Crowding Factor Effect to Solve Multiplexer Problems

Lubna Zaglul Bashir

Building and Construction Engineering Dept., University of Technology, Baghdad-Iraq

Received: 1/11/2011

Accepted: 14/2/2013

Abstract **–**Adaptive systems include a vast range of living natural and artificial systems. Reinforcement learning systems are one form of adaptive systems. The current work will focus on a particular kind of reinforcement learning system: the classifier system. A classifier system has the ability to categorize its environment and create rules dynamically, thus making it able to adapt to differing circumstances. This work investigates the effect of crowding factor on the classifier system to solve six-bit and eleven-bit multiplexer problems. The six bit multiplexer problem is defined as six signal lines that come into the multiplexer. The signals on the first two lines (the address or A-lines) are decoded as an assigned binary number. This address value is then used to indicate which of the four remaining signals (on the data or D-lines) is to be passed through the multiplexer output. The eleven bit multiplexer problem is defined as eleven signal lines that come into the multiplexer. The signals on the first three lines (the address or A-lines) are decoded as an assigned binary number. This address value is then used to indicate which of the eight remaining signals (on the data or D-lines) is to be passed through the multiplexer output. This work Investigates the classifier system rule learning with ***no crowding*** and normal ***crowding settings*** by comparing and contrasting the effectiveness of the rule sets learned and their composition in two cases. Experiment results show that the run using classifiers without crowding replacement is unable to perform as well as the run with crowding replacement. The time needed to match the signal is shorter when using classifiers with crowding replacement and we are more likely to achieve good results quickly.

.

Keywords–Reinforcement learning, adaptive system, classifiers, crowding, multiplexer.

1. Introduction

The ability to learn is one of the most fundamental attributes of intelligent behavior. The study and computer modeling of learning processes in their multiple manifestations constitute the subject matter of machine learning. These technologies derived from artificial intelligence are developing turbulent environment of increased specialization and competition, where understanding the human thought process is of prime importance. Learning is a multifaceted phenomenon, which includes the acquisition of new declarative knowledge, the development of motor and cognitive skills and the discovery of new facts and theories through observations and experimentation [1].

Classifier systems are a class of ***rule-based*** message processing systems. Rules are known as classifiers because they are mainly used to classify messages into general sets. Learning in classifier systems is achieved by two mechanisms: **Bucket brigade** and **Genetic Algorithms.** Bucket brigade allocates strength (credit) to the classifiers according to their usefulness in attaining system goals. Genetic Algorithms are used to search for new plausible classifiers [2].

This work investigates the effect of crowding on the Learning Classifier System to solve 6- bit and 11-bit multiplexer problems. Starting from the same population investigate classifier system rule learning with crowding factor values of 1 means ***(no crowding)*** and 3 means ***(normal crowding setting),*** we compare and contrast the effectiveness of the rule sets learned and their composition in two cases.

2. Adaptive Systems

A basic negative feedback loop is the simplest example of intelligent machines, i.e., a machine that appears to function with a definite goal in mind. A representation of a state of an airs that attempts to bring about a basic negative feedback loop has four elements:

a) A sensory signal measuring an environment variable.
b) A reference value.
c) A comparator.
d) An effector signal that alters the environment variable.

The comparator compares the sensory signal with the reference value and computes an effector signal. The effector signal works in such a way that it moves the environment variables closer to the reference value. Such a negative feedback control loop will tend to maintain the variable around the reference value equilibrium. A thermostat is a well known example of this kind of system (temperature = environment variable, dial position = reference value, bi-metallic strip = comparator, turn heating on/ off = effectors signal) [3], [4], [5].

3. A Brief Overview of Classifier Systems

Classifier systems are a form of adaptive systems as a form of domain independent learning system. A classifier is essentially a condition action rule with some associated values or parameters. These are typical estimates of the classifier which is utility to the system. The condition part of a classifier is represented by a ternary bit string composed from the set {0, 1, #}, while the action part is composed from {0, 1}. A classifier system generates, evaluates and

makes use of classifiers for decision making in interaction with some problem environment. These functions are carried out by three major subsystems, the performance system, the learning or credit assignment system, and the rule discovery system. Each classifier has a strength parameter which represents the system's evaluation of the utility of that classifier. In classifier systems, the strength parameter is used as a measure of utility in both performance and rule discovery [1], [6], [7], [8]. The learning classifier system is illustrated in Fig.1 [2].

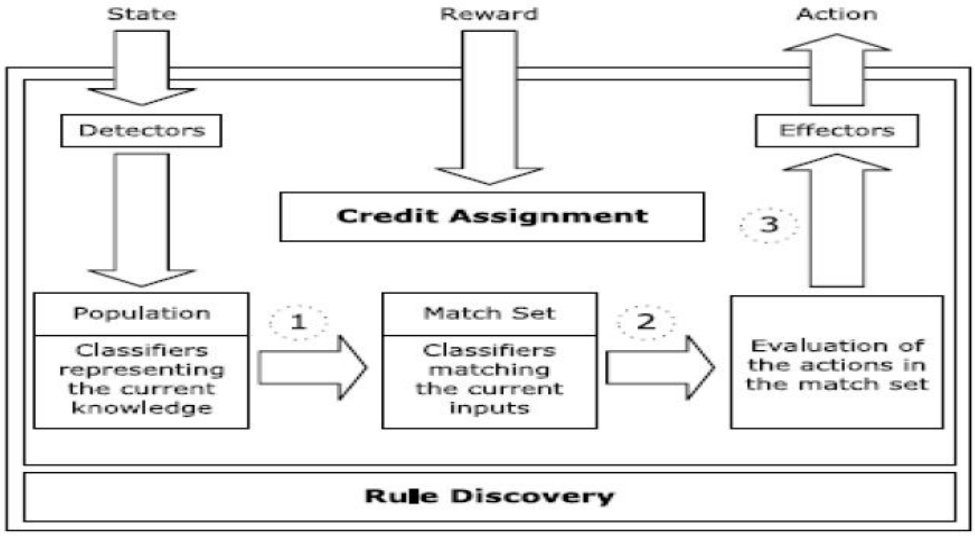


Figure1. Simplified architecture of learning classifier system

3.1. The Performance System (Rules and messages system)

The performance system is composed of four parts:

- Input Interface Generates messages in the form of binary bit strings representing detected environmental features.
- Message List Stores messages generated by the input interface and by classifiers.
- Classifier List A list of classifiers which are applied to the message list and which may add messages to it.
- Output Interface Translates action messages from the message list into actions in the environment [5], [6], [8], [9].

3.2. The Learning System

The performance system itself is not adaptive so the classifier system must include some form of learning system. In the traditional classifier system the bucket brigade algorithm is used to allow the system to perform credit assignment with the classifiers. The credit assignment problem is the problem of determining which components of a complex system are responsible for producing which outcomes the bucket brigade operates by adjusting the strength parameter of classifiers in response to the reward or payoff the system receives as a result of its actions. Classifier's strength is thus a prediction of the reward the system will

receive if it selects that classifier from the set of classifiers matching the current messages and allows it to post its message. Another way at looking at strength is a measure of how useful the classifier has been in the past [10].

To allow selective activation of some classifiers satisfied i.e. matched, during a major cycle an element of competition is introduced. Each classifier that has its condition part satisfied makes a bid to become active. Only the highest bidders are allowed to post their messages. The bid of a classifier depends on two factors:

- Its strength, which is a measure of the classifiers usefulness to the system. The strength of a classifier should be modified as a result of the systems experience with a particular task domain.

- Its specificity, the number of 1's and 0's in its condition part. Specificity can be thought of as a measure of a classifiers relevance to a particular set of messages.

  A classifiers bid is

  B = k * strength* specificity

k is a universal constant for all classifiers. The higher classifiers strength usefulness, the higher its bids and the more likely it will win the competition and post messages. The same applies to specificity.

Consequently the behavior of a classifier system can be modified by changing the strengths associated with classifiers. If the strength of classifiers that tend to lead to useful behavior can be increased and the strength of classifiers that tend to lead to useless behavior can be decreased, the system will learn to produce a more useful behavior. The bucket brigade algorithm is designed to bring about these types of changes in strength [5].

The basis for the bucket brigade is information from the environment about whether or not the classifier system as a whole is behaving correctly. This is achieved via rewards. The system receives positive reward from the environment when it produces the right behavior and negative reward when it produces incorrect behavior. In some situations, neither positive nor negative reward will be received, or it may be received some time in the future, and it may be contingent upon other factors, hence the credit assignment problem. The bucket brigade acts in two ways:

- When a reward is received the bucket brigade adds the reward value to the strength of all classifiers active during the major cycle. In other words, the algorithm changes the strength of all classifiers that were directly associated in time with the receipt of the reward. The mapping from classifier system state to reward is called the payoff function.

When a classifier is activated it pays the amount it bid to the classifiers that made it possible to become Active, i.e., those which posted the messages which matched the classifiers condition. Consequently, the strength of the active classifiers is decreased by the amount of its bid. In this way the bucket brigade also acts to increase the strength of classifiers that indirectly lead to useful behavior, i.e., rewards. The bucket brigade allows rewards to circulate back to antecedent classifiers that produce rewarding behaviors [11], [12].

3.3. The Rule Discovery System

A complete classifier system needs some means of generating new rules for use in the performance and learning systems. Well known genetic algorithm (GA) techniques have been used as the main source of rule discovery in CS. Genetic algorithms were inspired by natural selection and operate by evolving generations of individuals which are successively more than according to some fitness evaluation function. In traditional classifier systems, classifiers strength is taken as a measure of its fitness or utility in rule discovery, in addition to its use in the performance system. The basic operation of a genetic algorithm is summarized as follows:

- Select classifiers for reproduction. The probability of a classifier being selected as a parent is based on its strength.
- Apply genetic operators to the new classifiers. Copies of the parent classifiers are generated and transformed using genetic operators. The most commonly used operators are crossover, which combine elements of the bit strings of two parents as in sexual reproduction, and mutation which is a change effected probabilistically on some part of the bit string.

- Select classifiers for deletion in order for the population of classifiers to remain within some reasonable size limit. Existing classifiers must be deleted as new ones are introduced. There are various means of selecting classifiers for deletion, for example probabilistically based on an inverse function of the classifiers strength .i.e. weaker classifiers are more likely to be deleted [6], [12], [13].

4. The Six-bit Multiplexer Problem

Multiplexer functions are used as reinforcement learning problems for classifier and other reinforcement learning systems using the basic form of reinforcement learning, Classifier systems learn the multiplexer problems by generating evaluating and basing decisions on classifiers. Boolean multiplexer problems are highly nonlinear logical functions commonly used for testing machine learning systems. They provide a series of related problems of increasing difficulty which makes them suitable for evaluating the ability of a system to scale up, Further they are useful as a common measure of performance for different systems as they have been used with a variety of systems.

The six bit multiplexer problem is defined as follows: six signal lines come into the multiplexer. The signals on the first two lines (the address or A-lines) are decoded as an assigned binary integer. This address value is then used to indicate which of the four remaining signals (on the data or D-lines) is to be passed through the multiplexer output. Boolean 6- bit multiplexer functions are defined for binary strings of length $l = k + 2^k$. The function's value may be determined by treating the first k bits as an address that indexes into the remaining 2^kbits, and returning the indexed bit. For example, in the 6-multiplexer (l=6), the value for the input string 100011 is 1, since the "address", 11, indexes bit 3 of the remaining four bits. In disjunctive normal form, the 6-multiplexer is fairly complicated:

$$F_6 = a_0' a_1' d_0 + a_0' a_1 d_1 + a_0 a_1' d_2 + a_0 a_1 d_3 \quad (1)$$

Multiplication is a Boolean AND operation, addition is a Boolean OR operation and the prime is a Boolean NOT

operation. There are exactly eight classifiers that would give the right answer for the example string above. The most specific is 100011:1 and the most general is 1###11:1 (the other six replace one or more of the #s in the latter by 0s). The 1###11:1 is correct for all (eight) inputs it can match; in fact, it is maximally general in the sense that no further #s can be added to its condition without producing an error. The 64-string input space can be covered by exactly eight such maximally general classifiers, each having three #s in its condition so it matches eight strings. They are:

###000:0
##0#01:0
#0##10:0
0###11:0
###100:1
##1#01:1
#1##10:1
1###11:1

In the simple classifier system, the classifier system is presented with a randomly generated sequence of example signals. The classifier system then learns to emulate the 6-multiplexer. To do this, the system repeatedly responds to deferent signals, receiving or not receiving reward as it gives or does not give the correct answer. In this manner the apportionment of credit algorithm rewards existing rules depending on their effectiveness. Thereafter the genetic algorithm injects new rules to improve system performance [14].

The multiplexer problem is single-*step* in that external reward was received on every time-step and the environmental input for each time-step was completely independent of that for the prior time - step. Problems involving categorization of data examples are typically single-step, since a decision is made and reinforcement as to the quality of the decision is received, in a single time-step, and the examples to be categorized are usually independent. Fig.2 illustrates the 6-bit multiplexer function.

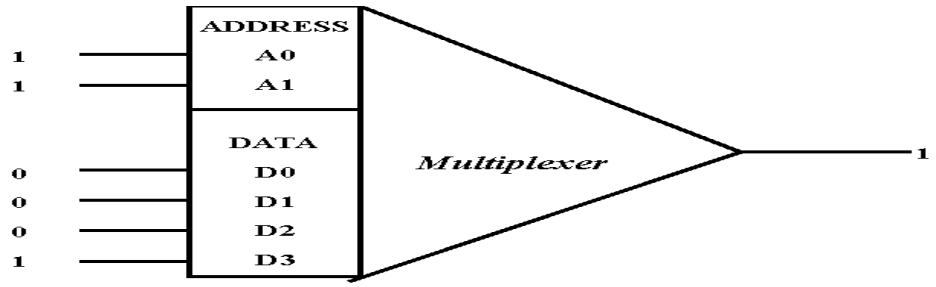


Figure 2. The six- bit multiplexer problem

5. The Eleven -bit Multiplexer Problem:

The eleven bit multiplexer problem is defined as follows: eleven signal lines come into the multiplexer. The signals on the first three lines (the address or A-lines) are decoded as an assigned binary integer. This address value is then used to indicate which of the eight remaining signals (on the data or D-lines) is to be passed through the multiplexer output. Boolean 11-bit multiplexer functions are defined for binary strings of length $l = k + 2^k$. The function's value may be

determined by treating the first *k* bits as an address that indexes into the remaining 2^kbits, and returning the indexed bit. For example, in the 11-multiplexer (*l*=11), the value for the input string 10000000111 is 1, since the "address", 111, indexes bit 7 of the remaining eight bits. In disjunctive normal form, the 11-multiplexer is complicated:

$$F_{11} = a_0a_1a_2d_0 + a_0a_1a_2d_1 + a_0a_1a_2d_2 + a_0a_1a_2d_3 + a_0a_1a_2d_4 + a_0a_1a_2d_5 + a_0a_1a_2d_6 + a_0a_1a_2d_7 \quad (2)$$

Where multiplication is a Boolean AND operation, addition is a Boolean OR operation and the prime is a Boolean NOT operation. There are exactly 16 classifiers that would give the right answer for the example string above. The most specific is 10000000111:1 and the most general is 1#######111:1 (the other 128 replace one or more of the #s in the latter by 0s). The 1#######111:1 is correct for all (128) inputs it can match; in fact, it is maximally general in the sense that no further #s can be added to its condition without producing an error. The 2048-string input space can be covered by exactly 16 such maximally general classifiers, each having three #s in its condition so it matches eight strings. They are:

#######0000:0
######0#001:0
#####0##010:0
####0###011:0
###0####100:0
##0#####101:0
#0######110:0
0#######111:0
#######1000:1
######1#001:1
#####1##010:1
####1###011:1
###1####100:1
##1#####101:1
#1######110:1
1#######111:1

In the simple classifier system, the classifier system is presented with a randomly generated sequence of example signals. The classifier system then learns to emulate the 11-multiplexer. To do this, the system repeatedly responds to deferent signals, receiving or not receiving reward as it gives or does not give the correct answer. In this manner the apportionment of credit algorithm rewards existing rules depending on their effectiveness. Thereafter the genetic algorithm injects new rules to improve system performance [14].

The multiplexer problem is single-*step* in that external reward was received on every time-step and the environmental input for each time-step was completely independent of that for the prior time-step. Problems involving categorization of data examples are typically single-step, since a decision is made and reinforcement as to the quality of the decision is received, in a single time-step, and the examples to be categorized are usually independent. Fig.3 illustrates the 11-bit multiplexer function [3].

6. The Rules and Messages System for (Multiplexer) Problem

As the Rules and Messages (performance) system is the heart of the classifier system, the matching procedures are the heart of the Rules and Messages system. The Rules and Messages system of the Multiplexer consists of a message list and classifier store. There is only a single message in the message list that is used to match against condition part of all classifiers in the classifier store and there is no message to be received in the current cycle until the system produces an action. The two routines are responsible for matching classifiers to the environment message: match and match classifiers.

The function match performs a match between a single condition and a single message and returns a Boolean true value if match succeeds. The function match is illustrated in Fig.4.

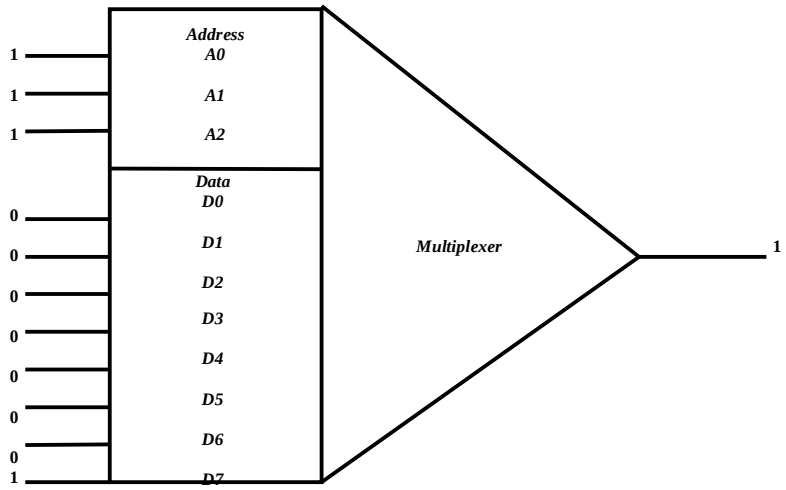


Figure.3 The Eleven - bit multiplexer problem

```
Function match;
{match single condition to single message}
begin
      matchtemp:=true;
      while(matchtemp=true) and(nposition>0) do
      begin
           matchtemp:=(condition[nposition]=wildcard) or          (condition[nposition]=message[nposition]);
           nposition:= nposition-1
       end;
  match:=matchtemp
end;
```

Figure 4. The Function Match

The procedure match classifiers match all classifiers against the environment message and construct the match list data structure. Fig.5 illustrates the procedure match classifiers.

```
Procedure match classifiers;
{ match all classifiers against environmental message and create match list}
begin with population do with match list do
       begin
          nactive:=0;
          for j:= 1 to nclassifier do with classifier [j] do
           begin
     matchflage:=match(condition,environmentmessage,nposition);
              if matchflag then
                  begin
                     nactive := nactive+1;
                     conditionlist[nactive]:=j
                  end
           end;
end   end;
```

Figure5. The Procedure Match Classifiers

7. Rule Discovery for (Multiplexer) Problem

GA is a way of injecting new possibly better rules into the system (Multiplexer). New rules are created by the rule discovery process, ***Reproduction, crossover, and mutation.*** These rules are then placed in the population and processed by the auction, payment and reinforcement learning to properly evaluate their role in the system.

- In the (Multiplexer) Problem a quantity called the selection proportion was used, where that proportion of the population is replaced at a given genetic algorithm invocation.

The number of mate pairs to select =proportion select*n classifier*0.5 (3)

- In the classic implementation of Classifier Systems, the genetic algorithm (i.e. reproduction, crossover and mutation) is called every T_{ga} cycles where T_{ga} is a constant whose optimal value is experimentally determined. The invocation of genetic algorithm learning may be conditioned on particular events such as lack of a match or poor performance. the basic execution cycle (the central loop) in Fig. 6 is as follows:

```
Procedure GA;
{Coordinate selection, mating, crossover, mutation, & replacement}
Begin with population do
    begin
      Statistics (population);      {get average, max,min, sumstrength}
      For j:=1 to nselect do with mating[j] do
          begin
              mate1:=select(population);          {pick mates}
              mate2:=select(population);
Crossover(classifier[mate1],classifier[mate2],child1,child2,
pcrossover,pmutation,sitecross,nposition,ncrossover,nmutation)
mort1:=crowding(child1,population,crawdingfactor,crawdingsubpop);
sumstrength:=sumstrength-classifier[mort1].strength+child1.strength
classifier[mort1]:=child1;
mort2:=crowding(child2,population,crawdingfactor,crawdingsubpop);
sumstrength:=sumstrength-classifier[mort2].strength+child2.strength
classifier[mort2]:=child2;
          end;
end   end;
```

Figure 6. The genetic procedure in LCS

- The selection process for (multiplexer) is performed using ***roulette wheel*** selection where each classifier's strength value S is used as its fitness. The sum of the population fitness is calculated .The expected value of an individual is the individual's fitness divided by the average fitness of the population. Roulette wheel selection algorithm is illustrated in Fig.7.
- The crossover operator is implemented by taking a two-parent string and generating two offspring strings. In (Multiplexer) problem crossover is implemented as follows: an integer position k along the string is selected uniformly at random between 1 and the string lengthless one [1, l-1].

Two new strings are created by swapping characters between position k+1 and l inclusively. Fig.8 illustrates Procedure Crossover.

```
Function roulette wheel;
 Begin
      P=0;
      J=0;
      Rand = Random * Sum of Strength;
      Repeat
            J=J+1;
            P = P + classifier [j]. Strength;
      Until ((P>Rand) OR (J = n classifier));
      Select j;
```

Figure.7: Roulette Wheel Selection Algorithm.

```
    Procedure crossover;
    Begin
          If flip (Pcrossover) then
             begin
             Sitecross: = rnd (1,nposition);
              ncrossover:=ncrossover+1;
             end
       else site cross: = nposition+1    {transfer but no cross}
       {transfer action part regardless of sitecross}
       Child1.action: = Parent1.action;
       Chiled2.action: = Parent2.action;
       {transfer and cross above cross site}
        j:= site cross;
       While (j<= n position ) do  begin
                Child2.condition [j]: = Parent1.condition [j];
                Child1.condition [j]: = Parent2.condition [j];
                j:=j+1
        end;
       j:=1;
       {Transfer only below cross-site}
       While (j< site cross) do begin
                Child1.condition [j]: = Parent1.condition [j];
                Child2.condition [j]: = Parent2.condition [j];
                j:=j+1
    end;
    end;
```

Fig.8: Procedure Crossover

- When a mutation is called it changes the mutated {0,1,#} character to one of other two with equal probability**.** The mutation algorithm is illustrated in Fig.9.

- **Selection of next Generation**

Now we are searching, not for the single best rule (classifier), but for a well-adapted set of rules. Therefore we use the "crowding replacement" algorithm to choose the classifiers that should die to make room for new offspring. (This implies combining the best of the parents and offspring.) Crowding replacement aims to replace a low performing classifier with a similar (potentially better classifier) [15], [16]. The crowding algorithm is illustrated in Fig.10.

```
for i= 1 to crowding factor do
        x:=find worst of a random set
        if this is not more similar to offspring then paste x
        set worst more similar to x
        end if
end for
Replace worst most similar with offspring
```

Figure.10. Crowding algorithm

8. Experimental Results

we examined the simple classifier system performance on the 6- bit multiplexer and 11-bit multiplexer problems, and performed two Simple classifier systems simulations, one with genetic algorithm using crowding replacement and one with the genetic algorithm without using crowding replacement. In both cases, we started from the same set of default hierarchical rules (general rules (those with many #'s) cover the general conditions and more specific, cover the exception).

- To achieve the crowding replacement by setting the following values: Crowding factor=3 /Crowding subpopulation=3
- To inhibit the crowding replacement, by setting the following values: Crowding factor=1 / Crowding subpopulation=1

The run of 6- bit and 11- bit Multiplexers using GA without crowding replacement is unable to perform as well as the run with crowding replacement. The time need to match a signal is shorter when using GA with crowding replacement and it is more likely to achieve good results quickly. When executing the 6-bit Multiplexer code, the system responds by presenting the initial report display in Fig.11 (a) the classifier system runs for 1000 iterations, terminating with the snapshot report display in Fig.11 (b). Executing the 11-bit multiplexer code, the system responds by presenting the initial report display in Fig.12 (a). The classifier system runs for 1000 iterations, terminating with the snapshot report display in Fig.12 (b). The performance of the system in the two cases is illustrated in Fig.13.

9. Conclusions

1. Crowding replacement aims to replace a low performing classifier with a better classifier**.**

2. The run using CS without crowding replacement is unable to perform as well as the run using CS with crowding replacement.

3. The learning rate of a serial implementation of a system is slow, because of increasing the number of similar classifiers on the population. This led to the fact that the performance of the system continues to decrease.

4. The time needed for convergence is shorter when using CS with crowding replacement and it is more likely to achieve good results quickly.

5. The number of iterations for 6-bit multiplexer when using crowding factor =300 cycles, whereas the number of iterations without crowding factor =850 cycles. The Number of iterations for 11-bit

multiplexer when using crowding factor =600 cycles, whereas the number of iterations without crowding factor =1000 cycles.

6. For 6-bit multiplexer with crowding factor the win rule is 111010:1 whereas the win rule is 1###10:0 without crowding factor. For 11-bit multiplexer with crowding factor the win rule is 1##0#1#0101:0:1 whereas the win rule is 1#######101:0 without crowding factor.

7. The system was able to learn rules for the given task using only a few training examples and starting with classifiers that were randomly generated.

```
---------------------------------------
A  6-bit Multiplexer Problem
---------------------------------------
population parameters
----------------------------
number of classifiers     =       10
number of positions       =        6
bid coefficient           = 0.1000
bid spread                = 0.0750
bidding tax               = 0.0100
existence tax             = 0.0000
generality probability    = 0.5000
bid specificity base      = 0.2500
bid specificity mult.     = 0.1250
edid specificity base     = 0.2500
ebid specificity mult.    = 0.1250
environmental parameters (multiplexer)
---------------------------------------
number of address lines =       2
number of data lines      =       4
total number of lines     =       6
apportionment of credit parameters
----------------------------------
bucket brigade flag   =     false

reinforcement parameters
------------------------
 reinforcement reward   =    10.0
Timekeeper parameters
---------------------
Initial iteration              =       0
Initial block                  =       0
Report period                  =      50
Console report period          =      50
Plot report period             =      50
Genetic algorithm period       =       5
Genetic Algorithm Parameters
----------------------------
Proportion to select/gen  = 0.8000
Number to select          =       4
Mutation probability      = 0.0200
Crossover probability     = 0.8000
  Crowding factor         =       3
  Crowding subpopulation  =       3
  [block: iteration]  -  [0:0]
  current 6-bit multiplexer status
  ------------------------------------
  signal                  =  000000
  decoded address         =       0
  multiplexer output      =       0
  classifier output       =       0
  environmental message:      000000
  no.     strength       bid      ebid   M  classifier
  ------------------------------------------------------------
    1     10.00      0.00      0.00       ###000:[0]
    2     10.00      0.00      0.00       ###100:[1]
    3     10.00      0.00      0.00       ##0#01:[0]
    4     10.00      0.00      0.00       ##1#01:[1]
    5     10.00      0.00      0.00       #0##10:[0]
    6     10.00      0.00      0.00       #1##10:[1]
    7     10.00      0.00      0.00       0###11:[0]
    8     10.00      0.00      0.00       1###11:[1]
    9     10.00      0.00      0.00       ######:[0]
   10     10.00      0.00      0.00       ######:[1]
  new winner[1] : old winner[1]
  [block: iteration]  -  [0:350]
  current 6-bit multiplexer status
  ------------------------------------
  signal                  =  111010
  decoded address         =       2
  multiplexer output      =       1
  classifier output       =       1
  environmental message:      111010

    no.      strength       bid      ebid      M  classifier
-------------------------------------------------------------------
    1   2030.61      0.00      0.00          111110:[1]
    2   1998.59      0.00      0.00          111110:[1]
    3   1554.69    157.04    156.96   x      111010:[1]
    4   2367.09      0.00      0.00          1101#1:[1]
    5   1534.79    171.32    171.32   x      111010:[1]
    6   2181.23      0.00      0.00          1101#0:[1]
    7   1970.77      0.00      0.00          110101:[1]
    8   2181.23      0.00      0.00          111111:[1]
    9   2182.84      0.00      0.00          1101#1:[1]
   10   1995.38      0.00      0.00          1#111#:[1]
  new winner[5] : old winner[3]
```

Fig.11.(a) : 6- bit Multiplexer with crowding factor

[block: iteration] - [0:350]
current multiplexer status

signal = 111010
decoded address = 2
multiplexer output = 1
classifier output = 1
environmental message: 111010

no.	strength	bid	ebid	M	classifier
1	174.42	0.00	0.00		###110:[1]
2	164.77	0.00	0.00		###110:[1]
3	158.74	0.00	0.00		###110:[1]
4	168.39	0.00	0.00		###100:[0]
5	174.42	0.00	0.00		###110:[1]
6	174.42	0.00	0.00		###110:[1]
7	176.27	0.00	0.00		###100:[1]
8	174.42	0.00	0.00		###110:[1]
9	147.52	9.31	9.34	x	1###10:[1]
10	**156.35**	**9.86**	**9.80**	**x**	**1###10:[1]**

new winner[10] : old winner[10]

Figure 11.(b) .6- bit Multiplexer without crowding factor

A 11-bit Multiplexer Problem

population parameters

number of classifiers = 18
number of positions = 11
bid coefficient = 0.1000
bid spread = 0.0750
bidding tax = 0.0100
existence tax = 0.0000
generality probability = 0.5000
bid specificity base = 0.2500
bid specificity mult. = 0.1250
edid specificity base = 0.2500
ebid specificity mult. = 0.1250
environmental parameters (multiplexer)

number of address lines = 3
number of data lines = 8
total number of lines = 11

apportionment of credit parameters

bucket brigade flag = false

reinforcement parameters

reinforcement reward = 10.0
Timekeeper parameters

Initial iteration = 0
Initial block = 0
Report period = 50
Console report period = 50
Plot report period = 50
Genetic algorithm period = 5

Genetic Algorithm Parameters

Proportion to select/gen = 0.8000
Number to select = 7
Mutation probability = 0.0200
Crossover probability = 0.8000
Crowding factor = 3
Crowding subpopulation = 3

[block: iteration] - [0:0]
current 11- bit multiplexer status

signal = 00000000000
decoded address = 0
multiplexer output = 0
classifier output = 0
environmental message: 00000000000

no.	strength	bid	ebid	M	classifier
1	100.00	0.00	0.00		######0000:[0]
2	100.00	0.00	0.00		######1000:[1]
3	100.00	0.00	0.00		#####0#001:[0]
4	100.00	0.00	0.00		#####1#001:[1]
5	100.00	0.00	0.00		####0##010:[0]
6	100.00	0.00	0.00		####1##010:[1]
7	100.00	0.00	0.00		###0###011:[0]
8	100.00	0.00	0.00		###1###011:[1]
9	100.00	0.00	0.00		##0####100:[0]
10	100.00	0.00	0.00		##1####100:[1]
11	100.00	0.00	0.00		#0#####101:[0]
12	100.00	0.00	0.00		#1#####101:[1]
13	100.00	0.00	0.00		0######110:[0]
14	100.00	0.00	0.00		1######110:[1]
15	100.00	0.00	0.00		0######111:[0]
16	100.00	0.00	0.00		1######111:[1]
17	100.00	0.00	0.00		##########:[0]
18	100.00	0.00	0.00		##########:[1]

new winner[1] : old winner[1]

[block: iteration] - [0:50]
current 11-bit multiplexer status

signal = 11001100101
decoded address = 5
multiplexer output = 0
classifier output = 0
environmental message: 11001100101

no.	strength	bid	ebid	M	classifier
1	106.50	0.00	0.00		###0####110:[0]
2	106.61	0.00	0.00		##0##01#1#0:[1]
3	106.61	0.00	0.00		##0##01#1#0:[1]
4	104.84	0.00	0.00		#101###0101:[0]
5	106.52	0.00	0.00		##0#####110:[0]
6	103.39	0.00	0.00		00####1#1#0:[1]
7	106.55	0.00	0.00		##01#01#1#0:[1]
8	105.80	0.00	0.00		1##0####110:[0]
9	**87.94**	**9.99**	**10.06**	**x**	**1##0#1#0101:[0]**
10	105.06	0.00	0.00		1##0#1#0100:[1]
11	105.83	0.00	0.00		##00####1#0:[0]
12	108.11	0.00	0.00		##0####0100:[0]
13	106.50	0.00	0.00		##01####110:[0]
14	106.61	0.00	0.00		1##0###0100:[0]
15	105.80	0.00	0.00		###0##1#1#0:[0]
16	106.55	0.00	0.00		##0#####110:[0]
17	106.52	0.00	0.00		###0####110:[0]
18	106.61	0.00	0.00		##0##01#1#1:[1]

new winner[9] : old winner[9]

Figure 12.(a). 11- bit Multiplexer with crowding factor

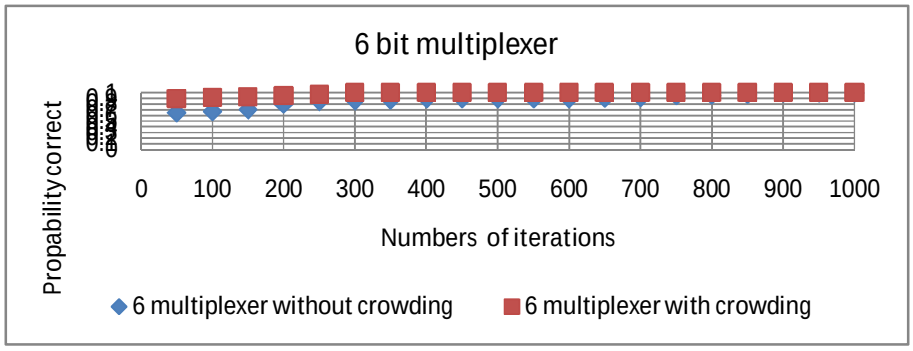
[block: iteration] - [0:200]
current 11-bit multiplexer status

signal = 11001100101
decoded address = 5
multiplexer output = 0
classifier output = 0
environmental message: 11001100101

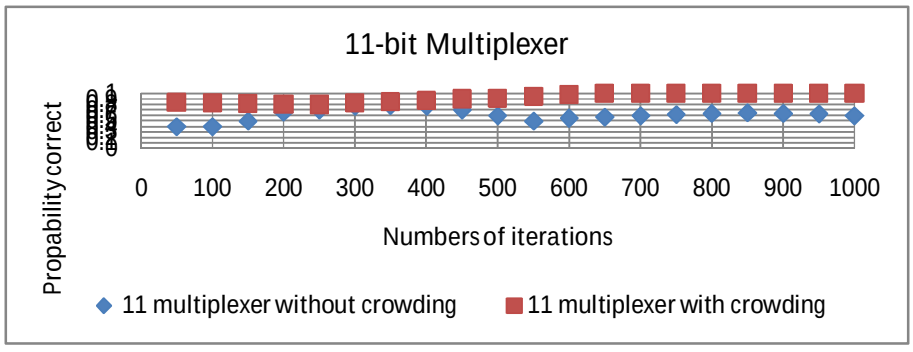
no.	strength	bid	ebid	M	classifier
1	87.40	0.00	0.00		110####0111:[0]
2	94.92	0.00	0.00		1#1#####101:[0]
3	94.92	0.00	0.00		1#0#10#1101:[1]
4	99.03	0.00	0.00		1#0#0##0101:[0]
5	79.76	8.06	8.02	x	110#####101:[1]
6	98.85	0.00	0.00		1#0##0##101:[0]
7	92.91	0.00	0.00		1#0##0##101:[0]
8	93.27	0.00	0.00		1#0##010101:[0]
9	98.36	0.00	0.00		1#0#10#1101:[1]
10	101.79	0.00	0.00		1#1#####101:[1]
11	98.11	0.00	0.00		1#1##0##101:[1]
12	**117.31**	**8.80**	**8.75**	**x**	**1#######101:[0]**
13	99.73	0.00	0.00		1#1#####101:[1]
14	90.93	0.00	0.00		1#1##0#0101:[0]
15	99.73	0.00	0.00		1#0#10#1101:[1]
16	87.40	0.00	0.00		1#1##0##101:[1]
17	104.55	0.00	0.00		1#1#####101:[1]
18	98.36	0.00	0.00		1#0#0##0101:[0]

new winner[12] : old winner[12]

Figure 12.(b) . 11- bit Multiplexer without crowding factor



(a)



(b)

Figure 13. (a) 6 – bit Multiplexer ; .(b) 11 – bit Multiplexer

References

[1] A.Yosuke, S.Yuji, "Reward Allotment Considered Roles for Learning Classifier System For Soccer Video Games" Hosei University, JAPAN,2009

[2] A. Kharmandar, A. Naeimi, A. M. Alizadeh, S. Jafari, S. Chavoshi", Soccer Simulation 2DTeam Description Proposal for Robocup, ,Payame Noor University, Iran,2011.

[3] B. Jaume, E. B. Mansilla, and M. V. Butz, "Learning Classifier Systems: Looking Back and Glimpsing Ahead" ASAP research group, School of Computer Science, Jubilee Campus, University of Nottingham, UK,2008.

[4] Kr. Amit, "Artificial intelligence and soft computing" behavioral and cognitive modeling of the human brain Book, 1998.

[5] R.J. Urbanowicz and J. H. Moore, "Learning Classifier Systems: A Complete Introduction, Review, and Roadmap" *Department of Genetics, Dartmouth College, Hanover, NH 03755, USA* Correspondence should be addressed to Jason H. Moore, jason.h.moore@dartmouth.edu, 2009.

[6] B. Jason, **"**Learning Classifier Systems**",** Technical Report 070514A,Complex Intelligent Systems Laboratory, Centre for Information Technology Research, Faculty of Information and Communication Technologies, Swinburne University of Technology Melbourne, Australiajbrownlee@ict.swin.edu.au, 2007.

[7] L. Samuel, S. Olivier, "A Comparison between AT NoSFERES and Learning Classifier Systems on non-Markov problems" Institute des Systems Intelligent et de Robotique, CNRS FRE 2507Université Pierre et Marie Curie - Paris 64 place JussieuF-75 252 Paris Cedex 05FranceAb, 2008.

[8] Z. Qing and P. Martin, "A Market-Based Rule Learning System" a Guang Dong Data Communication Bureau China Telecom 1 Dongyuanheng Rd., Yuexiunan, Guangzhou 510110, China, Department of Information Science, University of Otago, PO Box 56, Dunedin, New Zealand and/or improving the comprehensibility of the rules, 2004.

[9] N. Bhatia, P. Dixit, M. Sood, "GAMBLE: genetic algorithm based machine learning expert" Indian Institute of Technology, 2000.

[10] E. Anders, "Evolution of Meta-parameters in Reinforcement Learning" Master's Thesis in Computer Science, at the School of Computer Science and Engineering, Royal Institute of Technology, Stockholm, Sweden, 2002.

[11] H. Joh, "Learning classifier systems" Jay Dee Technology Ltd, 2002.

[12] Jn. Troels, "Classifier System Abstracts" Aarhus school of business, Denmark, 2004.

[13] M. Melanie, "An introduction to genetic algorithm" MIT Press, London England, book, 1998.

[14] W. Stewart, "Classifier Fitness Based on Accuracy" The Rowland Institute for Science 100 Edwin H. Land Blvd. Cambridge, MA 02142 (617) 497-4650 wilson@smith.rowland.org Submitted to Evolutionary Computation, To appear in Vol. 3, 1995.

[15] B. Larry, "Learning Classifier Systems: A Brief Introduction", Faculty of Computing, Engineering & Mathematical Sciences University of the West of England, Bristol BS16 1QY, U.K. Larry, 2004.

[16] R. E. Smith , M. K. Jiang, J. Bacardit , M. Stout , N. Krasnogor, J.D. Hirst," A learning classifier system with mutual-information-based fitness", UK Engineering and Physical Sciences Research Council (EPSRC), 2010.