Chapter 1
An Overview of Swarm Intelligence-Based Algorithms

Osheen Khare, Sumaiya Ahmed, and Yograj Singh

1 Introduction

In the past years, mathematicians, computer scientists, and research scholars have been increasingly engaged in researching the possibilities of emulating various natural systems to conceptualize and develop algorithms for the purpose of optimization. Owing to this trend, a prominent class of optimization techniques, known as nature-inspired algorithms, has emerged. The framework of such algorithms is designed to imitate the biological processes observed in nature, such as evolution, mutation, and societal behavior of insects, to arrive at the optimal solution. One of the emerging fields within nature-inspired algorithms has been the swarm intelligence (SI)-based algorithms. The term, swarm intelligence, introduced by G. Beni and J. Wang in 1989 [1] has been used to refer to the branch of optimization algorithms that models the collective behavior of animal colonies and the interactions (with the environment and other members of the swarm) of the members present in such a colony.

While the algorithms have gained widespread popularity today, they have been in work since the late 1980s. Ant colony optimization (ACO) was the first SI algorithm introduced by M. Dorigo and colleagues in the year 1991 to solve hard combinatorial optimization problems [10]. After that, J. Kennedy et al. [12, 20, 24] introduced particle swarm optimization (PSO) simulating the behavior of flocks of birds in 1995. Years later, in 2005, D. Karabago proposed the artificial bee colony algorithm (ABC) [17] in the family of SI algorithms. Over the years, research has increased the scope of swarm intelligence by observing and studying different groups of animals and algorithms, such as cuckoo search (CS), firefly algorithm (FA), dragonfly algorithm (DA) [30], and grey wolf optimizer (GWO), have emerged. The grey

O. Khare · S. Ahmed (✉) · Y. Singh
Department of Mathematics, Lady Shri Ram College for Women, University of Delhi, New Delhi, India

D. Singh et al. (eds.), *Design and Applications of Nature Inspired Optimization*, Women in Engineering and Science, https://doi.org/10.1007/978-3-031-17929-7_1

wolf optimizer is one of the algorithms which emulates the hierarchical system of an animal group. In each pack of grey wolves, there is an alpha that dominates the pack and a beta wolf who leads the pack in the absence of the alpha wolf. The delta and omega wolves follow the alpha and beta wolves. The algorithm draws inspiration from the hunting approach of a pack of grey wolves. These wolves hunt in an efficient manner by following a routine of steps: chasing, encircling, harassing, and attacking. This enables them to hunt bigger prey. GWO has found applications in medical and bioinformatics, machine learning, environmental applications, and networking applications. The developing SI-based algorithms such as the wolf pack algorithm are gaining wide popularity due to their global convergence and computational robustness.

The rapidly advancing swarm intelligence techniques can be applied to the optimization of telecommunication systems and business, military operations, engineering design problems, transportation systems, data mining, image segmentation, electric machines, and so on [31, 32]. Swarm robotics is a newly emerging field that draws inspiration from swarm intelligence. The field studies the design and interactions of simple robots with each other and their environment. The goal is to develop a collective behavior resulting from the coordination of multiple robots as a system. Such an approach can be applied to various scenarios, such as search and rescue, mapping, and demining.

Though there are ample instances of SI-based optimization techniques being used to solve real-world problems in the available literature, a little amount of work has been done with respect to conducting theoretical analysis of the algorithms to explain how they operate. Besides mathematical analysis, research also needs to be done in parameter tuning and parameter control to enhance the functioning of the algorithms.

The chapter aims to conduct an elaborate survey of SI-based algorithms and is divided into six sections. The first section gives the readers a brief introduction to SI-based optimization methods. The second section elaborates upon the key features and characteristics of SI-based algorithms, highlighting their advantages and limitations. The third section lays out the steps for implementing PSO and demonstrates the process by minimizing the Rosenbrock function. The fourth section details the stages involved in the execution of ABC and presents how the computational technique can be employed to minimize the Schwefel's function. The fifth subsection illustrates how the algorithms can be compared using the fixed iteration test. The last section summarizes the paper and presents the concluding remarks along with the further scope of the study.

2 Characteristics of SI-Based Algorithms

The swarm individuals show relatively simple intellectual abilities, but they are able to survive by using social interactions and certain behavioral patterns. These social interactions can either take place directly or indirectly. Direct interactions are

through audio or visual signals, such as the communication between birds of a flock. High-quality vision enables them to search for a food source and pass information related to it to the rest of the swarm. Indirect interactions are known as stigmergy, meaning communication through the environment, such as the pheromone trails of ants. Such a phenomenon occurs when a member of the colony reacts to the changes in the environment introduced by the other member.

Two of the notable characteristics of SI-based algorithms are division of labor and self-organization. To ensure proper division of labor, the entire colony is split into various groups, and each group is assigned a specialized task. Such a strategy results in a more structured and intensive exploration and exploitation of the search space. Self-organization indicates that the interactions among the members of the swarm take place on the basis of purely local information, fluctuations, and positive and negative feedback.

Over the years, SI algorithms have undergone continuous development, and hence, there has been a boom in the research exhibiting the rapid evolution of SI-based algorithms and successful implementation of SI algorithms to real-world optimization problems. Besides operations research [25], vast and diverse domains, such as machine learning [24], bioinformatics, medical informatics [8], business, and finance, have also started using computational modelling of swarms. SI algorithms are widely applied in problems of function optimization, optimal route and scheduling problems, engineering and structural optimization problems, data, and image analysis [13, 23]. Thus, they are considered to be one of the most promising optimization techniques owing to their following characteristics:

1. Scalability: SI algorithms are scalable in the sense that they can be applied to groups with a sufficient number of individuals to thousands of individuals. In other words, SI algorithms are independent of swarm size, as long as the size of the swarm isn't very small [3].
2. Adaptability: Owing to their inherent auto-configuration and self-organization abilities, SI algorithms are able to facilitate the swift adaptation of an individual to the variations in the environment on run-time.
3. Collective robustness: A single minor failure in a system can cause the failure of the entire system. In SI algorithms, such a risk is reduced as there is no single individual which is essential to the functioning of the colony. This makes SI algorithms highly robust with high fault tolerance [11].
4. Individual simplicity: SI algorithms comprises individuals who have rather simple and limited abilities of their own. However, change in the behavior of an individual level is sufficient to bring change in collective organized group behavior.

While SI algorithms can solve optimization problems with large data, compared to the other classes of nature-inspired optimization, they are still at an early stage of research. SI algorithms have certain demerits, such as:

1. Time-critical applications: SI algorithms are useful when it comes to solving non-time-critical problems with numerous repetitions of the same activity. Since SI

algorithms do have predefined and preprogrammed pathways to solutions, they are not suitable for time-critical applications, such as nuclear reactor temperature controllers.
2. Parameter tuning: This is one of the biggest drawbacks of SI algorithms. Most of the parameters involved in SI algorithms are dependent upon the problem; hence, they are either empirically selected using the trial and error method or adaptively adjusted on run-time [5].
3. Stagnation: SI algorithms exhibit a lack of central coordination; hence, they often suffer from stagnation or premature convergence to local optimum. However, the limitation can be overcome with better parameter tuning.

3 Particle Swarm Optimization Algorithm

PSO is a nature-inspired metaheuristic optimization algorithm that imitates the social behavior of a flock of birds. This population-based technique makes use of a set of flying particles that are birds with velocities. When a flock of birds moves in search of food, each bird adjusts its position according to its own historical performance as well as the flock's historical performance, which makes the flock move toward the most promising areas in the search space. Similarly, the particles in PSO dynamically adjust their position in order to reach the optimal solution efficiently [26, 33].

This experience sharing ability of a swarm makes PSO a rather efficient and successful algorithm for optimization. The algorithm begins with initializing n random particles with a certain position and velocity in the search space. At each iteration, in order to calculate the fitness of each of these particles, the objective function value is evaluated at their current position, and the personal best (*pbest*) and the global best (*gbest*) are identified. Then, the particle updates its velocity to imitate the *gbest* and *pbest* particles by moving closer to them. The formula for updating the velocity and position is, respectively, given by:

$$v_{n+1} = (w * v_n) + \left(c_1 * r_1 * \left(x_{pbest} - x_n\right)\right) + \left(c_2 * r_2 * \left(x_{gbest} - x_n\right)\right)$$

$$x_{n+1} = v_{n+1} + x_n$$

where

- v_{n+1} denotes the velocity of the successive particle.
- x_{n+1} denotes the position of the successive particle.
- x_n denotes the particle's current position.
- x_{pbest} denotes the historically personal best position of the particle.
- x_{gbest} denotes the position of the global best particle of the swarm.
- w denotes the given *inertia factor*, which controls the exploration capabilities of the algorithm. It strikes a balance between the global and local search.
- $r_1$and r_2 are random numbers uniformly generated within the range [0,1].

- c_1*and* c_2 are positive parameters called the *cognitive* and *social parameters* respectively. These parameters control the movement of the particle relative to its personal experience and the experience of the swarm [9]. The value of c_1 and c_2 can greatly affect the algorithm by biassing the particle's position towards *pbest* or *gbest*:
  - If $c_1 > c_2$, then the search behavior is biased toward the *pbest*.
  - If $c_1 < c_2$, then the search behavior is biased toward the *gbest*.
  - When high values of c_1 and c_2 are selected, then the particle's new positions are generated in distant regions of the search space, leading to a better global exploration, but it might lead to divergence of the particles.
  - When small values of c_1 and c_2 are selected, then the particle's new positions are generated close by as a result of limited movement, leading to a refined local search.

Before an iteration ends, the index of the swarm's *gbest* particle is updated in case the position of any particle in the swarm turns out to be better than the current position of the swarm's *gbest* particle. This iterative process is terminated when the stopping criterion is met, i.e., the maximum number of iterations is completed or a good enough fitness value is attained, or the algorithm has been giving the same result for a number of consecutive iterations. The fitness value of the *gbest* particle at the end of the process is taken as the optimized function value.

Pseudocode of PSO

1. Initialize the swarm of n random particles with arbitrary positions and velocity in the search space.
2. Define the bounds *(lb, ub)*, inertia factor (w), cognitive and social parameters (c_1, c_2), and maximum number of iterations to be executed.
3. Calculate the objective function value for each particle at their current position.
4. Update the particle's best position (x_{pbest}).
5. Identify the swarm's global best particle and update its position as x_{gbest}.
6. Update the velocity and position of the particle accordingly.
7. Repeat steps 3–6 until the particles converge to an optimal solution or the stopping criteria is met.

Minimizing Rosenbrock Function Using PSO

The implementation of PSO can be exhibited on a number of test functions. One such test function is Rosenbrock function, also known as the banana function. The function has a global minimum in a narrow parabolic valley at (1,1) with $f(x) = 0$.

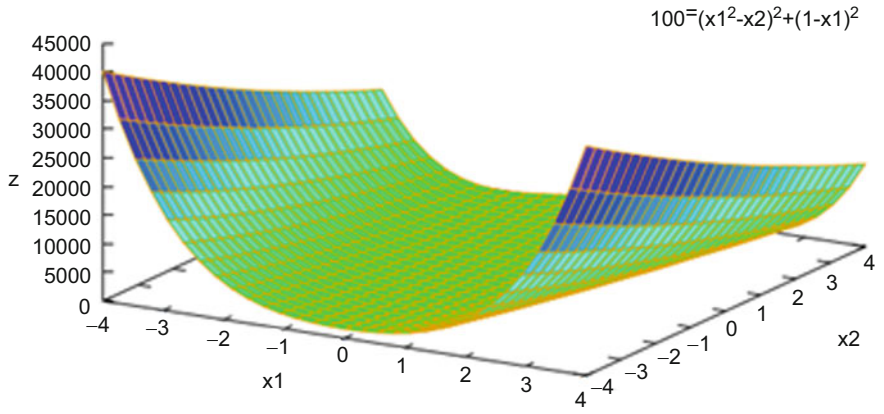


Fig. 1.1 Plotting Rosenbrock function using Maxima

Table 1.1 Optimal values obtained for variables by PSO

Variables	Optimal values
x_1	1
x_2	1
$f(x_1, x_2)$	0

Table 1.2 Values observed for different parameters during the execution of PSO

Parameters	Values
Best value	0
Worst value	4.8625
Mean	0.09602
Standard deviation	0.1635896857

While obtaining this global optimum is easy, the algorithms often get stuck to the local optimum; hence, it is used to assess the efficiency of optimization algorithms.

The general Rosenbrock function is given as:

$$f(x) = \sum_{i=1}^{d-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2 \right]$$

Here, we will implement PSO on a two-dimensional Rosenbrock function given as:

$$F = 100\left(x_1^2 - x_2\right)^2 + (1 - x_1)^2 \text{ where} - 2.048 \leq x_1,\ x_2 \leq 2.048.$$

The plot of the function can be observed in Fig. 1.1.

The algorithm was executed for 10 runs with 200 iterations. The number of iterations were fixed after observing the behavior and convergence rate of PSO. The results obtained while minimizing the Rosenbrock function using PSO can be summarized by Tables 1.1 and 1.2, and Fig. 1.2:

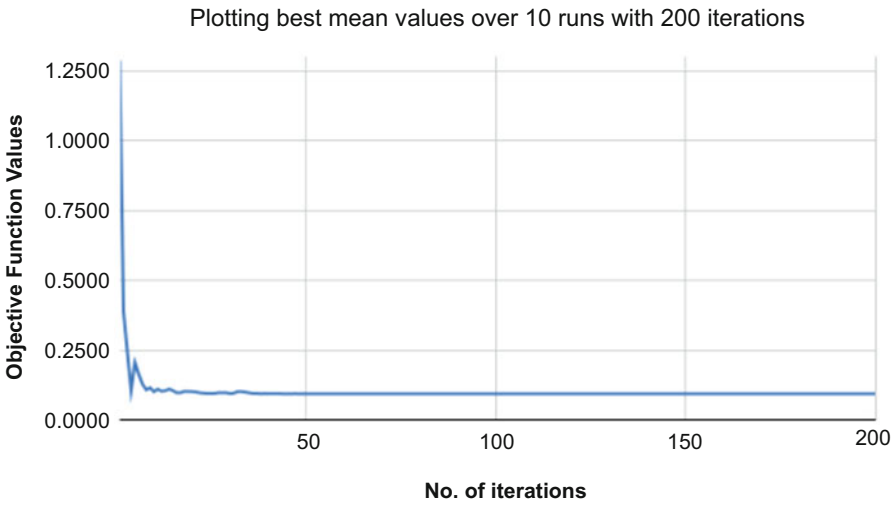


Fig. 1.2 Evolution of best mean results by PSO over 10 runs with 200 iterations

It can be observed that PSO has attained 0 as the best value. Thus, the algorithm has been successful in minimizing the function to the lowest value that can be achieved. The worst value and statistical measures, such as mean and standard deviation, are taken into account while assessing the performance of the algorithm as they indicate the stability and convergence of the solutions. If an algorithm attains a mean closer to the optimal value, a lower value for the worst objective function value and standard deviation, it indicates that the method was efficient and suitable as it was able to generate appropriately lower values (in case of a minimization problem) on sufficiently large numbers of iterations.

Figure 1.3 displays code snippets for executing PSO algorithm in MATLAB.

4 Artificial Bee Colony Algorithm

ABC is one of the most widely used SI-based optimization methods, which has been successfully applied to solve diverse complex numerical problems. The framework of the algorithm designed to emulate the foraging behavior of honeybees was first proposed by Karaboga in 2005 [15].

In ABC, the colony of bees is composed of three categories, i.e., employed, onlooker bees, and scout bees [18]. Employed bees find and exploit a particular food source. Here, the food source represents a feasible solution within the search space. Every employed bee produces a modification and assesses the nectar amount of a specific food source, i.e., fitness of the solution. After evaluating the quality of the food source, they pass this information to other bees in the hive. Onlooker bees, on receiving information related to the position, directions, and the quality of the food

Fig. 1.3 Implementing PSO using MATLAB

sources, select a food source based on a probability proportional to its fitness value. After a food source has been completely exhausted, it is abandoned, and the employed bee associated with that particular food source becomes a scout bee, which randomly chooses a new food source to exploit. Thus, it can be concluded that, while onlooker and employed bees perform the job of exploitation, scout bees ensure that the entire global region is intensively searched for an optimal solution.

To understand the method in-depth, we would now have a closer look at the three stages of the algorithm:

1. Employed Bees Phase: The process is initialized by generating random food source positions or feasible solutions within the search space. The parameters of the algorithms such as the swarm size, number of food sources, and limit are also defined. Each employed bee is assigned a particular food source to exploit. To examine the quality of the food source, the objective value function corresponding to the source/solution is calculated. The fitness values (fit_i) of the food sources are then derived using the following formula with the help of the objective function values (f_i) [16]:

$$\text{if } f_i \geq 0, \text{then } fit_i = \frac{1}{1+f_i}$$

$$\text{if } f_i < 0, \text{ then } fit_i = 1 + |f_i|$$

The employed bees alter the positions of the food sources to produce new solutions (X_{new}) by arbitrarily selecting a partner solution (X_p) and modifying a random variable (j^{th} variable) of the initial solution (X) with the help of the given formula:

$$X_{new}^j = X^j + rand(-1, 1) * \left(X^j - X_p^j\right)$$

The bees then implement greedy selection to discard the less suitable solutions. After retaining the richer food sources, the employed bees disseminate information concerning the quality of the food sources among the onlooker bees.

2. Onlooker Bees Phase: The onlooker bees, on the basis of the information received from the employed bees about the fitness of the solutions ($\forall n = 1, 2, \ldots, SN$), select a particular food source according to the probability (p_i) value assigned to it with the help of the given formula:

$$p_i = \frac{fit_i}{\sum_{n=1}^{SN} fit_n}$$

This ensures selection of better food sources. After choosing a food source to exploit, onlooker bees produce a modification in the food position, similar to the one carried by the employed bees. To determine which food sources to retain and which to abandon, the onlooker bees execute the greedy selection. At the end of the stage, the best solution obtained in the ongoing iteration is stored.

3. Scout Bees Phase: If the position of a particular food source hasn't been modified or if a solution hasn't been improved for a predetermined number of iterations/cycles (represented by the limit parameter), the food source is abandoned. The employed bee associated with the food source becomes a scout bee, which undertakes the random generation of new solutions to be exploited in the next iteration.

The same iterative process is continued till the loop is terminated and the stopping criterion is met, i.e., when the desired accuracy is attained, or a number of iterations get completed.

Minimizing Schwefel's Function Using Artificial Bee Colony Algorithm

*a*To examine the efficiency and utility of ABC, the algorithm has been implemented to minimize the rotated hyper-ellipsoid function, which is popularly known as the

Schwefel's function. The benchmark function is continuous, convex, and unimodal in nature and produces rotated hyper-ellipsoids when plotted. The function has been defined as:

$$f(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} X_j^2$$

The search area is usually restricted to $-65.536 \leq x_i \leq 65.536$ for $i = 1, \ldots, n$. The function attains its global minima, $f(x) = 0$, at $x_i = 0$ for $i = 1, \ldots, n$.

The algorithm has been executed to minimize the function in a two-dimensional space, where it takes the form of $f(x) = (x_1)^2 + ((x_1)^2 + (x_2)^2)$.

The plot of the function can be observed in Fig. 1.4.

Tables 1.3 and 1.4, and Fig. 1.5 summarize the observations and results attained while minimizing Schwefel's function using ABC over 10 runs with 200 iterations:

It can be observed that the best value obtained by ABC while minimizing the function is 0. Thus, the algorithm has been successful in minimizing the function to the lowest value that can be achieved.

Figure 1.6 provides a glimpse into how MATLAB can be programmed to implement ABC.

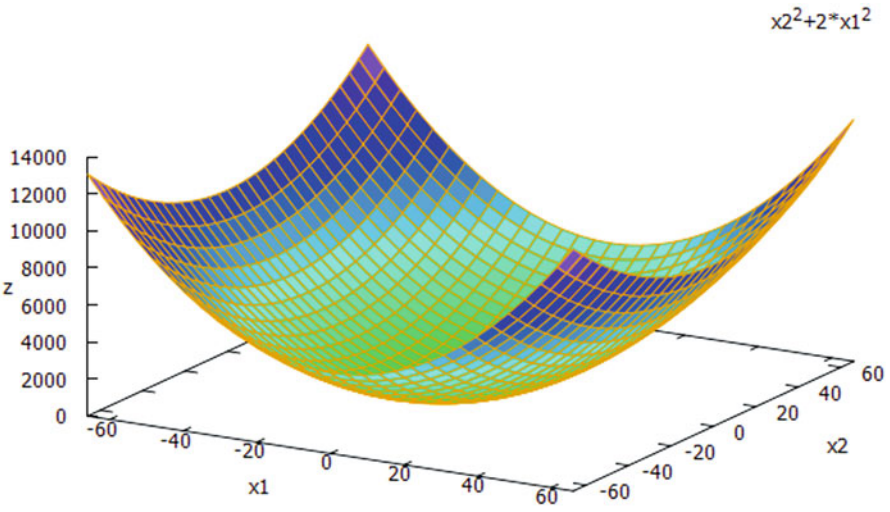


Fig. 1.4 Plotting Schwefel's function using Maxima

Table 1.3 Optimal values obtained for variables by ABC

Variables	Optimal values
x_1	0
x_2	0
$f(x_1, x_2)$	0

Table 1.4 Values observed for different parameters during the execution of ABC

Parameters	Values
Best value	0
Worst value	2.22184299
Mean	0.007044136895
Standard deviation	0.08366737868

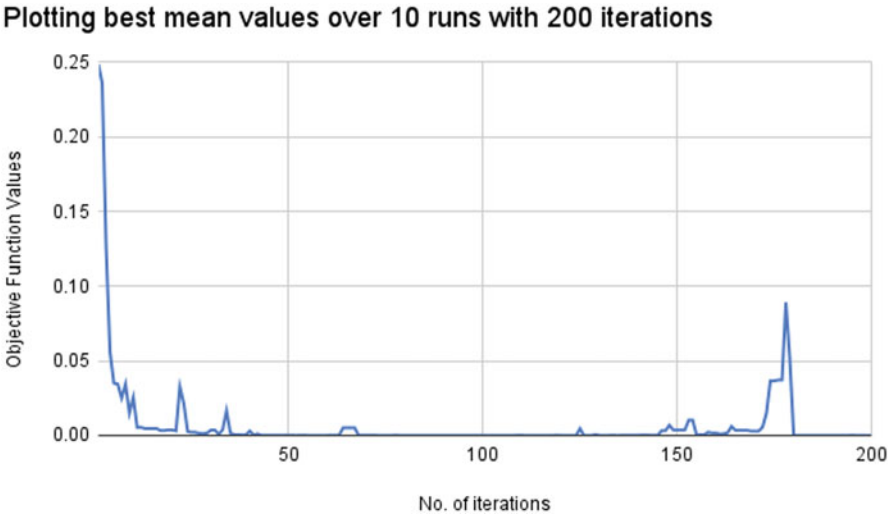


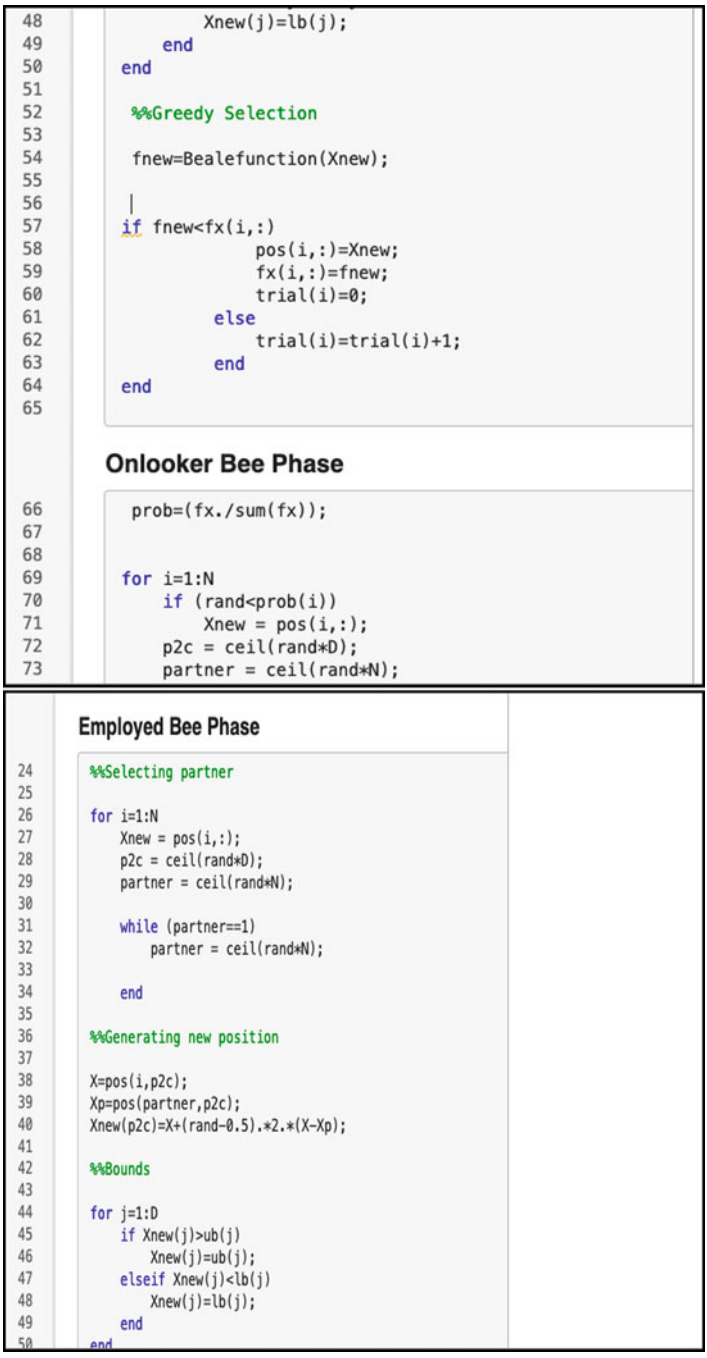
Fig. 1.5 Evolution of best mean results by ABC over 10 runs with 200 iterations

5 Comparative Analysis Using Fixed Iteration Test

The section lays out the framework to compare the performances of the algorithm using the fixed iteration test, wherein the results obtained over a specified no. of iterations and runs are analyzed to determine which algorithm performs better than its counterpart.

To establish a comparison between PSO and ABC algorithms, a fixed iteration test was conducted, wherein both SI-based computational methods were executed to minimize the axis parallel hyper-ellipsoid function. It is a continuous, convex, and unimodal function with a global minimum and no other local minima.

The general formulation of the function is given by:



```
48          Xnew(j)=lb(j);
49      end
50  end
51
52   %%Greedy Selection
53
54   fnew=Bealefunction(Xnew);
55
56   |
57  if fnew<fx(i,:)
58                  pos(i,:)=Xnew;
59                  fx(i,:)=fnew;
60                  trial(i)=0;
61           else
62                  trial(i)=trial(i)+1;
63           end
64  end
65
```

Onlooker Bee Phase

```
66    prob=(fx./sum(fx));
67
68
69  for i=1:N
70      if (rand<prob(i))
71          Xnew = pos(i,:);
72      p2c = ceil(rand*D);
73      partner = ceil(rand*N);
```

Employed Bee Phase

```
24  %%Selecting partner
25
26  for i=1:N
27      Xnew = pos(i,:);
28      p2c = ceil(rand*D);
29      partner = ceil(rand*N);
30
31      while (partner==1)
32          partner = ceil(rand*N);
33
34      end
35
36  %%Generating new position
37
38  X=pos(i,p2c);
39  Xp=pos(partner,p2c);
40  Xnew(p2c)=X+(rand-0.5).*2.*(X-Xp);
41
42  %%Bounds
43
44  for j=1:D
45      if Xnew(j)>ub(j)
46          Xnew(j)=ub(j);
47      elseif Xnew(j)<lb(j)
48          Xnew(j)=lb(j);
49      end
50  end
```

Fig. 1.6 Implementing ABC using MATLAB

$$f(x) = \sum_{i=1}^{d} ix_i^2$$

Here, we will be implementing the two algorithms on a two-dimensional sum squares function given by:

$$f(x) = x_1^2 + 2x_2^2$$

The function attains the minimum value of 0 at (0,0). Figure 1.7 shows the plot of the function.

The performances of the algorithm were evaluated and compared on the basis of the best and worst objective function value achieved and statistical parameters such as mean and standard deviation. With respect to optimization problems involving minimization of the given objective function value, the algorithm attaining a lower best objective function value and a higher worst objective function value is considered better. While mean reflects the average value achieved in an iteration, standard deviation measures the variability and dispersion of the dataset around the mean. These statistical parameters give an insight into the efficiency of the search processes employed in the algorithm. A mean closer to the optimal value with a lower standard deviation value indicates that the algorithm has higher convergence rates as it has been successful in producing lower objective functions close to the optimal value on a greater number of iterations.

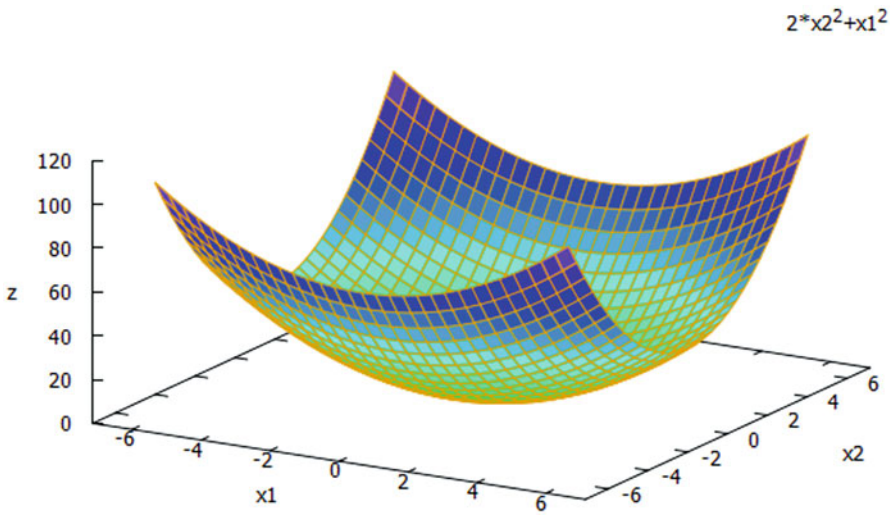


Fig. 1.7 Plotting axis parallel hyper-ellipsoid function using Maxima

Table 1.5 Comparison between performances of ABC and PSO

Parameters of comparison	PSO	ABC
Best value	0	0
Worst value	0.02119116	0.6271
Mean	0.000229697115	0.0015
Standard deviation	0.001051627142	0.01923517001

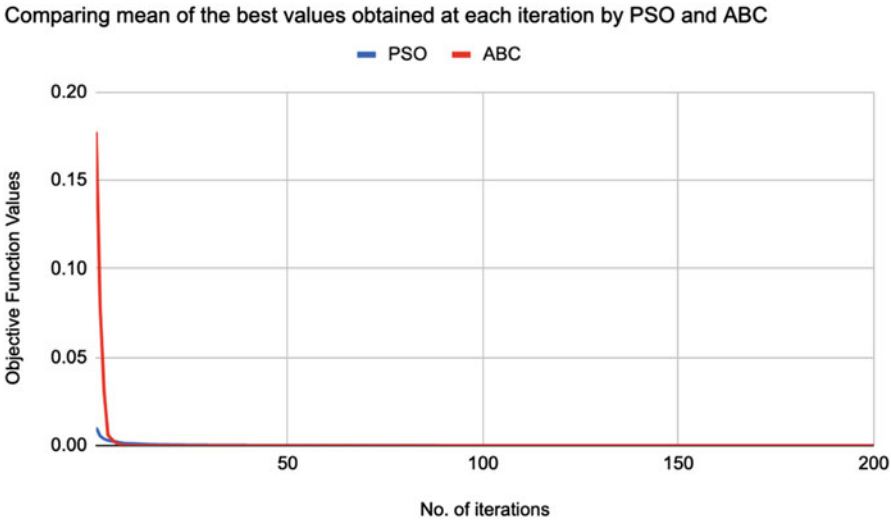


Fig. 1.8 Comparing best objective function values attained over 30 runs with 200 iterations

From Table 1.5 and Fig. 1.8, it can be concluded that PSO performs better than ABC while optimizing, and the given function as a higher worst value is achieved in case of ABC when compared to PSO. The mean calculated in the case of PSO is closer to the optimal value than ABC, which also obtains a greater value for standard deviation.

To further assess the suitability of the systems with respect to solving the minimization problem, the best variable values computed at each iteration were also compared.

By analyzing the results obtained for individual variables with respect to the predetermined parameters from Tables 1.6 and 1.7, Figs. 1.9 and 1.10, it can be deduced that PSO obtains better results in comparison to ABC. Thus, PSO is a more robust and efficient optimization tool than ABC for minimizing the sum squares function.

Table 1.6 Comparison between performances of ABC and PSO for x_1

Parameters of comparison	PSO	ABC
Best value	0	0
Worst value	0.1258	0.4663
Mean	0.000613	0.000642
Standard deviation	0.01077988038	0.01921224285

Table 1.7 Comparison between performances of ABC and PSO for x_2

Parameters of comparison	PSO	ABC
Best value	0	0
Worst value	−0.0986	0.5517
Mean	−0.0001	0.0003
Standard deviation	0.007526091968	0.02382160177

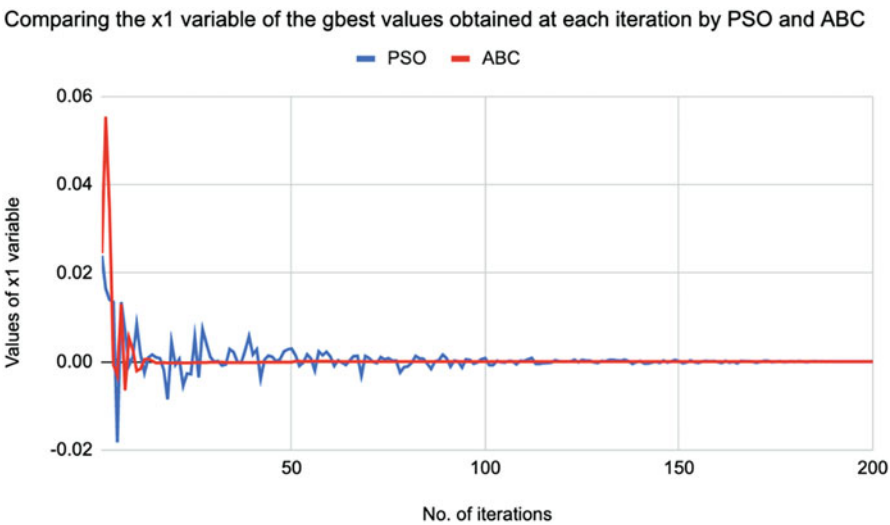


Fig. 1.9 Comparing best x_1 values attained over 30 runs with 200 iterations

6 Conclusion

The chapter offers a thorough overview of SI-based algorithms by highlighting its key features and points of merit and demerits and detailing and demonstrating the stages involved in implementing PSO and ABC to optimize unconstrained problems. The chapter also provides the readers with a blueprint for comparing algorithms by laying out the steps and requirements involved in conducting fixed interaction tests.

It is worth noting that both the algorithms attained an accurate optimal solution with their original structure, but their full potential can be unlocked only when necessary, and modifications are made to their structure. Hybridization of ABC with an evolutionary framework, stochastic methods, or deterministic methods can make

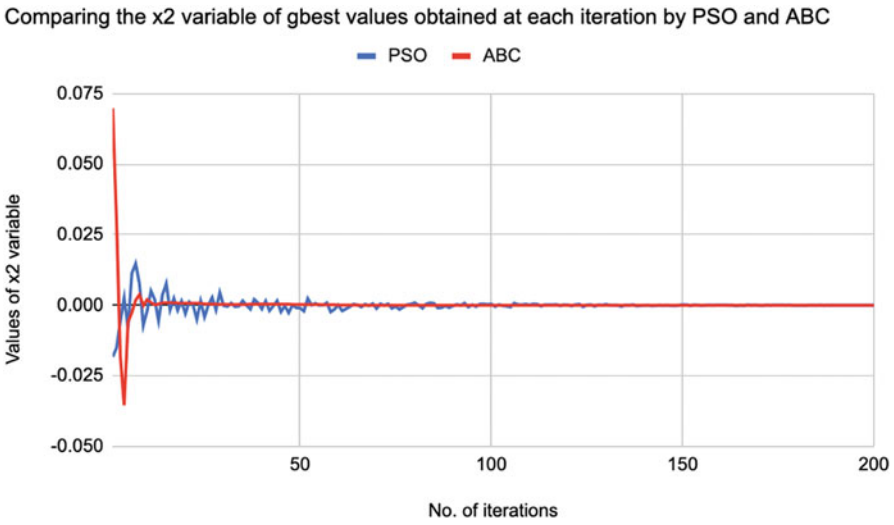


Fig. 1.10 Comparing best x_2 values attained over 30 runs with 200 iterations

the algorithm more efficient in the search process and allow parallel processing for time-saving. In order to improve the convergence rate of ABC, modifications in the production of new neighbor can be proposed. The search pool can be diversified with new strategies in the scout production phase. New selection strategies can also enhance the performance of ABC. On the other hand, hybridization of PSO with an evolutionary algorithm like genetic algorithm (GA) where the population of one algorithm is used as the initial population for the other one, instead of random population generation, produced finer results [28]. Adoption of new strategies of updating the velocity and particle position can lead to an increased efficiency of PSO. Along with this, further research can be conducted on fine parameter tuning for both PSO and ABC, since the results of the algorithms are heavily dependent on their parameters.

PSO and ABC have been evolving with time. The enhanced versions of the two algorithms have found them a place in a large number of applications. They have a huge scope in the field of neural networks [14], image and video analysis [4, 7], bioinformatics and medical applications [27], data mining [29], and much more [19].

While PSO and ABC are the most common examples of SI optimization algorithms, several other SI-based techniques of optimizations have been introduced in the recent years such as bacterial foraging [26], cat swarm optimization [6], artificial immune system [2], and glowworm swarm optimization [21, 22]. It should be interesting to realize that all these algorithms take inspiration from nature and help simulate an environment to solve real-world problems. At the same time, this very reason makes SI algorithms relatively weaker, because there exists an inadequacy of the theoretical analysis. The study of SI algorithms seems to be at an early stage, but with their growing popularity, one can believe that research and depth analysis will only improve the state of SI algorithms in the future.

References

1. Ahmed, H., Glasgow, J.: Swarm Intelligence: Concepts, Models and Applications. School of Computing, Queens University Technical Report (2012)
2. Bakhouya, M., Gaber, J.: An immune inspired-based optimization algorithm: application to the traveling salesman problem. Adv. Model. Optim. **9**(1), 105–116 (2007)
3. Bela, M., Gaber, J., El-Sayed, H., Almojel, A.: Swarm Intelligence. In: Handbook of Bio-inspired Algorithms and Applications, CRC Computer & Information Science, vol. 7. Chapman & Hall (2006)
4. Benala, T.R., Villa, S.H., Jampala, S.D., Konathala, B.: A novel approach to image edge enhancement using artificial bee colony optimization algorithm for Hybridized Smoothening Filters. In: World Congress on Nature & Biologically Inspired Computing, pp. 1071–1076. IEEE (2009
5. Bonabeau, E., Meyer, C.: Swarm intelligence: a whole new way to think about business. Harv. Bus. Rev. **79**(5), 105–115 (2001)
6. Buck, F.: Cooperative Problem Solving with a Distributed Agent System-Swarm Intelligence. (2007)
7. Chu, S.C., Tsai, P.W., Pan, J.S.: Cat swarm optimization. In: Pacific Rim International Conference on Artificial Intelligence, pp. 854–858. Springer (2006)
8. Das, S., Abraham, A., Konar, A.: Spatial information based image segmentation using a modified particle swarm optimization algorithm. In: 6th International Conference on Intelligent Systems Design and Applications, vol. 2, pp. 438–444. IEEE (2006)
9. Das, S., Abraham, A., Konar, A.: Swarm intelligence algorithms in bioinformatics. In: Computational Intelligence in Bioinformatics, pp. 113–147. Springer (2008)
10. Del Valle, Y., Venayagamoorthy, G.K., Mohagheghi, S., Hernandez, J.C., Harley, R.G.: Particle swarm optimization: basic concepts, variants and applications in power systems. IEEE Trans. Evol. Comput. **12**(2), 171–195. IEEE (2008)
11. Dorigo, M.: Optimization, Learning and Natural Algorithms (1992)
12. Dorigo, M.: Editorial. Swarm Intell. J. **1**(1) (2007)
13. Eberhart, R., Kennedy, J.: A new optimizer using particle swarm theory. In: MHS'95 Proceedings of the Sixth International Symposium on Micro Machine and Human Science, pp. 39–43. IEEE (1995)
14. Engelbrecht, A.P.: Computational Intelligence: An Introduction. Wiley (2007)
15. Irani, R., Nasimi, R.: Application of artificial bee colony-based neural network in bottom hole pressure prediction in underbalanced drilling. J. Petrol. Sci. Eng. **78**(1), 6–12. Elsevier (2011)
16. Karaboga, D.: An Idea Based on Honey Bee Swarm for Numerical Optimization. Technical report-TR06. Technical Report, Erciyes University (2005)
17. Karaboga, D., Akay, B.: Artificial bee colony algorithm for large-scale problems and engineering design optimization. J. Intell. Manuf. **23**, 1001–1014 (2010)
18. Karaboga, D., Basturk, B.: A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. J. Glob. Optim. **39**(3), 459–471. Springer (2007)
19. Karaboga, D., Basturk, B.: On the performance of artificial bee colony (ABC) algorithm. Appl. Soft Comput. **8**(1), 687–697 (2008)
20. Karaboga, D., Gorkemli, B., Ozturk, C., Karaboga, N.: A comprehensive survey: Artificial Bee Colony (ABC) algorithm and applications. Artif. Intell. Rev. **42**(1), 21–57. Springer (2014)
21. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of International Conference on Neural Networks, vol. 4, pp. 1942–1948. IEEE (1995)
22. Krishnanand, K., Ghose, D.: Glowworm swarm optimization for searching higher dimensional spaces. Innov. Swarm Intell. 61–75. Springer (2009)
23. Kulkarni, V.R., Desai, V.: ABC and PSO: a comparative analysis. In: IEEE International Conference on Computational Intelligence and Computing Research, pp. 1–7. IEEE (2016)
24. Lim, C.P., Dehuri, S.: Innovations in Swarm Intelligence, vol. 248. Springer Science & Business Media (2009)

25. Olivas, E.S., Guerrero, J.D.M., Martinez-Sober, M., Magdalena, B., Jose, R., Serrano, L.: Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques. IGI Global (2009)
26. Parsopoulos, K.E., Vrahatis, M.N.: Particle Swarm Optimization and Intelligence: Advances and Applications. IGI Global (2010)
27. Passino, K.M.: Biomimicry of bacterial foraging for distributed optimization and control. IEEE Control Syst. Magaz. **22**(3), 52–67. IEEE (2002)
28. Veeramachaneni, K., Osadciw, L.A., Varshney, P.K.: An adaptive multimodal biometric management algorithm. IEEE Trans. Syst. Man Cybern. C. Appl. Rev. **35**(3), 344–356. IEEE (2005)
29. Veeramachaneni, K., Peram, T., Mohan, C.K., Osadciw, L.A.: Optimization using particle swarms with near neighbor interactions. In: Genetic and Evolutionary Computation Conference, pp. 110–121. Springer (2003)
30. Wu, S., Lei, X., Tian, J.: Clustering PPI network based on functional flow model through artificial bee colony algorithm. In: 7th International Conference on Natural Computation, vol. 1, pp. 92–96. IEEE (2011)
31. Mirjalili, S.: Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems. Neural Comput. & Applic. **27**, 1053–1073 (2016)
32. Jevtić, A., Andina, D.: Swarm intelligence and its applications in swarm robotics. In: 6th WSEAS International Conference on Computational Intelligence, Man-Machine Systems and Cybernetics, pp. 41–46 (2007)
33. Shi, Y.: Feature article on particle swarm optimization. IEEE Neural Netw. Soc., 8–13 (2004)