Rooster Colony Algorithm: A two-step Multi-Swarm Optimization Approach

Miguel A. Salido[1], **Adriana Giret**[2] **Christian Perez**[3], **Carlos March**[1]
[1]Instituto de Automatica e Informatica Industrial, Universitat Politecnica de Valencia, Spain
[2]Instituto Valenciano de Inteligencia Articial, Universitat Politecnica de Valencia, Spain
[3]Valencian Graduate School of Artificial Intelligence - valgrAI, Spain
msalido@dsic.upv.es, agiret@dsic.upv.es, cripeber@upv.es, cmarmoy@etsinf.upv.es

Abstract

Particle Swarm Optimization is a metaheuristic optimization algorithm inspired by the collective behavior of animal swarms where a set of candidate solutions, called particles, are randomly initialized in the search space, and their movements are iteratively updated based on their individual best solutions and the global best solution found by the swarm. This paper proposes a Multi-Swarm rooster colony algorithm (RCA) that considers a set of roosters, each owning a group of hens to compose a team. Each team (rooster and its hens) competes for the resource (food) with the other teams. From the combinatorial optimization point of view, each team analyzes part of the search space by an independent PSO algorithm with the same objective function. The RCA algorithm concurrently executes all PSO algorithms with different inertial weights for exploring different regions and the best solution (Gbest) of each team will compose the initial population for a new further centralized PSO algorithm that will exploit the previous solutions to search for the optimal one. Thus, the proposed RCA is composed of two steps, based on exploration and exploitation strategies to find an optimized solution in the search space. The results show that the proposed algorithm is competitive in solving well-known optimization functions. The objective is to apply this technique to solving real-life scheduling problems.

Introduction to PSO and Optimization Algorithms

Particle Swarm Optimization (PSO) (Kennedy and Eberhart 1995) is a population-based optimization technique that is inspired by the collective behavior of social swarms such as birds flocking, fish schooling, or ants colonies. It is a stochastic algorithm used to optimize a problem by iteratively trying to improve a candidate solution with respect to a given measure of quality. In PSO, a set of candidate solutions, called particles, are randomly initialized in the search space, and their movements are iteratively updated based on their individual best solutions and the global best solution found by the swarm. Each particle adjusts its position in the search space according to its own best solution so far and the best solution found by any particle in the swarm.

It is a relatively simple and efficient algorithm, but its performance can be sensitive to the selection of parameters such as the number of particles, the weight coefficients, and the termination criterion.

In Particle Swarm Optimization (PSO), exploration and exploitation are two crucial concepts that determine the algorithm's ability to find the global optimum in a search space (Clerc, 2010). Exploration refers to the process of exploring different regions of the search space to find new solutions. In PSO, exploration is achieved by the particles moving randomly around the search space to discover new regions. Exploration is important to ensure that the algorithm does not get trapped in a local optimum and can continue to search for the global optimum. Exploitation, on the other hand, refers to the process of exploiting the current knowledge of the search space to refine the search and find better solutions. In PSO, exploitation is achieved by the particles adjusting their positions towards the best position found so far. Balancing exploration and exploitation is critical in PSO to achieve optimal performance. Several techniques are used in the literature to balance exploration and exploitation (Zhang and Wang, 2021), by applying a multi-swarm particle swarm optimization (Xia, Gui and Zhan, 2018), adjusting the swarm's parameters (Liu and Liu, 2019), switching between different topologies (Chen et. al., 2021) or introducing memory and mutation strategies (Zhu et al., 2021). The optimal balance between exploration and exploitation depends on the specific problem being solved, and finding the right balance is a challenging task in optimization.

Recently, several swarm intelligence optimization algorithms have been developed by simulating the social relationship structure in a chicken swarm. For instance, (Wu et al., 2016) demonstrated the convergence of the chicken swarm optimization algorithm using the Markov chain and proposed an enhanced version of the algorithm that adjusts the updated formula of chicks to learn from both their mother hens and roosters within their group. (Torabi and Safi-Esfahani, 2018) combined the chicken swarm optimization algorithm with an improved raven roosting optimization algorithm to improve the balance between global and local search capabilities. (Qu et al., 2017) replaced the Gaussian distribution in the updated formula of roosters with adaptive distribution to achieve a balance between global and local search abilities.

The Rooster colony Algorithm

The Rooster colony Algorithm (RCA) aims to balance exploration and exploitation in a two-step PSO search. Algorithm 1 shows the pseudocode of the proposed Rooster colony Algorithm. The input of the algorithm is composed of the number of roosters, the hens associate with each rooster, and four different arrays of parameters that can be tuned to improve its performance:

- $iter = \{iter_1, ..., iter_{\#Roosters}, iter_g\}$ Maximum number of iterations for each PSO execution: It limits the number of times the algorithm will run before stopping.

- $W = \{w_1, ..., w_{\#Roosters}, w_g\}$ Inertia weight (W): it controls the impact of the rooster/hens' previous velocity on the new velocity. A high inertia weight can help the rooster/hen move towards the global best solution, but may also result in overshooting the optimal solution. A low inertia weight can help the rooster/hen converge more slowly, but may also result in getting stuck in local optima. Here, each PSO has a different value that covers different inertia weights ($w_1 < w_2 < ... < w_{\#Roosters}$)

- $c1 = \{c1_1, ..., c1_{\#Roosters}, c1_g\}$ Cognitive parameter (c1): it Controls the impact of the particle's personal best position on its new velocity. A high cognitive parameter can help the particle move towards its personal best solution, but may also result in getting stuck in local optima.

- $c2 = \{c2_1, ..., c2_{\#Roosters}, c2_g\}$ Social parameter (c2): it controls the impact of the swarm's global best position on the particle's new velocity. A high social parameter can help the particle move towards the global best solution, but may also result in overshooting the optimal solution.

During the first step of RCA Algorithm 1 (lines 3-7) different PSO search algorithms will explore the search space in an independent way. The best solution for each rooster will feed the initial population of the second step algorithm, which will execute a PSO algorithm with the parameter given in lines 8. This second step is focused on the exploitation of the best solutions obtained in the first step.

Algorithm 1 The Rooster colony Algorithm

1: *input:* #Roosters, {hens}, iter, W, c1, c2
2: *output:* $Gbest$, $fitness$
3: $Population \longleftarrow \emptyset$
4: **for each Rooster** i **do**:
5: $Gbest_i \longleftarrow$**PSO** (Rooster$_i \cup hens_i, iter_i, w_i, c1_i, c2_i$)
6: $Population \longleftarrow Population \cup Gbest_i$
7: **end for**
8: $Gbest, Fitness \longleftarrow$**PSO**(Population, iter$_g, w_g, c1_g, c2_g$)

Algorithm 2 shows the pseudo-code of the PSO algorithm used in the Rooster colony Algorithm 1.

Algorithm 2 PSO

1: *input:* Population, iter, w, c1, c2
2: *output:* $Gbest$, $fitness$
3: Evaluate the fitness of each particle
4: Set personal best position and fitness for each particle
5: Set global best position and fitness for the swarm
6: $i \longleftarrow 0$
7: **while** $s <= iter$ **do**
8: **for** (each particle i in swarm)
9: Update velocity $v_{i,d}$
10: Update position $x_{i,d}$
11: Evaluate fitness of particle i
12: **if**(fitness is better than personal best fitness)
13: Update best position $pbest_{i,d}$ and fitness
14: **end if**
15: **if**(fitness is better than global best fitness)
16: Update global best position $Gbest$ and $fitness$
17: **end if**
18: **end for**
19: **end while**

Evaluation

To evaluate the behavior of the proposed algorithm, commonly used test functions for optimization algorithms were used. For example, the Rastrigin function where A was set to 10, Rosenbrock function where a and b were set to 0 and 100, respectively, and Griewank function. All these functions have many local minima and a global minimum of 0.

Table 1 shows the comparison of the solution quality of PSO and the proposed RCA. To this end, the number of particles was set to 5, and parameters c1 and c2 were set to 2.5 and 0.5, respectively, and the inertial weight W was set to 0.6 for PSO (according to Perez, and Behdinan 2007). For RCA, 5 roosters were considered, and the inertial weight was set to 0.1, 0.3, 0.5, 0.7, and 0.9, respectively and C1=c2=2 to explore the search space. For the second step, the inertial weight was set to 0.1 ($w_g = 0.1$) and c1=2, c2=1 to refine the search and find a better solution. The number of iterations was the same for both algorithms. RCA divided the available number of iterations equally in both step 1 and step 2. One hundred executions were randomly generated for each problem and the average fitness value is shown in Table 1. It can be observed that RCA outperforms the behavior of the PSO algorithm in all analyzed optimization functions.

Function	Iter	Fitness in PSO	Fitness in RCA
Rastrigin	30	0.005858950	**0.00000000**
Rosenbrock	20	2.373629581	**0.00000004**
Griewank	20	0.419812079	**0.00000001**

Table 1: Fitness comparison of PSO v RCA.

Acknowledgements

The authors gratefully acknowledge the financial support of the European Social Fund (Investing In Your Future), the Spanish Ministry of Science (PID2021-125919NB-I00).

References

Kennedy, J., and Eberhart, R. 1995. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks* 1942–1948. Perth, Australia.

Clerc, M. 2010. Particle swarm optimization. John Wiley & Sons.

Zhang, X., and Wang, S. 2021. Quantum-behaved particle swarm optimization with adaptive mutation operator. *IEEE Transactions on Evolutionary Computation*, 25(1), 25–39.

Xia, X, Gui, L. and Zhan, Z.H. 2018. A multi-swarm particle swarm optimization algorithm based on dynamical topology and purposeful detecting. *Applied Soft Computing*, 67, 126–140.

Liu, Y., and Liu, X. 2019. Dynamic multi-objective particle swarm optimization with hybrid operator selection. Soft Computing, 23(2), 477–494.

Chen, J., Li, H., Zeng, J., Wu, S., and Gao, L. 2021. A novel self-adaptive particle swarm optimization algorithm based on dynamic switching topologies. Engineering Applications of Artificial Intelligence, 102, 104331.

Zhu, Y., Lai, X., Yang, Q., and Jiao, L. 2021. Adaptive particle swarm optimization with memory and mutation for high-dimensional optimization problems. IEEE Transactions on Evolutionary Computation, 25(4), 606–619.

D. Wu, S. Xu, and F. Kong. 2016. Convergence analysis and improvement of the chicken swarm optimization algorithm, IEEE Access, vol. 4, pp. 9400—9412.

S. Torabi and F. Safi-Esfahani. 2018. A hybrid algorithm based on chicken swarm and improved raven Roosting algorithm, Soft Computing, vol. 23, pp. 100129—100171.

C. W. Qu, S. A. Zhao, Y. M. Fu, and W. He, 2017. Chicken swarm optimization based on elite opposition-based learning, Mathematical Problems in Engineering, vol. 2017, 19 pages.

Perez, R. and Behdinan, K. 2007. Particle Swarm Optimization in Structural Design. 10.5772/5114.