

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
УЖГОРОДСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ СИСТЕМ І МЕРЕЖ**

**ДОСЛІДЖЕННЯ КОМП'ЮТЕРНИХ СИСТЕМ
ШТУЧНОГО ІНТЕЛЕКТУ**

**Методичні вказівки до лабораторних робіт
для студентів 5-го курсу
спеціальності КСМ**

Ужгород - 2016

**Методичні вказівки до лабораторних робіт з курсу
Дослідження комп'ютерних систем штучного інтелекту для
студентів 5-го курсу
інженерно-технічного факультету,
спеціальності комп'ютерні системи і мережі**

Рецензент: **Маляр М.М., кандидат технічних наук, доцент кафедри
кібернетики і прикладної математики**

Укладачі: **Балога С.І., кандидат фізико-математичних наук,
доцент кафедри комп'ютерних систем і мереж**

Відповідальний за випуск: **Туряниця І.І., канд. фіз.-мат. наук, професор,
декан інженерно-технічного факультету**

**Дані методичні вказівки розглянуто та схвалено на засіданні кафедри
комп'ютерних систем та мереж, протокол № 5 від 28 січня 2016 р. та
методичної комісії інженерно-технічного факультету, протокол № 1 від
5 лютого 2016 р.**

ЛАБОРАТОРНА РОБОТА № 1

Тема: НЕЙРОННА МЕРЕЖА ЗВОРОТНЬОГО ПОШИРЕННЯ ПОХИБКИ. НЕЙРОЕМУЛЯТОР NEURAL NETWORK WIZARD

Neural Network Wizard 1.7 - це програмний емулятор нейрокомп'ютера. В Neural Network Wizard реалізовано багатошарову нейронну мережу, що навчається за алгоритмом зворотного поширення похибки (back propagation). Програма може застосовуватися для аналізу інформації, побудови моделей процесів і прогнозування. Для роботи із системою необхідно здійснити наступні операції:

- Зібрати статистику по процесу.
- Навчити нейромережу на приведених даних.
- Перевірити отримані результати.

Під час навчання нейромережа самостійно підбирає значення коефіцієнтів і будує таку модель, що найточніше відбиває процес дослідження.

ПІДГОТОВКА ВХІДНИХ ДАНИХ

Дані для навчання мережі мають бути сформовані в текстовому файлі з розділювачами (Tab чи пробіл). Кількість прикладів повинна бути досить великою і вибірка має бути репрезентативною. Крім того, потрібно забезпечити, щоб дані були не суперечливі. Вся інформація повинна бути представлена в числовому виді. Причому це стосується всіх даних. Якщо інформація представляється в текстовому виді, то необхідно застосувати певний метод, щоб перевести текстову інформацію у числову. Висока якість отриманих результатів досягається, якщо вжити передобробку даних. Якщо текстову інформацію можна якось ранжувати, то необхідно це враховувати. Наприклад, якщо ви кодуєте інформацію про міста, то можна ранжувати по чисельності населення і задати відповідне кодування: Київ = 1, Львів = 2, і т.д. Якщо ж дані не можна впорядкувати, то можна задати їм довільні номери. Взагалі, краще при кодуванні вхідної інформації збільшувати відстань між об'єктами (Київ = 1, Львів = 11) і визначати результат за відстанню між значенням, отриманим з нейромережі та кодом об'єкта. До

підготовки даних для нейромережі потрібно підходити дуже серйозно. Від цього залежить 90% успіху.

ОСОБЛИВОСТІ НАВЧАННЯ НЕЙРОМЕРЕЖІ

При навчанні нейромережі потрібно врахувати кілька факторів:

1. якщо подавати на вхід суперечливі дані, то мережа може взагалі ніколи нічому і не навчитися. Вона буде не в змозі зрозуміти, чому в одному випадку $2+2=4$, а в другому $2+2=5$. Від суперечливих даних у навчальній і тестовій вибірці слід позбутися.
2. кількість зв'язків між нейронами має бути меншою ніж кількість прикладів в навчальній вибірці. Інакше мережа не навчиться, а "запам'ятає" усі приведені приклади.
3. Якщо занадто довго навчати мережу, те вона може "перенавчитися". Необхідно визначати момент, коли процес буде вважатися завершеним.

В цілому, немає чітких правил як потрібно навчати нейромережу, щоб отримати найкращий результат. Для підбору найкращих параметрів навчання можна використовувати, наприклад, генетичні алгоритми, але вже зовсім інша тема

РОБОТА З СИСТЕМОЮ

Крок 1. Відкрийте базу даних

Виберіть файл із навчальною вибіркою... Інформація, що міститься в цьому файлі, використовується для навчання нейромережі. Можна відкрити txt-файл для навчання, або npw-файл – навчену нейромережу.

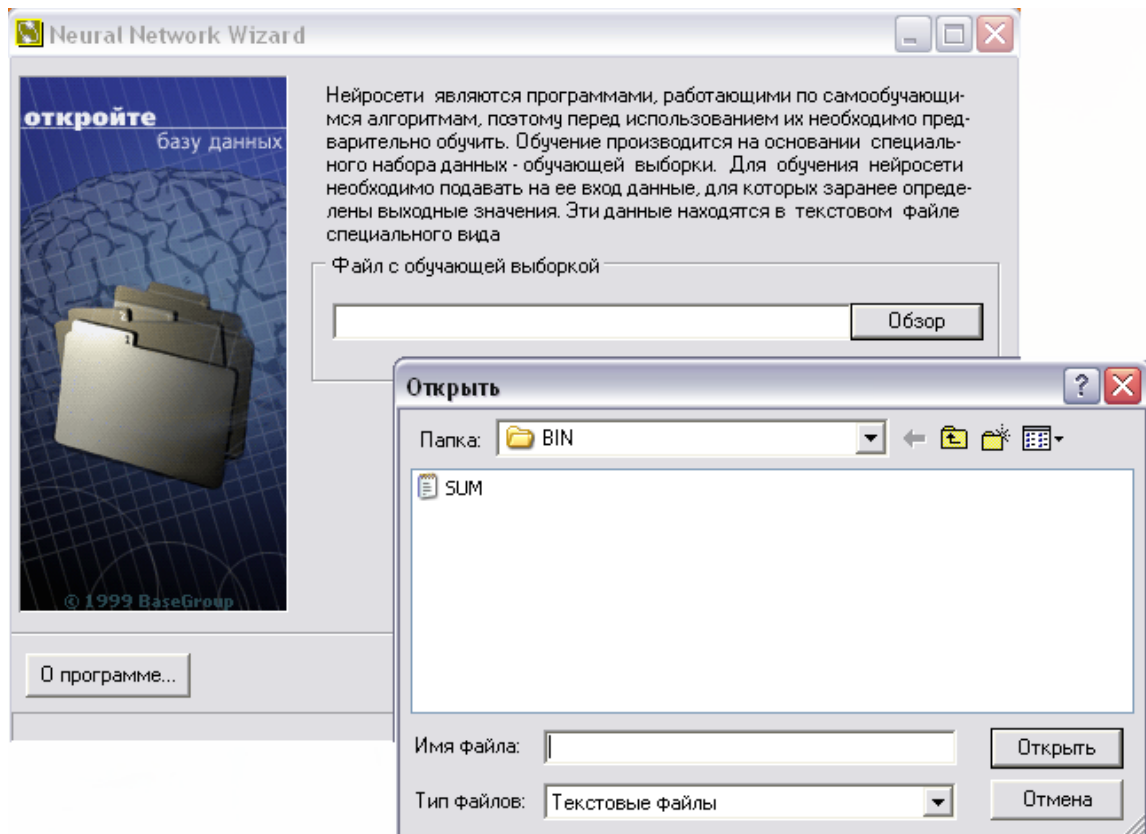


Рис. 1. Вибір файлу з навчальною вибіркою

Крок 2. Задайте поля і їх властивості

Виберіть поле в списку і вкажіть, як його обробляти.

Використовувати поле як...

Нейронна мережа складається з вхідного, вихідного і прихованого прошарків. Кількість нейронів у першому та останньому шарі залежить від того, які поля позначаються як вхідні та вихідні. Поля, що відзначено позначкою "не використовувати" у навчанні і тестуванні нейромережі застосовуватися не будуть.

Нормалізувати поле як...

На вхід нейромережі повинна подаватися інформація в нормалізованому виді. Тобто це числа в діапазоні від 0 до 1. Можна вибрати наступні методи нормалізації.

- $(X - \text{MIN}) / (\text{MAX} - \text{MIN})$ - лінійна нормалізація.
- $1 / (1 + \exp(-ax))$ - експонентна нормалізація.
- Авто $(x - \tilde{x}) / a$, $1 / (1 + \exp(a))$ - нормалізація, що заснована на статистичних характеристиках вибірки

- Без нормалізації - нормалізація не здійснюється

Параметри нормалізації...

Задайте значення, що використовуються у формулах нормалізації.

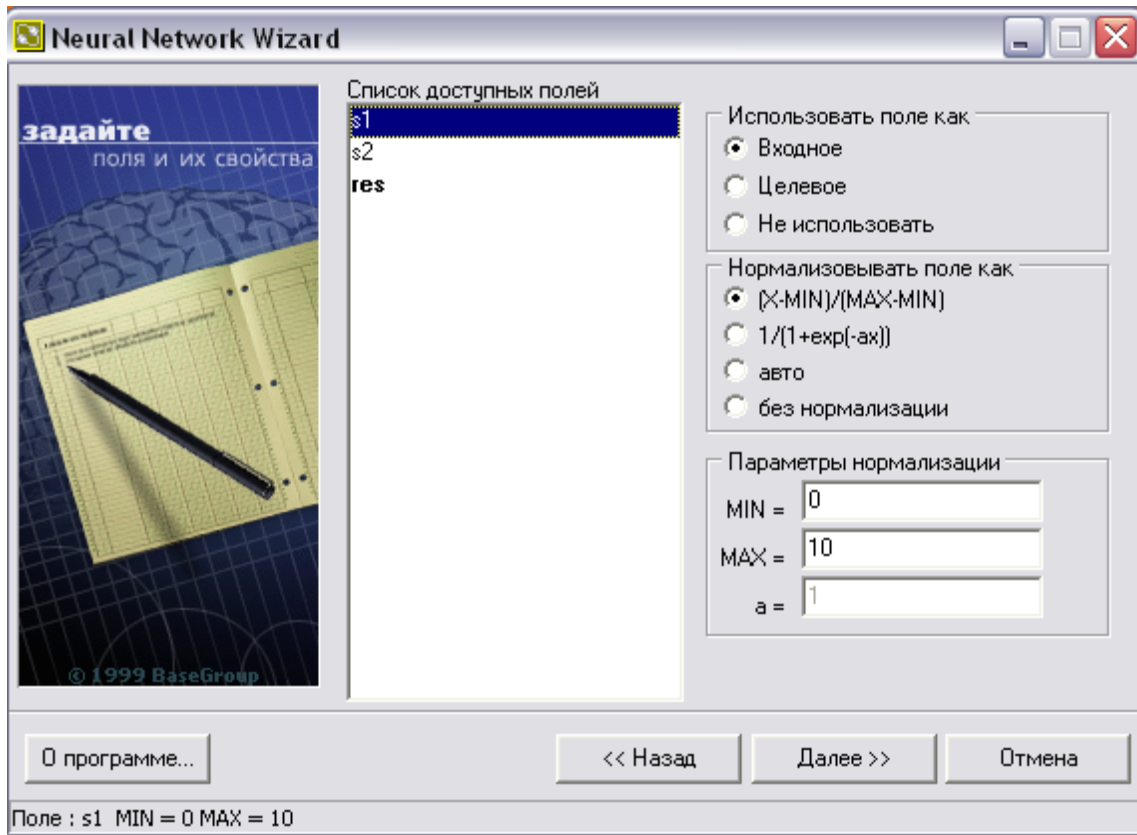


Рис. 2. Список доступних полів та його властивості

Крок 3. Задайте параметр нейромережі

Число прошарків нейромережі...

Нейронна мережа складається з прошарків – вхідного, вихідного і прихованих. Необхідно вказати кількість прихованих прошарків. Загального правила скільки повинно бути таких прошарків немає, зазвичай обирається 1-3 прихованих прошарків. Чим більш нелінійною є задача, тим більше прихованих прошарків повинно бути.

Прощарки, Число нейронів...

В Neural Network Wizard всі елементи попереднього прошарку зв'язані з усіма елементами наступного. Кількість нейронів у першому та останньому прошарках залежить від того, скільки полів вказано як вхідні та вихідні. Кількість нейронів в кожному прихованому прошарку необхідно задавати.

Загальних правил визначення кількості нейронів немає, але необхідно, щоб число зв'язків між нейронами було меншим за кількість прикладів в навчальній вибірці. Інакше нейромережа втратить здатність до узагальнення, а просто "запам'ятає" всі приклади з навчальної вибірки. Тоді при тестуванні на прикладах, наявних у навчальній вибірці вона буде демонструвати прекрасні результати, а на реальних даних – погані.

Параметр сигмоїди...

Сигмоїда застосовується для забезпечення нелінійного перетворення даних. У протилежному випадку, нейромережа може виділяти лише лінійно розділимі множини. Чим вище параметр, тим більше перехідна функція є подібною на порогову. Параметр сигмоїди підбирається емпірично.

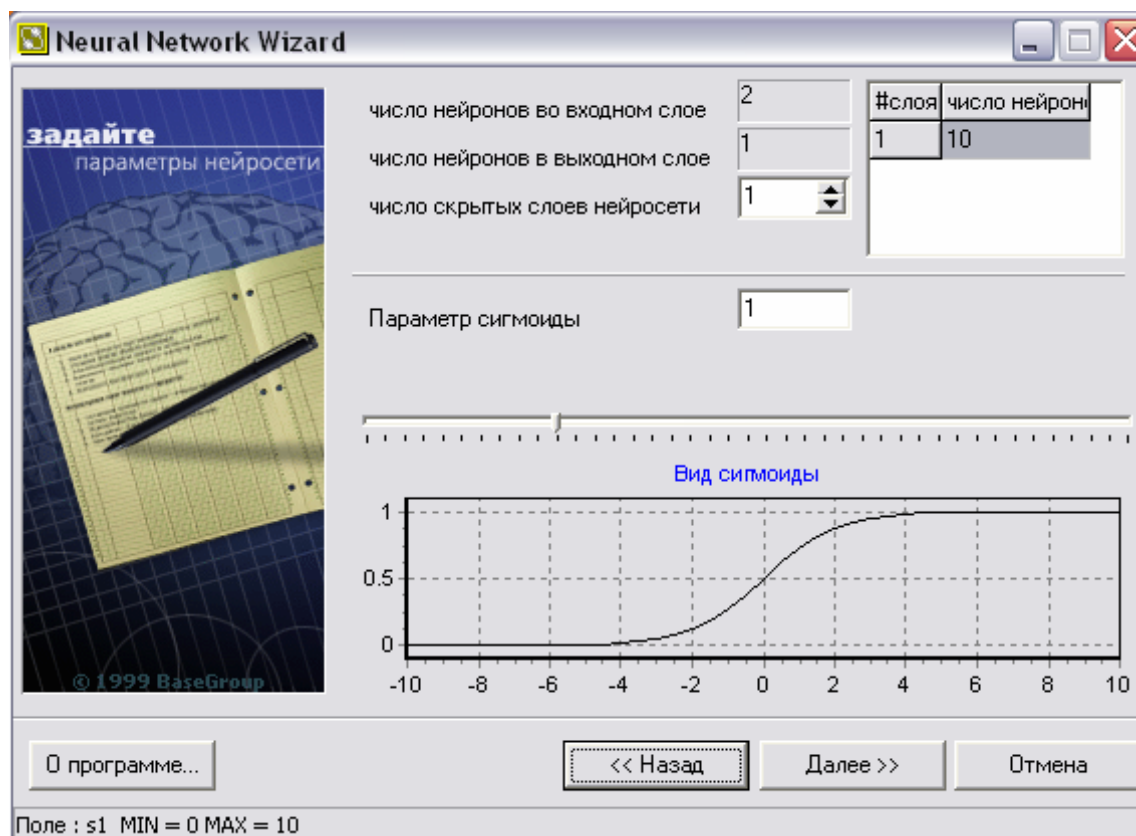


Рис. 3. Вибір параметра сигмоїди та параметрів нейромережі.

Крок 4. Задайте параметр навчання

Використовувати для навчання мережі % вибірки...

Всі приклади, що подаються на вхід нейромережі, поділяються на дві множини – навчальну та тестову. Задайте, скільки відсотків прикладів буде

використано в навчальній вибірці. Записи, що використовуються для тестування, вибираються випадково, але пропорції зберігаються.

Швидкість навчання...

Параметр визначає амплітуду корекції ваг на кожному кроці навчання.

Момент (імпульс)...

Параметр визначає ступінь впливу i -ої корекції ваг на $i+1$ -шу.

Розпізнано, якщо помилка за прикладом $< \dots$

Якщо результат прогнозування відрізняється від значення з навчальної множини ϵ меншим за вказану величини, то приклад вважається розпізнаним.

Використовувати тестову множину як валідаційну...

При встановлення цього прапорця, навчання буде припинено як тільки помилка на тестовій множині почне збільшуватися. Видається відповідне повідомлення. Це допомагає уникнути ситуації перенавчання нейромережі.

Критерії зупинки навчання...

Необхідно визначити момент, коли навчання буде закінчено.

The screenshot shows the 'Neural Network Wizard' window. On the left is a graphic with the text 'задайте параметры обучения' (set training parameters) and '© 1999 BaseGroup'. The main area contains the following settings:

- Использовать для обучения нейросети: 80 % выборки
- Скорость обучения: 0.1
- Момент: 0.9
- Распознано если ошибка по примерам: 0.05
- ☐ Использовать тестовое множество как валидационное

Критерии останковки обучения:

- ☐ Прошло: 10000 эпох
- ☐ Макс. ошибка при обучении <: 0.05
- ☐ Средняя ошибки при обучении <: 0.05
- ☐ Распознано: 80 % обучающей выборки
- ☐ Макс. ошибка при тестировании <: 0.05
- ☐ Средняя ошибки при тестировании <: 0.05
- ☐ Распознано: 80 % тестовой выборки

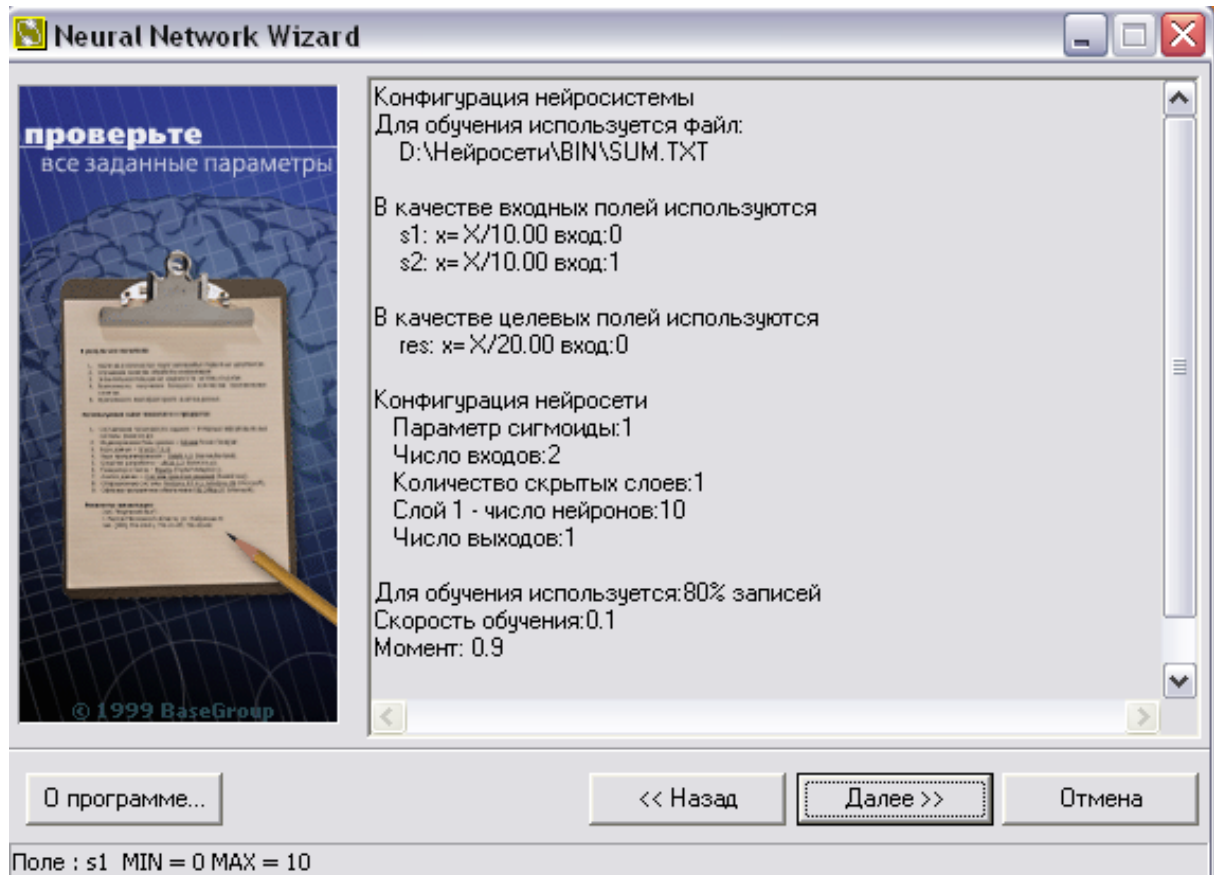
Buttons at the bottom: 'О программе...', '<< Назад', 'Далее >>', 'Отмена'.

Footer: Поле : s1 MIN = 0 MAX = 10

Рис. 4. Визначення параметрів навчання та закінчення процесу навчання

Крок 5. Перевірте всі задані параметри

Переконайтеся, що всі параметри вказано вірно.



**Рис. 5. Перевірка правильності конфігурації мережі
і параметрів навчання**

Крок 6. Запустіть навчання системи

Пуск навчання/зупинка навчання...

Запустіть процес. В таблиці над кнопкою можна спостерігати, як міняється помилка навчання.

Розподіл помилки...

У діаграмі відображається розподіл помилки. Зелені стовпці – помилка на навчальній вибірці, червоні – на тестовій вибірці. Чим правіше стовпець, тим вище значення помилки. Шкала від 0 до 1. Чим вище стовпець, тим більше прикладів із зазначеною помилкою.

Розподіл прикладів у навчальній/тестовій вибірці...

На цих графіках можна відслідковувати наскільки результати, що спрогнозовані нейронною мережею збігаються зі значеннями в навчальній

(ліворуч) і тестовій (праворуч) вибірці. Кожен приклад позначено на графіку точкою. Якщо точка попадає на виділену лінію (діагональ), то нейромережа передбачила результат з досить високою точністю. Якщо точка знаходиться вище діагоналі, значить нейромережа недооцінила результат, нижче – переоцінила. Необхідно домагатися, щоб точки розташовувалися якнайближче до діагоналі.

Крок 7. Розрахуйте кінцевий результат

У наборі вхідних параметрів введіть числа і натисніть на кнопку **"Розрахунок"**. У таблиці «розраховані параметри» висвічується результат. Слід пам'ятати, що не варто перевіряти нейромережу на числах, що виходять за границі навчальної і тестової вибірки. Якщо нейромережа навчена додавати числа в діапазоні від 0 до 10, то і тестувати нейромережу необхідно в тому самому діапазоні.

Якщо результати влаштовують, то натисніть на кнопку **"Зберегти"**. Neural Network Wizard зберігає всі параметри і налаштування у файлі з розширенням nnw.

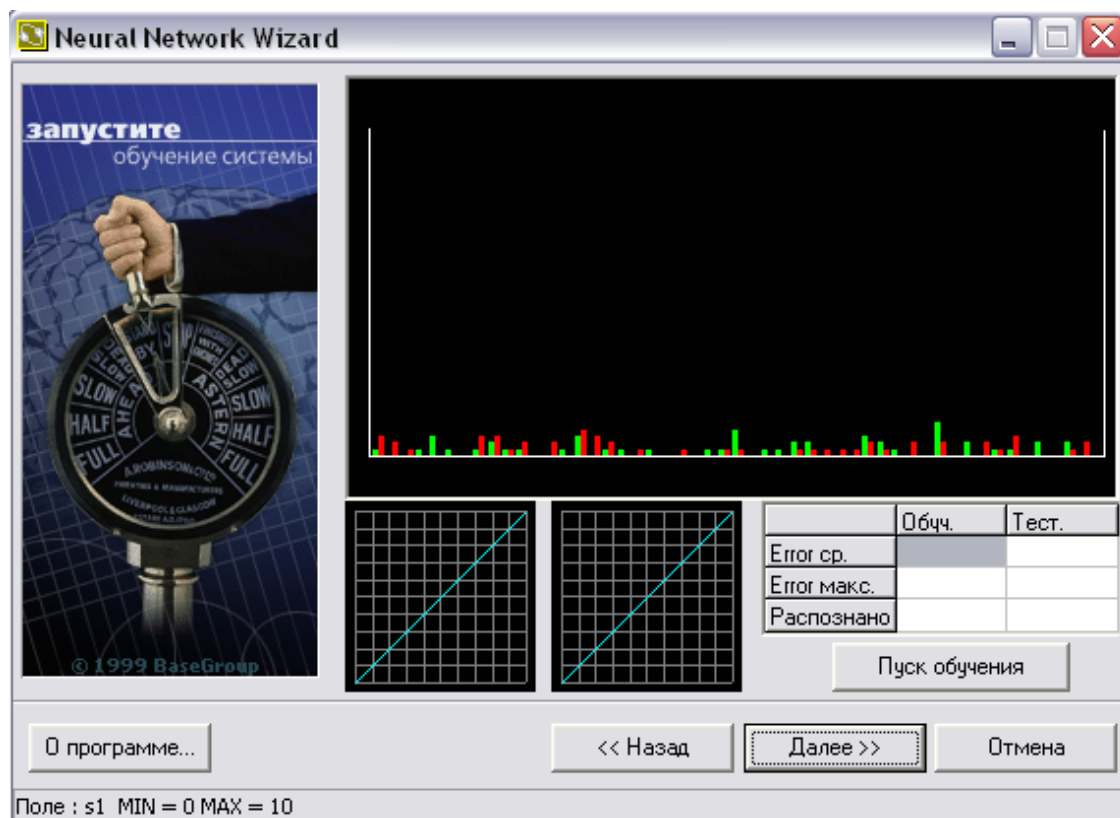


Рис. 6. Перший етап побудови моделі операції додавання. Запуск системи

Рис. 7. Ввід початкових параметрів и розрахунок кінцевих результатів.

Завдання для самостійної роботи

Завдання 1. ПРИКЛАД НАВЧАННЯ НЕЙРОННОЇ МЕРЕЖІ

Для прикладу розглянемо процес навчання нейронної мережі арифметиці, а точніше додаванню двох чисел. Розглянемо рішення цієї проблеми по кроках.

Крок 1. На вхід подаємо наступну інформацію:

S1	S2	RES
0	0	0
1	1	2
1	2	3
2	2	4
3	3	6
4	4	8
2	4	6

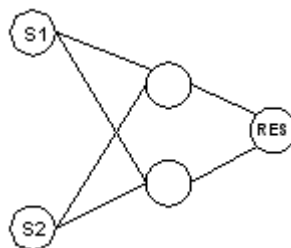
5	5	10
6	6	12
7	7	14
8	8	16
9	9	18
9	10	19
10	9	19
10	10	20
2	3	5
6	1	7
1	5	6
3	8	11
9	7	16
8	9	17

Крок 2. Вказуємо, що поле RES – цільове.

Тобто нейромережа буде намагатися визначити яким чином значення полів S1 і S2 впливають на поле RES.

Крок 3. Визначаємо конфігурацію нейронної мережі.

Задаємо кількість прихованих шарів – 1. Кількість елементів у 1-ому шарі – 2. В результаті маємо наступну конфігурацію



Крок 4. Визначаємо параметри навчання.

Зупинити навчання по проходженні 10000 epoch.

Крок 5. Перевіряємо правильність конфігурації мережі і параметрів навчання.

Крок 6. Запускаємо систему на навчання, під час якого система побудує модель операції додавання.

Крок 7. Після закінчення навчання тестуємо отриману модель.

Вводимо початкові параметри і розраховуємо результат.

Це загальні принципи роботи. На практиці отримати якісний результат на реальних даних не так просто. Для того, щоб отримати якісний результат при використанні нейромереж треба розуміти яка математична модель покладена в її основу і ретельно підходити до підбору статистичної вибірки, на якій нейромережа буде навчатися.

ПОРЯДОК РОБОТИ

1. Запустити нейроемулятор Neural Network Wizard.
2. Уважно ознайомитися з методикою навчання нейронної мережі.
3. Пройти всі кроки тестового завдання і проаналізувати отримані результати.
4. Сформувані власні дані для навчання.
5. Пройти всі кроки навчання на нових даних, але приділити більшої уваги налаштуванню параметрів нейроемулятора.
6. Проаналізувати отримані результати і зробити висновок.

Завдання 2. Розглянути процес навчання нейронної мережі для задач зі свого варіанту

1	$r = x - y$	9	$r = \cos x - y$
2	$r = x * y$	10	$r = \operatorname{tg} x - 2y$
3	$r = \frac{x}{y}$	11	$r = \operatorname{ctg} x + y$
4	$r = \sqrt{x} + y$	12	$r = x^2 - y$
5	$r = \sqrt[3]{x} + 2y$	13	$r = x - y^2$
6	$r = \sin x + y$	14	$r = x^2 - y^2$

7	$r = x + y^2$	15	$r = 2x - y$
8	$r = x^2 + y$		

Обчислити вихід двошарового перцептрону після двох ітерацій навчання за методом зворотного поширення помилки. Кількість входів 2 (із навчальної множини), кількість виходів 1; вагові множники першого і другого прошарків вибрати самостійно (малі додатні значення).

ЗМІСТ ЗВІТУ

1. Назва та мета виконання лабораторної роботи.
2. Можливості нейроемулятора Neural Network Wizard.
3. Особливості навчання нейронної мережі та вплив налаштувань параметрів на вихідні результати.
4. Характеристика основних параметрів налаштування.
5. Аналітичні висновки щодо властивостей програми та отриманих результатів.

Лабораторна робота №2

ТЕМА: РЕКУРЕНТНІ ТА САМООРГАНІЗОВАНІ НЕЙРОННІ МЕРЕЖІ

МЕТА: Отримання й закріплення знань, формування практичних навичок роботи з нейронними мережами зі зворотними зв'язками

Короткі теоретичні відомості

Мережа Хопфілда. Джон Хопфілд вперше представив свою асоціативну мережу у 1982 р. у Національній Академії Наук. На честь Хопфілда та нового підходу до моделювання, ця мережна парадигма згадується як мережа Хопфілда. Мережа базується на аналогії фізики динамічних систем. Початкові застосування для цього виду мережі включали асоціативну, або адресовану за змістом пам'ять та вирішували задачі оптимізації.

Мережа Хопфілда використовує три прошарки: вхідний, прошарок Хопфілда та вихідний прошарок. Кожен прошарок має однакову кількість нейронів. Входи прошарку Хопфілда під'єднані до виходів відповідних нейронів вхідного прошарку через змінні ваги з'єднань. Виходи прошарку

Хопфілда під'єднуються до входів всіх нейронів прошарку Хопфілда, за винятком самого себе, а також до відповідних елементів у вихідному прошарку. В режимі функціонування, мережа скеровує дані з вхідного прошарку через фіксовані ваги з'єднань до прошарку Хопфілда. Прошарок Хопфілда коливається, поки не буде завершена певна кількість циклів, і біжучий стан прошарку передається на вихідний прошарок. Цей стан відповідає образу, вже запрограмованому у мережу.

Навчання мережі Хопфілда вимагає, щоб навчальний образ був представлений на вхідному та вихідному прошарках одночасно. Рекурсивний характер прошарку Хопфілда забезпечує засоби корекції всіх ваг з'єднань. Недвійкова реалізація мережі повинна мати пороговий механізм у передатній функції. Для правильного навчання мережі відповідні пари "вхід-вихід" мають відрізнятися між собою.

Якщо мережа Хопфілда використовується як пам'ять, що адресується за змістом вона має два головних обмеження. По-перше, число образів, що можуть бути збережені та точно відтворені є строго обмеженим. Якщо зберігається занадто багато параметрів, мережа може збігатись до нового неіснуючого образу, відмінному від всіх запрограмованих образів, або не збігатись взагалі. Межа ємності пам'яті для мережі приблизно 15% від числа нейронів у прошарку Хопфілда. Другим обмеженням парадигми є те, що прошарок Хопфілда може стати нестабільним, якщо навчальні приклади є занадто подібними. Зразок образу вважається нестабільним, якщо він застосовується за нульовий час і мережа збігається до деякого іншого образу з навчальної множини. Ця проблема може бути вирішена вибором навчальних прикладів більш ортогональних між собою.

Структурна схема мережі Хопфілда приведена на рис. 1.

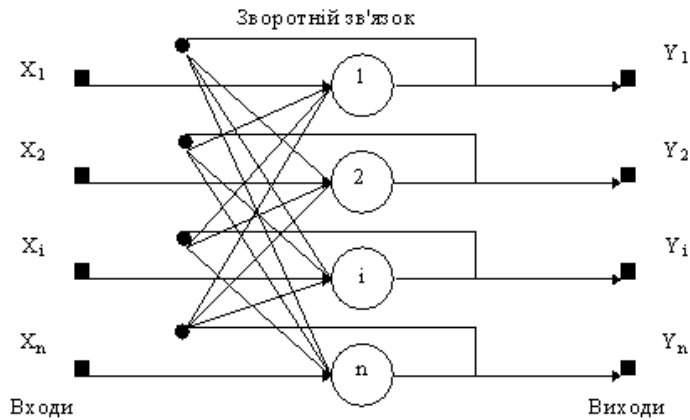


Рис. 1. Структурна схема мережі Хопфілда.

Задача, розв'язувана даною мережею в якості асоціативної пам'яті, як правило, формулюється таким чином. Відомий деякий набір двійкових сигналів (зображень, звукових оцифровок, інших даних, що описують якийсь об'єкти або характеристики процесів), вважають зразковим. Мережа повинна вміти з зашумленого сигналу, поданого на її вхід, виділити ("пригадати" по частковій інформації) відповідний зразок або "дати висновок" про те, що вхідні дані не відповідають жодному із зразків. У загальному випадку, будь-який сигнал може бути описаний вектором x_1, x_i, x_n, \dots , n - число нейронів у мережі і величина вхідних і вихідних векторів. Кожний елемент x_i дорівнює або $+1$, або -1 . Позначимо вектор, що описує k -ий зразок, через X^k , а його компоненти, відповідно, - x_i^k , $k=0, \dots, m-1$, m - число зразків. Якщо мережа розпізнає (або "пригадує") якийсь зразок на основі пред'явлених їй даних, її виходи будуть містити саме його, тобто $Y = X^k$, де Y - вектор вихідних значень мережі: y_1, y_i, y_n . У протилежному випадку, вихідний вектор не співпадає з жодним зразковим.

Якщо, наприклад, сигнали являють собою якесь зображення, то, відобразивши у графічному виді дані з виходу мережі, можна буде побачити картинку, що цілком збігається з однією зі зразкових (у випадку успіху) або ж "вільну імпровізацію" мережі (у випадку невдачі).

Алгоритм функціонування мережі

1. На стадії ініціалізації мережі вагові коефіцієнти синапсів встановлюються таким чином:

$$w_{ij} = \begin{cases} \sum_{k=1}^m x_i^k x_j^k, & i \neq j \\ 0, & i = j \end{cases}$$

Тут i і j - індекси, відповідно, предсинаптичного і постсинаптичного нейронів; x_i^k , x_j^k - i -ий і j -ий елементи вектора k -ого зразка.

2. На входи мережі подається невідомий сигнал (t - номер ітерації). Його поширення безпосередньо встановлює значення виходів:

$$y_i(0) = x_i, \quad i = 0 \dots n-1,$$

тому позначення на схемі мережі вхідних сигналів у явному виді носить чисто умовний характер. Нуль у дужці справа від y_i означає нульову ітерацію в циклі роботи мережі.

3. Розраховується новий стан нейронів

$$S(t+1) = \sum_{j=1}^n w_{ij} y_j(t), \quad j=0 \dots n-1$$

і нові значення виходів

$$y_j(t+1) = f[S_j(t+1)]$$

де f - передатна функція у виді порогової, приведена на рис. 2.

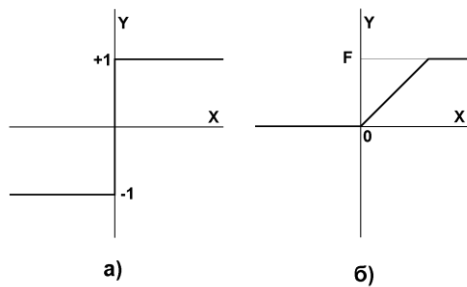


Рис. 2. Передатні функції

4. Перевіряємо чи змінилися вихідні значення виходів за останню ітерацію. Якщо так - перехід до пункту 2, інакше (якщо виходи стабілізувались) - кінець. При цьому вихідний вектор являє собою зразок, що найкраще відповідає вхідним даним.

Мережа Хеммінга є розширенням мережі Хопфілда. Ця мережа була розроблена Річардом Ліппманом (*Richard Lippman*) у середині 80-х рр. Мережа Хемінга реалізує класифікатор, що базується на найменшій похибці для векторів двійкових входів, де похибка визначається відстанню Хемінга.

Відстань Хемінга визначається як число бітів, які відрізняються між двома відповідними вхідними векторами фіксованої довжини. Один вхідний вектор є незашумленим прикладом образу, інший є спотвореним образом. Вектор виходів навчальної множини є вектором класів, до яких належать образи. У режимі навчання вхідні вектори розподіляються до категорій для яких відстань між зразковими вхідними векторами та біжучим вхідним вектором є мінімальною.

Мережа Хеммінга має три прошарки: вхідний прошарок з кількістю вузлів, скільки є окремих двійкових ознак; прошарок категорій (прошарок Хопфілда), з кількістю вузлів, скільки є категорій або класів; вихідний прошарок, який відповідає числу вузлів у прошарку категорій.

Мережа є простою архітектурою прямого поширення з вхідним рівнем, повністю під'єднаним до прошарку категорій. Кожен елемент обробки у прошарку категорій є зворотно під'єднаним до кожного нейрона у тому ж самому прошарку і прямо під'єднаним до вихідного нейрону. Вихід з прошарку категорій до вихідного прошарку формується через конкуренцію.

Навчання мережі Хеммінга є подібним до методології Хопфілда. На вхідний прошарок надходить бажаний навчальний образ, а на виході вихідного прошарку надходить значення бажаного класу, до якого належить вектор. Вихід містить лише значення класу до якої належить вхідний вектор. Рекурсивний характер прошарку Хопфілда забезпечує засоби корекції всіх ваг з'єднань.

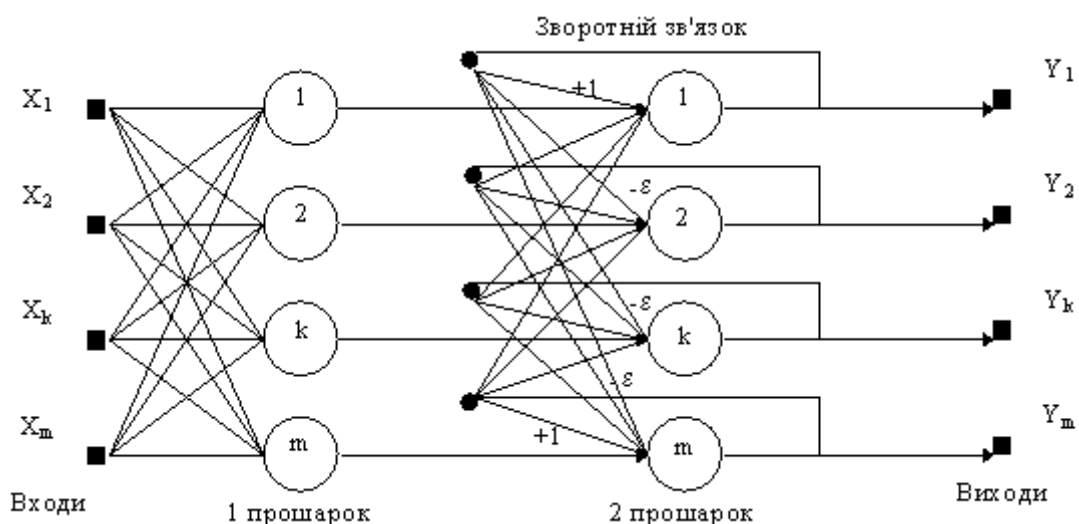


Рис. 3. Структурна схема мережі Хеммінга

Алгоритм функціонування мережі Хеммінга

1. На стадії ініціалізації ваговим коефіцієнтам першого прошарку і порогу передатної функції присвоюються такі значення:

$$W_{ik} = x_I^k / 2, i = 0 \dots n-1, k = 0 \dots m-1$$

$$b_k = n / 2, k = 0 \dots m-1$$

Тут x_i^k - i -ий елемент k -ого зразка.

Вагові коефіцієнти гальмуючих синапсів у другому прошарку беруть рівними деякій величині $0 < e < 1/m$. Синапс нейрона, пов'язаний із його ж виходом має вагу +1.

2. На входи мережі подається невідомий вектор $x_1, x_i, x_n \dots$. Розраховуються стани нейронів першого прошарку (верхній індекс у дужках вказує номер прошарку):

$$y_j^{(1)} = S_j^{(1)} = \sum_{i=1}^n w_{ij} x_i + b_j, j = 0 \dots m-1.$$

Після цього отримання значення ініціалізують значення виходів другого прошарку:

$$y_j^{(2)} = y_j^{(1)}, j = 0 \dots m-1$$

3. Обчислюються нові стани нейронів другого прошарку:

$$S_j^{(2)}(t+1) = y_j(t) - \varepsilon \sum_{k=1}^m y_k^{(2)}(t), k \neq j, j = 1 \dots m$$

і значення їх виходів:

$$y_j^{(1)}(t+1) = f[S_j^{(2)}(t+1)]$$

Передатна функція f має вид порога, причому величина b повинна бути достатньо великою, щоб будь-які можливі значення аргументу не призводили до насичення.

4. Перевіряється, чи змінилися виходи нейронів другого прошарку за останню ітерацію. Якщо так - перейти до кроку 3. Інакше - кінець.

Роль першого прошарку є умовною: скориставшись один раз на першому кроці 1 значеннями його вагових коефіцієнтів, мережа більше не вертається до нього, тому перший прошарок може бути взагалі виключений із мережі.

Мережа Хеммінга має ряд переваг над мережею Хопфілда. Вона реалізує оптимальний класифікатор мінімуму похибки, якщо похибки вхідних бітів є випадковими та незалежними. Для функціонування мережі Хеммінга потрібна менша кількість нейронів, оскільки середній прошарок вимагає лише один нейрон на клас, замість нейрону на кожен вхідний вузол. І, нарешті, мережа Хеммінга не страждає від неправильних класифікацій, які можуть трапитись у мережі Хопфілда. В цілому, мережа Хеммінга є як швидшою, так і точнішою за мережу Хопфілда.

Двоскерована асоціативна пам'ять була розроблена Бартом Козко (*Bart Kosko*) і розширює модель Хопфілда. Множина парних образів навчається за образами, що представлені як біполярні вектори. Подібно до мережі Хопфілда, коли представляється зашумлена версія одного образу, визначається найближчий образ, асоційований з ним.

На рис. 4 показаний приклад двоскерованої асоціативної пам'яті. Вона має стільки входів, скільки є вихідних нейронів. Два приховані прошарки містяться на двох окремих асоціативних елементах пам'яті і представляють подвоєний розмір вхідних векторів. Середні прошарки повністю з'єднуються один з одним. Вхідний та вихідний прошарки потрібні для реалізації засобів введення та відновлення інформації з мережі.

Середні прошарки розроблені для збереження асоційованих пар векторів. Коли зашумлений вектор образу вважається вхідним, середні прошарки коливаються до досягнення стабільного стану рівноваги, який відповідає найближчій навченій асоціації і буде генерувати початковий навчальний образ на виході. Подібно до мережі Хопфілда, двоскерована асоціативна пам'ять є схильною до неправильного відшукування навченого образу, якщо надходить невідомий вхідний вектор, який не був у складі навчальної множини.

Двоскерована асоціативна пам'ять відноситься до гетероасоціативної пам'яті. Вхідний вектор надходить на один набір нейронів, а відповідний вихідний вектор продукується на іншому наборі нейронів. Вхідні образи асоціюються з вихідними.

Для порівняння: мережа Хопфілда є автоасоціативною. Вхідний образ може бути відновлений чи виправлений мережею, але не може бути асоційований з іншим образом. У мережі Хопфілда використовується одношарова структура асоціативної пам'яті, у якій вихідний вектор з'являється на виході тих же нейронів, на які надходить вхідний вектор.

Двоскерована асоціативна пам'ять, як і мережа Хопфілда, здатна до узагальнення, виробляючи правильні вихідні сигнали, незважаючи на спотворені входи.

Розглянемо схему двоскерованої асоціативної пам'яті. Вхідний вектор A обробляється матрицею ваг W мережі, у результаті чого продукується вектор вихідних сигналів мережі B . Вектор B обробляється транспонованою матрицею W^T ваг мережі, яка продукує сигнали, що представляють новий вхідний вектор A . Цей процес повторюється доти, поки мережа не досягне стабільного стану, у якому ні вектор A , ні вектор B не змінюються.

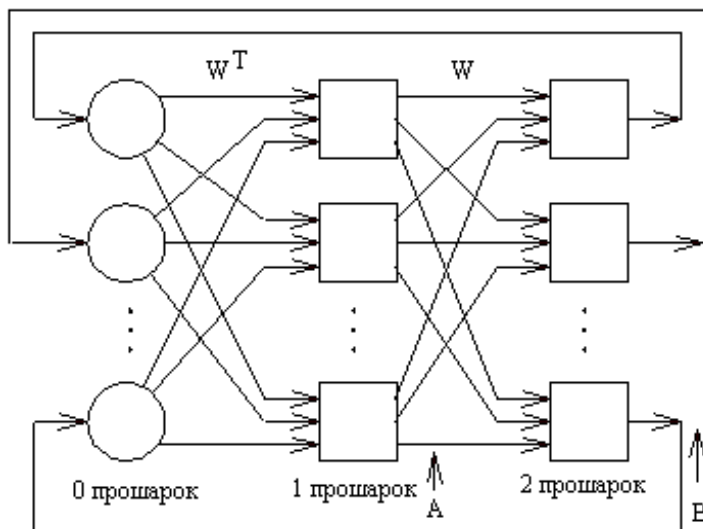


Рис. 4. Двоскерована асоціативна пам'ять

Нейрони в прошарках 1 і 2 функціонують, як і в інших парадигмах, обчислюючи суму зважених входів і значення передатної функції F :

$$b_j = F \sum_i a_i w_{ij}$$

або у векторній формі: $B = F(AW)$

де B - вектор вихідних сигналів нейронів прошарку 2, A - вектор вихідних сигналів нейронів прошарку 1, W - матриця ваг зв'язків між прошарками 1 і 2, F - передатна функція.

Аналогічно $A=F(BW^T)$, де W^T є транспозицією матриці W .

В якості передатної функції використовується експонентна сигмоїда.

Прошарок 0 не робить обчислень і не має пам'яті. Він є лише засобом розподілу вихідних сигналів прошарку 2 до елементів матриці W^T .

Формула для обчислення значень синаптичних ваг:

$$W = \sum_j A_j^T B_j$$

де A_j і B_j - вхідні і вихідні сигнали навчальної вибірки.

Вагова матриця обчислюється як сума добутків всіх векторних пар навчальної вибірки.

Системи зі зворотним зв'язком мають тенденцію до коливань. Вони можуть переходити від стану до стану, ніколи не досягаючи стабільності. Доведено, що двоскерована асоціативна пам'ять безумовно стабільна при будь-яких значеннях ваг мережі.

Області застосування. Асоціативна пам'ять, розпізнавання образів.

Недоліки. Ємність двоскерованої асоціативної пам'яті жорстко обмежена. Якщо n - кількість нейронів у вхідному прошарку, то число векторів, що можуть бути запам'ятовані в мережі не перевищує $L=n/2\log_2 n$. Так, якщо $n=1024$, то мережа здатна запам'ятати не більш 25 образів, кожний з яких повинен бути відновлюваним.

Двоскерована асоціативна пам'ять має деяку непередбачуваність у процесі функціонування, можливі помилкові відповіді.

Переваги:

- У порівнянні з автоасоціативною пам'яттю (наприклад, мережею Хопфілда), двоскерована асоціативна пам'ять дає можливість будувати асоціації між векторами A і B , що у загальному випадку мають різні розмірності. За рахунок таких можливостей гетероасоціативна пам'ять має більш широкий клас застосувань, ніж автоасоціативна пам'ять.

- Процес формування синаптичних ваг простий і швидкий. Мережа швидко збігається в процесі функціонування.

Нейронна мережа Кохонена

Мережа Кохонена – представник класу конкуруючих мереж або мереж, що змагаються, у яких ваги зв'язків змінюються в ході ітераційного процесу виділення нейронів-переможців. Структура цієї нейронної мережі, що має й інші назви – відображення, що самоорганізується, Кохонена, топологічно зберігає перетворення та карти ознак Кохонена, що самоорганізуються, була запропонована Кохоненом у 1982 році. Відмінна риса цієї мережі – відображати вхідну інформацію, зберігаючи відношення сусідніх вхідних елементів, тобто зберігаючи її топологічну структуру. Ця властивість характерна мозку, але використовується лише в деяких нейронних мережах, хоча вона часто буває необхідною при встановленні характеру взаємозв'язків, які людським оком важко вловлюються. У цьому випадку можливе застосування відображення Кохонена, що дозволяє від важко сприйманих людським зором взаємозв'язків елементів перейти, наприклад, до впорядкованого розташування елементів на прямокутній, гексагональній або будь-якій іншій відповідній сітці. Це широко використовується для перетворення багатовимірних вихідних даних в одно- або двовимірні карти ознак Кохонена (або карти Кохонена, що самоорганізуються). Розглянемо інший можливий ефективний випадок застосування перетворення Кохонена. Є безліч об'єктів, частина з яких дуже схожа одна на одну, а інша їх частина не схожа або схожа відносно. Необхідно виконати класифікацію таким чином, щоб було наочно видно, яким чином групуються об'єкти й наскільки вони близькі або далекі один від одного. Можливе також застосування відображення Кохонена для перетворення багатовимірних вихідних даних в одно- або двовимірні "карти ознак Кохонена", які можна розглядати як результат передобробки для інших нейронних мереж.

Мережа має два шари нейронів. Нейрони першого шару сприймають вхідну інформацію у вигляді n -мірних безперервних векторів та передають її A -нейронам, які впорядковані в одно- або двовимірному масиві як елементи

деяких кластерів. Під час процесу самоорганізації при пред'явленні вектора деякого вхідного зображення виділяється А-елемент кластера, що у сенсі мінімуму заданої відстані (наприклад, квадрата евклідової відстані) найбільше відповідає цьому вхідному вектору. Цей виділений або перемігший елемент та його найближчі сусіди (або в термінах топології – покриття елемента) за заданим правилом змінюють свої ваги, щоб ще краще відповідати вхідному вектору, тобто елемент переможець та його покриття в деякому сенсі близькі до вхідного вектора й прагнуть цю близькість збільшити.

Мережа розпізнає кластери в навчальних даних і розподіляє дані до відповідних кластерів. Якщо в наступному мережа зустрічається з набором даних, несхожим ні з одним із відомих зразків, вона відносить його до нового кластеру. Якщо в даних містяться мітки класів, то мережа спроможна вирішувати задачі класифікації. Мережі Кохонена можна використовувати і в задачах, де класи відомі - перевага буде у спроможності мережі виявляти подібність між різноманітними класами.

Мережа Кохонена має всього два прошарки: вхідний і вихідний, що називають самоорганізованою картою. Елементи карти розташовуються в деякому просторі - як правило двовимірному.

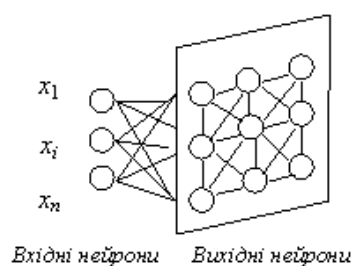


Рис. 5. Мережа Кохонена

Мережа Кохонена навчається методом послідовних наближень. Починаючи з випадковим чином обраного вихідного розташування центрів, алгоритм поступово покращується для кластеризації навчальних даних.

Проте, алгоритм може працювати і на іншому рівні. В результаті ітеративної процедури навчання мережа організовується таким чином, що елементи, які відповідають центрам, розташованим близько один від одного в просторі входів, будуть розташовані близько один від одного і на топологічній карті. Топологічний прошарок мережі можна уявити як двовимірну штахету, яку

потрібно так відобразити в N -вимірний простір входів, щоб по можливості зберегти вихідну структуру даних. Звісно ж, при будь-якій спробі відтворити N -вимірний простір на площині буде загублено багато деталей, але такий прийом дозволяє користувачу візуалізувати дані, що неможливо зрозуміти іншим засобом.

Основний ітераційний алгоритм Кохонена послідовно проходить ряд епох, на кожній епосі опрацьовується один навчальний приклад. Вхідні сигнали - вектори дійсних чисел - послідовно пред'являються мережі. Бажані вихідні сигнали не визначаються. Після пред'явлення достатнього числа вхідних векторів, синаптичні ваги мережі визначають кластери. Крім того, ваги організуються так, що топологічне близькі вузли чуттєві до схожих вхідних сигналів.

Для реалізації алгоритму необхідно визначити міру сусідства нейронів (окіл нейрона-переможця). На мал. 6 показані зони топологічного сусідства нейронів на карті ознак у різні моменти часу. $NE_j(t)$ - множина нейронів, що вважаються сусідами нейрона j у момент часу t . Зони сусідства зменшуються з часом.

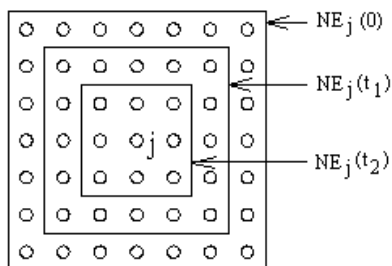


Рис.6.Зони топологічного сусідства на карті ознак у

різні моменти часу

Алгоритм функціонування мережі Кохонена:

1. Ініціалізація мережі. Ваговим коефіцієнтам мережі надаються малі випадкові значення. Початкова зона сусідства показана на рис. 6.
2. Пред'явлення мережі нового вхідного сигналу.
3. Обчислення відстані до всіх нейронів мережі:

Відстані d_j від вхідного сигналу до кожного нейрона j визначаються за формулою:

$$d_j = \sum_{i=1}^N (x_i(t) - w_{ij}(t))^2,$$

де x_i - i -ий елемент вхідного сигналу в момент часу t , $w_{ij}(t)$ - вага зв'язку від i -го елемента вхідного сигналу до нейрона j у момент часу t .

4. Вибір нейрона з найменшою відстанню:

Вибирається нейрон-переможець j^* , для якого відстань d_j найменше.

5. Налаштування ваг нейрона j^* і його сусідів:

Робиться налаштування ваг для нейрона j^* і всіх нейронів з його околу NE. Нові значення ваг:

$$w_{ij}(t+1) = w_{ij}(t) + r(t)(x_i(t) - w_{ij}(t))$$

де $r(t)$ - швидкість навчання, що зменшується з часом (додатне число, менше одиниці).

6. Повернення до кроку 2.

В алгоритмі використовується коефіцієнт швидкості навчання, який поступово зменшується, для тонкішої корекції на новій епосі. В результаті позиція центру встановлюється в певній позиції, яка задовільним чином кластеризує приклади, для яких даний нейрон є переможцем.

Властивість топологічної впорядкованості досягається в алгоритмі за допомогою використання поняття околу. Окіл - це декілька нейронів, що оточують нейрон-переможець. Відповідно до швидкості навчання, розмір околу поступово зменшується, так, що спочатку до нього належить досить велике число нейронів (можливо вся карта), на самих останніх етапах окіл стає нульовим і складається лише з нейрона-переможця. В алгоритмі навчання корекція застосовується не тільки до нейрона-переможця, але і до всіх нейронів з його поточного околу. В результаті такої зміни околу, початкові доволі великі ділянки мережі мігрують в бік навчальних прикладів. Мережа формує грубу структуру топологічного порядку, при якій схожі приклади активують групи нейронів, що близько знаходяться на топологічній карті. З кожною новою епохою швидкість навчання і розмір околу зменшуються, тим самим всередині ділянок карти виявляються більш тонкі розходження, що зрештою призводить до точнішого налаштування кожного нейрона. Часто навчання зумисне розбивають на дві фази: більш коротку, з великою швидкістю навчання і

великих околів, і більш тривалу з малою швидкістю навчання і нульовими або майже нульовими околами.

Після того, як мережа навчена розпізнаванню структури даних, її можна використовувати як засіб візуалізації при аналізі даних.

Області застосування. Кластерний аналіз, розпізнавання образів, класифікація.

Недоліки. Мережа може бути використана для кластерного аналізу тільки в тому випадку, якщо заздалегідь відоме число кластерів.

Переваги. Мережа Кохонена здатна функціонувати в умовах перешкод, тому що число кластерів фіксоване, ваги модифікуються повільно, налаштування ваг закінчується після навчання.

Модифікації. Одна з модифікацій полягає в тому, що до мережі Кохонена додається мережа MAXNET, що визначає нейрон з найменшою відстанню до вхідного сигналу.

Завдання для самостійної роботи

Варіант 1

1. Навчити дискретну мережу Хопфілда розпізнаванню різних букв вашого імені. Обґрунтувати вибір: • числа нейронів мережі; • алгоритму навчання мережі.
2. Дослідити можливості мережі щодо розпізнавання перекручених або зашумлених зображень, різної кількості зображень.

Варіант 2

1. Навчити нейронну мережу ДАП розпізнавання асоціацій на основі букв вашого імені. Обґрунтувати вибір: • числа нейронів мережі; • алгоритму навчання мережі.
2. Дослідити працездатність мережі ДАП при пред'явленні на її входи перекручених зображень, різної кількості зображень.

Варіант 3

1. Розробити нейронну мережу Хеммінга, що зможе розпізнавати не менше 6 різних букв вашого імені та вашого прізвища. При цьому необхідно

обґрунтувати вибір: • числа нейронів кожного шару; • види функцій активації нейронів кожного шару; • величини ваг зв'язків мережі Хеммінга.

2. Навчити нейронну мережу еталонним зображенням букв.
3. Дослідити можливості мережі з розпізнавання перекручених зображень.
4. Підібрати вхідне зображення, рівновіддалене за відстанню Хеммінга від двох еталонних зображень. Яка реакція при пред'явленні цього зображення?

Варіант 4

1. Виконати кластеризацію дев'яти двійкових векторів за допомогою нейронної мережі Кохонена на три класи. Двійкові вектори одержати в такий спосіб. Записати своє прізвище, ім'я та по батькові. Вибрати із цих трьох слів слово з найбільшим числом букв (якщо два або всі слова мають однакове найбільше число букв, то вибирається одне з них). Закодувати голосні й приголосні букви цього слова відповідно одиницями й нулями. Отриманий двійковий код стане першим вектором з дев'яти векторів, кластеризацію яких необхідно виконати за допомогою нейронної мережі Кохонена. Другий і третій вектори одержати з вектора 1 шляхом інверсії відповідно другого й третього розряду вектора 1. Потім аналогічним чином закодувати нулями й одиницями букви наступних двох слів. Якщо отримані двійкові вектори 4 і 5 коротші від векторів 1, 2, 3, то їхню довжину слід збільшити до довжини цих векторів, додаючи ліворуч або праворуч необхідне число двійкових розрядів. Вектори 6, 7 й 8, 9 одержати відповідно з векторів 4, 5 шляхом інверсії відповідно другого й третього розрядів цих векторів.
2. Розробити архітектуру нейронної мережі для ваших вхідних векторів.
3. Обґрунтувати попередній вибір параметрів $r(t)$ і NE, ваг зв'язків нейронної мережі для вашого випадку.
4. Дослідити працездатність мережі при різних значеннях $r(t)$.

Лабораторна робота №3

АЛГОРИТМ РОЮ ЧАСТИНОК

Мета роботи – дослідити особливості застосування алгоритму рою частинок до розв'язання задачі пошуку екстремуму функцій багатьох змінних, виконати аналіз швидкості збіжності й якості результату в залежності від параметрів алгоритму.

Алгоритм рою частинок (Particle Swarm Optimization - PSO) є однією з метаевристик, використовуваних для розв'язку оптимізаційних задач.

Нехай $A \in \mathbf{R}^n$ є простір пошуку й $f: A \rightarrow Y \in \mathbf{R}$ – цільова функція. Pso-Алгоритм заснований на множині (популяції) потенційних розв'язків задачі, які досліджуються в просторі пошуку одночасно. Популяція називається роєм, а її члени – частинками. Рій визначається як множина:

$$S = \{x_1, x_2, \dots, x_n\},$$

N частинок (розв'язків-кандидатів), кожне з яких є:

$$x_i = (x_{i1}, x_{i2}, \dots, x_{in})^T \in A, i = 1, 2, \dots, N.$$

Причому N є параметром алгоритму. Цільова функція $f(x)$ визначена на всіх точках A . Отже, кожна із частинок представляє єдине значення функції,

$$f_i = f(x_i) \in Y.$$

Частки переміщуються в просторі A ітеративно. Це здійснюється шляхом зміни її позиції за допомогою зсуву, який називають швидкістю:

$$v_i = (v_{i1}, v_{i2}, \dots, v_{in})^T, i = 1, 2, \dots, N.$$

Швидкість змінюється ітеративно, забезпечуючи частці потенційну можливість відвідати будь-яку область A . Нехай t позначає номер ітерації, тоді поточна позиція i -ї частки i її швидкість є $x_i(t)$ і $v_i(t)$ відповідно.

Швидкість змінюється на основі інформації, отриманої на попередньому кроці алгоритму. Це забезпечується тим, що запам'ятовує *кращу позицію*, яка була отримана в результаті попереднього пошуку. Для цього, крім рою S , який містить поточні позиції частинок, Pso-Алгоритм зберігає також множину:

$$P = \{p_1, p_2, \dots, p_n\},$$

кращих позицій: $p_i = (p_{i1}, p_{i2}, \dots, p_{in})^T \in A, i = 1, 2, \dots, N$, будь-коли відвіданих кожною частиною. Позначимо ці позиції $p_i(t) = f_i(t)$, де t є номер ітерації.

Оскільки в основі Pso-Алгоритму лежить модель соціально поведінки, повинен існувати механізм, що дозволяє частинкам обмінюватися інформацією. Зокрема, найважливішою інформацією є найкраща позиція із усіх відвіданих усіма частинками. Нехай g є індекс найкращої позиції з найменшим (у випадку завдання мінімізації) значенням цільової функції в P на даній ітерації t для всього рою чосток $p_{gi}(t)$, $p_{ij}(t)$ найкраща позиція для даної частки .

Тоді найпростіша версія Pso-Алгоритму визначається виразом

$$v_{ij}(t+1) = v_{ij}(t) + c_1 R_1(p_{ij}(t) - x_{ij}(t)) + c_2 R_2(p_{gj}(t) - x_{ij}(t)), \quad (1)$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1), \quad (2)$$

Псевдокод операції Pso-Алгоритму

Вхід: Число часток, N ; рою, S ; краці позиції, P .

Крок 1. Множина $T \leftarrow 1$.

Крок 2. Ініціалізація S і множини $P = S$.

Крок 3. Оцінка S і P , а також визначити індекс g кращої позиції.

Крок 4. While (критерії завершення не виконані)

Крок 5. Оновлення S з використанням рівнянь (1) і (2).

Крок 6. Оцініть S .

Крок 7. Оновлення P і перегляд індексу g .

Крок 8. Множина $T \leftarrow T + 1$.

Крок 9. End While

Крок 10. Вивід кращого знайденого розв'язку.

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

де t – номер ітерації; R_1 і R_2 – випадкові величини рівномірно розподілені на інтервалі $[0,1]$; c_1, c_2 – вагові множники, які називають когнітивним (*cognitive*) і соціальним (*social*) параметрами відповідно. На кожній ітерації після модифікації й обчислень цільової функції частинок, модифікується й найкраща позиція. Таким чином, нова найкраща позиція x_i на ітерації $t+1$ є:

$$p_i(t+1) = \begin{cases} x_i(t+1), & \text{if } f(x_i(t+1)) \leq f(p_i(t)), \\ p_i(t), & \text{otherwise.} \end{cases} \quad (3)$$

Визначення нового значення індексу g завершує ітерацію Pso-Алгоритму. Звичайно частки ініціалізуються випадковим чином у просторі A . Значення швидкості $v_{ij}(t)$ у правій частині виразу (1) являє собою інерцію частки, яка враховує переміщення частки на попередній ітерації. Це запобігає надмірному зсуву частки в напрямку найкращої позиції p_{ij} , тим самим, перешкоджає “застрягненню” алгоритму в локальному мінімумі.

Крім того, інерція служить як якийсь збурення для найкращої частки x_g . Насправді, якщо частинка x_i знаходить нову позицію з меншим значенням цільової функції, вона тим самим стає найкращою (тобто, $g \leftarrow i$) і її краща позиція p_i збігається з p_g і x_i на наступній ітерації. Як наслідок два стохастичних доданки в (1) перетворюються в 0. Під час відсутності інерції в (1) згадана частинка залишалася б в одній і тій же позиції протягом декількох ітерацій, поки нова найкраща частинка не була б знайдена іншою частиною.

Значення множників c_1 і c_2 впливають на пошукові можливості алгоритму, забезпечуючи зсув частки x_i у напрямку найкращих позицій p_i і p_g відповідно й змінюючи при цьому амплітуду пошуку. Очевидно, що амплітуда пошуку суттєво різниться у двох випадках. Якщо потрібно більш ретельний глобальний пошук, більші значення c_1 і c_2 забезпечують нові точки в більш далеких областях простору пошуку. З іншого боку, більш ретельний пошук у районі кращої позиції вимагає менших значень цих параметрів. Крім того, вибір, $c_1 > c_2$ забезпечить зсув напрямку пошуку в напрямку p_i , у той час як $c_1 < c_2$ – у напрямку p_g . Цей ефект може бути корисним, якщо є якась додаткова інформація про форму цільової функції. Наприклад, у випадку опуклої унімодальної функції, вибір значень параметрів, який зміщує пошук у напрямку p_g , у комбінації з відповідним значенням амплітуди пошуку буде більш ефективним.

Як правило, слід розглядати тільки позиції частки, які лежать у допустимих межах простору пошуку A . Якщо частинка переміщується за межі простору пошуку після застосування виразу (2), її слід повернути на границю. У найпростішому випадку простір пошуку може бути визначений як гіперкуб:

$$A = [a_1, b_1] \times [a_2, b_2] \times \dots \times [a_n, b_n],$$

де $a_i, b_i \in \mathbb{R}, i = 1, 2, \dots, n$

$$x_{ij}(t+1) = a_j, \text{ якщо } x_{ij}(t+1) < a_j \text{ й} \quad (4)$$

$$x_{ij}(t+1) = b_j, \text{ якщо } x_{ij}(t+1) > b_j$$

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

Модифікації Pso-Алгоритму

По-перше слід зазначити ефект, який називається вибух рою (*swarm explosion*), який означає неконтрольований ріст амплітуди швидкості. Для його запобігання вводиться механізм обмеження амплітуди швидкості за допомогою додаткового параметра алгоритму $v_{max} > 0$. Після визначення нового значення швидкості відповідно до (1), перед використанням (2), вводиться обмеження

$$|v_{ij}(t+1)| \leq v_{max}, i = 1, 2, \dots, N, j = 1, 2, \dots, n \quad (5)$$

У випадку порушення цього обмеження швидкість приводиться до її найближчої границі

$$v_{ij}(t+1) = v_{max}, \text{ якщо } v_{ij}(t+1) > v_{max} \text{ й}$$

$$v_{ij}(t+1) = -v_{max}, \text{ якщо } v_{ij}(t+1) < -v_{max}.$$

Звичайне значення v_{max} вибирається як якась частина простору пошуку по одній з координат. Якщо простір пошуку заданий як в (4) v_{max} може бути визначене як

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

або обмеження швидкості може бути введене для кожної з координат

$$v_{max} = \frac{b_i - a_i}{k}, i = 1, 2, \dots, n,$$

де k звичайно рівно 2. Обмеження швидкості вирішує проблему вибуху рою, однак залишається ще проблема збіжності алгоритму.

Мова йде про здатність рою концентрувати свої частки в районі найбільш перспективних позицій на останній стадії оптимізації викликана неможливістю контролювати швидкість.

Більш ретельний пошук у районі найбільш перспективних позицій вимагає більш сильного “притягання” до них частинок рою. Цього можна досягти зменшенням зміщення частинок від перспективних позицій, яке досягається введенням вагового множника інерції в (1). За рахунок цього вплив попереднього значення швидкості зменшується. Цей множник w називають вагою інерції. У такий спосіб маємо новий варіант виразу (1)

$$v_{ij}(t+1) = wv_{ij}(t) + c_1 R_1(p_{ij}(t) - x_{ij}(t)) + c_2 R_2(p_g(t) - x_{ij}(t)) \quad (6)$$

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

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

Вагу інерції слід вибирати таким чином, щоб ефект впливу $v_{ij}(t)$ зменшувався протягом виконання алгоритму. Звичайно в якості початкового значення w вибирають величину, трохи більшу 1 (наприклад, 1.2), щоб забезпечити більш інтенсивний пошук на початковій стадії. Нижньою границею w може служити, наприклад, 0.1. У ході виконання алгоритму величина w спадає лінійно

$$w(t) = w_H - (w_H - w_\epsilon) \frac{1}{T_{\max}}$$

де t – номер ітерації; w_ϵ і w_H – верхня й нижня границя w ; T_{\max} – максимальне число ітерацій.

ЗАВДАННЯ НА ДЛЯ САМОСТІЙНОЇ РОБОТИ

1. Написати код, що реалізує Pso-Алгоритм.
2. З його допомогою знайти екстремуми наступних функцій

№	Інтервал	Функція	Глобальний екстремум
1	$[-2.048, 2.048]$	$\max F = 3905.93 - 100(x_1^2 - x_2^2 - x_1^2)$	$X(1; 1)$ $F = 3905.93$
2	$[-2, 2]$	$\min F = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \cdot [30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$	$X(0; -1)$ $F = 3$

3	$\mathbb{I}5, 10\mathbb{I}$	$\min F = a(x_2 - bx_1^2 + cx_1 - d)^2 + e(1 - f)\cos x_1 + e,$ $a = 1, b = \frac{5.1}{4}\left(\frac{7}{22}\right)^2, c = \frac{35}{22}, d = 6, e = 10, f = \frac{1}{8} \cdot \frac{7}{22}$	$X(-\frac{22}{7}; 12.275)$ $X(\frac{22}{7}; 12.275)$ $X(\frac{66}{7}; 12.475)$ $F = 0.3977272$
4	$\mathbb{I}, 10\mathbb{I}$	$\min F = (x_1 - x_2)^2 + ((x_1 + x_2 - 10)/3)^2$	$X(5; 5)$ $F = 0$
5	$\mathbb{I}1.2, 1.2\mathbb{I}$ $\mathbb{I}10, 10\mathbb{I}$	$\min F = 100(x_1^2 - x_2)^2 + (1 - x_1)^2$	$X(1; 1)$ $F = 0$
6	$\mathbb{I}1.2, 1.2\mathbb{I}$	$\min F = \sum_{i=1}^3 (100(x_{i+1}^2 - x_i)^2 + (1 - x_i)^2)$	$X(1; 1; 1; 1)$ $F = 0$
7	$\mathbb{I}5.12, 5.12\mathbb{I}$	$\min F = \sum_{i=1}^6 x_i^2$	$X(0; 0; 0; 0; 0; 0)$ $F = 0$
8	$\mathbb{I}1.28, 1.28\mathbb{I}$	$\max F = \frac{100}{100(x_1^2 - x_2) + (1 - x_1)^2 + 1}$	
9	$\mathbb{I}5.12, 5.12\mathbb{I}$	$\max F = 20 + x_1^2 + x_2^2 - 10\cos 2\pi x_1 - 10\cos 2\pi x_2$	
10	$\mathbb{I}5.12, 5.12\mathbb{I}$	$\max F = \sum_1^5 [x_i]$	
11	$\mathbb{I}, \pi\mathbb{I}$	$\max F = \sqrt{3}\cos x_1 + \sin x_2$	
12	$\mathbb{I}5, 5\mathbb{I}$	$\max F = \sum_{i=1}^5 x_i^2$	
13	$\mathbb{I}1, 1\mathbb{I}$	$\min F = 10 + x_1^2 + x_2^2 - 10\cos 2\pi x_1 - 10\cos 2\pi x_2$	
14	$\mathbb{I}1.2, 1.2\mathbb{I}$	$\max F = 10(x_1^2 - x_2)^2 + (1 - x_1)^2$	
15	$\mathbb{I}1.28, 1.28\mathbb{I}$	$\min F = \frac{100}{100(x_1^2 - x_2) + (1 - x_1)^2 + 1}$	
16	$\mathbb{I}5.12, 5.12\mathbb{I}$	$\min F = \sum_{i=1}^2 x_i^2 + 2$	
17	$\mathbb{I}5, 5\mathbb{I}$	$\max F = -\sum_{i=1}^2 x_i^2 + 3$	

3. Побудувати графіки залежності швидкості збіжності алгоритму від розміру популяції.

Розглянемо приклад, який допоможе прояснити процес оновлення. Припустимо, потрібно знайти мінімум функції $3 + x_0^2 + x_1^2$. Графік цієї функції наведено на рис. 1. Основа куба на рис. 1 являє значення x_0 і x_1 , а вертикальна вісь - значення функції. Зауважимо, що поверхня графіка мінімізується при $f = 3$, коли $x_0 = 0$ і $x_1 = 0$.

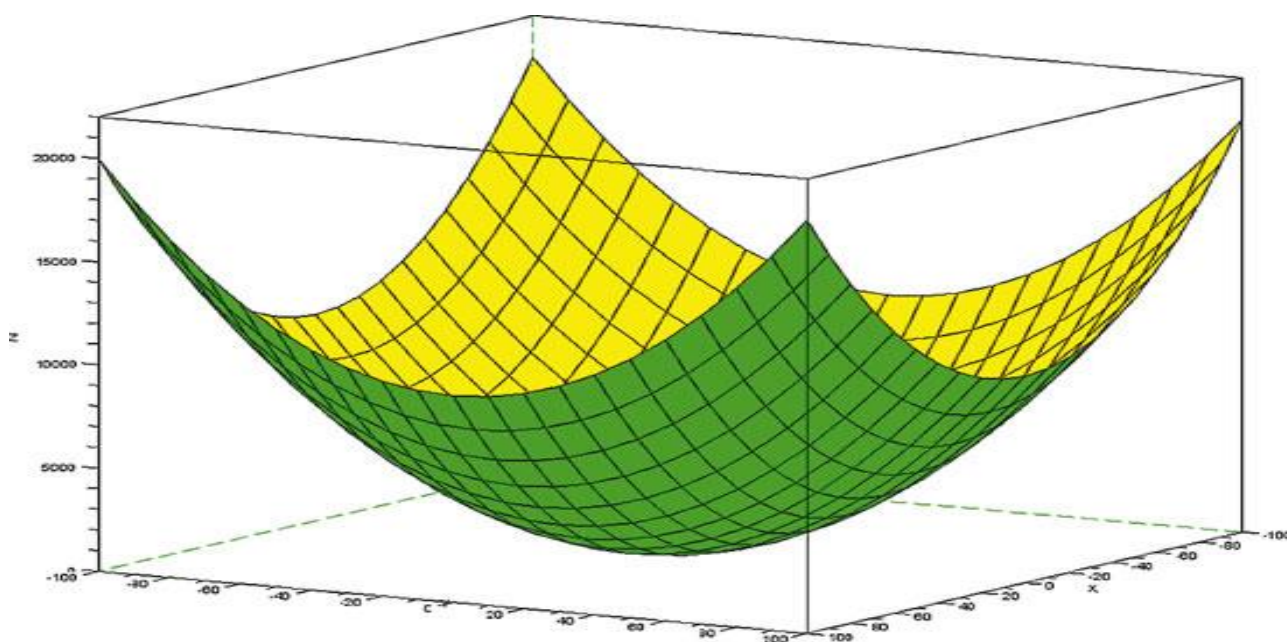


Рис. 1. Графік $f = 3 + x_0^2 + x_1^2$

Нехай поточна позиція частинки, $x(t)$, дорівнює $\{x_0, x_1\} = \{3,0, 4,0\}$, а її поточна швидкість, $v(t)$, - $\{-1,0, -1,5\}$. Також припустимо, що константа $w = 0,7$, константа $c_1 = 1,4$, константа $c_2 = 1,4$, а випадкові числа r_1 і r_2 відповідно рівні $0,5$ і $0,6$. Нарешті, припустимо, що краща відома позиція частинки - $p(t) = \{2,5, 3,6\}$ і глобальна краща відома позиція будь-якої частинки в рої - $g(t) = \{2,3, 3,4\}$. Тоді значення нової швидкості і позиції будуть наступними

$$v(t+1) = (0.7 * \{-1.0, -1.5\}) + (1.4 * 0.5 * \{2.5, 3.6\} - \{3.0, 4.0\}) + (1.4 * 0.6 * \{2.3, 3.4\} - \{3.0, 4.0\})$$

$$= \{-0.70, -1.05\} + \{-0.35, -0.28\} + \{-0.59, -0.50\} = \{-1.64, -1.83\}$$

$$x(t+1) = \{3.0, 4.0\} + \{-1.64, -1.83\} = \{1.36, 2.17\}.$$

Оптимальний розв'язок: $\{x_0, x_1\} = (0,0, 0,0)$. Зауважимо, що процес оновлення поліпшив колишню позицію / розв'язок з $\{3,0, 4,0\}$ до $\{1,36, 2,17\}$.

Видно, що нова швидкість дорівнює старій (помноженій на вагову частку) плюс множник, який залежить від кращої відомої позиції частинки, плюс інший множник, який залежить від кращої відомої позиції всіх часток в рої. Отже, нова позиція частинки зміщується в більш оптимальну на основі кращих відомих позицій даної частинки і всіх частинок. Графік на рис. 2 показує рух однієї з частинок протягом перших восьми ітерацій, виконаних програмою PSO. Частка починає з $x_0=100,0$, $x_1=80,4$ і рухається до оптимального розв'язку $x_0 = 0$, $x_1 = 0$. Спіральний рух типово для PSO.

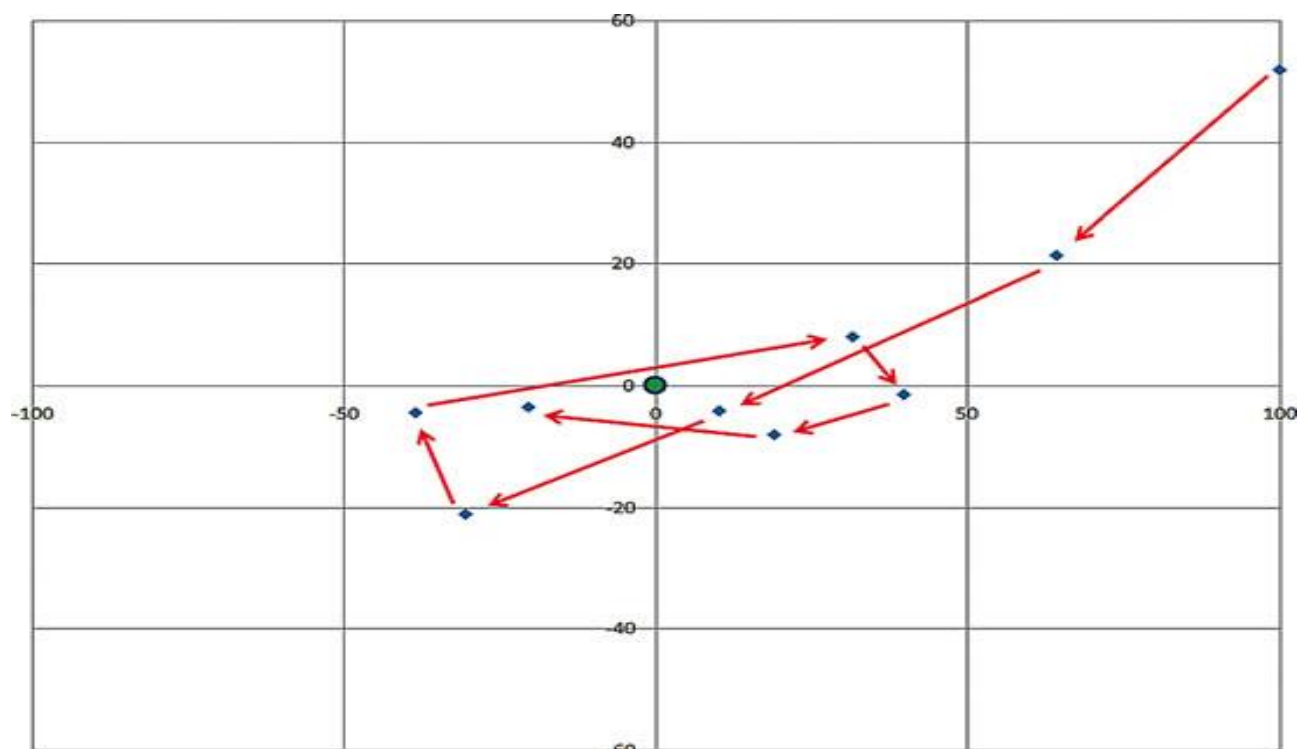


Рис. 2. Рух частинки до оптимального розв'язку

ЛАБОРАТОРНА РОБОТА №4

ТЕМА: ОСНОВИ РОБОТИ В СЕРЕДОВИЩІ SWI Prolog.

ОСНОВНІ СТРАТЕГІЇ РОЗВ'ЯЗУВАННЯ ЗАДАЧ. АЛГОРИМИ ПОШУКУ РОЗВ'ЯЗКІВ

Короткі теоретичні відомості.

SWI-Prolog-це вільна(відкрита) реалізація мови програмування Prolog, часто використовується для викладання та програм Semantic Web. Ця реалізація представляє багатий набір можливостей, бібліотеки для constraint logic programming, багатопоточності, юніт-тестування, GUI, інтерфейс до мови програмування Java, ODBC і т. д., підтримує літературне програмування, містить реалізацію веб-сервера, бібліотеки для SGML, RDF, RDFS, засоби розробника (включаючи IDE з графічними відладчиком і профілювальником), і обширну документацію. SWI-Prolog працює на платформах Unix, Windows, iMacintosh. SWI-Prolog постійно розвивається, починаючи з моменту створення в 1987 році. Його творцем і основним розробником є Jan Wielemaker. Назва SWI походить від Sociaal-Wetenschappelijke Informatica («Social Science Informatics»), початкової назви групи в Амстердамському університеті, де Wielemaker.

3.1. Основні можливості середовища розробки

Після установки вікно SWI-Editor має наступний вигляд(Рис. 1):

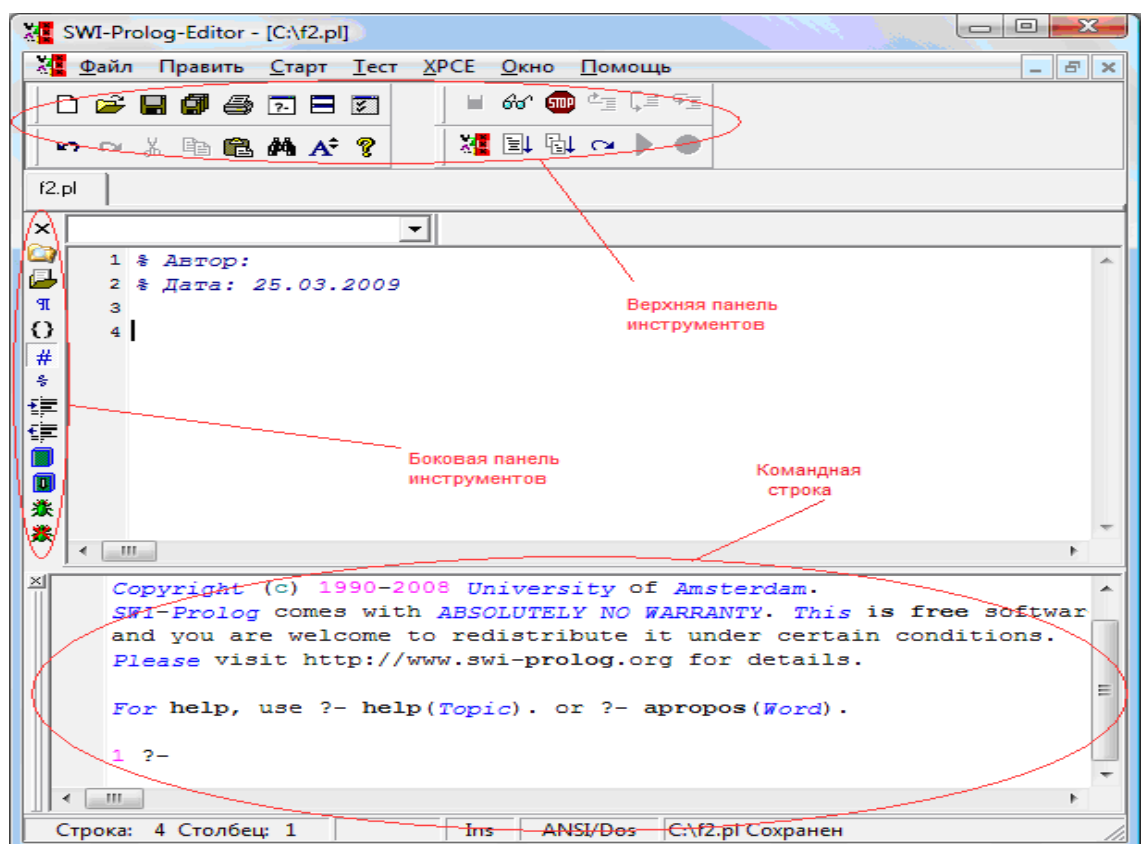


Рис. 1 вікно SWI-Editor

За замовчуванням у програмі виставлено англійську мову, але її можна змінити на російську, за допомогою Window-> Configuration, де вибрати вкладку Option і в поле Language File вказати шлях до файлу russian.ini (зазвичай поставляється разом з Prolog Editor і знаходиться в тій же папці, що і english.ini). Далі пояснення по інтерфейсу редактора будуть викладені з назвами пунктів російською мовою. Майже всі пункти контекстного меню за умовчанням для зручності винесені на верхню панель інструментів. Для створення нової програми необхідно вибрати пункт Файл-> Новый. Після написання програми у вікні редактора її можна виконати командою Старт-> Запустить, або натиснути F9. УВАГА: редактор не підтримує кирилицю в шляху до файлу, тому для успішного запуску програм необхідно, щоб у шляху до файлу програми не було символів кирилиці. Режим налагодження включається і вимикається пунктом Тест-> Отладка(On / Off). Також можливе включення і відключення покрокового виконання програми пунктом Тест-> Трасировка (On / Off). В режимі покрокового виконання для переміщення між командами використовуються кнопки «Следующий» (наступний функтор), «Шаг» (виконання одного кроку програми) і «Шаг наверх» (виконання до виходу на рівень вгору по стеку виклику). У командному рядку відображаються всі дії SWI Editor, що стосуються движка SWI Prolog. Насправді SWI Editor просто дає команди на виконання через командний рядок, тобто є надбудовою над консольної реалізацією середовища розробки SWI Prolog.

МЕТОД РЕКУРСІЇ. РОБОТА ЗІ СПИСКАМИ

Обчислення факторіала.

В математиці факторіал визначається

$$n! = (n-1)! * n;$$

$$0! = 1.$$

В термінах Прологу

$$\text{factorial}(0,1).$$

$$\text{factorial}(N,R):-N>0, N1=N-1, \text{factorial}(N1,R1), R=R1 \times N.$$

Приклад:

Goal: factorial(2,Res).

factorial(2,R):- 2>0, N1=2-1, factorial(N1,R1), R=2×R1.

В результаті отримуємо запит:

factorial(1,R1),

який повертає (зв'язує) R1 з 1. Далі

factorial(N,1):- N >0, N1=N-1, factorial(N1,R1), R=R1×N.

який повертає (зв'язує) R1 з 0. Починається обернений хід рекурсії, в результаті якого отримаємо $2 \times 1 = 1$.

Операції над списками

Списки - це проста, найбільш часто використовувана в нечисловому програмуванні структура, що представляє собою послідовність, складену із довільного числа елементів. Список є рекурсивною структурою:

- він або пустий,
- або
- складається із голови і хвоста,
- де хвіст є список.

Пустий список позначається []. В багатьох реалізаціях Пролога існує функтор «.», який об'єднує голову і хвіст - (1,.(2,.(3,[])).

Існує нотація, що дозволяє розділити голову і хвіст списка.

$L=[\text{Head}|\text{Tail}]$

Основні операції над списками.

До основних операцій над списками звичайно відносять:

- 1) Перевірка належності елемента спискові.
 - 2) З'єднання двох списків.
 - 3) Додавання/видалення елемента в/із списку.
 - 4) Відношення підсписок.
 - 5) Визначення довжини списку.
 - 6) Видалення всіх дублікатів і т.д.
- Відношення належності елемента X списку L можна представити:
X є голова L, або X належить хвосту.

member(X, [X|Tail]).

member(X, [H|T]): - member(X, T).

- Додати елемент add(X,L,[X|L]).
- Видалити елемент. Це відношення можна визначити виходячи із наступних міркувань:

- 1) Якщо видаляється голова, то результатом буде хвіст списку;
- 2) Якщо X знаходиться в хвості списку, то його потрібно видалити звідти.

del(X,[X|T],T).

del(X,[Y|T],[Y|T1]): - del(X,T,T1).

Якщо в списку декілька входжень елемента, то del видалить їх всі за допомогою повернення. Кожна альтернатива буде видаляти тільки одне входження.

Goal: del(a,[a,b,a,a],L).

L=[b,a,a]; L=[a,b,a]; L=[a,b,a]; no

Якщо використати del в оберненому напрямку, то можна додати елемент на довільні позиції. Яким повинен бути список L, щоб після видалення a одержався список [1,2,3] ?

Goal: del (a,L,[b,c,d]).

L=[a,1,2,3]; L=[1,a,2,3]; . . . no

- **Перестановка** - предикат з двома аргументами-списками, один із яких є перестановкою іншого. Перестановка здійснюється за допомогою механізму автоматичного перебору.

1) Якщо перший список пустий, то і другий повинен бути пустий.

2) Якщо перший список не пустий, тоді він має вигляд [X|L] і тоді спочатку отримаємо L1 - перестановку L, після чого внести X на довільну позицію L.

permut([],[]).

permut([X|L],P):- permut(L,L1), insert(X,L1,P).

Предикат insert можна визначити:

insert(X,List,BigList):- del(X,BigList,List).

- **Друк списку.**

Прямий порядок

```
type_list([]).
```

```
type_list([H|T]):-write(H), nl, type_list(T).
```

Обернений порядок .

```
type_list_rev ([]).
```

```
type_list_rev([H|T]):-type_list_rev(T), write(H), nl.
```

Приведемо текст Пролог-програми для наступної задачі. Із вихідного списку, що містить задану кількість елементів, отримати новий список, кожний елемент якого дорівнює збільшеному в два рази відповідному елементу вхідного списку. Обидва списки вивести в стандартному вигляді, тобто елементи розділити комами, заключити в квадратні дужки.

```
/* Ввід списку із N елементів */
```

```
read_list(0,[ ]).
```

```
read_list(N, [H|T]):-readreal(H), N1=N-1,
```

```
read_list(N1,T).
```

```
/* Вивід елементів списку. Після кожного елемента виводиться кома*/
```

```
wr_list([ ]).
```

```
wr_list([H|T]):-write(H,', '), wr_list(T).
```

```
/* Вивід списку в стандартній формі */
```

```
write_list(L):-write('[ '), wr_list(L), cursor(A,B),B1=B-1,cursor(A, B1),  
write(']').
```

```
/* Перетворення вхідного списку */
```

```
new_list([ ],[ ]).
```

```
new_list([H1|T1],[H2|T2]):-H2=H1*2,new_list(T1, T2).
```

```
result:-write("Введіть число елементів списку"), nl, write("N="),  
readint(N), nl,
```

```
write("Введіть елементи списку"), nl, read_list(N,L), new_list(L,L1),
```

```
write("Вхідний список L="), write_list(L),nl,nl,
```

```
write("Новий список L1="), write_list(L1).
```

ЗАВДАННЯ ДЛЯ САМОСТІЙНОЇ РОБОТИ №1

Варіант 1

1. Створити предикати, що відображають наступні родинні відносини: відношення ТРОЮРІДНИЙ БРАТ(X, Y) і відношення ТРОЮРІДНА СЕСТРА(X, Y) через відношення БАТЬКИ(X, Y), ЖІНОЧА_СТАТЬ(X) і ЧОЛОВІЧА_СТАТЬ(X). Записати як читаються такі відношення. Скласти запити, які виводять на екран всіх членів родини, які є чиїмись сестрами чи братами.
2. Видалити початок списку до заданого елемента X (включно). Удалити кінець списку, починаючи з заданого елемента X .

Варіант 2

1. Створити предикати, що відображають наступні родинні відносини: відношення ДВОЮРІДНА ТІТКА(X, Y) і відношення ДВОЮРІДНИЙ ДЯДЬКО(X, Y) через відношення БАТЬКИ(X, Y), ЖІНОЧА_СТАТЬ(X) і ЧОЛОВІЧА_СТАТЬ(X). Записати як читаються такі відношення. Тітка – рідна сестра батьків, а дядько – рідний брат батьків. Двоюрідна тітка – двоюрідна сестра батьків, а двоюрідний дядько – двоюрідний брат батьків. Скласти запити, які виводять на екран всіх членів родини, які є чиїмись тітками (дядьками).
2. Написати програму сортування списку методом вставки, результатом роботи програми повинен бути вхідний, відсортований та розвернутий списки.

Варіант 3

1. Створити предикати, що відображають наступні родинні відносини: відношення ПЛЕМІННИК(X, Y) і відношення ПЛЕМІННИЦЯ(X, Y) через відношення БАТЬКИ(X, Y), ЖІНОЧА_СТАТЬ(X) і ЧОЛОВІЧА_СТАТЬ(X). Записати як читаються такі відношення. Племінник – син рідного брата або сестри. Племінниця – дочка рідного

брата або сестри. Скласти запити, які виводять на екран всіх членів родини, які є чиймись племінницями чи племінниками.

2. Всі елементи списку зменшити на величину середнього арифметичного цих елементів.

Варіант 4

1. Створити предикати, що відображають наступні родинні відносини: відношення $ТЕСТЬ(X,Y)$ і відношення $ТЕЩА(X,Y)$ через відношення $БАТЬКИ(X,Y)$, $ШЛЮБ(X,Y)$, $ЖІНОЧА_СТАТЬ(X)$ і $ЧОЛОВІЧА_СТАТЬ(X)$. Записати як читаються такі відношення. Теща – мати дружини. Тесть – батько дружини. Скласти запити, які виводять на екран всіх членів родини, які є чиймись тестем чи тещею.
2. Написати програму сортування списку методом швидкого сортування, результатом роботи програми повинен бути вхідний, відсортований та розвернутий списки.

Варіант 5

1. Створити предикати, що відображають наступні родинні відносини: відношення $ОНУК(X,Y)$ і відношення $ОНУЧКА(X,Y)$ через відношення $БАТЬКИ(X,Y)$, $ЖІНОЧА_СТАТЬ(X)$ і $ЧОЛОВІЧА_СТАТЬ(X)$. Записати як читаються такі відношення. Скласти запити, які виводять на екран всіх членів родини, які є чиймись бабусею чи дідусем.
2. Обчислити суму додатних і від'ємних елементів списку та вивести списки додатних і від'ємних елементів.

Варіант 6

1. Створити предикати, що відображають наступні родинні відносини: відношення $НЕВІСТКА(X,Y)$ і відношення $ЗЯТЬ(X,Y)$ через відношення $БАТЬКИ(X,Y)$, $ШЛЮБ(X,Y)$, $ЖІНОЧА_СТАТЬ(X)$ і $ЧОЛОВІЧА_СТАТЬ(X)$. Записати як читаються такі відношення. Невістка – дружина сина по відношенню до батьків. Зять – чоловік дочки по відношенню до батьків. Скласти запити, які виводять на екран всіх членів родини, які є чиймись невісткою чи зятем.

2. Написати програму сортування списку бульбашковим методом, результатом роботи програми повинен бути вхідний, відсортований та розвернутий списки.

Варіант 7

1. Створити предикати, що відображають наступні родинні відносини: відношення МАЧУХА(X,Y) і відношення ВІТЧИМ(X,Y) через відношення БАТЬКИ(X,Y), ЧОЛОВІК(X,Y) і ДРУЖИНА(X,Y). Записати як читаються такі відношення. Вітчим – нерідний батько, чоловік матери по відношенню до її дітей від колишнього шлюбу. Мачуха – нерідна мати, жінка батька по відношенню до його дітей від колишнього шлюбу. Скласти запити, які виводять на екран всіх членів родини, які є чиїмись мачухою чи вітчимом.
2. Визначити різницю між максимальним і мінімальним елементами списку.

Варіант 8

1. Створити предикати, що відображають наступні родинні відносини: відношення КУЗЕН(X,Y) і відношення КУЗИНА(X,Y) через відношення БАТЬКИ(X,Y), ЖІНОЧА_СТАТЬ(X) і ЧОЛОВІЧА_СТАТЬ(X). Записати як читаються такі відношення. Кузен – двоюрідний брат. Кузина – двоюрідна сестра. Скласти запити, які виводять на екран всіх членів родини, які є чиїмись кузеном чи кузиною.
2. Написати програму, яка виводить n-й член ряду Фібоначчі (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,...), суму перших членів цього ряду та список, складений з них.

Варіант 9

1. Створити предикати, що відображають наступні родинні відносини: відношення СВЕКОР(X,Y) і відношення СВЕКРУХА(X,Y) через відношення БАТЬКИ(X,Y), ШЛЮБ(X,Y), ЖІНОЧА_СТАТЬ(X) і ЧОЛОВІЧА_СТАТЬ(X). Записати як читаються такі відношення. Свекор – батько чоловіка. Свекрха –мати чоловіка. Скласти запити, які виводять на екран всіх членів родини, які є чиїмись свекром чи свекрухою.

2. Підрахувати кількість додатних і від'ємних елементів списку та вивести списки додатних і від'ємних елементів.

Варіант 10

1. Створити предикати, що відображають наступні родинні відносини: відношення ПАДЧЕРИЦЯ(X, Y) і відношення ПАСИНОК(X, Y) через відношення БАТЬКИ(X, Y), ШЛЮБ(X, Y), ЖІНОЧА_СТАТЬ(X) і ЧОЛОВІЧА_СТАТЬ(X). Записати як читаються такі відношення. Падчериця – нерідна дочка одного з подружжя. Пасинок – нерідний син одного з подружжя. Записати як читається таке відношення. Скласти запити, які виводять на екран всіх членів родини, які є чиїмись падчерицею чи пасинком.
2. Обчислити суму елементів, що стоять в списку на непарних місцях і суму елементів, що стоять на парних місцях, та вивести два списки, складені з них.

Варіант 11

1. Створити предикати, що відображають наступні родинні відносини: відношення ДВОЮРІДНИЙ ПЛЕМІННИК(X, Y) і відношення ДВОЮРІДНА ПЛЕМІННИЦЯ(X, Y) через відношення БАТЬКИ(X, Y), ЖІНОЧА_СТАТЬ(X) і ЧОЛОВІЧА_СТАТЬ(X). Записати як читаються такі відношення. Двоюрідний племінник – син двоюрідного брата або сестри. Племінниця – дочка двоюрідного рідного брата або сестри. Скласти запити, які виводять на екран всіх членів родини, які є чиїмись племінницями чи племінниками.
2. Із заданного списку утворити поліндом. Поліндром - це список, який читається однаково як зліва направо, так і в протилежному напрямку. Приклад поліндрома - [1,2,3,2,1].

Варіант 12

1. Створити предикати, що відображають наступні родинні відносини: відношення ПРАОНУК(X, Y) і відношення ПРАОНУЧКА(X, Y) через

відношення БАТЬКИ(X, Y), ЖІНОЧА_СТАТЬ(X) і ЧОЛОВІЧА_СТАТЬ(X). Записати як читаються такі відношення. Скласти запити, які виводять на екран всіх членів родини, які є чиймись праонуком чи праонучкою.

2. Підрахувати кількість парних та кількість непарних елементів списку та вивести списки парних і непарних елементів.

Варіант 13

1. Створити предикати, що відображають наступні родинні відносини: відношення ДВОЮРІДНА ТІТКА(X, Y) і відношення ДВОЮРІДНИЙ ДЯДЬКО(X, Y) через відношення БАТЬКИ(X, Y), ЖІНОЧА_СТАТЬ(X) і ЧОЛОВІЧА_СТАТЬ(X). Записати як читаються такі відношення. Тітка – рідна сестра батьків, а дядько – рідний брат батьків. Двоюрідна тітка – двоюрідна сестра батьків, а двоюрідний дядько – двоюрідний брат батьків. Скласти запити, які виводять на екран всіх членів родини, які є чиймись тітками (дядьками).
2. Написати програму сортування списку методом злиття, результатом роботи програми повинен бути вхідний і відсортований списки.

Варіант 14

1. Створити предикати, що відображають наступні родинні відносини: відношення ПЛЕМІННИК(X, Y) і відношення ПЛЕМІННИЦЯ(X, Y) через відношення БАТЬКИ(X, Y), ЖІНОЧА_СТАТЬ(X) і ЧОЛОВІЧА_СТАТЬ(X). Записати як читаються такі відношення. Племінник – син рідного брата або сестри. Племінниця – дочка рідного брата або сестри. Скласти запити, які виводять на екран всіх членів родини, які є чиймись племінницями чи племінниками.

Варіант 15

1. Створити предикати, що відображають наступні родинні відносини: відношення ТЕСТЬ(X, Y) і відношення ТЕЩА(X, Y) через відношення

БАТЬКИ(X,Y), ШЛЮБ(X,Y), ЖІНОЧА_СТАТЬ(X) і ЧОЛОВІЧА_СТАТЬ(X). Записати як читаються такі відношення. Теща – мати дружини. Тесть – батько дружини. Скласти запити, які виводять на екран всіх членів родини, які є чиймись тестем чи тещею.

2. Обчислити суму ряду цілих парних та непарних чисел від 1 до n та списки, складені з них.

БІНАРНІ ДЕРЕВА

Бінарне дерево визначається рекурсивно, яке має ліве піддерево, корінь і праве піддерево. Ліве і праве піддерева самі є бінарними деревами. На рис. 1 показаний приклад бінарного дерева.

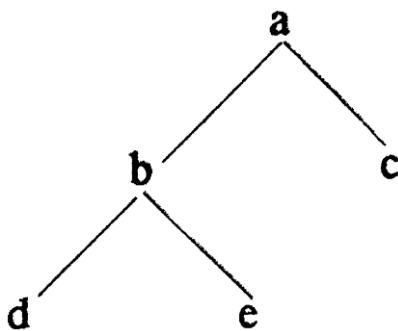


Рис. 1. Бінарне дерево

Такі дерева можна представити термами виду

бд(Лд, К, Пд), де Лд - ліве піддерево, К - корінь, а Пд - праве піддерево.

Для позначення порожнього бінарного дерева будемо використовувати атом **nil**.

Бінарне дерево на мал.1 має ліве піддерево **бд(бд(nil, d, nil), b, бд(nil, e, nil))**

праве піддерево **бд(nil, c, nil)** і записується як

бд(бд(бд(nil, d, nil), b, бд(nil, e, nil)), a, бд(nil,c,nil)).

Представлення множин за допомогою бінарних дерев

Опис множин у виді списків дозволяє використовувати для множин цільове твердження **належить**, визначене раніше для списків. Однак для множин, що складаються з великого числа елементів, спискові цільові твердження стають неефективними. Розглянемо, наприклад, як цільове твердження **належить** дозволяє моделювати належність множині. Нехай **L** -

список, що описує множин з перших **1024** натуральних чисел. Тоді при відповіді на запит

?- належить(3000,L).

прологові прийдеться перевірити всі **1024** числа, перш ніж вияснити, що такого числа немає. Представлення множини бінарним деревом дозволяє домогтися кращого результату. При цьому бінарне дерево повинне бути упорядковане таким чином, щоб будь-який елемент у лівому піддереві був меншим, ніж значення кореня, а будь-який елемент у правому піддереві - більшим. Дерево на рис.1 є неупорядкованим.

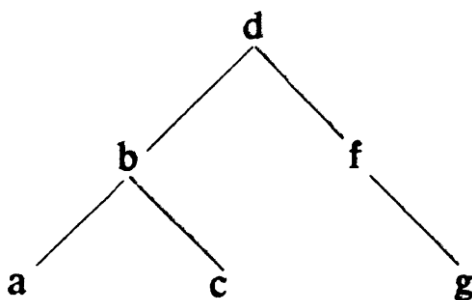


Рис. 2. Упорядковане бінарне дерево

Зверніть увагу, що упорядкування приводить не до єдиного варіанта представлення множини за допомогою дерева. Наприклад, на рис. 3 зображена та ж множина, що і на рис.2.

Будемо називати лінійним представлення на мал.3, і збалансованим - таке, як на рис.2.

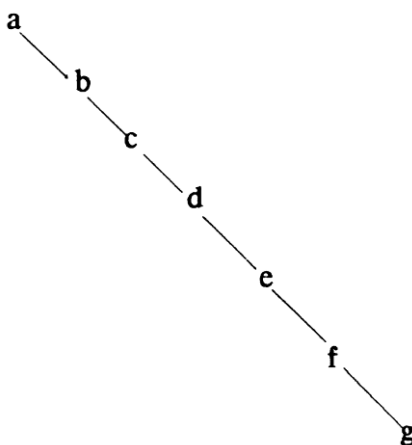


Рис.3. Лінійне представлення

Практичні завдання

1. Створити предикат для виведення бінарного дерева. Для відображення непустиого дерева необхідно відобразити праве піддерево з відступом вправо на відстані В, надрукувати корінь дерева і відобразити ліве піддерево з відступом вправо на відстані В.

друк(T):-друк1(T,0).

друк1(nil,_).

друк1(бд(Л, К, П),В):-В1 is В+2, друк1(П,В1), tab(В),write(К), nl, друк1(Л,В1).

?- друк(бд(бд(бд(nil, 1, nil),2,бд(nil, 3, nil)), 5, бд(бд(nil, 6,nil),7, бд(nil, 8, nil)))).

8

7

6

5

3

2

1

true.

2. Описати предикат пошуку елемента в бінарному дереві, використовуючи лекційний матеріал. Потім внести зміни в процедуру: якщо елемент належить дереву, знайти шлях між коренем і цим елементом. Змінена процедура матиме вигляд:

належить _бд1(Елем, бд(_, Елем, _), [Елем]).

належить _бд1 (Елем, бд(Лів, Корінь, Прав), [Корінь|Шлях]) :-

Корінь>Елем, належить _бд1 (Елем, Лів, Шлях).

належить _бд1 (Елем, бд(Лів,Корінь,Прав), [Корінь|Шлях]) :-

Елем>Корінь, належить _бд1(Елем, Прав, Шлях).

?- належить _бд1 (6, бд(бд(бд(nil, 1,nil),2,бд(nil, 3,nil)), 5, бд(бд(nil, 6,nil),7,бд(nil, 8, nil))), L).

L = [5, 7, 6]

Протестувати предикат для інших бінарних дерев.

3. Описати предикат введення елемента в бінарне дерево.

а) Найпростіший спосіб внесення на самий нижчий рівень, так щоб він став ЛИСТКОМ:

включ_бд(nil, X, бд(nil, X, nil)).

включ_бд(бд(Лд, К, Пд), X, бд(Лднов, К, Пд)) :-X<K, включ_бд(Лд, X, Лднов).

включ_бд(бд(Лд, К, Пд), X, бд(Лд, К, Пднов)) :- X>K, включ_бд(Пд, X, Пднов).

?- **включ_бд(бд(бд(бд(nil, 1,nil),2,бд(nil, 3,nil)), 5, бд(бд(nil, 6,nil),7, бд(nil, 8, nil))), 9, Бд).**

Бд = бд(бд(бд(nil, 1, nil), 2, бд(nil, 3, nil)), 5, бд(бд(nil, 6, nil), 7, бд(nil, 8, бд(nil, 9, nil)))).

?- **включ_бд(бд(бд(бд(nil, 1,nil),2,бд(nil, 3,nil)), 5, бд(бд(nil, 6,nil),7, бд(nil, 8, nil))), 9, Бд), друк(Бд).**

9
8
7
6
5
3

2

1

Бд = бд(бд(бд(nil, 1, nil), 2, бд(nil, 3, nil)), 5, бд(бд(nil, 6, nil), 7, бд(nil, 8, бд(nil, 9, nil)))).

?- включ_бд(бд(бд(бд(nil, 1,nil),3,бд(nil, 4,nil)), 5, бд(бд(nil, 6,nil),7, бд(nil, 8, nil))), 2, Бд), друк(Бд).

8

7

6

5

4

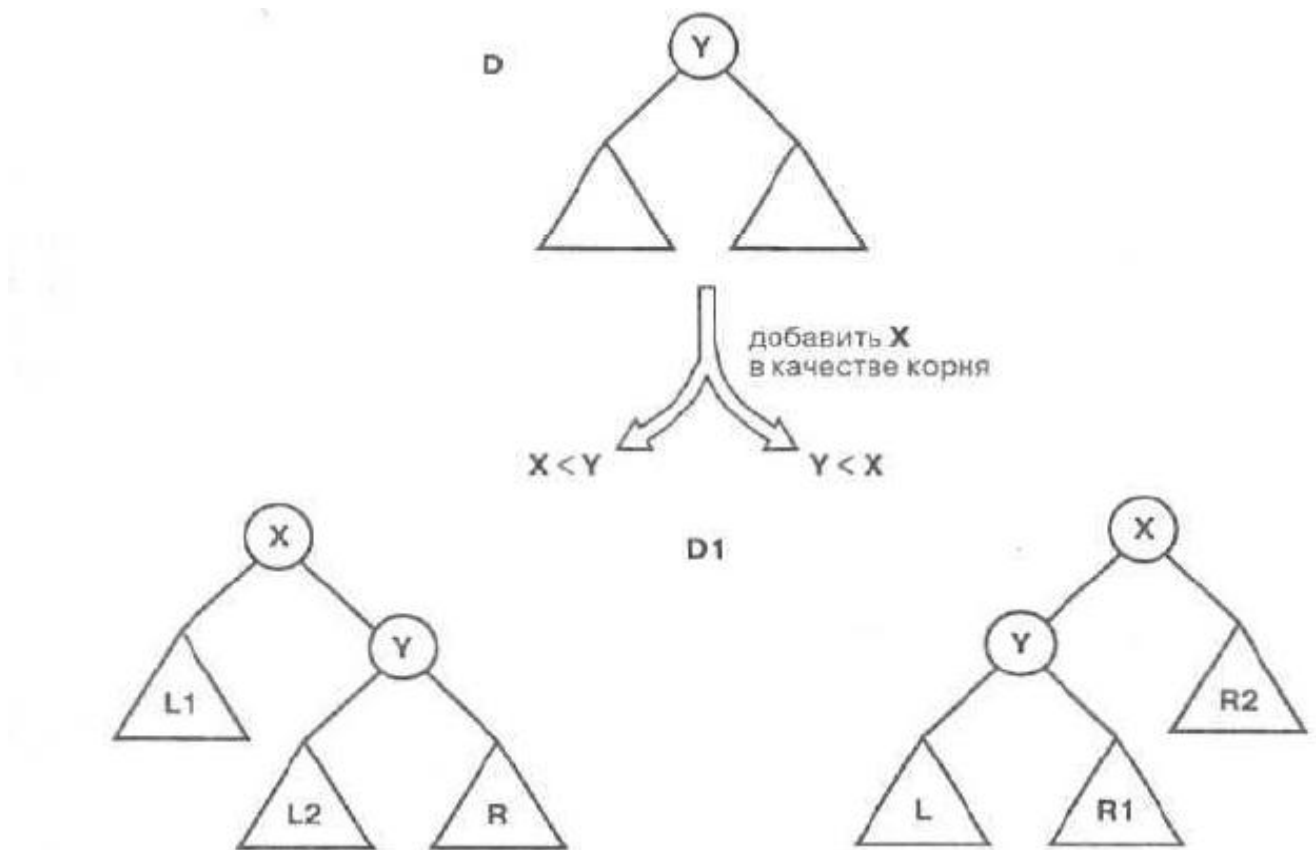
3

2

1

Бд = бд(бд(бд(nil, 1, бд(nil, 2, nil)), 3, бд(nil, 4, nil)), 5, бд(бд(nil, 6, nil), 7, бд(nil, 8, nil)))

б) Внести элемент на довільний рівень бінарного дерева. Для цього необхідно описати нову процедуру **включ_корінь(Д, Х, Д1)**, де Х - новий елемент, який включають в Д замість кореня, а Д1 – нове бінарне дерево з коренем Х.



включ_бд1(Д, Х,Д1):- включ_корінь(Д, Х, Д1).

включ_бд1(бд(Лд, К, Пд), Х, бд(Лднов, К, Пд)) :-X<К, включ_бд1(Лд, Х, Лднов).

включ_бд1(бд(Лд, К, Пд), Х, бд(Лд, К, Пднов)) :- X>К, включ_бд1(Пд, Х, Пднов).

включ_корінь(nil, Х, бд(nil, Х, nil)).

включ_корінь(бд(Лд, Y, Пд), Х, бд(Лд1, Х, бд(Лд2, Y, Пд))) :- Y>X, включ_корінь(Лд, Х, бд(Лд1, Х,Лд2)).

включ_корінь(бд(Лд, Y, Пд), Х, бд(бд(Лд, Y, Пд1), Х, Пд2)) :- Y<X, включ_корінь(Пд, Х, бд(Пд1, Х,Пд2)).

?- включ_бд1(бд(бд(бд(nil, 1,nil),3,бд(nil, 4,nil)), 5, бд(бд(nil, 6,nil),7, бд(nil, 8, nil))), 2, Бд), друк(Бд).

8

7

6

5

4

3

2

1

Бд = бд(бд(nil, 1, nil), 2, бд(бд(nil, 3, бд(nil, 4, nil)), 5, бд(бд(nil, 6, nil), 7, бд(nil, 8, nil))))

8

7

6

5

4

3

2

1

Бд = бд(бд(бд(nil, 1, nil), 2, бд(nil, 3, бд(nil, 4, nil))), 5, бд(бд(nil, 6, nil), 7, бд(nil, 8, nil))) ;

8

7

6

5

4

3

2

1

Бд = бд(бд(бд(бд(nil, 1, nil), 2, nil), 3, бд(nil, 4, nil)), 5, бд(бд(nil, 6, nil), 7, бд(nil, 8, nil))) ;

8

7

6

5

4

3

2

1

Бд = бд(бд(бд(nil, 1, бд(nil, 2, nil)), 3, бд(nil, 4, nil)), 5, бд(бд(nil, 6, nil), 7, бд(nil, 8, nil)))

СТРАТЕГІЇ РОЗВ'ЯЗАННЯ ЗАДАЧ

Простір станів - це граф, вершини якого відповідають ситуаціям, що зустрічаються в задачі ("проблемні ситуації"), а рішення задачі зводиться до пошуку шляху в цьому графі. Процес розв'язання задачі містить у собі пошук у графі, при цьому, як правило, виникає проблема, як обробляти альтернативні шляхи пошуку. Основні стратегії перебору альтернатив - це пошук у глибину і пошук у ширину.

1. Пошук у глибину

Почнемо розробку алгоритму і його варіантів з наступної простої ідеї: Для того, щоб знайти шлях із заданої вершини В в деяку цільову вершину, необхідно:

- " якщо В - це цільова вершина, то покласти Роз = [В], або
- " якщо для вихідної вершини В існує вершина-наступник В1, така, що можна провести шлях з В1 у цільову вершину, то покласти Роз = [В | Роз1].

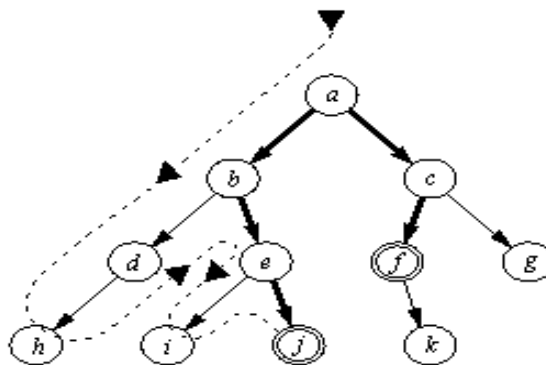


рис. 1. Приклад простору станів: а - стартова вершина, f i j - цільові вершини. Порядок, у якому відбувається прохід по вершинах простору станів при пошуку в глибину: а, b, d, h, e, i, j. Знайдено рішення [а, b, e, j]. Після повернення виявлене інше рішення: [а, с, f].

Ми говоримо "у глибину", маючи на увазі той порядок, у якому розглядаються альтернативи в просторі станів. Завжди, коли алгоритмові пошуку в глибину слід вибрати з декількох вершин ту, в яку варто перейти для продовження пошуку, він віддає перевагу саму "глибоку" з них. Найглибша вершина - це вершина, розташована далі інших від стартової вершини. На рис. 1 ми бачимо на прикладі, у якому порядку алгоритм проходить по вершинах. Пошук у глибину найбільш адекватний рекурсивному стилеві програмування, прийнятому в Пролозі. Причина цього полягає в тому, що, обробляючи цілі, пролог-система сама переглядає альтернативи саме в глибину. Пошук у глибину простий, його легко програмувати і він у деяких випадках добре працює. Пошук у глибину часто працює добре, як у розглянутому прикладі, однак наша проста процедура вирішити може потрапити в складне положення, причому багатьма способами. Чи станеться це чи ні - залежить від структури простору станів. Для того, щоб ускладнити роботу процедури в прикладі мал. 1 досить внести в задачу зовсім невелику зміну: додати дугу, що веде з h у d , щоб вийшов цикл. Отже необхідним є виявлення циклів.

Крім того, багато просторів станів нескінченні. У такому просторі алгоритм пошуку в глибину може "втратити" ціль, рухаючи вздовж нескінченної гілки графа. Програма буде нескінченно довго обстежувати цю нескінченну область простору, так і не наблизившись до мети.

Для того, щоб запобігти безцільному блуканню по нескінченних гілках необхідно ввести обмеження на глибину пошуку.

2. Пошук в ширину

На противагу пошукові в глибину стратегія пошуку в ширину передбачає перехід у першу чергу до вершин, найближчий до стартової вершини. У результаті процес пошуку має тенденцію розвиватися більш у ширину, ніж у глибину, що ілюструє мал.2. Пошук у ширину програмується не так легко, як пошук у глибину. Причина полягають у тому, що

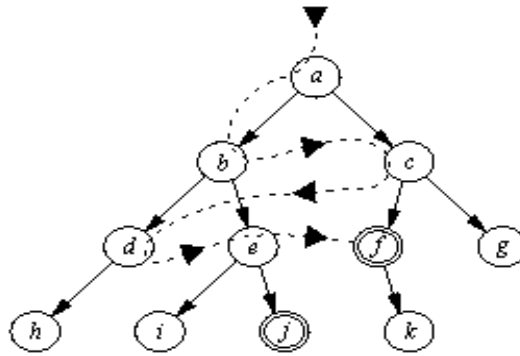


рис. 2. Простий простір станів: а - стартова вершина, f і j - цільові вершини. Застосування стратегії пошуку в ширину дає наступний порядок проходження по вершинах: а, b, c, d, e, f. Більш

коротке рішення [a, c, f] знайдене раніше, ніж більш довге [a, b, e, j]

нам приходится зберігати всю множину альтернативних вершин-кандидатів, а не тільки одну вершину, як при пошуку в глибину. Більш того, якщо ми бажаємо одержати за допомогою процесу пошуку вирішальний шлях, тієї однієї множини вершин недостатньо. Тому треба зберігати не множину вершин-кандидатів, а множину шляхів-кандидатів.

3. Пошук з перевагою

Програму пошуку з перевагою можна одержати як результат удосконалення програми пошуку в ширину. Подібно пошукові в ширину, пошук з перевагою починається зі стартової вершини і використовує множини шляхів-кандидатів. У той час, як пошук у ширину завжди вибирає для продовження самий короткий шлях (тобто переходить у вершини найменшої глибини), пошук з перевагою вносить у цей принцип наступне удосконалення: для кожного кандидата обчислюється оцінка і для продовження вибирається кандидат з найкращою оцінкою.

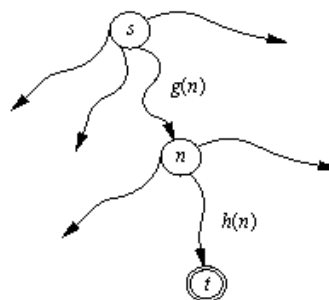


рис. 3. Побудова евристичної оцінки $f(n)$ вартості найдешевшого шляху з s у t, що проходить через n: $f(n) = g(n) + h(n)$.

Нехай f - це евристична оціночна функція, за допомогою якої ми одержуємо для кожної вершини n оцінку $f(n)$ "труднощі" цієї вершини. Тоді найбільш перспективним кандидатом варто вважати вершину, для якої f приймає мінімальне значення. Будемо використовувати тут функцію f спеціального виду, що приводить до добре відомого A^* -алгоритмові. Функція $f(n)$ буде побудована таким чином, щоб давати оцінку вартості оптимального вирішального шляху зі стартової вершини s до однієї з цільових вершин за умови, що цей шлях проходить через вершину n . Давайτε припустимо, що такий шлях існує і що t - це цільова вершина, для якої цей шлях мінімальний. Тоді оцінку $f(n)$ можна представити у виді суми з двох що складаються (мал.3): $f(n) = g(n) + h(n)$. Тут $g(n)$ - оцінка оптимального шляху з s у n ; $h(n)$ - оцінка оптимального шляху з n у t .

Коли в процесі пошуку ми попадаємо у вершину n , ми знаходимося в наступній ситуації: шлях з s у n уже знайдений, і його вартість може бути обчислена як сума вартостей складових його дуг. Що ж стосується $h(n)$, то про нього важко що-небудь сказати, оскільки до цього моменту область простору станів, що лежить між n і t , ще не "вивчена" програмою пошуку. Тому, як правило, про значення $h(n)$ можна тільки будувати здогаду на підставі евристичних розумінь, тобто на підставі тих знань про конкретну задачу, якими володіє алгоритм. Оскільки значення h залежить від предметної області, універсального методу для його обчислення не існує.

ЗАВДАННЯ ДЛЯ САМОСТІЙНОЇ РОБОТИ №2

Гра-головоломка "8". Цільова вершина:

1	2	3
8		4
7	6	5

Для розв'язання задач використати алгоритм пошуку з перевагами.

ЗМІСТ

Лабораторна робота №1. НЕЙРОННА МЕРЕЖА ЗВОРОТНЬОГО ПОШИРЕННЯ ПОХИБКИ.....	3
Лабораторна робота №2 РЕКУРЕНТНІ ТА САМООРГАНІЗОВАНІ НЕЙРОННІ МЕРЕЖІ.....	14
Лабораторна робота №3 АЛГОРИТМ РОЮ ЧАСТИНОК.....	29
Лабораторна робота №4 ОСНОВИ РОБОТИ В СЕРЕДОВИЩІ SWI Prolog. ОСНОВНІ СТРАТЕГІЇ РОЗВ'ЯЗУВАННЯ ЗАДАЧ. АЛГОРИМИ ПОШУКУ РОЗВ'ЯЗКІВ	37

Гарнітура Times New Roman. Папір офсетний.

Формат видання 60х84/16.

Умов. друк. арк.4,0 Зам. № 39. Наклад 25 прим.

Видруковано ПП «АУДИТОР-ШАРК»

88000, м. Ужгород, пл. Жупанатська, 15/1. Тел.: 3-51-25.

E-mail: office@shark.com.ua

Свідоцтво про внесення суб'єкта видавничої справи

до державного реєстру видавців,

виготівників і розповсюджувачів

видавничої продукції

Серія 3т № 40 від 29 жовтня 2012 року

