Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Сучасні концепції створення інтелектуальних інформаційних систем

Комп'ютерний практикум

Навчальний посібник

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня доктора філософії
за освітньою програмою «Інформаційні системи та технологій»
спеціальності 126 Інформаційні системи та технологій

Укладачі: П. І. Кравець, В. М. Шимкович

Електронне мережеве навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2024

УДК 681.3
І

Укладачі: *Кравець Петро Іванович*, канд. техн. наук, доц.
Шимкович Володимир Миколайович, канд. техн. наук, доц.

Рецензент Ткач М. М., канд. техн. наук, доц.,
КПІ ім. Ігоря Сікорського, факультет інформатики та обчислювальної техніки, кафедра інформаційних систем та технологій

Відповідальний редактор *Полторак В.П.*, канд. техн. наук, доц.

Рекомендовано Методичною радою ФІОТ КПІ ім. Ігоря Сікорського
(протокол № 10 від 21.06.2024 р.)
Рекомендовано Вченою радою факультету інформатики та обчислювальної техніки
(протокол № 12 від 24.06.2024 р.)

І **Сучасні концепції створення інтелектуальних інформаційних систем.** Лабораторний практикум [Електронний ресурс] : навч. посіб. для здобувачів ступеня д-ра філософії за освіт. програмою «Інформаційні системи та технологій» спец. 126 Інформаційні системи та технологій / КПІ ім. Ігоря Сікорського ; уклад.: П. І. Кравець, В. М. Шимкович. – Електрон. текст. дані (1 файл). – Київ : КПІ ім. Ігоря Сікорського, 2024. – 75 с.

Навчальний посібник охоплює теоретичний матеріал та практичні завдання, які необхідні для виконання лабораторного практикуму з дисципліни «Сучасні концепції створення інтелектуальних інформаційних систем»». В посібнику наводяться рекомендації по використанню програмних продуктів і засобів для виконанню робіт з наступної тематики: створення інтелектуальних інформаційних систем на основі методу мурашиних колоній, технології моделювання розвитку природних систем та інтелектуальних Web-агентів. Посібник може бути корисним студентам відповідних спеціальностей при вивченні дисциплін, пов'язаних з розробкою інформаційних систем на основі технологій штучного інтелекту, а також при виконанні бакалаврських робіт, курсових проектів, магістерських робіт, в яких використовуються відповідні технології.

УДК 681.3

Реєстр. № НП ХХ/ХХ-ХХХ. Обсяг Х,Х авт. арк.

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

ЗМІСТ

ВСТУП

Перед виконанням робіт практикуму студенти повинні ознайомитися з даними методичними вказівками та матеріалами рекомендованої літератури. Роботи виконуються на планових заняттях у відповідності з розкладом. Перед виконання роботи студенти проходять співбесіду, де повинні показати знання про мету роботи, про методи виконання кожного етапу роботи та по теорії тематики роботи. Після виконання роботи студенти надають викладачу оформлений звіт з текстами розроблених програм, файлами вхідних і вихідних даних, файлами програм, що виконуються.

Звіт має містити:

- титульний аркуш;
- варіант завдання, завдання і мету роботи;
- короткий опис теоретичних відомостей;
- інтерфейс користувача;
- текст програми, що виконується;
- вхідні та вихідні дані програми;
- змістовний аналіз отриманих результатів та висновки.

Для отримання заліку по кожній із робіт студент повинен продемонструвати викладачу результати роботи розроблених програм на конкретних прикладах і відповідно проаналізувати отримані результати. Залік з лабораторних робіт студент одержує після виконання і здачі всіх робіт.

РОБОТА № 1

Розробка інтелектуальної інформаційної системи на основі мурашиного алгоритму

Мета роботи: навчитися створювати інтелектуальні інформаційні системи на основі мурашиного алгоритму для розв'язування задач комбінаторної оптимізації.

Основні теоретичні відомості

Мурашині алгоритми досліджуються вченими із середини 1990-х років в рамках наукового напрямку Natural Computing – «Природні обчислення», що поєднує методи з природними механізмами прийняття рішень, а саме: Genetic Algorithms – генетичні алгоритми; Evolution Programming – еволюційне програмування; Neural Network Computing – нейромережеві обчислення; DNA Computing – ДНК обчислення; Cellular Automata – клітинні автомати; Ant Colony Algorithms – мурашині алгоритми. На сьогоднішній день мурашині алгоритми використовуються для оптимального вирішення 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 нестаціонарних системах, наприклад, трафіків у телекомунікаційних мережах [1,2].

Основу мурашиних алгоритмів складає емуляція процесу самоорганізації функціонування мурашиної колонії. Колонію мурах можна представити як багатоагентну систему, в якій кожен з агентів (мурах) функціонує самостійно по дуже елементарних правилах. При цьому майже примітивна поведінка агентів дозволяє організувати злагоджену та розумну поведінку всієї системи. Агенти спільно вирішують проблему й допомагають іншим агентам у подальшій оптимізації розв'язку вирішуваних завдань.

Методи мурах (Ant algorithm), або оптимізація за принципом мурашиної колонії, мають специфічні особливості, властиві мурахам, і використовують їх

для орієнтації у фізичному просторі. У методі мурашиних колоній передбачається, що навколишнє середовище являє собою закриту двовимірну графову 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 руху в мережі. Він підтримує список табу tList – список вузлів, які він вже відвідав, тобто агент повинен проходити через кожний вузол тільки один раз. Вузли в списку «поточної подорожі» tList розташовуються в тому порядку, у якому агент їх відвідував. Пізніше список використовується для визначення довжини шляху між пройденими вузлами.

Біологічні принципи поведінки мурашиної колонії. Мурах відносять до соціальних комах які живуть всередині організованого колективу – колонії. Кількість мурах в одній колонії коливається від 40 особин до кількох мільйонів. Такі колективні системи здатні вирішувати складні та динамічні задачі по виконанню спільної роботи, яка не могла би виконуватися окремими елементами системи окремо від інших в різноманітних середовищах без зовнішнього управління та контролю. В даному випадку такий феномен називають ройовим інтелектом, як спосіб кооперативної поведінки або стратегія виживання.

Основою поведінки мурашиної колонії є самоорганізація, яка дозволяє забезпечити досягнення загальної мети колонії, взявши за основу низькорівневу взаємодію. Варто зазначити, що колонія не має централізованого управління, і внаслідок цього виконується тільки обмін локальною інформацією між окремими особами шляхом непрямого обміну (прямий обмін – це візуальні та/або хімічні контакти), який і був покладений в основу мурашиних алгоритмів. Таким чином, в загальному випадку розглядаються сліпі мурахи, які не можуть відчувати близькість їжі.

Непрямий обмін являє собою рознесену в часі взаємодію, при якій одна особа змінює деяку частин середовища, а інші використовують залишену

інформацію пізніше, коли з нею (інформацією) стикаються. Біологами було встановлено, що в мурах така взаємодія відбувається за допомогою спеціальної хімічної речовини – феромону, який являє собою виділення спеціальних залоз і який відкладається і залишається на шляху під час переміщення мурах. Чим більше мурах рухається по деякому конкретному відрізку шляху, тим більша кількість феромону відкладається на ньому. Кількість даного феромону на шляху дозволяє мурахам визначити найкращий на даний момент шлях для руху, але його кількість з часом зменшується за рахунок випаровування і це приводить до зміни співвідношення кількості феромону на різних шляхах, що заставляє рухатись мурах по інших напрямках руху і власне визначає адаптивність поведінки мурах у виборі подальшого напрямку руху.

Опис мурашиних алгоритмів. Мурашині алгоритми є ймовірнісною жадною евристикою, де отримання рішень (східність алгоритму) гарантується, тобто в будь якому випадку можна отримати оптимальне рішення, проте швидкість цієї східності невідома. Покажемо принцип використання мурашиного алгоритму на вирішенні відомої задачі комівояжера, яка була першою комбінаторною задачею вирішеною мурашиними алгоритмами, оскільки вона легко інтерпретується з поведінкою мурах.

В мурашиних алгоритмах кожна мураха являє собою незалежний комівояжер, який вирішує свою окрему задачу. За одну ітерацію кожна мураха виконує повний маршрут комівояжера. Багатократність пошуку маршруту комівояжера забезпечується пошуком маршруту одночасно кількома мурахами, які починають свій рух з різних окремих точок мережі.

Кількість феромону, який відкладається мурахою на шляху є обернено пропорційною до довжини маршруту. Тобто при короткому маршруті буде відкладатися більше феромону на відповідних ребрах графу і більша кількість мурах буде їх вибирати для досягнення мети. Відкладений на коротких шляхах феромон виступає своєрідним підсилювачем, який забезпечує збереженість хороших та оптимальних маршрутів в глобальній пам’яті мурашника (І ці маршрути можуть бути покращені на наступних ітераціях алгоритму). Тобто

ймовірність включення ребра графа в маршрут мурахи пропорційна кількості феромону на цьому графі – це є основне правило позитивного зворотного зв’язку для задачі комівояжера.

З цього правила випливає інша складова самоорганізації – випадковість. Позитивний зворотній зв'язок можна описати як імітацію поведінки мурах з використанням феромонів – залишків слідів та переміщення по слідах. Тобто чим більше слідів буде залишено на ребрі графа, тим більше мурах буде по ній рухатись. Внаслідок таких дій на ребрі з’являються нові сліди, які приваблюють ще більшу кількість мурах.

Використовуючи позитивний зворотній зв'язок ми отримуємо передчасну східність рішень – випадок, коли всі мурахи рухаються одним і тим самим субоптимальним маршрутом. Щоб уникнути подібної ситуації використовують негативний зворотній зв'язок, який полягає в випаровуванні феромону. Варто зазначити, що час випаровування не повинен бути занадто тривалим, тому що виникає можливість східності популяції маршрутів до одного субоптимального рішення. І в той самий момент час випаровування не має бути занадто малим, тому що це призводить до швидкої втрати пам’яті колонії і відповідно до неузгодженої поведінки мурах. В мурашиній колонії узгоджена поведінка мурах має велике значення, тому що багато мурах досліджують різні ділянки простору рішень і передають отриманий досвід через зміни значень в пам’яті мурашника.

Для кожної з мурах переміщення з пункту I в пункт J залежить від таких складових: 1) сліду феромону; 2) пам’яті мурахи; 3) видимості,

Пам’ять мурахи - це перелік пунктів, які були відвідані мурахою і які відвідувати повторно заборонено. За допомогою цього переліку мураха не потрапить в один і той же самий пункт двічі. Описаний перелік росте в процесі пошуку маршруту при кожній ітерації алгоритму. Позначимо через $J_{i,k}$ перелік пунктів, які мураха k ще повинна відвідати і яка знаходиться в пункті i. $J_{i,k}$ – доповнення до пам’яті мурахи.

Видимістю описують величину, яка є зворотною до відстані: $\eta_{ij} = 1/D_{ij}$, тут $\boldsymbol{D}_{ij}$ – це відстань між пунктами *i* та *j*. Видимість – локальна статистична інформація, яка описує евристичне бажання мурахи перейти з пункту *j* в пункт *i*, і чим менша відстань між ними, тим більше бажання його відвідати.

Слід феромону на ребрі – це підтверджене мурашиним досвідом бажання відвідати пункт *j* з пункту *i*. Від видимості відрізняється тим що слід феромону є більш динамічною та глобальною інформацією, яка змінюється після кожної з ітерацій алгоритму і яка репрезентує набутий мурахою досвід. Позначимо $\tau_{ij}(t)$ кількість віртуального феромону на ребрі (*i, j*) на ітерації *t*. Тоді ймовірність переходу *k*-ї мурахи з пункту *i* в пункт *j* на *t*-й ітерації буде:

$$\begin{cases} P_{ij,k}(t) = \dfrac{\left[\tau_{ij}(t)\right]^{\alpha} \cdot \left[\eta_{ij}\right]^{\beta}}{\sum\limits_{l \in J_{i,k}} \left[\tau_{ij}(t)\right]^{\alpha} \cdot \left[\eta_{ij}\right]^{\beta}}, \quad якщо\ j \in J_{i,k}, \\ P_{ij,k}(t) = 0, \quad якщо\ j \notin J_{i,k}, \end{cases} \tag{1}$$

Тут α та β – параметри, які регулюються, і які задають вагу сліду феромону та видимості при виборі маршруту. При α=0 вибереться найближчий пункт. І це буде відповідати жадному алгоритму в теорії оптимізації. Якщо ж буде $\beta = 0$, то спрацює тільки підсилення феромону, і це призведе до швидкого виродження наявних маршрутів до одного субоптимального рішення.

Саме вибір пункту організовується наступним чином: кожен пункт має свій сектор з площею, яка описується в правилі (1), як визначення ймовірності вибору деякого маршруту. Далі для вибору пункту потрібно згенерувати випадкове число і тим самим визначити сектор.

Слід звернути увагу, що хоча описане вище правило не змінюється протягом ітерації, значення ймовірностей $P_{ij,k}(t)$ для мурах в одному і тому ж пункті можуть відрізнятися, оскільки $P_{ij,k}(t)$ є функцією від $J_{i,k}$ – списку ще не відвіданих пунктів мурахою *k*.

Після проходження маршруту кожна з мурах *k* залишає на ребрі (*i, j*) феромон, кількість якого описується наступним співвідношенням:

$$\Delta\tau_{ij,k}(t)=\begin{cases}\dfrac{Q}{L_k(t)}, якщо\ (i,j)\in T_k(t),\\ 0, якщо\ (i,j)\notin T_k(t),\end{cases} \quad (2)$$

де $T_k(t)$ – маршрут, який мураха k пройшла на ітерації t; Q – параметр, значення якого вибирається одного порядку з довжиною оптимального маршруту; $L_k(t)$ – довжина маршруту.

Щоб дослідити весь простір рішень потрібно виконувати випаровування феромону – зменшення в часі кількості відкладеного на попередніх кроках феромону. Тепер через $p\in[0,1]$ позначимо коефіцієнт випаровування феромону та визначемо правило оновлення феромону:

$$\tau_{ij}(t+1)=(1-p)\cdot\ \tau_{ij}(t)+\Delta\tau_{ij}(t)\ ,$$

де $\Delta\tau_{ij}(t)=\sum_{k=1}^{m}\Delta\tau_{ij,k}(t)$, m – кількість мурах в колонії.

Як правило на початку оптимізації кількість феромону береться рівним невеликому додатному числу τ_0. Сумарна кількість мурах в колонії залишається однаковою протягом виконання алгоритму. У випадку багаточисельної колонії виконується підсилення субоптимальних маршрутів, а коли мурах невелика кількість, з'являється ризик втрати злагодженої поведінки через недостатню взаємодію та швидке випаровування феромону.

Зазвичай кількість мурах, які приймають участь в алгоритмі, визначається кількістю пунктів, які потрібно відвідати, і, як правило, кожна мураха починає свій маршрут зі свого окремого пункту.

Модифікації мурашиного алгоритму. Мурашиний алгоритм може бути ефективним для рішення задач оптимізації, які допускають графову інтерпретацію, причому при збільшенні розмірності вирішуваних задач ефективність мурашиних алгоритмів зростає.

З метою зменшення часу реалізації алгоритму часто використовують модифікацію алгоритму з використанням «елітних мурах». Такі мурахи здатні підсилювати ребра найкращих маршрутів, які були знайдені на початку роботи алгоритму. Кількість феромону, який відкладається на ребрах найкращого

поточного маршруту $T+$, приймається рівним $Q/L+$, де $L+$ – довжина маршруту $T+$. Такий рівень феромону дозволяє мурахам досліджувати рішення, які містять кілька ребер найкращого на даний момент маршруту $T+$. Якщо в мурашнику є e «елітних» мурах, тоді ребра маршруту $T+$ будуть отримувати загальне підсилення $\Delta\tau_e = e \cdot Q/L^+$.

Досить хорошими шляхами покращення мурашиного алгоритму є адаптація параметрів з використанням бази нечітких правила та їх гібридизація, наприклад з генетичними алгоритмами. Така гібридизація може полягати в обміні найкращими рішеннями через деякі проміжки часу.

В мурашиних алгоритмах якість рішень залежить від параметрів правил відкладання та випаровування феромону, від налаштувань параметрів в ймовірнісно-пропорціональному правилі вибору шляху (1), та базується на поточній кількості феромону. Є припущення, що динамічне адаптаційне налаштування цих параметрів може дозволити покращити отримувані рішення. Також слід зазначити, що значний вплив на якість рішення має початкове розподілення феромону та вибір умовно оптимального рішення при ініціалізації.

Для реалізації мурашиного алгоритму потрібно:

1) сформулювати задачу в вигляді набору компонент та переходів як сукупності неорієнтованих зважених графів, на яких мурахи зможуть будувати рішення;

2) задати значення сліду феромону;

3) визначити евристику поведінки мурах під час побудови рішення;

4) реалізувати ефективний локальний пошук, якщо це можливо та необхідно;

5) вибрати специфічний мурашиний алгоритм та використати його для рішення задачі;

6) налаштувати параметри мурашиного алгоритму.

Метод мурашиних колоній виконується в наступній послідовності.

Крок 1. Задати параметри: 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 - ρ) 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 розподіляється рівномірно по вузлах мережі, такий рівномірний розподіл агентів між вузлами необхідний 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. Якщо агент ще не закінчив шлях, тобто не відвідав всі вузли мережі, для визначення наступної грані шляху використовується формула:

$$P = \frac{\tau_{ru}(t)^{\alpha} \cdot \eta_{ru}(t)^{\beta}}{\sum_{k \in J} \tau_{ru}(t)^{\alpha} \cdot \eta_{ru}(t)^{\beta}},$$

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

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

Крок 4. Розраховується довжина шляху пройденого агентом коли агент відвідає всі вузли мережі. Вона дорівнює сумі довжин всіх граней, по яких подорожував агент. А також визначається кількість феромону, що було залишено на кожній грані шляху i-м агентом, за формулою:

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

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

Отриманий результат є засобом оцінки шляху (короткий шлях характеризується високою концентрацією ферамону, а більш довгий шлях – більш низькою) та коригування (підсиленя) кількості феромону уздовж кожної грані пройденого *i*-им агентом шляху за формулою:

$$\tau_{ru}(t+1) = \tau_{ru}(t) + \left(\Delta\tau_{ru}^{i}(t) \cdot \rho\right),$$

де *r,u* – вузли, що утворюють грані які відвідав *i*-ий агент. Коєфіцієнт ρ приймає значення в межах 0…1.

Дана формула застосовується до всіх граней шляху, при цьому кожна грань отримує зміну значення кількості ферому пропорційно довжині шляху. Тому варто дочекатися, поки агент закінчить подорож і тільки потім оновити рівні феромону.

Далі кількість феромону на гранях корегується за рахунок випаровуваням відповідно до формули:

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

де $(1-\rho)$ – обернений коефіцієнт відновлення шляху.

Крок 5. Оцінка на досягнення оптимального результату. Оцінка може виконуватися для деякої постійної кількості запусків пошуку шляхів або до моменту, коли протягом декількох запусків не було отримано змін у значені довжини шляху. Якщо оцінка дала позитивний результат, то відбувається закінчення пошуку (перехід до кроку 7), у іншому випадку - перехід до кроку 6.

Крок 6. Повторний запуск. Після того як пошук шляху агентом завершений, грані оновлені відповідно до довжини шляху, відбулося випаровування феромону на всіх гранях, очищується список табу, довжина шляху прирівнюється нулю, а далі відбувається повторний запуск пошуку відповідно до кроку 3.

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

Завдання до роботи

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

2. Розробити за допомогою мов програмування програмне середовище, що реалізує метод мурашиних колоній. При програмній реалізації методу дотримуватися порядку методу мурашиних колоній.

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

- поточна позиція, в якій знаходиться агент;
- наступна позиція;
- список табу, 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 пошуку).

4. Передбачити можливість наглядного (графічного) відображення змін у поточних результатах роботи середовища у вигляді користувацького інтерфейсу.

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

-ініціалізація методу;
- моделювання переміщення агентів;
- вибір наступного пункту;
- оновлення шляхів;
- перезавантаження агентів.
- реалізувати окрему функцію розрахунку довжини шляху.

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

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

8. Оформити звіт з роботи.

9. Відповісти на контрольні питання.

Приклад виконання роботи

Розглянемо спочатку варіант, в якому кількість мурах рівна кількості міст. Нехай розташовується по одній мурашці в кожному місті, звідки вони і розпочнуть свій рух. Кількість міст та мурах відповідно повинна бути однаковою для такого варіанту мурашиної колонії.

Код програми:

```
ages = 50; % Кількість поколінь мурах
ants = 60; % Кількість мурах в одному поколінні
n = 60; % Кількість міст
a = 1; % Кількість феромону на шляху (коефіцієнт значимості шляху)
b = 3; % Пріоритет відстані над кількістю феромону
p = 0.5; % Кількість феромону на шляху, що лишає агент
Q = 1; % Кількість феромону, що лежить на шляху
pheromones_0 = 0.01; % Початкова кількість феромону
distance = zeros(n,n); % Матриця відстаней
reverse = zeros(n,n); % Матриця зворотніх відстаней
routes_1 = zeros(ants,n); % Матриця маршрутів в одному поколінні
distances_1 = zeros(ants,1); % Вектор відстаней одного покоління мурах
best_distance = inf; % Стартове значення найкращої дистанції
opt_routes = zeros(1,n); % Оптимальні шляхи
city_mix = randperm(n); % Перемішування списку міст
priority = zeros(1,n); % Матриця ймовірностей переходу між містами
cities = rand(n,2)*100; % Генеруємо координати міст
pher_on_routes = pheromones_0*(ones(n,n)); % Матриця початкової кількості феромонів
% Формуємо матриці відстаней в обидва боки
for i = 1:n
    for j = 1:n
        % Відстань вперед
        distance(i,j) = sqrt((cities(i,1)-cities(j,1))^2+(cities(i,2)-cities(j,2))^2);
            if i ~= j
            % відстань назад
            reverse(i,j) = 1/sqrt((cities(i,1)-cities(j,1))^2+(cities(i,2)-
cities(j,2))^2);
            end
    end
end
for ii = 1:ages % Кожне покоління
    for k = 1:ants % Кожна мураха з покоління
        routes_1(k,1) = city_mix(k); % Розташовуємо кожну мураху в окреме місто
        for s = 2:n % Шлях наступних мурах (після першої)
            city_id = routes_1(k,s-1); % Індекс міста
            % Обраховуємо ймовірність відвідування міст
```

```
            priority = pher_on_routes(city_id,:).^a .* reverse(city_id,:).^b;
            priority(routes_1(k,1:s-1)) = 0; % Враховуємо міста, в яких мурахи вже були
            priority = priority ./ sum(priority); % Обраховуємо кінцеву ймовірність
переходів
            next_city = find(cumsum(priority) >= rand, 1, 'first'); % Обрахунок
наступного переходу
            routes_1(k,s) = next_city; % Закріплюємо місто за мурахою
        end
        opt_routes = [routes_1(k,1:end),routes_1(k,1)]; % Обраховуэмо маршрут мурахи
        S = 0; % Довжина шляху
        for i = 1:n % Обраховуэмо довжину маршруту мурахи
            S = S + distance(opt_routes(i),opt_routes(i+1));
        end
        distances_1(k) = S; % Загальний шлях мурах в поколінні
        if distances_1(k) < best_distance % Перевіряємо результати нового покоління і
найкращого попереднього
            best_distance = distances_1(k); % Міняємо значення найкращого шляху
            best_route = routes_1(k,1:end); % Міняємо значення найкращого маршруту
            cities_final(:,[1,2]) = cities(best_route(:),[1,2]); % Формуємо маршрут
            % Будуємо графік маршрутів

plot([cities_final(:,1);cities_final(1,1)],[cities_final(:,2);cities_final(1,2)],'k-
o');
            % Заголовки для графіка
            title({['iteration ', num2str(ii)]; ['ant ', num2str(k)]; ['best route ',
num2str(distances_1(k))]});
            pause(0.1) % Час оновлення графіку
        end
    end
    pher_on_routes = (1-p)*pher_on_routes; % Оновлюємо кількість феромону
    for u = 1:ants % Кожна мураха
        opt_routes = [routes_1(u,1:end), routes_1(u,1)]; % Шлях кожної мурахи
        for t = 1:n % Кожне місто
            x = opt_routes(t); % Шлях до поточного міста
            y = opt_routes(t+1); % Шлях до наступного міста
            pher_on_routes(x,y) = pher_on_routes(x,y) + Q/distances_1(u); % Добавляємо
феромони на шляхи
        end
    end
end
disp(best_distance); % Виводимо на екран найкращу дистанцію
clearvars -except cities; % Очищуємо всі змінні окрім карти міст
```

Результати роботи програми для 10, 50, 100 та 200 міст та мурах:

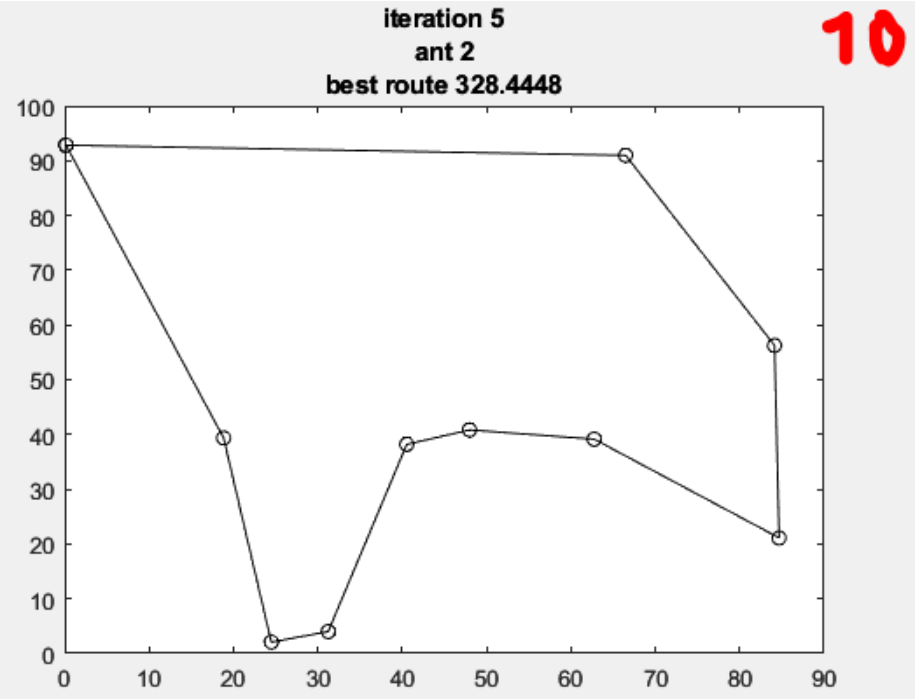


Рисунок 1 – Результати роботи програми для 10 міст та мурах

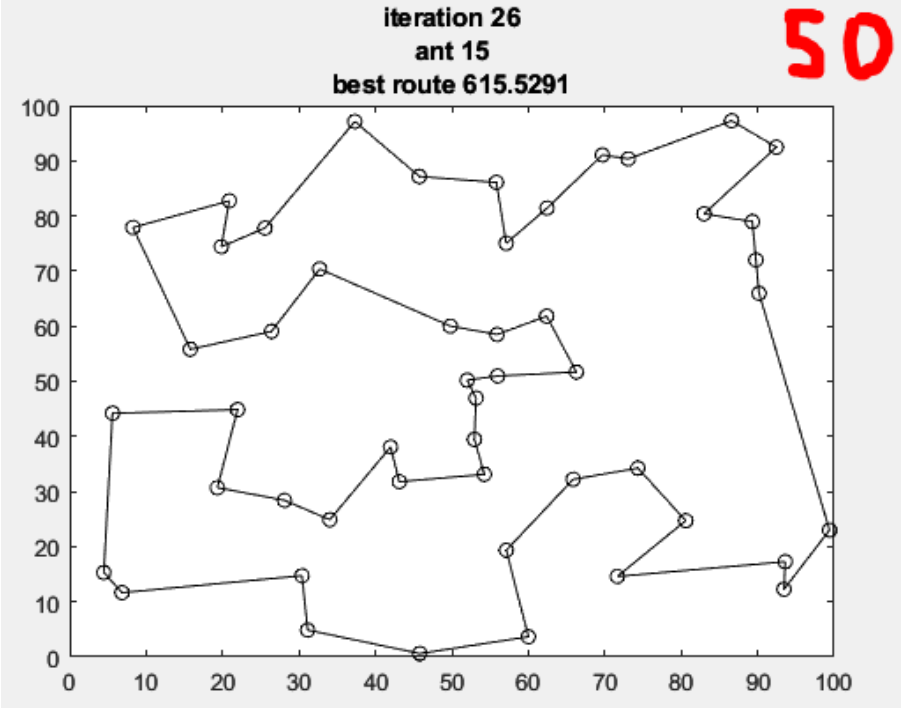


Рисунок 2 – Результати роботи програми для 50 міст та мурах

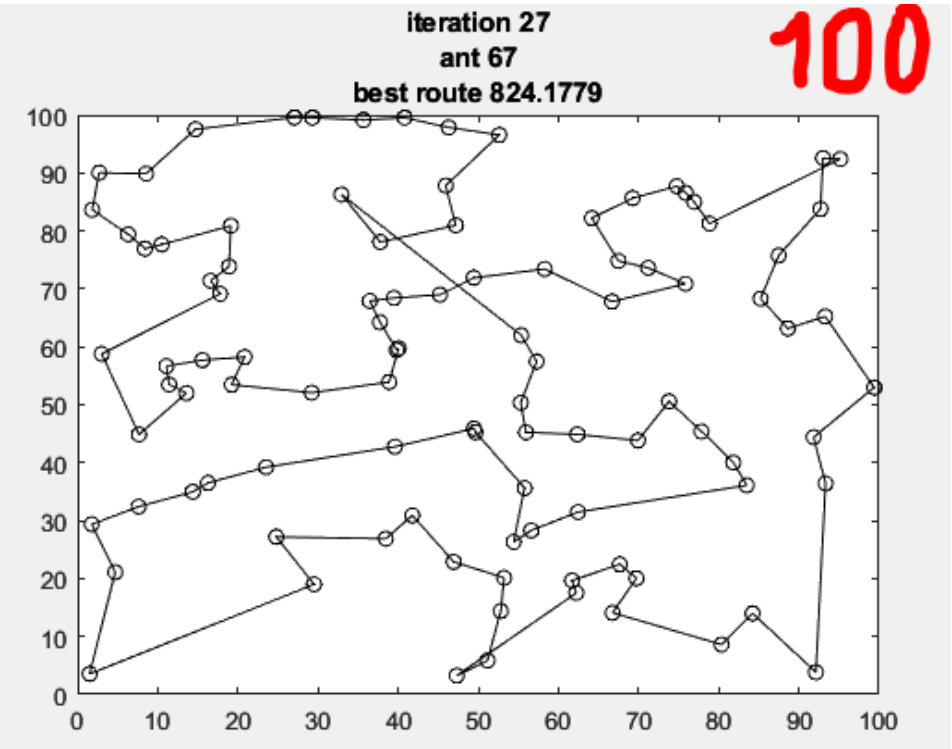


Рисунок 3 – Результати роботи програми для 100 міст та мурах

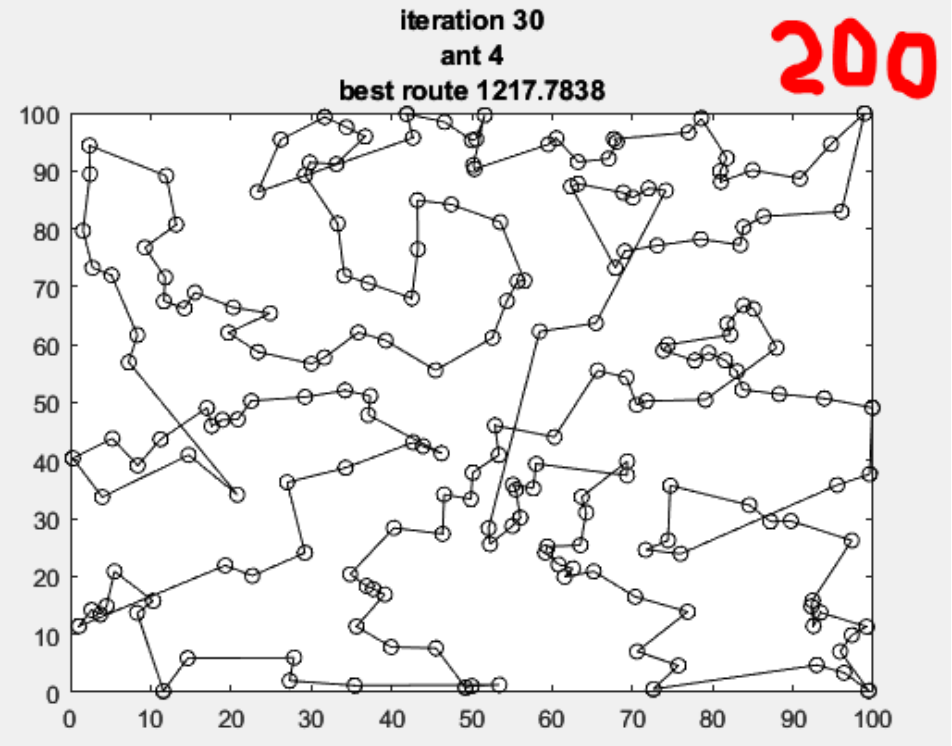


Рисунок 4 – Результати роботи програми для 200 міст та мурах

Можемо спостерігати, що чим більше міст, тим більше поколінь потрібно мурахам, щоб знайти найкращий шлях між всіма містами. Оскільки міст стає більше, то і зростає кількість шляхів, від чого росте і оптимальний шлях.

Розглянемо варіант, в якому мурахи розподіляються довільним шляхом між містами. Кількість міст та мурах може бути різною для такого варіанту мурашиної колонії, що дозволяє перевірити нам як вплине дефіцит чи надлишок мурах на оптимальний шлях. Для об'єктивної перевірки будемо використовувати одну і ту ж карту міст. Також будемо робити по 4 запуски, оскільки результати можуть відрізнятися.

Код програми:

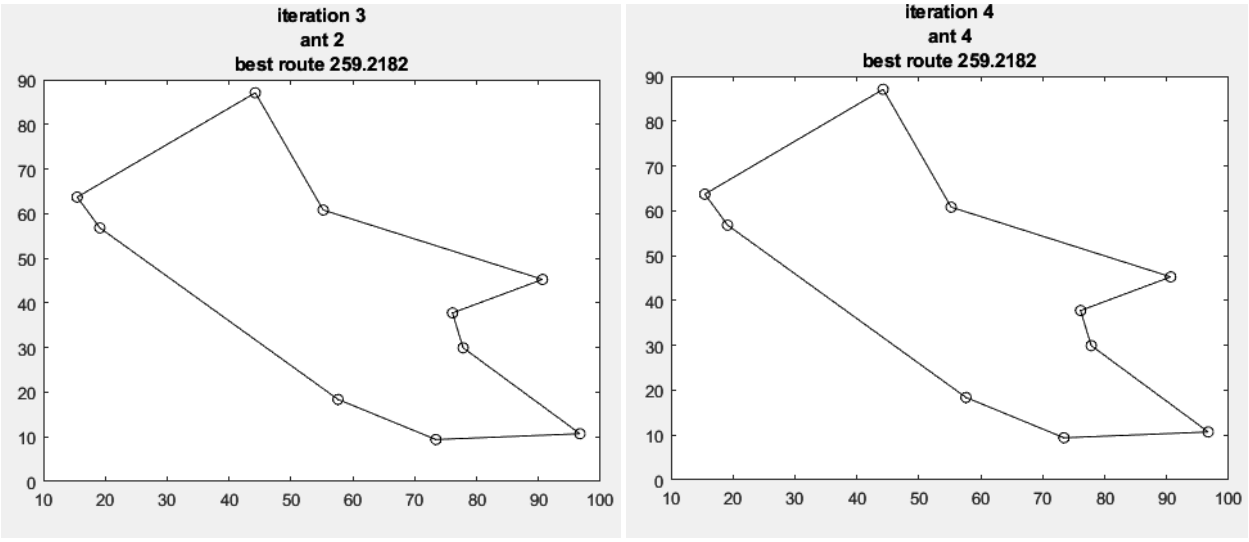
```
ages = 50; % Кількість поколінь мурах
ants = 1; % Кількість мурах в одному поколінні
n = 10; % Кількість міст
a = 1; % Кількість феромону на шляху (коефіцієнт значимості шляху)
b = 3; % Пріоритет відстані над кількістю феромону
p = 0.5; % Кількість феромону на шляху, що лишає агент
Q = 1; % Кількість феромону, що лежить на шляху
pheromones_0 = 0.01; % Початкова кількість феромону
distance = zeros(n,n); % Матриця відстаней
reverse = zeros(n,n); % Матриця зворотніх відстаней
routes_1 = zeros(ants,n); % Матриця маршрутів в одному поколінні
distances_1 = zeros(ants,1); % Вектор відстаней одного покоління мурах
best_distance = inf; % Стартове значення найкращої дистанції
opt_routes = zeros(1,n); % Оптимальні шляхи
city_mix = randperm(n); % Перемішування списку міст
priority = zeros(1,n); % Матриця ймовірностей переходу між містами
cities = rand(n,2)*100; % Генеруємо координати міст
pher_on_routes = pheromones_0*(ones(n,n)); % Матриця початкової кількості феромонів
% Формуємо матриці відстаней в обидва боки
for i = 1:n
    for j = 1:n
        % Відстань вперед
        distance(i,j) = sqrt((cities(i,1)-cities(j,1))^2+(cities(i,2)-cities(j,2))^2);
            if i ~= j
            % відстань назад
            reverse(i,j) = 1/sqrt((cities(i,1)-cities(j,1))^2+(cities(i,2)-
cities(j,2))^2);
            end
    end
end
for ii = 1:ages % Кожне покоління
    for k = 1:ants % Кожна мураха з покоління
        routes_1(k,1) = randi([1 n]); % Мурахи розподіляються довільно
        for s = 2:n % Шлях наступних мурах (після першої)
            city_id = routes_1(k,s-1); % Індекс міста
            % Обраховуємо ймовірність відвідування міст
            priority = pher_on_routes(city_id,:).^a .* reverse(city_id,:).^b;
            priority(routes_1(k,1:s-1)) = 0; % Враховуємо міста, в яких мурахи вже були
            priority = priority ./ sum(priority); % Обраховуємо кінцеву ймовірність
переходів
            next_city = find(cumsum(priority) >= rand, 1, 'first'); % Обрахунок
наступного переходу
            routes_1(k,s) = next_city; % Закріплюємо місто за мурахою
        end
        opt_routes = [routes_1(k,1:end),routes_1(k,1)]; % Обраховуэмо маршрут мурахи
        S = 0; % Довжина шляху
        for i = 1:n % Обраховуэмо довжину маршруту мурахи
```

```
            S = S + distance(opt_routes(i),opt_routes(i+1));
        end
        distances_1(k) = S; % Загальний шлях мурах в поколінні
        if distances_1(k) < best_distance % Перевіряємо результати нового покоління і
найкращого попереднього
            best_distance = distances_1(k); % Міняємо значення найкращого шляху
            best_route = routes_1(k,1:end); % Міняємо значення найкращого маршруту
            cities_final(:,[1,2]) = cities(best_route(:),[1,2]); % Формуємо маршрут
            % Будуємо графік маршрутів

plot([cities_final(:,1);cities_final(1,1)],[cities_final(:,2);cities_final(1,2)],'k-
o');
            % Заголовки для графіка
            title({['iteration ', num2str(ii)]; ['ant ', num2str(k)]; ['best route ',
num2str(distances_1(k))]});
            pause(0.1) % Час оновлення графіку
        end
    end
    pher_on_routes = (1-p)*pher_on_routes; % Оновлюємо кількість феромону
    for u = 1:ants % Кожна мураха
        opt_routes = [routes_1(u,1:end), routes_1(u,1)]; % Шлях кожної мурахи
        for t = 1:n % Кожне місто
            x = opt_routes(t); % Шлях до поточного міста
            y = opt_routes(t+1); % Шлях до наступного міста
            pher_on_routes(x,y) = pher_on_routes(x,y) + Q/distances_1(u); % Добавляємо
феромони на шляхи
        end
    end
end
disp(best_distance); % Виводимо на екран найкращу дистанцію
clearvars -except cities; % Очищуємо всі змінні окрім карти міст
```

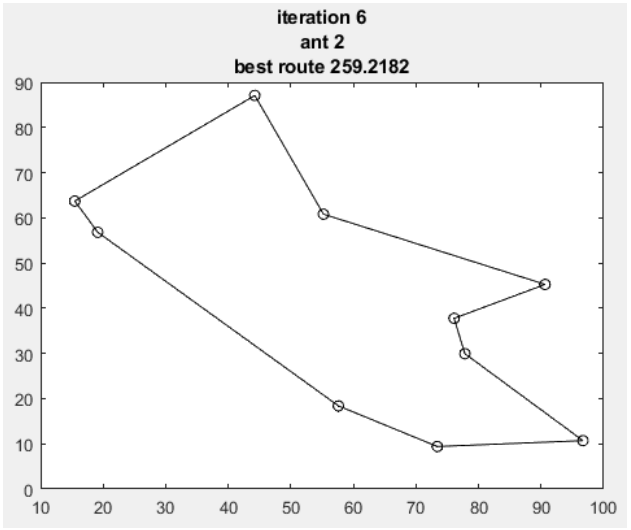
Спочатку зробимо дане дослідження з невеликою кількістю міст – 10.

Результати роботи для кількості мурах 10:

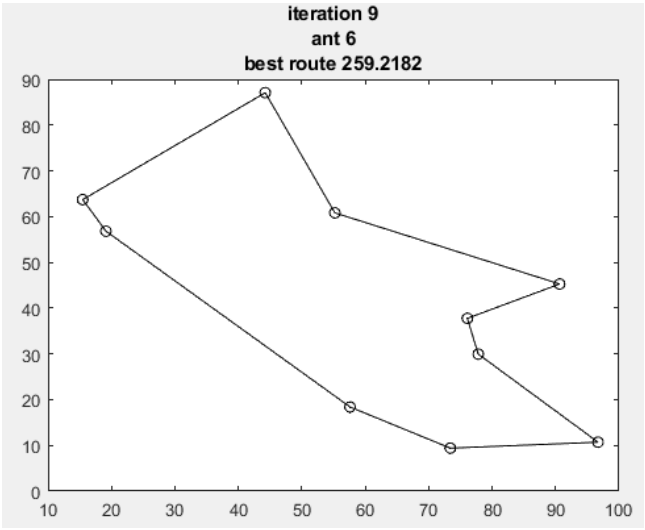


(*а*) (*б*)

Рисунок 5 –



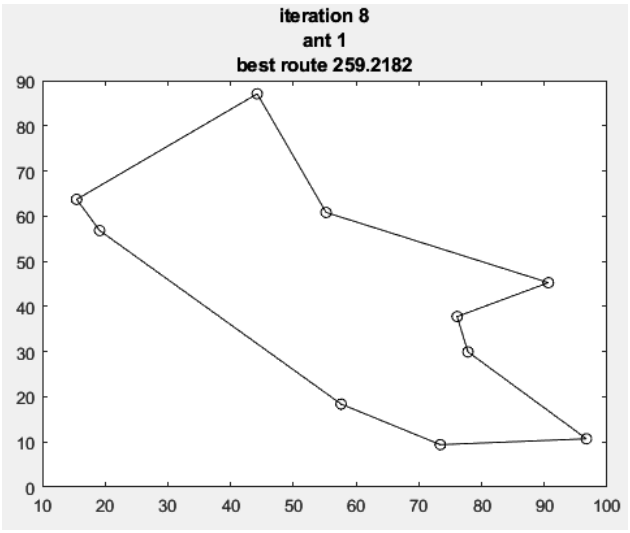
(*а*)



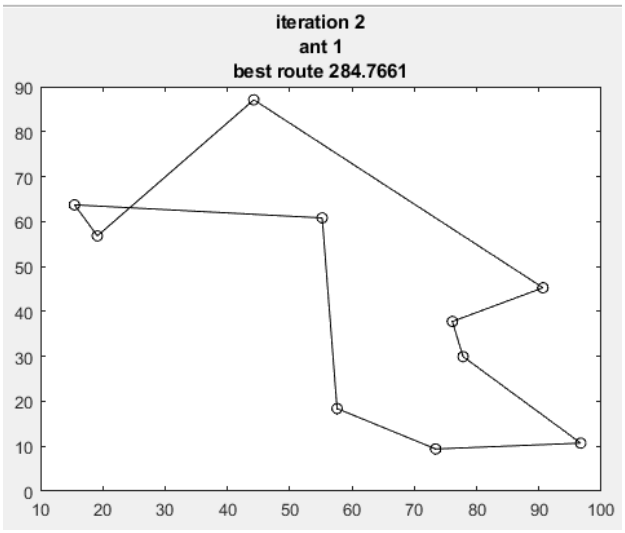
(*б*)

Рисунок 6 –

Бачимо, що шляхи побудовані однакові, відрізняється лише покоління, яке знайшло цей шлях. Тепер перевіримо цю ж карту з дифіцитом – 1 мураха на 10 міст:

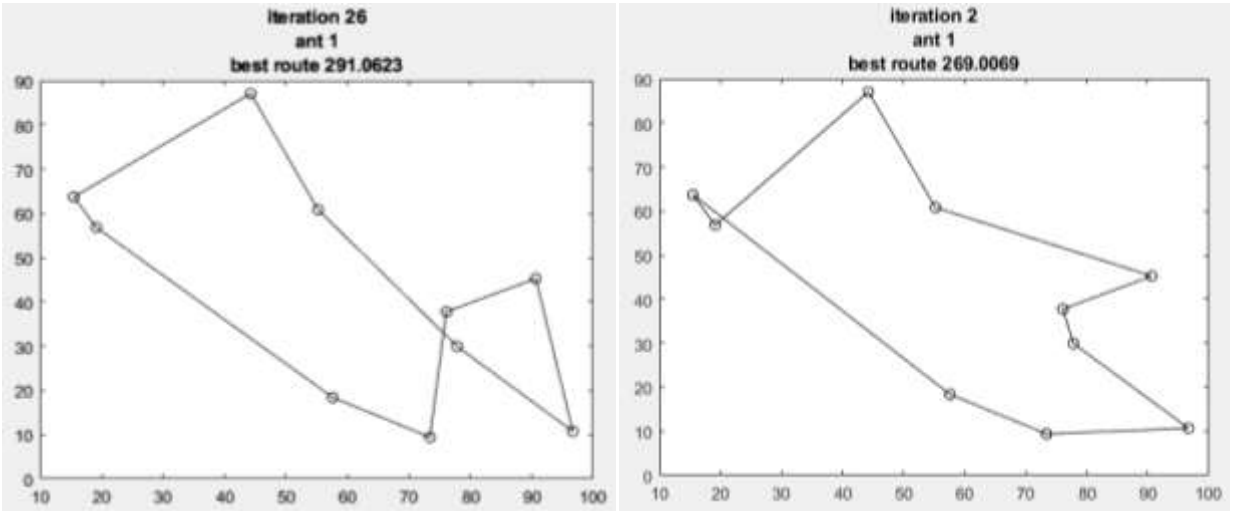


(*а*)



(*б*)

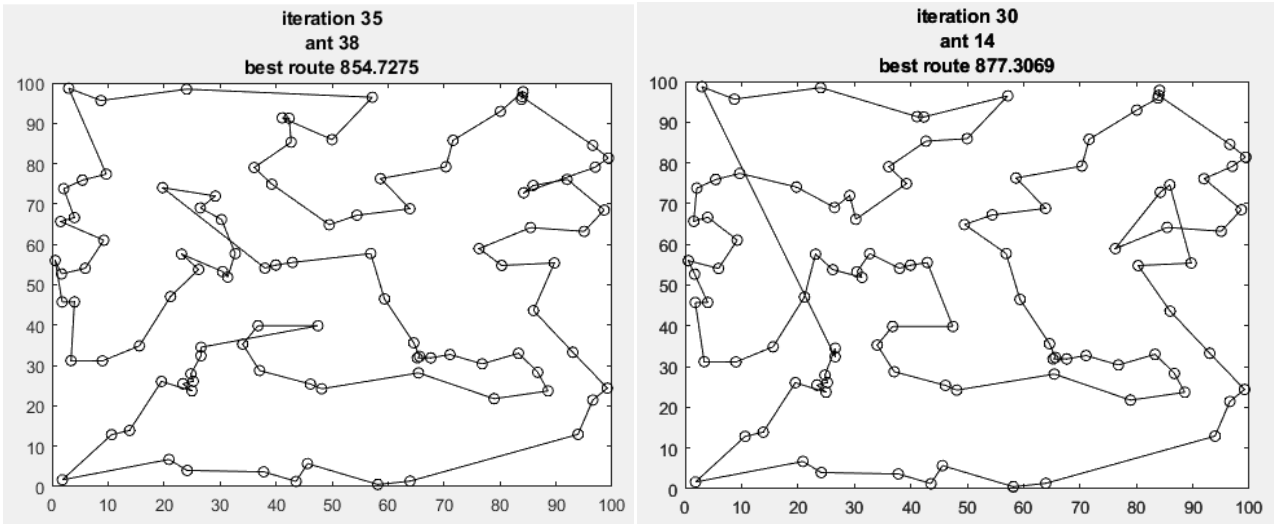
Рисунок 7 –



(*а*) (*б*)

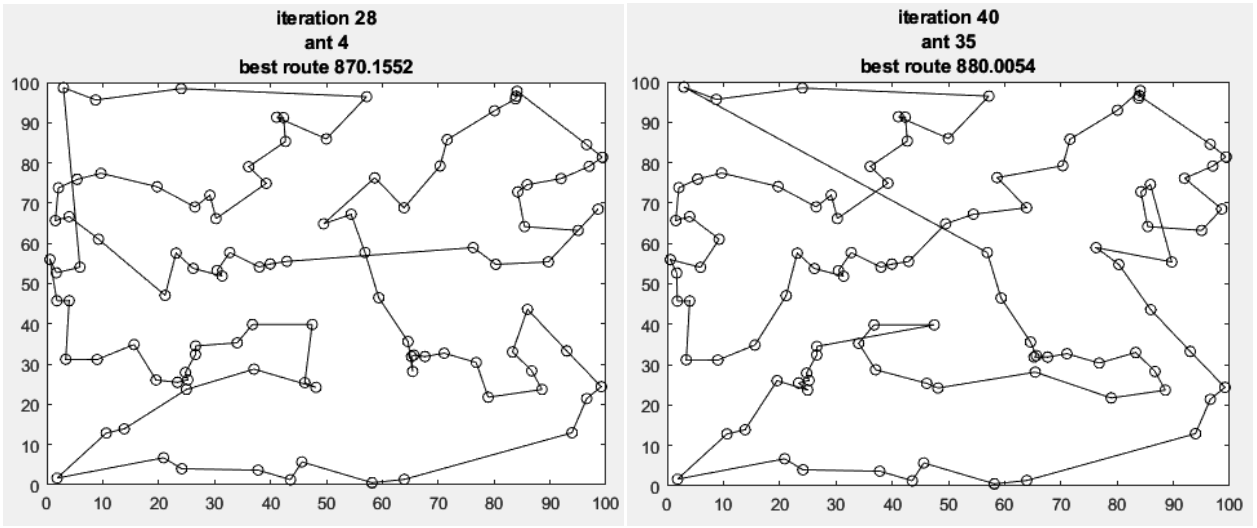
Рисунок 8 –

Бачимо, що одна мураха здатна побудувати такий же шлях з малою кількістю міст, але 3 з 4 запуски показали гірші результати. Тепер збільшимо кількість міст до 100 та кількість мурах до 100, щоб порівняння із більшою кількістю мурах було об’єктивнішим:



(*а*) (*б*)

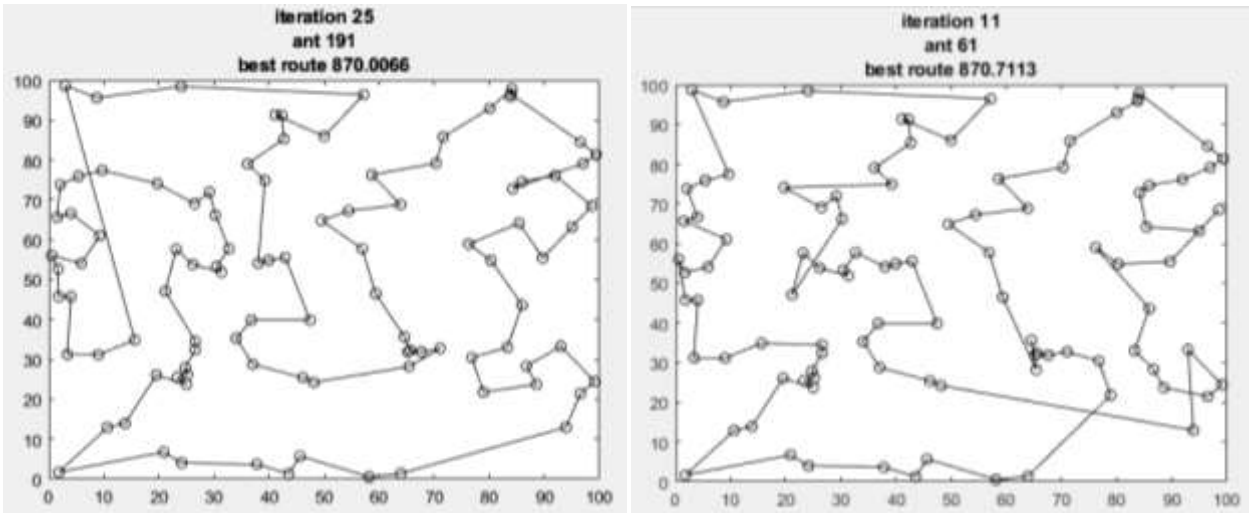
Рисунок 9 –



(*а*) (*б*)

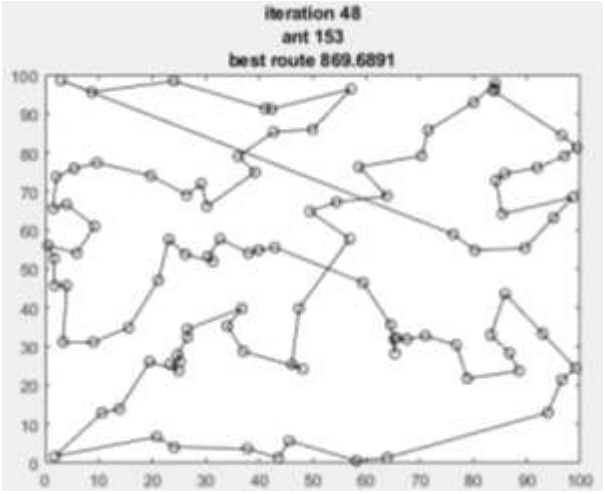
Рисунок 10 –

Бачимо, що найкращий шлях на цей раз відрізняється з кожним запуском. Тепер перевіримо чи стане генерація кращого маршруту кращою, якщо ми збільшимо кількість мурах до 200:

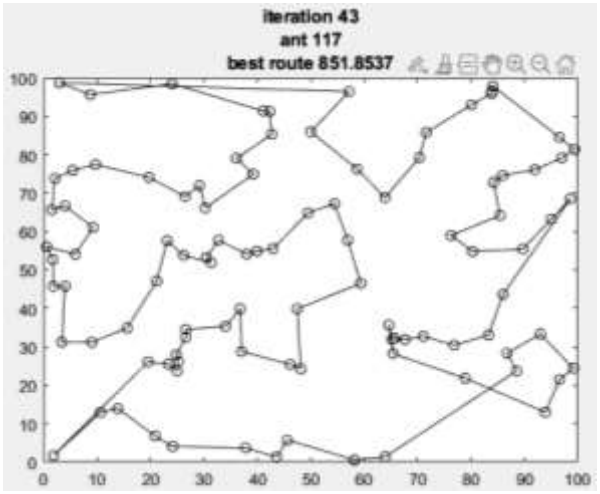


(*а*) (*б*)

Рисунок 11 –



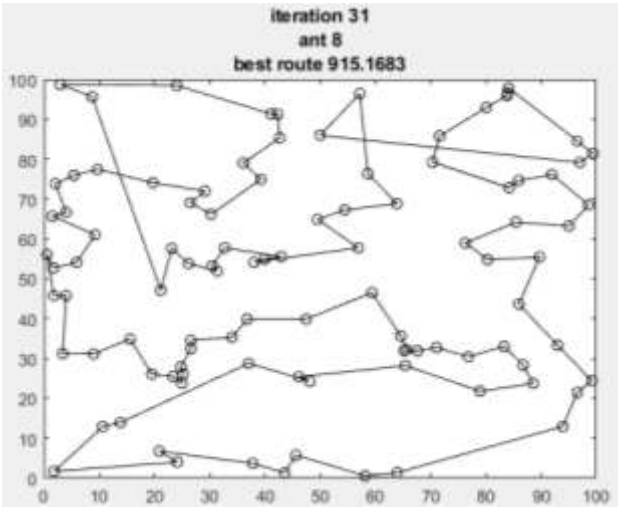
(*а*)



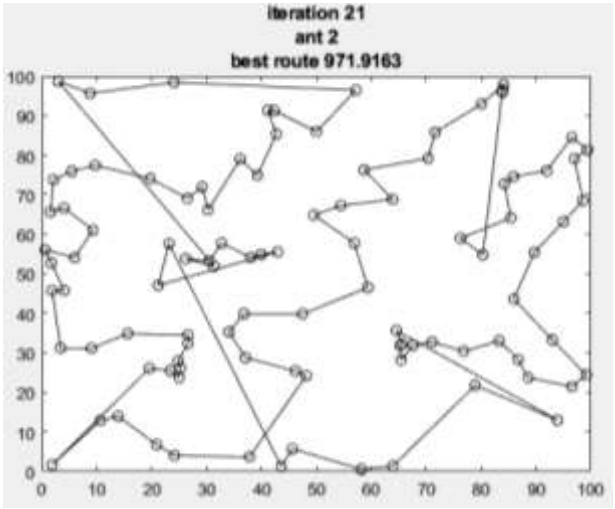
(*б*)

Рисунок 12 –

Бачимо, що в середньому найкращий шлях став коротший і навіть найкращий шлях був отриманий кращий ніж у випадку з однаковою кількістю мурах на однакову кількість міст. Проведемо тестування цієї ж карти міст із дифіцитом мурах – 10 мурах на 100 міст:

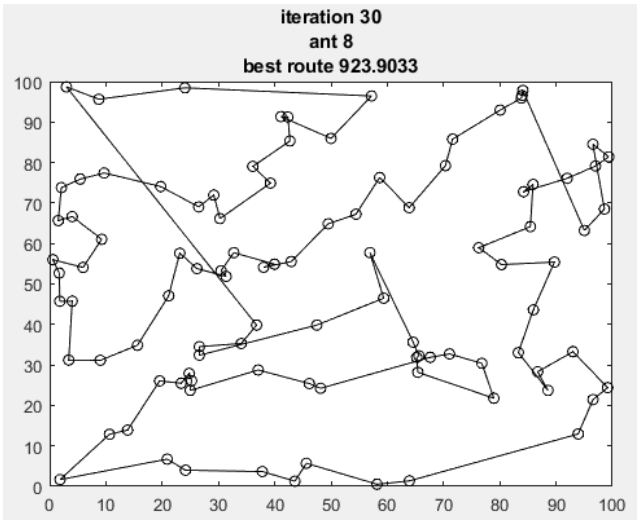


(*а*)

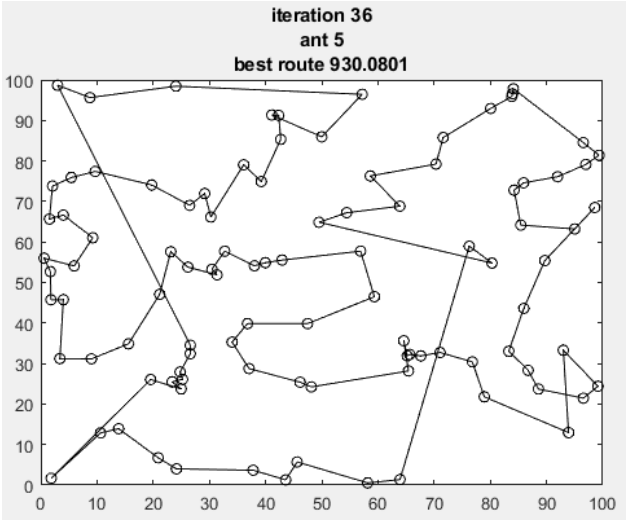


(*б*)

Рисунок 13 –



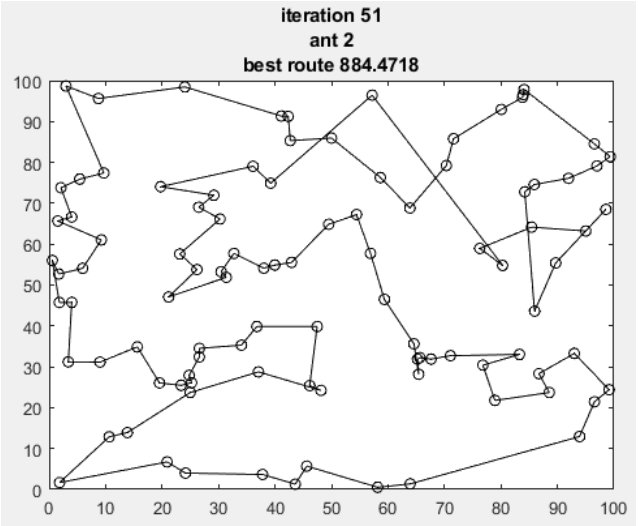
(*а*)



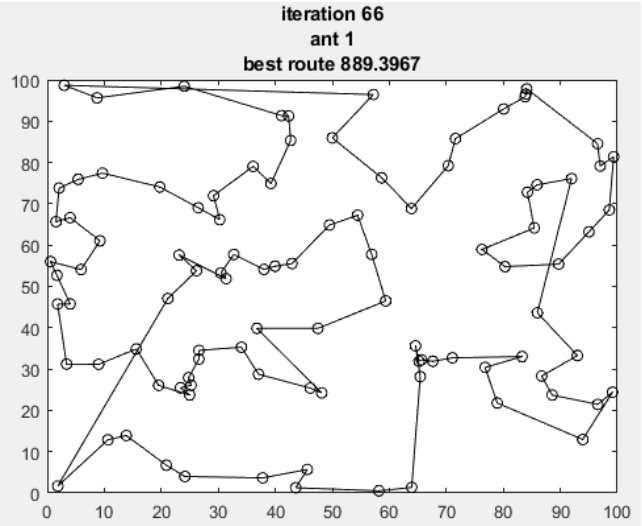
(*б*)

Рисунок 14 –

Бачим, що результати стали відрізнятися сильніше, а також середній результат значно погіршився. Відповідно можемо зробити висновок, що кількість мурах впливає на побудову оптимального шляху. Спробуємо надати 10-ти мурахам на даній карті міст більше поколінь - 500, щоб в них було більше часу збудувати оптимальний шлях:

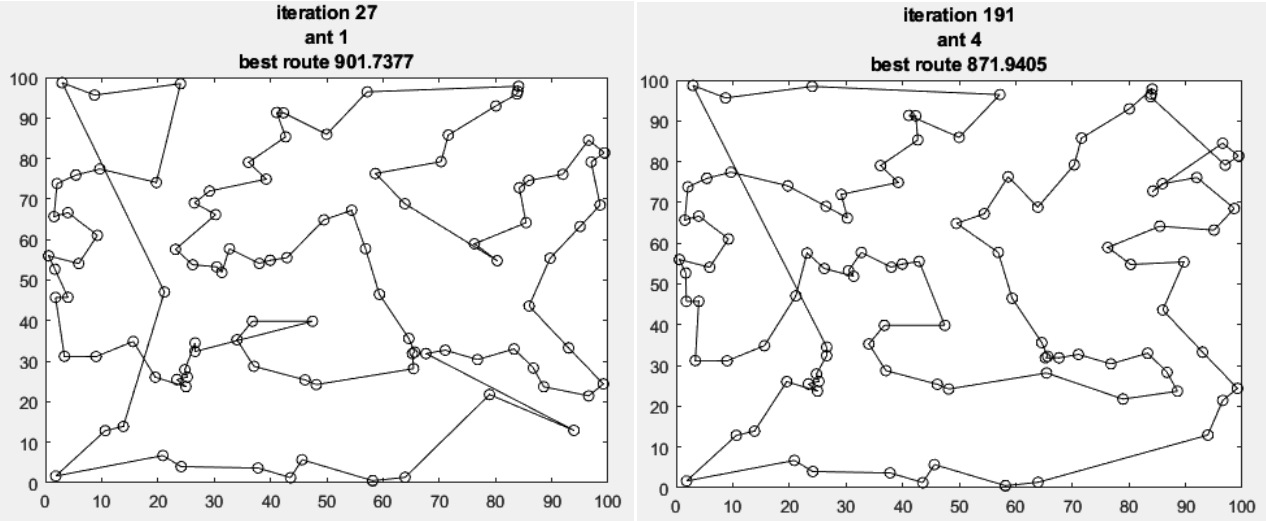


(*а*)



(*б*)

Рисунок 15 –



(*а*) (*б*)

Рисунок 16 –

Бачим, що середній результат став кращим і 3 запуски з 4 показали, що 50-ти поколінь для такої кількості мурах було замало. При одному з запусків 10 мурах змогли повторити найгірший результат 200 мурах, але їм для цього знадобилося майже в 4 рази більше поколінь.

Розглянемо варіант, в якому всі мурахи стартують з одного міста. Кількість міст та мурах може бути різною для такого варіанту мурашиної колонії, що знову дозволяє перевірити нам як вплине дефіцит чи надлишок мурах на оптимальний шлях. Для об’єктивної перевірки знову будемо використовувати одну і ту ж карту міст. Також будемо робити по 4 запуски, оскільки результати можуть відрізнятися.

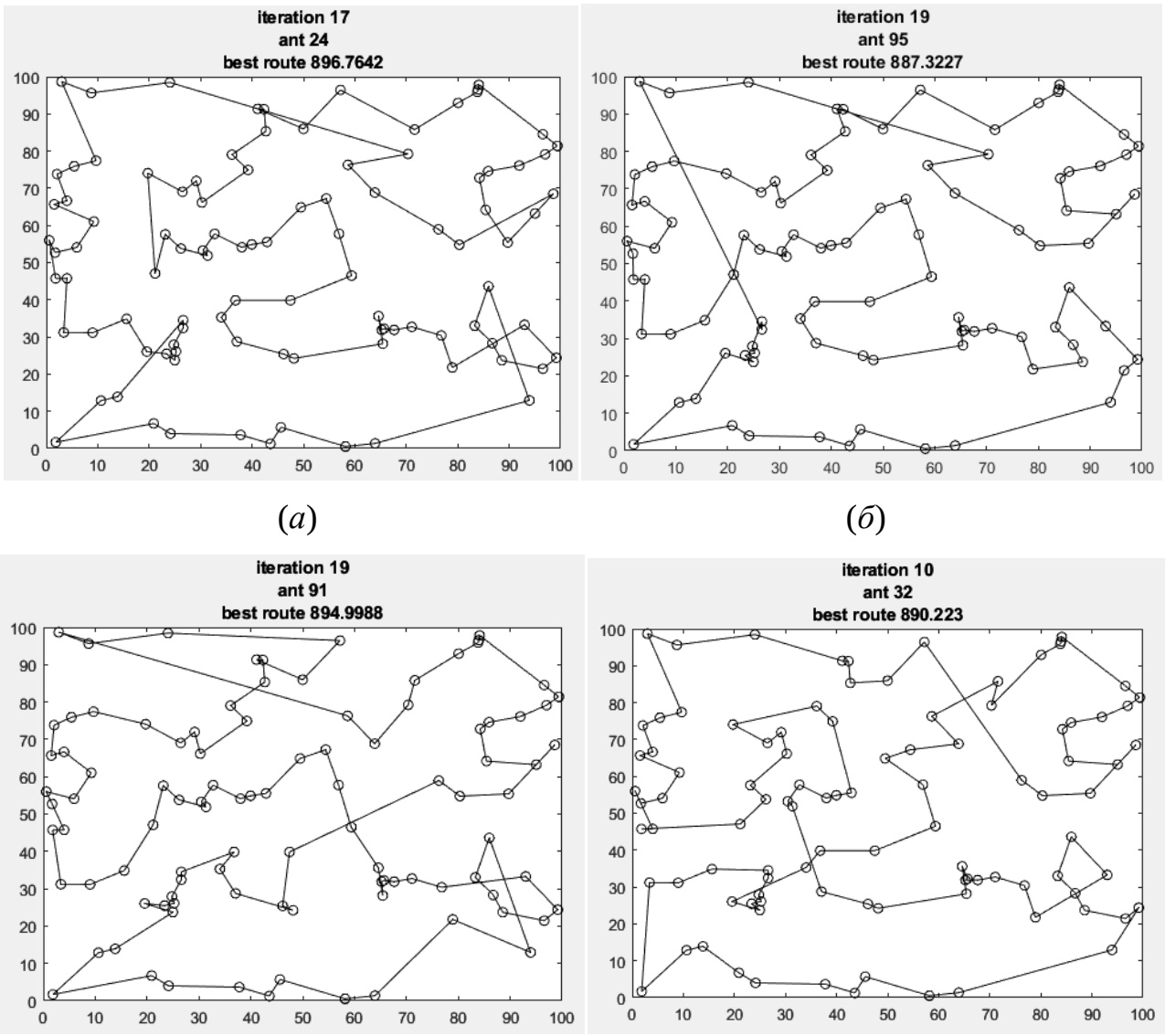
Код програми:

```
ages = 50; % Кількість поколінь мурах
ants = 50; % Кількість мурах в одному поколінні
n = 50; % Кількість міст
a = 1; % Кількість феромону на шляху (коефіцієнт значимості шляху)
b = 3; % Пріоритет відстані над кількістю феромону
p = 0.5; % Кількість феромону на шляху, що лишає агент
Q = 1; % Кількість феромону, що лежить на шляху
pheromones_0 = 0.01; % Початкова кількість феромону
distance = zeros(n,n); % Матриця відстаней
reverse = zeros(n,n); % Матриця зворотніх відстаней
routes_1 = zeros(ants,n); % Матриця маршрутів в одному поколінні
distances_1 = zeros(ants,1); % Вектор відстаней одного покоління мурах
best_distance = inf; % Стартове значення найкращої дистанції
opt_routes = zeros(1,n); % Оптимальні шляхи
city_mix = randperm(n); % Перемішування списку міст
priority = zeros(1,n); % Матриця ймовірностей переходу між містами
```

```
cities = rand(n,2)*100; % Генеруємо координати міст
pher_on_routes = pheromones_0*(ones(n,n)); % Матриця початкової кількості феромонів
% Формуємо матриці відстаней в обидва боки
for i = 1:n
    for j = 1:n
        % Відстань вперед
        distance(i,j) = sqrt((cities(i,1)-cities(j,1))^2+(cities(i,2)-cities(j,2))^2);
            if i ~= j
            % відстань назад
            reverse(i,j) = 1/sqrt((cities(i,1)-cities(j,1))^2+(cities(i,2)-
cities(j,2))^2);
            end
    end
end
for ii = 1:ages % Кожне покоління
    for k = 1:ants % Кожна мураха з покоління
        routes_1(k,1) = 7; % Мурахи виходять з конкретного міста
        for s = 2:n % Шлях наступних мурах (після першої)
            city_id = routes_1(k,s-1); % Індекс міста
            % Обраховуємо ймовірність відвідування міст
            priority = pher_on_routes(city_id,:).^a .* reverse(city_id,:).^b;
            priority(routes_1(k,1:s-1)) = 0; % Враховуємо міста, в яких мурахи вже були
            priority = priority ./ sum(priority); % Обраховуємо кінцеву ймовірність
переходів
            next_city = find(cumsum(priority) >= rand, 1, 'first'); % Обрахунок
наступного переходу
            routes_1(k,s) = next_city; % Закріплюємо місто за мурахою
        end
        opt_routes = [routes_1(k,1:end),routes_1(k,1)]; % Обраховуэмо маршрут мурахи
        S = 0; % Довжина шляху
        for i = 1:n % Обраховуэмо довжину маршруту мурахи
            S = S + distance(opt_routes(i),opt_routes(i+1));
        end
        distances_1(k) = S; % Загальний шлях мурах в поколінні
        if distances_1(k) < best_distance % Перевіряємо результати нового покоління і
найкращого попереднього
            best_distance = distances_1(k); % Міняємо значення найкращого шляху
            best_route = routes_1(k,1:end); % Міняємо значення найкращого маршруту
            cities_final(:,[1,2]) = cities(best_route(:),[1,2]); % Формуємо маршрут
            % Будуємо графік маршрутів

plot([cities_final(:,1);cities_final(1,1)],[cities_final(:,2);cities_final(1,2)],'k-
o');
            % Заголовки для графіка
            title({['iteration ', num2str(ii)]; ['ant ', num2str(k)]; ['best route ',
num2str(distances_1(k))]});
            pause(0.1) % Час оновлення графіку
        end
    end
    pher_on_routes = (1-p)*pher_on_routes; % Оновлюємо кількість феромону
    for u = 1:ants % Кожна мураха
        opt_routes = [routes_1(u,1:end), routes_1(u,1)]; % Шлях кожної мурахи
        for t = 1:n % Кожне місто
            x = opt_routes(t); % Шлях до поточного міста
            y = opt_routes(t+1); % Шлях до наступного міста
            pher_on_routes(x,y) = pher_on_routes(x,y) + Q/distances_1(u); % Добавляємо
феромони на шляхи
        end
    end
end
disp(best_distance); % Виводимо на екран найкращу дистанцію
clearvars -except cities; % Очищуємо всі змінні окрім карти міст
```

Використаємо ту ж карту на 100 міст, що використовувалася для тестування довільного розподілення мурах. Нехай в нас буде 100 мурах, в яких буде 50 поколінь для знаходження оптимального шляху. Стартова точка буде місто номер 1:

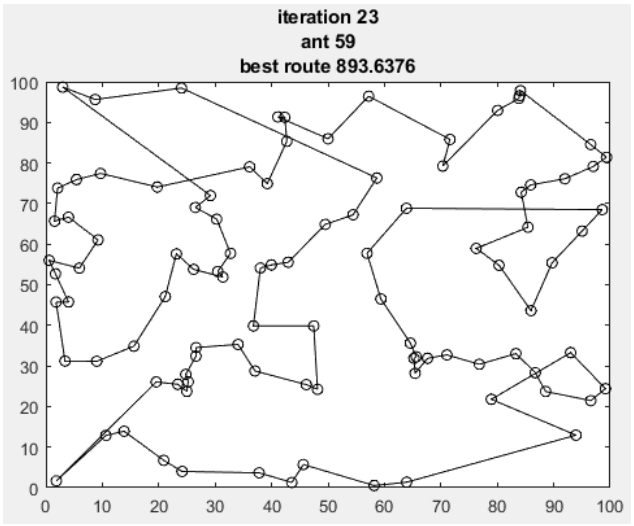


(*а*) (*б*)

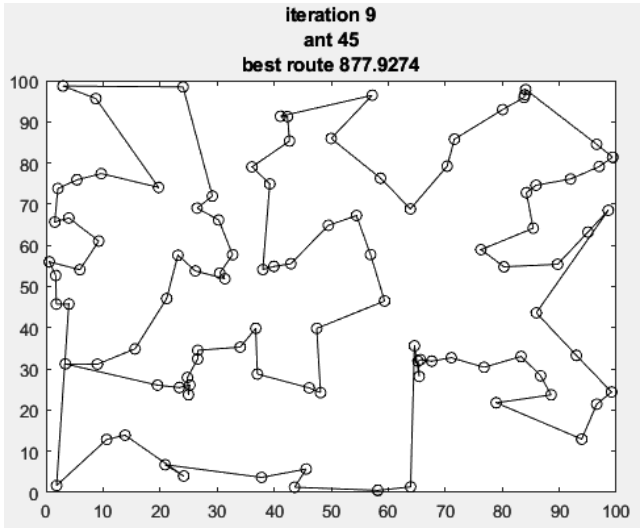
(*в*) (*г*)

Рисунок 17 –

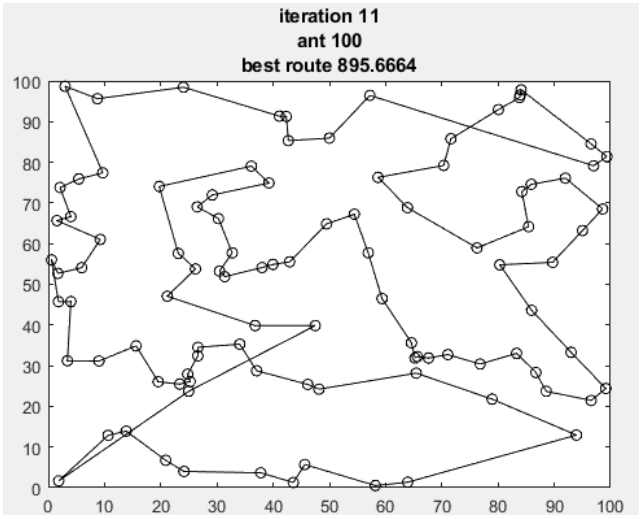
Бачим, що результати досить схожі, але гірші ніж були при довільному розподіленні мурах. Спробуємо змінити місто їх розташування:



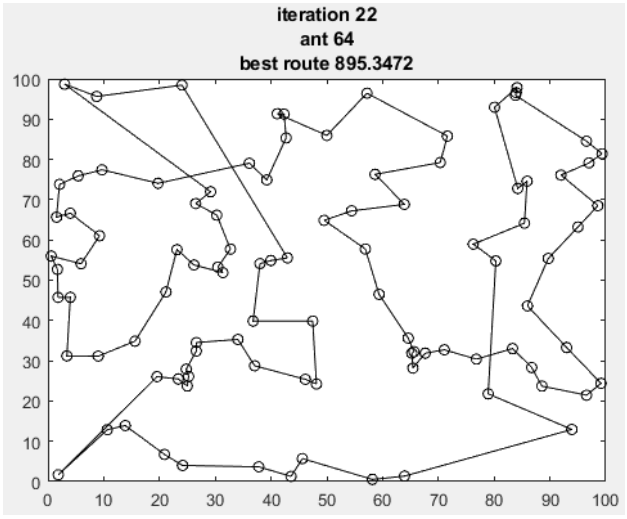
(*а*)



(*б*)



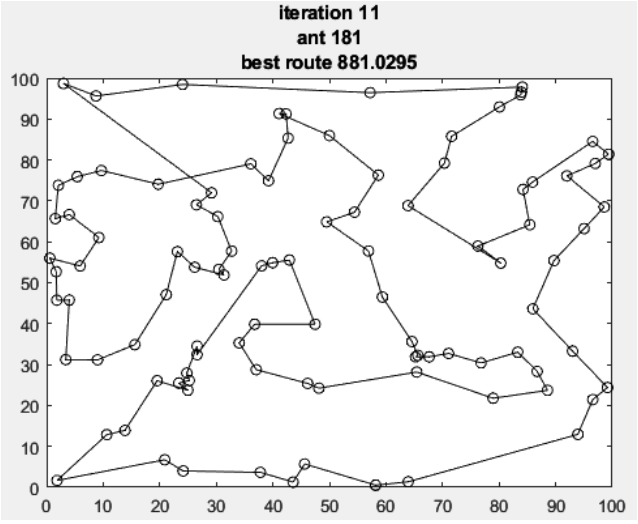
(*в*)



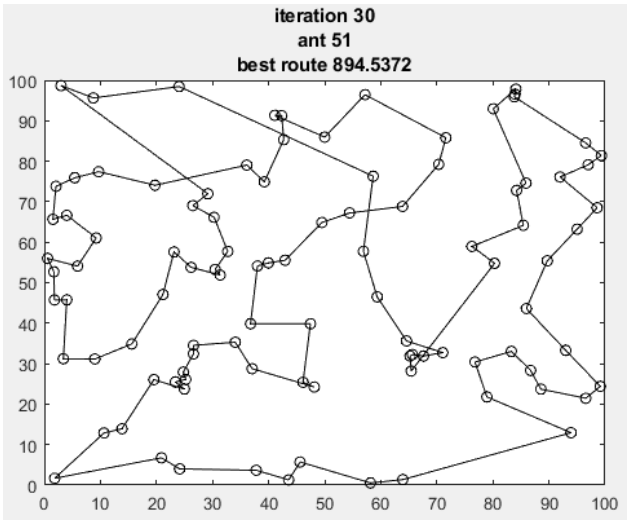
(*г*)

Рисунок 18 –

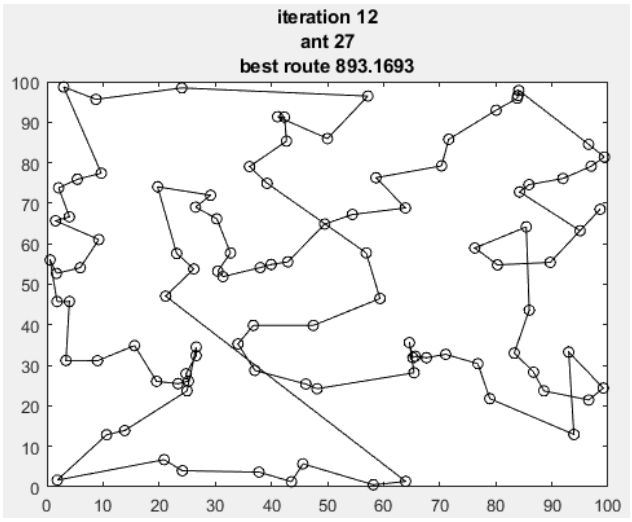
Бачим, що результати майже не змінилися. Спробуємо збільшити кількість мурах до 200:



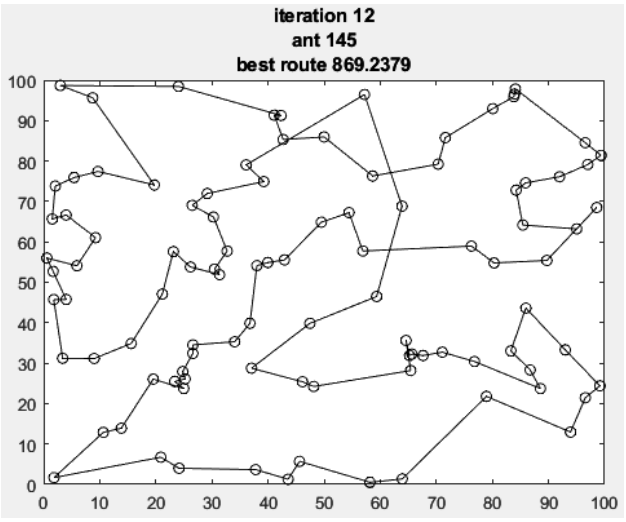
(*а*)



(*б*)



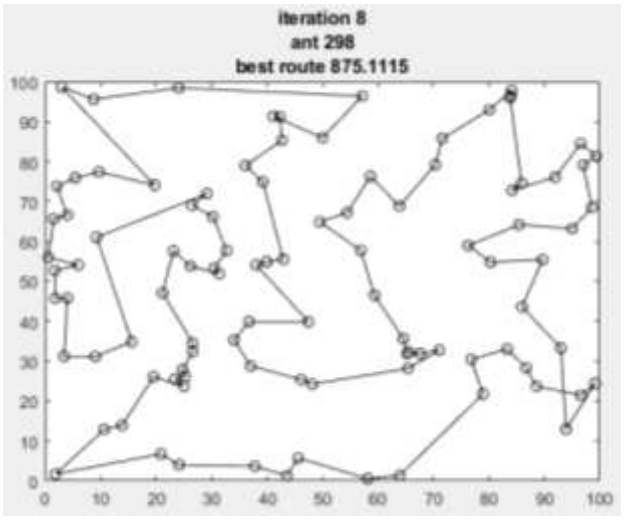
(*в*)



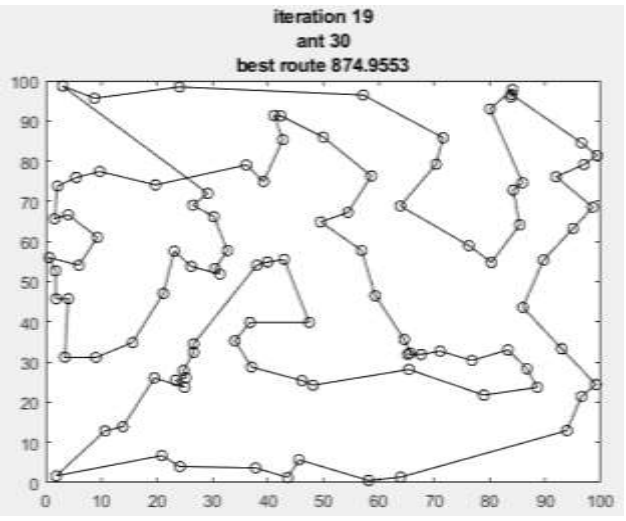
(*г*)

Рисунок 19 –

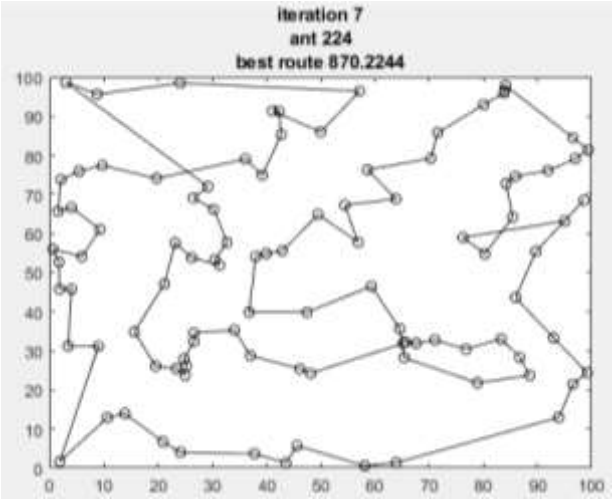
Бачим, що результати стали трішки кращими порівняно із кількістю мурах в 100 одиниць. Спробуємо збільшити кількість мурах до 500:



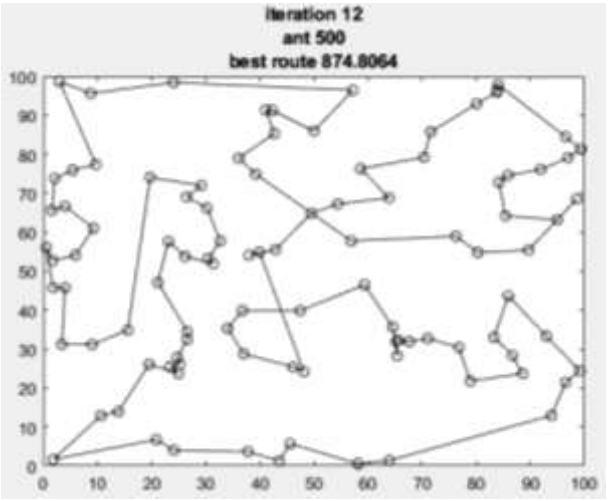
(*а*)



(*б*)



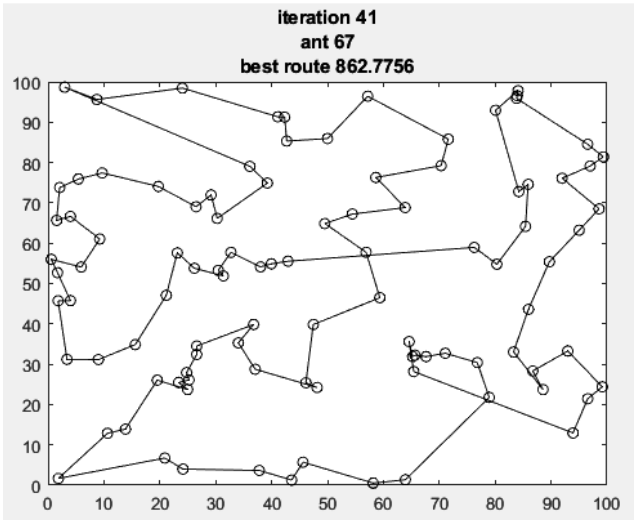
(*в*)



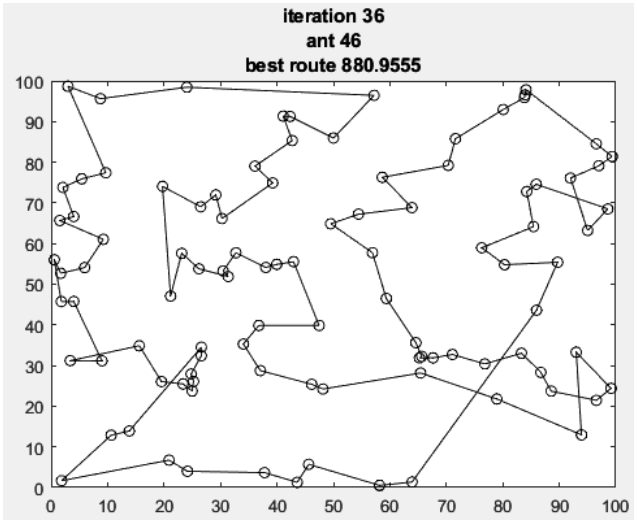
(*г*)

Рисунок 20 –

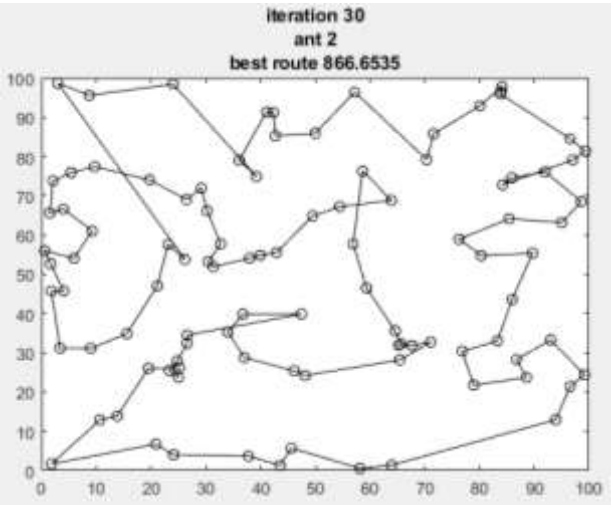
Бачим, що в середньому результати стали трішки кращі. Отже велика кількість мурах позитивно впливає на пошук найкращого шляху даним методом. Перевіримо як буде працювати з даною картою міст перша програма (з рівномірним розподіленням мурах по містах):

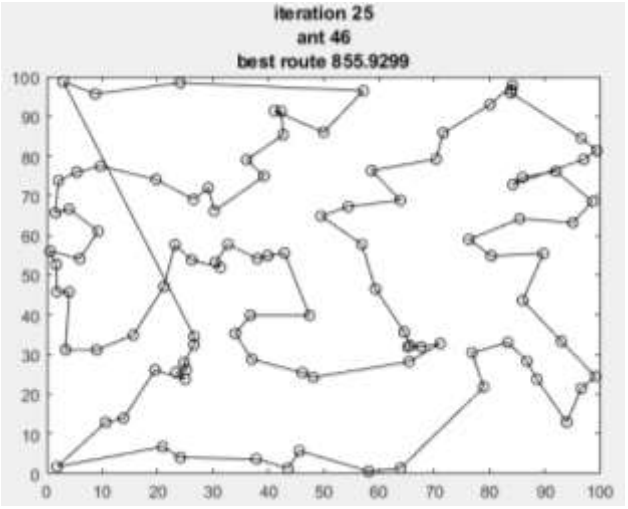


(*а*)



(*б*)





(*в*) (*г*)

Рисунок 22 –

Бачим, що результати дуже хороші, але не найкращі.

Зведемо всі отримані дані в таблицю (кількість мурах, міст та поколінь записано у вигляді a/b/c, де a – кількість мурах, b – кількість міст, c – кількісьб поколінь):

Тип	Довільне розподілення 100/100/50	Довільне розподілення 200/100/50	Довільне розподілення 10/100/50	Довільне розподілення 10/100/500	Старт з однієї точки 100/100/50	Старт з однієї точки 200/100/50	Старт з однієї точки 500/100/50	Рівномірний розподіл 100/100/50
1 запуск	855	870	915	884	897	881	875	864
2 запуск	877	871	972	890	887	895	875	881
3 запуск	870	870	924	902	895	893	870	867
4 запуск	880	852	930	872	890	869	875	856
Середнє значення	870,5	865,75	935,25	889	892,25	884,5	873,75	867

Контрольні питання

1. В чому полягає основна ідея методу мурашиних колоній?

2. Для розв’язання яких задач використовується метод мурашиних колоній?

3. В який умовах особливо ефективне використання мурашиних алгоритмів?

4. Порівняйте метод мурашиних колоній з іншими оптимізаційними методами.

5. Що являє собою навколишнє середовище в методі мурашиних колоній?

6. З якою метою використовується список табу?

7. Яким чином розташовуються вузли в списку поточної подорожі ?

8. Що таке феромон? Яке його призначення в методі мурашиних колоній?

9. Проаналізуйте послідовність виконання методу мурашиних колоній.

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

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.

РОБОТА № 2

Розробка інтелектуальної інформаційної систем на основі технології моделювання розвитку природних систем

Мета роботи: навчитися створювати інтелектуальні інформаційні системи на основі технології розвитку природних систем для моделювання процесів в економіці та живій природі, а також для моделювання взаємодії різноманітних об’єктів в конфліктних ситуаціях.

Основні теоретичні відомості

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 поводження 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 керувати створеним середовищем, перевіряти різні гіпотези й спостерігати, як системи й середовище реагують на зміни.

Моделювання харчових ланцюгів. Харчовий ланцюг описує ієрархію живих організмів в екосистемі. Найпростішим прикладом харчового ланцюгу може бути абстрактний ланцюг, що складається із трьох особин (Рисунок 23).

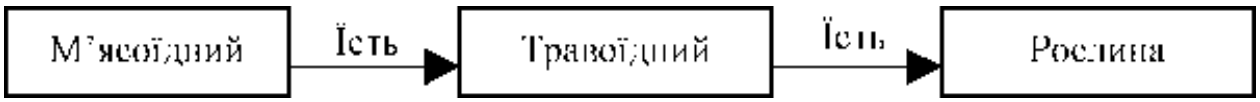


Рисунок 23 – Простий харчовий ланцюг

В нижній частині харчового ланцюгу перебувають рослини. Вони одержують енергію з навколишнього середовища (дощу, ґрунту й сонця). Наступний рівень займають травоїдні тварини - для виживання вони поїдають рослини. На верхньому щаблі перебувають хижаки. У цій моделі хижаки поїдають травоїдних 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 поводження в середовищі) і групу правил, які визначають, як і коли відбувається взаємодія.

Опис моделі. Модель простого харчового ланцюгу складається із середовища й трьох типів особин. Рослини являють собою нерухоме джерело їжі для травоїдних тварин. Травоїдні тварини є мігруючими агентами, які певним чином сприймають навколишнє середовище і їдять рослини. Іншими мігруючими агентами в середовищі є хижаки, що поїдають травоїдних тварин. Хижаки можуть їсти тільки травоїдних, а травоїдні 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.

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 (Рисунок 24).

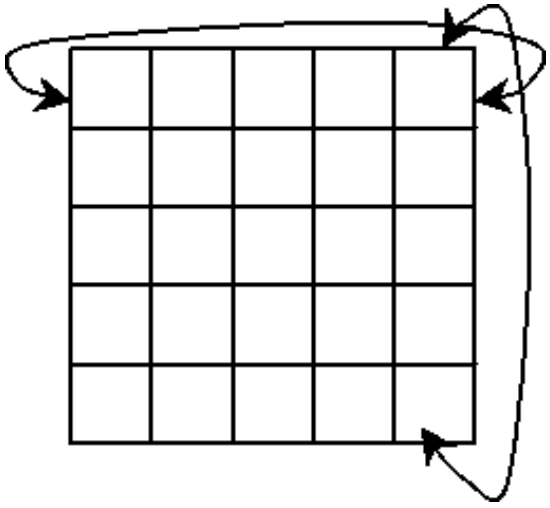
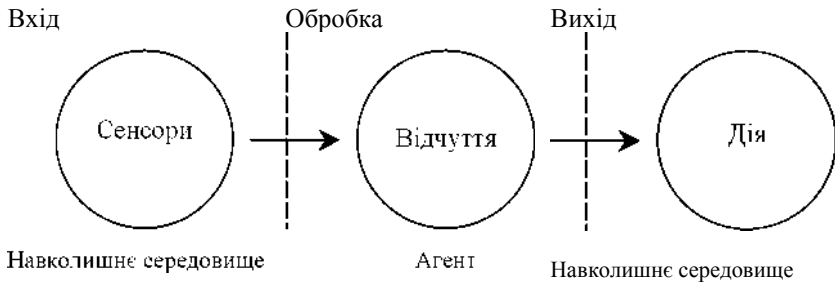


Рисунок 24 – Світ у вигляді сітки, побудованої за принципом тороїду

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

Анатомія агенту. Агент є генетичною особиною. Він може бути тільки певного типу (травоїдним або хижаком), але метод вивчення навколишнього середовища та порядок дій для всіх агентів однакові (Рисунок 25). Агента



можна розглядати як просту систему

Рисунок 25 – Модель систем з агентами

Як відображено на рисунку 25, 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 (рисунок 26).

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.

Агент може визначати вид об'єктів у полі зору. Тому для чотирьох областей пропонуються три числа, які дозволяють ідентифікувати типи наявних об'єктів (рослини, травоїдні й хижаки), тобто всього дванадцять входів.

Рисунок 26 – Область відчуття агенту. Агент «дивиться» на північ

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 – травоїдне на передньому7 плані;
- u_1 – хижак на передньому плані;
- u_2 – рослина на передньому плані;
- u_3 – травоїдне ліворуч;
- u_4 – хижак ліворуч;
- u_5 – рослина ліворуч;
- u_6 – травоїдне праворуч;
- u_7 – хижак праворуч;
- u_8 – рослина праворуч;
- u_9 – близькість травоїдного;
- u_{10} – близькість хижака;
- u_{11} – близькість рослини.

Така нейронна мережа має 4 виходи *oj*:

- 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:

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

Тобто, дія 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 (b_j). 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 налаштовані для виживання в середовищі.

Вибір дії агента. 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.

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 травоїдних, а травоїдні – 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 травоїдні – дві одиниці. Це означає, що для збереження життя травоїдним потрібно з'їдати у два рази більше їжі, ніж хижакам. Хоча хижакам не потрібно так багато їжі, їм ще потрібно її знайти. Травоїдні мають перевагу, що полягає в тому, що їхня їжа не переміщується по середовищу. Проте їм однаково потрібно відшукати свою їжу.

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.

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.

Завдання до роботи

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

2 Розробити програмне забезпечення, що моделює штучне життя (модель харчового ланцюгу). При розробці програмного забезпечення врахувати твердження, наведені в п.2.3.2.1-2.3.2.5.

3 Штучний світ моделюється за допомогою сітки розмірності NxN.

4 Існує всього 3 типи агентів:

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

2) травоїдні: вони мають становити максимально 50% від максимального можливого значення агентів і не можуть бути в кількості, меншій за 25% від максимально можливого значення агентів. При цьому травоїдні 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 травоїдних.

5 Агенти, що моделюють травоїдних 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 травоїдних 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 напряму, з’їсти іншого агенту. 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.

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

7 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.

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

9 Оформити звіт з роботи.

10 Відповісти на контрольні питання.

Приклад виконання роботи

Змоделюємо систему розміром 1000 на 1000 кілометрів, в якій будуть проживати 15000 рослин, 500 травоїдних тварин та 500 хижаків. Всі тварини мають свою особисту енергію та вік. Тварини будуть аналізувати все, що знаходиться на відстані до 2 кілометрів від них і на основі зібраних даних будуть визначати куди їм рухатися та що робити. За один день тварина може пройти до 2 кілометрів та втрачає енергію. Для того, щоб їй поповнити енергію їй потрібно шукати харчування. Травоїдні харчуються рослинами, хижаки харчуються травоїдними. Тварини можуть покинути ліс, якщо приймуть неправильне рішення, що призведе до голодної смерті в пустелі. В кінці життєдіяльності (протягом 50-ти днів) виводиться на екран поточна карта розміщення всіх жителів та кількість живих тварин.

Розглянемо код програми:

```
n = 1000; % Кількість клітинок
days = 50; % Кількість днів
gr_n = 15000; % Кількість рослин
gr_eat_n = 500; % Кількість травоїдних
wild_n = 500; % Кількість хижаків
age_gr_eat = zeros(gr_eat_n,1); % Покоління травоїдних
age_wild = zeros(wild_n,1); % Покоління хижаків
nrg_gr_eat = ones(gr_eat_n,1); % Енергія травоїдних
nrg_wild = ones(wild_n,1); % Енергія хижаків
gr_cords = rand(gr_n,2)*n; % Генеруємо координати рослин
gr_cords = round(gr_cords);
gr_eat_cords = rand(gr_eat_n,2)*n; % Генеруємо початкові координати травоїдних
gr_eat_cords = round(gr_eat_cords);
wild_cords = rand(wild_n,2)*n; % Генеруємо початкові координати хижаків
wild_cords = round(wild_cords);
% Перевіряємо що є в полі зору в кожного травоїдного
for d = 1:days
    for i = 1:gr_eat_n % Кількість травоїдних в системі
        w = 0;
        gr = 0; % Створюємо лічильники
        gr_sum = 0;
        w_sum = 0;
        long_gr_sum = 0;
```

```
        long_w_sum = 0;
        for j = 1:gr_n % Кількість рослин в системі
            % Перевіряємо присутність рослин під ногами
            if gr_eat_cords(i,1) == gr_cords(j,1) && gr_eat_cords(i,2) == gr_cords(j,2)
                gr = gr + 1;
            end
            % Перевіряємо ближню відстань на присутність рослин
            if gr_eat_cords(i,1) == gr_cords(j,1)+1 && gr_eat_cords(i,2) ==
gr_cords(j,2)+1
                gr_sum = gr_sum + 1;
            end
            if gr_eat_cords(i,1) == gr_cords(j,1) && gr_eat_cords(i,2) ==
gr_cords(j,2)+1
                gr_sum = gr_sum + 1;
            end
            if gr_eat_cords(i,1) == gr_cords(j,1)+1 && gr_eat_cords(i,2) ==
gr_cords(j,2)
                gr_sum = gr_sum + 1;
            end
            if gr_eat_cords(i,1) == gr_cords(j,1) && gr_eat_cords(i,2) ==
gr_cords(j,2)-1
                gr_sum = gr_sum + 1;
            end
            if gr_eat_cords(i,1) == gr_cords(j,1)+1 && gr_eat_cords(i,2) ==
gr_cords(j,2)-1
                gr_sum = gr_sum + 1;
            end
            if gr_eat_cords(i,1) == gr_cords(j,1)-1 && gr_eat_cords(i,2) ==
gr_cords(j,2)+1
                gr_sum = gr_sum + 1;
            end
            if gr_eat_cords(i,1) == gr_cords(j,1)-1 && gr_eat_cords(i,2) ==
gr_cords(j,2)
                gr_sum = gr_sum + 1;
            end
            if gr_eat_cords(i,1) == gr_cords(j,1)-1 && gr_eat_cords(i,2) ==
gr_cords(j,2)-1
                gr_sum = gr_sum + 1;
            end
            % Перевіряємо дальню відстань на присутність рослин
            if gr_eat_cords(i,1) == gr_cords(j,1)+2 && gr_eat_cords(i,2) ==
gr_cords(j,2)+2
                long_gr_sum = long_gr_sum + 1;
            end
            if gr_eat_cords(i,1) == gr_cords(j,1) && gr_eat_cords(i,2) ==
gr_cords(j,2)+2
                long_gr_sum = long_gr_sum + 1;
            end
            if gr_eat_cords(i,1) == gr_cords(j,1)+2 && gr_eat_cords(i,2) ==
gr_cords(j,2)
                long_gr_sum = long_gr_sum + 1;
            end
            if gr_eat_cords(i,1) == gr_cords(j,1) && gr_eat_cords(i,2) ==
gr_cords(j,2)-2
                long_gr_sum = long_gr_sum + 1;
            end
            if gr_eat_cords(i,1) == gr_cords(j,1)+2 && gr_eat_cords(i,2) ==
gr_cords(j,2)-2
                long_gr_sum = long_gr_sum + 1;
            end
            if gr_eat_cords(i,1) == gr_cords(j,1)-2 && gr_eat_cords(i,2) ==
gr_cords(j,2)+2
```

```
                long_gr_sum = long_gr_sum + 1;
            end
            if gr_eat_cords(i,1) == gr_cords(j,1)-2 && gr_eat_cords(i,2) ==
gr_cords(j,2)
                long_gr_sum = long_gr_sum + 1;
            end
            if gr_eat_cords(i,1) == gr_cords(j,1)-2 && gr_eat_cords(i,2) ==
gr_cords(j,2)-2
                long_gr_sum = long_gr_sum + 1;
            end
            if gr_eat_cords(i,1) == gr_cords(j,1)+2 && gr_eat_cords(i,2) ==
gr_cords(j,2)+1
                long_gr_sum = long_gr_sum + 1;
            end
            if gr_eat_cords(i,1) == gr_cords(j,1)+1 && gr_eat_cords(i,2) ==
gr_cords(j,2)+2
                long_gr_sum = long_gr_sum + 1;
            end
            if gr_eat_cords(i,1) == gr_cords(j,1)+2 && gr_eat_cords(i,2) ==
gr_cords(j,2)-1
                long_gr_sum = long_gr_sum + 1;
            end
            if gr_eat_cords(i,1) == gr_cords(j,1)+1 && gr_eat_cords(i,2) ==
gr_cords(j,2)-2
                long_gr_sum = long_gr_sum + 1;
            end
            if gr_eat_cords(i,1) == gr_cords(j,1)-1 && gr_eat_cords(i,2) ==
gr_cords(j,2)-2
                long_gr_sum = long_gr_sum + 1;
            end
            if gr_eat_cords(i,1) == gr_cords(j,1)-1 && gr_eat_cords(i,2) ==
gr_cords(j,2)+2
                long_gr_sum = long_gr_sum + 1;
            end
            if gr_eat_cords(i,1) == gr_cords(j,1)-2 && gr_eat_cords(i,2) ==
gr_cords(j,2)+1
                long_gr_sum = long_gr_sum + 1;
            end
            if gr_eat_cords(i,1) == gr_cords(j,1)-2 && gr_eat_cords(i,2) ==
gr_cords(j,2)-1
                long_gr_sum = long_gr_sum + 1;
            end
            if gr_sum ~=0 || long_gr_sum ~=0
                jj = j;
            end
        end
        for j = 1:wild_n
            % Перевіряємо присутність хижаків під ногами
            if gr_eat_cords(i,1) == wild_cords(j,1) && gr_eat_cords(i,2) ==
wild_cords(j,2)
                w = w + 1;
            end
            % Перевіряємо ближню відстань на присутність хижаків
            if gr_eat_cords(i,1) == wild_cords(j,1)+1 && gr_eat_cords(i,2) ==
wild_cords(j,2)+1
                w_sum = w_sum + 1;
            end
            if gr_eat_cords(i,1) == wild_cords(j,1) && gr_eat_cords(i,2) ==
wild_cords(j,2)+1
                w_sum = w_sum + 1;
            end
```

```
            if gr_eat_cords(i,1) == wild_cords(j,1)+1 && gr_eat_cords(i,2) ==
wild_cords(j,2)
                w_sum = w_sum + 1;
            end
            if gr_eat_cords(i,1) == wild_cords(j,1) && gr_eat_cords(i,2) ==
wild_cords(j,2)-1
                w_sum = w_sum + 1;
            end
            if gr_eat_cords(i,1) == wild_cords(j,1)+1 && gr_eat_cords(i,2) ==
wild_cords(j,2)-1
                w_sum = w_sum + 1;
            end
            if gr_eat_cords(i,1) == wild_cords(j,1)-1 && gr_eat_cords(i,2) ==
wild_cords(j,2)+1
                w_sum = w_sum + 1;
            end
            if gr_eat_cords(i,1) == wild_cords(j,1)-1 && gr_eat_cords(i,2) ==
wild_cords(j,2)
                w_sum = w_sum + 1;
            end
            if gr_eat_cords(i,1) == wild_cords(j,1)-1 && gr_eat_cords(i,2) ==
wild_cords(j,2)-1
                w_sum = w_sum + 1;
            end
            % Перевіряємо дальню відстань на присутність хижаків
            if gr_eat_cords(i,1) == wild_cords(j,1)+2 && gr_eat_cords(i,2) ==
wild_cords(j,2)+2
                long_w_sum = long_w_sum + 1;
            end
            if gr_eat_cords(i,1) == wild_cords(j,1) && gr_eat_cords(i,2) ==
wild_cords(j,2)+2
                long_w_sum = long_w_sum + 1;
            end
            if gr_eat_cords(i,1) == wild_cords(j,1)+2 && gr_eat_cords(i,2) ==
wild_cords(j,2)
                long_w_sum = long_w_sum + 1;
            end
            if gr_eat_cords(i,1) == wild_cords(j,1) && gr_eat_cords(i,2) ==
wild_cords(j,2)-2
                long_w_sum = long_w_sum + 1;
            end
            if gr_eat_cords(i,1) == wild_cords(j,1)+2 && gr_eat_cords(i,2) ==
wild_cords(j,2)-2
                long_w_sum = long_w_sum + 1;
            end
            if gr_eat_cords(i,1) == wild_cords(j,1)-2 && gr_eat_cords(i,2) ==
wild_cords(j,2)+2
                long_w_sum = long_w_sum + 1;
            end
            if gr_eat_cords(i,1) == wild_cords(j,1)-2 && gr_eat_cords(i,2) ==
wild_cords(j,2)
                long_w_sum = long_w_sum + 1;
            end
            if gr_eat_cords(i,1) == wild_cords(j,1)-2 && gr_eat_cords(i,2) ==
wild_cords(j,2)-2
                long_w_sum = long_w_sum + 1;
            end
            if gr_eat_cords(i,1) == wild_cords(j,1)+2 && gr_eat_cords(i,2) ==
wild_cords(j,2)+1
                long_w_sum = long_w_sum + 1;
            end
```

```
            if gr_eat_cords(i,1) == wild_cords(j,1)+1 && gr_eat_cords(i,2) ==
wild_cords(j,2)+2
                long_w_sum = long_w_sum + 1;
            end
            if gr_eat_cords(i,1) == wild_cords(j,1)+2 && gr_eat_cords(i,2) ==
wild_cords(j,2)-1
                long_w_sum = long_w_sum + 1;
            end
            if gr_eat_cords(i,1) == wild_cords(j,1)+1 && gr_eat_cords(i,2) ==
wild_cords(j,2)-2
                long_w_sum = long_w_sum + 1;
            end
            if gr_eat_cords(i,1) == wild_cords(j,1)-1 && gr_eat_cords(i,2) ==
wild_cords(j,2)-2
                long_w_sum = long_w_sum + 1;
            end
            if gr_eat_cords(i,1) == wild_cords(j,1)-1 && gr_eat_cords(i,2) ==
wild_cords(j,2)+2
                long_w_sum = long_w_sum + 1;
            end
            if gr_eat_cords(i,1) == wild_cords(j,1)-2 && gr_eat_cords(i,2) ==
wild_cords(j,2)+1
                long_w_sum = long_w_sum + 1;
            end
            if gr_eat_cords(i,1) == wild_cords(j,1)-2 && gr_eat_cords(i,2) ==
wild_cords(j,2)-1
                long_w_sum = long_w_sum + 1;
            end
                if gr ~= 0 && nrg_gr_eat(i) > 0 % Обраховуємо стратегію руху агента
                    nrg_gr_eat(i) = nrg_gr_eat(i) + 0.5; % Обраховуємо енергію для
кожного сценарію
                    gr_cords(jj,1) = rand()*n;
                    gr_cords(jj,1) = round(gr_cords(jj,1));
                    gr_cords(jj,2) = rand()*n;
                    gr_cords(jj,2) = round(gr_cords(jj,2));
                elseif gr_sum ~= 0 && nrg_gr_eat(i) > 0
                    nrg_gr_eat(i) = nrg_gr_eat(i) - 0.0005;
                    nrg_gr_eat(i) = nrg_gr_eat(i) + 0.5;
                    gr_eat_cords(i,1) = gr_cords(jj,1);
                    gr_eat_cords(i,2) = gr_cords(jj,2);
                    gr_cords(jj,1) = rand()*n;
                    gr_cords(jj,1) = round(gr_cords(jj,1));
                    gr_cords(jj,2) = rand()*n;
                    gr_cords(jj,2) = round(gr_cords(jj,2));
                elseif long_gr_sum ~= 0 && nrg_gr_eat(i) > 0
                    nrg_gr_eat(i) = nrg_gr_eat(i) - 0.001;
                    nrg_gr_eat(i) = nrg_gr_eat(i) + 0.5;
                    gr_eat_cords(i,1) = gr_cords(jj,1);
                    gr_eat_cords(i,2) = gr_cords(jj,2);
                    gr_cords(jj,1) = rand()*n;
                    gr_cords(jj,1) = round(gr_cords(jj,1));
                    gr_cords(jj,2) = rand()*n;
                    gr_cords(jj,2) = round(gr_cords(jj,2));
                elseif w ~= 0 && nrg_gr_eat(i) > 0
                    nrg_gr_eat(i) = nrg_gr_eat(i) - 0.001;
                    r = rand();
                    if r < 0.25
                        gr_eat_cords(i,1) = gr_cords(jj,1) - 2;
                        gr_eat_cords(i,2) = gr_cords(jj,2) - 2;
                    elseif r >= 0.25 && r <0.5
                        gr_eat_cords(i,1) = gr_cords(jj,1) + 2;
                        gr_eat_cords(i,2) = gr_cords(jj,2) - 2;
```

```
                    elseif r >= 0.5 && r <0.75
                        gr_eat_cords(i,1) = gr_cords(jj,1) - 2;
                        gr_eat_cords(i,2) = gr_cords(jj,2) + 2;
                    elseif r >= 0.75
                        gr_eat_cords(i,1) = gr_cords(jj,1) + 2;
                        gr_eat_cords(i,2) = gr_cords(jj,2) + 2;
                    end
                elseif w_sum ~= 0 && nrg_gr_eat(i) > 0
                    nrg_gr_eat(i) = nrg_gr_eat(i) - 0.001;
                    if gr_eat_cords(i,1) == wild_cords(j,1) + 1 && gr_eat_cords(i,2) ==
wild_cords(j,2)
                        gr_eat_cords(i,1) = wild_cords(j,1) - 2;
                        gr_eat_cords(i,2) = wild_cords(j,2);
                    elseif gr_eat_cords(i,1) == wild_cords(j,1) - 1 &&
gr_eat_cords(i,2) == wild_cords(j,2)
                        gr_eat_cords(i,1) = wild_cords(j,1) + 2;
                        gr_eat_cords(i,2) = wild_cords(j,2);
                    elseif gr_eat_cords(i,1) == wild_cords(j,1) && gr_eat_cords(i,2) ==
wild_cords(j,2) + 1
                        gr_eat_cords(i,1) = wild_cords(j,1);
                        gr_eat_cords(i,2) = wild_cords(j,2) - 2;
                    elseif gr_eat_cords(i,1) == wild_cords(j,1) && gr_eat_cords(i,2) ==
wild_cords(j,2) - 1
                        gr_eat_cords(i,1) = wild_cords(j,1);
                        gr_eat_cords(i,2) = wild_cords(j,2) + 2;
                    elseif gr_eat_cords(i,1) == wild_cords(j,1) + 1 &&
gr_eat_cords(i,2) == wild_cords(j,2) + 1
                        gr_eat_cords(i,1) = wild_cords(j,1) - 2;
                        gr_eat_cords(i,2) = wild_cords(j,2) - 2;
                    elseif gr_eat_cords(i,1) == wild_cords(j,1) + 1 &&
gr_eat_cords(i,2) == wild_cords(j,2) - 1
                        gr_eat_cords(i,1) = wild_cords(j,1) - 2;
                        gr_eat_cords(i,2) = wild_cords(j,2) + 2;
                    elseif gr_eat_cords(i,1) == wild_cords(j,1) - 1 &&
gr_eat_cords(i,2) == wild_cords(j,2) - 1
                        gr_eat_cords(i,1) = wild_cords(j,1) + 2;
                        gr_eat_cords(i,2) = wild_cords(j,2) + 2;
                    elseif gr_eat_cords(i,1) == wild_cords(j,1) - 1 &&
gr_eat_cords(i,2) == wild_cords(j,2) + 1
                        gr_eat_cords(i,1) = wild_cords(j,1) + 2;
                        gr_eat_cords(i,2) = wild_cords(j,2) - 2;
                    end
                elseif long_w_sum ~= 0 && nrg_gr_eat(i) > 0
                    nrg_gr_eat(i) = nrg_gr_eat(i) - 0.001;
                    if gr_eat_cords(i,1) == wild_cords(j,1) + 1 && gr_eat_cords(i,2) ==
wild_cords(j,2) + 2
                        gr_eat_cords(i,1) = wild_cords(j,1) - 1;
                        gr_eat_cords(i,2) = wild_cords(j,2) - 3;
                    elseif gr_eat_cords(i,1) == wild_cords(j,1) - 1 &&
gr_eat_cords(i,2) == wild_cords(j,2) + 2
                        gr_eat_cords(i,1) = wild_cords(j,1) + 1;
                        gr_eat_cords(i,2) = wild_cords(j,2) - 3;
                    elseif gr_eat_cords(i,1) == wild_cords(j,1) + 2 &&
gr_eat_cords(i,2) == wild_cords(j,2) + 1
                        gr_eat_cords(i,1) = wild_cords(j,1) - 3;
                        gr_eat_cords(i,2) = wild_cords(j,2) - 1;
                    elseif gr_eat_cords(i,1) == wild_cords(j,1) + 2 &&
gr_eat_cords(i,2) == wild_cords(j,2) - 1
                        gr_eat_cords(i,1) = wild_cords(j,1) - 3;
                        gr_eat_cords(i,2) = wild_cords(j,2) + 1;
                    elseif gr_eat_cords(i,1) == wild_cords(j,1) + 1 &&
gr_eat_cords(i,2) == wild_cords(j,2) - 2
```

```
                    gr_eat_cords(i,1) = wild_cords(j,1) - 1;
                    gr_eat_cords(i,2) = wild_cords(j,2) + 3;
                elseif gr_eat_cords(i,1) == wild_cords(j,1) - 1 &&
gr_eat_cords(i,2) == wild_cords(j,2) - 2
                    gr_eat_cords(i,1) = wild_cords(j,1) + 1;
                    gr_eat_cords(i,2) = wild_cords(j,2) + 3;
                elseif gr_eat_cords(i,1) == wild_cords(j,1) - 2 &&
gr_eat_cords(i,2) == wild_cords(j,2) - 1
                    gr_eat_cords(i,1) = wild_cords(j,1) + 3;
                    gr_eat_cords(i,2) = wild_cords(j,2) + 1;
                elseif gr_eat_cords(i,1) == wild_cords(j,1) - 2 &&
gr_eat_cords(i,2) == wild_cords(j,2) + 1
                    gr_eat_cords(i,1) = wild_cords(j,1) + 3;
                    gr_eat_cords(i,2) = wild_cords(j,2) - 1;
                elseif gr_eat_cords(i,1) == wild_cords(j,1) + 2 &&
gr_eat_cords(i,2) == wild_cords(j,2) + 2
                    gr_eat_cords(i,1) = wild_cords(j,1) - 3;
                    gr_eat_cords(i,2) = wild_cords(j,2) - 3;
                elseif gr_eat_cords(i,1) == wild_cords(j,1) + 2 &&
gr_eat_cords(i,2) == wild_cords(j,2)
                    gr_eat_cords(i,1) = wild_cords(j,1) - 3;
                    gr_eat_cords(i,2) = wild_cords(j,2);
                elseif gr_eat_cords(i,1) == wild_cords(j,1) - 2 &&
gr_eat_cords(i,2) == wild_cords(j,2)
                    gr_eat_cords(i,1) = wild_cords(j,1) + 3;
                    gr_eat_cords(i,2) = wild_cords(j,2);
                elseif gr_eat_cords(i,1) == wild_cords(j,1) - 2 &&
gr_eat_cords(i,2) == wild_cords(j,2) + 2
                    gr_eat_cords(i,1) = wild_cords(j,1) + 3;
                    gr_eat_cords(i,2) = wild_cords(j,2) - 3;
                elseif gr_eat_cords(i,1) == wild_cords(j,1) + 2 &&
gr_eat_cords(i,2) == wild_cords(j,2) - 2
                    gr_eat_cords(i,1) = wild_cords(j,1) - 3;
                    gr_eat_cords(i,2) = wild_cords(j,2) + 3;
                elseif gr_eat_cords(i,1) == wild_cords(j,1) - 2 &&
gr_eat_cords(i,2) == wild_cords(j,2) - 2
                    gr_eat_cords(i,1) = wild_cords(j,1) - 3;
                    gr_eat_cords(i,2) = wild_cords(j,2) - 3;
                elseif gr_eat_cords(i,1) == wild_cords(j,1) && gr_eat_cords(i,2) ==
wild_cords(j,2) - 2
                    gr_eat_cords(i,1) = wild_cords(j,1);
                    gr_eat_cords(i,2) = wild_cords(j,2) + 3;
                elseif gr_eat_cords(i,1) == wild_cords(j,1) && gr_eat_cords(i,2) ==
wild_cords(j,2) + 2
                    gr_eat_cords(i,1) = wild_cords(j,1) ;
                    gr_eat_cords(i,2) = wild_cords(j,2) - 3;
                end
            else
                nrg_gr_eat(i) = nrg_gr_eat(i) - 0.001;
                r = rand();
                if r < 0.25
                    gr_eat_cords(i,1) = gr_eat_cords(i,1) - 2;
                    gr_eat_cords(i,2) = gr_eat_cords(i,2) - 2;
                elseif r >= 0.25 && r <0.5
                    gr_eat_cords(i,1) = gr_eat_cords(i,1) + 2;
                    gr_eat_cords(i,2) = gr_eat_cords(i,2) - 2;
                elseif r >= 0.5 && r <0.75
                    gr_eat_cords(i,1) = gr_eat_cords(i,1) - 2;
                    gr_eat_cords(i,2) = gr_eat_cords(i,2) + 2;
                elseif r >= 0.75
                    gr_eat_cords(i,1) = gr_eat_cords(i,1) + 2;
                    gr_eat_cords(i,2) = gr_eat_cords(i,2) + 2;
```

```
                    end
                end
                if nrg_gr_eat(i) > 0 % Обраховуємо вік кожного агента
                    age_gr_eat(i) = age_gr_eat(i) + 1;
                else % Обраховуємо смерть кожного агента
                    gr_eat_cords(i,1) = 0;
                    gr_eat_cords(i,2) = 0;
                end
            end

        end
        for i = 1:wild_n % Виконуємо ті ж дії для хижаків, але з пошуком травоїдних
            gre = 0;
            gre_sum = 0;
            long_gre_sum = 0;
            for j = 1:gr_eat_n % Кількість травоїдних в системі
                % Перевіряємо присутність травоїдних під ногами
                if wild_cords(i,1) == gr_cords(j,1) && wild_cords(i,2) == gr_cords(j,2)
                    gre = gre + 1;
                end
                % Перевіряємо ближню відстань на присутність травоїдних
                if wild_cords(i,1) == gr_eat_cords(j,1)+1 && wild_cords(i,2) ==
gr_eat_cords(j,2)+1
                    gre_sum = gre_sum + 1;
                end
                if wild_cords(i,1) == gr_eat_cords(j,1) && wild_cords(i,2) ==
gr_eat_cords(j,2)+1
                    gre_sum = gre_sum + 1;
                end
                if wild_cords(i,1) == gr_eat_cords(j,1)+1 && wild_cords(i,2) ==
gr_eat_cords(j,2)
                    gre_sum = gre_sum + 1;
                end
                if wild_cords(i,1) == gr_eat_cords(j,1) && wild_cords(i,2) ==
gr_eat_cords(j,2)-1
                    gre_sum = gre_sum + 1;
                end
                if wild_cords(i,1) == gr_eat_cords(j,1)+1 && wild_cords(i,2) ==
gr_eat_cords(j,2)-1
                    gre_sum = gre_sum + 1;
                end
                if wild_cords(i,1) == gr_eat_cords(j,1)-1 && wild_cords(i,2) ==
gr_eat_cords(j,2)+1
                    gre_sum = gre_sum + 1;
                end
                if wild_cords(i,1) == gr_eat_cords(j,1)-1 && wild_cords(i,2) ==
gr_eat_cords(j,2)
                    gre_sum = gre_sum + 1;
                end
                if wild_cords(i,1) == gr_eat_cords(j,1)-1 && wild_cords(i,2) ==
gr_eat_cords(j,2)-1
                    gre_sum = gre_sum + 1;
                end
                % Перевіряємо дальню відстань на присутність травоїдних
                if wild_cords(i,1) == gr_eat_cords(j,1)+2 && wild_cords(i,2) ==
gr_eat_cords(j,2)+2
                    long_gre_sum = long_gre_sum + 1;
                end
                if wild_cords(i,1) == gr_eat_cords(j,1) && wild_cords(i,2) ==
gr_eat_cords(j,2)+2
                    long_gre_sum = long_gre_sum + 1;
                end
```

```
            if wild_cords(i,1) == gr_eat_cords(j,1)+2 && wild_cords(i,2) ==
gr_eat_cords(j,2)
                long_gre_sum = long_gre_sum + 1;
            end
            if wild_cords(i,1) == gr_eat_cords(j,1) && wild_cords(i,2) ==
gr_eat_cords(j,2)-2
                long_gre_sum = long_gre_sum + 1;
            end
            if wild_cords(i,1) == gr_eat_cords(j,1)+2 && wild_cords(i,2) ==
gr_eat_cords(j,2)-2
                long_gre_sum = long_gre_sum + 1;
            end
            if wild_cords(i,1) == gr_eat_cords(j,1)-2 && wild_cords(i,2) ==
gr_eat_cords(j,2)+2
                long_gre_sum = long_gre_sum + 1;
            end
            if wild_cords(i,1) == gr_eat_cords(j,1)-2 && wild_cords(i,2) ==
gr_eat_cords(j,2)
                long_gre_sum = long_gre_sum + 1;
            end
            if wild_cords(i,1) == gr_eat_cords(j,1)-2 && wild_cords(i,2) ==
gr_eat_cords(j,2)-2
                long_gre_sum = long_gre_sum + 1;
            end
            if wild_cords(i,1) == gr_eat_cords(j,1)+2 && wild_cords(i,2) ==
gr_eat_cords(j,2)+1
                long_gre_sum = long_gre_sum + 1;
            end
            if wild_cords(i,1) == gr_eat_cords(j,1)+1 && wild_cords(i,2) ==
gr_eat_cords(j,2)+2
                long_gre_sum = long_gre_sum + 1;
            end
            if wild_cords(i,1) == gr_eat_cords(j,1)+2 && wild_cords(i,2) ==
gr_eat_cords(j,2)-1
                long_gre_sum = long_gre_sum + 1;
            end
            if wild_cords(i,1) == gr_eat_cords(j,1)+1 && wild_cords(i,2) ==
gr_eat_cords(j,2)-2
                long_gre_sum = long_gre_sum + 1;
            end
            if wild_cords(i,1) == gr_eat_cords(j,1)-1 && wild_cords(i,2) ==
gr_eat_cords(j,2)-2
                long_gre_sum = long_gre_sum + 1;
            end
            if wild_cords(i,1) == gr_eat_cords(j,1)-1 && wild_cords(i,2) ==
gr_eat_cords(j,2)+2
                long_gre_sum = long_gre_sum + 1;
            end
            if wild_cords(i,1) == gr_eat_cords(j,1)-2 && wild_cords(i,2) ==
gr_eat_cords(j,2)+1
                long_gre_sum = long_gre_sum + 1;
            end
            if wild_cords(i,1) == gr_eat_cords(j,1)-2 && wild_cords(i,2) ==
gr_eat_cords(j,2)-1
                long_gre_sum = long_gre_sum + 1;
            end
            if gre ~= 0 && nrg_wild(i) > 0
                nrg_wild(i) = nrg_wild(i) + 1;
                gr_eat_cords(j,1) = 0;
                gr_eat_cords(j,2) = 0;
                nrg_gr_eat(j) = -1;
            elseif gre_sum ~= 0 && nrg_wild(i) > 0
```

```
                nrg_wild(i) = nrg_wild(i) - 0.0005;
                nrg_wild(i) = nrg_wild(i) + 1;
                wild_cords(i,1) = gr_eat_cords(j,1);
                wild_cords(i,2) = gr_eat_cords(j,2);
                gr_eat_cords(j,1) = 0;
                gr_eat_cords(j,2) = 0;
                nrg_gr_eat(j) = -1;
            elseif long_gre_sum ~= 0 && nrg_wild(i) > 0
                nrg_wild(i) = nrg_wild(i) - 0.001;
                nrg_wild(i) = nrg_wild(i) + 1;
                wild_cords(i,1) = gr_eat_cords(j,1);
                wild_cords(i,2) = gr_eat_cords(j,2);
                gr_eat_cords(j,1) = 0;
                gr_eat_cords(j,2) = 0;
                nrg_gr_eat(j) = -1;
            else
                nrg_wild(i) = nrg_wild(i) - 0.0005;
                r = rand();
                if r < 0.25
                    wild_cords(i,1) = wild_cords(i,1) - 3;
                    wild_cords(i,2) = wild_cords(i,2) - 3;
                elseif r >= 0.25 && r <0.5
                    wild_cords(i,1) = wild_cords(i,1) + 3;
                    wild_cords(i,2) = wild_cords(i,2) - 3;
                elseif r >= 0.5 && r <0.75
                    wild_cords(i,1) = wild_cords(i,1) - 3;
                    wild_cords(i,2) = wild_cords(i,2) + 3;
                elseif r >= 0.75
                    wild_cords(i,1) = wild_cords(i,1) + 3;
                    wild_cords(i,2) = wild_cords(i,2) + 3;
                end
            end
            if nrg_wild(i) > 0
               age_wild(i) = age_wild(i) + 1;
            else
               wild_cords(i,1) = 0;
               wild_cords(i,2) = 0;
            end
        end
    end
end
gr_eat_nn = 0;
for i = 1:gr_eat_n
    if gr_eat_cords(i,1) ~= 0 && gr_eat_cords(i,2) ~= 0
        gr_eat_nn = gr_eat_nn + 1;
    end
end
wild_nn = 0;
for i = 1:wild_n
    if wild_cords(i,1) ~= 0 && wild_cords(i,2) ~= 0
        wild_nn = wild_nn + 1;
    end
end
plot([gr_cords(:,1);gr_cords(1,1)],[gr_cords(:,2);gr_cords(1,2)],'ko',
[gr_eat_cords(:,1);gr_eat_cords(1,1)],[gr_eat_cords(:,2);gr_eat_cords(1,2)],'go',
[wild_cords(:,1);wild_cords(1,1)],[wild_cords(:,2);wild_cords(1,2)],'ro');
title({['alive gr eat sum: ', num2str(gr_eat_nn)]; ['alive wild sum: ',
num2str(wild_nn)]; ['age (days): ', num2str(days)]});
clearvars
```

Оскільки рух травоїдних обраховується перед рухом хижаків, то хижакам важче виживати в світі, але в той же час вони отримують вдвічі більше енергії за поїдання травоїдного.

Переглянемо результати роботи програми на прикладі 50 днів, щоб впевнитися, що тварини дійсно займаються поїданням харчів. Оскільки запасу енергії в них зі старту є на 40 днів, то до призначеного часу залишаться лише тварини, які щось їли.

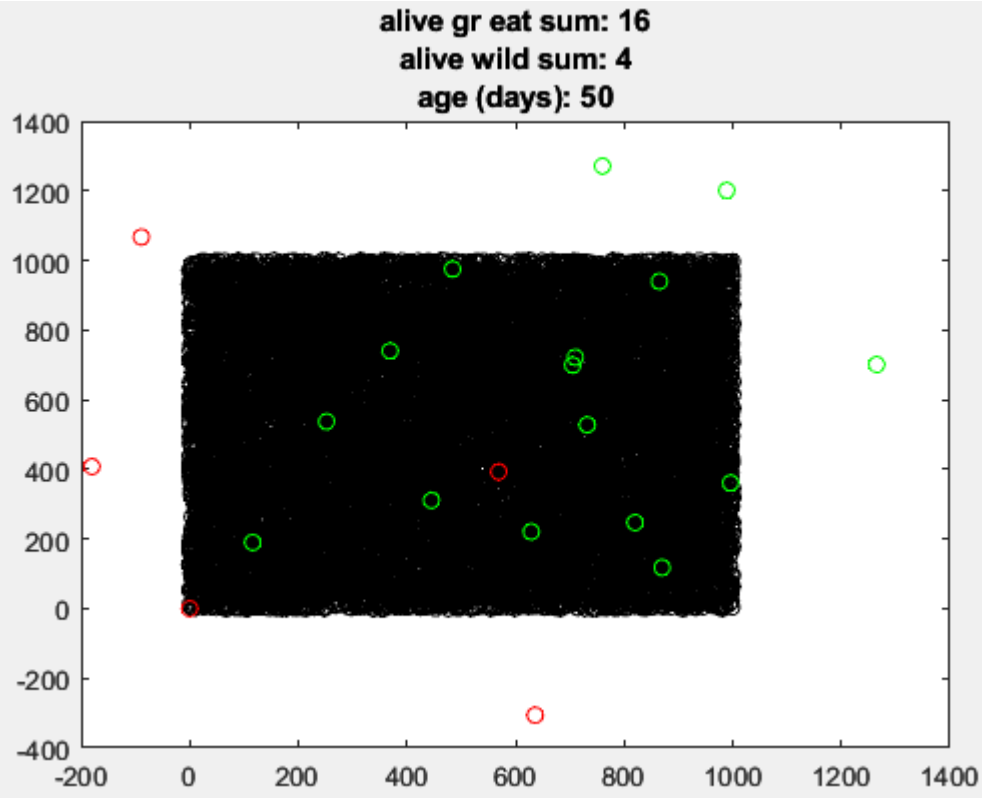


Рисунок 26 –

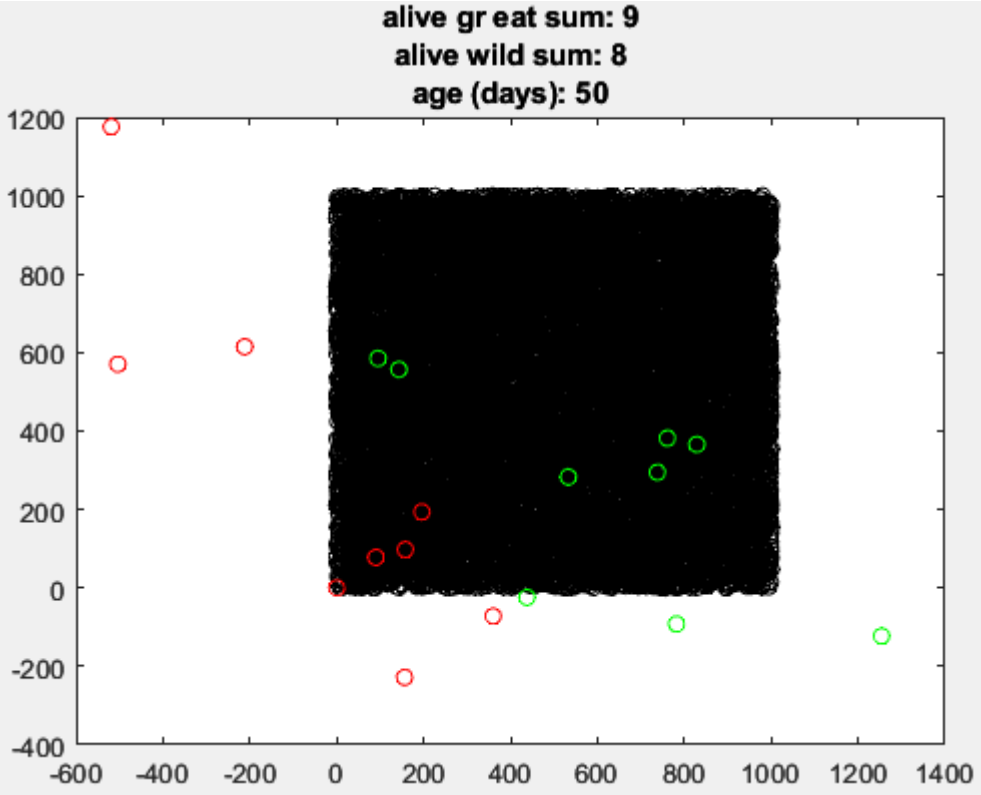


Рисунок 27 –

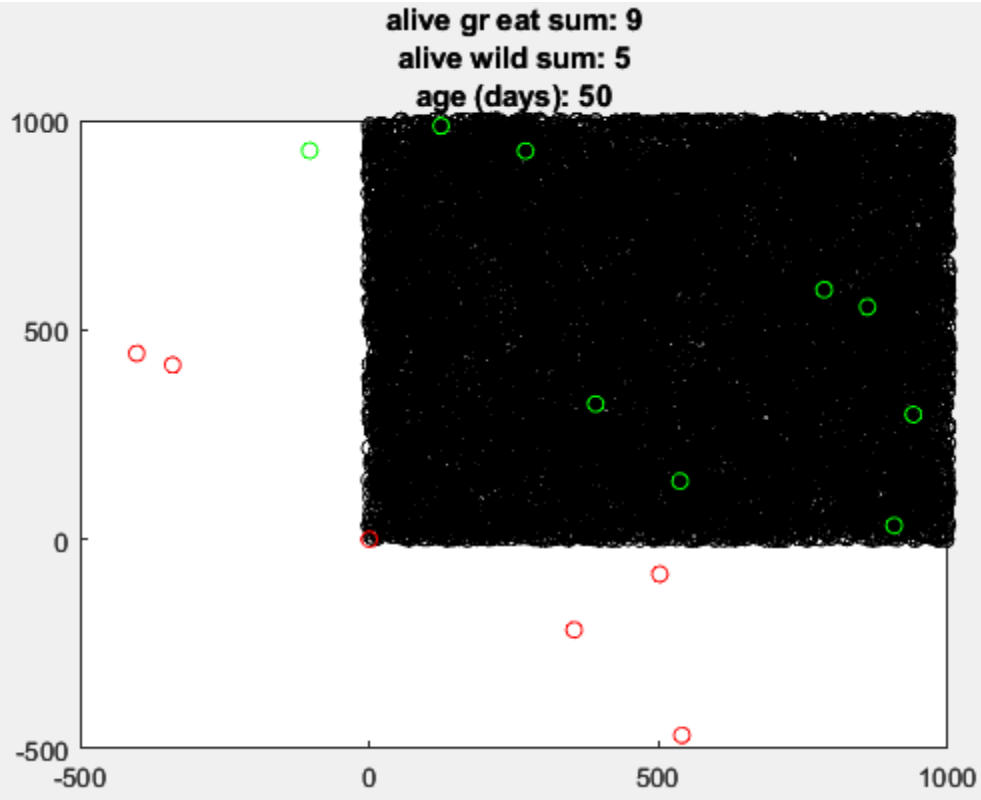


Рисунок 28 –

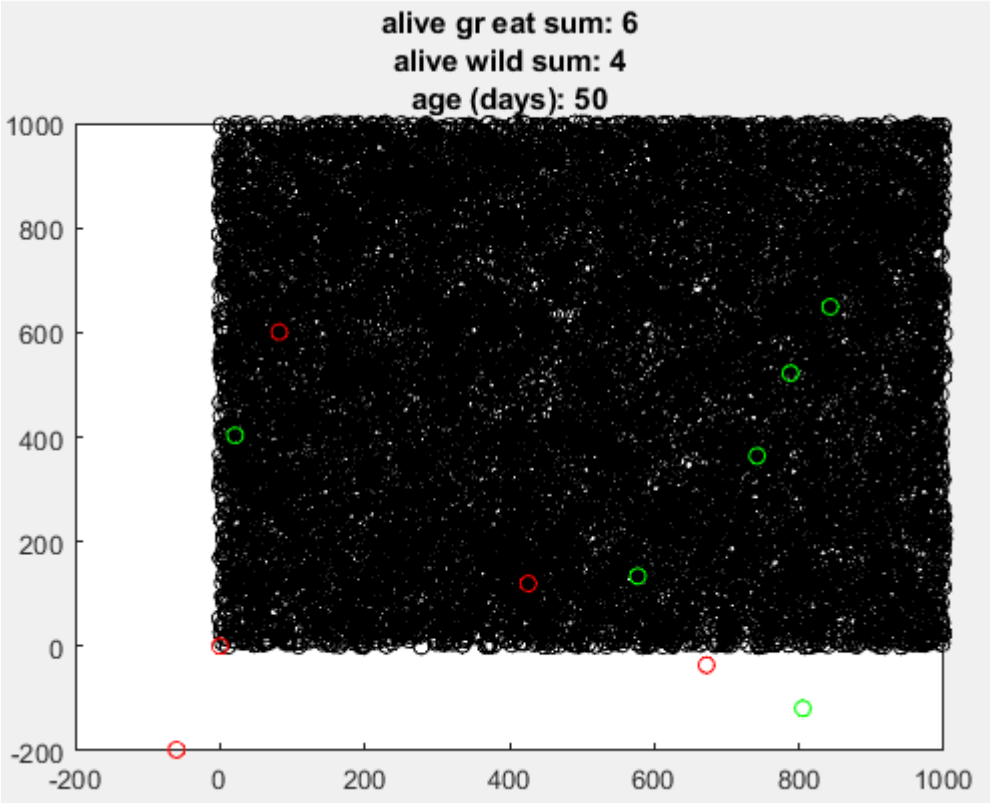


Рисунок 29 –

Бачимо, що за 50 днів життя виживає досить мало тварин. Велика кількість травоїдних помирає від голоду не зважаючи на те, що ліс дуже заповнений рослинами. Також хижаки не стоять на місці і шукають для себе харчування, що дуже зменшує популяцію травоїдних. Але оскільки травоїдні дуже боязкі і одразу тікають від хижаків, то хижакам досить важко впіймати травоїдного через що вони практично всі помирають від голоду за перші 40 днів. Також можемо зробити висновок, що тваринам, які знаходяться на межах лісу часто роблять неправильний вибір в напрямку, оскільки якщо вони нічого не бачать в полі зору, то йдуть навмання. Бачимо, що деякі тварини завдяки тому, що похарчувалися перед виходом з лісу змогли пройти дуже велику відстань за 50 днів.

Отже, була змодельована екосистема, в якій проживають певні жителі, які харчуються по правилах харчового ланцюга (хижаки їдять травоїдних, травоїдні їдять траву). Було визначено, що тваринам досить важко виживати в таких умовах, оскільки вони не завжди приймають правильні рішення у

випадку, якщо варіантів прийнятних рішень кілька. В пріоритеті у тварин це знайти харчування, після чого вони дбають про свою безпеку.

Контрольні запитання

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

2 Яким чином пов’язані штучне життя та синтетична наука про поведінку?

3 В чому полягає сутність моделювання харчових ланцюгів?

4 Проаналізуйте параметри, які визначаються при моделюванні харчових ланцюгів.

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

6 З якою метою використовуються сенсори та активатори?

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

8 Які дії може виконувати агент?

9 Що таке енергія та метаболізм?

10 При виконанні якої умови агент допускається до відтворення?

11 Якими способами агент може загинути?

12 Проаналізуйте засоби пакету Матлаб, які були використані при моделюванні харчового ланцюгу.

РОБОТА № 3

Розробка інтелектуальної інформаційної системи на основі інтелектуальних Web-агентів.

Мета роботи: навчитися створювати інтелектуальні інформаційні системи на основі інтелектуальних Web-агентів. які взаємодіють із певними службами в середовищі Internet і реалізують деякі корисні завдання для користувачів, наприклад, збір необхідної інформації

Основні теоретичні відомості

Агент - це апаратна або програмна сутність, здатна діяти в інтересах досягнення цілей, поставлених перед нею користувачем.

В межах мультиагентної парадигми програмні агенти являють собою автономні компоненти, що діють від імені користувача.

Агентів також називають розумними агентами (Intelligent agent), тому що розумність є ключовим чинником при їхньому створенні. Хоча агенти можуть приймати різні форми, але вони всі реалізують якісь корисні функції за людину. Агент може одержати завдання, яке необхідно виконати, і приймати потрібні рішення в процесі комунікації з іншими агентами (можливо, навіть із агентами, що представляють інтереси інших користувачів).

Властивості програмних агентів

Агенти можуть мати одну або декілька властивостей, наведених в табл.

Таблиця 3.1 - Властивості програмних агентів

Властивість	**Визначення**
Автономність	Можливість діяти незалежно від користувача
Адаптивність	Здатність до навчання під час роботи
Комунікативність	Здатність до комунікації з користувачем або іншими агентами
Здатність до співробітництва	Працює з іншими агентами для досягнення мети
Персоніфікованість	Поводиться природно (проявляє емоції)
Мобільність	Можливість переміщатися по навколишньому середовищу

3.1.

Однією з основних властивостей, які асоціюються із програмними агентами, є автономність (Autonomy). Агент вважається автономним, якщо він здатний діяти без прямого управління людиною. Поняття автономності включає також наявність в агента декількох цілей, яких він може досягти. Наявність задач також сприяє тому, що агент одержує можливість планувати.

Агент є адаптивним (Adaptivity), якщо він може змінювати своє поводження на підставі досвіду. Він повинен учитися й робити висновки залежно від стану навколишнього середовища й своїх знань.

Ще однією важливою характеристикою агентів є здатність до комунікації (Communicative). Агент повинен уміти спілкуватися з користувачем, щоб визначати свої завдання або ідентифікувати початкову інформацію. Крім того, агентові необхідно спілкуватися з навколишнім середовищем. Наприклад, пошуковий агент, що знаходить цікаві Web-сторінки, повинен уміти спілкуватися за допомогою протоколу HTTP, щоб підключатися до серверів і одержувати від них інформацію (шукані сторінки).

Агент здатний до співробітництва (Collaborative), якщо він може спілкуватися з іншими агентами для вирішення своїх завдань. Наприклад, агенти, які діють від імені користувачів на аукціонах, можуть брати участь в укладанні спільних угод, орієнтуючись на інформацію, отриману від користувачів. Подібні агенти звичайно входять у системи з більшою кількістю агентів, тому що для співробітництва потрібна безліч агентів.

Для деяких агентів найбільш істотною є здатність до персоналізації. Програми з їхнім застосуванням звичайно використовуються в розважальних цілях. Тому, якщо персоналізація агента буде змінювати його залежно від переданої інформації, користувач зможе зрозуміти те, що агент намагається повідомити. Наприклад, коли оповідач про що-небудь розповідає й вираз його особи при цьому не змінюється, слухачеві може бути важко його зрозуміти. Інформація, закодована в людських почуттях, дозволяє одержати додаткові

відомості, тому можливість персоналізації може бути важливою для агентів певного класу.

Деякі агенти мають мобільність, або можливість переміщення по навколишньому середовищу. Ця властивість є дуже корисною при створенні певних програм. Властивість мобільності використовується у програмних агентах, що одержують інформацію з віддаленої бази даних. При звичайних умовах хост приймає інформацію від бази даних, збирає потрібні відомості, а потім фільтрує їх за допомогою критеріїв, заданих користувачем. При одержанні мобільності агент може бути спрямований у віддалену базу даних, щоб автоматично відфільтрувати результати, а потім повернути тільки ті, які необхідні користувачеві. Це дозволяє як заощадити на пропускній здатності каналу, так і спростити архітектуру, особливо в тому випадку, якщо два хости можуть бути від’єднанні один від одного.

Агент необов'язково повинен мати всі властивості. У більшості випадків використовуються всього дві.

Класифікація агентів

Незалежні агенти можна класифікувати наступним чином.

1 Агенти штучного життя.

2 Програмні агенти.

3 Агенти для завдань.

4 Агенти для розваг.

5 Віруси.

6 Інші агенти.

Агент, використання якого залежить від завдання, застосовується для вирішення певної проблеми. Прикладом може служити пошуковий агент у мережі Internet. Крім пошуку за заданими критеріями, агент також стежить за тим, які зі знайдених посилань відкриє користувач. Це дозволяє створити шкалу цінностей пошуку, що сприяє кращому сортуванню результатів, а також прискорює одержання потрібного результату. Пошуковий агент може

працювати у фоновому режимі, повторюючи пошук, щоб знайти й надати користувачеві нову інформацію.

Пошукові агенти володіють не тільки автономністю, але й адаптивністю, оскільки вони вивчають найцікавіші для користувача теми, ґрунтуючись на його реакції (тобто посиланнях, які відкриває користувач).

Іншим прикладом може служити агент, що використовується для роботи на аукціоні. Він здатний діяти від імені користувача, купуючи товари в мережі Internet по найнижчій ціні. Він може робити це в співробітництві з іншими агентами. Знайшовши агентів, які намагаються купити схожі товари, можна зробити спільну оптову покупку групи товарів за зниженою ціною.

Розважальні агенти корисні для взаємодії у віртуальному світі або подання персонажа в якості інтерфейсу користувача.

Вірус по-іншому можна назвати небезпечним мобільним програмним агентом. Хоча вірус не обов'язково повинен бути адаптивним, він є автономним і мобільним. Замість того щоб використовувати спеціальний протокол для переміщення по мережі, вірус застосовує стандартні протоколи й найчастіше протокол передачі пошти.

Розумні агенти. Хоча зробити агента розумним досить складно, існує безліч методів, які можна використовувати для того, щоб додати агентові здатність до прийняття розумних рішень. До таких методів належать:

- нейронні мережі;
- системи, засновані на знаннях;
- дерева вирішувальних правил;
- еволюційні методи;
- нечітка логіка;
- алгоритми кластеризації.

Web-агент. Завдання Web-агенту полягає в тому, щоб взаємодіяти з певними службами у мережі Internet (наприклад, із службами новин) і збирати інформацію на підставі критеріїв, заданих користувачем. Після збирання інформації Web-агент надає її користувачеві в тому порядку, який найбільш

відповідає запиту (починаючи з даних, що мають найвищий рейтинг за умовами пошуку).

Web-агент використовує стандартні протоколи й передає користувачеві інформацію через звичайний браузер. Для спілкування з користувачем агент містить HTTP-сервер, для зв'язку із зовнішніми серверами є HTTP-клієнтом, а для одержання новин функціонує як клієнт протоколу NTTP (Рисунок 30).

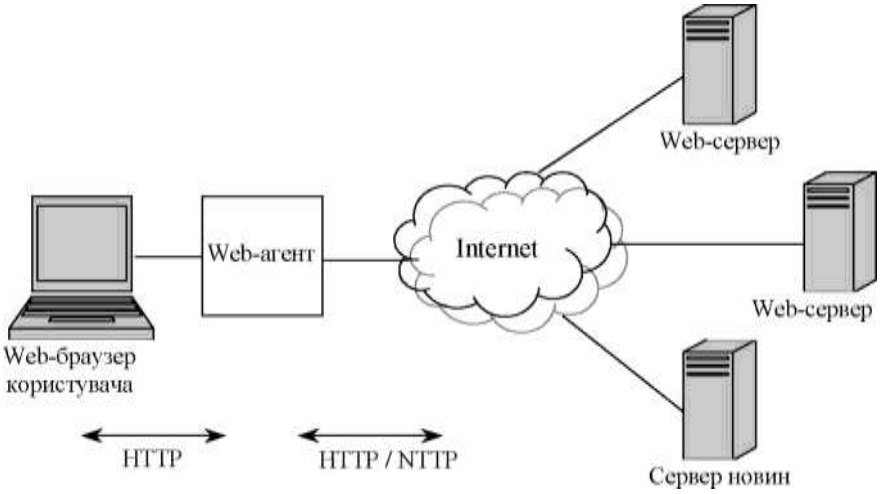


Рисунок 30 – Архітектура Web-агента

Web-агент буде відігравати роль додатка, що здійснює взаємодію між користувачем і мережею Internet при читанні новин Користувач змінює параметри агента за допомогою простого файлу конфігурації. У ньому користувач указує Web-сторінки для моніторингу (для видачі повідомлення у випадку їхньої зміни) і групи новин для перегляду й визначає специфічні критерії пошуку. Коли агент знаходить об'єкти, які відповідають заданому критерію, він зберігає їх і видає користувачеві у вигляді Web-сторінки, посилання на якій відсортовані в потрібному порядку. Користувач може клацнути по посиланню на певний Web-сайт або рядку новин і перейти до потрібної статті.

Завдання агента полягає в тому, щоб спростити читання новин, відфільтрувати непотрібну інформацію й представити новини в порядку, при якому найцікавіші дані розташовані на першому місці.

Web-агент є дуже простим і не має здатності до навчання. Він володіє двома властивостями: автономністю (здатний працювати у фоновому режимі

без прямого керування з боку користувача) і здатністю до комунікації із зовнішніми серверами при зборі інформації.

Сенсорами агента є стандартні протоколи, які дозволяють йому збирати потрібну інформацію за заданими критеріями пошуку. Для передачі результатів пошуку користувачеві використовується протокол HTTP. Потім користувач підключається до Web-агента, як до будь-якого іншого Web-сервера.

Для отримання інформації з середовища Internet Web-агент виступає в якості NTTP-клієнта, який реалізує набір інтерфейсів для взаємодії з Internet серверами, наприклад, із серверами новин. Ці інтерфейси дозволяють додатку з'єднуватись із сервером новин, задавати групи новин за певними критеріями, переходити до заголовку статті, зчитувати всю статтю, аналізувати повідомлення, пропускати поточну новину та переривати зв'язок із сервером новин.

Протокол NTTP є інтерактивним протоколом, який повністю базується на ASCII-командах. Нижче наведений приклад роботи із NTTP-сервером новин yellow.geeks.org за допомогою Telnet. Дані, що вводить користувач, виділені жирним текстом.

root@plato /root]# **telnet yellow.geeks.org 119**

S: 201 plato.mtjones.com DNEWS Version 5.5dl, SO, posting OK

C**: list**

S: 215 list.of newsgroups follows

S: control 2 3 y

S: control.cancel 2 3 y

S: my.group 10 3 y

S: new.group 6 3 y

S: .

C: **group my.group**

S: 211 8 3 10 my.group selected

C**: article 3**

S: 220 3 <3C36AF8E. 1BD3047E@mtjones.com> article retrieved

S: Message-ID: <3C36AF8E. 1BD3047E@mtjones.com>

S: Date: Sat, 05 Jan 2002 00:47:27 -0700

S: From: "M. Tim Jones" <mtj@mtjones.com>

S: X-Mailer: Mozilla 4.74 [en] (Win98; U)

S: X-Accept-Language: en

S: MIME-Version: 1.0

S: Newsgroups: my.group

S: Subject: this is my post

S: Content-Type: text/plain; charset=us-ascii

S: Content-Transfer-Encoding: 7bit

S: NNTP-Posting-Host: sartre.mtjones.com

S: X-Trace: plato.mtjones.com 1010328764 sartre.mtjones.com (6 Jan 2002 07:52:44 -0700)

S: Lines: 6

S: Path: plato.mtjones.com

S: Xref: plato.mtjones.com my.group:3

S:

S:

S: Hello

S:

S: This is my post.

S:

S:

S: .

C: **date**

S: 111 20020112122419

C: **quit**

S: 205 closing connection - goodbye!

У наведеному прикладі показано, як з'єднання з NTTP-сервером створюється за допомогою Telnet-клієнта. Сервер новин відповідає на запит, у якому вказується тип сервера й інша необхідна інформація. Тепер через з'єднання можна давати команди. Існує два базових типи відповідей, які можна чекати від сервера - з одного рядка й з декількох рядків. Щоб ідентифікувати кінець відповіді, протокол NNTP, як і протокол SMTP, застосовує символ “.” у рядку, (наприклад, відповідь на команди list і article).

Завдання до роботи

1. Ознайомитися з основними теоретичними відомостями за темою роботи.

2. Розробити програмного Web-агента, який взаємодіє із певними службами в Internet та збирає інформацію на основі критеріїв, що задані користувачем. Програмний Web-агент має відповідати вимогам, наведеним в п. 3.2.

3. За допомогою протоколу HTTP Web-агент має перевіряти зміни заданих Web сайтів, на яких відображається потрібна інформація (наприклад, новини). При цьому зв’язок з сайтами організовується за допомогою інтерфейсу сокетів. Протокол мережевого рівня - IP, протокол транспортного рівня - TCP, протокол прикладного рівня - NTTP (порт 80). Перевірку на зміну сайтів з новинами рекомендовано виконувати шляхом перевірки розміру файлу, що пропонується скачати.

4. За допомогою протоколу NTTP Web-агент має відбирати на вказаних серверах потрібну інформацію (наприклад, на сайті новин, - статті), яка відповідає критеріям, що задані користувачем. Критеріями можуть бути: група новин та окремі слова, що мають зустрічатися в новині. Зв’язок з сервером новин має бути організований за допомогою інтерфейсу сокетів. Протокол мережевого рівня - IP, протокол транспортного рівня - TCP, протокол прикладного рівня - NTTP (порт 119). При цьому врахувати наступні особливості серверів новин:

- після підключення до серверу має бути повернуте повідомлення, що

вміщує код 201 у випадку вдалого підключення;

- для встановлення заданої групи новин необхідно відправити команду: group - назва_групи, при цьому в результаті успішності має бути повернуте повідомлення, що вміщує наступну послідовність: код 211, загальну кількість повідомлень даної групи, номер першого та останнього повідомлення;

- для завантаження заголовку повідомлення необхідно відправити команду: head - номер_повідомлення, при цьому, якщо все було виконано успішно, то повертається код 221 та заголовок повідомлення;

- для завантаження усього повідомлення необхідно відправити серверу команду: article номер_повідомлення. Якщо повідомлення успішно передано, то повертається код 220 та саме повідомлення;

- для ідентифікації кінця відповіді сервера використовується рядок з символом “.”

5. Web-агент має надавати користувачу Web-інтерфейс. При цьому мають бути передбачені наступні можливості:

- можливість перегляду поточної конфігурації (тобто конфігурацію можна переглянути, прописавши в Web-браузері посилання наступного типу: http://192.168.115.2:8080/conf.html - в випадку, коли Web-агент працює на комп’ютері з ip-адресою 192.168.115.2 на порті 8080);

- можливість перегляду списку http-сайтів та nttp-серверів новин, на яких відбулися зміни (наприклад, http://192.168.115.2:8080/index.html). Тобто має бути відображена інформація в вигляді двох таблиць: таблиця http-сайтів, що змінилися, та таблиця повідомлень на серверах новин. При цьому для цих повідомлень має бути відображена група, до якої вони відносяться, а також задані слова, що були знайдені в цих повідомленнях. Послідовність відображення новин має відповідати їхній відповідності критеріям користувача;

- можливість перегляду повідомлення з серверу новин у web-браузері користувача;

- надати можливість користувачу вказати http-сервери, що було переглянуто, тобто які вже не треба виводити у таблиці до їх наступної зміни.

6. Робота зі збору інформації про стан серверів та робота із користувачем мають не заважати одна одній.

7. Конфігурація роботи Web-агента має розміщуватися в окремому конфігураційному файлі, в якому можна вказати наступні параметри:

- http-сайти з новинами;
- сервери новин;
- групи новин та відповідні їм слова, що мають знаходитися в середині повідомлень.

8. Виконати тестування розробленого програмного агента.

9. З метою створення підсистеми автоматичного поповнення бази знань інтегрувати створеного програмного агента до інтелектуальної системи підтримки прийняття рішень. При необхідності з метою забезпечення інтеграції розробити додаткове програмне, інформаційне, лінгвістичне, математичне забезпечення.

10. Виконати тестування розробленого програмного забезпечення (інтелектуальна система підтримки прийняття рішень з інтегрованим Web-агентом) шляхом звертання за допомогою Web-агенту до заданих користувачем Web-сторінок із динамічно змінюваною інформацією з метою поповнення фактів бази знань.

11. Проаналізувати роботу створеного програмного агенту. Виявити його переваги та недоліки.

12. Оформити звіт з роботи.

13. Відповісти на контрольні питання.

Приклад виконання роботи

Було реалізовано Web-агент, що аналізує інформацію на сайті з результатами тенісних турнірів на мові програмування Python з виводом результатів відбору інформації в Telegram. Файл з конфігурацією запиту зберігається окремо і легко може бути модифікований.

Основний код програми:

```
from parser import Parser, Match
import config
from parser import Match
import telebot
from DataBase.db import DataBase
import schedule
import time
import datetime
BOT = telebot.TeleBot(config.TOKEN)

def edit_message(match:Match, message_id:int, verification:str, new_score:str, db:DataBase):
    try:
        message = f"""
Лига: {match.championship}
{match.name_at} vs {match.name_ht}
Начало игры: {match.time_start_game}
{match.total} {verification}
С 1 по 3 партию
{match.url}
{", ".join([":".join(list(map(str, i))) for i in new_score])}
"""
        BOT.edit_message_text(message, config.PEER_ID, message_id)
        time.sleep(config.DELAY_BETWEEN_API_REQUESTS)
        db.verification_confirmation(match.hash_game, True)
    except Exception as err:
        print(err)

def check_strategy(db:DataBase):
    try:
        for game in db.get_unverified_games():
            match = Match(
                game[0],
                game[1],
                game[2],
                game[3],
                game[4],
                game[5],
                game[6],
                game[7],
                game[8],
                game[9],
            )

            stat_url = match.url
            parser = Parser()
            table = parser._table(stat_url)
            last_game = parser.get_day_ago_game(table)
            print(last_game)
            if isinstance(last_game, int) and last_game <= 0:
                print("edit message!")
                new_score = parser.get_score(table)[1]

                for i in new_score[0:2]:
                    if (sum(i) <= config.SUM_SCORE_OF_GAME + 0.5 and match.total == "Тм") or
                       (sum(i) >= config.SUM_SCORE_OF_GAME - 0.5 and match.total == "Тб"):
                        edit_message(match=match, message_id=game[11], verification="+",
new_score=new_score, db=db)
                    else:
                        edit_message(match=match, message_id=game[11], verification="-",
new_score=new_score, db=db)

    except Exception as err:
```

```
        print(err)

def main():
    try:
        parser = Parser()
        base = DataBase(config.PATCH_DB)
        for stats_match in parser.start_parsing():
            print(stats_match)
            if stats_match.day_ago_game <= config.DAY_AGO and stats_match.total_score in
config.TOTAL_SCORE and stats_match.total:
                text = f"""
Лига: {stats_match.championship}
{stats_match.name_at} vs {stats_match.name_ht}
Начало игры: {stats_match.time_start_game}
{stats_match.total}
{stats_match.url}
"""
                if  not base.game_on_table(stats_match):
                    result = BOT.send_message(config.PEER_ID, text=text)
                    print("send message!")
                    time.sleep(config.DELAY_BETWEEN_API_REQUESTS)
                    base.entry_message_id(stats_match.hash_game, result.message_id)
        check_strategy(db=base)
        print("Finished! Waiting for more...")
    except Exception as err:
        print(err)

def timer(minutes:int):
    schedule.every().hours.do(main)
    while True:
        schedule.run_pending()
        time.sleep(1)

if __name__ == "__main__":
    print("Bot is running!")
    main()
    timer(config.DELAY)
```

Клас Match та Parser:

```
import cloudscraper
import re
from dataclasses import dataclass
from bs4 import BeautifulSoup
import datetime
import hashlib
import config
import time

@dataclass
class Match:
    hash_game: str
    championship: str
    time_start_game: str
    day_ago_game: int
    name_at: str
    name_ht: str
    total_score: str
    score: list
    total: str
    url: str
```

```
class Parser:

    _scraper = cloudscraper.create_scraper()
    def _get_data(self, url):
        try:
            print(url)
            data = self._scraper.get(url)
            print(data.status_code)
            time.sleep(1)
            return data
        except Exception as err:
            print(err)
    def get_championship(self, path, key) -> str:
        try:
            return path.get(f"{key}").get("name_ch")
        except Exception as err:
            print(err)
    def get_time_start_game(self, path, key, id) -> str:
        try:
            return path.get(f"{key}").get("evts").get(id).get("date_ev_str")
        except Exception as err:
            print(err)
    def get_players(self, path, key, id) -> tuple:
        try:
            name_ht = path.get(f"{key}").get("evts").get(id).get("name_ht")
            name_at = path.get(f"{key}").get("evts").get(id).get("name_at")
            return name_ht, name_at
        except Exception as err:
            print(err)
    def _get_url(self, path, key, id) -> str:
        try:
            link = path.get(f"{key}").get("evts").get(id).get("stat_link")
            if link:
                return f"https://betcity.ru/ru/mstat/{link}"
        except Exception as err:
            print(err)
    def get_score(self, table):
        try:
            if table:
                all_score = table[-1].find("td", class_="score").text.replace(",", "").replace("(",
"").replace(")", "").split()
                score = [list(map(int, num.split(":"))) for num in all_score[1:]]
                total_score = str(all_score[0])
                return total_score, score
            else:
                return None, None
        except Exception as err:
            print(err)
    def _table(self, stats_url):
        try:
            if stats_url:
                stats = self._get_data(stats_url).text
                soup = BeautifulSoup(stats, "html5lib")
                table = soup.find_all("table")
                if table and len(table) >= 5:
                    return table
        except Exception as err:
            print(err)
    def get_day_ago_game(self, table) -> int:
        try:
            if table:
```

```
                date_last_game = table[-1].find("td", class_="date")
                if date_last_game:
                    date_now = datetime.datetime.today().date()
                    date_last_game = list(map(int, date_last_game.text.split(".")))
                    date_last_game = datetime.datetime(date_last_game[2]+2000, date_last_game[1],
date_last_game[0]).date()
                    day_ago_game = (date_now-date_last_game).days
                    return day_ago_game
        except Exception as err:
            print(err)
    def get_total(self, total_score:str, score:list) -> str:
        try:
            if total_score in config.TOTAL_SCORE:
                counttm = 0
                counttb = 0
                for num in score:
                    if sum(num) >= config.SUM_SCORE_OF_GAME:
                        counttb += 1
                    else: counttm += 1
                if counttb >= 3 or counttm >= 3:
                    total = "Тм" if counttb >= 3 else "Тб"
                    return total
        except Exception as err:
            print(err)

    def filter_championship(self, championship_name:str) -> bool:
        try:
            for keyword in config.IN_NAME_CHAMPIONSHIP:
                if re.search(keyword, championship_name) == None:
                    for notkeyword in config.NOT_IN_NAME_CHAMPIONSHIP:
                        if re.search(notkeyword, championship_name) != None:
                            return False
            return True
        except Exception as err:
            print(err)
    def start_parsing(self) -> Match:
        try:
            main_path =
self._get_data("https://ad.betcity.ru/d/on_air/soon").json().get("reply").get("sports").get("46").get("
chmps")
            for key in main_path.keys():
                championship = self.get_championship(main_path, key)

                if self.filter_championship(championship_name=championship):
                    for id_ in main_path.get(f"{key}").get("evts"):
                        url = self._get_url(main_path, key, id_)
                        table = self._table(url)

                        time_start_game = self.get_time_start_game(main_path, key, id_)
                        day_ago_game = self.get_day_ago_game(table)
                        name_at, name_ht = self.get_players(main_path, key, id_)
                        total_score, score = self.get_score(table)

                        if None not in (championship, url, time_start_game, day_ago_game, name_at,
name_ht, score, total_score):
                            text_for_get_hash =
f"{championship}{time_start_game}{day_ago_game}{name_at}{name_ht}{total_score}{score}{url}"
                            hash_game = hashlib.md5(text_for_get_hash.encode('utf-8')).hexdigest()
                            total = self.get_total(total_score, score)
                            if total:
                                yield Match(
                                    hash_game,
```

```
                                    championship,
                                    time_start_game,
                                    day_ago_game,
                                    name_at,
                                    name_ht,
                                    total_score,
                                    score,
                                    total,
                                    url
                                )
        except Exception as err:
            print(err)

if __name__ == "__main__":
    for i in Parser().start_parsing():
        print(i)
```

Файл конфігурації:

```
TOTAL_SCORE = ("1:3", "3:1", "3:0", "0:3", "3:2", "2:3")
DAY_AGO = 15
SUM_SCORE_OF_GAME = 75.5
NOT_IN_NAME_CHAMPIONSHIP = ("Росія", "Білорусія", "TT Elite Series")
IN_NAME_CHAMPIONSHIP = ("TT-Cup", "Чехія", "Ліга Про", "Чехії")
DELAY = 60
TOKEN = "2143279582:AAGT4RWvIaJots4KcF7IFC6QLfQx7v5f64I"
PEER_ID = -1001821864423
PATCH_DB = "table.db"
DELAY_BETWEEN_API_REQUESTS = 3
```

В даному випадку робимо пошук за фінальними рахунками, давністю матчу, загальним рахунком, вказуємо заборонені дані в новині, вказуємо обов’язкові дані в новині (В даному випадку це чеські матчі ліги Про), вказуємо затримку для повторної перевірки, вказуємо токен нашого Телеграм боту, який буде публікувати цю новину в спільноті, вказуємо ІД спільноти, в яку буде публікуватися дана новина, вказуємо ім’я файлу з зібраною базою даних, вказуємо затримку між кожним запитом.

Для демонстрації роботи був створений канал https://t.me/laba_3_Nesteruk, який містить в собі бота https://t.me/laba234bot, що раніше вже використовувався в лабораторних роботах з інших предметів.

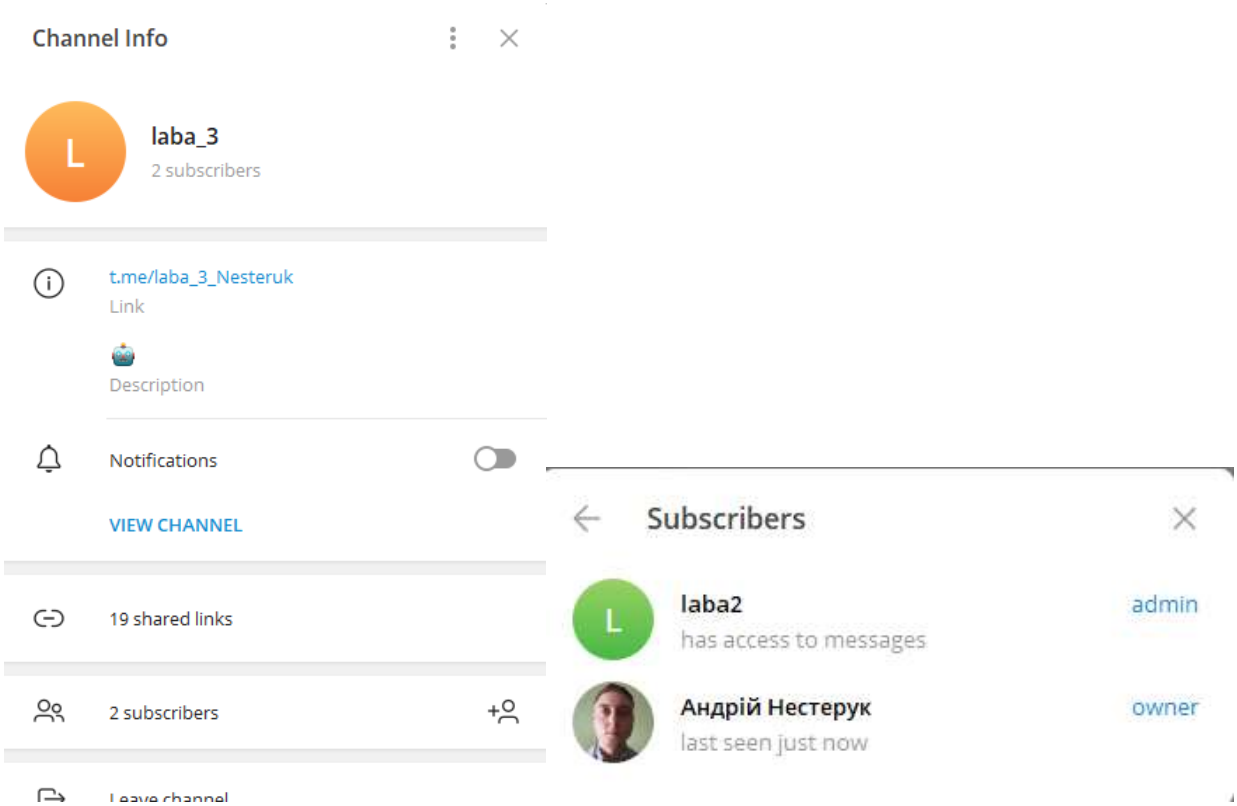


Рисунок 31 –

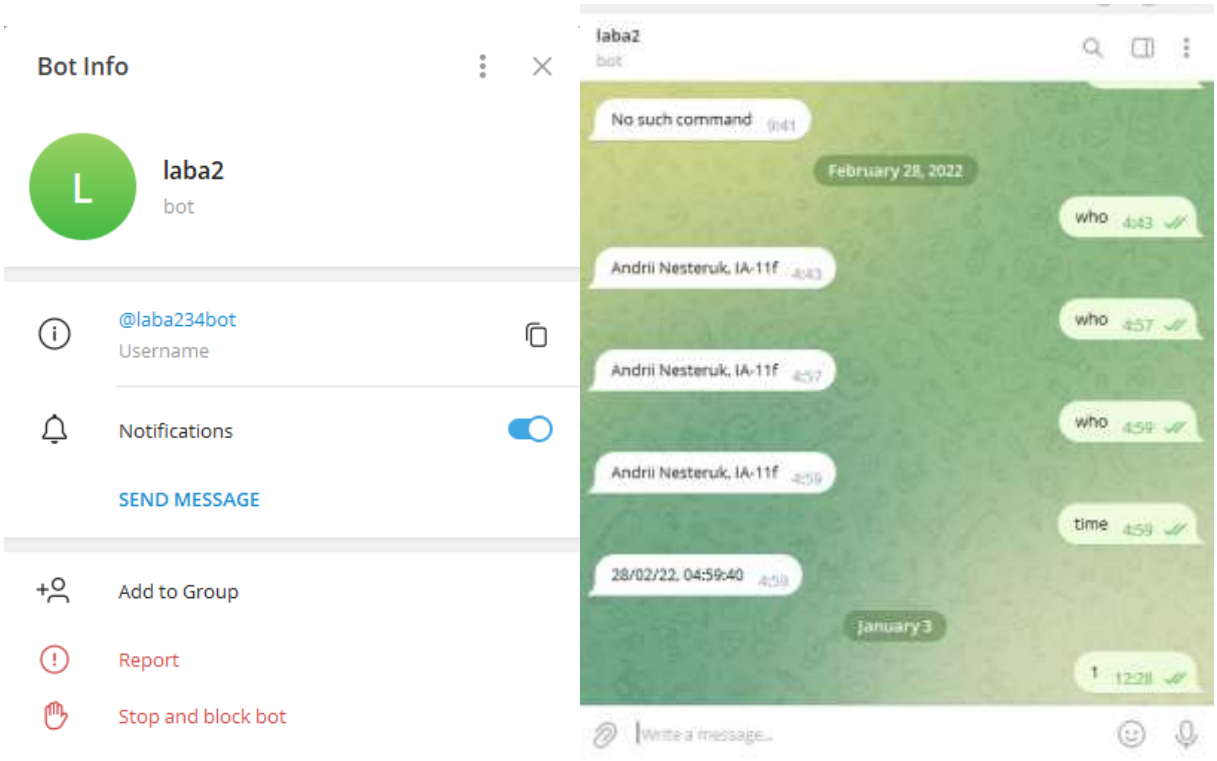


Рисунок 32 –

Бачимо, що програма показує код 200, що означає, що запит виконаний успішно і у випадку, якщо проаналізована новина співпадає з нашим запитом,

то інформація про неї виводиться в консоль, а також заповнюється в базу даних, яка надсилається боту, який її публікує.

Рисунок 33 –

Можемо бачити, що посилання співпадають в консолі, та опубліковані ботом.

Рисунок 34 –

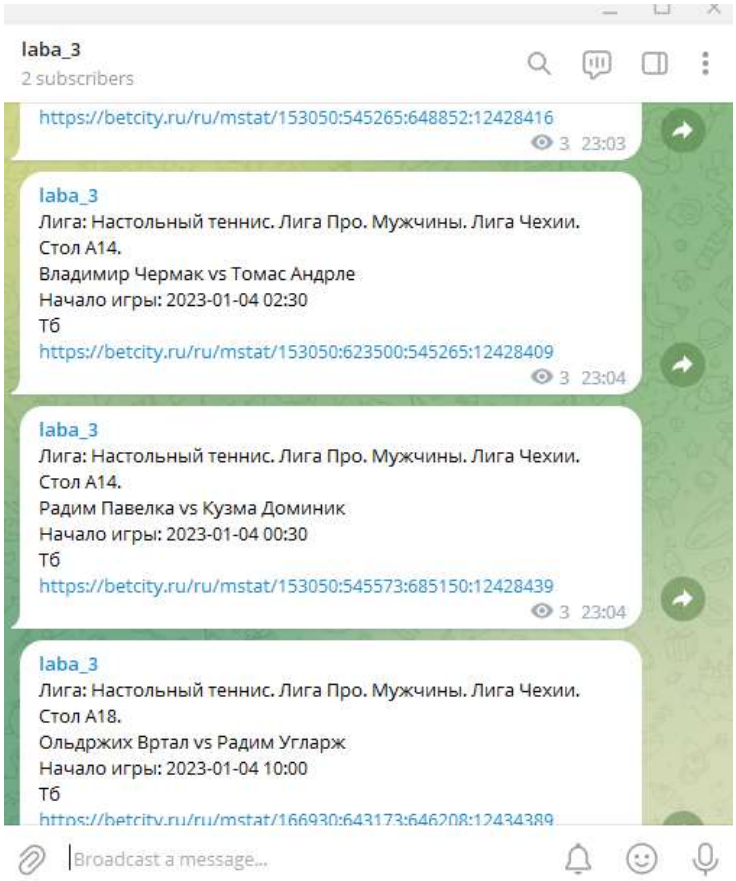


Рисунок 35 –

Отже, було створено Web-агент, який аналізує новини по заданим параметрам, які можна легко змінювати. При знаходженні відповідних новин, програма надсилає дані для публікації.

Контрольні запитання

1 Що являє собою програмний агент?

2 Поясніть та проаналізуйте властивості агентів: автономність, адаптивність, комунікативність, здатність до співробітництва, персоніфікованість, мобільність.

3 З якою метою застосовуються агенти, використання яких залежить від завдань?

4 Що таке розважальні агенти?

5 Проаналізуйте комп’ютерні віруси як програмні агенти.

6 Які методи використовуються для надання інтелектуальності програмним агентам?

7 Які протоколи використовує Web-агент?

8 Виконайте аналіз архітектури Web-агенту?

9 Якими властивостями володіє Web-агент?

10 Проаналізуйте протокол, який використовується для передачі користувачу результатів пошуку за допомогою Web-агенту.

11 Наведіть і поясніть схему потоків усередині Web-агенту?

12 Проаналізуйте основні функції обраного інструментарію розробки Web-агенту: внутрішня структура, параметри, основні змінні, їх призначення та використання.

ПЕРЕЛІК ДЖЕРЕЛ

1. Джонс М. Т. Программирование искусственного интеллекта в приложениях / М. Тим Джонс ; Пер. с англ.Осипов А. И. – М. : ДМК Пресс, 2011.– 312 с.: ил.
2. Згуровський М.З..Зайченко Ю.П. Системи і методи штучного інтелекту [Електронний ресурс] : підруч. для здобувачів ступеня магістра за спец. галузі знань 12 Інформаційні технології / М. З. Згуровський, Ю. П. Зайченко ; КПІ ім. Ігоря Сікорського. – Електрон. текст. дані (1файл). – Київ : КПІ ім. Ігоря Сікорського, 2024. – 710 с.