symmetry

Article

BCA: Besiege and Conquer Algorithm

Jianhua Jiang [1,*], Xianqiu Meng [2], Jiaqi Wu [1], Jun Tian [1], Gaochao Xu [2] and Weihua Li [3]

1 Center for Artificial Intelligence, Jilin University of Finance and Economics, Changchun 130117, China; 5221191008@s.jlufe.edu.cn (J.W.); 6231193030@s.jlufe.edu.cn (J.T.)
2 College of Computer Science and Technology, Jilin University, Changchun 130012, China; mengxq23@mails.jlu.edu.cn (X.M.); xugc@jlu.edu.cn (G.X.)
3 School of Engineering, Computer and Mathematical Sciences, Auckland University of Technology, Auckland 1010, New Zealand; weihua.li@aut.ac.nz
* Correspondence: jjh@jlufe.edu.cn; Tel.: +86-155-2685-5899

Abstract: This paper introduces a bio-inspired meta-heuristic algorithm, the Besiege and Conquer Algorithm (BCA), developed to tackle complex and high-dimensional optimization problems. Drawing inspiration from the concept of symmetry and guerrilla warfare strategies, the BCA incorporates four core components: besiege, conquer, balance, and feedback. The besiege strategy strengthens exploration, while the conquer strategy enhances exploitation. Balance and feedback mechanisms maintain a dynamic equilibrium between these capabilities, ensuring robust optimization performance. The algorithm's effectiveness is validated through benchmark test functions, demonstrating superior results in comparison with existing methods, supported by Friedman rankings and Wilcoxon signed-rank tests. Beyond theoretical and experimental validation, the BCA showcases its real-world relevance through applications in engineering design and classification problems, addressing practical challenges. These results underline the algorithm's strong exploration, exploitation, and convergence capabilities and its potential to contribute meaningfully to diverse real-world domains.

Keywords: besiege and conquer algorithm; meta-heuristics optimizer; swarm intelligence; computational intelligence

check for updates

Academic Editor: Theodore E. Simos

Received: 2 January 2025
Revised: 23 January 2025
Accepted: 26 January 2025
Published: 1 February 2025

Citation: Jiang, J.; Meng, X.; Wu, J.; Tian, J.; Xu, G.; Li, W. BCA: Besiege and Conquer Algorithm. *Symmetry* **2025**, *17*, 217. https://doi.org/10.3390/sym17020217

1. Introduction

Optimization is the systematic approach of selecting a strategy to achieve the most favorable outcome under defined or uncertain constraints [1]. Optimization problems can be broadly categorized into constrained and unconstrained types. In the context of Artificial Neural Networks (ANNs), optimization is a high-dimensional and non-linear task, where the objective is to minimize the loss function by adjusting parameters such as weights and biases. Due to the presence of multiple local minima in the loss landscape, achieving optimal model performance is a global optimization problem. Conventional methods, such as gradient descent, often struggle with convergence to global minima, becoming trapped in local optima. Conversely, meta-heuristic algorithms, with their global search capabilities, have shown promise in navigating these complex landscapes and are designed to reach the global optimal solution, thus addressing one of the primary challenges in ANN optimization. These advanced approaches offer robust alternatives by diversifying search mechanisms to explore solutions beyond local regions, enhancing overall model accuracy and stability across diverse problem domains.

The Multi-Layer Perceptron (MLP) is a foundational model in deep learning and machine learning [2], widely applied across various domains [3,4], including classification, regression, and pattern recognition. Its architecture, consisting of multiple layers of interconnected neurons, enables MLPs to learn complex non-linear relationships in data, making them highly effective for tasks that require intricate pattern recognition. This flexibility has positioned MLPs as essential components in advancing machine-learning methodologies, contributing to the development of powerful predictive models [5]. However, MLPs are often regarded as "black-box" models, making the interpretation of internal weights and feature extraction processes challenging. Furthermore, the performance of MLPs can vary significantly depending on the problem, often requiring task-specific optimization and adjustments to prevent overfitting or underfitting in complex, high-dimensional datasets. Although MLPs have reached a level of maturity in their application within deep learning, they continue to present valuable areas for research and development. MLPs can become even more robust and adaptable to a wider range of applications, meeting the increasing demands for interpretable and efficient machine-learning models.

Meta-heuristic algorithms can effectively solve complex optimization problems such as high-dimensional, non-convex, and multimodal problems by balancing global search and local development [6,7]. Their strong adaptability and innovation have demonstrated excellent problem-solving capabilities in practical problems such as resource allocation, path planning, and hyperparameter optimization. Meta-heuristics methods, such as Gray Wolf Optimization (GWO) [8], Ant Colony Optimization (ACO) [9], and Genetic Algorithms (GA) [10], have become increasingly prominent in neural network training, particularly in optimizing MLPs [11]. The application and optimization in optimizing the parameters of MLPs offers significant advantages, particularly in complex and non-linear optimization landscapes where traditional gradient-based methods may struggle. Meta-heuristics methods excel in global search capabilities, systematically exploring expansive parameter spaces to identify optimal solutions, a robustness that is particularly valuable in neural network training for circumventing the challenges of local minima in high-dimensional spaces [12]. Despite the above advantages, they also face notable challenges in MLP optimization. Firstly, these methods often experience slower convergence rates, especially in high-dimensional parameter spaces, which can significantly increase training complexity. Secondly, these methods may still become trapped in suboptimal regions in highly rugged or complex landscapes. Finally, due to the vast diversity for their architectures and applications, meta-heuristic methods may require tailored adaptations to achieve optimal performance. These methods may need to be redesigned or fine-tuned to effectively address specific network structures, data distributions, or optimization objectives [13]. To address the above challenges, recent research focuses on hybrid approaches, adaptive parameter tuning, and dynamic meta-heuristic methods variations. As the application of meta-heuristic methods continues to evolve, research on these advancements is expected to address existing limitations and fully exploit the potential in optimizing MLPs and other neural network architectures [14,15].

These algorithms typically combine multiple strategies or processes, which in turn amplifies interdependencies among the individual components [16]. Furthermore, the intrinsic complexity of hybrid frameworks constrains their scalability, making it challenging to implement these algorithms for larger or more complex problems without significant alterations. In contrast, newly proposed single algorithms often feature a simplified structure and reduced parameter count to facilitate both their design and implementation [17]. Unlike the generalized frameworks of hybrid algorithms, single algorithms are typically tailored to address specific challenges or bottlenecks in particular problem domains. These specialized algorithms frequently exhibit enhanced performance on specific problems, as their design is more closely tailored to meet the particular demands of the optimization task [18,19].

With the accelerated development of machine learning, traditional optimization algorithms frequently demonstrate limitations for high-dimensional data and intricate model structures. Particularly in the training of MLPs, conventional optimization methods often struggle with challenges such as suboptimal convergence rates and a pronounced susceptibility to local optima [20]. These constraints may impede the performance of the model, thereby diminishing its efficiency and generalizability in tackling intricate, real-world challenges. As models become increasingly complex and the dimensions of data expand, these challenges may intensify the computational requirements, rendering the training process both resource-intensive and time-consuming. It is imperative to address these challenges in order to enhance the applicability of MLPs in domains that demand efficient, robust, and scalable solutions. This underscores the need for the development of more advanced optimization techniques that are specifically designed to meet the evolving requirements of contemporary neural network architectures.

This study proposes an intelligent optimization algorithm, termed the Besiege and Conquer Algorithm (BCA), which incorporates innovative strategies, including besiege, conquer, balance, and feedback. The proposal of the BCA is inspired by the collaborative dynamics and tactical behaviors observed between armies and soldiers during warfare, effectively translating these principles into a computational framework for optimization. The BCA introduces distinctive "besiege" and "conquer" strategies based on the symmetry concept. The mutual assistance of besiege and conquer strategies in the search process can solve the local stagnation problem of traditional methods for high-dimensional problems, such as PSO and ACO. The BCA is based on a feedback mechanism and the adaptive update of the Besiege and Conquer Balance (*BCB*), which makes it more adaptable when solving complex optimization problems. In addition, the BCA can effectively optimize the weights and biases. This approach not only fosters a more efficient global search but also enhances local refinement to achieve fast convergence and efficient training performance. The BCA exhibits considerable promise in improving adaptability and resilience in various intricate training contexts, thereby serving as a viable alternative for parameter optimization in high-dimensional, non-linear optimization challenges. Eventually, the main **contributions** of this research can be summarized as follows:

- A methodology grounded in human behavior is proposed, and a thorough besiege and conquer strategy is conducted.
- All mechanisms are modeled mathematically, including besiege, conquer, balance, and feedback strategies. The **besiege** strategy contributes to exploration, while the **conquer strategy** is dedicated to exploitation. The **balance** and **feedback** strategies enhance the balance between exploration and exploitation capabilities.
- The BCA introduces the parameter *BCB*, which controls the balance mechanism to speed up convergence.
- The superiority of the BCA is verified on IEEE CEC 2017 benchmark test functions, two engineering designs, and three classification problems.

The performance of the BCA is tested by the IEEE CEC 2017 benchmark functions and compared with some other meta-heuristic algorithms, such as weIghted meaN oF vectOrs (INFO) [21], Reptile Search Algorithm (RSA) [22], Self-Organizing Migrating Algorithm Team To Team Adaptive (SOMA T3A) [23], Butterfly Optimization Algorithm (BOA) [24], GWO [8], Differential Evolution (DE) [25], Genetic Algorithm (GA) [10], and PSO [26], using Friedman ranking and the Wilcoxon signed statistical test. In addition, the BCA's applicability is demonstrated by two engineering designs, including the Tension/Compression Spring [27] and Gear Train Design [28] problems, and three classification problems, including the XOR [29], Ballon [30], and Tic-Tac-Toe [31] datasets.

2. Related Work

2.1. Meta-Heuristic Algorithms

Meta-heuristic algorithms constitute an advanced category of optimization methodologies aimed at discovering optimal or near-optimal solutions for intricate optimization challenges. These algorithms utilize the principles of computational intelligence and are inspired by a wide range of natural phenomena and human behaviors, drawing from various fields, including biology, physics, and the social sciences [32,33]. Meta-heuristic algorithms can be categorized into (1) evolutionary algorithms, (2) swarm intelligence algorithms, (3) human-based algorithms, and (4) physics-based algorithms. Through these classifications, meta-heuristic algorithms demonstrate significant versatility in tackling a diverse array of optimization challenges, making them essential instruments in disciplines such as engineering, finance, and artificial intelligence. Their adaptive characteristics and ability to conduct global searches enhance their effectiveness in managing the complexities and uncertainties that are intrinsic to real-world optimization issues.

Evolutionary algorithms based on evolution mainly simulate the evolutionary law of survival of the fittest in nature (Darwin's law) to achieve the overall progress of the population and finally solve the optimal solution. A brief review is shown in Table 1. Among them, the two most prominent ones are GA [10] and DE [25].

Table 1. A brief review of evolutionary algorithms.

Algorithm	Abbreviations	Authors and Year
Evolution Strategy	ES	Rechenberg et al., 1973 [34]
Genetic Algorithm	GA	Holland et al., 1992 [10]
CoEvolutionary Algorithm	CEA	Hillis et al., 1990 [35]
Differential Evolution	DE	Storn et al., 1997 [25]
Imperialist Competitive Algorithm	ICA	Atashpaz-Gargari et al., 2007 [36]
Differential Search Algorithm	DSA	Civicioglu et al., 2012 [37]
Backtracking Search Optimization Algorithm	BSA	Civicioglu et al., 2013 [38]
Stochastic Fractal Search	SFS	Salimi et al., 2015 [39]
Synergistic Fibroblast Optimization	SFO	Dhivyaprabha et al., 2018 [40]
Wildebeests Herd Optimization	WHO	Motevali et al., 2019 [41]
Learner Performance based Behavior Algorithm	LPB	Rahman et al., 2021 [42]

Human-based algorithms are primarily derived from various aspects of human behavior, including teaching, social interaction, learning processes, emotional responses, and management practices. Notable examples of such algorithms encompass Teaching–Learning-Based Optimization (TLBO) [43], Group Search Optimizer (GSO) [44], Colliding Bodies Optimization (CBO) [45], League Championship Algorithm (LCA) [46], and Queuing Search Algorithm (QSA) [47], among others, as illustrated in Table 2.

Table 2. A brief review of human-based algorithms.

Algorithms	Abbreviations	Authors and Year
Imperialist Competitive Algorithm	ICA	Atashpaz-Gargari et al., 2007 [36]
Human-Inspired Algorithm	HIA	Zhang et al., 2009 [48]
League Championship Algorithm	LCA	Kashan et al., 2014 [46]
Teaching–Learning-Based Optimization	TLBO	Rao et al., 2011 [43]
Anarchic Society Optimization	ASO	Shayeghi et al., 2012 [49]
Human Mental Search	HMS	Mousavirad et al., 2017 [50]
Volleyball Premier League	VPL	Moghdani et al., 2018 [51]
Gaining Sharing Knowledge	GSK	Mohamed et al., 2020 [52]
Coronavirus Herd Immunity Optimizer	CHIO	Al-Betar et al., 2021 [53]
Ali baba and the Forty Thieves	AFT	Braik et al., 2022 [54]

Physics-based algorithms, such as Simulated Annealing (SA) [55], Gravitational Local Search Algorithm (GLSA) [56], Central Force Optimization (CFO) [57], and others in Table 3, are based on physics and chemistry and are mainly derived from the physical rules and chemical reactions in the universe [55].

Table 3. A brief review of physics-based algorithms.

Algorithms	Abbreviations	Authors and Year
Simulated Annealing	SA	Kirkpatrick et al., 1983 [55]
Variable Neighborhood Search	VNS	Mladenović et al., 1997 [58]
Big Bang–Big Crunch	BB-BC	Erol et al., 2006 [59]
Central Force Optimization	CFO	Formato et al., 2007 [57]
Gravitational Search Algorithm	GSA	Rashedi et al., 2009 [60]
Black Hole Algorithm	BHA	Hatamlou et al., 2013 [61]
Colliding Bodies Optimization	CBO	Kaveh et al., 2014 [45]
Lightning Search Algorithm	LSA	Shareef et al., 2015 [62]
Multi-Verse Optimizer	MVO	Mirjalili et al., 2016 [63]
Thermal Exchange Optimization	TEO	Kaveh et al., 2017 [64]
Equilibrium Optimizer	EO	Faramarzi et al., 2020 [65]

Swarm Intelligence (SI) algorithms are designed to achieve the global optimal solution by simulating swarm intelligence [66]. In these algorithms, each group is representative of a biological population. These populations are capable of executing tasks that individual members cannot accomplish independently, as exemplified by the methodologies. These algorithms leverage the cooperative behaviors exhibited by individuals within the population. The SI techniques outlined in Table 4 draw inspiration from the hunting and movement patterns observed in natural biological systems [67]. The process initiates with a random initialization of particles, which subsequently engage in a search for the global optimal solution within the designated search space. The core of SI algorithms is the combined concepts of exploration and exploitation [68,69]. Given that the optimal solution may be located anywhere within the search space, exploration involves a comprehensive examination of this space [70]. Generally, SI algorithms strive to achieve an optimal balance between exploitation and exploration.

Table 4. A brief review of swarm-intelligence-based algorithms.

Algorithms	Abbreviations	Authors and Year
Ant Colony Optimization	ACO	Dorigo et al., 1991 [9]
Particle Swarm Optimization	PSO	Kennedy et al., 1995 [26]
Firefly Algorithm	FA	Yang Xin-She, 2009 [71]
Fruit Fly Optimization	FOA	Pan Wen-Tsao, 2012 [72]
Ant Lion Optimizer	ALO	Mirjalili 2015 [73]
Tree-Seed Algorithm	TSA	Kiran, 2015 [74]
Dragonfly Algorithm	DA	Mirjalili et al., 2016 [75]
Whale Optimization Algorithm	WOA	Mirjalili et al., 2016 [76]
Grasshopper Optimization Algorithm	GOA	Saremi et al., 2017 [77]
Salp Swarm Algorithm	SSA	Mirjalili et al., 2017 [78]
Butterfly Optimization Algorithm	BOA	Arora et al., 2019 [24]
Bald Eagle Search Algorithm	BES	Alsattar et al., 2020 [79]
Harris Hawks Optimizer	HHO	Abualigah et al., 2021 [80]
Red Fox Optimizer	RFO	Połap et al., 2021, [81]
Dingo Optimization Algorithm	DOA	Bairwa et al., 2021 [82]
Chameleon Swarm Algorithm	CSA	Braik, 2021 [83]
Reptile Search Algorithm	RSA	Abualigah et al., 2022 [22]
White Shark Optimizer	WSO	Braik Malik et al., 2022 [84]

2.2. *Multi-Layer Perceptron (MLP)*

MLP is a fundamental architecture, shown in Figure 1, typically comprising three components: an input layer, one or more hidden layers, and an output layer. Neurons in each layer are interconnected through weighted connections, facilitating the flow of information across the network. The key characteristics of the MLP include the following:

- Feedforward Architecture: Information flows from the input layer through the hidden layers to the output layer without any feedback connections.
- Non-linear Activation Functions: Each neuron typically employs an activation function, such as ReLU, sigmoid, or tanh, to introduce non-linearity, enabling the network to learn complex functional relationships.
- Backpropagation Training: The network is trained using the backpropagation algorithm, which calculates gradients to update weights, thereby minimizing the discrepancy between the predicted outputs and the actual target values.

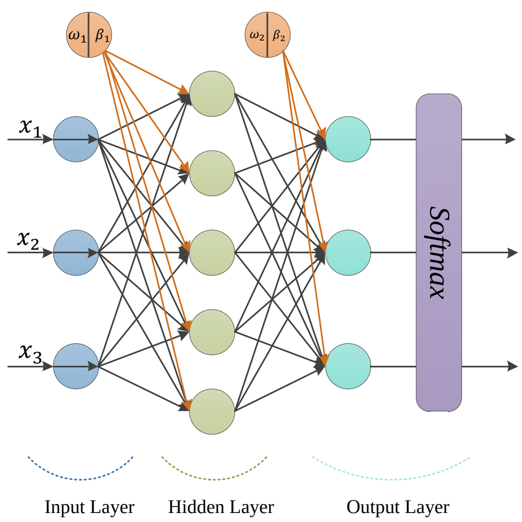


Figure 1. A simple Multilayer Perceptron (MLP) model.

2.3. Enhancing MLP Optimization Using Meta-Heuristic Optimization Methods

Meta-heuristic optimization methods offer a promising approach for optimizing MLPs by effectively navigating the complex parameter space and addressing the limitations in traditional optimization techniques. However, due to the high-dimensional and non-convex nature of the weight space, traditional methods like gradient descent may struggle with issues such as slow convergence and local optima. Meta-heuristic algorithms, including PSO [85], GA [86], and DE [87], can overcome the above challenges by leveraging their global search capabilities. These algorithms use population-based search processes that maintain multiple potential solutions and iteratively refine them, which enhances exploration and reduces the likelihood of getting trapped in local minima. Moreover, meta-heuristic algorithms can be adapted to diverse MLP architectures and problem requirements through flexible parameter settings and dynamic search strategies. This adaptability allows these methods to adjust the complexity for different tasks, such as classification or regression, and provides a generalized approach for high-dimensional, complex data.

In summary, meta-heuristic optimization techniques present considerable benefits in the training of MLPs. These approaches are proficient in performing effective global searches, display adaptability to various problems, and are resilient in avoiding local optima. Although the computational demands of these methods can be substantial, their ability to deliver high-quality solutions highlights their importance in optimizing MLPs. This is particularly relevant in fields that require robust, scalable, and efficient machine-learning models, further emphasizing their crucial role in advancing optimization methodologies.

3. Besiege and Conquer Algorithm (BCA)

3.1. Inspiration

Besiege strategy. An essential strategy in force maneuvering involves coordinating frontline units to attack the enemy's flanks or rear, creating a siege scenario [88]. Besiege tactics can be classified by scale (strategic, battle, and tactical) and style (one-wing, two-wing, four-breadth, and vertical besiegement). The goal is to isolate and immobilize the enemy, enabling a decisive outcome. The besiege duration is determined by strategic plans, objectives, and battlefield conditions. Initially, when enemy defenses are strong, the main forces launch a concentrated assault, while smaller units target weak points. During the later stages, it becomes a war of attrition, with coordinated attacks from all directions—east, west, north, and south, as shown in Figure 2.

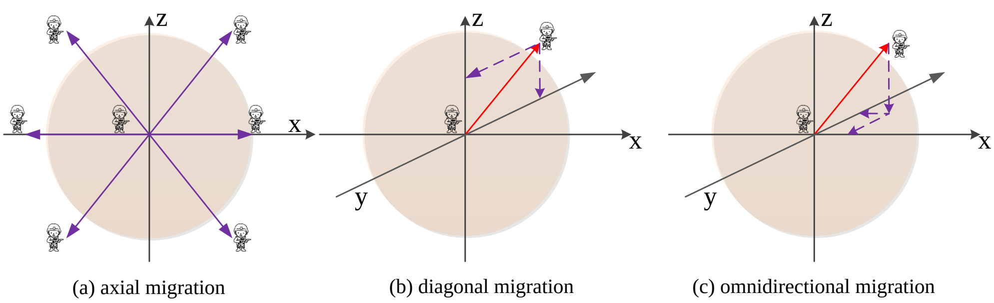


Figure 2. Soldier's besiege behavior in three-dimensional space.

Conquer strategy. Conquering the enemy is a core war strategy, involving coordinated attacks to establish order across all units [89]. Optimal force deployment aims to maximize battlefield effectiveness and utilize resources efficiently. The central strategy focuses on seizing the battlefield initiative by engaging enemy strengths and exploiting weaknesses. This approach involves dividing forces strategically and dispersing them to mount centripetal attacks from multiple directions [90]. By effectively timing and positioning breakthroughs, units can disrupt the enemy's influence and control on the battlefield.

Approach strategy. The approach strategy in warfare enables troops to advance effectively, securing critical terrain and targets to gain battlefield initiative. By swiftly occupying strategic positions and controlling advantageous areas, this strategy disrupts enemy defenses, disrupts plans, and forces the enemy into a passive stance. Various tactical maneuvers, such as penetration, flanking, and encirclement, increase unpredictability, supporting the execution of a successful conquest strategy. This approach maximizes speed, flexibility, and concentration of forces to exploit enemy weaknesses, achieving both tactical and strategic objectives.

Balance strategy. The balance strategy in warfare aims to optimize resource allocation and ensure a stable equilibrium between offense and defense, short- and long-term goals, and local and overall objectives. By diversifying tactics and deploying forces across multiple points, this approach reduces dependency on a single tactic and minimizes the risk of total failure due to localized losses. It emphasizes sustained combat capability, allowing commanders to respond flexibly and adjust across different battlefields. Key benefits include resource efficiency, enhanced adaptability, comprehensive combat readiness, sustained morale, and improved decision-making, all crucial for success in complex battlefield environments.

3.2. Initialization Phase

The initialization operation is the first step in space search based on the symmetry concept; the set of a randomly generated army is computed by Equation (1).

$$A_{i,d} = lb + (ub - lb) * rand \tag{1}$$

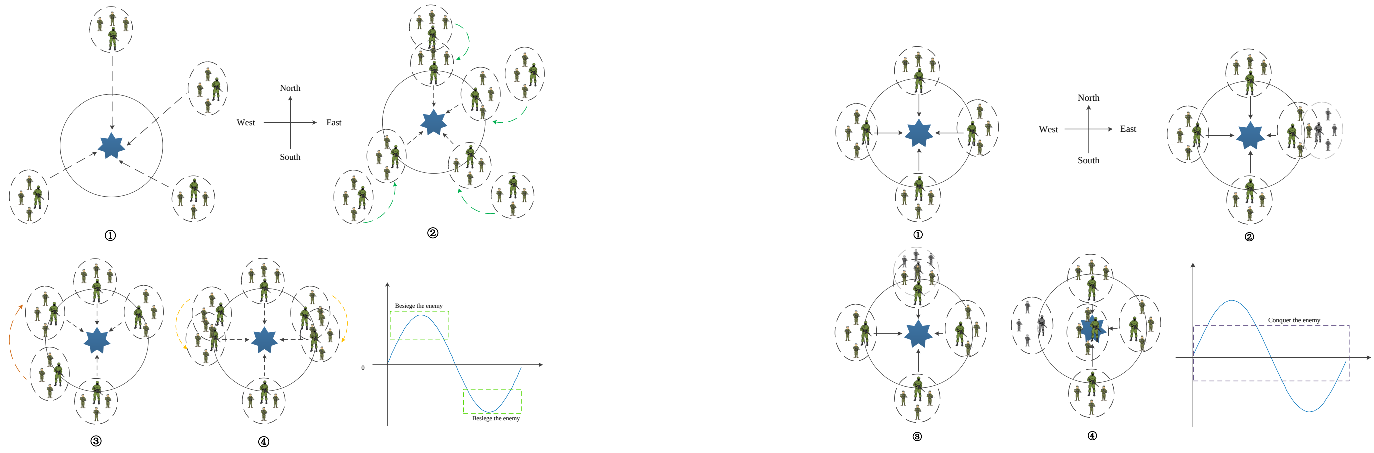
where *rand* is a random constant and *lb* and *ub* present the lower and upper bounds of the given search space, respectively. Parameter $A_{i,d}$ is denoted to the d_{th} dimension of the i_{th} army, and the random army solution in Equation (2) is generated stochastically.

$$A = \begin{bmatrix} A_{1,1} & .. & A_{1,m} & .. & A_{1,j-1} & .. & A_{1,d} \\ A_{2,1} & .. & A_{2,m} & .. & A_{2,j-1} & .. & A_{2,d} \\ .. & .. & .. & .. & & & \\ A_{i-1,1} & .. & A_{i-1,m} & .. & A_{i-1,j-1} & .. & A_{i-1,d} \\ A_{i,1} & .. & A_{i,m} & .. & A_{i,j-1} & .. & A_{i,d} \end{bmatrix} \tag{2}$$

There are several soldiers in each army, and the number of soldiers is set to *nSoldiers*. Soldiers are moved based on the best or a random army. Each army surrounds the enemy by dividing the army into scattered soldiers. Scattered soldiers form a unit, surrounded by a second-level scatter, and then disperses in the n_{th} level until it surrounds the enemy and launches an attack until the enemy is destroyed. The army has control over dispersed soldiers, forces formed by randomly distributed soldiers to keep moving toward their targets, and dispersing as they progress until they surround and attack the enemy.

3.3. Besiege and Conquer Strategies

The besiege mechanism of soldiers determines the updating direction of the army, that is, the direction of finding the optimal solution. The position approach of a soldier will also change when influenced by different rules. In the BCA, three soldier position besiege mechanisms are designed, as shown in Figure 3.



(**a**) (**b**)

Figure 3. Besiege and conquer mechanisms controlled by parameter k. (**a**) Besiege strategy from four directions: east, west, north, and south. (**b**) Conquer strategy from four directions: east, west, north, and south.

Migration mechanism. The purpose of this mechanism is to gradually approach the global optimal solution through concentrated search. The besiege mechanism mainly consists of surrounding the best army (discovered enemy), forming an updated circle around it. The generation position of soldiers can be computed by Equation (3).

$$S_{j,d}^{t+1} = B_d^t + |A_{r,d}^t - A_{i,d}^t| * k_1 \quad when \begin{cases} |\, k_1 \,| < 0.5, & Conquer \quad operator \\ 0.5 < |\, k_1 \,| < 1, & Besiege \quad operator \end{cases} \tag{3}$$

where $S_{j,d}^{t+1}$ is the j_{th} soldier of the d_{th} dimension with $(t+1)_{\text{th}}$ iterations, B_d^t is the current best army (discovered enemy) with t_{th} iterations, $A_{i,d}^t$ is the i_{th} army of the d_{th} dimension with t_{th} iterations, $A_{r,d}^t$ is a random army of the d_{th} dimension with t_{th} iterations, and k_1 is the cover coefficient.

Approach mechanism. Improving the diversity of global search to effectively balance exploration and exploitation, Equation (4) is adopted to generate new soldiers to facilitate the search for optimal solutions. This soldier position focuses on the best army and the current army, promoting soldiers to achieve the best army and speeding up convergence capability. It reaches a nearby position of the best army by the parameter Besiege and Conquer Balance (*BCB*).

$$S_{j,d}^{t+1} = BCB^t * B_d^t + \alpha^t * A_{i,d}^t \tag{4}$$

where α^t is computed by Equation (5).

$$\alpha^t = 1 - BCB^t \tag{5}$$

Cover mechanism. Exploration and exploitation are two critical parts. This mechanism ensures that the BCA explores and exploits a significant amount of search space. The other strategy is computed by Equation (6) to increase the chance of finding the global optimal solution. Additionally, it can increase population diversity and exploration capability.

$$S_{j,d}^{t+1} = A_{r,d}^t + |A_{r,d}^t - A_{i,d}^t| * k_2 \tag{6}$$

where k_2 is the cover coefficient.

However, discarding the soldiers generated beyond the scope of the search space will reduce the diversity of the population and slow down the convergence speed. Therefore, considering these above problems, soldiers are randomly generated in the search space to compensate for the diversity loss and increase their exploration capability when soldiers cross the border. The randomly generated soldier is computed by Equation (7).

$$S_{j,d} = lb + (ub - lb) * rand \tag{7}$$

where *rand* is a random constant in the range of [0, 1]. To speed up the convergence rate, a trigonometric function (including *sine* and *cosine* functions) is added to disturb the soldiers' migration position. Since the trigonometric function changes in both positive and negative directions in the range, it promotes the diversity of soldiers in location migration and avoids local stagnation. The right of Figure 4 shows the wave pattern of trigonometric functions. As shown in Figure 4, the effects of the *sine* and *cosine* functions with the range in $[-1, 1]$ illustrate the direction of the soldier's migration location, whether away from or near the current best army. This mechanism ensures that the BCA explores and exploits a significant amount of search space. The parameters are implemented with Equations (8) and (9).

$$k_1 = sin(2 * pi * rand) \tag{8}$$

$$k_2 = cos(2 * pi * rand) \tag{9}$$

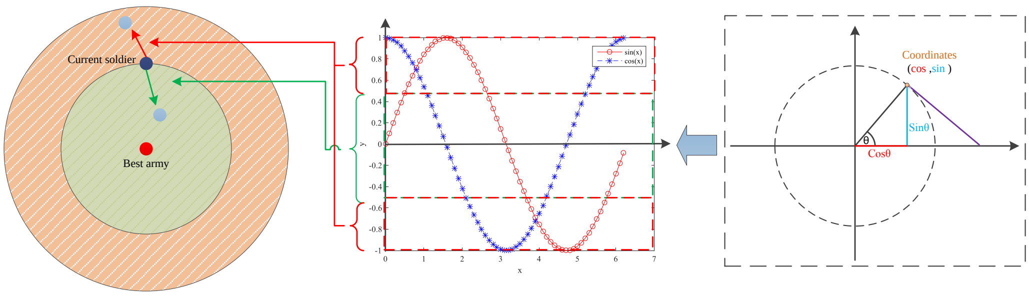


Figure 4. Besiege and conquer mechanisms with the influence of *sine* and *cosine* functions.

Unlike linear functions, $sin(2 * pi * rand)$ and $cos(2 * pi * rand)$ are mathematical functions that describe smooth repetitive oscillations. The values of the sine function are randomly distributed as a sinusoidal wave. This mechanism enhances the exploration capability to find the global optimum. k_1 and k_2 control the direction and length of the cover mechanism, which can balance the exploration and exploitation capabilities. $|k| < 0.5$ contributes to the conquer mechanism, and $0.5 < |k| < 1$ is conducive to the besiege mechanism. k_1 in Equation (3) controls the exploration direction, and k_2 in Equation (6) guides the exploitation direction. These two parameters control the approach strategy and influence the movement of the army to find the global optimum position.

3.4. Balance Strategy

Adopting which besiege or conquer mechanism is decided by the relationship between *rand* and *BCB*. The allocation of the besiege and conquer mechanisms to update the soldiers' positions are shown in Figure 5 . The design of the balance mechanism is conducive to achieving an equilibrium between exploration and exploitation capabilities. The selection of different strategies can help find the global optimal solution. Meanwhile, there is an 80% probability to further update the position of the soldiers with the best army, and a remaining 20% probability of using random soldiers as benchmarks to approach the soldiers. The two different strategies complement each other and can help avoid local stagnation and accelerate convergence. Equation (10) shows how to choose the approach strategy to attack the enemy in different circumstances. The practical cooperation of the two strategies can promote the capability of global search and increase its effectiveness in finding the optimal solution.

$$S_{j,d}^{t+1} = \begin{cases} B_d^t + |A_{r,d}^t - A_{i,d}^t| * k_1 & , \quad if \quad rand < BCB \\ A_{r,d}^t + |A_{r,d}^t - A_{i,d}^t| * k_2 & , \quad else \end{cases} \tag{10}$$

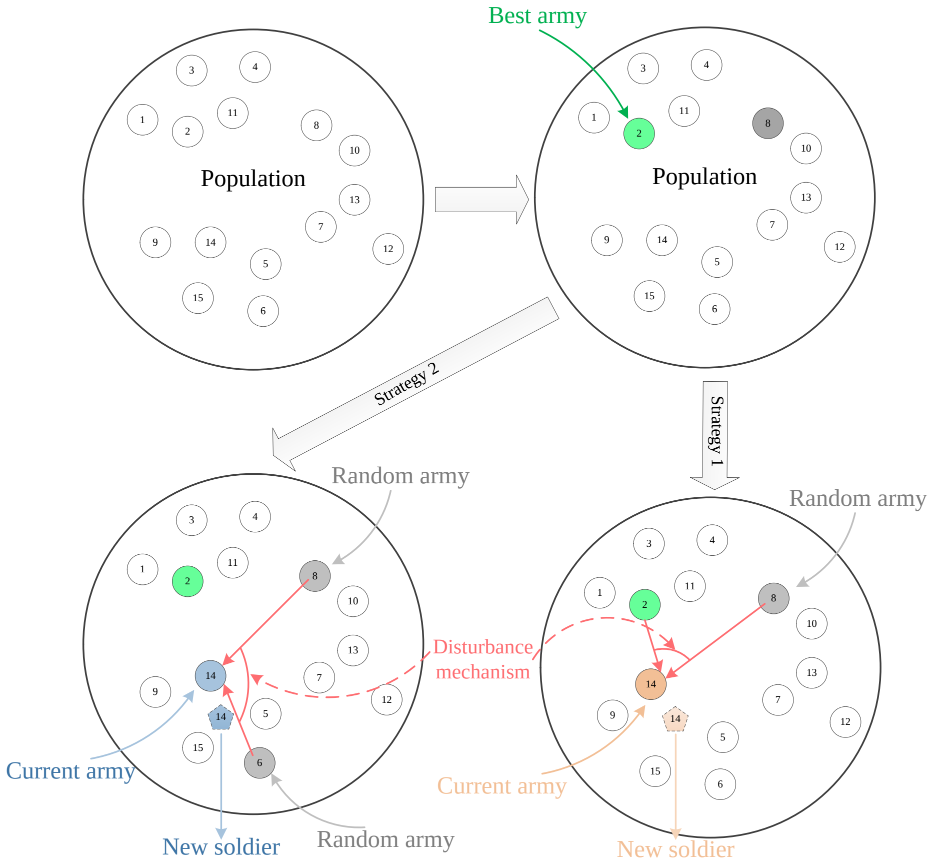


Figure 5. Allocation methods of the strategy selection mechanism.

3.5. Feedback Strategy

In the BCA, the parameter *BCB* is adjusted dynamically according to the distance between the current army and the best army to effectively achieve a balance between exploration and exploitation capabilities. *BCB* is used for balancing the behavior between besiege and conquer, that is, exploration and exploitation. According to the Pareto Principle [91], the parameter *BCB* is initially set to 0.8. *BCB* controls both the exploration and the exploitation capabilities. A higher value of *BCB* enhances a powerful local search and convergence speed, while a lower value of *BCB* results in slow convergence but a powerful global search. In other words, the BCA's exploration and exploitation capabilities are controlled by the *BCB*. The *BCB* in Equation (4) controls the approach strategy of soldiers, which influences the exploration ability and convergence speed. By adjusting the parameter of *BCB* through the feedback mechanism, the speed of finding the optimal solution can be accelerated. The BCA contains mechanisms that complement each other, and the relationship between these mechanisms is shown in Figure 6.

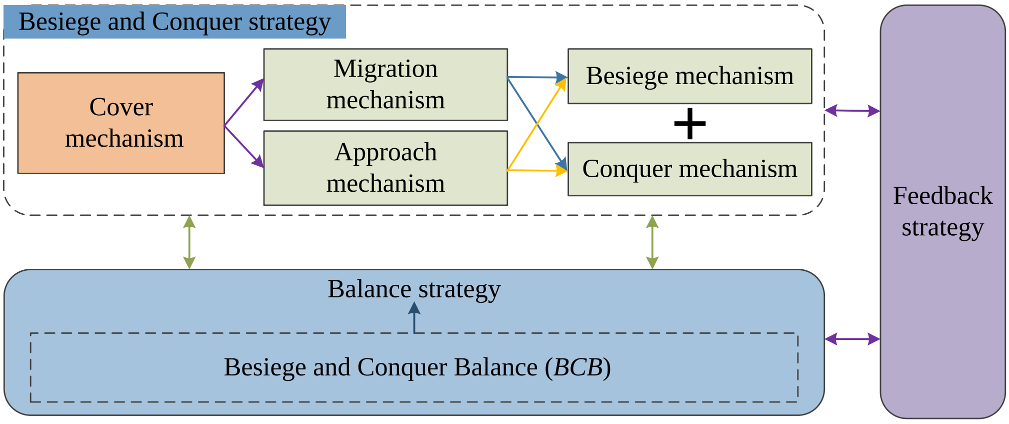


Figure 6. The cooperative relationship of these four mechanisms.

All of the parameters and explanations involved in this study are shown in Table 5.

Table 5. Notations

Notation	Meaning
$A_{i,d}$	The d_{th} dimension of the i_{th} army
$S_{j,d}^{t+1}$	The j_{th} soldier of the d_{th} dimension with $(t+1)_{\text{th}}$ iteration
B_d^t	The current best army (discovered enemy) with t_{th} iteration
$A_{r,d}^t$	A random army of the d_{th} dimension with t_{th} iteration
α^t	Regularization parameter
k_1, k_2	The cover coefficient
lb,ub	The lower and upper bound of the given search space
$nSoldiers$	The number of soldiers
BCB	Besiege and Conquer Balance

3.6. Computational Complexity

The computational complexity of the BCA involves definition, initialization, soldier evaluation, and soldier update. It is mainly influenced by the number of iterations (T), the problem dimension (D), the population (army) number (N), the number of soldiers ($nSoldiers$), and the function assessment's cost (c). The computational complexity can be given as follows. Note that the computational complexity of the initialization processes, with $\frac{N}{nSoldier}$, is $O(\frac{N}{nSoldier})$. The computational complexity of the updating process is $O(nSoldier \times T \times D)$. Therefore, the computational complexity of the BCA can be computed by Equation (11).

$$O(BCA) = O(problem.define) + O(intilization) + O(army.cost) + O(soldier.update) \quad (11)$$

where the computational complexities of the components of Equation (12) can be defined as follows:

- Problem definition is set as O(1).
- Initialization of the population demands $O(\frac{N}{nSoldier} \times D)$.
- Generation of soldiers demands $O(nSoldier \times D \times T)$.
- Evaluation of solutions demands $O(T \times \frac{c}{nSoldier} \times D)$.

$$O(BCA) = O(1 + \frac{N \times D}{nSoldier} + T \times nSoldier \times D + \frac{T \times c \times N}{nSoldier}) \quad (12)$$

The flow of the BCA is shown in Figure 7, and its pseudo code is shown in Algorithm 1.

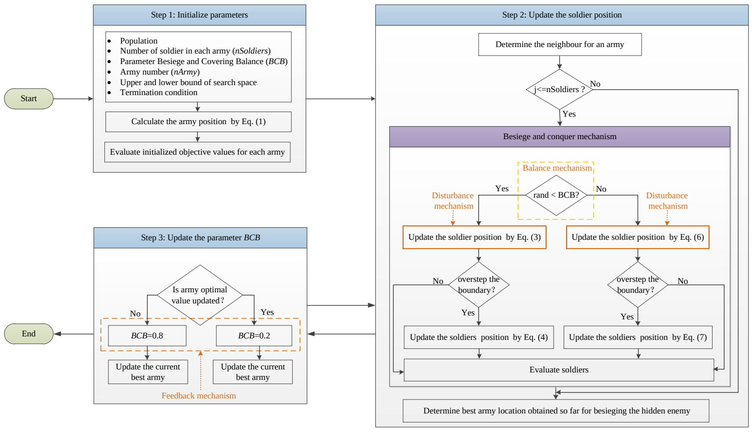


Figure 7. The workflow of the BCA.

Algorithm 1 BCA: Besiege and Conquer Algorithm

Step 1: Initialize parameters:
  1.1 Initialize population and the number of soldiers (*nSoldiers*).
  1.2 Put up Besiege and Conquer Balance (*BCB*) parameter.
  1.3 Set the iteration number (*MaxIteration*).
  1.4 Set the number of army (*nArmy*).
  1.5 Initialize the upper bound (*ub*) and lower bound (*lb*) of the search space.
  1.6 Determine the termination condition (*MaxIteration*).
  1.7 Initialize the army position through Equation (1).
  1.8 Evaluate initialized objective values for each army.
Step 2:
  While $t < MaxIter$
  2.1 **For** i: $nArmy$
    2.1.1 Determination of the neighbor for i_{th} army
    2.1.2 **For** j: $nSoldiers$
        For d: dim
          If $rand <= BCB$
            Update the position of the i_{th} soldier by Equation (3)
            If $S_{j,d} > ub \quad || \quad S_{j,d} < lb$
              Update the position of the i_{th} soldier by Equation (4)
            End If
          Else
            Update the position of the i_{th} soldier by Equation (6)
            If $(S_{j,d} > ub \quad || \quad S_{j,d} < lb)$
              Update the position of the i_{th} soldier by Equation (7)
            End If
          End For
        End For
    End For
    2.1.3 Evaluate soldiers' objectives in each army
  End While
Step 3:
  3.1 Determine the best army location obtained so far.
  3.2 Judge whether the army optimal value is updated.
  3.3 **If** army optimal value is updated
      $BCB(t+1) = 0.2;$
    Else
      $BCB(t+1) = 0.8;$
    End If
Step 4: Update the best army.

---

4. Experiment Setting

The experiment codes are executed in the Matlab R2015b environment under the Windows 10 operating system; all simulations were performed on a computer with Intel Core(TM) i3-6100 CPU @ 3.70 GHz, and its memory was 8 GB.

4.1. Experimental Test Functions

The IEEE CEC 2017 benchmark function test set, which consists of a diverse collection of benchmark functions, including uni-model, multi-model, hybrid, and composition functions, is commonly employed to assess and verify the performance of the BCA. These functions are carefully designed to test the robustness and versatility of optimization methods across a wide range of problem complexities.

4.2. Comparative Algorithms

The performance of the proposed algorithm and its superiority compared to other algorithms are verified by comparison with classical, popular, and recent algorithms, such as INFO [21], RSA [22], SOMA T3A [23], BOA [24], GWO [8], DE [25], GA [10], and PSO [26]. The selected algorithms are introduced in detail as follows. These algorithms represent a variety of optimization approaches, such as EAs, SI, and DE, which have been widely applied to similar optimization problems. By comparing with these algorithms, we aim to demonstrate the effectiveness and efficiency of our method across different optimization paradigms.

For all comparative algorithms, the population number is set to 30 and the maximum number of iterations is set to 500. For different comparison methods, the hyper-parameter settings of the comparison methods mainly refer to the proposed references, and their parameters are set in Table 6. In addition, each algorithm is executed 30 times to ensure statistics. "Mean" represents the mean of the best value obtained by the algorithm, "Std." represents the standard deviation of the best value, and "Median" represents the median of the best value. In addition, the Friedman ranking test is used to obtain the average and final ranking. The Wilcoxon signed-rank test is used to check the algorithm's effectiveness and whether or not it is significantly different from other algorithms.

Table 6. List of parameter settings for comparative algorithms.

Algorithms	Parameter	Value	Reference
BCA	BCB $nSoldiers$	0.8 3	
INFO	No hyperparameter settings		[21]
RSA	Evolutionary sense Sensitive parameter controlling the exploration accuracy Sensitive parameter controlling the exploitation accuracy	$2 \times randn \times (1 - (iter/maxiter))$ 0.005 0.1	[22]
GWO	a	Liner from 2 to 0	[8]
BOA	Power exponent Sensory modality Probability switch (p)	0.1 0.01 0.8	[24]
SOMA T3A	Step PRT	$0.2 + 0.05 \times cos(2 \times \pi \times FEs_{count}/FEs_{Max})$ $0.05 + 0.95 \times (FEs_{count}/FEs_{Max})$	[23]

Table 6. *Cont.*

Algorithms	Parameter	Value	Reference
PSO	Cognitive component Social component	2 2	[26]
DE	Scale factor primary Scale factor secondary Scale factor secondary Crossover rate	0.6 0.5 0.3 0.8	[25]
GA	CrossPercent MutatPercent ElitPercent	70% 20% 10%	[10]

5. Experimental Results and Discussion

5.1. Computational Complexity Analysis

Computational complexity is also one of the criteria for evaluating algorithm performance. The computational complexity of traditional algorithms is affected by the number of iterations (T), the problem dimension (D), the population number (N), and the function assessment's cost (c). The computational complexities of the components can be defined as follows:

1. Initialization of problem definition demands $O(1)$.
2. Initialization of population demands $O(N \times D)$.
3. Assessment of the cost function demands $O(T \times c \times N)$.
4. Evaluation of solutions demands $O(T \times N \times D)$.

Thus, the general computational complexity of traditional algorithm can be expressed in Equation (13).

$$O(Algorithm) = O(1 + N \times D + T \times c \times N \times D + T \times N \times D) \tag{13}$$

The computational complextity of the BCA can be computed: $O(BCA) = O(1 + \frac{N \times D}{nSoldier} + T \times nSoldier \times D + \frac{T \times c \times N}{nSoldier})$.

The $\frac{N \times D}{nSoldier} <= N \times D$, $\frac{T \times c \times N}{nSoldier} <= T \times c \times N$, and $T \times nSoldier \times D <= T \times N \times D$ demonstrate that the BCA needs less computational cost when compared with traditional swarm intelligence algorithms.

5.2. Parameters Sensitivity

The BCA's optimization framework proposed in this study models the concept of multiple soldiers operating within each army to enhance the search process. Specifically, the parameter *nSoldiers* determines the number of soldiers assigned to each army, playing a pivotal role in guiding the army's progress toward updating the optimal global solution. This mechanism leverages the collaborative behaviors of soldiers to enhance the exploration and exploitation capabilities, helping to avoid local stagnation and improving the efficiency of convergence. Increasing the number of soldiers per army diversifies the search, allowing exploration of a broader solution space and mitigating the risk of being trapped in local optima. However, this comes at the cost of computational resources, as a larger number of soldiers requires additional iterations and evaluations. Therefore, while a more substantial number of soldiers theoretically enhances the robustness, it also introduces diminishing returns due to the escalating computational overhead. As detailed in Table 7, the experimental results demonstrate that different values of *nSoldiers* significantly impact the BCA's ability to find the global optimum. When the number of soldiers is too low, the BCA risks insufficient exploration, leading to premature convergence and suboptimal solutions. Conversely, when the number of soldiers is too high, the computation time and resource consumption increase disproportionately, leading to inefficiency without significant gains in solution quality. Through extensive experimentation, it was found that the optimal configuration occurs when *nSoldiers* is set to three (i.e., three soldiers per army). This configuration strikes an effective balance between exploration and exploitation, achieving high-quality solutions while minimizing computational costs. The results underscore that this choice avoids local stagnation, accelerates convergence, and ensures computational efficiency, making it the most effective setup for the algorithm in the context of the problems tested.

5.3. Exploitation Analysis

The unimodal functions (F_1 and F_2) can test the exploitation ability of the BCA, and it can be seen from Table 8 that the BCA can find the best solution on F_1 and F_2 functions compared with other algorithms. Tables 9–11 show the obtaining results of the BCA and other comparative algorithms. It can be seen that the BCA can effectively solve single objective problems and has a stronger exploitation capability than other algorithms. This is due to two different strategy selection mechanisms in the BCA.

5.4. Exploration Analysis

The exploration capability can be verified by multi-model functions (F_3 to F_{19}); experimental results show that the BCA is superior to the other algorithms in Table 8. "*w*" means that the BCA is superior to (win) the comparative algorithms, "*l*" means that the BCA is weaker (lose) than the comparative algorithms, and "*e*" means that the BCA is equal to the comparative algorithms. It can be seen from Tables 9–11 that the BCA can obtain the best global solution. Compared to other algorithms, the BCA can quickly find the global optimal solution and has a strong exploration capability. The strong exploration capability benefits from the cover mechanism, which helps to jump out from local solutions, generate new soldiers' positions, increase the diversity of global search, and find the optimal solution in multimodal functions.

5.5. Local Minima Avoidance Analysis

Local minima avoidance is a standard characteristic to evaluate algorithms. It can be seen from Figures 8–10 that the BCA, compared to other comparison algorithms, shows more rapid convergence and has a stronger capability of avoiding local stagnation. According to Table 8, the BCA is superior to other algorithms in terms of hybrid and composition benchmark functions with 30D, 50D, and 100D; hence, it shows that the BCA has a good balance between exploration and exploitation. Due to the cooperation between balance and feedback mechanisms, the search capability can be improved adaptively.

Table 7. Results of the various *nSoldiers* with 10D, 30D, 50D, and 100D.

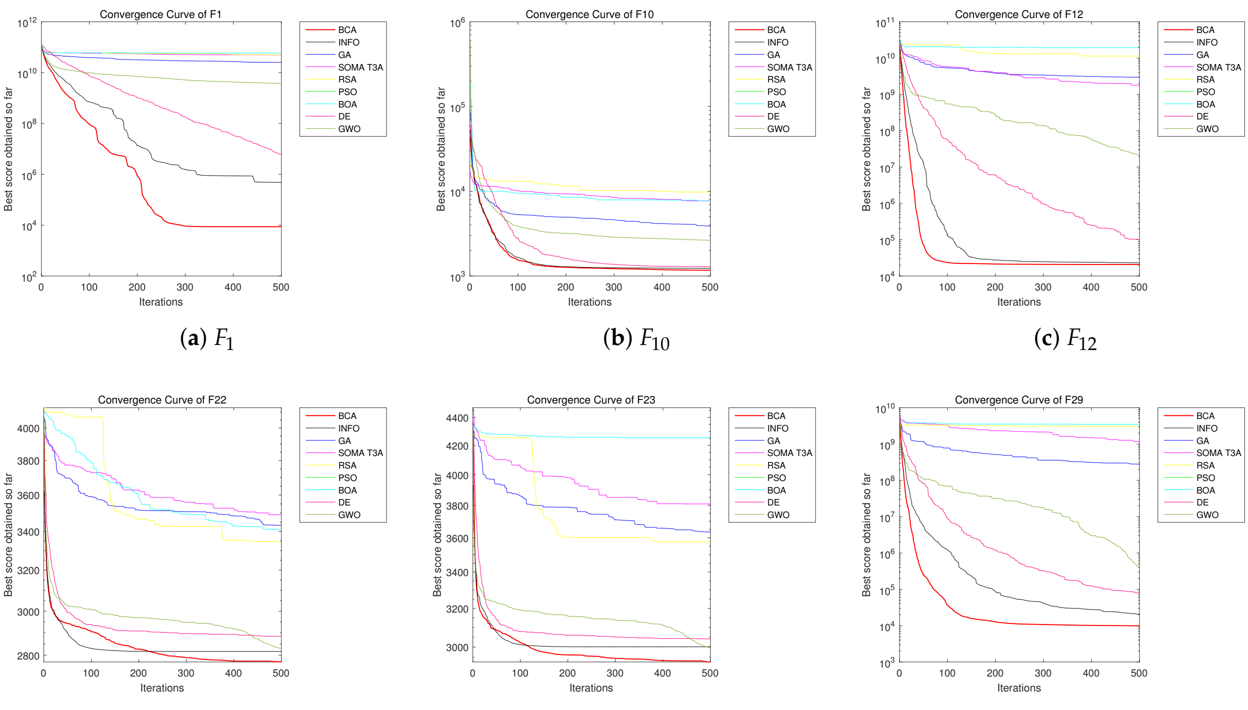
Function		10D				Function		30D			
		***nSoldier* = 2**	***nSoldier* = 3**	***nSoldier* = 4**	***nSoldier* = 5**			***nSoldier* = 2**	***nSoldier* = 3**	***nSoldier* = 4**	***nSoldier* = 5**
Unimodal Functions	F_1	3.0846×10^3	2.7349×10^3	7.7035×10^4	1.3394×10^7	Unimodal Functions	F_1	5.4825×10^3	5.8610×10^3	1.1128×10^9	4.2021×10^9
	F_2	3.0949×10^2	3.0000×10^2	2.4666×10^3	1.0033×10^4		F_2	9.5782×10^4	9.1813×10^4	8.8295×10^4	9.6163×10^4
Multimodal Functions	F_3	4.0602×10^2	4.0906×10^2	4.3333×10^2	4.2246×10^2	Multimodal Functions	F_3	4.8535×10^2	5.1259×10^2	7.6105×10^2	1.4731×10^3
	F_4	5.1145×10^2	5.1430×10^2	5.2292×10^2	5.2810×10^2		F_4	6.9397×10^2	6.3094×10^2	6.3101×10^2	6.8227×10^2
	F_5	6.0000×10^2	6.0032×10^2	6.0344×10^2	6.1100×10^2		F_5	6.0028×10^2	6.0157×10^2	6.2252×10^2	6.3103×10^2
	F_6	7.3803×10^2	7.2268×10^2	7.3195×10^2	7.4333×10^2		F_6	9.6636×10^2	9.4231×10^2	1.0266×10^3	1.1529×10^3
	F_7	8.1197×10^2	8.1401×10^2	8.2305×10^2	8.2462×10^2		F_7	1.0044×10^3	9.1677×10^2	9.2094×10^2	9.5984×10^2
	F_8	9.0035×10^2	9.0559×10^2	1.0147×10^2	1.0088×10^3		F_8	9.6527×10^2	1.2018×10^3	3.4503×10^3	4.0112×10^3
	F_9	1.6642×10^3	1.5155×10^3	1.6689×10^3	1.7918×10^3		F_9	8.6455×10^3	8.0815×10^3	6.6932×10^3	5.7858×10^3
Hybrid Functions	F_{10}	1.1060×10^3	1.1169×10^3	1.1516×10^3	1.6462×10^3	Hybrid Functions	F_{10}	1.2447×10^3	1.1850×10^3	3.3216×10^3	4.0884×10^3
	F_{11}	1.4101×10^4	1.8262×10^4	7.6760×10^5	2.7037×10^6		F_{11}	5.6230×10^5	9.7535×10^5	1.2471×10^7	1.2526×10^8
	F_{12}	6.3975×10^3	1.0754×10^4	8.7964×10^3	1.2086×10^4		F_{12}	2.1185×10^4	2.3155×10^4	1.7233×10^6	3.0925×10^7
	F_{13}	1.4446×10^3	1.4436×10^3	1.4575×10^3	2.2201×10^3		F_{13}	4.5091×10^4	7.8492×10^4	1.0781×10^6	1.6536×10^6
	F_{14}	1.6004×10^3	1.5665×10^3	2.0493×10^3	6.5510×10^3		F_{14}	1.2666×10^4	1.1432×10^4	1.3389×10^4	3.9439×10^5
	F_{15}	1.6235×10^3	1.6946×10^3	1.8341×10^3	1.8097×10^3		F_{15}	3.2972×10^3	2.8115×10^3	2.6830×10^3	2.8585×10^3
	F_{16}	1.7339×10^3	1.7370×10^3	1.7720×10^3	1.7691×10^3		F_{16}	2.0290×10^3	2.0641×10^3	2.1830×10^3	2.3095×10^3
	F_{17}	7.6635×10^3	7.5756×10^3	9.1449×10^3	8.5387×10^3		F_{17}	1.8141×10^6	1.4815×10^6	3.4402×10^6	5.0139×10^6
	F_{18}	2.0163×10^3	1.9412×10^3	2.9480×10^3	3.3431×10^3		F_{18}	1.1171×10^4	1.4182×10^4	1.5584×10^4	6.6029×10^5
	F_{19}	2.0093×10^3	2.0347×10^3	2.1076×10^3	2.1114×10^3		F_{19}	2.4423×10^3	2.4354×10^3	2.5215×10^3	2.5333×10^3
Composition Functions	F_{20}	2.3135×10^3	2.2993×10^3	2.3069×10^3	2.3106×10^3	Composition Functions	F_{20}	2.5061×10^3	2.4294×10^3	2.4247×10^3	2.4567×10^3
	F_{21}	2.3542×10^3	2.2991×10^3	2.4485×10^3	2.3745×10^3		F_{21}	3.6015×10^3	5.3314×10^3	5.1472×10^3	5.8761×10^3
	F_{22}	2.6134×10^3	2.6172×10^3	2.6336×10^3	2.6467×10^3		F_{22}	2.7878×10^3	2.7657×10^3	2.8392×10^3	2.9193×10^3
	F_{23}	2.7260×10^3	2.7278×10^3	2.7333×10^3	2.7556×10^3		F_{23}	3.0183×10^3	2.9604×10^3	2.9892×10^3	3.0420×10^3
	F_{24}	2.9366×10^3	2.9315×10^3	2.9410×10^3	2.9384×10^3		F_{24}	2.8892×10^3	2.8986×10^3	3.0963×10^3	3.2930×10^3
	F_{25}	3.1356×10^3	3.3707×10^3	3.1790×10^3	3.2100×10^3		F_{25}	4.7202×10^3	4.3892×10^3	5.4223×10^3	6.7499×10^3
	F_{26}	3.1030×10^3	3.1240×10^3	3.1397×10^3	3.1405×10^3		F_{26}	3.2282×10^3	3.2482×10^3	3.3226×10^3	3.3470×10^3
	F_{27}	3.3097×10^3	3.2913×10^3	3.3675×10^3	3.4173×10^3		F_{27}	3.2279×10^3	3.2397×10^3	3.5564×10^3	3.9394×10^3
	F_{28}	3.2144×10^3	3.2056×10^3	3.2466×10^3	3.2717×10^3		F_{28}	3.7674×10^3	3.7521×10^3	4.0859×10^3	4.1870×10^3
	F_{29}	2.0827×10^5	2.8017×10^5	6.0841×10^5	1.0338×10^6		F_{29}	1.2610×10^4	1.0956×10^4	1.5343×10^5	2.0374×10^7

Table 7. *Cont.*

Function		50D				Function		100D			
		***nSoldier* = 2**	***nSoldier* = 3**	***nSoldier* = 4**	***nSoldier* = 5**			***nSoldier* = 2**	***nSoldier* = 3**	***nSoldier* = 4**	***nSoldier* = 5**
Unimodal Functions	F_1	2.2155×10^6	2.0239×10^5	6.7977×10^9	2.1753×10^{10}	Unimodal Functions	F_1	4.2844×10^9	2.5720×10^9	6.2050×10^{10}	1.2140×10^{11}
	F_2	2.4021×10^5	2.4341×10^5	2.3537×10^5	2.3822×10^5		F_2	6.4573×10^5	6.3313×10^5	6.5750×10^5	6.7979×10^5
Multimodal Functions	F_3	5.8195×10^2	5.9412×10^2	1.8270×10^3	3.6785×10^3	Multimodal Functions	F_3	1.5671×10^3	1.1256×10^3	9.3277×10^3	2.1264×10^4
	F_4	9.0606×10^2	8.1345×10^2	7.7596×10^2	8.8940×10^2		F_4	1.6604×10^3	1.4450×10^3	1.3806×10^3	1.6253×10^3
	F_5	6.0550×10^2	6.0710×10^2	6.3175×10^2	6.4129×10^2		F_5	6.3002×10^2	6.2828×10^2	6.4901×10^2	6.6447×10^2
	F_6	1.2670×10^3	1.2209×10^3	1.5840×10^3	2.0320×10^3		F_6	2.4678×10^3	2.5133×10^3	3.8345×10^3	5.0119×10^3
	F_7	1.2240×10^3	1.0672×10^3	1.1088×10^3	1.1795×10^3		F_7	1.9327×10^3	1.7600×10^3	1.7474×10^3	1.9928×10^3
	F_8	4.0432×10^3	5.3850×10^3	1.3424×10^4	1.9343×10^4		F_8	3.5799×10^4	3.8626×10^4	6.4847×10^4	7.6842×10^4
	F_9	1.5238×10^4	1.5087×10^4	1.2615×10^4	1.1332×10^4		F_9	3.2727×10^4	3.2433×10^4	3.1304×10^4	2.9131×10^4
Hybrid Functions	F_{10}	1.8119×10^3	1.5757×10^3	5.9873×10^3	1.4609×10^4	Hybrid Functions	F_{10}	1.4601×10^5	1.4255×10^5	9.7561×10^4	1.1785×10^5
	F_{11}	7.5257×10^6	5.7793×10^6	9.4934×10^8	3.9893×10^9		F_{11}	3.5939×10^8	1.1284×10^8	1.2216×10^{10}	2.7504×10^{10}
	F_{12}	9.1517×10^3	1.2298×10^4	1.1167×10^8	7.9810×10^8		F_{12}	1.5045×10^4	1.3561×10^5	6.7898×10^8	3.4474×10^9
	F_{13}	2.7398×10^5	5.6463×10^5	3.3394×10^6	6.5692×10^6		F_{13}	4.8115×10^6	2.7004×10^6	1.8884×10^7	2.8560×10^7
	F_{14}	7.7526×10^3	8.2671×10^3	4.3185×10^6	3.8452×10^7		F_{14}	7.3906×10^3	7.3133×10^3	8.2907×10^7	8.4903×10^8
	F_{15}	5.0489×10^3	4.2202×10^3	3.4537×10^3	3.9827×10^3		F_{15}	1.1266×10^4	1.0296×10^4	7.4809×10^3	8.4029×10^3
	F_{16}	3.9683×10^3	3.6335×10^3	3.3731×10^3	3.5296×10^3		F_{16}	7.8248×10^3	7.4606×10^3	6.8917×10^3	1.5298×10^4
	F_{17}	8.3843×10^6	4.4784×10^6	9.7898×10^6	2.2358×10^7		F_{17}	1.9926×10^7	1.3042×10^7	1.8313×10^7	3.738×10^7
	F_{18}	1.9051×10^4	1.5268×10^4	8.9817×10^4	2.3109×10^7		F_{18}	1.1467×10^4	9.5134×10^3	7.4317×10^7	6.1740×10^8
	F_{19}	4.0439×10^3	3.6484×10^3	3.3779×10^3	3.487×10^3		F_{19}	7.6536×10^3	7.7604×10^3	7.0747×10^3	7.2755×10^3
Composition Functions	F_{20}	2.7426×10^3	2.6121×10^3	2.6046×10^3	2.6857×10^3	Composition Functions	F_{20}	3.4163×10^3	3.2773×10^3	3.3503×10^3	3.5862×10^3
	F_{21}	1.5769×10^4	1.5592×10^4	1.3809×10^4	1.3377×10^4		F_{21}	3.5191×10^4	3.492×10^4	3.0837×10^4	2.9716×10^4
	F_{22}	3.0911×10^3	3.0069×10^3	3.2211×10^3	3.3350×10^3		F_{22}	3.8555×10^3	3.5242×10^3	4.0349×10^3	4.1911×10^3
	F_{23}	3.3510×10^3	3.2801×10^3	3.3271×10^3	3.4428×10^3		F_{23}	4.4885×10^3	4.1094×10^3	4.9017×10^3	5.3913×10^3
	F_{24}	3.0758×10^3	3.0885×10^3	4.1700×10^3	5.9169×10^3		F_{24}	4.3383×10^3	3.8788×10^3	1.0377×10^4	1.6568×10^4
	F_{25}	7.6214×10^3	6.0522×10^3	8.8032×10^3	1.0473×10^4		F_{25}	1.7191×10^4	1.4683×10^4	2.1762×10^4	2.7959×10^4
	F_{26}	3.3861×10^3	3.4761×10^3	3.8879×10^3	4.1217×10^3		F_{26}	3.7895×10^3	3.6972×10^3	4.3947×10^3	4.7517×10^3
	F_{27}	3.3468×10^3	3.3626×10^3	5.2080×10^3	6.6049×10^3		F_{27}	5.3290×10^3	4.6914×10^3	1.4082×10^4	1.9022×10^4
	F_{28}	4.6264×10^3	4.34027×10^3	5.1836×10^3	5.6978×10^3		F_{28}	1.0077×10^4	7.9907×10^3	1.00667×10^4	1.4889×10^4
	F_{29}	1.1449×10^6	1.1202×10^6	1.2465×10^7	6.4638×10^7		F_{29}	3.5244×10^5	9.0104×10^4	4.7855×10^8	3.1963×10^9

Table 8. Statistics for the BCA and its comparative algorithms.

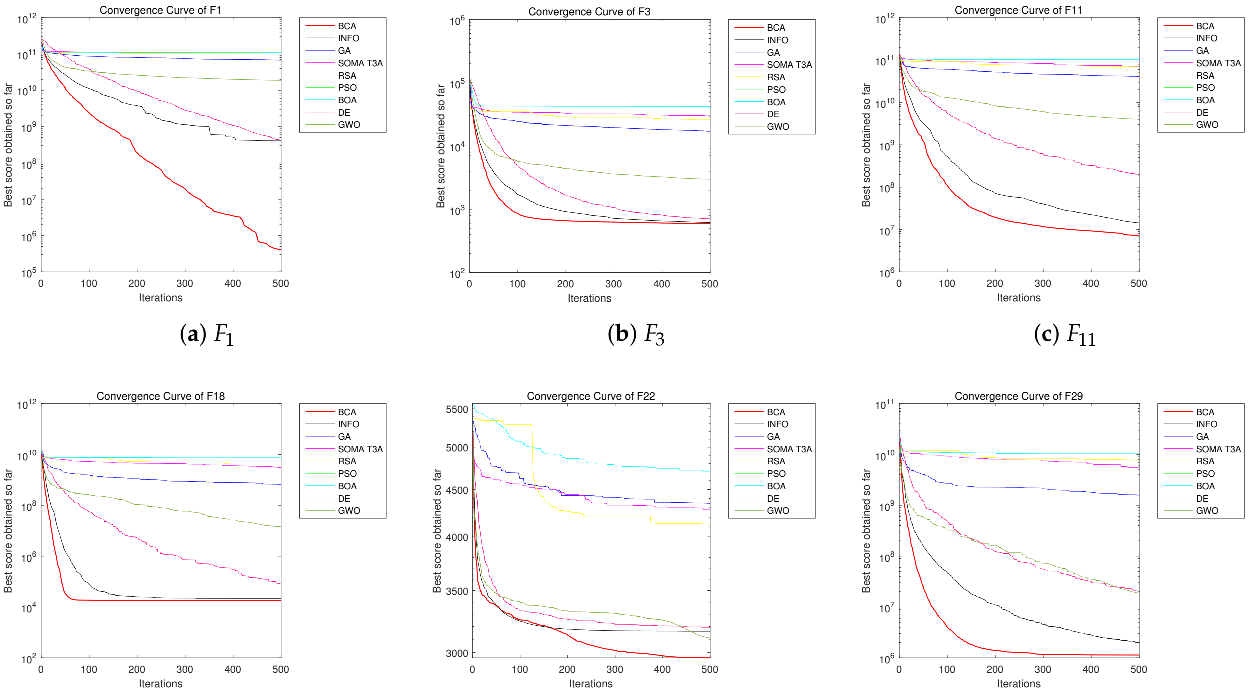
	30D							
Function type	**BCA vs. INFO (w/l/e)**	**BCA vs. RSA (w/l/e)**	**BCA vs. SOMA T3A (w/l/e)**	**BCA vs. GWO (w/l/e)**	**BCA vs. BOA (w/l/e)**	**BCA vs. DE (w/l/e)**	**BCA vs. PSO (w/l/e)**	**BCA vs. GA (w/l/e)**
Uni-model Function	1/1/0	1/1/0	2/0/0	1/1/0	2/0/0	2/0/0	2/0/0	1/1/0
Multi-model Function	5/2/0	7/0/0	7/0/0	5/2/0	7/0/0	5/2/0	5/2/0	6/1/0
Hybrid Functions	5/5/0	10/0/0	10/0/0	9/1/0	10/0/0	5/5/0	8/2/0	10/0/0
Composition Functions	8/2/0	10/0/0	10/0/0	7/3/0	10/0/0	8/2/0	10/0/0	10/0/0
Total	19/11/0	28/1/0	29/0/0	22/7/0	29/0/0	20/9/0	25/4/0	27/2/0
	50D							
Function type	**BCA vs. INFO (w/l/e)**	**BCA vs. RSA (w/l/e)**	**BCA vs. SOMA T3A (w/l/e)**	**BCA vs. GWO (w/l/e)**	**BCA vs. BOA (w/l/e)**	**BCA vs. DE (w/l/e)**	**BCA vs. PSO (w/l/e)**	**BCA vs. GA (w/l/e)**
Uni-model Function	1/1/0	2/0/0	1/1/0	2/0/0	2/0/0	2/0/0	2/0/0	1/1/0
Multi-model Function	5/2/0	7/0/0	7/0/0	6/1/0	7/0/0	5/2/0	6/1/0	6/1/0
Hybrid Functions	4/6/0	10/0/0	10/0/0	7/3/0	10/0/0	9/1/0	6/4/0	9/1/0
Composition Functions	8/2/0	10/10/0	10/10/0	8/2/0	10/10/0	9/1/0	9/1/0	9/1/0
Total	19/11/0	29/0/0	28/1/0	23/6/0	29/0/0	25/4/0	23/6/0	25/4/0
	100D							
Function type	**BCA vs. INFO (w/l/e)**	**BCA vs. RSA (w/l/e)**	**BCA vs. SOMA T3A (w/l/e)**	**BCA vs. GWO (w/l/e)**	**BCA vs. BOA (w/l/e)**	**BCA vs. DE (w/l/e)**	**BCA vs. PSO (w/l/e)**	**BCA vs. GA (w/l/e)**
Uni-model Function	1/1/0	1/1/0	1/1/0	2/0/0	2/0/0	2/0/0	2/0/0	1/1/0
Multi-model Function	2/5/0	6/1/0	6/1/0	3/4/0	7/0/0	5/2/0	6/1/0	6/1/0
Hybrid Functions	4/6/0	10/0/0	9/1/0	8/2/0	10/0/0	9/1/0	6/4/0	9/1/0
Composition Functions	9/1/0	9/1/0	9/1/0	7/3/0	9/1/0	9/1/0	9/1/0	9/1/0
Total	16/13/0	26/3/0	25/4/0	20/9/0	28/1/0	25/4/0	23/6/0	25/4/0



(**a**) F_1 (**b**) F_{10} (**c**) F_{12}

(**d**) F_{22} (**e**) F_{23} (**f**) F_{29}

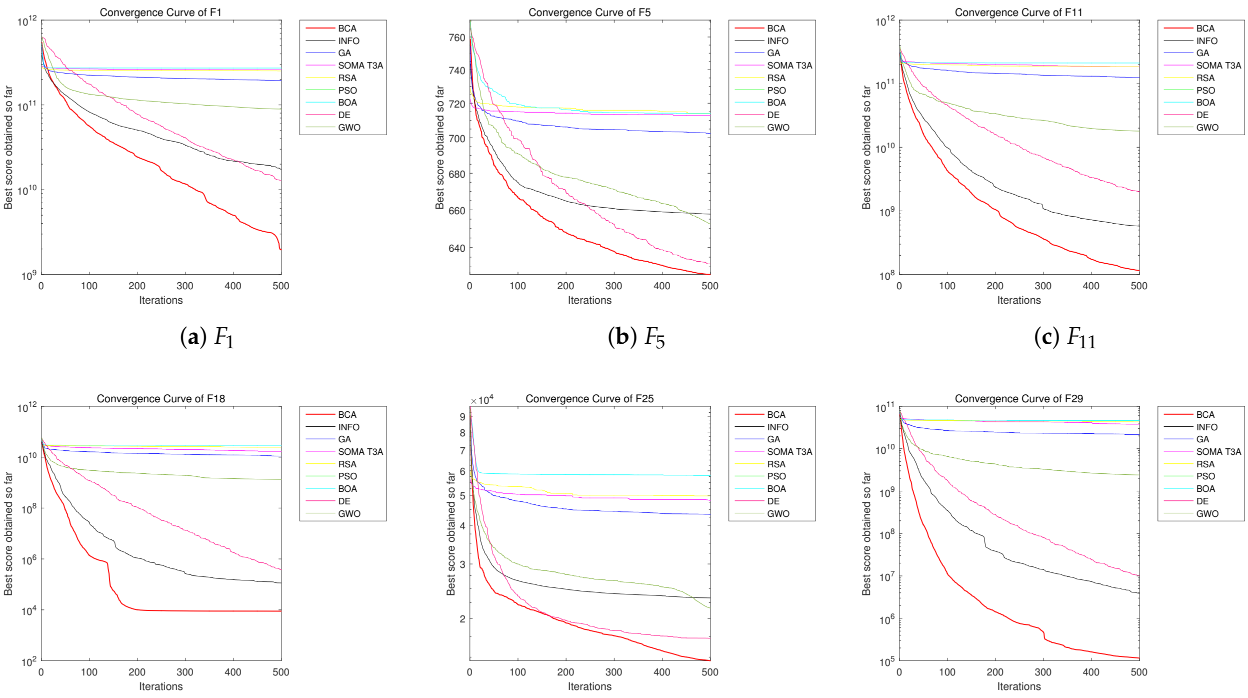
Figure 8. Convergence curve of the BCA and its comparative algorithms with 30D.



(**a**) F_1 (**b**) F_3 (**c**) F_{11}

(**d**) F_{18} (**e**) F_{22} (**f**) F_{29}

Figure 9. Convergence curve of the BCA and its comparative algorithms with 50D.



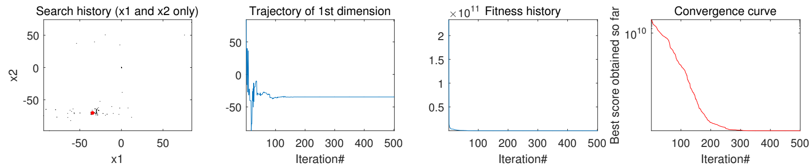
(**a**) F_1 (**b**) F_5 (**c**) F_{11}

(**d**) F_{18} (**e**) F_{25} (**f**) F_{29}

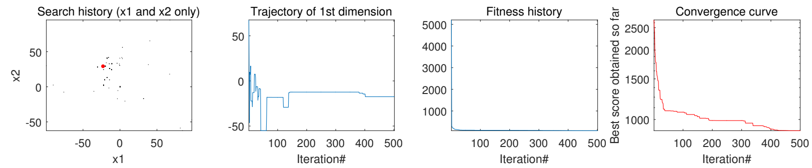
Figure 10. Convergence curve of the BCA and its comparative algorithms with 100D.

5.6. Qualitative Analysis

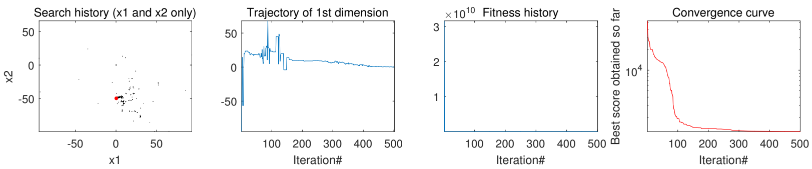
Figures 11–13 show the different qualitative indicators of BCA convergence. In addition, the first columns in Figures 11–13 show the search history graph in different dimensions (30D, 50D, 100D), which can explain the behavior and interaction among soldiers, population, and armies. It can be observed that the soldiers tend to extensively search promising regions of the search spaces and exploit the best ones. It can be seen from the experimental results that the BCA has strong diversity in the optimization process and can fully explore the search space to avoid losing the best solution.



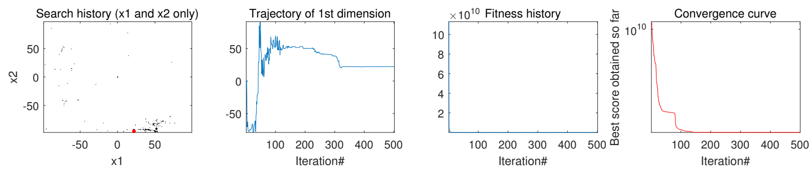
(a) F_1



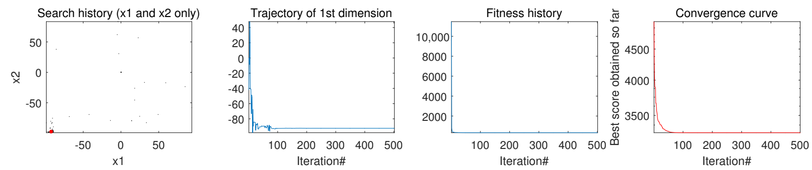
(b) F_6



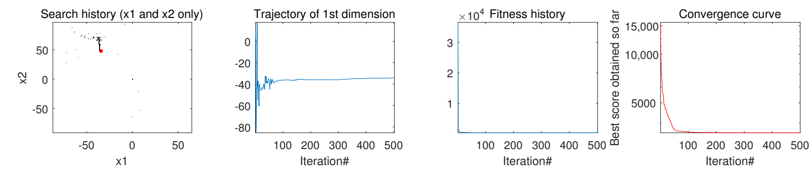
(c) F_{10}



(d) F_{12}



(e) F_{26}



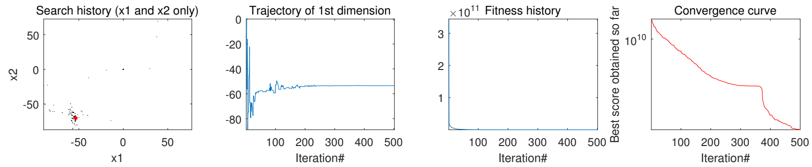
(f) F_{27}

Figure 11. Qualitative results for the studied problems with 30D.

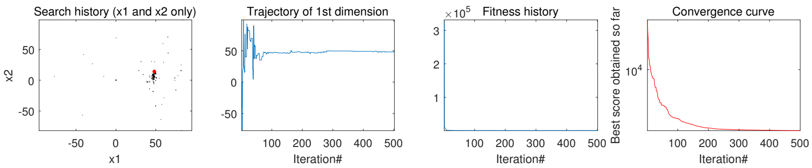
The second columns of Figures 11–13 show the trajectory of the first particle, in which changes of the first search agent in its first dimension can be observed. It can be seen from the trajectory curve that the BCA is not able to easily jump into local optimal in the search process, and soldier and army positions can be effectively updated.

The third columns of Figures 11–13 show the convergence curve. According to the convergence curve, it can be seen that the BCA can quickly converge to avoid local stagnation, and it can be seen from Figures 8–10 that the BCA converges faster than other algorithms and is able to easily find the global optimal solution.

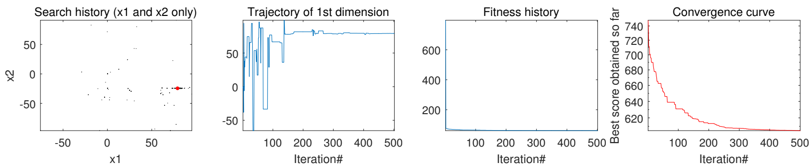
In conclusion, according to the experimental results, it can be proven that the BCA is superior to other comparative algorithms in all aspects. The besiege mechanism promotes the effective position update of soldiers, the cover mechanism ensures the avoidance of local stagnation, and the feedback mechanism effectively balances the capabilities of exploration and exploration.



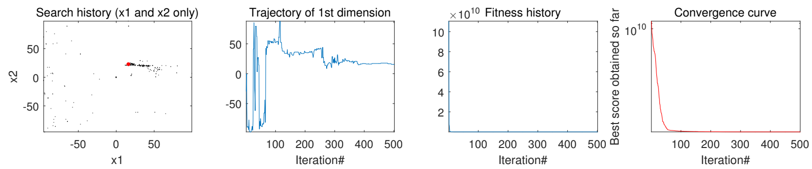
(a) F_1



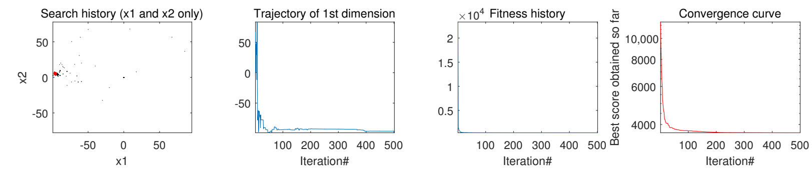
(b) F_3



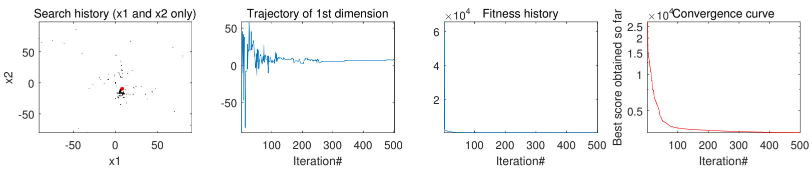
(c) F_5



(d) F_{14}

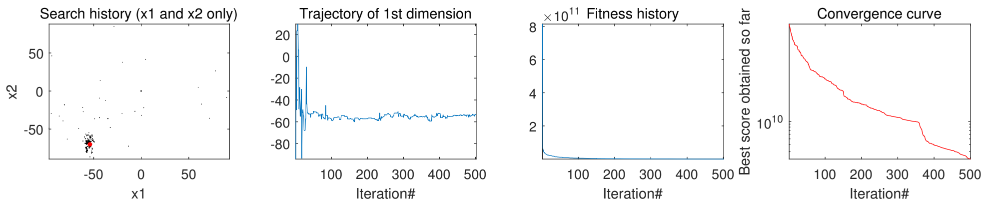


(e) F_{26}

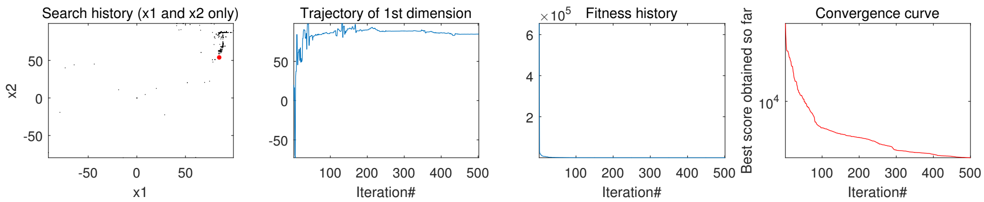


(f) F_{27}

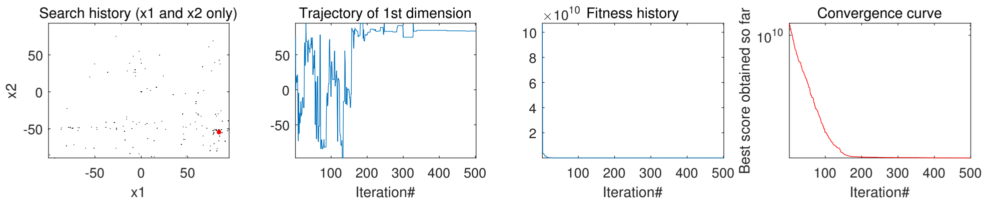
Figure 12. Qualitative results for the studied problems with 50D.



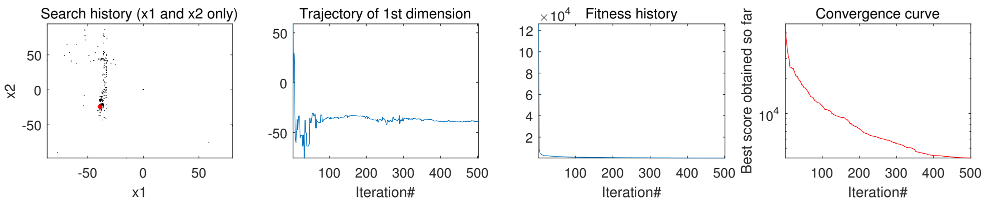
(**a**) F_1



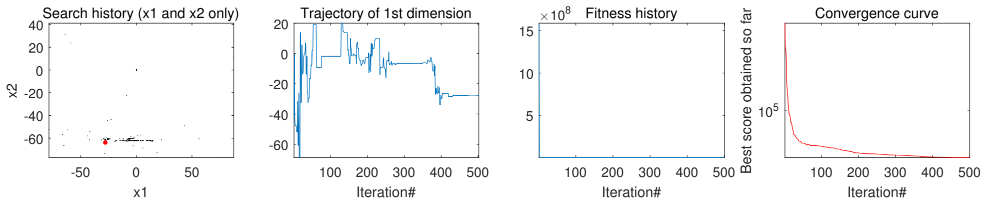
(**b**) F_3



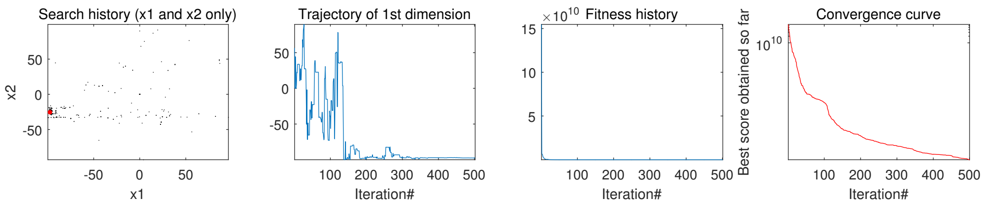
(**c**) F_{18}



(**d**) F_{27}



(**e**) F_{28}



(**f**) F_{29}

Figure 13. Qualitative results for the studied problems with 100D.

5.7. Quantitative Analysis

This section quantitatively analyzes the optimizing performance of the BCA. Tables 9–11 show the optimal values obtained by the algorithm in different dimensions (i.e., 30D, 50D, and 100D). According to the experimental results, the BCA has certain advantages in obtaining the optimal solution. In terms of Friedman ranking, the average and final rankings can obtain first place in 30D and 50D. At 100D, it can get the same ranking as INFO. INFO uses three operators to update the position of vectors in each generation. The effective collaboration of the three operators gives the INFO algorithm an advantage in high-dimensional search spaces. However, according to the statistical test results of Table 12, when the p-value is less than α, the BCA is obviously better than the comparative algorithm. This is sufficient to prove that the BCA can effectively obtain the optimal values of different types of function in the benchmark functions.

Tables 13–15 show the influence of the different population numbers ($N = 30, 60, 90$) on obtaining the optimal solution in different dimensions ($D = 30, 50, 100$). According to the experimental results in Tables 13–15, the BCA can obtain different optimal solutions when there are different populations. The higher the population, the more the number of each army will increase, which will affect the accuracy of finding the optimal solution. However, an excessive population will cause local stagnation.

5.8. Limitation Analysis

Although the BCA has global search capabilities in theory, it can easily fall into local stagnation. In addition, the BCA is sensitive to parameter settings and may require a significant amount of experimental adjustments to achieve optimal performance. The detailed analysis is as follows.

(1) The key parameters in the BCA (e.g., BCB, α, k_1, k_2, etc.) play a vital role in the BCA's performance, and the settings of these parameters significantly affect the convergence speed and global search capability. Improper parameter adjustment may lead to local optimum and reduce search efficiency.

(2) The BCA requires global exploration in the besiege phase and local exploitation in the conquer phase. Since the strategies in these phases rely on predefined parameters and rules, it may lead to an imbalance between exploration and exploitation. Too much emphasis on exploration in the early phase leads to slow convergence. Too much focus on exploitation in the later phase may miss better solutions.

The above discussion on parameter sensitivity and mechanism deficiencies leads to the following problems encountered by the BCA when dealing with low-dimensional problems. In Figure 14, the area marked by the yellow box shows that the convergence speed of the BCA slows down significantly in some iteration intervals, or even almost stops decreasing. This may indicate that the BCA has entered the local optimal solution region while exploring the global optimal solution and lacks an effective mechanism to jump out of this region. The area marked by the red box shows that the objective function value decreases rapidly in the early iteration stage, but then the change tends to be flat or stagnant, indicating that the BCA may converge to a suboptimal solution too early.

F2: The curve decreases rapidly overall, but is obviously in the local stagnation stage (yellow box), which may be due to the dimension or complexity of the problem causing the search to get stuck in the local area.

F21: It shows a rapid decline in the initial stage (red box), but then converges prematurely. The algorithm may not fully explore the solution space.

F26: Overall, there is both premature convergence (red box) and local stagnation (yellow box), which indicates that the algorithm has weak adaptability to complex functions.

Table 9. Comparison of the BCA with other algorithms on IEEE CEC 2017 benchmark functions with D = 30.

Function			BCA	INFO	RSA	SOMA_T3A	GWO	BOA	DE	PSO	GA
Unimodal Functions	F_1	Mean	**7.3880** × 10^3	1.4797 × 10^6	4.8720 × 10^{10}	4.9633 × 10^3	4.1568 × 10^9	5.9652 × 10^3	6.0843 × 10^6	3.9834 × 10^6	2.6017 × 10^3
		Std.	6.7454 × 10^6	8.0437 × 10^6	6.5562 × 10^9	3.4129 × 10^9	1.8381 × 10^9	6.4292 × 10^9	3.7530 × 10^6	3.9821 × 10^6	3.3396 × 10^9
	F_2	Mean	8.3146 × 10^4	**1.5641** × 10^4	8.2286 × 10^4	8.3233 × 10^4	7.6636 × 10^4	8.1579 × 10^4	1.8618 × 10^5	1.0133 × 10^5	6.9717 × 10^4
		Std.	1.7206 × 10^4	6.5204 × 10^3	4.6242 × 10^3	5.5701 × 10^3	1.4066 × 10^4	3.6184 × 10^3	2.6071 × 10^4	2.6407 × 10^4	7.6528 × 10^3
Multimodal Functions	F_3	Mean	**4.9649** × 10^2	5.0578 × 10^2	1.0047 × 10^4	1.2526 × 10^4	7.3329 × 10^2	2.1430 × 10^4	4.9515 × 10^2	6.0999 × 10^2	6.8051 × 10^3
		Std.	3.0714 × 10^1	2.4421 × 10^1	3.1019 × 10^3	1.2582 × 10^3	2.6180 × 10^2	3.4920 × 10^3	1.1336 × 10^1	5.0173 × 10^1	1.0775 × 10^3
	F_4	Mean	**6.2601** × 10^2	6.5256 × 10^2	9.3392 × 10^2	9.1887 × 10^2	6.4466 × 10^2	9.3711 × 10^2	7.2164 × 10^2	6.2390 × 10^2	8.3551 × 10^2
		Std.	6.4823 × 10^1	2.9596 × 10^1	2.7766 × 10^1	1.4446 × 10^1	4.1253 × 10^1	1.9692 × 10^1	1.5936 × 10^1	5.1134 × 10^1	2.7043 × 10^1
	F_5	Mean	**6.0272** × 10^2	6.2577 × 10^2	6.9182 × 10^2	6.9286 × 10^2	6.1685 × 10^2	6.9796 × 10^2	6.0317 × 10^2	6.0440 × 10^2	6.7563 × 10^2
		Std.	2.7117	1.0257 × 10^1	5.1185	3.0113	4.7764	5.4640	1.0001	1.8947	5.9692
	F_6	Mean	**8.9593** × 10^2	9.9630 × 10^2	1.3910 × 10^3	1.4453 × 10^3	9.1927 × 10^2	1.4259 × 10^3	9.8754 × 10^2	9.0824 × 10^2	1.2033 × 10^3
		Std.	6.9389 × 10^1	7.8008 × 10^1	3.4784 × 10^1	3.3319 × 10^1	5.1047 × 10^1	3.1782 × 10^1	1.4285 × 10^1	4.5985 × 10^1	4.9971 × 10^1
	F_7	Mean	9.3177 × 10^2	**9.2555** × 10^2	1.1447 × 10^3	1.1491 × 10^3	9.1408 × 10^2	1.1366 × 10^3	1.0281 × 10^3	9.3753 × 10^2	1.0674 × 10^3
		Std.	7.0344 × 10^1	3.3423 × 10^1	1.8699 × 10^1	1.3349 × 10^1	2.7180 × 10^1	1.3730 × 10^1	1.1465 × 10^1	4.9395 × 10^1	2.5073 × 10^1
	F_8	Mean	**1.3907** × 10^3	3.0992 × 10^3	1.1193 × 10^4	1.2359 × 10^4	3.1211 × 10^3	9.8685 × 10^3	1.2164 × 10^3	2.0726 × 10^3	7.7142 × 10^3
		Std.	5.7378 × 10^2	7.1942 × 10^2	9.0623 × 10^2	8.5897 × 10^2	1.3575 × 10^3	9.1532 × 10^2	1.0076 × 10^2	1.3533 × 10^3	1.1240 × 10^3
	F_9	Mean	8.1898 × 10^3	**5.2763** × 10^3	8.4987 × 10^3	8.6665 × 10^3	5.7265 × 10^3	9.1584 × 10^3	8.7828 × 10^3	7.1740 × 10^3	8.1345 × 10^3
		Std.	1.1729 × 10^3	6.3960 × 10^2	5.6988 × 10^2	3.1658 × 10^2	1.6905 × 10^3	3.5099 × 10^2	3.1398 × 10^2	1.2008 × 10^3	6.0967 × 10^2
Hybrid Functions	F_{10}	Mean	**1.1835** × 10^3	1.2740 × 10^3	9.7759 × 10^3	7.4048 × 10^3	2.2520 × 10^3	6.5269 × 10^3	1.2146 × 10^3	1.3422 × 10^3	3.7805 × 10^3
		Std.	5.8982 × 10^1	5.6408 × 10^1	4.1619 × 10^3	7.2237 × 10^2	9.6268 × 10^2	1.6504 × 10^3	3.0307 × 10^1	7.1024 × 10^1	5.4228 × 10^2
	F_{11}	Mean	**9.5317** × 10^5	1.1902 × 10^6	1.4036 × 10^{10}	8.1473 × 10^9	1.1483 × 10^8	1.5110 × 10^{10}	5.3128 × 10^7	5.3493 × 10^6	5.2030 × 10^9
		Std.	1.0400 × 10^6	1.4872 × 10^6	3.2013 × 10^9	1.1724 × 10^9	1.1814 × 10^8	3.8408 × 10^9	1.3264 × 10^7	7.0306 × 10^6	1.0609 × 10^9
	F_{12}	Mean	**2.0148** × 10^4	2.3429 × 10^4	1.1569 × 10^{10}	2.0260 × 10^9	1.9977 × 10^7	1.5656 × 10^{10}	1.0655 × 10^5	1.2126 × 10^5	3.0901 × 10^9
		Std.	1.8459 × 10^4	2.3964 × 10^4	5.0479 × 10^9	3.4694 × 10^8	4.2831 × 10^7	6.9907 × 10^9	5.1243 × 10^4	5.9190 × 10^5	1.0280 × 10^9
	F_{13}	Mean	8.1822 × 10^4	**8.9611** × 10^3	7.1491 × 10^6	2.6733 × 10^6	5.1371 × 10^5	1.2091 × 10^6	1.5327 × 10^3	6.4990 × 10^4	7.8694 × 10^5
		Std.	9.6417 × 10^4	8.3701 × 10^3	6.2921 × 10^6	7.9807 × 10^5	7.3396 × 10^5	3.2678 × 10^6	3.0669 × 10^1	4.7071 × 10^4	3.5142 × 10^5
	F_{14}	Mean	1.0747 × 10^4	**8.7682** × 10^3	6.2763 × 10^8	3.2771 × 10^8	1.2451 × 10^6	9.1247 × 10^8	2.0125 × 10^3	3.8490 × 10^4	4.2598 × 10^6
		Std.	9.7030 × 10^3	8.2320 × 10^3	3.5718 × 10^8	8.4128 × 10^7	2.0567 × 10^6	3.2735 × 10^8	3.1098 × 10^2	9.4447 × 10^4	4.8209 × 10^6
	F_{15}	Mean	2.9198 × 10^3	**2.7770** × 10^3	5.4631 × 10^3	5.6578 × 10^3	2.9158 × 10^3	9.8574 × 10^3	3.2026 × 10^3	2.8450 × 10^3	4.6878 × 10^3
		Std.	5.0390 × 10^2	3.3337 × 10^2	6.3234 × 10^2	4.8396 × 10^2	4.2457 × 10^2	1.9100 × 10^3	1.5115 × 10^2	3.5540 × 10^2	4.6143 × 10^2
	F_{16}	Mean	**2.0399** × 10^3	2.3911 × 10^3	6.6210 × 10^3	3.4163 × 10^3	2.1926 × 10^3	5.1271 × 10^3	2.6787 × 10^3	2.0744 × 10^3	2.9505 × 10^3
		Std.	1.1135 × 10^2	2.9662 × 10^2	4.7490 × 10^3	1.7524 × 10^2	2.7291 × 10^2	1.9614 × 10^4	2.1667 × 10^2	2.1001 × 10^2	3.2960 × 10^2
	F_{17}	Mean	7.7814 × 10^5	**1.3282** × 10^5	4.5941 × 10^7	3.8286 × 10^7	3.4731 × 10^6	2.8351 × 10^7	5.4324 × 10^5	2.0066 × 10^6	7.3208 × 10^6
		Std.	1.0304 × 10^6	9.0093 × 10^4	3.7883 × 10^7	2.0043 × 10^7	3.2317 × 10^6	7.6202 × 10^7	1.4745 × 10^5	1.7028 × 10^6	4.5066 × 10^6
	F_{18}	Mean	1.2889 × 10^4	**1.0950** × 10^4	7.4788 × 10^8	6.4076 × 10^8	3.7638 × 10^6	8.5804 × 10^8	2.1355 × 10^3	1.9417 × 10^4	1.2621 × 10^7
		Std.	1.1564 × 10^4	1.1168 × 10^4	6.2283 × 10^8	2.5103 × 10^8	8.3869 × 10^6	4.7651 × 10^8	1.7647 × 10^3	1.7074 × 10^4	9.9661 × 10^6
	F_{19}	Mean	**2.4143** × 10^3	2.6088 × 10^3	3.0591 × 10^3	2.9896 × 10^3	2.5280 × 10^3	3.0625 × 10^3	2.2733 × 10^3	2.4472 × 10^3	2.7151 × 10^3
		Std.	2.122 × 10^2	1.9812 × 10^2	1.5895 × 10^2	9.5183 × 10^1	1.4077 × 10^2	1.3445 × 10^2	1.6974 × 10^2	2.3705 × 10^2	1.4118 × 10^2

Table 9. *Cont.*

Function			**BCA**	**INFO**	**RSA**	**SOMA_T3A**	**GWO**	**BOA**	**DE**	**PSO**	**GA**
Composition Functions	F_{20}	Mean	2.4371×10^{3}	$\mathbf{2.4318} \times 10^{3}$	2.7361×10^{3}	2.7216×10^{3}	2.4191×10^{3}	2.5979×10^{3}	2.5109×10^{3}	2.4451×10^{3}	2.6487×10^{3}
		Std.	6.9269×10^{1}	3.3990×10^{1}	5.4538×10^{1}	3.0677×10^{1}	2.7440×10^{1}	4.8889×10^{1}	1.3871×10^{1}	4.3882×10^{1}	4.1045×10^{1}
	F_{21}	Mean	5.6464×10^{3}	$\mathbf{4.6169} \times 10^{3}$	8.8274×10^{3}	8.8268×10^{3}	6.6580×10^{3}	6.5895×10^{3}	9.9666×10^{3}	5.6468×10^{3}	6.5363×10^{3}
		Std.	3.5173×10^{3}	2.2771×10^{3}	1.2110×10^{3}	3.1078×10^{2}	2.3309×10^{3}	9.0012×10^{2}	3.1172×10^{3}	3.2973×10^{3}	6.1926×10^{2}
	F_{22}	Mean	$\mathbf{2.7655} \times 10^{3}$	2.8282×10^{3}	3.3377×10^{3}	3.5206×10^{3}	2.8358×10^{3}	$3.6649 \times 10^{3}3$	2.8781×10^{3}	2.7909×10^{3}	3.3941×10^{3}
		Std.	5.7598×10^{1}	4.6297×10^{1}	7.9028×10^{1}	6.2823×10^{1}	5.2785×10^{1}	1.9079×10^{2}	1.4794×10^{1}	4.4467×10^{1}	1.0907×10^{2}
	F_{23}	Mean	$\mathbf{2.9328} \times 10^{3}$	2.9853×10^{3}	3.4752×10^{3}	3.8190×10^{3}	3.0511×10^{3}	4.0122×10^{3}	3.0386×10^{3}	3.0085×10^{3}	3.6716×10^{3}
		Std.	6.7951×10^{1}	5.7706×10^{1}	1.8902×10^{2}	7.1766×10^{1}	7.3606×10^{1}	2.9513×10^{2}	1.2820×10^{1}	3.9095×10^{1}	7.1213×10^{1}
	F_{24}	Mean	2.9019×10^{3}	2.9172×10^{3}	4.9411×10^{3}	4.5437×10^{3}	3.0264×10^{3}	5.4129×10^{3}	$\mathbf{2.8896} \times 10^{3}$	2.9377×10^{3}	3.6190×10^{3}
		Std.	1.9859×10^{1}	2.3514×10^{1}	6.4920×10^{2}	1.9325×10^{2}	8.3053×10^{1}	4.6183×10^{2}	2.6780	2.6898×10^{1}	9.7065×10^{1}
	F_{25}	Mean	$\mathbf{4.5451} \times 10^{3}$	5.6475×10^{3}	1.0502×10^{4}	1.0450×10^{4}	5.0378×10^{3}	1.0610×10^{4}	5.7637×10^{3}	5.1020×10^{3}	8.9843×10^{3}
		Std.	8.7765×10^{2}	1.0821×10^{3}	8.5372×10^{2}	4.5798×10^{2}	7.4092×10^{2}	8.3320×10^{2}	1.2686×10^{2}	5.0098×10^{2}	4.7159×10^{2}
	F_{26}	Mean	$\mathbf{3.2483} \times 10^{3}$	3.2820×10^{3}	3.9878×10^{3}	4.3977×10^{3}	3.2000×10^{3}	4.7381×10^{3}	3.2120×10^{3}	3.2753×10^{3}	4.1949×10^{3}
		Std.	1.9161×10^{1}	5.4519×10^{1}	4.219×10^{2}	1.7630×10^{2}	2.4229×10^{-4}	3.1335×10^{2}	7.6240	2.8524×10^{1}	1.9942×10^{2}
	F_{27}	Mean	$\mathbf{3.2433} \times 10^{3}$	3.2468×10^{3}	6.5226×10^{3}	6.8475×10^{3}	3.3544×10^{3}	7.6036×10^{3}	3.2720×10^{3}	3.3028×10^{3}	5.1823×10^{3}
		Std.	2.8094×10^{1}	2.5404×10^{1}	7.9387×10^{2}	3.0725×10^{2}	1.3707×10^{2}	4.9561×10^{2}	2.7703×10^{1}	4.4193×10^{1}	2.3464×10^{2}
	F_{28}	Mean	$\mathbf{3.7791} \times 10^{3}$	4.2619×10^{3}	6.6501×10^{3}	6.9804×10^{3}	3.6817×10^{3}	7.4031×10^{3}	4.5047×10^{3}	3.9346×10^{3}	5.8452×10^{3}
		Std.	2.2075×10^{2}	2.9220×10^{2}	1.0196×10^{3}	4.8584×10^{2}	3.0165×10^{2}	9.1913×10^{3}	2.3321×10^{2}	2.3988×10^{2}	3.9740×10^{2}
	F_{29}	Mean	$\mathbf{1.0876} \times 10^{4}$	1.9525×10^{4}	2.9254×10^{9}	1.2403×10^{9}	2.1538×10^{6}	3.3004×10^{8}	7.1202×10^{4}	3.7456×10^{4}	2.3894×10^{8}
		Std.	3.8180×10^{3}	1.3548×10^{4}	1.1521×10^{9}	3.6092×10^{8}	5.8043×10^{6}	1.2809×10^{9}	4.1259×10^{4}	3.7595×10^{4}	1.3164×10^{8}
Average Ranking			**2.13**	2.70	7.60	7.63	3.87	8.13	3.73	3.33	5.90
Total Ranking			**1**	2	7	8	5	9	4	3	6

Table 10. Comparison of the BCA with other algorithms on IEEE CEC 2017 benchmark functions with D = 50.

Function			BCA	INFO	RSA	SOMA_T3A	GWO	BOA	DE	PSO	GA
Unimodal Functions	F_1	Mean	**8.0698** $\times 10^{6}$	1.2081×10^{8}	1.0054×10^{11}	1.0541×10^{11}	1.9953×10^{10}	9.9579×10^{10}	4.2853×10^{8}	4.7099×10^{8}	6.7499×10^{10}
		Std.	2.5848×10^{7}	3.3163×10^{8}	8.7185×10^{9}	4.1479×10^{9}	6.6208×10^{9}	7.6477×10^{9}	1.0631×10^{8}	4.3556×10^{8}	3.6694×10^{9}
	F_2	Mean	2.1874×10^{5}	**1.0352** $\times 10^{5}$	1.7072×10^{5}	1.7426×10^{5}	2.9008×10^{5}	3.0258×10^{5}	4.0362×10^{5}	2.4174×10^{5}	1.6088×10^{5}
		Std.	4.4722×10^{4}	3.3469×10^{4}	1.4597×10^{4}	1.4382×10^{4}	9.6064×10^{4}	1.2568×10^{5}	6.6267×10^{4}	5.1004×10^{4}	1.4334×10^{4}
Multimodal Functions	F_3	Mean	**5.8938** $\times 10^{2}$	6.4763×10^{2}	2.7104×10^{4}	2.9864×10^{4}	3.0583×10^{3}	4.1718×10^{4}	6.6851×10^{2}	8.8720×10^{2}	1.7406×10^{4}
		Std.	7.0100×10^{1}	8.8827×10^{1}	4.5861×10^{3}	2.4337×10^{3}	1.0917×10^{3}	3.6393×10^{3}	4.1228×10^{1}	1.1478×10^{2}	2.6666×10^{3}
	F_4	Mean	**7.8413** $\times 10^{2}$	8.0011×10^{2}	1.1750×10^{3}	1.2143×10^{3}	7.9093×10^{2}	1.2140×10^{3}	9.6133×10^{2}	8.0109×10^{2}	1.0797×10^{3}
		Std.	1.2354×10^{2}	5.3138×10^{1}	2.5710×10^{1}	1.3604×10^{1}	3.4174×10^{1}	2.5134×10^{1}	2.1992×10^{1}	7.7571×10^{1}	4.2529×10^{1}
	F_5	Mean	**6.0651** $\times 10^{2}$	6.4537×10^{2}	7.0573×10^{2}	6.9849×10^{2}	6.3233×10^{2}	7.0448×10^{2}	6.1448×10^{2}	6.1507×10^{2}	6.8897×10^{2}
		Std.	3.0238	6.7509	4.3169	2.0565	6.6989	4.4172	2.0835	3.4453	5.222
	F_6	Mean	1.2372×10^{3}	1.3888×10^{3}	1.9726×10^{3}	2.0541×10^{3}	1.1988×10^{3}	2.0184×10^{3}	1.2313×10^{3}	1.2571×10^{3}	**1.7464** $\times 10^{3}$
		Std.	1.7023×10^{2}	1.0297×10^{2}	3.3972×10^{1}	3.3665×10^{1}	9.2951×10^{1}	3.4824×10^{1}	2.5992×10^{1}	8.0061×10^{1}	8.6661×10^{1}
	F_7	Mean	1.0984×10^{3}	**1.0977** $\times 10^{3}$	1.5119×10^{3}	1.4972×10^{3}	1.1061×10^{3}	1.5184×10^{3}	1.2211×10^{3}	1.1243×10^{3}	1.4017×10^{3}
		Std.	1.3220×10^{2}	5.3863×10^{1}	2.1038×10^{1}	1.5377×10^{1}	6.7371×10^{1}	2.8109×10^{1}	1.8749×10^{1}	8.4016×10^{1}	4.1217×10^{1}
	F_8	Mean	5.5744×10^{3}	9.2101×10^{3}	3.8790×10^{4}	4.0799×10^{4}	1.6457×10^{4}	3.7342×10^{4}	3.0368×10^{3}	**6.7222** $\times 10^{3}$	3.0784×10^{4}
		Std.	3.6592×10^{3}	2.3518×10^{3}	2.3156×10^{3}	2.4841×10^{3}	4.4410×10^{3}	2.6924×10^{3}	7.5760×10^{2}	5.4127×10^{3}	3.8685×10^{3}
	F_9	Mean	1.4690×10^{4}	**8.1527** $\times 10^{3}$	1.5335×10^{4}	1.4691×10^{4}	1.0845×10^{4}	1.5445×10^{4}	1.5134×10^{4}	1.4055×10^{4}	1.3846×10^{4}
		Std.	1.3087×10^{3}	9.1292×10^{2}	4.0800×10^{2}	3.2265×10^{2}	3.3112×10^{3}	4.7646×10^{2}	5.7015×10^{2}	1.1317×10^{3}	7.4803×10^{2}
Hybrid Functions	F_{10}	Mean	1.5325×10^{3}	**1.4936** $\times 10^{3}$	2.1183×10^{4}	1.9607×10^{4}	7.3545×10^{3}	2.7563×10^{4}	1.8245×10^{3}	2.1715×10^{3}	1.5010×10^{4}
		Std.	4.1856×10^{2}	3.2047×10^{2}	2.6847×10^{3}	1.5181×10^{3}	2.5495×10^{3}	1.9258×10^{3}	1.7100×10^{2}	4.2295×10^{2}	2.3156×10^{3}
	F_{11}	Mean	**6.0038** $\times 10^{6}$	1.5783×10^{7}	7.4291×10^{10}	7.2731×10^{10}	5.0761×10^{9}	1.0244×10^{11}	1.7041×10^{8}	4.7613×10^{7}	4.1717×10^{10}
		Std.	3.9285×10^{6}	1.1161×10^{7}	1.9085×10^{10}	5.4454×10^{9}	4.0055×10^{9}	1.2312×10^{10}	8.4455×10^{7}	2.8385×10^{7}	5.6714×10^{9}
	F_{12}	Mean	**8.8823** $\times 10^{3}$	3.3263×10^{4}	4.5817×10^{10}	3.9844×10^{10}	7.8005×10^{8}	7.6975×10^{10}	1.2000×10^{5}	2.9367×10^{6}	1.7893×10^{10}
		Std.	8.3569×10^{3}	4.2739×10^{4}	1.4543×10^{10}	5.9399×10^{9}	1.5196×10^{9}	1.0671×10^{10}	4.0879×10^{6}	1.5111×10^{7}	3.9607×10^{9}
	F_{13}	Mean	7.1093×10^{5}	9.3353×10^{4}	5.9906×10^{7}	2.5106×10^{7}	3.1367×10^{6}	1.6226×10^{8}	5.8469×10^{4}	**8.3739** $\times 10^{5}$	2.7865×10^{7}
		Std.	7.0911×10^{5}	9.7111×10^{4}	3.9631×10^{7}	8.9211×10^{6}	4.5229×10^{6}	9.3788×10^{7}	2.8929×10^{4}	1.0091×10^{6}	1.4505×10^{7}
	F_{14}	Mean	9.5976×10^{3}	1.0522×10^{4}	6.6059×10^{9}	5.9406×10^{9}	4.1580×10^{7}	**1.3721** $\times 10^{10}$	1.2541×10^{5}	8.4425×10^{3}	1.5639×10^{9}
		Std.	7.5456×10^{3}	6.5294×10^{3}	2.9172×10^{9}	4.6824×10^{8}	6.0041×10^{7}	3.2457×10^{9}	1.2170×10^{5}	7.5388×10^{3}	4.9284×10^{8}
	F_{15}	Mean	4.1961×10^{3}	3.7242×10^{3}	8.4170×10^{3}	8.9178×10^{3}	3.6754×10^{3}	1.5434×10^{4}	5.6286×10^{3}	4.0180×10^{3}	**7.2506** $\times 10^{3}$
		Std.	1.1946×10^{3}	4.4344×10^{2}	1.4172×10^{3}	5.6007×10^{2}	5.3702×10^{2}	1.7538×10^{3}	2.9782×10^{2}	7.1457×10^{2}	6.0709×10^{2}
	F_{16}	Mean	3.6761×10^{3}	**3.2831** $\times 10^{3}$	1.1912×10^{4}	1.1882×10^{4}4	3.2836×10^{3}	9.4433×10^{3}	3.8683×10^{3}	3.4948×10^{3}	4.4646×10^{3}
		Std.	5.0328×10^{2}	3.3696×10^{2}	4.0412×10^{3}	1.8850×10^{3}	4.3853×10^{2}	1.2499×10^{4}	2.4086×10^{2}	5.1286×10^{2}	4.6452×10^{2}
	F_{17}	Mean	3.7145×10^{6}	**6.5369** $\times 10^{5}$	2.2561×10^{8}	5.3734×10^{7}	1.8814×10^{7}	2.3829×10^{8}	7.2114×10^{6}	9.1979×10^{6}	5.1911×10^{7}
		Std.	3.4216×10^{6}	5.7179×10^{5}	8.4785×10^{7}	1.0627×10^{7}	2.4697×10^{7}	1.1768×10^{8}	3.5520×10^{6}	7.7993×10^{6}	1.1650×10^{7}
	F_{18}	Mean	**1.7769** $\times 10^{4}$	2.0862×10^{4}	4.2811×10^{9}	3.2215×10^{9}	2.1014×10^{7}	7.5964×10^{9}	2.4024×10^{4}	7.7060×10^{4}	5.6002×10^{8}
		Std.	1.4976×10^{4}	1.2405×10^{4}	1.2925×10^{9}	6.7984×10^{8}	5.2307×10^{7}	1.5056×10^{9}	4.6330×10^{4}	3.3516×10^{5}	1.9099×10^{8}
	F_{19}	Mean	3.9894×10^{3}	3.3028×10^{3}	4.2539×10^{3}	4.0362×10^{3}	3.2912×10^{3}	4.4428×10^{3}	4.3908×10^{3}	3.7272×10^{3}	**3.6308** $\times 10^{3}$
		Std.	3.1532×10^{2}	3.7555×10^{2}	2.2889×10^{2}	1.5749×10^{2}	5.1545×10^{2}	1.7880×10^{2}	1.8545×10^{2}	3.8193×10^{2}	2.6442×10^{2}

Table 10. *Cont.*

Function			BCA	INFO	RSA	SOMA_T3A	GWO	BOA	DE	PSO	GA
Composition Functions	F_{20}	Mean	2.6289×10^3	$\mathbf{2.6137} \times 10^3$	3.1229×10^3	3.1833×10^3	2.6297×10^3	3.1620×10^3	2.7704×10^3	2.6479×10^3	3.0632×10^3
		Std.	1.4251×10^2	5.9255×10^1	8.5638×10^1	3.2547×10^1	7.3004×10^1	7.3181×10^1	2.5858×10^1	7.9332×10^1	4.3429×10^1
	F_{21}	Mean	1.6405×10^4	$\mathbf{1.0111} \times 10^4$	1.7409×10^4	1.6761×10^4	1.3105×10^4	1.6839×10^4	1.6646×10^4	1.4385×10^4	1.5936×10^4
		Std.	9.7241×10^2	6.4693×10^2	3.9289×10^2	3.2032×10^2	3.0878×10^3	7.4231×10^2	4.0475×10^2	2.9685×10^3	7.9201×10^2
	F_{22}	Mean	$\mathbf{2.9959} \times 10^3$	3.1934×10^3	4.0747×10^3	4.2654×10^3	3.0818×10^3	4.8310×10^3	3.2146×10^3	3.0819×10^3	4.3706×10^3
		Std.	1.2535×10^2	1.2322×10^2	1.9257×10^2	7.5957×10^1	9.1497×10^1	1.9073×10^2	1.8505×10^1	7.3836×10^1	1.0892×10^2
	F_{23}	Mean	3.2918×10^3	$\mathbf{3.2877} \times 10^3$	4.4812×10^3	5.0063×10^3	3.3843×10^3	6.0799×10^3	3.3377×10^3	3.3251×10^3	4.7415×10^3
		Std.	1.1679×10^2	9.0870×10^1	6.6486×10^2	9.4557×10^1	1.1332×10^2	3.4054×10^2	2.0458×10^1	4.8423×10^1	1.2742×10^2
	F_{24}	Mean	$\mathbf{3.0860} \times 10^3$	3.1876×10^3	1.3539×10^4	1.4018×10^4	4.3759×10^3	1.5817×10^4	3.1449×10^3	3.2742×10^3	9.3090×10^3
		Std.	3.2256×10^1	4.9994×10^1	1.6051×10^3	7.1673×10^2	7.1319×10^2	8.6365×10^2	3.2658×10^1	7.0189×10^1	4.8423×10^2
	F_{25}	Mean	$\mathbf{6.5998} \times 10^3$	1.0058×10^4	1.6336×10^4	1.6454×10^4	7.2661×10^3	1.8529×10^4	8.4696×10^3	7.3589×10^3	1.4126×10^4
		Std.	1.1763×10^3	1.7772×10^3	8.1251×10^2	4.5176×10^2	1.0119×10^3	4.6464×10^2	3.5477×10^2	8.7881×10^2	7.1047×10^2
	F_{26}	Mean	3.4756×10^3	3.7370×10^3	5.9281×10^3	6.9078×10^3	3.2000×10^3	6.6606×10^3	3.3844×10^3	3.6645×10^3	$\mathbf{6.4421} \times 10^3$
		Std.	9.5517×10^1	1.8797×10^2	1.0353×10^3	3.6858×10^2	2.1386×10^{-4}	5.6742×10^2	7.7249×10^1	1.0106×10^2	4.2455×10^2
	F_{27}	Mean	$\mathbf{3.3779} \times 10^3$	3.5239×10^3	1.1726×10^4	1.2045×10^4	3.3962×10^3	1.1800×10^4	3.5984×10^3	3.5421×10^3	8.9990×10^3
		Std.	5.7735×10^1	1.0620×10^2	1.5292×10^3	5.8127×10^2	3.8952×10^2	1.1263×10^3	9.5782×10^2	1.1275×10^2	4.9551×10^2
	F_{28}	Mean	$\mathbf{4.3124} \times 10^3$	5.1213×10^3	5.7134×10^4	3.3590×10^4	4.6454×10^3	1.4499×10^5	5.6166×10^3	4.7325×10^3	1.4915×10^4
		Std.	4.0225×10^2	5.0900×10^2	8.1925×10^4	8.3978×10^3	6.2813×10^2	3.2654×10^5	2.6784×10^2	5.0232×10^2	2.9872×10^3
	F_{29}	Mean	$\mathbf{1.2805} \times 10^6$	1.7690×10^6	7.3830×10^9	5.8149×10^9	5.1098×10^7	1.0356×10^{10}	2.1101×10^7	8.5554×10^6	1.8142×10^9
		Std.	3.1411×10^5	9.3920×10^5	2.5616×10^9	8.3036×10^8	2.0667×10^7	2.3883×10^9	1.0839×10^7	5.0240×10^6	4.4706×10^8
Average Ranking			**2.20**	2.40	7.43	7.47	3.53	8.53	4.10	3.57	5.77
Total Ranking			**1**	2	7	8	3	9	5	4	6

Table 11. Comparison of the BCA with other algorithms on IEEE CEC 2017 benchmark functions with D = 100.

Function			BCA	INFO	RSA	SOMA_T3A	GWO	BOA	DE	PSO	GA
Unimodal Functions	F_1	Mean	**2.4262** $\times 10^9$	1.5827×10^{10}	2.5051×10^{11}	2.6181×10^{11}	8.1909×10^{10}	2.8481×10^{11}	1.2168×10^{10}	1.0303×10^{10}	1.9283×10^{11}
		Std.	2.1697×10^9	5.9449×10^9	6.8445×10^9	4.6747×10^9	1.1466×10^{10}	9.0918×10^9	2.3400×10^9	2.0291×10^9	8.5353×10^9
	F_2	Mean	6.5836×10^5	3.7586×10^5	**3.4850** $\times 10^5$	3.5384×10^5	1.3649×10^6	9.5001×10^5	9.2191×10^5	7.3736×10^5	3.5431×10^5
		Std.	8.5490×10^4	6.1650×10^4	1.1314×10^4	1.0242×10^4	5.1518×10^5	1.1966×10^5	1.0117×10^5	9.8121×10^4	2.4768×10^4
Multimodal Functions	F_3	Mean	**1.1402** $\times 10^3$	2.1230×10^3	8.6240×10^4	8.5233×10^4	1.1234×10^4	1.0456×10^5	1.9129×10^3	2.2175×10^3	5.5811×10^4
		Std.	1.6994×10^2	4.5247×10^2	1.1066×10^4	7.1279×10^3	2.9543×10^3	9.9881×10^3	3.6458×10^2	3.5647×10^2	4.8785×10^3
	F_4	Mean	1.4523×10^3	**1.3003** $\times 10^3$	2.0685×10^3	2.1246×10^3	1.3395×10^3	2.1023×10^3	1.4984×10^3	1.4913×10^3	1.9419×10^3
		Std.	3.0949×10^2	6.0707×10^1	4.5174×10^1	3.3107×10^1	5.4478×10^1	2.7335×10^1	4.1121×10^1	1.3418×10^2	6.3060×10^1
	F_5	Mean	**6.2673** $\times 10^2$	6.5983×10^2	7.1255×10^2	7.1278×10^2	6.5269×10^2	7.1582×10^2	6.3221×10^2	6.4121×10^2	7.0139×10^2
		Std.	7.8883	6.0012	3.7904	1.4522	4.5112	3.3421	3.8317	7.7823	3.7522
	F_6	Mean	2.4105×10^3	2.8567×10^3	3.9083×10^3	4.0947×10^3	**2.3550** $\times 10^3$	3.9955×10^3	2.2631×10^3	2.6027×10^3	3.5544×10^3
		Std.	3.2586×10^2	2.4278×10^2	9.0787×10^1	5.3391×10^1	1.3874×10^2	7.1554×10^1	8.6077×10^1	1.7078×10^2	1.5662×10^2
	F_7	Mean	1.7870×10^3	1.7241×10^3	2.5308×10^3	2.5713×10^3	**1.6996** $\times 10^3$	2.5950×10^3	1.9096×10^3	1.7887×10^3	2.3853×10^3
		Std.	2.8372×10^2	1.0907×10^2	5.9205×10^1	3.0255×10^1	8.2933×10^1	4.5855×10^1	3.500×10^1	1.5337×10^2	6.1557×10^1
	F_8	Mean	3.3261×10^4	2.6853×10^4	8.2162×10^4	8.2940×10^4	5.4341×10^4	8.5244×10^4	**2.1295** $\times 10^4$	4.5034×10^4	7.2344×10^4
		Std.	1.0941×10^4	3.0790×10^3	3.6701×10^3	3.2867×10^3	1.4229×10^4	3.5794×10^3	5.6631×10^3	2.4891×10^4	4.7622×10^3
	F_9	Mean	3.2281×10^4	**1.7869** $\times 10^4$	3.2081×10^4	3.1765×10^4	2.5030×10^4	3.3361×10^4	3.3478×10^4	3.1371×10^4	3.0695×10^4
		Std.	1.4270×10^3	1.7704×10^3	9.0578×10^2	5.3961×10^2	6.1326×10^3	5.5069×10^2	5.9817×10^2	1.1393×10^3	1.0018×10^3
Hybrid Functions	F_{10}	Mean	1.2551×10^5	**3.2673** $\times 10^4$	2.1884×10^5	1.8807×10^5	1.2569×10^5	1.1115×10^6	3.4259×10^5	1.2228×10^5	1.5227×10^5
		Std.	3.0498×10^4	7.6137×10^3	2.8094×10^4	1.7728×10^4	2.8048×10^4	2.5365×10^5	4.5553×10^4	3.0212×10^4	2.0980×10^4
	F_{11}	Mean	**1.0159** $\times 10^8$	8.2837×10^8	1.7894×10^{11}	1.8373×10^{11}	2.6146×10^{10}	2.2615×10^{11}	1.2339×10^9	1.4037×10^9	1.2383×10^{11}
		Std.	5.6502×10^7	8.3954×10^8	2.2378×10^{10}	7.2630×10^9	8.4709×10^9	1.2059×10^{10}	5.5010×10^8	6.2157×10^8	1.0713×10^{10}
	F_{12}	Mean	**1.1851** $\times 10^4$	3.5256×10^5	4.6528×10^{10}	4.3217×10^{10}	3.7958×10^9	5.0376×10^{10}	1.9034×10^6	1.6131×10^7	2.5314×10^{10}
		Std.	8.1518×10^3	1.0772×10^6	5.4012×10^9	2.4149×10^9	2.6778×10^9	3.5562×10^9	1.9179×10^6	4.6544×10^7	2.4457×10^9
	F_{13}	Mean	2.9782×10^6	**1.5074** $\times 10^6$	9.3691×10^7	4.3262×10^7	9.8780×10^6	1.7488×10^8	2.3123×10^7	1.1788×10^7	1.8978×10^7
		Std.	2.0700×10^6	7.3534×10^5	4.2511×10^7	9.3987×10^6	4.7507×10^6	5.3129×10^7	8.3497×10^6	7.3408×10^6	4.1156×10^6
	F_{14}	Mean	**6.1051** $\times 10^3$	1.8254×10^4	2.3222×10^{10}	2.1381×10^{10}	8.4784×10^8	2.8782×10^{10}	1.3875×10^6	8.3210×10^6	1.0750×10^{10}
		Std.	4.3314×10^3	2.3018×10^4	3.7678×10^9	1.9326×10^9	1.1504×10^9	3.5521×10^9	2.3709×10^6	2.6313×10^7	1.1435×10^9
	F_{15}	Mean	9.8380×10^3	**6.3684** $\times 10^3$	2.0929×10^4	2.0376×10^4	9.1569×10^3	2.5379×10^4	1.1466×10^4	9.3651×10^3	1.8110×10^4
		Std.	2.2106×10^3	7.4683×10^2	3.1865×10^3	1.1518×10^3	1.7239×10^3	1.8175×10^3	4.0190×10^2	1.6248×10^3	1.6440×10^3
	F_{16}	Mean	7.3028×10^3	6.2763×10^3	1.2741×10^7	**2.2644** $\times 10^6$	9.0987×10^3	3.0086×10^6	8.1420×10^3	6.7173×10^3	5.6565×10^5
		Std.	1.2345×10^3	9.5553×10^2	1.1567×10^7	7.8162×10^5	4.6655×10^3	1.6597×10^7	3.0139×10^2	9.0206×10^2	3.2081×10^5
	F_{17}	Mean	9.2284×10^6	**2.1110** $\times 10^6$	1.5856×10^8	1.0757×10^8	1.5438×10^7	2.2938×10^8	5.3408×10^7	1.8286×10^7	3.6796×10^7
		Std.	7.9516×10^6	1.2088×10^6	8.5792×10^7	2.4989×10^7	8.9871×10^6	1.3454×10^8	1.9901×10^7	1.1332×10^7	9.9945×10^6
	F_{18}	Mean	**1.6481** $\times 10^4$	1.0110×10^5	2.3574×10^{10}	1.8007×10^{10}	1.0655×10^9	2.9969×10^{10}	4.9848×10^6	3.6414×10^5	1.0713×10^{10}
		Std.	4.0296×10^4	2.0092×10^5	4.9719×10^9	1.5611×10^9	1.5996×10^9	3.5853×10^9	1.5144×10^7	1.2319×10^6	1.8233×10^9
	F_{19}	Mean	7.6005×10^3	**5.5823** $\times 10^3$	7.7664×10^3	7.4699×10^3	6.1301×10^3	8.2367×10^3	7.1889×10^3	7.4633×10^3	7.1668×10^3
		Std.	3.2761×10^2	5.2891×10^2	2.6622×10^2	2.3863×10^2	1.4393×10^3	3.1704×10^2	2.8736×10^2	5.2791×10^2	3.2952×10^2

Table 11. *Cont.*

Function			BCA	INFO	RSA	SOMA_T3A	GWO	BOA	DE	PSO	GA
Composition Functions	F_{20}	Mean	**3.2061** $\times 10^3$	3.3173×10^3	4.6039×10^3	4.7637×10^3	3.3123×10^3	4.9988×10^3	3.4553×10^3	3.3432×10^3	4.7138×10^3
		Std.	3.0861×10^2	1.5987×10^2	2.1284×10^2	5.8958×10^1	9.0393×10^1	1.8975×10^2	3.8367×10^1	1.3888×10^2	1.5033×10^2
	F_{21}	Mean	3.4888×10^4	**2.1203** $\times 10^4$	3.4798×10^4	3.4291×10^4	2.9534×10^4	3.4654×10^4	3.3976×10^4	3.3656×10^4	3.3430×10^4
		Std.	6.8231×10^2	2.0965×10^3	5.9158×10^2	5.0831×10^2	5.5320×10^3	5.0334×10^2	5.0728×10^2	1.4131×10^3	1.0478×10^3
	F_{22}	Mean	**3.4270** $\times 10^3$	3.9970×10^3	5.7098×10^3	6.7390×10^3	3.9791×10^3	6.8537×10^3	3.9121×10^3	3.7042×10^3	6.7026×10^3
		Std.	1.0831×10^2	1.8032×10^2	2.2234×10^2	1.3785×10^2	1.1420×10^2	3.2125×10^2	4.3860×10^1	1.0324×10^2	3.0047×10^2
	F_{23}	Mean	**4.1427** $\times 10^3$	4.8633×10^3	9.0247×10^3	1.0304×10^4	4.9121×10^3	1.4845×10^4	4.4334×10^3	4.3668×10^3	1.1008×10^4
		Std.	2.2216×10^2	3.0941×10^2	2.2474×10^3	4.1372×10^2	1.4091×10^2	1.1676×10^3	5.7606×10^1	1.1141×10^2	5.3080×10^2
	F_{24}	Mean	**3.9451** $\times 10^3$	4.4219×10^3	2.5541×10^4	2.6442×10^4	9.0625×10^3	2.8948×10^4	5.2123×10^3	5.3468×10^3	1.8131×10^4
		Std.	2.4271×10^2	2.7817×10^2	1.6708×10^3	1.2230×10^3	1.6244×10^3	2.0025×10^3	4.7429×10^2	6.0285×10^2	6.0790×10^2
	F_{25}	Mean	**1.4253** $\times 10^4$	2.4819×10^4	5.0168×10^4	4.7764×10^4	2.2388×10^4	5.7504×10^4	1.6418×10^4	1.6479×10^4	4.3008×10^4
		Std.	2.2698×10^3	3.6772×10^3	3.5322×10^3	1.7066×10^3	1.8713×10^3	2.1206×10^3	5.2302×10^2	1.1404×10^3	2.0140×10^3
	F_{26}	Mean	3.6732×10^3	4.1662×10^3	1.2084×10^4	1.3883×10^4	**3.2000** $\times 10^3$	1.6125×10^4	3.7738×10^3	4.1689×10^3	1.2293×10^4
		Std.	1.1192×10^2	2.4580×10^2	2.7715×10^3	5.0937×10^2	2.7202×10^{-4}	1.1549×10^3	1.9439×10^2	1.6458×10^2	8.7161×10^2
	F_{27}	Mean	4.4414×10^3	5.6510×10^3	2.9822×10^4	2.7928×10^4	**3.5548** $\times 10^3$	3.9325×10^4	1.0193×10^4	7.2703×10^3	2.5580×10^4
		Std.	7.1057×10^2	1.0350×10^3	2.0328×10^3	6.0543×10^2	1.3957×10^3	1.9787×10^3	2.4050×10^3	2.0799×10^3	9.1464×10^2
	F_{28}	Mean	**7.8602** $\times 10^3$	8.4401×10^3	7.6068×10^5	3.4879×10^5	7.9197×10^3	1.3111×10^6	1.0405×10^4	9.1411×10^3	5.5861×10^4
		Std.	1.3546×10^3	9.1056×10^2	5.7523×10^5	9.2666×10^4	1.6097×10^3	4.3956×10^5	4.9541×10^2	9.8512×10^2	1.6627×10^4
	F_{29}	Mean	**1.0003** $\times 10^5$	3.0133×10^6	4.1761×10^{10}	3.9200×10^{10}	2.8882×10^9	4.7457×10^{10}	8.3342×10^6	1.2042×10^7	2.2226×10^{10}
		Std.	6.6358×10^4	2.0704×10^6	4.4631×10^9	1.7074×10^9	2.3192×10^9	4.5899×10^9	4.2568×10^6	1.9619×10^7	3.6108×10^9
	Average Ranking		**2.47**	**2.47**	7.17	7.10	3.67	8.80	4.07	3.63	5.67
	Total Ranking		**1**	**1**	8	7	4	9	5	3	6

Table 12. Results of the Wilcoxon signed-rank test for the BCA and other algorithms on IEEE CEC 2017 benchmark functions.

	30D								
BCA	vs. SCO	vs. INFO	vs. GA	vs. SOMA_T3A	vs. RSA	vs. PSO	vs. BOA	vs. DE	vs. GWO
	1.7344×10^{-6}	2.5364×10^{-1}	1.4936×10^{-5}	1.7344×10^{-6}	5.2165×10^{-6}	3.3173×10^{-4}	5.7517×10^{-6}	3.8723×10^{-2}	5.7064×10^{-4}
	Yes	NO	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
	Yes	NO	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
	50D								
BCA	vs. SCO	vs. INFO	vs. GA	vs. SOMA_T3A	vs. RSA	vs. PSO	vs. BOA	vs. DE	vs. GWO
	1.7344×10^{-6}	5.5774×10^{-1}	7.5137×10^{-5}	1.3601×10^{-5}	1.2381×10^{-5}	3.1618×10^{-3}	1.7344×10^{-6}	2.0515×10^{-4}	2.7653×10^{-3}
	Yes	NO	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
	Yes	NO	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
	100D								
BCA	vs. SCO	vs. INFO	vs. GA	vs. SOMA_T3A	vs. RSA	vs. PSO	vs. BOA	vs. DE	vs. GWO
	1.7344×10^{-6}	9.4261×10^{-1}	5.3070×10^{-5}	2.5967×10^{-5}	1.9729×10^{-5}	8.3071×10^{-4}	2.1266×10^{-6}	1.0570×10^{-4}	2.9575×10^{-3}
	Yes	NO	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
	Yes	NO	**Yes**	**Yes**	**Yes**	**Yes**	Yes	**Yes**	**Yes**

Table 13. Results of different populations with 30D.

Function		N = 30			N = 60			N = 90		
		Mean	**Std.**	**Median**	**Mean**	**Std.**	**Median**	**Mean**	**Std.**	**Median**
Unimodal Functions	F_1	7.3880×10^3	6.7454×10^3	3.8861×10^3	5.4449×10^3	5.6765×10^3	3.5046×10^3	4.6746×10^3	1.4123×10^3	5.7977×10^3
	F_2	8.3146×10^4	1.7206×10^4	8.1638×10^4	7.7592×10^4	1.1306×10^4	7.5372×10^4	6.6083×10^4	6.2693×10^4	1.4721×10^4
Multimodal Functions	F_3	4.9649×10^2	3.0714×10^1	4.9252×10^2	4.9179×10^2	2.2574×10^1	4.8842×10^2	4.8495×10^2	4.8633×10^2	2.7164×10^1
	F_4	6.2601×10^2	6.4823×10^1	6.0346×10^2	6.7939×10^2	5.7299×10^1	7.0420×10^2	6.9770×10^2	7.0603×10^2	3.7821×10^1
	F_5	6.0272×10^2	2.7117	6.0198×10^2	6.0004×10^2	9.8761×10^{-2}	6.0001×10^2	6.0004×10^2	6.0003×10^2	2.9621×10^{-2}
	F_6	8.9593×10^2	6.9389×10^1	8.9541×10^2	9.5491×10^2	2.8263×10^1	9.6248×10^2	9.5526×10^2	9.5864×10^2	1.6341×10^1
	F_7	9.3177×10^2	7.0344×10^1	9.0080×10^2	9.9036×10^2	5.1629×10^1	1.0037×10^3	1.0078×10^3	1.0112×10^3	3.0123×10^1
	F_8	1.3907×10^3	5.7378×10^2	1.1280×10^3	9.0601×10^2	7.9865	9.0440×10^2	9.0141×10^2	9.0101×10^2	1.4670
	F_9	8.1898×10^3	1.1729×10^3	8.4436×10^3	8.4277×10^3	3.4190×10^2	8.4837×10^3	8.2718×10^3	8.2482×10^3	3.6665×10^2
Hybrid Functions	F_{10}	1.1835×10^3	5.8982×10^1	1.1776×10^3	1.2044×10^3	5.0040×10^1	1.1950×10^3	1.2334×10^3	1.2427×10^3	3.2662×10^1
	F_{11}	9.5317×10^5	1.0400×10^6	6.3189×10^5	2.6591×10^5	2.8870×10^5	1.5407×10^5	2.4397×10^5	1.6314×10^5	2.1469×10^5
	F_{12}	2.0148×10^4	1.8459×10^4	1.4413×10^4	1.3349×10^4	1.3905×10^4	8.0189×10^3	1.6795×10^4	1.1447×10^4	1.6274×10^4
	F_{13}	8.1822×10^4	9.6417×10^4	4.7288×10^4	3.847×10^4	3.3347×10^4	3.6679×10^4	2.5751×10^4	1.4199×10^4	2.4443×10^4
	F_{14}	1.0747×10^4	9.7030×10^3	7.1497×10^3	1.1307×10^4	1.0356×10^4	7.4763×10^3	1.4489×10^4	1.2136×10^4	1.1371×10^4
	F_{15}	2.9198×10^3	5.0390×10^2	3.0275×10^3	3.1022×10^3	4.1633×10^2	3.2082×10^3	3.1699×10^3	3.2143×10^3	2.012×10^2
	F_{16}	2.0399×10^3	1.1135×10^2	2.0585×10^3	1.9230×10^3	1.4829×10^2	1.9132×10^3	1.9150×10^3	1.8817×10^3	1.2865×10^2
	F_{17}	7.7814×10^5	1.0304×10^6	4.2460×10^5	1.1602×10^6	8.7514×10^5	9.2794×10^5	1.8026×10^6	1.3069×10^6	1.6463×10^6
	F_{18}	1.2889×10^4	1.1564×10^4	8.2227×10^3	1.4547×10^4	1.5462×10^4	8.2788×10^3	1.1386×10^4	8.5105×10^3	9.7387×10^3
	F_{19}	2.4143×10^3	2.1220×10^2	2.4204×10^3	2.2592×10^3	1.7011×10^2	2.2061×10^3	2.2178×10^3	2.1996×10^3	2.0358×10^2
Composition Functions	F_{20}	2.4371×10^3	6.9269×10^1	2.4561×10^3	2.4795×10^3	5.1795×10^1	2.4966×10^3	2.4993×10^3	2.5033×10^3	1.7737×10^1
	F_{21}	5.6464×10^3	3.5173×10^3	4.3274×10^3	3.4979×10^3	2.7257×10^3	2.3000×10^3	2.5517×10^3	2.3000×10^3	1.3769×10^3
	F_{22}	2.7655×10^3	5.7598×10^1	2.7475×10^3	2.7889×10^3	7.8776×10^1	2.8320×10^3	2.8359×10^3	2.8518×10^3	5.4288×10^1
	F_{23}	2.9328×10^3	6.7951×10^1	2.9132×10^3	3.0039×10^3	5.7480×10^1	3.0292×10^3	3.0291×10^3	3.0306×10^3	1.4249×10^1
	F_{24}	2.9019×10^3	1.9859×10^1	2.8905×10^3	2.8867×10^3	1.8345	2.8871×10^3	2.8878×10^3	2.8871×10^3	4.8380
	F_{25}	4.5451×10^3	8.7765×10^2	4.6629×10^3	4.6645×10^3	9.0179×10^2	4.5846×10^3	4.9001×10^3	5.2268×10^3	8.6492×10^2
	F_{26}	3.2483×10^3	1.9161×10^1	3.2453×10^3	3.2207×10^3	1.5836×10^1	3.2194×10^3	3.2151×10^3	3.2136×10^3	1.0483×10^1
	F_{27}	3.2433×10^3	2.8094×10^1	3.2344×10^3	3.2195×10^3	3.8101×10^1	3.2183×10^3	3.2019×10^3	3.2045×10^3	3.5352×10^1
	F_{28}	3.7791×10^3	2.2075×10^2	3.7431×10^3	3.6631×10^3	2.0532×10^2	3.5980×10^3	3.6891×10^3	3.6264×10^3	2.0704×10^2
	F_{29}	1.0876×10^4	3.8180×10^3	1.0020×10^4	1.1098×10^4	5.8710×10^3	8.8924×10^3	1.3314×10^4	1.101×10^4	7.2877×10^3

Table 14. Results of different populations with 50D.

Function		N = 30			N = 60			N = 90		
		Mean	**Std.**	**Median**	**Mean**	**Std.**	**Median**	**Mean**	**Std.**	**Median**
Unimodal Functions	F_1	8.0698×10^6	2.5848×10^7	8.5769×10^4	5.6533×10^4	5.4529×10^4	3.8752×10^4	1.6929×10^6	2.5291×10^6	7.7395×10^5
	F_2	2.1874×10^5	4.4722×10^4	2.1759×10^5	2.086×10^5	2.5998×10^4	2.0824×10^5	1.9983×10^5	2.4721×10^4	1.9739×10^5
Multimodal Functions	F_3	5.8938×10^2	7.0100×10^1	5.9366×10^2	5.4685×10^2	4.6212×10^1	5.4588×10^2	5.8317×10^2	4.8414×10^1	5.7983×10^2
	F_4	7.8413×10^2	1.2354×10^2	7.7960×10^2	9.5733×10^2	2.7532×10^1	9.6231×10^2	9.4155×10^2	4.0125×10^1	9.5446×10^2
	F_5	6.0651×10^2	3.0238	6.0599×10^2	6.0357×10^2	1.7040	6.0326×10^2	6.0443×10^2	1.9290	6.0391×10^2
	F_6	1.2372×10^3	1.7023×10^2	1.2142×10^3	1.2462×10^3	2.7839×10^1	1.2492×10^3	1.2554×10^3	3.1171×10^1	1.2627×10^3
	F_7	1.0984×10^3	1.3220×10^2	1.0452×10^3	1.2465×10^3	5.9646×10^1	1.2548×10^3	1.2486×10^3	2.7830×10^1	1.2516×10^3
	F_8	5.5744×10^3	3.6592×10^3	5.0946×10^3	2.1500×10^3	9.4137×10^2	1.821×10^3	2.2068×10^3	7.1179×10^2	1.9962×10^3
	F_9	1.4690×10^4	1.3087×10^3	1.5137×10^4	1.4969×10^4	4.2448×10^2	1.5050×10^4	1.4696×10^4	5.2522×10^2	1.4790×10^4
Hybrid Functions	F_{10}	1.5325×10^3	4.1856×10^2	1.4082×10^3	1.4942×10^3	1.2734×10^2	1.4626×10^3	1.5806×10^3	1.4222×10^2	1.5572×10^3
	F_{11}	6.0038×10^6	3.9285×10^6	5.2296×10^6	4.6185×10^6	2.6767×10^6	4.4667×10^6	6.7136×10^6	3.4054×10^6	6.9069×10^6
	F_{12}	8.8823×10^3	8.3569×10^3	5.8793×10^3	8.9058×10^3	1.0131×10^4	3.3792×10^3	6.8728×10^3	5.1488×10^3	5.4100×10^3
	F_{13}	7.1093×10^5	7.0911×10^5	4.7577×10^5	1.764×10^5	1.5576×10^5	1.4294×10^5	1.8604×10^5	1.5231×10^5	1.2970×10^5
	F_{14}	9.5976×10^3	7.5456×10^3	9.4456×10^3	6.2588×10^3	4.800×10^3	4.5249×10^3	9.6673×10^3	5.6008×10^3	8.5047×10^3
	F_{15}	4.1961×10^3	1.1946×10^3	4.6432×10^3	4.9000×10^3	5.0821×10^2	5.0144×10^3	5.0119×10^3	4.0126×10^2	5.0726×10^3
	F_{16}	3.6761×10^3	5.0328×10^2	3.8736×10^3	3.7952×10^3	4.5891×10^2	3.9097×10^3	3.917×10^3	1.8772×10^2	3.9074×10^3
	F_{17}	3.7145×10^6	3.4216×10^6	2.1930×10^6	6.1150×10^6	5.1799×10^6	5.0918×10^6	6.2919×10^6	3.6692×10^6	6.0346×10^6
	F_{18}	1.7769×10^4	1.4976×10^4	1.6746×10^4	1.2614×10^4	1.0910×10^4	1.0209×10^4	1.7584×10^4	1.2659×10^4	1.5127×10^4
	F_{19}	3.9894×10^3	3.1532×10^2	4.0624×10^3	4.0121×10^3	1.7131×10^2	4.0342×10^3	3.9080×10^3	1.6453×10^2	3.9554×10^3
Composition Functions	F_{20}	2.6289×10^3	1.4251×10^2	2.6798×10^3	2.7164×10^3	7.8921×10^1	2.7452×10^3	2.7498×10^3	2.2452×10^1	2.7450×10^3
	F_{21}	1.6405×10^4	9.7241×10^2	1.6771×10^4	1.5945×10^4	2.6014×10^3	1.6382×10^4	1.5775×10^4	2.5859×10^3	1.6241×10^4
	F_{22}	2.9959×10^3	1.2535×10^2	2.9727×10^3	3.1145×10^3	1.0888×10^2	3.1561×10^3	3.1700×10^3	4.3416×10^1	3.1780×10^3
	F_{23}	3.2918×10^3	1.1679×10^2	3.3428×10^3	3.3416×10^3	2.6905×10^1	3.3404×10^3	3.3468×10^3	1.7065×10^1	3.3473×10^3
	F_{24}	3.0860×10^3	3.2256×10^1	3.0836×10^3	3.0413×10^3	3.4239×10^1	3.0383×10^3	3.0566×10^3	3.3287×10^1	3.0536×10^3
	F_{25}	6.5998×10^3	1.1763×10^3	6.0904×10^3	7.6085×10^3	1.0757×10^3	7.9554×10^3	7.7630×10^3	1.2142×10^3	8.2386×10^3
	F_{26}	3.4756×10^3	9.5517×10^1	3.4703×10^3	3.3717×10^3	7.7598×10^1	3.3570×10^3	3.3469×10^3	4.9444×10^1	3.3398×10^3
	F_{27}	3.3779×10^3	5.7735×10^1	3.3638×10^3	3.3182×10^3	2.8599×10^1	3.3120×10^3	3.3166×10^3	2.8689×10^1	3.3176×10^3
	F_{28}	4.3124×10^3	4.0225×10^2	4.3761×10^3	4.7976×10^3	7.7415×10^2	4.6373×10^3	4.9083×10^3	6.1360×10^2	5.1620×10^3
	F_{29}	1.2805×10^6	3.1411×10^5	1.2535×10^6	1.1792×10^6	4.2818×10^5	1.1050×10^6	1.0317×10^6	2.3300×10^5	9.6863×10^5

Table 15. Results of different populations with 100D.

Function		N = 30			N = 60			N = 90		
		Mean	**Std.**	**Median**	**Mean**	**Std.**	**Median**	**Mean**	**Std.**	**Median**
Unimodal Functions	F_1	2.4262×10^9	2.1697×10^9	1.4349×10^9	2.1735×10^9	1.6368×10^9	1.9405×10^9	4.8626×10^9	2.2534×10^9	4.5258×10^9
	F_2	6.5836×10^5	8.5490×10^4	6.5079×10^5	6.0051×10^5	4.8053×10^4	5.9572×10^5	5.6070×10^5	5.0478×10^4	5.6712×10^5
Multimodal Functions	F_3	1.1402×10^3	1.6994×10^2	1.1296×10^3	1.0799×10^3	1.7728×10^2	1.0504×10^3	1.5534×10^3	6.5074×10^2	1.409×10^3
	F_4	1.4523×10^3	3.0949×10^2	1.5810×10^3	1.5998×10^3	1.3127×10^2	1.6277×10^3	1.6358×10^3	6.5052×10^1	1.6395×10^3
	F_5	6.2673×10^2	7.8883	6.2470×10^2	6.2634×10^2	6.3532	6.2555×10^2	6.3272×10^2	6.1965	6.3235×10^2
	F_6	2.4105×10^3	3.2586×10^2	2.3947×10^3	2.2629×10^3	1.4796×10^2	2.2637×10^3	2.3654×10^3	1.2481×10^2	2.3764×10^3
	F_7	1.7870×10^3	2.8372×10^2	1.9033×10^3	1.9130×10^3	1.4320×10^2	1.9451×10^3	1.9346×10^3	6.5322×10^1	1.9489×10^3
	F_8	3.3261×10^4	1.0941×10^4	3.0414×10^4	2.4906×10^4	6.5603×10^3	2.4154×10^4	2.8735×10^4	8.0231×10^3	2.9131×10^4
	F_9	3.2281×10^4	1.4270×10^3	3.2546×10^4	3.2493×10^4	4.6601×10^2	3.2445×10^4	3.2077×10^4	5.5939×10^2	3.2266×10^4
Hybrid Functions	F_{10}	1.2551×10^5	3.0498×10^4	1.2013×10^5	1.2502×10^5	2.2621×10^4	1.2619×10^5	1.2094×10^5	1.9187×10^4	1.1893×10^5
	F_{11}	1.0159×10^8	5.6502×10^7	9.3947×10^7	1.0162×10^8	4.2322×10^7	1.0274×10^8	3.5831×10^8	1.9544×10^8	3.1847×10^8
	F_{12}	1.1851×10^4	8.1518×10^3	9.4589×10^3	9.3345×10^3	8.3468×10^3	5.5971×10^3	9.3405×10^3	7.8067×10^3	5.2999×10^3
	F_{13}	2.9782×10^6	2.0700×10^6	2.4561×10^6	2.8133×10^6	1.3232×10^6	2.6904×10^6	7.9611×10^6	6.1426×10^6	5.2904×10^6
	F_{14}	6.1051×10^3	4.3314×10^3	4.9656×10^3	7.0866×10^3	5.5325×10^3	5.6302×10^3	4.8590×10^3	3.1298×10^3	3.8430×10^3
	F_{15}	9.8380×10^3	2.2106×10^3	1.0919×10^4	1.0982×10^4	9.0023×10^2	1.1099×10^4	1.1047×10^4	3.8279×10^2	1.1076×10^4
	F_{16}	7.3028×10^3	1.2345×10^3	7.7119×10^3	7.5733×10^3	7.8489×10^2	7.7642×10^3	7.5426×10^3	3.3576×10^2	7.6183×10^3
	F_{17}	9.2284×10^6	7.9516×10^6	6.2699×10^6	1.5234×10^7	8.0396×10^6	1.2651×10^7	2.1013×10^7	1.3007×10^7	1.9431×10^7
	F_{18}	1.6481×10^4	4.0296×10^4	4.9390×10^3	8.7120×10^3	9.0183×10^3	3.5736×10^3	9.2523×10^3	7.0218×10^3	6.9492×10^3
	F_{19}	7.6005×10^3	3.2761×10^2	7.6956×10^3	7.5911×10^3	2.1343×10^2	7.5550×10^3	7.5043×10^3	1.9804×10^2	7.5278×10^3
Composition Functions	F_{20}	3.2061×10^3	3.0861×10^2	3.1123×10^3	3.451×10^3	8.7610×10^1	3.4666×10^3	3.4806×10^3	6.2116×10^1	3.4850×10^3
	F_{21}	3.4888×10^4	6.8231×10^2	3.4930×10^4	3.4561×10^4	6.0301×10^2	3.4606×10^4	3.4332×10^4	7.2741×10^2	3.4458×10^4
	F_{22}	3.4270×10^3	1.0831×10^2	3.409×10^3	3.7837×10^3	2.2633×10^2	3.7978×10^3	3.906×10^3	1.5031×10^2	3.9565×10^3
	F_{23}	4.1427×10^3	2.2216×10^2	4.1747×10^3	4.3501×10^3	2.5350×10^2	4.4379×10^3	4.4577×10^3	1.1226×10^2	4.4796×10^3
	F_{24}	3.9451×10^3	2.4271×10^2	3.8940×10^3	$3.8960 \times 10^3 3$	2.6916×10^2	3.8134×10^3	4.2877×10^3	2.7775×10^2	4.2707×10^3
	F_{25}	1.4253×10^4	2.2698×10^3	1.4234×10^4	1.6057×10^4	2.3513×10^3	1.7050×10^4	1.7742×10^4	9.7393×10^2	1.7790×10^4
	F_{26}	3.6732×10^3	1.119×10^2	3.6588×10^3	3.6297×10^3	9.8681×10^1	3.6233×10^3	3.8026×10^3	1.3702×10^2	3.7729×10^3
	F_{27}	4.4414×10^3	7.1057×10^2	4.1383×10^3	4.0945×10^3	3.5898×10^2	3.9512×10^3	4.6269×10^3	6.2025×10^2	4.4988×10^3
	F_{28}	7.8602×10^3	1.3546×10^3	7.4896×10^3	1.0043×10^4	1.179×10^3	1.0306×10^4	1.0371×10^4	5.4174×10^3	1.0520×10^4
	F_{29}	1.0003×10^5	6.6358×10^4	7.9011×10^4	2.0147×10^5	2.0424×10^5	1.1431×10^5	6.3088×10^5	5.7825×10^5	4.1689×10^5

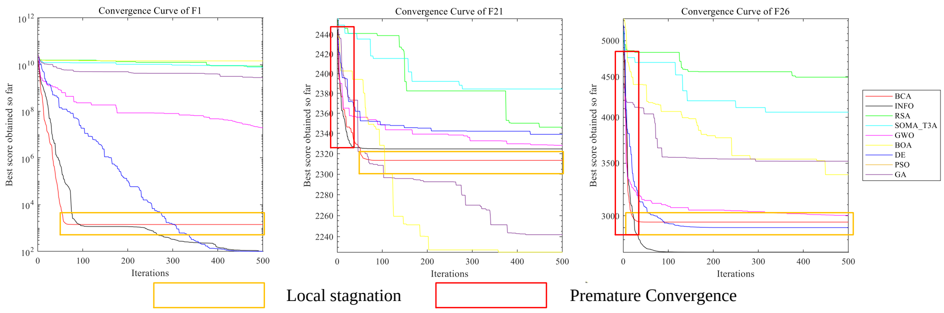


Figure 14. Convergence curve of the BCA for solving simple unimodal functions.

6. Real-World Engineering Problems

6.1. Optimization Process

In addressing engineering design problems, the BCA leverages its optimization mechanisms to identify optimal design parameter combinations, effectively satisfying the multiple objectives and constraints inherent in engineering design scenarios. The specific solution process can be summarized as follows.

(1) According to the specific engineering problem, the objective function and constraint conditions are defined to develop the mathematical modeling.

(2) An optimization algorithm is used to find the optimal feasible solution (i.e., a set of design parameters) step by step in each iteration.

(3) Evaluate the effect of the final design solution and check that it satisfies all constraints.

6.2. Tension/Compression Spring Design Problem

The Tension/Compression Spring Design problem (T/CSD) is introduced in Ref. [27], as shown in Figure 15. The problem variables used to design the problem are mean coil diameter (D), wire diameter (d), and several active coils (N). This problem can be expressed as follows:

Consider: x = $[x_1, x_2, x_3] = [d, D, N]$

Minimize: $f(x) = (N+2)Dd^2$

Subject to:

$$g_1(x) = 1 - \frac{D^3 N}{71785D^4} \leq 0$$

$$g_2(x) = \frac{4D^2 - dD}{12566(Dd^3 - d^4)} + \frac{1}{5108d^2} - 1 \leq 0$$

$$g_3(x) = 1 - \frac{140.45d}{D^2 N} \leq 0$$

$$g_4(x) = \frac{D+d}{1.5} - 1 \leq 0$$

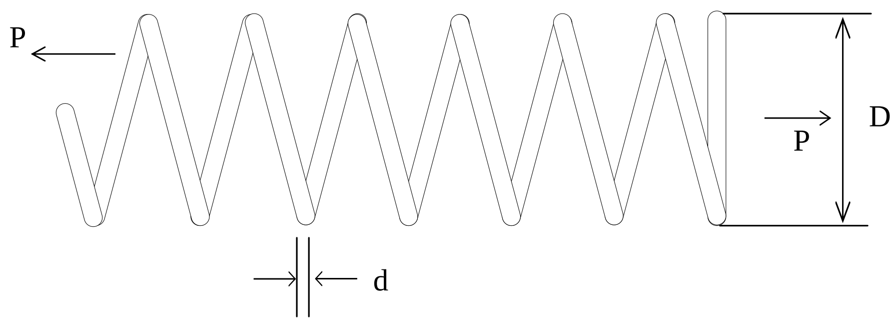


Figure 15. The Tension/Compression Spring Design problem (T/CSD).

Table 16 displays the experimental results for comparing the BCA with other algorithms. Although the BCA did not achieve the best results when compared with traditional algorithms, such as PSO, it can still outperform some methods in terms of parameter settings, such as the RSA.

Table 16. Results of Tension/Compression Spring Design (T/CSD) problem for comparative algorithms.

	Variables			g				f_{cost}			
	d	**D**	**N**	g_1	g_2	g_3	g_4	**Mean**	**Std.**	**Best**	**Worst**
BCA	0.058	0.589	4.682	−0.008	−0.032	−4.692	−0.568	0.013	0.001	0.01	0.015
GWO	0.05	0.3744	8.5615	−0.0013	-1.32×10^{-4}	−4.8521	−0.717	0.0099	6.77×10^{-6}	0.0099	0.0099
PSO	0.0539	0.4120	8.6554	-2.22×10^{-16}	−0.1220	−4.1508	−0.6894	0.0135	0.0010	0.0127	0.0175
RSA	0.050	0.336	13.077	−0.085	−0.100	−3.851	−0.742	0.013	0.001	0.011	0.013
GA	0.0506	0.3862	14.5477	−0.7745	−0.0082	−2.2784	−0.7088	0.0185	0.0037	0.012	0.0285
DE	0.0517	0.3567	11.2913	-4.43×10^{-9}	−0.1343	−4.0537	−0.7278	0.0127	2.03×10^{-5}	0.0127	0.0128
INFO	0.0533	0.4567	6.0900	-9.52×10^{-9}	-4.25×10^{-10}	−4.8952	−0.6600	0.0101	0.0003	0.0099	0.0108
BOA	0.0503	0.3740	10.7850	−0.2279	−0.0185	−3.6833	−0.7172	0.0118	0.0010	0.0010	0.0151

6.3. Gear Train Design Problem

The gear train design problem has four integer variables as a discrete problem [28]. Figure 16 shows details of the gear train design problem, which can be defined by Equation (14).

$$GearRatio = \frac{T_d T_b}{T_a T_f} \tag{14}$$

where T_i denotes the number of teeth of the gearwheel i, and they are all integers varying in the range 12–60. The mathematical formulation is defined as follows:

$$f(T_d, T_b, T_a, T_f) = (\frac{1}{6.931} - \frac{T_d T_b}{T_a T_f}) \tag{15}$$

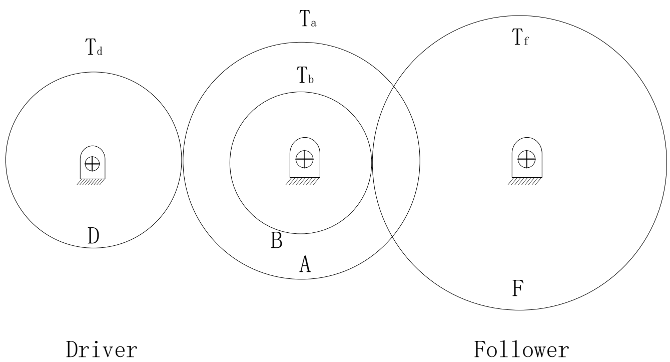


Figure 16. The gear train design problem.

The experimental results are shown in Table 17 to better verify the applicability of the proposed BCA and show that the BCA can obtain the best experimental results and has a relatively stable optimal result. Meanwhile, these experimental results demonstrate that the BCA has certain advantages in solving engineering design problems.

The advantages of the BCA can be seen from the tension/compression spring design (T/CSD) and gear train design problems. The experimental results show that the BCA not only has certain advantages in solving engineering case design problems but can also quickly find the optimal solution in solving unconstrained and constrained problems. To summarize the above experimental results, compared with other algorithms, the BCA can obtain the global optimal solution, and its mechanisms can help to avoid local stagnation and achieve fast convergence with lowest design cost for engineering design problems.

Table 17. Results of Tension/Compression Gear Train Design problem for comparative algorithms.

	Variables				f_{cost}			
	T_d	T_b	T_a	T_f	**Mean**	**Std.**	**Best**	**Worst**
BCA	49	19	18	49	2.86×10^{-12}	9.63×10^{-12}	1.43×10^{-24}	3.85×10^{-11}
GWO	38	20	16	59	6.56×10^{-12}	8.69×10^{-12}	1.02×10^{-15}	3.20×10^{-11}
PSO	57	27	14	47	4.79×10^{-24}	2.07×10^{-23}	0	1.09×10^{-22}
RSA	37	14	16	42	6.69×10^{-8}	1.53×10^{-7}	1.72×10^{-12}	6.08×10^{-7}
GA	47	17	18	47	5.66×10^{-9}	2.65×10^{-8}	1.29×10^{-20}	1.45×10^{-7}
DE	42	18	20	60	6.64×10^{-3}	8.82×10^{-3}	1.13×10^{-7}	2.86×10^{-2}
INFO	44	25	15	58	1.35×10^{-26}	7.42×10^{-26}	0	4.06×10^{-25}
BOA	56	17	24	55	4.97×10^{-3}	8.10×10^{-3}	2.54×10^{-8}	3.75×10^{-2}

7. BCA for Training MLPs

To improve its capability of solving classification problems, the proposed BCA is used to optimize the weights and biases term of MLP to avoid local stagnation in the classification process, and it will improve the classification rate and minimize the error rate. This section selects three standard datasets from the University of California Irvine (UCI) machine-learning repository (https://archive.ics.uci.edu/, accessed on 1 January 2024), including XOR [29], Ballon [30], and Tic-Tac-Toe Endgame [31]. The BCA algorithm optimizes the weights and biases of Multi-Layer Perceptron (MLP) to construct the classification model.

7.1. Optimization Process

The BCA can be effectively utilized to optimize the training process of MLPs by fine-tuning weights and biases to achieve enhanced model performance. Through its robust global search capabilities, the BCA avoids premature convergence to local optima, ensuring better exploration of the solution space. Its adaptive balance between exploration and exploitation enables the identification of optimal parameter configurations, which improves the accuracy and generalization of the MLP. The main process can be summarized as follows.

Problem Formulation. The goal is to optimize the weights and biases of an MLP to minimize a predefined loss function (Mean Squared Error, *MSE*) over a given dataset. *MSE* [92,93], as the criteria to evaluate the performance, is computed by Equation (16).

$$MSE = \sum_{i=1}^{m} (o_i^k - d_i^k)^2 \tag{16}$$

where m is the number of outputs, d_i^k and o_i^k are the desired output and actual output of the i_{th} input using the k_{th} training sample, and the $\overline{MSE}$ is average MSE computed by Equation (17) to enhance fairness.

$$\overline{MSE} = \sum_{k=1}^{s} \frac{\sum_{i=1}^{m} (o_i^k - d_i^k)^2}{s} \tag{17}$$

where s is the number of training samples. The training of an MLP consists of multiple variables and functions, where $\overline{MSE}$ for the BCA is achieved by Equation (18).

$$minimize : F(\overrightarrow{V}) = \overline{MSE} \tag{18}$$

In addition to *MSE*, the test error is employed for the approximation problem, and the classification accuracy rate is computed by Equation (19).

$$Accuracy\ rate = \frac{Number\ of\ correctly\ classified\ objects}{Number\ of\ objects\ in\ the\ dataset} \tag{19}$$

Initialization. The initial population is randomly generated within a predefined range, ensuring diverse starting points across the parameter space. Each candidate solution in the BCA's population represents a set of MLP weights and biases.

Fitness Evaluation. Each candidate solution is decoded into the MLP's weight and bias, followed by forward propagation of a training data batch using these parameters. The loss is then calculated by comparing the MLP's output to the truth labels, and this loss serves as the fitness value, where lower values indicate better performance.

Iterative Process. The iterative process involves repeating the besiege and conquer phases until the maximum number of iterations is reached, ensuring the optimization criteria are met.

Result Extraction. After the optimization process concludes, the best solution is mapped back to the MLP's parameters, and the optimized model is validated on a test dataset to evaluate its generalization capability.

7.2. Experimental Results on Three Datasets

7.2.1. Xor Dataset

The XOR dataset [29] has three different characters as input and one output. The results of this dataset are illustrated in Table 18. The best classification rate, average $\overline{MSE}$, and deviation belong to the BCA-MLP model. These results demonstrate that the BCA-MLP model has the strongest capability to avoid local stagnation. The BCA-based trainer is very competitive when compared with the other algorithms.

Table 18. Comparative results on XOR dataset.

	BCA-MLP	BOA-MLP	SMA-MLP	RSA-MLP	PSO-MLP	SCA-MLP
Classification accuracy	**96.6667%**	20.4167%	22.9167%	24.1667%	40.4167%	51.6667%
$\overline{MSE}$	**0.0004**	0.1301	0.2007	0.1605	0.1333	0.0414
Std.	**0.0008**	0.0437	0.0271	0.0389	0.0685	0.0324

7.2.2. Ballon Dataset

The Balloon dataset [30] has four features as input and two outputs. According to Table 19, the accuracy of the BCA-MLP model reaches 100%, and obtains the best $\overline{MSE}$ and standard deviation. The experimental results illustrate that the BCA-MLP model performs better than the other methods in training the MLP and can find the global optimal solution in a more stable manner.

Table 19. Comparative results on Ballon dataset.

	BCA-MLP	BOA-MLP	SMA-MLP	RSA-MLP	PSO-MLP	SCA-MLP
Classification accuracy	**100%**	85.1667%	98%	51.1667%	70.5%	100%
$\overline{MSE}$	$\mathbf{5.4761} \times 10^{-10}$	4.7226×10^{-3}	3.1037×10^{-4}	1.5287×10^{-2}	6.1734×10^{-2}	2.8029×10^{-6}
Std.	$\mathbf{2.8897} \times 10^{-9}$	9.1955×10^{-3}	8.5071×10^{-4}	1.7109×10^{-2}	6.8740×10^{-2}	3.9437×10^{-6}

7.2.3. Tic-Tac-Toe Endgame Dataset

The Tic-Tac-Toe Endgame dataset [31] encodes a complete set of possible board configurations at the end of a tic-tac-toe game, where x is assumed to play first. The goal is to win x (i.e., this is true when there are eight possible ways for x to make three times in a row). Table 20 shows that the BCA-MLP model can obtain the highest classification rate and also demonstrates that the BCA-MLP model has the stronger exploration to find the best solution.

Table 20. Comparative results on Tic-Tac-Toe Endgame dataset.

	BCA-MLP	BOA-MLP	SMA-MLP	RSA-MLP	PSO-MLP	SCA-MLP
Classification accuracy	**97.2690%**	64.4444%	81.6822%	73.4268%	93.8837%	96.2098%
$\overline{MSE}$	1.2609×10^{-2}	1.4308×10^{-2}	1.0694×10^{-2}	1.2134×10^{-2}	1.4620×10^{-2}	1.3243×10^{-2}
Std.	2.3301×10^{-3}	8.5751×10^{-4}	2.3023×10^{-3}	1.4956×10^{-3}	1.5816×10^{-3}	1.4103×10^{-3}

In summary, according to the results of the above three classification experiments, it can be seen that the BCA can train MLP well and optimize the weights and biases terms to obtain a classification model with high accuracy. The BCA, with all mechanisms, can promote MLP optimization to achieve local stagnation avoidance, fast convergence, and high accuracy.

8. Conclusions and Future Work

This paper proposes a novel BCA optimization algorithm, including besiege, conquer, balance, and feedback strategies, inspired by the soldiers and armies motivation strategy. The BCA's design incorporates promising exploration and exploitation regions in its besiege and conquer mechanisms to update the positions of the soldiers. To highlight the capability of the proposed BCA, some classical, popular meta-heuristics, such as INFO, RSA, SOMA T3AM, GWO, BOA, DE, PSO, and GA, are employed for comparison. These algorithms are tested on IEEE CEC 2017 benchmark functions to verify their performance. Four metrics (i.e., search history, average fitness function, the trajectory of the first dimension, and convergence curve) are implemented to qualitatively investigate the proposed BCA. In addition, the Friedman ranking and Wilcoxon signed-rank tests are used to quantitatively verify the efficiency of the algorithms. The comparative experimental results demonstrate that the BCA can determine the global optima for the majority of unimodal, multimodal, hybrid, and composite functions.

Furthermore, to demonstrate the excellent efficiency of the BCA in the benchmark functions, complex engineering design problems are considered in order to display its practicability in tackling real-world problems in practice, including Tension/Compression Spring Design and Gear Train Design problems. Then, the BCA trains the MLP model to handle classification problem, such as XOR, Ballon, and Tic-Tac-Toe datasets, to improve the classification accuracy. The experimental results show that the BCA classification model (BCA-MLP) is better than the comparative methods. To sum up, the above superior results are attributed to the following several aspects.

- The besiege strategy can increase population diversity to enhance the exploration capability.
- The conquer strategy facilitates exploitation and delegates to the local search.
- The balance and feedback strategies not only enhance the balance between exploitation and exploration but also help to find the best solutions.
- The introduction of parameter *BCB* assists in gradually shifting its focus from exploitation to exploration, and avoiding local stagnation.

The practical applications of the BCA extend well beyond neural networks and engineering design. Leveraging its robust exploration and exploitation capabilities, the BCA excels in identifying precise segmentation boundaries and cluster centers, particularly in scenarios involving high-dimensional and complex data distributions, such as medical image segmentation and customer segmentation. Additionally, the BCA's powerful global optimization capabilities, adaptability, and resilience make it highly suitable for addressing challenges in diverse domains, including data clustering, segmentation, and financial investment optimization. These attributes underscore its significant potential as a versatile tool in solving complex real-world optimization problems.

For future work, we will do the following. The BCA will also be applied to solve multi-objective optimization problems. Apart from this, the proposal of binary or many objective versions of the BCA could also be significant contributions.

Author Contributions: Conceptualization and supervision, J.J. and G.X.; methodology, X.M. and W.L.; writing—editing, J.T. and J.W. All authors have read and agreed to the published version of the manuscript.

Funding: The authors thank the financial support from the Foundation of the Jilin Provincial Department of Science and Technology (No. YDZJ202201ZYTS565).

Data Availability Statement: The datasets analysed during the current study are available in the University of California Irvine (UCI) machine-learning repository (https://archive.ics.uci.edu/, accessed on 1 January 2024). The code is at https://www.jianhuajiang.com/projects/besiege-and-conquer-algorithm, accessed on 1 January 2024.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Bernal, C.P.; Angel, M.; Moya, C.M. Optimizing energy efficiency in unrelated parallel machine scheduling problem through reinforcement learning. *Inf. Sci.* **2024**, *693*, 121674.
2. Mazroua, A.A.; Bartnikas, R.; Salama, M. Neural network system using the multi-layer perceptron technique for the recognition of pd pulse shapes due to cavities and electrical trees. *IEEE Trans. Power Deliv.* **1995**, *10*, 92–96.
3. Chen, C.; da Silva, B.; Yang, C.; Ma, C.; Li, J.; Liu, C. Automlp: A framework for the acceleration of multi-layer perceptron models on FPGAs for real-time atrial fibrillation disease detection. *IEEE Trans. Biomed. Circuits Syst.* **2023**, *12*, 1371–1386.
4. Rojas, M.G.; Olivera, A.C.; Vidal, P.J. A genetic operators-based ant lion optimiser for training a medical multi-layer perceptron. *Appl. Soft Comput.* **2024**, *151*, 111192.
5. Erdogmus, D.; Fontenla-Romero, O.; Principe, J.C.; Alonso-Betanzos, A.; Castillo, E. Linear-least-squares initialization of multilayer perceptrons through backpropagation of the desired response. *IEEE Trans. Neural Netw.* **2005**, *16*, 325–337.
6. Zervoudakis, K.; Tsafarakis, S. A global optimizer inspired from the survival strategies of flying foxes. *Eng. Comput.* **2023**, *39*, 1–34.
7. Połap, D.; Woźniak, M. Polar bear optimization algorithm: Meta-heuristic with fast population movement and dynamic birth and death mechanism. *Symmetry* **2017**, *9*, 203.
8. Mirjalili, S.; Mirjalili, S.M.; Lewis, A. Grey wolf optimizer. *Adv. Eng. Softw.* **2014**, *69*, 46–61.
9. Dorigo, M.; Birattari, M.; Stutzle, T. Ant colony optimization. *IEEE Comput. Intell. Mag.* **2006**, *1*, 28–39.
10. Holland, J.H. Genetic algorithms. *Sci. Am.* **1992**, *267*, 66–73.
11. Ali, A.; Assam, M.; Khan, F.U.; Ghadi, Y.Y.; Nurdaulet, Z.; Zhibek, A.; Shah, S.Y.; Alahmadi, T.J. An optimized multilayer perceptron-based network intrusion detection using gray wolf optimization. *Comput. Electr. Eng.* **2024**, *120*, 109838.
12. Li, X.-D.; Wang, J.-S.; Hao, W.-K.; Wang, M.; Zhang, M. Multi-layer perceptron classification method of medical data based on biogeography-based optimization algorithm with probability distributions. *Appl. Soft Comput.* **2022**, *121*, 108766.
13. Zervoudakis, K.; Tsafarakis, S. A mayfly optimization algorithm. *Comput. Ind. Eng.* **2020**, *145*, 106559.
14. Oyelade, O.N.; Aminu, E.F.; Wang, H.; Rafferty, K. An adaptation of hybrid binary optimization algorithms for medical image feature selection in neural network for classification of breast cancer. *Neurocomputing* **2024**, *617*, 129018.
15. Xue, Y.; Zhang, C. A novel importance-guided particle swarm optimization based on mlp for solving large-scale feature selection problems. *Swarm Evol. Comput.* **2024**, *91*, 101760.
16. Lu, Y.; Tang, Q.; Yu, S.; Cheng, L. A multi-strategy self-adaptive differential evolution algorithm for assembly hybrid flowshop lot-streaming scheduling with component sharing. *Swarm Evol. Comput.* **2025**, *92*, 101783.
17. Sörensen, K. Metaheuristics—The metaphor exposed. *Int. Trans. Oper. Res.* **2015**, *22*, 3–18.
18. Tong, M.; Peng, Z.; Wang, Q. A hybrid artificial bee colony algorithm with high robustness for the multiple traveling salesman problem with multiple depots. *Expert Syst. Appl.* **2025**, *260*, 125446.
19. Ansah-Narh, T.; Nortey, E.; Proven-Adzri, E.; Opoku-Sarkodie, R. Enhancing corporate bankruptcy prediction via a hybrid genetic algorithm and domain adaptation learning architecture. *Expert Syst. Appl.* **2024**, *258*, 125133.
20. Liu, W.; Zhang, L.; Xie, L.; Hu, T.; Li, G.; Bai, S.; Yi, Z. Multilayer perceptron neural network with regression and ranking loss for patient-specific quality assurance. *Knowl.-Based Syst.* **2023**, *271*, 110549.

21. Ahmadianfar, I.; Heidari, A.A.; Noshadian, S.; Chen, H.; Gandomi, A.H. INFO: An efficient optimization algorithm based on weighted mean of vectors. *Expert Syst. Appl.* **2022**, *195*, 116516.
22. Abualigah, L.; Abd Elaziz, M.; Sumari, P.; Geem, Z.W.; Gandomi, A.H. Reptile search algorithm (RSA): A nature-inspired meta-heuristic optimizer. *Expert Syst. Appl.* **2022**, *191*, 116158.
23. Diep, Q.B. Self-organizing migrating algorithm team to team adaptive–SOMA T3A. In Proceedings of the 2019 IEEE Congress on Evolutionary Computation (CEC), Wellington, New Zealand, 10–13 June 2019; pp. 1182–1187.
24. Arora, S.; Singh, S. Butterfly optimization algorithm: A novel approach for global optimization. *Soft Comput.* **2019**, *23*, 715–734.
25. Storn, R.; Price, K. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *J. Glob. Optim.* **1997**, *11*, 341–359.
26. Kennedy, J.; Eberhart, R. Particle swarm optimization. In Proceedings of ICNN'95-International Conference on Neural Networks, Perth, WA, Australia, 27 November–1 December 1995; Volume 4, pp. 1942–1948.
27. Belegundu, A.D.; Arora, J.S. A study of mathematical programming methods for structural optimization. *Part I Theory Int. J. Numer. Methods Eng.* **1985**, *21*, 1583–1599.
28. Prayoonrat, S.; Walton, D. Practical approach to optimum gear train design. *Comput. Des.* **1988**, *20*, 83–92.
29. McGarry, K.J.; Wermter, S.; MacIntyre, J. Knowledge extraction from radial basis function networks and multilayer perceptrons. In Proceedings of the IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No. 99CH36339), Washington, DC, USA, 10–16 July 1999; Volume 4, pp. 2494–2497.
30. Pazzani, M.J. Influence of prior knowledge on concept acquisition: Experimental and computational results. *J. Exp. Psychol. Learn. Mem. Cogn.* **1991**, *17*, 416–432.
31. Aha, D.W. Incremental constructive induction: An instance-based approach. In *Machine Learning Proceedings 1991*; Morgan Kaufmann: Burlington, MA, USA, 1991; pp. 117–121.
32. Renkavieski, C.; Parpinelli, R.S. Meta-heuristic algorithms to truss optimization: Literature mapping and application. *Expert Syst. Appl.* **2021**, *182*, 115197.
33. Jian, Z.; Zhu, G. Affine invariance of meta-heuristic algorithms. *Inf. Sci.* **2021**, *576*, 37–53.
34. Rechenberg, I. *Evolution Strategy: Optimization of Technical Systems by Means of Biological Evolution*; Fromman-Holzboog: Stuttgart, Germany, 1973; Volume 104, pp. 15–16.
35. Hillis, W.D. Co-evolving parasites improve simulated evolution as an optimization procedure. *Phys. D Nonlinear Phenom.* **1990**, *42*, 228–234.
36. Atashpaz-Gargari, E.; Lucas, C. Imperialist competitive algorithm: An algorithm for optimization inspired by imperialis-tic competition. In Proceedings of the 2007 IEEE Congress on Evolutionary Computation, Singapore, 25–28 September 2007; IEEE: Piscataway, NJ, USA, 2007; pp. 4661–4667.
37. Civicioglu, P. Transforming geocentric cartesian coordinates to geodetic coordinates by using differential search algorithm. *Comput. Geosci.* **2012**, *46*, 229–247.
38. Civicioglu, P. Backtracking search optimization algorithm for numerical optimization problems. *Appl. Math. Comput.* **2013**, *219*, 8121–8144.
39. Salimi, H. Stochastic fractal search: A powerful meta-heuristic algorithm. *Knowl.-Based Syst.* **2015**, *75*, 1–18.
40. Dhivyaprabha, T.; Subashini, P.; Krishnaveni, M. Synergistic fibroblast optimization: A novel nature-inspired computing algorithm. *Front. Inf. Technol. Electron. Eng.* **2018**, *19*, 815–833.
41. Motevali, M.M.; Shanghooshabad, A.M.; Aram, R.Z.; Keshavarz, H. Who: A new evolutionary algorithm bio-inspired by wildebeests with a case study on bank customer segmentation. *Int. J. Pattern Recognit. Artif. Intell.* **2019**, *33*, 1959017.
42. Rahman, M.C.; Rashid, A.T. A new evolutionary algorithm: Learner performance based behavior algorithm. *Egypt. Inform. J.* **2021**, *22*, 213–223.
43. Rao, R.V.; Savsani, V.J.; Vakharia, D. Teaching–learning-based optimization: A novel method for constrained mechanical design optimization problems. *Comput. Des.* **2011**, *43*, 303–315.
44. He, S.; Wu, Q.H.; Saunders, J.R. Group search optimizer: An optimization algorithm inspired by animal searching behavior. *IEEE Trans. Evol. Comput.* **2009**, *13*, 973–990.
45. Kaveh, A.; Mahdavi, V.R. Colliding bodies optimization: A novel meta-heuristic method. *Comput. Struct.* **2014**, *139*, 18–27 .
46. Kashan, A.H. League championship algorithm (LCA): An algorithm for global optimization inspired by sport champi-onships. *Appl. Soft Comput.* **2014**, *16*, 171–200.
47. Zhang, J.; Xiao, M.; Gao, L.; Pan, Q. Queuing search algorithm: A novel metaheuristic algorithm for solving engineering optimization problems. *Appl. Math. Model.* **2018**, *63*, 464–490.
48. Zhang, L.M.; Dahlmann, C.; Zhang, Y. Human-inspired algorithms for continuous function optimization. In Proceedings of the 2009 IEEE International Conference on Intelligent Computing and Intelligent Systems, Shanghai, China, 20–22 November 2009; IEEE: Piscataway, NJ, USA, 2009; Volume 1, pp. 318–321.

49. Shayeghi, H.; Dadashpour, J. Anarchic society optimization based pid control of an automatic voltage regulator (avr) system. *Electr. Electron. Eng.* **2012**, *2*, 199–207.
50. Mousavirad, S.J.; Ebrahimpour-Komleh, H. Human mental search: A new population-based meta-heuristic optimization algorithm. *Appl. Intell.* **2017**, *47*, 850–887.
51. Moghdani, R.; Salimifard, K. Volleyball premier league algorithm. *Appl. Soft Comput.* **2018**, *64*, 161–185.
52. Mohamed, A.W.; Hadi, A.A.; Mohamed, A.K. Gaining-sharing knowledge based algorithm for solving optimization problems: A novel nature-inspired algorithm. *Int. J. Mach. Learn. Cybern.* **2020**, *11*, 1501–1529.
53. Al-Betar, M.A.; Alyasseri, Z.A.A.; Awadallah, M.A.; Abu Doush, I. Coronavirus herd immunity optimizer (CHIO). *Neural Comput. Appl.* **2021**, *33*, 5011–5042.
54. Braik, M.; Ryalat, M.H.; Al-Zoubi, H. A novel meta-heuristic algorithm for solving numerical optimization problems: Ali baba and the forty thieves. *Neural Comput. Appl.* **2022**, *34*, 409–455.
55. Kirkpatrick, S.; Gelatt, C.D., Jr.; Vecchi, M.P. Optimization by simulated annealing. *Science* **1983**, *220*, 671–680.
56. Rashedi, E.; Rashedi, E.; Nezamabadi-Pour, H. A comprehensive survey on gravitational search algorithm. *Swarm Evol. Comput.* **2018**, *41*, 141–158.
57. Formato, R.A. Central force optimization. *Prog Electromagn Res.* **2007**, *77*, 425–491.
58. Mladenović, N.; Hansen, P. Variable neighborhood search. *Comput. Oper. Res.* **1997**, *24*, 1097–1100.
59. Erol, O.K.; Eksin, I. A new optimization method: Big bang–big crunch. *Adv. Eng. Softw.* **2006**, *37*, 106–111.
60. Rashedi, E.; Nezamabadi-Pour, H.; Saryazdi, S. GSA: A gravitational search algorithm. *Inf. Sci.* **2009**, *179*, 2232–2248.
61. Hatamlou, A. Black hole: A new heuristic optimization approach for data clustering. *Inf. Sci.* **2013**, *222*, 175–184.
62. Shareef, H.; Ibrahim, A.A.; Mutlag, A.H. Lightning search algorithm. *Appl. Soft Comput.* **2015**, *36*, 315–333.
63. Mirjalili, S.; Mirjalili, S.M.; Hatamlou, A. Multi-verse optimizer: A nature-inspired algorithm for global optimization. *Neural Comput. Appl.* **2016**, *27*, 495–513.
64. Kaveh, A.; Dadras, A. A novel meta-heuristic optimization algorithm: Thermal exchange optimization. *Adv. Eng. Softw.* **2017**, *110*, 69–84.
65. Faramarzi, A.; Heidarinejad, M.; Stephens, B.; Mirjalili, S. Equilibrium optimizer: A novel optimization algorithm. *Knowl.-Based Syst.* **2020**, *191*, 105190.
66. Jiang, J.; Meng, X.; Chen, Y.; Qiu, C.; Liu, Y.; Li, K. Enhancing tree-seed algorithm via feed-back mechanism for optimizing continuous problems. *Appl. Soft Comput.* **2020**, *92*, 106314.
67. Mavrovouniotis, M.; Li, C.; Yang, S. A survey of swarm intelligence for dynamic optimization: Algorithms and applications. *Swarm Evol. Comput.* **2017**, *33*, 1–17.
68. Jiang, J.; Liu, Y.; Zhao, Z. TriTSA: Triple tree-seed algorithm for dimensional continuous optimization and constrained engineering problems. *Eng. Appl. Artif. Intell.* **2021**, *104*, 104303.
69. Jiang, J.; Meng, X.; Qian, L.; Wang, H. Enhance tree-seed algorithm using hierarchy mechanism for constrained optimization problems. *Expert Syst. Appl.* **2022**, *209*, 118311.
70. Jiang, J.; Zhao, Z.; Liu, Y.; Li, W.; Wang, H. DSGWO: An improved grey wolf optimizer with diversity enhanced strategy based on group-stage competition and balance mechanisms. *Knowl.-Based Syst.* **2022**, *250*, 109100.
71. Yang, X.-S. Firefly algorithms for multimodal optimization. In *International Symposium on Stochastic Algorithms*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 169–178.
72. Pan, W.-T. A new fruit fly optimization algorithm: Taking the financial distress model as an example. *Knowl.-Based Syst.* **2012**, *26*, 69–74.
73. Mirjalili, S. The ant lion optimizer. *Adv. Eng. Softw.* **2015**, *83*, 80–98.
74. Kiran, M.S. TSA: Tree-seed algorithm for continuous optimization. *Expert Syst. Appl.* **2015**, *42*, 6686–6698.
75. Mirjalili, S. Dragonfly algorithm: A new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems. *Neural Comput. Appl.* **2016**, *27*, 1053–1073.
76. Mirjalili, S.; Lewis, A. The whale optimization algorithm. *Adv. Eng. Softw.* **2016**, *95*, 51–67.
77. Saremi, S.; Mirjalili, S.; Lewis, A. Grasshopper optimisation algorithm: Theory and application. *Adv. Eng. Softw.* **2017**, *105*, 30–47.
78. Mirjalili, S.; G.; omi, A.H.; Mirjalili, S.Z.; Saremi, S.; Faris, H.; Mirjalili, S.M. Salp swarm algorithm: A bio-inspired optimizer for engineering design problems. *Adv. Eng. Softw.* **2017**, *114*, 163–191.
79. Alsattar, H.A.; Zaidan, A.; Zaidan, B. Novel meta-heuristic bald eagle search optimisation algorithm. *Artif. Intell. Rev.* **2020**, *53*, 2237–2264.
80. Abualigah, L.; Yousri, D.; Abd Elaziz, M.; Ewees, A.A.; Al-Qaness, M.A.; Gandomi, A.H. Aquila optimizer: A novel meta-heuristic optimization algorithm. *Comput. Ind. Eng.* **2021**, *157*, 107250.
81. Połap, D.; Woźniak, M. Red fox optimization algorithm. *Expert Syst. Appl.* **2021**, *166*, 114107.
82. Bairwa, A.K.; Joshi, S.; Singh, D. Dingo Optimizer: A nature-inspired metaheuristic approach for engineering problems. *Math. Probl. Eng.* **2021**, *2021*, 2571863.

83. Braik, M.S. Chameleon swarm algorithm: A bio-inspired optimizer for solving engineering design problems. *Expert. Syst. Appl.* **2021**, *174*, 114685.
84. Braik, M.; Hammouri, A.; Atwan, J.; Al-Betar, M.A.; Awadallah, M.A. White shark optimizer: A novel bio-inspired meta-heuristic algorithm for global optimization problems. *Knowl.-Based Syst.* **2022**, *243*, 108457.
85. Guan, X.; Hu, T.; Zhang, Z.; Wang, Y.; Liu, Y.; Wang, Y.; Hao, J.; Li, G. Multi-layer perceptron-particle swarm optimization: A lightweight optimization algorithm for the model predictive control local planner. *Int. J. Adv. Robot. Syst.* **2024**, *21*, 17298806241301581.
86. Kumari, V.; Kumar, P.R. Optimization of multi-layer perceptron neural network using genetic algorithm for arrhythmia classification. *Communications* **2015**, *3*, 150–157.
87. Piotrowski, A.P. Differential evolution algorithms applied to neural network training suffer from stagnation. *Appl. Soft Comput.* **2014**, *21*, 382–406.
88. Stober, E. O. The hunger games: Weaponizing food. *Mark. Ideas* **2022**, *37*, 1–8.
89. Madhi, W.T.; Abd, S.M. Characteristics of the hero in contemporary arabic poetry. *PalArch's J. Archaeol. Egypt/Egyptology* **2021**, *18*, 1846–1862.
90. Janson, H.W. The equestrian monument from cangrande della scala to peter the great. In *Aspects of the Renaissance*; University of Texas Press: Austin, TX, USA, 2021; pp. 73–86.
91. Sanders, R. The pareto principle: Its use and abuse. *J. Serv. Mark.* **1987**, *1*, 37–40.
92. Hernández, G.; Zamora, E.; Sossa, H.; Téllez, G.; Furlán, F. Hybrid neural networks for big data classification. *Neurocomputing* **2020**, *390*, 327–340.
93. Meng, X.; Jiang, J.; Wang, H. AGWO: Advanced GWO in multi-layer perception optimization. *Expert Syst. Appl.* **2021**, *173*, 114676.