An Adaptive Knowledge Evolution Strategy for Finding Best Solutions of Specific Problems

Yo-Ping Huang[1], Yueh-Tsun Chang[2], Shang-Lin Hsieh[2], and Frode Eika Sandnes[3]

[1]Dept. of Electrical Engineering, National Taipei University of Technology
Taipei, Taiwan 10608
yphuang@ntut.edu.tw
[2]Dept. of Computer Science and Engineering, Tatung University
Taipei, Taiwan 10451
tennno@gmail.com; slhsieh@ttu.edu.tw
[3]Faculty of Engineering, Oslo University College
Oslo, Norway
frodes@hio.no

Abstract-- Most real-world problems cannot be mathematically defined and/or structured modularly for peer researchers in the same community to facilitate their work. This is partially because there are no concrete defined methods that can help researchers clearly describe their problems and partially because one method fits one problem but does not apply to others. In order to apply someone's research results to new domains and for researchers to collaborate with each other more efficiently, a well-defined architecture with self-adaptive evolution strategies is proposed. It can automatically find the best solutions from existing knowledge and previous research experiences. The proposed architecture is object-oriented that in turn become foundations of the community interaction evolution strategy and knowledge sharing mechanism. They make up an autonomous evolution mechanism using a progressive learning strategy and a common knowledge packaging definition. The architecture defines fourteen highly modular classes that allow users to enhance collaboration with others in the same or similar research community. The presented evolution strategies also integrate the merits of users' predefined algorithms, group interaction and learning theory to approach the best solutions of specific problems. Finally, resource limitation problems are tackled to verify both the re-usability and flexibility of the proposed work. Our results show that even without using any specific tuning of the problems, optimal or near-optimal solutions are feasible.

Keywords: Self Adaptation, Evolution Computation, Knowledge Sharing, Learning Strategy, Community Interaction.

1. Introduction

In recent year, many research results have been reported in the field of artificial intelligence. Most of these are inspired by natural ecological system, such as ant colony systems [1]. Some researchers use max-min ant system to enhance classification, reduce completion time and the use of computing resources [2-3]. Ecosystems are also the inspiration behind artificial immune system and cellular automata. Recent articles describe compact classifier system using simple artificial immune system [4] to model the simulation of computer network for evaluating connectivity and system reliability and computing the shortest package routing path [5].

Ecosystems are also the original inspiration behind genetic algorithm which was first

described by Prof. John Holland in 1975. Many researchers are continually enhancing the performance of genetic algorithm [6-12] and developing new applications [13]. Moreover, ecosystem ideas lead Dr. Eberhart and Dr. Kennedy to develop particle swarm optimization (PSO) in 1995, which is inspired by social behavior of bird flocking. PSO has been applied to many applications [14-17] and the architecture of PSO allows researchers to enhance the performance using different domain knowledge [18-20]. Inspired by biological evolution, genetic programming is an evolutionary methodology for generating computer programs that solve certain user-defined tasks [21-23], and many enhancements to the basic genetic programming paradigm have been proposed [24-28]. Based on the concept of adaptation and evolution, evolution strategy (ES) was proposed in the early 1960s and 1970s. Although ES has been around for some time, the application and enhancement are still being proposed [29-32].

Moreover, learning classifier systems that were first described by Prof. John Holland consist of computing rules which are composed of binary, real-valued, neural network, and other representations. Recently, researchers have proposed an enhancement using self-organized map (SOM) [33] and neural network [34] on supervised learning classifier systems (UCS). UCS is also used for the classification problem in data mining [35].

There is a vast body of research being published every year, which means that the research results in different kinds of fields are growing. Although Internet technology has shortened the distance between people, the amount of information that becomes available is growing. People only have limited time to absorb information. For example, there are various subjects in artificial intelligence, such as neural network, genetic algorithm, data mining and gray system, etc. For each of these categories a vast body of knowledge is developed for solving different problems.

Current media technologies are unable to hide the information chaos, and the expanding information spreads unsystematically. Consequently, a gap emerges between research groups which prevent interactions. As a result, many experiences, results and algorithms will be lost or redundantly developed.

Researchers usually search for existing solutions when confronted by a new problem. During the search, a solution is invented by knowledge combination with different process orders. Time and rigorous methods are needed to discover the appropriate solution, and researchers can also add new algorithms to the combination for gaining better solution. However, the progress is not easily shared with others because of the information gap. In general, it is often difficult to learn from previous mistakes and experiences to avoid repeating the same mistakes.

Most results are simply improvements upon previous research. The improvements include adding, deleting, reordering or modifying steps and algorithms of previous studies. The improvement depends on which algorithms, modules or algorithms are selected.

Past research provides the foundations for new inventions. Research is more valuable if it is understandable and reusable. Researchers can discover better solutions by combining past knowledge. Past records can help researchers avoid mistakes conducted in previous research.

The probability of obtaining better solutions rises when the previous research progresses and results are better understood. This is the reason for acquiring knowledge and discussing problems with others. Irrespective of the types of knowledge, they always have chances to provide brilliant ideas for solutions. In order to gather the required knowledge, it is crucial that researchers interact.

Researchers are restricted by limited time and space to find the optimal combination from a seemingly unlimited number of sources. It is also impossible to know if the combination is suitable for other problems or not.

Based on the principles of object oriented programming (OOP), a common format can be

defined to package knowledge as basic components representing best solutions. Computers can then be used to assist researchers finding new solutions to problems.

Even if two algorithms have similar architectures, they could use different knowledge. Take neural networks and genetic algorithms for example. Neural networks can be divided into three layers including input, hidden and output layers. The output layer sends unsupervised or supervised feedback back to the hidden layer for self-adaption. On the other hand, genetic algorithms can also be divided into four operations including initialization, mutation, crossover and selection operations. The selection and initialization also have a self-adaptive mechanism using performance filters and a competition strategy to generate new solutions. Clearly, both evolution algorithms present similar compositions using self-adaption feedback. In many studies, researchers also applied these two algorithms to various problems together for developing better solutions.

The operations of the proposed strategy emulate the crossover operation of genetic algorithm. The operations use several evolution steps with independent modules to find better solutions. The highly modular knowledge definition allows researchers to both improve the final solutions using proposed evolution strategy and also efficiently share their research experiences with others.

The proposed evolution strategy provides several means of knowledge exchange. Central to the strategy is the ability to find better knowledge combinations and solutions to various problems based on huge banks of knowledge using computing power. The basic concept of the proposed strategy and architecture is similar to independent research. Normally, researchers obtain independent research results or find reliable solutions and ideas by themselves before discussing, competing or cooperating with other researchers. In the proposed architecture, researchers from different regions can cooperate using the highly modular components.

This paper is organized as follows. The definitions and the characteristics of the modules of the proposed strategy and architecture are introduced in Section 2. Three evolution strategies are presented for the proposed architecture. The components of the architecture which are called *Community Knowledge Modules* (CKM) are presented in Section 3. Section 4 demonstrates the usability of the proposed strategy and architecture using the parameters optimization problem with two different functions as examples. Finally, future work is discussed in the conclusion section.

2. The Architecture of Community Knowledge Evolution

People usually learn from others in class, through discussions and by reading books. The proposed strategy mimics this human learning mechanism through 14 modules that are the evolution operating components.

2.1 Architecture

The proposed architecture is called the *Community Knowledge Modules* (CKM). These modules are used by the *Community Interaction Strategy* (CIS) for finding the optimal solution to problems. CKM has two primary parts, namely the *Domain Definition* (DD) and the *Evolution Modules* (EM). These are used for defining problems and evolution operations, respectively. The CKM both abstracts the evolution progress and avoids the rewriting of similar programs for solving different data formats. The CKM forces the user to focus on the algorithm rather than a particular data format.

2.2 Domain Definition

Irrespective of the problem that the researcher wants to solve, the researcher has to

determine the basic data format to be used in the program. The DD provides several common formats suitable for solving different classes of problems. The format employs object oriented ideas where the user has to override and to inherit the defined interfaces and abstract classes to get access to the basic operation classes which are used by the CIS.

During the evolution, the CIS uses the defined interfaces to operate on the real data which are packaged in the basic operational class. The user must provide operations for translating real data into the defined format which follows the defined interfaces.

The *Domain Definition* has two parts, namely the *Case Definition* (CD) and the *Knowledge Definition* (KD). The CD defines five input data package definitions which represent different types of research problems. These definitions include *Domain*, *Topic*, *Case*, *Unit Set* and *Particle* classes.

The *Knowledge Definition* contains a group of interfaces which satisfies the needs of different knowledge domains. Users can exploit the standard KD to implement *Knowledge* classes, such as, mutation and crossover operation of GA, back-propagation neural network or any algorithm for specific needs. For different missions, the modules can be divided into two types of modules, *Generator type* and *Common type* modules. *Generator type* modules generate initial solutions for *Common type* modules to process.

After the required modules are implemented, the modules are managed by the *Knowledge Map* module for storing and managing the modules. The map preserves the entity of modules and manages the connection between relational *Knowledge* modules. The connection is decided by the module creator. Before the module is implemented, the creator has to decide the relation of the implementing and the published modules.

Before the evolution progress is run, the *Knowledge Map* creates a *Start Point Knowledge* module which is designed as the entry to the progress. The *Start Point Knowledge* module can only be hooked to *Generator type* modules. *Common type* modules can hook to multiple modules except from *Start Point Knowledge* modules. The final processing order will be recommended by *Memes* which memorizes the process histories and results as a human brain. The details of the evolution process are discussed in Section 3.

2.3 Evolution Modules

Evolution Modules contain a group of modules which are designed for the proposed strategy. These modules are *Community, Member Group, Knowledge Map, Member, Memes, Solution, Evolution Path* and *Particle Unit*. These modules provide various operations and running histories for the evolution process. For different problems, the restrictions and the data format of the specific case are provided by the *Case* classes which are created by the users. Some operations are supported by the implemented CD classes. Since real data are packaged into the *Particle* module, the EM can handle different data formats rather than having to rewrite the experiment programs.

2.4 The Relation between Domain Definition and Evolution Modules

The relation between the EM and DD modules is similar to the relation of operating components and problem definition. These modules support the initial conception for providing the idea of *Community Interaction Evolution* (CIE). As shown in Fig. 2.1, a research community usually focuses on a specific research domain. A community also contains independent member groups which focus on one or multiple topics of the research domain. A topic also is studied by one or multiple member groups.

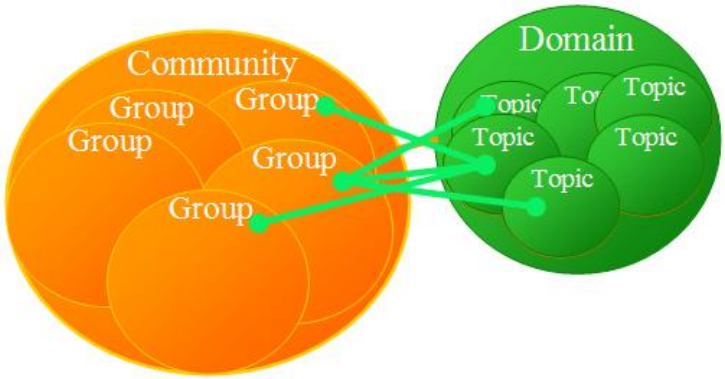

Fig. 2.1. The relation between the community module and the domain definition.

The role of the *Community* module is to serve as an association which has many research groups. Each group in the *Community* module has their specific research interests. The *Community* module provides both operating interfaces and controls the life cycle of the *Member Group*s entities and the *Community Interaction* operation. The *Member Group* module can be seen as a research group who focuses on a specific case. The *Member Group* module uses the *Knowledge Map* module to manage the implemented *Knowledge* entities.

After the *Member Evolution has* finished, the *Group Interaction* operation is launched. In the *Group Interaction* operation, the *Member* entities of the same *Member Group* entity exchange their personal best solutions with each other to find the best solution of the *Member Group* entity.

The evolution which is evolved inside the *Member* entity is called *Member Evolution*. This is similar to the situation where researchers independently investigate the problem before discussing it with others. Although the investigations are independent, researchers usually have the same elementary knowledge of the research field. Each *Member* entity of the *Member Group* entity can retrieve suitable *Knowledge* entities from the *Knowledge Map* entity of the *Member Group* entity and then, launch the *Member Evolution* operation for producing solutions. These solutions are stored in the *Memes* entity of the *Member* entity. The relations of *Case*, *Knowledge Map*, *Member* and *Memes* are shown in Fig. 2.2.

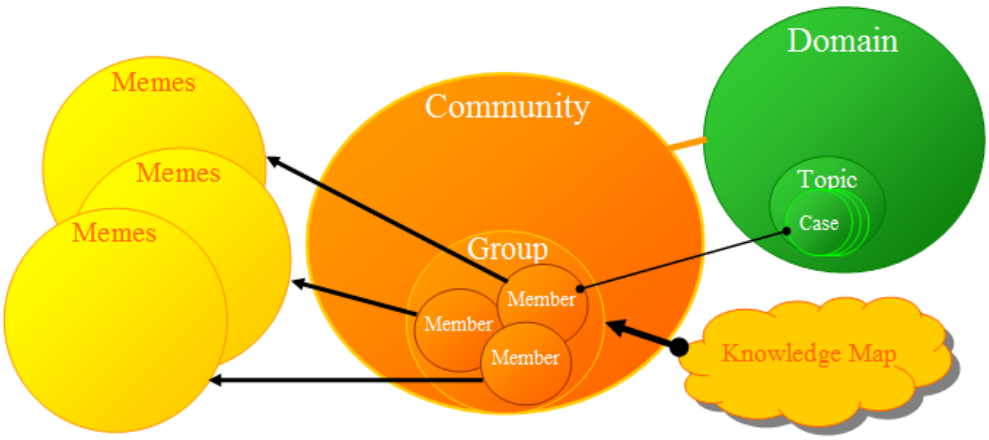

Fig. 2.2. The relations of Case, Knowledge Map, Member and Memes.

As shown in Fig. 2.3, each *Member* module has only one *Memes* module. The *Memes* module is both responsible for recommending appropriate *Knowledge* entity and *Activity* operations, and store *Solution* entities which record evolution history and the personal best solution of the *Member* entity. In order to store the process history and the personal best solutions, the *Solution* module is composed of two modules, namely the *Evolution Path* and the *Unit Set* modules.

The *Evolution Path* module records the global unique identification (GUID) and the processing order of operated *Knowledge* entities which have been used during the *Member Evolution*. The *Unit Set* module contains the *Particle Unit* module which represents the basic operating unit in the *Member Evolution*. Each *Particle Unit* module has one to multiple *Particle* modules. The relation between the *Particle Unit* and the *Particle* modules is very similar to the coordinates and its values. The user has to implement the *Particle* module following the definition of the CD to provide the basic operating unit. According to the need of the processing problem, the user must implement the appropriate *Particle* module to package the data, and then, use the *Particle Unit* entity to manage *Particle* entities.

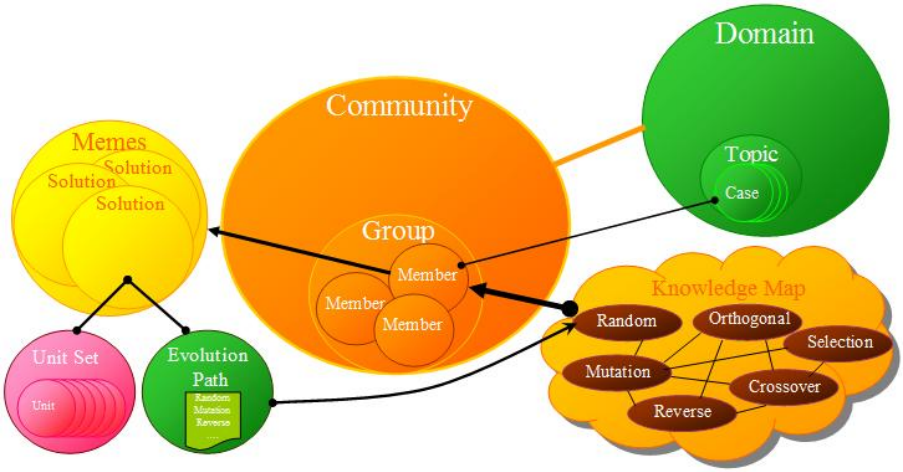

Fig. 2.3. The relation of evolution modules.

3. The Implementation and Evolution Strategy of CKE

Before the modules are implemented, the creator should understand the properties and restrictions of the processing problem. The creator overrides, inherits or implements the modules of *Domain Definition*.

3.1 The Implementation of Domain Definition Modules

The creator must inherit and implement the *IDomain*, *ITopic* and *ICase* interfaces respectively. The *IDomain* interface definition defines basic information and the domain definition, and manages all registered *Topic* modules which are implemented from *ITopic*. Moreover, the *ITopic* interface contains the definition of basic information, and responses for the management of all registered *Case* entities which follows the *ICase* interface. The *ICase* interface contains definitions of solution region setting and constraints.

The definitions include the configurations of basic information, solution region, optimal and worst results, the length of *Unit Set*, the composition of *Particle Unit* and the accuracy of final results. The user can inherits original interfaces or add new functions for the specific case. After the *Case* module has been implemented, *ICase* endows with *Evolution Modules* the ability to create *Solution*, *Unit Set*, *Particle Unit* and *Particle* entities.

Except for the *ICase* interface, the user must inherit *IUnitSet* and *IParticle* too. Because problems use different data formats and compositions, user has to provide appropriate data format and composition for dealing with the problem through the implementation of *IUnitSet* and *IParticle* interfaces.

As shown in Fig. 3.1 the left part presents the modules which must be inherited or overridden, the right part are the example of implemented entities. The example in Fig. 3.1 shows that three sensors need to be allocated the position in 2D coordinates. At the beginning, the user must use *Unit Set* module to form the composition of the three sensors respectively,

and then, package up the location of the sensor using the *Particle Unit* module which holds two *Particle* modules to represent the coordinates of position X and Y, respectively.

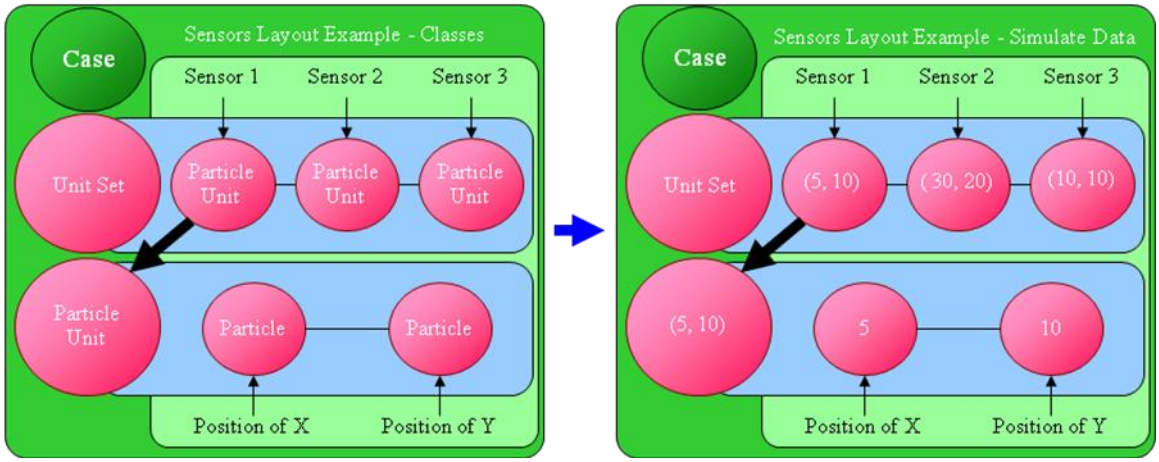

Fig. 3.1. The relations between Unit Set, Particle Unit and Particle modules.

For the sake of calculation, comparison and evaluation operations, the *CIS* has to know the discrimination of different solutions. Therefore, the interface uses the "*double*" format to represent the performance score, because "*double*" can be translated into most other types of data formats.

Beside of the *Case Definition* described above, the *Knowledge Map* module is responsible for providing the *Knowledge* module during the *Member Evolution*. The user can follows the *IKnowledge* interface to implement new *Knowledge* modules and adds the module into a *Knowledge Map* entity.

There are five interfaces defined in the *IKnowledge* definition which are the interfaces for starting, executing and hooking *Knowledge* module, the interface of providing *Knowledge Map* to gather the children *Knowledge* and the interface of storing the processed *Knowledge* information into the *Evolution Path* entity. The user can implement the *IKnowledge* interface to create new *Knowledge* modules and use the existing *Knowledge* modules. If user wants, the new module can be used by *CIS*, the module needs to set the relations with other implemented *Knowledge* modules and registers itself into the *Knowledge Map* entity. Only after the relations have been built, the *CIS* will be able to retrieve the appropriate entity from the *Knowledge Map* entity during the evolution.

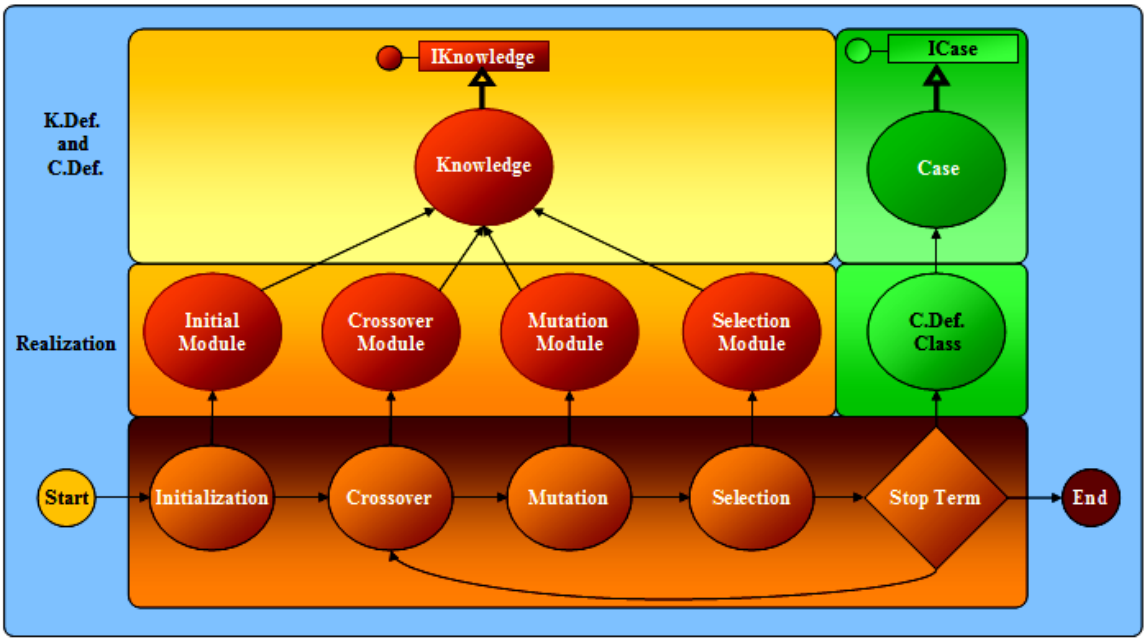

Fig. 3.2. The operations of genetic algorithm to knowledge definition.

Fig. 3.2 provides an example that transforms the conventional operations of genetic algorithm (GA) into *Knowledge Definition* modules. At the bottom of the figure are the GA operations which include Initialization, Crossover, Mutation, Selection operations and the Stop Term validation. Because the first four operations represent different evolution strategies, the user can transform these operations into *Knowledge* modules respectively. However, the Stop Term validation is an examining method rather than an evolution strategy. The validation should be transformed into a *Case* module as an evaluation operation of the processing problem.

3.2 The Evolution Strategy of the CKE

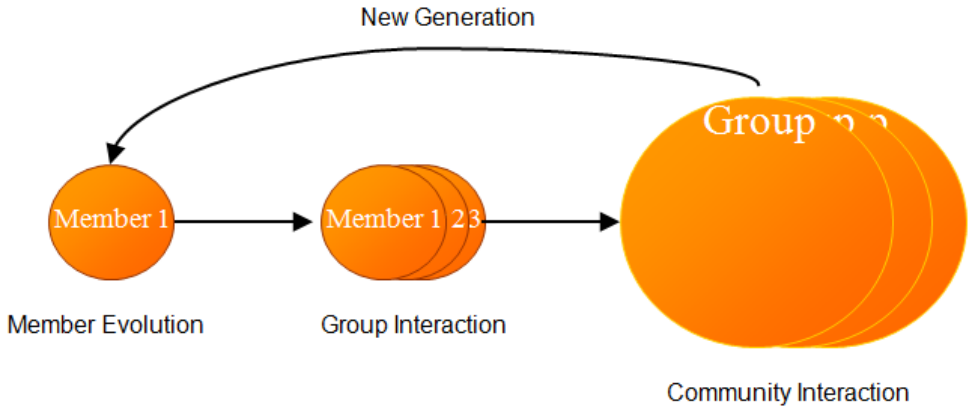

Fig. 3.3. The evolution strategy of community knowledge evolution.

In the *Community Interaction Strategy*, the evolution strategy has been divided into three steps, namely *Member Evolution*, *Group Interaction* and *Community Interaction*. The *Member Evolution* step involves self-adaptive evolution using the *Knowledge* modules. The *Member* entity uses the supported *Knowledge* entities which are recommended by its *Memes* entity to evolve new solutions. Every *Member* entity of the same *Group* entity generates independent solutions to the same problem. After each *Member* of every *Group* entities entity

finished *Member Evolution*, *Group Interaction* will use the best *Solution* entity of each *Member* entity to evolve better solutions.

As the *Member* entity uses the *Knowledge* entity, *Group* and *Community Interaction*s use four *Activity* operations to generate better solutions. The *Activity* operations are applied to exchange the best solutions of *Member* entities which are produced from *Member Evolution*. In order to preserve the individualism and avoid local optimal, *CIS* simulates human learning interactive behaviors. *Member* entities computes local optimal independently during *Member Evolution*, and then, preserves a suitable diversity for final optimal solutions in *Group Interaction* and *Community Interaction*. The working flow is shown in Fig. 3.3.

3.3 Member Evolution using Knowledge Modules

Fig. 3.4 shows the workflow of the *Member Evolution* operation. The operation starts from the registration of *Topic* and *Case* entities. After the *Member* entity loads the processing *Case* entity, the *Memes* entity provides the recommended *Knowledge* entities. When the *Knowledge Map* entity decides to stop the evolution, one or more *Solution* entities are generated. Totally, each *Member* entity has at least three *Solution* entities. When all of the *Solution* entities are generated, the *Member* entity ranks all *Solution* entities into *Best*, *Normal* and *Worse* categories. The *CIS* uses *Best* and *Worse Solution* entities in the operations of comparison and convergence verification and preserves the diversity of solutions using *Normal Solution* entities. After *CIS* has finished the *Member Evolution* operation of all the *Member* entities of all *Group* entities which process the same problem, the *Group Interaction* operation will be launched to find the group-level optimal solutions.

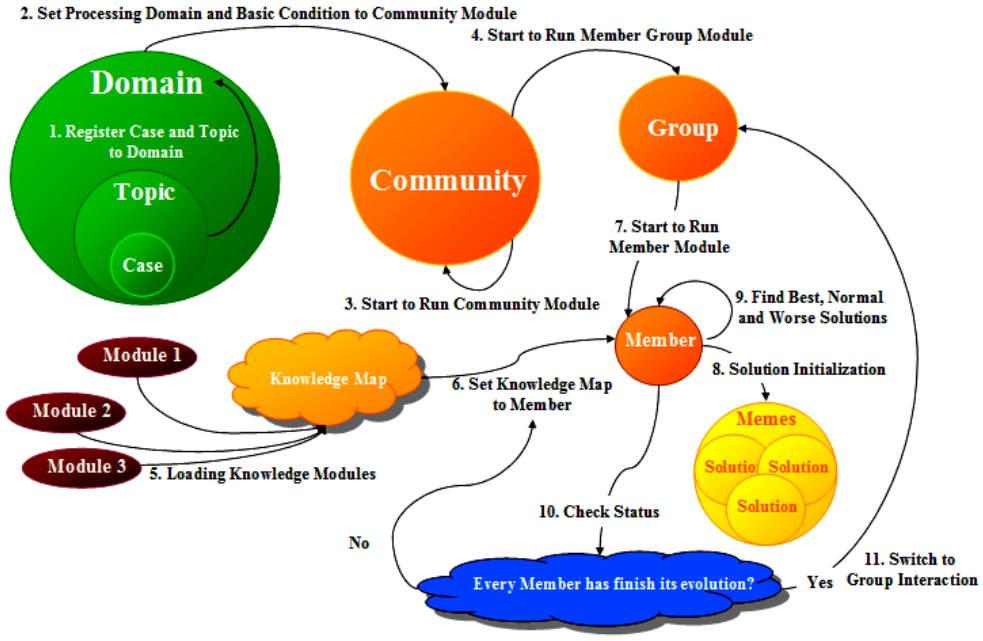

Fig. 3.4. The workflow of the member evolution operation.

3.4 Knowledge Map

The circle key point of the Member Evolution is the *Knowledge Map* module which is responsible for the management of *Knowledge* entities. The *Knowledge Map* entity loads and registers all *Knowledge* entities which are listed in the configuration setting. After all the entities are loaded, the *Knowledge Map* creates the *Start Point Knowledge* entity as the entry of the evolution. After the entity is created, the *Knowledge Map* entity continually loads *Registered Knowledge* entities and builds the relations to other *Registered Knowledge* entities according to the pre-decided configuration of loaded *Knowledge* entity. After all of the

Registered Knowledge entities are loaded and the relations are built, the *Knowledge Map* entity is created.

After the *Knowledge Map* entity is created, the *Member* entity is able to execute the selected *Knowledge* entities which are recommended by the *Memes* entity based on the needs for processing the problem and the running history of processed *Knowledge* entities.

In the recommendation procedure shown in Fig. 3.4, the *Knowledge Map* entity delivers the processing *Case* entity and the previously processed *Knowledge* entity to *Member* entity's *Memes* entity. The *Memes* will randomly select a *Knowledge* entity from the relation list of the processed *Knowledge* entity when the running history has not any records of the processing *Case* and the processed *Knowledge*. Otherwise, *Memes* will select an appropriate *Knowledge* entity as the recommended *Knowledge* entity according to the *Reference Probability* which is generated from the processed history. For example, two processed records of *Knowledge* entity A and B are 40 and 60, the selected probability of the entities are 0.4 and 0.6, respectively.

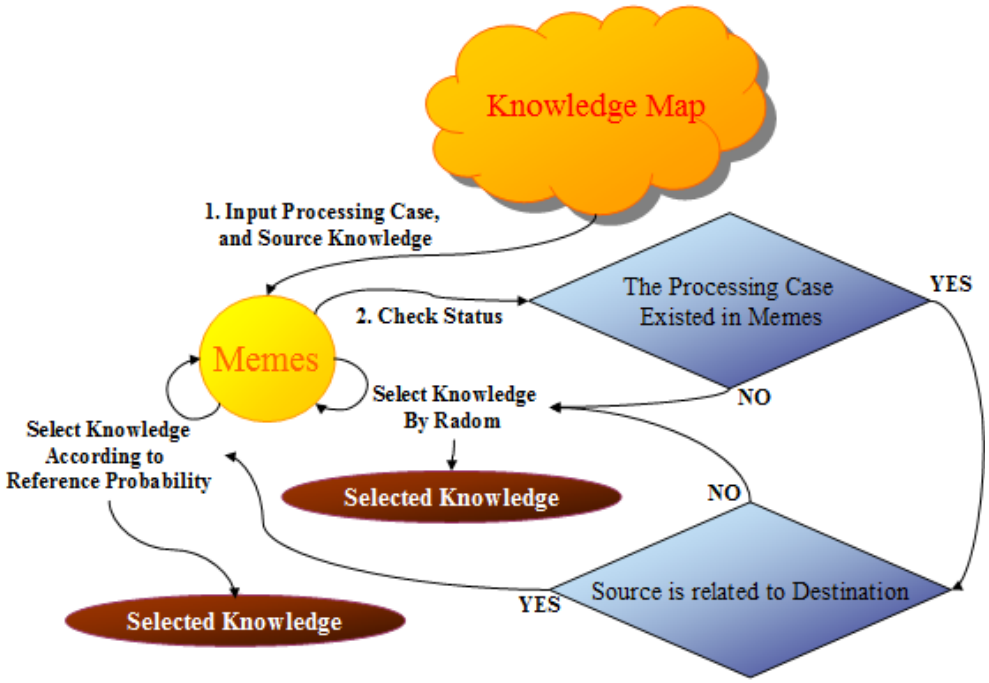

Fig. 3.5. The workflow for getting recommended knowledge entities.

After the *Knowledge Map* entity gets the recommended *Knowledge* entity, the *Knowledge Map* entity will deliver the *Solution* entity to the recommended *Knowledge* entity and launch the recommended *Knowledge* entity procedure. When the procedure has finished, the *Memes* entity will update the running history of the recommended *Knowledge* and switch to other status after the updating. If the processed *Knowledge* entity is recommended by random selection, the *Memes* entity will check the *Stopping Threshold*. If the *Knowledge* is not selected by random, the *Memes* will load the relation list from the recommended *Knowledge* to retrieve the next recommendation. The *Memes* uses the processed result from the recommended *Knowledge* to generate the *Stopping Threshold*. When the result approaches the optimal solution, the *Stopping Threshold* is flagged. For example, when the optimal solution is 100 and the process result is 90 then the *Stopping Threshold* will be ten. After the *Stopping Threshold* is generated, the *Knowledge Map* entity generates a random value in the interval 0 to 100. If the random value is less than ten, the *Memes* continues to find the next recommended *Knowledge* entity. Otherwise, the evolution is considered converged.

After the recommended *Knowledge* finishes, the *Knowledge Map* stores the GUID of

the *Knowledge* and the score of *Superior* solution into the *Memes* entity for updating the running history which is used for recommendation.

3.5 The Running History of the Memes Module

In biology, the memes represents thoughts, symbols and activity patterns of the culture. The memes affects the evolution of culture such as gene evolution using selection, mutation, crossover and recombination. During human learning and researching, a nerve conduction pattern of the human's brain will be built into the memes for the specific problem. In the proposed architecture, the *Memes* module stores the running history of the *Knowledge* entities which records the information of each processed *Case* entity with used symbols and the running history of processed *Knowledge* entities.

The history records helps the *Member* entity to shorten the searching time for evolving the solution and maintain the diversity of solutions provided by different *Member* entities during *Group Interaction* and *Community Interaction*.

The *Memes* module manages the running history using "*Symbol*" which means the basic data format character. For example, *Symbols* includes 0 to 9 and the decimal point in digital format and 0 to 9 and A to Z are valid string characters. More examples are listed in Table 3.1. Because different problems use different symbols, the user must provide the specific symbols for *CIS* as the basic unit for recording the evolution history in the *Symbol Frequency Set (SFS)* during *Member Evolution*.

Table 3.1. Symbol examples.

Type	Example	Symbols
Binary	010110	{0, 1}
Hexadecimal	E9	{0, 1, 2, 3, …, 8, 9}, {A, B, C, D, E, F}
Integer	-10	{0, 1, 2, 3, …, 8, 9}, { "+", "-"}
Float	33.143	{0, 1, 2, 3, …, 8, 9}, { "+", "-", "."}
String	Hello	{0, 1, 2, 3, …, 8, 9}, {A, B, C, …, Y, Z}, { … }

The *Symbol* management of the *Memes* module recommends the best symbols of the solution for *CIS* and manages the *SFS*. The purpose of the *SFS* is to record the symbol usage frequency of symbols which appears in different positions. As the example in Fig. 3.6 shows, the first coordinate of the second group in *Set A* is ten. The value can be divided into two symbols which are 1 and 0. Therefore, the usage frequencies of used symbols in different positions will become the recommendation reference for the *Memes* entity to generate the *Reference Probability* for recommending an appropriate symbol to the *Member* entity. In Fig. 3.6, the probabilities of the first positions of the first coordinate in the second group can be translated into 3.3% ("0"), 34.8% ("1"), 22.3% ("2") and 38.6% ("3"). Therefore, the higher recommended symbols of the first position are "1" and "3".

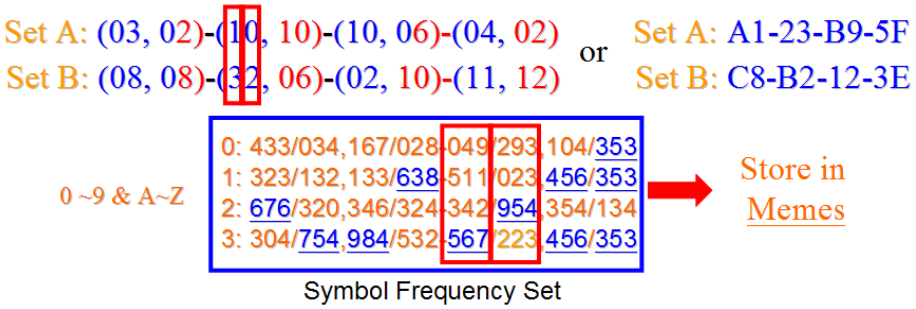

Fig. 3.6. An example of symbol frequency set.

3.6 Activity Operations

Activity operations are the interaction methods between *Member* entities in the *Group Interaction* and the *Community Interaction*. The purpose of the *Activity* operations is to preserve a suitable diversity of solutions. Default *Activity* operations of the proposed strategy are *Teach_To*, *Learn_From*, *Collaboration* and *Competition* operations. In Fig. 3.7, *Member* entity A uses the *Teach_To* operation as the interactive *Activity* operation. *Member* B automatically uses the *Learn_From* operation as the interaction operation. These *Activity* operations simulate the human learning procedure. This is analogous to teachers who provide the best solutions to students, which students accept without questions.

The *Member* entity uses the *Teach_To* operation to retrieve the best *Solution* entity from its *Memes* and deliver the entity to the opposite *Member* entity. After the entity has received by *Learn_From*, the opposite *Member* entity unconditionally stores the received *Solution* entity to its *Memes* entity.

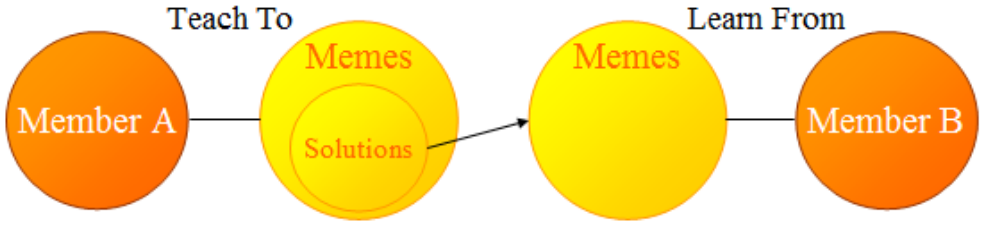

Fig. 3.7. Teach_To activity and Learn_From activity operations.

In the real world, the relationship between teacher and students is not only teaching and learning, but also discussion, collaboration and competition. In order to simulate these situations, a *Collaboration* operation is created to provide a cooperation activity for *Member* entities. As the procedure in Fig. 3.8 shows, *Member* A and B propose the best *Solution* entities and generate a new *Unit Set* entity by comparing the difference parts of their *Unit Set* entities.

The new *Unit Set* entity employs two steps. First, the new *Unit Set* entity stores the same *Particle Unit* entity parts which are selected from the *Unit Set* A and B entities, and then retrieves the remaining *Particle Unit* entities randomly from *Unit Set* A and B entities. After a new *Unit Set* entity has been created, the new entity is compared with *Unit Set* A and B entities. If the score of the new *Unit Set* entity is better than the best score of *Unit Set* A and B entities, the *Memes* entities of *Member* A and B entities will store the new *Unit Set* entity. Otherwise, the new entity is abandoned.

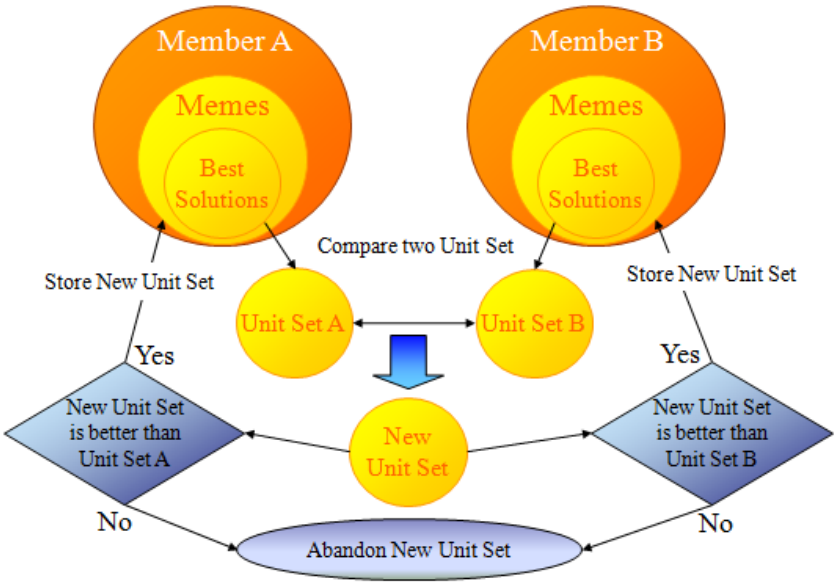

Fig. 3.8. Collaboration operation.

Competition is also an important facet of human learning. The *Competition* operation retrieves the best *Solution* entity of the processing *Case* entity from the *Member* A and B entities, and then each *Memes* entity unconditionally stores the better *Solution* entity. In Fig. 3.9, the *Memes* entity of *Member* A or B stores *Solution* B or A entity when the score of *Solution* B or A is better than the *Best Solution* of itself.

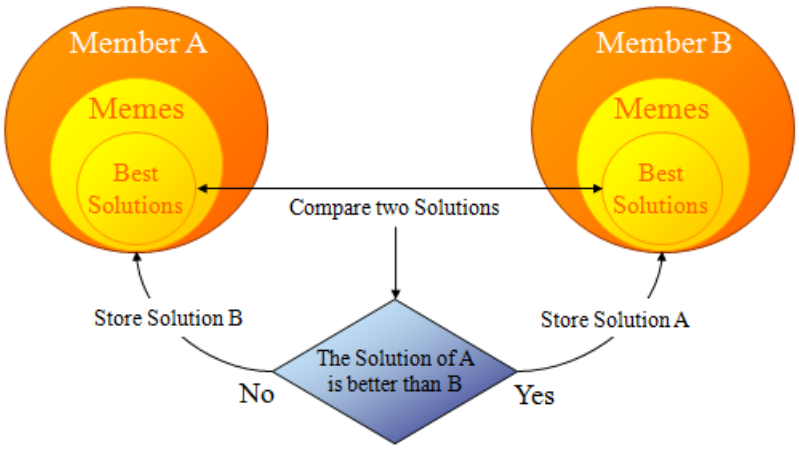

Fig. 3.9. Competition operation.

3.7 Group Interaction using Activity

In the *Group Interaction*, the major responsibility of the *Group* module is to manage the interaction of the *Member* entities. The *Member* entity uses its *Meme* entity and opposite *Member* entity's *Memes* to decide which *Activity* operations should be applied. After the independent procedure of *Member Evolution* and *Group Interaction*, *Member* entity can find the best solution for the processing problem.

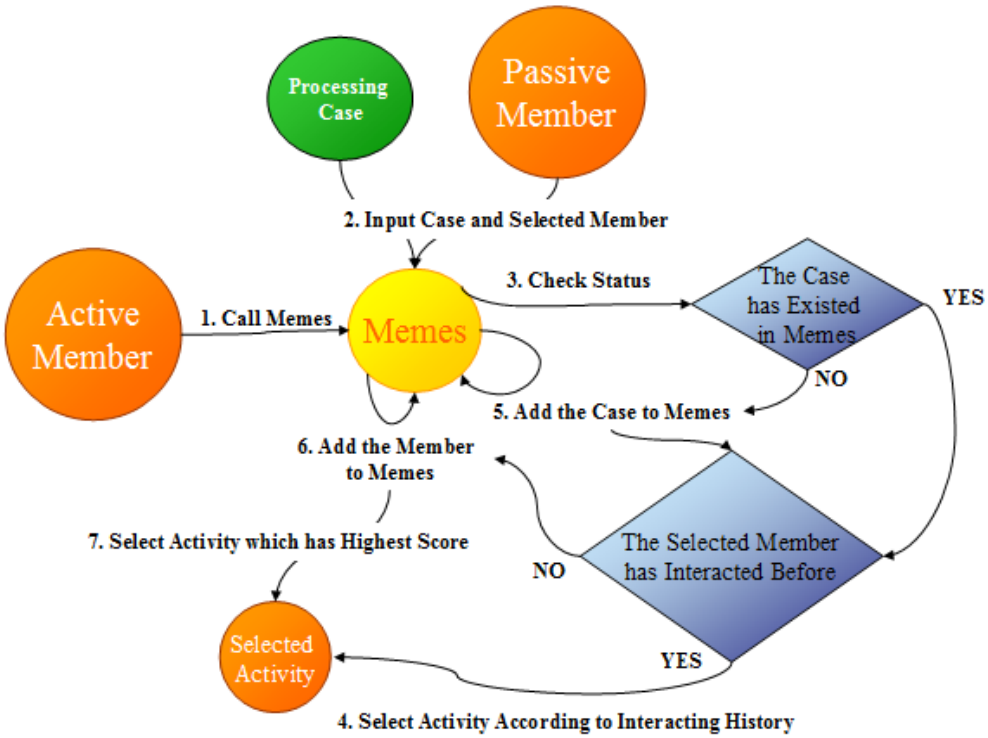

Fig. 3.10. The workflow of activity recommendation operation.

As shown in Fig. 3.10, all *Member* entities of the same *Group* entity will be set as alternate *Active* and *Passive* characters. The recommended *Activity* operation is selected according to the running history of the interacting *Member* entities. The running history records the result of all operated *Activity* operations which have been applied to the interaction with other *Member* entities. The selected probability of the recommended *Activity* operation is calculated using the records.

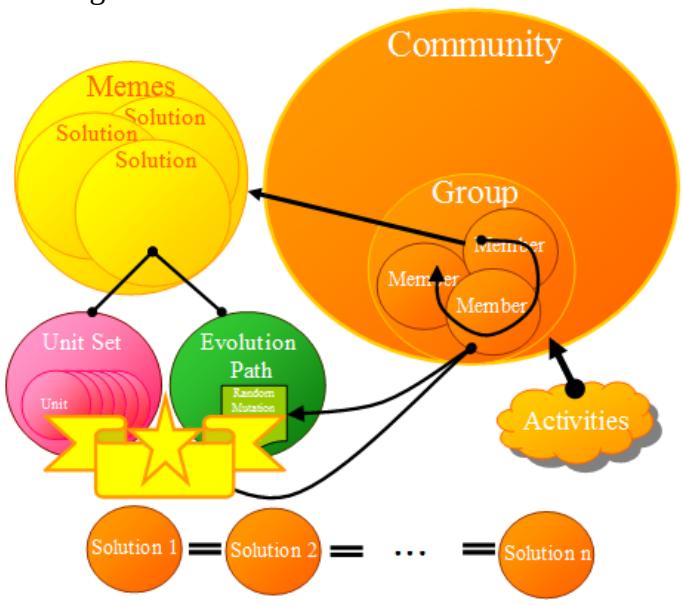

Fig. 3.11. Group interaction operation.

All modules in the *Group Interaction* operation are shown in Fig. 3.11. The *Group* module loads the default *Activity* operations for the *Member* entity to generate new solutions until all the solutions of every *Member* entity are identical. The *Group* entity selects a pair of *Member* entities randomly, and then, the pair selects an appropriate *Activity* operation for the

processing problem and generates new solutions after the interaction. After all the *Member* entities finished the interactions, the *Group* entity checks all solutions of each *Member* entities. If all solutions' score of the *Member* entity are the same, the *Member* entity is treated as a convergent entity. When all *Member* entities in each *Group* entities are convergent, the *Community Interaction* operation is launched.

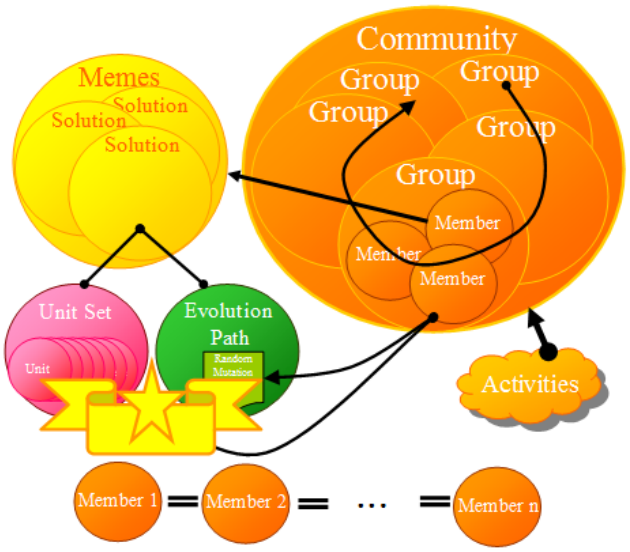

Fig. 3.12. Community interaction operation.

3.8 Community Interaction using Activity

The *Community Interaction* uses the same *Activity* operations as the *Group Interaction*. The recommended *Activity* operation is also selected by the *Member* pair which comprises the best *Member* entities from two different *Group* entities.

Fig. 3.12 describes the concept of *Community Interaction*. When all the *Member* entities have converged, the *Group* entity is treated as convergent. The interaction continues until all the *Group* entities are convergent. To ensure that the process finishes, the *Community* entity calculates the difference between the best solution and optimal solution to produce a *Stopping Threshold* which is used by the *Member Evolution*.

4. Simulation Results

In order to demonstrate the practicality of the proposed architecture, 11 operations from past research results have been implemented into 11 *Knowledge* modules which are two *Generator* and nine *Common Knowledge* modules. Only two Generator modules connect to the *Start Point* module as its parent, and then connect other 11 modules that every module are connected each other.

4.1 Function Parameters Optimization

a. Curve-fitting Problem

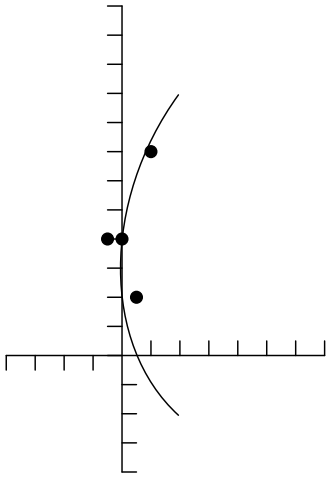

Fig. 4.1. Curve-fitting problem.

In the first experiment, assuming there are four points, (-1,8), (0,8), (1,4), and (2,16), in the plane shown in Fig. 4.1. The problem is to find a quadratic function that fits those data points. Equations 4.1 to 4.5 illustrate how to apply the linear algebra for solving the problem. By plugging the four data points into Eq.(4.5), the least mean-squared value is 20. Denote the quadratic function by:

$$f(t) = c_0 + c_1 t + c_2 t^2, \tag{4.1}$$

where C_0, C_1, and C_2 are coefficients for the polynomial. Fitting the data into the above equation, we get the following form:

$$\boldsymbol{A}\begin{bmatrix} \boldsymbol{C_0} \\ \boldsymbol{C_1} \\ \boldsymbol{C_2} \end{bmatrix} = \boldsymbol{B}, \tag{4.2}$$

where

$$A = \begin{bmatrix} 1 & -1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \end{bmatrix}, \quad B = \begin{bmatrix} 8 \\ 8 \\ 4 \\ 16 \end{bmatrix}, C = \begin{bmatrix} C_0 \\ C_1 \\ C_2 \end{bmatrix} \tag{4.3}$$

By using the least mean-squared method, we then get the following solution:

$$\boldsymbol{C} = \begin{bmatrix} \boldsymbol{C_0^*} \\ \boldsymbol{C_1^*} \\ \boldsymbol{C_2^*} \end{bmatrix} = (\boldsymbol{A^T A})^{-1} \boldsymbol{A^T B} = \begin{bmatrix} \boldsymbol{5} \\ \boldsymbol{-1} \\ \boldsymbol{3} \end{bmatrix}. \tag{4.4}$$

It means that the quadratic equation is:

$$f(t) = 5 - t + 3t^2. \tag{4.5}$$

b. Generalized Rastrigin's Function

$$F(\vec{x}) = A \cdot n + \sum_{i=1}^{n} x_i^2 - A \cdot \cos(\omega \cdot x_i)$$
$$A = 10 \; ; \; \omega = 2 \cdot \pi \; ; \; x_i \in [-5.12, 5.12]$$

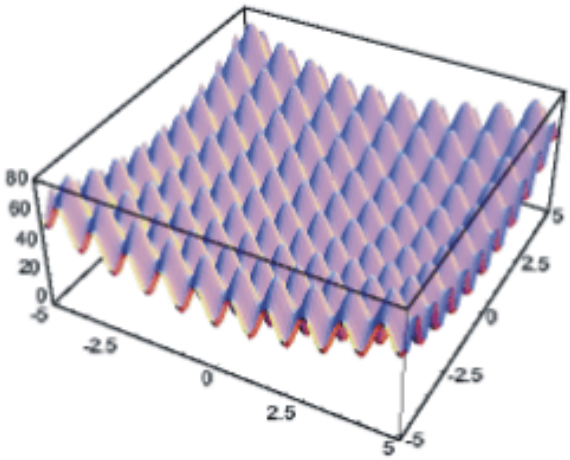

Fig. 4.2. Generalized Rastrigin's function [36].

Rastrigin function was first proposed by Rastrigin [36] and has been generalized by Muhlenbein etc [38]. Due to the large search space and its large number of local minima, it has been seen as a difficult problem. The optimal value of generalized Rastrigin's function is zero.

4.2 CKE+ Fuzzy Adaptive Genetic Algorithm

Fuzzy adaptive genetic algorithm (FAGA) has been proposed in [39]. FAGA not only uses fuzzy membership functions and an elite strategy to preserve the diversity of the population and to enhance the performance, but also modified the mutation and crossover operations using a sequence frequency set which is the predecessor of the proposed symbol frequency set. In order to adapt the different situation automatically, FAGA has abandoned mutation and crossover rates, and recurrence. The population size is also adaptively adjusted during the processing according to the situation. In the experiment, FAGA uses the evolution results of CKE as its initial population to enhance the performance. The flowchart of FAGA is shown in Fig. 4.3.

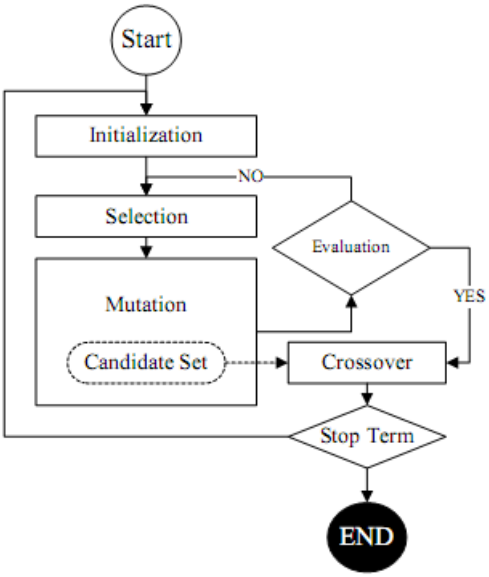

Fig. 4.3. The flowchart of FAGA.

4.3 Results with TGA, FAGA and CKE+FAGA

The experiment ran on a laptop with an Intel Centrino 2 and 2G DDRII RAM. In the experiment, each value shown in Fig. 4.4 to 4.7 are the average from 30 independent repetitions. The results in Table 4.1 and 4.2 are the average value of Fig. 4.4 to 4.7. The necessary parameters used in the traditional genetic algorithm are summarized as follows:

Recurrence	Population Size	Mutation Rate	Crossover Rate
500	50	0.05	0.8

As can be seen from Table 4.1 TGA spent the most time to approach the optimal unsuccessfully rather than FAGA and CKE+FAGA. Although, the average result of CKE+FAGA only shows little improvement, its processing time is half that of FAGA. The result confirms the efficiency of CKE and demonstrates that CKE can help shorten the processing time without any penalties.

Table 4.1. Results of curve-fitting problem.

	Traditional GA	FAGA	CKE+FAGA
Average Time (s)	18.0322908	13.3131773	6.1632032067
Average Value	23.45417767	20.45089747	20.39634233

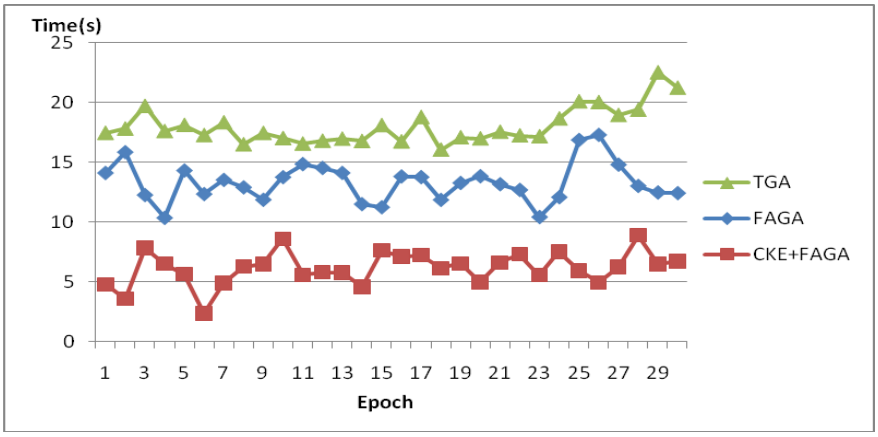

Fig. 4.4. Processing time for the curve-fitting problem.

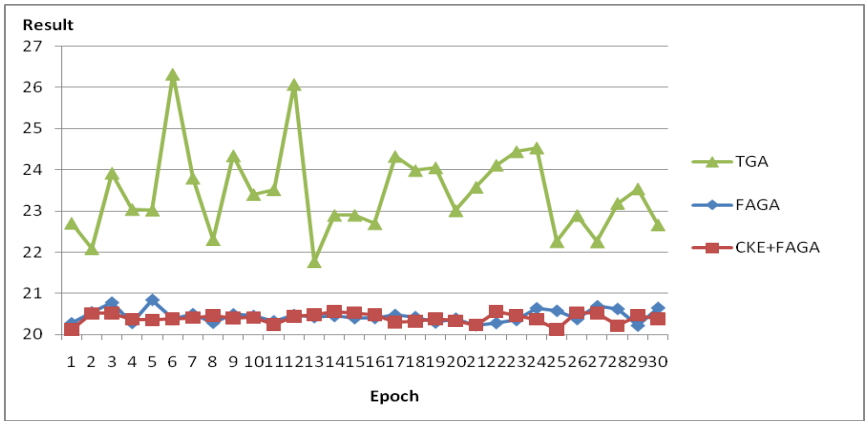

Fig. 4.5. Simulation results for the curve-fitting problem.

The average results of generalized Rastrigin's function show that the performance of TGA is too far away from the result of FAGA and CKE+FAGA. As a result, Fig. 4.7 only

shows the values of FAGA and CKE+FAGA. The results show that CKE+FAGA only needs half the time to find the optimal value which is also better than the average value for FAGA and TGA.

Table 4.2. Results for the generalized Rastrigin's function.

	Traditional GA	FAGA	CKE+FAGA
Average Time (s)	16.81403913	3.222696367	1.469646367
Average Value	3.265926667	1.17667E-05	1.333E-07

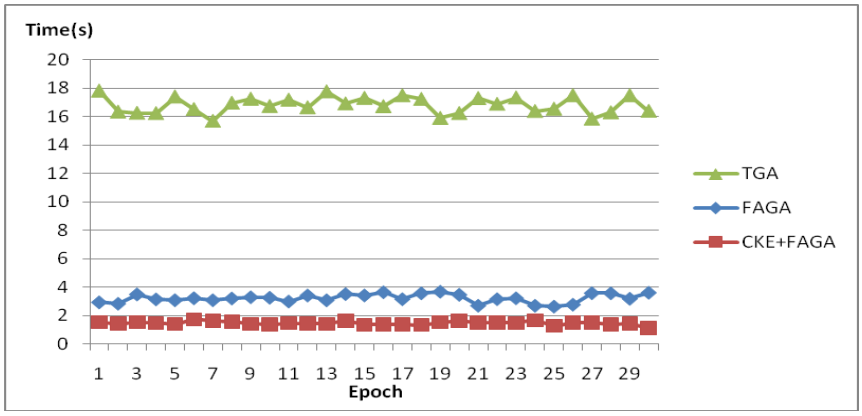

Fig. 4.6. Processing time for the generalized Rastrigin's function.

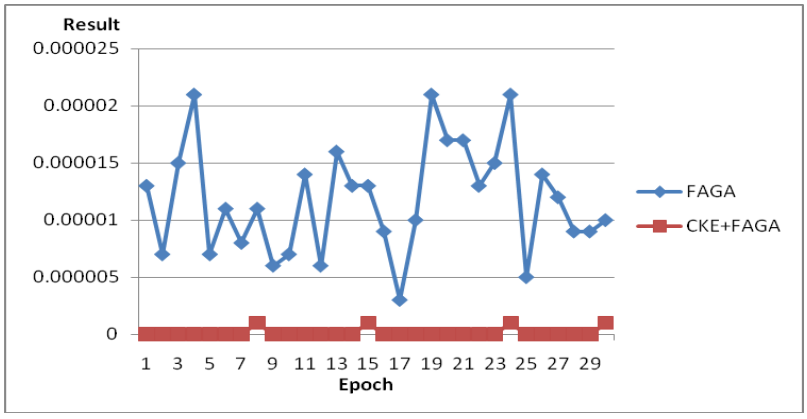

Fig. 4.7. Simulation results for the generalized Rastrigin's function.

5. Conclusion

In this paper, we proposed a strategy and an architecture which can find new algorithms and solutions automatically. The proposed evolution strategy uses the proposed architecture to deal with problems. The user can propose new knowledge and solutions following the definitions which are described as *Knowledge Definitions* and *Case Definitions*. The *Knowledge* module can package up the domain knowledge which is generated by various researchers. Although the proposed strategy can recombine various *Knowledge* processing compositions, the implementation of the *Knowledge* modules still need to be manually created and hence needs human intervention. Human can use the strategy and architecture to save time. As a result, the strategy and architecture are proposed as a framework to assist researchers in finding optimal knowledge processing compositions from vast collections of existing knowledge. Moreover, the framework also provides common definitions for

convenient information exchange.

Based on the strategies from the *Member Evolution* operation to the *Group Interaction* and the *Community Interaction*, different *Community* entities could exchange their solutions as the reference to each other. Future work includes building a *Community Network* to provide an interaction environment for *Community* modules and their research results to be easily shared with other researchers.

In the proposed strategy, the default *Activity* operations are very simple and not sufficient to represent all the interactivity of human beings. Hence, in future work, the *Activity* operations will be separated from the management of *Member* module and be designed as an interface similar to *IKnowledge*. Through the interface, researcher can implement various *Activity* modules to simulate the activity of human beings, organizations and communities. Researchers can exchange their *Activity* modules.

References

[1] Birattari, M., Pellegrini, P. and Dorigo, M., "On the Invariance of Ant Colony Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, pp.732-742, Dec. 2007.

[2] Gang Wang, Wenrui Gong, DeRenzi, B. and Kastner, R., "Ant Colony Optimizations for Resource- and Timing-Constrained Operation Scheduling," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 6, pp.1010-1029, June 2007.

[3] Martens, D., De Backer, M., Haesen, R., Vanthienen, J., Snoeck, M. and Baesens, B., "Classification With Ant Colony Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 5, pp.651-665, Oct. 2007.

[4] Leung, K., Cheong, F. and Cheong, C., "Generating Compact Classifier Systems Using a Simple Artificial Immune System," IEEE Transactions on Systems, Man, and Cybernetics, Part B, vol. 37, no. 5, pp.1344-1356, Oct. 2007.

[5] Mardiris, V., Sirakoulis, G.Ch., Mizas, C., Karafyllidis, I. and Thanailakis, A. "A CAD System for Modeling and Simulation of Computer Networks Using Cellular Automata," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 38, no. 2, pp.253-264, March 2008.

[6] Mattiussi, C. and Floreano, D., "Analog Genetic Encoding for the Evolution of Circuits and Networks," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 5, pp.596-607, Oct. 2007.

[7] Sanchez, L.and Couso, I., "Advocating the Use of Imprecisely Observed Data in Genetic Fuzzy Systems," *IEEE Transactions on Fuzzy Systems*, vol. 15, no. 4, pp.551-562, Aug. 2007.

[8] Lymperopoulos, D.G., Tsitsas, N.L. and Kaklamani, D.I., "A Distributed Intelligent Agent Platform for Genetic Optimization in CEM-Applications in a Quasi-Point Matching Method," *IEEE Transactions on Antennas and Propagation*, vol. 55, no. 3, Part 1, pp.619-628, March 2007.

[9] Aguilar-Ruiz, J.S., Giraldez, R. and Riquelme, J.C., "Natural Encoding for Evolutionary Supervised Learning," *IEEE Transactions on Evolutionary Computation*, Vol. 11, no. 4, pp.466-479, Aug. 2007.

[10] Shiu Yin Yuen and Chi Kin Chow, "A Genetic Algorithm That Adaptively Mutates and Never Revisits," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 2, pp.454-472, April 2009.

[11] Ali, H., Doucet, A. and Amshah, D. I., "GSR-A New Genetic Algorithm for Improving Source and Channel Estimates," *IEEE Transactions on Circuits and Systems I: Regular*

Papers, vol. 54, no. 5, pp.1088-1098, May 2007.
[12] Malossini, A., Blanzieri, E. and Calarco, T., "Quantum Genetic Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 2, pp.231-241, April 2008.
[13] Perales-Gravan, C. and Lahoz-Beltra, R., "An AM Radio Receiver Designed With a Genetic Algorithm Based on a Bacterial Conjugation Genetic Operator," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 2, pp.129-142, April 2008.
[14] Hua-Liang Wei, Billings, S.A., Yifan Zhao and Lingzhong Guo, "Lattice Dynamical Wavelet Neural Networks Implemented Using Particle Swarm Optimization for Spatio–Temporal System Identification," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp.181-185, Jan. 2009.
[15] Faa-Jeng Lin, Li-Tao Teng, Jeng-Wen Lin and Syuan-Yi Chen, "Recurrent Functional-Link-Based Fuzzy-Neural-Network-Controlled Induction-Generator System Using Improved Particle Swarm Optimization," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 5, pp.1557-1577, May 2009.
[16] Goudos, S.K., Zaharis, Z.D., Kampitaki, D.G., Rekanos, I.T. and Hilas, C.S., "Pareto Optimal Design of Dual-Band Base Station Antenna Arrays Using Multi-Objective Particle Swarm Optimization With Fitness Sharing," *IEEE Transactions on Magnetics*, vol. 45, no. 3, pp.1522-1525, March 2009.
[17] [Zielinski, K., Weitkemper, P., Laur, R. and Kammeyer, K.-D., "Optimization of Power Allocation for Interference Cancellation with Particle Swarm Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 1, pp.128-150, Feb. 2009.
[18] Ramezani, M., Haghifam, M.-R., Singh, C., Seifi, H. and Moghaddam, M.P., "Determination of Capacity Benefit Margin in Multiarea Power Systems Using Particle Swarm Optimization," *IEEE Transactions on Power Systems*, vol. 24, no. 2, pp.631-641, May 2009.
[19] Zhan, Z.-H., Zhang, J., Li, Y. and Chung, H. S.-H., "Adaptive Particle Swarm Optimization," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 2009.
[20] Sharma, K.D., Chatterjee, A. and Rakshit, A., "A Hybrid Approach for Design of Stable Adaptive Fuzzy Controllers Employing Lyapunov Theory and Particle Swarm Optimization," *IEEE Transactions on Fuzzy Systems*, vol. 17, no. 2, pp.329-342, April 2009.
[21] Yang Zhang and Rockett, P.I., "Application of Multiobjective Genetic Programming to the Design of Robot Failure Recognition Systems," *IEEE Transactions on Automation Science and Engineering*, vol. 6, no. 2, pp.372-376, April 2009.
[22] Shintemirov, A., Tang, W. and Wu, Q.H., "Power Transformer Fault Classification Based on Dissolved Gas Analysis by Implementing Bootstrap and Genetic Programming," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 39, no. 1, pp.69-79, Jan. 2009.
[23] Darren Doherty and Colm O'Riordan, "Effects of Shared Perception on the Evolution of Squad Behaviors," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 1, no. 1, pp. 50-62, March 2009.
[24] Walker, J.A. and Miller, J.F., "The Automatic Acquisition, Evolution and Reuse of Modules in Cartesian Genetic Programming," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 4, pp.397-417, Aug. 2008.
[25] Chion, C., Landry, J.-A. and Da Costa, L., "A Genetic-Programming-Based Method for Hyperspectral Data Information Extraction: Agricultural Applications," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 46, no. 8, pp. 2446-2457, Aug. 2008.
[26] Sobester, A., Nair, P. B. and Keane, A. J., "Genetic Programming Approaches for

Solving Elliptic Partial Differential Equations," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 4, pp.469-478, Aug. 2008.
[27] Hasegawa, Y. and Iba, H., "A Bayesian Network Approach to Program Generation," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 6, pp.750-764, Dec. 2008.
[28] Day, P. and Nandi, A.K., "Binary String Fitness Characterization and Comparative Partner Selection in Genetic Programming," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 6, pp.724-735, Dec. 2008.
[29] Alcala, R., Ducange, P., Herrera, F., Lazzerini, B. and Marcelloni, F., "A Multi-Objective Evolutionary Approach to Concurrently Learn Rule and Data Bases of Linguistic Fuzzy Rule-Based Systems," *IEEE Transactions on Fuzzy Systems*, 2009.
[30] Hansen, N., Niederberger, A.S.P., Guzzella, L. and Koumoutsakos, P., "A Method for Handling Uncertainty in Evolutionary Optimization With an Application to Feedback Control of Combustion," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 1, pp.180-197, Feb. 2009.
[31] Yasakethu, S.L.P., Fernando, W.A.C. and Kondoz, A., "Evolution strategy based rate controlling for off-line video coding," *Electronics Letters*, vol. 45, no. 4, pp.204-205, Feb. 2009.
[32] Yasakethu, S.L.P., Fernando, W.A.C. and Kondoz, A.M., "Rate controlling in off line 3D video coding using evolution strategy," *IEEE Transactions on Consumer Electronics*, vol. 55, no. 1, pp.150-157, Feb. 2009.
[33] Rojanavasu, P., Hai Huong Dam, Abbass, H.A., Lokan, C. and Pinngern, O., "A Self-Organized, Distributed, and Adaptive Rule-Based Induction System," *IEEE Transactions on Neural Networks*, vol. 20, no. 3, pp.446-459, March 2009.
[34] Dam, H.H., Abbass, H.A., Lokan, C. and Xin Yao, "Neural-Based Learning Classifier Systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, no. 1, pp.26-39, Jan. 2008.
[35] Bernado-Mansilla, E. and Garrell-Guiu, J.M., "Accuracy-Based Learning Classifier Systems: Models, Analysis, and Applications to Classification Tasks," *Evolutionary Computation*, vol. 11, no. 3, pp.209-238, 2003.
[36] The Generalized Rastrigin Function - http://tracer.lcc.uma.es/problems/rastrigin/rastrigin.html
[37] A. Törn and A. Zilinskas. "Global Optimization," *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 1989.
[38] Mühlenbein, H., Schomisch, D., and Born, J., "The Parallel Genetic Algorithm as Function Optimizer," *Parallel Computing*, vol. 17, pp. 619-632, 1991.
[39] Chang, Y.-T., Huang, Y.-P., and Sandnes, F.E., "Efficient Layout of Multisensors using Fuzzy Adaptive Genetic Algorithm," *in Proc. NAFIPS*, San Diego, CA, USA, pp.216-221, June 2007.