Продолжение таблицы 2

1	2	3	4	5
Visual Basic.NET	-	+	+	+
Delphi/Object Pascal	+	-/+	+	-
Perl	+/-	+	-/+	+

Примечание:
+ – указанная возможность присутствует;
- – указанная возможность отсутствует;
-/+ – возможность поддерживается очень ограниченно;
+/- – возможность поддерживается не полностью.

Выводы. Проведен общий анализ современных ЯП. Выполнено исследование особенностей типизации наиболее популярных (рейтинговых) ЯП для правильного выбора необходимого программного обеспечения при решении широкого класса задач. Результаты исследований показали низкую заинтересованность разработчиков программного обеспечения в нетипизированных языках.

ЛИТЕРАТУРА

1. TIOBE Index for February 2017. [Электронный ресурс] – Режим доступа: https://www.tiobe.com/tiobe-index/.
2. Жульковская И.И. К выбору стратегии преподавания информатики для инженерных специальностей университетов / Жульковская И.И., Жульковский О.А. // Зб. наукових праць ДДТУ (технічні науки). – Дніпродзержинськ: ДДТУ. – 2013. – №.1 (21). – С.171-176.
3. Пирс Б. Типы в языках программирования / Пирс Б.; перевод с англ. – М.: Издательство «Лямбда пресс»-«Добросвет», 2011. – 680с.
4. Груздев Д. Ликбез по типизации в языках программирования [Электронный ресурс] / Груздев Д. – Режим доступа: https://habrahabr.ru/post/161205/.

Поступила в редколлегию 29.03.2017.

УДК 519.688

ПИШНИЙ М.А., аспірант
МАРЧЕНКО О.О., магістр
КОСУХІНА О.С., к.ф.-м.н., доцент
ГУЛЄША О.М., к.пед.н., доцент

Дніпровський державний технічний університет, м. Кам'янське

ІНТЕЛЕКТУАЛЬНІ МЕТОДИ ОБРОБКИ ДАНИХ: МУРАШИНІ АЛГОРИТМИ

Вступ. В останні два десятиліття при вирішенні комбінаторно-логічних задач проектування, конструювання та штучного інтелекту, а також при оптимізації складних систем дослідники все частіше застосовують природні механізми пошуку оптимальних рішень. Багато методик, які стосуються м'яких обчислень, експлуатують ідею того, що множина відносно простих об'єктів, працюючих за цілком зрозумілими правилами, об'єднуючись, демонструють поведінку, що набагато перевищує за інтелектуальністю поведінку окремого індивідуума. Мурашині алгоритми – це перспективний метод оптимізації, що базується на моделюванні поведінки колонії мурах. Оригінальний алгоритм (Ant Colony Optimization) виник в процесі спостереження за тим, як живі мурахи виду Argentine Ant обстежують територію навколо мурашника, знаходять їжу і несуть її в мурашник, постійно оптимізуючи (скорочуючи) шлях, який проходить кожен з мурах. Ці дослідження проводилися в 1989-му році Госсом і в 1990-му – Денеборгом [1]. Перша

математична формалізація алгоритму запропонована в 1992-му році Марко Доріго [2]. Застосування мурашиних алгоритмів для різного роду задач, таких як задача комівояжера, задача календарного планування, транспортна задача, квадратична задача про призначення, задача оптимізації мережевих трафіків, задача розмальовки графа, розробка оптимальної структури мереж GPS в рамках створення високоточних геодезичних та знімальних технологій тощо, розглядаються і в роботах вітчизняних вчених, наприклад, Штовби С.Д., Рудого О.М. [3-4], Колеснікова К.В, Карапетяна А.Р., Кравченка О.В. та ін.

Колонія мурах може розглядатися як багатоагентна система, в якій кожний агент (мураха) функціонує автономно за дуже простими правилами. На противагу майже примітивній поведінці агентів поведінка всієї системи виходить напрочуд розумною.

Живі мурахи під час пошуків їжі ходять навколо мурашника випадковим чином по стежках, які не є фізичними доріжками, «протоптаними» поколіннями комах. Основна орієнтація у мурах відбувається за рахунок феромонів, до яких вони дуже чутливі і якими вони позначають все навколо. Стежки стають все більш помітними. Але іноді мурашки збиваються з шляху і тоді випадковим чином шукають місце, позначене феромонами. Це може призводити до знаходження коротшого шляху. У цей час відбувається процес випаровування феромону. Якби його не було, то початковий шлях завжди мав би найсильніший запах, і процес пошуку коротшого шляху не відбувався.

До найбільш вдалих вдосконалених алгоритмів відносять:

- Elitist Ant System. Вдосконалення полягає у введенні в алгоритм так званих «елітних мурах». Досвід показує, що проходячи ребра, які входять в короткі шляхи, мурахи з більшою ймовірністю будуть знаходити коротші шляхи. Ефективною стратегією є штучне збільшення рівня феромонів на найвдаліших маршрутах. Для цього на кожній ітерації алгоритму кожна з елітних мурах проходить шлях, який є найкоротшим із знайдених на даний момент. Експерименти показують, що, до певного рівня, збільшення числа елітних мурах є досить ефективним, дозволяючи значно скоротити число ітерацій алгоритму. Однак, якщо число елітних мурах занадто велике, то алгоритм досить швидко знаходить субоптимальне рішення і застряє в ньому. Як і інші змінні параметри, оптимальне число елітних мурах слід визначати дослідним шляхом;
- Ant-Q. Цей мурашиний алгоритм отримав свою назву за аналогією з методом машинного навчання Q-learning. В основі алгоритму лежить ідея про те, що мурашину систему можна інтерпретувати як систему навчання з підкріпленням. Ant-Q підсилює цю аналогію, запозичуючи багато ідей з Q-навчання. Алгоритм зберігає Q-таблицю, що зіставляє кожному з ребер величину, яка визначає «корисність» переходу по цьому ребру. Q-таблиця змінюється в процесі роботи алгоритму – відбувається навчання системи. Значення корисності переходу по ребру обчислюється, виходячи із значень корисності переходу по наступних ребрах в результаті попереднього визначення можливих наступних станів. Після кожної ітерації корисності оновлюються, виходячи з довжин шляхів, до складу яких було включено відповідні ребра;
- Ant Colony System. Для підвищення ефективності в порівнянні з класичним алгоритмом введено три основних зміни: по-перше, рівень феромонів на ребрах оновлюється не лише в кінці чергової ітерації, але і при кожному переході мурах з вузла у вузол; по-друге, наприкінці ітерації рівень феромонів підвищується тільки на найкоротшому із знайдених шляхів; по-третє, алгоритм використовує змінене правило переходу: або, з певною часткою ймовірності, мураха безумовно вибирає краще ребро у відповідності до довжини і рівня феромонів, або робить вибір так само, як і в класичному алгоритмі;
- Max-min Ant System. У цьому мурашиному алгоритмі підвищення концентрації феромонів відбувається тільки на кращих шляхах з тих, що пройшли мурахи. Така велика увага до локальних оптимумів компенсується введенням обмежень на максимальну і мінімальну концентрації феромонів на ребрах, які вкрай ефективно захищають алгоритм від передчасної збіжності до субоптимальних рішень. На етапі ініціалізації концентрація феромонів на всіх ребрах встановлюється рівною максимальній. Після кож-

ної ітерації алгоритму тільки одна мураха залишає за собою слід: або найбільш успішна на даній ітерації, або, аналогічно алгоритму Elitist Ant System, елітна. Цим досягається, з одного боку, більш ретельне дослідження області пошуку, з іншого – його прискорення.

Постановка задачі. У роботі представлена алгоритмічна реалізація мурашиного алгоритму для задачі комівояжера, та її програмне втілення за допомогою мови програмування C#.

Задача комівояжера формулюється як задача пошуку мінімального за вартістю замкнутого маршруту за всіма вершинами без повторень на повному зваженому графі з n вершинами. Змістовно вершини графа є місцями, які повинен відвідати комівояжер, а вага ребер відображає відстань (довжину) або вартість проїзду.

Розглянемо більш формально один з найпростіших мурашиних алгоритмів. Лабіринт задається графом (вершини і ребра). Вважаємо, що мурахи шукають оптимальний шлях (найкоротший) між двома вершинами (мурашник і їжа), поки (не виконані умови виходу):

1 – створюємо мурах. Початкові точки, куди поміщаються мурахи, залежать від обмежень задачі. У найпростіших випадках ми можемо їх всіх помістити в одну точку або випадково розподілити по площині графа. На цьому ж етапі кожне ребро графа позначається невеликим позитивним числом, що характеризує запах феромону. Це потрібно для того, щоб на наступному кроці у нас не було нульових ймовірностей;

2 – мурахи шукають рішення. Визначаємо ймовірність переходу з вершини i у вершину j за формулою

$$P_{ij}(t) = \frac{\left[\tau_{ij}(t)\right]^{\alpha}\left[d_{ij}\right]^{-\beta}}{\Sigma_{l \in connected\_nodes}\left[\tau_{ij}(t)\right]^{\alpha}\left[d_{ij}\right]^{-\beta}}, \quad (1)$$

де $\tau_{ij}(t)$ – рівень феромону; d_{ij} – евристична відстань; α, β – константи.

Якщо $\alpha = 0$, то найбільш вірогідним є вибір найближчого сусіда, і алгоритм стає «жадібним». У разі, коли $\beta = 0$, найбільш імовірний є вибір тільки на основі рівня феромону, що призводить до того, що мурахи залишаються на вже «протоптаних» шляхах. Як правило, використовується деяке компромісне значення цих величин, яке підбирається експериментально для кожної задачі. Застосування такого ймовірнісного правила забезпечує реалізацію однієї з основних складових самоорганізації – випадковості;

3 – оновлення рівня феромону відбувається за формулою

$$\tau_{ij}(t+1) = (1-p)\tau_{ij}(t) + \Sigma_{k \in used\_edge(i,j)} \frac{Q}{L_k(t)}, \quad (2)$$

де p – параметр, що задає інтенсивність випаровування; L_k – ціна поточного розв'язання для *k*-го мурахи; Q – характеризує порядок ціна оптимального рішення. Таким чином вираз $\frac{Q}{L_k(t)}$ визначає кількість феромону, яким мураха *k* відмітив ребро (i, j) [5].

Результати роботи. Для розв'язання задачі комівояжера розроблено програму в C# з використанням Win Forms, оскільки задачі, пов'язані з графами, незручно розглядати в консолі. Створена програма складається із декількох основних модулів.

Вихідні дані. Маємо граф з сімнадцятьма вершинами (випадкове значення), кожна з вершин повинна бути включеною в маршрут (відповідно до умов задачі комівояжера). Вершини розміщуються випадковим чином, так само як обираються початкові параметри системи.

За допомогою мурашиного алгоритму та блок-схеми, зображеної на рис.1, знаходимо оптимальний маршрут.

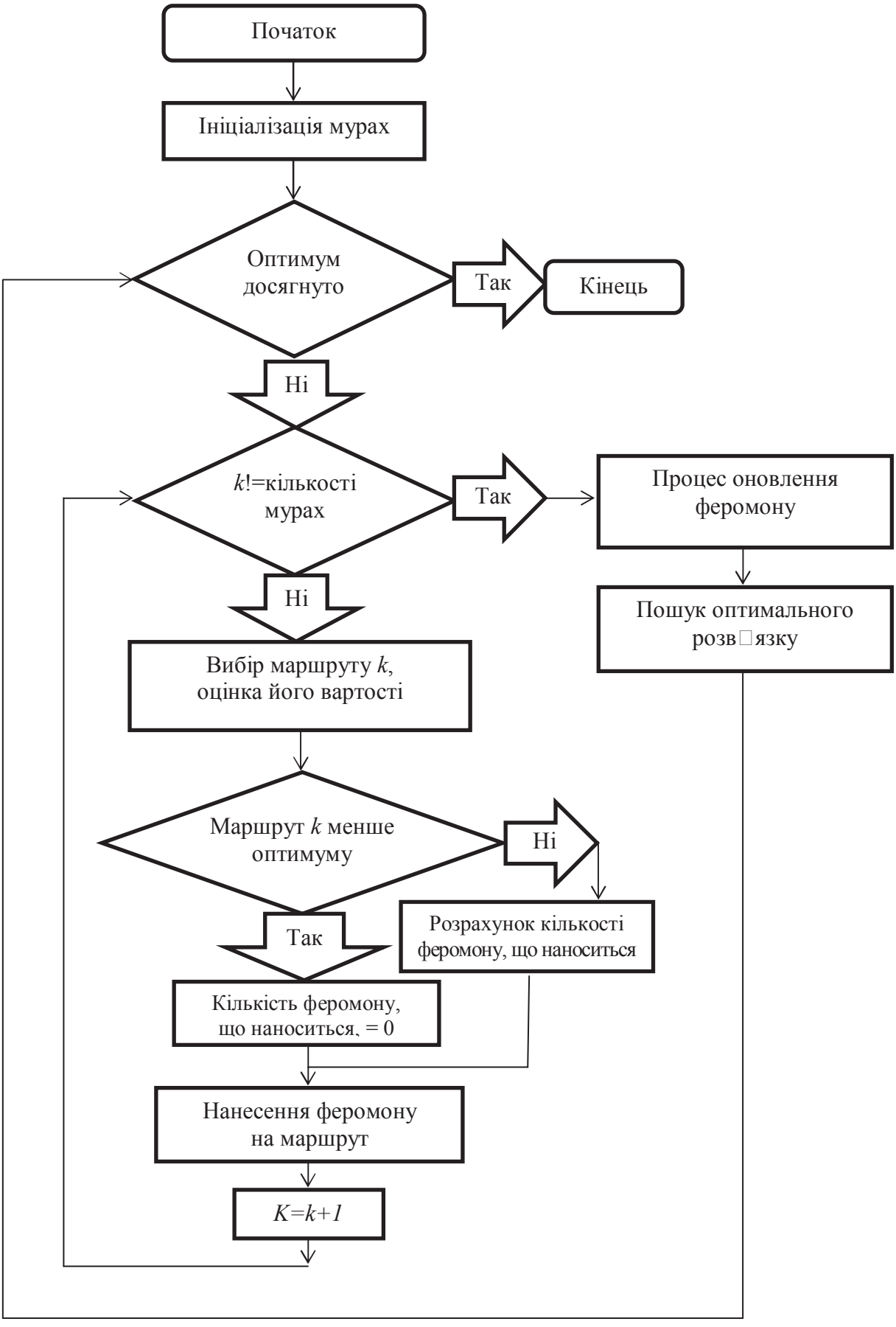

Рисунок 1 – Детальна блок-схема мурашиного алгоритму

На рис.2 зображено кращий, як для третього кроку алгоритму, маршрут та відображено довжину кращого шляху, яка складає приблизно 4.9 одиниць.

Рис.3 дає змогу спостерігати чотирьохсотий крок алгоритму, який відображає фінальну стадію задачі. З рис.3 видно, що довжина знайденого оптимального шляху зменшилася приблизно до 3.89 одиниць, що є на 8% (вісім відсотків) краще, ніж демонстрував рис.2. Проте, зважаючи на те, що різниця між рис.2 і 3 полягає у 311 кроках алгоритму, можна стверджувати, що для досягнення фінальної стадії оптимізації мурашиному алгоритму знадобився, можливо, надмірно великий час з огляду на порівняно невеликий відсоток оптимізації.

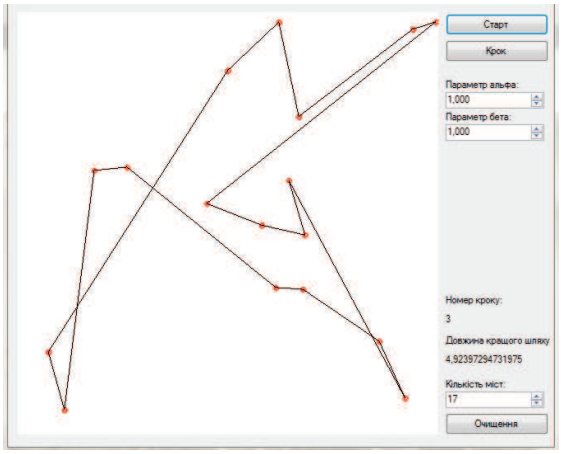

Рисунок 2 – Початок роботи програми

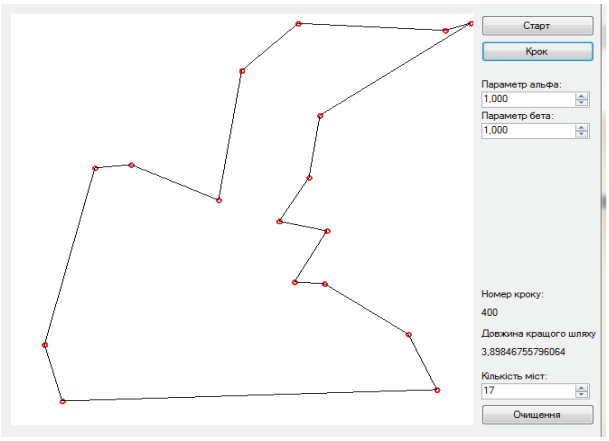

Рисунок 3 – Оптимальний маршрут знайдено

Звісно, кожний випадок буде мати свою швидкість збіжності, і для деяких з них оптимізація буде швидшою або повільнішою. Також не можна відкидати вірогідність того, що навіть після 12 тисяч ітерацій можливо отримати більш оптимальний маршрут. Проте, якщо казати про більш тривіальні випадки, наприклад, наведений вище, то можна стверджувати, що рис.2, 3 показують усереднену динаміку рішення задачі, а також, що основна оптимізація у класичному мурашиному алгоритмі, застосованому до графа з невеликою кількістю (приблизно до 50) вершин, відбувається протягом перших 100 ітерацій.

У роботі проаналізовано також вплив деяких параметрів на ефективність отриманих рішень. Підібрати оптимальні значення параметрів α та β можна експериментальним шляхом. Для довільно згенерованих 60 вершин графа в табл.1 відображено отримані значення.

Таблиця 1 – Залежність ефективності рішень від вибраних параметрів на прикладі задачі з 60 вершинами графа

Параметри	Найкраще рішення	Найгірше рішення	Середній результат
1	2	3	4
$\alpha = 1$ та $\beta = 1$			
100 ітерацій	7.04	16.83	11.94
1000 ітерацій	6.52	6.82	6.67
$\alpha = 2$ та $\beta = 1$			
100 ітерацій	15.68	6.68	11.18
1000 ітерацій	6.68	6.68	6.68

Продовження таблиці 1

1	2	3	4
α = 5 та β = 1			
100 ітерацій	8.04	7.04	7.54
1000 ітерацій	6.99	6.89	6.94
α = 0 та β = 1			
100 ітерацій	7.57	8.39	7.99
1000 ітерацій	7.39	7.86	7.63
α =1 та β = 2			
100 ітерацій	3.01	3.34	3.16
1000 ітерацій	2.67	2.82	2.74
α = 1 та β = 5			
100 ітерацій	2.63	3.49	2.66
1000 ітерацій	2.60	2.61	2.61
α = 1 та β = 0			
100 ітерацій	11.66	11.68	11.67
1000 ітерацій	9.56	10.05	9.86

Вихідні дані. Маємо граф з 60 вершинами (випадкове значення), кожна з вершин повинна бути включеною в маршрут (відповідно до умов задачі комівояжера). Вершини розміщуються випадковим чином, так само як обираються початкові параметри системи. На рис.4, 5 наведемо отримані результати для α=1, β=1.

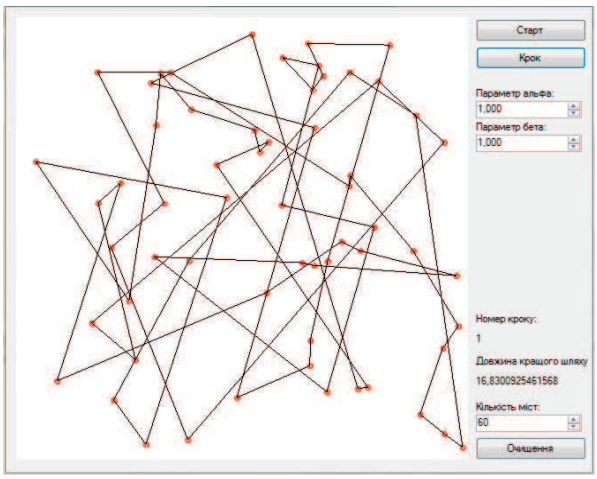

Рисунок 4 – Найгірше значення для 100 ітерацій при α=1, β=1

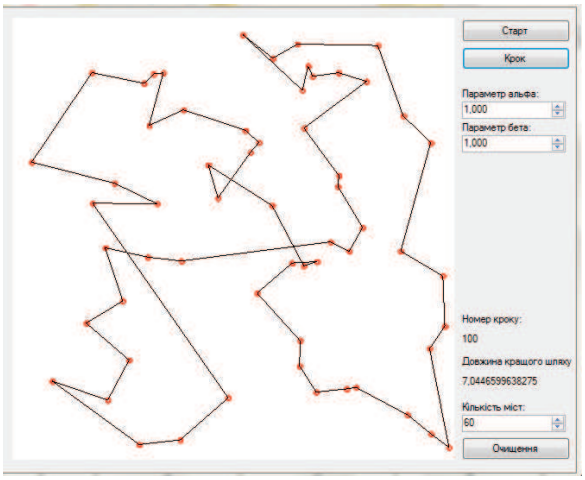

Рисунок 5 – Найкраще значення для 100 ітерацій при α=1, β=1

З наведених в табл.1 значень робимо висновок, що зміна значень параметрів α та β призводить до суттєвих змін в отриманих рішеннях. Найкращі результати (для 60 вершин графа) були отримані за наступних значень параметрів: (α=2, β=1) та (α=1, β=5). Збільшення значень параметра α (збільшується залежність отриманого рішення від концентрації феромону) призводить до збільшення діапазону рішень. Збільшення значень параметра β (збільшується залежність отриманого рішення від маршруту) призводить до зменшення діапазону рішень. Робота мурашиного алгоритму носить еврис-

тичний характер при більшому діапазоні рішень та „точний” характер – при невеликому діапазоні рішень.

Висновки. В роботі на прикладі задачі комівояжера розглянуто алгоритмічну реалізацію мурашиного алгоритму з впровадженням складових самоорганізації мурах, таких як випадковість, багатократність взаємодії, негативна і позитивна складові зв’язку. Програмна реалізація даного алгоритму здійснена за допомогою мови програмування C#.

ЛІТЕРАТУРА

1. Self-Organized Shortcuts in the Argentine Ant / Goss S., Aron S., Deneubourg J.L., Pasteels J.M. // Naturwissenshaften. – 1989. – № 76. – P.579-581.
2. Dorigo M. Swarm Intelligence, Ant Algorithms and Ant Colony Optimization / Dorigo M. // Reader for CEU Summer University Course «Complex System». – Budapest: Central European University. – 2001. – P.1-38.
3. Штовба С.Д. Муравьиные алгоритмы / С.Д.Штовба // Математика в приложениях. Exponenta Pro. – 2003. – №4. – С.70-75.
4. Штовба С.Д. Муравьиные алгоритмы оптимизации (на укр. языке) / С.Д.Штовба, О.М.Рудый // Вестник ВПИ. – 2004. – № 4. – С.62-69.
5. Шумейко А.А. Интеллектуальный анализ данных (Введение в Data Mining): учеб. пособ. / А.А.Шумейко, С.Л.Сотник. – Днепропетровск: издатель Белая Е.А. – 2012. – 210с.

Надійшла до редколегії 25.04.2017.

УДК 004.031.43

ЯШИНА К.В., к.т.н., доцент
ЯЛОВА К.М., к.т.н., доцент
СУГАЛЬ Є.О., студент

Дніпровський державний технічний університет, м. Кам’янське

ОГЛЯД ГНУЧКИХ МЕТОДОЛОГІЙ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Вступ. Agile – це сімейство методологій розробки програмних продуктів, що орієнтовані на використання ітеративної технології з динамічним формуванням та уточненням системи вимог на кожному ітераційному кроці. Дискретизація локальних процесів передбачає організацію їх взаємодії в середині робочих груп, створених з кваліфікованих фахівців різного профілю. У світі розробки програмного забезпечення (ПЗ) Agile методології прийнято називати «гнучкими» або «легкими».

Гнучкі методології розробки ПЗ є досить молодими. Маніфест Agile прийнято у лютому 2001 року. Цей маніфест формулює чотири основні ідеї методології:

- особистісний підхід (особистості та їх взаємодії є важливішими, ніж процеси та інструменти);
- забезпечення працездатності продукту, що розробляється (ПЗ, що ефективно працює є важливішим, ніж наявність детальної документації);
- забезпечення конструктивного діалогу між розробником та замовником (співпраця із замовником є важливішою, ніж жорсткі контрактні зобов’язання);
- гнучкість методів розробки з урахуванням вимог замовника, що динамічно оновлюються (реакція на зміни є важливішою, ніж чітке дотримання початкового плану) [1].

На сьогоднішній день гнучкі методології – основа сучасного світу розробки програмного забезпечення.