2025 16th International Conference on Electrical and Electronics Engineering (ELECO) | 979-8-3315-4694-6/25/$31.00 ©2025 IEEE | DOI: 10.1109/ELECO69582.2025.11329296

A Modified Artificial Bee Colony Algorithm for Dynamic Optimization Problems

Abdulbaki KARTAL[1,2], Bahriye AKAY[1], Dervis KARABOGA[1]

[1]Erciyes University, Faculty of Engineering, Department of Computer Engineering, Kayseri, Türkiye
[2]Erciyes University, Graduate School of Natural and Applied Sciences, Department of Computer Engineering, Kayseri, Türkiye
abdulbakikartal@erciyes.edu.tr, bahriye@erciyes.edu.tr, karaboga@erciyes.edu.tr

Abstract

Dynamic optimization problems (DOPs) are characterized by time-varying search spaces, which pose challenges to traditional algorithms. To address these challenges, a modified Artificial Bee Colony (ABC) algorithm is proposed in which the best solution is re-evaluated at the beginning of each cycle. When a change is detected, all candidate solutions are updated, the best one is retained, and the remaining solutions are transferred to the scout phase to explore new solutions in the search space. New solutions are compared, and the best one is preserved. Experiments were conducted on the dynamic Rastrigin, Rosenbrock, Sphere, and Griewank functions. The proposed method was shown to achieve faster convergence and higher solution quality compared to the standard ABC, while diversity and convergence stability were maintained relative to recent DOP-ABC variants.

1. Introduction

Optimization problems in time-varying environments, where the problem definition or the fitness landscape changes over time, are referred to as Dynamic Optimization Problems (DOPs). Unlike static problems, DOPs require not only locating the global optimum but also adapting rapidly to changes. For this reason, DOPs are of critical importance in many real-time applications such as industrial process control, resource scheduling, and evolutionary robotics.

Among nature-inspired metaheuristics, the Artificial Bee Colony (ABC) algorithm has attracted significant attention due to its simple structure and effective balance between exploration and exploitation [1-3]. The ABC algorithm mimics the foraging behavior of honeybees through three types of agents - employed bees, onlooker bees, and scout bees - whose interactions collectively generate new solutions and update the population [1].

In this study, we modified scout bee phase within the ABC framework to be able to explicitly detect environmental changes in dynamic problems. The goal is to maintain computational efficiency while enhancing both population diversity and convergence stability. In the proposed approach, the best solution is re-evaluated at the beginning of each cycle. When a change is detected, solutions are adaptively re-generated, thereby minimizing the loss of previously acquired information.

The remainder of this paper is organized as follows: In Section 2, related work is presented, Section 3 explains the proposed method, Section 4 reports the experimental study. Section 5 summarizes the results, and section 6 presents the conclusion.

2. Related Work

Drawing on searches across Google Scholar, ScienceDirect, IEEE Xplore, and Web of Science using the keywords ABC, DOP, and Modified ABC, the most pertinent 27 studies were identified in the literature, some of them are summarized below for use in this paper.

Nishida (2013) adapted ABC to time-varying functions by adding two simple steps: (i) per-cycle re-evaluation of employed solutions within the employed phase, and (ii) time-step best update by removing the "($f_{ij} > f_{best}$)" condition. These changes refresh information immediately when the objective shifts, introduce no new parameters, add negligible overhead, preserve baseline ABC on static tasks, and markedly improve tracking on dynamic uni/multimodal functions [4].

Jia *et al.* (2016) proposed a multi-swarm ABC for dynamic environments: change is detected by re-evaluating the global best each generation, the population is partitioned into sub-swarms by position, and the onlooker phase uses a local search radius ($r_x = 0.1, (u-1)/p_s$); the scout bee phase is omitted in dynamics since poor solutions may become efficient after changes. On the Moving Peaks benchmark, it outperforms canonical ABC in offline error, convergence speed, and robustness [5].

Nakano *et al.* (2015) augmented ABC with a memory scheme for dynamic environments: employed and onlooker pools are decoupled, the onlooker phase uses a greedy selection, and a similarity threshold stores non-duplicate elites in memory. Each cycle re-evaluates a few candidates to detect changes; upon detection, stored elites overwrite employed solutions while the rest are reinitialized, effectively removing the scout phase. Results show faster adaptation and stable convergence on dynamic benchmarks [6].

Bose *et al. (2012)* adapted ABC to dynamic environments via multiple sub-colonies and a perturbation strategy pool (source-rand/1, rand/1, rand-fittest/1), while maintaining diversity with quantum individuals, Levy-based Brownian moves, exclusion, and improvement-based re-initialization. Change detection is triggered by re-evaluating a non-optimizing reference vector each generation; upon detection, parameters/sub-colonies are re-initialized to promptly track shifting optima [7].

Jia (2021) integrated ABC with a culture (belief) space atop a multi-swarm framework: situational, normative, and temporal knowledge guide sub-swarms; the onlooker update is scaled by normative ranges, and if sub-swarm elites collide within an exclusion radius, the weaker one is re-initialized around temporal knowledge. Environmental change is detected when the global best deteriorates. The multi-swarm design removes

Authorized licensed use limited to: ULAKBIM UASL - Erciyes Universitesi. Downloaded on February 19,2026 at 10:45:32 UTC from IEEE Xplore. Restrictions apply.

the scout phase. On Moving Peaks, CMABC is competitive or superior to baseline ABC and peer methods [8].

Biswas et al. (2012) adapted ABC to dynamic environments via k-means clustering and an archive memory: the swarm is hard-partitioned, perturbations use nearest/farthest neighbor moves, and a selection factor (S=0.6) enables multi-parameter updates per cycle; a dedicated change_detector particle provides change detection. After a change, the population is refreshed with normal noise, archive elites replace poor solutions, and converged clusters are reinitialized to maintain diversity [9].

While the first stage of the proposed method—detecting problem changes—aligns with approaches in the literature, the second stage diverges in the procedure followed after a change is detected. Contrary to some prior studies, the scout-bee phase is not eliminated; rather, we show that more frequent use of the scout phase is necessary to maintain an appropriate exploration–exploitation balance.

3. Methods

3.1. Basic ABC Algorithm Steps

ABC algorithm is a metaheuristic inspired by the foraging behavior of honeybees in discovering and exploiting food sources. In this framework, each solution in the population represents a food source. These solutions are updated through three main types of bees: employed, onlooker, and scout bees. The procedural steps of the ABC algorithm, derived from the natural behaviors of honeybees, are summarized as follows:

Step 1 – Initialization of the Population

In the search space, random solution vectors (food sources) are generated within the lower and upper bounds. Each solution is initialized by Eq. 1,

$$x_{ij} = x_{minj} + rand(0,1).\left(x_{maxj} - x_{minj}\right) \quad (1)$$

where, x_{minj} and x_{maxj} denote the bounds of dimension j and $rand(0,1)$ is a uniform deviate, yielding a uniform sample of x_{ij} over $[x_{minj}, x_{maxj}]$.

The value of the objective function is evaluated by Eq. 2.

$$f_i = f(x_i) \quad (2)$$

The fitness value is determined by Eq. 3.

$$fitness(f) = \begin{cases} \frac{1}{1+f}, f \geq 0 \\ 1 + |f|, f < 0 \end{cases} \quad (3)$$

For each solution, the trial counter is reset to zero Eq. 4.

$$trial_i = 0 \quad (4)$$

The solution with the lowest objective function value is recorded as the global optimum.

Step 2 – Employed Bee Phase

Each employed bee generates a new candidate solution in the neighborhood of its associated solution by Eq. 5.

$$v_{ij} = x_{ij} + \phi_{ij} \cdot \left(x_{ij} - x_{kj}\right) \quad (5)$$

ϕ_{ij} is a scalar coefficient independently sampled from a uniform distribution $U[-1,1]$ at each neighbor-generation step.

Here, x_k denotes a randomly selected, distinct solution. If the generated candidate violates the bounds, it is repaired; if it improves upon the incumbent, the solution is updated; otherwise, the corresponding trial counter is incremented by one.

Step 3 – Onlooker Bee Phase

Each onlooker bee selects a candidate solution based on the information provided by the employed bees by Eq. 6.

$$p_i = \frac{f_i}{\sum_{k=1}^{N} f_k} \quad (6)$$

Candidate solutions are selected via roulette-wheel selection; then, as in Step 2, a neighboring candidate is generated and greedy selection is performed.

Step 4 – Scout Bee Phase

If a solution cannot be improved for an extended period (i.e., its trial counter reaches or exceeds the predefined limit), it is abandoned and replaced with a newly generated random solution by Eq. 7.

$$\begin{gathered} if\ trial_i \geq limit \\ x_{ij} = x_j^{min} + rand(0,1)\left(x_j^{max} - x_j^{min}\right) \end{gathered} \quad (7)$$

Step 5 – Updating the Best Solution

At each cycle, all candidate solutions are compared against the current global best; if a lower objective value is obtained, the global best is updated.

3.2. Modified ABC Algorithm: Proposed Phases

This study augments the standard ABC algorithm with additional steps to make it responsive to environmental changes. The proposed modifications are tailored for dynamic optimization problems and comprise the following new steps:

Step A – Function-Change Detection Mechanism

At the beginning of each cycle, the global best obtained in the previous cycle is re-evaluated. If the newly computed objective value for this solution differs from its prior value, it is assumed that the objective function has changed.

$$f_{new}(x_{best}) \neq f_{old}(x_{best}) \quad (8)$$

A change in the objective function is detected by Eq. 8.

This control enables the algorithm to detect the environmental changes.

Step B – Population-Level Update After Change Detection

Authorized licensed use limited to: ULAKBIM UASL - Erciyes Universitesi. Downloaded on February 19,2026 at 10:45:32 UTC from IEEE Xplore. Restrictions apply.

Upon detection of a function change, the entire population—not only the current best—is re-evaluated with respect to the new objective function. For each solution:

- The objective function value is re-evaluated;
- The fitness value is re-evaluated;
- The trial counter is reset to zero.

Step C – Scout Bee Assignment Mechanism

In the presence of a function change, a prespecified fraction of the population is reinitialized via the scout-bee phase to adapt to the new environment, as follows:

- All candidate solutions are ranked according to their objective function values recomputed against the changed environment.
- Starting from the worst solutions, the selected fraction is designated as scout bees and reinitialized at random positions within the search space.

This procedure substantially enhances the exploration capacity of the basic ABC algorithm and accelerates convergence toward emerging optima.

When no function change is detected, the algorithm follows the classical ABC scheme: the solution that fails to improve within a predefined limit of trials is reinitialized via the scout-bee mechanism, which is applied to the individual with the largest trial counter.

Step D – Recording the Best Solution

In the event of a change in the objective function, the previous global best is discarded at the end of the cycle in which the change is detected, and the solution with the lowest objective value among the current evaluations is assigned as the new global best.

In cycles where no change is detected, the procedure follows the standard ABC: the global best is updated whenever a solution with a lower objective value is found.

In summary, with these modifications the algorithm can:

- Detect changes in the objective function,
- Re-evaluate and update the entire set of candidate solutions,
- Adaptively broaden its re-exploration via the scout-bee phase,
- Switch to its classical ABC behavior when the environment becomes stationary.

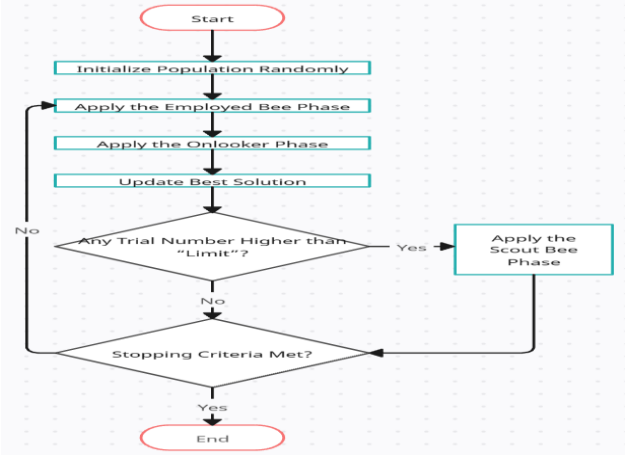


Fig. 1: Flowchart of Standart (Basic) ABC

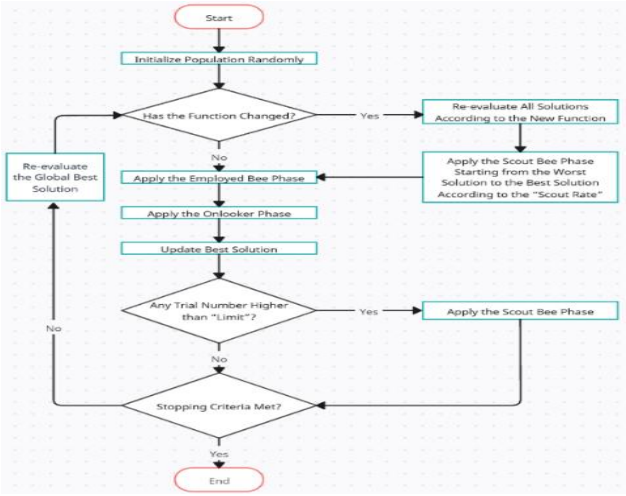


Fig. 2: Flowchart of Modified ABC

4. Experimental Study

4.1. Experimental Setup

Experimental settings were as follows: the colony size was 100 (50 employed bees and 50 onlooker bees), yielding 50 food sources (SN = 50); the problem dimension was D = 100; the maximum number of cycles was 30.000; we executed 30 independent runs; the limit parameter was set to 500, computed as SNxDx0.1; and the initial seed was chosen at random.

4.2. Dynamic Benchmarks and Change Schedule

Beyond static forms, we employ dynamic variants by applying a time-dependent shift (m_t) to the decision vector. At cycle t:

$$f(x,t) = f(x - m_i^t) \tag{9}$$

Here m_t is a piecewise-constant shift vector defined by Eq. 9. The environment is piecewise stationary: updates occur only at pre-specified change cycles; m_t remains fixed. No rotation, offset, or evaluation noise is applied.

4.2.1 Shift-Vector Generation and Bounding

We model dynamics via a shift vector $m^t \in [L,U]^D$ that specifies the time-varying optimum. The objective is evaluated by translating the static landscape by Eq. 10.

$$f(x,t) = f_{static}(x - m^t). \tag{10}$$

$m^0 \sim Uniform([L,U])$ is initialized. At predefined change cycles $t \in \theta$, each component is updated by Eq. 11.

$$m_i^{t+1} = m_i^t + \delta_i^t, \delta_i^t \sim N(\mu, \sigma^2), \tag{11}$$

Feasibility is ensured via rejection sampling (equivalently, δ_i^t is drawn from a truncated normal on $[L_i - m_i^t, U_i - m_i^t]$) so that $m_i^{t+1} \in [L_i, U_i]$. For $t \notin \theta$, keep $m^{t+1} = m^t$. The mean μ

Authorized licensed use limited to: ULAKBIM UASL - Erciyes Universitesi. Downloaded on February 19,2026 at 10:45:32 UTC from IEEE Xplore. Restrictions apply.

controls directional drift and the standard deviation σ controls change magnitude; bounding m_t ensures the moving optimum remains within the feasible region and yields controlled, comparable dynamic difficulty.

4.2.2 Function Definitions (Dynamic Forms)

Dynamic Rastirigin function is given by Eq. 12:

$$f(x,t) = 10D + \sum_{i=1}^{D}\left[(x_i - m_i^t)^2 - 10cos\left(2\pi(x_i - m_i^t)\right)\right] \quad (12)$$

Dynamic Rosenbrock function defined by Eq. 13:

$$f(x,t) = \sum_{i=1}^{D-1}\begin{bmatrix}\left(1-(x_i - m_i^t)\right)^2 + \\ 100\left((x_{i+1} - m_{i+1}^t) - \right)^2 \\ (x_i - m_i^t)^2\end{bmatrix} \quad (13)$$

Dynamic Griewank function given by Eq. 14:

$$f(x,t) = 1 + (1/4000)$$
$$\sum_{i=1}^{D}(x_i - m_i^t)^2 - \prod_{i=1}^{D} cos\left((x_i - m_i^t)/\sqrt{(i)}\right) \quad (14)$$

Dynamic Sphere function defined by Eq. 15:

$$f(x,t) = \sum_{i=1}^{D}\left(x_i^2 - m_i^t\right) \quad (15)$$

4.2.3 Change Cycles

Functions are changed at the following cycle indices:

Rastrigin, Griewank, Sphere:
(20000, 21000, 22200, 22250, 22300, 24000, 24400, 24450, 26300, 28150)

Rosenbrock:
(20900, 22200, 22850, 22950, 24500, 25650, 24800, 25500, 26750, 27000, 27300, 29100, 29500, 29900)

At each $t \in \theta_f$, m_t is set to the next shift vector in the sequence and recorded; between changes, m_t remains constant.

4.3 Pre-computed Shiftes and Feasibility

Shift vectors are read form a fixed file and reused across runs. For each benchmark, $m_t$vectors are validated against the canonical box constraints as given in Table 1:

Function Name	Lower Bound	Upper Bound
Sphere	-100	+100
Rastrigin	-5.12	+5.12
Griewank	-600	+600
Rosenbrock	-2.048	+2.048

Table 1: Lower and Upper Bound of Functions

4.4 Scout Rate (SR)

Upon detected function changes, the scout rate (SR) is treated as an experimental parameter. Under the same dynamic setup, all other hyperparameters are fixed while SR is varied over: SR ∈ {0%, 50%, 99%, 100%}

5. Results

The experiments on dynamic Rastrigin, Rosenbrock, Griewank, and Sphere indicate a consistent ordering across scout-rate (SR) settings (see Figs. 3–6). Notably, the Standard ABC does not detect function changes; it keeps exploiting around the previously best location and thus continues searching in the wrong region after each change, with recovery occurring only when the scout mechanism eventually relocates solutions. In particular, SR 99%—which retains the current best solution and reinitializes nearly all remaining solutions—delivers the fastest recovery after a change and maintains the lowest trajectories over time. This behavior aligns with the intuition that, in dynamic landscapes, preserving a single high-quality "anchor" enables directed exploitation while a broad restart injects the diversity required for rapid tracking of the new optimum. In contrast, SR 100% removes the anchor altogether and turns the restart into an undirected exploration, which explains the larger excursions and slower reconvergence observed in the figures. SR 50% improves over SR 0% yet typically falls short of SR 99%; the Rosenbrock landscape appears most sensitive to SR, whereas Griewank and Sphere separate more smoothly but with the same overall ordering.

Relative to prior ABC variants that rely on change detection, memory archives, or distribution-based steps, our results suggest a complementary design principle: when a change is detected, the entire population, is re-evaluated keep the best-so-far is kept, and reset almost all other solutions are reset. This policy offers a strong exploration–exploitation balance without introducing heavy parameterization. From an implementation standpoint, the additional overhead is modest, as re-evaluation and randomized restarts are linear in the population size.

This study is limited by the chosen dynamics (shift-only changes with pre-specified change cycles) and synthetic benchmarks at a single dimension (D=100). Future work should examine rotated/reshaped landscapes, stochastic observation noise, and application-driven dynamics, as well as real-time constraints and heterogeneous change magnitudes. Extending the analysis with statistical tests and runtime/memory profiling would further substantiate the observed trends.

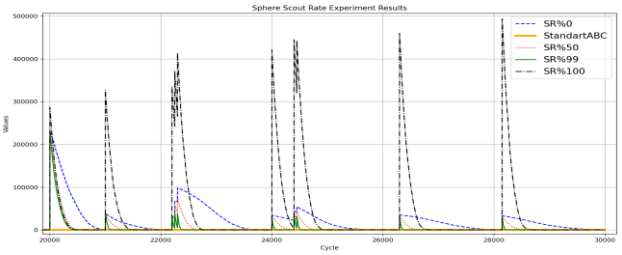


Fig. 3. Dynamic Sphere — effect of scout rates

Authorized licensed use limited to: ULAKBIM UASL - Erciyes Universitesi. Downloaded on February 19,2026 at 10:45:32 UTC from IEEE Xplore. Restrictions apply.

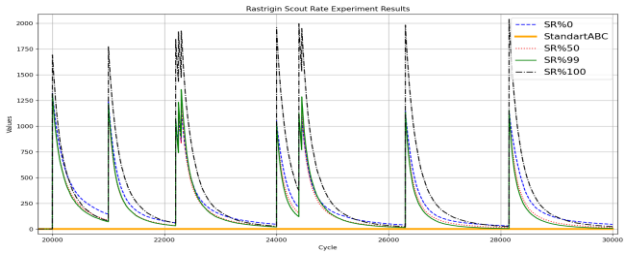


Fig. 4. Dynamic Rastrigin — effect of scout rates

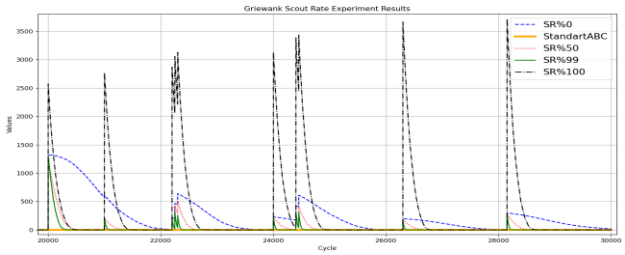


Fig. 5. Dynamic Griewank — effect of scout rates

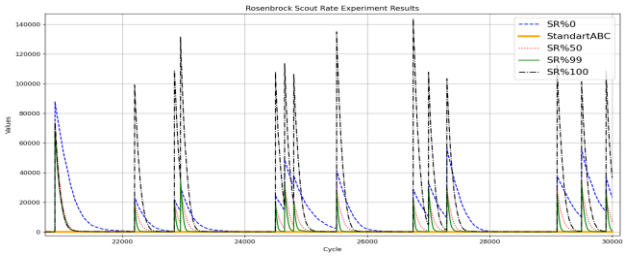


Fig. 6. Dynamic Rosenbrock — effect of scout rates

6. Conclusion

Across four dynamic benchmark functions (Rastrigin, Rosenbrock, Griewank, Sphere), varying the scout-assignment rate (SR ∈ {0%, 50%, 99%, 100%}) yields a consistent pattern: SR 99% produces the fastest post-change recovery and the lowest tracking error. Here, SR 99% means retaining the single best solution after each detected change and reinitializing almost all remaining solutions as scouts. This policy injects strong exploration while preserving one elite "anchor," thereby maintaining exploitation capacity. The advantage is most pronounced on multimodal landscapes (Rastrigin, Griewank), where steep drops and rapid reconvergence are observed. On predominantly unimodal landscapes (Rosenbrock, Sphere), performance gaps are narrower, yet SR 99% still attains the quickest recovery and the smallest residual error.

By contrast, SR 100% (turning every solution into a scout) effectively resets the population, erasing useful structure and yielding the largest spikes and the slowest reconvergence. Importantly, SR 0% is *not* the same as Standard ABC. In our SR framework, SR 0% still detects function changes and re-evaluates the population, but it launches no extra scouts beyond the baseline; adaptation is therefore slower than SR 50%/99% yet remains change-aware. Standard ABC, on the other hand, has no change detector and keeps exploiting around the prior best, continuing to search in the wrong region until the limit-triggered scout mechanism eventually relocates solutions. The intermediate SR 50% shows middling behavior—typically faster than SR 0% but not as efficient as SR 99%.

Overall, for dynamic environments we recommend SR 99%, avoiding SR 100%, and using SR 0% only as a change-aware lower bound—not as a substitute for Standard ABC. These findings highlight a general principle: after a change, broad exploration is critical, but preserving a single elite solution is equally decisive for sustained exploitation and stable tracking.

Future work should examine rotated/reshaped landscapes, stochastic observation noise, and application-driven dynamics, as well as real-time constraints and heterogeneous change magnitudes. Extending the analysis with rigorous statistical testing and runtime/memory profiling would further validate the observed trends. It will also focus on evaluating real-time applications and assessing adaptability across diverse dynamic scenarios.

6. References

[1] D. Karaboga, "An Idea Based on Honey Bee Swarm for Numerical Optimization," Tech. Rep. TR06, Erciyes Univ./ Eng. Faculty, Comput. Eng. Dept., Kayseri, Turkey, 2005.

[2] D. Karaboga and B. Basturk, "A Powerful and Efficient Algorithm for Numerical Function Optimization: Artificial Bee Colony (ABC) Algorithm," J. Glob. Optim., vol. 39, no. 3, pp. 459-471, 2007.

[3] D. Karaboga and B. Basturk, "On the Performance of Artificial Bee Colony (ABC) Algorithm," Appl. Soft Comput., vol 8, no. 1, pp. 687-697, Jan. 2008.

[4] T. Nishida, "Modification of ABC Algorithm for Adaptation to Time-Varying Functions," Electron. Comm. Jpn., vol. 96, no. 11, pp. 44-53, 2013.

[5] D.-L. Jia, S.-X. Qu, and L.-Y. Li, "A Multi-swarm Artificial Bee Colony Algorithm for Dynamic Optimization Problems," in *Proc. Int. Conf. on Information System and Artificial Intelligence (ISAI)*, pp. 439–443, 2016.

[6] H. Nakano, M. Kojima, and A. Miyauchi, "An Artificial Bee Colony Algorithm with a Memory Scheme for Dynamic Optimization Problems," in *Proc. IEEE*, 2015.

[7] D. Bose, S. Biswas, S. Kundu, and S. Das, "A Strategy Pool Adaptive Artificial Bee Colony Algorithm for Dynamic Environment through Multi-population Approach," in Proc. SEMCCO 2012, LNCS 7677, pp. 611–619, 2012.

[8] D. Jia, "A Culture-Based Artificial Bee Colony Algorithm for Optimization in Dynamic Environments," *J. Adv. Comput. Intell. Intell. Informatics*, vol. 26, no. 1, pp. 23–27, 2022.

[9] S. Biswas, D. Bose, and S. Kundu, "A Clustering Particle Based Artificial Bee Colony Algorithm for Dynamic Environment," in *Proc. SEMCCO 2012*, LNCS 7677, pp. 151–159, 2012.

Authorized licensed use limited to: ULAKBIM UASL - Erciyes Universitesi. Downloaded on February 19,2026 at 10:45:32 UTC from IEEE Xplore. Restrictions apply.