|	|	|	|	|	|
Swapping prob:	0.4	0.2	0.4	0.1	0.1	0.2
biased flipping:	|	|	|	|	|	|
Created mask:	1	0	1	0	0	0
Applying mask:	|		|			
Parent P1:	0	1	0	1	1	1
Parent P2:	1	1	1	1	1	0
Swapping:	|		|			
Child C1:	1	1	1	1	1	1
Child C2:	0	1	0	1	1	0

Рисунок 8 – SANUX

Random Respectful Crossover (RRC)

Він вибирає двох батьків для схрещування і потомство генерується на основі вектора подібності батьків. Спочатку він створює вектор подібності $S_{ab} = (S_{1ab}, \ldots, S_{nab})$ таким чином, що якщо обидва гени батьків мають однакові значення, то вектор подібності містить значення батьківського, інакше вектор подібності містить нульове значення цього гена[4].

Після створення вектора подібності S створюється два дочірніх за значеннями вектора подібності. Якщо вектор подібності містить 1, то ген обох дітей встановлюється на 1, а якщо він містить 9, то гени обох дітей встановлюється в 0. Крім цього, якщо вектор вектор подібності містить нульове значення для будь-якого гена, тоді дочірній ген вибирається за допомогою рівномірно рандомного числа. Якщо це число < 0.5, то зверігається 1, інакше зберігається 0.

```
Parent 1:      1 1 1 0 1 0 0 1 0
Parent 2:      1 0 0 0 1 0 1 1 0
Offspring 1:   1 1 0 0 1 0 1 1 0
Offspring 1:   1 0 0 0 1 0 0 1 0
```

Рисунок 9 – Random respectful crossover

Цей алгоритм дублює гени батьків у нащадку в кожній позиції, де б вони не були ідентичними.

Masked Crossover (MX)

Оператор MX використовує вектор маски, щоб визначити, які біти якого з батьківських елементів успадковуються нащадком. Перший крок - це дублювання бітів батьків. Біти першого батька копіюються до першого нащадка і, відовідно, другого батька до другого нащадка. На другому кроці нащадки обмінюються між собою бітами в тих позиціях, де вектори маски батьківського елемента дорівнюють 1, що вказує на домінування цього батька в цій позиції, а вектори маски іншого батька дорівнюють 0.

Вектори маски ініціюються в P(0) випадковим чином. Під час кожної ітерації GA вектори маски успадковуються кожним нащадком від свого батька. Потім модифікуються вектори маски потомства, а також батьків. Процес модифікації заснований на порівняння придатності нащадків і батьків. Якщо створено гарне потомство, маски батьків не потрібно змінювати, і маски потомства можуть бути дуже схожими на маски потоства можуть бути дуже схожими на маски батьків. У ситуації, коли було створено погане потомтво, маски батьків, а також нащадків потребують модифікації.

1-Bit Adaptation Crossover (1BX)

У методі 1 BX останній біт вектора рішення зарезервовано для коду одного з двох застосованих операторів кросовера. Припускаючи, що “0” відповідає оператору Uniform Crossover (UX), а “1” відповідає оператору 2-Point Crossover (2-PX), вибір одного з них здійснюється за правилом: якщо останній біт батьків вимкнути те саме значення, а потім виберіть оператор, зазначений цим бітом. В іншому випадку вибирає оператор шляхом виділення методом жеребкування, тобто обирає однорідне випадкове дійсне число від 0 до 1. Якщо це значення < 0.5, тоді виконується рівномірний кросовер, інакше використовується 2-Point Crossover. Застосування описаної схеми кросовера поєднує вибір оператора з вектором рішення. Причому цей вибір здійснюється окремо для кожної батьківської пари; тому ця схема називається локальною адаптацією. Також була представлена глобальна адаптаційна версія, але, як підкреслив автор, її застосування дало значно гірші результати.

Multivariate Crossover (MC)

MC ділить весь батьківський рядок на q підрядки. Кросовер виконується на основі випадкового значення, вибраного для кожного підрядка. Якщо це значення $< P_c$, то виконується кросовер, інші тільки батьківські гени копіюються в дітей-нащадків. Він виконує стандартний 1-Point Crossover для підрядка, коли умова виконується[4].

Найбільш принципова відмінність між оператором MC та іншими операторами, що використовуються рекомбінацію змінної до змінної, полягає в тому, що відповідь на питання “чи схрещувати” перевіряється в методі MC окремо для кожного підрядка. Що стосується інших операторів, то відповідь на це питання стосується батьківського вектора в цілому.

Homologous Crossover (HX)

Оператор HX заснований на стандартному операторі схрещування k-Point. Запроваджена модифікація спирається на той факт, що лише рядки бітів, які мають принаймні певну довжину або допустимий ступінь подібності, можуть брати участь у кросовері. Визначення ступеня подібності здійснюється на основі оператора XOR. Ця стратегія спрямована на передачу рядків із заданими параметрами наступному поколінню. У HX значення o і r визначається апріорі як постійні або динамічно змінені під час виконання GA.

Count Preserving Crossover (CPC)

Оператор CPC виконує своє завдання припкскаючи, що кількість бітів, що дорівнює “1” у кожній хромосомі в початковій популяції $P(0)$, однакова. CPC може гарантувати збереження постійної кількості бітів, що дорівнює “1” завдяки застосуванню двох списків із зазначенням відмінностей між батьками. Списо L_{up} містить позиції тих бітів, на яких є відмінності між батьками, але перший батько в даній позиції утримує біт, рівний “0”, а другий дорівнює “1”. Процес створення потомства, який використовує ці списки заснований на обміні бітами між нащадками в тих позиціях, які вказуються наступними парами елементів зі списків L_{up} і L_{down}. Кількість елементів у L_{up} і L_{down} однакова, що є прямим результатом припущення, що кількість бітів, що дорівнює “1”, є постійною для всіх хромосом у $P(0)$.

Elitist Crossover (EX)

У стандартному генетичному алгоритмі процесу відбору завжди передує процес кросинговеру. У методі EX обидва процеси інтегровані. Під час першого кроку вся популяція випадковим чином перемішується. Потім з кожної наступної пари батьківських векторів за допомогою кросовера створюється два

нових вектора. Із створеної “сім’ї” виділяються два найкращі вектори, які впроваджують у наступну популяцію як потомство. Застосування елітарного відбору традиційним способом на рівні всієї сукупності часто може бути причиною передчасної ковергеції алгоритму. EX елітарний відбір на “сімейному” рівні, на думку авторів, усуває цю проблему.

4.3. Мутації

Інколи може здаватися, що значення функції придатності (fitness) отриманих нащадків стагнує навколо оптимальної точки[3]. Тому що одні і ті ж рядки можуть повторюватися знов і знов в нащадку, що і призводить до застою. Цю проблему можна вирішити, застосувавши оператор мутації, який допомагає досягти найкращої оптимальної точки Одним з найбільш відомих механізмів створення варіацій є мутація, коли нові пробні рішення створюються шляхом внесення невеликих випадкових змін у представлення попередніх пробних рішень. Якщо використовується двійкове представлення, то мутація досягається шляхом випадкового перевертання одного біта.

Функція придатності (fitness) вимірює якість згенерованого рішення. Функція fitness завжди залежить від проблеми. Після визначення функції пристосованості ГА переходить до поколінь, що повторюються, застосовуючи оператори схрещування, мутації інверсії та відбору для оптимального рішення. Функція придатності - це функція мінімізації або максимізації, яка використовується для обчислення відносного значення отриманого рішення з очікуваним.

4.4. Селекція

Фаза відбору визначає, які особини вибираються для спарювання(розмноження) і скільки потомства дає кожна обрана особина. Основний принцип стратегії відбору - “чим краще індивід, тим вище його шанс стати батьком” 6] . Це процес, який визначає, які індивіди мають бути збережені та дозволені для відтворення, а які мають зникнути. Основна мета операції відбору полягає у тому, щоб зробити акцент на хороших рішеннях і усунути погані рішення в сукупності, зберігаючи при цьому розмір популяції незмінним.

Відбір вводить вплив функції пристосованості на процес оптимізації генетичного алгоритму. Відбір має використовувати придатність даної особи, оскільки придатність є мірою “добротності” особи. Однак вибір не може ґрунтуватися лише на виборі найкращої особини, оскільки найкращий індивід може бути не дуже близьким до оптимального рішення. Натомість певна ймовірність того, що відносно непридатні особини будуть відібрані, має бути збережена, щоб гарантувати, що гени, які несуть ці непридатні особи, не “втрачені” передчасно з популяції. Для оптимізації генетичного алгоритму було розроблено та використано ряд стратегій відбору.

Proportionate roulette wheel selection

Теорема схеми, наведена Holland[7], передбачає, що хромосоми вибираються на основі їхніх ймовірностей, які пропорційні їх значенню пристосованості. Його принцип вибору подібний до рулетки. Імовірність виділення сектора в рулетці пропорційна величині центрального кута сектора. Аналогічно в генетичному алгоритмі вся популяція розділена на колесо, і кожен сектор представляє індивіда. Частка придатності індивіда до загальних значень придатності всієї популяції визначає ймовірність відбору цієї особини в

наступному поколінні. Це визначає площу, яку займатиме індивід на колесі[8]. Схематично роботу методо зображено на рисунку 4.2.

Нижче наведено кроки для вибору колеса рулетки:

- Обчислити суму значень придатності кожної особини в популяції
- Обчисліть пристосованість кожної особи та її ймовірність вибору, поділивши пристосованість окремої хромосоми на суму значень пристосованості всієї популяції
- Розбити колесо рулетки на сектори відповідно до ймовірностей, розрахованих на другому кроці
- Поверніть колесо n кількість разів. Коли рулетка зупиняється, сектор, на який вказує вказівник, відповідає вказаній особині

Імовірність вибору індивіда розраховується наступною формулою

$$ps(a_i) = \frac{f(a_i)}{\sum_{i=1}^{n} f(a_j)}; j = 1,2, \dots, n$$

де n - це розмір популяції

$f(a_i)$ - значення пристосованості індивіда i

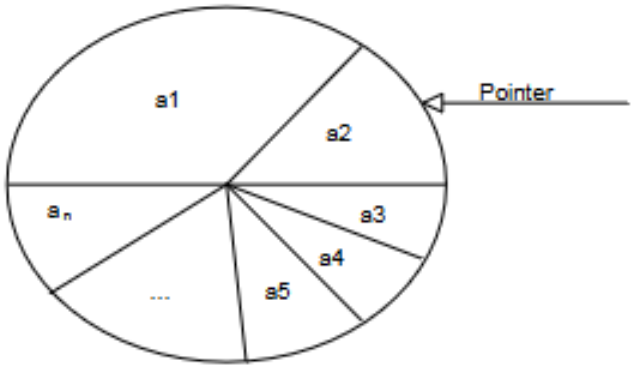


Рисунок 10 – Селекція методом рулетки

Linear Ranking Selection

Даний метод ввів Baker[9] з метою усунення недоліків пропорційного відбору, описаного вище. У методі відбору лінійного рейтингу особи спочатку сортується відповідно до їхньої придатності, а потім їм присвоюються ранги. Найкраща особа отримує ранг N, а найгірша отримує ранг 1. Імовірність відбору потім лінійно призначається індивідам відповідно до їх рангу.

$$P_i = \frac{1}{N}(n^+ - n^-)\frac{i-1}{N-1}; i \in \{1, \dots, N\}$$

P_i- ймовірність і-ого індивіда.

$\frac{n^-}{N}$ - імовірність відбору найгіршого індивіда

$\frac{n^+}{N}$ - імовірність відбору найкращого індивіда

Кожна особина отримує різний ранг, навіть якщо її ймовірності однакові.

Процес рейтингування складається з двох етапів. На першому кроці ми сортуємо сукупність, а на другому - присвоюємо ранги у порядку, що відповідає пропорційному відбору. Найкращий алгоитм сортування займає $O(n * \log n)$ часової складності. Таким чином, часова складність алгоритму буде $O(n * \log n)$ + час вибору (що становить приблизно між $O(n)$ і $O(n^2)$)

Expotential Ranking Selection

Ця методика відрізняється від техніки лінійного ранжирування тим, що ймовірності тут експотенціально зважені. Основою показника є c, де $0 < c < 1$

$$p_i = \frac{c^{N-i}}{\sum_{j=1}^{N} c^{N-j}}; i \in \{1, \dots, N\}$$

$\sum_{j=1}^{N} c^{N-j}$ – сума нормалізованих ймовірностей для визначення $\sum_{i=1}^{N} p_i = 1$

Алгоритм як лінійного, так і експотенційного рейтингу однакові. Різниця лише в обчисленні ймовірностей відбору [6]. У цьому також найкращій особині присвоюється ранг N а найгіршій 1

4.5. Обмін

Іноді навіть після багатьох ітераційних результатів рядки у вихідній позиції батьків у популяції залишаються незмінними навіть через багато поколінь і можуть призвести до стагнації. Цього можна уникнути, використовуючи оператор підкачки. Обмін - це особливий тип генетичного оператора, який використовується в цьому дослідженні, щоб розширити простір пошуку замість того, щоб зосередитися на конкретних локальних оптимах.

4.6. Припинення

Це етап, на якому визначено оптимальне рішення. Зазвичай алгоритм припиняє роботу, коли або створено максимальну кількість поколінь, або досягнуто задовільного придатності для популяції. Умови припинення можуть відрізнятися від програми до програми, і деякі з загальновживаних умов припинення є такими:

- Якщо рішення досягло мінімальних критеріїв, тобто значення функції придатності досягає бажаного мінімального значення.
- Якщо досягнуто фіксоване число генерацій
- Якщо послідовні ітерації більше не дають кращих результатів, значення функції пристосованості залишається незмінним для безперервних ітерацій
- Якщо виділеного часу/грошей/вартості досягнуто. Шляхом ручного огляду.

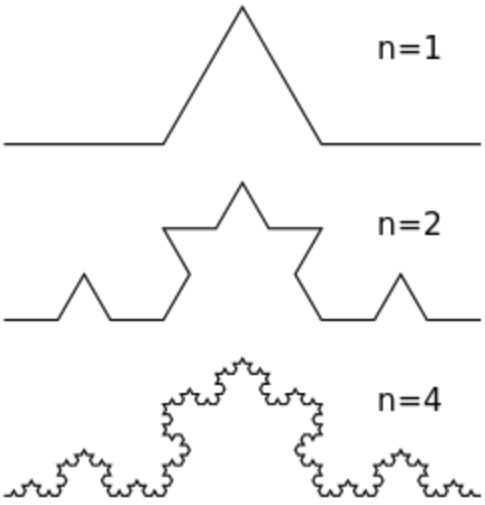
ДОДАТОК А

Чисельне наближення переміщення Леві (Levy Flight)

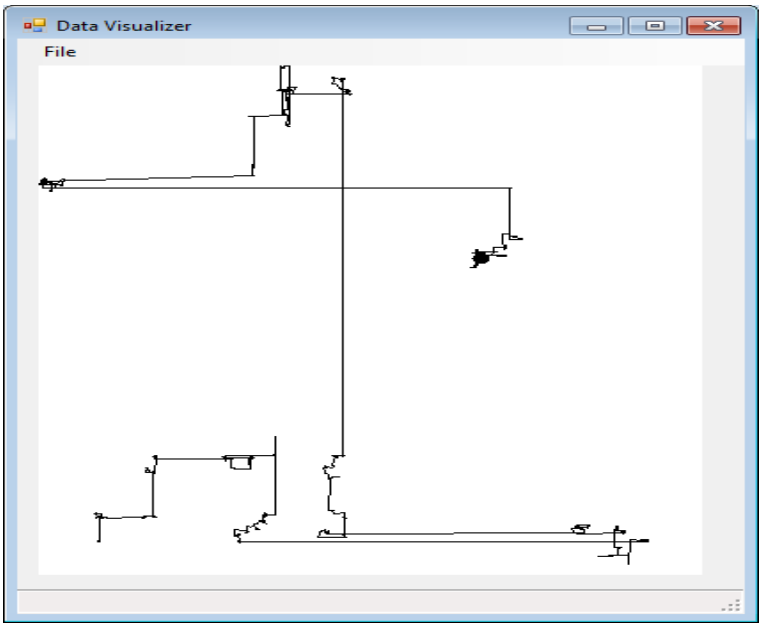
Поль Пьєр Леві (1886-1971), французький математик, розглядав у тридцятих роках XX століття наступну проблему: «Що, якщо можливо, що щільність імовірності N незалежна, тотожна розподіленню випадкових величин $X_1, X_2, \dots, X_N$ і задовольняє вимозі, що щільність ймовірності їх суми $X_1 + X_2 + \cdots + X_N$ має таку ж форму функціонального характеру?»

Сьогодні ми могли б сказати, що Леві намагався знайти клас самоподібних об’єктів, відомих як фрактали, оскільки Бенуа Мандельбро винайшов їх значно пізніше 1968 році.

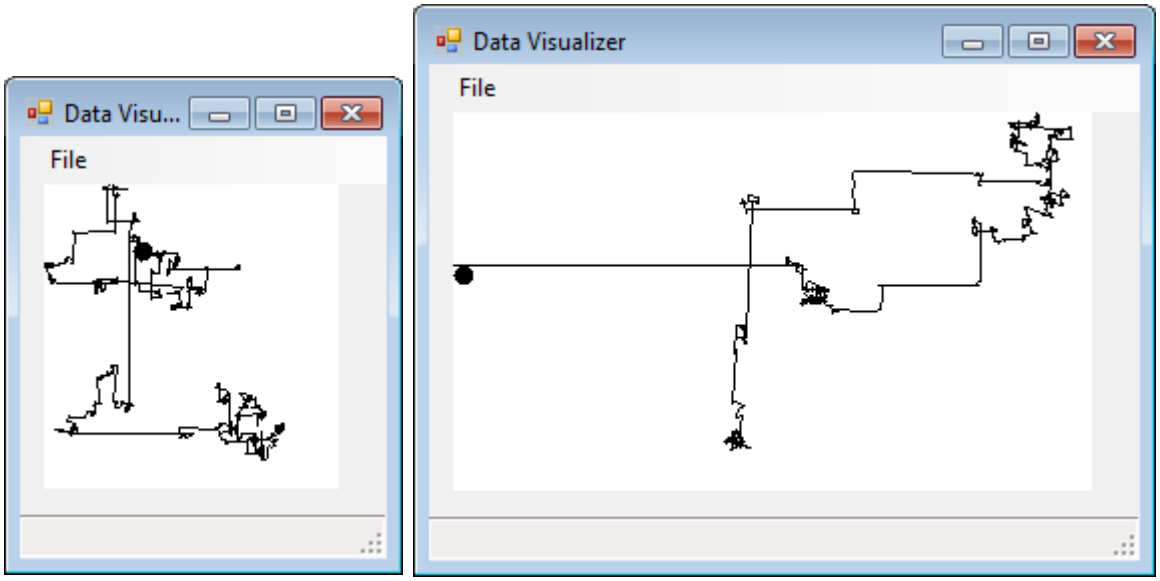
Фрактал – природне явище або математичний набір, який демонструє повторюваний шаблон, який відображається у кожному масштабі. Якщо реплікація точно така ж в кожному масштабі, вона називається самоподібною схемою.



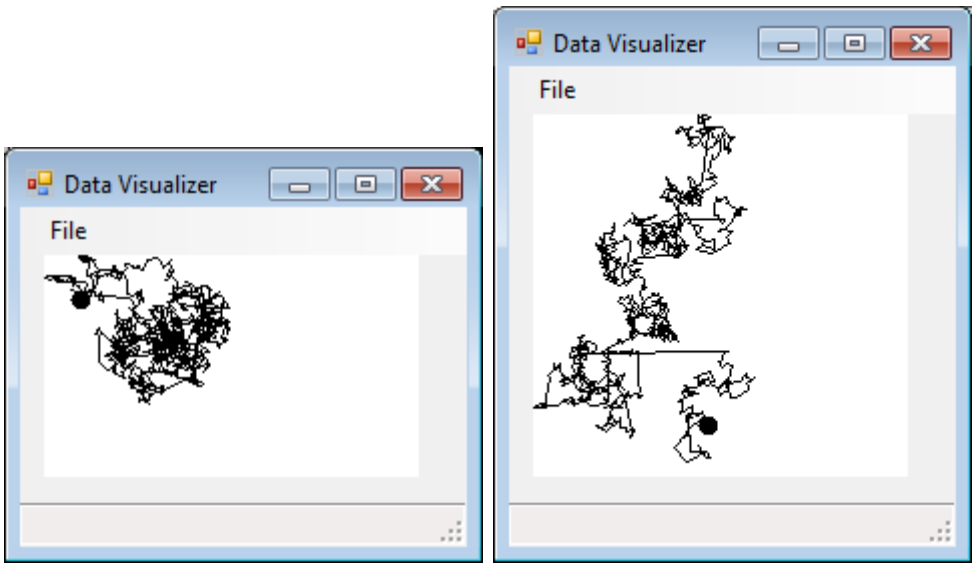
Тому використання ширини кроків, згенерованих відповідно до одного з розподілів Леві, замість рівномірного або Гаусового розподілу має бути вигідним. Співвідношення між двома класами може бути легко контрольоване шляхом відповідного вибору індексу λ.



$\lambda = 0{,}9$



$\lambda = 1{,}3$



$\lambda = 2{,}5$

Ідеальний розподіл Леві важко реалізувати в комп'ютерному коді, але в приблизний формі це легко.

Щоб перетворити рівномірно розподілену випадкову величину в інший розподіл, потрібно використовувати зворотну кумулятивну функцію розподілу. Тобто, якщо F – сукупна функція розподілу, яка відповідає щільності імовірності f, а u – однакова випадкова величина в [0,1], то

$$X = F^{-1}(u)$$

розподілена за F. Функція кумулятивної розподілу для чистого розподілу має вигляд

$$F(x) = 1 - \left(\frac{x}{x_{min}}\right)^{-\alpha},$$

де x_{min} – це мінімальне значення, яке може прийняти випадкова величина, і тому функцією зворотного розподілу є

$$F^{-1}(u) = x_{min}(1 - \text{г})^{-\frac{1}{\alpha}}.$$

Кумулятивна функція розподілу (CDF). Кумулятивна функція розподілу показує ймовірність того, що цільове значення менше вказаного значення або дорівнює йому. Вона доступна тільки для кількісних цільових значень.

Якщо необхідно запровадити штучний мінімум, можна розглянути двостороннє правило розподілу. Це не чисті силові закони (необхідно затиснути розподіл навколо нуля, щоб щільність не була нескінченною), але вони є асимптотичними в хвостах.

Одним з таких розподілів є t-розподіл Стьюдента зі ступенями свободи, який поводить себе як закон сили з показником ν, оскільки його значення прямують до $\pm\infty$.

Фактично, оскільки u рівномірно розподілена, можна отримати точно такий же розподіл за допомогою $F^{-1}(u) = u^{-1/\alpha}$, що має коротшу форму подання.

Найпростіший спосіб згенерувати переміщення Леві:

```
double levy_random(double x, double lambda, double alpha){

    double rnd = RAND;
    double f = pow(rnd, -1 / lambda);
    x += alpha * f * (RAND - .5);
    return x;
}
```

Алгоритм Мантегни

Алгоритм Мантегни генерує випадкові числа відповідно до симетричного стабільного розподілу Леві. Алгоритм потребує параметрів розподілу $\alpha \in [0{,}3; 1{,}99]$, $c > 0$ та кількості ітерацій n. Він також вимагає значення кількості точок для створення. Якщо цю кількість не вказано, алгоритм генерує лише одну точку. В алгоритмі Мантегни довжина кроку розраховується як

$$v = \frac{x}{|y|^{\frac{1}{\alpha}}},$$

де x і y нормально розподілені випадкові змінні $x \sim N(0, \sigma_x)$, $y \sim N(0, \sigma_y)$, де

$$\sigma_x(\alpha) = \left[\frac{\Gamma(\alpha+1)\sin\frac{\pi\alpha}{2}}{\Gamma\left(\frac{\alpha+1}{2}\right)\alpha 2^{\frac{\alpha-1}{2}}} \right]^{\frac{1}{\alpha}}, \sigma_y = 1$$

Спрощена версія алгоритму

Алгоритм починається з отримання одного за одним рішень з початкової популяції, а потім заміни його новим вектором, який генерується за допомогою кроків, описаних нижче:

$$stepsize = 0.01 * v.* (s - current\,current)$$

$$newsoln = oldsoln + stepsize.* z$$

де параметр v такий же, як і вищезазначеного алгоритму Мантегни з розрахунком для $\alpha = 3/2$, а z – нормально розподілена стохастична змінна.

Спрощена версія алгоритму з P-best

В алгоритмі спрощеної версії концепція глобального найкращого (G-best) використовується аналогічним чином, як і у алгоритмі МРЧ. Але це не включає поточний кращий вектор (P-best), який використовується в МРЧ для створення нових рішень.

ДОДАТОК Б

Метод Box-Muller:

```
double gaussian_random(double mue /*mean*/, double sigma/*stsndard deviation*/){
	double x1, x2, w, y;

	do {
		x1 = 2.0 * rand()/(RAND_MAX + 1) - 1.0;
		x2 = 2.0 * rand()/(RAND_MAX + 1) - 1.0;
		w = x1 * x1 + x2 * x2;
	} while ( w >= 1.0);
	double llog = log( w );
	w = sqrt( (-2.0 * llog ) / w );
	y = x1 * w;

	return mue + sigma * y;
}
```

Приклад набору параметрів:

```
//Archive
void Run(){
		int iter = 0;
		Archive ^tmp = gcnew Archive(k + colony_size);
		for(; iter < maxiter; iter++){
			for(int i = 0; i < colony_size; i++){
				ants[i]->ConstructSolution( archive, ksi);

			for(int i = 0; i < colony_size; i++){
				//add new solutions to buffer
			}
			for(int i = k; i < k + colony_size; i++){
				//recalc fitness			}
			}
			tmp->Sort();

			for(int i = 0; i < k; i++){
				//replace the colony size best solution
				//in archive
			}
			archive->CalcOmegas();

			iter++;
			if (iter > maxiter)
				return;
		}
	}

//Ant
void ConstructSolution(Archive ^ar, double ksi){
		array<double>^ omegas = ar->GetOmegas();
		for(int i = 0; i < dimentions; i++){
			//get Gaussian function for i-th decision variable
			int gil = roulette_wheel( omegas );
			ArchiveItem^ ai = ar->GetItem( gil );
			//returns j-th decision variable
			double mue = ai->GetXiJ( i );

			double sum = 0;
			for(int j = 0; j < k; j++){
				if( j == gil )
					continue;
				sum += fabs( solution[i]-ar->GetXiJ( i.j ));
			}
			//get variance for Gaussian function
			double sigma = ksi * sum / (k - 1);

			//sample decision variable from Gaussian kernel – формула 3
			double x = gaussian_random(mue, sigma);
			//check boundaries
			solution[i] = x;
		}
	}
```

ДОДАТОК В

Покрокове пояснення ABC

Нехай необхідно виконати наступну оптимізацію:

Мінімізувати $f(x) = x_1^2 + x_2^2$, де $-5 \leq x_1, x_2 \leq 5$.

Керуючі параметри алгоритму встановлено як:

- Розмір колонії $CS = 6$
- Ліміт для розвідника $L = \frac{CS \cdot D}{2} = 6$, та вимірність $D = 2$.

Спочатку довільно ініціалізуємо координати трьох джерел нектару ($CS/2$) робочих бджіл, використовуючи рівномірний розподіл в діапазоні $(-5,5)$.

Фітнес-функція обчислюється як

$$fit_i = \begin{cases} \frac{1}{1+f_i}, f_i \geq 0 \\ 1 + |f_i|, f_i < 0 \end{cases}$$

Маємо

x_1	x_2	f	fit
1.4112	-2.5644	8.5678	0.1045
0.4756	1.4338	2.2820	0.3047
-0.1824	-1.0323	1.0990	0.4764

Максимальне значення фітнес-функції – 0.4764 – якість найкращого джерела нектару.

1. $Iter = 1$
2. Фаза робочих бджіл

   2.1.Робоча бджола #1

   Згідно (2) обчислюємо нові координати.

   $k = 1$ // довільно вибраний індекс

$j = 0$ // довільно вибрана вимірність функції (0, 1 або 2)

$\varphi = 0.8050$ // довільне число з $[-1,1]$

$v_0 = (2.1644, -2.5644)$

$f(v_0) = 11.2610$

$fit(v_0) = 0.0816$

Застосовуємо жадібний механізм для вибору між x_0 та v_0: $0.0816 < 0.1045$. Розв'язок #0 не може бути покращений, збільшуємо номер спроби.

2.2. Робоча бджола #2

Згідно (2) обчислюємо нові координати.

$k = 2$ // довільно вибраний індекс в околиці i

$j = 1$ // довільно вибрана вимірність функції (0, 1 або 2)

$\varphi = 0.0762$ // довільне число з $[-1,1]$

$v_1 = (0.4756, -1.6217)$

$f(v_1) = 2.8560$

$fit(v_1) = 0.2593$

Застосовуємо жадібний механізм для вибору між x_1 та v_1: $0.2593 < 0.3047$. Розв'язок #1 не може бути покращений, збільшуємо номер спроби.

2.3. Робоча бджола #3

Згідно (2) обчислюємо нові координати.

$k = 0$ // довільно вибраний індекс в околиці i

$j = 0$ // довільно вибрана вимірність функції (0, 1 або 2)

$\varphi = -0.0671$ // довільне число з $[-1,1]$

$v_2 = (-0.0754, -1.0323)$

$f(v_2) = 1.0714$

$fit(v_2) = 0.4828$

Застосовуємо жадібний механізм для вибору між x_2 та v_2: $0.4828 > 0.4764$. Розв'язок #2 був покращений, замінюємо x_2 на v_2, а також обчислюємо p_i згідно (1).

x_1	x_2	f	fit	p_i
1.4112	-2.5644	8.5678	0.1045	0.1172
0.4756	1.4338	2.2820	0.3047	0.3416
-0.0754	-1.0323	1.0714	0.4828	0.5412

3. Фаза бджіл-спостерігачів

Генеруємо нові координати v_i для спостерігачів з координат x_i залежно від p_i.

3.1. Бджола-спостерігач #1 ($i = 0$)

$v_2 = (-0.0754, -2.2520)$

$f(v_2) = 5.0772$

$fit(v_2) = 0.1645$

Застосовуємо жадібний механізм для вибору між x_2 та v_2: $0.1645 < 0.4828$. Розв'язок #2 не може бути покращений, збільшуємо номер спроби.

3.2. Бджола-спостерігач #2 ($i = 1$)

$v_1 = (0.1722, 1.4338)$

$f(v_1) = 2.0855$

$fit(v_1) = 0.3241$

Застосовуємо жадібний механізм для вибору між x_1 та v_1: $0.3241 > 0.3047$. Розв'язок #1 був покращений, замінюємо x_1 на v_1, а також встановлюємо номер спроби = 0.

x_1	x_2	f	fit
1.4112	-2.5644	8.5678	0.1045
0.1722	1.4338	2.0855	0.3241
-0.0754	-1.0323	1.0714	0.4828

3.3.Бджола-спостерігач #3 ($i = 2$)

$v_2 = (0.0348, -1.0323)$

$f(v_2) = 1.0669$

$fit(v_2) = 0.4838$

Застосовуємо жадібний механізм для вибору між x_2 та v_2: $0.4838 > 0.4828$. Розв'язок #2 був покращений, замінюємо x_2 на v_2, а також встановлюємо номер спроби = 0.

x_1	x_2	f	fit
1.4112	-2.5644	8.5678	0.1045
0.1722	1.4338	2.0855	0.3241
0.0348	-1.0323	1.0669	0.4838

Запам'ятовуємо найкраще значення: $Best = (0.0348, -1.0323)$.

4. Фаза бджіл-розвідників

Кількості спроб покращення джерел нектару: $(1, 0, 0)$.

Жодне джерело нектару не має бути покинутим оскільки $L = 6$.

Якщо існує джерело, що має бути покинутим (кількість спроб для нього більша ніж $L = 6$), то генеруємо нові координати і замінюємо ним покинутий.

5. $Iter = Iter + 1$

6. Процедура продовжується до досягнення критерія завершення.

ДОДАТОК Г

Приклад ітерації

Нехай у якості цільової функції є $f(x, y) = -(x^2 + y^2)$.

Знак мінус в даному випадку вибрано, щоб у функції був глобальний максимум, а не мінімум. Глобальний (і єдиний) максимум цієї функції знаходиться в точці $(0; 0)$, причому $f(0,0) = 0$.

Решта необхідних параметрів:

- Кількість бджіл-розвідників: 10
- Кількість бджіл, що відправляються на найкращі ділянки: 5
- Кількість бджіл, що відправляються на перспективні ділянки: 2
- Кількість найкращих ділянок: 2
- Кількість перспективних ділянок: 3
- Розмір кожної ділянки: 10

Нехай розвідники натрапили на наступні ділянки (список впорядкований за спаданням цільової функції):

- $f(15, 18) = -549$
- $f(-30, -15) = -1125$
- $f(22, -31) = -1445$
- $f(18, 40) = -1924$
- $f(-25, 47) = -2834$
- $f(60, 86) = -10996$
- $f(-91, -99) = -18082$
- $f(17, -136) = -18785$
- $f(-152, -1) = -22501$
- $f(-222, 157) = -73933$.

Спочатку буде обрано 2 найкращі точки:

- $f(15, 18) = -549$
- $f(-30, -15) = -1125$

Потім будуть обрані інші 3 перспективні ділянки:

- $f(22, -31) = -1445$
- $f(18, 40) = -1924$
- $f(-25, 47) = -2834$

В околиці найкращих точок буде надіслано по 5 бджіл:

Для першої найкращої точки значеннями координат, якими обмежується ділянка будуть:

- $[15 - 10 = 5;\ 15 + 10 = 25]$ для першої координати
- $[18 - 10 = 8;\ 18 + 10 = 28]$ для другої координати

І для другої точки:

- $[-30 - 10 = -40;\ -30 + 10 = -20]$ для першої координати
- $[-15 - 10 = -25;\ -15 + 10 = -5]$ для другої координати

Аналогічно розраховуються інтервали для обраних ділянок:

- $[12;\ 32]\ [-41;\ -21]$
- $[8;\ 28]\ [30;\ 50]$
- $[-35;\ 15]\ [37;\ 57]$

Зауважимо, що тут для кожної з координат розмір області однаковий і дорівнює 20, в реальності це не обов'язково так. До кожного з найкращих інтервалів відправляємо по 5 бджіл, а на перспективні ділянки по 2 бджоли. Причому, ми не будемо змінювати координати бджіл, які знайшли найкращі і перспективні ділянки, оскільки існує ймовірність того, що на наступній ітерації максимальне значення цільової функції буде гірше, ніж на попередньому кроці.

Нехай тепер на першій з найкращих ділянок ми маємо наступних бджіл:

- $f(15, 18) = -549$

- $f(7,12) = 193$
- $f(10,10) = 100$
- $f(16,24) = 832$
- $f(18,24) = 900$

Як видно, вже серед цих нових точок є такі, які кращі за попередній розв’язок. Ті самі дії виконуємо і з другою з найкращих ділянок, а потім аналогічно і з перспективними ділянками. Після чого серед усіх нових точок знову визначаються найкращі і перспективні, а процес повторюється знову.

ДОДАТОК Д

Припустимо, що задано наступну матрицю відстаней:

	1	2	3	4
1	0	2.06	4.03	6.32
2	2.06	0	2.50	4.12
3	4.03	2.50	0	2.24
4	6.32	4.12	2.24	0

Крок 1. На першому кроці кожен об'єкт являє собою окремий кластер: |1|, |2|, |3| і |4|. Згідно з критерієм класифікації, об'єднання відбувається між кластерами, відстань між найближчими представниками яких найменша: кластери |1| і |2|. Відстань, при якій відбулося об'єднання – 2,06. Необхідно виконати перерахунок матриці відстаней з урахуванням нового кластера:

	1,2	3	4
1,2	0	2.50	4.12
3	2.50	0	2.24
4	4.12	2.24	0

Ми пам'ятаємо, що за допомогою методу одиничного зв'язку подібність двох кластерів визначається як мінімальна відстань між будь-якими двома точками двох ластерів.

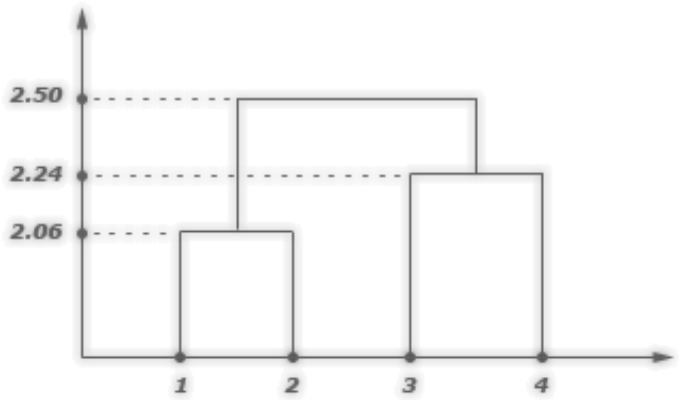
Відстань між, наприклад (p_3, p_6) і p_1 буде розрахована наступним чином:

$$dist\big((1,2),3\big) = \min\big(dist(1,3), dist(2,3)\big) = \min(4.03, 2.50) = 2.50.$$

Крок 2. Кластери на даному етапі: |1,2|, |3| і |4|. Згідно з новою матрицею відстаней, кластери |3| і |4| є найближчими. Відстань об'єднання – 2,24. Необхідно виконати перерахунок матриці відстаней з урахуванням нового кластера:

	1,2	3,4
1,2	0	2.50
3,4	2.50	0

Крок 3. Кластери на даному етапі: |1,2| і |3,4|. Відстань між кластерами дорівнює 2,5 – це відстань між об’єктами 2 і 3. Формування кластерів закінчено. Результат класифікації методом найближчого сусіда представлений у вигляді дендрограми:



При використанні методу найближчого сусіда особливу увагу слід приділяти вибору міри відстані між об’єктами. На основі неї формується початкова матриця відстаней, яка і визначає весь подальший процес класифікації.

Метод найбільш віддалених сусідів або метод повного зв’язку

Перед початком роботи алгоритму розраховується матриця відстаней між об’єктами. На кожному кроці в матриці відстаней знаходиться мінімальне значення, що відповідає відстані між двома найближчими кластерами. Знайдені кластери об’єднуються, утворюючи новий кластер. Ця процедура повторюється до тих пір, поки не будуть об’єднані всі кластери. Припустимо, задана наступна матриця відстаней:

	1	2	3	4
1	0	2.06	4.03	6.32
2	2.06	0	4.12	2.25
3	4.03	4.12	0	3.50
4	6.32	2.25	3.50	0

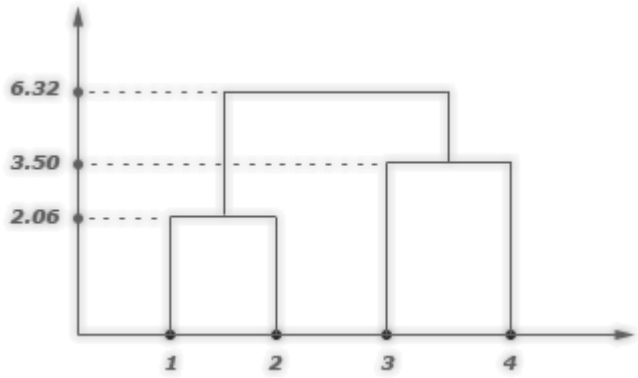
Крок 1. На першому кроці кожен об'єкт являє собою окремий кластер. Згідно з критерієм класифікації, об'єднання відбувається між кластерами, відстань між якими найменша. Таким чином на цьому кроці об'єднуються кластери |1| і |2|. Відстань об'єднання – 2,06. Необхідно виконати перерахунок матриці відстаней з урахуванням нового кластера (нагадаємо, що відстань між класами визначається як відстань між їх найбільш віддаленими представниками):

	1,2	3	4
1,2	0	4.12	6.32
3	4.12	0	3.50
4	6.32	3.50	0

Крок 2. Кластери на даному етапі: |1,2| і |3,4|. Згідно з новою матрицею відстаней, кластери |3| і |4| є найближчими. Відстань об'єднання – 3,5. Необхідно виконати перерахунок матриці відстаней з урахуванням нового кластера:

	1,2	3,4
1,2	0	6.32
3,4	6.32	0

Крок 3. Кластери на даному етапі: |1,2| і |3,4|. Відстань між кластерами дорівнює 6,23 – це відстань між об’єктами 1 і 4. Формування кластерів закінчено. Результат роботи алгоритму представлений у вигляді дендрограми:



Метод зваженого попарного середнього (WPGMA)

Метод зваженого попарного середнього – Weighted Pair-Group Method Using Arithmetic Averages або скорочено WPGMA.

Нехай необхідно виконати класифікацію заданої множини об’єктів методом зваженого попарного середнього.

Перед початком роботи алгоритму розраховується матриця відстаней між об’єктами. На кожному кроці в матриці відстаней знаходиться мінімальне значення, що дорівнює відстані між двома найближчими кластерами. Знайдені кластери u і v об’єднуються, утворюючи новий кластер. Рядки і стовпці, що відповідають кластерам u і v, видаляються з матриці відстаней, і додається новий рядок та новий стовпець, що відповідає кластеру k. В результаті матриця зменшується на один рядок і один стовпець. Ця процедура повторюється до тих пір, поки не будуть об’єднані всі кластери. Нехай задана наступна матриця відстаней:

	1	2	3	4	5
1	0	2.06	4.03	6.32	2.08
2	2.06	0	3.50	4.12	5.43
3	4.03	3.50	0	2.25	3.65
4	6.32	4.12	2.25	0	4.81
5	2.08	5.43	3.65	4.81	0

Нехай кластери, u, v і k містять, T_u, T_v і T_k об'єктів відповідно. Кластер k утворений шляхом об'єднання кластерів u і v, тоді $T_k = T_u + T_v$. Необхідно розрахувати віддаленість кластера k від деякого кластера w. Відстань між цими кластерами визначається згідно з формулою:

$$D\big((u,v),w\big) = \frac{T_u D_{u,w} + T_v D_{v,w}}{T_u + T_v}.$$

Крок 1. На першому кроці, коли кожен об'єкт являє собою окремий кластер: |1|, |2|, |3|, |4| і |5|. Згідно з критерієм класифікації, об'єднання відбувається між кластерами, відстань між якими найменша. Таким чином на цьому кроці об'єднуються кластери |1| і |2|. Відстань об'єднання – 2,06. Необхідно виконати перерахунок матриці відстаней з урахуванням нового кластера:

	1,2	3	4	5
1,2	0	3.765	5.22	3.755
3	3.765	0	2.25	3.65
4	5.22	2.25	0	4.81
5	3.755	3.65	4.81	0

Наведемо приклад розрахунку

Відстані між кластерами $k = |1,2|$ і $w = |3|$. Кластер k утворений шляхом об'єднання кластерів $u = |1|$ і $v = |2|$. Відстані $D(u,w)$ і $D(v,w)$ беремо з початкової матриці відстаней. Підставивши отримані значення в формулу, отримаємо:

$$D = \frac{1 \cdot 4{,}03 + 1 \cdot 3{,}5}{1+1} = 3{,}765.$$

Крок 2. Згідно з новою матриці відстаней, кластери $|3|$ і $|4|$ є найближчими. Відстань об'єднання – 2,36. Необхідно виконати перерахунок матриці відстаней з урахуванням отриманого кластера:

	1,2	3,4	5
1,2	0	4.4925	3.755
3,4	4.4925	0	4.23
5	3.755	4.23	0

Наведемо приклад розрахунку

Відстані між кластерами $k = |3,4|$ і $w = |1,2|$. Кластер k утворений шляхом об'єднання кластерів $u = |3|$ і $v = |4|$. Відстані $D(u,w)$ і $D(v,w)$ беремо з матриці відстаней попереднього кроку. Підставивши отримані значення в формулу, отримаємо:

$$D = \frac{1 \cdot 3{,}765 + 1 \cdot 5{,}22}{1+1} = 4{,}4925.$$

Крок 3. Кластери на даному етапі: $|1,2|$ і $|3,4|$ і $|5|$. Кластери $|1,2|$ і $|5|$ є найближчими. Відстань об'єднання – 3,755. Необхідно виконати перерахунок матриці відстаней з урахуванням нового кластера:

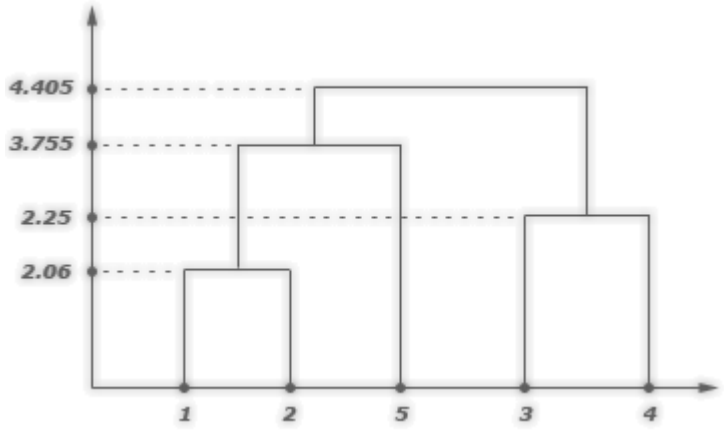
	1,2,5	3,4
1,2,5	0	4.405
3,4	4.405	0

Наведемо приклад розрахунку

Відстані між кластерами $k = |1,2,5|$ і $w = |3,4|$. Кластер k утворений шляхом об'єднання кластерів $u = |1,2|$ і $v = |5|$. Відстані $D(u,w)$ і $D(v,w)$ беремо з матриці відстаней попереднього кроку. Підставивши отримані значення в формулу, отримаємо:

$$D = \frac{2 \cdot 4{,}4925 + 1 \cdot 4{,}23}{2+1} = 4{,}405.$$

Крок 4. На останньому кроці об'єднуються два кластери, що залишилися: $|1,2,5|$ і $|3,4|$. Відстань об'єднання – 4,405. Результат роботи алгоритму представлений у вигляді дендрограми:



Метод незваженого попарного середнього (UPGMA)

Метод невиваженого попарного середнього – Unweighted Pair-Group Method Using Arithmetic Averages або скорочено UPGMA.

Перед початком роботи алгоритму розраховується матриця відстаней між об’єктами. На кожному кроці в матриці відстаней знаходиться мінімальне значення, що відповідає відстані між двома найближчими кластерами. Знайдені кластери u і v об’єднуються, утворюючи новий кластер k. Рядки і стовпці, що відповідають кластерам u і v, видаляються з матриці відстаней, і додається новий рядок та новий стовпець, що відповідає кластеру k. В результаті матриця зменшується на один рядок і один стовпець. Ця процедура повторюється до тих пір, поки не будуть об’єднані всі кластери. Нехай задана наступна матриця відстаней:

	1	2	3	4	5
1	0	2.06	4.03	6.32	2.08
2	2.06	0	3.50	4.12	5.43
3	4.03	3.50	0	2.25	3.65
4	6.32	4.12	2.25	0	4.81
5	2.08	5.43	3.65	4.81	0

Нехай кластер k утворений шляхом об’єднання кластерів u і v. Необхідно розрахувати віддаленість кластера k від деякого кластера w. Відстань між цими кластерами визначається згідно з формулою:

$$D\big((u,v),w\big) = \frac{D_{u,w} + D_{v,w}}{2}.$$

Крок 1. На першому кроці кожен об’єкт являє собою окремий кластер: $|1|$, $|2|$, $|3|$, $|4|$ і $|5|$. Згідно з критерієм класифікації, об’єднання відбувається між кластерами, відстань між якими найменша. Таким чином на цьому кроці

об’єднуються кластери |1| і |2|. Відстань об’єднання – 2,06. Необхідно виконати перерахунок матриці відстаней з урахуванням нового кластера:

	1,2	3	4	5
1,2	0	3.765	5.22	3.755
3	3.765	0	2.25	3.65
4	5.22	2.25	0	4.81
5	3.755	3.65	4.81	0

Наведемо приклад розрахунку

Відстані між кластерами $k = |1,2|$ і $w = |3|$. Кластер k утворений шляхом об’єднання кластерів $u = |1|$ і $v = |2|$. Відстані $D(u,w)$ і $D(v,w)$ беремо з початкової матриці відстаней. Підставивши отримані значення в формулу, отримаємо:

$$D = \frac{4,03 + 3,5}{2} = 3,765.$$

Крок 2. Згідно з новою матриці відстаней, кластери |3| і |4| є найближчими. Відстань об’єднання – 2,25. Необхідно виконати перерахунок матриці відстаней з урахуванням отриманого кластера:

	1,2	3,4	5
1,2	0	4.4925	3.755
3,4	4.4925	0	4.23
5	3.755	4.23	0

Наведемо приклад розрахунку відстані між кластерами $k = |3,4|$ і $w = |1,2|$. Кластер k утворений шляхом об’єднання кластерів $u = |3|$ і $v = |4|$.

Відстані $D(u,w)$ і $D(v,w)$ беремо з матриці відстаней попереднього кроку. Підставивши отримані значення в формулу, отримаємо:

$$D = \frac{3{,}765 + 5{,}22}{2} = 4{,}4925.$$

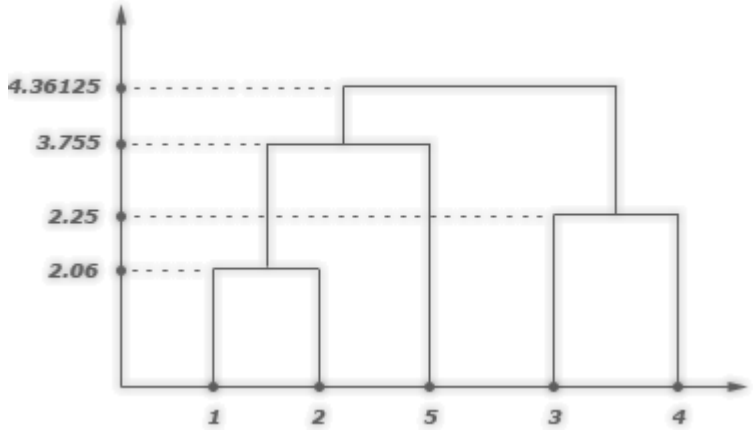
Крок 3. Кластери на даному етапі: |1,2| і |3,4| і |5|. Кластери |1,2| і |5| є найближчими. Відстань об'єднання – 3,755. Необхідно виконати перерахунок матриці відстаней з урахуванням нового кластера:

	1,2,5	3,4
1,2,5	0	4.36125
3,4	4.36125	0

Наведемо приклад розрахунку відстані між кластерами $k = |1,2,5|$ і $w = |3,4|$. Кластер k утворений шляхом об'єднання кластерів $u = |1,2|$ і $v = |5|$. Відстані $D(u,w)$ і $D(v,w)$ беремо з матриці відстаней попереднього кроку. Підставивши отримані значення в формулу, отримаємо:

$$D = \frac{4{,}4925 + 4{,}23}{2} = 4{,}36125.$$

Крок 4. На останньому кроці об'єднуються два кластери, що залишилися: |1,2,5| і |3,4|. Відстань об'єднання – 4,36125. Результат роботи алгоритму представлений у вигляді дендрограми:



ЛІТЕРАТУРНІ ДЖЕРЕЛА

Розділ 1

1. Ch. Blum, M.J.B. Aguilera, A. Roli. Hybrid Metaheuristics. An Emerging Approach to Optimization ,Berlin: Springer-Verlag, 2008,289 p
2. Маньєццо, Вітторіо; Штюцле, Томас; Voß, Stefan (Eds.), Гібридизуюча метаевристика та математичне програмування. Серія: Аннали інформаційних систем, вип. 10, Springer, 2009
3. Б'янкі, Леонора; Марко Доріго; Лука Марія Гамбарделла; Вальтер Дж. Гутяр, Опитування з метаевристики для стохастичної комбінаторної оптимізації, Природні обчислення, **8** (2),2009,с. 239–287
4. В. Б. Гітіс, К. Ю. Гудкова. Методи штучного інтелекту: навч. посіб, Краматорськ: ДДМА, 2018. — 136 с.
5. Субботін С. О., Олійник А. О., Олійник О. О. Неітеративні, еволюційні та мультиагентні методи синтезу нечіткологічних і нейромережних моделей: Монографія / Під заг. ред. С. О. Субботіна. — Запоріжжя: ЗНТУ, 2009. — 375 с
6. Eberhart R.C., Shi Y. (2000). Comparing inertia weights and constriction factors in particle swarm optimization. Proceedings of the Congress on Evolutionary Computation **1**. с. 84–88.
7. Poli R. An analysis of publications on particle swarm optimisation applications ,Technical Report CSM-469, Department of Computer Science, University of Essex, UK, 2007.
8. Poli, R. . Analysis of the publications on the applications of particle swarm optimisation. Journal of Artificial Evolution and Applications:,2008, pp. 1–10.
9. Carlisle A., Dozier G. (2001). An Off-The-Shelf PSO. Proceedings of the Particle Swarm Optimization Workshop. с. 1–6.
10. Clerc M., Kennedy J. (2002). The particle swarm - explosion, stability, and convergence in a multidimensional complex space. IEEE Transactions on Evolutionary Computation **6** (1): 58–73.
11. Trelea, I.C. (2003). The Particle Swarm Optimization Algorithm: convergence analysis and parameter selection. Information Processing Letters **85**: 317–325.
12. Bratton D., Blackwell T. (2008). A Simplified Recombinant PSO. Journal of Artificial Evolution and Applications.
13. Evers, G. (2009). An Automatic Regrouping Mechanism to Deal with Stagnation in Particle Swarm Optimization (Master's thesis). The University of Texas - Pan American, Department of Electrical Engineering. Архів оригіналу за 18 травня 2011. Процитовано 8 червня 2011.

14. Pedersen, M.E.H. (2010). Tuning & Simplifying Heuristical Optimization (PhD thesis). University of Southampton, School of Engineering Sciences, Computational Engineering and Design Group.
15. Pedersen, M.E.H.; Chipperfield, A.J. (2010). Simplifying particle swarm optimization. Applied Soft Computing **10**: 618–628. Архів оригіналу за 24 січня 2014. Процитовано 8 червня 2011.
16. Lovbjerg M., Krink T. (2002). The LifeCycle Model: combining particle swarm optimisation, genetic algorithms and hillclimbers. Proceedings of Parallel Problem Solving from Nature VII (PPSN). с. 621–630.
17. Niknam T., Amiri B. (2010). An efficient hybrid approach based on PSO, ACO and k-means for cluster analysis. Applied Soft Computing **10** (1): 183–197.
18. Xinchao, Z. (2010). A perturbed particle swarm algorithm for numerical optimization. Applied Soft Computing **10** (1): 119–124.
19. Zhan Z-H., Zhang J., Li Y., Chung H.S-H. (2009). Adaptive Particle Swarm Optimization. IEEE Transactions on Systems, Man, and Cybernetics **39** (6): 1362–1381.
20. Waldner, Jean-Baptiste (2008). Nanocomputers and Swarm Intelligence. London: ISTE John Wiley & Sons. p. 215
21. Yang X. S. A New Metaheuristic Bat-Inspired Algorithm. Nature Inspired Cooperative Strategies for Optimization (NICSO 2010). 2010. Vol. 284. P. 65–74.
22. Yang X.-S., A New Metaheuristic Bat-Inspired Algorithm, in: Nature Inspired Cooperative Strategies for Optimization (NISCO 2010) (Eds. J. R. Gonzalez et al.), Studies in Computational Intelligence, Springer Berlin, 284, Springer, 65–74 (2010)
23. X. S. Yang and S. Deb, Multiobjective cuckoo search for design optimization, Computers and Operations Research, October ,2011
24. Ян, Х. S. Натхненні природою метаевристичні алгоритми. Luniver Press.,2008
25. Ariyaratne M., Pemarathne W. Огляд останніх досягнень алгоритму світлячка: сучасний натхненний алгоритм, Матеріали 8-ї міжнародної науково-дослідної конференції, 61–66, КДУ, Опубліковано в листопаді 2015 р.
26. Marco., Dorigo, (2004). Ant colony optimization. Cambridge, Mass.: MIT Press.
27. Larranaga, P. and Lozano, J. A. (eds): Estimation of distribution algorithms, Kluwer Academic Publishers (2002).
28. Schatzman, M., Taylor, J., and Schatzman, M.: Numerical Analysis: A Mathematical Introduction, Oxford Univ Press (2002).

29. Arvin, Farshad; Samsudin, Khairulmizam; Ramli, Abdul Rahman; Bekravi, Masoud (2011). Imitation of Honey bee Aggregation with Collective Behavior of Swarm Robots.. International Journal of Computational Intelligence Systems **4** (4). с. 739
30. S. L. Ukasik and AK. SÃlawomirZ, Firefly algorithm for Continuous Constrained Optimization Tasks, 1st International Confernce on Computational Collective Intelligence, Semantic Web, Social Networks and Multiagent Systems, Springer-Verlag Berlin, Heidelberg 2009. – p.169-178.
31. M. Dhivya and M. Sundarambal, Cuckoo search for data gathering in wireless sensor networks,Int. J. Mobile Communications, Vol. 9, pp. 642-656 (2011).
32. .Dorigo, M., Stutzle, T., Ant Colony Optimization. MIT Press, Cambridge, MA, 2004.
33. Karaboga D. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm / D. Karaboga, B. Basturk // Journal of Global Optimization. – 2007. – Vol. 39. – P. 459–471.

Розділ 2.

1. Estivill-Castro, Vladimir. Why so many clustering algorithms – A Position Paper. ACM SIGKDD Explorations Newsletter **4** (1): 65–75.
2. Jain A., Murty M., Flynn P. Data Clustering: A Review. // ACM Computing Surveys. 1999. Vol. 31, no. 3.
3. Clustering with Minimum Spanning Trees / Y.Zhou, O.Grygorash, T.F.Hain та ін. .— Elsevier, 2010 .— 22 p.
4. CHIH-TANG CHANG1 , JIM Z. C. LAI2 AND MU-DER JENG, A Fuzzy K-means Clustering Algorithm Using Cluster Center Displacement , JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 27, 995-1009 (2011)
5. Д.В. Ланде, І.Ю. Субач, Ю.Є. Боярінова Основи теорії і практики інтелектуального аналізу даних у сфері кібербезпеки: навчальний посібник, Київ: ІСЗЗІ КПІ ім. Ігоря Сікрорського, 2018. - 300 с. ISBN 978-966-2577-12-9

Розділ 3

1. Глибовець М. М., Олецький О.В.Штучний інтелект,Києво-Могилянська академія, 2002,364 с.
2. В. Б. Гітіс, К. Ю. Гудкова Методи штучного інтелекту: навч. Посіб,Краматорськ: ДДМА, 2018,136 с.
3. Н. Б. Шаховська, Р. М. Камінський, О. Б. Вовк.Системи штучного інтелекту: навч. Посіб, Львів: Львівська політехніка, 2018,392 с..
4. Системи штучного інтелекту: навч. посіб. / Ю. В. Нікольський, В. В. Пасічник, Ю. М. Щербина ; за наук. ред. В. В. Пасічника ; М-во

освіти і науки, молоді та спорту України.,Львів: Магнолія-2006, 2013,279 с.
5. Stuart J. Russell, Peter Norvig. Artificial Intelligence: A Modern Approach,Pearson,2015
6. Alan Bundy, Rod Burstall. Artificial Intelligence: An Introductory Course,Revised, Edinburgh University Press, 1984,200 с.
7. Nils J. Nilsson. The Quest for Artificial Intelligence,Cambridge University Press, 2009, 578 с.
8. М.А. Новотарський Б.Б. Нестеренко, Штучні нейронні мережі, Інститут математики НАН України, Київ, 2004, 207с.

Розділ 4

1. Mitchell M. An Introduction to Genetic Algorithm, MIT Press. 1996, P.158
2. Bortfeldt A and Gehring A Hybrid Genetic Algorithm for Container Loading Problem, European Journal of Operational Research, 2001, vol. 131, issue 1, 143-161
3. Goldberg D.E. Genetic Algorithm in SearchOptimization and Machine Learning,1988, volume 3, pp.95–99
4. Tomasz Dominik Gwiazda, Genetic Algorithms Reference", Volume –I, Poland: Tomasz Gwiazda, 2006, P.410
5. Shengxiang Yang, Adaptive Non-Uniform Crossover Based on Statistics for Genetic Algorithms, Proceedings of the Genetic and Evolutionary Computation Conference, 2002, pp. 650-657
6. Blickle, Tobias, and Lothar Thiele. A comparison of selection schemes used in genetic algorithms,1995, P. 65
7. John Holland. Adaptation in natural and artificial systems. The MIT Press Cambridge, Massachusetts London, England ,1992, P.207
8. Zhong, Jinghui, et al. «Comparison of performance between different selection strategies on simple genetic algorithms.» Computational Intelligence for Modelling, Control and Automation, 2005 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on. Vol. 2. IEEE, 2005.
9. Baker, James Edward. Adaptive selection methods for genetic algorithms, Proceedings of an International Conference on Genetic Algorithms and their applications,1985, P.234