Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

journal homepage: www.elsevier.com/locate/swevo

oBABC: A one-dimensional binary artificial bee colony algorithm for binary optimization

Fangfang Zhu [a,b], Zhenhao Shuai [c], Yuer Lu [c], Honghong Su [d], Rongwen Yu [c], Xiang Li [a], Qi Zhao [e,*], Jianwei Shuai [c,f,*]

[a] Department of Physics, and Fujian Provincial Key Laboratory for Soft Functional Materials Research, Xiamen University, Xiamen, 361005, China
[b] National Institute for Data Science in Health and Medicine, and State Key Laboratory of Cellular Stress Biology, Innovation Center for Cell Signaling Network, Xiamen University, Xiamen, 361005, China
[c] Wenzhou Institute, University of Chinese Academy of Sciences, Wenzhou, 325001, China
[d] Yangtze Delta Region Institute of Tsinghua University, Jiaxing, 314006, China
[e] School of Computer Science and Software Engineering, University of Science and Technology Liaoning, Anshan, 114051, China
[f] Oujiang Laboratory (Zhejiang Lab for Regenerative Medicine, Vision and Brain Health), Wenzhou, 325001, China

A R T I C L E I N F O

Keywords:
Artificial bee colony
Binary optimization
Swarm intelligence
UFLP
Max-Cut

A B S T R A C T

Artificial bee colony (ABC) algorithm is a widely utilized swarm intelligence (SI) algorithm for addressing continuous optimization problems. However, most binary variants of ABC (BABC) algorithms may suffer from issues such as invalid searches and high complexity when applied to binary problems. To address these challenges, we first establish a set of criteria for developing a BABC algorithm. Following these criteria, we propose a novel BABC algorithm, denoted as oBABC, which not only adheres to the defined criteria but also successfully inherits the advantages of original ABC algorithm. To evaluate the performance of oBABC and verify its effectiveness, experiments are conducted on two typical binary problems: uncapacitated facility location problem (UFLP) and maximum cut problem (Max-Cut). The experimental results reveal the following findings: 1) The validity of the criteria and the accuracy of the theoretical analysis are confirmed. oBABC exhibits high search efficiency with an invalid learning rate (*ILR*) of 0 %, while the *ILR*s of other BABC algorithms almost exceeds 20 %. 2) In terms of search efficiency and capability, oBABC exhibits a significant improvement in search efficiency and consistently ranks at the top in terms of optimization capability. These results suggest that oBABC may be a highly efficient and effective tool for solving binary problems.

1. Introduction

Optimization problems are ubiquitous in engineering fields and arise in a multitude of tasks such as hyperparametric search [1,2], filter design [3], resource allocation [4,5], classification problem [6], clustering problem [7], and biological signal processing [8,9], among others. Due to their vast state space, many of these problems are considered NP-hard (Non-deterministic Polynomial-time hard), indicating no existing algorithms can efficiently solve them within polynomial time. SI algorithms emerge as a promising alternative for addressing such challenges [10], which including particle swarm optimization (PSO) [11,12], ant colony optimization (ACO) [13], artificial bee colony (ABC) [14,15], firefly algorithm (FA) [16], gray wolf optimization (GWO) [17], bat algorithm (Bat) [18], battle royale optimization (BRO) [19], and many others. Among them, ABC stands out as a prominent member of SI algorithms, successfully addressing optimization problems with various characteristics [20]. It models the collective intelligence of a honey bee colony during a foraging task, with the objective of maximizing nectar collection from food sources. Not only does it possess a simple structure and easy adaptability to different problems, but it also exhibits several outstanding features. First, the solution search formula updates only one dimension value at a time, enabling a finer-grained search around the current solution and enhancing the algorithm's exploitation performance. Second, employee bees recruit onlooker bees to profitable sources through positive feedback, which facilitates the exploitation of better suboptimal solutions and accelerates convergence speed. Finally, over-exploited solutions are reset by scouting bees through negative feedback, allowing the ABC

* Corresponding author.
E-mail addresses: zhaoqi@lnu.edu.cn (Q. Zhao), shuaijw@wiucas.ac.cn (J. Shuai).

https://doi.org/10.1016/j.swevo.2024.101567
Received 23 September 2023; Received in revised form 11 March 2024; Accepted 31 March 2024

algorithm to escape local optimal solutions and introduce fluctuation. In summary, the ABC algorithm is an excellent optimization algorithm that balances exploitation and exploration capabilities through positive feedback, negative feedback, and fluctuation properties.

These advantages have positioned the ABC algorithm as a new research hotspot [21–24]. All of them focus on improving the search efficiency and convergence characteristics of ABC algorithms. Some variants introduce multi-strategies to guide search by using multi-elite guidance without losing population diversity [25,26]. Some introduce the strategy of multi-dimensional updates at different stages of ABC algorithm to accelerate the convergence of the algorithm and enhance the exploration ability of the algorithm [26,27]. Some hybridize the search engine of ABC algorithm with that of other SI algorithms, combining the advantages of each algorithm to achieve better performance [28]. However, most of the research is devoted to improving the performance of ABC algorithm in continuous optimization problems, and only a few are devoted to exploring the application of ABC algorithm in binary problems.

1.1. Motivations

Binary optimization problems differ from continuous optimization problems as the solutions are belong to a binary space where variables take values of either 0 or 1. Most SI algorithms are originally proposed for continuous domain problems, where the solution search formula yields real-valued solutions. Therefore, when applying SI algorithms to solve binary problems, it is necessary to address the mapping issue between the real domain and the binary domain. Most SI algorithms have corresponding binary versions, such as binary versions of PSO [29,30], binary DE [31], discrete FA [32], binary BRO [33], BABC [27,34-36], and others. To the best of our knowledge, there is a lack of well-defined design criteria for the development of binary versions of SI algorithms, particularly for algorithms like ABC that rely on one-dimensional updates. This characteristic brings new challenges to the development of binary versions. Therefore, the main aim of this work is to present comprehensive criteria for the development of binary versions of SI algorithms. Moreover, based on these criteria, we try to propose a novel binary version of the ABC algorithm.

1.2. Contribution

In this work, we define the key criteria for developing binary versions of SI algorithms and propose oBABC algorithm. The main contributions of this work are as follows:

i) We initially offer criteria for developing binary versions of SI algorithms, which encompass three aspects: (I) minimizing computational complexity, (II) maximizing space efficiency, and (III) employing appropriate bionic strategies. Criterion I emphasizes the importance of reducing the complexity of the search formula. Criterion II necessitates the design of a search formula that ensures exploration of new solution spaces. Criterion III calls for the incorporation of more intelligent strategies to enhance the likelihood of discovering optimal solutions within a limited number of epochs.
ii) The optimal solution search formula is derived for one-dimensional updated BABC algorithm, enabling exploration of previously unvisited rich sources. Any binary SI algorithm with one-dimensional update characteristic should adopt this formula, otherwise it will result in invalid search.
iii) Multiple interaction models for selecting the direction of movement have been developed in oBABC. A decision model is developed to decide whether to move towards or away from the neighborhood solution, which enhances the exploitation performance of oBABC.

1.3. Organization

The rest of this paper is structured as follows: Section 2 provides a comprehensive review of the relevant literature. Section 3 offers a detailed introduction to the basic ABC algorithm. Section 4 outlines the essential criteria for binarization and introduces oBABC algorithm. Experimental results and their discussion are presented in Section 5. The paper concludes with Section 6, summarizing the study’s findings and suggesting potential avenues for future research for both scholars and practitioners.

2. Literature review

There are two ways to develop binary versions of SI algorithms: mapping-based method and binary-operator method. Recently, several BABC algorithms have been introduced, which are discussed below.

2.1. Mapping-Driven binary variants

In the mapping-based approach, search agents navigate a continuous solution space, with candidate solutions being converted into binary form prior to objective function evaluation. This approach serves as a straightforward post-processing technique to binarize SI algorithms, exemplified by the discrete PSO (DPSO) [29] and binary differential evolution (BDE) [31]. The first binary version of the basic ABC algorithm, known as DABC, is based on mapping-based method, which has been successful in feature selection [37]. Additionally, various binary ABC algorithms utilizing transfer functions have been introduced, including the angle-modulated ABC (AMABC) [34] and ABCbin [38]. AMABC utilizes angle modulation to create a homomorphic mapping between continuous and binary spaces, enhancing optimization for numerical functions. Nonetheless, the adoption of transfer functions escalates computational complexity due to the extensive use of floating-point and even exponential operations. While ABCbin attempts to mitigate this complexity by implementing a mod-round-mod operation for mapping, it still incurs a higher computational load compared to methods based on binary operations.

2.2. Binary-operator-driven binary variants

In the binary-operator method, search agents navigate a binary-structured space directly, eliminating the need for mapping between disparate spaces. This method functions as a preprocessing strategy, characterized by its low computational complexity. Notable examples of the binary-operator method are binABC [35], bitABC [36] and PBABC [27], which incorporate logical operations such as ’and’, ’or’, and ’not’, along with their combinations. This approach replaces the arithmetic operations—’add’, ’subtract’, ’multiply’, and ’divide’—typical of the ABC algorithm’s continuous variants. By foregoing the mapping process, this method significantly reduces computational complexity, allowing computations to be executed directly within the binary space and maintaining the inherent binary characteristics of the solution. This ensures efficient processing without necessitating further transformations or conversions. Additional algorithms based on the binary-operator method, including DisABC [39], NBABC [40], ibinABC [41] iBABC [42] and GBABC [43], apply statistical theory to mutate multiple bits of the solution. This maintains the search updates strictly within the binary space and obviates the need for conversions between different numerical spaces. However, a significant challenge of this approach is the redesign of the search operator for binary contexts. Although insights can be drawn from the continuous versions, the distinct characteristics of binary spaces necessitate a cautious and deliberate approach in the design of binary-specific search operators.

2.3. Binary variants of ABC

While numerous BABC algorithm have been developed, they still face challenges that compromise their effectiveness on binary problems. A notable feature of the ABC algorithm is its one-dimensional update mechanism [14,20,44], wherein search agents adjust only one variable dimension at a time. This approach enhances the algorithm's granularity, enabling closer proximity searches without missing adjacent superior solutions. However, this feature significantly increases the likelihood of invalid searches across all BABC variants, such as DABC [37], binABC [35], bitABC [36], ABCbin [38], etc., that adopt this one-dimensional update strategy. We will delve into this issue in depth and propose relevant design criteria in the following sections. Moreover, the one-dimensional update characteristic of the ABC algorithm slows down convergence due to the minimal Euclidean distance covered per search step. This challenge can be mitigated by adopting multi-dimensional updates, as seen in algorithms like DisABC [39], NBABC [40], ibinABC [41] iBABC [42] and GBABC [43]. These approaches improve convergence speed but may compromise the algorithm's precision, leading to premature optimization. An alternative strategy involves simulating intelligent population behavior to more accurately direct search efforts. In response to these challenges, oBABC algorithm employs the binary-operator method alongside multiple interaction models for directional selection. This innovation holds promise for enhancing optimization across a variety of problems, including data analysis [45] and prediction [46,47], positioning oBABC as a pivotal tool for optimization challenges.

3. The basic abc algorithm

3.1. ABC algorithm

The ABC algorithm is mainly inspired by the foraging behavior of the honey bee colony. It is divided into four phases: initialization, employed bees, onlooker bees, and scout bees, with the last three phases corresponding to the duties of the bees during foraging.

Initialization: Assuming that the food sources, which are possible solutions for optimization problem, are distributed in a d-dimensional space. A food source or a solution can be expressed as Eq. (1). The initial solution can be achieved by Eq. (2).

$$x_i^t = \left(x_{i1}^t, x_{i2}^t, \cdots, x_{iD}^t\right) \in R^D \tag{1}$$

$$x_{ij} = x_{\cdot j}^{\min} + r_{ij} \times \left(x_{\cdot j}^{\max} - x_{\cdot j}^{\min}\right) \tag{2}$$

Where $i\epsilon\{1, 2, \cdots, N\}$, N is the number of employed bees and D is the dimensionality of the optimization problem. x_{ij}^t is the jth dimension variable of x_i at epoch t. $x_{\cdot j}^{min}$ and $x_{\cdot j}^{max}$ are the lower and upper bounds of x_{ij}, respectively. r_{ij} is a uniform random number in range of [0,1].

Employed bees' stage: The employed bees are responsible for exploring higher quality food source locations around current site. Once the food sources are found, employed bees establish a one-to-one correspondence with the food sources and search for a better food source near the current one. Corresponding to the ABC algorithm, the search formula for the location of food sources can be described as Eq. (3).

$$x_{ij}^{t+1} = x_{ij}^t + \varphi_{ij} \times \left(x_{ij}^t - x_{kj}^t\right) \tag{3}$$

Where $k \neq i \ \in \{1, 2, \cdots, N\}$ is a randomly chosen neighbor index that is different from i. $j \in \{1, 2, \cdots, D\}$ is a randomly determined dimension index and φ_{ij} is a uniform random number in range of $[-1,1]$. x_i^t is the current solution, x_k^t is a neighbor one which is the guiding solution, and x_i^{t+1} is the candidate solution generated by the solution search formula (3).

Onlooker bees' stage: In ABC algorithm, onlooker bees search for more optimal food sources near those identified as superior among all food sources. Specifically, the ABC algorithm utilizes the roulette wheel selection method, a fitness-based technique defined by Eq. (4), to choose superior food sources. This method favors sources with higher nectar amounts, increasing the probability that onlooker bees will explore these areas.

$$p_i = 0.9 \times \frac{f_i}{\max\{f_1, f_2, \cdots, f_N\}} + 0.1 \tag{4}$$

Where f_i is the objective function value of the solution i. For minimization problems, the quality of food would be converted as follows:

$$f_i = \begin{cases} 1/(1 + f(x_i))\text{if}(f(x_i) > 0) \\ 1 + abs(f(x_i)), \text{otherwise} \end{cases} \tag{5}$$

Where $f(x_i)$ is the objective function value associated to x_i.

Scout bees' stage: During foraging, when an employed bee exhausts a source that it was exploiting, it abandons the source and transforms into a scout bee that ventures out to explore new, potentially rich sources. Similarly, the ABC algorithm memorizes the number of nearby searches around each source in the system. If a food source fails to improve after a predefined number of trials, represented by a constant parameter "*Limit*" that source is eliminated via pruning. A new food source is then generated randomly using Eq. (2).

3.2. The pseudo-code of the basic ABC algorithm

The ABC system will iterate through the employed bees' phase, onlooker bees' phase, and scout bees' phase until the specified termination criteria are satisfied. The pseudo-code for the ABC algorithm is outlined in Algorithm 1.

Most ABC algorithms adhere to a similar algorithmic framework, as depicted in Algorithm 1. However, they differ in terms of the employed search strategies, which are executed iteratively and contribute significantly to their effectiveness and computational complexity. Algorithm 2 illustrates the search strategy of the basic ABC algorithm. It involves the random selection of the updated dimension j and the neighborhood solution k. Notably, the basic ABC algorithm utilizes a unique one-dimensional updated search formula, as described in Eq. (3), setting it apart from other SI algorithms.

4. Criteria & oBABC

4.1. The key criteria

Binary optimization problems are prevalent in various research fields, where the solution space is restricted to binary values. In other words, each decision variable in the solution can only take on a value of either 0 or 1. To extend the binary variant of the SI algorithms, several key issues must be addressed, as described below.

- How to obtain the initial solutions which belong to binary space.
- How to ensure that the new solution obtained by the solution search formula also belongs to binary space.

In the initialization phase of binary optimization algorithms, it is a commonly adopted practice to transform Eq. (2) into Eq. (6) for the purpose of generating binary values for individual decision variables.

$$x_{ij} = \begin{cases} 0, \text{ifrand} < 0.5 \\ 1, \text{otherwise} \end{cases} \tag{6}$$

Where "rand" is a uniform random number in range of [0,1].

Another crucial issue is to develop an effective search strategy that guarantees that the newly obtained solution derived from the solution search formula belongs to the binary domain. To address this issue more

Algorithm 1
Pseudo-code of the basic ABC algorithm.

```
Input: Problem model, $N$, max-epochs
  Output: best solution, best objective value
  {Initialization}
      Initialize all the parameters
      Initialize the food sources' positions by Eq. (2), then evaluate them.
      Record the best solution as $x(t)$ and its fitness as $f(t)$
      set Trail = 0
    Repeat: (evaluations = $t = 0$)
    {Employed bees' phase}
      for $i = 1: 1: N$
        Apply the solution search strategy (Algorithm 2).
        {Greedy selection}
        if $f_{new}$ better than $f_i$
          $x_i^{t+1} = x_i^{t+1}$
          $f_i = f_{new}$
          $trail_i = 0$
        else
          $x_i^{t+1} = x_i^t$
          $trail_i = trail_i + 1$
        end
      end
    {Onlooker bees' phase}
      Calculate the probability values $p_i$ by Eq. (4).
      $n = 0$, $i = 0$.
      while $n < N$
        if random $< p_i$
          Apply the solution search strategy (Algorithm 2).
          {Greedy selection}
          if $f_{new}$ better than $f_i$
            $x_i^{t+1} = x_i^{t+1}$
            $f_i = f_{new}$
            $trail_i = 0$
          else
            $x_i^{t+1} = x_i^t$
            $trail_i = trail_i + 1$
          end
          $n = n + 1$
        end
        $i = (i + 1) \bmod N$
      end
    {Scout bees' phase}
      if $\max(trail_i) > Limit$
        Reset $x_i^{t+1}$ by Eq. (2) and evaluate $x_i^{t+1}$
        $trail_i = 0$
        $t = t + 1$
      end
      Record the best solution as $x(t)$ and it's fitness as $f(t)$
      $t = t + 2*N$
    Until meet stop conditions
    Output the final solution
```

Algorithm 2
The solution search strategy.

```
Input: $x_i^t$
  Output: $x_i^{t+1}$
  Neighbor selection: Randomly select $k \epsilon [1, N],\ k \neq i$
    Direction selection: Randomly select $j \epsilon [1, D]$.
    Create a new candidate position $x_i^{t+1}$ by Eq. (3).
    Calculate the new position's fitness $f_{new}$
```

effectively, we establish key criteria for the search strategy stated below.

Criterion I: Minimizing computational complexity.

As known, minimizing the complexity is imperative as it is a fundamental metric used to evaluate the efficacy of an algorithm. In SI algorithms, the complexity is determined by the search formula, which is executed iteratively. Therefore, it is crucial to optimize the search formula for better algorithmic performance.

Criterion II: Maximizing space efficiency.

It is necessary to promise that each search is valid. If the candidate solution is the same as the current ones, it will be a waste of computational resources, and the optimization performance of the algorithm will be compromised.

Criterion III: Employing appropriate bionic strategies.

SI algorithms distinguish themselves from stochastic search by simulating swarm behavior to guide the search process. By incorporating appropriate bionic strategies, these algorithms can expedite convergence and enhance their capabilities, including the ability to escape local optima and perform global search.

To satisfy Criterion II, it is necessary to establish relevant metrics for evaluation purposes. These metrics include but are not limited to under-learning rate, over-learning rate, and invalid-learning rate (*ILR*). These measures enable us to assess the effectiveness of binary SI algorithms in meeting the predefined criterion accurately.

The search process of an SI algorithm typically consists of two primary components: input solutions (x_i^t, x_j^t, x_k^t, etc.) and output solutions (x_i^{t+1}). The input solution covers all the solutions involved in the computation of the output solution and can be further classified into the current solution (x_i^t) and the guiding solutions (x_j^t, x_k^t, etc.). Current solution is typically selected sequentially from the existing population, while guiding solutions are selected based on various strategies and may involve multiple solutions. The output solution, also referred to as the candidate solution, represents the outcome of the entire search process. Based on these components, the relevant evaluation metrics are defined as follows.

Under-learning α: It occurs when the Euclidean distance between the candidate solution x_i^{t+1} and the current solution x_i^t is 0, indicating an invalid search that fails to explore new solution spaces. Mathematically, this metric can be modeled as follows.

$$\alpha = \begin{cases} 1, & \sum_{j=1}^{D}\left(x_{ij}^{t+1} - x_{ij}^{t}\right)^2 > 0 \\ 0, & otherwise \end{cases} \tag{7}$$

Where, $\alpha == 0$ indicates that this is an invalid search, otherwise the update is valid.

Over-learning β: It occurs when the Euclidean distance between the candidate solution x_i^{t+1} and the guiding solutions (x_j^t, x_k^t, etc.) is 0, indicating an invalid search that fails to explore new solution spaces. This is again an invalid search and can be modeled as follows.

$$\beta_k = \begin{cases} 1, & \sum_{j=1}^{D}\left(x_{ij}^{t+1} - x_{kj}^{t}\right)^2 > 0 \\ 0, & otherwise \end{cases} \tag{8}$$

Where $i \neq k$, $\beta_k == 0$ signifies that the current search is invalid due to the Euclidean distance being 0 from the guiding solutions x_k^t.

In either case, it would be an invalid search. We can define the *ILR* as Eq. (9). It is evident that as *ILR* decreases, the goodness-of-fit to criterion II improves. The optimal value of *ILR* is 0 %.

$$ILR = \left(1 - \frac{1}{M}\sum_{m=1}^{M}\left(\alpha_m + \sum_{i=1}^{P}\beta_{mi}\right)\right) \times 100\% \tag{9}$$

Where $m \in \{1, 2, \cdots, M\}$, M is the number of executions of the solution search formula and M is the total number of executions of the solution search formula in the ABC algorithm. P represents the number of guiding solutions, $P = 1$ in most ABC algorithms.

In Fig. 1, an illustrative example of *ILR* is presented. Let us consider x_i^t as the current solution, x_j^t and x_k^t as the guiding solutions, and x_i^{t+1} as the candidate solution, typical cases would be as follows.

Case 1: When the Euclidean distance between the candidate solution x_i^{t+1} and the selected current solution x_i^t is 0, it indicates that the search did not deviate from the current solution, signifying an invalid search due to under-learning.

Case 2: If the Euclidean distance between the candidate solution x_i^{t+1} and a guiding solution x_j^t is 0, it implies that x_i^{t+1} is an exact replica of x_j^t, indicating an invalid search resulting from over-learning.

Case 3: Similarly, when the Euclidean distance between the candidate solution x_i^{t+1} and another guiding solution x_k^t is 0, it suggests that x_i^{t+1} is an exact duplicate of x_k^t, also indicating an invalid search due to over-learning.

Case 4: Conversely, if the new solution obtained differs from all other solutions, with a non-zero Euclidean distance from x_i^t, x_j^t, and x_k^t, it signifies that x_i^{t+1} explores a new solution space and represents a valid update.

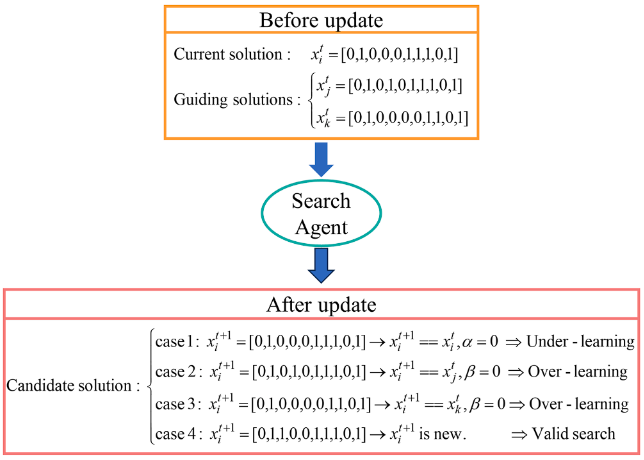
Before update
Current solution : $x_i^t = [0,1,0,0,0,1,1,1,0,1]$
Guiding solutions : $x_j^t = [0,1,0,1,0,1,1,1,0,1]$, $x_k^t = [0,1,0,0,0,0,1,1,0,1]$
Search Agent
After update
Candidate solution :
case 1 : $x_i^{t+1} = [0,1,0,0,0,1,1,1,0,1] \rightarrow x_i^{t+1} == x_i^t, \alpha = 0 \Rightarrow$ Under - learning
case 2 : $x_i^{t+1} = [0,1,0,1,0,1,1,1,0,1] \rightarrow x_i^{t+1} == x_j^t, \beta = 0 \Rightarrow$ Over - learning
case 3 : $x_i^{t+1} = [0,1,0,0,0,0,1,1,0,1] \rightarrow x_i^{t+1} == x_k^t, \beta = 0 \Rightarrow$ Over - learning
case 4 : $x_i^{t+1} = [0,1,1,0,0,1,1,1,0,1] \rightarrow x_i^{t+1}$ is new. $\Rightarrow$ Valid search

Fig. 1. Examples illustrating under-learning and over-learning scenarios. When a search agent is deployed, any candidate solution that aligns with previously generated solutions is considered an invalid search, as depicted in cases 1, 2, and 3.

These scenarios serve to distinguish between valid and invalid search processes.

4.2. *Criteria validated on the BABC algorithms*

This section is dedicated to the theoretical analysis of the criteria values presented in Section 4.1. Various BABC algorithms are employed as examples to assess the level of alignment between these algorithms and the proposed criterion. Furthermore, a quantitative analysis is conducted using the *ILR* metric.

4.2.1. *Analysis of DABC*

DABC is an example of mapping-based method. For this method, the solution search formula is the same as the basic ABC algorithm, a transfer-function is introduced to map $[-\infty, +\infty]$ to (0,1), defined as Eq. (10). And the real value obtained by Eqs. (3) and (10) represents the probability that a dimension variable should be 1, which is then used to obtain the final value as outlined in Eq. (11).

$$sig\left(x_{ij}\right) = \frac{1}{\left(1 + e^{-x_{ij}}\right)} \tag{10}$$

$$x_{ij}^{t+1} = \begin{cases} 0, & sig\left(v_{ij}^{t+1}\right) < rand \\ 1, & sig\left(v_{ij}^{t+1}\right) \geq rand \end{cases} \tag{11}$$

$$v_{ij}^{t+1} = x_{ij}^t + \varphi_{ij} \times \left(x_{ij}^t - x_{kj}^t\right) \tag{12}$$

To facilitate analysis, Eq. (3) is rephrased as Eq. (12) with its original meaning intact. This enables enumeration of the various input and output combinations arising from the search strategy employed by DABC, as summarized in Table 1.

Let $P(x_{ij}^{t+1}/(x_{ij}^t, x_{kj}^t))$ denotes the probability that a new candidate solution x_{ij}^{t+1} takes when the input is x_{ij}^t and x_{kj}^t. Then $P(0/(0,0)) = 0.5$, $P(0/(0,1)) = 0.5$, $P(1/(1,0)) = 0.5$, $P(1/(1,1)) = 0.73$. If x_{ij}^{t+1} is equal to x_{ij}^t, it is an invalid search. In this case, the probability of invalid search can be expressed as $P(ILR) = P(0/(0,0)) \times P(0,0) + P(0/(0,1)) \times P(0,1) + P(1/(1,0)) \times P(1,0) + P(1/(1,1)) \times P(1,1) = 0.5 \times 0.25 + 0.5 \times 0.25 + 0.5 \times 0.25 + 0.73 \times 0.25 = 0.5575$.

As per our analysis, it is evident that DABC does not provide a guarantee for validity in solution update at each search due to its rela-

Table 1
The analysis of the solution update strategy in DABC.

		Case 1	Case 2	Case 3	Case 4
Input	x_{ij}^t	0	0	1	1
	x_{kj}^t	0	1	0	1
Procedure	S1: after Eq. (12)	0	$-\varphi_{ij}^t$	φ_{ij}^t	1
	S2: after Eq. (10)	0.5	[0.27,0.73]	[0.27,0.73]	0.73
Output	$P(x_{ij}^{t+1}=0)$	50 %	50 %	50 %	27 %
	$P(x_{ij}^{t+1}=1)$	50 %	50 %	50 %	73 %

Note: φ_{ij}^t is a uniformly distributed random number in the range [−1,1]. Hence, after applying Eq. (10) in both case 2 and case 3, the resulting values of $sig(\pm\varphi_{ij}^t)$ will belong to the [0.27, 0.73] range, presenting a symmetrical distribution centered on 0.5. Due to this, it can be theoretically concluded that the probability of x_{ij}^{t+1} being either 0 or 1 is equal.

tively high probability of invalid searches, which has been calculated as 55.75 %. Moreover, the incorporation of exponential operations within the algorithm adds to its computational complexity [48], which scales as $O(n \times log_2^2(n) \times log_2(log_2(n)))$.

4.2.2. Analysis of ABCbin

ABCbin employs mod-round-mod operations. This approach achieves lower computational complexity compared to DABC. Specifically, the mod-round-mod operation is defined by Eq. (13), which corresponds to Eqs. (10) and (11) as utilized in DABC.

$$x_{ij}^{t+1} = round\left(\left|v_{ij}^t \bmod 2\right|\right) \bmod 2 \tag{13}$$

The input and output combinations for ABCbin's solution search process are listed in Table 2. An interesting issue has been identified wherein x_{ij}^{t+1} would obtain a certain value in case 1 and 4. The probability values for each combination are as follows: $P(0/(0,0)) = 1$, $P(0/(0,1)) = 0.5$, $P(1/(1,0)) = 0.5$, $P(1/(1,1)) = 1$. Then $P(ILR) = P(0/(0,0)) \times P(0,0) + P(0/(0,1)) \times P(0,1) + P(1/(1,0)) \times P(1,0) + P(1/(1,1)) \times P(1,1) = 1 \times 0.25 + 0.5 \times 0.25 + 0.5 \times 0.25 + 1 \times 0.25 = 0.75$.

ABCbin does not guarantee to generate new solution at each search, as its search process has a 75 % probability of invalidity. However, compared to DABC, ABCbin utilizes a modulus operation rather than an exponential operation, which significantly reduces computational requirements. Specifically, its complexity is $O(n \times log_2(n) \times log_2(log_2(n))$, which is primarily determined by Eq. (12). Whereas criterion I shows slight improvement, criterion II does not meet the requirement with $P(ILR) = 62.5$ %.

During our investigation, we have identified an issue where variable v_{ij}^t remains consistently below 2. To address this inefficiency, we propose the following update formula:

$$x_{ij}^{t+1} = round\left(\left|v_{ij}^t\right|\right) \bmod 2 \tag{14}$$

4.2.3. Analysis of binABC

binABC applies binary bitwise operations instead of real arithmetic operations in the basic ABC algorithm. The search formula for updated solutions is presented as Eq. (15).

$$x_{ij}^{t+1} = x_{ij}^t \otimes \left(\varphi\left(x_{ij}^t \otimes x_{kj}^t\right)\right) \tag{15}$$

Where '⊗' stands for a 'xor' operator and 'φ' is the logic NOT gate with 50 % probability. If φ is less than 0.5, the result obtained by $(x_{ij}^t \otimes x_{kj}^t)$ is inverted; otherwise, the result is not inverted. Here, φ is a random number uniformly selected from the interval [0, 1].

As shown in Table 3, the probabilities of $P(0/(0,0)) = 0.5$, $P(0/(0,1)) = 0.5$, $P(1/(1,0)) = 0.5$, $P(1/(1,1)) = 0.5$. The value of P(*ILR*) is subsequently determined as follows: $P(ILR) = P(0/(0,0)) \times P(0,0) + P(0/(0,1)) \times P(0,1) + P(1/(1,0)) \times P(1,0) + P(1/(1,1)) \times P(1,1) = 0.5 \times 0.25 + 0.5 \times 0.25 + 0.5 \times 0.25 + 0.5 \times 0.25 = 0.5$.

The complexity of Eq. (15) is o(3). Criterion I is effectively met, however, at P(*ILR*) = 50 %, criterion II is not satisfied.

4.2.4. Analysis of bitABC

bitABC introduces a solution update formula as Eq. (16), which differs from Eq. (3) by employing bitwise logical operators instead of arithmetic ones. Specifically, the 'xor' operator replaces 'addition', the 'and' operator replaces 'multiplication', and the 'or' operator replaces 'subtraction'. Moreover, binary values for ϕ_{ij}^t are obtained using Eq. (17).

$$x_{ij}^{t+1} = x_{ij}^t \otimes \left(\phi_{ij}^t \& \left(x_{ij}^t \middle| x_{kj}^t\right)\right) \tag{16}$$

$$\phi_{ij} = \begin{cases} 1, & rand < r \\ 0, & rand \geq r \end{cases} \tag{17}$$

Where, '&' stands for a 'and' operator, and '|' represents a 'or' operator in a binary bitwise operation. *r* is a real control parameter within the range [0,1], the default value is 0.5.

As shown in Table 4, it can be deduced that $P(0/(0,0)) = 1$, $P(0/(0,1)) = 0.5$, $P(1/(1,0)) = 0.5$, $P(1/(1,1)) = 0.5$. Then $P(ILR) = P(0/(0,0)) \times P(0,0) + P(0/(0,1)) \times P(0,1) + P(1/(1,0)) \times P(1,0) + P(1/(1,1)) \times P(1,1) = 1 \times 0.25 + 0.5 \times 0.25 + 0.5 \times 0.25 + 0.5 \times 0.25 = 0.625$.

The computational complexity of Eq. (16) is o(3). Although criterion I has been significantly improved, criterion II fails to meet the required condition, with P(*ILR*) being 62.5 %.

Table 2
The anlysis of ABCbin.

		Case 1	Case 2	Case 3	Case 4
Input	x_{ij}^t	0	0	1	1
	x_{kj}^t	0	1	0	1
Procedure	S1: after Eq. (12)	0	$-\varphi_{ij}^t$	φ_{ij}^t	1
	S2-(a): after Eq. (13) Under $\varphi_{ij}^t \in (-0.5, 0.5)$	0	0	0	1
	S2-(b): after Eq. (13) Under others	0	1	1	1
Output	$P(x_{ij}^{t+1}=0)$	100 %	50 %	50 %	0
	$P(x_{ij}^{t+1}=1)$	0	50 %	50 %	100 %

Note: φ_{ij}^t is a uniformly distributed random number in the range [−1,1]. Hence, the occurrence probability of both S2-(a) and S2-(b) is identical. Thus, in case 2 and case 3, there exists an equal theoretical probability for the value of x_{ij}^{t+1} to be either 0 or 1.

Table 3
The anlysis of binABC.

		Case 1	Case 2	Case 3	Case 4
Input	x_{ij}^t	0	0	1	1
	x_{kj}^t	0	1	0	1
Procedure	S1: after $x_{ij}^t \otimes x_{kj}^t$	0	1	1	0
	S2-(a): after Eq. (15) Under $\varphi < 0.5$	1	0	1	0
	S2-(b): after Eq. (15) Under $\varphi \geq 0.5$	0	1	0	1
Output	$P(x_{ij}^{t+1}=0)$	50 %	50 %	50 %	50 %
	$P(x_{ij}^{t+1}=1)$	50 %	50 %	50 %	50 %

Note: φ is a uniformly distributed random number in the range [0,1]. Therefore, both S2-(a) and S2-(b) occur with identical probability. Consequently, in case 2 and case 3, there is an equal theoretical likelihood for the value of x_{ij}^{t+1} to be either 0 or 1.

Table 4
The anlysis of bitABC.

		Case 1	Case 2	Case 3	Case 4
Input	x_{ij}^t	0	0	1	1
	x_{kj}^t	0	1	0	1
Procedure	S1: after $x_{ij}^t \| x_{kj}^t$	0	1	1	1
	S2-(a): after Eq. (16) Under ϕ_{ij}^t=1	0	1	0	0
	S2-(b): after Eq. (16) Under ϕ_{ij}^t=0	0	0	1	1
Output	$P(x_{ij}^{t+1} = 0)$	100 %	50 %	50 %	50 %
	$P(x_{ij}^{t+1} = 1)$	0	50 %	50 %	50 %

Note: According to the definition of Eq. (17), the probability of ϕ_{ij}^t being 0 or 1 is equal. Thus, in case 2, case 3 and case 4, there exists an equal theoretical probability for the value of x_{ij}^{t+1} to be either 0 or 1.

4.3. *Summary and analysis of criteria for binarization*

Section 4.2 presents a theoretical evaluation of several BABC algorithms, analyzing them against Criteria I and II, with the outcomes detailed in Table 5. Regarding Criterion I, it is clear that the computational complexity indicator shows the binABC and bitABC algorithms substantially outperform the DABC and ABCbin algorithms. This advantage is mainly due to the inherently lower computational complexity of binary logic operations compared to operations with real numbers. Thus, the binary-operator method, by virtue of its reduced computational complexity relative to the mapping-based method, is recommended for binary algorithm design.

With respect to Criterion II, the *ILR* is identified as a crucial metric for evaluation. The *ILR* values for the algorithms DABC, ABCbin, binABC, and bitABC all exceed 50 %, indicating, based on mathematical expectation, their failure to meet Criterion II and to assure the validity of each search. The task of theoretically deriving *ILR* for multi-dimensional update BABC algorithms, such as ibinABC, NBABC, GBABC, iBABC, and PBABC, is notably challenging. Their *ILR* values will be determined empirically in subsequent experiments.

Regarding Criterion III, despite the absence of quantitative metrics, it is clear that the current BABC algorithms lack the integration of a bio-inspired model in their update dimension selection strategies. To address this, the oBABC algorithm is proposed based on the established criteria. The theoretical insights derived will be subjected to further scrutiny and empirical validation in subsequent experiments to underscore the theoretical arguments and demonstrate the advantages of the oBABC algorithm.

4.4. *oBABC algorithm*

4.4.1. *Search formula based on criteria I & II*

One of the distinguishing features of the basic ABC algorithm is that it updates only one-dimensional variable at each search. Motivated by Criterion I, we opt for a search formula design that utilizes binary operations. Following this, in alignment with Criterion II, we undertook a theoretical analysis of the search formula for the one-dimensional BABC.

Table 5
Summary of theoretical performance of one-dimensional updated BABC algorithms.

Algorithm	Complexity	*ILR*	Criterion I	Criterion II
DABC	$O(n \times log_2^2(n) \times log_2(log_2(n)))$	55.75 %	High	Not
ABCbin	$O(n \times log_2(n) \times log_2(log_2(n)))$	75 %	Median	Not
binABC	$O(3)$	50 %	Low	Not
bitABC	$O(3)$	62.50 %	Low	Not

Table 6
The analysis of validity of solution update.

	Case 1	Case 2	Case 3	Case 4
x_{ij}^t	0	0	1	1
x_{ij}^{t+1}	0	1	0	1
Reward	invalid	valid	valid	invalid

Table 6 presents a summary regarding the validity of solution updates. If x_{ij}^{t+1} is equal to x_{ij}^t, then x_i^{t+1} will be equal to x_i^t, resulting in under-learning. An example is depicted in Fig. 2. The search path (a) is deemed invalid due to under-learning, while path (b) explores an alternative solution and be valid. Specifically, $x_{i6}^{t+1} = x_{i6}^t = 1$ would yield $f(x_i^{t+1}) = f(x_i^t)$, including but not limited to the computation of $f(x_i^{t+1})$. Then the entire computational process outlined in Algorithm 2 would become futile. These spurious calculations merely increase the computation time, but offer no benefits to the optimization process. Instead, one can refer to path (b) in Fig. 2, where setting x_{i6}^{t+1}=0 would lead to $x_i^{t+1} \neq x_i^t$, potentially yielding a different value of $f(x_i^{t+1})$ from $f(x_i^t)$. Clearly, considering solutions that differ from the current one can significantly aid in searching for unvisited space.

Hence, the one-dimensional updated BABC algorithm possesses a theoretically optimal search equation, as exemplified by Eq. (18). This equation incorporates reverse operators to guarantee a valid search process during each search. Specifically, Eq. (18) explicitly indicates that x_i^{t+1} will adopt a unique value distinct from its previous state, x_i^t. Importantly, the output obtained from this equation is directly binarized, eliminating the need for additional binarization procedures.

$$x_{ij}^{t+1} = \sim x_{ij}^t \tag{18}$$

4.4.2. *Dimension selection strategy based on criterion III*

SI algorithms provide a significant advantage over stochastic algorithms as they facilitate guided search through interactions, thereby enhancing the chances of attaining optimal solutions within desired evaluations. Considering Criterion III, we introduce a module that employs a biomimetic strategy for selecting the update dimension, instead of the random selection method utilized in the traditional ABC algorithm. Inspired by the natural waggle dance, which communicates the direction of fruitful hives to attract observers, we introduce two key modules: the difference degree and the difference vector.

Difference vector Δ: Δ is vector of *D*, where Δ_j=1 indicates that x_{ij}^t and x_{kj}^t have identical value, while Δ_j=0 indicates that x_{ij}^t and x_{kj}^t have distinct values. The calculation of Δ can be performed using Eq. (19).

$$\Delta = x_i^t \otimes x_k^t \tag{19}$$

Difference degree *l*: The Euclidean distance between two solutions, which can be calculated by Eq. (20).

$$l = \sum_{j=1}^{D} \Delta_j \tag{20}$$

Subsequently, the update dimension *j* is identified such that Δ_j==1 denotes movement towards neighboring solutions, whereas selecting Δ_j==0 as the update dimension *j* implies moving away from them. The directionality, whether towards or away from the neighboring solution, is determined by the difference degree *l* in Eq. (20), and the corresponding mathematical formulation is presented as Eq. (21).

$$d = \begin{cases} rswhere(\Delta, 0), \text{if} l < \gamma \\ rswhere(\Delta, 1), \text{else} \end{cases} \tag{21}$$

Where *rswhere*(Δ, *v*) is direction selection engine, it means that randomly selects an output *d* from where $\Delta_d = v$. *γ* is the control parameter for direction selection, named direction selection agent.

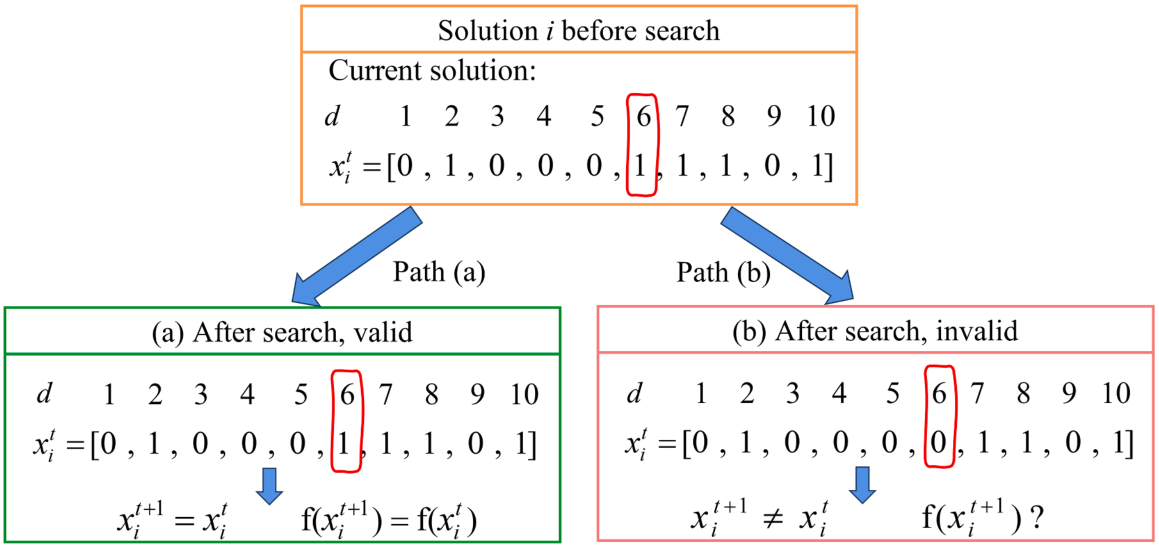
Solution *i* before search
Current solution:
d 1 2 3 4 5 6 7 8 9 10
$x_i^t = [0, 1, 0, 0, 0, 1, 1, 1, 0, 1]$
Path (a)
Path (b)
(a) After search, valid
d 1 2 3 4 5 6 7 8 9 10
$x_i^t = [0, 1, 0, 0, 0, 1, 1, 1, 0, 1]$
$x_i^{t+1} = x_i^t$
$f(x_i^{t+1}) = f(x_i^t)$
(b) After search, invalid
d 1 2 3 4 5 6 7 8 9 10
$x_i^t = [0, 1, 0, 0, 0, 0, 1, 1, 0, 1]$
$x_i^{t+1} \neq x_i^t$
$f(x_i^{t+1})$?

Fig. 2. Search efficiency analysis of a 10-Dimensional problem with dimension 6 as the update dimension. Row '*d*' represents the dimension index. Two possible paths are shown: an invalid path (a) where no search gain is achieved as x_i^{t+1} remains unchanged from x_i^t, and a valid path (b) where x_i^{t+1} takes a different value from x_i^t, enabling exploration of an unvisited $f(x_i^{t+1})$ in this search.

In contrast to the other ABC algorithms, oBABC integrates a direction selection agent γ as a novel control parameter. This agent facilitates determining whether the algorithm directs the search towards or away from neighboring solutions. As the value of γ increases, the search is directed towards neighboring solutions, promoting colony aggregation, and thereby enhancing exploitation performance, consequently accelerating convergence. Conversely, smaller values of γ direct the search away from neighbors, stimulating the colony to explore diverse locations, thus improving exploration performance.

Fig. 3 illustrates typical examples of direction selection engines, which show that γ should fall within the acceptable range of [2, *D*], rather than [0, *D*]. As indicated in Fig. 3(a) and (b), the value of γ must not be less than 2. Specifically, when *l* is 0, the operation $rswhere(\Delta, 1)$ becomes illegal, whereas when *l* is 1, $rswhere(\Delta, 1)$ results in an invalid search of over-learning. Thus, when *l* equals 0 or 1, $rswhere(\Delta, 0)$ should be promised. In other words, γ must be no less than 2. Furthermore, Fig. 3(c) indicates that when *l* is *D*, updating x_i^t to move away from x_k^t would be optimal choice, consistent with Eq. (21). Lastly, Fig. 3(d) provides an example where *v* is determined by γ which belonging to [2, *D*].

Furthermore, Eq. (22) provides a model for determining the value of γ that is dependent on *D*.

$$\gamma = round[\tau \times (D-2)] + 2 \tag{22}$$

Where $\tau \in [0, 1]$ determines the value of γ.

4.4.3. Elaboration of oBABC

oBABC is an extension of the basic ABC algorithm in binary space, with the same framework as described in Algorithm 1, but incorporating an enhanced binary search strategy, which can be found in Algorithm 3. To enhance the clarity of oBABC, we provide a flowchart in Fig. 4. Similar to the ABC algorithm, oBABC comprises three modules: employee bees, onlooker bees, and scout bees. The employee bees and onlooker bees employ distinct strategies for food source selection, determining the index of the current solution to be updated. Fig. 4(a) illustrates the main loop of the algorithm, while Fig. 4(b), Fig. 4(c), and Fig. 4(d) represent specific stages within the algorithm: the employee bees' module, the onlooker bees' module, and the scout bees' module, respectively. In the employee bees' module, the selection of x_i follows a sequential distribution without repetition. Conversely, the onlooker bees' module utilizes the roulette method to select x_m, allowing for

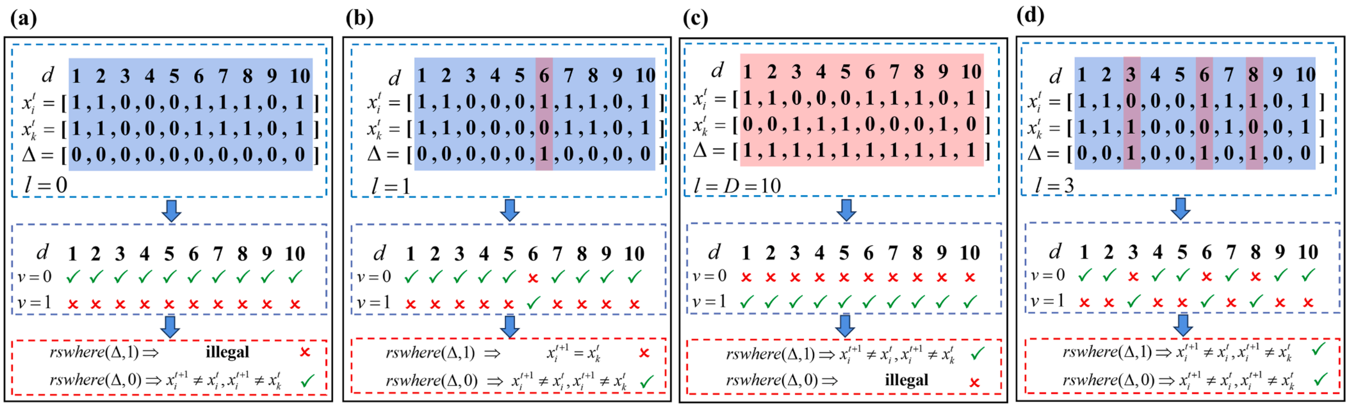
(a)
d 1 2 3 4 5 6 7 8 9 10
$x_i^t = [1,1,0,0,0,1,1,1,0,1]$
$x_k^t = [1,1,0,0,0,1,1,1,0,1]$
$\Delta = [0,0,0,0,0,0,0,0,0,0]$
$l = 0$
v=0 ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓
v=1 × × × × × × × × × ×
$rswhere(\Delta,1) \Rightarrow$ illegal ×
$rswhere(\Delta,0) \Rightarrow x_i^{t+1} \neq x_i^t, x_i^{t+1} \neq x_k^t$ ✓
(b)
d 1 2 3 4 5 6 7 8 9 10
$x_i^t = [1,1,0,0,0,1,1,1,0,1]$
$x_k^t = [1,1,0,0,0,0,1,1,0,1]$
$\Delta = [0,0,0,0,0,1,0,0,0,0]$
$l = 1$
v=0 ✓ ✓ ✓ ✓ ✓ × ✓ ✓ ✓ ✓
v=1 × × × × × ✓ × × × ×
$rswhere(\Delta,1) \Rightarrow x_i^{t+1} = x_k^t$ ×
$rswhere(\Delta,0) \Rightarrow x_i^{t+1} \neq x_i^t, x_i^{t+1} \neq x_k^t$ ✓
(c)
d 1 2 3 4 5 6 7 8 9 10
$x_i^t = [1,1,0,0,0,1,1,1,0,1]$
$x_k^t = [0,0,1,1,1,0,0,0,1,0]$
$\Delta = [1,1,1,1,1,1,1,1,1,1]$
$l = D = 10$
v=0 × × × × × × × × × ×
v=1 ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓
$rswhere(\Delta,1) \Rightarrow x_i^{t+1} \neq x_i^t, x_i^{t+1} \neq x_k^t$ ✓
$rswhere(\Delta,0) \Rightarrow$ illegal ×
(d)
d 1 2 3 4 5 6 7 8 9 10
$x_i^t = [1,1,0,0,0,1,1,1,0,1]$
$x_k^t = [1,1,1,0,0,0,1,0,0,1]$
$\Delta = [0,0,1,0,0,1,0,1,0,0]$
$l = 3$
v=0 ✓ ✓ × ✓ ✓ × ✓ × ✓ ✓
v=1 × × ✓ × × ✓ × ✓ × ×
$rswhere(\Delta,1) \Rightarrow x_i^{t+1} \neq x_i^t, x_i^{t+1} \neq x_k^t$ ✓
$rswhere(\Delta,0) \Rightarrow x_i^{t+1} \neq x_i^t, x_i^{t+1} \neq x_k^t$ ✓

Fig. 3. Analysis of the direction selection engine $rswhere(\Delta, v)$ described in Eq. (21) at different *l*. The top row shows the initial information of x_i^t and x_k^t. The middle row displays the possibility of selecting a dimension under values of *v*. The bottom row illustrates the validity of $rswhere(\Delta, v)$. *d* represents the dimension index. '✓' indicates feasibility or legality while '×' indicates the opposite. (a) $l = 0$, *v* must be 0. (b) $l = 1$, *v* should be 0 to ensure that x_i^{t+1} differs from both x_i^t and x_k^t. (c) $l = D$, *v* must be 1. (d) $l = 3$, *v* is determined by γ.

Algorithm 3
The solution search method in oBABC.

```
Input: $x_i^t$
  Output: $x_i^{t+1}$
  Neighbor selection: Randomly select $k \epsilon [1, N]$, $k \neq i$
    Direction selection: select j by Eq. (21), Eq. (19) and Eq. (20)
    Create a new candidate position $x_i^{t+1}$ by Eq. (18).
    Calculate the new position's fitness $f_{new}$
```

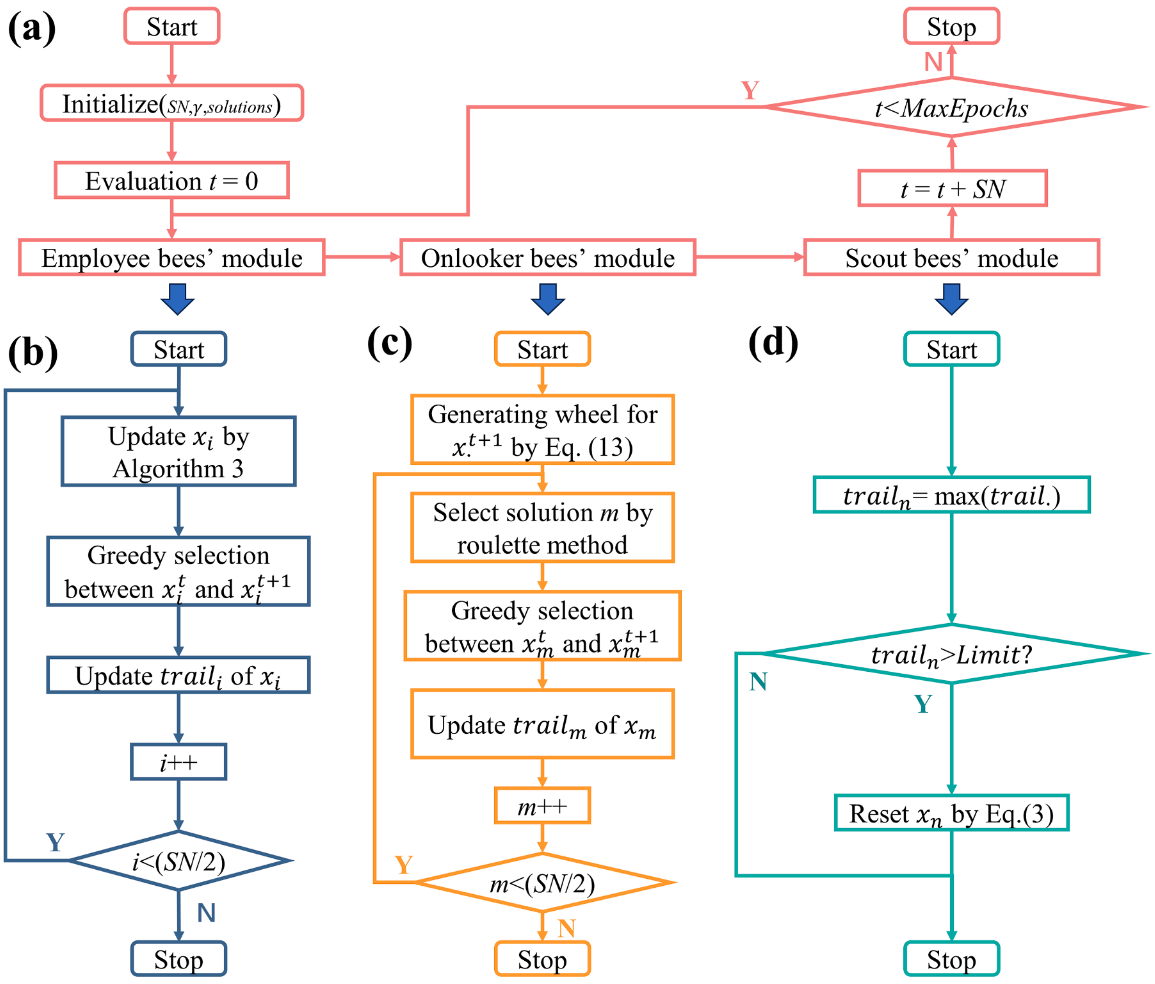
(a)
Start
Initialize(SN,γ,solutions)
Evaluation t = 0
Employee bees' module
Onlooker bees' module
Scout bees' module
t = t + SN
t<MaxEpochs
Y
N
Stop
(b)
Start
Update x_i by Algorithm 3
Greedy selection between x_i^t and x_i^{t+1}
Update $trail_i$ of x_i
i++
i<(SN/2)
Y
N
Stop
(c)
Start
Generating wheel for $x_{\cdot}^{t+1}$ by Eq. (13)
Select solution m by roulette method
Greedy selection between x_m^t and x_m^{t+1}
Update $trail_m$ of x_m
m++
m<(SN/2)
Y
N
Stop
(d)
Start
$trail_n$= max(trail.)
$trail_n$>Limit?
N
Y
Reset x_n by Eq.(3)
Stop

Fig. 4. Flowchart of oBABC. (a) Main loop. (b) Employee bees' module. (c) Onlooker bees' module. (d) Scout bees' module. While x_i in employee bees' module is selected in order, x_m in onlooker bees' module is selected by roulette method.

probability-based selection that may involve repetitions. For example, in a colony with 5 solutions, the sequences of current solution indices can be observed as i= [1,2,3,4,5] for the sequential distribution without repetition, and m= [1,1,3,3,5] for the probability-based selection that permits potential repetition.

An example of oBABC is shown in Fig. 5. In this case, the difference vector Δ is [1,1,1,1,0,1,0,1,0,0]. Applying the search methodology of oBABC with τ=0.1, an update dimension 4 is generated by Eq. (21) and a new solution is outputted by Eq. (18) as [0,1,1,0,0,1,1,1,0,1].

5. Experiment

This section provides a detailed explanation of the experimental materials, performance metrics, and results. The experiments have been conducted using MATLAB R2021b on an Intel CPU Core i5–12400F @ 2.50 GHz (16 GB RAM) system running on Windows 11. To assess the performance of oBABC, we employed the UFLP and Max-Cut problems. We compare the results obtained from oBABC with state-of-the-art variants of the BABC algorithms, including one-dimensional updated BABC algorithms such as DABC [37], ABCbin [38], binABC [35], bitABC [36], as well as multi-dimensional updated BABC algorithms like NBABC [40], ibinABC [41] iBABC [42], GBABC [43], PBABC [27]. Additionally, we include DPSO [29] for comparison purposes. The same control parameters are used for all algorithms to ensure fair comparison, with a colony size of 40 (The ABC-based algorithms comprise a population of 20 employee bees and 20 onlooker bees, whereas DPSO algorithm population consists of 40 particles.). For UFLP, the maximum number of function evaluations is set as 80,000. While for Max-Cut, it is limited to 20,000.

5.1. Experimental material

Binary optimization problems form a crucial subset of numerical optimization problems. UFLP [49] and Max-Cut are widely recognized

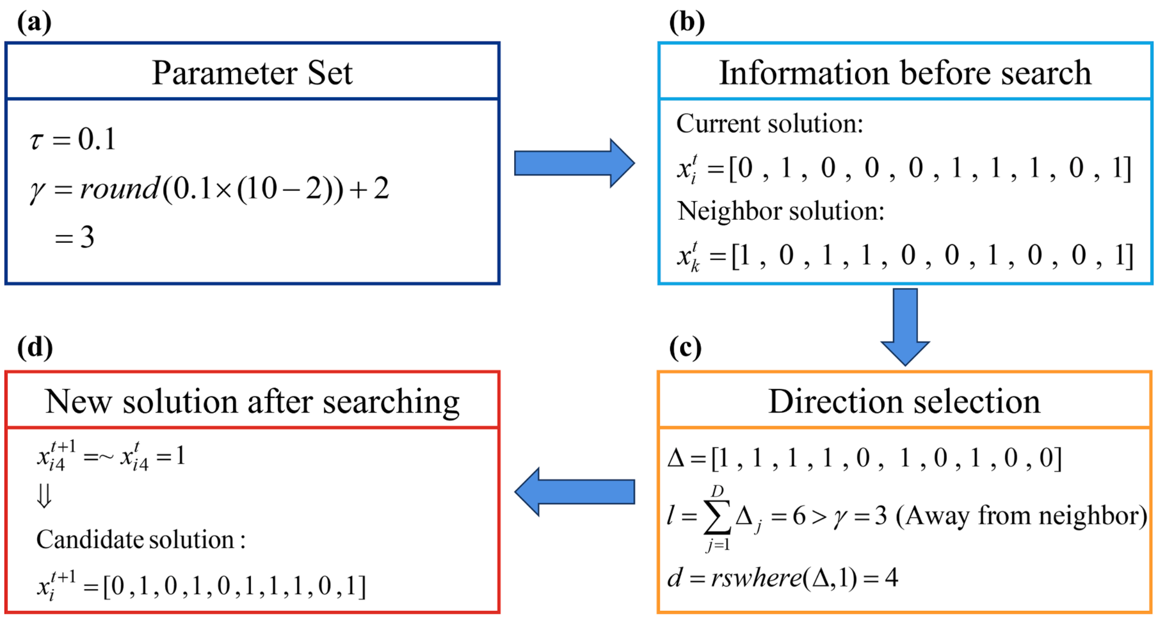
(a)
Parameter Set
$\tau = 0.1$
$\gamma = round(0.1\times(10-2))+2 = 3$
(b)
Information before search
Current solution:
$x_i^t = [0,1,0,0,0,1,1,1,0,1]$
Neighbor solution:
$x_k^t = [1,0,1,1,0,0,1,0,0,1]$
(c)
Direction selection
$\Delta = [1,1,1,1,0,1,0,1,0,0]$
$l = \sum_{j=1}^{D}\Delta_j = 6 > \gamma = 3$ (Away from neighbor)
$d = rswhere(\Delta,1) = 4$
(d)
New solution after searching
$x_{i4}^{t+1} = \sim x_{i4}^t = 1$
Candidate solution :
$x_i^{t+1} = [0,1,0,1,0,1,1,1,0,1]$

Fig. 5. An example illustrates search strategy in oBABC. (a) Direction selection agent γ is 3, calculated by using Eq. (22) with τ=0.1. (b) x_k^t is randomly selected as the learning neighbor for the current solution x_i^t. (c) Update dimension 4 is generated by Eq. (21). (d) A new solution x_i^{t+1} is outputted by the search strategy of oBABC.

as representative benchmark problems for assessing the performance of binary optimization algorithms.

UFLP is a constrained optimization problem, and the binary variables are used to represent the assignment of facilities and the satisfaction of capacity constraints. The problem can be mathematically expressed using Eq. (23). Further, Table 7 provides a summary of optimal costs along with details regarding test problem sizes and names for UFLP.

$$\min f = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}\times x_{ij} + \sum_{i=1}^{n} fix_i \times y_i \tag{23}$$

Where fix_i refers to the fixed cost associated with the opening of facility i, while c_{ij} denotes the cost incurred when facility i caters to the needs of customer j. Here, x_{ij}=1 indicates that facility i serves customer j, and y_i=1 implies that facility i is open.

The Max-Cut problem serves as another binary problem that can be utilized to assess the effectiveness of optimization algorithms. The objective is to find the maximum cut in an undirected graph $G = (V, E)$, where V represents the set of vertices and E represents the set of edges. The maximum cut aims to locate a cut between two distinct sets of vertices, S and its complement, $\overline{S} = V\backslash S$, such that the size of the cut is larger than any other possible cut in the graph. Specifically, a cut in a graph represents the number of edges separating two sets of vertices viewed as separate and complementary partitions. A maximum cut refers to the cut with the most considerable weight, and can be defined mathematically via Eq. (24). Reference characteristics related to the Max-Cut problem are presented in Table 8.

$$\max W(S,\overline{S}) = \sum_{i\in S, j\in \overline{S}} w_{ij} \tag{24}$$

Where $w_{ij} = w_i \times w_j$ is the weight of the edge between node i and node j, w_i is the weight of vertex i and $w_i \epsilon \{1, -1\}$, which depending on the partition that w_i belongs to.

5.2. Performance evaluation metrics

In this subsection, a specific set of performance metrics has been presented to achieve a more precise quantification of the validity of the theoretical analysis discussed in Section 4.3 and to conduct a comprehensive comparative analysis with several state-of-the-art BABC algorithms. The performance metrics are as follows.

Invalid-learning Rate (*ILR*): Refers to the rate at which the newly generated solution is deemed invalid, either due to it being identical to the current solution or its neighbor, as defined as Eq. (9) in Section 4.1.

Best: Represents the best value of the output objective function over

Table 7
Test problem sets for the UFLP, used in the experiments.

Level	Problem Name	Problem Size	Cost of Optimal Solution
Small-size	Cap71	16×50	932,615.75
	Cap72	16×50	977,799.4
	Cap73	16×50	1,010,641.45
	Cap74	16×50	1,034,976.98
Medium-size	Cap101	25×50	796,648.44
	Cap102	25×50	854,704.20
	Cap103	25×50	893,782.11
	Cap104	25×50	928,941.75
Large-size	Cap131	50×50	793,439.56
	Cap132	50×50	851,495.33
	Cap133	50×50	893,076.71
	Cap134	50×50	928,941.75
Extra-large-size	CapA	100×1000	17,156,454.48
	CapB	100×1000	12,979,071.58
	CapC	100×1000	11,505,594.33

Table 8
Test problem sets for the Max-Cut problem, used in the experiments. n and d are respectively the problem size and the density.

Problem Name	Solution	Problem Name	Solution	Problem Name	Solution
n = 100, d = 0.1		n = 100, d = 0.5		n = 100, d = 0.9	
pw01–100.0	2019	pw05–100.0	8190	pw09–100.0	13,585
pw01–100.1	2060	pw05–100.1	8045	pw09–100.1	13,417
pw01–100.2	2032	pw05–100.2	8039	pw09–100.2	13,461
pw01–100.3	2067	pw05–100.3	8139	pw09–100.3	13,656
pw01–100.4	2039	pw05–100.4	8125	pw09–100.4	13,514
pw01–100.5	2108	pw05–100.5	8169	pw09–100.5	13,574
pw01–100.6	2032	pw05–100.6	8217	pw09–100.6	13,640
pw01–100.7	2074	pw05–100.7	8249	pw09–100.7	13,501
pw01–100.8	2022	pw05–100.8	8199	pw09–100.8	13,593
pw01–100.9	2005	pw05–100.9	8099	pw09–100.9	13,658

multiple runs.

Worst: Represents the worst value of the output objective function over multiple runs.

Mean: Represents the mean value of the output objective function over multiple runs.

Std.Dev.: Represents the standard deviation of the output objective function over multiple runs.

Matching Rate (*MR*): Represents the degree of similarity between the objective function value of the current solution and that of the optimal solution, defined by Eq. (25). Specifically, *MR* is defined as the best value achieved by an algorithm in a single run. *MR*(*fit*) denotes the value of *MR* under the agreed rules. For instance, *MR*(*Mean*), *MR*(*Best*), and *MR*(*Worst*) represent the average, best, and worst *MR* values obtained by the algorithm across multiple runs, respectively. *MR*(*f*(*t*)) represents the best *MR* value attained by an algorithm up to evaluation *t*.

$$MR(fit) = 1 - \left|\frac{(fit - Optimum)}{Optimum}\right| \times 100\% \tag{25}$$

HitTimes(*x*): Quantifies the number with which the *MR* of an algorithm reaches a specified value, denoted by *x*. For illustration, *HitTimes* (99 %) indicates the number of runs in which the *MR* achieves 99 % of the optimal solution. This metric is invaluable for assessing the search capabilities of algorithms, as it measures the frequency with which they attain the desired optimal solution.

HitFirst(*x*): Defined as the minimum number of evaluations required for an optimized algorithm to reach a certain *MR*, represented by *x*. For instance, *HitFirst*(99 %) would correspond to the earliest evaluation *t* when $MR(f(t)) = 99\,\%$. *HitFirst*(*x*) provides crucial information about the search efficiency of algorithms in reaching the desired *MR*. By measuring the minimum number of evaluations needed for an algorithm to achieve a certain level of optimization, we can get insight into the search efficiency of the algorithm.

5.3. Results

5.3.1. Validation of ILR for various algorithms

According to the analysis in Section 4.3, and based on criterion II, it has been revealed that non-ideal search methods primarily lead to invalid searches. The theoretical estimates of *ILR* values for different algorithms are shown in Table 5. To validate this finding, a series of tests are carried out on UFLP and Max-Cut problems with varying sizes, and the outcomes are depicted in Fig. 6. oBABC has been designed to mitigate the occurrence of invalid searches, and the experimental results confirm its effectiveness in this regard. oBABC exhibits high search efficiency with an *ILR* of 0 %, while the other algorithms remain stuck in invalid searches with higher *ILR* values. Specifically, in the context of the Max-Cut problem, the *ILR* values of the one-dimensional updated ABC algorithm are closely aligned with the corresponding theoretical values. However, when considering the UFLP problem, the experimental *ILR* values significantly exceeds the theoretical values. This discrepancy can be attributed to the tendency of many algorithms to quickly converge to local optima or rapidly identify the optimal solution in the UFLP problem, resulting in a reduction in population diversity. In contrast, the Max-Cut problem poses a challenge in finding the optimal solution and provides a richer solution space for the algorithm to explore.

Considering multi-dimensional updated BABC algorithm, deriving their *ILR* values theoretically poses a challenge. Nonetheless, experimental results presented in this section show that *ILR* values surpass 50 % for the UFLP and generally exceed 20 % for the Max-cut problem. Such findings highlight the issue of invalid updates in BABC algorithms that feature multi-dimensional updates, thus failing to meet Criterion II. Although *ILR* is not the sole indicator of an algorithm’s optimization efficacy, it points to potential computational resource wastage. In summary, these critical insights emphasize the superior search efficiency of the oBABC algorithm regarding *ILR*, relative to other BABC algorithms, marking a notable advancement in algorithmic design.

5.3.2. On the choice of the direction agent γ

This subsection examines the effect of γ on solution quality obtained by oBABC. As its value determined by τ, we vary τ from 0 to 1 in increments of 0.1 to evaluate algorithm performance. The convergence plot in Fig. 7(a) illustrates that smaller τ leads to faster convergence. In terms of solution quality and robustness under different τ, we analyze mean and standard deviation in Fig. 7(b)~(c). A lower objective function value indicates better solution quality for the UFLP problems. Based on Fig. 7(b) and Fig. 7(c), when τ=0.1, the search capability and robustness are best balanced, indicating that setting τ can balance the exploitation and exploration, and enhance search efficiency and capability of oBABC.

5.3.3. Analysis of computation time

The computation time of an algorithm is a crucial metric for assessing its performance, and it is a key aspect of criterion I. In Fig. 8(a), the time taken by various algorithms to complete a function evaluation on UFLP and Max-Cut problems is presented. The results show that most algorithms have similar runtimes, which can be attributed to advancements in computer processing power. Fig. 8(b) displays the time needed

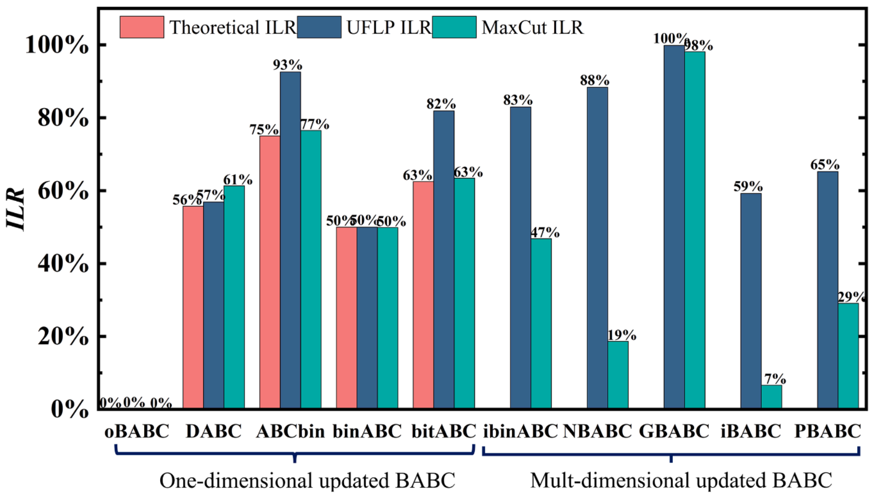

Fig. 6. Comparison of invalid-learning rate (*ILR*) for different algorithms. Theoretical values are quoted from Table 5, while experimental values are obtained from statistical analysis of the UFLP and Max-Cut problems using Eq. (9).

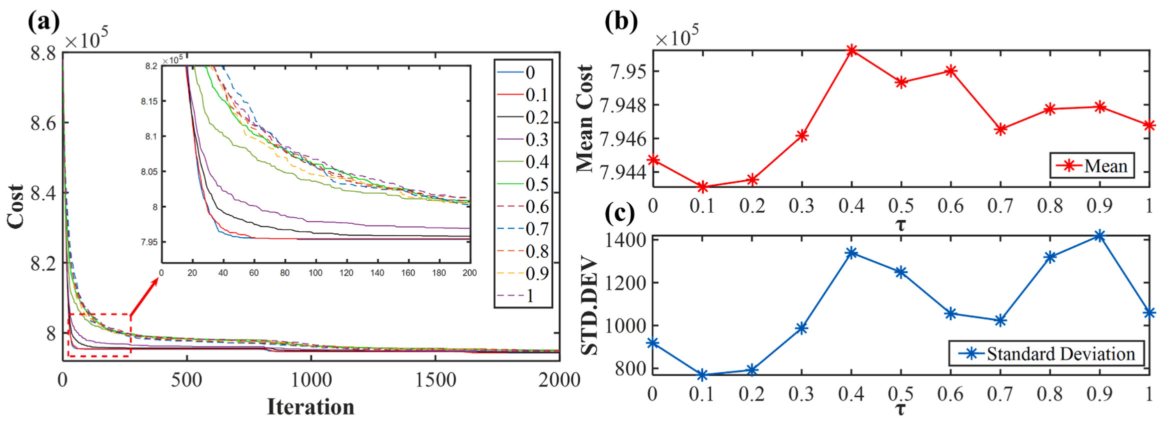

Fig. 7. Impact of varying τ on Cap131 dataset. (a) Convergence curves for different values of τ, showcasing differences in convergence speed as highlighted in the zoomed-in plot. (b) Mean cost across 30 runs for each value of τ. (c) Standard deviation across 30 runs for each value of τ.

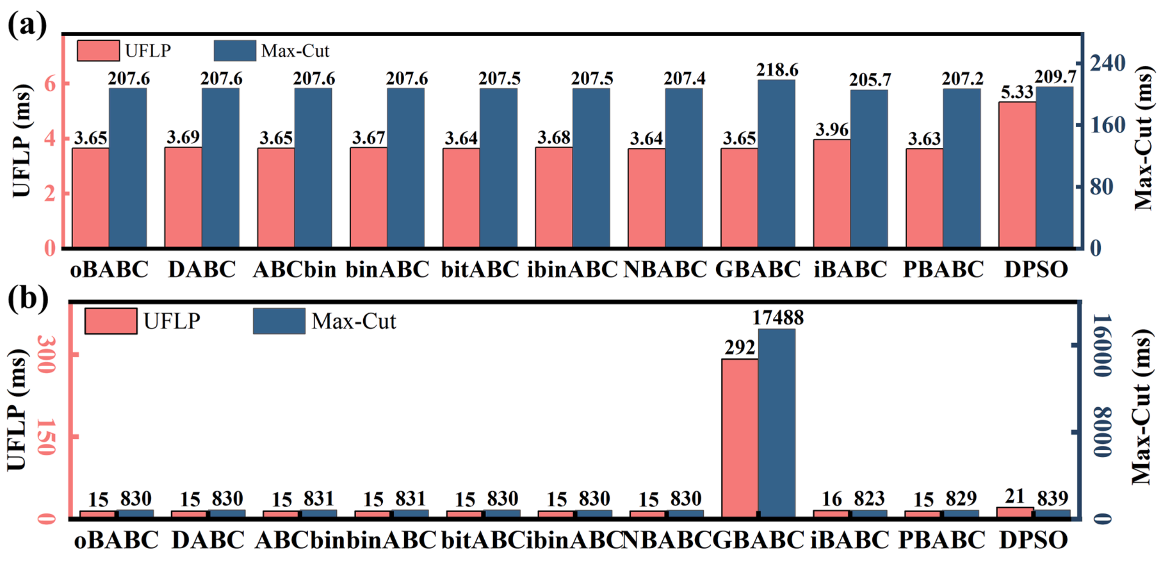

Fig. 8. The runtime of different algorithms for solving UFLP and Max-Cut problems. (a) The time taken by various algorithms to complete a function evaluation on these problems. (b) The time taken by various algorithms per epoch for solving these problems.

for different algorithms to finish a population iteration. GBABC stands out with a significantly longer runtime, about 20 times greater than the other algorithms. This difference arises because each solution update in GBABC involves evaluating 10 children and 10 grandchildren candidate solutions, leading to an epoch time twenty times longer than that of other algorithms. While this strategy improves population diversity, it also introduces additional computational overhead. To ensure fairness, we employ the maximum number of function evaluations as a stopping criterion for the comparative analysis of algorithm performance in subsequent experiments.

5.3.4. *Performance on UFLP problems*

Numerical experimental results obtained from UFLP datasets are summarized in Table S1 (Supplementary Materials Table S1). As observed in Table S1, oBABC, like other algorithms, successfully finds the optimal solution in at least one out of 30 independent runs. The distinction among these algorithms primarily lies in their search efficiency. To provide a more intuitive comparison of their performance, we utilize the convergence curve, a commonly employed tool for evaluating algorithm efficiency and performance. Fig. 9 depicts the convergence curves for different problem scales. It is evident from Fig. 9(a)-(b) that oBABC exhibits comparable convergence for small and medium-sized problems, while DPSO and multi-dimensional updated BABC converges faster in the initial stage but requires more search epochs to escape local optima and find the global optimum. For large-scale problems, both oBABC and NBABC demonstrate competitive performance, as illustrated in Fig. 9(c). However, as shown in Fig. 9(d), for extra-scale problems, oBABC converges at a slower rate compared to the partially multi-dimensional updated BABC algorithms (GBABC, NBABC, iBABC). The convergence curves for all 15 problems can be found in Fig. S1. They indicate that oBABC performs comparably to other BABC algorithms in terms of search performance and successful identification of the optimal solution. In comparison to the one-dimensional updated BABC algorithms, it exhibits clear advantages in terms of convergence speed and optimization outcomes. While the multi-dimensional updated BABC algorithms demonstrate slightly faster convergence, suggesting that the multi-dimensional update and multiple candidate solutions in each search epoch contribute to improved population diversity. In summary, most BABC algorithms demonstrate superior optimization performance compared to DPSO. Regarding search efficiency, oBABC shows a significant enhancement over the one-dimensional updated BABC algorithm and performs comparably to the multi-dimensional updated BABC algorithm. These results indicate that the oBABC algorithm offers a competitive solution for UFLP problems.

To quantitatively analyze the search efficiency of different algorithms, we conduct a statistical analysis of the minimum number of

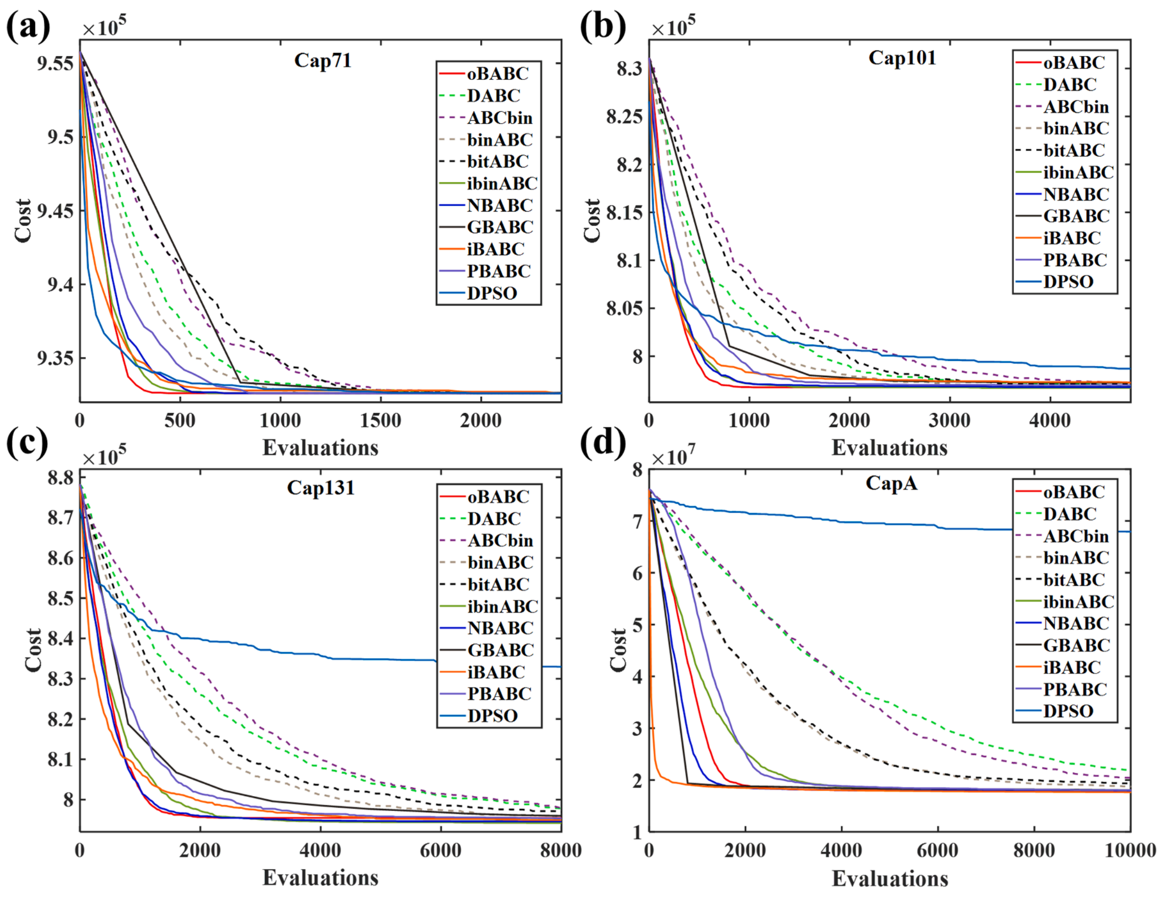

Fig. 9. Convergence curves of different algorithms for typical UFLP problems with varied sizes. The problems used include (a) Cap 71 as a representative of small-size problems, (b) Cap 101 as a representative of medium-size problems, (c) Cap 131 as a representative of large-size problems. (d) Cap A as a representative of extra-large-size UFLP problems.

evaluations required to reach the target solution for problems of varying sizes. For small and medium-sized problems, as shown in Fig. 10(a), oBABC achieves the optimal solution within 2500 evaluations, which get a competitive result. In the case of large-scale problems, as depicted in Fig. 10(b), oBABC converges to the target solution with a 99 % matching degree only in approximately 1000 evaluations, while most one-dimensional updated BABC algorithms require more than 4000 evaluations. However, for extra-large-scale problems, achieving the optimal solution in every run is not guaranteed by any algorithm. We conduct a statistical analysis on the output matching degree value, *MR* (Fig. 10(c)), which clearly demonstrates that oBABC holds a distinct advantage over the one-dimensional updated BABC algorithms. Compared to the multi-dimensional updated BABC algorithms (iBABC, ibinABC), oBABC exhibits comparable performance.

5.3.5. *Performance on max-cut problems*

Max-cut is another highly challenging binary optimization problem for which most SI algorithms do not guarantee the discovery of an optimal solution. Fig. 11(a) presents an analysis of the algorithms' ability to find optimal or near-optimal solutions. Among the algorithms, oBABC demonstrates the highest success rate, achieving optimal solutions in at least one run ("*MR*(*Best*)=100 % in 30 runs") for 13 out of 30 problems. In contrast, the other algorithms fail to reach the optimal solution. Furthermore, oBABC outperforms the rest in terms of "*MR* (*Best*)>99 % in 30 runs," solving all 30 problems with nearly-optimal solutions and securing the top position. Evaluating "*MR*(*Mean*)>99 % in 30 runs," oBABC provides solutions for 20 problems, nearly double the number achieved by the second-ranked binABC algorithm. These findings highlight the better capability and robustness of oBABC in finding optimal or near-optimal solutions. Considering search efficiency, as illustrated in Fig. 11(b), oBABC consistently requires fewer evaluations to achieve a 98 % convergence to the optimal solution compared to other algorithms. The performance of oBABC is more robustness, and its box width is reduced by about 28 % compared to other algorithms. Examining the search capability, the analysis of *MR* values depicted in Fig. 11(c) reveals that oBABC consistently achieves *MR* values primarily above 99 %, while the other BABC algorithms mostly fall below 99 % in most runs. This observation highlights the superiority of oBABC in terms of search performance and efficiency in the Max-cut problem.

Turning to the convergence curves, depicted in Fig. 12, it is evident that oBABC outperforms the others in terms of both convergence rate and results. Due to space limitations, only a subset of these comparisons is presented in Fig. 12, while additional figures can be found in Fig. S2 and Fig. S3. Furthermore, most of the multi-dimensional updated BABC algorithms demonstrate better performance compared to their one-dimensional counterparts. Notably, oBABC consistently exhibits superior performance in terms of both convergence rate and results, followed by NBABC. These findings highlight the exceptional ability of oBABC to achieve superior results compared to other algorithms, characterized by faster convergence rates and higher solution quality.

The detailed results obtained from the Max-Cut problem are summarized in Table S2. To conduct a comprehensive statistical analysis, we employ the Wilcoxon signed-rank test to analyze the output of the algorithm. As shown in Table 9, oBABC consistently outperforms the other algorithms across almost all problems, with only binABC, ibinABC, and NBABC demonstrating comparable performance on 4, 6, and 5 of the 30 problems, respectively. However, the remaining problems exhibit notably inferior performance compared to oBABC. DABC, ABCbin, and

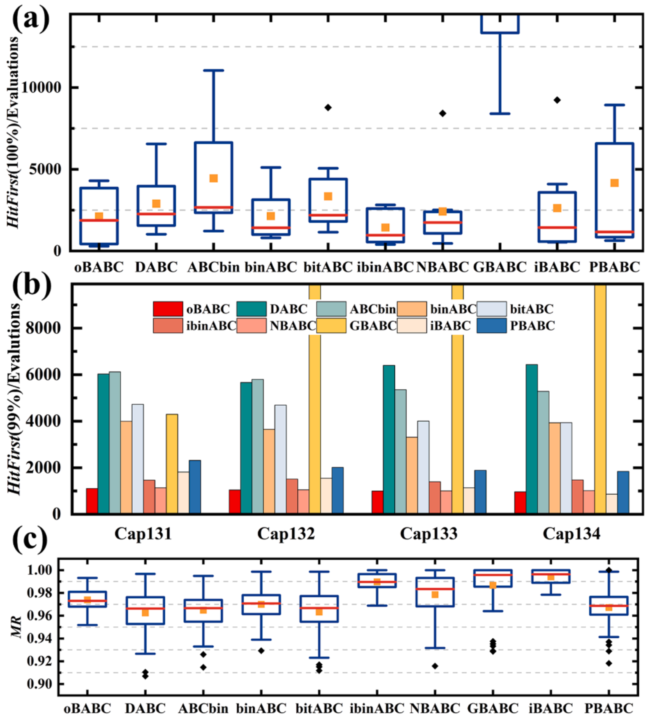

Fig. 10. Statistical analysis of performance of various algorithms across different UFLP problems. (a) Distribution of *HitFirst*(100) values for small-size and medium-size problems. (b) Comparison of search efficiency using *HitFirst* (99) on large-size problems. (c) The distribution of *MR* achieved by each algorithm on extra-large-size problems in 30 runs.

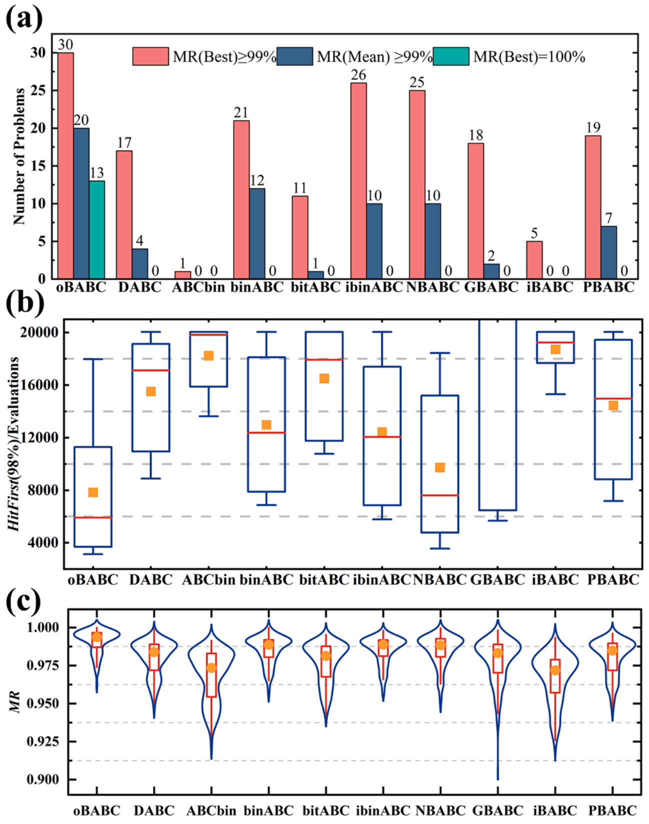

Fig. 11. Statistical analysis of algorithm performance across different Max-Cut problems. (a) The number of problems in which the algorithms can identify specific near-optimal solutions, with thresholds $MR(Best) \geq 99$ %, $MR(Mean) \geq 99$ %, and $MR(Best) = 100$ %. (b) Distribution of *HitFirst*(98 %) values for all Max-Cut problems. (c) Distribution of *MR* values for each algorithm on a Max-Cut problems.

DPSO consistently demonstrate inferior rankings compared to all other algorithms, and their results are not included in Table 9 due to space constraints, but can be found in Table S3. Moreover, a *t*-test is employed to provide a quantitative assessment by establishing a confidence interval for the difference between the means. The results of *t*-test, presented in Table S4, further validate the superior performance of oBABC in comparison to the others. Specifically, the confidence intervals of *t*-test reveal a clear advantage for oBABC, despite the Wilcoxon signed-rank test indicating their equivalence. Due to the extensive size of *t*-test tables, they are provided in the Supplementary Materials.

Furthermore, Table S2 provides detailed evidence supporting the superior performance of oBABC across various metrics, including the *Mean, Best, Worst,* and *Std.Dev* of the objective function, as well as the *HitFirst*(98 %), *HitTimes*(99 %), and *HitTimes*(100 %) metrics. We rank these algorithms based on the presented metrics, which are illustrated in the form of a radar chart in Fig. 13. The chart clearly demonstrates oBABC's superiority in terms of search efficiency (*HitFirst*(98 %)), search capability (*HitTimes*(99 %), *HitTimes*(100 %)), and robustness (*Best, Worst, Mean, Std.Dev*) compared to the other algorithms.

5.4. Discussion

In this work, we have formulated criteria for designing binary variants of SI algorithms and introduced oBABC algorithm. Criterion I highlights the necessity of minimizing the complexity of the search formula. Despite technological advancements in computing power, memory, and parallel computing, which could mitigate the impact of computational efficiency, it is crucial to ensure low computational complexity in the design phase. Our analysis indicates that binary-operator approaches are more computationally efficient than mapping-based methods in developing binary variants of SI algorithms. As a result, oBABC algorithm adopts a binary operation methodology.

Criterion II necessitates the design of a search formula that ensures exploration of new solution spaces. Experimental results reveal a significant presence of invalid searches in BABC algorithms. For one-dimensional updated BABC algorithms, the *ILR* values align closely with theoretical expectations, consistently exceeding 50 %. For multi-dimensional updated BABC algorithms, despite the theoretical challenges in estimating *ILR* values, experimental evidence suggests that *ILR* exceeds 50 % for the UFLP and 20 % for the Max-Cut problem. Although *ILR* alone does not determine an algorithm's optimization effectiveness, it points to potential computational resource wastage. The oBABC algorithm demonstrates superior search efficiency with an ambition of achieving a zero *ILR*, marking a notable advancement in algorithmic design.

Criterion III promotes the integration of biomimetic strategies to increase the likelihood of quickly identifying optimal solutions. The biomimetic model is what distinguishes SI algorithms from stochastic algorithms. oBABC algorithm employs a biomimetic strategy in its update dimension selection process, representing a considerable improvement over the conventional ABC algorithm. Further experiments in UFLP and Max-Cut problems have verified oBABC's enhanced convergence rate compared to one-dimensional update BABC algorithms, requiring significantly fewer function evaluations to achieve the *HitFirst*(100 %) and *HitFirst*(99 %) benchmarks—less than 25 % of that needed by one-dimensional updating BABC algorithms. In challenging Max-Cut problems, oBABC has demonstrated a higher probability of finding optimal solutions, outperforming other algorithms in terms of reaching 99 % of solutions and the function evaluations required to achieve 99 % sub-optimal solutions, thereby highlighting its performance advantage. These initial findings indicate oBABC's potential as an efficient and effective solution for binary problems.

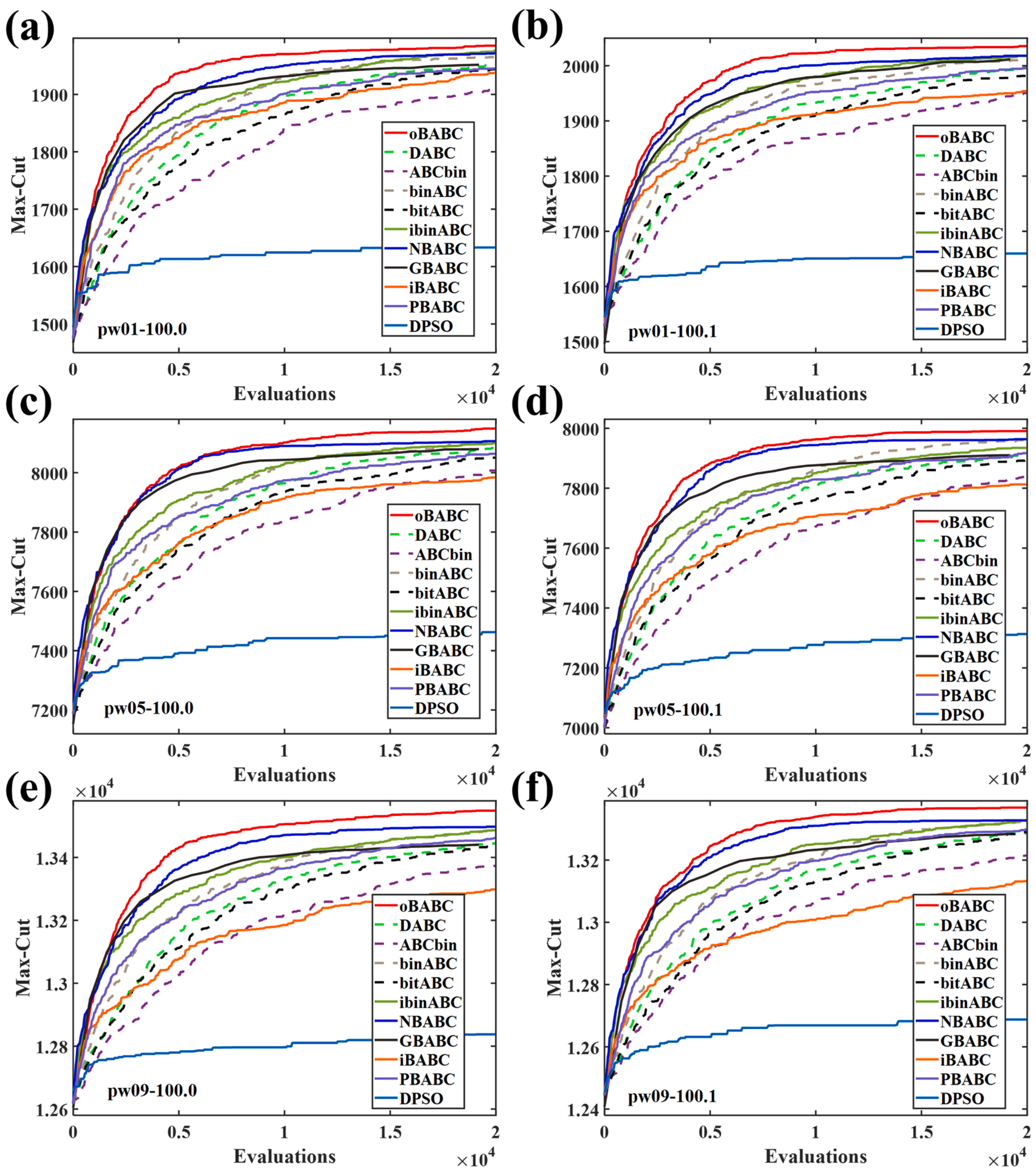

Fig. 12. Convergence curves of various algorithms for Max-Cut datasets with varying densities. The figure displays convergence performance plots for different algorithms running on typical Max-Cut datasets with(a)∼(b) density $d = 0.1$, (c)∼(d) density $d = 0.5$, (e)∼(f) density $d = 0.9$.

6. Conclusion and future works

Our work introduces oBABC, a novel one-dimensional BABC algorithm designed to solve binary problems. And our work makes three core contributions. First, we recommend binarization criteria that have not been previously identified in the literature to our knowledge. To validate these criteria, several typical BABC algorithms are analyzed according to the criteria, and some experiments are designed to verify the results of theoretical analysis. Second, we find a theoretically optimal one-dimensional update search formula, which enables exploration of the solution space surrounding current solutions while avoiding mapping problem spaces to other domains. This formula results in significant improvements in search efficiency. Third, we propose an innovative update dimension selection strategy that emulates multiple interactions among bees. This approach directs the direction of updates and leads to significant improvements in convergence rates. oBABC represents a new milestone of one-dimensional update BABC algorithm. Specifically, when tested on UFLP and Max-cut problems, oBABC outperforms current state-of-the-art binary variants of ABC algorithm by achieving the optimum or near-optimum solution. Furthermore, oBABC displays stronger search capability in terms of solution quality and robustness, as evidenced by the best, mean, worst, and standard deviation metrics, thus surpassing other algorithms. Therefore, it suggests that oBABC has potential for the application of solving complex binary problems.

On the other hand, we have observed that the multi-dimensional update strategy offers certain advantages in terms of optimization performance. In our future research, we aim to investigate effective ways to synergistically combine the concepts of one-dimensional update and multi-dimensional update. The goal is to preserve the fine search capability inherent in the one-dimensional update while integrating the

Table 9
Detailed statistical results obtained by the Wilcoxon signed-rank test on Max-cut problems (α = 0.05). A p-value of ’1′ indicates that oBABC outperforms the other algorithms, ’0′ indicates comparable performance between the algorithms, and ’−1′ signifies that oBABC performs worse. The results of the DABC and ABCbin algorithms show significantly worse performance. Due to space limitations, these results are not presented here and presented in Table S3.

Problem	binABC	bitABC	ibinABC	NBABC	GBABC	iBABC	PBABC
	p-value(h)	p-value(h)	p-value(h)	p-value(h)	p-value(h)	p-value(h)	p-value(h)
pw01–100–0	4.88e-03(1)	9.77e-04(1)	*8.40e-02(0)*	*1.03e-01(0)*	1.95e-03(1)	1.95e-03(1)	9.77e-04(1)
pw01–100–1	4.88e-03(1)	1.95e-03(1)	2.73e-02(1)	2.64e-02(1)	4.39e-02(1)	9.77e-04(1)	9.77e-04(1)
pw01–100–2	*5.12e-01(0)*	9.77e-04(1)	*2.78e-01(0)*	1.86e-02(1)	9.77e-03(1)	2.93e-03(1)	4.88e-03(1)
pw01–100–3	1.86e-02(1)	9.77e-04(1)	3.91e-03(1)	1.95e-03(1)	1.95e-03(1)	9.77e-04(1)	9.77e-04(1)
pw01–100–4	1.95e-02(1)	9.77e-04(1)	7.81e-03(1)	3.22e-02(1)	4.88e-03(1)	9.77e-04(1)	1.37e-02(1)
pw01–100–5	*3.26e-01(0)*	9.77e-04(1)	*2.08e-01(0)*	3.03e-02(1)	1.95e-03(1)	9.77e-04(1)	2.44e-02(1)
pw01–100–6	*1.21e-01(0)*	1.95e-03(1)	*8.01e-02(0)*	*9.28e-02(0)*	9.77e-04(1)	9.77e-04(1)	9.77e-04(1)
pw01–100–7	1.95e-03(1)	9.77e-04(1)	*7.62e-02(0)*	4.49e-02(1)	9.77e-04(1)	9.77e-04(1)	9.77e-04(1)
pw01–100–8	3.22e-02(1)	9.77e-04(1)	9.77e-04(1)	*5.27e-02(0)*	9.77e-04(1)	9.77e-04(1)	1.95e-03(1)
pw01–100–9	2.54e-02(1)	9.77e-04(1)	*6.54e-02(0)*	*3.85e-01(0)*	1.95e-03(1)	1.95e-03(1)	9.77e-04(1)
pw05–100–0	1.95e-03(1)	9.77e-04(1)	1.95e-03(1)	6.84e-03(1)	2.93e-03(1)	9.77e-04(1)	9.77e-04(1)
pw05–100–1	6.84e-03(1)	9.77e-04(1)	1.95e-03(1)	9.77e-03(1)	9.77e-04(1)	9.77e-04(1)	9.77e-04(1)
pw05–100–2	2.93e-03(1)	9.77e-04(1)	9.77e-04(1)	1.95e-03(1)	9.77e-04(1)	9.77e-04(1)	9.77e-04(1)
pw05–100–3	9.77e-04(1)	9.77e-04(1)	4.88e-03(1)	1.95e-03(1)	9.77e-04(1)	9.77e-04(1)	2.93e-03(1)
pw05–100–4	9.77e-04(1)	9.77e-04(1)	2.93e-03(1)	9.77e-03(1)	9.77e-04(1)	9.77e-04(1)	1.95e-03(1)
pw05–100–5	9.77e-04(1)	9.77e-04(1)	9.77e-04(1)	9.77e-04(1)	9.77e-04(1)	9.77e-04(1)	9.77e-04(1)
pw05–100–6	9.77e-04(1)	9.77e-04(1)	9.77e-04(1)	1.95e-03(1)	4.88e-03(1)	9.77e-04(1)	9.77e-04(1)
pw05–100–7	2.34e-02(1)	9.77e-04(1)	9.77e-04(1)	9.77e-04(1)	9.77e-04(1)	9.77e-04(1)	9.77e-04(1)
pw05–100–8	4.49e-02(1)	9.77e-04(1)	9.77e-04(1)	9.77e-04(1)	9.77e-04(1)	9.77e-04(1)	9.77e-04(1)
pw05–100–9	9.77e-04(1)	9.77e-04(1)	2.93e-03(1)	1.86e-02(1)	9.77e-04(1)	9.77e-04(1)	1.95e-03(1)
pw09–100–0	1.95e-03(1)	9.77e-04(1)	9.77e-04(1)	9.77e-04(1)	9.77e-04(1)	9.77e-04(1)	9.77e-04(1)
pw09–100–1	5.86e-03(1)	9.77e-04(1)	9.77e-04(1)	9.77e-03(1)	1.95e-03(1)	9.77e-04(1)	2.93e-03(1)
pw09–100–2	*5.27e-02(0)*	9.77e-04(1)	3.91e-03(1)	1.95e-03(1)	9.77e-04(1)	9.77e-04(1)	1.95e-03(1)
pw09–100–3	6.84e-03(1)	9.77e-04(1)	9.77e-04(1)	5.27e-02(0)	1.95e-03(1)	9.77e-04(1)	1.95e-03(1)
pw09–100–4	1.86e-02(1)	9.77e-04(1)	4.88e-03(1)	1.95e-02(1)	1.95e-03(1)	9.77e-04(1)	9.77e-04(1)
pw09–100–5	6.84e-03(1)	9.77e-04(1)	9.77e-04(1)	1.86e-02(1)	9.77e-04(1)	9.77e-04(1)	9.77e-04(1)
pw09–100–6	1.37e-02(1)	9.77e-04(1)	1.07e-02(1)	1.37e-02(1)	9.77e-04(1)	9.77e-04(1)	1.95e-03(1)
pw09–100–7	3.22e-02(1)	9.77e-04(1)	9.77e-04(1)	4.88e-03(1)	3.91e-03(1)	9.77e-04(1)	9.77e-04(1)
pw09–100–8	9.77e-04(1)	1.95e-03(1)	4.88e-03(1)	1.95e-03(1)	2.93e-03(1)	9.77e-04(1)	9.77e-04(1)
pw09–100–9	6.84e-03(1)	9.77e-04(1)	1.95e-03(1)	9.77e-04(1)	9.77e-04(1)	9.77e-04(1)	9.77e-04(1)
All	1.53e-36(1)	4.79e-51(1)	1.64e-42(1)	2.42e-36(1)	1.11e-48(1)	4.80e-51(1)	5.40e-50(1)

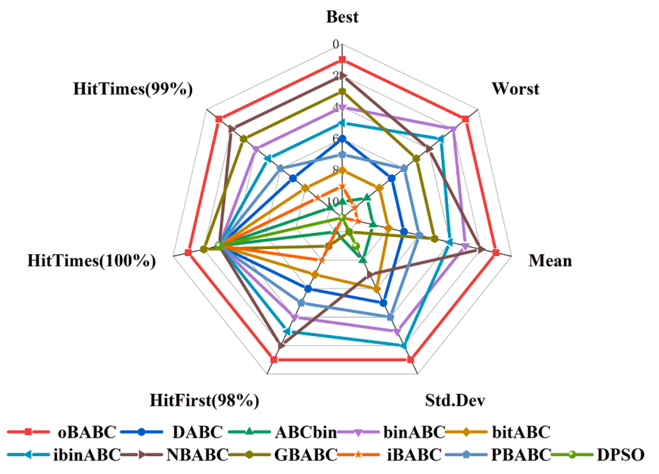

Fig. 13. Overall performance ranking of the algorithms operating on Max-Cut datasets.

rapid convergence and global exploration abilities of the multi-dimensional update. Additionally, we plan to extend the application scope of oBABC to address a broader range of engineering problems, including feature/attribute space reduction and classification/clustering problems.

Data availability

The codes and datasets are available online at https://github.com/PHD-Fang/oBABC.

CRediT authorship contribution statement

Fangfang Zhu: Conceptualization, Methodology, Data curation, Writing – original draft. **Zhenhao Shuai:** Data curation, Validation, Visualization, Investigation. **Yuer Lu:** Data curation, Validation. **Honghong Su:** Investigation, Resources. **Rongwen Yu:** Project administration, Resources. **Xiang Li:** Visualization, Investigation. **Qi Zhao:** Conceptualization, Methodology, Writing – review & editing, Funding acquisition. **Jianwei Shuai:** Conceptualization, Supervision, Funding acquisition, Project administration, Writing – review & editing.

Declaration of competing interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Data availability

Data will be made available on request.

Funding

This work is supported by the Ministry of Science and Technology of the People’s Republic of China (STI2030-Major Projects2021ZD0201900), National Natural Science Foundation of China (Grant No. 12090052), Natural Science Foundation of Liaoning Province (Grant No. 2023-MS-288), Fundamental Research Funds for the Central Universities (Grant No. 20720230017), and Basic Public Welfare Research Program of Zhejiang Province (Grant No. LGF20F030005).

Supplementary materials

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.swevo.2024.101567.

References

[1] U. Erkan, A. Toktas, D. Ustun, Hyperparameter optimization of deep CNN classifier for plant species identification using artificial bee colony algorithm, J. Ambient. Intell. Humaniz. Comput. (2022) 1–12.
[2] Q. He, C.Q. Zhong, X. Li, H. Guo, Y. Li, M. Gao, R. Yu, X. Liu, F. Zhang, D. Guo, F. Ye, T. Guo, J. Shuai, J. Han, Dear-DIA(XMBD): deep Autoencoder enables Deconvolution of data-independent acquisition proteomics, Research (Wash D C) 6 (2023) 0179.
[3] F. Latifoğlu, A novel singular spectrum analysis-based multi-objective approach for optimal FIR filter design using artificial bee colony algorithm, Neural Comput. Appl. 32 (2020) 13323–13341.
[4] S. Somesula, N. Sharma, A. Anpalagan, Artificial Bee optimization aided joint user association and resource allocation in HCRAN, Appl. Soft Comput. 125 (2022) 109152.
[5] B. Zhou, Z. Zhao, An adaptive artificial bee colony algorithm enhanced by Deep Q-Learning for milk-run vehicle scheduling problem based on supply hub, Knowl. Based Syst. 264 (2023) 110367.
[6] E. Zorarpacı, S.A. Özel, Privacy preserving rule-based classifier using modified artificial bee colony algorithm, Expert Syst. Appl. 183 (2021) 115437.
[7] Q. Pu, C. Xu, H. Wang, L. Zhao, A novel artificial bee colony clustering algorithm with comprehensive improvement, Vis. Comput. 38 (2022) 1395–1410.
[8] H. Hu, Z. Feng, H. Lin, J. Zhao, Y. Zhang, F. Xu, L. Chen, F. Chen, Y. Ma, J. Su, Q. Zhao, J. Shuai, Modeling and analyzing single-cell multimodal data with deep parametric inference, Brief Bioinform. 24 (2023) bbad005.
[9] J. Zhao, J. Sun, S.C. Shuai, Q. Zhao, J. Shuai, Predicting potential interactions between lncRNAs and proteins via combined graph auto-encoder methods, Brief Bioinform. 24 (2023) bbac527.
[10] E. Osaba, E. Villar-Rodriguez, J. Del Ser, A.J. Nebro, D. Molina, A. LaTorre, P. N. Suganthan, C.A. Coello Coello, F. Herrera, A tutorial on the design, experimentation and application of metaheuristic algorithms to real-World optimization problems, Swarm Evol. Comput. 64 (2021) 100888.
[11] R. Eberhart, J. Kennedy, A new optimizer using particle swarm theory, in: Proceedings of the sixth international symposium on micro machine and human science, Ieee, Nagoya, Japan, 1995, pp. 39–43.
[12] Y.-J. Gong, J. Zhang, O. Liu, R.-Z. Huang, H.S.-H. Chung, Y.-H. Shi, Optimizing the vehicle routing problem with time windows: a discrete particle swarm optimization approach, IEEE Trans. Syst. Man Cybernet Part C (Appl. Rev.) 42 (2011) 254–267.
[13] M. Dorigo, M. Birattari, T. Stutzle, Ant colony optimization, IEEE Comput. Intell. Mag. 1 (2006) 28–39.
[14] D. Karaboga, B. Basturk, On the performance of artificial bee colony (ABC) algorithm, Appl. Soft Comput. 8 (2008) 687–697.
[15] F. Zhao, Z. Wang, L. Wang, T. Xu, N. Zhu, Jonrinaldi, A multi-agent reinforcement learning driven artificial bee colony algorithm with the central controller, Expert Syst. Appl. 219 (2023) 119672.
[16] J. Wu, Y.-G. Wang, K. Burrage, Y.-C. Tian, B. Lawson, Z. Ding, An improved firefly algorithm for global continuous optimization problems, Expert Syst. Appl. 149 (2020) 113340.
[17] S. Mirjalili, S.M. Mirjalili, A. Lewis, Grey Wolf Optimizer, Adv. Eng. Softw. 69 (2014) 46–61.
[18] X.-S. Yang, A New Metaheuristic bat-Inspired Algorithm, in: Nature inspired Cooperative Strategies for Optimization (NICSO 2010), Springer, 2010, pp. 65–74.
[19] S. Agahian, T. Akan, Battle royale optimizer for training multi-layer perceptron, Evolv. Syst. 13 (2022) 563–575.
[20] B. Akay, D. Karaboga, B. Gorkemli, E. Kaya, A survey on the artificial bee colony algorithm variants for binary, integer and mixed integer programming problems, Appl. Soft Comput. 106 (2021) 107351.
[21] T. Ye, W. Wang, H. Wang, Z. Cui, Y. Wang, J. Zhao, M. Hu, Artificial bee colony algorithm with efficient search strategy based on random neighborhood structure, Knowl. Based Syst. 241 (2022) 108306.
[22] H. Zhao, C. Zhang, X. Zheng, C. Zhang, B. Zhang, A decomposition-based many-objective ant colony optimization algorithm with adaptive solution construction and selection approaches, Swarm Evol. Comput. 68 (2022) 100977.
[23] Z. Yong, H. Chun-lin, S. Xian-fang, S. Xiao-yan, A multi-strategy integrated multi-objective artificial bee colony for unsupervised band selection of hyperspectral images, Swarm Evol. Comput. 60 (2021) 100806.
[24] Y. Yu, F.-Q. Zhang, G.-D. Yang, Y. Wang, J.-P. Huang, Y.-Y. Han, A discrete artificial bee colony method based on variable neighborhood structures for the distributed permutation flowshop problem with sequence-dependent setup times, Swarm Evol. Comput. 75 (2022) 101179.
[25] H. Wang, Z. Wu, S. Rahnamayan, H. Sun, Y. Liu, J.-s. Pan, Multi-strategy ensemble artificial bee colony algorithm, Inf. Sci. (Ny) 279 (2014) 587–603.
[26] X. Zhou, J. Lu, J. Huang, M. Zhong, M. Wang, Enhancing artificial bee colony algorithm with multi-elite guidance, Inf. Sci. (Ny) 543 (2021) 242–258.
[27] M.S. Kiran, A binary artificial bee colony algorithm and its performance assessment, Expert Syst. Appl. 175 (2021) 114817.
[28] S.S. Jadon, R. Tiwari, H. Sharma, J.C. Bansal, Hybrid artificial bee colony algorithm with differential evolution, Appl. Soft Comput. 58 (2017) 11–24.
[29] J. Kennedy, R.C. Eberhart, A discrete binary version of the particle swarm algorithm, in: 1997 IEEE International conference on systems, man, and cybernetics. Computational cybernetics and simulation, IEEE, 1997, pp. 4104–4108.
[30] S. Mirjalili, A. Lewis, S-shaped versus V-shaped transfer functions for binary particle swarm optimization, Swarm Evol. Comput. 9 (2013) 1–14.
[31] A.P. Engelbrecht, G. Pampara, Binary differential evolution strategies, in: 2007 IEEE congress on evolutionary computation, IEEE, 2007, pp. 1942–1947.
[32] X. Wang, H. Ren, X. Guo, A novel discrete firefly algorithm for Bayesian network structure learning, Knowl. Based Syst. 242 (2022) 108426.
[33] T. Akan, S. Agahian, R. Dehkharghani, Binbro: binary battle royale optimizer algorithm, Expert Syst. Appl. 195 (2022) 116599.
[34] G. Pamparà, A.P. Engelbrecht, Binary artificial bee colony optimization, in: 2011 IEEE Symposium on Swarm Intelligence, IEEE, 2011, pp. 1–8.
[35] M.S. Kiran, M. GÜNdÜZ, XOR-based artificial bee colony algorithm for binary optimization, Turkish J. Electric. Eng. Comput. Sci. 21 (2013) 2307–2328.
[36] D. Jia, X. Duan, M.K. Khan, Binary Artificial Bee Colony optimization using bitwise operation, Comput. Ind. Eng. 76 (2014) 360–365.
[37] Y. Marinakis, M. Marinaki, N. Matsatsinis, A hybrid discrete artificial bee colony-GRASP algorithm for clustering, in: 2009 International Conference on Computers & Industrial Engineering, IEEE, 2009, pp. 548–553.
[38] M.S. Kiran, The continuous artificial bee colony algorithm for binary optimization, Appl. Soft Comput. 33 (2015) 15–23.
[39] M.H. Kashan, N. Nahavandi, A.H. Kashan, DisABC: a new artificial bee colony algorithm for binary optimization, Appl. Soft Comput. 12 (2012) 342–352.
[40] C.J. Santana Jr, M. Macedo, H. Siqueira, A. Gokhale, C.J. Bastos-Filho, A novel binary artificial bee colony algorithm, Future Generat.Comput. Syst. 98 (2019) 180–196.
[41] R. Durgut, Improved binary artificial bee colony algorithm, Front. Inf. Technol. Electron. Eng. 22 (2021) 1080–1091.
[42] A. Telikani, A.H. Gandomi, A. Shahbahrami, M.Naderi Dehkordi, Privacy-preserving in association rule mining using an improved discrete binary artificial bee colony, Expert Syst. Appl. 144 (2020) 113097.
[43] C. Ozturk, E. Hancer, D. Karaboga, A novel binary artificial bee colony algorithm based on genetic operators, Inf. Sci. (Ny) 297 (2015) 154–170.
[44] E. Kaya, M.S. Kiran, An improved binary artificial bee colony algorithm, in: 2017 15th International Conference on ICT and Knowledge Engineering (ICT&KE), IEEE, 2017, pp. 1–6.
[45] H. Hu, Z. Feng, H. Lin, J. Cheng, J. Lyu, Y. Zhang, J. Zhao, F. Xu, T. Lin, Q. Zhao, J. Shuai, Gene function and cell surface protein association analysis based on single-cell multiomics data, Comput. Biol. Med. 157 (2023) 106733.
[46] W. Wang, L. Zhang, J. Sun, Q. Zhao, J. Shuai, Predicting the potential human lncRNA-miRNA interactions based on graph convolution network with conditional random field, Brief Bioinform. 23 (2022) bbac463.
[47] H. Gao, J. Sun, Y. Wang, Y. Lu, L. Liu, Q. Zhao, J. Shuai, Predicting metabolite–disease associations based on auto-encoder and non-negative matrix factorization, Brief. Bioinform. 24 (2023) bbad259.
[48] R.P. Brent, Fast multiple-precision evaluation of elementary functions, J. ACM (JACM) 23 (1976) 242–251.
[49] J.E. Beasley, Lagrangean heuristics for location problems, Eur. J. Oper. Res. 65 (1993) 383–399.