Computational algorithms inspired by biological processes and evolution

*M. Janga Reddy and D. Nagesh Kumar**

In recent times computational algorithms inspired by biological processes and evolution are gaining much popularity for solving science and engineering problems. These algorithms are broadly classified into evolutionary computation and swarm intelligence algorithms, which are derived based on the analogy of natural evolution and biological activities. These include genetic algorithms, genetic programming, differential evolution, particle swarm optimization, ant colony optimization, artificial neural networks, etc. The algorithms being random-search techniques, use some heuristics to guide the search towards optimal solution and speed-up the convergence to obtain the global optimal solutions. The bio-inspired methods have several attractive features and advantages compared to conventional optimization solvers. They also facilitate the advantage of simulation and optimization environment simultaneously to solve hard-to-define (in simple expressions), real-world problems. These biologically inspired methods have provided novel ways of problem-solving for practical problems in traffic routing, networking, games, industry, robotics, economics, mechanical, chemical, electrical, civil, water resources and others fields. This article discusses the key features and development of bio-inspired computational algorithms, and their scope for application in science and engineering fields.

Keywords: Algorithms, biological processes, evolutionary computation, nonlinear optimization, swarm intelligence.

IN the last few decades, several novel computational methods have been proposed for the solution of complex real-world problems. The development of various biologically inspired computational algorithms has been recognized as one of the important advances made in the field of science and engineering. These algorithms can provide an enhanced basis for problem-solving and decision-making.

It is well recognized that the complexity of today's real-world problems exceeds the capability of conventional approaches[1]. The popular conventional methods that have been widely used include mathematical optimization algorithms (such as Newton's method and gradient descent method that use derivatives to locate a local minimum), direct search methods (such as the simplex method and the Nelder–Mead method that use a search pattern to locate optima), enumerative approaches such as dynamic programming (DP), etc. Each of these techniques in general requires making several assumptions about the problem in order to suit a particular method, and may not be flexible enough to adapt the algorithm to solve a particular problem as it is, and may obstruct the possibility of modelling the problem closer to reality[2]. Many science and engineering problems generally involve nonlinear relationships in their representation; so linear programming (LP) may not be a suitable approach to solve most of the complex practical problems. The enumerative-based DP technique poses 'curse-of-dimensionality' for a higher dimensional problem, due to exponential increase in computational time with increase in the number of state variables and the corresponding discrete states. The gradient-based nonlinear programming methods can solve problems with smooth nonlinear objectives and constraints. However, in large and highly nonlinear environment, these algorithms often fail to find feasible solutions, or converging to suboptimal solutions depending upon the degree of nonlinearity and initial guess. Also, the conventional nonlinear optimization solvers are not applicable for problems with non-differentiable and/or discontinuous functional relationships. The efficiency of algorithms varies depending on the complexity of the problem. Thus, for one reason or the other, conventional methods have several limitations and may not be suitable for a broad range of practical problems.

M. Janga Reddy is in the Department of Civil Engineering, Indian Institute of Technology Bombay, Powai, Mumbai 400 076, India and D. Nagesh Kumar is in the Department of Civil Engineering, and the Centre for Earth Sciences, Indian Institute of Science, Bangalore 560 012, India.
*For correspondence. (e-mail: nagesh@civil.iisc.ernet.in)

To surmount these problems, in recent times stochastic search and optimization algorithms inspired by nature and biological processes have been proposed and applied in various fields of science and engineering. In the following section details of these methods are briefly discussed.

Algorithms inspired by biological processes

The ideas from nature and biological activities have motivated the development of many sophisticated algorithms for problem-solving. These algorithms are broadly classified as evolutionary computation and swarm intelligence (SI) algorithms. Evolutionary computation is a term used to describe algorithms which were inspired by 'survival of the fittest' or 'natural selection' principles[3]; whereas 'swarm intelligence' is a term used to describe the algorithms and distributed problems-solvers which were inspired by the cooperative group intelligence of swarm or collective behaviour of insect colonies and other animal societies[4].

Evolutionary algorithms

Evolutionary algorithms (EAs) are computational methods inspired by the process and mechanisms of biological evolution. According to Darwin's natural selection theory of evolution, in nature the competition among individuals for scarce resources results in the fittest individuals dominating over the weaker ones (i.e. survival of the fittest). The process of evolution by means of natural selection helps to account for the variety of life and its suitability for the environment. The mechanisms of evolution describe how evolution actually takes place through the modification and propagation of genetic material (proteins). EAs share properties of adaptation through an iterative process that accumulates and amplifies beneficial variation through a trial and error process. Candidate solutions represent members of a virtual population striving to survive in an environment defined by a problem-specific objective function. In each case, the evolutionary process refines the adaptive fit of the population of candidate solutions in the environment, typically using surrogates for the mechanisms of evolution such as genetic recombination and mutation[3].

In EAs, a population of individuals, each representing a search point in the space of feasible solutions, is exposed to a collective learning process which proceeds from generation to generation. The population is randomly initialized and subjected to the process of selection, recombination and mutation through stages known as generations, such that the newly created generations evolve towards more favourable regions of the search space. The progress in the search is achieved by evaluating the fitness of all the individuals in the population, selecting the individuals with the highest fitness value and combining them to create new individuals with increased likelihood of improved fitness. After some generations, the solution converges and the best individual represents optimum (or near-optimum solution). The standard structure of EA[5] is shown in Figure 1.

EAs provide solutions to many real-world, complex, optimization problems that are difficult to tackle using the conventional methods, due to their nature that implies discontinuities of the search space, non-differentiable objective functions, imprecise arguments and function values[2]. EAs can be applied to many types of problems, viz. continuous, mixed-integer, combinatoric, etc. Furthermore, these algorithms can be easily combined with the existing techniques such as local search and other exact methods. In addition, it is often straightforward to incorporate domain knowledge in the evolutionary operators and in the seeding of the population. Moreover, EAs can handle problems with any combination of the challenges that may be encountered in real-world application such as local optima, multiple objectives, constraints, dynamic components, etc. The main paradigms of nature-inspired evolutionary computation[5] include genetic algorithm (GA), genetic programming (GP), evolutionary programming (EP), evolutionary strategies (ES), differential evolution (DE), etc.

Genetic algorithms

The most well-known paradigm of EAs is GA, having widespread popularity in science, engineering and industrial applications. The GA is inspired by population genetics (including heredity and gene frequencies) and evolution at the population level as well as Mendelian understanding of the structure (such as chromosomes, genes, alleles) and mechanisms (such as recombination and mutation)[6]. Individuals of a population contribute their genetic material (genotype) proportional to the suitability of their expressed genome (phenotype) to the environment, in the form of offspring. The next generation is created through a process of mating that involves the recombination of genomes of two individuals in the population with the introduction of random perturbation

```
Begin
    $t \leftarrow 0$
    initialize population $P(t)$
    evaluate $P(t)$
    While (not termination-condition) do
    Begin
        $t \leftarrow t + 1$
        select $P(t)$ from $P(t - 1)$
        alter $P(t)$
        evaluate $P(t)$
    End
End
```

Figure 1. Basic structure of an evolutionary algorithm.

(called mutation). This iterative process may result in an improved adaptive fit between the phenotypes of individuals in a population and the environment.

The basic version of GA was developed by Holland[3] based on binary encoding of the solution parameters (called binary-coded GA). Later many variants of GA were developed, including real-coded GAs, which work directly with the real values of the parameters[7]. The main steps involved in GA are:

(1) Initialize population using random generation.
(2) Evaluate the fitness of each individual in the population.
(3) Repeat the following steps (for evolution) until the termination criteria are satisfied:
    (a) Select the best-fit individuals for reproduction.
    (b) Perform genetic operations, crossover and mutation to generate new offspring.
    (c) Evaluate the individual fitness of new members.
    (d) Replace the least fit individuals with new individuals.
(4) Report the best solution of the fittest individual.

The algorithm can be terminated when the fitness value has reached some predefined threshold or maximum allowed time has elapsed or maximum number of generations is passed. The GA can be applied in many different ways to solve a wide range of problems. The success of GA to solve a specific problem depends on two major decisions: proper representation of the problem and defining a good measure of fitness. GAs have successful applications in civil, mechanical, electrical, manufacturing, economics, physics, chemistry, bioinformatics, water resources, etc.[1,5,7].

Genetic programming

GP is an inductive automatic programming technique for evolving computer programs to solve problems[8]. The objective of the GP algorithm is to use induction to devise a computer program. This is achieved by using evolutionary operators on candidate programs with a tree structure to improve the adaptive fit between the population of candidate programs and an objective function. The GP is well suited to symbolic regression, controller design and machine-learning tasks under the broader name of function approximation.

In GP, a population is progressively improved by selectively discarding the not-so-fit population and breeding new children from better populations. Like other EAs, the GP solution starts with a random population of individuals (equations or computer programs). Each possible solution set can be visualized as a 'parse tree' comprising the terminal set (input variables) and functions (generally operators such as +, –, *, /, logarithmic or trigonometric). The 'fitness' is a measure of how closely a trial solution solves the problem. The objective function, say, the minimization of error between estimated and observed value, is the fitness function. The solution set in a population associated with the 'best fit' individuals will be reproduced more often than the less-fit solution sets. The subsequent generation of new population is achieved through different genetic operations like reproduction, crossover and mutation. Selection is generally made by ranking the individuals according to their fitness. Individuals with better fitness are carried over to the next generation. In crossover operation, the branches in the 'parse tree' of two parent individuals are interchanged to generate two possible solution sets. The new solution sets (children) depict some characteristics of their parents, and genetic information is exchanged in the process; whereas mutation operation simply consists of random perturbation of node (function) in a tree with a probability known as mutation probability. Mutation of a parse tree will not change the tree structure, but changes the information content in the parse tree. The mutation helps to explore new domains of search and avoids the chances of local trapping. The overall success of GP in solving a given problem depends on proper selection and fine-tuning of the operators and parameters.

There are several variants of GP, like linear genetic programming (LGP), gene expression programming (GEP), multi-expression programming (MEP), Cartesian genetic programming, etc. Among these, the LGP and GEP are widely used in various fields of science and engineering[8,9].

Differential evolution

DE is a modern optimization technique in the family of EAs introduced by Storn and Price[10]. It was proposed as a variant of EAs to achieve the goals of robustness in optimization and faster convergence to a given problem. DE algorithm differs from other EAs in the mutation and recombination phases. Unlike GAs, where perturbation occurs in accordance with a random quantity, DE uses weighted differences between solution vectors to perturb the population.

A typical DE works as follows: after random initialization of the population (where the members are randomly generated to cover the entire search space uniformly), the objective functions are evaluated and the following steps are repeated until a termination condition is satisfied. At each generation, two operators, namely mutation and crossover are applied on each individual to produce a new population. In DE, mutation is the main operator, where each individual is updated using a weighted difference of a number of selected parent solutions; and crossover acts as background operator where crossover is performed on each of the decision variables with small probability. The offspring replaces the parent only if it improves the fitness value; otherwise the parent is carried over to the new population.

There are several variants of DE, depending on the number of weighted differences considered between solution vectors for perturbation and the type of crossover operator (binary or exponential). For example, in *DE*/*rand-to-best*/*1*/*bin* variant of DE, perturbation is made with the vector difference of the best vector of the previous generation (*best*) and current solution vector, plus single vector differences of two randomly chosen vectors (*rand*) among the population. The DE variant uses binomial (*bin*) variant of crossover operator, where the crossover is performed on each of the decision variables whenever a randomly picked number between 0 and 1 is within the crossover constant (CR) value. More details of DE and their applications can be found in Price *et al.*[11].

Swarm intelligence

The term swarm is used for the assembling of animals such as fish schools, bird flocks and insect colonies (such as ants, termites and honey bees) performing collective activities. In the 1980s, ethologists conducted several studies and modelled the behaviour of a swarm and came up with some interesting observations[12]. The individual agents of a swarm act without supervision, and each of these agents has a stochastic behaviour due to its perception in the neighbourhood. Simple local rules, without any relation to the global pattern, and interactions between systematic or self-organized agents led to the emergence of collective intelligence called 'swarm intelligence' (SI). Swarms use their environment and resources effectively by collective intelligence. Self-organization is the key characteristic of a swarm system, which results in global-level response by means of local-level interactions[4].

The SI algorithms are based on intelligent human cognition that derives from the interaction of individuals in a social environment. In the early 1990s, researchers noticed that the main idea of socio-cognition can be effectively applied to develop stable and efficient algorithms for optimization tasks[2]. In a broad sense, SI is an artificial intelligence tool that focuses on studying the collective behaviour of a decentralized system made up of a population of simple agents interacting locally with each other and with the environment. The SI methods are also called behaviourally inspired algorithms.

The main principles to be satisfied by a swarm algorithm to have an intelligent behaviour are[13]:

- The swarm should be able to do simple space and time computations (the proximity principle).
- The swarm should be able to respond to quality factors in the environment, such as the quality of food or safety of location (the quality principle).
- The swarm should not allocate all of its resources along excessively narrow channels and it should distribute resources into many nodes (the principle of diverse response).
- The swarm should not change its mode of behaviour upon every fluctuation of the environment (the principle of stability).
- The swarm must be able to change behaviour mode when it matters (the principle of adaptability).

The main paradigms of biologically inspired SI algorithms include ant colony optimization (ACO), particle swarm optimization (PSO), artificial bee colony (ABC), honey-bee mating optimization (HBMO) algorithms, etc. The SI algorithms have many features in common with the EAs. Similar to EAs, SI models are population-based methods. The system is initialized with a population of individuals (i.e. potential solutions). These individuals are then manipulated over many generations by way of mimicking the social behaviour of insects or animals, in an effort to find the optima. Unlike EA, SI models do not use evolutionary operators such as crossover and mutation. A potential solution simply flies through the search space by modifying itself according to its relationship with other individuals in the population and the environment. In the 1990s, mainly two approaches: (i) based on ant colony described by Dorigo[14], and (ii) based on fish schooling and bird flocking introduced by Eberhart and Kennedy[15] had vastly attracted the interest of researchers. Both approaches have been studied by many researchers and their new variants have been introduced and applied for solving several problems in different areas. Brief descriptions of ACO, PSO and ABC algorithms with their basic working principles are presented next.

Ant colony optimization

The ACO is inspired by the foraging search behaviour of real ants and their ability in finding the shortest paths. It is a population-based general search technique for the solution of difficult combinatorial optimization problems. The first ant system algorithm was proposed based on the foraging behaviour exhibited by real ant colonies in their search for food[14]. The algorithm is stimulated by the pheromone trail-laying and training behaviour of real ant colonies. The process by which the ants are able to find the shortest paths[2] is illustrated in Figure 2.

Let H be home, F be the food source and A–B be an obstacle in the route. At various times, the movement of ants can be described as follows: (a) at time $t = 0$, the ants choose left and right-side paths uniformly in their search for food source; (b) at time $t = 1$, the ants which have chosen the path F–B–H reach the food source earlier and are retuning back to their home, whereas ants which have chosen path H–A–F are still halfway in their journey to

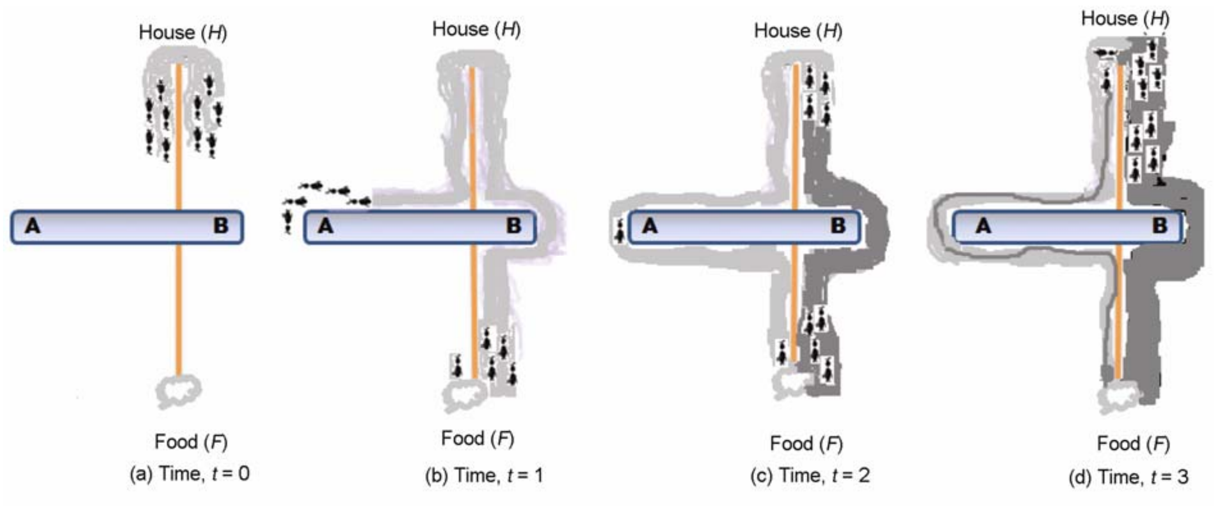

Figure 2. Illustration of ant colony principle – how real ants find the shortest path in their search for food.

the food source; (c) at time $t = 2$, since the ants move at approximately constant speed, those which chose the shorter, right-side path (*H–B–F*) reach home more rapidly, depositing more pheromone in the *H–B–F* route; (d) at time $t = 3$, pheromone accumulates at a higher rate on the shorter path (*H–B–F*), which is therefore automatically preferred by the ants and consequently all ants will follow the shortest path. The darkness of the shading is approximately proportional to the amount of pheromone deposited by ants.

In general, ACO has many features, which are similar to GA[16,17]:

- Both are population-based stochastic search techniques.
- GA works on the principle of natural evolution or survival of the fittest, whereas ACO works on pheromone trail-laying behaviour of ant colonies.
- GA uses crossover and mutation as prime operators in its evolution for the next generation, whereas ACO uses pheromone trail and heuristic information.
- In ACO, trial solutions are constructed incrementally based on the information contained in the environment and the solutions are improved by modifying the environment through a form of indirect communication called stigmergy; on the other hand, in GA the trial solutions are in the form of strings of genetic materials and new solutions are obtained through the modification of previous solutions.
- In GA, the memory of the system is embedded in the trial solutions, whereas in ACO algorithms the system memory is contained in the environment itself.

ACO algorithm: The ant system is the first ACO algorithm[14] that was proposed in 1991. The algorithm searches for global optimum using a population of trial solutions. First the system is randomly initialized with a population of individuals (each representing a particular decision point). These individuals are then manipulated over many iterations using some guiding principles in their search, in an effort to find the optima. The guiding principle used within the ACO is a probability function based on the relative weighting of pheromone intensity and heuristic information (indicating the desirability of the option) at a decision point. At the end of each iteration, each of the ants adds pheromone to its path (set of selected options). The amount of pheromone added is proportional to the quality of the solution (for example, in the case of minimization problems, lower-cost solutions are better; hence they receive more pheromone). The pseudo-code of a simple ACO algorithm[17] is given in Figure 3.

In early stages, these were mainly used for discrete combinatorial optimization tasks, but later modified and used for continuous optimization also. Thus, ACO is exhibiting interesting results for numerical problems as well as for real-life applications in science and engineering[16]. An important characteristic one should be aware of about ACO is that it is a problem-dependent application[17]. In

```
Begin
   Initialize population
   Evaluate fitness of population
   While (stopping criterion not satisfied) do
      Position each ant in a starting node
      Repeat
         For each ant do
            Choose next node by applying the state transition rule
            Apply step by step pheromone update
         End for
      Until every ant has built a solution
      Evaluate fitness of population
      Update best solution
      Apply offline pheromone update
   End While
End
```

Figure 3. Pseudo-code of the ant colony optimization algorithm.

order to adopt ACO and apply it to a particular problem, the following steps are required:

- Problem representation as a graph or a similar structure easily covered by ants.
- Assigning a heuristic preference to the generated solutions at each time step.
- Defining a fitness function to be optimized.
- Selection of appropriate model and parameters for the ACO algorithm.

Particle swarm optimization

PSO is a SI method inspired by social behaviour of bird flocking or fish schooling. The word 'particle' denotes individuals in a swarm, for example, birds in a flock. Each individual (particle) in a swarm behaves in a distributed way using its own intelligence and the collective (group) intelligence of the swarm. As such, if one particle discovers a good path to food, the rest of the swarm will also be able to follow the good path instantly, even if their location is far away in the swarm.

The PSO was originally proposed in 1995 by Eberhart and Kennedy[15] as a population-based heuristic search technique for solving continuous optimization problems. The PSO shares many similarities with the GA. PSOs are initialized with a population of random solutions and searches for optima by updating generations. However, in contrast to methods like GA, in PSO, no operators inspired by natural evolution are applied to extract a new generation of candidate solutions; instead PSO relies on the exchange of information between individuals (particles) of the population (swarm). Thus, each particle adjusts its trajectory towards its own previous best position and towards the best previous position attained by any other members in its neighbourhood (usually the entire swarm)[18].

PSO algorithm: In the PSO algorithm, each particle is associated with a position in the search space and velocity for its trajectory. For defining this, let us assume that the search space is D-dimensional; then the ith individual (particle) of the population (swarm) can be represented by a D-dimensional vector, $X_i = (x_{i1}, x_{i2}, \ldots, x_{iD})^T$. The velocity (position change) of this particle can be represented by another D-dimensional vector, $V_i = (v_{i1}, v_{i2}, \ldots, v_{iD})^T$. The best previously visited position of the ith particle is denoted as $P_i = (p_{i1}, p_{i2}, \ldots, p_{iD})^T$. Defining g as the index of the best particle in the swarm (i.e. the gth particle is the best), and superscripts denoting the iteration number, the swarm is manipulated according to the following two equations[19]:

Velocity updating:

$$v_{id}^{n+1} = \chi[w\, v_{id}^{n} + c_1 \text{rand}_1()(p_{id}^{n} - x_{id}^{n}) + c_2 \text{rand}_2()(p_{gd}^{n} - x_{id}^{n})]. \qquad (1)$$

Position updating:

$$x_{id}^{n+1} = x_{id}^{n} + v_{id}^{n+1}, \qquad (2)$$

where $d = 1, 2, \ldots, D$ is the index for decision variables; $i = 1, 2, 3, \ldots, N$, the index for the swarm population; N the size of the swarm; χ a constriction coefficient; w the inertial weight; the symbol g represents the index of the best particle among all particles in the population; c_1 and c_2 are called acceleration coefficients (namely cognitive and social parameters); '$\text{rand}_1()$' and '$\text{rand}_2()$' are uniform random numbers between zero and 1, and n is the iteration number. In eq. (1), the first term is the momentum part, the second term is the cognitive component (personal knowledge), and the third term is the social component (group knowledge). The general effect of the equation is that each particle oscillates in the search space, between its previous best position and the best position of its best neighbour, attempting to find the new best point in its trajectory. A time-step of unity is assumed in the velocity term of eq. (2).

In the PSO algorithm, first each particle is initialized with a random swarm of particles and random velocity vectors. Then the fitness of each particle is evaluated by the fitness function. Two 'best' values are defined, the global and the personal best. The global best (GBest) is the highest fitness value in the entire population (best solution so far), and the personal best (PBest) is the highest fitness value achieved by a specific particle over the iterations. Each particle is attracted towards the location of the 'best fitness achieved so far' across the whole population. In order to achieve this, a particle stores the previously reached 'best' positions in a cognitive memory. The relative 'pull' of the global and the personal best is determined by the acceleration constants c_1 and c_2. After this update, each particle is then revaluated. If any fitness is greater than the global best, then the new position becomes the new global best. If the particle's fitness value is greater than the personal best, then the current value becomes the new personal best. This procedure is repeated till the termination criteria are satisfied. The pseudo-code of the PSO algorithm[19] is presented in Figure 4.

```
Begin
    Initialize swarm position $X(0)$ and velocities $V(0)$
    Set iteration counter, $n = 0$
    Repeat
        Compute fitness function for each individual of swarm
        Compute PBest($n$) and GBest
        Begin (Perform PSO operations)
            Compute $V(n + 1)$ using velocity update rule
            Compute $X(n + 1)$ using position update rule
        End
        Set $n = n + 1$
    Until termination criteria satisfied
End
```

Figure 4. Pseudo-code of the particle swarm optimization (PSO) algorithm.

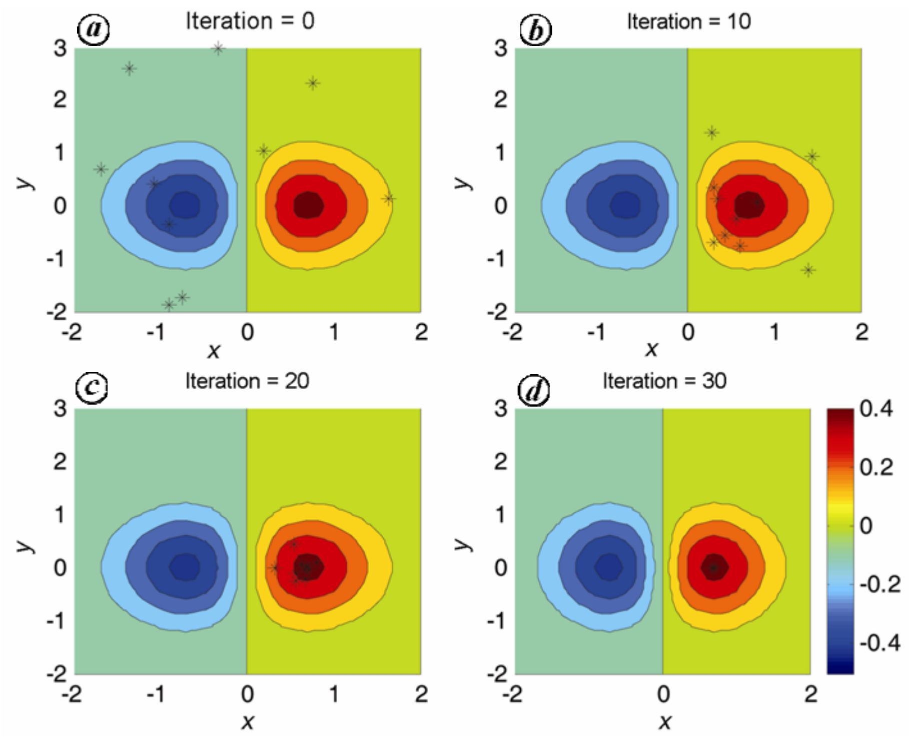

Figure 5. Illustration of the working of the PSO algorithm, showing over the iterations how the particles explore the global optima. The colour scheme gives the quality of fitness value (objective value). Blue colour corresponds to minimum *f*, whereas dark red to maximum *f*.

For illustration of the PSO working principle, let us consider a simple function $f(x, y) = x\mathrm{e}^{(-x^2-y^2)}$ for maximization. The working of the PSO at various iterations is depicted in Figure 5. Let (a) at iteration = 0, ten particles are randomly initialized uniformly in their search space; (b) at iteration = 10, the particles are guided by the social and individual experiences gained over the few iterations and are exploring the better functional value; (c) at iteration = 20, the particles are moving towards better fitness value and nearing the optimal value; (d) at iteration = 30, almost all the particles have reached the best solution in the search space. The darkness of colour from blue to red shows the variation of objective function value from minimum to maximum.

The solution of real-world problems is often difficult and time-consuming. Thus the application of bio-inspired algorithms in combination with the conventional optimization methods has also been receiving attention, with wider applications to practical problems[1].

Bee-inspired optimization

These optimization algorithms are inspired by honey bees. There are two main classes of honey-bee optimization algorithms: algorithms that utilize genetic and behavioural mechanisms underlying the mating behaviour of the bee, and algorithms that take their inspiration from the foraging behaviour of the bee. The first class of optimization algorithms makes use of the fact that a honeybee colony comprises a large number of individuals that are genetically heterogeneous due to the queen mating with multiple males. Many of the mating-inspired algorithms extend the principles of optimization algorithms from the field of evolutionary computation by introducing bee-inspired operators for mutation or crossover.

Among the bee swarm algorithms for optimization, the HBMO algorithm that was inspired from the natural mating process of honey bees[20], and the ABC algorithm that was inspired from simulating foraging behaviour of bees[21] are receiving wider applications in different areas of science and engineering for solving various optimization problems such as optimization of continuous functions, data-mining, vehicle routing, image analysis, protein structure prediction, etc.

ABC algorithm: This is a population-based algorithm inspired by the foraging behaviour of honey-bees[21]. In this metaphor, bees are the possible solutions to the problem, and they fly within the environment (the search space) to find the best food source location (best solution).

Honey-bee colonies have a decentralized system to collect food and can adjust the searching pattern precisely in order to enhance the collection of nectar. Honey bees collect nectar from flower patches as a food source for the hive from vast areas around their hive (more than 10 km), and usually the number of bees going out is proportional to the amount of food available at each patch. Bees communicate with each other at the hive via a waggle dance that informs other bees in the hive as to the direction, distance and quality rating of food sources. The exchange of information among bees is most important in the formation of the collective knowledge.

The basic idea concerning the algorithms based on the bee foraging behaviour is that foraging bees have a potential solution to an optimization problem in their memory (i.e. a configuration for the problem decision variables). This potential solution corresponds to the location of a food source and has an aggregated quality measure (i.e. value of the objective function). The food source quality information is exchanged through the waggle dance that probabilistically biases other bees to exploit food sources with higher quality.

The ABC algorithm works with a swarm of *n* solutions and *x* (food sources) of dimension *d* that are modified by the artificial bees. The bees aim at discovering places of food sources *v* (locations in the search space) with high amount of nectar (good fitness). In the ABC algorithm there are three types of bees: the scout bees that fly randomly in the search space without guidance; the employed bees that exploit the neighbourhood of their food sources selecting a random solution to be perturbed and the onlooker bees that are placed on the food sources using a probability based selection process. As the nectar amount of a food source increases, the probability value P_i with which the food source is preferred by onlookers also increases. If the nectar amount of a new source is higher than that of the previous one in their memory, they update the new position and forget the previous one. If a solution is not improved by a predetermined number of trials controlled by the parameter limit, then the food source is abandoned by the corresponding employed bee, and it becomes a scout bee. Each cycle of the search consists of moving the employed and onlooker bees onto the food sources and calculating their nectar amounts, and determining the scout bees and directing them onto possible food sources. The ABC algorithm seeks to balance the exploration and exploitation by combining local search methods (accomplished by employed and onlooker bees), with global search methods (dealt by scout bees)[21]. The pseudo-code of the ABC algorithm is shown in Figure 6.

A brief comparison of different evolutionary computation algorithms and SI algorithms is given in Table 1. Detailed reviews of different EAs and their applications can be found in the literature[22–25].

Multi-objective optimization

Recently, bio-inspired algorithms are becoming increasingly popular for solving multi-objective optimization problems, and ensued in the development of various multi-objective evolutionary algorithms (MOEAs) and multi-objective swarm algorithms (MOSAs). This is due to their efficiency and easiness to handle nonlinear and nonconvex relationships of real-world problems[1]. Also, these algorithms have some advantages over the conventional approaches, such as, use of population of solutions in each iteration helps to offer a set of alternatives in a single run, and randomized initialization and stochastic

```
Begin
    Initialize the food positions randomly $x_i$, $i = 1, 2, \dots, n$
    Evaluate fitness $f(x_i)$ of the individuals
    While stop condition not met Do
        Employed phase:
            Produce new solutions with $k$, $j$ and $\varphi$ at random
            $v_{ij} = x_{ij} + \varphi_{ij} \cdot (x_{ij} - x_{kj}),\ k \in \{1, 2, \dots, n\}, j \in \{1, 2, \dots, d\}, \varphi \in [0, 1]$
            Evaluate solutions
            Apply greedy selection process for the employed bees
        Onlooker phase:
            Calculate probability values for the solutions $x_i$
            $P_i = \frac{f_i}{\sum_{j=1}^{n} f_j}$
            Produce new solutions from $x_i$ selected using probability $P_i$
            Evaluate solutions
            Apply greedy selection for the onlookers
        Scout phase:
            Find abandoned solution: If limit exceeds, replace it with a new random solution
            Memorize the best solution achieved so far
    End While
        Output the results
End
```

Figure 6. Pseudo-code of the artificial bee colony algorithm.

Table 1. Characteristics of different evolutionary computation and swarm intelligence (SI) algorithms

Characteristic of algorithm	Genetic algorithm	Differential evolution	Genetic programming	Ant colony optimization	Particle swarm optimization	Artificial bee colony algorithm
Algorithm type	Genotypic/ phenotypic	Phenotypic	Phenotypic	Phenotypic	Phenotypic	Phenotypic
Developed by	Holland[3]	Storn and Price[10]	Koza[8]	Dorigo *et al.*[14]	Eberhart and Kennedy[15]	Karaboga[21]
Basic principle	Natural selection or survival of the fittest	Survival of the fittest	Survival of the fittest	Cooperative group intelligence of swarm	Cooperative group intelligence of swarm	Collective knowledge of bees
Solution representation	Binary/real-valued	Real-valued	Expression trees	Graph or a similar structure for path-covering of ants	Real-valued	Real-valued
Fitness	Scaled objective value	Objective function value	Scaled objective value	Scaled objective value	Objective function value	Objective function value
Evolutionary operators	Mainly crossover (other operator, mutation)	Mainly mutation (other operator, crossover)	Crossover and mutation	None	None	None
Selection process	Probabilistic, preservative	Deterministic, extinctive	Probabilistic, extinctive	Probabilistic, preservative	Deterministic, extinctive	Probabilistic, preservative
Type of decision variables	Applicable to both real values and/or discrete values	Mainly for real values (can be used for discrete variables)	Mainly for real values	Mainly for discrete values	Mainly for real values (applicable for discrete variables)	Applicable to both discrete and real values
Applicability to problems	For all types of problems (linear/ nonlinear)	For all types of problems (linear/ nonlinear)	For all types of problems (linear/ nonlinear)	For all types of problems (linear/ nonlinear)	For all types of problems (linear/ nonlinear)	For all types of problems (linear/ nonlinear)

search in their operation helps to overcome local optima. These special characteristics are helping the bio-inspired algorithms to achieve well-spread and well-diverse pareto optimal solutions in a single run quickly. Hence they are receiving wider applications in different areas ranging from robotics to water resources using MOEAs[1,26–28] and MOSAs[18,29–31].

Artificial neural networks

Artificial neural network (ANN) is another important computational method that was developed in the 1970s and 1980s, and is gaining popularity as a modern statistical data-modelling tool for many nonlinear, difficult-to-represent and complex problems in science and engineering. The ANNs are inspired from a close examination of the central nervous system and the neurons, axons, dendrites and synapses, which constitute the processing elements of biological neural networks as investigated by neuroscience experts[32]. In the ANN, simple artificial nodes (called neurons) are connected together to form a network of nodes mimicking the biological neural networks. A neural network consists of an interconnected group of artificial neurons, and it processes information using a connectionist approach to computation. The neural models are usually used to model complex relationships between inputs and outputs (called function approximation), or to find patterns in data (called pattern recognition)[33].

In general, the ANN is an adaptive system that changes its structure based on external or internal information that flows through the network during the learning phase. To achieve robust learning from the given set of patterns, various kinds of neural network mechanisms are explored. These include feed-forward neural networks (FFNNs), recurrent neural networks, time-delayed neural networks, real-time recurrent neural networks, etc. A standard architecture of the FFNN is shown in Figure 7.

Network architecture mainly denotes the number of input and output variables, the number of hidden layers, and the number of neurons in each hidden layer. It

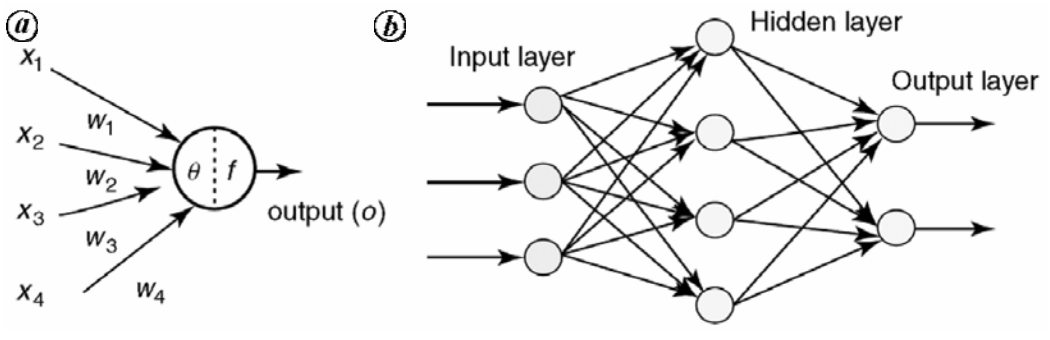

Figure 7. Architecture of artificial neural network. ***a***, Artificial neuron; ***b***, Multilayered feed-forward neural network.

determines the number of connection weights and the way information flows through the network. The sole role of the nodes of the input layer is to relay the external inputs to the neurons of the hidden layer. Hence the number of input nodes corresponds to the number of input variables. The outputs of the last hidden layer are passed to the output layer which provides the final output of the network.

Depending on the procedure through which ANNs establish the given task of function of approximation or pattern recognition, there are mainly two classes of network training known as supervized and unsupervized learning. In supervized training, in order to learn the relationships, inputs and outputs are specified for each pattern during the training period (e.g. FFNN); whereas in unsupervized training only inputs are specified to the neural networks and it should be able to evolve itself to achieve a specific task such as pattern recognition or classification (e.g. self-organizing maps). There are many methods to find optimal weights of neural networks[33], such as error back-propagation algorithm, conjugate gradient algorithm, cascade correlation algorithm, quasi-Newton method, Levenberg–Marquardt algorithm, radial basis function algorithm, etc. Apart from this, EAs have also been proposed for finding the network architecture and weights of neural networks, and have been applied to various problems[34,35].

ANNs are receiving increasing attention with wider applications for modelling complex and dynamic systems in science and engineering. Since any modelling effort will have to be based on an understanding of the variability of the past data, ANNs have some special useful characteristics in this regard. In contrast to conventional modelling approaches, ANNs do not require an in-depth knowledge of the driving processes, nor do they require the form of the model to be specified a priori[25]. Over the last two decades, ANNs have been used extensively to model complex nonlinear dynamics, which is not adequately represented by linear models[33,35]. As the cited papers also include discussion on various applications, the interested reader may refer them for more details on specific applications.

Concluding remarks

Researchers have developed various algorithms for solving complex problems by modelling the behaviours of nature and biological processes, which resulted in several evolutionary computation and SI algorithms. EAs are inspired from Darwin's principle of evolution – 'survival of the fittest'. SI algorithms are inspired from biological activities such as food searching by the ants, bird flocking, fish schooling, honey-bee mating process, etc. Algorithms such as GA based on the theory of survival of the fittest, ACO based on ant swarm, PSO based on bird flock and fish schooling, and HBMO based on honey-bee mating have been proposed in various studies to solve optimization problems in science and engineering. These computational algorithms can provide acceptable optimal solutions to many complex problems that are difficult to cope using conventional methods (due to their nature that may imply discontinuities of the search space, non-differentiable objective functions, nonlinear relationships or imprecise arguments and function values). Thus the use of these computational algorithms for solving practical problems is becoming more popular. Still there is a lot of scope for research and their applications in different areas of science, engineering and industrial problems. By considering the specific advantages of the EA and SI algorithms, it will be a wise idea to take benefit of the special advantages of these methods in solving practical problems.

1. Deb, K., *Multi-objective Optimization using Evolutionary Algorithms*, John Wiley, Chichester, UK, 2001.
2. Janga Reddy, M., Swarm intelligence and evolutionary computation for single and multiobjective optimization in water resource systems. PhD thesis, Indian Institute of Science, Bangalore, 2006.
3. Holland, J. H., *Adaptation in Natural and Artificial Systems*, The MIT Press, 1975.
4. Bonabeau, E., Dorigo, M. and Theraulaz, G., *Swarm Intelligence: From Natural to Artificial Systems*, Oxford University Press, New York, 1999.
5. Michalewicz, Z. and Fogel, D. B., *How to Solve It: Modern Heuristics*, Springer, 2004.
6. Brownlee, J., *Clever Algorithms: Nature-inspired Programming Recipes*, LuLu.com, Australia, 2011, ISBN: 978-1-4467-8506-5.

7. Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison Wiley, Reading, NY, 1989.
8. Koza, J. R., *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, The MIT Press, Cambridge, MA, 1992.
9. Ferreira, C., Gene expression programming: a new adaptive algorithm for solving problems. *Complex Syst.*, 2001, **13**, 87–129.
10. Storn, R. and Price, K., Differential evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report, TR-95-012, International Computer Science Institute, Berkley, 1995.
11. Price, V. K., Storn, R. M. and Lampinen, J. A., *Differential Evolution: A Practical Approach to Global Optimization*, Springer-Verlag, Berlin, 2005.
12. Seeley, T., *Honeybee Ecology: A Study of Adaptation in Social Life*, Princeton University Press, Princeton, 1985.
13. Millonas, M. M., Swarms, phase transitions, and collective intelligence. In *Artificial Life III*, Addison-Wesley, Reading, 1994, pp. 417–445.
14. Dorigo, M., Maniezzo, V. and Colorni, A., Positive feedback as a search strategy. Technical Report 91-016, Politecnico di Milano, Italy, 1991.
15. Eberhart, R. C. and Kennedy, J., A new optimizer using particle swarm theory. In Proceedings Sixth Symposium on Micro Machine and Human Science, IEEE Service Center, Piscataway, NJ, 1995, pp. 39–43.
16. Dorigo, M. and Stutzle, T., *Ant Colony Optimization*, MIT Press, Cambridge, MA, 2004.
17. Nagesh Kumar, D. and Janga Reddy, M., Ant colony optimization for multipurpose reservoir operation. *Water Resour. Manage*., 2006, **20**, 879–898.
18. Kennedy, J., Eberhart, R. C. and Shi, Y., *Swarm Intelligence*, Morgan Kaufmann, San Francisco, 2001.
19. Nagesh Kumar, D. and Janga Reddy, M., Multipurpose reservoir operation using particle swarm optimization. *J. Water Resour. Plan. Manage., ASCE*, 2007, **133**, 1–10.
20. Abbass, H. A., Marriage in honey bees optimization: a haplometrosis polygynous swarming approach. In The Congress on Evolutionary Computation, CEC2001, Seoul, Korea, 2001, vol. 1, pp. 207–214.
21. Karaboga, D., An idea based on honey bee swarm for numerical optimization. Technical Report-TR06, Erciyes University, Turkey, 2005.
22. Labadie, J. W., Optimal operation of multireservoir systems: state-of-the-art review. *J. Water Resour. Plan. Manage., ASCE*, 2004, **130**, 93–111.
23. Rani, D. and Moreira, M. M., Simulation–optimization modelling: a survey and potential application in reservoir systems operation. *Water Resour. Manage*., 2010, **24**, 1107–1138.
24. Nicklow, J. *et al.*, State of the art for genetic algorithms and beyond in water resources planning and management. *J. Water Resour. Plan. Manage*., *ASCE*, 2011, **136**, 412–432.
25. Karaboga, D. and Akay, B., A survey: algorithms simulating bee swarm intelligence. *Artif. Intell. Rev.*, 2009, **31**, 61–85.
26. Janga Reddy, M. and Nagesh Kumar, D., Optimal reservoir operation using multi objective evolutionary algorithm. *Water Resour. Manage.*, 2006, **20**, 861–878.
27. Janga Reddy, M. and Nagesh Kumar, D., Multi-objective differential evolution with application to reservoir system optimization. *J. Comp. Civ. Eng*., *ASCE*, 2007, **21**, 136–146.
28. Janga Reddy, M. and Nagesh Kumar, D., Evolving strategies for crop planning and operation of irrigation reservoir system using multi-objective differential evolution. *Irrig. Sci.*, 2008, **26**, 177–190.
29. Janga Reddy, M. and Nagesh Kumar, D., Multi-objective particle swarm optimization for generating optimal trade-offs in reservoir operation. *Hydrol. Proc.*, 2007, **2**, 2897–2909.
30. Janga Reddy, M. and Nagesh Kumar, D., An efficient multi-objective optimization algorithm based on swarm intelligence for engineering design. *Eng. Opt.*, 2007, **39**, 49–68.
31. Janga Reddy, M. and Nagesh Kumar, D., Performance evaluation of elitist-mutated multi-objective particle swarm optimization for integrated water resources management. *J. Hydroinf.*, 2009, **11**, 78–88.
32. Hertz, J., Palmer, R. G. and Krogh, A. S., *Introduction to the Theory of Neural Computation*, Perseus Books, 1990.
33. Haykin, S., *Neural Networks: A Comprehensive Foundation*, Prentice Hall, 1999.
34. Nagesh Kumar, D., Janga Reddy, M. and Maity, R., Regional rainfall forecasting using large scale climate teleconnections and artificial intelligence techniques. *J. Intell. Syst.*, 2007, **16**, 307–322.
35. Maier, H. R., Jain, A., Dandy, G. C. and Sudheer, K. P., Methods used for the development of neural networks for the prediction of water resources variables in river systems: Current status and future directions. *Environ. Model. Soft.*, 2010, **25**, 891–909.

Received 14 December 2011; revised accepted 20 June 2012