A CONCURRENT NEURAL NETWORK - GENETIC PROGRAMMING MODEL FOR DECISION SUPPORT SYSTEMS

AJITH ABRAHAM[1], CRINA GROSAN[2], CONG TRAN[3], LAKHMI JAIN[3]

[1]*School of Computer Science and Engineering,*
Chung-Ang University, Korea, ajith.abraham@ieee.org

[2]*Department of Computer Science,*
Babes-Bolyai University Cluj-Napoca, Romania, cgrosan@cs.ubbcluj.ro

[3]*School of Electrical and Information Engineering*
University of South Australia, Australia, lakhmi.jain@unisa.edu.au

Abstract. This paper suggests a decision support system for tactical air combat environment using a combination of unsupervised learning for clustering the data and three well known genetic programming techniques to classify the different decision regions accurately. The genetic programming techniques used are: Linear Genetic programming (LGP), Multi Expression Programming (MEP) and Gene Expression Programming (GEP). The clustered data is used as the inputs to the genetic programming algorithms. Some simulation results demonstrating the difference of these techniques and are also performed. Experiment results reveal that the proposed method is efficient.

1. Introduction

Several decision support systems have been developed mostly in various fields including medical diagnosis, business management, control system, command and control of defense, air traffic control and so on [9][1] [1]. Usually previous experience or expert knowledge is often used to design decision support systems. Several adaptive learning frameworks for constructing intelligent decision support systems have been proposed [1][2][3][4][1]. To develop an intelligent decision support system, we need a holistic view on the various tasks to be carried out including data management and knowledge management (reasoning techniques) [1][1]. The focus of this paper is to develop a Tactical Air Combat Decision Support System (TACDSS) with minimal prior knowledge, which could also provide optimal decision scores. As shown in Figure 1, we propose a concurrent unsupervised neural network to cluster the decision regions and genetic programming techniques to automatically generate the decision scores. Section 2 presents the problem of decision making in tactical air combat system. In Section 3, we introduce some theoretical concepts of Self Organizing Map (SOM) followed by the genetic programming techniques namely Linear Genetic programming (LGP), Multi Expression Programming (MEP) and Gene Expression programming [1]. Experimentation results are provided in Section 4 and some conclusions are also provided towards the end.

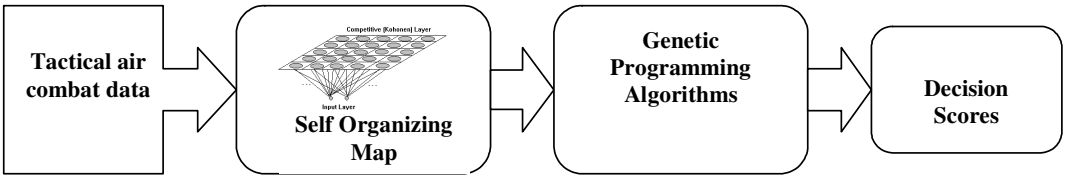


Figure 1. Concurrent unsupervised and genetic programming models for decision support systems

2. The Tactical Air Combat Environment

Figure 2 presents a typical scenario of air combat tactical environment. The Airborne Early Warning and Control (AEWC) is performing surveillance in a particular area of operation. It has two hornets (F/A-18s) under its control at the ground base as shown "+" in the left corner of Figure 2. An air-to-air fuel tanker (KB707) "Í" is on the station and the location and status are known to the AEWC. Two of the hornets are on patrol in the area of Combat Air Patrol (CAP). Sometime later, the AEWC on-board sensors detects 4 hostile aircrafts (Mig-29) shown as "O". When the hostile aircrafts enter the surveillance region (shown as dashed circle) the mission system software is able to identify the enemy aircraft and its distance from the Hornets in the ground base or in the CAP. The mission operator has few options to make a decision on the allocation of hornets to intercept the enemy aircraft.

- Send the hornet directly to the spotted area and intercept,
- Call the hornet in the area back to ground base and send another Hornet from the ground base

- Call the hornet in the area to refuel before intercepting the enemy aircraft

The mission operator will base his decisions on a number of decision factors, such as:

- Fuel used and weapon status of hornet in the area
- Interrupt time of Hornet in the ground base and the Hornet at the CAP to stop the hostile
- The speed of the enemy fighter aircraft and the type of weapons it posses
- The information of enemy aircraft with type of aircraft, weapon, number of aircraft

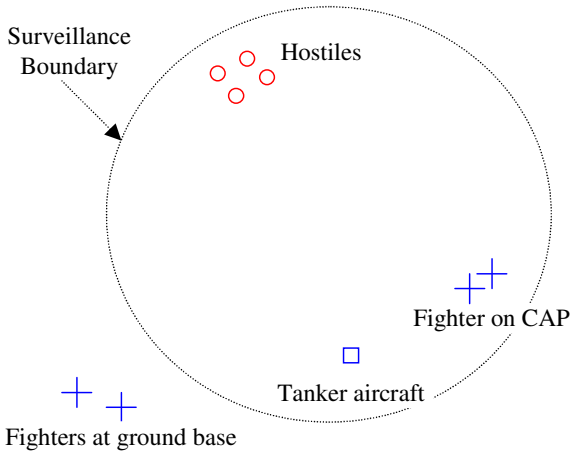


Figure 2. A simple scenario of the air combat

From the above simple scenario, it is evident that there are several important decision factors of the tactical environment that might directly affect the air combat decision. In the simple tactical air combat, the four decision factors that could affect the decision options for calling the Hornet in the CAP or the Hornet at the ground base are the following:

- '*fuel status*' - quantity of fuel available to perform the intercept,
- '*weapon possession status*' – quantity of weapons available in the Hornet,
- '*interrupt time*' - time required by the hornet to interrupt the hostile and
- '*danger situation*' - information of the Hornet and the hostile in the battlefield.

Each factors has difference range of unit such as the *fuel status* (0 to 1000 litres), *interrupt time* (0 to 60 minutes), *weapon status* (0 to 100 %) and *danger situation* (0 to 10 points). We used the following two expert rules for developing the fuzzy inference system.

- The decision selection will have small value if the *fuel status* is low, the *interrupt time* is slow, the hornet has low *weapon status*, and the *danger situation* is high.
- The decision selection will have high value if the *fuel status* is full, the *interrupt time* is fast, the hornet has high *weapon status* and the *danger situation* is low.

Table 1. Decision factors for the tactical air combat

Fuel used	Time Intercept	Weapon Status	Danger Situation	Decision
Full	Fast	Sufficient	Very Dangerous	Good
Half	Normal	Enough	Dangerous	Acceptable
Low	Slow	Insufficient	Endanger	Bad

In a tactical air combat environment, decision-making is always based on all states of decision factors. But sometimes, a mission operator or commander could make a decision based on an important factor, such as the fuel used is too low, the enemy has more powerful weapons, quality and quantity of enemy aircraft and so on. Table 1 shows some typical scores (decision selection point) taking into account of the various tactical air combat decision factors.

3. Learning Decision Regions Using Hybrid Un-Supervised and supervised Learning Paradigms

1.0. *Self-Organizing Feature Maps*

The Kohonen's projection algorithm is the fundamental ideas of unsupervised, competitive learning, self-organization, and global ordering [8]. An input from the original high-dimensional space causes dominant response of one neuron in the 2D array of neurons, and only this "*winning*" neuron together with its *neighboring neurons* get to adjust their weights. For example, adjusting weights of neurons in a local neighborhood around the winning neuron leads to global ordering through continuous learning. This operation of the SOM algorithm shows the ability of biological neurons that perform global ordering based on local interactions. This global order leads to the creation of natural structures and biologically motivated configurations and shapes, which are created according to laws of minimum energy, time, or complexity.

2.0. *Genetic Programming*

Once the clusters are defined using SOM, the next step is to apply the Genetic Programming (GP) models to learn the different decision regions for the given input data. Linear Genetic Programming (LGP), Multi Expression Programming (MEP) and Gene Expression Programming (MEP) are the three well known genetic programming techniques explored in this paper.

1.2.0. *Linear Genetic Programming (LGP)*

Linear genetic programming is a variant of the GP technique that acts on linear genomes [15]. Its main characteristics in comparison to tree-based GP lies in that the evolvable units are not the expressions of a functional programming language (like LISP), but the programs of an imperative language (like c/c ++). An alternate approach is to evolve a computer program at the machine code level, using lower level representations for the individuals. This can tremendously hasten the evolution process as, no matter how an individual is initially represented, finally it always has to be represented as a piece of machine code, as fitness evaluation requires physical execution of the individuals. The basic unit of evolution here is a native machine code instruction that runs on the floating-point processor unit (FPU). Since different instructions may have different sizes, here instructions are clubbed up together to form instruction blocks of 32 bits each. The instruction blocks hold one or more native machine code instructions, depending on the sizes of the instructions. A crossover point can occur only between instructions and is prohibited from occurring within an instruction. However the mutation operation does not have any such restriction. LGP uses a specific linear representation of computer programs. Instead of the tree-based GP expressions of a functional programming language (like *LISP*) programs of an imperative language (like *C*) are evolved. A LGP individual is represented by a variable-length sequence of simple *C* language instructions. Instructions operate on one or two indexed variables (registers) *r*, or on constants *c* from predefined sets.
The result is assigned to a destination register, for example, *ri = rj* c*.
Here is an example LGP program:

```
void LGP(double v[8])
[0] = v[5] + 73;
v[7] = v[3] – 59;
if (v[1] > 0)
if (v[5] > 21)
v[4] = v[2] . v[1];
v[2] = v[5] + v[4];
v[6] = v[7] . 25;
v[6] = v[4] – 4;
v[1] = sin(v[6]);
if (v[0] > v[1])
v[3] = v[5] . v[5];
v[7] = v[6] . 2;
v[5] = v[7] + 115;
if (v[1] <= v[6])
v[1] = sin(v[7]);
}
```

A LGP can be turned into a functional representation by successive replacements of variables starting with the last effective instruction. The maximum number of symbols in a LGP chromosome is 4 * Number of instructions. Evolving programs in a low-level language allows us to run those programs directly on the computer processor, thus avoiding the need of an interpreter. In this way the computer program can be evolved very quickly. An important LGP parameter is the number of registers used by a chromosome. The number of registers is usually equal to the number of attributes of the problem. If the problem has only one attribute, it is impossible to obtain a complex expression such as the quartic polynomial. In that case we have to use several supplementary registers. The number of supplementary registers depends on the complexity of the expression being discovered. An inappropriate choice can have disastrous effects on the program being evolved. LGP uses a modified steady-state algorithm. The initial population is randomly generated. The following steps are repeated until a termination criterion is reached: Four individuals are randomly selected from the current population. The best two of them are considered the winners of the tournament and will act as parents. The parents are recombined and the offspring are mutated and then replace the losers of the tournament. We used a LGP technique that manipulates and evolves a program at the machine code level. The settings of various linear genetic programming system parameters are of utmost importance for successful performance of the system. The population space has been subdivided into multiple subpopulation or demes. Migration of individuals among the subpopulations causes evolution of the entire population. It helps to maintain diversity in the population, as migration is restricted among the demes. Moreover, the tendency towards a bad local minimum in one deme can be countered by other demes with better search directions. The various LGP search parameters are the mutation frequency, crossover frequency and the reproduction frequency: The crossover operator acts by exchanging sequences of instructions between two tournament winners. Steady state genetic programming approach was used to manage the memory more effectively.

2.2.0. *Multi Expression Programming (MEP)*

MEP genes are (represented by) substrings of a variable length [14]. The number of genes per chromosome is constant. This number defines the length of the chromosome. Each gene encodes a terminal or a function symbol. A gene that encodes a function includes pointers towards the function arguments. Function arguments always have indices of lower values than the position of the function itself in the chromosome. The proposed representation ensures that no cycle arises while the chromosome is decoded (phenotypically transcripted). According to the proposed representation scheme, the first symbol of the chromosome must be a terminal symbol. In this way, only syntactically correct programs (MEP individuals) are obtained. An example of chromosome using the sets F= {+, *} and T= {a, b, c, d} is given below:

1: a
2: b
3: + 1, 2
4: c
5: d
6: + 4, 5
7: * 3, 6

The maximum number of symbols in MEP chromosome is given by the formula:

$Number\_of\_Symbols = (n + 1) * (Number\_of\_Genes - 1) + 1$,

where n is the number of arguments of the function with the greatest number of arguments. The maximum number of effective symbols is achieved when each gene (excepting the first one) encodes a function symbol with the highest number of arguments. The minimum number of effective symbols is equal to the number of genes and it is achieved when all genes encode terminal symbols only.

The translation of a MEP chromosome into a computer program represents the phenotypic transcription of the MEP chromosomes. Phenotypic translation is obtained by parsing the chromosome top-down. A terminal symbol specifies a simple expression. A function symbol specifies a complex expression obtained by connecting the operands specified by the argument positions with the current function symbol.

For instance, genes 1, 2, 4 and 5 in the previous example encode simple expressions formed by a single terminal symbol. These expressions are:

$E_1 = a$,
$E_2 = b$,
$E_4 = c$,
$E_5 = d$,

Gene 3 indicates the operation + on the operands located at positions 1 and 2 of the chromosome. Therefore gene 3 encodes the expression: $E_3 = a + b$. Gene 6 indicates the operation + on the operands located at positions 4 and 5. Therefore gene 6 encodes the expression: $E_6 = c + d$. Gene 7 indicates the operation * on the operands located at position 3 and 6. Therefore gene 7 encodes the expression: $E_7 = (a + b) * (c + d)$. E_7 is the expression encoded by the whole chromosome.
There is neither practical nor theoretical evidence that one of these expressions is better than the others. This is why each MEP chromosome is allowed to encode a number of expressions equal to the chromosome length (number of genes). The chromosome described above encodes the following expressions:
$E_1 = a$,
$E_2 = b$,
$E_3 = a + b$,
$E_4 = c$,
$E_5 = d$,
$E_6 = c + d$,
$E_7 = (a + b) * (c + d)$.
The value of these expressions may be computed by reading the chromosome top down. Partial results are computed by dynamic programming and are stored in a conventional manner.
Due to its multi expression representation, each MEP chromosome may be viewed as a forest of trees rather than as a single tree, which is the case of Genetic Programming.

Fitness assignment

As MEP chromosome encodes more than one problem solution, it is interesting to see how the fitness is assigned. The chromosome fitness is usually defined as the fitness of the best expression encoded by that chromosome. For instance, if we want to solve symbolic regression problems, the fitness of each sub-expression E_i may be computed using the formula:

$$f(E_i) = \sum_{k=1}^{n} \left| o_{k,i} - w_k \right|,$$

where $o_{k,i}$ is the result obtained by the expression E_i for the fitness case k and w_k is the targeted result for the fitness case k. In this case the fitness needs to be minimized. The fitness of an individual is set to be equal to the lowest fitness of the expressions encoded in the chromosome:
When we have to deal with other problems, we compute the fitness of each sub-expression encoded in the MEP chromosome. Thus, the fitness of the entire individual is supplied by the fitness of the best expression encoded in that chromosome.

MEP strengths

A GP chromosome generally encodes a single expression (computer program). By contrast, a MEP chromosome encodes several expressions. The best of the encoded solution is chosen to represent the chromosome (by supplying the fitness of the individual). The MEP chromosome has some advantages over the single-expression chromosome especially when the complexity of the target expression is not known. This feature also acts as a provider of variable-length expressions. Other techniques (such as Gramatical Evolution (GE) [15] or Linear Genetic Programming (LGP) [5]) employ special genetic operators (which insert or remove chromosome parts) to achieve such a complex functionality.

3.2.0. *Gene Expression Programming (GEP)*

The individuals of gene expression programming [1] are encoded in linear chromosomes which are expressed or translated into expression trees (branched entities). Thus, in GEP, the genotype (the linear chromosomes) and the phenotype (the expression trees) are different entities (both structurally and functionally) that, nevertheless, work together forming an indivisible whole. In contrast to its analogous cellular gene expression, GEP is rather simple. The main players in GEP are only two: the chromosomes and the Expression Trees (ETs), being the latter the expression of the genetic information encoded in the chromosomes. As in nature, the process of information decoding is called translation. And this translation implies obviously a kind of code and a set of rules. The genetic code is very simple: a one-to-one relationship between the symbols of the chromosome and the functions or terminals they represent. The rules are also very simple: they determine the spatial organization of the functions and terminals in the ETs and the type of interaction between sub-ETs. GEP uses linear chromosomes that store

expressions in breadth-first form. A GEP gene is a string of terminal and function symbols. GEP genes are composed of a *head* and a *tail*. The head contains both function and terminal symbols. The tail may contain terminal symbols only. For each problem the head length (denoted h) is chosen by the user. The tail length (denoted by t) is evaluated by:
$t = (n - 1)h + 1$,
where n is the number of arguments of the function with more arguments.
Let us consider a gene made up of symbols in the set S:
$S = \{., /, +, -, a, b\}$.
In this case $n = 2$. If we choose $h = 10$, then we get $t = 11$, and the length of the gene is $10 + 11 = 21$. Such a gene is given below:

*C*GEP = +* *ab* - +*aab* + *ababbbababb.*

The *expression* encoded by the gene C_{GEP} is:

$E = a + b \,.\, ((a + b) - a)$.

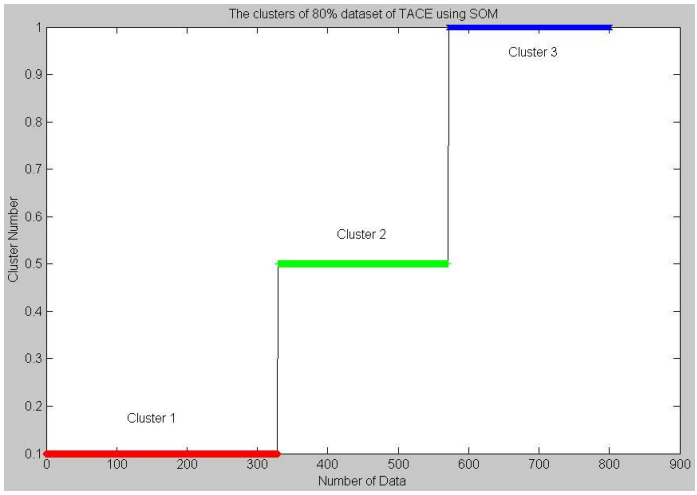
GEP genes may be linked by a function symbol in order to obtain a fully functional chromosome. In the current version of GEP the linking functions for algebraic expressions are addition and multiplication. A single type of function is used for linking multiple genes. Genetic operators are the core of all genetic algorithms, and two of them are common to all evolutionary systems: selection and replication. Although the center of the storm, these operators, by themselves, do nothing in terms of evolution. In fact, they can only cause genetic drift, making populations less and less diverse with time until all the individuals are exactly the. So, the cornerstone of all evolutionary systems is modification, or more specifically, the genetic operators that cause variation. And different algorithms create this modification differently. GEP uses mutation, recombination and transposition. GEP uses a generational algorithm. The initial population is randomly generated. The following steps are repeated until a termination criterion is reached: A fixed number of the best individuals enter the next generation (elitism). The mating pool is filled by using binary tournament selection. The individuals from the mating pool are randomly paired and recombined. Two offspring are obtained by recombining two parents. The offspring are mutated and they enter the next generation. There are some problems regarding multigenic chromosomes. Generally, it is not a good idea to assume that the genes may be linked either by addition or by multiplication. Providing a particular linking operator means providing partial information to the expression which is discovered. But, if all the operators *{+,-, ., /}* are used as linking operators, then the complexity of the problem substantially grows (since the problem of determining how to mix these operators with the genes is as difficult as the initial problem). Furthermore, the number of genes in the GEP multigenic chromosome raises a problem. As can be seen in [6], the success rate of GEP increases with the number of genes in the chromosome. But, after a certain value, the success rate decreases if the number of genes in the chromosome is increased. This happens because we cannot force a complex chromosome to encode a less complex expression. A large part of the chromosome is unused if the target expression is short and the head length is large. Note that this problem arises usually in systems that employ chromosomes with a fixed length.

4. Experiment Results and Performance Analysis

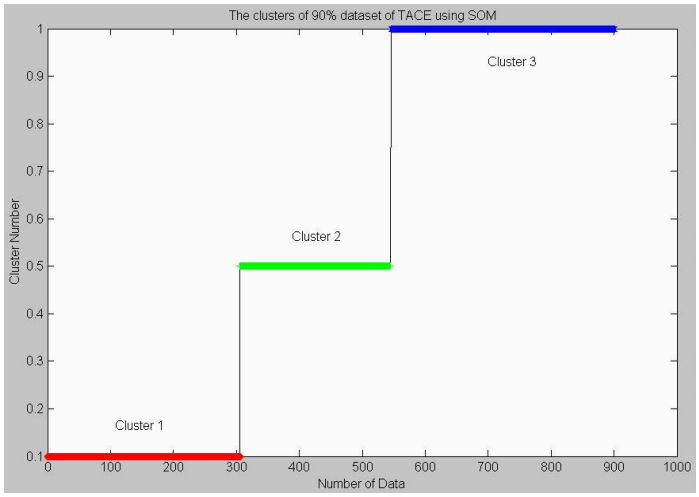
Our master data set comprises of 1000 numbers. To avoid any bias on the data, from the master dataset, we randomly created two sets of training (Dataset A - 90% and Dataset B- 80%) and test data (10% and 20%). In addition to the four input variables (*fuel used, time intercept* *weapon status* and *danger situation*) as illustrated in Table 1, we also used the cluster information generated using SOM algorithm to train the GP models. All the experiments were repeated three times and the average errors are reported.

1.0. *Unsupervised Training of SOM*

The SOM algorithm provides three clusters: C_1, C_2 and C_3. The developed clusters for the two data sets A and B are shown in Figures 3 (a) and (b).



(a)



(b)

Fig. 3. Developed clusters using SOM

2.0. *Learning the Decision Regions*

Parameters used by LGP, MEP and GEP are presented in Tables 2, 3 and 4 respectively. Root Mean Squared Errors (RMSE) values obtained for first and second data sets using LGP, MEP ad GEP are presented in Table 5.

Parameter		**Value**
Population size		50
Mutation frequency		95%
Crossover frequency		95%
Number of demes		10
Program size	Initial	80
	maximum	1024

Table 2. Parameters used by LGP.

Parameter	**Value**
Population size	50
Number of mutations per chromosome	3
Crossover probability	0.8
Code length	40
Number of generations	50
Tournament size	4

Table 3. Parameters used by MEP.

Parameter	Value
Population size	50
Mutation probability	0.044
Crossover probability (one point crossover)	0.3
Number of genes	3
Genes recombination	0.1
Genes transposition	0.1
Inversion	0.1

Table 4. Parameters used by GEP.

RMSE	LGP	MEP	GEP
First data set			
Test	0.09989	0.07225	0.06693
Train	0.05912	0.05930	0.06626
Second data set			
Test	0.056864	0.05927	0.07666
Train	0.057904	0.05725	0.06540

Table 5. RMSE of decision scores using GP models for the test data set

The functions evolved by MEP (combining these variables and also using some constants) are reported below:

- for first data set, the derived function is: var_3 (where var_1, var_2, var_3 var_4 are the variables);
- for second data set, the derived function is: 0.93786 + var_1 - 0.79537 - 0.044892;

Relationship between the number of generations and the value of fitness function (RMSE) for training data obtained by MEP for first data set and second data set are depicted in Figures 4 and 5 respectively. Figures 6 and 7, illustrate LGP models showing the evolution of best training and test fitness values and the average and best code length for first and second data set respectively during 200,000 tournaments.

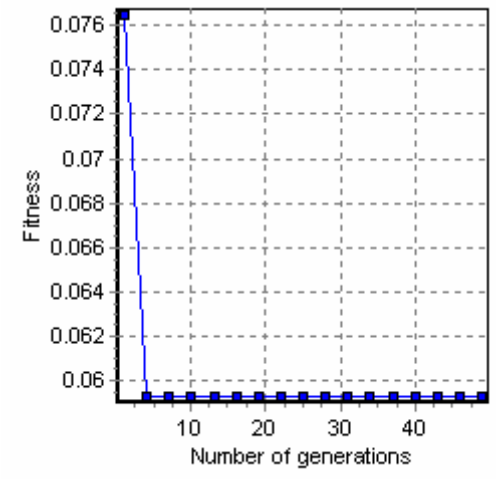


Figure 4. First data set: relationship between fitness function and generations obtained by MEP.

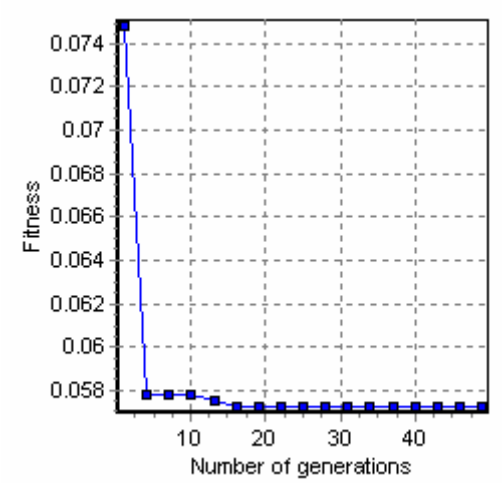


Figure 5. Second data set: relationship between fitness function and generations obtained by MEP.

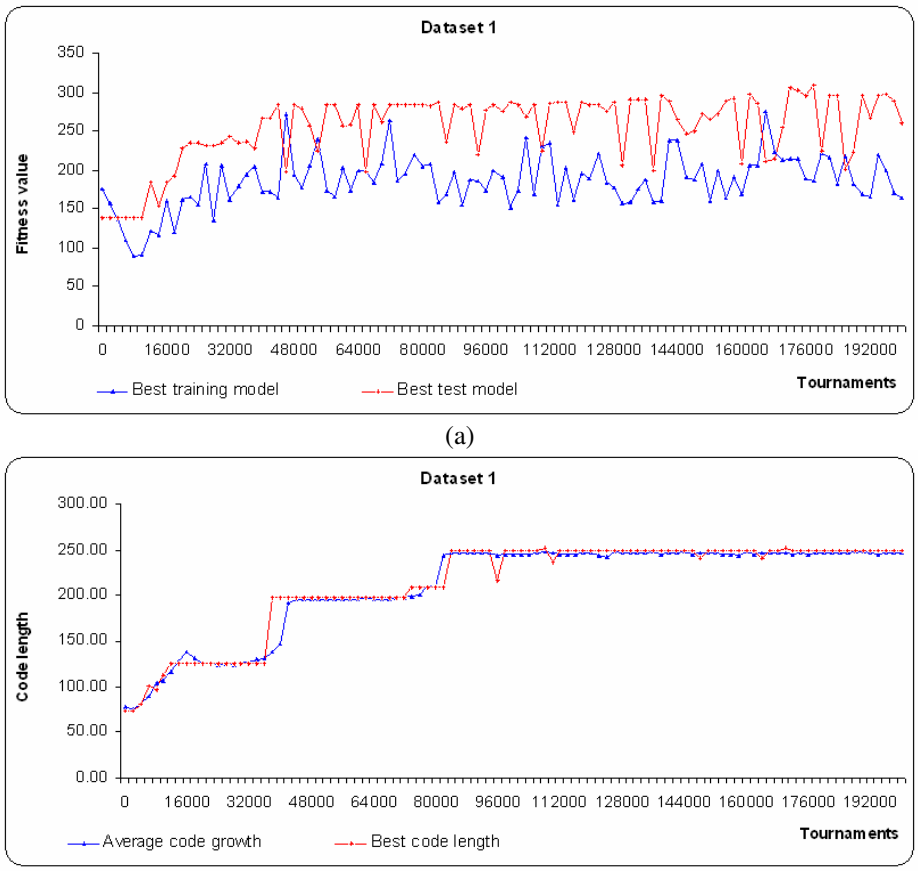


Figure 6 (a). First data set: LGP models showing the evolution of best training and test fitness values during 200,000 tournaments **(b)** average and best code length

(a)

(b)

Figure 7 (a). Second dataset: LGP models showing the evolution of best training and test fitness values during 200,000 tournaments **(b)** average and best code length

5. Conclusion and future research

In this paper, we proposed a hybrid unsupervised-supervised training method to develop a decision support system for a tactical air combat environment. We also investigated the performance of three different genetic programming techniques (Linear Genetic Programming, Multi Expression Programming and Gene Expression Programming) to learn the different decision regions. Two data sets were considered in the experiments. Empirical results reveal that Gene Expression Programming (GEP) outperforms Linear Genetic Programming (LGP) and Multi Expression Programming (MEP) for the first data set and for the second data set LGP obtains the best results. MEP performed well for both data sets while GEP (the best algorithms for first data set) is the worst for second data set and LGP (the worst for first data set) is the best for second data set. Our future research is targeted to investigate and develop how to initiate optimal cluster centers with and without prior knowledge.

References

[1] Cong Tran, Ajith Abraham and Lakhmi Jain, Modeling Decision Support Systems Using Hybrid Neurocomputing, Neurocomputing Journal, Elsevier Science, Netherlands, Volume 61C, pp. 85-97, 2004.

[2] Cong Tran, Lakhmi Jain and Ajith Abraham, Adaptive Database Learning in Decision Support System Using Evolutionary Fuzzy Systems: A Generic Framework, Hybrid Information Systems, Advances in Soft Computing, Physica Verlag, Germany, pp. 237-252, 2001.

[3] Cong Tran, Ajith Abraham and Lakhmi Jain, TACDSS: Adaptation Using a Hybrid Neuro-Fuzzy System, 7th Online World Conference on Soft Computing in Industrial Applications (WSC7), Advances in Soft Computing: Engineering Design and Manufacturing, Benitez, J. M. et al (Eds.), Springer Verlag, Germany, pp. 53-62, 2002.

[4] Cong Tran, Ajith Abraham and Lakhmi Jain, Adaptation of Mamdani Fuzzy Inference System Using Neuro - Genetic Approach for Tactical Air Combat Decision Support System, 15th Australian Joint Conference on Artificial Intelligence (AI'02), LNAI 2557, Springer Verlag, Germany, pp. 672-679, 2002.

[5] Alamo, D. J. L, (1993), A comparison among eight different techniques to achieve an optimum estimation of electrical grounding parameters in two-layered earth, , IEEE Transactions on Power Delivery, Vol. 8, Iss. 4 , pp. 1890 – 1899.

[6] Cattral R., Oppacher F. and Deogo D (1999), Rule acquisition with a genetic algorithm", Proceedings of the congress on Evolution computation, CEC99, Vol. 1, pp. 125-129.

[7] Eom, S.B., (1999), Decision support systems research: current state and trends, Industrial Management and Data Systems, Vol. 99, Iss. 5, pp. 213-221.

[8] Kohonen T., (1982), Self organized formation of topologically correct feature maps, Biological Cybernetics, Vol. 43, pp-59-69.

[9] Leal de Matos, P.A. and Powell, P.L., (2003), Decision Support for Flight Re-routing in Europe, Decision Support Systems 34(4), pp. 397-412.

[10] Moynihan, G.P., Purushothaman, P., McLeod, R.W. and Nichols, W.G., (2002), DSSALM: A decision support system for asset and liability management Decision Support Systems Vol. 33, Iss.1, pp. 23-38.

[11] Nemati, H.R., Steiger, D.M., Iyer, L.S. and Herschel, R.T., (2002), Knowledge Warehouse: an architectural integration of knowledge management, Decision Support, Decision Support Systems, Vol. 33, Issue. 2, pp. 143-161.

[12] Papamichail, K.N. and French, S., (2003), Explaining and justifying the advice of a decision support system: a natural language generation approach, Expert Systems with Applications, Vol. 24, Iss.1, pp. 35-48.

[13] Parker, David B., (1985), "Learning-logic", Report TR-47, Cambridge, MA: Massachusetts Institute of Technology, Center for Computational Research in Economics and Management Science.

[14] M. Oltean, C. Grosan. Evolving Evolutionary Algorithms using Multi Expression Programming, Proceedings of the 7th European Conference on Artificial Life, Dortmund, Germany, pp. 651-658, 2003.

[15] Banzhaf. W., Nordin. P., Keller. E. R., Francone F. D. , Genetic Programming : An Introduction on The Automatic Evolution of Computer Programs and its Applications, Morgan Kaufmann Publishers, Inc., 1998.

[1] Ferreira C., Gene Expression Programming: A new adaptive algorithm for solving problems — Complex Systems, Vol. 13, No. 2, pp. 87–129, 2001.