**Application of an Artificial Fish Swarm Algorithm in Symbolic Regression**

Qing LIU†a), Tomohiro ODAKA†, Jousuke KUROIWA†, and Hisakazu OGURA†, Members

**SUMMARY** An artificial fish swarm algorithm for solving symbolic regression problems is introduced in this paper. In the proposed AFSA, AF individuals represent candidate solutions, which are represented by the gene expression scheme in GEP. For evaluating AF individuals, a penalty-based fitness function, in which the node number of the parse tree is considered to be a constraint, was designed in order to obtain a solution expression that not only fits the given data well but is also compact. A number of important conceptions are defined, including distance, partners, congestion degree, and feature code. Based on the above concepts, we designed four behaviors, namely, randomly moving behavior, preying behavior, following behavior, and avoiding behavior, and present their respective formalized descriptions. The exhaustive simulation results demonstrate that the proposed algorithm can not only obtain a high-quality solution expression but also provides remarkable robustness and quick convergence.

**key words:** artificial fish swarm algorithm, symbolic regression, parse tree, optimization, penalty

1. Introduction

In science and industry, regression analysis [1] is a basic and important means of scientific investigation, enabling identification of the functional relationship between independent variables and the dependent variable. This always requires a predefined assumption of a functional relationship. However, as practical problems are usually complex, non-linear, and full of random noise, in many cases, it is almost impossible to predefine a reasonable assumption. Taking the functional relationship between the share index and the time sequence as an example, the functional relationship is far too complex to predefine a rational assumption. Even for a seasoned expert, this is an impossible task. In this case, regression analysis is no longer applicable. Fortunately, an entirely different method, symbolic regression, does not require an assumption. From this perspective, symbolic regression has more general applicability. Symbolic regression is the problem of seeking to discover an exact symbolic expression of a function that fits (or approximately fits) the experimental data set from an unknown curve [2]. Symbolic regression is an important problem because it is related to general machine learning problems [3]. In addition, the importance of symbolic regression in science and industry has been well recognized [4]–[6]. Symbolic regression will become increasingly important due to the increasing complexity of problems in science and industry.

At present, genetic programming (GP), pioneered by John Koza [2], is a well-known and effective solution for solving symbolic regression problems. Genetic programming is an evolutionary computing technique for solving problems automatically. In fact, GP can be considered as an extension of genetic algorithm (GA). Like general GA, GP relies on a population and selects superior individuals according to their fitness, generation by generation. That is to say, GP works under the same framework as GA. On the other hand, unlike general GA, GP adopts parse trees to represent the candidate solutions of the problem, instead of linear chromosomes of a fixed length string. The genotype in GP is more complex than that in GA, and the definition of the phenotype in GP is more abstract than that in GA. In the parse tree, leaf nodes are referred to as terminals, whereas the other nodes are referred to as primitive functions. For symbolic regression problems, the terminals are usually independent variables and constants, whereas the primitive functions are usually basic mathematical operations, such as $+$, $-$, $\ast$, $/$, sin, exp, and ln. The primitive functions and terminals are organized into a certain expression from the root node to the leaf nodes. Since there is a distinct difference between the chromosome structure of GP and that of GA, several genetic operations of GA, such as mutation and crossover, cannot be used to directly operate parse trees. Therefore, they were redefined in order to adapt the parse tree representation. The fitness of the parse tree individual depends on the error when it fits the unknown curve. Superior chromosomes generate less error than other chromosomes. By evolving, the error will decrease. This is an error-driven evolutionary mechanism. Under the above described framework, GP evolves the population iteratively until a desirable expression is generated. A number of studies on the successful use of GP for solving symbolic regression problems have been reported, e.g., Refs. [4] and [5]. However, GP is not flawless and has two major shortcomings can be summarized as follows:

1. **Code bloat**

Due to the iterative action of the crossover and mutation operators, the chromosomes of the parse tree may grow indefinitely, regardless of whether or not the growth acts to improve the solutions [7]. This phenomenon is referred to as code bloat. Code bloat may lead to the exhaustion of memory and reduced efficiency of genetic operators. Moreover, code bloat significantly expands the search space of the problem, so that searching for an optimal solution will be more difficult.

---

Manuscript received September 27, 2012.

†The authors are with University of Fukui, Fukui-shi, 910–8501 Japan.

a) E-mail: liu@m.his.u-fukui.ac.jp

DOI: 10.1587/transinf.E96.D.872
(2) Limitation of genetic operations

In GP, the parse tree is not only the genotype but also the phenotype. Genetic operators act directly on parse trees. If the phenotype exhibits a certain amount of functional complexity, it is extremely difficult to reproduce the genotype with modification [8]. Since genetic operators cannot guarantee that invalid parse trees will not occur, in many cases, genetic operators are limited.

In 2001, Ferreira proposed an improved version of GP, referred to as gene expression programming (GEP) [8]. Gene expression programming shares the same diagram representation of GP (parse trees), but produces different entities, which are referred to as gene expressions. The gene expression is a novel scheme that uses a linear chromosome with a fixed length to represent the parse tree. Owing to the fixed length of the chromosome, the problem of code bloat in GP is not a problem in GEP. In addition, since none of the genetic operators can result in an invalid parse tree by acting on gene expression representation, genetic operators will no longer be limited. Obviously, the two shortcomings of GP have been avoided in GEP, and the remarkable performance obtained by using GEP to solve symbolic regression problems has proven that GEP is superior to GP [8], [9]. However, the difference between GEP and GP is only the genotype of the chromosome. That is to say, from GP to GEP, the improvement only relies on a more superior scheme of chromosome encoding (gene expression), instead of other aspects. In other words, GEP still works under the same framework as GA. In essence, GEP is still an extension of GA. Therefore, GEP unavoidably possesses all of the shortcomings of GA, especially premature and slow convergence. These are the native shortcomings of GA, which cannot be avoided by adopting only a superior encoding scheme.

The artificial fish swarm algorithm (AFSA) is a novel optimization algorithm, which was first proposed by X.L. Li in 2002 [10]. Inspired by the activities of fish in water, the AFSA generally adopts three famous rules that were summarized by Reynolds [11] in order to simulate various behaviors of fish individuals and provides a flexible optimization framework for solving difficult problems. As a swarm intelligence optimization algorithm, the AFSA also relies on a population, which consists of a number of artificial fish (AF) individuals. Each AF individual of the population is an agent that searches the solution space via various behaviors, such as preying, following, and swarming. The optimization mechanism of the AFSA is very different from that of the GA. The core of the optimization mechanism of GA is competition, whereas that of the AFSA is cooperation. Each AF individual searches its own neighborhood via individual behaviors and exchanges and shares optimization experiences with other individuals via social behaviors, which reflect the abovementioned cooperation. Up to now, there have been several studies on using the AFSA to solve various problems [12]–[15]. In these reported studies, the AFSA always performs much better than GA, even if the same encoding scheme is adopted. Obviously, the cooperation-based framework of the AFSA is superior to the competition-based framework of GA. Therefore, for further advancement of solving symbolic regression problems, combining the gene expression scheme of GEP and the cooperation-based framework provided by the AFSA follows naturally.

To the best of our knowledge, studies on the usage of the AFSA for solving symbolic regression problems have not yet been reported. The purpose of the present paper is to investigate the applicability and efficiency of the AFSA for solving symbolic regression problems. In the present study, we use the gene expression scheme in GEP to represent AF individuals in the AFSA. In order to evaluate AF individuals more reasonably, a novel penalty-based fitness function is designed. Several important concepts are defined and on the basis of the defined concepts, several behaviors of AF individuals are developed. We herein present a detailed description of the proposed AFSA and demonstrate its validity through exhaustive simulation experiments.

The remaining part of the present paper is organized as follows. In Sect. 2, the gene expression representation is briefed and our proposed algorithm is detailed. In Sect. 3, the proposed algorithm is tested through exhaustive simulation experiments in order to investigate various aspects of performance and for comparison with GEP. Section 4 presents a discussion of how to rationally set the parameters of the proposed algorithm and an analysis of various behaviors of the AF individual. Finally, conclusions are presented in Sect. 5.

2. Artificial Fish Swarm Algorithm for Symbolic Regression

2.1 Encoding Method of the AF Individual

The key to using AFSA to solve problems is how to encode a potential solution of problems into an AF individual representation in AFSA. For symbolic regression problems, a potential solution is an expression that is represented by a parse tree. How is a parse tree encoded into an AF individual? In this paper, we adopt a gene expression scheme, which is a linear string of fixed length, to represent an AF individual. Here, we briefly review the scheme. For additional details, see Ref. [8].

A gene expression representation consists of two parts, a head and a tail. The head consists of both functions (elements from the function set $F$) and terminals (elements from the terminal set $T$), whereas the tail consists of only terminals. The length of the head $h$ is chosen according to the problem, whereas the length of the tail $t$ depends on $h$. Actually, $t = h \cdot (n - 1) + 1$, where $n$ represents the number of arguments of the function with the most arguments. A simple example will be presented for the purpose of explanation. Consider, for instance, the following algebraic expression:

$$f(x) = e^{\sin x - x} + \pi \cdot \ln x$$

which can be represented as a parse tree, as shown in Fig. 1.

Obviously, for this parse tree, $F = \{+, -, \ast, \ln, \sin, \exp\}$, $T = \{x, \pi\}$, in this case, $n = 2$. For example, for $h = 6$ and
of the parse tree can be easily represented as follows:

+  \text{exp}  \*  -  \pi  \ln  \sin  x  x  x  x  \pi  x

|   0   |   1   |   2   |   3   |   4   |   5   |   6   |   7   |   8   |   9   |  10  |  11  |  12  |
|------|------|------|------|------|------|------|------|------|------|------|------|------|
|      | exp  |   *  |   -  | \pi  | \ln  | \sin |   x  |   x  |   x  |   x  |   \pi |   x  |

Open reading frame (ORF)   || Noncoding region

where the part from suffix 0 to suffix 9 represents the structure of the parse tree and is referred to as the open reading frame (ORF). Apart from ORF, the remaining part (shaded) is referred to as the noncoding region, which is redundant for representing the parse tree of Fig. 1. In fact, it is very common for gene expression representation to have a noncoding region. Owing to the noncoding region, any modifications of this kind of representation without restrictions will not result in an invalid parse tree. Therefore, the noncoding region is absolutely the essence of gene expression scheme. Moreover, the length of the ORF may occasionally be equal to the length of the gene expression representation. In other words, a gene expression representation may not have a noncoding region. From this simple example, we know that the start site of the ORF is always the first point of the gene expression scheme, whereas the termination point of the ORF does not always coincide with the last position of the representation.

The gene expression scheme is intuitionistic and allows modification without restrictions. This is why we adopt such a scheme to represent AF individuals.

2.2 Evaluation of AF Individuals

The task of this subsection is to design a reasonable fitness function by which to evaluate whether an AF individual is superior. In this paper, an AF individual is a potential solution of a symbolic regression problem, which is essentially an algebraic expression. The goal of symbolic regression is to find an expression that fits data of an unknown curve within an acceptable error of the correct value, so the fitness of the AF individual should depend on the actual error of the correct value. This is also why no matter GP or GEP adopts the error-driven mechanism to evolve population.

In the proposed algorithm, we use the average relative error to evaluate the fitness of AF individuals. Assume that the number of data points to be fitted is \( n \), and each data point can be noted as \((x_i, y_i)\). Here, \( x_i \) denotes the independent variable of the \( i \)-th data point, whereas \( y_i \) denotes the dependent variable of the \( i \)-th data point. This way, the fitness function should be expressed as follows:

\[
\text{fitness} = \frac{1}{n} \sum_{i=1}^{n} \frac{|f(x_i) - y_i|}{|y_i|}
\]

where \( f(\cdot) \) denotes the exact algebraic expression, which is represented by the AF individual to be evaluated. Note that, for a perfect fitting, independent of the form of \( f(\cdot) \), the expression \( f(x_i) = y_i \) holds for each data point. This means that \( \text{fitness} = 0 \). Thus, the lower the average relative error is, the better the AF individual will be. In this case, the symbolic regression problem is a minimum problem.

We also noticed a problem in that studies on symbolic regression focus only on the precision of the expression to be found, but ignore its concision. However, the concision of the obtained expression should also be an expectation in symbolic regression. Taking the application of symbolic regression in knowledge discovery in databases (KDD) as an example, a compact expression is more easily explained and understood than complex expressions. Thus, the optimal expression should not only fit the data well but should also be compact, or at the very least, not too complex. Therefore, we consider concision as a constraint for symbolic regression. Then, the problem of how to describe the concision degree of an expression should first be solved. The concision of an expression depends on the total number of nodes of the parse tree representation of the expression. The fewer nodes the parse tree has, the more compact the expression will be. For counting the number of nodes of the parse tree, there is no need to decode the gene expression representation of the AF individual into a parse tree. Instead, we can directly count the number of nodes of the parse tree from the gene expression representation. Assume that the parse tree has a total of \( k \) nodes and that each node takes \( n_i \) arguments. In particular, the number of arguments taken by the terminal node is considered to be 0, where \( k \) and \( n_i \) \((i = 1, 2, \cdots, k)\) satisfy the following relationship:

\[
k - 1 = \sum_{i=1}^{k} n_i
\]

According to Eq. (3), we can obtain the algorithm for counting \( k \). The pseudocode of this counting algorithm is as follows:

```
Algorithm of counting the number of the nodes of a parse tree
0  sum_Args=0;  // sum of number of arguments of all nodes
1  for i=0 to (h+t)-1
2     sum_Args+=n_i;
3  if ( i == sum_Args )
4     break;
5  end if
6  end for
7  output sum_Args+1;  // k-1=sum_Args
```
Hence, we modified the former fitness function into a penalty-based fitness function:

\[
\text{fitness} = \begin{cases} 
\frac{1}{n} \sum_{i=1}^{n} \frac{|f(x_i) - y_i|}{|y_i|} + p \cdot (k - K) & (k > K) \\
\frac{1}{n} \sum_{i=1}^{n} \frac{|f(x_i) - y_i|}{|y_i|} & (k < K)
\end{cases}
\] (4)

where \( p \) is a penalty factor, \( K \) is a constraint of \( k \) and can be chosen according to the problem. In addition, this fitness function requires that each \( y_i \) of the given data not be 0. Otherwise, a div0 error will occur. However, the requirement that each \( y_i \) cannot be 0 is inurbane. Therefore, in order to avoid the div0 error, we replace \( y_i \) with \( 0.000001 \cdot \min\{y_k | k = 1, 2, \ldots n \} \) when \( y_i = 0 \).

2.3 Definitions of Concepts

In order to introduce the proposed algorithm, some important concepts should first be introduced.

2.3.1 Distance

In the AFSA, distance is a very basic and important concept. Actually, distance is used to describe the degree of similarity between any two points in the solution space. A shorter distance means a higher degree of similarity. The standard AFSA is used for the continuous function optimization, so that the distance is measured as Euclidean distance [10]. However, for the combinatorial optimization problem, the Hamming distance was adopted [14]. There are also a number of other methods for measuring the distance in other AFSA applications, e.g. the method described in Ref. [12]. Obviously, the method used to measure the distance depends on the property of problem. Generally speaking, symbolic regression is a combinatorial optimization problem, which involves determining an optimal combination of various symbols to represent an expression. Although the Hamming distance would be a native way to measure distance, due to the adopted gene expression scheme, the Hamming distance is not always rational for this measurement. The reason for this is explained using three different AF individuals, the genotypes and the phenotypes of which are shown in Table 1 and Fig. 2, respectively.

Table 1 and Fig. 2 show that each AF individual is different from the other two AF individuals. The difference in genotypes between AF individual 1 and AF individual 2, and that between AF individual 1 and AF individual 3, are only 1 bit. However, the extents of the differences in phenotype (expression) are not the same. On one hand, the phenotype of AF individual 1 is very similar to that of AF individual 2, this phenomenon can be reasonably explained by their very close distance in Hamming space (Hamming distance: 1). On the other hand, although the Hamming distance between the genotypes of AF individual 1 and AF individual 3 is also 1, the phenotype of AF individual 1 is quite different from that of AF individual 3. This indicates that the Hamming distance is not always rational for measuring the similarity of any two AF individuals. However, the Hamming distance is absolutely rational for measuring the distance between any two AF individuals which have the same parse tree structure. Note that the representations of the parse tree structures of AF individual 1 and AF individual 2 are the same and that the Hamming distance representing the similarity of the two individuals is rational. This is not a coincidence, but rather is inevitable. Conversely, the Hamming distance is irrational only when measured be-
between two individuals with different parse tree structures. According to the above analysis, in this paper, we use the Hamming distance to measure the similarity between two solutions, but only for measuring the similarity between individuals with same parse tree structures and only for the part of ORF, without considering the noncoding region. Given two AF individuals, $X_1$ and $X_2$, with the same parse tree structure, the distance between these individuals is expressed as $\text{Distance}(X_1, X_2)$. As for solutions with different parse tree structures, we do not need to measure the distance between any two of them in the proposed algorithm.

2.3.2 Partners and Congestion Degree

In the AFSA, AF Individuals search the solution space collaboratively when they exhibit social behaviors. This means that each individual requires a number of partners. In this paper, the concept of partners is very important in the proposed AFSA. In terms of data storage, whatever the AFSA, or any other algorithm that relies on population, we usually store the population with an array of objects, which are AF individuals for the case of the AFSA. In order to explain how we define partners of a certain AF individual, we present the following example:

Array of AF individuals

$0 \ 1 \ 2 \ \cdots \ i-r \ \cdots \ i \ \cdots \ t+r \ \cdots \ \text{popsize-1}$

$\uparrow \text{current AF individual}$

Assume that the $i$th element of the array is the current AF individual. The partners of the current AF individual can then be defined with these elements from suffix $i - r$ to suffix $i + r$, as indicated by the shaded area. In this case, the current AF individual has $2r$ partners. Most of the AF individuals stored in the array have $2r$ partners. However, a few individuals have fewer than $2r$ partners. Taking the 0th and (popsize−1)th individuals as examples, each of these examples has only $r$ partners. Since the 0th individual has no left neighbors and the (popsize−1)th individual has no right neighbors. In order to ensure that each AF individual has $2r$ partners, we convert the suffix of this array, as follows:

For any suffix $s \in [i - r, i + r]$ ($i \in [0, \text{popsize} - 1]$)

i) If $s < 0$, then $s \leftarrow s + \text{popsize};$

ii) If $s > 0$, then $s \leftarrow s \% \text{popsize}.$

Thus, the linear structure of the array is mapped to the ring topology shown in Fig. 3. The 0th and (popsize−1)th individuals are joined. Then, each individual on the ring has $2r$ partners. Based on the concept of partners, we define the congestion degree as follows:

$$c = \frac{n_s}{2r}$$

where $c$ represents the current congestion degree and $n_s$ represents the number of the partners that have the same parse tree structure as the current AF individual. The congestion degree describes the diversity of the parse tree structures of the partners of the current AF individual from contrary. A larger congestion degree indicates poorer diversity of parse tree structures. In order to compute $n_s$, we should determine the method of judging whether the parse tree structures of two AF individuals are the same. Actually, it is not essential to first decode the two individuals into parse tree representations. This can be achieved very easily, as described in detail in the next subsection.

2.3.3 Feature Code and Visual Field

In order to determine whether one AF individual has the same parse tree structure as another AF individual, we simply need to determine whether all the feature codes of the AF individuals are the same. The feature code is the number of arguments taken by each operator. In particular, the feature code of the terminal is considered to be 0. For example, see Table 2.

Table 2 Feature code. (shaded area: noncoding region)

| Feature code | (a) AF individual 1 | (b) AF individual 2 |
|--------------|--------------------|--------------------|
| Gene expression | exp − sin / x x p x x | exp + ln * x p x x |
| Feature code | 1 2 1 2 0 0 NULL | 1 2 1 2 0 0 NULL |

Table 2 shows that the feature codes of AF individual 1 are the same as those of AF individual 2, whereas the gene expressions of these AF individuals are different. However, Fig. 4 shows that the parse tree structures of two individuals are the same. In other words, as long as individuals have the same feature codes, they must have the same parse tree structure. This example reveals that the feature codes can be easily used to investigate the parse tree structure of an AF individual without decoding.
Then, we define the visual field of an AF individual with the set of all potential solutions, which have the same feature codes to the mentioned AF individual. In other words, all potential solutions in the visual field of an AF individual have the same parse tree structure. It is rational to measure the Hamming distance between any two points in the visual field of an AF individual, as described in Sect. 2.3.1.

2.4 Description of Various Behaviors of AF Individuals

AF individuals are agents who search the solution space via various behaviors. In the process of the AFSA, each individual of the population iteratively implements behaviors according to a certain behavior strategy. Assuming that the current AF individual is \( X \), the fitness of \( X \) is expressed as \( \text{fitness}(X) \).

2.4.1 Randomly Moving Behavior

Randomly moving behavior is behavior in which AF individual \( X \) randomly moves \( m \) steps (Hamming distance) in its own visual field, whereas \( m \) is the number of random steps that satisfy \( \text{MinStep} < m < \text{MaxStep} \). We implement the motion of the AF individual through modification of its gene expression. In addition, moving in one’s own visual field means that modification of its gene expression should not change its parse tree structure. In other words, feature codes ought not to be changed. We present an example in order to explain how to implement a randomly moving operator. Table 3 is a diagram to show the process that AF individual \( X \) moves to a new position \( X' \) by performing randomly moving behavior. The gene expression and feature code of AF individual \( X \) is shown in Table 3. Suppose that AF individual \( X \) will randomly move 3 steps in its own visual field before arriving at \( X' \). The underlined bits of its gene expression are to be modified. For each of these bits randomly generate a new gene that has the same feature code as the original gene. Then, \( X \) arrives at \( X' \), without changing the parse tree structure. As shown in Table 3, we see that \( \text{Distance}(X, X') = 3 \), which means that AF individual \( X \) really moved 3 steps in its visual field.

Must be emphasized, it is always forbidden to modify the bits in the noncoding region because modification in this region cannot change the expression represented by the AF individual.

| Gene expression | \( \exp \) | \( - \sin \) | \( / \) | \( \times \) | \( \times \) | \( \times \) | \( \times \) | \( \times \) | NULL |
|-----------------|---------|--------|-----|-----|-----|-----|-----|-----|
| Feature code    | 1       | 2      | 1   | 2   | 0   | 0   | 0   | \( \times \) | NULL |
| AF individual \( X' \) (Shaded area: noncoding region) |
| Gene expression | \( \cos \) | \( - \sin \) | \( \pm \) | \( \times \) | \( \times \) | \( \times \) | \( \times \) | NULL |
| Feature code    | 1       | 2      | 1   | 2   | 0   | 0   | 0   | \( \times \) | NULL |

2.4.2 Preying Behavior

Preying behavior is behavior in which the AF individual looks for a better position and then moves to this position. When preying, AF individual \( X \) tries another position \( X' \) that is chosen at random in its visual field. If the tried position \( X' \) is better than the current position, i.e., \( \text{fitness}(X') < \text{fitness}(X) \), then replace \( X \) by \( X' \). Otherwise, try another time. If AF individual \( X \) cannot find a better position after at most \( \text{try Number} \) times, then the position remains unchanged, where \( \text{try Number} \) is a parameter.

In order to obtain a tried position \( X' \) in the visual field of an AF individual, let a copy of the AF individual perform randomly moving behavior once and then consider the copy to be the tried position \( X' \).

2.4.3 Following Behavior

Following behavior is behavior in which an AF individual imitates and learns a superior individual among its partners. When following, first find out the most superior individual among the partners of the current AF individual \( X \). Assume \( X' \) to be the most superior AF individual.

1) If \( X' \) is not superior to \( X \), then allow \( X \) to prey.
2) If \( X' \) is superior to \( X \), then perform the following procedure. Generate random step \( m \), which satisfies \( \text{MinStep} < m < \text{MaxStep} \). Then randomly choose \( m \) bits from the part of ORF of \( X' \) to replace corresponding bits of \( X \). In this operation, without considering the correspondence of feature codes, the parse tree structure of the current AF individual may be changed.

2.4.4 Avoiding Behavior

Avoiding behavior is behavior in which AF individuals avoid the fish swarm to be much too crowded. When performing the avoiding behavior, the number of the partners which have the same parse tree structure as the current AF individual is first determined. The value of the congestion degree is then computed according to Eq. (5). If the value is less than \( \text{MaxC} \), which is the acceptable upper bound of the congestion degree, leave \( X \) unchanged. Otherwise, randomly select several bits of the part of ORF and then randomly generate new genes (either operators or terminals) that have different feature codes from those of the original genes for the chosen bits so that the parse tree structure must be changed.

2.4.5 Behavioral Strategy and AF Specimen

The AFSA is a flexible open optimization framework in which each behavior of the AF individual is a type of search pattern. Each AF individual performs behaviors according to a behavioral strategy during algorithm execution. For different problems, the adopted behavioral strategy may not be fixed. Generally speaking, we can freely adopt various
behavioral strategies so long as it is beneficial for problem solving, according to the property of the problem to be solved. In this paper, we first make each AF individual perform avoiding behavior. Then, preying behavior is performed. If the AF individual does not improve via preying, then perform following behavior. Then, if it still doesn’t arrive at a better position in solution space, perform randomly moving behavior.

As the algorithm progresses, the best AF individual of the population in each iteration is recorded as an AF specimen. After performing various behaviors during one iteration, once any AF individual has become superior to the AF specimen, we use the superior AF individual to update the AF specimen.

2.5 Algorithm Steps

(0) Input data set for symbolic regression.
(1) Initialize AF population and estimate each individual.
(2) Set the AF specimen.
(3) for iteration ← 1 to MaxIteration
   | for i ← 1 to Population
   | | AF[i].Avoiding();
   | | AF[i].Preying();
   | | if(not get better)
   | | | | AF[i].Following();
   | | | | if (not get better)
   | | | | | | AF[i].RandomlyMoving();
   | | | | if (AF[i] is better than AF specimen)
   | | | | | | AF[i].Update(AF specimen);
   | | end if
   | | end if
   | end for
end for
(4) Output the optimal solution of the AF population (AF specimen).

3. Simulations

We implemented the proposed algorithm in C++. All of the experiments were performed on a computer with an Intel Core i7-2600k 3.4 GHz (Turbo boost 3.7 GHz) CPU and 6 GB of RAM. The operating system running on this computer is windows 7 sp1 (x64).

3.1 A Computational Example

In order to verify the search ability of the proposed algorithm, we first create a computational example of the symbolic regression problem and then use the proposed AFSA to solve the problem. In Table 4, the values of independent variable x and the corresponding dependent variable y is shown. The functional relationship between x and y is unknown. We use the proposed algorithm to determine the exact functional relationship. The options of the parameters of the proposed algorithm are shown in Table 5.

During execution of the algorithm, we depicted the function curve presented by the AF specimen of each iteration in Fig. 5, where the AF specimen is the one who can best fit the given data in AF population. This figure shows that as the number of iterations increases, the evolving expression fit of the given data improves. Finally, the proposed algorithm found an optimal expression, the curve of which is also depicted in black in Fig. 5. As shown in the figure, the given data is fit perfectly, and the exact expression is as follows:

\[ y = \cos x \sqrt{\exp(\sin x + \exp(\log(x) + \exp(\log(\log(x)))) \cdot x)} \]

We also ran the proposed algorithm 50 times and depicted the average iterative curve in Fig. 6 in order to...
investigate its convergence. As shown in Fig. 6, the average curve quickly appears near the 0-error at approximately the 300th iteration and almost touches the 0-error at approximately the 800th iteration. Finally, the average relative error (fitness) is exactly 1.375%. This accuracy is satisfactory for most applications in science and industry. In other words, the proposed AFSA has a remarkable ability to converge to the optimal solution area of the solution space with high accuracy.

In order to investigate the robustness of the proposed algorithm, we did not randomly initialize the AF population, but instead initialized all of the AF individuals using the same expression, \( y = \ln x \). This is a very improper expression for fitting the given data shown in Table 4. Not only that, the diversity of the initial population is extremely poor. Under this pre-condition, we ran the proposed algorithm to determine an expression that can fit the given data well. Since this task is very difficult, we reset the MaxIteration parameter of the algorithm to 5,000 iterations, while leaving the rest of the parameters unchanged. As before, the function curves presented by the best AF individual of each iteration are depicted in Fig. 7.

The curve of \( y = \ln x \), which is used to initialize AF individuals, is shown in Fig. 7. At the first iteration, the curves of the entire population were the same, i.e., \( y = \ln x \). This is why the curve of \( y = \ln x \) appears in Fig. 7. Apart from this, we can also see that the fit of the evolving solution for the given data improves until finally coinciding with the given data, despite of the extremely poor diversity of the initial population. In addition, we also depicted the average iterative curve of 50 runs in Fig. 8. This figure shows that the average curve gradually approaches the 0-error and that the final average relative error (fitness) is 1.04%. Although the average precision is not extremely high, it is still high enough for many applications in science and industry. Moreover, it is essential to restate that this precision is obtained through an extremely difficult test, because the diversity of the initial population is extremely poor. Based on this investigation, we can state that the proposed AFSA has excellent robustness.

In the present study, a penalty-based fitness evaluation function is adopted because we consider the concision as a constraint for symbolic regression. For investigating the effectiveness of the constraint, we run the proposed AFSA with different values of parameter \( K \), which is the upper bound of the number of nodes of the parse tree represented by the AF individual. Apart from parameter \( K \), the other parameters of the proposed algorithm were left unchanged. (The values are the same as those in Table 5.) The obtained solutions for different constraints are shown in Table 6. As the value of parameter \( K \) decreases, the obtained expressions become increasingly compact. However, the precisions with which these expressions fit the given data are the same, namely, 0-error. This indicates that, owing to the adoption of the penalty-based fitness evaluation function, the proposed AFSA can effectively determine a compact solution expression. This performance is significant. A compact expression is always easier to understand and explain.

3.2 Comparison with GEP

The improved performance of GEP, as compared to GP, has resulted in GEP being the best known approach for symbolic regression. In order to justify the advantage of the proposed AFSA, as compared to GEP, we experimented on two algorithms with a symbolic regression problem from Ref. [9]. The given data are shown in Table 3 of Ref. [9].

We first used the proposed algorithm to find out the exact functional relationship. Here, the options of the
Table 6 Solutions for different constraints and different values of $K$.

| $K$ | Solutions |
|-----|-----------|
| 25  | $\sqrt{\frac{\sin x - \ln \left(\frac{x}{x+\pi}\right)}{\sqrt{e^{\sin x} + e^{\sin x}}}} \cdot \cos \left(\sqrt{\cos x - \cos x}\right)$ |
| 22  | $\sqrt{\frac{\sin x - \ln \left(\frac{x}{x+\pi}\right)}{\sqrt{e^{\sin x} + e^{\sin x}}}} \cdot \cos \left(\sqrt{\cos x - \cos x}\right)$ |
| 20  | $\sqrt{\frac{\sin x - \ln \left(\frac{x}{x+\pi}\right)}{\sqrt{e^{\sin x} + e^{\sin x}}}} \cdot \cos \left(\sqrt{\cos x - \cos x}\right)$ |
| 15  | $\sqrt{\frac{\sin x - \ln \left(\frac{x}{x+\pi}\right)}{\sqrt{e^{\sin x} + e^{\sin x}}}} \cdot \cos \left(\sqrt{\cos x - \cos x}\right)$ |

Table 7 Options of the parameters of the proposed algorithm for the comparison.

| Parameter          | Value |
|--------------------|-------|
| Population         | 20    |
| $2r$ (partners' number) | 6     |
| try_Number         | 3,000 |
| MinStep            | 3     |
| MaxStep            | 6     |
| MaxC               | 0.55  |
| MaxIteration       | 5,000 |
| $H$                | 30    |
| $p$                | 0.05  |
| $K$                | 35    |
| $x$, $0.811$, $0.618$, $\pi$, $1.112$, 2, 0.5 |
| $+$, $\times$, $\div$, $\ln$, $\exp$, $\sin$, $\cos$, $\sqrt{x}$, $\sqrt{\sin(x)}$, $\sqrt{\cos(x)}$, $\text{pow}(x, 2)$, $\text{abs}(x)$ |

Figure 9 shows the evolution process of the best expression, during execution of the algorithm. Based on this figure, we can see that the proposed algorithm finally obtained a solution that fits the given data perfectly after a series of evolutions. The obtained exact expression is as follows:

$$y = \ln(x^2) - \left(\frac{0.811}{e^0.618}\right)^2 \left(\sin \frac{1.112}{\pi} - \frac{x^2}{10^{0.618-0.811-0.811}}\right) - e^{2+1}$$

For comparison, we also show the exact expression obtained by GEP as follows [9]:

$$y = \left(\ln\left(10^{x+0.613}\right) \cdot \frac{x^2}{0.203}\right) + \left(e^{(x^2+0.811)}\right) + \left(10^{\sin(x)}\right).$$

Since Ref. [9] adopts a different fitness evaluation function from that adopted in this paper, we cannot compare the quality of the solutions obtained by the proposed AFSA and GEP by directly comparing the fitness of the solutions. Therefore, in order to compare the precision of the obtained solution with that obtained by GEP [9], we show the function curves of the two solutions in Fig. 10 (a), which is an overall comparison. As shown in the figure, it is almost impossible to distinguish which algorithm provides the better solution. However, if we zoom in, as shown in Figs. 10 (b), 10 (c), and 10 (d), comparison of the algorithms indicate that the proposed algorithm fits the given data better than GEP. This indicates that the proposed AFSA has a better ability to search for solutions with high quality.

Literature [16], by proposing a new representation scheme based on prefix notation that overcomes the original GEP’s drawbacks, presented an improved GEP, referred to as P-GEP, which has faster convergence with higher accuracy than the general GEP. In order to compare the convergence performance of our proposed algorithm with that of P-GEP, we also implemented P-GEP in C++. We tested both P-GEP and our proposed algorithm with a simple symbolic regression problem provided in Ref. [16]. The simple symbolic regression problem is shown in Eq. (6). A set of twenty-one fitness cases equally spaced along the x axis from $-10$ to $10$ were chosen for the test. The options of the parameters of the two algorithms for testing are shown in Table 8. Hereinto, except for very few parameters, the most ones for P-GEP are set with the same option as Ref. [16], and the most parameters for our proposed AFSA are set with
the same option as shown in Table 5.

\[ y = 3 \cdot (x + 1)^3 + 2 \cdot (x + 1)^2 + (x + 1) \tag{6} \]

In this test, since the fitness of a solution in P-GEP is defined as residual, instead of what we defined in this paper, we couldn’t intuitively compare the convergence accuracy by the iterative curves unless a common fitness function could be adopted for both P-GEP and the proposed AFSA. We thus used Eq. (2) to be the fitness function of both P-GEP and our proposed AFSA. And the reason why we didn’t use Eq. (4) here is that the representation scheme of the parse tree adopted in P-GEP doesn’t support our proposed counting algorithm based on Eq. (3). Obviously, such modification helps to intuitively compare the convergence accuracy by the iterative curves, but will not influence the searching ability of both P-GEP and our proposed AFSA at all. In addition, since the time complexity of our proposed AFSA is far different from that of P-GEP, the time consumption of the AFSA is accordingly different from that of P-GEP, even if we allow the both algorithms run the same iterations. Therefore, to be fair, the iterative curves with respect to CPU-time (s), instead of the number of iterations, were plotted. We respectively ran P-GEP and the proposed AFSA for 50 times for the above mentioned problem instance. Each run of each algorithm was allowed to last 5 seconds. The iterative curve of each run was depicted in Fig. 11.

As shown in Fig. 11, the most of the iterative curves of the proposed AFSA decline more quickly than those of P-GEP. This obviously indicates that the AFSA has a faster convergence speed, comparing with P-GEP. Beyond this,
from the perspective of the convergence accuracy, our proposed AFSA also performed a remarkable performance, which is obviously superior to P-GEP. From this figure, we can see that all the runs of our proposed AFSA finally achieved near-0-error within 5.0 seconds, while the most runs of P-GEP didn’t. In addition, the iterative curves of the 50 runs of the proposed AFSA are relatively more concentrated, while those of P-GEP are dispersed. This means that our proposed AFSA also possesses a better stability.

Time of evaluating fitness is a commonly used index for estimating evolutionary algorithms’ search efficiency. Since a candidate solution’s fitness will always be evaluated when it was searched by algorithms, the more times of evaluating fitness mean that the more points in solution space are searched. In other word, the larger region in solution space is explored by the algorithms. Obviously, an evolutionary algorithm with the searching mechanism of high-efficiency should be able to return a high-quality solution but only explore as smaller region in solution space as possible. In consideration of this, for further investigating our proposed AFSA’s optimization mechanism from this perspective, we also relatively depicted 50 iterative curves with respect to fitness evaluation times of our proposed AFSA and those of P-GEP in Fig. 12. These iterative curves are based on the same test as the previous one. The only difference is that the pausing condition of the tested algorithms for this comparison is the 200,000 times of evaluating fitness.

From Fig. 12, we can see that the proposed AFSA still performed better than P-GEP on the aspect of search efficiency. Within 200,000 times of evaluating fitness, most of the iterative curves of our proposed AFSA converged to a higher accuracy than those of P-GEP. This indicates that, although the proposed AFSA explored the same sized region in solution space as P-GEP, it can still return a more superior solution than P-GEP. In other word, preying behavior is the basis of convergence. This conclusion coincides with what described in Ref. [17], the doctoral thesis of X.L. Li, who firstly proposed the general AFSA. Apart from preying behavior, following behavior is another very important operator for the proposed algorithm. In other word, preying behavior is the basis of convergence. This conclusion coincides with what described in Ref. [17], the doctoral thesis of X.L. Li, who firstly proposed the general AFSA. Apart from preying behavior, following behavior is another very important operator for the proposed algorithm. In order to depict the average iterative curves for the cases with respect to CPU-time (s), we sampled the fitness of the best AF individual among the AF population every 0.005 s and the average iterative curves are shown in Fig. 13. To be a standard, the average iterative curve of 50 runs of the intact AFSA was also depicted.

As what we see from Fig. 13, due to the deletions, the performances of the algorithm respectively drop with various extents. Thereinto, the performance of the algorithm without preying behavior operator drops most, followed by that without following, avoiding, and randomly moving. This indicates that preying behavior is the most important operator for our proposed algorithm. In other word, preying behavior is the basis of convergence. This conclusion coincides with what described in Ref. [17], the doctoral thesis of X.L. Li, who firstly proposed the general AFSA. Apart from preying behavior, following behavior is another very important operator for the proposed algorithm, in consideration of the poor performance of the algorithm without following behavior. Although preying behavior is the basis of convergence, it is only local search. When implementing preying, AF individuals only move in their own visual field. Instead, implementing following behavior makes it possible to spread superior pattern of genotype from one individual to others. Obviously, such a mechanism makes
our proposed algorithm possess global search ability. As for avoiding behavior and randomly moving behavior, they are also indispensable components for our proposed algorithm, although they seem not as important as preying and following. From Fig. 13, it is not difficult to see that, no matter avoiding behavior or randomly moving behavior, each of them really helps to improve the performance of our proposed algorithm.

4. Discussion and Analysis

4.1 Option of Parameters

There are several important parameters in the proposed AFSA, and the performance of the proposed algorithm depends on the rational option of the parameters. Therefore, it is necessary to discuss how to reasonably set the parameters in order to obtain the maximal performance. Until now, to the best of our knowledge, there is no consensus on how to properly set the parameters of the AFSA. Moreover, some concepts of the proposed algorithm are completely different from the general AFSA. Therefore, we have to set parameters according to our experience and the property of the problem to be solved.

(1) **Population** – The AFSA is a swarm-based algorithm. In order to ensure that AF individuals can collaboratively search the solution space, a certain scale population is essential. Usually, we set the Population according to the difficulty of the problem. A large Population can be used to solve difficult problems, whereas a small Population can be used to solve easy problems. A rational range for Population is between 20 and 100.

(2) **try.Number** – This parameter is the number of times that an AF individual randomly tries a new position in its own visual field. A larger try.Number allows an AF individual to more fully explore a new position in its own visual field, as compared with a smaller try.Number. However, a larger try.Number is more time consuming. According to the property of the problem, we should set an appropriate value, such that we could balance well the time consumption and intensity of exploration.

(3) **MinStep** and **MaxStep** – These two parameters are, respectively, the lower bound and the upper bound of random steps that an AF individual moves. According to our optimization experience, a shorter step-length usually causes an AF individual to obtain a solution with high precision, whereas a longer step-length causes the algorithm to converge quickly but prematurely. Therefore, we tend to allow AF individual moves with relatively shorter random step-length. That is why we respectively set MinStep = 3 and MaxStep = 6 so that the random step-length will be relatively shorter. Of course, the interval is not completely absolute and depends on the problem.

(4) **2r (number of partners)** and **MaxC** – The number of partners, 2r, can be set to be approximately 1\( \frac{1}{3} \) of Population. In addition, the rational value of MaxC can be set in a relatively wide range of from approximately 0.4 to approximately 0.7.

(5) Other parameters (\( H, p, K, \) etc.) – Freely set the values of these parameters according to the requested precision and concise degree of the conceivable expression.

As discussed above, generally speaking, the proposed AFSA is not very sensitive to the parameters. Therefore, we can set the values of these parameters relatively freely in a relatively wide range, according to the properties of the problem.

4.2 Analysis of Behaviors

In AFSA, each AF individual is an agent that searches the solution space via various behaviors, which are various search patterns. Randomly moving behavior is the simplest search pattern and is a completely random search in the local space of the current AF individual. The abovementioned local space is a set of solutions that have the same parse tree structure as the current AF individual. Therefore, randomly moving is similar to the mutation operator in GA. Preying behavior is actually a type of local search via repeatedly and randomly trying a new point in local space. It is not difficult to see that both preying and randomly moving are individual behaviors, rather than group behaviors. When performing both preying and randomly moving behaviors, an AF individual does not share and use the experience of its partners, instead explores on its own. The result of exploring in the way of working on one’s own depends on fortune. Therefore, individual behaviors are inefficient. In addition, neither preying nor randomly moving can change the parse tree structure represented by an AF individual. In other words, either behavior is simply a local search pattern. However, even with what we talked above, these behaviors are still very important for the proposed algorithm, because these behaviors are the foundation of the optimization. Based on searching via individual behaviors, AF individuals share and use each others’ optimization experience to tremendously enhance the searching efficiency through group behaviors, which are, respectively, following behavior and avoiding behavior. Thus, an AF individual learns optimization experience from the best individual among its partners and uses the obtained optimization experience to improve itself through following behavior. The optimization experience is actually the bits chosen from the superior partner, according to the description of the following behavior (Sect. 2.4.3). First, because of directly and conveniently obtaining the beneficial experience of the superior partner, rather than randomly exploring on its own, the search efficiency is significantly increased. Moreover, the obtained experience also allows an AF individual to move from a local space into another space because the obtained new bits may change the parse tree structure represented by the individual who performs following behavior. According to the above two features, following behavior is a very important search pattern for exploiting the solution space. As for avoiding behavior, when an AF individual performs such a behavior in order to avoid overcrowding, various new parse tree structures may be created. This is beneficial to maintain the diversity of parse tree
structures throughout the population. In avoiding behavior, the experiences of partners are also used, not for imitating, but for innovation. Based on the above considerations, it is not difficult to see that following behavior and avoiding behavior severally reflect two methods of cooperation among AF individuals.

Through a behavioral strategy, the four behaviors are organized in order to form a good optimization mechanism, which maintains a good balance between exploring and exploiting the solution space.

5. Conclusion

In this present study, we described an implementation for solving symbolic regression problems within an artificial fish swarm algorithm (AFSA). Since the gene expression scheme in GEP is a brilliant approach for encoding a parse tree, we adopted this scheme as the encoding scheme to represent AF individuals. Unlike other studies on symbolic regression, we considered both the accuracy and concision of the solution. In this paper, the proposed implementation method is such that we designed a penalty-based fitness evaluation function that considers the number of nodes of the parse tree represented by an AF individual as a constraint. Under such a constraint, the obtained solution does meet the demands of both accuracy and concision. Before describing the proposed algorithm, we first defined several important concepts and then described them in detail. Based on the defined conceptions, we defined four behaviors and presented a formalized description of these behaviors, which are randomly moving behavior, preying behavior, following behavior, and avoiding behavior. Through the proposed behavior strategy, the four behaviors are organized together to form a well-run optimization framework. In order to investigate the performance of the proposed algorithm with respect to various aspects, we tested the proposed algorithm through exhaustive simulation experiments. The detailed results obtained in the simulation experiments indicate that the proposed algorithm can obtain a higher-precision solution with quicker convergence, as compared with GEPs. Beyond this, the proposed algorithm also exhibited remarkable robustness. Since several parameters are adopted in the proposed algorithm, which may significantly influence the performance of the algorithm, we also discussed how to set the parameters relatively reasonably in order to increase the performance as much as possible. In addition, we simply analyzed the mechanisms of the four behaviors.

Research on the use of the AFSA proposed in this paper for solving symbolic regression has opened up a new route for symbolic regression problems and broke new ground in the application of the AFSA. In the future, research on using the AFSA to solve symbolic regression will focus primarily on the followings: (1) proposing a better representation scheme for encoding a parse tree that is different from the adopted gene expression scheme, (2) designing other behaviors to enrich the optimization mechanism, (3) combining other superior heuristic algorithms (such as the simulated annealing algorithm) with the proposed algorithm to form a hybrid algorithm for improving its local search ability, so that the performance can be further enhanced, and (4) investigating a method by which to avoid overfit when the given data are polluted by noise.

References

[1] D.A. Freedman, Statistical Models: Theory and Practice, Cambridge University Press, 2005.
[2] J. Koza, Genetic programming: On the programming of computers by means of natural selection, MIT Press, 1992.
[3] M. Schmidt and H. Lipson, “Learning noise,” GECCO’07, pp.1680–1685, 2007.
[4] P. Barmopalexis, K. Kachrimanis, A. Tsakonas, and E. Georgarakis, “Symbolic regression via genetic programming in the optimization of a controlled release pharmaceutical formulation,” vol.107, pp.75–82, 2011.
[5] E. Hemberg, M. O’Neill, L. Ho, and H. Claussen, “A symbolic regression approach to manage femtocell coverage using grammatical genetic programming,” CECCO’11, pp.639–646, 2011.
[6] B. Dolin, F.H. Bennett, and E.G. Rieffel, “Co-evolving an effective fitness sample: Experiments in symbolic regression and distributed robot control,” SAC2002, pp.553–559, 2002.
[7] T. Soule, J.A. Foster, and J. Dickinson, Code Growth in Genetic Programming, MIT Press, Cambridge, MA, 1996.
[8] C. Ferreira, “Gene expression programming: A new adaptive algorithm for solving problems,” Complex Systems, vol.13, issue2, pp.87–129, 2001.
[9] C. Ferreira, “Gene expression programming in problem solving,” WSC6 tutorial, 2001.
[10] X.L. Li, Z.J. Shao, and J.X. Qian, “An optimizing method based on autonomous animats: Fish-swarm algorithm,” Systems Engineering - Theory & Practice, vol.22, no.11, pp.32–38, 2002.
[11] C.W. Reynolds, “Flocks, herds and schools: A distributed behavioral model,” Computer Graphics (ACM SIGGRAPH’87 conference proceedings), vol.21, no.4, pp.25–34, 1987.
[12] Y.L. Zhang, H. Ogura, J. Kuroiwa, and T. Odaka, “Fish swarm optimization method for the two-dimensional guillotine cutting problem,” J. Signal Processing, vol.15, no.3, pp.225–234, 2011.
[13] G. Huang, H. Zhu, and J. Zhou, “An optimization method of multistage stations locating in oil transportation based on fish-swarm algorithm,” Systems Engineering-Theory & Practice, vol.28, no.3, pp.94–101, 2008.
[14] X. Ma and Q. Liu, “Artificial fish swarm algorithm for multiple knapsack problem,” J. Computer Applications, vol.30, no.2, pp.469–471, 2010.
[15] X. Ma and Q. Liu, “An artificial fish swarm algorithm for Steiner tree problem,” IEEE International Conference on Fuzzy Systems, pp.59–63, 2009.
[16] X. Li, C. Zhou, W.M. Xiao, and P.C. Nelson, “Prefix gene expression programming,” Genetic and Evolutionary Computation Conference (GECCO’05), Washington, DC, USA, June 2005.
[17] X.L. Li, A new intelligent optimization method — artificial fish school algorithm, ZheJiang University, China, 2003.
Qing Liu received his B.E. degree from Xi’an University of Technology, China, in 2006. In 2010, he received his M.E. degree from the same university. Now, he is a doctoral student of the Department of Nuclear Power and Energy Safety Engineering at the Graduate School of Engineering, University of Fukui, Japan. His main interests and research activities focus on intelligence optimization algorithms’ optimizing mechanisms and their applications.

Tomohiro Odaka received a Ph.D. degree in Engineering from Waseda University, Japan, in 1990. He is professor of the Graduate Program of Nuclear Power and Energy Safety Engineering at the Graduate School of Engineering, University of Fukui Graduate School, Japan. He is a member of IEEE CS, ACM, IPSJ, JSME, and AESJ. His research activities focus on user modeling, network security, biomedical signal processing, intelligent human interface, and computer aided instruction.

Jousuke Kuroiwa received a Dr. Eng. degree from Tohoku University, Sendai, Japan, in 1996. He joined the Department of Human and Artificial Intelligent Systems, Fukui University, Fukui, Japan, where he is now an Associate Professor and engaged in research on nonlinear dynamics and dynamical neural computations. Dr. Kuroiwa is a member of IPSJ, JNNS, PSJ and RISPJ.

Hisakazu Ogura received his D.Sc. degree from Kyoto University, Japan, in 1977. He was retired from the University of Fukui, Japan, and now is a guest professor of the university. His main interests and research activities are in knowledge representation and processing in the fields of artificial intelligence, medical informatics and medical image processing by applying genetic algorithms, artificial neural networks, symbol processing or fuzzy set theory. He is a member of IPS Japan and SOFT Japan.