Genetic Algorithms for Protocol Validation

Enrique Alba and José M. Troya

Dpto. de Lenguajes y Ciencias de la Computación, Univ. de Málaga
Campus de Teatinos (2.2.A.6), 29071- MÁLAGA, España
{eat, troya}@lcc.uma.es

Abstract. We present a first attempt in applying a genetic algorithm for checking the correctness of communication protocols (expressed as a pair of communicating FSMs). The GA measures the fitness of a given string by making use of a protocol simulator. Every string in the population is a trace of the execution of the protocol. The simulator evaluates the trace by running it, provoking changes and messages exchanges in the states of every FSM. The fitness of a string (trace) is high if the string detects a deadlock or if useless states or transitions are encountered. We are interested in testing the suitness of the GA search in such a domain. We have tested this genetic validation on a hand-made protocol and on the Transmission Control Protocol (TCP).

1 Introduction

The validation of a newly developed protocol is a major issue in the field of communication protocols. The existing work in this field always begins with a first phase of unambiguous specification of the protocol with a formal technique: Finite State Machines (FSMs), Petri Nets or specification languages as DRL[4], LOTOS[7] or STELLE[8] in order to facilitate the second phase, namely validation. Since the space of all possible system states is very large, recent works are shifting to the application of heuristics as an alternative to exhaustive or other existing search procedures. The applications of GAs [3][5] on different domains [1][2] are growing. In this work we want to study how a GA (GAVOR v3.0) can validate protocols.

In order to reach this goal we must define the genotype representation for any partial solution and the kind of fitness computations for characterising it. The magnitude of the search for a medium sized protocol is very high. When a communicating entity resides in a given state of its FSM, the set of possible states the other entity could reside, the contents of its (remote) messages queue and the local messages queue conform a very large search space. We are not evolving FSMs as in [6], but instead we are validating the protocol they represent.

In the design of truly working protocols (videoconferencing, multimedia, etc.) it is very important the validation of the protocol, that is, the formal warranty that the protocol does not hang up in deadlock, no messages between the PPEs are ignored or lost, and that all the devised states and transitions of the FSMs are useful. We want to complain with all these goals with the exception of lost messages (for the moment). It is very frequent that the protocol can be modelled as a FSM that is executed by the client and the server nodes. Stated in a more formal way, the communicating Peer Protocol Entities (PPEs) residing in a given level of the hierarchy of layers being developed are exchanging messages. This exchange is driven by the user actions or by the actual FSM state and input message. Of course every PPE can be in a different state of its underlying FSM. Fig. 1 depicts the whole scenario.

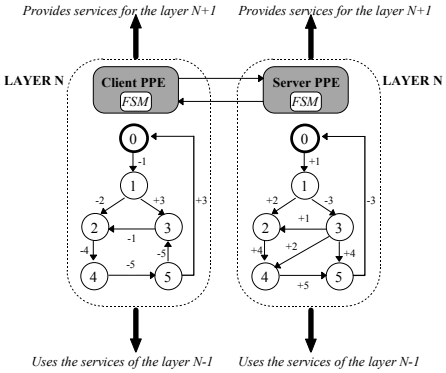


Fig. 1. *Architecture of a sample communication protocol (EXP protocol) with 2 FSMs representing both the client and the server ends. Positive values represent input messages and negative values are output messages.*

Fig. 1 shows the two PPEs each one executing its own FSM. In the most simple situation every transition is an input (positive) or output (negative) reception/transmission of a message. Messages of the same type share the same absolute value. In the rest of the paper we first state the meaning of the genotype and how crossover works. We then present the details of our measurement of the fitness and finally we expose the results and some considerations on the future work.

2 Genotype and Crossover

The first stage we face is to devise a chromosomal representation of a solution to our validation problem. Since we are interested in detecting deadlock, non-visited states and non-fired transitions, we define the strings in the population as encoding a trace of the execution of the system. This means that we will have to manage variable-length strings in the same population. Every gene in the chromosome (string position) is coded as a byte (values ranging from 0 to 255). If a gene has value `x` then the `x`-th firable transition of the actual state in the FSM will be fired. In this way the chromosome represents a sequence of system transitions.

Two problems arise from this gene interpretation. The former is concerned with its numeric value, because if a gene indicates, for example, that 56-th transition must fire, and the actual state has only 2 or 3 firable transitions (or even 0!) some mapping from the string to the actual value is needed. We make a kind of fixed remapping [3] by means of which we take `(x mod ft)` as the transition to fire, where `x` is the string value and `ft` is the *number of firable transitions* for the present state in the FSM (Fig. 2a).

The later problem with this encoding is related to the number of processes. At present we consider only two communicating processes in the system, thus we should need two different traces. One possible solution could have been to use diploid genomes and a dominance operator like in [5]. But we have decided to utilise the same haploid string as representing the two traces, because the two FSMs will be in general in different states and will have different sets of firable transitions. In this

way the same gene of the string defines the next state for every FSM (Fig. 2b). This is thought to be a better choice for the representation in terms of the BBs hypothesis.

Some final considerations must be made about what we call a **firable transition**. For any given present state in one FSM, a transition is *firable* if it consists in sending an output message to the other process **or** if it consists in receiving a given message **and** its associated messages queue contains this type of message. When a process sends a message, the simulator stores it in the queue of the other process.

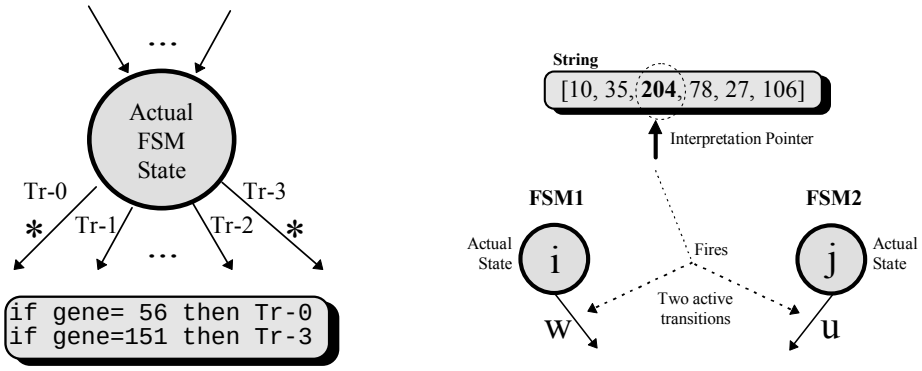


Fig. 2. *Genotype representation for genetic validation.* **(2a)** *Every gene is a positive integer. We take the* **mod** *value with respect to the number of firable transitions. The transitions with the * symbol are active. So if the current gene in the string is an even number then transition 0 will fire and if it is an odd number transition 3 will fire. In* **(2b)** *the interpretation of a string is showed. The pointer moves from left to right firing different transitions in the two FSMs.*

If we want to validate more complicated protocols then we need also to devise a more elaborated conception of a transition. We need, in every transition, an input and output parts. The FSM proceeds from a state to another one only if the input message specified in the transition is also present in its messages queue. When the transition fires the message in the output part is sent to the other process. In order to model user commands or messages from an upper layer of the protocol family, we define an **event** to be a special input message that can be always considered firable.

v2.0

```
2 //Nr. of processes
process 0:                process 1:
8 //Nr. of transits.     9
0 1 -1                    0 1 +1
1 2 -2                    1 2 +2
1 3 +3                    1 3 -3
2 4 -4                    2 4 +4
3 2 -1                    3 2 +1
4 5 -5                    3 4 +2
5 3 -5                    3 5 +4
5 0 +3                    4 5 +5
//ist-ost-iomsg           5 0 -3
```

v3.0

```
2 //Nr. of processes
process 0:                process 1:
8 //Nr. of transits.     9
0 1 ^0 -1                 0 1 +1 -0
1 2 ^0 -2                 1 2 +2 -0
1 3 +3 -0                 1 3 ^0 -3
2 4 ^0 -4                 2 4 +4 -0
3 2 ^0 -1                 3 2 +1 -0
4 5 ^0 -5                 3 4 +2 -0
5 3 ^0 -5                 3 5 +4 -0
5 0 +3 -0                 4 5 +5 -0
//ist-ost-imsg-omsg       5 0 ^0 -3
```

Fig. 3. *Two versions of files representing the FSMs in the fig. 1. The files contain the number of processes, the number of transitions per process and the transitions themselves. Every line is composed of the actual state, the next state and the transition (v2.0 in/out as a single message and v3.0 in-out as two separate messages). The first version (v2.0, left) considers transitions as input or output exchanges of messages. The later version (v3.0, right) introduces the event ^0 as the input part of an output transition. For input transitions we output a NULL (-0) message.*

We have developed a version of GAVOR for every kind of transition. In the above figure two input protocol files with the FSMs of Fig. 1 are showed. The two files represent the same sample protocol but in different versions of our GAVOR package. You can see that transitions with only output messages (always firable) in version 2 are encoded in version 3 with NULL input events (also always firable). With our coding we have tried double point crossover (DPX) because of its easy application to this domain and its widely recognised benefits. In the crossover implementation we encourage the creation of new traces.

3 Fitness Computations

For the computation of fitness values we need a complete simulator of the protocol being validated. For every trace (string) we run the simulator module and add a `D_FITNESS` quantity if a deadlock is encountered. For every non-visited state and every non-fired transition (it is not the same) we add the values `NV_FITNESS` and `NF_FITNESS` respectively.

With this evaluation scheme we need a value for `D_FITNESS` much larger than the other two values, because a deadlock detection is far more important than a useless state or transition.. One straightforward drawback is that if several enough states and transitions are not visited/fired by the trace the computed fitness can be larger than for traces that detect deadlock. This situation can be avoided if we choose `D_FITNESS`>>`NV_FITNESS`>`NF_FITNESS`. The GA maximises this fitness.

Fitness = D_FITNESS*deadlock_present + NV_FITNESS*number_nv_states + NF_FITNESS*number_nf_transitions

We have tried another kind of fitness computation consisting in penalising bad strings (the inverse of the above function but removing first the deadlock term). The results have been bad (we need a refinement) and are not reported here.

We also need to impose lower and upper limits to the initial length of the traces created for the initial population, since a too short trace will fail in representing a good tour over the FSM states and transitions while a very long trace will spent a great deal of computational resources unnecessarily. At runtime we promote (the crossover operator does it) traces of short length but do not impose limits on length.

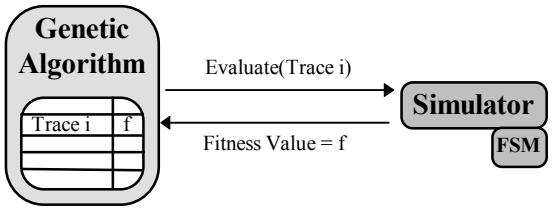


Fig. 4. *Interaction between the GA and the Simulator. The GA terminates execution when a user given number of steps has been reached. The best string stands for the maximum number of protocol violations. If the best final fitness is 0 then the GA has validated the protocol.*

4 Analysis of Results

In this section we study the efficacy and efficiency of the devised GA for the validation of two different protocols. The first protocol is the example protocol (EXP) depicted in Fig. 1 and the second one is the well-known TCP from the TCP/IP protocol family. We will perturb the original protocol by introducing deadlocks and by disabling some states and transitions in order to study the efficacy of the GA in detecting such specification errors. The EXP considers two different FSMs while the TCP protocol has a unique FSM that is run at both ends. Also the complexity of the TCP protocol is considerably greater, thus we only will use the EXP for a detailed study. TCP will be used for testing the bounds of the applicability in GA validation.

4.1 The EXP Protocol

Despite the small number of states (6+6 in total) in the EXP protocol it presents the main characteristics of a general protocol, with the added interest of using two different finite state machines (for the client and server ends).

Validating The Correct EXP Protocol.

First we want to validate the correct version of this protocol (GA parameters shown in the Table 1). Although the protocol is correct at a first glance (for a human) the GA validator had some troubles with respect to the initial length of the traces. With a value of between 20 and 100 for the initial length of strings we have always got some non-fired transitions and 1 or 2 non-visited states (typically the state 2 of FSM_1). Fig. 5 shows the results for different ranges of length.

Table 1.- *Basic Parameters for the tests with GAVOR v3.0*

Selection:	Roulette Wheel and Random Parents
Replacement:	Steady-State
Crossover:	Double Point with P_c=1.0
Mutation:	Random Transition with P_m=0.1
PopSize:	50 individuals
Nrecombin.:	100
Seeds:	{3, 7, 11, 17, 19, 23, 34, 57, 93, 137}
D_FITNESS=10	**NV_FITNESS**=2 **NF_FITNESS**=1

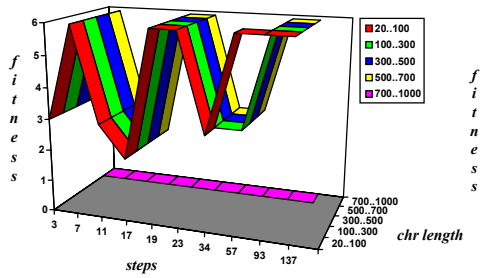


Fig. 5. *Influence of the initial string length on the suitness of the validation. The EXP protocol is correct but low string lengths make some states and transitions to appear as useless. Only 700..1000 validates.*

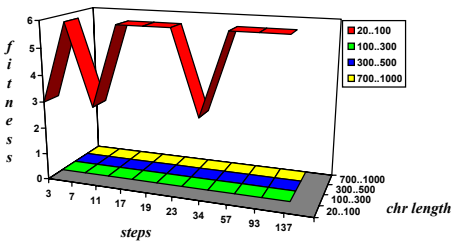


Fig. 6. *All the initial string lengths above 100 allow to validate the EXP protocol (this is the correct result since we are validating a correct protocol) when a population of 100 individuals is used.*

Since we suspected that length from 700 to 1000 was a somewhat excessive value for this relatively small protocol we enlarged the population size up to use 100 individuals and then we got a perfect validation for any initial string length greater than 100 (Fig. 6). Only the traces with lengths between 20 and 100 failed in recognising that the protocol was correct albeit the significance of their final error values is minor (two non-fired transitions and one non-visited state).

Disturbing the Correct EXP Protocol.

In these tests we want to show that GAVOR performs different levels of validation. First we have changed in FSM_0 the transition from the states 1 to 2 by removing the -2 output message and by adding a +2 input message (Fig. 7). In all the 10 runs GAVOR found that this transition was now non-firable. But also it concluded that the transitions 1-2 and 3-4 from the FSM_1 had also became non-firable because our disruption in FSM_0 had eliminated all the sources of the -2 messages and therefore any transition with a +2 condition was non-firable. This means a fitness of 3 (1+1+1).

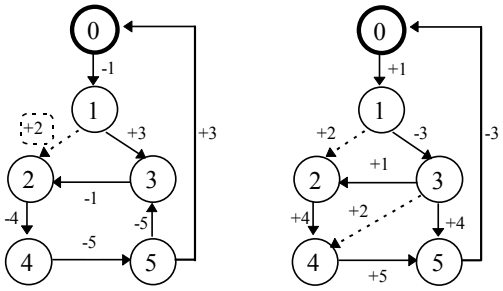


Fig. 7. *Useless Transition: FSM_0[1-2:-2⇨+2]. GAVOR v3.0 found it.*

The second kind of disruption we have introduced in the original EXP is to change in FSM_0 the transition 1-2 from -2 to +2 and also the transition 2-4 from -4 to +4 (Fig. 8). These changes are known to introduce a deadlock in the system. We want GAVOR to discover this deadlock. The runs perform very consistently in computing traces of fitness 29 (11 non-fired transitions plus 4 non-visited states plus deadlock fitness ⇨ 11*1 +2*4+10=29). We have tried initial lengths of 2..10, 5..10, 10..20 and 20..100 with 10 and with 50 individuals in the population and for every one of the 10 different seeds. The most usual trace do not visit the states 4 and 5 in any of the FSMs. In fact this is true because it is impossible to visit them!

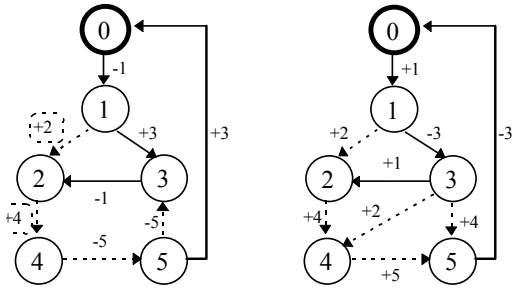


Fig. 8. *Deadlock: FSM_0[1-2:-2⇨+2* ***and*** *2-4:-4⇨+4]. GAVOR v3.0 detected this deadlock.*

In these runs we have also detected an interesting side-effect of the implementation in that when the actual state in the first FSM has two firable transitions, (O and I messages), it always fires the one with the output message. This is because we always simulate FSM_0 first and then FSM_1. With a random selection of the first FSM to simulate for every transition fire, this problem has been removed.

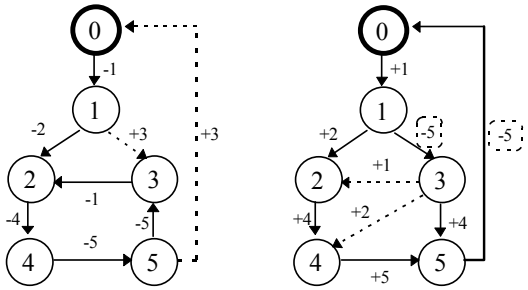


Fig. 9. *Livelock: FSM_1[1-3:-3⇨-5* ***and*** *5-0:-3⇨-5]. GAVOR v3.0 detected this condition.*

The last disruption we have tried consists in changing in the original EXP protocol the FSM_1 by making transitions 1-3 and 5-0 to send a -5 message instead of a -3 message thus removing any source in FSM_1 of the message type 3 (Fig. 9). Our validator quickly (always below 50 steps) found that the transitions 1-3 and 5-0 of FSM_0 are now non-firable because they wait a +3 message input that never arrives.

Also the validator found a non-intuitive scenario that reveals that transitions 3-2 and 3-4 in FSM_1 cannot be fired. This is because FSM_0 is always looping in its internal cycle 2-4-5-3-2-4-5-3-2... and FSM_1 (except for the first time in that it visits 0-1-2-4-5-0) is always making the same (its only permissible) loop: 0-1-3-5-0-1-3... consuming the -1 message from FSM_0 in the 0-1 and not in its 3-2 transition.

This kind of hidden synchronisation can make some transitions useless. In our example, FSM_1 receives messages of type 1 but they are never consumed in the 3-2 transition, but always in the 0-1 transition. Known or not, this scenario provokes a sort of *livelock* in that there exists at least one possible infinite execution that makes transition 3-2 non-firable although messages of type 1 are being received. Livelocks are typically undesired factors in any parallel system and GAVOR can detect them.

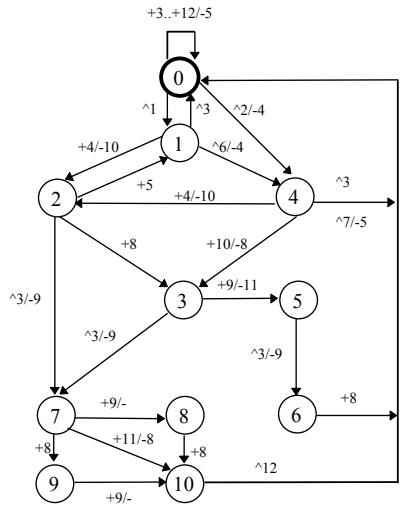
4.2 Validating The TCP

In this section we want to validate the Transmission Control Protocol. This is a broadly extended protocol that performs the functions of the transport layer (after the OSI terminology) and that is present in a very large number of LANs and in the Internet. In the tests first we try to validate the correct protocol and then we study the GA efficacy in detecting abnormal conditions introduced in the correct protocol.

In this protocol the same state machine defines the behaviour of both communication ends. This is a difference with respect to the EXP protocol. Another difference is its complexity (much higher than for EXP). Also here we have, besides the I/O normal transitions, a new kind of event transition, used for modelling the interactions with the user. For example the user can `open` a communication path (^2) or `close` the communication (^3) or a `time-out` can expire (^12) -see Fig. 10-.

Validating The Correct TCP.

In this section we want to validate the original TCP. Due to the magnitude of the search for this protocol, we consider a hit if we approach near enough to a perfect genetic validation (fitness 30 or so). Fig. 10 depicts the basic TCP finite state machine we are using. Of course a trace of fitness 0 only indicates that, at least, there exists a given good execution. If we do not find deadlock we are not absolutely sure that it does not exist, but when averaged on all the runs, if no trace finds deadlock we are reasonably sure it doesn't exist (if we find deadlock we only know it is invalid).



MEANING OF THE STATES

```
0  CLOSED
1  LISTEN
2  SYN RECVD
3  ESTABLISHED
4  SYN SENT
5  CLOSE WAIT
6  LAST ACK
7  FIN WAIT-1
8  CLOSING
9  FIN WAIT-2
10 TIMED WAIT
```

MEANING OF THE I/O MESSAGES

```
1  passive_open  7  time-out
2  active_open   8  ack
3  close         9  fin
4  syn           10 syn+ack
5  reset         11 fin+ack
6  send          12 wait
```

Fig. 10. *Finite State Machine forTCP. Since GAVOR v3.0 doesn't allow more than 1 transition between a given couple of states we have considered for the transitions 0➔0 only the +8/-5 (ack/reset). The same holds for the two transitions 4➔0 in that we only consider the ^3 (close).*

The conclusions about the results are somewhat difficult because we have only partial statistics on the whole execution of the GA. We still need to consider the quantitative value for the probability of making good interpretations of the results. For example, if we detect a deadlock we are sure (Pr=1) that deadlock exists. If we do not detect deadlock, which is the probability (Pr=?) that deadlock is not really present in the protocol?. We are still developing a numeric interpretation.

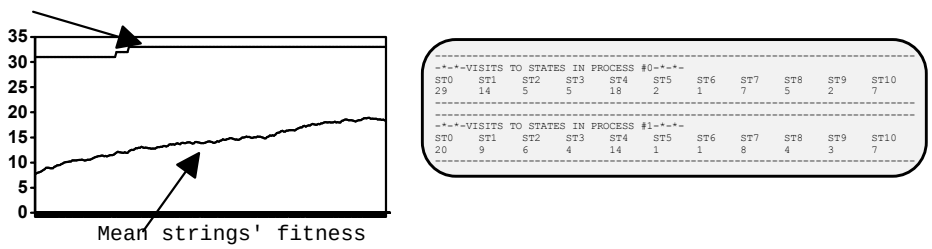
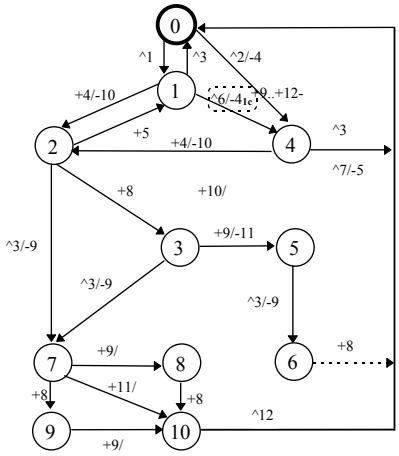


Fig. 11. *Genetic validation of TCP. The graphic shows the worst and mean search that strings perform for 500 recombinations of 100 strings with between 500 and 700 of initial length. The GA cannot find any deadlock or other errors (this protocol is error free).*

Disturbing the Correct TCP.

We have tried to validate three independent and incorrect versions of the original TCP. First we have changed the event ^6 from the states 1 to 4 to a new +6 (never-firable) message reception. The second disruption consists in inducing a deadlock by eliminating the only output transition (+8) from the state 6. The last disruption removes the -8 messages from the FSMs (`ack`'s won't be sent).

For the change ^6⇨+6 all the computed traces report that transition 1-4 is always inactive. We have used the parameters in Table 1 and also we have tried initial lengths of 100..500 and 100 individuals instead of only 50. The execution is slower (10 minutes) but the traces worked out more accurate results.



5 Conclusions and Future Work

In this work we have attempted the validation of two communication protocols (EXP and TCP) by means of a genetic algorithm (GAVOR v3.0 package). The genotype we have used is a variable-length string (initially generated with bounded length) representing a trace (for the 2 FSMs) that defines the dynamic behaviour of the protocol. The fitness of every trace is measured by simulating the trace on the FSMs and by adding some reward value depending on how many non-visited states, non-fired transitions and deadlocks are detected.

Our results in the genetic validation are very promising and robust. This approach has proved to be good in detecting useless states and transitions in the protocol specification that represent common errors in the designer's work. Deadlocks are also detected with efficiency and livelocks are (indirectly) reported.

The technique does need some refinements with respect to the fitness computation, a different remapping from genotype to phenotype and the parameterization of the GA. Finally, the genetic search presents some problems in refining the results for large protocols. We are testing several improvements on this approach: to count for the loss of messages, to decide on how to manage the messages that remain in queue when simulation ends and studying the degree of confidence. Also a global and continued interpretation (statistics and runtime decisions) of every string worked out could allow a best validation (at present we are trying this).

References

[1] Alba, E., Aldana, J. F. and Troya, J. M**., Genetic Algorithms as Heuristics for Optimising ANN Design**. R.F. Albrecht, C.R. Reeves and N.C. Steele (eds), Artificial Neural Nets and Genetic Algorithms, Innsbruck. Springer-Verlag, pp 683-690. (1993)

[2] Alba, E., Aldana, J. F. and Troya, J. M., **Load Balancing and Query Optimisation in Dataflow Parallel Evaluation of Datalog Programs**. Lionel M. Ni (ed), Proceedings of the International Conference on Parallel and Distributed Systems, Taiwan. IEEE Computer Society Press, pp 682-688. (1994)

[3] Beasley, D., Bull, D. R. and Martin, R. R., **An Overview of Genetic Algorithms: Part 1 (Fundamentals) and Part 2 (Research Topics)**. University Computing, pp 58-69 15(2) and 15(4) pp 170-181. (1993)

[4] Díaz, M. and Troya, J.M., **A Parlog Based Real-Time Distributed Environment**. Future Generation Computer Systems 9, pp 201-218, North-Holland. (1993)

[5] Goldberg, D. E., **Genetic Algorithms, in Search, Optimization and Machine Learning**. Addison-Wesley. (1989)

[6] Fogel, L. J., Owens, A. J. and Walsh, M. J., **Artificial Intelligence Through Simulated Evolution**. New York, John Wiley. (1966)

[7] ISO, **LOTOS-A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour**. ISO IS 8807. (1989)

[8] ISO, **ESTELLE-A Formal Description Technique Based on State Transition Model**. ISO 9074. (1989)