A METHOD FOR REAL-TIME EMULATION OF GENETIC ALGORITHM-OPTIMIZED CONTROLLERS

Nicolae NECULA[1]

Articolul prezintă o abordare bazată pe Programarea Genetică pentru descoperirea structurii unor controleri optimizați cu Algoritmi Genetici, permițând astfel proiectanților să emuleze si să implementeze în timp real comportarea acestor controleri . Metoda este ilustrată prin rezultatele experimentale obținute în urma simulării în Matlab a unui controler pentru alocarea adaptivă a resurselor, util pentru planificarea optimă a pachetelor într-un sistem TDMA.

This paper describes a Genetic Programming (GP)-based approach for the reverse engineering of Genetic Algorithm (GA)-optimized controllers, allowing the designer to emulate and implement in real-time the behavior of such controllers. The method is illustrated by the experimental results generated in the Matlab simulation of an adaptive resource allocation controller, useful for optimal packet scheduling in a TDMA system.

Keywords: genetic algorithms, genetic programming, symbolic regression, TDMA optimal resource allocation, Matlab simulation

1. Introduction

Genetic algorithms (GAs) are heuristic global optimization techniques derived from the principles of natural selection and evolution [1]. They are powerful tools for solving complex non-linear optimization problems, subject to different type of constraints. Quality of the obtained solution depends on how the solution "fitness" is defined, and what values are chosen for algorithm parameters, such as: population size and the required number of generations.

GAs have been theoretically and empirically proven to be robust search techniques, capable of finding solutions that are better than those obtained by many other heuristic methods, being closer to the theoretical optimum for various realistic complex optimization problems [2].

Most network design and traffic engineering problems are formulated and solved as constrained optimization problems, many of them being best solved by GAs [3], particularly the combinatorial optimization problems that do not have to be run in real-time.

[1] Prof., Faculty of Electronics, Telecommunications and Information Technology, Department of telecommunication University POLITEHNICA of Bucharest, Romania

Optimal packet scheduling problems, like the TDMA (Time Division Multiplex Access) problem, can be solved by GAs [4], but the real-time implementation of a controller running such an algorithm is impossible due to the huge amount of computation that needs to be performed in a very short time, i.e. once per TDM frame.

However, the behaviour of such a controller is worth investigating in order to attempt to emulate it in real-time. In two previous papers [5,6], a "reverse engineering" approach has been used by the two authors for identifying and implementing the adaptive control mechanism that has been "discovered" offline by the GA. The necessary stages of this approach are:

- Control function definition. A simple and efficient elementary control function should be capable of allocating all timeslots, once per frame, to two competing sets of traffic sources, according to the current state of the system. A good characterization of the system state is given by the total current length of the queues that store the packets generated by the two sets of sources. These two sets should comprise sources in high and low QoS classes, respectively. Repeated, hierarchical utilization of such an elementary control function can easily partition the given set of timeslots into any number of blocks (subsets) for servicing optimally as many QoS classes as desired.
- Control function(s) dataset specification. By running offline a GA with an appropriately defined fitness function, for optimal per TDM frame packet scheduling, sample points are generated for all the control functions defined above. In order to smooth the unavoidable statistical fluctuations, these points are averaged over multiple runs of the GA.
- Control function(s) identification. Empirical approximation by a two-variable, one-parameter algebraic expression for all the desired control functions is finally performed, where the parameter value is specific to each individual function.

This paper illustrates the reverse engineering process applied to a particular GA-generated non-real-time optimal controller. It also describes a more systematic approach for control function identification using Genetic Programming (GP). Since GP is capable of performing symbolic regression for identifying any unknown functions from a given subset of its data points [7], it is the natural choice for performing the identification task.

Control function specification is briefly described in Section 1 for a particular application: the GA-based optimal timeslot allocation controller. However, similar steps can be performed for other types of GA-based optimal controllers, as well.

Section 2 describes a simplified implementation in Matlab of the GP for symbolic regression.

Experimental results obtained by running this simplified GP for the specified control function are illustrated and analyzed in Section 3.

Finally, some possible directions for further work and concluding remarks are included.

2. Control function definition and specification

Optimization of the tradeoff between priority-based service and service fairness during periods of high load with large bursts of data, in a TDMA system, is one of the challenging problems that can be solved off-line using a GA.

Conventional solutions allocate statically a fixed number of timeslots per frame to the different QoS (quality of service) classes, and use the remaining timeslots for strict priority-based dynamic allocation, when needed [8,9].

The main drawback of this type of solutions is the poor fairness obtained when the sources in the higher priority classes generate large bursts of data at the same time.

By accepting a slight degradation of the quality of service for higher priority classes as the price paid for a better fairness offered to the lower priority classes during the congested periods, a better timeslot allocation can be obtained.

An off-line controller of this type can be obtained by using a GA, where each chromosome represents a possible per frame timeslot allocation, and the fitness assigned to a chromosome is calculated, according to [4], by:

$$C_F = \sum_{i=1}^{s} \Phi_i \tag{1}$$

where:

s = chromosome length = the number of timeslots per frame,

Φ_i = fitness assigned to the i*th* time slot , with regard to the traffic source serviced by this time slot:

$$\Phi_i = \frac{P_i * Q_i}{\sqrt{f_i}} \tag{2}$$

where:

P_i = priority of the traffic source serviced by the i*th* time slot,

Q_i = dynamic length of the queue used by this traffic source

f_i = number of time slots assigned to this traffic source in the current frame.

By maximizing the above fitness, the GA attempts to find, in every frame, an optimized chromosome capable to assign all the available timeslots in such a manner that achieves an optimal balance between the opposite requirements regarding both priority-based service and fairness.

An elementary control function for timeslot allocation is defined as follows: let r = $F(q_1,q_2)$ partition dynamically, once per frame, the set of s timeslots into two subsets of r*s and (1-r)s timeslots, that are allocated to real-time (*rt)* and non-real-time (*nrt)* traffic sources, respectively. Value of r is a function of the current length of the queues assigned to these sources. Queue length is measured in traffic units, where one unit equals the timeslot bandwidth.

By using the fitness function defined above, together with a mix of *rt, nrt and BE* traffic sources, the GA-based scheduler is run once per frame, and all relevant data points are collected for identifying the control function that describes how the GA-based scheduler partitions adaptively and (almost) optimally the set of available timeslots into two subsets.

The following choices have been used for the simulation scenario:

- TDMA physical layer:
  - Link bandwidth = 20 Mbps
  - Frame duration = 1 ms
  - Timeslots per frame s=20
  - 1000 bits per timeslot = conventionally defined packet size
  - 100% link load
- Traffic sources (3+1 QoS classes):
  - Highest priority (Constant bit rate): 9 sources
  - Second priority (Real-time variable bit rate): 14 MPEG2 sources
  - Next priority (Non-real-time variable bit rate): 10 FTP sources
  - Lowest priority (Best effort): 2 sources
  - Bandwidth demand: 30% for CBR sources, 34% for VBRrt sources, 30% for VBRnrt sources, and 6% for BE sources.
- Genetic Algorithm characteristics:
  - Population size = 50
  - Maximum number of generations = 50
  - Gene crossover probability = 0.7
  - Gene mutation probability = 0.01
  - Elitist selection, using 3 classes of fitness: highest (30% of population size), intermediate (40%) and lowest (30%).
- Simulation characteristics:
  - Simulation duration = 10 seconds = 10000 frames.
  - 10 successive runs, with no reinitialization of the Matlab random number generator.

The simulation steps, together with some additional steps (shown in bold characters), that are necessary for specifying the elementary control function defined above, are:

Repeat for 10 times
Repeat for 10000 frames
Insert data packets into the service queues
Run GA to generate opt_chrom (optimal timeslot allocation)
Extract data packets from the queues specified by opt_chrom
Calculate and store triplet $\{q_{1i},q_{2i},r_i\}$ ***for i=1 to 10*10000***
At the end, convert the final array of triplets into two p×p matrices

The resulting array of triplets describes the dynamic behaviour of the GA when handling *rt* vs. *all other types* of packets.

This very large array is then converted into two easier to handle p×p matrices F(X,Y) and C(X,Y), by scaling down the original range of values of q_1 and q_2 ($[q_1,q_{1.max}]$ and $[q_2,q_{2.max}]$, respectively) to just p=20 uniform quantization levels (i.e. X,Y=1,2,…,20). C(i,j) is the number of points that fit into the respective compressed range {[i,i+1),[j,j+1)}, and F(i,j) is the average value of r for all these C(i,j) points.

Other possible options for obtaining matrices F and C may use non-uniform quantization of the q_1 and q_2 values, and/or different values for p.

After identifying the unknown expression of the function specified by its points in F(X,Y), either empirically or by symbolic regression, a simple real-time adaptive scheduler can be developed to emulate the above GA – optimized scheduler.

The basic idea of this adaptive scheduling algorithm is to allocate dynamically, once per frame, the available resources to just two QoS classes or groups of classes (e.g.: real-time classes vs. all other classes), according to the following formulas that emulate the behavior of the GA-based scheduler:

IF $q_1+q_2>s$ THEN

$$r = F(q_1,q_2) \quad (3)$$

$$n_1=\min(\text{round}(r*s),q_1) \quad (4)$$

$$n_2=\min(s-n_1,q_2) \quad (5)$$

ELSE $n_1=q_1$, $n_2=q_2$; ENDIF

$$n_3=\max(1,s-(n_1+n_2)) \quad (6)$$

where:

- s = the number of per frame un-reserved timeslots,
- q_1 = total current length of all real-time queues,
- q_2 = total current length of all other queues,

(q_1 and q_2 are conventionally measured in traffic units (tu's), where one tu equals the timeslot size in bits),

- r = % of frame timeslots allocated to real-time sources,

- n_1, n_2 = the number of timeslots allocated to the two QoS classes,
- n_3 = the number of timeslots allocated to the BE class.

If the total queue lengths q_1+q_2, corresponding to the number of packets waiting to be dispatched from the queues is less than s, then all sources obtain their required number of timeslots and no resource partitioning algorithm is needed.

Otherwise, formula (3) provides the desired tradeoff between priority-based service and fair service: when $q_1>>q_2$, r is close to 1, and almost all timeslots are allocated to the first class, but when q_1 and q_2 are comparable, then the second class gets more timeslots allocated to it. Formulas (2) and (3) provide the work conserving characteristic of the algorithm by never allowing the number of timeslots allocated to any non-BE class to exceed the current length of the queue used by the respective class.

A simple empirical formula obtained in [5] for function $F(q_1,q_2)$ is:

$$r = 1/(1 + c*q_2/ q_1) \quad (7)$$

where the value of parameter c determines the timeslot allocation policy, as follows:

- c=0 ensures strict priority-based allocation
- As c increases in value between 0 and 1, the allocation fairness between the two QoS classes gets better, but the price paid is the increased degradation of the service offered to the first class.
- When r=0.5, complete fairness is provided to the two QoS classes, i.e. round robin scheduling.

Resource allocation to more than two QoS classes requires obviously a repeated application of the above algorithm in a multi-level hierarchical manner, with more control functions of type (7), each with its specific value of parameter c.

3. A simplified implementation of genetic programming

Genetic Programming (GP) [7] represents another set of heuristic global optimization techniques, similar to GAs, but better suited for solving symbolic problems. Generally, GP is capable to “discover” optimized data processing programs for, practically, any imaginable type of data processing. Symbolic regression is just a particular type of application where the program describes an algebraic expression, and the objective is to find an optimal algebraic expression that describes an unknown function, specified by a number of its data points. A possible criterion of optimality is the mean square error in all the data points.

Since the chromosomes and genes used for representing the evolving solutions in GP are based on rooted trees with ordered branches, they are much more difficult to handle than the ones used in GAs. This is the reason why a

simpler form of GP, well suited to the particular application considered in this paper, has been defined and implemented by the present author.

The following simplified definition has been defined by the present author for describing the relatively simple algebraic expressions needed:

- The chromosome is a linear sequence of g genes, with each gene corresponding to an element of the expression (variable, constant or operator), and specifying by its position the location of this element within the expression.
- There are three types of genes:
  - arithmetic operator genes that specify one of the 4 possible basic arithmetic operations: add, subtract, multiply, and divide with saturation (for overflow avoidance), encoded by values 1 thru 4,
  - identifiers of variables that specify by an index (1,2,…) which variable occurs in the respective location of the algebraic expression,
  - constant specifiers that specify both the location and the value of each constant, if any, in the algebraic expression. Only positive constants are defined and encoded, by using the negative values of the respective constants in order to distinguish them from the variable identifiers. A negative constant can also be specified in an expression by just using the subtract operator gene in front of the gene specifying a positive constant.
- The total number of genes in a chromosome is g=L+(L-1), where L is the number of all entries specified for variables and constants, and L-1 is the number of arithmetic operators connecting all successive pairs of variables and/or constants. Genes are conventionally ordered in the chromosome, starting with the L location specification genes and ending with the L-1 operator specification genes.

When evaluating the numerical value of an expression specified by a chromosome, the multiply and divide operators are used first, from the right of the expression to the left, and then, the add and subtract operators are used from the left of the resulting expression to the right. As a result of this simplified representation and evaluation procedure, expression a/b*c is equivalent with a/(b*c), and a*b+a*c is equivalent with a(b+c), but a/(b+c) is not possible because it cannot be handled by the simplified GP.

Since the empirical approximation of F(X,Y) suggests an expression of the type shown in formula (7), which cannot be handled by the simplified implementation of GP used in this paper, the identification has been performed for the reciprocal of F: G(X,Y)=1/F(X,Y).

As an example, consider the following 13-gene chromosome describing an algebraic expression for a 2-variable function G(X,Y):

crom(1:11) = [-81 1 -50 2 1 1 -80 1 4 4 4 3 2]

The corresponding algebraic expression is:

G(X,Y) = 81+X/50/Y/X*X-80 = 1 + 0.02*Y/X

After simplifications, the above size-13 expression converts to the denominator of formula (7) for c=0.02, X=q_1 and Y=q_2. Since a size of 13 is too large for this particular function, some redundant sub-expressions have been generated by GP: (81-80) for constant=1, and X/(50/(Y/(X*X))) for 0.02*Y/X.

Generally, before applying GP to solve the identification problem, the following five data sets have to be determined:

- The set of independent variables of the unknown function = {X,Y}.
- The set of primitive functions = {+,-,* and /}.
- The chromosome fitness measure = reciprocal of the mean square error, defined by:

$$\phi = 1/(\sum_{i=1}^{N}(E(X_i, Y_i) - 1/F_i)\text{^}2) \tag{8}$$

where: N = number of selected data points

F_i = *i*th value in the set selected from F(X,Y) matrix

$E(X_i,Y_i)$ = value of the algebraic expression evaluated in the *i*th data point in the set selected from F(X,Y).

The parameters for controlling the run: population size, maximum number of generations to be run, etc.

The number of runs and the method for designing a result

4. Control function identification by symbolic regression

Fig. 1 illustrates the data points generated by the GA for the implicit elementary control function specified in Section 1.

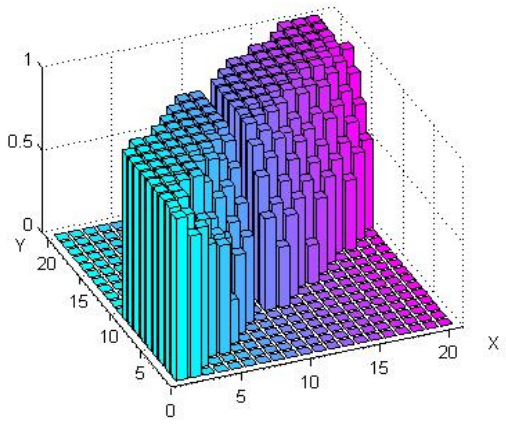

Fig. 1. Data points for the elementary control function F(X,Y): X,Y = values of *rt* and *nrt* queue lengths after uniform quantization; F(X,Y): values are averaged over 10 GA runs for link load=1

The large number of saturation points, with F(X,Y)≈1, that are present in this graph, contribute to making F rather difficult to approximate.

Since GA does not generate points for all possible (X,Y) pairs, F(X,Y) is plotted as 0 for all the undefined pairs. In [5], F(X,Y) looks somewhat different than in Fig.1 because it is extrapolated as F(X,Y)=1 for all the undefined pairs with $Y \geq Y_{max}/2$.

The frequency of occurrence of the F(X,Y) data points in 10 successive GA runs is illustrated in Fig.2. It can be noticed that many of the saturation points have a high frequency of occurrence.

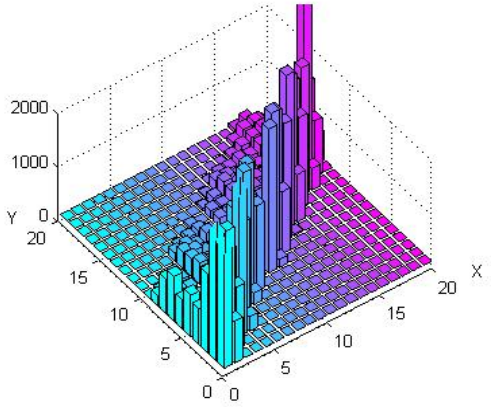

Fig. 2. Number of occurences of the F(X,Y) points in 10 GA runs (some of these numbers are greater than 2000)

This makes the appropriate selection of the N data points for symbolic regression rather difficult. Out of the many possible options for data point selection, after experimentally choosing a value of N, the following have been tested:

- Select the most frequently generated points (with the largest values of C(i,j)). This option did not work well, as expected, because too many of the selected data points are saturation points, that are practically useless for the identification of F.

- Select all points that are most frequently generated in a given range of values. This option eliminates all saturation points (with F(i,j)≈1) and all statistically irrelevant (very infrequent) points, allowing a much easier identification of F.

Symbolic regression is then performed for the set of N=40 appropriately selected data points, and the only assumed a priori knowledge about the unknown function G(X,Y)=1/F(X,Y) consists of:

- Number of variables is 2, according to function specificication.

- The maximum expected size of the algebraic expression of G(X,Y) has been chosen as S=13. The denominator in formula (7) has a size of only 7, with 2 identifiers of variables (used just once per variable), 2 constant values and 3 arithmetic operators. By choosing a maximum expected size of 13, GP is allowed to discover more complex formulas for G. If such formulas do not exist, GP generates redundant terms in the optimized solution, as shown below.

The following experimentally chosen values have been used for the simulation scenario:

- GP characteristics:
  - Population size = 500
  - Maximum number of generations = 500
  - Chromosome size = 13
  - Probability of generating a constant = 0.5
  - Maximum value of a constant = 100
  - Gene crossover probability = 0.7
  - Gene mutation probability = 0.03
  - Elitist selection, using 3 classes of fitness: highest (30% of population size), intermediate (50%) and lowest (20%).
- Simulation characteristics:
  - 10 successive runs, with no reinitialization of the Matlab random number generator.
  - N=40 data points for the unknown function specification

The following sequence of steps is then performed for running the GP:

- Generate randomly an initial population of chromosomes
- Repeat for the chosen number of generations:
  - Evaluate the fitness of all chromosomes in the current generation
  - Create the next generation of chromosomes by using the elitist approach and by applying the two basic genetic operations: crossover and mutation.
- Designate the chromosome with the highest fitness as the (approximate) solution to the problem

The results obtained in 10 GP runs for N=40 points of 1/F(X,Y), selected in the range $0.5<F(X,Y)<1$, are illustrated in Table 1.

It can be noticed that GP has used “ingenious” ways to create constants, redundant terms and subexpressions, as illustrated by various examples in this table:

Y/Y=1, 81-80=1, X-X=0

53/X/Y/4*40 = 53/(X/(Y/(4*40))) = 0.3*Y/X

Table 1

Symbolic regression results for 1/F(X,Y) identification

Run #	Algebraic expression described by the optimal chromosome	Simplified expression of 1/F(X,Y)
1	Y / X * 2 / 65 / 98 + X / X	1 + 0.33*Y/X
2	Y / Y + 63 * Y / X * 95 * 2	1 + 0.3*Y/X
3	Y / X + 24 / 83 - 49 / 59 * X	Error
4	Y / Y + X + Y / X * 3 - X	1 + 0.33*Y/X
5	X / 3 / Y / X * X + 81 - 80	1 + 0.33*Y/X
6	45 / 31 + Y / 15 - X * 7 / 79	Error
7	99 / 56 - X / 50 - 65 / Y * 52	Error
8	Y / Y + X / X * 3 * X / Y	1 + 0.33*Y/X
9	X / X + 53 / X / Y / 4 * 40	1 + 0.3* Y/X
10	84 / 54 - X / Y * Y / X / 5	Error

The four runs that are labelled as errors, are not really errors, but less useful expressions. Their occurrence is very likely due to the use of uniform quantization when generating matrix F(X,Y). It is possible that non-uniform quantization with small steps for smaller data values and large steps for large data values may have worked better.

The other six of the GP runs converged to results that are very similar to the one obtained empirically in [5]. For problems where the unknown function is completely unknown, i.e. it has no empirically determined algebraic expression, a convenient experimentally chosen criterion like mean square error or minimal maximum absolute error needs to be used to select the best solutions generated by the GP in 10 (or more) runs.

5. Conclusions

Despite the fact that GA and GP offer very few or no convergence guarantees, they are still capable of generating good, nearly optimal solutions for complex problems, and are easily adapted to new problem domains.

The steps used for the reverse engineering of the particular GA-generated controller described above can be successfully used for solving many other types of optimal real-time resource allocation and/or scheduling problems. Further work is however required in identifying specific such controllers and emulating their behavior in real-time.

The numerous experiments required to select the best possible combinations of GA and GP parameters and characteristics, initial data (number

of data points, number of quantization levels, type of quantization) and a priori assumptions is a very demanding task, but the results are worth obtaining, as the adaptive scheduling algorithm described in [5,6] proves it.

This algorithm has been a useful addition to the numerous existing scheduling algorithms, because it can provide a better fairness in servicing the lower priority classes during (severe) link overload conditions, by enforcing just minor (acceptable) quality degradations of the service offered to the higher priority classes.

R E F E R E N C E S

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

[2] *M. Ericsson, M.G.C. Resende, P.M. Pardalos,* A Genetic Algorithm for the Weight Setting Problem in OSPF Routing, J.Combinatorial Optimization, **Vol.6**, No.3, pp.299-333, 2002.

[3] *Michal Pioro, Deepankar Medhi,* Routing, Flow and Capacity Design in Communication and Computer Networks, Morgan Kaufman Publishers, 2004.

[4] *S.Thilakawardana, R.Tafazolli,* Use of Genetic Algorithms in Efficient Scheduling for Multi-Service Classes, European Wireless Conference 2004, Barcelona, Spain, 24-27 Feb., 2004.

[5] *N. Necula, E. Borcoci*, Adaptive packet scheduling for QoS support in IEEE 802.16 wireless access systems, UPB Scientific Bulletin, Series C, **Vol.70**, No.1, 2008, pp.3-16.

[6] *N.Necula, E.Borcoci*, Performance analysis of an adaptive scheduler for IEEE 802.16 wireless access systems, UPB Scientific Bulletin, Series C, **Vol.71**, No.1, 2009, pp.3-20.

[7] *J.R. Koza, F.H. Benett III, D. Andre, M.A. Keane,* Genetic Programming III: Darwinian Invention and Problem Solving, Morgan Kaufman Publishers, 1999.

[8] *Kitty Wongthavarawat, Aura Ganz*, Packet scheduling for QoS support in IEEE 802.16 broadband wireless access systems, Int.J.Commun.Syst., **vol.16**, issue 1, pp.81-96, Feb. 2003.

[9] *Alexander Sayenko, Olli Alanen, Juha Karhula, Timo Hämäläinen*, Ensuring the QoS requirements in 802.16 scheduling, Proc. of Nineth ACM International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems, Torremolinos, Spain, Oct. 2006, pp.108-117.