

Міністерство освіти і науки України
Запорізький національний технічний університет

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних та практичних робіт
з дисципліни

“Мультиагентні системи і технології”

для студентів спеціальностей

121 “Інженерія програмного забезпечення” та

122 “Комп’ютерні науки та інформаційні технології”
(всіх форм навчання)

2016

Методичні вказівки до виконання лабораторних та практичних робіт з дисципліни “Мультиагентні системи і технології” для студентів спеціальності 121 “Інженерія програмного забезпечення” та 122 “Комп’ютерні науки та інформаційні технології” (всіх форм навчання) / А.О. Олійник, В.М. Льовкін, Є.М. Федороченко. – Запоріжжя: ЗНТУ, 2016. – 44 с.

Автори: А.О. Олійник, к.т.н., доцент
В.М. Льовкін, к.т.н., доцент
Є.М. Федороченко, ст. викладач

Рецензент: С.О. Субботін, д.т.н., професор

Відповідальний
за випуск: С.О. Субботін, д.т.н., професор

Затверджено
на засіданні кафедри
програмних засобів

Протокол №1
від “16” серпня 2016 р.

ЗМІСТ

Вступ	5
1 Лабораторна робота № 1 Метод мурашиних колоній.....	6
1.1 Мета роботи	6
1.2 Основні теоретичні відомості.....	6
1.2.1 Основні поняття.....	6
1.2.2 Послідовність виконання методу	7
1.3 Завдання до роботи.....	9
1.4 Зміст звіту.....	10
1.5 Контрольні питання.....	11
2 Лабораторна робота № 2 Метод PSO	12
2.1 Мета роботи	12
2.2 Основні теоретичні відомості.....	12
2.2.1 Основні поняття.....	12
2.2.2 Підхід gbest PSO	13
2.2.3 Підхід lbest PSO	16
2.3 Завдання до роботи.....	18
2.4 Зміст звіту.....	19
2.5 Контрольні питання.....	19
3 Лабораторна робота № 3 Моделювання природних систем.....	21
3.1 Мета роботи	21
3.2 Основні теоретичні відомості.....	21
3.2.1 Поняття штучного життя	21
3.2.2 Моделювання харчових ланцюгів.....	22
3.2.2.1 Опис моделі.....	22
3.2.2.2 Навколишнє середовище	23
3.2.2.3 Анатомія агенту	23
3.2.2.4 Енергія та метаболізм.....	26
3.2.2.5 Відтворення.....	27
3.2.2.6 Смерть.....	27
3.3 Завдання до роботи.....	27
3.4 Зміст звіту.....	29
3.5 Контрольні питання.....	29
4 Лабораторна робота № 4 Паралельне мультиагентне програмування	31
4.1 Мета роботи	31
4.2 Основні теоретичні відомості.....	31

4.2.1 Сімейство мов Golog та розширення MIndiGolog	31
4.2.2 Розв'язання задачі мультиагентного програмування на основі MIndiGolog	33
4.3 Завдання до роботи	36
4.4 Зміст звіту	37
4.5 Контрольні питання	37
5 Лабораторна робота № 5 Недетерміноване мультиагентне програмування	38
5.1 Мета роботи	38
5.2 Основні теоретичні відомості	38
5.3 Завдання до роботи	41
5.4 Зміст звіту	42
5.5 Контрольні питання	43
Література	44

ВСТУП

Дане видання призначене для вивчення та практичного освоєння студентами усіх форм навчання основ мультиагентних систем.

Відповідно до графіка студенти перед виконанням лабораторної роботи повинні ознайомитися з конспектом лекцій та рекомендованою літературою.

Для одержання заліку по кожній роботі студент здає викладачу цілком оформлений звіт, а також 3,5-дюймову дискету у форматі MS – DOS / Windows, перевірену на відсутність вірусів, з текстами розроблених програм, файлами програм, що виконуються, файлами даних і текстом звіту.

Звіт має містити:

- титульний аркуш;
- мету, варіант і завдання роботи;
- лаконічний опис теоретичних відомостей;
- текст програми, що обов'язково містить коментарі;
- вхідні та вихідні дані програми;
- змістовний аналіз отриманих результатів та висновки.

Звіт виконують на білому папері формату А4 (210 x 297 мм). Текст розміщують тільки з однієї сторони листа. Поля сторінки з усіх боків – 20 мм. Аркуші скріплюють за допомогою канцелярських скріпок. Для набору тексту звіту використовують редактор MS Word: шрифт Times New Roman, 12 пунктів. Міжрядковий інтервал: полуторний – для тексту звіту, одинарний – для лістингів програм, таблиць і роздруківок даних.

Під час співбесіди студент повинний виявити знання про мету роботи, по теоретичному матеріалу, про методи виконання кожного етапу роботи, по змісту основних розділів розробленого звіту з демонстрацією результатів на конкретних прикладах. Студент повинний вміти правильно аналізувати отримані результати. Для самоперевірки при підготовці до виконання і здачі роботи студент повинний відповісти на контрольні питання, приведені наприкінці опису відповідної роботи. Загальний залік студент одержує після виконання і здачі останньої роботи.

1 ЛАБОРАТОРНА РОБОТА № 1

МЕТОД МУРАШИНИХ КОЛОНІЙ

1.1 Мета роботи

1.1.1 Вивчити метод мурашиних колоній.

1.1.2 Навчитися розв'язувати задачі комбінаторної оптимізації за допомогою методу мурашиних колоній.

1.2 Основні теоретичні відомості

1.2.1 Основні поняття

Мурашині алгоритми серйозно досліджуються європейськими вченими із середини 1990-х років. На сьогоднішній день вже отримано гарні результати для оптимізації таких складних комбінаторних задач, як задача комівояжера, задача оптимізації маршрутів вантажівок, задача розфарбування графа, квадратична задача про призначення, задача оптимізації сіткових графіків, задача календарного планування і т. ін. Особливо ефективні мурашині алгоритми при динамічній оптимізації процесів у розподілених нестационарних системах, наприклад, трафіків у телекомунікаційних мережах.

Метод мурашиних колоній заснований на взаємодії декількох *мурах* (програмних агентів, що є членами великої колонії) і використовується для вирішення різних оптимізаційних задач. Агенти спільно вирішують проблему й допомагають іншим агентам у подальшій оптимізації розв'язку.

Методи мурахи (Ant algorithms), або оптимізація за принципом мурашиної колонії, мають специфічні особливості, властиві мурахам, і використовують їх для орієнтації у фізичному просторі. Природа пропонує різні методики для оптимізації деяких процесів. Методи мурашиних колоній особливо цікаві тому, що їх можна використовувати для вирішення не тільки статичних, але й динамічних задач, наприклад, задач маршрутизації в мінливих мережах.

Базова ідея методу мурашиних колоній полягає у вирішенні оптимізаційної задачі шляхом застосування непрямого зв'язку між автономними агентами.

У методі мурашиних колоній передбачається, що навколишнє середовище являє собою закриту двовимірну мережу – групу вузлів, з'єднаних за допомогою граней. Кожна грань має вагу, що позначається як відстань між двома вузлами, з'єднаними нею. Граф – двонаправлений, тому агент може подорожувати по грані в будь-якому напрямку.

Агент забезпечується набором простих правил, які дозволяють йому вибирати шлях у графі. Він підтримує список табу *tList* – список вузлів, які він вже відвідав. Таким чином, агент повинен проходити через кожний вузол тільки один раз.

Вузли в списку “поточної подорожі” *tList* розташовуються в тому порядку, у якому агент відвідував їх. Пізніше список використовується для визначення довжини шляху між вузлами.

Справжня мураха під час переміщення по шляху залишає за собою деяку кількість *феромону*. У методі мурашиних колоній агент залишає феромон на гранях мережі після завершення подорожі.

1.2.2 Послідовність виконання методу

Метод мурашиних колоній виконується в наступній послідовності кроків.

Крок 1. Задати параметри методу: α – коефіцієнт, що визначає відносну значимість шляху (кількість феромона на шляху); β – параметр, що означає пріоритет відстані над кількістю феромона; ρ – коефіцієнт кількості феромона, що агент залишає на шляху, де $(1-\rho)$ показує коефіцієнт випару феромона на шляху після його завершення; Q – константу, що відноситься до кількості феромона, яку було залишено на шляху.

Крок 2. Ініціалізація методу. Створення популяції агентів. Після створення популяція агентів нарівно розподіляється по вузлах мережі. Необхідно рівномірний розподіл агентів між вузлами, щоб всі вузли мали однакові шанси стати відправною точкою. Якщо всі агенти почнуть рух з однієї точки, це буде означати, що дана точка вважається оптимальною для старту, а насправді вона такою може і не бути.

Крок 3. Рух агентів. Якщо агент ще не закінчив шлях, тобто не відвідав всі вузли мережі, для визначення наступної грані шляху використовується формула:

$$P_{ru} = \frac{\tau_{ru}(t)^\alpha \cdot \eta_{ru}(t)^\beta}{\sum_{k \in J} \tau_{rk}(t)^\alpha \cdot \eta_{rk}(t)^\beta} > rand(0;1),$$

де J – множина граней, ще не відвіданих агентом; $\tau_{ru}(t)$ – інтенсивність феромона на грані між вузлами r і u , які утворюють k -у грань, у момент часу t ; $\eta_{ru}(t)$ – функція, що представляє собою зворотну величину відстані грані.

Агент подорожує тільки по вузлах, які ще не були відвідані ним, тобто по вузлах, яких не має у списку табу $tList$. Тому ймовірність розраховується тільки для граней, які ведуть до ще не відвіданих вузлів.

Крок 4. Пройдений агентом шлях відображається, коли агент відвідає всі вузли мережі. Цикли заборонені, оскільки в метод включений список табу $tList$. Після завершення може бути розрахована *довжина шляху*. Вона дорівнює сумі всіх граней, по яких подорожував агент. *Кількість феромону*, що було залишено на кожній грані шляху i -го агента, визначається за формулою:

$$\Delta\tau_{ru}^i(t) = \frac{Q}{L^i(t)},$$

де $L^i(t)$ – довжина шляху i -го агента.

Результат є засобом вимірювання шляху: короткий шлях характеризується високою концентрацією феромону, а більш довгий шлях – більш низкою. Потім отриманий результат використовується для збільшення кількості феромону уздовж кожної грані пройденого i -им агентом шляху:

$$\tau_{ru}(t) = \tau_{ru}(t-1) + \left[\sum_{i \in J_{ru}} \Delta\tau_{ru}^i(t) \right] \cdot \rho,$$

де r, u – вузли, що утворюють грані, які відвідав i -ий агент; J_{ru} – множина агентів, що відвідали грань (r,u) ; N_{ru} – кількість агентів, що відвідали грань (r,u) .

Дана формула застосовується до всього шляху, при цьому кожна грань позначається феромоном пропорційно довжині шляху. Тому варто дочекатися, поки агент закінчить подорож і тільки потім оновити

рівні феромону, у противному випадку дійсна довжина шляху залишиться невідомою. Константа ρ приймає значення між 0 і 1.

$$\tau_{ru}(t) = \tau_{ru}(t) \cdot (1 - \rho).$$

Тому для випаровування феромону використовується зворотний коефіцієнт відновлення шляху $(1 - \rho)$.

Крок 5. Перевірка на досягнення оптимального результату. Перевірка може виконуватися для постійної кількості шляхів або до моменту, коли протягом декількох запусків не було отримано повторних змін у виборі найкращого шляху. Якщо перевірка дала позитивний результат, то відбувається закінчення роботи методу (перехід до кроку 7), у противному випадку – перехід до кроку 6.

Крок 6. Повторний запуск. Після того як шлях агента завершений, грані оновлені відповідно до довжини шляху, й відбулося випаровування феромону на всіх гранях, метод виконується повторно. Список табу очищується, і довжина шляху онулюється. Після цього виконується до кроку 3.

Крок 7. Кінець. Визначається кращий шлях, що і є розв'язком.

1.3 Завдання до роботи

1.3.1 Ознайомитися з літературою та основними теоретичними відомостями за темою роботи.

1.3.2 Розробити за допомогою пакету Matlab програмне забезпечення, що реалізує метод мурашиних колоній. Рекомендовані значення параметрів: $\alpha \approx 1$, $\beta \approx 5$, $\rho \approx 0,5$, $Q \approx 1000$ при умові, що координати пунктів призначення знаходяться в діапазоні $(0; 100]$ При програмній реалізації методу дотримуватися вимог, наведених в п.1.3.2.1 – 1.3.2.4.

1.3.2.1. Для гнучкості програмного забезпечення всі параметри методу (наприклад, кількість агентів, коефіцієнт випаровування і т. ін.) зберігати в змінних або константах.

1.3.2.2. Для реалізації агентів, що моделюють поведінку мурах, використовувати структуру, що має наступні поля:

- поточна позиція, в якій знаходиться агент;
- наступна позиція;
- список табу (бітовий рядок), в якому зберігаються ті пункти, в яких вже побував агент;

- кількість пунктів, що вже відвідав агент;
- масив подорожі, де зберігається послідовність пунктів, в яких побував агент;
- довжина шляху, що пройшов агент. Розраховується після закінчення подорожі.

1.3.2.3. Виділити основні етапи методу в окремі функції, що реалізуються у відповідних .m-файлах:

- ініціалізація методу;
- моделювання переміщення агентів;
- вибір наступного пункту;
- оновлення феромонів на шляхах;
- перезавантаження агентів.

Також необхідно реалізувати окрему функцію розрахунку довжини шляху.

1.3.2.4. Передбачити можливість наглядного (графічного) відображення змін у поточних результатах роботи методу.

1.3.3 Виконати тестування розробленого програмного забезпечення за допомогою вирішення конкретних прикладів задачі комівояжера. Задачі (не менше трьох) для виконання тестування програми сформулювати самостійно. Вибір тестових задач обґрунтувати.

1.3.4 Порівняти одержані результати вирішення різних прикладів задачі комівояжера. Результати порівняльного аналізу звести до таблиці, попередньо розробивши систему критеріїв порівняння результатів вирішення задачі комівояжера.

1.3.5 Оформити звіт з роботи.

1.3.6 Відповісти на контрольні питання.

1.4 Зміст звіту

1.4.1 Тема та мета роботи.

1.4.2 Короткі теоретичні відомості.

1.4.3 Текст розробленого програмного забезпечення з коментарями.

1.4.4 Результати роботи програмного забезпечення (таблиця порівняльного аналізу та графіки з поясненнями).

1.4.5 Висновки, що містять відповіді на контрольні запитання, а також відображують результати виконання роботи та їх критичний аналіз.

1.5 Контрольні питання

1.5.1 В чому полягає основна ідея методу мурашиних колоній?

1.5.2 Для розв'язання яких задач використовується метод мурашиних колоній?

1.5.3 В яких умовах особливо ефективно використання мурашиних алгоритмів?

1.5.4 Порівняйте метод мурашиних колоній з іншими оптимізаційними методами.

1.5.5 Що являє собою навколишнє середовище в методі мурашиних колоній?

1.5.6 З якою метою використовується список табу?

1.5.7 Яким чином розташовуються вузли в списку “поточної подорожі”?

1.5.8 Що таке феромон? Яке його призначення в методі мурашиних колоній?

1.5.9 Проаналізуйте послідовність виконання методу мурашиних колоній.

1.5.10 Як розраховується кількість феромону, що було залишено на кожній грані шляху i -го агенту?

1.5.11 Поясніть особливості модифікації методу мурашиних колоній при вирішенні таких задач комбінаторної оптимізації, як задача комівояжера, задача оптимізації маршрутів вантажівок, задача розфарбування графа, квадратична задача про призначення, задача оптимізації сіткових графіків, задача календарного планування.

1.5.12 Проаналізуйте засоби пакету Matlab, які були використані при розробці програмного забезпечення, що реалізує метод мурашиних колоній.

2 ЛАБОРАТОРНА РОБОТА № 2

МЕТОД PSO

2.1 Мета роботи

2.1.1 Вивчити метод PSO.

2.1.2 Навчитися знаходити глобальних оптимум багатомірної функції за допомогою методу PSO.

2.2 Основні теоретичні відомості

2.2.1 Основні поняття

Оптимізація з використанням рою частинок (Particle Swarm Optimization, PSO) – це метод пошуку, що базується на понятті популяції, та моделює поведінку птахів у зграї та косяків риб. Першочерговою метою використання концепції рою частинок була графічна імітація красивого та непередбачуваного руху птахів та риб у зграї, з метою пошуку базових принципів, завдяки яким птахи літають (риби плавають) синхронно та вміють змінювати напрям руху з перегрупованням в оптимальні формації. З тих часів концепція розвинулась у простий та перспективний оптимізаційний метод.

В методі PSO особини, що називаються частинками, переміщуються у багатомірному просторі рішень. Зміна координат частинок всередині простору пошуку обумовлюються природною соціально-психологічною тенденцією частинок конкурувати між собою. Отже, зміна в стані частинки залежить від досвіду та знань її сусідів. В даному випадку, слово “знання” є синонімом “інформації”. Результат такого моделювання полягає в тому, що процес пошуку мотивує частинки недетермінованим чином повертатися в оптимальні області простору рішень.

Особини в рої частинок володіють дуже простою поведінкою: вони прагнуть перевершити досягнення сусідніх частинок й покращити власні. Таким чином, емергентна властивість даної системи полягає в дослідженні оптимальних областей багатомірного простору пошуку.

Метод PSO керує роєм частинок й кожна частинка уявляє собою потенційне рішення. За аналогією з еволюційними стратегіями, рій можна розглядати як популяцію, а частинку як індивіда (чи хромосому). Тобто частинка “літає” у багатомірному просторі рішень, та її позиція визначається виходячи з власного досвіду та досвіду своїх сусідів.

За допомогою $x_i(t)$ позначається позиція i -ої частинки в просторі пошуку рішень в момент часу t (t визначає дискретне значення часу). Позиція частинки змінюється з додавання швидкості $v_i(t)$ до поточної позиції:

$$x_i(t + 1) = x_i(t) + v_i(t + 1).$$

Початковий стан визначається наступним чином: $x_i(0) = U(x_{\min}, x_{\max})$, де $U(a, b)$ є функцією випадкових чисел з діапазону $[a, b]$. Дана формула уявляє собою вектор швидкості й визначає сам оптимізаційний процес, а також відбиває використання як досвідчених знань частинки, так й обмін інформацією з сусідніми частинками. Власні знання самої частинки, що також називаються *когнітивним компонентом* формули швидкості, прямо пропорційні поточній відстані частинки від її найкращого положення, яке було знайдено з моменту старту її життєвого циклу. А обмін інформацією даної особини з іншими являється *соціальним компонентом* формули швидкості.

Історично було розроблено два підходи, які фактично являються різновидами базового методу PSO: *gbest* та *lbest*; вони відрізняються ступенем зв'язності частинок у просторі пошуку.

2.2.2 Підхід gbest PSO

У різновиді *gbest* PSO кожна частинка зв'язана з усім роєм. Частинки утворюють так звану соціальну мережу, яка в *gbest* відповідає топології типу “зірка” (рис. 2.1). Кожна частинка може взаємодіяти з усіма іншими частинками, і вона схильна до кращого рішення усього рою. Частинка імітує загальне оптимальне рішення, тому її швидкість залежить від інформації, отримуваної від усіх

інших. В даному випадку, соціальним компонентом швидкості являється найкраща досягнута позиція рою в просторі рішень, і вона позначається як $y^*(t)$.

В методі *gbest* PSO швидкість частинки розраховується за формулою:

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[y^*_j(t) - x_{ij}(t)],$$

де $v_{ij}(t)$ – це швидкість i -ої частинки у вимірі $j=1, \dots, n_x$ в момент часу t ; $x_{ij}(t)$ – це позиція i -ої частинки у просторі j ; c_1 та c_2 – позитивні константи прискорення, які використовуються для варіювання ваг когнітивної та соціальної компонент швидкості частинки відповідно; $r_{1j}(t), r_{2j}(t) = U(0,1)$ являються випадковими значення з діапазону $[0,1]$. Ці випадкові величини додають стохастичний елемент у роботу методу.

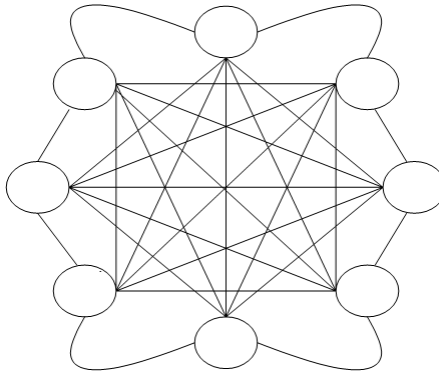


Рисунок 2.1 – Топологія типу “зірка”

Величина y_i являється найкращою позицією i -ої частинки, яку вона відвідала, починаючи з першої ітерації. Наступна оптимальна позиція i -ої частинки в момент часу $t+1$ розраховується за формулою:

$$y_i(t+1) = \begin{cases} y_i(t), & \text{якщо } f(x_i(t+1)) \geq f(y_i(t)); \\ x_i(t+1), & \text{якщо } f(x_i(t+1)) < f(y_i(t)), \end{cases}$$

де $f: \mathcal{R}^{n_x} \rightarrow \mathcal{R}$ – фітнес-функція. Так само, як і в еволюційних підходах, вона являється мірою близькості даного рішення до оптимального, також фітнес-функція визначає продуктивність або якість частинки (рішення).

Глобальний найкращий оптимум $y^*(t)$, в момент часу t , визначається так:

$$y^*(t) \in \{y_0(t), \dots, y_{n_s}(t)\} \mid f(y^*(t)) = \min \{f(y_0(t)), \dots, f(y_{n_s}(t))\},$$

де n_s – загальна кількість частинок у рою. Важливо відзначити, що відповідно до даної формули y^* – це найкраща позиція, що була віднайдена будь-якою з частинок. Глобальний оптимум також може бути розрахований на основі інформації про частинки з даного рою:

$$y^*(t) = \min \{f(x_0(t)), \dots, f(x_{n_s}(t))\}.$$

Метод *gbest* PSO можна описати у вигляді послідовності наступних кроків.

Крок 1. Якщо умови закінчення виконані, то вихід, в іншому випадку – перехід на крок 2.

Крок 2. Створити та ініціалізувати n_x -мірний рій.

Крок 3. Встановити: $i = 1$.

Крок 4. Визначення локально найкращої позиції. Якщо виконується умова $f(x_i) < f(y_i)$, то встановити $y_i = x_i$.

Крок 5. Визначення глобально найкращої позиції. Встановити: $y^* = y_i$.

Крок 6. Встановити: $i = i + 1$.

Крок 7. Якщо $i < n_s$, то перехід на крок 3, в іншому випадку – на крок 8.

Крок 8. Встановити: $i = 1$.

Крок 9. Оновити швидкість частинки, використовуючи формулу зміни швидкості частинки для *gbest* PSO.

Крок 10. Оновити позицію частинки.

Крок 11. Встановити: $i = i + 1$.

Крок 12. Якщо $i < n_s$, то виконати перехід на крок 9, в іншому випадку на крок 1.

2.2.3 Підхід lbest PSO

У локально найкращому методі PSO, або *lbest* PSO, частинки утворюють соціально-мережеву топологію типу “кільце” (рис. 2.2). та для кожної частинки задається менша кількість сусідів, ніж при *gbest* підході. Кожна частинка намагається імітувати кращого із своїх сусідів, тяготячись до нього. Соціальний компонент швидкості відбиває обмін інформації між сусідами частинки. Відповідно до формули зміни позиції вклад у швидкість частинки пропорційний відстані між нею та найкращим рішенням, знайденим її сусідами. Швидкість розраховується за формулою:

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[y_{ij}^*(t) - x_{ij}(t)],$$

де $y_{ij}^*(t)$ – найкраща позиція, знайдена сусідами частинки i в вимірі j . Локально найкраща позиція частинки y_i^* , тобто краща позиція, знайдена у сусідстві N_i і визначається як:

$$y_i^*(t+1) \in \{N_i \mid f(y_i^*(t+1))\} = \min\{f(x)\}, \forall x \in N_i,$$

де сусідство визначається за наступною формулою:

$$N_i = \{y_{i-n_{N_i}}(t), y_{i-n_{N_i}+1}(t), \dots, y_{i-1}(t), y_i(t), y_{i+1}(t), \dots, y_{i+n_{N_i}}(t)\},$$

для сусідства розмірності n_{N_i} .

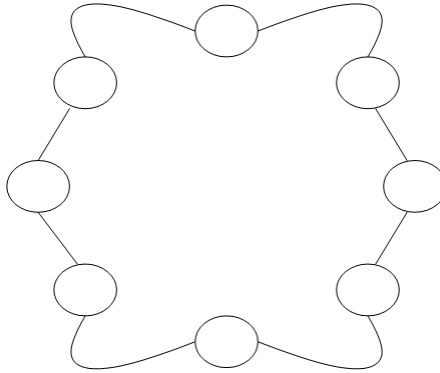


Рисунок 2.2 – Топологія типу “кільце”

Метод *lbest* є майже еквівалентним *gbest*, з тією різницею, що на кроці 9 швидкість частинки оновлюється за формулою для *lbest* PSO, а зв'язані частинки визначаються більш складним чином.

Крок 1. Якщо умови закінчення виконані, то вихід, в іншому випадку – перехід на крок 2.

Крок 2. Створити та ініціалізувати n_x -мірний рій.

Крок 3. Встановити: $i = 1$.

Крок 4. Визначення локально найкращої позиції. Якщо виконується умова $f(x_i) < f(y_i)$, то встановити $y_i = x_i$.

Крок 5. Визначення глобально найкращої позиції. Встановити: $y^* = y_i$.

Крок 6. Встановити: $i = i + 1$.

Крок 7. Якщо $i < n_s$, то перехід на крок 3, в іншому випадку – на крок 8.

Крок 8. Встановити: $i = 1$.

Крок 9. Оновити швидкість частинки, використовуючи формулу зміни швидкості частинки для *lbest* PSO.

Крок 10. Оновити позицію частинки.

Крок 11. Встановити: $i = i + 1$.

Крок 12. Якщо $i < n_s$, то виконати перехід на крок 9, в іншому випадку на крок 1.

Закінчення виконання методу PSO відбувається, коли виконується хоча б один з критеріїв зупину. Існує два аспекти, котрі мають бути враховані при виборі критеріїв.

1. Умова зупину не має сприяти передчасній збіжності методу PSO. В іншому випадку будуть отримані субоптимальні рішення.

2. При перевірці критеріїв зупину фітнес-функція має викликатися мінімальну кількість разів, оскільки від цього дуже сильно залежить швидкість роботи методу PSO.

Звичайно використовують наступні критерії зупину:

- досягнення максимальної кількості ітерацій;
- задовільне рішення знайдено;
- вихід при відсутності покращень на протязі великої кількості ітерацій;
- вихід при умові, що перша похідна фітнес-функції в околиці частинки чи частинок дорівнює нулю.

2.3 Завдання до роботи

2.3.1 Ознайомитися з літературою та основними теоретичними відомостями за темою роботи.

2.3.2 Розробити за допомогою пакету Matlab програмне забезпечення, що реалізує метод PSO, як *gbest*, так і *lbest*. При програмній реалізації методу дотримуватися вимог, наведених в п.2.3.2.1 – 2.3.2.3.

2.3.2.1. Для гнучкості програмного забезпечення всі параметри методу (наприклад, кількість агентів, коефіцієнти прискорення і т. ін.) зберігати в змінних або константах.

2.3.2.2. Виділити основні етапи методу в окремі функції, що реалізуються у відповідних .m-файлах.

2.3.2.3. Передбачити можливість наглядного (графічного) відображення змін у поточних результатах роботи методу.

2.3.3 Виконати тестування розробленого програмного забезпечення за допомогою вирішення конкретних прикладів задачі оптимізації багатомірної функції. Тестові функції обрати самостійно. Вибір тестових функцій обґрунтувати.

2.3.4 Порівняти одержані результати оптимізації різних тестових функцій для підходів *gbest* та *lbest*. Результати порівняльного аналізу звести до таблиці, попередньо розробивши систему критеріїв порівняння результатів пошуку оптимуму.

2.3.5 Оформити звіт з роботи.

2.3.6 Відповісти на контрольні питання.

2.4 Зміст звіту

2.4.1 Тема та мета роботи.

2.4.2 Короткі теоретичні відомості.

2.4.3 Текст розробленого програмного забезпечення з коментарями.

2.4.4 Результати роботи програмного забезпечення (таблиця порівняльного аналізу та графіки з поясненнями).

2.4.5 Висновки, що містять відповіді на контрольні запитання, а також відображують результати виконання роботи та їх критичний аналіз.

2.5 Контрольні питання

2.5.1 В чому полягає основна ідея методу PSO?

2.5.2 Для розв'язання яких задач використовується метод PSO?

2.5.3 Порівняйте метод PSO з іншими оптимізаційними методами.

2.5.5 Що являє собою частинка в методі PSO?

2.5.6 Які частини компоненти формули швидкості частинок є і яка їх роль?

2.5.7 Які два основні підходи методу PSO існують?

2.5.8 Порівняйте *gbest* та *lbest* PSO.

2.5.9 Назвіть особливості ініціалізації та зупину роботи методу PSO.

2.5.10 Проаналізуйте вплив констант на роботу методу PSO.

2.5.11 Якими принципами треба користуватися при виборі значень констант роботи методу PSO?

3 ЛАБОРАТОРНА РОБОТА № 3 МОДЕЛЮВАННЯ ПРИРОДНИХ СИСТЕМ

3.1 Мета роботи

3.1.1 Навчитися моделювати природні системи.

3.1.2 Вивчити методи, які застосовуються до моделювання природних систем.

3.2 Основні теоретичні відомості

3.2.1 Поняття штучного життя

Штучне життя (Artificial life) – поняття, що ввів Кріс Лангтон для позначення множини комп'ютерних механізмів, котрі використовуються для моделювання природних систем. Штучне життя використовується для моделювання процесів у економіці, поведінки тварин та комах, а також для взаємодії різноманітних об'єктів.

Штучне життя являє собою цілу науку з безліччю аспектів. До штучного життя відноситься синтетична наука про поведінку, що представляє собою підхід до вивчення поведінки тварин, при якому прості синтетичні організми певним чином діють у синтетичному світі.

Штучне життя може бути описане як теорія й практика моделювання біологічних систем. Розробники, які ведуть дослідження в даній сфері, сподіваються, що шляхом моделювання біологічних систем можна бути зрозуміти, чому і як вони працюють. За допомогою моделей дослідники можуть управляти створеним середовищем, перевіряти різні гіпотези й спостерігати, як системи й середовище реагують на зміни.

3.2.2 Моделювання харчових ланцюгів

Харчовий ланцюг описує ієрархію живих організмів в екосистемі. Найпростішим прикладом харчового ланцюгу може бути абстрактний ланцюг, що складається із трьох особин (рис. 3.1).

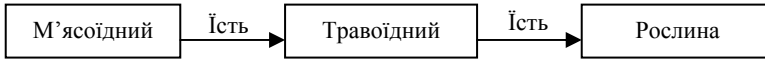


Рисунок 3.1 – Простий харчовий ланцюг

В нижній частині харчового ланцюгу перебувають рослини. Вони одержують енергію з навколишнього середовища (дощу, ґрунту й сонця). Наступний рівень займають травоядні тварини – для виживання вони поїдають рослини. На верхньому щаблі перебувають хижаки. У цій моделі хижаки поїдають травоядних тварин, щоб вижити.

Щоб змоделювати простий харчовий ланцюг, необхідно визначити деякі параметри: навколишнє середовище (фізичний простір, у якому взаємодіють агенти), самих агентів (а також їхнє сприйняття й поведінки в середовищі) і групу правил, які визначають, як і коли відбувається взаємодія.

3.2.2.1 Опис моделі

Модель простого харчового ланцюгу складається із середовища й трьох типів особин. Рослини являють собою нерухоме джерело їжі для травоядних тварин. Травоядні тварини є мігруючими агентами, які певним чином сприймають навколишнє середовище і їдять рослини. Іншими мігруючими агентами в середовищі є хижаки, що поїдають травоядних тварин. Хижаки можуть їсти тільки травоядних, а травоядні можуть їсти тільки рослини. Якщо який-небудь агент живе в середовищі певний час і не одержує їжі, він гине від голоду. Коли агент поглинає достатню кількість їжі, він може розмножуватися. Таким чином, у середовищі створюється новий агент певного типу. Відбувається еволюція, при якій мутує мозок агенту (проста нейронна мережа).

Важливо відзначити, що агенти спочатку не знають, як потрібно виживати в середовищі. Вони не знають, що поїдання їжі дозволить їм прожити довше. Також вони не знають, що повинні уникати тих, хто їх їсть. Агенти повинні освоїти всі ці знання за допомогою еволюції.

3.2.2.2 Навколишнє середовище

Агенти живуть у світі, побудованому за принципом сітки, грані якої з'єднані за аналогією з тороїдом. Якщо агент переміщається за грань у певному напрямку, він з'являється на іншій стороні (рис. 3.2).

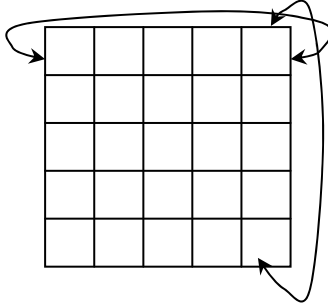


Рисунок 3.2 – Світ у вигляді сітки, побудованої за принципом тороїду

Рослини займають унікальні чарунки в середовищі, однак декілька агентів можуть займати одну чарунку (травоїдна тварина й/або хижак).

3.2.2.3 Анатомія агенту

Агент є генетичною особиною. Він може бути тільки певного типу (травоїдним або хижак), але метод вивчення навколишнього середовища та порядок дій для всіх агентів однакові (рис. 3.3). Агента можна розглядати як просту систему з набором входів (його відчуттям світу), реакцій на навколишній світ (його мозком) та дій.

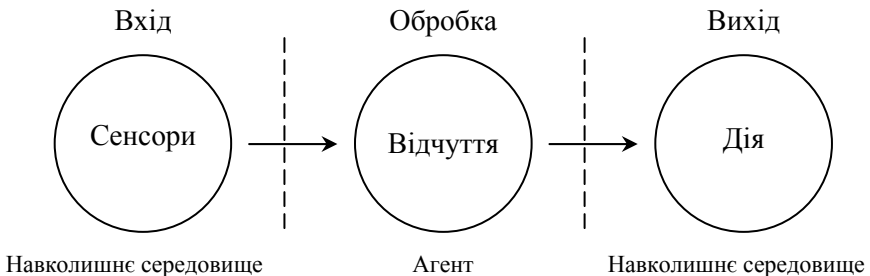


Рисунок 3.3 – Модель систем з агентами

Як відображено на рис. 3.3, агент складається із трьох окремих частин. Це сенсори, відчуття (визначення того, яку дію вибрати) і дія. Модель агента реагує на навколишнє середовище. Агенти не можуть планувати й навчатися. Навіть у такій простій моделі навчання відбувається за принципом, що називається еволюцією Ламарка. При відтворенні характеристики батька передаються потомству.

Сенсори

Агенти можуть відчувати, що відбувається навколо них у середовищі. Однак агент не бачить все середовище, він реагує тільки на групу чарунок навколо нього (рис. 3.4). Локальне середовище, що може відчувати агент, розділене на чотири окремі області. Найближча до агенту область називається *областю близькості*, саме в цій області агент може виконувати певні дії (наприклад, з'їсти інший об'єкт). Область поперед агента (5 чарунок) називається *фронтом*.

	Фронт	Фронт	Фронт	Фронт	Фронт	
	Ліворуч	Близькість	Близькість	Близькість	Праворуч	
	Ліворуч	Близькість	Агент	Близькість	Праворуч	

Рисунок 3.4 – Область відчуття агенту. Агент “дивиться” на північ

Агент може визначати вид об'єктів у полі зору. Тому для чотирьох областей пропонуються три числа, які дозволяють

ідентифікувати типи наявних об'єктів (рослини, трав'яні й хижак), тобто всього дванадцять входів.

Активатори

Агент може виконувати обмежене число дій у середовищі: переміститися на одну чарунку (у заданому напрямку), повернутися ліворуч або праворуч або з'їсти об'єкт, що перебуває в області близькості. Дія, яку виконує агент, визначається його мозком при оцінці входів, отриманих на рівні сенсорів.

Мозок агента

Мозок агента може бути однією з численних комп'ютерних конструкцій. Існуючі симуляції штучного життя використовують принцип кінцевих автоматів, системи класифікації або нейронні мережі. Щоб зберегти аналогію з біологічною мотивацією, при моделюванні харчових ланцюгів використовується проста нейронна мережа, побудована за принципом “переможець отримує все”. Входи такої мережі u_i інтерпретуються наступним чином:

- u_0 – трав'яне на передньому плані;
- u_1 – хижак на передньому плані;
- u_2 – рослина на передньому плані;
- u_3 – трав'яне ліворуч;
- u_4 – хижак ліворуч;
- u_5 – рослина ліворуч;
- u_6 – трав'яне праворуч;
- u_7 – хижак праворуч;
- u_8 – рослина праворуч;
- u_9 – близькість трав'яного;
- u_{10} – близькість хижака;
- u_{11} – близькість рослини.

Така нейронна мережа має 4 виходи o_j :

- o_0 – повернути ліворуч;
- o_1 – повернути праворуч;
- o_2 – переміститися;
- o_3 – з'їсти.

Входи сенсорів відображають кількість агентів, які перебувають у полі зору в певній області. Після того як всі входи були отримані із середовища, програма “просуває” їх через мережу до виходів. Це виконується за допомогою формули:

$$o_j = b_j + \sum_{i=0}^n u_i w_{ij}.$$

Тобто, для кожного виходу (o_j) мережі підсумовуються результати входів (u_i), які множаться на ваги з'єднань від входів до виходів (w_{ij}). Також додається зсув для виходу. У результаті на виходах мережі буде отриманий набір значень, які потім використовуються елементом дії агента.

Початкові ваги нейронної мережі агента вибираються випадковим чином. У результаті відтворення ваги повинні бути настроєні для виживання в середовищі.

Вибір дії агенту

Агент може виконувати одну дію із чотирьох можливих, що відповідають виходам нейронної мережі. Процес вибору дії полягає в пошуку вихідної чарунки з найбільшим значенням і виконанні відповідної дії. Це і є принцип “переможець отримує все” стосовно до мережі. Після виконання дії навколишнє середовище змінюється (якщо на нього впливали), і процес продовжується.

3.2.2.4 Енергія та метаболізм

Щоб вижити в навколишньому середовищі, агентам потрібна адекватна енергія. Якщо внутрішня енергія агента стає рівною нулю, агент вмирає. Агенти створюють енергію, з'їдаючи інші об'єкти в середовищі. Агент може з'їсти тільки той об'єкт, що допускається харчовим ланцюгом. Хижаки можуть їсти тільки травоядних, а травоядні – тільки рослини. Агенти також мають метаболізм, тобто коефіцієнтом поглинання енергії, що дозволяє їм зберігати життя. За кожну одиницю часу хижаки поглинають одну одиницю енергії, а травоядні – дві одиниці. Це означає, що для збереження життя травоядним потрібно з'їдати у два рази більше їжі, ніж хижакам. Хоча хижакам не потрібно так багато їжі, їм ще потрібно її знайти. Травоядні мають перевагу, що полягає в тому, що їхня їжа не переміщується по середовищу. Проте їм однаково потрібно відшукати свою їжу.

3.2.2.5 Відтворення

Якщо агент поглинає достатню кількість їжі, щоб досягти показника 90% від максимального рівня енергії, він допускається до участі у відтворенні. Відтворення дозволяє агентам, які змогли вижити в навколишньому середовищі, створити потомство (природний відбір). При створенні потомства агенти змінюють ваги своїх нейронних мереж за допомогою випадкової мутації. Навчання в середовищі недоступно, однак те, що агент може відтворювати себе, означає, що його нейронна мережа буде передана його дитині. Це повторює принцип еволюції Ламарка, оскільки характеристики агента передаються його потомству (дитина успадковує нейронну мережу свого батька).

Відтворення має наслідок, який полягає в тому, що: батько й дитина розділяють наявну енергію батька (енергія батька ділиться навпіл). При цьому агент не зможе безупинно відтворювати себе.

3.2.2.6 Смерть

Агент може вмерти двома способами: він не може знайти їжу й помирає від голоду або його з'їдає інший агент, що стоїть вище в харчовому ланцюгу. У кожному разі мертвий агент видаляється з моделі.

3.3 Завдання до роботи

3.3.1 Ознайомитися з літературою та основними теоретичними відомостями за темою роботи.

3.3.2 Розробити за допомогою пакету Matlab програмне забезпечення, що моделює штучне життя (модель харчового ланцюгу). При розробці програмного забезпечення врахувати твердження, наведені в п.3.3.2.1–3.3.2.5.

3.3.2.1 Штучний світ моделюється за допомогою сітки розмірності $N \times N$.

3.3.2.2 Існує всього 3 типи агентів (на протязі всього процесу моделювання):

1) рослини (при цьому в процесі моделювання, коли одну рослину з'їдають, має моделюватися поява нової рослини);

2) травоядні: вони мають становити максимально 50% від максимального можливого значення агентів і не можуть бути в кількості, меншій за 25% від максимального можливого значення агентів. При цьому травоядні можуть їсти рослини, в той же час вони є їжею для 3-го типу агентів – м'ясоїдних;

3) м'ясоїдні: обмеження на їхню кількість такі ж, як і для травоядних.

3.3.2.3 Агенти, що моделюють травоядних та м'ясоїдних, мають бути реалізовані у вигляді структури, що містить наступні поля:

- тип;
- енергія;
- вік – кількість ітерацій, на протязі яких жив агент;
- покоління: спочатку встановлюється в 1. Збільшується, якщо агент є нащадком іншого (тобто збільшується в процесі розмноження);
- координата X;
- координата Y;
- напрям погляду агенту: може бути північним, східним, західним або південним;
- входи нейромережі: на входи поступають кількості інших агентів відповідно до їх положення відносно даного агенту. Тобто на вхід 1 поступає кількість травоядних на передньому плані, на вхід 2 – кількість м'ясоїдних на передньому плані, на вхід 3 – кількість рослин на передньому плані, на входи 4-6 – кількості ті ж агентів, що і для входів 1-3, але вже ліворуч, 7-9 – праворуч, 10-12 – у безпосередній близькості;
- ваги нейромережі, які вибираються випадковим чином один раз при ініціалізації моделювання;
- зміщення для нейромережі – вибираються випадковим чином при ініціалізації;
- 4 виходи нейромережі, значення яких розраховуються за наведеною в основних теоретичних відомостях формулою. Значення виходів відповідають можливим діям агенту: повернути ліворуч (тобто лише змінити напрям погляду), повернути праворуч, рух вперед відповідно до напрямку, з'їсти іншого агента. При цьому, якщо було прийнято рішення з'їсти іншого агента, то необов'язково хтось буде з'їденим, оскільки дане рішення може бути помилковим. Обирається та дія, значення якої є найбільшим.

3.3.2.4 При моделюванні штучного життя вважати, що при поїданні рослини отримується енергія, вдвічі менша за поїдання травоядного;

3.3.2.5 Зберігати наступну статистику:

- кількість агентів відповідних типів на даній ітерації;
- максимальний вік за весь час моделювання агентів відповідних типів;
- кількість розмножень агентів для кожного типу.

3.3.3 Обґрунтовано сформувати набір даних для тестування розробленого програмного забезпечення. Виконати тестування. Результати тестування оформити в вигляді таблиць та графіків.

3.3.4 Оформити звіт з роботи.

3.3.5 Відповісти на контрольні питання.

3.4 Зміст звіту

3.4.1 Тема та мета роботи.

3.4.2 Короткі теоретичні відомості.

3.4.3 Текст розробленого програмного забезпечення з коментарями.

3.4.4 Обґрунтовано обраний набір даних для обробки (якщо він великий – навести фрагмент).

3.4.5 Результати роботи програмного забезпечення (таблиця порівняльного аналізу з поясненнями, графіки).

3.4.6 Висновки, що містять відповіді на контрольні запитання, а також відображують результати виконання роботи та їх критичний аналіз.

3.5 Контрольні питання

3.5.1 Що таке штучне життя? З якою метою ведуться дослідження в галузі штучного життя?

3.5.2 Яким чином пов'язані штучне життя та синтетична наука про поведінку?

3.5.3 В чому полягає сутність моделювання харчових ланцюгів?

3.5.4 Проаналізуйте параметри, які визначаються при моделюванні харчових ланцюгів.

3.5.5 Дайте визначення понять: область близькості агенту, фронт агенту.

3.5.6 З якою метою використовуються сенсори та активатори?

3.5.7 Які математичні засоби можуть бути використані для моделювання мозку агента?

3.5.8 Які дії може виконувати агент?

3.5.9 Що таке енергія та метаболізм?

3.5.10 При виконанні якої умови агент допускається до відтворення?

3.5.11 Якими способами агент може загинути?

4 ЛАБОРАТОРНА РОБОТА № 4 ПАРАЛЕЛЬНЕ МУЛЬТИАГЕНТНЕ ПРОГРАМУВАННЯ

4.1 Мета роботи

4.1.1 Вивчити основні можливості сімейства мов декларативного агентного програмування Golog та зокрема MIndiGolog.

4.1.2 Навчитися розв'язувати задачі на основі паралельного мультиагентного програмування за використання MIndiGolog.

4.2 Основні теоретичні відомості

4.2.1 Сімейство мов Golog та розширення MIndiGolog

Golog (Algol in logic) – це мова декларативного агентного програмування, що є стандартним підходом до визначення складних форм поведінки під час ситуаційного числення. Відзначається широким спектром застосування та великою кількістю розширень для забезпечення додаткової функціональності. Під Golog розуміють стандартне сімейство мов програмування, що базується на даному принципі і включає ConGolog, IndiGolog.

Ситуаційне числення використовується для представлення середовищ, що динамічно змінюються, у яких будь-які зміни є результатом виконання деяких дій. Основні елементи ситуаційного числення:

- дії, які можуть бути виконані у середовищі, визначаються функціональним символом (наприклад, *walk (loc)* – пройти до місця *loc*);

- ситуації, які відзначаються можливою історією середовища (S_0 – початкова ситуація перед виконанням будь-яких дій, а $do(a, s)$ – ситуація, яка дозволяє отримати результат від виконання дії a у ситуації S);

- змінні, які описують стан середовища (наприклад, *close-To(x, y, s)*: x близький до y у ситуації s).

Прикладом ситуації може слугувати наступна: $do(put(A,B), do(put(B,C), S_0))$ – ситуація, коли А поклали на В після того, як В поклали на С у деякій початковій ситуації S_0 .

Виконати програму на Golog означає знайти таку послідовність примітивних дій, що виконання їх, розпочате у деякій початковій ситуації S призведе до ситуації S' , тобто $Do(A, S, S')$, при чому дана послідовність дій дозволить роботу виконати відповідні дії у своєму середовищі.

Щоб запрограмувати агента, використовуючи Golog, необхідно визначити ситуаційне числення для кожної дії та програму, що складається з дій, об'єднаних такими програмними конструкторами як if-then-else, цикл while та недетермінований вибір. У таблиці 4.1 представлено стандартні оператори сімейства мов Golog.

Таблиця 4.1 – Оператори, що використовуються у Golog та його інтерпретаціях

Оператор	Значення
Nil	Порожня програма
a	Виконати дію a
$u?$	Виконати, якщо умова u вірна
$\delta_1 : \delta_2$	Виконати δ_1 , після чого δ_2
$\delta_1 \delta_2$	Виконати δ_1 або δ_2
$\pi(x, \delta(x))$	Недетермінований вибір аргументів для δ
δ^*	Виконати δ 0 або більше разів
$if\ u\ then\ \delta_1\ else\ \delta_2$	Якщо u справедливо, то виконати δ_1 , інакше – δ_2 .
$while\ u\ do\ \delta$	Виконати δ , доки справедливо u
$proc\ P(\vec{x})\sigma(\vec{x})end$	Визначення процедури
$\delta_1 \parallel \delta_2$	Паралельне виконання
$\delta_1 << \delta_2$	Паралельне виконання з пріоритетом
$\delta^{\parallel}$	Паралельні ітерації
$\Sigma(\delta)$	Планувати виконання офлайн

MIndiGolog – мультиагентний IndiGolog, який направлений на те, щоб надати можливість команді агентів паралельно планувати виконання та виконувати розподіляємо програму на Golog.

Для кращого моделювання динаміки мультиагентної команди у MIndiGolog інтегровано 3 розширення ситуаційного числення, що використовується у семантиці IndiGolog. Дані розширення дозволяють агентам представляти час, виконувати дії паралельно та виконувати природні дії реальним чином. Введення часу дозволяє координувати дії агентів. Концепція виконання природних дій дозволяє агентам передбачувати поведінку інших учасників команди та середовища.

Існує два базові підходи до використання Golog з мультиагентним виконанням.

Перший підхід полягає в тому, що поведінка загальної системи визначається як паралельне виконання програм окремих агентів:

$$\delta = \delta_{agt1} \parallel \delta_{agt2} \parallel \dots \parallel \delta_{agtN}.$$

Другий підхід до використання Golog для мультиагентного виконання програм, що застосовується у MIndiGolog, полягає в тому, що всі агенти взаємодіють з метою спільного виконання однієї розподіляємої програми. У такому випадку програма представляє паралельне виконання декількох завдань:

$$\delta = \delta_{task1} \parallel \delta_{task2} \parallel \dots \parallel \delta_{taskN}.$$

4.2.2 Розв’язання задачі мультиагентного програмування на основі MIndiGolog

Розглянемо задачу мультиагентного програмування. Іванна запросила до себе на обід гостей. Добрий інженер, але посередній кухар, вона вирішила створити команду роботів-кухарів, щоб допомогти їй приготувати їжу. Їй необхідно запрограмувати роботів для виконання обов’язків, при чому роботи мають автоматизовано розподіляти роботу між собою таким чином, що вони можуть діяти незалежно, де це можливо, та синхронізувати свої дії, де необхідно. Обід складається з декількох страв, які декілька роботів-кухарів мають приготувати на одній кухні, використовуючи наявний набір кухонного приладдя та інгредієнтів.

Кожен кухар представлений окремим агентом, наприклад:

$\forall agt : agt = Hemingway \vee agt = Marquez \vee agt = Remarque.$

Різноманітні типи інгредієнтів та посуду представлені окремими термами (наприклад, *Lettuce*, *Bowl*). Індивідуальні об'єкти даних типів мають назви: *Lettuce1*, *Bowl2*.

Доступні дії: *release*, *acquire*, *placeIn*, and *transfer*, що дозволяють відповідно вивільнити, заволодіти об'єктом, помістити у нього та перемістити з нього вміст.

Існує два завдання: *chop(cont)* – нарізати вміст посудини та *mix(cont, t)* – перемішати вміст посудини за даний час. При цьому вмістом посудини є набір продуктів, які поміщені у неї.

Програма, що розв'язує дану задачу на основі використання MIndiGolog, складається з наступних файлів:

- *sitcalc.pl* – деякі базові предикати *sitcalc*;
- *midigolog.pl* – семантика MIndiGolog;
- *domain.pl* – аксіоматизація приготування;
- *program.pl* – деякі програмні визначення (колекція корисних кулінарних процедур);
- *main.pl* – головна вхідна точка програми.

Для виконання даної програми необхідно спочатку інсталиювати SWI Prolog та запустити інтерпретатор MIndiGolog, після чого запустити SWI Prolog та завантажити *main.pl*, двічі клікнувши на файл '*main.pl*' (у випадку використання Windows) або виконавши наступні команди (у випадку використання консольної версії):

?-consult(main).

?-main(X).

Друга команда дозволяє виконати головний предикат.

Після виконання даної програми на екрані буде отримано роздруковку дій, виконаних агентами:

```
do[acquire_object(thomas,lettuce1),acquire_object(richard,tomato1),
acquire_object(harriet,egg1)] at 1
do[acquire_object(thomas,board1),acquire_object(richard,carrot1),
acquire_object(harriet,bowl2)] at 2
do[place_in(thomas,lettuce1,board1),acquire_object(richard,board2),
place_in(harriet,egg1,bowl2)] at 3
do[begin_task(thomas,chop(board1)),place_in(richard,carrot1,board2),
release_object(harriet,bowl2)] at 4
do[begin_task(richard,chop(board2)),end_task(thomas, chop(board1))]] at 7
do [acquire_object(thomas,bowl1),end_task(richard,chop(board2))]] at 12
```

do [transfer(thomas,board1,bowl1),acquire_object(richard,flour1)] at 13
 do [release_object(thomas,board1),acquire_object(richard,bowl2)] at 14
 do [release_object(thomas,bowl1),place_in(richard,flour1,bowl2)] at 15
 do [acquire_object(richard,board1)] at 16
 do [place_in(richard,tomato1,board1)] at 17
 do [begin_task(richard,chop(board1))] at 18
 do [end_task(richard,chop(board1))] at 23
 do [acquire_object(richard,bowl1)] at 24
 do [transfer(richard,board1,bowl1)] at 25
 do [release_object(richard,board1)] at 26
 do [release_object(richard,bowl1)] at 27
 do [acquire_object(richard,bowl1)] at 28
 do [transfer(richard,board2,bowl1)] at 29
 do [release_object(richard,board2)] at 30
 do [release_object(richard,bowl1)] at 31
 do [acquire_object(thomas,bowl1),release_object(richard,bowl2)] at 32
 do [begin_task(thomas,mix(bowl1,1)),acquire_object(richard,sugar1)] at 33
 do [acquire_object(richard,bowl2),end_task(thomas,mix(bowl1,1))] at 34
 do [release_object(thomas,bowl1),place_in(richard,sugar1,bowl2)] at 35
 do [release_object(richard,bowl2)] at 36
 do [acquire_object(thomas,bowl2)] at 37
 do [begin_task(thomas,mix(bowl2,5))] at 38
 do [end_task(thomas,mix(bowl2,5))] at 43
 do [release_object(thomas,bowl2)] at 44
 do [acquire_object(thomas,oven1)] at 45
 do [acquire_object(thomas,bowl2)] at 46
 do [place_in(thomas,bowl2,oven1)] at 47
 do [set_timer(thomas,cakeTimer,35)] at 48
 do [ring_timer(cakeTimer)] at 83
 do [acquire_object(thomas,board1)] at 84
 do [transfer(thomas,oven1,board1)] at 85
 do [release_object(thomas,oven1)] at 86
 do [release_object(thomas,board1)] at 87

Дії, які виконуються паралельно, можливі, якщо вони можливі індивідуально і якщо ні яка пара дій не є конфліктуючою:

$$Poss(c\#t;s) \equiv \forall a;a' \in c :$$

$$Poss(a\#t;s) \wedge Poss(a'\#t;s) \wedge \neg Conflicts(a;a';s).$$

Дії конфліктують, якщо вони виконуються одним агентом або якщо виникає необхідність заволодіти одним об'єктом різними агентами одночасно:

$$\text{Conflicts}(a; a'; s) \equiv \text{actor}(a) = \text{actor}(a')$$

$$\forall \exists \text{agt}; \text{agt}', \text{obj} : a = \text{acquire}(\text{agt}, \text{obj}) \wedge a' = \text{acquire}(\text{agt}', \text{obj}).$$

Деякий процес може бути заблоковано на період часу, доки інші процеси не будуть виконані і не розблокують його.

4.3 Завдання до роботи

4.3.1 Ознайомитися з літературою та основними теоретичними відомостями за темою роботи.

4.3.2 Виконати програму, що розв'язує задачу приготування обіду (4.2.2) на основі використання декількох агентів, детально розглянувши отримані результати та приділивши особливу увагу паралельному виконанню дій агентами.

4.3.3 Детально ознайомитись з текстом програми, приділивши особливу увагу можливостям, які надає MIndiGolog порівняно зі стандартною мовою Golog, а також тому, яким чином на MIndiGolog виконується визначення рецептів окремих страв, як агенти виконують елементарні дії, беручи контроль над об'єктом, звільняючи його, встановлюючи таймер тощо, яким чином агенти виконують свої дії паралельно і як розв'язуються конфліктні ситуації, коли декілька агентів намагаються взяти контроль над одним об'єктом в той самий момент часу.

4.3.4 Модифікувати відповідні файли програми (main.pl, domain.pl, mindigolog.pl, program.pl та sitcalc.pl за необхідністю), щоб в результаті виконання програми кухарі приготували яєчню з помідорами та салат. Для приготування яєчні з помідорами можна використати наступний рецепт: нарізати помідори, розбити яйця, зробити суміш з яєць, помідорів та солі, засмажити протягом 5 хвилин та викласти на тарілку. Для приготування салату можна використати салат, помідори та моркву, які необхідно нарізати, а потім ретельно перемішати.

4.3.5 Вивести на екран інгредієнти та короткий рецепт приготування кожної зі страв (наприклад, у такому вигляді: “Dish1: mixed([chopped(lettuce1), chopped(tomato1),chopped(carrot1)],1)”).

4.3.6 Оформити звіт з роботи.

4.3.7 Відповісти на контрольні питання.

4.4 Зміст звіту

4.4.1 Тема та мета роботи.

4.4.2 Короткі теоретичні відомості.

4.4.3 Текст розробленого програмного забезпечення з коментарями.

4.4.4 Результати роботи програмного забезпечення.

4.4.5 Висновки, що містять відповіді на контрольні запитання, а також відображують результати виконання роботи та їх критичний аналіз.

4.5 Контрольні питання

4.5.1 Що таке Golog?

4.5.2 Чим відрізняються дії від ситуацій в термінах ситуаційного числення?

4.5.3 Що означає вираз “виконати програму на Golog”?

4.5.4 Яким чином виконується програмування агенту на Golog?

4.5.5 Які стандартні оператори використовуються у Golog та його інтерпретаціях?

4.5.6 Які підходи до використання Golog з мультиагентним виконанням розрізняють?

4.5.7 Які особливості має MIndiGolog порівняно з Golog?

4.5.8 У чому полягає принцип використання Golog для мультиагентного програмування, що застосовується у MIndiGolog?

4.5.9 У яких випадках можливо виконати дії, які мають бути виконані паралельно?

4.5.10 Які дії називаються конфліктуючими?

4.5.11 Як розв’язуються конфліктуючі ситуації?

5 ЛАБОРАТОРНА РОБОТА № 5 НЕДЕТЕРМІНОВАНЕ МУЛЬТИАГЕНТНЕ ПРОГРАМУВАННЯ

5.1 Мета роботи

5.1.1 Вивчити можливості недетермінованого програмування на основі MIndiGolog.

5.1.2 Навчитися розв'язувати задачі на основі недетермінованого мультиагентного програмування.

5.2 Основні теоретичні відомості

У застосуваннях, які дозволяють розв'язувати розподілені задачі, команда агентів повинна тісно взаємодіяти з метою досягнення спільної мети. Такі команди часто можуть бути представлені одним агентом з розподіленими можливостями, що призводить до двох-етапної методики програмування таких команд: пристосувати формалізм одно-агентного програмування для визначення завдань, які необхідно виконати, після чого забезпечити стратегію координації, що дозволяє команді виконувати специфікацію паралельно. Даний підхід успішно застосовується такими платформами як STEAM, SharedPlans та TAEMS, які визначають завдання за допомогою варіювання формалізму ієрархічних мереж завдань, альтернативою яким є спосіб виконання програм на високому рівні, що реалізовано у мові програмування Golog і який набирає популярність.

Під програмою високого рівня розуміють програму, примітивні компоненти якої є діями, визначеними для даного домену, при чому дані дії з'єднуються стандартними програмними конструкціями і можуть містити недетерміновані оператори.

Нехай $Do(a, s, s')$ – складна дія, при чому виконання дії a у ситуації s призводить до ситуації s' . Семантику складних дій можна представити наступним чином:

- примітивні дії a :

$$\overset{def}{Do(a, s, s')} = Poss(a, s) \wedge s' = do(a, s);$$

– послідовність:

$$\overset{def}{Do([a_1; a_2], s, s')} = (\exists s'') Do(a_1, s, s'') \wedge Do(a_2, s'', s');$$

– недетермінований вибір з двох дій:

$$\overset{def}{Do([a_1; a_2], s, s')} = Do(a_1, s, s') \vee Do(a_2, s, s');$$

– недетермінований вибір параметрів дій:

$$\overset{def}{Do((\pi x)a(x), s, s')} = (\exists x) Do(a(x), s, s');$$

– умовні дії:

$$\overset{def}{if\ p\ then\ a_1\ else\ a_2} = [p?; a_1] || [\neg p?; a_2];$$

– цикли:

$$\overset{def}{while\ p\ a} = [p?; a]^*; \neg p?;$$

– тести:

$$\overset{def}{Do(p?, s, s')} = p[s] \wedge s' = s';$$

– недетерміновані ітерації:

$$\overset{def}{Do(a^*, s, s')} = (\forall P)[(\forall s_1) P(s_1, s_1)] \wedge [(\forall s_1, s_2, s_3) P(s_1, s_2) \wedge Do(a, s_2, s_3) \supset P(s_1, s_3)] \supset P(s, s').$$

Деякі дії можуть бути не виконуваними в даних ситуаціях. Наприклад, неможливо покласти предмет, якщо агент не тримає його в руках в даний момент. Тому дії мають передумови – достатні умови, яким повинні відповідати стани середовища перед тим, як дії будуть виконані в цих станах.

Обмеження на виконання дій змодельовані за допомогою літералів форми $Poss(a, s)$, де a – дія, s – ситуація і $Poss$ – спеціальний бінарний предикат, що позначає можливість виконання дії в заданій ситуації. Наприклад, умова про те, що покласти предмет можна тільки в тому випадку, якщо ти його тримаєш, моделюється таким чином:

$$Poss(drop(o), S) \longleftrightarrow is_carrying(o, S).$$

Багато операцій у Golog є недетермінованими і можуть бути виконані декількома різними шляхами. Задачею агента є розроблення детермінованого виконання програми, послідовності дій, які можуть бути легально виконані у середовищі.

Визначення завдань для агентів-кухарів вимагає комбінації можливостей. Так є багато процедурних знань про рецепти, які мають бути закодовані так точно, як тільки можливо. Проте є багато деталей про те, хто який крок виконує, або коли саме вони мають виконуватись, і такі деталі не мають бути точно визначені програмістом.

Сімейство мов Golog забезпечує потужний формалізм для визначення завдань. Контрольований недетермінізм, що забезпечує дане сімейство мов, може бути використаний для приховування цих деталей, при цьому зберігаючи їх процедурну природу.

Розглянемо декілька прикладів визначення мультиагентного завдання на Golog.

Приклад 1. Приготування салату командою агентів.

```
procMakeSalad(dest)
  π (agt;ChopTypeInto(agt;Lettuce; dest)) ||
  π (agt;ChopTypeInto(agt;Carrot; dest)) ||
  π (agt;ChopTypeInto(agt; Tomato; dest))] ;
  π (agt; [acquire(agt; dest) ;
  beginTask(agt; mix(dest; 1)) ;
  endTask(agt; mix(dest; 1)) ;
  release(agt; dest)]) end
```

Дана програма у своїх інструкціях на основі недетермінованих операторів визначає виконання завдання приготування одного салату таким чином, що “Хтось нарізає салат, хтось нарізає моркву та хтось нарізає помідор. Потім хтось перемішує їх”. Необхідно зазначити, що оператори програми дозволяють нарізати три інгредієнти окремо, при цьому кожне під-завдання може бути виконано будь-яким доступним агентом.

Приклад 2. Нарізання об’єкту деякого типу.

```
procChopTypeInto(agt; type; dest)
  [π (board; IsType(board;Board)? ;
  π (obj; IsType(obj; type)? ;
  acquire(agt; board) ;
  acquire(agt; obj) ;
```

```

placeIn(agt; obj; board) ;
beginTask(agt; chop(board)) ;
endTask(agt; chop(board)) ;
acquire(agt; dest) ;
transfer(agt; board; dest) ;
release(agt; board) ;
release(agt; dest)))] end

```

У даному прикладі агент має вибрати та зайняти об'єкт визначеного типу, так само як і дошку для нарізання. Після цього він має покласти об'єкт на дошку, нарізати його та перемістити до посудини.

Програми не визначають, який агент яке завдання має виконувати. Бібліотека процедур може бути скомпонована дуже гнучко щодо визначення поведінки агентів, а результуюча програма може бути віддана на виконання будь-якій команді агентів. Такий механізм є дуже зручним для моделювання взаємодії агентів. Однак, використання в даних цілях мови Golog не є оптимальним, адже за такого варіанту тільки один агент може виконувати одну дію в будь-який момент часу, в той час як можливості усіх інших агентів не використовуються. MIndiGolog дозволяє виконувати всі можливі дії паралельно, ініціюючи виконання декількох дій у будь-який момент часу.

Як тільки програма управління досягає точки недетермінованого вибору, викликається планувальник для визначення того, що необхідно робити далі. Таким чином, програміст може визначати частково задані контролери та спиратися на використання вбудованих планувальників для виконання остаточного вибору плану управління.

5.3 Завдання до роботи

5.3.1 Ознайомитися з літературою та основними теоретичними відомостями за темою роботи.

5.3.2 Використовуючи текст програми, що розв'язує задачу приготування обіду на основі використання декількох агентів, отриманий в лабораторній роботі № 4, модифікувати відповідні файли програми (main.pl, domain.pl, mindigolog.pl, program.pl та sitcalc.pl за необхідністю) наступним чином:

5.3.2.1 Виконати приготування салату, тістечка та яєчні з помідорами паралельно.

5.3.2.2 Збільшити число кухарів.

5.3.2.3 Зменшити число інгредієнтів, доступних для приготування страв (наприклад, зменшити кількість наявних на кухні яєць), або навпаки збільшити число інгредієнтів, необхідних для приготування деяких страв (наприклад, використати більше яєць для приготування яєчні з помідорами);

5.3.2.4 Зменшити число наявних на кухні ножів або дошок, на яких виконується нарізання продуктів.

5.3.3 Виконати програму в різних режимах (за різних варіантів вимог за п. 5.3.2.2-5.3.2.4), з'ясувати, яким чином впливає зміна тих чи інших параметрів на отримані результати, та пояснити різницю в отриманих результатах.

5.3.4 Використовуючи текст розробленої програми, виконати наступні дії:

5.3.4.1 Привести приклади виконання недетермінованого вибору.

5.3.4.2 Пояснити, яким чином можна інтерпретувати дані оператори.

5.3.4.3 Продемонструвати на результатах виконання програми, як було виконано вибір планувальником.

5.3.5 Оформити звіт з роботи.

5.3.6 Відповісти на контрольні питання.

5.4 Зміст звіту

5.4.1 Тема та мета роботи.

5.4.2 Короткі теоретичні відомості.

5.4.3 Текст розробленого програмного забезпечення з коментарями.

5.4.4 Результати роботи програмного забезпечення в різних режимах з коментарями, що пояснюють різницю в отриманих результатах.

5.4.5 Висновки, що містять відповіді на контрольні запитання, а також відображують результати виконання роботи та їх критичний аналіз.

5.5 Контрольні питання

5.5.1 Що таке програма високого рівня?

5.5.2 Що таке недетермінований вибір?

5.5.3 Для чого необхідні недетерміновані оператори?

5.5.4 Що відбувається, коли програма управління досягає точки недетермінованого вибору?

5.5.5 Для чого необхідні обмеження на виконання дій?

5.5.6 У чому полягають переваги використання MIndiGolog для мультиагентного паралельного програмування з можливістю виконання недетермінованого вибору?

5.5.7 Чим відрізняється недетермінований вибір з двох дій від недетермінованого вибору параметрів дій?

5.5.8 Які конструкції називаються ітеративними, які умовними, а які недетермінованими?

ЖИТЕПАТҮҮҮ

1. Abraham A., Grosan G. *Swarm Intelligence in Data Mining*. – Berlin: Springer, 2006. – 267 p.
2. Beni G., Wang J. *Swarm Intelligence // Annual Meeting of the Robotics Society: Proceedings of Seventh International Conference*. – Tokyo: RSJ Press, 1989. – P. 425-428.
3. Bonabeau E., Dorigo M., Theraulaz G. *Swarm Intelligence: From Natural to Artificial Systems*. – New York: Oxford University Press, 1999. – 320 p.
4. Bullnheimer B., Hartl R.F., Strauss C. *Applying the ant system to the vehicle routing problem // Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. – Boston: Kluwer, 1998. – P. 109–120.
5. Colorni A., Dorigo M., Maniezzo V., Trubian M. *Ant system for job-shop scheduling // Belgian Journal of Operations Research, Statistics and Computer Science (JORBEL)*. – 1994. – №34. – P. 39–53.
6. Costa D., Hertz A. *Ants can colour graphs // Journal of the Operational Research Society*. – 1997. – №48. – P. 295–305.
7. Di Caro G., Dorigo M. *Two ant colony algorithms for best routing in datagram networks // Proceedings of the Tenth IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS'98) / Eds: Y. Pan, S.G. Akl, K. Li*. – Anaheim: IASTED/ACTA Press, 1998. – P. 541–546.
8. Dorigo M. *Optimization, Learning and Natural Algorithms*. – Milano: Politecnico di Milano, 1992. – 140 p.
9. Dorigo M., Gambardella L.M. *Ant colonies for the traveling salesman problem // BioSystems*. – 1997. – №43. – P. 73–81.
10. Eberhart R., Kennedy J. *A New Optimizer using Particle Swarm Theory // Symposium on Micro Machine and Human Science: Proceedings of the Sixth International Symposium on Micro machine and Human Science*. – Nagoya: IEEE Service Center, 1995. – P. 39-43.
11. Ferber J., Müller J.P. *Influences and reactions: A model of situated multiagent systems // Proceedings of the 2nd International Conference on Multi-Agent Systems (ICMAS-96)*. – Menlo Park: IEEE Computer Society Press, 1996. – P. 72-79.

12. Franks N.R. Teams in social insects: group retrieval of prey by army ants (*Eciton burchelli*, Hymenoptera: Formicidae) // Behavioral Ecology and Sociobiology. – №18. – 1986. – P. 425–429.
13. Gambardella L.M., Dorigo M. HAS-SOP: An hybrid ant system for the sequential ordering problem. – Lugano: CH, 1997. – P. 237–255.
14. Gambardella L.M., Taillard E., Agazzi G. Macs-vrptw: A multiple ant colony system for vehicle routing problems with time windows // New Methods in Optimisation. – McGraw-Hill, 1999. – P. 63–79.
15. Gasser L., Hill R.W. Coordinated problem solvers // Annual Review of Computer Science, – 1990. – №4. – P. 203–253.
16. Griffin D.R. Animal Minds. – Chicago: The University of Chicago Press, 2001. – 376 p.
17. Hackman J.R. Groups That Work (and Those That Don't). – San Francisco, California: Jossey-Bass, 1990. – 544 p.
18. Kennedy J., Eberhart R.C. Particle swarm optimization // Proceedings of the 1995 IEEE International Conference on Neural Networks, Vol. 4. – NJ: IEEE Press, 1995. – P. 1942–1948.
19. Leguizamón G., Michalewicz Z. A new version of Ant System for subset problems // Proceedings of the 1999 Congress on Evolutionary Computation (CEC'99). – New Jersey: IEEE Press, 1999. – P. 1459–1464.
20. Maniezzo V. Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. – Bologna: Università di Bologna, 1998 – 102 p.
21. Maniezzo V., Colomi A., Dorigo M. The ant system applied to the quadratic assignment problem. – Bruxelles: Université Libre de Bruxelles, 1994. – 24 p.
22. Michel R., Middendorf M. An ACO algorithm for the shortest common supersequence problem // New Methods in Optimisation. – McGraw-Hill, 1999. – P. 525–537.
23. Parunak H.V.D. “Go to the ant”: Engineering principles from natural multi-agent systems // Annals of Operation Research – 1997. – №75. – P. 69–101.
24. Passino K.M., Seeley T.D. Modeling and analysis of nest-site selection by honey bee swarms: The speed and accuracy trade-off // Behavioral Ecology and Sociobiology. – 2006. – №59. – P. 427–442.
25. Peer E., Engelbrecht A. Using Neighborhoods with the Guaranteed Convergence PSO // Swarm Intelligence Symposium:

Proceedings of the IEEE SIS-2003. – Indianapolis: IEEE Press, 2003. – P. 235–242.

26. Subbotin S., Oleynik A. Ant Colony Optimization for Feature Selection Based on Operations with Crisp Sets // Modern problems of radio engineering, telecommunications and computer science: Proceedings of the XI International Conference TCSET'2008 (19–23 February 2008). – Lviv: Publishing house of Lviv Polytechnic, 2008. – P. 117–118.

27. Subbotin S., Oleynik A. Modifications of Ant Colony Optimization Method for Feature Selection // The experience of designing and application of CAD systems in Microelectronics: Proceedings of the IX International Conference CADSM-2007 (20–24 February 2007). – Lviv: Publishing house of Lviv Polytechnic, 2007. – P. 493–494.

28. Wilson E.O. The Insect Societies. – Cambridge, Massachusetts: The Belknap Press, 1974. – 562 p.

29. Джонс М.Т. Программирование искусственного интеллекта в приложениях. – М.: ДМК Пресс, 2004. – 312 с.

30. Олейник Ал.А. Сравнительный анализ методов оптимизации на основе метода муравьиных колоний // Комп'ютерне моделювання та інтелектуальні системи: Збірник наукових праць / За ред. Д.М. Пізи, С.О. Субботіна. – Запоріжжя: ЗНТУ, 2007. – С. 147–159.

31. Олейник Ал.А. Сравнительный анализ модификаций метода муравьиных колоний для отбора признаков // Радіоелектроніка і молодь в ХХІ сторіччі: Матеріали 11-го міжнародного молодіжного форуму (10–12 квітня 2007 р.). – Харків: ХНУРЕ, 2007. – Ч.1. – С. 353.

32. Олейник Ал.А., Олейник Ан.А., Субботин С.А., Яценко В.К. Синтез моделей коэффициента упрочнения деталей авиадвигателей после алмазного выглаживания на основе оптимизационного подхода // Вісник двигунобудування. – 2005. – №3. – С. 25–30.

33. Пожужева И.С., Субботин С.А., Олейник А.А. Эволюционная оптимизация многомерных функций // Нові матеріали і технології в металургії та машинобудуванні. – 2006. – № 1. – С. 70–72.

34. Kelly R.F. Asynchronous Multi-Agent Reasoning in the Situation Calculus: Ph.D. Thesis. – Melbourne, 2008. – 226 p.