АВТОМАТИЗАЦІЯ ТА КОМП'ЮТЕРНО-ІНТЕГРОВАНІ ТЕХНОЛОГІЇ

УДК 004.89 DOI: 10.30977/BUL.2219-5548.2022.98.0.37

БІОНАТХНЕННІ МЕТОДИ ПЛАНУВАННЯ ШЛЯХУ МОБІЛЬНИХ РОБОТІВ

Гурко О. Г., Гурко В. О.
Харківський національний автомобільно-дорожній університет

Анотація. *Біонатхненні інтелектуальні алгоритми є різновидом методів оптимізації, що знайшли широке використання у мобільній робототехніці. У даній роботі представлено огляд біонатхненних алгоритмів, що використовуються для планування шляху автономних мобільних роботів у заздалегідь невідомій місцевості. Запропоновано класифікацію біонатхненних алгоритмів оптимізації. Проведено аналіз основних алгоритмів ройового інтелекту та наведено відповідні їм псевдокоди.*

Ключові слова*: мобільний робот, планування шляху, методи оптимізації, біонатхненні алгоритми, алгоритми ройового інтелекту.*

Вступ

Мобільна робототехніка є однією з галузей, що зазнає бурхливого розвитку. Автономні мобільні роботи (АМР) знаходять застосування у різних сферах людської діяльності та виконують різноманітні функції: від побутової та розважальної до рятувальної та військової. Щоб виконувати покладені на них завдання АМР повинні в реальному часі будувати мапу оточуючого середовища та знаходити своє положення у цьому середовищі (так зване завдання SLAM – Simultaneous Localization and Mapping), будувати шлях від початкової точки простору до кінцевої, оминаючи перешкоди [1] і забезпечувати рух за визначеним шляхом.

Складність завдання побудови шляху АМР залежить від типу робота та оточуючого середовища [2], що може бути статичним (з нерухомими перешкодами) та динамічним (що містить як рухомі, так і нерухомі перешкоди). Робот може рухатися у площині або у тривимірному просторі (повітряні або підводні роботи). Крім того, оточуюче АМР середовище можна розділити на детерміноване та невизначене відповідно до кількості наявної щодо нього інформації [3]. Складність завдання побудови шляху зростає за необхідності побудови оптимального шляху. Наприклад, АМР повинен знайти найкоротший шлях між кінцевими точками або витратити на рух мінімум часу або/та енергії.

Необхідність урахування наведених факторів визначає ефективність використання певних алгоритмів планування шляху АМР.

Нещодавно зросла популярність використання для планування рухів АМР метаевристичних алгоритмів, що побудовані на основі спостережень за поведінкою зграй тварин, птахів, риб або комах у боротьбі за їжу та партнерів, тому їх часто називають біонатхненними алгоритмами (БНА) або алгоритмами ройового інтелекту [4–6]. Справа в тому, що інтелект зграї можна трактувати як колективний розум групи відносно простих істот. Для досягнення мети ці істоти взаємодіють і демонструють складну поведінку, що дозволяє більш ефективно досягти мети, ніж окремим істотам. Такі методи також зазвичай мають вищу ефективність, ніж традиційні методи штучного інтелекту. Дана стаття представляє аналіз можливості та ефективності використання різноманітних БНА при плануванні шляху АМР.

Аналіз публікацій

Аналіз літератури [2, 3, 7–12], присвяченої методам планування шляху АМР, показав, що вказані методи можна класифікувати за різними ознаками (рис. 1), які можуть комбінуватися між собою.

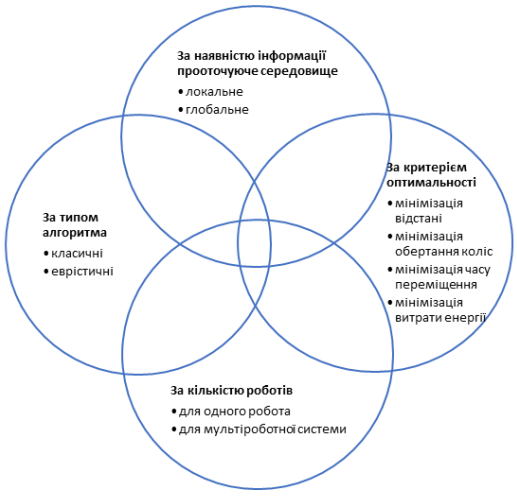

Рис. 1. Методи планування шляху АМР

Мета і постановка завдання

Як зазначено у попередньому розділі, великою групою методів, що використовуються для розв'язання оптимізаційних задач при побудові шляху АМР, є методи, що ґрунтуються на використанні так званих біонатхненних алгоритмів (БНА). На жаль, на даний час дуже мало публікацій українською мовою, що присвячені застосуванню БНА у робототехніці. Тому метою даної роботи є аналіз основних БНА, що застосовуються для планування шляху АМР.

Решту статті структуровано таким чином. Коротка класифікація БНА наведена у наступному розділі. Потім виконано аналітичний огляд та надано приклади використання для побудови шляху АМР алгоритмів, що імітують поведінку зграї тварин. Наприкінці статті наведені висновки за роботою та напрямок подальших досліджень.

Класифікація біонатхненних алгоритмів

Останнім часом зросла кількість досліджень з розробки та використання у робототехнічних додатках БНА оптимізації, що базуються на фізичних принципах, теорії еволюції та певній поведінці живих істот. Ці алгоритми використовуються для розпізнавання образів, планування шляху, керування рухом роботів тощо. Оскільки БНА перебувають у стадії розробки, то їх однозначної класифікації не існує. Станом на 2020 рік нараховувалося 257 БНА [13]. Багато авторів, наприклад [14, 15], виділяють два основні типи БНА: еволюційні та роєві алгоритми, тоді як у деяких роботах окремо виділяють алгоритми оптимізації переміщенням бактерій (Bacterial Foraging Optimization Algorithms) [16], алгоритми на основі поведінки рослин [17].

Для того щоб повніше класифікувати БНА, необхідно дати їм визначення. Одним із найвдаліших, на наш погляд, є визначення БНА як інтелектуальних обчислювальних методів, що імітують функцію та структуру організму, індивідуальну поведінку та поведінку рою, а також процес еволюції життя та суспільства [18]. Виходячи з наведеного визначення, БНА можна класифікувати таким чином (рис. 2): 1) еволюційні алгоритми; 2) алгоритми, що натхненні структурою організму; 3) алгоритми, що імітують поведінку окремих особин; 4) алгоритми ройового інтелекту; 5) комбіновані алгоритми.

Еволюційними є алгоритми, що натхненні складними еволюційними процесами: зміною генетичних ознак популяції між поколіннями, розселенням рослин або тварин, а також розвитком суспільства: генетичні алгоритми, еволюційна стратегія, еволюційне програмування, хімічне генетичне програмування, алгоритм заливного поля, алгоритми розповсюдження бур'янів, алгоритми біогеографічної оптимізації, що застосовують для вирішення завдань оптимізації, математичні алгоритми, які описують процес розповсюдження живих організмів на сусідніх островах [19], алгоритми, що імітують розвиток культури людства тощо.

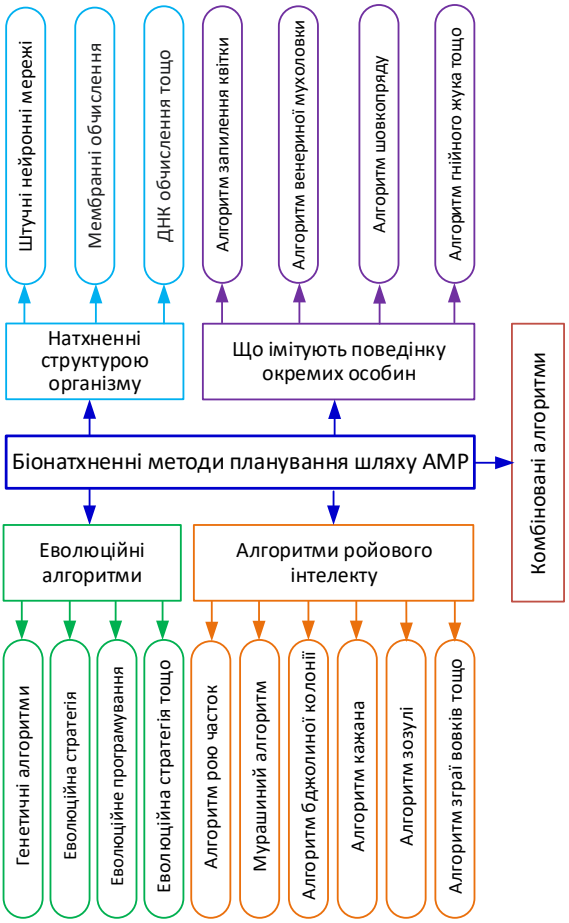

Рис. 2. Класифікація біонатхненних алгоритмів

Серед наведеного переліку напевно найвідомішим є генетичний алгоритм (ГА), що оснований на імітації процесу природного відбору. Основна ідея ГА полягає у виживанні найсильніших членів популяції хромосом, де кожна хромосома є можливим рішенням задачі оптимізації. Можливість використання хромосом оцінюється шляхом вимірювання їх функції придатності. ГА оновлює свою популяцію, посилюючи знайдені рішення (хромосоми), створюючи потомство на основі своїх найкращих хромосом за допомогою операторів схрещування (кросинговер) та мутації.

Одним із цікавих застосувань ГА при плануванні шляху робота є [20], де невеликі за розміром роботи грають у футбол. Тут за допомогою ГА планується шлях від поточної точки положення робота до воріт в обхід динамічних перешкод – інших гравців. При цьому алгоритм не потребує суттєвих обчислювальних ресурсів. Еволюційна стратегія, еволюційне програмування, хімічне генетичне програмування дещо нагадують ГА.

До групи алгоритмів, що натхненні *структурою живих організмів,* можна віднести штучні нейронні мережі, які імітують структуру мереж нервових клітин, та механізм передачі інформації в них; мембранні обчислення, що виникли в результаті вивчення біологічних клітин та клітинних мембран; ДНК обчислення, що ґрунтуються на сучасних знаннях про будову та функції молекули ДНК та операції, які виконуються в живих клітинах над молекулами ДНК за допомогою різних ферментів; штучні імунні системи, алгоритми яких натхненні принципами та процесами імунної системи хребетних, і т.ін.

До алгоритмів, що *імітують поведінку окремих особин,* можна віднести, наприклад такі, що основані на побудові шляху деякими комахами або тваринами, які будують свій шлях, відстежуючи хімічний слід (запах). Відомо, що багато комах виробляють феромони, які використовуються для обміну інформацією та пошуку шляху у навколишньому середовищі до їжі або партнера. Прикладом використання подібних алгоритмів у робототехніці є алгоритм шовкопряда, гнійного жука, алгоритм омара і т.ін. [21–23]. У якості заміни феромонів використовуються хімічні сполуки, світловий промінь тощо.

До цієї ж групи алгоритмів можна віднести алгоритми, основані на поведінці хижих рослин: алгоритм запилення квітки, алгоритми венериної мухоловки, всмоктування пузирчаткою [24] та ін.

Останнім часом все ширшу увагу привертають алгоритми, побудовані *на основі соціальної поведінки групи* (рою, зграї) організмів. Справа в тому, що інтелект групи деяких відносно простих живих істот (агентів) можна трактувати як колективний розум. Для досягнення мети ці істоти взаємодіють і демонструють складну та ефективну поведінку, стійку до втрати окремих індивідів [25, 26]. До алгоритмів цієї групи належать алгоритми рою часток, зграї вовків, мурашиний алгоритм, алгоритм зозулі, сірого вовка, алгоритм метелика і полум'я, алгоритм оптимізації переміщенням бактерій та багато інших.

Комбіновані алгоритми поєднують різні підходи, наприклад, нейронні мережі та алгоритми ройового інтелекту [27] або відстеження хімічного сліду та алгоритми ройового інтелекту [28].

Запропонована класифікація БНА, що використовуються при побудові шляху АМР, може розвиватися та доповнюватися з появою і розвитком нових алгоритмів. Так, існують алгоритми, що ґрунтуються на поведінці неживих природних об'єктів. Наприклад, алгоритм інтелектуальних крапель [29], що знаходить оптимальний або близький до оптимального шлях на кшталт того, як це роблять краплі води, коли вона тече руслом річки. При урахуванні подібних алгоритмів необхідно додати до наведеної класифікації ще одну групу, хоча деякі автори [16] відносять алгоритм інтелектуальних крапель до групи алгоритмів ройового інтелекту. Нижче надано більш детальний опис алгоритмів ройового інтелекту, що найбільш розповсюджені у мобільній робототехніці.

Алгоритм рою часток

Серед БНА, що належать до алгоритмів ройового інтелекту, одним із самих розповсюджених є алгоритм рою часток (англ. Particle Swarm Optimization – PSO). Цей алгоритм привабливий простотою ідеї та реалізації. На час написання даної статті у базі даних SCOPUS за ключовими словами «pso&robot& path&planning» знайдено 288 029 публікацій.

Основна концепція алгоритму рою часток полягає у такому [25]. Рій містить популяцію часток, які є кандидатами розв'язку задачі оптимізації. Стан кожної i-ї частки в просторі пошуку X, що є множиною всіх можливих рішень у певний час t, описується векторами положення $x_i(t)$ та швидкості $v_i(t)$ (рис. 3). Кожна частка пам'ятає своє найкраще положення $p_{besti}(t)$. Найкраще положення серед всіх часток рою зберігається у відповідному векторі глобальних найкращих положень $g_{best}(t)$.

Рух частки до нового поточного положення в момент часу $(t + 1)$ базується на трьох векторах (рис. 3): векторі поточної швидкості $v_i(t)$, векторі від поточного положення до поточного особистого найкращого положення $p_{besti}(t) - x_i(t)$ та векторі поточного глобального найкращого положення $g_{best}(t)$. Очікується, що нове положення $x_i(t + 1)$ частки буде кращим, ніж попереднє положення $x_i(t)$, оскі-

льки воно ґрунтується на попередньому рішенні про рух цієї частки, попередньому досвіді самої частки та попередньому досвіді всього рою часток.

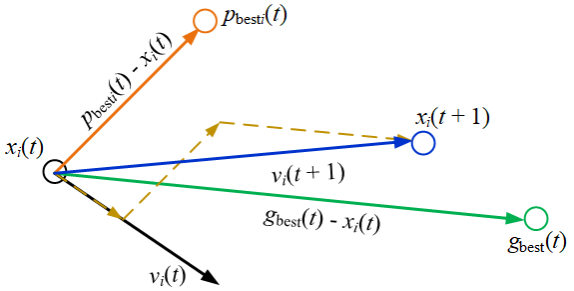

Рис. 3. Ілюстрація алгоритму рою часток [25]

З 1995 р., коли було введено поняття рою часток [30], алгоритм багато разів удосконалювався і на даний час класичною вважається така формула, що описує процес пошуку оптимального рішення таким методом:

$$v_i(t+1) = \omega v_i(t) + r_1 c_1 \left(p_{besti}(t) - x_i(t) \right) + \\ + r_2 c_2 \left(g_{best}(t) - x_i(t) \right), \quad (1)$$

$$x_i(t+1) = x_i(t) + v_i(t+1), \quad (2)$$

де ω – коефіцієнт інерції i для швидкості i-ї частки; c_1, c_2 – позитивні коефіцієнти прискорення; r_1, r_2 – рівномірно розподілені випадкові числа в діапазоні [0, 1]. Таким чином, другий доданок у (1) – це когнітивний компонент, а третій – соціальний компонент.

Псевдокод, що відповідає алгоритму рою часток, наведено на рис. 4.

Як уже зазначалося, на даний час існує велика кількість досліджень, що ґрунтуються на використанні методу рою часток для побудови шляху АМР. Ці роботи відрізняються або удосконаленням алгоритму, наприклад, підходами до вибору коефіцієнтів у (1) [31, 32], використанням різноманітних цільових функцій [33, 34] для розв'язання задачі оптимізації або комбінацією цього методу з іншими [35, 36].

Мурашиний алгоритм

Цей алгоритм був вперше запропонований у докторській дисертації М. Доріго у 1992 р. для моделювання поведінки колонії мурах під час пошуку їжі [37].

Починаючи пошук їжі, мураха рухається у випадковому напрямку, випускаючи шляхом феромон, який з часом випаровується. Чим ближче джерело їжі, тим менше часу витрачає мураха на дорогу туди і назад, і тим більша концентрація феромона. Інші мурахи вловлюють концентрації феромонів, залишених попередніми мурахами, і з більшою ймовірністю виберуть той шлях, де ця концентрація більша.

```
// Ініціалізація:
Ініціалізувати розмір популяції nPop;
Ініціалізувати максимальне число ітерацій maxIt
for i = 1 to nPop
    Ініціалізувати випадкові значення x_i з діапазону (x_min, x_max);
    Ініціалізувати випадкові значення v_i з діапазону (v_min, v_max);
    p_besti = x_i;
end for
Оцінка кожної частки;
Визначити глобальне найкраще положення g_best;
// Основний цикл:
while maxIt do
    for i = 1 to nPop do
        оновлення v_i(t + 1) за формулою (1);
        оновлення x_i(t + 1) за формулою (2);
        if v_i(t + 1) > v_max then v_i(t + 1) = v_max end if
        if v_i(t + 1) < v_min then v_i(t + 1) = v_min end if
        if x_i(t + 1) > x_max then x_i(t + 1) = x_max end if
        if x_i(t + 1) < x_min then x_i(t + 1) = x_min end if
        p_besti(t + 1) = p_besti(t);
        g_best(t + 1) = g_best(t);
        розрахунок цільової функції fitness(x_i(t + 1));
        if fitness(p_besti(t + 1)) < fitness(x_i(t + 1)) then
          оновити p_besti(t + 1);
        end if
        if fitness(g_best(t+1)) < fitness(p_besti(t+1)) then
        оновити g_best(t+1);
        end if
    end for
end while
return g_best
```

Рис. 4. Псевдокод для реалізації алгоритму рою часток

Таким чином, чим більше мурах проходить певним шляхом, тим більш привабливим він стає для їхніх родичів.

При реалізації алгоритму розглядається граф, вузли якого відображають початкову, кінцеву та проміжні точки шляху, а ребра – шляхи між цими точками. Щоб мурахи не вибирали шлях назад, кожна з них зберігає в пам'яті пройдені нею вузли у вигляді таблиці табу і не розглядає їх, як можливі для переходу.

Нехай k-та мураха знаходиться у вузлі i. Ймовірність її переходу з вузла i в j визначається виразом:

$$p_{ij}^k = \begin{cases} \dfrac{[\tau_{ij}^k(t)]^\alpha [\eta_{ij}^k(t)]^\beta}{\sum_{s \in N} [\tau_{is}^k(t)]^\alpha [\eta_{is}^k(t)]^\beta}, & j \in N \\ 0, & j \notin N \end{cases} \quad (3)$$

де τ_{ij} – концентрація феромона на шляху з вузла i у вузол j; η_{ij} – евристичне значення, що

вказує на відстань між вузлами i та j; α, β – вагові коефіцієнти, які встановлюють важливість інтенсивності феромона та евристичного значення відповідно; s – поточний вузол шляху; N – множина вузлів, які мураха може вибрати на наступному кроці (які не перебувають у таблиці табу).

З (3) видно, що чим більше значення коефіцієнта α, тим важливіша для вибору шляху концентрація феромона і мураха буде більш схильна вибирати шлях, яким пішла більшість мурах; чим більше значення β, тим мураха більш схильна вибирати найкоротший шлях. Евристичне значення η_{ij} визначається за такою формулою:

$$\eta_{ij}(t) = \frac{1}{d_{jg}} \qquad (4)$$

де d_{jg} – відстань від j-го вузла до кінцевої точки g:

$$d_{jg} = \sqrt{(x_i - x_g)^2 + (y_i - y_g)^2}\,. \qquad (5)$$

Як було зазначено вище, феромон поступово випаровується. Концентрація феромона на наступному кроці визначається формулами (6) та (7):

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \rho\Delta\tau_{ij}(t)\,, \qquad (6)$$

$$\Delta\tau_{ij}(t) = \sum_{k=1}^{m}\Delta\tau_{ij}^{k}(t)\,, \qquad (7)$$

де ρ – коефіцієнт випаровування феромона, $\rho \in (0, 1)$. Чим більше значення ρ, тим інтенсивніше випаровується феромон; $\Delta\tau_{ij}(t)$ – приріст феромона від i-го вузла до j-го вузла в момент часу t:

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} \dfrac{Q}{L_k}, & P(i,j) \in P \\ 0, & P(i,j) \notin P \end{cases}, \qquad (8)$$

де L_k – загальна довжину шляху k-ї мурахи; P – вузли на шляху; $P(i, j)$ – шлях між вузлами i та j; Q – концентрація феромона.

Формула (8) показує, що тільки мурахи, які досягли місця призначення, можуть оновлювати феромони, а концентрація феромонів обернено пропорційна довжині шляху.

Псевдокод, що відповідає наведеному мурашиному алгоритму, наданий на рис. 5. Після ініціалізації вихідних даних перевіряється умова припинення роботи алгоритму (рядок 2). Цією умовою може бути, наприклад, максимальне число поколінь maxGen або максимальне число поколінь без поліпшення рішення [38]. Рядки 2–13 охоплюють окреме покоління. У кожному поколінні всі мурахи в колонії переміщуються між окремими вузлами. Спочатку всі вузли вважаються невідвіданими (рядок 4). Алгоритм передбачає відвідування всіх вузлів один раз. У рядку 6 за виразом (3) обчислюється ймовірність, яка визначає шанс мурахи відвідати всі невідвідані вузли, що залишилися, а у рядку 7 відповідно до цієї ймовірності вибирається певний вузол. Мураха відвідує обраний вузол і позначає його як відвіданий.

Потім, якщо найкраще рішення, знайдене в даному поколінні, краще за попереднє найкраще рішення, воно зберігається (рядок 9). Наприкінці кожного покоління рівень феромона випаровуються (рядок 10) відповідно до формули (6), а в рядку 11 рівень феромона оновлюється. Потім починається наступне покоління, поки не буде виконано умову завершення. Наприкінці алгоритму в рядку 14 повертається найкраще знайдене рішення.

```
1: //Ініціалізація: початковий розмір популяції nPop;
                    // максимальне число поколінь maxGen;
                    // початковий рівень феромону та
                    // коефіцієнт випаровування ρ
2: while maxGen
3:     for k = 1 to nPop do
4:       встановити всі вузли як невідвідані
5:       while кількість невідвіданих вузлів > 0 do
6:         розрахувати за (3) ймовірність p  вибору мурахою вузла
7:         вибрати вузол у відповідності до ймовірності p
8:         відвідати обраний вузол
9:         зберегти найкраще рішення, якщо воно знайдено
10:        випаровування феромону за (6)
11:        оновити рівень феромону
12:    end for
13: end while
14: return найкраще рішення
```

Рис. 5. Псевдокод для реалізації мурашиного алгоритму

Мурашиний алгоритм широко використовується для пошуку шляху мобільних роботів. Зокрема, у [39] показано, що ефективність використання цього алгоритму при плануванні шляху АМР залежить від мапи середовища, оточуючого робота.

У [40] запропоновані такі удосконалення в мурашиному алгоритмі підвищення його збіжності під час пошуку шляху АМР. На початковому етапі використовується нерівномірна концентрація феромона, яка залежить від відстані між проміжним вузлом та лінією почa-

ток-кінець, що дозволяє уникнути випадкового вибору шляху на початковому етапі пошуку. Щоб прискорити швидкість збіжності алгоритму, евристичне значення η у (3) помножується на коефіцієнт, який поступово зменшує роль евристичної інформації у міру збільшення кількості ітерацій.

У роботі [41] представлений удосконалений мурашиний алгоритм, в якому для підвищення його збіжності до рівняння (3) введено стимулюючу ймовірність, що допомагає мурасі вибирати шлях до наступного вузла з великою кількістю виходів до цілі. Ця ймовірність залежить від кількості перешкод: чим їх менше, тим більше значення стимулюючої ймовірності і тим привабливіша ця ділянка шляху. Крім того, автори використовують модифіковане правило оновлення та випаровування феромонів. Згідно із результатами моделювання, запропонований у [41] покращений мурашиний алгоритм значно перевершує класичний варіант алгоритму з точки зору швидкості збіжності та довжини шляху.

У той же час у [42] підвищення ефективності побудови шляху АМР на складних картах запропоновано поєднання мурашиного алгоритму з елементами алгоритму A*. Також запропоновано новий вираз для визначення евристичного значення η,, метою якого є оптимізація гладкості шляху.

3. Алгоритм кажана

Алгоритм був запропонований 2010 р. [43] та базується на використанні кажанами для орієнтації у просторі принципу ехолокації. При реалізації алгоритму кажана робляться такі припущення [43, 44]:

1. Кажани використовують ехолокацію, щоб визначати відстань до об'єктів і розрізняти здобич від перешкод.

2. Під час пошуку здобичі кожен віртуальний кажан літає випадково зі швидкістю $\mathbf{v}_i$ у точці (рішенні) $\mathbf{x}_i$ з фіксованою частотою $f_{\min}$, змінною довжиною хвилі λ і гучністю A_0. Залежно від відстані до цілі кажан змінює довжину (або частоту) хвилі випромінюваних імпульсів і регулює швидкість імпульсів r.

3. По мірі наближення до цілі гучність змінюється від великого і позитивного значення A_0 до мінімального постійного значення $A_{\min}$.

Для більш ефективного застосування алгоритму кажана для задач оптимізації рекомендується зробити деякі додаткові припущення. Загалом припускаємо, що частота f еволюціонує на обмеженому інтервалі $[f_{\min}, f_{\max}]$. Це означає, що довжина хвилі λ також обмежена, оскільки f і λ пов'язані між собою співвідношенням: λf = const. З практичних міркувань також зручно, щоб найбільша довжина хвилі була порівняна з розміром простору пошуку. Якщо для простоти прийняти, що $f_{\min} = 0$, то $f \in [0, f_{\max}]$. Швидкість імпульсів належить діапазону $r \in [0, 1]$, де 0 означає повну відсутність імпульсів, а 1 означає максимальну швидкість випромінювання імпульсів. Псевдокод, що реалізує алгоритм кажана з урахуванням наведених вище припущень, наведений на рис. 6.

```
1:  // Ініціалізація: Розмір популяції: P
2:  // Максимальне число поколінь: maxG
3:  // Гучність: A
4:  // Швидкість імпульсів: r
5:  // Максимальна частота: f_max
6:  // Розмірність задачі: d
7:  // Цільова функція: ϕ(x), x = (x_1, …, x_d)^T
8:  // Випадкове число: θ ∈ U(0, 1)
9:  g ← 0
10: Ініціалізація x_i, v_i, i = (1, …, n)
11: Визначення частоти імпульсів f_i у x_i
12: Ініціалізація частоти імпульсів r_i та гучності A_i
13: while g < maxG do
14:   for i = 1 to P do
15:     Генерація нових рішень, регулюючи частоту,
16:     та оновлюючі швидкості та положення за формулами (9)-(11)
17:       if θ > r_i then
18:         s^best ← s^g    // вибір найкращого поточного рішення
19:         ls^best ← ls^g // вибір локального рішення навколо s^best
20:       end if
21:       Генерація нового рішення локальним випадковим блуканням
21:       if θ < A_i and ϕ(x_i) < ϕ(x*) then
22:         Прийняти нові рішення
23:         Збільшити r_i та зменшити A_i
24:       end if
25:   end for
26:   g ← g + 1
27: end while
28: Ранжування кажанів та визначення найкращого положення x*
29: return x*
```

Рис. 6. Псевдокод алгоритму кажана

Алгоритм розглядає популяцію з P кажанів (рядок 1). Кожен кажан, що представляє потенційне рішення задачі оптимізації, має своє положення x_i та швидкість v_i. Алгоритм ініціалізує ці змінні (рядок 10) випадковими значеннями в просторі пошуку. Потім для кожного окремого кажана обчислюються частота імпульсів, швидкість імпульсів та гучність (рядки 11, 12). Потім протягом поколінь популяція розвивається (рядки 13, 26) поки не буде досягнуто максимальної кількості поколінь max$G$ (рядок 27). Для кожного покоління g і кожного кажана (рядок 14) обчислюються нова частота, положення та швидкість (рядки 15,16) відповідно до таких рівнянь:

$$f_i^g = f_{\min}^g + \beta(f_{\max}^g - f_{\min}^g), \qquad (9)$$

$$\mathbf{v}_i^g = \mathbf{v}_i^{g-1} + [\mathbf{x}_i^{g-1} - \mathbf{x}^*] f_i^g, \qquad (10)$$

$$\mathbf{x}_i^g = \mathbf{x}_i^{g-1} + \mathbf{v}_i^g, \qquad (11)$$

де $\beta \in [0, 1]$ підпорядковується випадковому рівномірному розподілу, а x^* представляє поточне глобальне найкраще положення (рішення), яке отримано шляхом оцінки цільової функції для всіх кажанів і ранжування значень їх придатності. Верхній індекс $(.)^g$ позначає поточне покоління g.

Найкраще поточне рішення та локальне рішення навколо нього вибираються ймовірнісно відповідно до певних заданих критеріїв (рядки 17 – 20). Потім пошук посилюється локальним випадковим блуканням (рядок 21). Для цього локального пошуку, як тільки рішення вибрано серед поточних найкращих рішень, воно збурюється випадковим блуканням:

$$\mathbf{x}_{\text{new}} = \mathbf{x}_{\text{old}} + \varepsilon A^g, \qquad (12)$$

де ε – випадкове число з рівномірним розподілом на інтервалі [-1, 1]; A_g – середня гучність усіх кажанів у поколінні g.

Якщо нове отримане рішення краще, ніж попереднє найкраще, воно приймається новим найкращим. У цьому випадку алгоритм збільшує частоту імпульсів та зменшує гучність (рядки 21 – 24). Цей процес повторюється протягом заданої кількості поколінь. Загалом, коли кажан знаходить свою здобич (тобто коли знайдено нове найкраще рішення), гучність зменшується та швидкість випромінювання імпульсу збільшується. Для простоти зазвичай використовуються такі значення: $A_0 = 1$ і $A_{\min} = 0$, припускаючи, що останнє значення означає, що кажан знайшов здобич і тимчасово припинив видавати будь-які звуки. Правила еволюції для гучності та частоти імпульсів такі:

$$A_i^{g+1} = \alpha A_i^g, \qquad (13)$$

$$r_i^{g+1} = r_i^0[1 - e^{-\gamma g}], \qquad (14)$$

де α та γ є константами, причому для будь-яких $0 < \alpha < 1$ та $\gamma > 0$ маємо:

$$A_i^g \to 0, \; r_i^g \to r_i^0 \text{ при } g \to \infty.$$

Необхідно відзначити, що кожний кажан повинен мати різні значення гучності та швидкості випромінювання імпульсу, що може бути досягнуто шляхом рандомізації. Для цього можна прийняти початкову гучність $A \in (0, 2)$ у той час як початкова швидкість випромінювання імпульсів r_i^0 може мати будь-яке значення в інтервалі [0, 1]. Гучність і швидкість випромінювання буде оновлено лише за умови руху кажанів до оптимального рішення. Таким чином, налаштуванням параметрів, можна керувати поведінкою зграї кажанів для знаходження оптимального рішення.

Однією з основних переваг алгоритму кажана є швидкість його виконання, завдяки чому він знайшов широке використання для планування шляху АМР [44–46].

Алгоритм бджолиної колонії

Алгоритм бджолиної колонії (АБК) є ще одним ройовим алгоритмом оптимізації, що широко застосовується у робототехніці [47, 48]. Алгоритм натхненний тим, як медоносні бджоли знаходять нові найбагатші та найближчі джерела їжі, коли бджоли у колонії розділяються на три групи:

1) робочі бджоли, що збирають нектар з певного джерела їжі;

2) спостерігачі, які спостерігають за танцем бджіл-розвідників у вулику, щоб вибрати джерело їжі;

3) розвідники, які випадково шукають нові джерела їжі.

Спочатку кілька бджіл-розвідників вилітають із вулика у різних випадково обраних напрямках. У природі, як тільки виявлено джерело їжі, бджола-розвідник повертається у вулик і виконує танець у вигляді вісімки, в якому передається закодована інформація про відстань до знайденого джерела їжі, про напрямлення до цього джерела, про якість та кількість знайденого там нектара. За інтенсивністю танцю розвідників бджоли-спостерігачі вибирають найбагатші джерела їжі (які найбільше задовольняють заданому критерію) та вилітають у потрібному напрямку. При цьому кількість надісланих бджіл залежить від багатства знайденого джерела їжі. На місці збору їжі бджоли-спостерігачі перетворюються на робочих бджіл. Вони облітають це місце та оцінюють з точки зору кількості їжі. Далі можливі два варіанти:

1) бджола знаходить поблизу інше, більш привабливе джерело їжі і збирає її там, а про старе місце забуває. Потім вона повертається до вулика, де в танці повідомляє бджолам-спостерігачам координати нового джерела;

2) бджола не знаходить кращого місця та збирає нектар там, куди спочатку летіла. Потім вона розповідає іншим бджолам про те саме місце.

Після того, як джерело їжі вичерпано, бджола, яка його використовувала, стає бджолою-розвідником і знову шукає інші джерела їжі.

Наведений процес повторюється багаторазово. Через деякий час усі бджоли зосередяться на найкращих джерелах, а про менш привабливі забудуть.

Відповідно до наведеної процедури в АБК виділяються етапи.

На першому етапі (етапі ініціалізації) задаються: а) випадковим чином кількість *SN* джерел їжі, яка дорівнює кількості робочих бджіл, тобто у кожного джерела їжі є власна робоча бджола; б) кількість перевірок *lim*, після яких бджоли забувають про джерело їжі; в) умова завершення виконання алгоритму.

На другому етапі кожна робоча бджола, коли їх загальна кількість дорівнює половині джерел їжі, починає шукати нове джерело їжі в районі поточного джерела:

$$v_{ij} = x_{ij} + \varphi_{ij}(x_{ij} - x_{kj}), \qquad (15)$$

де φ_{ij} – дійсне випадкове число в діапазоні [-1, 1]; $k \in \{1, 2, \ldots, SN\}$ – індекс випадково обраного рішення, причому $k \neq i$; $j \in \{1, 2, \ldots, D\}$, D – розмірність задачі.

Як видно з (15), зі зменшенням значення $(x_{ij} - x_{kj})$ зменшується і збурення по положенню x_{ij}. Таким чином, коли рішення наближається до оптимального, довжина кроку пошуку нового рішення зменшується.

Після знаходження нового джерела v_{ij} його придатність порівнюється з x_{ij}. Якщо придатність нового джерела дорівнює або більше придатності старого, то нове джерело замінює в пам'яті старе. В інакшому випадку старе джерело їжі зберігається в пам'яті.

На наступному етапі бджола-спостерігач оцінює інформацію про продуктивність всіх джерел, знайдених робочими бджолами, та вибирає джерело їжі x_i за такою формулою:

$$p_i = \frac{f_i}{\sum_{n=1}^{SN} f_n}, \qquad (16)$$

де f_i – придатність рішення x_i, тобто значення цільової функції.

Очевидно, що чим більше значення цільової функції f_i, тим більша вірогідність вибору i-го джерела їжі.

Після того, як бджола-спостерігач обрала джерело їжі. Визначається нове сусіднє джерело за формулою (15) та обчислюється його придатність.

Якщо після закінчення заданої кількості ітерацій *lim* рішення на даній ділянці не покращується, то вона виключається з подальшого розгляду, а бджоли-дослідники цієї ділянки стають бджолами-розвідниками і відбір рішень здійснюється випадково за такою формулою:

$$x_i^j = x_{\min}^j + \text{rand}(0,1)(x_{\max}^j - x_{\min}^j). \qquad (17)$$

Відповідний псевдокод зображено на рис. 7 [49].

```
// Ініціалізація:
x_s: s-те рішення у популяції, s = 1, …, SN
v_s: отримане рішення, що відповідає s-му рішенню в популяції
maxIt: максимальна кількість ітерацій
limit_s: граничний лічильник рішень у популяції, s = 1, …, ps
ilimit: граничне значення, яке має бути перевищено
  для випуску бджіл-розвідників
r_s: рівномірно розподілене випадкове число в діапазоні [0, 1]
Згенерувати випадкову початкову популяцію за рівнянням (17)
iteration = 1;
while iteration < maxIt do
// фаза робочої бджоли
  s = 1;
  while s < SN do
    if r_s < mr then
      Побудувати новий розв'язок v_s, що відповідає розв'язку x_s
      за рівнянням (15)
        if f_{v_s} < f_{x_s} then limit_s = 0 and x_s ← v_s
        else limit_s = limit_s + 1
        end if
      s = s + 1
    end if
  end while
  // фаза бджоли-спостерігача
  l = 1
  while l < SN do
    Обрати рішення з популяції, використовуючи рівняння (16)
    Знайти нове рішення v_l що відповідає обраному рішенню x_l
    за рівнянням (15)
        if f_{v_l} < f_{x_l} then limit_l = 0 and x_l ← v_l
        else limit_l = limit_l + 1
        end if
      l = l + 1
  end while
  // фаза бджоли-розвідниці
  if mod(iteration, spp) = 0 then
    m = 1
    while m < SN do
      if limit_m > ilimit then
        Згенерувати нове рішення замість x_m за рівнянням (17)
      end if
      m = m + 1
    end while
  end if
  iteration = iteration + 1
end while
```

Рис. 7. Псевдокод для алгоритму бджолиної колонії

Алгоритм починається з генерації рівномірно розподіленої випадкової початкової популяції *SN* розв'язків за допомогою рівняння (17) в межах допустимого діапазону (рядок 8).

У фазі робочої бджоли (рядки 11–21) з поточного розв'язку формується новий розв'язок з імовірністю, що залежить від швидкості модифікації (*mr*), де $mr \in [0, 1]$ – керуючий параметр алгоритму, що обмежує зміну розмірності задачі. Більша швидкість модифікації приводить до суттєвої зміни поточного рішення, тоді як менша швидкість зміни приводить до незначної зміни. На основі попереднього рішення, що знаходиться в пам'яті, за допомогою рівняння (15) отримується новий розв'язок.

На етапі бджоли-спостерігача (рядки 22–31) використовується механізм відбору для ймовірнісного вибору кращого рішення (16), яке буде використано бджолою-спостерігачем. Як у фазі робочих бджіл, так і у фазі бджіл-спостерігачів, якщо бджоли не можуть покращити рішення, відповідне значення лічильника збільшується, інакше краще рішення, знайдене бджолами, замінює попереднє рішення в популяції. Фази робочої бджоли та бджоли-спостерігача продовжуються до тих пір, поки всі бджоли не закінчать процес пошуку.

У фазі бджоли-розвідниці (рядки 32–41) для керування процесом розвідки використовується ще один контрольний параметр – період розвідки (*spp*). У кожному циклі *spp*, якщо раніше розраховане значення лічильника перевищує задане граничне значення *ilimit*, бджола-розвідник вилітає для випадкового дослідження ще невідкритого простору рішень, тобто відбувається очищення пам'яті. Наведені три фази продовжуються до тих пір, поки не буде перевищена задана кількість ітерацій (*maxIt*).

Алгоритм зозулі

Алгоритм зозулі (англ. Cuckoo search algorithm) запропонований у 2009 р. [50] і завдяки простоті та здатності розв'язувати як лінійні, так і нелінійні задачі, широко використовується для планування шляху АМР [51, 52, 53], а також для розв’язання багатьох інших задач оптимізації. Натхненням для роботи цього алгоритму послужила поведінка зозуль, коли вони підкидають свої яйця до чужих гнізд. Зазвичай зозуля в чуже гніздо несе по одному яйцю і цю процедуру виконують всі зозулі. Однак, якщо це яйце виявляє господар гнізда, він або знищить усі яйця, або побудує нове гніздо. У результаті виживають тільки кілька яєць зозуль. Ці факти лягли в основу трьох основних правил алгоритму зозулі:

1) кожна зозуля відкладає одне яйце за один раз у випадково вибране гніздо;

2) найкращі гнізда з яйцями високої якості (придатними рішеннями) переходять на наступне покоління;

3) кількість доступних гнізд фіксована, а яйце зозулі може бути виявлене господарем гнізда з імовірністю $p_a \in [0, 1]$. Виявлені яйця виключають із подальшого розгляду.

Кожне яйце в гнізді являє собою рішення, а яйце зозулі – нове рішення. Метою оптимізації є знаходження нових потенційно кращих рішень (яєць зозулі), щоб замінити гірші рішення (яйця господарів) у гніздах.

Ґрунтуючись на цих трьох правилах, основні кроки алгоритму зозулі можуть бути узагальнені у вигляді псевдокоду, показаного на рис. 8.

1: // Ініціалізація:
2: Визначити цільову функцію $f(x)$, $x = (1, 2, ..., x_d)^T$
3: Створити початкову популяцію з n гнізд господарів x_i ($i = 1, 2, ..., n$)
4: **while** ($t < maxGen$) **or** (Критерій припинення) **do**
5: Випадковий вибір дох гнізд x та y
6: **for** $i = 1$ **to** n **do**
7: Випадковий політ зозулі до гнізда x_i^{t+1} за формулою (18)
8: Розрахунок значення цільової функції $f(x_i^{t+1})$
9: **if** $f(x_i^{t+1})$ краще ніж $f(y_i)$
10: Замінити y_i новим рішенням x_i^{t+1}
11: **end if**
12: **end for**
13: Покинути частину p_a найгірших гнізд та побудувати нові
14: Залишити найкращі рішення (гнізда з кращими рішеннями)
15: Ранжування рішень та знаходження поточного найкращого рішення
16: **end while**
17: **return** Найкраще рішення

Рис. 8. Псевдокод для алгоритму зозулі

Під час перельоту i-ї зозулі в інше місце (пошуку нового рішення) $x_i^{(t+1)}$ використовуються польоти Леві [54]:

$$x_i^{(t+1)} = x_i^{(t)} + \alpha \oplus levy(\lambda), \qquad (18)$$

де t – покоління, α – розмір кроку, який залежить від масштабу завдання (зазвичай $\alpha = 1$), $\oplus$ – покомпонентний добуток двох

векторів, λ – коефіцієнт, який залежить від постановки завдання.

Як виявилося [54], пошук нового рішення за допомогою польотів Леві ефективніший за традиційне випадкове блукання, що використовується в інших розглянутих вище ройових алгоритмах, оскільки в довгостроковій перспективі довжина його кроку набагато більша:

$$levy(\lambda) = t^{-\lambda}, \ (1 < \lambda \le 3). \qquad (19)$$

Це означає, що значна частина нових рішень буде сформована досить далеко від поточних найкращих рішень, що прискорює збіжність алгоритму.

Алгоритм сірих вовків

Алгоритм сірих вовків запропонований у 2014 р. [55] і швидко набирає популярності.

Даний алгоритм натхненний соціальною ієрархією сірих вовків у природі та їх поведінкою при полюванні на здобич. Відповідно до соціальної ієрархії сірі вовки поділяються на чотири рівні (α, β, δ та ω). α – домінуючий, він відповідає за прийняття рішень у тому числі під час полювання, зграя виконує його вказівки. β – це вовки, які допомагають альфам у прийнятті рішень. δ – вовки підпорядковуються вовкам α і β та відповідальні за виконання таких завдань, як пошук здобичі й полювання. ω – сірі вовки найнижчого рангу, вони відповідають за утримання вовчої зграї. Полювання сірого вовка поділяється на вистежування, переслідування та напад на здобич.

Вовки завжди полюють зграями і тримаються разом, виконуючи команди ватажка зграї, спілкуючись за допомогою виття, тон якого індивідуальний для кожної зграї. Така поведінка використовується для передачі інформації про місцезнаходження здобичі. Як тільки член зграї сірих вовків ідентифікує такий тон, α-, β- і δ-вовки негайно починають переслідування, щоб виявити місце виття і випадковим чином змінюють своє положення навколо неї. Кінцевим положенням буде положення всередині кола.

Таким чином під час виконання алгоритму сірих вовків на кожній ітерації оновлюються положення α-вовка, β-вовка та δ-вовка відповідно до таких формул [55]:

$$\vec{D} = \left| \vec{C} \cdot \vec{X}_p(t) - \vec{X}(t) \right|, \qquad (20)$$

$$\vec{X}(t+1) = \vec{X}_p(t) - \vec{A} \cdot \vec{D}, \qquad (21)$$

де t – поточна ітерація; $\vec{X}_p$ та $\vec{X}$ – відповідно вектори положення здобичі та вовка на ітерації t; $\vec{D}$ – відстань вовка до здобичі; $\vec{A}$ і $\vec{C}$ – вектори коефіцієнтів.

Таким чином, рівняння (20) описує відстань між сірим вовком та здобиччю, в той час як (21) є формулою оновлення положення сірого вовка. Вектори коефіцієнтів $\vec{A}$ і $\vec{C}$ розраховуються за цими рівняннями:

$$\vec{A} = 2\vec{a} \cdot \vec{r}_1 - \vec{a}, \qquad (22)$$

$$\vec{C} = 2 \cdot \vec{r}_2, \qquad (23)$$

де $\vec{r}_1$, $\vec{r}_2$ – випадкові вектори у діапазоні [0, 1], основна роль яких полягає у збільшенні випадковості руху сірого вовка; $\vec{a}$ – вектор коефіцієнтів збіжності, його компоненти лінійно зменшуються від 2 до 0 по мірі роботи алгоритму.

$$a = 2(1 - t/t_{\max}), \qquad (24)$$

де $t_{\max}$ – максимальна кількість ітерацій в алгоритмі.

При виконанні алгоритму на кожній ітерації вважається, що кращі положення мають α-, β- і δ-вовки. Однак ω-вовки також переміщуються у напрямку α, β і δ. Для коригування положення ω-вовків відносно інших вовків використовуються такі співвідношення:

$$\begin{cases} \vec{D}_\alpha = \left| \vec{C}_1 \cdot \vec{X}_\alpha - \vec{X} \right|, \\ \vec{D}_\beta = \left| \vec{C}_2 \cdot \vec{X}_\beta - \vec{X} \right|, \\ \vec{D}_\delta = \left| \vec{C}_2 \cdot \vec{X}_\delta - \vec{X} \right|, \end{cases} \qquad (25)$$

де $\vec{X}_\alpha$, $\vec{X}_\beta$ та $\vec{X}_\delta$ – вектори положення α-, β- і δ-вовків відповідно; $\vec{C}_1$, $\vec{C}_2$, $\vec{C}_3$ – випадково згенеровані вектори; $\vec{X}$ – вектор положення поточної особини. За рівнянням (25), таким чином, обчислюють відстані між положенням поточної особини та положенням особин α, β і δ. Вектори кінцевого положення поточної особини розраховуються таким чином:

$$\begin{cases} \vec{X}_1 = \vec{X}_\alpha - \vec{A}_1 \cdot (\vec{D}_\alpha), \\ \vec{X}_2 = \vec{X}_\beta - \vec{A}_2 \cdot (\vec{D}_\beta), \\ \vec{X}_3 = \vec{X}_\delta - \vec{A}_3 \cdot (\vec{D}_\delta), \end{cases} \quad (26)$$

$$\vec{X}(t+1) = \frac{\vec{X}_1 + \vec{X}_2 + \vec{X}_3}{3}, \quad (27)$$

де $\vec{A}_1$, $\vec{A}_2$, $\vec{A}_3$ – випадково згенеровані вектори.

Псевдокод алгоритму сірих вовків наведено на рис. 9 [55]. Переваги алгоритму сірих вовків полягають у тому, що він простий, потребує налаштування лише двох керуючих параметрів ($\vec{a}$ і $\vec{C}$; зміною параметра $\vec{A}$ налаштовується процес пошуку здобичі), гнучкий, надійний і легко реалізується. Тому даний алгоритм швидко привернув увагу дослідників при вирішенні різноманітних завдань, у тому числі й при плануванні шляху АМР [56–58].

1: Ініціалізація популяції сірого вовка X_i (i = 1, 2, …, n)
2: Ініціалізація a, A та C
3: Розрахунок значення цільової функції для кожного пошукового агента
4: X_α = кращий пошуковий агент
5: X_β = другий кращий пошуковий агент
6: X_δ = третій кращий пошуковий агент
7: **while** $t < t_{max}$
8: **for** кожного пошукового агента
9: Оновити положення поточного пошукового агента за формулою (27)
10: **end for**
11: Оновити a, A та C
12: Розрахунок значення цільової функції для кожного пошукового агента
13: Оновити X_α, X_β та X_δ
14: $t = t + 1$
15: **end while**
16: **return** X_α

Рис. 9. Псевдокод алгоритму сірих вовків

Однак алгоритм сірих вовків має й недоліки, наприклад, повільну швидкість збіжності та невисоку точність рішення. З цієї причини алгоритм постійно покращується [59, 60].

Висновки

Планування шляху мобільного робота є одним із найбільш важливих завдань мобільної робототехніки. Правильне планування шляху забезпечує безпеку робота і його оточення, ефективність виконання роботом покладених на нього завдань, дозволяє заощаджувати час та енергію, що витрачаються на виконання завдань, тощо. Тому постійно проводяться дослідження щодо використання нових та удосконалення існуючих методів оптимізації для їх використання при плануванні шляху мобільного робота.

Застосування для побудови шляху АМР класичних методів оптимізації обмежене їх суттєвими недоліками, такими як обчислювальна складність і тривалий час пошуку оптимального шляху. Для усунення проблем, що виникли під час використання класичних методів оптимізації, було розроблено евристичні, а потім і метаевристичні методи. Останні можна спрощено розглядати як алгоритми пошуку оптимальних або близьких до оптимальних рішень, доки не буде виконано якусь умову або досягнуто заданої кількості ітерацій. Серед метаевристичних методів все більшу популярність набувають біонатхненні методи оптимізації, в основі яких є еволюційні процеси в природі, а також поведінка живих організмів.

У цій роботі коротко розглянуто біонатхненні методи оптимізації, які застосовуються для побудови шляху мобільного робота. Усі методи поділено на чотири основні групи: 1) еволюційні алгоритми; 2) алгоритми, натхненні структурою організму; 3) алгоритми, що імітують поведінку окремих особин; 4) алгоритми ройового інтелекту; 5) комбіновані алгоритми.

Особливу увагу приділено алгоритмам ройового інтелекту, у яких відносно проста поведінка окремих агентів, що взаємодіють між собою та з навколишнім середовищем, що дає змогу досягти колективу (рою) агентів заданої мети. Детально розглянуто алгоритм рою часток, мурашиний алгоритм, алгоритми кажана, бджолиної колонії, зозулі та сірих вовків. Усі ці алгоритми застосовуються для панування шляху роботів. Наведені відповідні вказаним алгоритмам псевдокоди спрямовані на спрощення програмної реалізації алгоритмів.

Подальша робота пов’язана з розробкою програмного забезпечення та порівняльного аналізу ефективності розглянутих алгоритмів для планування шляху мобільного робота.

Література/ References

1. T. T. Mac, C. Copot, D. T. Tran and R. De Keyser. Heuristic approaches in robot path planning: A survey, *Robotics and Autonomous Systems*, 2016, vol. 86, pp. 13–28. https://doi.org/10.1016/j.robot.2016.08.001
2. K. Karur, N. Sharma, C. Dharmatti and J. E. Siegel. A survey of path planning algorithms for mobile robots, *Vehicles*, 2021, vol. 3, no 3, pp. 448–468. https://doi.org/10.3390/vehicles3030027
3. M. S. Abed, O. F. Lutfy and Q. Al-Doori. A review on path planning algorithms for mobile robots. *Engineering and Technology Journal*, 2021.

vol. 39, no 5A, pp. 804–820. https://doi.org/10.30684/etj.v39i5a.1941
4. J. Kallannan. *Bio-Inspired Algorithms in PID Controller Optimization*. CRC Press, 2018. https://doi.org/10.1201/9780429486579
5. J. Ni, L. Wu, X. Fan and S. X. Yang. Bioinspired intelligent algorithm and its applications for mobile robot control: A survey. *Computational Intelligence and Neuroscience*, 2016, pp. 1–16. https://doi.org/10.1155/2016/3810903
6. M. N. Ab Wahab, S. Nefti-Meziani and A. Atyabi. A comprehensive review of swarm optimization algorithms. *Plos One*, 2015, vol. 10, no 5, e0122827. https://doi.org/10.1371/journal.pone.0122827
7. N. Adzhar, Y. Yusof and M. A. Ahmad. A review on autonomous mobile robot path planning algorithms. *Advances in Science, Technology and Engineering Systems Journal*, 2020, vol. 5, no 3, pp. 236–240. https://doi.org/10.25046/aj050330
8. M. N. A. Wahab, S. Nefti-Meziani, A. Atyabi. A comparative review on mobile robot path planning: Classical or meta-heuristic methods? *Annual Reviews in Control*, 2020, vol. 50, pp. 233–252. https://doi.org/10.1016/j.arcontrol.2020.10.001
9. I. Skrjanc, G. Klancar, A. Zdesar, andS. Blazic. *Wheeled Mobile Robotics: From Fundamentals Towards Autonomous Systems*. Elsevier Science & Technology Books, 2017.
10. A. S. Matveev, A. V. Savkin, M. Hoy and C. Wang. Survey of algorithms for safe navigation of mobile robots in complex environments. *Safe Robot Navigation Among Moving and Steady Obstacles*. Elsevier, 2016, pp. 21–49. https://doi.org/10.1016/b978-0-12-803730-0.00003-2
11. L. Armesto, et al. Mobile robot obstacle avoidance based on quasi-holonomic smooth paths. *Advances in Autonomous Robotics*. Springer Berlin Heidelberg, 2012, pp. 244–255. https://doi.org/10.1007/978-3-642-32527-4_22
12. A. Muhammad, A. H. A. Mohammed, and I. H. Shanono. Path planning methods for mobile robots: A systematic and bibliometric review, *ELEKTRIKA-Journal of Electrical Engineering*, 2020, vol. 19, no 3, pp. 14–34. https://doi.org/10.11113/elektrika.v19n3.225
13. D. Molina, et al. Comprehensive taxonomies of nature- and bio-inspired optimization: inspiration versus algorithmic behavior, critical analysis recommendations. *Cognitive Computation*, 2020, vol. 12, no 5, pp. 897–939. https://doi.org/Я10.1007/s12559-020-09730-8
14. C. Guo, H. Tang, B. Niu and C. Boon Patrick Lee. A survey of bacterial foraging optimization. *Neurocomputing*, 2021, vol. 452, pp. 728-74. https://doi.org/10.1016/j.neucom.2020.06.142
15. S. Das, A. Biswas and S. Dasgupta, A. Abraham. Bacterial foraging optimization algorithm: Theoretical foundations, analysis, and applications. *Foundations of Computational Intelligence. Vol. 3*. Springer Berlin Heidelberg, 2009, pp. 23–55. https://doi.org/10.1007/978-3-642-01085-9_2
16. D. Rai and K. Tyagi. Bio-inspired optimization techniques. *ACM SIGSOFT Software Engineering Notes*, 2013, vol. 38, no 4, pp. 1–7. https://doi.org/10.1145/2492248.2492271
17. T. Sarkar, et al. Application of bio-inspired optimization algorithms in food processing. *Current Research in Food Science*, 2022, vol. 5, pp. 432–450. https://doi.org/10.1016/j.crfs.2022.02.006
18. J. Ni, L. Wu, X. Fan and S. X. Yang. Bioinspired intelligent algorithm and its applications for mobile robot control: A survey. *Computational Intelligence and Neuroscience*, 2016, vol. 2016, pp. 1–16. https://doi.org/10.1155/2016/3810903
19. D. Simon. Biogeography-based optimization. *IEEE Transactions on Evolutionary Computation*, 2008, vol. 12, no 6, pp. 702–713. https://doi.org/10.1109/tevc.2008.919004.
20. H. Burchardt, and R. Salomon. Implementation of path planning using genetic algorithms on mobile robots. *In Proceedings of the 2006 IEEE International Conference on Evolutionary Computation*, Vancouver, Canada, 16–21 July 2006; pp. 1831–1836. https://doi.org/10.1109/CEC.2006.1688529
21. R. Deveza, D. Thiel, A. Russell and A. Mackay-Sim. Odor sensing for robot guidance. *The International Journal of Robotics Research,* 1994, v. 13, no 3, pp.232-239. https://doi.org/10.1177/027836499401300305
22. X. Chen, J. Huang. Odor source localization algorithms on mobile robots: A review and future outlook. *Robotics and Autonomous Systems,* 2019, vol. 112, pp. 123-136. https://doi.org/10.1016/j.robot.2018.11.014.
23. L. Wang, S. Pang and J. Li. Olfactory-based navigation via model-based reinforcement learning and fuzzy inference methods. In *IEEE Transactions on Fuzzy Systems*, 2021, vol. 29, no. 10, pp. 3014-3027, https://doi.org/10.1109/TFUZZ.2020.3011741
24. R. Gowri and R. Rathipriya. Non-swarm plant intelligence algorithm: bladderworts suction (BWS) algorithm, *2018 International Conference on Circuits and Systems in Digital Enterprise Technology* (*ICCSDET*), 2018, pp. 1-7. https://doi.org/10.1109/ICCSDET.2018.8821225
25. A. Gurko and Y. Petrenko. A PSO-based controller tuning for a laser technical vision system. *2022 IEEE 3rd KhPI Week on Advanced Technology* (*KhPIWeek*), 2022, pp. 1-5, https://doi.org/10.1109/KhPIWeek57572.2022.9916393.
26. J. Kaliannan, A. Baskaran, N. Dey and A. S. Ashour. Bio-inspired algorithms in PID controller optimization. London: CRC Press, 2020.
27. C. Luo, G. E. Jan, Z. Chu and X. Li. Biologically inspired intelligence with applications on robot navigation. In *Artificial Intelligence - Emerging Trends and Applications*. London, United Kingdom: IntechOpen, 2018 [Online]. Available: https://www.intechopen.com/chapters/61601 doi: 10.5772/intechopen.75692

28. Y. Liu, X. Zhang, X. Guan and D. Delahaye. Potential odor intensity grid based UAV path planning algorithm with particle swarm optimization approach. *Mathematical Problems in Engineering*, 2016, vol. 2016, Article ID 7802798, 16 pages. https://doi.org/10.1155/2016/7802798
29. S. Salmanpour, H. Omranpour and H. Motameni. An intelligent water drops algorithm for solving robot path planning problem. *2013 IEEE 14th International Symposium on Computational Intelligence and Informatics* (*CINTI*), 2013, pp. 333-338, https://doi.org/10.1109/CINTI.2013.6705216
30. J. Kennedy and R. Eberhart. Particle swarm optimization. *Proceedings of ICNN'95 - International Conference on Neural Networks*, 1995, vol. 4, pp. 1942-1948. https://doi.org/10.1109/ICNN.1995.488968.
31. Wei Li and Gai-Yun Wang. Application of improved PSO in mobile robotic path planning. *2010 International Conference on Intelligent Computing and Integrated Systems*, 2010, pp. 45-48. https://doi.org/10.1109/iciss.2010.5655007.
32. B. Song, Z. Wang and L. Zou. An improved PSO algorithm for smooth path planning of mobile robots using continuous high-degree Bezier curve. *Applied Soft Computing*, 2021, vol. 100, p. 106960. https://doi.org/10.1016/j.asoc.2020.106960.
33. B. Deepak and D. Parhi. PSO based path planner of an autonomous mobile robot. *Open Computer Science*, 2012, vol. 2, no 2, pp. 152-168. https://doi.org/10.2478/s13537-012-0009-5.
34. M. K. Rath and B. B. V. L. Deepak. PSO based system architecture for path planning of mobile robot in dynamic environment. *2015 Global Conference on Communication Technologies* (*GCCT*), 2015, pp. 797-801, https://doi.org/10.1109/GCCT.2015.7342773.
35. F. H. Ajeil, I. K. Ibraheem, M. A. Sahib and A. J. Humaidi. Multi-objective path planning of an autonomous mobile robot using hybrid PSO-MFB optimization algorithm. *Applied Soft Computing*, 2020, vol. 89, p. 106076. https://doi.org/10.1016/j.asoc.2020.106076.
36. W. Jatmiko, K. Sekiyama and T. Fukuda. A pso-based mobile robot for odor source localization in dynamic advection-diffusion with obstacles environment: theory, simulation and measurement. In *IEEE Computational Intelligence Magazine*, 2007, vol. 2, no. 2, pp. 37-51. https://doi.org/10.1109/MCI.2007.353419.
37. M. Dorigo and T. Stützle. Ant colony optimization: Overview and recent advances. In *Handbook of Metaheuristics*. Boston, MA: Springer US, 2010, pp. 227–263. https://doi.org/10.1007/978-1-4419-1665-5_8.
38. P. Stodola, J. Mazal, M. Podhorec and O. Litvaj. Using the Ant Colony Optimization algorithm for the Capacitated Vehicle Routing Problem. *Proceedings of the 16th International Conference on Mechatronics - Mechatronika 2014*, 2014, pp. 503-510, https://doi.org/10.1109/MECHATRONIKA.2014.7018311
39. Yee Zi Cong and S. G. Ponnambalam. Mobile robot path planning using ant colony optimization. *2009 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, 2009, pp. 851-856, https://doi.org/10.1109/AIM.2009.5229903.
40. Sh. Li, G. Zhao and W. Yue. Research on path planning for mobile robot based on improved ant colony algorithm. *Journal of Physics: Conference Series,* 2021, vol. 2026, no. 1, p. 012049. https://doi.org/10.1088/1742-6596/2026/1/012049.
41. K. Akka and F. Khaber. Mobile robot path planning using an improved ant colony optimization. *International Journal of Advanced Robotic Systems*, 2018, vol. 15, no 3. https://doi.org/10.1177/1729881418774673.
42. X. Dai, S. Long, Z. Zhang and D. Gong. Mobile robot path planning based on ant colony algorithm with A* heuristic method. *Frontiers in neurorobotics*, 2019, vol. 13, article 15. https://doi.org/10.3389/fnbot.2019.00015.
43. X.-S. Yang. A new metaheuristic bat-inspired algorithm. In *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*. Springer Berlin Heidelberg, 2010, pp. 65–74. https://doi.org/10.1007/978-3-642-12538-6_6.
44. P. Suárez, A. Iglesias and A. Gálvez. Make robots be bats: specializing robotic swarms to the bat algorithm. *Swarm and Evolutionary Computation*, 2019, vol. 44, pp. 113-129. https://doi.org/10.1016/j.swevo.2018.01.005.
45. J. Perez, P. Et al. Trajectory optimization for an autonomous mobile robot using the bat algorithm. In *Fuzzy Logic in Intelligent System Design*. Cham: Springer International Publishing, 2017, pp. 232–241. https://doi.org/10.1007/978-3-319-67137-6_25.
46. F. H. Ajeil, I. K. Ibraheem, A. J. Humaidi and Z. H. Khan. A novel path planning algorithm for mobile robot in dynamic environments using modified bat swarm optimization. *The Journal of Engineering*, 2021, no 1, pp. 37–48. https://doi.org/10.1049/tje2.12009
47. M.A. Contreras-Cruz, V. Ayala-Ramirez and U. H. Hernandez-Belmonte. Mobile robot path planning using artificial bee colony and evolutionary programming. *Applied Soft Computing,* 2015, vol. 30, pp. 319-328. https://doi.org/10.1016/j.asoc.2015.01.067.
48. Y.Y. Cui, W. Hu and A. Rahmani. Fractional-order artificial bee colony algorithm with application in robot path planning. *European Journal of Operational Research*, 2022. https://doi.org/10.1016/j.ejor.2022.11.007
49. C. B. Kalayci, O. Ertenlice, H. Akyer and H. Aygoren. An artificial bee colony algorithm with feasibility enforcement and infeasibility toleration procedures for cardinality constrained portfolio optimization, Expert Systems With Applications, 2017, vol. 85, pp. 61–75. https://doi.org/10.1016/j.eswa.2017.05.018
50. X.-S. Yang and Suash Deb. Cuckoo Search via Lévy flights. *2009 World Congress on Nature & Biologically Inspired Computing (NaBIC)*, 2009, pp. 210-

214. https://doi.org/10.1109/NABIC.2009.5393690.
51. P. K. Mohanty and D. R. Parhi. Optimal path planning for a mobile robot using cuckoo search algorithm, *Journal of Experimental & Theoretical Artificial Intelligence*, 2014, vol. 28, no 1-2, pp. 35–52. https://doi.org/10.1080/0952813x.2014.971442.
52. P. K. Mohanty and D. R. Parhi. Cuckoo search algorithm for the mobile robot navigation. In *Swarm, Evolutionary, and Memetic Computing*. Cham: Springer International Publishing, 2013, pp. 527–536. https://doi.org/10.1007/978-3-319-03753-0_47
53. K. Sharma, S. Singh and R. Doriya. Optimized cuckoo search algorithm using tournament selection function for robot path planning. *International Journal of Advanced Robotic Systems*, 2021, vol. 18, no 3, p. 172988142199613. https://doi.org/10.1177/1729881421996136.
54. X. S. Yang and S. Deb. Engineering optimisation by cuckoo search. *International Journal of Mathematical Modelling and Numerical Optimisation*, 2010, vol. 1, no 4, pp. 330. https://doi.org/10.1504/ijmmno.2010.035430.
55. S. Mirjalili, S. M. Mirjalili and A. Lewis. Grey wolf optimizer. *Advances in Engineering Software*, 2014, vol. 69, pp. 46–61. https://doi.org/10.1016/j.advengsoft.2013.12.007.
56. A. Mallikarjuna Rao, K. Ramji and T. Naveen Kumar. Intelligent navigation of mobile robot using grey wolf colony optimization. *Materials Today: Proceedings*, 2018, vol. 5, no 9, pp. 19116–19125. https://doi.org/10.1016/j.matpr.2018.06.265
57. M. Petrović, A. Jokić, Z. Miljković and Z. Kulesza. Multi-objective scheduling of single mobile robot based on grey wolf optimization algorithm. *SSRN Electronic Journal*, 2022. 29 p. [Online]. Available: https://doi.org/10.2139/ssrn.4058009
58. L. Doğan and U. Yüzgeç. Robot path planning using gray wolf optimizer. *International Conference on Advanced Technologies, Computer Engineering and Science* (*ICATCES'18*). [Online]. Available: http://indexive.com/uploads/papers/icatces2018-17.pdf
59. E. Malayjerdi, M. Yaghoobi and M. Kardan. Mobile robot navigation based on fuzzy cognitive map optimized with grey wolf optimization algorithm used in augmented reality. In *2017 5th RSI International Conference on Robotics and Mechatronics (ICRoM)*, Tehran, Iran, 25–27 Oct. 2017. IEEE. [Online]. Available: https://doi.org/10.1109/icrom.2017.8466169
60. R. Kumar, L. Singh and R. Tiwari. Path planning for the autonomous robots using modified grey wolf optimization approach. *Journal of Intelligent & Fuzzy Systems*, 2021, vol. 40, no 5, pp. 9453–9470 https://doi.org/10.3233/jifs-201926.

Гурко Олександр Геннадійович, д.т.н., проф., зав. каф. автоматизації та комп'ютерно-інтегрованих технологій, gurko@khadi.kharkov.ua. ORCID: 0000-0001-9905-8584.
Гурко Володимир Олександрович, студент магістратури, каф. автоматизації та комп'ютерно-інтегрованих технологій, pzkpmfv@gmail.com.
Харківський національний автомобільно-дорожній університет, 61002, Україна, м. Харків, вул. Ярослава Мудрого, 25.

Bio-inspired methods for planning the path of mobile robots

Abstract. Problem. *The issue of path planning for a mobile robot is one of the most important ones of mobile robotics. Proper path planning ensures the safety of the robot and its environment, the efficiency of the tasks carried out by a robot, saves time and energy consumption for these tasks, etc. Therefore, research is constantly conducted on the implementation of new and improving existing optimization methods for the path planning for a mobile robot. The utilization of classical optimization methods is limited by their significant drawbacks, such as computational complexity and long time for searching the optimal path. To eliminate these issues, heuristic and then metaheuristic methods have been developed. Among metaheuristic methods, bio-inspired optimization methods, which are based on evolutionary processes in nature, as well as the behaviour of living organisms, are becoming increasingly popular.* ***Goal****. This paper aims to analyse the most popular bio-inspired algorithms used for mobile robot path planning.* ***Methodology****. The paper briefly reviews the bio-inspired optimization methods that are applicable to the path planning of a mobile robot. Particular emphasis is given to swarm intelligence algorithms, in which the relatively simple behaviour of individual agents interacting with each other and with the environment allows a swarm of these agents to achieve a given goal.* ***Results****. A classification of bio-inspired optimization methods used for mobile robot path planning is proposed. Pseudocodes for swarm optimization algorithms that are most frequently applied in mobile robotics are presented.* ***Originality****. This paper is one of the first in Ukraine to offer a comprehensive overview of bio-inspired methods of optimization used for mobile robot path planning.* ***Practical value.*** *The implementation of the considered algorithms in mobile robot control systems will improve the efficiency of robots in performing their assigned tasks. The given pseudocodes will simplify the development of software to implement the mentioned algorithms.*
Key words: *mobile robot, path planning, optimization methods, bio-inspired algorithms, swarm intelligence algorithms.*

Gurko Alexander, professor, Doct. of Science, Automation and Computer-Integrated Technologies Department, gurko@khadi.kharkov.ua. ORCID: 0000-0001-9905-8584.
Hurko Volodymyr, master student, pzkpmfv@gmail.com.
Kharkiv National Automobile and Highway University, 25, Yaroslava Mudrogo str., Kharkiv, 61002, Ukraine.