Fuzzy Classifier System and Genetic Programming on System Identification Problems

Jose Aguilar
CEMISID, Dpto. de Computación,
Facultad de Ingeniería.
Universidad de Los Andes
Mérida, VENEZUELA.

Mariela Cerrada
CEMISID, Dpto. de Control,
Facultad de Ingeniería.
Universidad de Los Andes
Mérida, VENEZUELA.

Abstract

In this work, two techniques of Computational Intelligence, Fuzzy Classifier System and Genetic Programming, are compared on system identification problems. By using a Fuzzy Classifier System, we pretend to find an input-output identification fuzzy model (composed of fuzzy rules). The Fuzzy Classifier System uses a genetic algorithm in order to adapt an initial population of fuzzy rules. In Genetic Programming, a set of analysis trees (the nodes are a set of mathematical symbols: constants, functions, variables, operators, etc.) is the population manipulated by the evolutionary algorithm. These analysis trees describe the possible different identification models. In both cases, the initial population is generated based on intuitive knowledge about the dynamic of the system. A set of historical data about input and output signals is used to adapt that population

1 INTRODUCTION

In processes control are required models which describe the dynamic behavior of the system in order to carry out control tasks [7,11,13]. Identification techniques propose an approximated model of a real system, based on linguistic or mathematical expressions, or an algorithm. Identification models that only manipulate input and output signals is one of the possible identification schemes (Input-Output Identification Models). In control theory, there are many techniques to solve this problem [10]. In this work, two intelligent mechanisms based on Evolutionary Computation (EC) are proposed in order to solve the input-output identification problem of dynamical system, one of these based on Genetic Programming (GP) and the other one based on Fuzzy Classifier System (FCS). In the case of GP, this approach proposes the evolution of a set of possible models that characterize the system. In specific, the evolutive process manipulates a population of analysis trees, which describe the possible models. In the case of FCSs, an input-output identification fuzzy model is generated from an initial population of fuzzy rules. Genetics Algorithms (GAs) are used to propose a new population of rules through an iterative cycle of states, until minimizing the identification error.

2 SYSTEM IDENTIFICATION (SI)

In control tasks, it is necessary to known the system model that describes the behavior of the system [7, 10, 11, 13]. The identification methods develop models which are capable of describe the essential properties of a system, taking into account its static and dynamic behavior during an interval of time. Such models can be used in control tasks, fault tolerance, etc.

There are many identification methods, several of them based on the control theory [10], or on the computational intelligence [1, 2, 12]. The identification models can be defined as a non-linear function of the current input (u(t)) and previous inputs (u(t-1), u(t-2) and so forth) and outputs (y(t-1), y(t-2) and so forth) (these models are called input-output identification models) [10]. The classical scheme for system identification is shown in the figure 1. The error signal between the real output and the estimated output is used to update the model parameters.

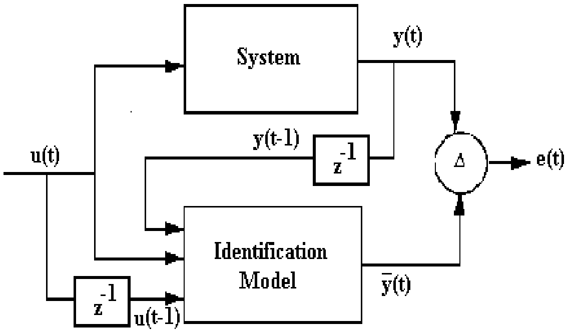


Figure 1: System identification scheme.

3 FCS AND GP ON IDENTIFICATION PROBLEMS

3.1 FCS-BASED IDENTIFICATION MECHANISM

In a previous work [3], a FCS for fault tolerance in industrial processes has been designed. Some ideas of that work have been used in order to design our FCS approach for system identification problems. The FCS generates an input-output identification fuzzy model, which is obtained from historical data about the input and output variables of the system. Our identification scheme based on FCS is shown in the figure 2.

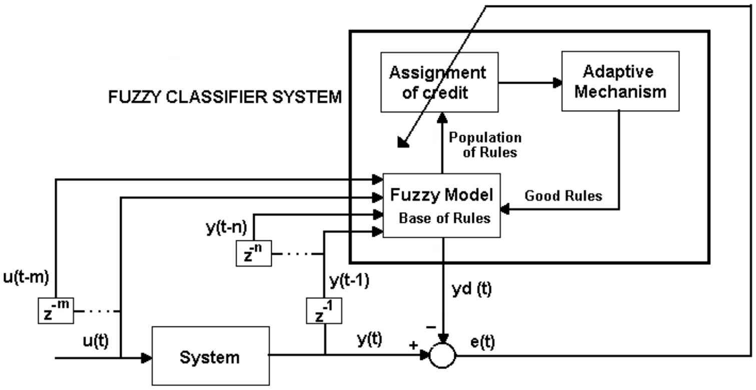


Figure 2: Our identification scheme based on FCS

In this design, we suppose that both the generic structure of the fuzzy rules and the membership functions of the fuzzy sets are known. Then, the FCS only finds the best instances of this generic structure.

3.1.1 Algorithm of the FCS.

For each training pattern, according to the historical data of the system and a population of “n” fuzzy rules, we follow the next steps:

1. Compute the activation grade of each rule.
2. Compute the credit of each activated rule.
3. Defuzzification of the output fuzzy set obtained by the fuzzy inference mechanism.
4. Compute the identification error *er*.
5. Compute the average error *ep*, when all patterns have been processed.
6. If average error is bigger than the error limit given by the user, then the FCS uses the adaptive mechanism based on GAs.
    6.1 Choose the parents (rules with high credit value).
    6.2 Apply the genetic operators (mutation and crossover).
    6.3 Replace the olds individuals for the new individuals, according to some replacement mechanism.

This procedure is repeated until that the identification average error reaches a minimum value given by the user or a maximum number of iterations have been accomplished.

3.1.2 The identification error calculation

The equation (1) is used to calculate the identification error associated to each pattern. The average error for all training patterns is given by the equation (2).

$$er = |(y_s - y_d)/y_s| \qquad (1)$$

$$ep = \sum_{i=1}^{m} er/m \qquad (2)$$

where y_s is the output of real system, y_d is the output of the fuzzy model and m is the number of patterns.

3.1.3 The fitness function definition.

The credit value of each fuzzy rule is computed based on the fitness function given by the equation (3):

$$S_i(t+1)=S_i(t)+Act_i(t)*\mu y_i/ea \qquad (3)$$

where $S_i(t)$ is the credit value of the fuzzy rule *i* at time t, $Act_i(t)$ is the activation grade of the fuzzy rule *i* at time t, *ea* is the absolute error ($ea=y_s-y_d$) and μy_i is the membership grade of the crisp value of the fuzzy model output. This fitness function permits the evaluation of the weight of the output fuzzy set of a rule into the crisp value given by the fuzzy model. So, a good credit value is obtained for those rules which give a minor identification error.

3.1.4 The adaptive mechanism

Each rule is codified as a vector of finite length, as it is shown in the figure 3.

Ve_1	CD_1	....	Ve_3	CD_3	Vs	CDs

Figure 3: Codification of a rule as an individual

where Ve_i is the input variable i, CD_i is the fuzzy set of the input variable Ve_i, Vs is the output variable and CDs is the fuzzy set of the output variable Vs.

In this work, we propose a set of changes into the fuzzy sets of the input and output variables in order to create new rules. The genetic operators of crossover and mutation are used in order to accomplish this task [6]. At the end, the new population is composed of $n+k$ rules (individuals), where n is the number of rules of the previous population and k is the number of new rules. In order to have n rules, we must eliminate k rules. We eliminate a rule according to its probability of elimination, given by the equation (4):

$$Pr(R_t)= Fr(R_t)/ \sum_{i=1}^{m} Fr(R_i) \qquad (4)$$

where P_r is the replacement probability of the rule R_i, F_r is the replacement factor of the rule R_i and m is the number of rules of the population (m=n+k). The replacement factor is given by the equation (5):

$$Fr(R_i)=1-FA_i/\sum_{j=1}^{m} FA_i \qquad (5)$$

where FA_i is the credit value of the rule R_i.

3.2 GP-BASED IDENTIFICATION MECHANISM

In this section it is proposed a method based on PG to develop identification models. In our approach, each individual is defined by a Multiple Interaction Programs (MIP) model. In the MIP model, each node is one equation, which is represented by an analysis tree. The identification mechanism proposes a simultaneous evolution of each analysis tree [1].

In our model, the terminal set of each node has input variables, constants or outputs from some precedent equations. In the figure 4.b is shown an analysis tree for T3, where *In1* y *In2* are input values of the problem. *T1* y *T2* are the outputs of these equations, which precede T3 (see figure 4.a). This model is easy to implement in GP, through the utilization of the ADF (Automatic Definition Function) technique. This extension of GP permits to define functions to evolve in parallel with the main procedure. These functions can be called by other functions, or by the main procedure, during the evolution. In our case, the MIP model defines the relationship among the functions. The population evolution follows the next algorithm:

1. Define a given MIP model for the individuals.
2. Generate, randomly, a population of individuals. Each one of the individuals is defined by a set of analysis trees according to the MIP model.
3. Evaluate each individual in order to determine its performance. The evaluation function is the average error between the historical output of the system and the output of the identification model (individual).
4. Select the parents (individuals with the smallest average error).
5. Apply the genetic operators to these parents in order to reproduce new individuals.
6. Replace the old worst individuals for the new individuals.

4 EXPERIMENTS

In this section, we present an example in order to compare both proposed identification methods. The example is a distillation system that uses a distillation column in continuous operation of multiple stages.

4.1 SYSTEM DESCRIPTION

The objective of a distillation system is to separate a mixture in two or more fractions with different boiling points. The function of the continuous distillation system can be seen with details in [7]. In the figure 5 is shown the structure of this distillation column. The feeding input (composed by benzene and toluene) is introduced in the second plate, and the distilled product is obtained in the first plate on the top of the column.

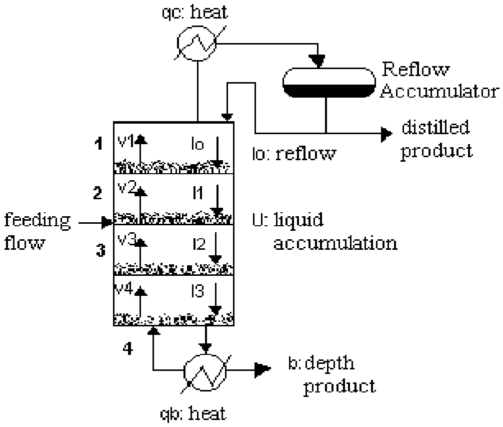


Figure 5: Distillation column

The constant input signal (feeding rate) is modeled with a step function with amplitude equal to ten (U(t) = 10). The theoretical model of this system is given by the equation (6) [7]:

$$X(t) = 1.1148*X(t-1) + 0.2525*X(t-2) - 0.3823*X(t-3) + 0.3294e-4*U(t-1) \quad (6)$$

where X(t) represents the output of the system. The output is the concentration of benzene on the top. The output signal from this model is shown in the figure 6.

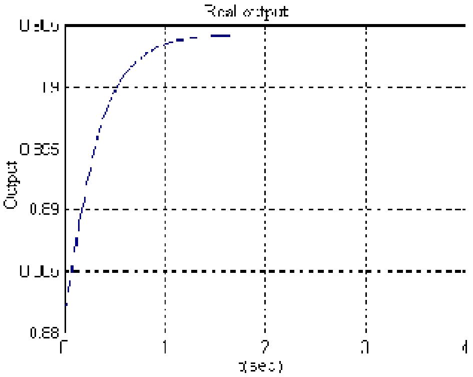


Figure 6: Output signal from theoretical model

4.2 IDENTIFICATION MODEL BASED ON GP

In order to develop the computational program, we have used the "The Genetic Programming Kernel" library designed by A. Fraser en 1994 [5]. This library permits the utilization of ADFs.

In this experiment, the MIP model is composed by two equations (M1 y M2), where M2 represents the ADF and M1 represents the main program (main tree), which can depend of M2 or not. The function set used by M1 and M2 is {+,-,*, %, *sin, cos}*. The terminal set of the main tree is composed by St(M1)={*u, xa1, xa2, xa3, xa4, xa5, s_M2*}, where *u* is the input signal at the time *t, xa1* is the output signal at the time t-1 *(X(t-1)), xa2* is the output at the time t-2 *(X(t-2)),* and so on, and s_M2 is the output of the ADF. The terminal set of the ADF only has two elements St(M2)={*xa1, xa2*}. The trigonometric functions are supposed with input values given in radians.

The historical values of the input and output signals have been obtained using the theoretical model defined by the equation (6). The aptitude of each individual was determined based on the average error between the output historical values and the outputs of the model proposed by the individual for the same set of input signals. A population of 300 individuals has been evolved through 50 generations. Finally, the individual with the smallest average error is selected. In the table 1 is shown the models obtained (the best individuals) using our identification method, for different terminal sets.

Table 1: Identification Models

CASES	IDENTIFICATION MODEL	THEORETICAL MODEL	ERROR
AP: C_T={u, a1, xa2, xa3, s_M2}	M1 = 2*xa1 – xa2*s_M2	Equation (6)	1.59254e-4
ADF: C_T={x1, x2}	M2 = $(xa1)^2$ / xa2		
AP: C_T={u, xa, xa2, s_M2}	M1 = 2*xa1 – xa2* s_M2	Equation (6)	2.3965e-4

ADF: C_T={x1, x2}	M2 = Equation (7)		
AP: C_T={u, xa1, a2, xa3, xa4, s_M2}	M1 = (xa1 / xa2)*xa1	Equation (6)	2.86043e-4
ADF: C_T={x1, x2}	M2 = xa1+xa2-xa3		
AP: C_T={u, xa1, xa2, xa3, xa4, xa5, s_M2}	M1 = 2*xa1 – xa2 *s_M2	Equation (6)	1.59254e-4
ADF: C_T={x1, x2}	M2 = $(xa1)^2$ / xa2		

where:

$$M2=xa1-\sin(\sin(\sin(\sin(\sin(\sin(\sin(\sin(\sin(\sin(\sin(\sin(\sin(\sin((xa2-xa1))))))))))))))) \tag{7}$$

The identification models obtained in the cases 1 and 4 are similar, and they are the best models. In the second case, the ADF model is different to the previous ones, but the value of the error is acceptable. In the case 3, the identification model do not depends of the ADF. In general, in all cases the best individual depends of the output signal at the times (*t-1*) and (*t-2*), and it does not depend of the input signal. In the second case, the identification model is more complex.

The identification error signal obtained by the model proposed in the case 2 is shown in the figure 7. The input signal is a constant function U(t)=10, and the initial conditions for the variables xa1 y xa2 was randomly selected near to the real initial conditions. At t=2 sec., the identification error converges to zero.

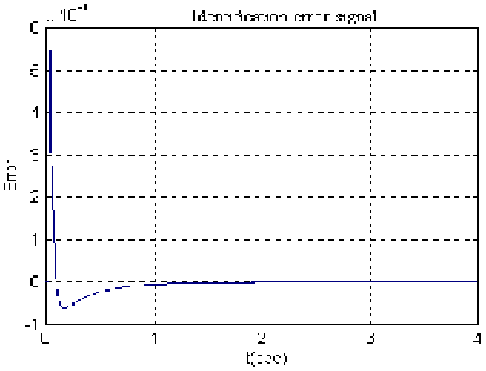


Figure 7: Identification error using the second model

4.3 IDENTIFICATION MODEL BASED ON FCS

In our approach, we suppose the following generic structure for the fuzzy rules:

$$\text{If } U(t) \text{ and } Y(t-1) \text{ then } Y(t) \tag{8}$$

where U(t) denotes the input variable at time t, Y(t-1) denotes the output variable at time t-1 and Y(t) denotes the output variable at time t. For such variables, we previously define their fuzzy sets according to their historical data values. The membership functions of these fuzzy sets are shown in the figure 8.

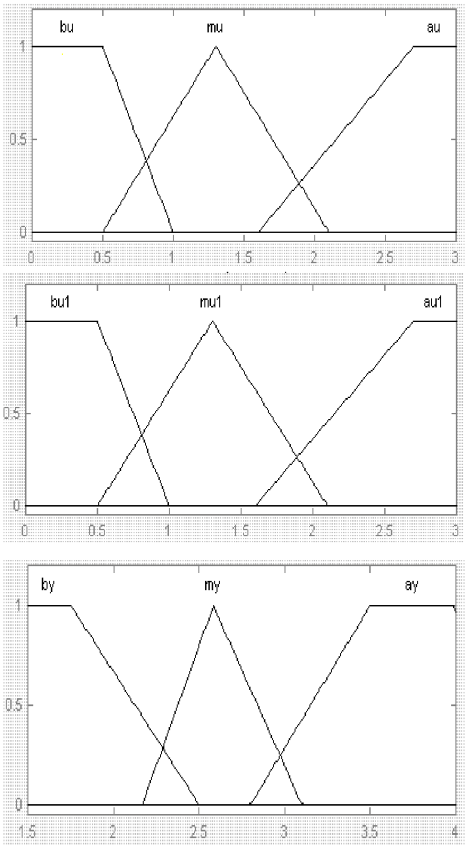


Figure 8: Membership functions of the fuzzy sets for U(t), Y(t-1) y Y(t).

Different experiments have been made from an initial population of fuzzy rules and 800 training patterns. The best fuzzy model according to the identification average error is the following:

If U(t) is mu and Y(t-1) is bu1 then Y(t) is ay

If U(t) is au and Y(t-1) is mu1 then Y(t) is by

If U(t) is mu and Y(t-1) is au1 then Y(t) is my

If i U(t) is au and Y(t-1) is au1 then Y(t) is ay

If i U(t) is mu and Y(t-1) is mu1 then Y(t) is ay

If i U(t) is au and Y(t-1) is au1 then Y(t) is my

If i U(t) is mu and Y(t-1) is bu1 then Y(t) is my

If i U(t) is bu and Y(t-1) is mu1 then Y(t) is my

If i U(t) is mu and Y(t-1) is mu1 then Y(t) is by

This fuzzy model has been found in the iteration number 87, with an average training error of 0.13. The output of this fuzzy identification model, for the input signal U(t)=10, is shown in the figure 9.

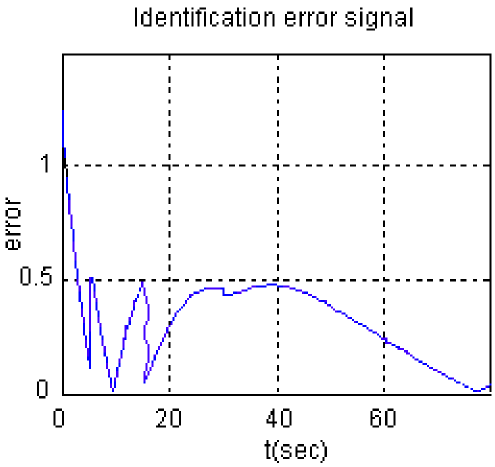


Figure 9: Identification error signal based on FCS.

In the figure 9, we can observe the identification error signal of the fuzzy model. The accuracy is not very good. We remark that the membership functions have not been adjusted. We can reach a good accuracy if these membership function are suitably adjusted. If we compare the identification error for both approaches (see figures 7 and 9), we can see that the identification error based on PG is better than the other one based on FCS.

5 CONCLUSIONS

This work shows the capabilities of the GP and the FCS in system identification problems. This application is very useful when the knowledge about the system is poor and when we not have the expert knowledge about the relationships between the system variables. The identification models that we have obtained by using these approaches are suitable.

FCS has been "training" out of line, therefore, the fuzzy model is an universal generic model. It is necessary to test our identification mechanism with different structures of the fuzzy rules in order to give more information at the FCS (more delayed input and outputs signal as inputs variables) . We can observe that there are many repeated fuzzy rules into the model, then the elimination algorithm must be improved. In the future, we will incorporate a membership function adaptive mechanism.

In the case of the GP, it depends of the function and terminal sets that are used, and the relationship established in the MIP model. In the future, we are going to test one extension of our approach where the MIP model evolves such that the evolution determines the optimal relation between the equations/variables.

Based on the experimental results, the GP-based identification mechanism is more efficient than the FCS-based mechanism, but we must remark that the FCS have not the membership function adaptive mechanism. This is a serious limitation that we must improve. Finally, other experiments will be tested in order to determine the efficiency of each proposed technique in different types of problems.

Acknowledgment

This work was partially supported by CONICIT-CONIPET grant *97003817* and CDCHT-ULA grant *I-621-98-02-A*.

References

[1] J. Aguilar, M. Cerrada, " GP-Based Approach for System Identification", accepted for publication, WSES International Conference on Evolutionary Computation, Tenerife, Spain, February 2001.

[2] P. Angeline, D. Fogel, "An Evolutionary Program for the Identification of Dynamical System", Technical Report, Natural Selection Company, 1998.

[3] M. Cerrada, J. Aguilar. "Fuzzy Classifier System: An Applications for Fault Tolerance in Industrial Processes", Proceedings of IASTED International Conference of Artificial Intelligence and Soft Computing, pp. 453-456, 1998.

[4] D. Fogel, "Evolutionary Computation: Toward a New Philosophy Through Simulated Evolution", IEEE Press, 1995

[5] A. Fraser, "Genetic programming in C++", Technical Report, University of Salfrod, Cybernetics Research Institute, 1994.

[6] D. Goldberg, "Genetic Algorithms in Optimization and Machine Learning", Addison-Wesley Publishing Company, 1989.

[7] C. Holland, "Fundamentos y Modelos de Procesos de Separación", Prentice Hall, 1997.

[8] J. R. Koza, "Genetic Programming: On the Programming of Computers by Means of Natural Selection", MIT Press, 1992.

[9] J. R. Koza, "Genetic Programming II: Automatic Discovery of Reusable Programs". MIT Press, 1994.

[10] L. Lennart, "System Identification, Theory for the User", Prentice Hall, 1997.

[11] K. Ogata, "Ingeniería de Control Moderna", McGraw Hill, 1992.

[12] P. Sastry, G. Santharam, K. Unnikrishnan, "Memory neuron networks for identification and control of dynamical system", IEEE Transaction on Neural Networks, Vol. 5, No. 2, p.p. 306-319, 1994.

[13] A. Wittenmark. "Computer Control System", Addison Wesley, 1989.

Multi-Objective Evolutionary Optimization of Flexible Manufacturing Systems

Jian-Hung Chen
Dept. of Information Engineering
Feng Chia University
Taichung, Taiwan 407, R.O.C.
jh.chen@ieee.org

Shinn-Ying Ho
Dept. of Information Engineering
Feng Chia University
Taichung, Taiwan 407, R.O.C.
syho@fcu.edu.tw

Abstract

This paper describes multi-objective evolutionary optimization of process planning in flexible manufacturing systems (FMSs). FMS can be described as an integrated manufacturing system consisting of machines, computers, robots, and automated guided vehicles (AGVs). While FMSs give great advantages through the flexibility, FMSs pose complex problems on multi-objective process planning. An evolutionary approach using a multi-objective evolutionary algorithm with a new elite clearing mechanism is proposed for solving the multi-objective process planning problems (MOPPPs). The experimental results demonstrate that our algorithm can solve MOPPPs efficiently.

1 INTRODUCTION

A flexible manufacturing system (FMS) is a production system consisting of a set of identical and/or complementary numerically controlled machines which are connected through an automated guided vehicle (AGV) system. Since FMS is capable of producing a variety of part types and handling flexible routing of parts instead of running parts in a straight line through machines, FMS gives great advantages through the flexibility, such as dealing with machine and tool breakdowns, changes in schedule, product mix, and alternative routes. Flexible manufacturing is of increasing importance in advancing factory automation that keeps a manufacturer in a competitive edge.

While FMS offers many strategic and operational benefits over conventional manufacturing systems, its efficient management requires solutions to complex process planning problems with multiple objectives and constraints. The aim of process planning is to develop a cost effective and operative process plan over the planning phases. Decisions regarding the process planning problem have to be made before the start of actual production, and consists of organizing the limited production resource constraints efficiently. Generally, the process planning includes routing optimization, equipment optimization and machine optimization (Tempelmeier and Kuhn, 1993).

During the past decades, a number of computer-aided process planning (CAPP) systems have been developed for the automated planning and increased efficiency of process planning, considering only a single objective. However, from a system designer's point of view, it is very desirable to obtain optimal solutions considering all the objectives. Moreover, obtaining a set of non-dominated solutions provides the flexibility for reconfigureable manufacturing.

Recently, some authors applied genetic algorithms (GAs) (Goldberg, 1989) to the process planning. Awadh et al. proposed a CAPP model based on GAs (Awadh, Sepehri and Hawaleshka, 1995). Moon et al. proposed an evolutionary algorithm for solving the flexible process sequencing problems with two objectives (Moon, Li and Gen, 1998). Brandimarte proposed a two-objective hierarchical approach based on a decomposition into a machine loading and a scheduling sub-problem (Brandimarte, 1999). However, the simpified/bicriteria model considered does not satisify the needs of FMSs.

Considering the practical manufacturing environments, we formulate the problems of process planning in FMSs as multi-objective process planning problems (MOPPPs), and propose an evolutionary approach using multi-objective evolutionary algorithm with a new elite clearing mechanism for solving MOPPPs.

This paper is organized as follows: Section 2 introduces the flexible manufacturing system and the mathematical formulation of MOPPPs. Section 3 presents the multi-objective evolutionary algorithm for the problem. Section 4 presents the experimental analysis of the proposed algorithms, and Section 5 summarizes our conclusions.

2 FLEXIBLE MANUFACTURING SYSTEM

A brief description of the flexible manufacturing environment and the mathematical formulation of MOPPPs are given in this section.

2.1 THE FLEXIBLE MANUFACTURING ENVIRONMENT

An FMS consists of a set of identical and/or complementary numerically controlled machines and possibly tool storage. All components are connected through an AGV system. Figure 1 from (Tempelmeier and Kuhn, 1993) shows the layout of a simple FMS with several machines and a tool system.

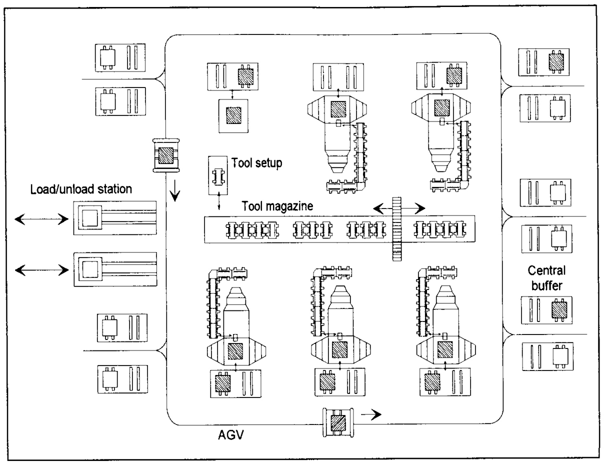


Figure 1: FMS with two AGVs and a central tool magazine.

In order to tackle the process planning in FMSs, the environment within which the FMS under consideration operates can be described as follows:

(1) A part type requires a number of operations. There is a number of part types will be manufactured simultaneously in batches. Parts can choose one or more machines at each of their operation stages, and the transportation of the parts within different machines is handled by an AGV system.

(2) The types and number of machines are known. There is sufficient input/output buffer space at each machine.

(3) A machine type can perform several types of operations, and an operation can be performed on alternative machine types.

(4) A machine type can only process an operation at one time. Operations to be performed in the machine type are nonpreemptive. Operation lot splitting is ignored.

(5) The tool costs of operations in machine types are known. Processing, times of operations in machine types are available and are deterministic.

(6) Workload on each machine is contributed by those operations assigned to a machine.

(7) A load/unload (L/U) station serves as a distribution center for parts not yet processed and as a collection center for parts finished. All vehicles start from the L/U station initially and return to there after accomplishing all their assignments. There is sufficient input/output buffer space at the (L/U) station.

(8) Number of AGVs is given and the transportation time of AGVs are known. AGVs carry a limited number of products at a time. They move along predetermined paths, with the assumption of no delay because of congestion. Preemption of trips is not allowed.

(9) It is assumed that all the design, layout and set-up issues within FMS have already been resolved.

(10) Real-time issues, such as traffic control, congestion, machine failure or downtime, scraps, rework, and vehicle dispatches for battery changer are ignored here and left as issues to be considered during real-time control.

2.2 MATHEMATICAL FORMULATION

Multi-objective process planning problems (MOPPPs) are concerned with the selection of individual process plans for all the parts with minimizing the total flow time, balancing the machine workload, minimizing the machine workload and minimizing the total equipment cost. MOPPPs can be formulated as follows.

2.2.1 Notations

In order to formulate MOPPPs, the following notations are introduced:

i : part index ($i = 1, 2, 3, \ldots, np$)

j : operation index for part type i ($j = 1, 2, 3, \ldots, no_i$)

k, l : machine index ($k, l = 1, 2, 3, \ldots, nt$)

pv_i : production volume (unit) for part type i

pt_{ijk} : processing time per unit to perform operation j of part type i using machine type k

tw_k : workload in machine k, $tw_k = pt_{ijk} * pv_i$

ew : average workload of machines

$$s_{ikl}:\begin{cases}1, \text{ if part type } i \text{ is to transfer from machine } k \text{ to } l\\ 0, \text{ otherwise}\end{cases}$$

$$x_{ijk}:\begin{cases}1, \text{ if machine type } k \text{ selected to perform}\\ \quad \text{operation } j \text{ of part type i}\\ 0, \text{ otherwise}\end{cases}$$

abl : available capacity of AGV per trip

n_{ikl} : the number of trips between machine types *k* and *l* for part type *i*, $n_{ikl} = s_{ikl} \times \left\lceil \frac{pv_i}{abl} \right\rceil$

tm_{kl} : transportation time from machine *k* to *l*

t_{ikl} : total transportation time between machines *k* and *l* for part type *i*, $t_{ikl} = n_{ikl} \times tm_{kl}$

c_{ijk} : tool costs to perform operation *j* of part type *i* using machine type *k*

2.2.2 Objectives

There are four objectives to be optimized in FMSs according to the suggestion of (Tempelmeier and Kuhn, 1993), described as follows.

(1) Minimization of the total flow time. This objective is to minimize the processing time and transportation time for producing the parts. The total machine processing time (f_1) is defined as Equation (1), the transportation time (f_2) is defined as Equation (2), and the total flow time (F_1) is defined as Equation (3). Transportations between unlinked machines are penalized in f_2.

$$f_1 = \sum_{i}^{np} \sum_{j}^{no_i} \sum_{k}^{nt} pv_i \cdot pt_{ijk} \cdot x_{ijk} \qquad (1)$$

$$f_2 = \sum_{i}^{np} \sum_{j}^{no_i - 1} \sum_{k}^{nt} \sum_{l}^{nt} t_{ikl} \cdot x_{ijk} \cdot x_{i(j+1)l} \qquad (2)$$

$$F_1 = f_1 + f_2 \qquad (3)$$

(2) Minimization of the deviations of machine workload. Balancing the machine workload can avoid creating bottleneck machines. The objective function (F_2) is defined as Equation (4).

$$F_2 = \sum_{k}^{nt} (tw_k - ew)^2 \qquad (4)$$

(3) Minimization of the greatest machine workload. Pursuing this objective also implies attempting to minimize the total flow time. The objective function (F_3) is defined as Equation (5).

$$F_3 = \max (tw_k) \qquad (5)$$

(4) Minimization of the tool costs. Tool costs consider the consumptions of tools. Owing to some unique tools are expensive, it is necessary to consider the tool life. The objective function (F_4) is defined as Equation (6).

$$F_4 = \sum_{i}^{np} \sum_{j}^{no_i} \sum_{k}^{nt} c_{ijk} \cdot x_{ijk} \qquad (6)$$

2.2.3 Multi-objective Mathematical Model

The overall multi-objective mathematical model of MOPPPs can be formulated as follows.

minimize F_1, F_2, F_3, F_4

subject to

$$\sum_{k}^{nt} x_{ijk} = 1, \forall (i, j) \qquad (7)$$

The operation flexibility is concerned with an operation can be performed on alternative machines with the different processing time and transportation time. The constraint, Equation (7), ensures that only one machine type is selected for each operation of a part type.

3 MULTI-OBJECTIVE EVOLUTIONARY APPROACH

Multi-objective evolutionary algorithms have been recognized to be particularly suitable for solving MOOPs because the ability to exploit and explore multiple solutions in parallel, and the ability to find an entire set of Pareto-optimal solutions in a single run.

We applied and refined the generalized multi-objective evolutionary algorithm (GMOEA) proposed by us (Ho and Chang, 1999), and propose a new clearing elite mechanism to reduce the non-dominated set. The advantages of GMOEA are:

(1) Elitism: GMOEA incorporates with two populations: the current population and the elite population, called the tentative set of non-dominated solutions (TSONS).

(2) Fitness assignment strategy: The generalized Pareto-based scale-independent (GPSI) fitness function can assign discriminative fitness value to individuals.

(3) Intelligent crossover (IC): IC is introduced to improve the performance of GMOEA on solving problems with a large number of parameters.

The representation of the chromosome is presented in Section 3.1. The fitness assignment strategy and IC are described in Sections 3.2 and 3.3, respectively. The mutation operation approach is described in Section 3.4. Section 3.5 presents the new elite clearing mechanism. The flow of our algorithm is provided in Section 3.6.

3.1 CHROMOSOME REPRESENTATION

The chromosome representation is defined a series of the operations for all the parts. In the chromosome, each gene stands for a machine type number for the machining operations. The assignment of machine types to operations is made by generating random numbers within the range [1, *nt*], so that the corresponding machine numbers are determined. Take Figure 2 for example, [4 2 3 … *nt*] stands for the process plan of part 1, [3 4 *nt* 2 3 … 1] stands for the process plan of part 2, [4 5 1 …] stands for the process plan of part *n*p.

Part *1*	Part *2*	…	Part *np*
4 2 3… *nt*	3 4 *nt* 2 3 … 1	…	4 5 1 …

Figure 2: The representation of a chromosome.

3.2 FITNESS ASSIGNMENT

The fitness assignment strategy of GMOEA uses a generalized Pareto-based scale-independent (GPSI) fitness function considering the quantitative fitness values in Pareto space for both dominated and non-dominated individuals.

Let GPSI fitness function be a tournament-like score for an individual x_u at the l^{th} evaluation operation with corresponding objective vector u. The current position of x_u in the individuals' score can be given by

$$\text{score}\ (x_u, l) = p_u^l - q_u^l + C \qquad \textbf{(8)}$$

where p_u^l is the number of individuals which can be dominated by x_u and q_u^l is the number of individuals which can dominate x_u in the current Pareto space. The constant C is used to obtain the positive fitness value (generally assign the number of the participant individuals).

3.3 INTELLIGENT CROSSOVER

Two parents breed two children using IC at a time by means of orthogonal array (OA). OA is an array of numbers whose columns are pairwise orthogonal. In every pair of columns all ordered pairs of numbers occur the same number of times.

An OA used in IC is described as follows. Let there be α factors, with two levels (or treatments) for each factor. The total number of experiments is 2^{α} for the popular "one-factor-at-a-time" study. The columns of two factors are orthogonal when the four pairs, (1,1), (1,2), (2,1), and (2,2), occur equally frequently over all experiments. When any two factors in an experimental set are orthogonal, the set is called an OA. To establish an OA of α factors with two levels, we obtain an integer $\beta = 2^{\lceil \log_2(\alpha+1) \rceil}$, build an orthogonal array $L_{\beta}(2^{\beta-1})$ with β rows and (β*-1*) columns, and use the first α columns. For instance, Table I shows an orthogonal array L_8 (2^7). Orthogonal experiment design can reduce the number of experiments for factor analysis. Generally, levels 1 and 2 of a factor represent selected genes from parents 1 and 2, respectively.

Table 1: Orthogonal array $L_8(2^7)$

	Factors							
Exp. no.	1	2	3	4	5	6	7	Function Evaluation value
1	1	1	1	1	1	1	1	y_1
2	1	1	1	2	2	2	2	y_2
3	1	2	2	1	1	2	2	y_3
4	1	2	2	2	2	1	1	y_4
5	2	1	2	1	2	1	2	y_5
6	2	1	2	2	1	2	1	y_6
7	2	2	1	1	2	2	1	y_7
8	2	2	1	2	1	1	2	y_8

Let y_t be the positive function evaluation value of experiment no. *t*. Define the main effect of factor *j* with level k , S_{jk},

$S_{jk} = \sum_{t=1}^{\beta} Y_t^2 \times$[the level of experiment number t of factor j is k],

where

$$[condition] = \begin{cases} 1 & if\ \ the\ \ condition\ \ is\ \ true \\ 0 & oterwise\ , \end{cases}$$

and $Y_t = \begin{cases} y_t & \text{if the function is to be maximized} \\ 1/y_t & \text{if the function is to be minimized} \end{cases}$

The steps to use the OA to achieve the IC is described as follows:

Step 1: Select the first α columns of OA $L_{\beta}(\,2^{\beta-1})$ where $\beta = 2^{\lceil \log_2(\alpha+1) \rceil}$. Note that let the chromosome be uniformly separated into α sub-strings and each sub-string of a chromosome be regarded as a factor in OA.

Step 2: Let level 1 and level 2 of factor j represent the j^{th} sub-string of a chromosome coming from the parent 1 and parent 2, respectively. Generate by-product individuals by means of OA.

Step 3: Calculate the values of the *l* objectives for each solution in the β by-product individuals. Compute their fitness value y_t for experiment no. *t* where *t* = 1, 2, …, β, and then update TSONS.

Step 4: Compute the main effect S_{jk} where *j = 1, 2, …,* α and *k = 1, 2.*

Step 5: Determine the best level for each sub-string. Select level 1 for the j^{th} sub-string if $S_{j1} > S_{j2}$.

Otherwise, select level 2.

Step 6: The chromosome of the first child is formed from the best combinations of the better sub-string from the derived corresponding parents.

Step 7: Rank the most effective factors from rank 1 to rank α. The factor with large (MED) has higher rank.

Step 8: The chromosome of the second child is formed similarly as the first child except that the sub-string with the lowest rank adopts the other level.

Since the machine index can be duplicated in a chromosome, no infeasible solutions will be generated when applying IC.

3.4 MUTATION

The procedure of mutation operator is as follows:

Step 1: Randomly select a machine index in the chromosome. Let the machine index be *i*.

Step 2: Replace the machine index *i* by randomly generate an integer value from the range [1, *nt*].

3.5 THE ELITE CLEARING MECHANISM

The main idea of the elite population is to improve the performance of the algorithm. Therefore, the individuals in the elite population may influences the behavior of the algorithm. However, the elite population may biased towards certain regions of the search space, leading to the unbalanced search directions of the algorithm. Thus, pruning the elite population to encourage the search directions toward unexplored regions is necessary.

Based on the idea of encouraging the search toward unexplored regions, the current population is used to represent the explored regions. If a non-dominated individual covers more individuals in the current population, it implies that the covered region is well explored. Therefore, once the size of the elite population exceeds the upperbound, these non-dominated individuals can be cleared from the elite population. By the way, it is also necessary to keep the boundary individuals, because the boundary individuals in each objective are the representative points for guiding the search directions. The procedure of elite clearing mechanism is as follows:

Step 1: For each non-dominated individual *i* in the elite population, calculate the number of individuals it dominates, *di*, in the current population. Let *di* of the boundary individuals in each objective be –1, so that they are always survived.

Step 2: Select a individual with larger d_i to be cleared by binary tournament selection. Clear a number of non-dominated individuals until the elite population achieves the upperbound.

3.6 MULTI-OBJECTIVE EVOLUTIONARY ALGORITHM

The flow of GMOEA is as follows:

Step 1: Initialization: Randomly generates an initial population of N_{pop} solutions. Let the initial TSONS be empty.

Step 2: Evaluation: Calculate the values of the *l* objectives for each solution in the current population and then compute the GPSI fitness function as the fitness values of all individuals.

Step 3: Update TSONS: Copy the non-dominated individuals and remove the dominated individuals in TSONS. Reduce the number of individuals by means of the elite clearing mechanism.

Step 4: Selection: Select $(N_{pop} - N_{ps})$ individuals from the population by binary tournament selection, and select N_{ps} non-dominated solutions from the TSONS randomly to form the new population, where N_{ps} is equal to $N_{pop} * P_s$.

Step 5: Crossover: Select $(N_{pop} * p_c)$ parents for crossover operations. Apply IC for all the selected pairs of parents.

Step 6: Mutation: Apply the mutation operator.

Step 7: Termination test: If the termination conditions are satisfied, end the algorithm. Otherwise, return to Step 2.

4 EXPERIMENT RESULTS

In order to investigate the performance of GMOEA, GMOEA is tested with the multi-objective 0/1 knapsack problems (Zitzler and Thiele, 1999). SPEA is also implemented to solve MOPPPs in order to make a direct comparison. The performance measure of algorithms we used is the coverage ratio of two set (A, B) by (Zitzler and Thiele, 1999). The coverage ratio of set (A, B) is calculated as follows:

$$C(A,B) := \frac{\text{the number of individuals of B dominated by A}}{\text{the number of individuls of B}}$$

The value C(A, B) = 1 means that all individuals in B are dominated by A. The opposite, C(A, B)=0, denotes that none of individuals in B are dominated by A.

4.1 COMPARSIONS OF MULTI-OBJECTIVE KNAPSACK PROBLEMS

The parameter settings of GMOEA for solving the multi-objective 0/1 knapsack problem with 750 items are as follows:

Current population size	:	50
Upperbound size of TSONS	:	50
Selection rate (Ps)	:	0.25
Crossover rate (P_c)	:	0.8

Mutation rate (P_m) : 0.01
Columns of OA (α) : 15

30 independent runs were performed per test problems, compared with same function evaluation times of SPEA. The raw results of SPEA are from the author's website. The experimental result of 2 knapsack-750 items is shown in Figure 3. The results concerning the C measure are shown in Table 2.

Generally, the simulation results of knapsack problems prove that GMOEA do better than SPEA. While SPEA use a large number of population size (250,300,350), none of solutions found by GMOEA are dominated by the solutions of SPEA.

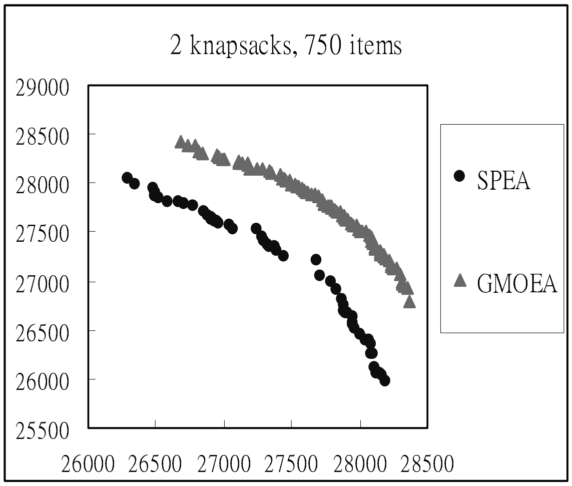


Figure 3: Trade-off fronts out from 30 runs.

Table 2: The C measure of GMOEA and SPEA.

Knapsacks problems	2-750	3-750	4-750
Number of solutions found by SPEA	37	426	1751
Number of solutions found by GMOEA	94	301	372
C(GMOEA, SPEA)	1 (37/37)	0.57 (244/426)	0.72 (1261/1751)
C(SPEA, GMOEA)	0 (0/94)	0 (0/301)	0 (0/372)

4.2 COMPARSION OF MOPPPS

Since MOPPPs are related to the generalized assignment problem (GAP) (Tempelmeier and Kuhn, 1993) (Barndimarte, 1999). Therefore, we used the benchmark problem instances of GAP, which are provided by OR-Library. Two instances, (20 agents, 100 jobs) and (20 agents, 200 jobs) in the benchmark – gapd are derived and formulated. Let agents be machines, jobs be operations, the cost of allocating job to agent be the processing time pt_{ijk}, and the resource requirement be the tool costs c_{ijk} in FMS. Assume a part is consists of 5 operations, so that the first instance has 20 parts, the second instance has 40 parts. The production volume (PV_i) of each part types is given as follows: {45, 43, 39, 46, 42, 56, 37, 33, 61, 30, 55, 43, 24, 39, 29, 44, 30, 45, 29, 30, 55, 33, 37, 43, 62, 36, 42, 44, 53, 40, 35, 41, 34, 29, 38, 49, 43, 25, 69, 41}, $i = 0, 1, \dots, 40$. Let the available capacity of AGV, *abl*, be 10. Considering the real manufacturing environment, the transportation time of AGV is given in Table 4. The transportation time within the same machine is to reflect that a machine unit may be a combination of several machines.

The parameter settings of GMOEA are as follows.

Current population size : 50
Upperbound size of TSONS : 50
Selection rate (Ps) : 0.2
Crossover rate (P_c) : 0.6
Mutation rate (P_m) : 0.05
Columns of OA (α) : 15

The parameter settings of SPEA are the same as the settings of GMOEA, except the population size of SPEA is 150 and the elite population is 50. 30 independent runs were performed per test problems, compared with function evaluation times = 100000.

Table 4: The C measure of GMOEA and SPEA.

MOPPPs	20 machines 100 operations	20 machines 200 operations
Number of solutions found by SPEA	415	199
Number of solutions found by GMOEA-N	392	250
Number of solutions found by GMOEA	465	313
C(GMOEA-N, SPEA)	0.71 (295/415)	0.90 (180/199)
C(SPEA, GMOEA-N)	0 (0/392)	0 (0/250)
C(GMOEA, SPEA)	1 (414/415)	1 (199/199)
C(SPEA, GMOEA)	0 (0/465)	0 (0/313)

In order to investigate the affects of the elite clearing mechanism, GMOEA without the elite clearing mechanism (GMOEA-N) is also performed. Moreover, box plots are used to visualize the distribution of solutions in each objective.

Box plots of MOPPP with 20 machines and 200 operations are shown as Figure 4, 5, 6 and 7. The results concerning the C measure are shown in Table 4. The simulation results of MOPPPs indicate that all the non-dominated solutions found by SPEA are dominated by GMOEA, and the elite clearing mechanism improves the distribution of solutions while maintaining the quality of solutions.

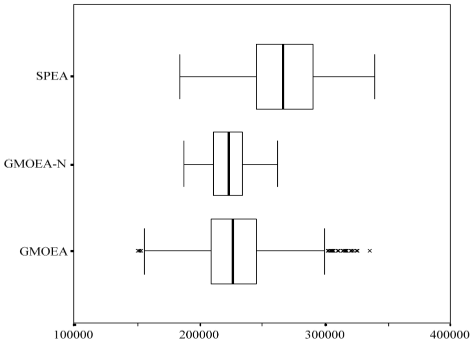


Figure 4: The distribution of solutions in F_1.

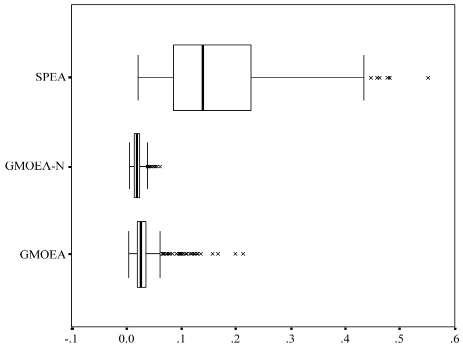


Figure 5: The distribution of solutions in F_2.

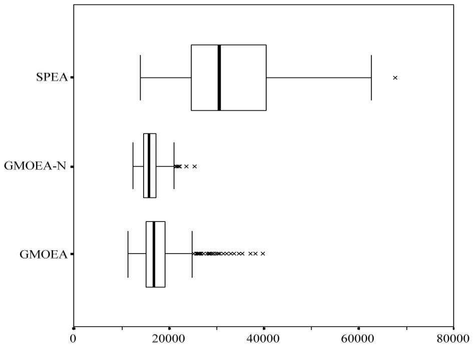


Figure 6: The distribution of solutions in F_3.

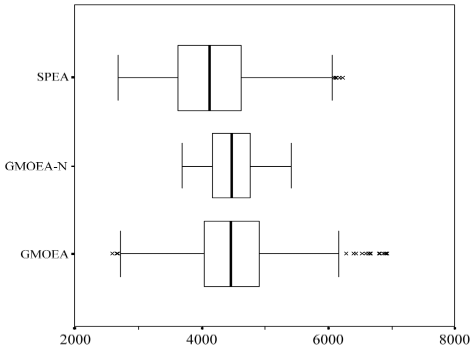


Figure 7: The distribution of solutions in F_4.

4.3 DISCUSSIONS

From the reported results, it is shown that:

(1) The quality of non-dominated solutions obtained GMOEA is superior to SPEA, and GMOEA outperforms SPEA in convergence speed and high accuracy within the same function evaluation times.

(2) GMOEA uses a compact population while SPEA uses a larger number of population, and no sharing or clustering technique is used in GMOEA. Therefore, the actual computation time of GMOEA is lesser than SPEA, because the complexity of identifying the non-dominated solutions is $O(N^2)$.

(3) From the experimental results of GMOEA and GMOEA-N. It is shown that the elite clearing mechanism is capable to encourage the algorithms to explore the unexplored search regions, so that the distribution of solutions can be improved. Moreover, the elite clearing mechanism is simple and efficient than the clustering technique used in SPEA.

5 CONCLUSIONS

Multi-objective process planning problems (MOPPPs) is an important problem in the pre-release planning phase of flexible manufacturing systems. This paper has presented an evolutionary approach using multi-objective evolutionary algorithm with a new elite clearing mechanism for solving MOPPPs. Objectives considering the flow time, machine balancing, machine workload and tool cost are optimized simultaneously. Experimental results demonstrated the proposed approach is suitable to solve the complex industrial problems with a large number of parameters.

References

S.-Y. Ho and X.-I. Chang (1999). An efficient generalized multiobjective evolutionary algorithm *GECCO-99: Proceeding of Genetic and Evolutionary Computation Conference*, pp. 871-878.

H. Tempelmeier and H. Kuhn (1993). *Flexible manufacturing systems: decision support for design and operation*. John Wiley & Sons.

B. Awadh, N. Sepehri and O. Hwalwshka (1995) A computer-aided process planning model based on genetic algorithms. *Computers and Operations Research*, 22(8): 841-856.

C. Moon, Y.-Z. Li and M. Gen (1998). Evolutionary algorithm for flexible process sequencing with multiple objectives. *Proceeding of IEEE International Conference on Computational Intelligence*, pp. 27-32.

E. Zitzler and L. Thiele (1999). Multiobjective evolutionary algorithms: a comparative case study and the strengthen Pareto approach. *IEEE Transaction on Evolutionary Computation*, **3**(4): 257-271.

P. Brandimarte (1999). Exploiting process plan flexibility in production scheduling: a multi-objective approach. *European Journal of Operational Research* (114): 59-71.

D. E Goldberg (1989). *Genetic Algorithms in search, Optimization and Machine Learning,* Addison – Wesley Publishing Company.

G. Taguchi and S. Konishi (1987). *Orthogonal Arrays and Linear Graphs. Dearbon, MI: American Supplier Institute*.

Table 3: Transportation time of AGV from machine to machine.

-: Represent there are no routing path between machines.

From\To	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	4	11	17	8	15	16	14	-	6	8	13	12	16	6	-	8	17	16	8	14
2	6	3	15	9	17	8	7	16	3	6	18	-	3	6	17	-	13	11	-	8
3	6	18	5	7	13	11	15	5	17	14	17	13	11	12	16	13	6	9	10	12
4	17	15	16	3	5	4	-	13	13	4	3	3	7	17	16	6	12	11	4	6
5	13	15	4	11	4	7	4	18	15	8	10	-	17	11	12	18	17	8	9	16
6	11	8	15	8	13	5	17	12	13	16	5	14	11	16	17	16	15	-	9	18
7	4	-	18	9	13	5	3	5	3	9	10	18	15	12	6	7	3	-	-	11
8	9	16	18	16	9	14	8	3	-	18	13	11	16	3	-	6	16	11	-	3
9	12	-	5	7	12	17	5	11	4	-	7	18	7	17	11	4	11	9	15	9
10	-	9	3	11	9	-	-	3	7	3	7	13	18	3	15	10	17	6	16	9
11	9	17	13	12	-	5	8	10	-	18	4	14	-	15	14	-	8	15	10	4
12	4	14	6	15	3	17	3	4	3	7	6	3	-	12	-	15	18	12	18	4
13	-	4	12	7	16	10	4	17	17	18	12	-	3	18	4	3	8	15	11	-
14	13	4	15	-	12	4	15	15	5	8	9	8	4	3	15	17	8	-	8	4
15	4	-	17	18	4	10	-	18	16	10	18	16	9	12	4	10	13	8	12	18
16	3	4	18	10	6	4	3	11	7	9	-	15	12	17	9	3	4	11	6	11
17	-	11	17	15	6	5	4	-	8	12	10	9	16	3	-	18	4	8	-	5
18	15	6	9	6	14	6	17	14	5	-	9	10	17	3	3	3	5	3	12	6
19	3	10	9	10	16	-	15	18	-	4	13	9	-	18	11	5	3	12	4	13
20	5	15	-	18	16	17	12	13	-	17	-	16	5	-	16	-	18	5	16	3

A Genetic Algorithm for the P-Median Problem

Elon Santos Correa
Departamento de Matematica
Colegio Militar de Curitiba
Timoteo Jose Ferreira, 72
Curitiba-PR, Brazil
ZIP Code: 82600-590
Tel. (55) (41) 256-5917
elonsc@yahoo.com
www.geocities.com/elonsc

Maria Teresinha A. Steiner
Departamento de Matematica
Universidade Federal do Parana
Centro Politecnico
Curitiba-PR, Brazil
ZIP Code: 81531-990
Tel. (55) (41) 361-3403
tere@mat.ufpr.br

Alex A. Freitas
Departamento de Informatica
Pontificia Universidade Catolica do Parana
Imaculada Conceicao, 1155
Curitiba-PR, Brazil
ZIP Code: 80215-901
Tel. (55) (41) 330-1669
alex@ppgia.pucpr.br
www.ppgia.pucpr.br/~alex

Celso Carnieri
Departamento de Matematica
Universidade Federal do Parana
Centro Politecnico
Curitiba-PR, Brazil
ZIP Code: 81531-990
Tel. (55) (41) 361-3403
carnieri@mat.ufpr.br

Abstract

Facility-location problems have several applications in telecommunications, industrial transportation and distribution, etc. One of the most well-known facility-location problems is the p-median problem. This work addresses an application of the capacitated p-median problem to a real-world problem. We propose a genetic algorithm (GA) to solve the capacitated p-median problem. The proposed GA uses not only conventional genetic operators but also a new heuristic “hypermutation” operator proposed in this work. The proposed GA is compared with a tabu search algorithm.

Keywords: facility location, p-median problem, genetic algorithms, tabu search.

1 INTRODUCTION

Facility-location problems have several applications in telecommunications, industrial transportation and distribution, etc. One of the most well-known facility-location problems is the p-median problem. This problem consists of locating p facilities in a given space (e.g. Euclidean space) which satisfy n demand points in such a way that the total sum of distances between each demand point and its nearest facility is minimized. In the non-capacitated p-median problem, one considers that each facility candidate to median can satisfy an unlimited number of demand points. By contrast, in the capacitated p-median problem each candidate facility has a fixed capacity, i.e. a maximum number of demand points that it can satisfy. The p-median problem is *NP-hard* [Kariv and Hakimi, 1979]. Therefore, even heuristic methods specialized in solving this problem require a considerable computational effort.

In this work we apply the capacitated p-median problem to a real-world problem, namely the selection of facilities for a university’s admission examination. The goal is to select 26 facilities among 43 available facilities. Each facility has a fixed capacity, i.e. a maximum number of students who can take an exam at that facility. Each student must be assigned to exactly one facility. The selected facilities must satisfy 19710 candidate students (i.e. students who have applied to the university’s admission exam). In addition, the 26 facilities must be selected in such a way that the total sum of the distances between each student’s home and the facility to which the student is assigned is minimized.

In order to solve this problem we propose a genetic algorithm (GA) specific for the capacitated p-median problem. The proposed GA is compared with a tabu search algorithm proposed by Glover (unpublished work).

This paper is organized as follows. Section 2 formally defines the p-median problem and the real-world application addressed in this work. Section 3 introduces the proposed GA. Section 4 reports computational results. Section 5 discusses related work. Finally, section 6 concludes the paper.

2 THE P-MEDIAN PROBLEM

Informally, the goal of the p-median problem is to determine p facilities in a predefined set with n ($n > p$) candidate facilities in order to satisfy a set of demands, so that the total sum of distances between each demand point and its nearest facility is minimized. The p facilities composing a solution for the problem are called medians.

Formally, assuming all vertexes of a graph are potential medians, the p-median problem can be defined as follows. Let $G = (V, A)$ an undirected graph where V are the vertexes and A are the edges. The goal is to find a set of vertexes $V_p \subset V$ (median set) with cardinality p, such that the sum of the distance between each remaining vertex in $\{V - V_p\}$ (demand set) and its nearest vertex in V_p be minimized.

We present below a formulation of the p-median problem in terms of Integer Programming proposed by Revelle and Swain (1970). This formulation allows that each vertex be considered, at the same time, as demand and facility (potential median), but in many cases (including our real-world application) demand and facilities belong to disjoint sets.

$$\text{Min} \sum_{i=1}^{n} \sum_{j=1}^{n} a_i d_{ij} x_{ij} \tag{2.1}$$

subject to:

$$\sum_{j=1}^{n} x_{ij} = 1, \quad i = 1, 2, ..., n \tag{2.2}$$

$$x_{ij} \le y_j, \quad i, j = 1, 2, ..., n \tag{2.3}$$

$$\sum_{j=1}^{n} y_j = p \tag{2.4}$$

$$x_{ij}, y_j \in \{0, 1\}, \quad i, j = 1, 2, ..., n \tag{2.5}$$

where,

n = total number of vertexes in the graph

a_i = demand of vertex j.

d_{ij} = distance from vertex i to vertex j.

p = number of facilities used as medians.

$$x_{ij} = \begin{cases} 1, \text{if the vertex i is assigned to facility j} \\ 0, \text{otherwise} \end{cases}$$

$$y_j = \begin{cases} 1, \text{if the vertex j is a facility used as a median} \\ 0, \text{otherwise} \end{cases}$$

The objective function (2.1) minimizes the sum of the (weighted) distances between the demand vertexes and the median set. The constraint set (2.2) guarantees that all demand vertexes are assigned to exactly one median. The constraint set (2.3) forbids that a demand vertex be assigned to a facility that was not selected as a median. The total number of median vertexes is defined by (2.4) as equal to p. Constraint (2.5) guarantees that the values of the decision variables x and y are binary (0 or 1).

Assuming all vertexes of a graph are potential medians, the p-median problem can be formally defined as follows. Let $G = (V, A)$ an undirected graph where V are the vertexes and A are the edges. The goal is to find a set of vertexes $V_p \subset V$ (median set) with cardinality p, such that: (a) the sum of the distance between each remaining vertex in $\{V - V_p\}$ (demand set) and its nearest vertex in V_p be minimized; and (b) all demand points are satisfied without violating the capacity restrictions of the median facilities. By comparison with the p-median problem, the capacitated p-median problem has the following additional constraints: (1) Each facility can satisfy only a limited number of demands (capacity restrictions); and (2) All demand points must be satisfied by respecting the capacities of the facilities selected as medians.

2.1 A REAL-WORLD APLICATION

The Federal University of Parana (UFPR), located in Curitiba, Brazil, was founded in 1912 as the first federal Brazilian university. It currently offers 61 undergraduate courses, 84 specialization courses (at the graduate level), 37 M.Sc. or M.A. courses and 21 Ph.D. courses. Undergraduate students are selected via a written admission exam applied to all candidate students. For the 2001 admission exam it has been proposed an optimization in the assignment of candidate students to the facilities where they will take the exam. The goal was to assign 19710 candidate students to facilities as close as possible to their corresponding homes. (In order to obtain the distance between each candidate student's home and each facility, all the addresses in question have been precisely located in a digitized map of the city of Curitiba). It was previously determined, for operational and economic reasons, that an algorithm should select 26 facilities to satisfy all 19710 candidate students, among a set of 43 candidate facilities. We cast this problem as a capacitated p-median problem, as follows:

1. The set of 43 facilities (potential exam locations) is the set V ($|V| = 43$) of all facilities candidate to median (actual exam locations).
2. Let $Vp \subset V$ ($|Vp| = 26$) be the set of the 26 selected exam locations.
3. Each of the 43 potential exam locations can satisfy only a limited number of candidate students.
4. The goal is to select a set $Vp \subset V$ that minimizes the total sum of distances between each candidate student's home and its nearest exam location (median).

3 THE PROPOSED GA

This section describes our proposed GA for the capacitated p-median problem, Cap-p-Med-GA.

3.1 INDIVIDUAL REPRESENTATION

Each individual (chromosome) has exactly p genes, where p is the number of medians, and the allele of each gene represents the index (a unique id number) of a facility selected as median. For instance, consider a problem with 15 facilities (potential medians) represented by the indexes 1,2,...,15. Suppose one wants to select 5 medians.

In our GA, the individual [2, 7, 5, 15, 10] represents a candidate solution for the problem where facilities 2, 5, 7, 10 and 15 have been selected as medians. In Cap-p-Med-GA the genome is interpreted as a set of facility indexes, in the mathematical sense of set - i.e. there are no duplicated indexes and there is no ordering among the indexes.

3.2 FITNESS EVALUATION

In essence, the fitness of an individual is given by the value of the objective function for the solution represented by the individual - as measured by formula (2.1). However, there is a caveat in the computation of the fitness of an individual. Note that Cap-p-Med-GA is used only to optimize the choice of the 26 medians, out of the 43 facilities. However, the computation of formula (2.1) requires that each of the 19710 candidate students be assigned to exactly one of the selected medians (i.e. the facility where the student will take the admission exam).

This assignment is done by a procedure that is used by Cap-p-Med-GA as a black box. Since this procedure is orthogonal to the use of a GA, it will not be described in detail here. For details the reader is referred to Correa (2001). Here we just mention the basic idea of this procedure. Once the 26 medians are selected, this procedure tries to assign each candidate student to the median (exam location) that is the nearest one to its home. The problem is that, since each median has a fixed capacity, some candidate students will have to be assigned to the second (or third, fourth, ...) nearest median to their homes. Suppose there is an assignment conflict - e.g. there is just one vacancy in one median, and that median is the nearest one for two candidate students. In this case the student-assignment procedure prefers to assign to that median the student that would be most prejudiced if she was assigned to its second nearest median. A student is "prejudiced" to the extent of the difference between two distances, namely the distance between her home and her nearest median and the distance between her home and her second nearest median. Once the student-assignment procedure is complete, the fitness of an individual can be computed by formula (2.1).

3.3 SELECTION

We use a ranking-based selection method proposed by Mayerle (1996), given by the formula below.

$$\text{Select(R)}=\left\{ r_j \in R \,/\, j = P - \left\lfloor \frac{-1+\sqrt{1+4.rnd(P^2+P)}}{2} \right\rfloor \right\}, \qquad (3.1)$$

where R is a list R = (r_1, r_2, ..., r_p), with P individuals sorted in increasing order by fitness value, rnd $\in$ [0, 1) is a uniformly-distributed random number and the symbol $\lfloor b \rfloor$ denotes the greatest integer smaller than or equal to b. Formula (3.1) returns the position in the list R of the individual to be selected. The formula is biased to favor the selection of individuals in early positions of the list - i.e. the best (smallest fitness) individuals.

The population evolves according to the steady-state method. The offspring produced by crossover (and possibly mutation) is inserted into the population only if they have a better (smaller) fitness than the worst individual of the current population.

3.4 CROSSOVER

As a preprocessing step for the possible application of crossover, Cap-p-Med-GA computes two exchange vectors, one for each parent, as follows. For each gene of parent 1, Cap-p-Med-GA checks whether the allele (facility index) of that gene is also present (in any position) at the genome of parent 2. If not, that facility index is copied to the exchange vector of parent 1. This means that facility index may be transferred to parent 2 as a result of crossover, since this transfer would not create any duplicate facility indexes in parent 2's genotype. The same procedure is performed for each facility index in the genotype of parent 2. For instance, let the two parents be the facility-index vectors [1, 2, 3, 4, 5] and [2, 5, 9, 10, 12]. Their respective exchange vectors are: vp_1 = [1, 3, 4] and vp_2 = [9, 10, 12]. Once the facility indexes that can be exchanged have been identified, the crossover operator can be applied, as follows.

No fixed crossover probability is used in Cap-p-Med-GA. Crossover is performed whenever the two parents are not equal to each other, i.e. whenever there is at least one facility index in the exchange vectors of parent 1 and parent 2. If the two parents are equal to each other, i.e. their exchange vectors are empty, one of the parents is reproduced unaltered for the next generation and the other parent is deleted, to avoid that duplicate individuals be inserted into the population.

Crossover is performed as follows. A random natural number c, varying from 1 to the number of elements in the exchange vectors minus 1, is generated. This number c determines how many facility indexes of each exchange vector will be actually swapped between the two parents. We emphasize that this procedure guarantees that there will be no duplicate facility index in any of the two children produced by crossover.

3.5 MUTATION

Mutation is performed as follows. The gene being mutated has its current allele replaced by another randomly-generated allele (a facility index), subject to the restriction that the new facility index is not present in the current genotype of the individual.

3.6 HEURISTIC HYPERMUTATION

This is a new heuristic operator proposed in this work. It is based on knowledge about the problem being solved.

This operator is applied right after the random generation of the initial population, and after that it is applied with a fixed probability (e.g. 0.5%) to each iteration of the steady-state method (i.e. each selection of two parents, possibly followed by crossover and conventional mutation). This operator starts by randomly selecting a percentage (e.g. 10%) of the individuals of the population. Then it tries to improve the fitness of each of the selected individuals as follows. For each gene of the individual, it tries to replace its facility index by each facility index that is not currently present in the genotype of the individual. For each gene, the replacement that most improves the individual's fitness is performed. Note that this is a very computationally expensive operator, since each time it is applied a large number of fitness functions needs to be performed. The cost-effectiveness of this application-specific, computationally-expensive operator will be evaluated in section 4.

More precisely, the heuristic hypermutation operator proposed in this work is implemented as follows:

```
Procedure HYPERMUTATION:
Step 1.
    Randomly select a subset of 10% of the individuals
    from the entire population.
Step 2.
    FOR EACH individual X selected in Step 1  DO
      Let H be the set of facility indexes that are not
      currently present in the genotype of individual X
      FOR EACH facility index "i" included in set H  DO
        BEST = X
        FOR EACH facility index "j" that is currently
        present in the genotype of the individual X  DO
          Let Y be a new individual with the set of
          facilities given by: $(X - \{j\}) \cup \{i\}$
          Calculate the fitness of Y
          If  fitness(Y) < fitness(BEST) then
            BEST = Y
        END FOR
        If fitness(BEST) < fitness(X) then
          X = BEST
      END FOR
      Insert the new X into the population,  replacing the
      old X
    END FOR
```

To illustrate the use of the hypermutation operator, consider a very simple example with only 5 facilities, labeled {1, 2, 3, 4, 5}, out of which we want to select 3 medians. Consider an individual *X*, selected to undergo hypermutation, containing the facilities {1, 4, 5}. Hence, the set *H* is the set {2, 3}, and *BEST* = *X* = {1, 4, 5}. The algorithm first let $j = 2$, so that the following new individuals are evaluated: {2, 4, 5}, {1, 2, 5} and {1, 4, 2}. Suppose the best of these 3 individuals is {1, 2, 5}, which is also better than the original {1, 4, 5}. Then the algorithm let *BEST* = {1, 2, 5}. At this point the algorithm let $j = 3$, so that the following new individuals are evaluated: {3,2, 5}, {1, 3, 5}, {1, 2, 3}. Suppose the best of these 3 individuals is {3,2,5}, but this individual is not better than the previously best individual {1,2,5}. Then *BEST* remains associated with the individual {1,2,5}. At this point all indexes in *H* have been tried, so the current value of *BEST*, {1,2,5}, replaces the original individual *X* in the population. This process is performed for each individual undergoing hypermutation.

4 COMPUTATIONAL RESULTS

As mentioned earlier, the problem being solved consists of selecting 26 medians out of 43 facilities. Therefore, there are C_{43}^{26} = 421,171,648,758 (roughly 421 billion) candidate solutions.

The proposed GA was evaluated by comparing it with another heuristic algorithm developed for the problem, namely a tabu search algorithm. The tabu search algorithm used here is our implementation of the algorithm proposed by (Glover, personal communication). In essence, this tabu search algorithm works as follows.

Consider the set V of all candidate facilities and $V_p \subset V$, $|V_p| = p$, the initial set of randomly-selected medians. Each "move" (operator) of the tabu search is a procedure that consists of adding (ADD), removing (REMOVE) or swapping (SWAP) in V_p the median that leads to the best (smallest) value of the objective function (2.1). The moves of adding, removing and swapping are sequentially performed, so that the number of medians in the set V_p, will vary in the range: $p - 1 \le |V_p| \le p + 1$.

This phenomenon is called "strategic oscillation". It helps to avoid a convergence to a local optimum.

The ADD, REMOVE and SWAP moves are implemented as follows:

Procedure ADD:

> Select a candidate facility from $\{V - Vp\}$ which when added to Vp results in the best possible value of solution. Then add this candidate facility to Vp. Note that each ADD move considers $|V - Vp|$ facilities as candidate to be added to the current solution (i.e. 17 or 18 facilities for the real-world problem addressed in this work).

Procedure REMOVE:

> Select a median from Vp which when removed from Vp results in the best possible value of solution. Then move this median into $\{V - Vp\}$ (removing it from Vp). Note that each REMOVE move considers $|Vp|$ medians as candidate to be removed from the current

solution (i.e. 26 or 27 medians for the real-world problem addressed in this work).

Procedure SWAP:

Select two facilities, one median from Vp and one facility from $\{V - Vp\}$, which, when swapped, result in the greatest improvement in the feasible solution value (all possible pair-wise exchanges are considered). Each SWAP move considers $|Vp|$ x $|V - Vp|$ pairs of facilities as candidate to be swapped (i.e. 26 x 17 = 442 candidate pairs for the real-world problem addressed in this work).

A tabu list memorizes the number of the iteration in which each median was added to a solution. During a certain number of iterations (called tabu tenure), it is forbidden to re-insert that median to the current solution, i.e. the corresponding move is a tabu (forbidden) move. The aspiration criterion used consists of allowing the tabu restriction to be ignored if the quality of the new solution produced by a tabu move is better than the quality of the best solution generated up to now by the search.

For a comprehensive, detailed discussion about tabu search in general the reader is referred to the book by Glover and Laguna (1997).

The experiments involved a comparison between two versions of Cap-p-Med-GA and the above-described tabu search algorithm. The first version of Cap-p-Med-GA is a full version of the algorithm, using all the genetic operators described in section 3. This version can be considered a hybrid GA/local search algorithm, since the heuristic hypermutation operator effectively incorporates problem-dependent knowledge into the GA. By contrast, the second version of Cap-p-Med-GA is a pure GA, which was obtained by simply switching off the heuristic hypermutation operator - i.e. this operator is never applied. In other words, it uses all the genetic operators described in section 3 except the heuristic hypermutation operator. This second version of Cap-p-Med-GA was included in our experiments to evaluate the cost-effectiveness of our proposed heuristic hypermutation operator in a controlled manner.

All results reported in this section were obtained on a Pentium III PC with 550MHz and 128 Mbytes of RAM. In order to make the comparison between the three algorithms (the two versions of Cap-p-Med-GA and the tabu search) as fair as possible, we have carefully determined the number of iterations performed by each algorithm in such a way that all the three algorithms evaluate roughly the same number of candidate solutions. This is fair because in the three algorithms the majority of processing time is by far taken by candidate-solution evaluation. More precisely, the algorithms' parameters determining the number of evaluated candidate solutions were set as follows:

Cap-p-Med-GA with heuristic hypermutation:

Population Size = 100
Number of iterations = 1000
Probability of conventional mutation = 1%
Probability of heuristic hypermutation = 0.5%
Number of individuals that are selected for undergoing hypermutation = 10% of Population Size = 10

Cap-p-Med-GA without heuristic hypermutation:

Population Size = 100
Number of iterations = 12100
Probability of conventional mutation = 1%

Tabu Search

Number of iterations = 150
Tabu tenure = 10

Note that Cap-p-Med-GA without heuristic hypermutation performs many more iterations than Cap-p-Med-GA with heuristic hypermutation, to compensate for the fact that, when heuristic hypermutation is applied at a given iteration, a very large number of candidate solutions are evaluated in that iteration. The small number of iterations of tabu search also reflects that fact that in a single iteration of the search (consisting of all possible adding, removing and swapping moves) many different candidate solutions are evaluated.

The computational results obtained by the three algorithms are reported in Table 4.1.

Table 4.1: Computational Results

	GA with heuristic hypermutat.	GA without heuristic hypermutat.	Tabu search
No. of eval. solutions	24,200	24,300	24,301
run time	01:43:34	01:43:21	01:23:37
average distance	2.33 Km	2.40 Km	2.37 Km
total distance	45,999 Km	47,313 Km	46,660 Km
% nearest facility	83%	79%	82%

The first row of Table 4.1 indicates the number of candidate solutions evaluated by each algorithm. The second row indicates the run time taken by each algorithm, in the format hours:minutes:seconds. Note that the three algorithms had about the same run time. This is a result of our having carefully determined the number of iterations of each algorithm so that each one evaluates roughly the same number of candidate solutions, as mentioned above. Therefore, a comparison among the three algorithms with respect to the quality of their produced solution is fair. The other rows of Table 4.1 are indicators of quality of the produced solution, as follows.

The third and fourth rows report respectively the average and total distance traveled by the students, measured in Km. The distance traveled by each student is the distance between the student's home and the facility (median) to which the student was assigned. The average distance is simply the total distance traveled by all 19710 students divided by 19710. The fifth row reports the percentage of students that were assigned to the facility that is indeed the facility nearest to the student's home, which is the ideal assignment for a student. Overall the three algorithms did a good job, managing to assign about 80% of the students to their ideal (nearest) facility.

With respect to both the minimization of average (or total) distance traveled by students and maximization of the percentage of students assigned to their nearest facility, the best algorithm was Cap-p-Med-GA with the heuristic hypermutation operator. The second best algorithm was tabu search. The worst algorithm was Cap-p-Med-GA without the heuristic hypermutation operator. Therefore, these results are evidence (in this application) for the cost-effectiveness of extending a conventional GA with a problem-dependent, heuristic operator.

5 RELATED WORK

Hosage and Goodchild (1986) (H&G) seem to have been the first researchers to develop a GA for the p-median problem. They used a simple GA, with conventional genetic operators. Each candidate solution was represented by a binary string, where each bit corresponds to a facility index. Each allele (1 or 0) indicates whether or not the corresponding facility is selected as a median. If the number of bits set to "1" is different from p the solution is deemed invalid and a penalty (proportional to the extent of restriction violation) is applied to the fitness of the individual. H&G tested their GA in a problem where the goal was to select 3 medians out of 20 facilities (i.e. $n = 20$, $p = 3$). They used a population of 25 individuals ($P = 25$), and did experiments with different numbers of generations, varying from 120 to 210. In experiments with randomly-generated problem instances, the GA obtained the optimal solution in about 70% and 90% of the problem instances, when running the GA for 120 and 210 generations, respectively. At first glance these are good results. However, the GA uses a classic binary individual representation, which is not very suitable for this problem. It wastes memory and processing time. The problem instances used to evaluate the algorithm had only 1140 candidate solutions (C_{20}^{3}). However, the GA generates and evaluates 2905 and 5065 solutions, when it is run for 120 and 210 generations, respectively. Although there are only 1140 candidate solutions, the search space for the GA is 2^{20} (all possible binary strings of length 20). Roughly 99.9% of the possible individuals are invalid solutions, and the GA wastes time analyzing them. Our work clearly avoids this problem, since the individual representation used in our work considers only candidate solutions with exactly the desired number of medians.

Dibble and Densham (D&D) (1993) proposed a GA with an individual representation more suitable for the p-median problem. Each individual has exactly p genes, and each gene represents a facility index. This is the same representation as the one used in our work. They used only conventional genetic operators. By contrast, we have developed a problem-dependent operator for the p-median problem, as discussed earlier. D&D applied their GA to a problem where the goal was to select 9 medians among 150. They used population size $P = 1000$ and 150 generations. They compared the results of their GA with the results obtained by the heuristic algorithm of Teitz and Bart (1968), which is a heuristic algorithm specialized for the p-median problem. Although the GA took a considerably longer processing time, both algorithms produced similar solutions.

Moreno-Perez et al. (1994) also developed a GA for the p-median problem. The individual representation is the same as the one used by D&D. They used only conventional genetic operators. Once again, this is in contrasts with our work, which proposed a problem-dependent operator for the p-median problem, as discussed earlier. One distinguishing feature of the GA proposed by Moreno-Perez et al. is that they used multiple population groups (colonies), which exchange candidate solutions with each other (via migration). The authors claim that this method helps to avoid premature convergence to a local optima. In the above reference the authors did not compare their proposed GA with any other algorithm, so it is difficult to say how cost-effective the algorithm is.

Erkut et al. (2001) also developed a GA for the p-median problem. Each individual also has exactly p genes representing a set of p selected medians. In addition to conventional genetic operators, they use the "*String-of-Change Operator*" independently suggested by Booker (1987) and Fairley (1991). This operator uses a string of change, which consists of a binary vector generated for each parent of a crossover. The parents are passed to an exclusive OR (XOR) operator. The expression a XOR b is defined as 1 if $a \neq b$ and 0 otherwise. For instance, applying XOR to the parents [10, 9, 12, 24, 7, 3] and [10, 9, 7, 8, 12, 3] one would obtain the binary vector [0, 0, 1, 1, 1, 0]. In order to avoid that crossover produces offspring identical to the parents, only the genes between the first "1" and the last "1" in the parents can be selected as crossover points.

The basic idea of this string-of-change operator is conceptually similar to the exchange vector used in our work. However, we believe our exchange vector is more suitable for the p-median problem, based on the following rationale. In order to identify the facility indexes that can be swapped between the parents, our exchange vector mechanism considers that each individual contains a (*unordered*) *set* of facility indexes. By contrast, the string-of-change, XOR mechanism considers that each individual contains a (*ordered*) *list* of facility indexes. For instance, in the above example, the facility indexes 12 and 7 were identified as possible crossover points by the string-of-change operator, despite the fact that they are present in both parents, since the position of their occurrence in the genotype is different in the two parents. By contrast, those two facility indexes would not be included in our exchange vector, since they occur in both parents. After all, the position of a facility index in the genotype is arbitrary, from the viewpoint of specifying a candidate solution. E.g., the *set* of medians {7, 12} represents the same solution as the *set* of medians {12, 7}, which is not recognized by the string-of-change operator.

6 CONCLUSIONS AND FUTURE WORK

We have proposed a GA for the capacitated p-median problem, and have applied it to a real-world problem with a quite large search space, containing roughly 421 billion ($4,21 \times 10^{11}$) candidate solutions. Our GA uses an individual representation and genetic operators developed specifically for the p-median problem.

In particular, we have proposed a heuristic hypermutation operator, to be used in addition to crossover and conventional mutation operators. We did experiments comparing two versions of our GA, one with this new operator and the other one without it, with a tabu search algorithm. The results show that: (a) the tabu search algorithm outperforms the GA without the heuristic hypermutation operator; but (b) the GA with the proposed heuristic hypermutation operator outperforms the tabu search algorithm. These results are evidence for the cost-effectiveness of the proposed heuristic operator, since all three algorithms evaluated roughly the same number of candidate solutions during their search. The user considered the solution produced by the GA (with the heuristic operator) very satisfactory.

Some directions for future research are as follows. Concerning the p-median problem, it seems worthwhile to develop new algorithms for this problem based on relatively new heuristic algorithms, such as *Scatter Search* and *Path Relinking*. These new heuristic algorithms, also related to evolutionary algorithms, have produced better results than GAs and tabu search in some combinatorial optimization algorithms (Glover, 1999).

Concerning the real-world application problem addressed in this paper, it would be interesting to extend the problem definition to find high-quality solutions (i.e. keeping the distance traveled by the students as small as possible) with a smaller number of selected medians. This would lead to a reduction in the costs of application of the university's admission exam, without increasing too much the distance traveled by the students. Going further, a more elaborated algorithm could perhaps directly consider the trade-off between minimizing the distance traveled by the students (which suggests selecting a larger number of medians) and minimizing the costs of the admission exam (which suggests selecting a smaller number of medians). Finally, from a GA viewpoint, an interesting research direction is to investigate whether the heuristic hypermutation operator proposed in this work can be adapted to work, in a cost-effective manner, with other combinatorial optimization problems.

ACKNOWLEDGMENTS

We are very grateful to Fred Glover and Erhan Erkut, for having sent us papers on tabu search and GA algorithms applied to the p-median problem. In particular, as mentioned in the text, Fred Glover suggested us the tabu search algorithm implemented in this work.

REFERENCES

BOOKER, L. B. Improving Search in Genetic Algorithms. In: Genetic Algorithms and Simulated Annealing (Edited by L. Davis), 61-73, Morgan Kauffmann, Los Altos, CA, 1987.

CORREA, Elon S. Algoritmos Geneticos e Busca Tabu Aplicados ao Problema das P-Medianas. (In Portuguese) M.Sc. Mestrado em Metodos Numericos em Engenharia. Universidade Federal do Parana (UFPR). Curitiba, Brazil. 2000.

DIBBLE, C.; DENSHAM, P. J. Generating Interesting Alternatives in GIS and SDSS Using Genetic Algorithms. *GIS/LIS* symposium, University of Nebraska, Lincoln, 1993.

ERKUT, Erhan; BOZKAYA, Burçin; ZHANG, Jianjun. An Effective Genetic Algorithm for the p-median Problem. (In press.) 2001.

FARLEY, A. Comparison of Choosing the Crossover Point in the Genetic Crossover Operation. Technical Report. Dept. of Computer Science, University of Liverpool, 1991.

GLOVER, Fred; LAGUNA, Manuel. Tabu Search. *Kluwer Academic Publishers*, University of Colorado, 1997.

GLOVER, Fred. Scatter Search and Path Relinking. Graduate School of Business, University of Colorado, Boulder, *Technical Report*, 1999.

GLOVER, Fred. Tabu Search for the p-median Problem. (Unpublished paper).

GOLDBERG, David E. *Genetic Algorithms in Search, Optimization and Machine Learning*. Menlo Park, CA: Addison-Wesley, 1989.

GOODCHILD, M. F.; NORONHA, V. Location-Allocation for Small Computers. University of Iowa, *Monograph* number 8, 1983.

HOSAGE, C. M.; GOODCHILD, M. F. Discrete Space Location-Allocation Solutions from Genetic Algorithms. *Annals of Operational Research*, 6, 35-46, 1986.

KARIV, O.; HAKIMI, S. L. The p-median problems. In: An Algorithmic Approach to Network Location Problems. SIAM *Journal on Applied Mathematics*, Philadelphia, 37, 539-560, 1979.

MAYERLE, S. F. Um algoritmo genetico para o problema do caixeiro viajante. (In Portuguese) Technical Report. Florianopolis-SC, Brazil: UFSC, 1996.

MORENO-PEREZ, J. A.; MORENO-VEGA, J. M.; MLADENOVIC, N. Tabu Search and Simulated Annealing in p-median Problems. *Talk at the Canadian Operational Research Society Conference*, Montreal, 1994.

REVELLE, C.; SWAIN, R. Central Facilities Location. *Geographical Analysis*, 2, 30-42, 1970.

TAITZ, M. B.; BART, P. Heuristic Concentration: Two-Stage Solution Construction. *Operational Research Society*, London, 16, 955-961, 1968.

Global geometry optimization of atomic and molecular clusters by genetic algorithms

Bernd Hartke
Institut für Theoretische Chemie, Universität Stuttgart
Pfaffenwaldring 55, 70569 Stuttgart, Germany
e-mail: hartke@theochem.uni-stuttgart.de, phone: +49-711-6854409

Abstract

One of the numerous NP-hard global optimization problems in theoretical chemistry is that of finding the global minimum energy structure of an atomic or molecular cluster. This paper describes our method of solving this problem with specialized genetic algorithms, both on the level of empirical model potentials and in combination with extremely expensive calculations from first principles. Discussing several current application examples, we show that our approach is a valuable new tool in hot topics of cluster chemistry.

1 Introduction

With its basic theoretical foundations known since the first half of the 20th century, chemistry may easily be mistaken for a "closed" science. However, the reduction of chemistry to the quantum mechanics of electrons and nuclei has also emphasized that the complexity of chemistry cannot be due to the complexity of its basic laws. In fact, all chemically relevant forces on the elementary particles are simple Coulomb attractions and repulsions between point charges. Instead, the complexity of chemistry results from the complexity of many-body interactions prevalent in chemical systems. Therefore, it comes as no surprise that many problems in theoretical chemistry turn out to be NP-hard [1, 2], with exact solutions requiring computational resources that increase exponentially with the number of particles. This problem class includes electronic structure determination [3], protein folding [4, 5], and cluster structure determination.

Clusters are assemblies of a few to many atoms or molecules [6]. They are not called molecules themselves since the forces between their constituent particles need not be chemical bonds in the usual sense but can also be hydrogen bonds or van-der-Waals "non-bonded interactions". In recent years, clusters have been recognized to play central roles in many everyday processes, up to the destruction of ozone via heterogeneous catalysis on the surface of water–sulphuric acid clusters in polar stratospheric clouds. Also, a proper understanding of cluster properties will be instrumental for the upcoming nanotechnology.

More basically, clusters serve as a bridge between single atoms or molecules and the "infinitely" extended solid. They are not simply small pieces of solids, neither in their structure nor in their physical or chemical properties. With the number n of particles increasing, one usually observes several transitions in cluster structure and properties. The value of n for the onset of solid state structure and properties depends both on the system and on the property under study. Also, clusters with particular values of n often show enhanced stability or a particularly low reactivity, which is usually explained as shell closure, following some particular structural principle. None of these basic issues has been properly resolved or understood as yet.

Clearly, an investigation into cluster properties cannot even begin without a proper understanding of cluster structures. Arguably, the cluster structure most likely to be found in experiments is that with the globally minimal potential energy. This hypothesis ignores quantum mechanical zero-point energy and finite temperature effects, which turn out to be important only in a few cases (e.g. for the water hexamer [7]). More importantly, it also ignores cluster formation dynamics, which may trap the cluster in a local minimum, depending on the experimental formation conditions (e.g. again for the water hexamer [8]). However, a statistically correct and quantum mechanically accurate simulation of the formation dynamics of clusters of non-trivial size is far beyond today's supercomputer capabilities. Luckily, current experience indicates that experiments produce clusters typically in their global minimum energy structures; local minimum structures are sometimes found but these are then always very close in energy to the global minimum.

Independent of the forces (and hence the potential) acting between the particles, it has been shown that finding the structure of an atomic cluster with globally minimal energy is an NP-hard problem [1,9,10]. In line with this result, exact global optimization methods so far could be applied only up to cluster sizes which are trivial for the heuristic methods considered in this paper. For example, a DC transformation approach [11] was managable in exact form only up to $n = 7$, while a deformation (homotopy) method [12] managed to reach (the still trivial) $n = 31$ but failed for three smaller cases. The `PHENIX` method presented in this paper has found the correct global minima of atomic clusters up to $n = 250$ within a few CPU days on a personal computer.

In molecular clusters (even if the molecules are treated as rigid bodies), the additional orientational degrees of freedom of the molecules provide another exponentially increasing search space [13], not separable from the first one constituted by the positional coordinates. Nevertheless, the `PHENIX` method presented here is feasible even for such systems, albeit only up to about $n = 25$ (with a comparable amount of computational resources).

The results mentioned so far deal with the situation that the forces between the particles are computed very quickly as analytical derivatives of a given empirical model potential of simple functional form. This is actually a strong approximation: As mentioned above, the forces between the particles should properly be calculated from first principles quantum mechanics (ab-initio). Since this is, however, yet another complex many-body problem, such a force calculation is typically 5–6 orders of magnitude more expensive, even with state-of-the-art quantum chemical methods. This would inflate the typical running time of our algorithm from about 1.5 CPU hours for $n = 60$ with an empirical potential to about 1000 CPU years on the ab-initio level. Nevertheless, some groups have applied brute-force simulated annealing in combination with ab-initio calculations to the problem of global cluster geometry optimization [14–17]. However, these applications always had characteristic deficiencies: Either the ab-initio treatment was not sufficiently accurate, or the reliability in locating global minimum structures remained in doubt, or the clusters considered were simply too small to make global optimization methods really necessary. In fact, to the knowledge of the present author, typical global optimization developers always implicitly assume that the cost function is cheap to evaluate; the case of a very expensive cost function is not considered, presumably because such a problem appears to be intractable from the outset. However, since we are facing exactly such a problem here, the present author has proposed and successfully applied an iterative procedure `GAGA` that solves this intractable problem by circumventing it.

The following section 2 mentions briefly the development and context of these two methods `PHENIX` and `GAGA`, and then presents their inner workings in detail, in relation to standard genetic algorithm (GA) paradigms. Besides giving references to benchmark applications, section 3 is devoted to applications of these methods to real-world physical chemistry examples, in direct relation to cluster experiments.

2 Algorithms

2.1 Brief historical background

The numerous different approaches applied so far to the global cluster geometry optimization problem have been reviewed recently [1] for the benchmark system of Lennard-Jones clusters. Similarly, applications of GAs to chemical problems have also been reviewed a few years ago [18]. This will not be repeated here.

After GAs had been used for the global optimization of dihedral angles in small biomolecules [19], they were employed first by the present author [20] and then also by other groups [21] for the global optimization of all degrees of freedom in atomic and molecular clusters. In these early papers, an actually genetic representation of the optimization problem was used by concatenating the cartesian coordinates of all particles in the cluster, and by operating with standard forms of single-point crossover and mutation on these coordinate strings. As already pointed out in Ref. [20], this representation does not achieve the best possible separability. Nevertheless, adding in local search steps (effectively generating an algorithm that would now be called “memetic”), we were able to push the size scaling of such a GA down to $n^{4.5}$, for the benchmark system of Lennard-Jones clusters [22], but we had to add in a seed growth method to get beyond $n = 20$ within reasonable computer time. Deaven and Ho [23] managed to eliminate the representation issue altogether, by applying crossover and mutation operators directly on the clusters in physical space. This enabled them to treat Lennard-Jones clusters up to $n = 100$ [24], but they arrived at wrong solutions for $n = 75, 76, 77$. Other groups [25] introduced minor variations to this basic recipe, but still could not solve those hard cases without seeds, i.e. without a priori external knowledge. By employing the GA concept of niches temporarily in the selection step, the present author [26] demonstrated that a phenotype algorithm is able to find all currently accepted global minima of Lennard-Jones clusters up to at least $n = 150$ without any use of external prior knowledge. At the same time, the size scaling of the method could be pushed below n^3, further improving the access to larger clusters, now standing at $n = 250$. This easily beats the best competing global optimization algorithm, "basin hopping", which is not based on the GA paradigm [27]: Their algorithm scales like n^5 at best, and hence they could reach only $n = 110$. Recently, our methodology was extended to molecular clusters and applied to the benchmark system of TIP4P water clusters [13].

2.2 Details of the methods

2.2.1 PHENIX

The backbone of our `PHENIX` algorithm is a standard GA with generational replacement. We are using rather small population sizes of typically only $m =$ 10–30 individuals. Following Deaven and Ho [23], we compensate for this limitation by a reorganization of the way the next generation is generated from the previous one: Instead of selecting $m/2$ parent pairs using some combination of fitness and random criteria, we generate all possible unique pairs (even including pseudo-pairs of each individual with a copy of itself). Using a crossover operator, and with a chance of about 15% also a mutation operator (both described below), each pair produces two children. From these larger, intermediate pool of children, m individuals are selected for the actual next generation, using several different fitness measures, see below.

The whole algorithm operates without genetic strings (we call this a "phenotype algorithm", hence the first part of its name). Instead, all operators are applied directly to the clusters, in 3-dimensional physical space. This automatically ensures a relatively high degree of separability of the optimization problem (since the fitness of each cluster is dictated to a large extent by the short-range interactions of its consituents, while long-range interactions only play a minor role), and it also makes the design of application-specific operators easier and more intuitive. The main fitness criterion is the potential energy of each cluster, generated from the position (and orientation) coordinates of its constituent atoms (or molecules) via a given (empirical, classical-mechanical) potential energy function. Accordingly, the main objective is finding the global minimum of this potential energy hypersurface. In the spirit of "memetic" algorithms, local potential energy minimizations of the clusters turn out to be essential for an effective operation of the whole algorithm. Therefore, local minimizations are performed in several places, employing standard conjugate-gradient or quasi-Newton routines: in the production of generation zero, after crossover (and mutation) as last step in the assembly of each new child in each intermediate generation, and after each postprocessing operation (see below) in each generation.

Generation zero is produced by setting all coordinates of all cluster constituents to random numbers, followed by a local minimization of each cluster. In order to avoid having to deal with clusters dissociated into many small parts (such configurations are very unlikely to lead to global minimum structures anyway), we draw random numbers only within a preset coordinate range, thus inevitably placing some initial bias on compact structures. Also, we enforce a minimal distance between atoms (molecules), in order to avoid numerical difficulties in the ensuing local minimization, due to the typcially steeply repulsive branches of most potential energy functions at close pair distances.

Our crossover operator is the generalization of the simple one-point string crossover operator to 3-dimensional physical space: Again following Deaven and Ho [23], each cluster in a pair is cut in two parts by a plane, and two children are made by reassembling these parts in a cross-over fashion. Extending the idea of Deaven and Ho, our cutting plane does not need to pass through the center of mass of the cluster, generating cluster halves. Instead, we allow deviations from a 50:50 partitioning, with a Gaussian distribution around the exact halves, such that even a 90:10 partitioning does occur with a non-negligible frequency. After reassembly of the parts, the distance between them and their relative rotational orientation is optimized, before a full local minimization is applied to the new child cluster. Therefore, crossovers of each cluster with a copy of itself do make sense, and are also done here. Another important extension of the basic idea results in two variants of this operator (one or the other is applied in each instance, chosen at random): in one variant, the orientation of the cutting plane is chosen at random for each cluster; in the other variant, the cutting plane is oriented deterministically, such that it separates the best part of the cluster from the worst part. Best and worst parts can be determined easily, since typical potential energy functions are build from atom (molecule) pair contributions: Half of each pair contribution is assigned to each atom (molecule) in the pair. Of course, this can also be generalized to 3-body and higher potential terms. The sum of all these contributions for one specific cluster constituent is the contribution of this constituent to the total potential energy. This allows the definition of a "center of quality" analogous to a center of mass. The best cluster half is then simply the one where the center of quality resides. Tests have shown that this deterministic crossover tends to generate a very good child (from the two better halves) and a fairly bad one (from the two worse halves), compared to the fitness of the parents, while the random crossover variant tends to generate children with a fitness more similar to each other and to their parents. Best overall algorithm performance is achieved with a roughly 50:50 mixture of both crossover variants, ensuring a suitable balance of exploitation and exploration.

Our mutation operator again is the analogue of its string mutation operator counterpart: A small number of constituents of the cluster is chosen at random (typically not more than 30% of the total). Each of those is moved by a random distance in a random direction (and rotated at random, for the case of molecular clusters), subject to the limitations that such a move must not place the moved constituent far away from the cluster or too close to another constituent.

The selection of the actual next generation from the larger intermediate pool is done with two basic criteria: the usual fitness measure based on minimal potential energy, and strong geometric diversity using (temporary, dynamic) niches (hence the second part of the name of our algorithm). To this end, the whole intermediate pool is sorted by potential energy, and, in addition, each cluster is assigned one (or several) numbers classifying its geometry (see below). Starting from the clusters with the lowest potential energy, each cluster of the intermediate pool is inspected and compared to the clusters already selected into the next generation. If its geometry classification number(s) deviate(s) more than a given difference from the corresponding classification number(s) of the already selected clusters, it is also selected, irrespective of its potential energy, and constitutes a new geometrical niche of its own. If its geometry classification number(s) is (are) closer than a given difference to the one(s) of an already selected cluster (i.e., if it falls within an already established geometrical niche), it is selected only if the number of clusters in this niche does not exceed a given limit and if its potential energy differs from the other clusters in this niche by more than a given amount. This selection process continues until a total of m clusters has been selected (constant overall population size). To ensure a minimum of exploration, there is also a special niche for mutants, which is filled with clusters that were operated upon by the mutation operator; there are no other (geometrical) criteria for this niche, but a minimum energy difference criterion also applies. Note that these geometrical niches differ from more usual ones in several respects: They exist only in the selection step (i.e., there is constant interbreeding between all niches); they are temporary and dynamic also in the sense that they are re-determined from scratch in each generation and then discarded again; and all user input needed for their determination is one measure characterizing the clusters other than their potential energy (or several of these) and two numbers: a niche-defining maximum difference in this measure, and a maximum capacity for each niche. As mentioned in the applications section, the general need for these niches as well as the choice of niche measures depends (and should depend) on the application under study.

It is important to add yet another ingredient to this algorithm: After each new generation is established, it is subjected to a set of postprocessing operators, applied in random selection to each cluster in turn, in an attempt to further refine the selected clusters. One obvious operator is a simple repetition of local minimization but this time with a much tighter threshold (this allows for the use of rather loose thresholds in the more numerous applications of local minimization in the remainder of the algorithm). Other operators became obvious during several different applications: Often the algorithm quickly converges to the vicinity of the correct global minimum solution but then takes a long time to move a very small number of misplaced atoms or molecules into their optimal places. This can be fixed efficiently by a "directed mutation" operator: Again using the above-mentioned partitioning of the total potential energy onto the cluster constituents, a very small number (between one and four) of the worst constituents is removed from the cluster and re-introduced into the most promising positions, discovered by a loose 3-dimensional grid search over the whole surface and interior of the cluster (for molecular clusters, also the optimal orientation is determined before re-introduction). For the case of molecular clusters, our experience agrees with that of the literature [28] in that it is advantageous to treat the position and orientation coordinates not only together but also separately. Therefore, in this case, another important postprocessing operator is a copy of the whole `PHENIX` algorithm operating solely on the orientation coordinates, with the position coordinates held fixed at the values of the particular cluster under study.

2.2.2 GAGA

As mentioned in the introduction, clusters should properly be treated not with empirical potential energy functions but with an explicit ab-initio treatment of their electrons. This is many orders of magnitude more expensive, however, and `PHENIX` as well as every other global optimization scheme currently in existence requires far too many function (and derivative) evaluations for this to be practical and reliable for clustes of interesting sizes. The present author has recently proposed an iterative algorithm `GAGA` that circumvents this problem in a non-rigorous fashion [29]. The main idea is very simple: Global cluster geometry optimization is still done on an empirical model potential. The potential energy of the resulting geometry, however, is then also calculated at a suitable ab-initio level. This information is then used to globally re-optimize the parameters in the model potential, by minimizing the difference between the model and ab-initio potential energies. This generates a modified model potential. On this modified potential, another global cluster geometry optimization is started, etc. This scheme is iterated until the model potential parameters converge or until no new types of cluster geometries appear. All substantially different cluster geometries generated in this fashion (which are global minima on different model potential energy surfaces, but not necessarily minima on the ab-initio level) are then refined by local (!, not global) geometry minimizations on the ab-initio level. The hypothesis is that among the resulting geometries there will also be the global minimum geometry on the ab-initio level. Another result of this combined strategy obviously is an improved model potential.

This is a meta-algorithm, employing two different global minimizations, one for cluster geometries and one for model potential parameters. In principle, it can be used with any global optimization method. We are currently employing GAs in both steps, and hence we call this meta-algorithm `GAGA`.

Obviously, the choice of the model potential energy functional form strongly influences the results of this scheme. However, this restriction is less severe than it may appear at first: Assuming that the global geometry optimization phase is deliberately not pushed to its limits and hence typically finds either the global minimum or a low-energy local minimum, it is actually sufficient if the model potential has a low-energy local minimum in the basin of attraction of the ab-initio global minimum, at least in some not too narrow range of model potential parameters. In practice, this actually turns out to be the case, for model potentials that are not overly simplistic: In all our applications attempted so far [29–31], the `GAGA` scheme was always able to locate the true ab-initio global minimum energy structures. Also, with more basic research into possible physical interactions between cluster constituents, it should be possible to find model potential functional forms that guarantee that the ab-initio global minimum will be found in such an iterative `GAGA` scheme.

3 Selected Applications

3.1 Lennard-Jones clusters

Lennard-Jones (LJ) clusters are a simplified model for noble gas clusters, with the potential energy function being given solely by pairwise van-der-Waals type interactions. LJ clusters have been established as a virtually obligatory benchmark system for global cluster geometry optimization. The very many attempts at treating this system have been reviewed recently [1], and full information (energies and geometries) for accepted global minima are available on the internet [32]. To date, however, only "basin-hopping" [27] and the `PHENIX` method presented here [26] were able to find all global minima in the size range up to $n = 110$ without using biasing a-priori information. The reasons for this have been analyzed in detail by Doye, Miller and Wales [33]: For most cluster sizes n, the potential energy hypersurface is dominated by one geometry species, the Mackay icosahedra. At certain narrow ranges of n, other structural types compete with this dominating type, and for $n = 38$ the fcc type wins, while for $n = 75, 76, 77$ and $n = 102, 103, 104$ the decahedral type wins. However, these other types win only by a very close margin, and most of the potential energy hypersurface is still dominated by the icosahedral types even in these cases, with high energy barriers to the narrow regions were the other geometry types reside. This makes it extremely difficult for all global optimization methods to treat these particular cluster sizes. Both "basin-hopping" and `PHENIX` without niches can treat these cases successfully, but need about two orders of magnitude more computer time for them than for the neighboring cluster sizes. The full `PHENIX` implementation with niches manages these cases almost within the same time as the simpler cases, as already published [26].

As mentioned in the introduction, `PHENIX` with n^3 also has a more favorable size scaling than "basin-hopping" with about n^5. This enables us to treat much larger clusters. Romero et al. [34] have employed a GA-based placement search on given icosahedral, decahedral and fcc lattices, arriving at proposals for global minimum structures up to $n = 309$. Without using such prior information, we performed an explorative `PHENIX` study beyond $n = 150$, by limiting the number of generations to 100. Nevertheless, up to $n = 190$, we managed to reproduce about 50% of the energies given by

Romero et al.; the largest cluster for which we have found agreement so far is $n = 250$. Computer times for finding these values are still tolerable (at most a couple of days on a single-processor PentiumIII PC) and well within the established n^3 size scaling.

Within this study, we also managed to improve upon the global minima originally given by Romero et al. for $n = 185, 186, 187$. They originally proposed decahedral structures for these three cases, with the energies given in Table 1 (energies are reported in units of the pair potential well depth); their structures for $n = 185, 186$ constituted a new geometry subtype for Lennard-Jones global minima, namely that of a decahedral core with an outer layer in fcc positions. Our unbiased search managed to find lower-lying minima of the more usual icosahedral type, thus eliminating this new structural subtype. Our proposal for $n = 185$ was recently again improved by R. H. Leary, to an energy value of -1125.4938.

Table 1: Improving Lennard-Jones minima

size n	Romero et al.	this work
185	-1125.299820	-1125.304876
186	-1132.503199	-1132.669966
187	-1139.240017	-1139.455696

It is also instructive to look at the cases where our algorithm failed to find the structures and energies of Romero et al. (or better ones) within the prescribed 100 generations: Contrary to original expectations in the LJ cluster community, global minimum structures with incomplete inner cores were recently found [35]. Structures of this type were proposed by Romero et al. for $n = 169, 170, 171, 172$, with 4,3,2,1 core holes, respectively. Of these structures, our algorithm correctly found those for $n = 169$ and $n = 171$ within the prescribed 100 generations, but constructed complete cores for the other two cases, leading to higher energies. We could trace this down to the "directed mutation" operator, which this time was overdoing his otherwise vital job of effectively filling core holes. Presumably, a less effective version of this operator or a counterbalancing operator that removes atoms from the core and places them into the outer layer could remedy the situation. In line with the much-discussed "no free lunch" theorems, this stresses the importance of application-specific program development.

For $n = 163$ and $n = 164$, our algorithm finished at a peculiar structural type higher in energy than the geometries proposed by Romero et al.. The same structural type also occured as low-lying local minimum in our optimization runs at many other values of n. At that time, we had not yet heard about the new tetrahedral structural type discovered by Leary and Doye [36] for $n = 98$. Comparing our structures with theirs revealed that they follow exactly the same pattern. Thus, unintentionally and before their discovery, we have shown that our algorithm is able to find also this new, fourth basic structure of LJ clusters.

3.2 Mercury clusters

Along with cluster structures, cluster properties also change with cluster size. For mercury clusters, there even seems to be a change in bonding character from van-der-Waals to covalent around $n = 13$, according to current interpretations of experimental results [37]. Employing our `GAGA` method in conjunction with a quantum-classical hybrid model including relativistic effects, we are currently modelling small mercury clusters in this size range [38]. Preliminary results already indicate that even below the presumed transition size, globally minimal cluster structures violate the usual icosahedral growth characteristic for van-der-Waals interactions. As an example, Fig. 1 depicts our best structures for Hg_7 and Hg_{10}, which are clearly not based on the pentagonal bipyramid (which is the basic motif of icosahedral growth in this size range). However, most theoretical interpretations of mercury cluster experiments have so far assumed icosahedral cluster structures [37] even for larger clusters; apparently, those works now need to be revised. According to our preliminary results, it may be possible to arrive at better agreement with experiment based on our new and entirely unexpected structures.

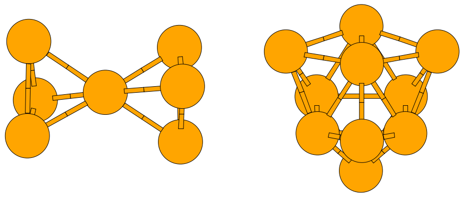


Figure 1: Globally optimal mercury clusters

3.3 Pure water clusters

To a surprising degree of qualitative accuracy, pure water clusters can be described using the simple classical TIP4P model potential. Wales and Hodges [28] have presented a large-scale global optimization benchmark study with this potential up to $n = 21$, employing the "basin-hopping" method. We have shown that a generalization of `PHENIX` to molecular clusters achieves at least comparable reliability and accuracy for this benchmark case [13]. Using our `GAGA` meta-strategy, we were also able to confirm global minimum energy structures on a suitable ab-initio level [31], for $n = 5, 6, 7$, and, at the same time, to qualitatively improve the performance of a physically well-founded model potential that previously worked well only for smaller clusters.

3.4 Water hetero clusters

Water solvation clusters of ions in the vacuum are currently investigated experimentally as models for a better understanding of bulk solvation. However, theoretical proposals for preferred structures of such entities have so far been largely speculative. We are currently applying our `PHENIX` global optimization strategy to alkali cations in TIP4P-modelled water clusters, in direct collaboration with experimentalists [39]. Preliminary results indicate that many traditional pictures may need revision. As an example, Fig. 2 shows our best structure for $Na^+(H_2O)_{20}$.

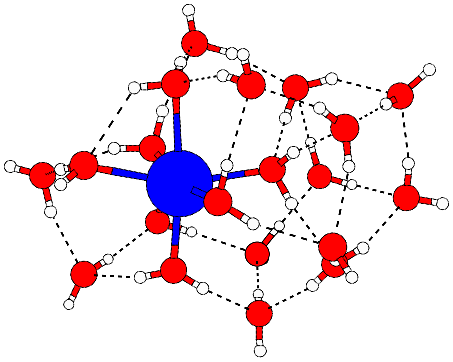


Figure 2: Globally optimal sodium ion – water microsolvation cluster

This structure violates two entrenched paradigms at the same time: Contrary to chemical intuition, at this cluster size the cation clearly is not at the center of the cluster. Also, the whole structure bears no resemblance whatsoever to the favorite hypothesis of a regular dodecahedral clathrate-like solvation hull traditionally expected to form at this cluster size. Interestingly, our first calculations in which we replaced Na^+ by its larger cousins K^+ and Cs^+ do show dodecahedral clathrate structures as global minima [40]. This seems to support the experimental observation that for these water solvation clusters n=20 is a magic number for K^+ and Cs^+ but not for Na^+. Further research into these systems tries to confirm these observations by ab initio calculations and to uncover reasons for these structural differences [40].

4 Conclusions

We have given a brief introduction to successful applications of GA-based methods to global cluster structure optimization. Comparing to other general-purpose global optimization algorithms, we have found our methods to be more easily adaptable to specific problems and at least as reliable and efficient. For the specific case of Lennard-Jones clusters, there is evidence that our method has a superior size scaling, enabling us to tackle larger problems.

Our global optimization methods are computationally much cheaper than more traditional methods in this area (e.g. molecular dynamics). Also, their output can be extended beyond mere listings of minimum structures towards thermodynamical information [41, 42]. Thus, GA-based global optimization can now be regarded as a new tool in chemistry and cluster physics, offering reliable information at a point where usual chemical intuition starts to fail badly.

References

[1] L. T. Wille, in: “Annual Reviews of Computational Physics VII”, D. Stauffer (Ed.), World Scientific, Singapore (2000); p. 25.

[2] J. C. Culberson, Evol. Comp. J. **6** (1998) 109.

[3] F. Jensen: “Introduction to Computational Chemistry”, Wiley, Chichester, 1999.

[4] A. Neumaier, SIAM Rev. **39** (1997) 407.

[5] C. M. Dobson, A. Šali and M. Karplus, Angew. Chem. **110** (1998) 908.

[6] “Clusters of Atoms and Molecules”, H. Haberland (Ed.), Springer, Berlin (1994).

[7] K. Liu, M. G. Brown, C. Carter, R. J. Saykally, J. K. Gregory and D. C. Clary, Nature (London) **381** (1996) 501.

[8] K. Nauta and R. E. Miller, Science **287** (2000) 293.

[9] L. T. Wille and J. Vennik, J. Phys. A **18** (1985) L419.

[10] G. W. Greenwood, Z. Phys. Chem. **211** (1999) 105.

[11] C. D. Maranas and C. A. Floudas, J. Chem. Phys. **97** (1992) 7667.

[12] J. Pillardy and L. Piela, J. Phys. Chem. **99** (1995) 11805.

[13] B. Hartke, Z. Phys. Chem. **214** (2000) 1251.

[14] D. Hohl, R. O. Jones, R. Car and M. Parrinello, J. Chem. Phys. **89** (1988) 6823.

[15] U. Röthlisberger and W. Andreoni, J. Chem. Phys. **94** (1991) 8129.

[16] B. Hartke and E. A. Carter, J. Chem. Phys. **97** (1992) 6569.

[17] B. Hartke and E. A. Carter, Chem. Phys. Lett. **216** (1993) 324.

[18] R. S. Judson, Rev. Comput. Chem. **10** (1997) 1.

[19] R. S. Judson, M. E. Colvin, J. C. Meza, A. Huffer and D. Gutierrez, Int. J. Quant. Chem. **44** (1992) 277.

[20] B. Hartke, J. Phys. Chem. **97** (1993) 9973.

[21] Y. Xiao and D. E. Williams, Chem. Phys. Lett. **215** (1993) 17.

[22] S. K. Gregurick, M. H. Alexander and B. Hartke, J. Chem. Phys. **104** (1996) 2684.

[23] D. M. Deaven and K. M. Ho, Phys. Rev. Lett. **75** (1995) 288.

[24] D. M. Deaven, N. Tit, J. R. Morris and K. M. Ho, Chem. Phys. Lett. **256** (1996) 195.

[25] M. D. Wolf and U. Landman, J. Phys. Chem. A **102** (1998) 6129.

[26] B. Hartke, J. Comput. Chem. **20** (1999) 1752.

[27] D. J. Wales and J. P. K. Doye, J. Phys. Chem. A **101** (1997) 5111.

[28] D. J. Wales and M. P. Hodges, Chem. Phys. Lett. **286** (1998) 65.

[29] B. Hartke, Chem. Phys. Lett. **258** (1996) 144.

[30] B. Hartke, Theor. Chem. Acc. **99** (1998) 241.

[31] B. Hartke, M. Schütz and H.-J. Werner, Chem. Phys. **239** (1998) 561.

[32] The Cambridge Cluster Database, D. J. Wales, J. P. K. Doye, A. Dullweber and F. Y. Naumkin, `http://brian.ch.cam.ac.uk/CCD.html`

[33] J. P. K. Doye, M. A. Miller and D. J. Wales, J. Chem. Phys. **111** (1999) 8417.

[34] D. Romero, C. Barrón and S. Gómez, Comput. Phys. Comm. **123** (1999) 87; `http://www.vetl.uh.edu/~cbarron/LJ_cluster/LJpottable.html`

[35] C. Barrón, S. Gómez and D. Romero, Appl. Math. Lett. **10** (1997) 25.

[36] R. H. Leary and J. P. K. Doye, Phys. Rev. E **60** (1999) R6320.

[37] Y. Wang, H.-J. Flad and M. Dolg, Phys. Rev. B **61** (2000) 2362.

[38] B. Hartke, H.-J. Flad and M. Dolg, manuscript in preparation.

[39] A. Charvat, B. Abel, M. Reich, T. Bergmann and B. Hartke, Phys. Chem. Chem. Phys., submitted (March 2001).

[40] F. Schulz, diploma thesis and Ph.D. thesis, University of Stuttgart, in preparation.

[41] D. J. Wales, Mol. Phys. **78** (1993) 151

[42] J. P. K. Doye and D. J. Wales, J. Chem. Phys. **102** (1995) 9659.

Finding near optimal parameters for Linear Congruential Pseudorandom Number Generators by means of Evolutionary Computation

Hernndez J.C., Ribagorda, A., Isasi P., Sierra. J.M.
Computer Science Dept.
Carlos III University
28911, Leganés, Madrid, Spain
{jcesar,arturo,isasi,sierra}@ia.uc3m.es

Abstract

Linear Congruential Generators (LCG's) are one model of pseudorandom number generators used in a great number of applications. They strongly depend on, and are completely characterized by, some critical parameters. The selection of good parameters to define a LCG is a difficult task mainly done, nowadays, by consulting tabulated values or by trial and error.

In this work, the authors present a method based on genetic algorithms that can automatically solve the problem of finding good parameters for a LCG. They also show that the selection of an evaluation funtion for the generated solutions is critical to the problem and how a seemingly good function such as entropy could lead to poor results. Finally, other fitness function are proposed and one of them is shown to produce very good results. Some other possibilities and variations that may produce fine linear congruential generators are also mentioned.

1 introduction

Since they were first proposed by Lehmer, in 1948 [1], Linear Congruential Generators (LCGs) are the most widely used Pseudorandom Number Generators (PRNGs). They are, also, one of the best analyzed models. Used in a great number of applications such as simulation, numerical analysis and optimization, they are frequently present where some PRNG is need.

A typical Lineal Congruential Generator has the form:

$$X_{n+1} \ = \ (a * X_n + b) \, mod \, m \ with \ X_0 = seed$$

It is completely characterized by the parameters $(X_0,\ a,\ b,\ m)$. Actually, the value of X_0 is nearly irrelevant, as we will see when analysing the MaxPeriodTheorem.
Although their cryptographic limitations [2, 3, 4, 5] are well known, for example they are not recommended as key generators for stream ciphers, and behave poorly in some statistical tests, they remain in wide use, mainly because of their ease of implementation and very high speed. The best ones are, also, hard to find because a badly chosen parameter usually makes the associated LCG worthless and, depending on the applications that will use it, even dangerous. In [9] one finds "Despite the large amount of theoretical research already done on this subject, many of the generators currently in use [...] are seriously flawed. Even some recently proposed or evaluated generators have a very weak theoretical justification." So selecting good parameters to define a LCG is quite a difficult task [6], and nowadays, if one wants to implement a LCG and knows something about the great importance of choosing good parameters, the only option available is to search throught tabulated triples $(a,\ b,\ m)$ until finding one that suits one's needs. These tabulated triplets are chosen because of their good statistical properties when exposing the generated sequence of numbers to a battery of tests, but the tests vary too much in number and significance.

Lots of programmers found themselves tempted to choose these parameters at random, which is a very bad idea that implies that dozens of very poor LGC's around the world are doing their job badly. It seems that a new way of automatically finding good LCG's will be very useful.

The main idea of this work is to design a procedure for automatically finding good parameters $(a,\ b,\ m)$ for a LCG by using genetic algorithms, completely avoiding the use of tables. We also illustrate the feasibility of using genetic algorithms as a useful tool to find good parameters for certain algorithms, in this case a particular model of a pseudorandom number generator that is widely spread throught thousands of applications around the world. We offer a new way to find different, but excellent and new LCGs. Also, more experienced users who want to use other LGCs that suits better their objectives than the tabulated one's can take advantage of this method.

2 LCGs Evaluation

Genetic algorithm-based methods are a natural approach when one must find the best (or at least a very good) solution to a problem between a really wide number of possibilities.

In the design of LCGs the solutions consists on the asignation of particular values to the three parameters a, b and m. Therefore one can also think of genetic algorithms as a way of doing a guided search through the space of tridimensional vectors (a, b, m) in natural numbers. With this scheme an individual consists on a binary representation of three natural numbers corresponding with the parameters to be found.

It is well known that the main problem when working with a genetic algorithm is usually the determination of the *fitness function*. The fitness function must contain as much problem domain knowledge as possible, because it has the most important bias for the searching process.

In this case, we have moreover an additional problem, because good or bad LCG's are sometimes difficult to distinguish and, frequently, this classification depends highly on the application that this LCG is going to have. Good LCG's for a certain application can be considered bad for others and viceversa. So, the ideal fitness function is one capable of perfectly mesuring the randomness of a LCG in question.

Unfortunately this function, one universal randomness-mesuring function, simply does not exist. One cannot measure randomness directly, in a single variable. Randomness is a hard concept which is, also, highly multidimensional. Therefore we must try different aproximations.

For a given pseudorandom number generator, be it a LCG or other, the usual rule of thumb is to make it pass a given battery of tests and see the results. These tests rejects most of the generators as non-random, but offer no guarantees on the generators that pass the tests. One generator (or in our case, a triplet of parameters a, b and m) that passes one hundred tests and performs as being random can behave very poorly in the next test and be rejected with strong evidence as being non-random. This battery testing method is the only way of accepting or rejecting generators, but it is clearly not good.

3 Generation of Linear Congruential Generators

3.1 Statistical test

In order to have a measure of the goodness of the acieved LCGs the following *statistical tests* have been used:

1. Entropy: The information density of the output of one cycle of the generator, expressed as the number of bits by byte (pack of 8 bits) generated, so it can take any value between 0.0 and 8.0, being higher values better but exponentially harder to reach by chance. It is one widely used way to mesure one aspect of randomness. Mathematically, it is expressed by the formula:

   $\sum_{x \in GF(2^8)} \; -p(x) log_2 p(x)$

2. Chi-Square percentil: The chi-square test is a very commonly used test for the randomness of data because it is extremely sensitive to errors in pseudorandom number generators. Its statistic compares the observed probabilities of every byte in a cycle of the generator against the expected ones if it were uniformingly distributed. The test evaluates the probability that the observed value of the statistic will be obtained from a uniform distribution. Values higher than 99% or less than 1% indicate that the sequence is almost certainly not random, and values between 99% and 95% or between 5% and 1% suggest that the sequence is suspect. The ideal value is 50% but values obtained in the 25%-75% interval can be considered good enough to pass the test. No LCG pass this test.

3. Arithmetic mean: Simply the arithmetic mean of all the values in a cycle of a generator. As every value between 0 and 255 must be equiprobable, results close to 255/2=127.5 must be obtained if the generator had good properties.

4. Monte Carlo value for Pi: This tests simply takes each sucessive sequence of six bytes and transform

it into two 24 bits coordinates (X,Y) that generate a point within a square. If the distance of this pseudorandomly generated point is less than the radius of the circle inscribed whitin the square, this point is considered a hit. Repeating this process a large number of times, the percentage of hits can be used to calculate the value of Pi and if the sequence is close to random this calculation will slowly converge to the real value of Pi.

5. Serial correlation coefficient: This quantity measures the extend to which each byte in the file depends upon the previous byte. For random sequences, this value (being it positive or negative) must be close to zero.

6. Maximal Period: A LCG determined by parameters a, b and m has maximal period if and only if the lenght of its period, that is, the lenght of the sequence that it produces before repeating, is m. To do this test we examine the sequence, looking for subcycles or degenerate periods or some mathematical properties, to determine its period. If we found a maximum period we labeled this column with a 'Y', otherwise a 'N'.

7. Percentage of the maximum period achieved: It is a little redundant with test 6 but useful if the generator does not have maximal period.

3.2 Entropy as fitness function

We have slightly modified Shannon's classical definition of entropy to have, in our case, the information density of the output of one cycle of a given generator, expressed mathematically by the formula

$$\sum_{x\in GF(2)} -p(x)log_2p(x)$$

where $p(x)$ is the observed probability of x in one cycle of the LCG, and $GF(2^n) = \{x|x \in \{0,1\}^n\}$.

We thought that entropy would be a very good fitness function. The measure of the entropy of a LCG is intuitively strongly related with its randomness. A LCG can be seen as a source of randomness and, obviously, higher entropy values are better. One thing worth mentioning is how entropy is measured. If we measure per bit entropy (as in the formula above) our maximum value will be 1.0 and will be easily reached by all sequences with the same number of 0's and 1's. Intuitively, this seems not to be a good choice as something like 10101010101010101010.....10 will have maximum entropy, being obviously quite non-random. So it is important to mesure entropy at higher, more difficult, levels. We found that per byte entropy (measuring the entropy of consecutive blocks of 8 bits) was quite adequate, since reaching the maximum of 8.0 was exponentially more difficult and much more informative. So our new formula is

$$\sum_{x\in GF(2^8)} -p(x)log_2p(x)$$

We also limited the range of integers to study because we defined our chromosome lengh to be 45 bits long. These 45 bits where logically divided into three groups to include the binary representation of a, b and m (in this order) so this bounded a, b and m to be less or equal than 2^{15}-1. This has no effect on our results except for the fact that better LCG's are achievable when larger integers can be used: larger periods can be obtained if we can choose larger m's and better or equal multipliers can be found if we have a higher bound. So we must compare our results against LCG's of our size. Anyway, our results are easily generalyzed to larger values just by extending the chromosome lengh.

Using this entropy-based fitness function we started to run our genetic algorithm implementation with what we considered the best internal parameters, found after some trials that pointed out that a crossover probability of 1.0, a low mutation probability (around 0.05), and a population size of 50 individuals were a good choice. We also used some elitism (the two best individuals always passed to the next generation) and tournament selection.

In Table 1 below we see the results of the statistical tests on a series of recommended LCG's that we got from tables in [7] and in Table 2 we see the results of this same test on the LCG's we obtained with this entropy only based approach.

At first sight the results on Tables 3,4 are much better than the results on Tables 1,2: all Chi-square percentils in Table 3 are equal or better than those in Table 1, the worst values for the entropy, arithmetic mean, and Monte Carlo pi estimation on Table 3 are much better than the best in Table 1, etc... So one is tempted to conclude research at this point pretending to have much better LGC's than the tabulated (and recommended) ones.

This is not true. These generators look quite impressive, but they are very bad indeed. In spite of its well-known use in coding theory and cryptography, and against first intuition, *entropy is not a definite randomness measuring function*. It is obvious that, in our case, generators that produce very low entropy values must be rejected as non-random (because they produce non uniform output), but, as it happens with

a	b	m	1	2	3	4
106	1283	6075	7.955	0.01	133.8	4.29%
1366	1283	6075	7.957	0.01	133.1	2.56%
936	1399	6655	7.961	0.01	133.5	3.43%
211	1663	7875	7.954	0.01	136.5	6.94%
421	1663	7875	7.954	0.01	136.0	7.20%
430	2531	11979	7.972	0.01	133.5	3.42%
859	2531	11979	7.973	0.01	133.6	4.14%
1741	2731	12960	7.977	0.01	132.2	2.08%
1541	2957	14000	7.973	0.01	133.6	3.33%
967	3041	14406	7.971	0.01	134.2	4.24%
1291	4621	21870	7.983	0.01	131.8	2.48%
419	6173	29282	7.979	0.01	133.6	2.81%
1255	6173	29282	7.979	0.01	133.9	4.93%
625	6571	31104	7.975	0.01	134.4	4.55%

Table 1: Test 1 to 4 for LCGs recomended in the literature

any other test, high values of entropy do not prove randomness at all. In fact, *entropy measures are nothing more than another randomness test, not better nor worst than any other.*

In our case, not only entropy values are significantly better on Table 2, as one can expect when using genetic algorithms to maximize entropy, also most of the rest. The worst value on Table 2 for tests 1, 2, 3 and 4 is much better (or at least equal) that the best one in Table 1. *The only significant exception is test 5 (serial correlation coefficient), but apparently, correlation values in Table 2 are quite good too (very near to zero).* It seems as if there is strong evidence in favor of Table 2 generators, but in fact they are very poor. They do not have maximal periods (as shown on Table 2 column 6 and 7). The values of the serial correlation coefficient reflect this undesirable behaviour.

Not having maximal period is not really so bad, if the period is quite close to this maximum. But this is not the case. A look at Table 2 column 7 shows these generators are not good because none of them achieve even a 13% of its theoretical maximum period. We must reject all of them as not useful for any uses (except, perhaps, exemplification of the limited value of entropy measures and, by extension, randomness tests).

How can we know *a priori* if a given LCG will have maximum period? The characterization of a maximal period LCG is done by the next

MaxPeriodTheorem: (Taken from [8])

a	b	m	5
106	1283	6075	-0.0127
1366	1283	6075	0.0026
936	1399	6655	-0.0036
211	1663	7875	-0.0059
421	1663	7875	-0.0199
430	2531	11979	0.0010
859	2531	11979	0.0061
1741	2731	12960	-0.0081
1541	2957	14000	-0.0154
967	3041	14406	-0.0078
1291	4621	21870	-0.0043
419	6173	29282	-0.0052
1255	6173	29282	-0.0106
625	6571	31104	-0.0133

Table 2: Test 5 for LCGs recomended in the literature

a	b	m	1	2	3	4
7476	3206	21497	7.991	0.01	129.5	1.79%
6943	5593	22873	7.990	0.01	128.0	1.06%
15131	6914	22873	7.993	0.01	128.6	1.05%
7476	2019	21331	7.992	0.01	129.7	0.96%
26977	10516	20817	7.993	2.50	129.3	0.46%
23989	1911	21331	7.993	0.50	129.1	0.91%
14684	29655	29241	7.993	0.01	127.5	0.87%

Table 3: Test 1 to 4 for LCGs generated by the GA with entropy fitness function

The linear congruential sequence:

$$X_{n+1} = (a * X_n + b) \, mod \, m \text{ with } X_0 = seed$$

defined by m, a, b and X_0 has maximal period (i.e. period length m) if and only if:

i) b is relatively prime to m

ii) $a-1$ is a multiple of p, for every prime p dividing m

iii) $a-1$ is a multiple of 4 if m is a multiple of 4

3.3 New fitness functions

Using an entropy-only based fitness function we do not get maximal period generators, contrary to our first beliefs that entropy is a definite randomness measure. So if we want maximal period generators we must ask

a	b	m	5	6	7
7476	3206	21497	-0.0163	N	6.80%
6943	5593	22873	0.0664	N	3.11%
15131	6914	22873	0.0119	N	6.23%
7476	2019	21331	-0.0210	N	12.30%
26977	10516	20817	-0.0227	N	6.23%
23989	1911	21331	0.0014	N	12.30%
14684	29655	29241	-0.0512	N	3.51%

Table 4: Test 5 to 7 for LCGs generated by the GA with entropy fitness function

for it in the fitness function. We must obvioulsy change our fitness function to include something related to the period of the LCG. One of our first attempts was the use of

$$fitness = entropy + period$$

but this function was quickly rejected because our search degenerated soon to nearly maximal $m's$ (in our implementation, maximum value for m is 2^{15}-1 = 32767). This was due to the very low significance that $0.0 \leq entropy \leq 8.0$ had in this fitness function where $0 \leq period \leq 32767$ become much more important (32767 parts of 32775, a weighted value of about 99.98%).

A better idea is to use a normalized measure of the period. Then, our fitness could be

$$fitness = entropy + \frac{period}{maxperiod} = entropy + \frac{period}{m}$$

(as m is the maximum period achievable by our LCG) and will have a maximum of 9.0 We tried this fitness function and found that, although it produced interesting results, it suffered from nearly the same problems as our first only entropy-dependent function, as it gave much more importance to entropy (8 parts of 9) than to period (1 parts of 9) and we felt both measures were equally important.

The next and final fitness funtion we used was, then,

$$fitness = entropy * \frac{period}{maxperiod} = entropy * \frac{period}{m}$$

This balances nicely both measures. Now our new maximum was 8.0 (but it was more difficult to obtain at random because $0 \leq \frac{period}{maxperiod} \leq 1$)

When using this final fitness function cited above, we got lots of interesting generators, some of which are shown in Table 5

a	b	m	1	2	3	4
15005	8371	19993	7.983	0.01	132.7	4.26%
6237	10697	21023	7.985	0.01	132.9	5.12%
14359	9654	21569	7.985	0.01	132.3	3.49%
12586	11658	21023	7.985	0.01	132.7	5.25%
4518	15578	21179	7.985	0.01	132.0	2.30%
1271	10331	20983	7.985	0.01	132.3	3.69%
6533	4712	21011	7.985	0.01	132.9	4.32%
14945	6262	21089	7.985	0.01	132.5	2.10%
1	19568	19647	7.984	0.01	132.8	3.47%

Table 5: Test 1 to 4 for LCGs generated by the GA with the new designed fitness function

a	b	m	5	6	7
15005	8371	19993	-0.0007	Y	100.00%
6237	10697	21023	-0.0017	Y	100.00%
14359	9654	21569	-0.0024	Y	100.00%
12586	11658	21023	0.0025	Y	100.00%
4518	15578	21179	-0.0005	Y	100.00%
1271	10331	20983	-0.0022	Y	100.00%
6533	4712	21011	-0.0103	Y	100.00%
14945	6262	21089	-0.0089	Y	100.00%
1	19568	19647	0.0052	Y	100.00%

Table 6: Test 5 to 7 for LCGs generated by the GA with the new designed fitness function

The results of the statistical tests performs on these generators are better but much less spectacular than those of Tables 3,4. If we compare our generators of Tables 5,6 against those shown (and recommended in the literature as good generators) on Tables 1,2 we see they perform very close, our's slightly better, so it is very likely that they will be adequate and perform equally good in most applications. They even pass the difficult serial correlation test the generators of Tables 3,4 failed. Our aim was to provide an automatic method of generating LCG's as good as the tabulated ones and the results on Tables 5,6 prove that, at least from these tests point of view, we have managed to do it.

We believe it is worthy to mention two curious facts: All the $m's$, except the last, obtained by our entropy+period approach and shown on Tables 5,6 have one interesting property in common: they are all prime numbers. That is a very direct and clever way of assuring that all the requisites in MaxPeriodTheorem are satisfied. Our genetic algorithm has found an intelligent and direct approach to meet all these requisites. Genetic algorithms have shown extremely high power in finding extraordinary bad LGCs with incredible good entropy, that is, finding exceptions. They also usually prefer the shortest, quickest solutions. In our case, during crossover, due to the representation chosen, around two in three times the m of one of the parents is passed to the child. Being m so important as to assure maximal period (if m prime and $b > 0$), good $m's$ are quickly spread through the population as a very good characteristic. When two individuals with the same (prime) m have crossover applied, the same m will be inherited by the child, thus guaranteeing a good average fitness, and only a or b are changed, thus searching for optimal $a's$ and $b's$ for a given m.

Obviously, m is not forced to be prime by the genetic algorithm. It can take any other value, producing different generators, but prime $m's$ are more likely. This is simply a shortcut that our genetic algorithm has found to easily achieve MaxPeriodTheorem requisites.

In our last example we have a non-prime m, but in this case maximum period is achieved by selecting $a = 1$, thus assuring all those requisites on $a - 1$ (in this case 0, so multiple of any number) of the MaxPeriodTheorem in a very nice and direct way.

4 Further Research

Some changes could be made that may drastically improve the results obtained, without changing the main ideas behind this research. We will mention here some of these changes that will probably lead to further and interesting research:

-Increasing the size of the chromosome will obviously produce better results. This can be acomplished easily, and results on longer chromosome implementations could probably compete against the best LCG's known nowadays.

-We believe the classical one point crossover mechanism that we have implemented can be improved somehow, perhaps using a biased-towards-the-better aproximation that also takes adventage of the very different relevance of the three parameters we have to study ($m >> a >> b$).

-Another interesting possibility is to change the fitness function such that it reflects that we prefer greater values of m, say adding a term like $\frac{1}{15} * \log(m)$.

-We believe that, in the scope of these kind of generators, a fitness function related with the result of the Spectral Test in a number of dimensions may do even better. For example, if using a normalized spectral test for different dimensions, an interesting fitness function to maximize could be the minimum on the test for these 30 dimensions.

5 Conclusions

We believe we have provided strong evidence that passing a series of tests, call them poker test, chi-square or even entropy, don't assure the passing of futher tests and that excellent results on some tests do not necessarily mean anything about the randomness of a generator. In fact, extraordinarily good results in one test must put the serious researcher in state of alarm.

We also have shown that a method relatively new in this field, the use of genetic algorithms, can be of great help when looking for good parameters. After the work of Pierre L'Ecuyer [8] in combining LCG's to improve their properties, this automatic way of getting good LCG's drastically increases its interest, as it could be used to generate automatically a pool of quite good LCG's that then can be combined to make an even better generator.

The authors strongly believe that variants of this method can be easily developed to help in the parameter choosing of other types of generators like Lagged Fibonacci Generators, IGC's, EIGC's, etc...

<u>References</u>

[1] D.H. Lehmer. Mathematical methods in large-scale computing units. Proc 2nd Sympos on Large-Scale Digital Calculating Machinery Cambridge MA 1949. pages 141-146, Cambridge MA 1951. Harvard University Press

[2] J.B. Plumstead. Inferring a Sequence Generated by a Linear Congruence Proceedings of the 23rd IEEE Symposium on the Foundations of Computer Science 1982

[3] D.E. Knuth. Deciphering a Linear Congruential Encryption. Technical Report 024800. Standford University. 1980

[4] J. Reeds. Cracking a Random Number Generator Cryptologia Vol. 1. January 1977. [5] J. Boyar Inferring sequences produced by pseudorandom number generators. J. Assoc. Comput. Mach. 36 N1, pp. 129-141, 1989

[6]Park & Miller . Random Number Generators: Good Ones Are Hard to Find Communications of the ACM, October 1988, p. 1192.

[7] B. Scheirer Applied Cryptography. 2nd Edition, p.

348. John Wiley & Sons 1996

[8] D.E. Knuth. The Art of Computer Programming. Second Edition, p.16 Addison-Wesley Series in C.S. and I.P, 1981

[9] Pierre L'Ecuyer. Efficient and Portable Combined Random Number Generators. Communications of the ACM Vol 31 pp. 742-749, 1988

Genetic Algorithms as Algorithm Adversaries

Elizabeth L. Johnson
Dept. of Math & Computer Science
Xavier University
Cincinnati, OH 45207
`ejohnson@xu.edu`

Abstract

This paper describes a genetic algorithm-based test case generator for use in the empirical analysis of algorithms. The fitness function for the genetic algorithm is the number of operations executed when the algorithm is run with the test case as input – the worse the performance, the higher the fitness. The goal of the generator is to produce test cases that are pathological for the algorithm. We present results from experiments using a GA-based generator for maximum cardinality bipartite matching and show that the results are better than the results using randomly generated test cases.

1 INTRODUCTION

Empirical analysis of algorithms is of increasing interest to the algorithms community. This type of research focuses on the performance of algorithms in practice, as analyzed through computational experiments. Such analysis leads not only to a better understanding of how well existing algorithms perform, but can generate new practical as well as theoretical improvements to the algorithms (Johnson and McGeouch 1993; Cherkassky, Goldberg, and Radzik 1993; McGeouch 1986). Just as techniques such as average case and worse case analysis were developed for studying the asymptotic behavior of algorithms, there is a need to develop robust techniques to study and characterize the behavior of algorithms in practice (Hook 1994).

One important area of study is the development of methods to generate cases for testing implementations of algorithms (Goldberg 1998). Traditionally, this input has been of three forms: 1) random input based on a general probability model, 2) real-world input, or 3) random input of a given structure (such as grid graphs or graphs with a certain degree sequence). While each of these input types is important to understanding the behavior of an algorithm, they do not yield a very complete picture of behavior because they focus on either average case or performance on input with a well-defined structure. Performance on infrequently occurring input or on input with different structures may not be fully explored by such methods.

Two questions of interest to algorithm designers are: 1) does the new algorithm do better than other algorithms on a given set of input cases, and 2) what are the input cases on which the new algorithm performs poorly? With the random generation, structured input, and real-world data approaches, the input is static. When a new algorithm version is being tested, we can only compare its performance to the old version on a given set of inputs, leaving open the question of which inputs cause the new algorithm to perform badly. We could try to generate more random input in an attempt to find pathological cases, but doing a random search of a large input space might take a long time without yielding any bad cases. If our interest is data of a certain structure or real-world data, we could examine the second question by trying input cases with various structures, but that would require making assumptions about what affects performance and designing a new input set for each new algorithm version.

Our work focuses on developing a more dynamic system for generating input in which the generator acts as an adversary to the algorithm, producing test cases on which the algorithm performs poorly. This generator uses a genetic algorithm (GA), a search technique based on the idea of *survival of the fittest* (Goldberg 1989). An initial population of objects representing test cases is generated, each object's fitness is evaluated, and depending on that fitness it may survive to reproduce and create the next generation. Objects with higher fitness have a greater chance of reproducing. In our work, a test case object's fitness is based on how the algorithm performs on that test case -- the worse the performance, the higher the fitness. So the final best individual after many generations of reproduction is the test case object on which the algorithm performed worst. We define performance by counting operations of interest performed by the algorithms, but performance could also be based on other measures such as resource usage (in terms of execution time or space) and quality of output (especially in the case of heuristics).

With the GA approach, we can readily compare algorithms and study the relationship between input and performance for a given algorithm. First, in order to compare algorithms, we can run the new algorithm on the final population generated by the GA for the old algorithm and vice versa. This allows us to test each algorithm on input cases that were hard for the other algorithm. Second, because the GA generator adapts its output to a particular algorithm, we can study performance by using the new algorithm for the GA fitness function and running the adversarial generator again. This allows us to answer the second question by providing a set of test cases on which the new algorithm performs poorly.

Obviously, the utility of this GA-based approach depends on the strength of the relationship between input structures and algorithm performance. For example, it would not be appropriate for algorithms that perform the same number of operations on all input. Fortunately, for most algorithms of interest, there is some relationship between input and performance and we can gain insight into algorithm behavior by studying the structure of the GA-generated input.

2 RELATED WORK

In previous empirical work on algorithms, test cases have been primarily of three types: 1) randomly generated data based on a particular probability model, 2) randomly generated data of a particular structure, and 3) data from real-world applications. Most of the experimental work in recent years has used the latter two types. In particular, if a pathological structure for a particular algorithm is known researchers will construct a generator which creates test cases with that structure (Moret and Shapiro 1994; Cherkassky, Goldberg, and Radzik 1993).

In 1991, the first DIMACS Implementation Challenge, which focused on empirical testing of network flow and matching algorithms, was held (Johnson and McGeoch 1993). Generators produced by participants and collected by DIMACS produced either random graphs or graphs which had particular structures, such as grid graphs and acyclic graphs. Several also produced known pathological cases for particular algorithms. In panel discussions following the formal presentations, participants lamented the lack of challenging test cases for the algorithms. In particular, they expressed concern that by using a limited set of test cases, algorithm implementations could be tuned to certain structures in the test cases. This would ensure good performance on the existing cases, but contribute little in understanding algorithm behavior on other non-represented structures.

Another problem with generators that produce structured cases is that they may introduce unintentional features into the input because of the construction procedure. These difficulties are described in (Sanchis 1994) in relation to a generator for vertex cover, an NP-hard problem. The generator was designed to produce test cases with known solutions by creating a minimal graph with a cover set of a given size and then adding edges to that graph. Because of the underlying properties of the original method used to generate graphs, degrees of noncover vertices were consistently lower than degrees of cover vertices. Since one of the heuristics being tested used a greedy rule that repeatedly chose vertices of highest degree for the cover, it did very well on these graphs. Correcting this problem necessitated modifying the generator so that it kept the average degrees of the cover and non-cover vertices nearly equal.

GAs have been used previously in generating test cases for real-time software systems (Wegener, et. al., 1997). In these systems, execution time is part of the correctness definition of the software. For example, in a rental car reservation system, calculations of available vehicles must be completed within a certain time in order for the system to be acceptable. Testing needs to characterize the time bounds of such calculations. Traditional testing methods would require the development of these best and worst cases. This is a difficult task due, in part, to the impact of the system environment on execution time. The GAs generated test cases that could be used to establish wider bounds on execution time than test cases produced by random generators. While the purpose of these experiments was to test the temporal correctness of the programs, they demonstrate the utility of GAs in generating extreme test cases for problems.

3 BIPARTITE MATCHING

The algorithms under study in our current work are variations of a push-relabel algorithm that solves the maximum cardinality bipartite matching problem. In this section we describe the basic problem as well as the algorithms of interest.

3.1 PROBLEM DESCRIPTION

The maximum cardinality bipartite matching problem is defined as follows: given an undirected bipartite graph with vertex set N, divided into two sets N_1 and N_2 , find the maximum cardinality set M' of edges (u,v) such that $u \in N_1$ and $v \in N_2$ and for all other edges $(u',v') \in M'$, $u \neq u'$ and $v \neq v'$. So no vertex is matched with more than one other vertex. A matching which includes or *covers* all vertices in N_1 and N_2 is called a *perfect matching*. The maximum size matching possible for a given graph is $min(|N_1|,|N_2|)$. We restrict our graphs so that $|N_1|=|N_2|$. Applications of this problem include job scheduling, pattern matching in images and strings, and resource allocation.

3.2 ALGORITHM DESCRIPTION

The bipartite matching problem can be transformed into a special case of the maximum flow problem by adding a source vertex, s, and edges from s to each vertex in N_1 and adding a sink vertex t and edges from each vertex in N_2 to t. In the maximum flow problem, the goal is to transport as much flow as possible from the source to the sink, given the constraint that edges have limited capacity. In the bipartite matching application, each edge is given unit capacity. The Goldberg-Tarjan push-relabel algorithm (Goldberg and Tarjan 1986) computes flow on the edges that meet the following constraints:

- *capacity constraint*: the flow on each edge must be less than the capacity of the edge,
- *flow antisymmetry constraint*: the flow on an edge is equal to the negation of the flow on its reversed edge, and
- *flow conservation constraint:* the flow entering a vertex is equal to the flow leaving a vertex (does not apply to source or sink).

The first and last constraints ensure that the flow on edges between vertices in N_1 and vertices in N_2 represents the maximum matching of the graph because each edge can only have one unit of flow. Since only one edge goes into each vertex in N_1 (from the source), flow can occur at most on one edge going out. The same is true of the edges going into the vertices in N_2.

The Goldberg-Tarjan algorithm works by initially pushing one unit of flow along each edge from the source to the N_1 vertices. The flow conservation constraint is relaxed to allow this N_1-unit *preflow* to exist as excess at the vertices in the network. Vertices are given a distance label, which approximates their distance from the source or sink. Initially the vertex label is set to 0 for all vertices except the source. The label of s is set to N_1+N_2+2. An edge (u,v) in the residual network (i.e. edges which have no flow on them) is *eligible* for a push operation if the distance label of $u(h(u))$ is equal to $1+h(v)$. By adjusting distance labels for each vertex through *relabel* operations and pushing flow along eligible edges using *push* operations, the algorithm sends as much flow as possible from the source to the sink. When all flow possible has gone to the sink, excess flow in the network is pushed back to the source in order to convert the preflow into a legal flow. The edges (between N_1 vertices and N_2 vertices) that have flow in the final network form the maximum cardinality matching set M'.

At any point in the algorithm, several edges may be eligible for push operations. One set of algorithm variations focuses on the decision about which vertex to process next when pushing flow through the network. The variation we explored uses *minimum distance vertex selection* (Goldberg and Kennedy 1994). In this algorithm, the vertex selected for processing is the vertex with the minimum distance label. In addition, a periodic global updating of the distance labels is done whenever the distance label of the vertex chosen is higher than any previous selected vertex. This global relabel computes an exact distance from each vertex to the sink in the residual network using a backwards breadth-first search and sets the vertex label to that value. A second backwards breadth-first search is done from the source to those vertices which had no path to the sink in the residual network. The labels of these vertices are set to $|N|$ + the distance to the source. Processing continues after each global relabel as before until the maximum flow has been computed. The Goldberg-Tarjan algorithm with minimum distance label vertex selection and global relabeling has an O(m $sqrt(n)$) bound, where n is the number of vertices and m is the number of edges.

We have also included a preprocessing step suggested in (Setubal 92). This has no known effect on the worst-case theoretical bound, but the preprocessing does appear to help in practice. Before any push/relabel operations begin, an initial greedy matching is done. In this matching, each N_1 vertex is paired with the first unmatched N_2 vertex in its edge list. If no unmatched node is found, a match is made with the first node in the edge list, even though it is also matched with one or more other vertices. Each vertex matching is counted as a push operation and labels are updated so that N_1 vertices have label $h(u)=1$ and the edges in the matching have one unit of flow. After this, the Goldberg-Tarjan algorithm is used to finish computing the flow (and therefore the legal matching) on the revised network.

4 GENETIC ALGORITHM TEST CASE GENERATOR

In implementing a genetic algorithm, we use SGA (Simple Genetic Algorithm), a set of functions written in C which implement the GA described in Goldberg's classic book (Goldberg 1989; Smith, Goldberg, and Earickson 1994). This code has been modified extensively to include crossover and mutation variations specific to this problem. User input to the basic SGA program includes: population size, chromosome length, number of generations, crossover probability, mutation probability, and random number generator state file. We also incorporated the MRANDOM pseudorandom number generator code to facilitate reproduction of results (Thomborson 1993). This program contains a pseudorandom number generator based on `prand`, developed by Bentley and Knuth. State files can be saved between runs so the problems with initial random number sequences can be avoided. This also allows us to repeat past runs by starting the random number generator with an old state file.

The sequence of operations in the genetic algorithm itself is the standard one(Goldberg 1989):

1. Generate initial population of chromosomes.

2. Evaluate fitness of each chromosome in the initial population.

3. Using selection, choose members of the current population as parents and combine them using a crossover operator to produce members of the new population.

4. Mutate individual chromosomes in the new population.

5. Evaluate the fitness of the chromosomes in the new population.

6. If this is final generation, stop and output the chromosome with best fitness. Otherwise, go to step 3.

Generation of the initial population is discussed below as are the crossover and mutation operators. Fitness for an individual is calculated using the number of pushes in the push/relabel bipartite matching algorithm. This measure was chosen because pushes are the bottleneck operations in this algorithm. We use a modified elitist selection where the two best individuals are automatically copied to the new population without undergoing change through crossover and mutation. Roulette wheel selection is used to choose the parents for the rest of the new population.

4.1 REPRESENTATION

As described above, a bipartite graph is a collection of nodes (*N*) and edges (*M*) in which the nodes can be divided in two groups N_1 and N_2 such that for any edge$(u,v) \in M$, $u \in N_1$ and $v \in N_2$. In our experiments, we represent a graph as a bitstring of length $|N_1|$x$|N_2|$, consisting of 0's and 1's. We can think of this as a flattened adjacency matrix where a 1 in a position indicates the presence of an edge and a 0 indicates the absence. Beginning in position 0 (the leftmost position), the first $|N_2|$ positions represent the edges from the first node in N_1 to the nodes in N_2. Figure 1 shows a bipartite graph with its corresponding bitstring. The substring "00010" for vertex *0* in N_1 means that there is an edge from that vertex to only vertex 3 in N_2 while the substring "10001“ for vertex 2 in N_1 represents edges from that vertex to vertices 0 and 4 in N_2. Note that this graph has a perfect matching which includes the edges (0,3), (1,0), (2,4), (3,1), and (4.2).

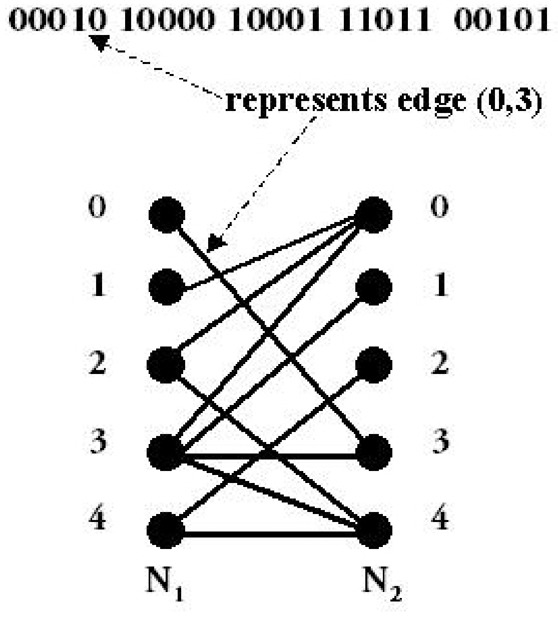


Figure 1: A bipartite graph and its chromosome

5 EXPERIMENTS

In our experiments we used two different genetic algorithm generators. The first generates graphs based on a fixed number of nodes *N* and an edge probability *pedge*. The second generates graphs with a fixed *N* and fixed number of edges *M*. Both use the representation for bipartite graphs described. They differ in their initial generation, crossover, and mutation methods.

5.1 VARIABLE EDGE COUNT GRAPHS

For the variable edge count GA, the initial bitstrings are generated with each position set to 1 with probability *pedge*. Otherwise the position is set to 0.

The crossover method is called *partial-v crossover*. The idea behind this method is to preserve some of the edge structure of the parents when creating the children. First, the randomly generated crossover point *p* can only occur at N_1 vertex boundaries in the bitstring. Second, we use *p* in determining where to begin crossover exchange on the N_2 vertices as well as the N_1 vertices. This means that the edge structure of the vertices above the crossover point *p* is maintained in the respective children while the edge structure for the vertices below the point is the result of combining the two parents. This method can result in a change in the number of edges in each graph, but the change is acceptable, given the probability model. Figure 2 illustrates the crossover method. In the children, the edges contributed by parent 1 are shown as solid lines while edges contributed by parent 2 are shown as dotted lines. The crossover point *p* appears as a dashed line.

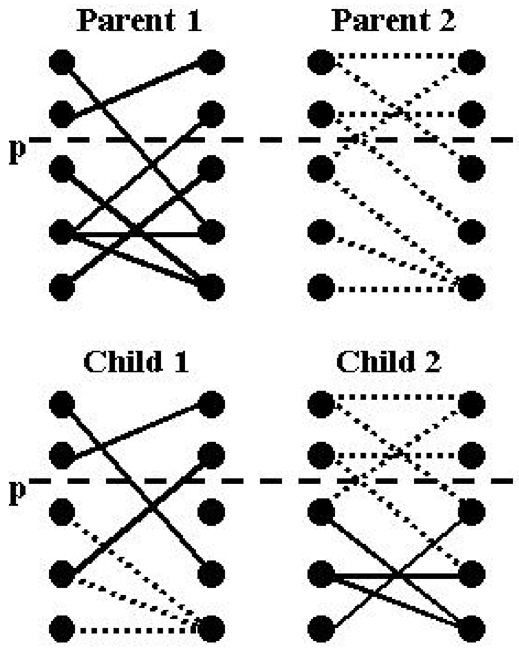


Figure 2: Partial V-Crossover

The mutation method is designed to preserve the qualities of the probability model as much as possible. In particular, the edge probability *pedge* is used to determine what the value of the mutated bit position should be. The current value of the bit position is ignored and a random number *r* between 0 and 1 is generated. If *r* <= *pedge* then the bit position is set to 1, otherwise the bit position is set to 0.

5.2 FIXED EDGE COUNT GRAPHS

Our second GA maintains a fixed number of edges in each graph representation. For generation of the initial population, the following technique is used.

1. Set all positions in the chromosome to 0 initially.

2. For each of *M* edges, generate a random number from 0 to *lchrom*-1.

3. If that position in chromosome is currently 0, change it to 1; otherwise repeat the number generation until it indicates a position that is 0. The repetition is necessary because we may have duplicates in the random number sequence so we need to choose a different edge to add.

The crossover method is as follows, with processing beginning at the first position in the chromosome:

1. Set the variable *deficit* to 0. This variable indicates which child has one less edge than the others currently.

2. If the position is a 0 in each parent chromosome, then make the position 0 in both children. Likewise, if the position is a 1 in each parent chromosome, then make the position 1 in both children.

3. If position is different in each parent:

a) if *deficit*=0 then generate a random number *r* between 0 and 1. If $r < 0.5$, then set the position to 1 in child 1 and 0 in child 2. Set *deficit* to 2. If $r \geq 0.5$, do the opposite and set *deficit* to 1.

b) if *deficit* = 1, then set position to 1 in child 1 and 0 in child 2. Reset *deficit* to 0.

c) if *deficit* = 2, then set position to 0 in child 1 and 1 in child 2. Reset *deficit* to 0.

Because the number of edges (and therefore the number of 1's in the bitstring) must remain constant, the mutation method must ensure that if an edge is added, another edge is removed and vice versa:

1. For each mutated bit position, flip the bit by changing 0 to 1 and vice versa.

2. If 0 was changed to 1 (so an edge was added), generate a random number between 0 and *lchrom*-1 and check the bit at that position. If the bit is 1 then change it to 0. Otherwise continue to generate a random number *r* until the bit at position *r* is 1. Change that bit to 0.

An analogous operation is done when an edge is deleted during mutation.

5.3 RESULTS

Our first experiments consisted of simply running the GA-based generator and the random generator using identical input parameters and producing the same number of graphs. We found that the worst graphs from the GA generator were consistently of higher fitness than the ones from the random generator. Early results showed that the worst cases occurred when one or more of the vertices had only one or two edges and another vertex was matched to the neighbor first. For example, suppose vertex 8 in group N_2 has one edge (2, 8), but edges (2, 6) and (2, 7) are also in the graph. During the initial greedy matching, vertices 2 and 6 are matched, leaving vertex 8 to become an extra match with 2. This conflict can cause a chain reaction of rematchings until vertex 8 is finally matched with the only vertex possible. In looking at these graphs, we decided that giving priority to low-degreed vertices during matching might alleviate this problem. This, of course, also makes sense intuitively -- if there are only two possible matches for a vertex, there is a high probability that the final matching will include one of those edges.

To that end, we developed three new versions of the basic algorithm described in section 3.2:

- *Vertex-ordered greedy matching*: during the initial greedy matching, order the N_1 vertices by degree, from lowest to highest.
- *Degree-ordered edge selection*: during all processing, maintain the edges lists of both the N_1 and N_2 vertices in order. For edges *(u, v),* this order is determined by the degree of *v* and is in increasing order.
- *Combination:* Combine the two approaches.

Tables 1, 2, and 3 show the results of experiments with the basic algorithm and the three variations of density for variable edge count graphs. Algorithms 1 through 4 are the basic algorithm, vertex-ordered algorithm, edge-ordered algorithm, and combination algorithm respectively.

We ran two sets of experiments. First, we ran the GA generator for *g* generations and recorded the maximum fitness reached. We did this 10 times for each set of parameters. The numbers in the table represent the maximum, mean, and standard deviation of this data. Reasoning that longer chromosomes may require more time to converge, we used *g* = 200, 300, 400, 500 for *n* = 32, 64, 128, 256 respectively. The other GA parameters were *popsize*= 100, *pcross*= .9 and *pmutation*= .01. For the variable edge count graphs, we used three different densities for the tests, setting the edge probability so that: 1) $E(m) = n$, 2) $E(m) = n \log n$, and 3) $E(m) = n\, sqrt(n)$. For the fixed edge count graphs, we used three comparable densities: 1) $m = n$, 2) $m = n \log n$, and 3) $m = n\, sqrt(n)$.

In the second set of experiments, we used a random graph generator with the same edge probabilities to produce a comparable number of graphs. For *n* = 32, 64, 128, we generated 20100, 30100, and 40100 random graphs respectively. For each density and value of *n,* we ran the algorithm variation 10 times and recorded the maximum fitness produced. As with the GA data, the RAN information in the table includes the maximum, mean and standard deviation of these 10 runs.

Because we were using the `MRANDOM` random number generator, we were able to save the states of the random number generator at various points. Before starting the experiments, we created and saved 10 random number generator states. We did this by seeding the random number generator and running it for 100,000 iterations. We then saved the state. For the next state, we retrieved the first one and iterated 100,000,017 more times, saving the state at the end. For each of the other 8 states we used the previous state and iterated 100,000,017 times. We used this approach rather than reseeding the random number generator for two reasons. First, (Thomborson 1993) has noted problems with nonrandomness in the beginning of the sequences with various generators. Second, determining on a truly random number with which to seed each new start is virtually impossible. With these two considerations, we believe that our approach is the soundest.

For each set of parameters, run 1 started with the first random number state, run 2 started with the second, and so on. We did this so that we could be assured that the comparable genetic algorithms were starting out with the same initial populations. We did run into one problem in our original set of experiments. The number of iterations between each state and the next was too small so that they overlapped. This resulted in cycles in the fitnesses of the graphs generated using the random generators. We chose a large iteration in the states for the final experiments in order to avoid this problem.

Our goal in doing these two sets of experiments was twofold. First, we wanted to determine whether the GA-based generator was capable of finding harder graphs (in terms of the number of pushes required) than the random generator. Second, we wanted to compare the algorithm variations to see whether the same graphs are difficult for each.

In looking at Tables 1-3, it is clear that, especially at the lowest density, the GA generator is able to find significantly harder cases for the algorithm variations than are found using the random generator. At that density, the maximums found by the GA are approximately 1.7 to 3.9 times worse than the maximums found by the random generator. The means are 1.6 to 3.1 times worse. The GA-based generator is not as successful at the higher densities, but in most cases, the graphs found are more difficult than those produced by the random generator. Graphs with higher densities are easier to match than graphs with low densities so it is not surprising that the maximum fitness decreases as we move to the right in the table.

Table 1: Maximum/Mean(Std. Dev.) Fitnesses for Variable Edge Count Graphs – *E(m)=n*

n=32	**GA**	**RANDOM**
Alg. 1	184/151(28.3)	76/70(3.6)
Alg 2	158/111(30.1)	66/60(4.0)
Alg 3	168/126(23.1)	70/59(7.1)
Alg 4	130/82(24.3)	52/49(1.9)
n=64	**GA**	**RANDOM**
Alg 1	588/378(111.5)	156/137(10.8)
Alg 2	472/257(99.7)	156/125(13.5)
Alg 3	592/383(116.1)	146/120(13 2)
Alg 4	232/169(32.5)	136/102(13.1)
n=128	**GA**	**RANDOM**
Alg 1	1060/635(199.8)	308/241(26.1)
Alg 2	886/442(168.9)	234/214(11.7)
Alg 3	962/616(171.6)	248/221(13.8)
Alg 4	448/343(66.9)	216/201(6.5)
n=256	**GA**	**RANDOM**
Alg 1	1072/729(163.6)	564/458(51.7)
Alg 2	1322/641(252.8)	444/397(29.7)
Alg 3	1044/724(183.5)	536/438(52.6)
Alg 4	866/601(160.3)	438/378(31.5)

Table 2: Maximum/Mean(Std. Dev.) Fitnesses for Variable Edge Count Graphs - *E(m)=n log n*

n=32	**GA**	**RANDOM**
Alg. 1	118/101(7.5)	66/60(3.9)
Alg 2	100/90(6.6)	68/54(7.9)
Alg 3	70/53(6. 6)	42/42(0.0)
Alg 4	52/41(3.8)	40/40(0.8)
n=64	**GA**	**RANDOM**
Alg 1	238/197(15.3)	138/129(5,3)
Alg 2	172/159(8.7)	126/115(6.4)
Alg 3	164/122(26.4)	114/106(7.4)
Alg 4	160/105(27.3)	102/86(7.7)
n=128	**GA**	**RANDOM**
Alg 1	368/340(14.7)	352/255(35.4)
Alg 2	350/301(22.5)	236/225(7.4)
Alg 3	316/290(36.5)	224/209(11.4)
Alg 4	286/255(15.2)	210/184(11.4)
n=256	**GA**	**RANDOM**
Alg 1	686/628(33.0)	508/480(15.3)
Alg 2	560/518(27.7)	482/442(16.3)
Alg 3	756/ 605(63.7)	442/400(20.8)
Alg 4	532/489(22.5)	400/363(22.7)

Table 3: Maximum/Mean(Std. Dev.) Fitnesses for Variable Edge Count Graphs - *E(m)=n sqrt(n)*

n=32	**GA**	**RANDOM**
Alg. 1	96/79(21.8)	42/40(0.6)
Alg 2	40/40(0.0)	40/39(1.1)
Alg 3	40/38(1.5)	38/37(1.0)
Alg 4	36/36(0.0)	36/36(0.0)
n=64	**GA**	**RANDOM**
Alg 1	196/192(2.7)	98/86(5.4)
Alg 2	170/134(24.7)	100/78(8.4)
Alg 3	92/85(5.2)	76/73(1.4)
Alg 4	74/72(0.6)	72/72(0.6)
n=128	**GA**	**RANDOM**
Alg 1	388/320(49 9)	176/168(3.8)
Alg 2	312/236(40.4)	196/164(11.6)
Alg 3	186/167(8.8)	156/151(3.7)
Alg 4	166/159(4.7)	152/144(4.9)
n=256	**GA**	**RANDOM**
Alg 1	638/472(78.2)	346/327(8.6)
Alg 2	406/362(24.2)	324/316(5.3)
Alg 3	324/307(7.8)	312/301(6.8)
Alg 4	338/299(14.5)	302/292(5.2)

In relation to our second question, the results indicate that considering the vertex degrees in processing may yield an improvement in performance. In almost all cases, the hardest graph found for the degree-based variations is not as difficult as the hardest graph found for the basic algorithm. But this difference could possibly be attributed to the GA not being able to find and exploit structures that yield hard cases for the variations. We tested this conjecture by running one of the variations (algorithm 4) on the graphs produced by the GA-based generator when the basic algorithm was used in the fitness function. Table 4 shows these average percentage improvements. Clearly, the graphs that are difficult for basic algorithm are not as difficult for the degree-based variation. So the improvement to the algorithm shows promise in handling the structures that produce problems for the basic algorithm.

We repeated the first two experiments for the fixed edge graphs. The results are not shown here, but we found that the GA generator did not do nearly as well against the random generator on these graphs. Improvement over random generation ranged from 0 to 2.8 times for the maximums and 0 to 2.1 times for the means. The reason for these results probably lies in the crossover method in use for the fixed edge graphs, which differs from the partial-v crossover used for the variable edge count graphs. Because we need to maintain a constant number of edges, we ignore the vertex boundaries in our representation, placing edges at random in either child 1 or child 2. This causes loss of graph degree structure from generation to generation. Since this degree structure may be important in determining the difficulty of the graph for the algorithm, the disappointing performance was not unexpected.

Table 4: Mean Improvement of Combination Algorithm over Basic Algorithm - Variable Edge Count Graphs

	GA	**RAN**
	E(m)=n	
n=32	77%	53.2%
n=64	80.5%	50.9%
n=128	74.8%	45.3%
	E(m) = n log n	
n=32	67.9%	45.8%
n=64	66.7%	48.9%
n=128	61.1%	48.2%
	E(m)=n sqrt(n)	
n=32	57.3%	20.1%
n=64	66.5%	24.9%
n=128	58.2%	22.7%

6 CONCLUSIONS

The GA-based generator shows promise as a tool for exploring pathological cases for algorithms. In particular, it succeeded in producing cases that were significantly harder (up to 3.9 times) than those generated by a random generator for the same algorithm. While we focussed on a particular graph-based problem, the same approach can be taken on other problems where the structure of the input affects the performance of the algorithm.

The work also suggested an adjustment to the Goldberg/Tarjan algorithm that may lead to a faster running time on some input. Early experiments show promise, but more examination of the variation is needed to ensure that the time for processing does not cause too much overhead in performance.

REFERENCES

Cherkassky, B. and Goldberg, A. V. and Radzik, T., "Shortest Paths Algorithms: Theory and Experimental Evaluation," Technical Report STAN-CS-93-1480, Stanford University, 1993.

Goldberg, A. V., "Selecting Problems for Algorithm Evaluation," Technical Report #98-142, NEC Research Institute, Inc., 1998.

Goldberg, A. V. and Kennedy, R., "Global price updates help," Technical Report CS-TR-94-1509, Stanford University, March 1994.

Goldberg, A.V. and Tarjan, R. E., "A new approach to the maximum flow problem," Proceedings of the 18th Annual

ACM Symposium on Theory of Computing, 1986, 136-146.

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

Smith, R.E., Goldberg, D.E., and Earickson, J.A., “SGA-C: a C-language implementation of a simple genetic algorithm,” Technical Report TCGA, No. 91002, The Clearinghouse for Genetic Algorithms, The University of Alabama, March 1994.

Hooker, J. N., "Needed: An Empirical Science of Algorithms, " Operations Research, 1994, 42, 201-212.

Johnson, D. S. and McGeoch, C. C., editors, "Network Flows and Matching: First DIMACS Implementation Challenge,” AMS, 1993.

McGeoch, C., “Analyzing Algorithms by Simulation: Variance Reduction Techniques and Simulation Speedups," ACM Computing Surveys, 1992, 24(2), 195-212.

Mitchell, Melanie, An Introduction to Genetic Algorithms, The MIT Press, 1996.

Moret, B.M.E. and Shapiro, H. D., “Empirical Analysis of algorithms for constructing a minimum spanning tree,” Computational Support for Discrete Mathematics, AMS, 1994, 99-117.

Sanchis, L.A., “Test construction for the vertex cover problem,” Computational Support for Discrete Mathematics, AMS, 1994, 315-326.

Setubal, J. C., “New experimental results for bipartite matching,” Technical Report DCC-07/92, State University of Campinas, November 1992.

Thomborson, C., “An introduction to `mrandom` 3.0,” unpublished manuscript, 1993.

Wegener, J., Sthamer, H., Jones, B., and Eyres, D., “Testing real-time systems using genetic algorithms,” Software Quality Journal, 6, 1997, 127-135.

Adaptive and Dynamic Elevator Group Control with a Genetic Algorithm

Jung-Hwan Kim and Byung-Ro Moon
School of Computer Science and Engineering
Seoul National University
Shilim-dong, Kwanak-gu, Seoul, 151-742 Korea
{aram,moon}@soar.snu.ac.kr

Abstract

A new elevator-group-control system is proposed. The system learns dynamic traffic flows by analyzing passenger traffic without prespecified patterns. A genetic algorithm continuously generates dispatch functions according to changes in passenger traffic. By considering the status inside elevators, the directions of passenger movement, and the number of waiting passengers, the system occasionally allocates *multiple* elevators for a single hall call, which assists in reducing passengers' waiting time. Experimental results showed up to 25% improvement over a system without the above features.

1 Introduction

In a tall building with multiple elevators, it is a notoriously difficult task to control the elevators in the most efficient manner. In general, the objectives of elevator control systems differ from building to building; the most common goals are to minimize passengers' average waiting time, to minimize average riding time, and to balance crowding in elevators. Optimizing an elevator-group-control system to achieve these objectives is difficult for various reasons including the following [20]: coordination of multiple cars, constraints on elevators' movements, incomplete information (e.g., after a button is pressed at a floor, it is impossible to know how many passengers are waiting at that floor), unknown passenger traffic patterns, and the existence of special-purpose elevators or floors [23]. In order to deal with these difficulties, conventional elevator-group-control systems have used fuzzy systems [10][12][13][16], artificial neural networks [2][6][17][22], genetic algorithms [7][8][23], etc.

Passenger traffic is conventionally classified into the following four patterns [5][20][22]: i) uppeak-traffic: most passengers move up from the first floor and downward movements are rare (mostly in the early morning), ii) downpeak-traffic: most passengers move down to the first floor and upward movements are rare (mostly in the evening), iii) lunchtime-traffic: many passengers move up from and down to the first floor, and iv) interfloor-traffic: passengers move up and down freely among several different floors with few specific patterns. Some studies have fully or partially focused on uppeak-traffic patterns [2][5][20] or on lunchtime-traffic patterns [7]. Other researchers [10][17] have proposed adaptation techniques for dynamic flows, which prepare a set of prespecified traffic patterns in advance and switch between policies appropriate for the specific patterns.

The system proposed in this paper contains two key ideas. First, a genetic algorithm (GA) continuously generates dispatch functions by adaptation to changes in passenger traffic. Although the systems of [10] and [17] also change dispatch functions during the running of the systems, they employ *prespecified* traffic patterns. The proposed system does not prepare any such patterns in advance. Second, it tries to reduce passengers' waiting time by *multiple elevator allocation* when it is expected that one elevator cannot serve all the passengers for a hall call at a floor. This is supported by setting a camera at each floor and estimating the numbers of passengers for upward and downward movement.

The GA optimizes the parameters of elevator-dispatch functions and contain a local improvement heuristic to help fine-tuning. By occasionally allocating multiple elevators for a single call at a floor, the system can reduce passengers' average waiting time. However, as it is based on prediction, the accuracy of the prediction is critical. This strategy is helpful when the gain as a result of good predictions is greater than the unavoidable loss due to wrong predictions.

The remainder of this paper is organized as follows. In the next section, we provide an overview of elevator-group-control systems that have been previously proposed. In Section 3, we describe our proposed elevator-group-control system and the key ideas in detail. In Section 4, we present our experimental results and mention our conclusions in Section 5.

2 Previous Work

A *hall call* is an event resulting from a passenger pushing one of the up or down buttons at a floor; a *car call* is an event caused by a passenger pushing a destination-floor button inside a car (elevator).

When a hall call is issued, the control system has to evaluate the attractiveness of each car. Some studies [12][13][16] used fuzzy systems to generate attractiveness evaluation functions. They evaluate each car using a fuzzy function and assign a car with the greatest function value to the hall call. Some of the studies considered just passengers' waiting time [12][16]; some additionally considered passengers' riding time [13]. Fujino *et al.* [7][8] used genetic algorithms to optimize the control system with preferential floors.

Passenger traffic continuously changes over time. A uniform control policy not considering the traffic patterns has an inevitable limit in reducing the passengers' waiting time. Dewen *et al.* [6] and Markon *et al.* [17] proposed learning paradigms by neural networks. The systems choose a control policy by having neural networks identify the most similar traffic pattern to the current flow among a set of ready-prepared traffic patterns.

Since the uppeak traffic pattern is relatively simple and occurs with great frequency, a number of studies have been done based on it [3][5][20]. A possible policy is that each car serves a particular group of floors [5], where the groups are usually disjoint from each other. Pepyne *et al.* [20] also assumed uppeak traffic and devised a policy in which each elevator waits until the number of passengers inside it reaches a threshold. These policies helped reduce the waiting time.

If we could know the destination floor of every passenger, we may be able to further reduce passengers' waiting time. However, conventional elevator systems have only two hall-call buttons (upward and downward) at a floor, and it is not possible to predict passengers' movements or to guess how many passengers are waiting. Amano *et al.* [2] proposed an elevator system where there are destination-floor buttons at each floor. A passenger pushes the specific destination-floor button. This provides more information than the "up/down button"-based systems. Nonetheless such a system still cannot know the number of passengers who want to go to the destination floor.

It is important to handle dynamic passenger traffic in order to reduce passengers' waiting time. Elevator allocation must be done in real time, but deciding on a dispatching strategy does not have to be done in real time (e.g., a one minute delay presents no difficulties). Thus, a genetic algorithm (which cannot easily provide a real-time solution) can be used on a semi-online basis. Previous work did (could) not consider the number of waiting passengers at each floor. The greater the number of waiting passengers, the longer is the expected waiting time. In our study, we set a camera at each floor and obtain information on the number of waiting passengers.

3 The Proposed System

In previous work, when a hall call is issued, the system just knows that the number of waiting passengers is at least one. This information is intrinsically insufficient. We thus install a camera at each floor to obtain more information. The cameras for this purpose are not very expensive. Moreover, these days more and more buildings set cameras in the halls (particularly around the elevator entrances) for the purpose of security. The sharply expanding market of DVR (Digital Video Recorder) is an evidence. The suggested system can take an almost free ride in this case. State-of-the-art pattern recognition techniques have no difficulty in counting the number of people from an image captured in a bounded area [14][15][24]. However, we still cannot clearly know all passengers' intentions (moving directions). In our work, we predict the numbers of upward and downward passengers by a simple rule. Rather than describing the rule in overt detail, we sketch it using a simple example. If there are k people on a floor with only one of the two buttons pushed, all the people are for the same direction. After that, if another button is also pushed and the number of people grows to m, the number of people for the two directions are divided into $k+w_1(m-k)$ and $(1-w_1)(m-k)$ where w_1 and $1-w_1$ are weighting factors set based on the past history of activity at the floor. After a car serves l people without filling its capacity, all the remaining $m-l$ people are for the other direction. This information is utilized in deciding a dispatching strategy.

The proposed system is composed of two units (Figure 1). The first unit, GCU (Group Control Unit), selects

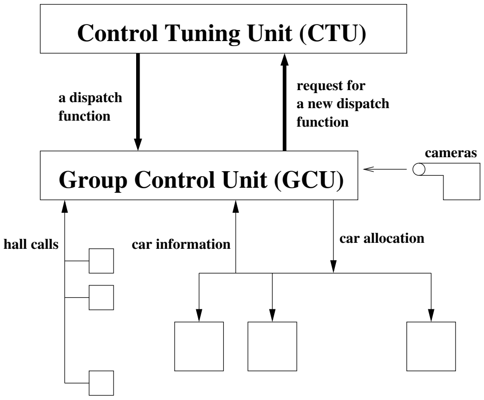


Figure 1: Structure of the system

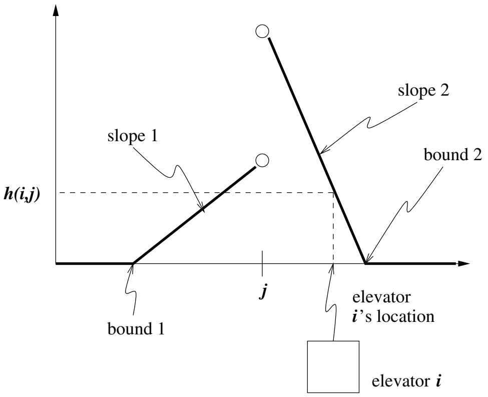


Figure 2: The shape of $h(i,j)$

a car using a dispatch function. This dispatcher keeps being updated by the other unit CTU. CTU (Control Tuning Unit) is a background procedure that produces a dispatch function considering passenger traffic. GCU controls all the cars and keeps checking the traffic flow. If the passenger traffic has changed remarkably, GCU requests CTU to generate a new dispatch function. GCU is a real time procedure; on the other hand, CTU is a semi-online procedure with a time budget of a few minutes.

3.1 Dispatch Function Generation

In selecting a car to serve a hall call, there are a number of factors to consider. For example, if a car is distant from the hall-called floor, it is desirable to have some penalty; if the car has been assigned to serve a floor near the hall-called floor, it is desirable to have some reward; if the car has car calls for a number of floors between its current location and the hall-called floor, it is desirable to have some penalty; the current crowding in the car also affects its merit. We need to have a dispatch function which considers all these factors. It is almost impossible to have an efficient dispatch function in advance because we do not have information on the passenger traffic and diverse combinations of the above factors. In our system, the passenger traffic is carefully monitored and the elevator-dispatch function keeps changing (in CTU) based on the traffic.

In the following, we describe the function that is tuned by a GA. We assume that there is a hall call in the floor j and the system wants to evaluate the merit of the car i. We denote by $f(i,j)$ the merit function of the car i for a hall call at the floor j. The function has 12 parameters for tuning as follows:

$$f(i,j) = w_1 h(i,j) + w_2 g(i,j) + w_3 c(i) + w_4 t(i)$$

where

- $h(i,j)$: a function that reflects the distance between the floor j and the elevator i's current location. Figure 2 shows the shape of the function $h(i,j)$. In the function, the reward of a floor (car i's location) depends on the slopes and bounds showed in the figure. Thus $h(i,j)$ itself has four parameters–two slopes and two bounds–to be tuned.
- $g(i,j)$: a function that gives some reward if the elevator i is already assigned to serve a floor near the floor j. This has a similar shape to $h(i,j)$ and also has four parameters.
- $c(i)$: the elevator i's crowding.
- $t(i)$: the number of car calls for the floors between the floor j and the elevator i's current location.

CTU tunes, by means of the GA, the four weighting factors (w_1 through w_4) and the eight parameters of $h(i,j)$ and $g(i,j)$ based on recent traffic. The GA procedure for tuning these parameters is described in Section 3.2. When a considerable change in the passenger traffic is detected, GCU gives CTU the information and requests the generation of a new dispatch function. The system can approximately guess the arrival rates of people by periodically tracking the total numbers of people (in the halls and the cars). If the moving averages of these rates considerably change, the system judges that the traffic has changed. It takes one minute or so for CTU to generate a new function by the GA; taking one minute to prepare a new function is not so critical unless the traffic fluctuates too frequently.

```
Create initial population of fixed size;
do {
    choose parent1 and parent2 from population;
    offspring = crossover(parent1, parent2);
    mutation(offspring);
    local-improvement(offspring);
    replace(population, offspring);
} until (stopping condition);
return the best solution;
```

Figure 3: The hybrid genetic algorithm framework we used

3.2 Tuning the Adaptive Function by a GA

Genetic Algorithms (GAs) are stochastic algorithms which mimic the natural evolution of population genetics in problem solving or simulation. A GA is known to have wide search capability and a good balance between exploitation and exploration of the search space [9][19].

In this study, the GA seeks for the optimal set of parameters for the elevator group controller (described in Section 3.1). The goal is to search parameter values that minimize the average passengers' waiting time. We use a steady-state hybrid genetic algorithm for tuning parameters of a dispatch function $f(i, j)$. The template of the GA is shown in Figure 3. Two parents are selected according to their probabilities that are proportional to their fitness values. The probability that the best solution is chosen is given four times that of the worst solution is chosen. This selection scheme prevents severe discrimination against poor solutions, and it is a common selection technique in genetic algorithm design. The offspring is produced through a traditional multi-point crossover. After an offspring is modified by a mutation operator, it is locally improved. The local improvement approach is described in Section 3.2.2. The local improved offspring replaces a solution in the population by the following rule [4]: the more similar parent to the offspring is replaced if the offspring is better, otherwise, the other parent is replaced if the offspring is better, if not again, the worst chromosome in the population is replaced. The rational behind this is to maintain the population diversity to the extent that not too much time is wasted [4]. In this experiment, the population size is set to 50 and a linear real-number encoding scheme is used. The GA stops after a fixed number of generations. To evaluate a chromosome, the GA simulates the group control with the corresponding parameters.

3.2.1 Non-Uniform mutation

Usually, steady-state GAs converge faster than generational GAs with more chances of genetic drift. Stronger mutation can alleviate steady-state GAs' genetic drift. Generally the solutions have poor qualities in early generations. As the generation grows, the qualities of the solutions get better. Consequently, a large mutation rate rarely contributes to the improvement of quality in the latter stage of a GA. In this work, we used a non-uniform mutation. This mutation changes the rates of perturbation as the generations go. The non-uniform mutation is performed as follows [18]: if $s_v^t = < v_1, \cdots, v_m >$ is chromosome (t is the generation number) and the elements v_k was selected for this mutation, the result is a vector $s_v^{t+1} = < v_1, \cdots, v_k', \cdots, v_m >$, such that

$$v_k' = \begin{cases} v_k + \Delta(t, UB - v_k) & \text{if a random digit is 0,} \\ v_k - \Delta(t, v_k - LB) & \text{if the random digit is 1} \end{cases}$$

where LB, UB are the lower and upper bounds of the variable v_k. The function $\Delta(t, y)$ returns value in the range $[0, y]$ such that the probability of $\Delta(t, y)$ being close to 0 increases as t increases. We used the following function as in Michalewicz [18]:

$$\Delta(t, y) = y \cdot \left(1 - r^{\left(1 - \frac{t}{T}\right)^b}\right)$$

where r is a random number from $[0, 1]$, T is the maximal generation number, and b is a constant value. We used $T = 200$ and $b = 5$.

3.2.2 Local Improvement

For a dispatch function f, we have 12 parameters $x_1, x_2, \cdots, x_{12}$ to be tuned by a GA. Each x_i corresponds to a gene in the GA. We devised a simple discrete improvement heuristic. First, we fix the parameters $(x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_{12})$ except x_i. The parameter x_i is modified to $x_i' \in \mathcal{N}(x_i)$ where $\mathcal{N}(x_i) = \{x| \mid x - x_i \mid \leq \alpha\}$, $\alpha \in \mathbb{R}$. Then, CTU simulates the group control with $(x_1, \cdots, x_i', \cdots, x_{12})$ and evaluates the attractiveness. This process is repeated with a number of x_i's which changes with a step size, say Δx, in the range $[x_i - \alpha, x_i + \alpha]$. Finally, the most attractive x_i' is selected. Since the simulation is not very cheap, we choose only one x_i in a generation. The index i is chosen at random in each generation.

3.3 Prediction-Based Multiple Allocation

When a car arrives at the destination floor and the remaining capacity of the car turns out to be not enough to serve all the waiting passengers, some passengers

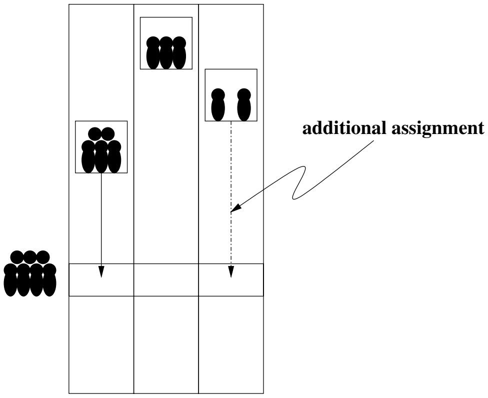


Figure 4: Multiple elevator allocation based on prediction

have to wait until another car comes. In previous work, the systems select another car to serve the remaining passengers right after this problem has occurred. This is an important factor contributing towards the increase of passengers' waiting time according to our investigation. If a system predicts the number of waiting passengers at the hall-called floor in advance, it can result in greater efficiency. This is an important feature of the proposed system, which periodically analyzes and predicts passengers' movements with the help of cameras.

Assume the car i is allocated in response to a hall call at the floor j. Let t_{ij} be the expected time for the car i to arrive at the floor j, w_{ij} be the expected number of waiting passengers at the floor j after t_{ij}, and c_{ij} be the expected crowding after t_{ij}. The system judges based on w_{ij} and c_{ij} whether or not the car i can serve all the passengers at the floor j. If it is not expected to be able to serve all the passengers, the system selects another car (Figure 4).

The proposed control system decides control strategies based on prediction. If the predictions are accurate, it reduces passengers' waiting time; if not, it may do harm to the system. To enhance the accuracy of prediction, the system periodically analyses passengers' movements with the help of cameras. When, contrary to prediction, the first car turns out to be able to carry all the waiting passengers at the floor j and the second car does not have a car call to the floor j, the system promptly cancels the second car's schedule for the floor j. Note that it is still not possible to get perfect information since there are only two hall-call buttons (upward/downward). This strategy is helpful when the gain by good predictions is greater than the unavoidable loss by wrong predictions.

Table 1: Simulation Conditions

Items	Settings
# of Floors	18
# of elevators	6
capacity	20 people/car
moving speed	$2t$ /floor
open-close time	$2t$
boarding time	$1.3t$

* t : the unit time of a Poisson process

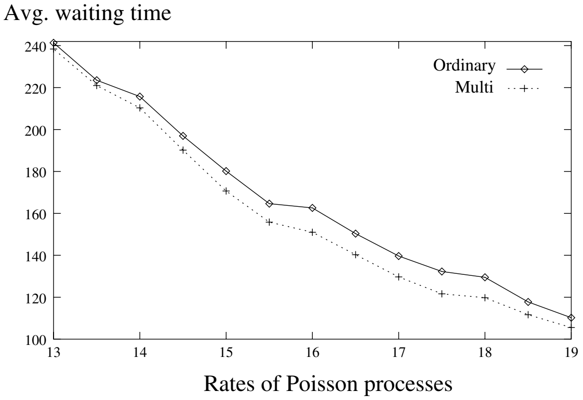


Figure 5: Average waiting time according to traffic patterns

4 Experimental Results

Table 1 shows the experimental conditions used in this paper. There are 18 floors and 6 elevators. Each elevator can serve up to 20 people. In practice, the number of passengers in a car is usually measured by an onboard scale.

The test was performed by simulations following the tradition. The passenger traffic followed a Poisson process as usual [1][8][11]. Let $N(t)$ be the number of passenger arrivals in any interval of length t. Then it is defined by a Poisson process that for all $s, t \geq 0$

$$P\{N(t+s)-N(s)=n\} = e^{-\lambda t}\frac{(\lambda t)^n}{n!}, \qquad n = 0, 1, \cdots.$$

The mean time between two arrivals is known to be λt. λ is called the rate of the process [21]. We did not assume any constraint in the Poisson process; thus it is similar to the interfloor traffic (regarding to the four traffic patterns in the introduction) as in most other researches.

We first investigate the effect of multiple allocation with cameras; this is shown in Figure 5. In the fig-

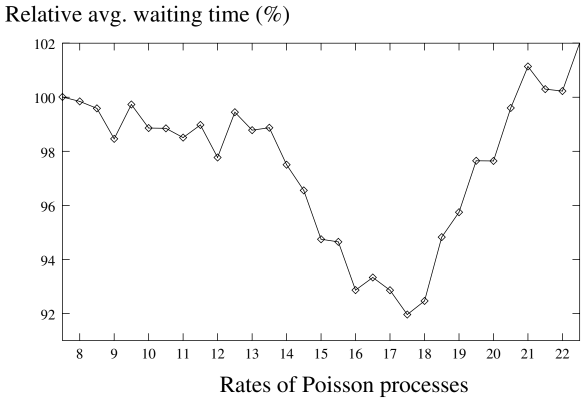


Figure 6: Relative average waiting time of the multi-allocation-based system against the system with no camera

ure, "Ordinary" represents the system without multiple allocations and "Multi" represents the system with multiple allocations. Both systems used dispatch functions that were generated by a genetic algorithm. The only difference is the existence of multiple allocations. The horizontal axis represents the rates (λ) of Poisson processes and the vertical axis represents the average waiting time in seconds. The result shows that the existence of multiple allocations significantly decreased passengers' average waiting time. The absolute waiting time is not an indicator for the usefulness of the suggested system. The average waiting time depends on the experimental settings. Rather, if the situation "not enough space in the assigned car and one more hall call for the next car" occurs occasionally, the suggested system would be helpful.

Figure 6 expands Figure 5 over a wider spectrum of traffic patterns. It shows the multi-allocation-based system's *relative* performance against the ordinary system. The average waiting time of the ordinary system was set to 100. When passengers arrive at each floor with process rate 17.5, the multi-allocation-based system showed greater than 8% improvement. The improvement was not visible in cases of too heavy or too light traffic. When the traffic is very heavy, the multi-allocation mechanism even did slight harm to the system. The results of Figure 5 and Figure 6 are the average from 100 runs for each process rate.

Usually the traffic flow fluctuates over time in most buildings during a day. Although a Poisson process handles irregular passenger arrivals, it is not uncommon to have far more fluctuating traffic than a Poisson process can handle. To simulate a tougher situation, we also created a nonhomogeneous Poisson process [21] where the rate (λ) of the process itself changes according to another Poisson process. In other words, the events of the rate-change follow another Poisson process. This is a harder situation for the control system to adapt to. If the system successfully adapts itself to the dynamic traffic flows, it may find a better dispatch function and the passengers' waiting time may decrease. When a considerable change in the passenger traffic is detected, GCU asks CTU to run a GA and generate a new dispatch function.

Table 2: Types of Systems

Types	GA adaptation to traffic flows	Multi-allocation
Type 1	N	N
Type 2	N	Y
Type 3	Y	N
Type 4	Y	Y

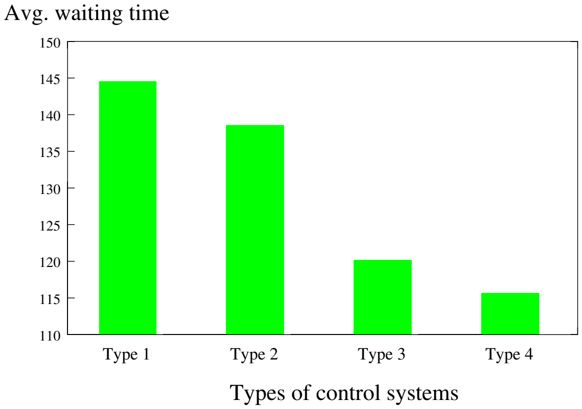


Figure 7: Average waiting times of four versions

In order to examine the effects of GA adaptation and of the multiple allocation, we tested four versions of systems tabulated in Table 2. Figure 7 shows the average waiting times of the four systems. The result shows that the GA adaptation to dynamic flows greatly affected the performance independent of the existence of cameras. Figure 8 shows the average crowding inside a car. The elevators were more evenly utilized in the cases with GA adaptation.

5 Conclusion

There are two key ideas in this paper. First, the dispatching function continuously changes by a genetic algorithm that carefully considers passenger traffic. Second, multi-elevator allocations sometimes occur with the help of cameras. The proposed system adapts itself to dynamic traffic flows, which led to significant

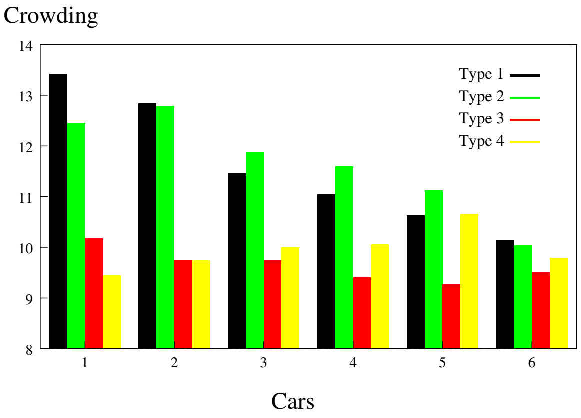


Figure 8: Crowding of elevators

improvement on the average. When combined with multi-elevator allocation, further improvement was observed.

Note that the multiple allocation were not useful when the traffic flow was extremely heavy or light. The experimental results showed that the use of multiple allocation notably decreased the average waiting time when the average crowding of elevators reached between 45% and 75% of their capacity. The waiting time did not decrease at all when the average crowding was, e.g., 30% or 85%. But when the average crowding was around 55%, the average waiting time decreased over 8%.

Incorporating both of the key ideas, up to 25% improvement was observed. If the system can predict situations more accurately, one can expect further improvement. More accurate prediction is left for future study.

Acknowledgement

This work was supported by Brain Korea 21 Project.

References

[1] J. Alander, J. Ylinen, and T. Tyni. Optimizing elevator control parameters. In *Second Finnish Workshop on Genetic Algorithms and Their Applications*, pages 105–114, 1994.

[2] M. Amano, M. Yamazaki, and H. Ikejima. The latest elevator group supervisory control system. *Mitsubishi Electric ADVANCE*, 67:88–95, 1996.

[3] G. Barney and S. Santos. *Elevator Traffic Analysis, Design and Control, 2nd Ed.* Peter Peregrinus, 1985.

[4] T. N. Bui and B. R. Moon. Genetic algorithm and graph partitioning. *IEEE Trans. on Computers*, 45(7):841–855, 1996.

[5] W. Chan and T. So. Dynamic zoning for intelligent supervisory control. *International Journal of Elevator Engineering*, 1:47–59, 1996.

[6] Z. Dewen, J. Li, Z. Yuwen, S. Guanghui, and H. Kai. Modern elevator group supervisory control systems and neural networks technique. In *1997 IEEE International Conference on Intelligent Processing Systems*, pages 528–532, 1997.

[7] A. Fujino, T. Tobita, K. Segawa, K. Yoneda, and A Togawa. An elevator group control system with floor attribute control method and system optimization using genetic algorithms. In *21st International conference on Industrial Electronics, Control, and Instrumentation*, pages 1502–1507, 1995.

[8] A. Fujino, T. Tobita, K. Segawa, K. Yoneda, and A. Togawa. An elevator group control system with floor-attribute control method and system optimization using genetic algorithms. *IEEE Transactions on Industrial Electronics*, 44(4):546–552, 1997.

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

[10] R. Gudwin, F. Gomide, and M. Netto. A fuzzy elevator group controller with linear context adaptation. In *1998 IEEE International Conference on Fuzzy Systems*, pages 481–486, 1998.

[11] T. Hikihara and S. Ueshima. Emergent synchronization in multi-elevator system and dispatching control. *IEICE Transactions on Fundamentals*, E80-A(9):1548–1553, 1997.

[12] K. Igarashi, S. Take, and T. Ishikawa. Supervisory control for elevator group with fuzzy expert system. In *IEEE International Conference of Industrial Technology*, pages 133–137, 1994.

[13] M. Kaneko, T. Ishikawa, and Y. Sogawa. Supervisory control for elevator group by using fuzzy expert system. In *23rd International Conference on Industrial Electronics, Control, and Instrumentation*, pages 370–376, 1997.

[14] V. Kettnaker and R. Zabih. Counting people from multiple cameras. In *IEEE International Conference on Multimedia Computing and Systems*, volume 2, pages 267–271, 1999.

[15] L. Khoudour, J. Deparis, and L. Duvieubourg. Linear image sequence analysis for passengers counting in public transport. In *International Conference on Public Transport Electronic Systems*, number 425, pages 100–104, 1996.

[16] K. Kurosawa and K. Hirasawa. Intelligent and supervisory control for elevator group. *Transactions of Information Processing Society of Japan*, 26(2):278–287, 1985.

[17] S. Markon, H. Kita, and Y. Nishikawa. Adaptive optimal elevator group control by use of neural networks. *Transaction of the Institute of System, Control and Information Engineers*, 7:487–497, 1994.

[18] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolutionary Programs.* Springer, 1992.

[19] M. Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, 1996.

[20] D. Pepyne and C. Cassandras. Optimal dispatching control for elevator systems during uppeak traffic. *IEEE Transactions on Control Systems Technology*, 5(6):629–643, 1997.

[21] S. Ross. *Introduction to Probability Models, 5th Ed.* Academic Press, 1993.

[22] A. So, J. Beebe, W. Chan, and S. Liu. Elevator traffic pattern recognition by artificial neural network. *Elevator Technology 6*, pages 122–131, 1995.

[23] T. Tobita, A. Fujino, K. Segawa, K. Yoneda, and Y. Ichikawa. A parameter tuning method using genetic algorithms for an elevator group control system. In *22nd International Conference on Industrial Electronics, Control, and Instrumentation*, pages 823–828, 1996.

[24] X. Zhang and G. Sexton. Automatic human head location for pedestrian counting. In *Sixth International Conference on Image Processing and Its Applications*, volume 2, pages 535–540, 1997.

An Evaluation of Negative Selection in an Artificial Immune System for Network Intrusion Detection

Jungwon Kim and Peter J. Bentley

Department of Computer Science
University College London
Gower Street, London
U.K.
Email: {J.Kim, P.Bentley}@cs.ucl.ac.uk

Abstract

This paper investigates the role of negative selection in an artificial immune system (AIS) for network intrusion detection. The work focuses on the use of negative selection as a network traffic anomaly detector. The results of the negative selection algorithm experiments show a severe scaling problem for handling real network traffic data. The paper concludes by suggesting that the most appropriate use of negative selection in the AIS is as a filter for invalid detectors, not the generation of competent detectors.

1 INTRODUCTION

The biological immune system has been successful at protecting the human body against a vast variety of foreign pathogens (Tizard, 1995). A growing number of computer scientists have carefully studied the success of this competent natural mechanism and proposed computer immune models for solving various problems including fault diagnosis, virus detection, and mortgage fraud detection (Dasgupta, 1998; Kephart et al,1995).

Among these various areas, intrusion detection is a vigorous research area where the employment of an artificial immune system (AIS) has been examined (Dasgupta, 1998; Kim and Bentley, 1999b; Hofmeyr, 1999; Hofmeyr and Forrest, 2000; Forrest and Hofmeyr, 2000). The main goal of intrusion detection is to detect unauthorised use, misuse and abuse of computer systems by both system insiders and external intruders. Currently many network-based intrusion detection systems (IDS's) have been developed using diverse approaches (Mykerjee et al, 1994). Nevertheless, there still remain unresolved problems to build an effective network-based IDS (Kim and Bentley, 1999a). As one approach of providing the solutions of these problems, previous work (Kim and Bentley, 1999a) identified a set of general requirements for a successful network-based IDS and three design goals to satisfy these requirements: being distributed, self-organising and lightweight. In addition, Kim and Bentley (1999a) introduced a number of remarkable features of human immune systems that satisfy these three design goals. It is anticipated that the adoption of these features should help the construction of an effective network-based IDS.

An overall artificial immune model for network intrusion detection presented in (Kim and Bentley, 1999b) consists of three different evolutionary stages: negative selection, clonal selection, and gene library evolution. This model is not the first attempt to develop an AIS for network intrusion detection. Various approaches to build an AIS have been attempted mainly by implementing only a small subset of overall human immune mechanisms (Dasgupta, 1998). This is because the nature of human immune systems is very complicated and sophisticated and thus it is very difficult to implement perfect human immune processes on a computer. However, as seen from other immunology literature (Paul, 1993; Tizard, 1995), an overall immune reaction is the carefully co-ordinated result of numerous components such as cells, chemical signals, enzyme, etc. Therefore, the omission of crucial components in order to make the development of AIS simpler and more applicable may detrimentally affect the performance of an AIS. This implies that appropriate artificial immune responses can be expected only if the roles of crucial components of human immune systems are correctly understood and they are implemented in the right way.

In this paper, we continue our effort to understand the roles of important components of artificial immune systems especially for providing appropriate artificial immune responses against network intrusions. Following our previous work identifying three different evolutionary stages: negative selection, clonal selection, and gene library evolution, of AIS by extensive literature study (Kim and Bentley, 1999a; 1999b), this paper focuses on the investigation of the roles of first stage: negative selection. With implementation details of this stage, this work presents how and which aspects of negative selection can contribute to the development of an effective network-based IDS.

2 BACKGROUND

2.1 NEGATIVE SELECTION OF THE HUMAN IMMUNE SYSTEM

An important feature of the human immune systems is its ability to maintain diversity and generality. It is able to detect a vast number of antigens with a smaller number of antibodies. In order to make this possible, it is equipped with several useful functions (Kim and Bentley, 1999a). One such function is the development of mature antibodies through the gene expression process. The human immune system makes use of gene libraries in two types of organs called the thymus and the bone marrow. When a new antibody is generated, the gene segments of different gene libraries are randomly selected and concatenated in a random order, see figure 1. The main idea of this gene expression mechanism is that a vast number of new antibodies can be generated from new combinations of gene segments in the gene libraries.

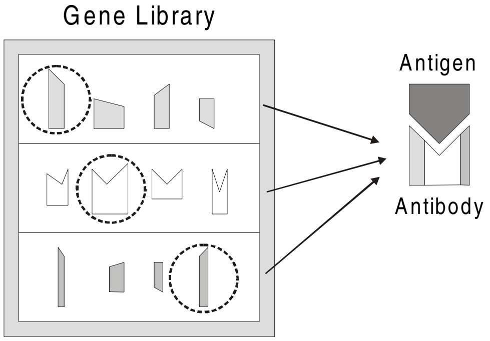


Figure 1 Gene Expression Process

However, this mechanism introduces a critical problem. The new antibody can bind not only to harmful antigens but also to essential self cells. To help prevent such serious damage, the human immune system employs negative selection. This process eliminates immature antibodies, which bind to self cells passing by the thymus and the bone marrow. From newly generated antibodies, only those which do not bind to any self cell are released from the thymus and the bone marrow and distributed throughout the whole human body to monitor other living cells. Therefore, the negative selection stage of the human immune system is important to assure that the generated antibodies do not to attack self cells.

2.2 THE NEGATIVE SELECTION ALGORITHM

Forrest et al (1994; 1997) proposed and used a negative selection algorithm for various anomaly detection problems. This algorithm defines 'self' by building the normal behaviour patterns of a monitored system. It generates a number of random patterns that are compared to each self pattern defined. If any randomly generated pattern matches a self pattern, this pattern fails to become a detector and thus it is removed. Otherwise, it becomes a 'detector' pattern and monitors subsequent profiled patterns of the monitored system. During the monitoring stage, if a 'detector' pattern matches any newly profiled pattern, it is then considered that new anomaly must have occurred in the monitored system.

This negative selection algorithm has been successfully applied to detect computer viruses (Forrest et al., 1994), tool breakage detection and time-series anomaly detection (Dasgupta, 1998) and network intrusion detection (Hofmeyr, 1999; Hofmeyr and Forrest, 2000; Forrest and Hofmeyr, 2000). Besides these practical results, D'haeseleer (1997) showed several advantages of negative selection as a novel distributed anomaly detection approach.

3 ALGORITHM OVERVIEW

This work used a negative selection algorithm to build an anomaly detector. This was achieved by generating detectors containing non-self patterns. The overview of this algorithm is provided in figure 2 and 3. The negative selection algorithm for network intrusion detection used in this paper follows the algorithm of Forrest et al (1994, 1997), described in the previous section. 'Self' was built by profiling the activities of each single network connection. The detail of self profiling is described in the next section.

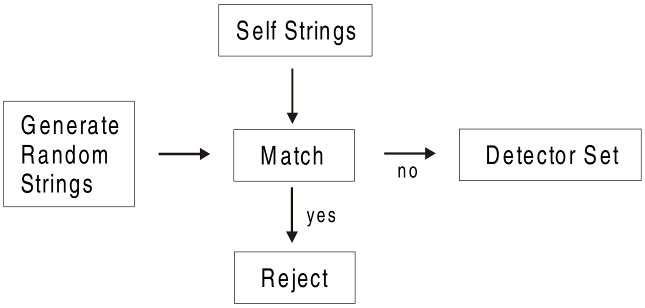


Figure 2 Detector Set Generation of
a Negative Selection Algorithm (Forrest et al, 1995)

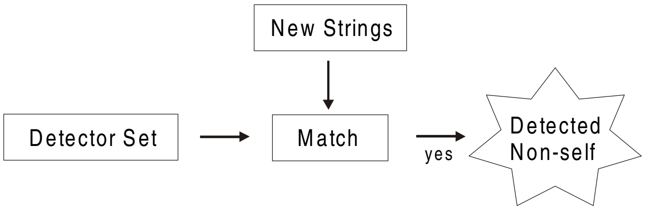


Figure 3 Non-Self Detection by a Detector Set

Even though this work follows the implementation details of Forrest et al's negative selection algorithm, there are two implementation details different from Forrest et al (1994, 1997). In the encoding of detectors, each gene of a detector has an alphabet of cardinality 10 with values from '0' to '9' and the allele of this gene indicates the 'cluster number' of corresponding field of profiles. As presented in the next section, the self profile built from the first data set has 33 fields and this number determines the total number of corresponding genes in the detectors. From these 33 fields, the values of 28 fields are continuous and the values of the other 5 fields are discrete. Specifically, the continuous values of 28 fields show a wide range of values. In order to handle this various and broad range of values, an overall range of real values for each field is sorted. Then, this range is discretised into a predefined number of clusters. The lower bound and higher bound of each cluster are determined by ensuring that each cluster contains the same number of records. This modification is necessary in order to save the length of encoded detector.

Furthermore, our implementation of measuring the similarity between a generated detector and a self profile is operated at the phenotype level while Forrest et al's (1994, 1997) is performed at the genotype level. In order to measure the similarity between a given detector and a self, the genotype of a detector is mapped onto a phenotype. The phenotype mapped from the evolved genotype is represented in a form of a detector pattern. As shown in figure 5, a field of a detector phenotype is represented by an interval having a lower bound and a higher bound while a field of a self phenotype is described by one specific value. Hence, the first step of measuring the similarity checks whether a value of each field of a self pattern belongs to a corresponding interval of a detector phenotype. When any value of a self pattern field is not included in its corresponding interval of a detector phenotype, these two fields are not matched. Similarly, for a nominal type of field, two fields match when the values of fields are identical.

The final degree of similarity between a given detector and self example follows the same matching function of Forrest et al (1994), the r-contiguous matching function. Thus, the degree of similarity is measured simply by counting the matching corresponding fields. For instance, if an activation threshold, r, is set as 2, the detector phenotype and self phenotype in the figure 4 will match since two contiguous fields, "Number of Packet" and "Duration", match and this number of contiguous matching fields equals to the activation threshold. However, if this threshold is set as 3, it is regarded that two phenotypes do not match.

Detector Phenotype =
(Number of Packet = [10, 26], Duration = [0.3, 0.85],
Termination = `half closed', … etc)

Self Phenotype =
(Number of Packet = 14, Duration = 0.37,
Termination = `normal', ….etc)

Figure 4: A Detector Phenotype and a Self Phenotype

4 NETWORK TRAFFIC DATA VS NETWORK INTRUSION SIGNATURE

The data chosen for this work was collected for a part of the 'Information Exploration Shootout', which is a project providing several data sets publicly available for exploration, discovery and collecting the results of participants[1]. The set used here was created by capturing TCP packet headers that passed between the intra-LAN and external networks as well as within the intra-LAN. This set consists of five different data sets. The TCP packet headers of the first set were collected when no intrusion occurred and the other four sets were collected when four different intrusions were simulated. These intrusions are: *IP spoofing attack, guessing rlogin or ftp passwords, scanning attack* and *network hopping attack.* The details of attack signatures and attack points of the four different attacks are not available.

The data originally had the fields of network packets capturing tool's format such as time stamp, source IP address, source port, destination IP address, destination port, etc. However, the primitive fields of captured network packets were not enough to build a meaningful profile. Consequently, it was essential to build a data-profiling program to extract more meaningful fields, which can distinguish "normal" and "abnormal". Many researchers have identified the security holes of TCP protocols (Porras and Valdes, 1998; Lee, 1999) and so the fields used by our profiles were selected based on the extensive study of this research. They were usually defined to describe the activities of each single connection.

The automated profile program was developed to extract the connection level information from TCP raw packets and it was used to elicit the meaningful fields of the first data set.

For each TCP connection, the following fields were extracted:

- Connection identifier: each connection is defined by four fields, initiator address, initiator port, receiver address and receiver port. Thus, these four fields are included in the profile first in order to identify each connection.
- Known port vulnerabilities: many network intrusions attack using various types of port vulnerabilities. There are fields to indicate whether an initiator port or a receiver port potentially holds these known vulnerabilities.
- 3-way handshaking: TCP protocol uses 3-way handshaking for a reliable communication. When some network intrusions attack, they often violate the 3-way handshaking rule. Thus, there are fields to check the occurrences of 3-way handshaking errors.
- Traffic intensity: network activities can be observed by measuring the intensity over one connection. For example, number of packets and number of kilobytes

---

[1] Available at http://iris.cs.uml.edu:8080/ network.html.

for one specific connection can describe the normal network activity of that connection.

Thus, in total, self profile fields had 33 different fields for the data set. Even though the network profile fields were extracted to describe a single connection activity, the data used in this research was too limited to apply this initial profile. The limit was that the data was collected for a quite short time, around 15~20 minutes. During this brief period, most different connections were established only once. An insufficient quantity of data was collected to build different connection profiles. Therefore, it was necessary to group different connections into several *meaningful* categories until each category had a *sufficient number* of connections to build a profile. Consequently, a total number of connections for each potential profile category were counted.

First of all, the data was categorised into two different groups: 'inter-connection' and 'intra-connection'. Inter-connection was the group of connections that were established between internal hosts and external hosts, and intra-connection was the group of connections that were established between internal hosts. Furthermore, to preserve anonymity, all internal hosts had a single fake address '2' and any extra information about external hosts and network topology was not provided. Therefore, the profiles according to specific hosts were insufficient. Instead, in this research, only the profiles of specific ports on any hosts were considered.

According to various possible categories, the established connection number of each profile was counted. From each case, apart from a profile class that had more than 100 connections, other profile classes were again grouped into other different classes until each class had more than 100 connections. Finally, 13 different self profiles were built. Their class names and the number of established connections are shown in table 1.

In table1, the class column of inter-connection is shown as: {(a,b),(c,d)}, where 'a' is an internal host, 'b' is a internal port number, 'c' is a external host address and 'd' is an external port number. Hence, the connection is established between (a,b) and (c, d). For the class column of intra-connection, 'a' is an internal host address, 'b' is an internal port number, 'c' is an internal host address and 'd' is an internal, port number. * indicates 'any' host address and 'any' port number. In addition, "well-known" shows the ports in the range 0 to 1023 are trusted ports. These ports are restricted to the superuser: a program must be running as root to listen to a connection. The port numbers of commonly used IP services, such as *ftp, telnet, http*, are fixed and belong to this range. But, many common network services employ an authentication procedure and intruders often use them to sniff passwords. It is worthwhile to monitor these ports separately from the other ports. Therefore, if the number of connections for any profile category, which is based on a specific port on any hosts, is not sufficient, these categories are regrouped into two new classes, a "well-known" port and a "not well-known" port.

Table 1: Self Profiles

Inter-connection	
Class	Number of Connection
{(2, *), (*, 80)}	5292
{(2, *), (*, 53)}	919
{(2, *), (*, 113)}	255
{(2, *), (*, 25)}	192
{(2, *), (*, well-known)}	187
{(2, *), (*, not well-known)}	756
{(2, 53), (*, *)}	940
{(2, 25), (*, *)}	352
{(2, 113), (*, *)}	145
{(2, well-known), (*, *)}	114
{(2, not well-known), (*, *)}	6050
Intra-connection	
{(2, *), (2, well-known)}	190
{(2, *), (2, not well-known)}	189

5 EXPERIMENT OBJECTIVE

Although previous work using a negative selection algorithm for anomaly detection (Forrest et al 1994; Dasgupta 1998; Hofmeyr, 1999) showed promising results, there had been little effort to apply this algorithm on vast amounts of data. One distinctive feature of a network intrusion detection problem is that the size of data, which defines "self" and "non-self", is enormous. In order for this algorithm to be adopted to a network-based IDS, it is important to understand whether this algorithm is capable of generating detectors in a reasonable computing time. In addition, it is essential to examine whether its tuning method, which derives an appropriate number of detectors to gain a good non-self detection rate, works when it is used on the huge size of real network data. Therefore, a series of experiments were performed to investigate these two significant features of the negative selection algorithm.

6 DATA AND PARAMETER SETTING

6.1 SETTING

As presented in section 4, the data used in this work produced thirteen different self profiles. From 13 different self sets, one self set, {(2, *), (*, 25)} in table 1, which has relatively smaller number of examples, 192, was selected for the following experiments. From the total of 192 examples of the selected self profile, 154 examples were used for generating detectors and 38 examples were applied for testing generated detectors. In addition, the detectors were tested on five different test sets. The first four sets were collected when four different intrusions were simulated (as explained in section 4) and the last set was created by generating random strings. These five sets have 273, 190, 1151, 273 and 500 examples respectively.

As described in section 3, the negative selection algorithm used in this paper employed the r-contiguous matching function. For the following experiments, its matching threshold should be defined. In order to define this number, the formulas to approximate the appropriate number of detectors when a false negative error is fixed (D'haeseleer, 1997; Forrest et al, 1994) were used. These formulas are as follows (Forrest et al, 1994) :

$$P_m \approx m^{-r}[(l-r)(m-1)/m+1] \text{ . …… (1)}$$

$$P_m \approx \frac{1}{N_s} \text{ ……………………….. (2)}$$

where,

P_m = the matching probability between a detector string and a randomly chosen self string,

N_s = the number of self strings,

m = the detector genotype alphabet cardinality,

l = the detector genotype string length and

r = the threshold of r-contiguous matching function.

Since N_s, m, l are already known, r can be calculated by using equation (1) and (2). The calculated r was used in the following equation in order to derive an appropriate number of detectors, N_r, and a total number of trials to generate these detectors, N_{r_0}, when the false negative error, P_f, is fixed (Forrest et al, 1994).

$$N_r = \frac{-\ln P_f}{P_m} \text{ …………………(3) and}$$

$$N_{r_0} = \frac{-\ln P_f}{P_m \times (1-P_m)^{N_S}} \text{ …………(4)}$$

The selected self set, {(2, *), (*, 25)}in table 1, was used for calculating N_r and N_{r_0} when P_f is fixed. Table 2 shows calculated N_r and N_{r_0} using (3) and (4) when P_f and r have various values.

(D'haeseleer, 1997; Forrest, et al, 1994) showed that the larger matching threshold drives the creation of less general detectors and thus it requires a larger number of detectors but a smaller number of detector generation retrials. This is because less general detectors are easier to avoid the matching a self profile. N_r and N_{r_0} in table 2 follows the same tendency.

Table 2 Number of required detectors, N_r, and number of trials to generate required number of detectors, N_{r0} when false negative error, P_f, and the threshold r of r-contiguous matching function are given. These numbers are calculated when a self string length, $l = 33$, an alphabet cardinality, $m = 10$ and the number of self strings, $N_S = 192$.

P_f	$r = 3$		$r = 4$	
	N_r	N_{r_0}	N_r	N_{r_0}
0.2	51	21953	535	955
0.1	73	31382	766	1366
0.05	95	40829	997	1777
0.01	146	62765	1532	2733

Even though this formula is clearly useful to predict the appropriate number of detectors and its generation number, its predicted number showed how infeasible this approach is when it is applied on a more complicated but more realistic search space. For instance, when the expected false negative error rate is fixed as 20%, its predicted detector generation trial number is 51 and the appropriate number of generated detectors is 21935 for the matching threshold is 3. Similarly, when we define the matching threshold as 4, it predicted 535 for the former and 955 for the latter. In addition, it was observed that when we fixed the matching threshold number as four and ran the system, the system could not manage to generate any single valid detector after one day. None of these cases seem to provide any feasible test case in terms of computing time. This results certainly did not follow the predicted detector generation trial number.

Thus, for the following experiments, we generated valid detectors by setting a matching threshold number that allowed a system to generate a valid detector in a reasonable time. It was observed that the average time of single successful detector generation took about 70sec CPU time and the average number of trials to generate a valid detector was 2~3 when a matching threshold was nine. These results were gained after running the negative selection algorithm for preliminary experiments. This number is used as the matching threshold for the following experiments. The details of these experiment results are described in the next section.

7 EXPERIMENT RESULT

Five different sets of detectors were generated after the AIS with the negative selection was run five times. Even though the matching threshold, 9, gave reasonable computing time to generate a valid detector, it requires a large number of detectors to gain a good non-self

Table 3 The mean and variance values of intrusion and self detection rates when detector set size varies
The means values are followed by the variances in the parentheses.

Num. Of Detectors	Intrusion1 (%)	Intrusion 2 (%)	Intrusion 3 (%)	Intrusion 4 (%)	Intrusion 5 (%)	Test Self Set (%)
100	9.45(2.11)	10.11(8.50)	11.14(9.44)	10.62(4.03)	0.48(0.012)	7.89(17.31)
200	11.72(5.37)	11.58(13.71)	12.98(11.52)	12.89(10.43)	0.88(0.092)	9.47(36.70)
300	12.53(4.25)	11.89(13.24)	13.73(9.48)	13.63(9.15)	1(0.12)	10(29.08)
400	13.33(2.79)	12.32(11.30)	14.58(10.18)	14.36(6.87)	1.28(0.112)	10.53(31.16)
500	13.55(3.15)	12.74(13.63)	14.89(10.40)	14.51(7.35)	1.36(0.068)	11.05(25.62)
600	13.77(3.80)	13.16(11.91)	15.07(10.24)	14.65(8.12)	1.68(0.412)	11.58(29.78)
700	13.77(3.80)	13.16(11.91)	15.26(9.46)	14.65(8.12)	2.04(0.388)	11.58(29.78)
800	13.92(4.09)	13.26(11.27)	15.45(10.09)	14.80(8.22)	2.04(0.388)	11.58(29.78)
900	14.14(4.13)	13.47(10.47)	15.67(9.69)	15.02(8.52)	2.08(0.352)	12.63(46.40)
1000	**14.21(4.32)**	**14.08(11.52)**	**15.90(8.71)**	**15.09(8.68)**	**2.28(0.312)**	**12.63(46.40)**

detection rate. After taking into account practically reasonable time to generate a whole data set, up to 1000 valid detectors were generated per run. All experiments were run on a PC with AMD K6-2 400Mhz processor and 128M RAM.

Table 4 Time is an avarage time of single detector generation and Trial is an average trial number to generate a single detector. The average values are followed by the standard deviations in parentheses.

System Run	**Time (Sec)**	**Detector Generation Trial**
1	58.71(26.85)	2.80(2.16)
2	67.29(28.88)	2.21(1.65)
3	73.75(33.72)	2.81(2.22)
4	78.48(39.86)	3.12(2.69)
5	69.64(26.62)	2.72(2.07)
Average	**71.81(32.75)**	**2.63(2.14)**

Table 3 shows the average time of single successful detector generation and the average number of trials to generate a valid detector. Compared to the result when the matching threshold is four, which did not generate any single detector after 24 hours, these results certainly look more applicable. We monitored five different non-self sets and one previously unseen self sets after every 100 detector generation and the monitor results of five different runs are shown in table 4. The overall non-self detection rate was very poor: less than 16%. In particular, the non-self detection rate for the last intrusion set, which was artificially generated by random strings, is extremely low and its maximum average non-self detection rate reaches only 2.28%. In addition, its average false positive detection rate, which is self detection rate by a detector set, shows 12.63% and this rate is not hugely different from the other four average non-self detection rates except intrusion 5. This implies that the collected self and non-self sets perhaps have some overlapping patterns because they showed quite similar detection rates. Thus generated detector sets completely failed to distinguish the hidden self and non-self patterns.

These poor results were anticipated. This is because the matching threshold was set in order to obtain a reasonable detector generation time. If, for example, we wanted a more usable 80% non-self detection rate, 643775165 detectors would be required (this number is also obtained from equation 3). The largest size of a generated detector set, 1000, was much smaller than this number and this caused such poor results. In addition, each run already took about 20 hours[2] to generate 1000 detectors. If we wished to generate 643775165 detectors, it would require 12517850.4 hours, or about 1,429 years on the same computer. According to Moore's Law, the processing speed of computers doubles every 18 months. We would have to wait around 35 years before the average processing speed of computers became fast enough to generate these detectors in an hour - and this is for just 15~20 minutes of a tiny subset of the network traffic data.

8 ANALYSIS

In contrast to the promising results shown in Hofmeyr's negative selection algorithm for network intrusion detection (Hofmeyr, 1999; Hofmeyr and Forrest, 2000), the results of these experiments raise doubt whether this algorithm should be used for network intrusion detection. In order to answer this question, the negative selection algorithm for network intrusion detection is analysed in detail.

The main problem of the negative selection algorithm is a severe scaling problem. Unlike previous work using

---

[2] Since it took, on average, 72 seconds to generate each detector, 72000 seconds were needed to produce 1000 detectors. 72000 seconds are 20 hours.

the negative selection algorithm for anomaly detection, here we apply a much larger "self" set to the negative selection algorithm. The definition of larger "self" set was essential to cover diverse types of network intrusions. For instance, (Hofmeyr 1999; Hofmeyr and Forrest, 2000) defines "self" as a set of normal pairwise connections between computers. These include connections between two computers in the LAN and between one computer in the LAN and external computers. The connection between computers is defined by "data-path-triple": (the source IP address, the destination IP address, the port called for this connection). This self definition is chosen based on the work by (Heberlein, et al, 1990). However, as other IDS literature pointed out (Lee, 1999), this self definition is very limited in order to detect various types of network intrusions and it will certainly be impossible to detect some intrusions that occur within a single normal connection such as unauthorised access from a remote machine.

However, as observed in section 4, when the self definition widens, a binary string to encode a detector lengthens. As the result of long length of binary detectors, an appropriate number of detectors to gain an acceptable false negative error becomes huge and thus requires an unacceptably long computation time. Our previous experiment results clearly show this problem.

It should be noted that Hofmeyr (1999) developed a refined theory and multiple secondary representations and these help to reduce the number of trials to generate detectors on structured self as much as three orders magnitude less. These methods made the distribution of a self set clump and it resulted in the reduction of the number of detector generation trials. However, the refined theory and secondary representations add extra space and computing time. More importantly, all of the suggested secondary representations, such as pure permutation, imperfect hashing and substring hashing, are matching rules which check matching only on genotypes. Unfortunately, matching rules that operate only at the genotype level have a weakness to be applied for a network intrusion detection problem. This deficiency can be explained by unravelling the problem of r-contiguous matching function.

We used the r-contiguous rule to check the match between a given detector and antigen. The main purpose of using it was in order to employ the formula to approximate an appropriate number of detectors to gain a certain non-self detection rate. However, the r-contiguous matching rule is too simple to determine the matching between rather complicated and high-dimensional patterns. It has been already known that most rules to represent intrusion signatures describe correlation among significant network connection events and temporal co-occurrences of events (Lee, 1999; Porras, 1998). Since the r-contiguous bit matching only measures the contiguous bits of genotypes of given two strings, it is hard to guarantee that the r-contiguous bit matching can catch this kind of correlation from given self and non-self patterns. The wider range of self definition shown in section 4 is also suggested in order to extract this type of correlation from given self and non-self network traffic examples.

But, if any new matching function is employed, D'haeseleer's (1997) formula is no longer valid. There is no way to tune the right number of detectors for negative selection. Therefore, this difficulty may force the negative selection algorithm to adopt an arbitrary number of detectors and this may cause an unexpectedly low detection accuracy or inefficient computation by generating more than sufficient number of detectors. In addition, D'haeseleer's (1997) new detector generation algorithms using a linear-time algorithm and a greedy algorithm that guarantees a liner time of detector generation is also not applicable when a different matching function is used.

In summary, it is necessary to use a more sophisticated matching function to determine the degree of correlation among significant network connection events and temporal co-occurrences of events. This requires deriving a new way to tune an appropriate number of detectors, which can be used for more sophisticated matching function.

These drawbacks of the negative selection algorithm made the AIS struggle to monitor vast amount of a network self set despite its other important features[3]. Consequently, the initial results of our experiments motivated us to re-define the role of negative selection stage within an overall network-based IDS and design a more applicable negative selection algorithm, which follows a newly defined role. As much of the other immunology literature (Tizard, 1995) addresses that the antigen detection powers of human antibodies rise from the evolution of antibodies via a clonal selection stage. While the negative selection algorithm allows the AIS to be an invaluable anomaly detector, its infeasibility to be applied on a real network environment is caused from allocating a rather overambitious task to it. To be more precise, the job of a negative selection stage should be restricted to tackle a more modest task that is closer to the role of negative selection of human immune system. That is simply filtering the harmful antibodies rather than generating competent ones. This view has been corroborated by further work (Kim and Bentley, 2001) which has recently shown how succesful the use of clonal selection with a negative selection operator can be for this type of problem.

---

[3] Hofmeyr and Forrest (2000)'s final system employs some other extensions to support the operation of AIS under a real network environment. Among them, affinity maturation and memory cell generation follow the clonal selection concept and these provide a kind of evolution of a detector set distributed on monitored hosts. However, it still uses only the negative selection algorithm to generate an initial detector set. Even though it may conform to human immune systems more closely, this approach could require excessive computation time to generate the initial detector set, if a broader definition of self is used. In addition, the usefulness of initial detectors is not proven before they are distributed to other hosts. This may also cause a waste of other computing resources.

9 CONCLUSIONS

This paper has investigated the role of negative selection in an artificial immune system (AIS) for network intrusion detection. The negative selection stage within our AIS was implemented following the algorithm created by Forrest et al (1994; 1997) and applied to real network data. The experiments showed the infeasibility of this algorithm for this application: the computation time needed to generate a sufficient number of detectors is completely impractical.

This result directs this research to re-define the role of negative selection algorithm within our overall artificial immune system framework. Current work is now investigating the intrusion detection mechanism of the clonal selection stage. A new understanding of the task of the clonal selection stage has now resulted in the development of a more appropriate use for negative selection as an operator within a novel clonal selection algorithm (Kim and Bentley, 2001).

References

D'haeseleer, P, (1997), "A Distributed Approach to Anomaly Detection", *ACM Transactions on Information System Security*. http://www.cs.unm.edu/~patrik/

Dasgupta, D., (1998), "An Overview of Artificial Immune Systems and Their Applications", In Dasgupta, D. (editor). *Artificial Immune Systems and Their Applications*, Berlin: Springer-Verlag, pp.3-21.

Forrest, S. et al, (1994) "Self-Nonself Discrimination in a Computer", *Proceeding of 1994 IEEE Symposium on Research in Security and Privacy,* Los Alamos, CA: IEEE Computer Society Press.

Forrest, S., et al, (1997), "Computer Immunology", *Communications of the ACM*, 40(10), 88-96.

Forrest, S and Hofmeyr, S. (2000) "Immunology as Information Processing", in *Design Principles for Immune Systems and Other Distributed Autonomous Systems*, (Ed) Segal, L.A. & Cohen, I. R. eds., Oxford University Press.

Heberlein, L. T., et al. (1990), "A Network Security Monitor", *Proceeding of 1990 Symposium on Research in Security and Privacy*, Oakland, CA, pp.296-304, May, 1990.

Hofmeyr, S., (1999) *An Immunological Model of Distributed Detection and Its Application to Computer Security*, Phd Thesis, Dept of Computer Science, University of New Mexico.

Hofmeyr, S., and Forrest, S., (2000), "Architecture for an Artificial Immune System", *Evolutionary Computation*, vol.7, No.1, pp.45-68.

Kephart, J. O., et al, (1995), "Biologically Inspired Defenses Against Computer Viruses", *the Proceeding of 14th Intl. Joint Conf. on Artificial Intelligence, Montreal, August*, pp.985-996.

Kim, J. and Bentley, P. (1999a), "The Human Immune System and Network Intrusion Detection", *7th European Conference on Intelligent Techniques and Soft Computing (EUFIT '99), Aachen, Germany*.

Kim, J. and Bentley, P. (1999b), "The Artificial Immune Model for Network Intrusion Detection, *7th European Conference on Intelligent Techniques and Soft Computing* (*EUFIT'99), Aachen, Germany*.

Kim, J. and Bentley, P. (2000), "Negative Selection within an Artificial Immune System for Network Intrusion Detection", *the 14th Annual Fall Symposium of the Korean Information Processing Society, Seoul, Korea.*

Kim, J. and Bentley, P. (2001), The Artificial Immune System for Network Intrusion Detection: An Investigation of Clonal Selection with a Negative Selection Operator. Submitted to CEC2001, the Congress on Evolutionary Computation, Seoul, Korea, May 27-30, 2001.

Lee, W., (2000) *A Data Mining Framework for Constructing Features and Models for Intrusion Detection Systems*, PhD Thesis, Dept of Computer Science, Columbia University.

Mykerjee, B., et al, (1994), "Network Intrusion Detection", *IEEE Network,* Vol.8, No.3, pp.26-41.

Paul, W. E., (1993), "The Immune System: An Introduction", in *Fundamental Immunology* 3rd Ed., W. E. Paul (Ed), Raven Press Ltd.

Porras, P. A., (1992), *STAT: A State Transition Analysis Tool for Intrusion Detection*, MSc Thesis, Department of Computer Science, University of California Santa Babara.

Porras, P. A. and Valdes, A., (1998), "Live Traffic Analysis of TCP/IP Gateways", *Proceeding of ISOC Symposium of Network and Distributed System Security.* http://www.csl.sri.com/emerald/downloads.html

Tizard, I. R., (1995), *Immunology: Introduction*, 4th Ed, Saunders College Publishing.

A RISC Processor for High-Speed Execution of Genetic Algorithms

Shinya Koizumi† **Shin'ichi Wakabayashi**† **Tetsushi Koide**‡
Kazunari Fujiwara† **Norimichi Imura**‡
† Graduate School of Engineering ‡ Research Center for Nanodevices and Systems
Hiroshima University
4-1 Kagamiyama 1 chome, Higashi-Hiroshima 739-8527, Japan E-Mail: wakaba@computer.org

Abstract

This paper proposes a new RISC processor for high speed execution of genetic algorithms (GAs). The proposed RISC processor is designed based on the DLX instruction set, and a set of new instructions, which are effective to high-speed execution of GAs, are added. Since a GA is implemented as software on the proposed processor, any type of GA can be realized. Using the instruction set of the proposed processor, more than 90 % reduction of the number of clocks to execute GA operators such as 2-point crossover can be achieved. The processor has been designed with the Verilog Hardware Description Language to be implemented as a VLSI chip with a $0.35\mu m$ standard cell technology.

1 Introduction

Genetic algorithms (GAs) were invented by John Holland in 1970s as search algorithms based on the mechanics of natural selection and natural genetics [1]. GAs are known to be robust and effective search algorithms for large-scale, complex optimization problems, and many results on applications of GAs in various areas of engineering have been reported.

However, the major drawback of GAs is their slow execution speed when they were implemented and executed on a conventional computer. To overcome this drawback, several approaches have been reported. Several authors have proposed parallel GAs, which may be classified into three classes, namely, massively parallel GAs, parallel island model GAs, and parallel hybrid GAs [6]. In general, parallel processing of GAs achieves a good performance. In addition, since parallel GAs are usually implemented as software, it is easy to implement any kind of GAs. The main disadvantage of parallel GAs is that it often requires a large amount of computer resources.

Another common approach to high-speed execution of a GA is to implement it as hardware. For example, Scott *et al.* proposed a hardware-based GA, which was an implementation of a steady-state GA using field programmable gate arrays (FPGAs) [7]. Yoshida *et al.* proposed a VLSI for GA, which realizes coarse-grained parallel processing of GA execution [12]. We have also proposed an LSI implementation of adaptive GAs [11]. Implementation of GAs as hardware generally achieves a very good performance. The major disadvantage of this approach, however, is the difficulty to realize the programmability. In fact, in most of hardware GAs, GA operators such as crossover and mutation were fixed in advance. To achieve a good performance of GA execution, appropriate GA operators and GA parameters should be selected and tuned for given problems.

In this paper, we propose a new RISC processor, whose instruction set is tailored to the efficient execution of GAs [4]. The proposed RISC processor is designed based on the DLX instruction set [3], and we add several special instructions, which are effective to high-speed execution of GAs. Newly added instructions can be classified into three groups. The first group consists of bit-oriented instructions, because GA operators such as crossover often require bit-oriented operations. The second group consists of instructions concerning with random numbers. Since a GA frequently uses random numbers, the computation time for generating a pseudo-random number has a heavy effect on the performance of GA execution. The proposed processor has a pseudo-random number generation circuit, and in each clock cycle, a pseudo-random number is generated. The processor has several instructions using random numbers, which are very effective to shorten the computation time of selection, crossover, and mutation. Finally, the third group of instructions added to the proposed processor consists of SIMD instructions, which are mainly used to implement a crossover operation.

Since a GA is implemented as software on the proposed processor, any kind of GA can be realized. Preliminary experiments show that, using the instruction set of the proposed processor, more than 90 % reduction of the number of clocks to execute GA operators such as 2-point crossover can be achieved. The processor has been designed with the Verilog Hardware Description Language to be implemented as a VLSI chip with a $0.35\mu m$ standard cell technology.

This paper is organized as follows. Section 2 presents the architecture and the instruction set of the proposed RISC processor. Section 3 discusses the VLSI design and the performance estimation on GA execution of the proposed processor. Finally, in Section 4, we conclude this paper.

2 Processor Architecture

2.1 Genetic Algorithms

Genetic Algorithm (GA) is known to be a robust search algorithm, which deals with the individuals (chromosomes) of candidate solutions (population) encoded in the problem independent representation [1]. During the genetic process, new candidate solutions are composed by using the genetic operators such as crossover and mutation. When solving the specific application by a GA, it is often necessary to use complex representation of individual and genetic operators tailored to the problem to search solutions efficiently. Thus, programmability is indispensable when developing general-purpose GA systems to solve optimization problems.

Programmability could be realized with either software or programmable logic devices. For software implementation of GAs, any kind of GAs could be programmed, but the performance is restricted due to properties of GAs such as frequent usage of bit operations and random number generation, which could not be efficiently performed on the ordinary processors. To realize programmability on the hardware-based GAs, using programmable logic devices such as field programmable logic arrays (FPGAs) to construct a GA engine is one solution, and several authors have reported the implementation of GAs on general-purpose FPGA-based systems [2]. The main disadvantage of this approach is its large design time. Designing a GA on an FPGA-based machine is more difficult than developing a GA software on a general-purpose processor, since hardware design includes not only coding with the hardware description language (HDL), but also placement and routing under timing and hardware resource constraints. In addition, presently, the level of HDL coding is generally lower than the level of programming with a high level programming language such as C, although there has been much effort to improve the productivity of HDL-based LSI design.

In this paper, to realize full programmability with high performance of GA execution, we propose a new RISC processor. Since any type of GA can be realized as software on the proposed processor, full programmability is assured. To achieve efficient execution of a GA, a special instruction set is devised.

2.2 Properties of a GA

To develop a processor tailored for executing a GA efficiently, first, we summarize the general properties of GAs. That is, we discuss what types of operations are frequently used in executing a GA.

1. Bit-oriented operations

   A GA generally requires not only normal word-oriented operations like AND, ADD, etc, but also a number of bit-oriented operations, that is, operations are applied not only to a whole memory word, but also to a part of a word. For example, when a chromosome consists of n bits, and a 2-point crossover is executed, each of parent chromosomes is divided into three parts, and the middle ones are exchanged. Cut points are normally specified as bit positions.

2. Random numbers

   A GA frequently uses random numbers in various stages of the algorithm execution such as crossover, mutation, and selection. Since pure random number generation is hard to realize, pseudo-random numbers are generally used, which are generated with some pseudo-random number generators. Meysenburg and Foster showed that the quality of pseudo-random number generation has little effect on the performance of a simple genetic algorithm [5]. This means that it is not necessary to adopt high-quality, complex, time-consuming random number generating methods for GAs.

3. SIMD operations

   A set of chromosomes forms a population, and it is often the case that the same operation is applied to all chromosomes in a population. Those operations could be regarded as SIMD (single instruction multiple data) operations.

When a GA is implemented as software on a general-purpose processor such as Pentium-III, UltraSPARC, etc., due to the properties listed above, we would face some difficulties to implement a GA efficiently. First, since general-purpose processors were normally designed for word-oriented operations, it is inefficient to realize bit-oriented operations.

Second, in the usual software implementation of a GA, a random number is generated with some pseudo-random number generation algorithm. A pseudo-random number generation algorithm is implemented with, at least, 10 instructions, and normally, more than 50 instructions. Since a GA frequently requires random numbers in its execution, we cannot neglect the performance overhead of random number generation.

Third, although some GA operations such as crossover could be realized as SIMD instructions, a general-purpose processor does not support any SIMD instructions.

2.3 Instruction Set

From the observations described in the previous subsection, an instruction set of the proposed processor was designed so that a GA can be implemented as software to realize the high-speed execution of it. Since instructions which ordinary general-purpose processors supported such as arithmetic and logical instructions, load/store instructions, and so on, are also required in the proposed processor, we adopt the DLX architecture, proposed in [3], as a base architecture, and all DLX instructions except floating point arithmetic ones are also supported in the proposed processor. Floating point arithmetic instructions will require a large amount of hardware resources, and since we have a restriction that the processor will be implemented on a CMOS standard cell with $4.9 \times 4.9mm^2$ chip area, we have to exclude them. In addition to the original DLX instruction set, we add a set of 27 new instructions tailored to execute a GA efficiently. Those instructions are classified into three categories listed below. Table 1 summarizes those new instructions. Due to the lack of space, only a few instructions in each categories are actually explained.

1. Bit-oriented instructions

In this category of instructions, only a specified part of a word is treated as an operand. In this category, there are 16 instructions consisting of arithmetic, logical, and move instructions. Examples are shown below:

MOVB [rs][rd][ra][rb][rc]

This instruction moves some bits in the source register to the destination register. That is,

$$\text{rs[ra:ra+rc-1]} \leftarrow \text{rd[rb:rb+rc-1]}.$$

ANDB [rs][rd][ra][rb][rc]

This instruction performs AND for the specified bits in the source and destination registers. That is,

$$\text{rs[ra:ra+rc-1]} \leftarrow \text{rs[ra:ra+rc-1] and rd[ra:ra+rc-1]}.$$

2. RNG related instructions.

This category includes 5 instructions related with the random number generation (RNG). As explained in the next subsection, the proposed processor has a pseudo-random number generator, which generates a 96-bit pseudo-random number in each clock cycle.

RSTRNG [ra][rb][rc]

This instruction initializes the random number generator by setting a number stored in registers ra, rb, and rc as its initial value.

SRNI [rs][imd]

This instruction sets a random number to register rs, whose range is [0 . . . imd-1].

3. SIMD instructions

This category of instructions contains 4 SIMD instructions.

RR [rs1][rs2][imd]

This instruction rotates registers rs1 and rs2 simultaneously in the right direction with imd bits.

The original DLX architecture has three instruction formats, denoted I-type, R-type, and J-type. To implement the new instructions described above, we add three new instruction formats, called I’-type, R’-type, and R”-type, shown in Figure 1.

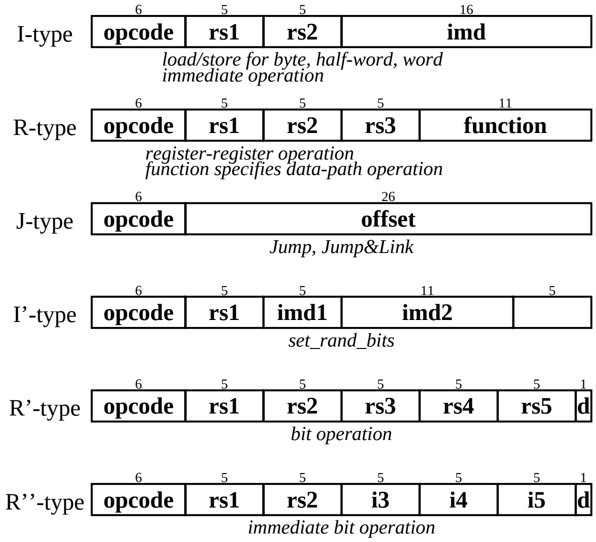


Figure 1: Instruction formats.

Table 1: DLX-GA new instruction set.

Mnemonic	Explanation
Bit-oriented instructions	**Operate on only a specified part of a word**
XCB, XCBI	Exchange bits, exchange bits immediate
CMPB, CMPBI	Compare bits, compare bits immediate; the least significant 1 bit specifies the destination register among R29 and R30
MOVB, MOVBI	Move bits, move bits immediate
XTRCB, XTRCBI	Extract bits, extract bits immediate
NOTB, NOTBI	Not bits, not bits immediate
ANDB, ANDBI	And bits, and bits immediate
ORB, ORBI	Or bits, or bits immediate
XORB, XORBI	Exclusive or bits, exclusive or bits immediate
RNG related instructions	**Generate various random numbers using the random number generator (RNG)**
RSTRNG	Reset the random number generator
SRN, SRNI	Set random number, set random number immediate
SRB	Set random bit
SRBS	Set random bits
SIMD instructions	**Perform the same operation on two operands**
RL	Rotate in the left direction
RR	Rotate in the right direction
XCRL	Exchange and rotate in the left direction
XCRR	Exchange and rotate in the right direction
Interrupt related instructions	**Control the interrupt**
RFI	Return from interrupt
SIEF	Set interrupt enable flag

2.4 GA-oriented RISC Architecture

Figure 2 shows the overall architecture of the proposed RISC processor, called DLX-GA. Specifications of the DLX-GA processor are given in Table 2.

DLX-GA is a 32-bit RISC processor. As mentioned, DLX-GA is designed based on the DLX processor [3], and hence most of the characteristics of DLX-GA is the same as DLX. DLX-GA has a uniform 32-bit instruction format, and a load/store architecture. We adopt the Harvard architecture, and the instruction memory bus and the data memory bus are separated. There is an on-chip instruction cache, but no data cache due to the restriction of hardware resources. The register file implements general-purpose registers with 32 words of 32 bits.

The datapath of DLX-GA is embedded in a 6-stage pipeline consisting of an instruction fetch stage, an instruction decode stage, two execution stages, a memory access stage, and a write-back stage. To implement SIMD instructions, two ALUs are equipped. Furthermore, a pseudo-random number generator (RNG) based on a cellular automaton based algorithm [8] is included in the datapath. This RNG generates a 96-bit pseudo random number in each clock cycle. The generated number is them multiplied with a constant given by the instruction to produce a random number within the specified range. As noted before, the quality of random number generations will merely effect on the GA performance [5], and hence a pseudo-random number generator (RNG) based on cellular automaton is sufficient to generate random numbers for GA execution. In fact, we have compared extensively our random number generator with the standard pseudo-random number generation algorithm written in C, and there was no evidence that the latter was statistically better than the former when each was used in a GA implementation.

Interrupt handling is also supported in the processor. When accepting an interrupt from outside of the processor, the processor starts an interrupt handling routine. With the interrupt mechanism, it is possible to construct a parallel GA system by connecting several DLX-GA processors.

3 VLSI Implementation and Performance Estimation

3.1 VLSI Implementation

The DLX-GA processor will be fabricated as a standard cell LSI with a 0.35μm 3 metal layer CMOS technology, which will be fabricated by Rohm Corporation by July 2001.

The DLX-GA chip was firstly designed in the hardware description language Verilog-HDL [10] on register transfer level (RTL). Each pipeline stage of the DLX-GA was designed as a module. In addition to these modules, there are

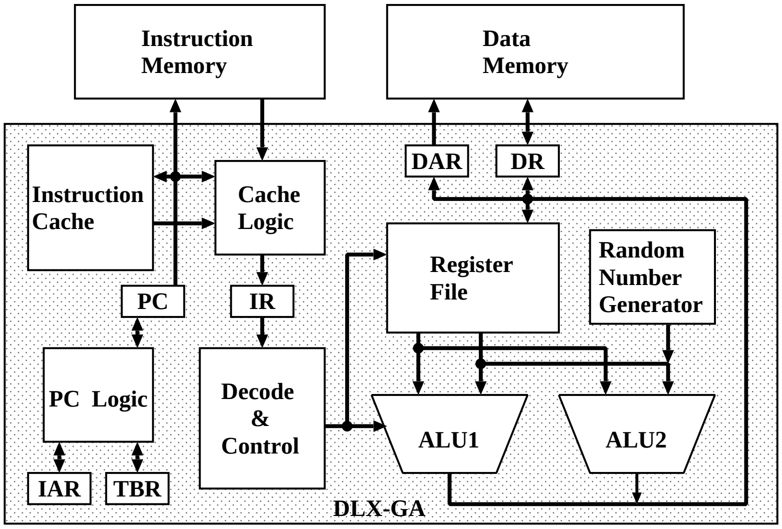


Figure 2: DLX-GA architecture.

Table 2: Specifications of DLX-GA.

Instruction memory address bus	15 bits
Instruction memory data bus	32 bits
Data memory address bus	22 bits
Data memory data bus	32 bits
Instruction cache	direct mapping, 512 lines
Register file	32 bits × 32 words
Interrupt	2 level
Clock	100 MHz

some submodules and some sub-submodules. For instance, the ID stage includes a register file and a forwarding controller as its internal submodules, and the EX stage has two ALUs and the RNG as its internal submodules. Each ALU has a bit-oriented operation module, a shift operation module and an ordinary arithmetic and logic operation module as its internal submodules.

After HDL coding of modules, each module of the DLX-GA was simulated separately on the functional level with Cadence's Verilog-XL simulator. After this initial verification, several modules were combined to a larger unit and simulated again. Finally, simulation of the complete DLX-GA was carried out. Several test programs were developed with a DLX-GA assembler which we have developed, and simulated on the whole HDL description of the DLX-GA.

The verified Verilog description on register transfer level was fed to Synopsys' Design Compiler to synthesize a gate level circuit. We use a CMOS 0.35μm standard cell library for the synthetic cells and make use of 5 8-bit 512-word memory macro cells to compose the 40-bit 512-word instruction cache. To obtain good synthesis results, modules containing more than 20,000 gates were synthesized separately. Finally, all of the hierarchy of the HDL description of the DLX-GA were removed, and the whole circuit was synthesized as a flat structure.

Table 3: Synthesis result of DLX-GA.

Number of cells	20,401
Number of nets	20,507
Number of FFs	2,074
Combinational area	32,708
Noncombinational area	14,527
Total area	47,512
Critical path [ns]	9.25

Table 3 shows the synthesis result of the DLX-GA chip except for the memory macro cells. Table 4 shows the synthesis result of each pipeline stage. Here, EX1 stage and EX2 stage were synthesized together. A cell in these tables means a basic gate such as AND gate, OR gate, a compound gate such as AND-OR-INV, and a latch such as D-FF. And a unit area is corresponding to a 2-input NAND

Table 4: Synthesis result of each pipeline stage.

Pipeline stage	IF	ID	EX1+EX2	MEM	WB
Number of cells	717	6,659	12,680	297	133
Number of nets	895	6,873	13,246	446	206
Number of FFs	164	1291	525	117	64
Combinational area	894	12,894	22,026	283	93
Noncombinational area	1,164	9,164	3,731	856	469
Total area	2,070	22,168	25,931	1,144	565
Critical path [ns]	8.59	4.13	9.05	8.97	1.05

gate. From Table 3, the whole circuit except the memory macro cells consists of around 20,000 standard cells, which is roughly equivalent to 48,000 2-input NAND gates, and is 1.6 times larger than the original DLX processor. From Table 4, the critical path of EX1+EX2 is 9.05ns and is roughly the same as that of IF and MEM. This is due to the division of the EX stage of the original DLX pipeline into two stages. Therefore, the two EX stages (EX1+EX2), which perform several complex operations (e.g., the bit-oriented operations), does not become a bottleneck in the 6-stage pipeline of the DLX-GA.

The layout of the DLX-GA was carried out with Avant!'s ApolloXO layout tool. Starting from the floorplan, the placement and routing were controlled by the timing constraints (e.g., target clock frequency). The placement was executed by employing ApolloXO's static timing analysis engine. After the placement, clock tree synthesis was carried out.

The chip image of the DLX-GA is shown in Figure 3. The chip size is $4.9 \times 4.9\ mm^2$. Standard cell logic was placed in the middle of the chip and memory macro cells were placed on the boundary of the chip because it was prohibited to route signal wires over memory macro cells.

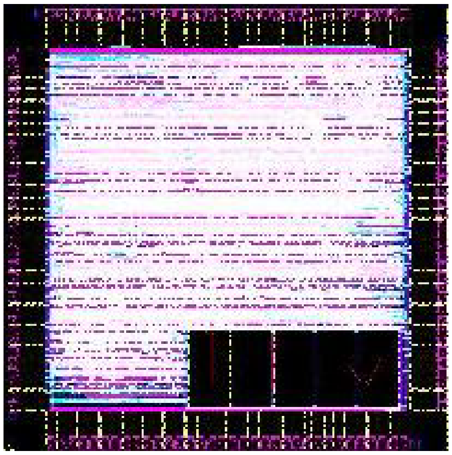


Figure 3: Chip image.

After the placement and routing, using back-annotated delay data in the Standard Delay Format (SDF), static timing analysis was carried out on Synopsys' Design Compiler. Similarly, using SDF, post-layout simulation was also carried out. These verifications proved that the timing was met to the given constraint.

3.2 Performance Evaluation

In this subsection, we show how much effective the proposed instruction set of the DLX-GA is. We wrote two typical GA functions with the C language, one of which is the typical 2-point crossover, and the other is the swap mutation for the traveling salesman problem. Then, two source programs were compiled with the GNU C Compiler (gcc) to obtain the assembly codes of the DLX processor. For the proposed DLX-GA processor, since no compiler was currently available, we got the assembly codes from the DLX assembly codes by rewriting the assembly codes by hand, and translating them to the machine codes by using the DLX-GA assembler. Then, we compare the number of clocks to execute the respective programs. Table 5 shows the results. From the table, the proposed processor achieved more than 90% reduction of the number of clocks. Figure 4 shows the assembly codes of two processors for 2-point crossover. Note that, even the codes for the DLX processor were produced by the compiler, we have checked that it was very hard to shorten the codes by hand.

Table 5: Comparison of DLX with DLX-GA.

processor	2-point crossover	swap mutation
DLX	138	75
DLX-GA	11	7

Next, we observed the percentage of CPU time of selection, crossover, mutation, evaluation, and random number generation in the total CPU time when executing the simple genetic algorithm (SGA) [9]. Table 6 shows the simulation condition and Table 7 shows the result. In this table, problem 1 is the maximization of function x^{10} and prob-

```
L2_LF0:
    lw     r1,-32(r30)
    lw     r2,-36(r30)
    slt    r1,r1,r2
    bnez   r1,L5_LF0
    j      L3_LF0
L5_LF0:
    lw     r1,-32(r30)
    snei   r2,r1,#0
    beqz   r2,L6_LF0
    lw     r1,-20(r30)
    slli   r2,r1,#0x4
    sw     -20(r30),r2
L6_LF0:
    lw     r1,-20(r30)
    addi   r2,r1,#15
    sw     -20(r30),r2
L4_LF0:
    lw     r2,-32(r30)
    addi   r1,r2,#1
    add    r2,r0,r1
    sw     -32(r30),r2
    j      L2_LF0
L3_LF0:
    nop
    lw     r1,-36(r30)
    sw     -32(r30),r1
L7_LF0:
    lw     r1,-32(r30)
    lw     r2,-40(r30)
    slt    r1,r1,r2
    bnez   r1,L10_LF0
    j      L8_LF0
L10_LF0:
    lw     r1,-20(r30)
    slli   r2,r1,#0x1
    sw     -20(r30),r2
L9_LF0:
    lw     r2,-32(r30)
    addi   r1,r2,#1
    add    r2,r0,r1
    sw     -32(r30),r2
    j      L7_LF0
L8_LF0:
    nop
    lw     r1,-40(r30)
    sw     -32(r30),r1
L11_LF0:
    lw     r1,-32(r30)
    lw     r2,-44(r30)
    slt    r1,r1,r2
    bnez   r1,L14_LF0
    j      L12_LF0
L14_LF0:
    lw     r1,-20(r30)
    slli   r2,r1,#0x4
    sw     -20(r30),r2
    lw     r1,-20(r30)
    addi   r2,r1,#15
    sw     -20(r30),r2
L13_LF0:
    lw     r2,-32(r30)
    addi   r1,r2,#1
    add    r2,r0,r1
    sw     -32(r30),r2
    j      L11_LF0
L12_LF0:
    lw     r1,-12(r30)
    lw     r2,-20(r30)
    and    r1,r1,r2
    lw     r3,-20(r30)
    sub    r2,r0,r3
    subi   r2,r2,#1
    lw     r3,-16(r30)
    and    r2,r2,r3
    or     r1,r1,r2
    sw     -24(r30),r1
    lw     r1,-16(r30)
    lw     r2,-20(r30)
    and    r1,r1,r2
    lw     r3,-20(r30)
    sub    r2,r0,r3
    subi   r2,r2,#1
    lw     r3,-12(r30)
    and    r2,r2,r3
    or     r1,r1,r2
    sw     -28(r30),r1
```

(a) 2-point crossover with the DLX instruction set.

```
lw     r3,-16(r30)
lw     r4,-20(r30)
sub    r4,r4,r3
lw     r5,-24(r30)
multi  r3,r3,r5
multi  r4,r4,r5
lw     r1,-8(r30)
lw     r2,-12(r30)
move_bits  r1,r2,r3,r3,r4
sw     -8(r30),r1
sw     -12(r30),r2
```

(b) 2-point crossover with the new instruction set.

Figure 4: Assembly codes of 2-point crossover.

lem 2 is the maximization of 32-dimensional vector length. From Table 7, the CPU time of the random number generation (rng) occupied more than 50% in the total CPU time. Normally, on a general-purpose processor, a pseudo-random number generation algorithm is implemented with more than 50 instructions, but on the DLX-GA processor, a pseudo-random number can be generated with only 1 instruction. Therefore, the RNG related instructions of the DLX-GA vastly contribute the performance improvement of GA execution.

Table 6: Simulation condition.

Number of generations	100
Number of chromosomes	100
Chromosome length	100 bits
Crossover probability	0.6
Mutation probability	0.01
Selection operator	roulette selection
Crossover operator	1-point crossover
Mutation operator	point mutation

Table 7: Percentage of each genetic operation.

problem	selection	crossover + mutation	evaluation	rng
1	10.8	13.6	19.4	56.2
2	11.3	15.7	20.5	52.5

Next, as a more practical case, we compared the execution time of two simple GA programs, one of which is composed of the DLX instructions only and the other is composed of the DLX and DLX-GA instructions. Simulation was done to execute both programs on the DLX-GA processor. The objective of the problem is to set all bits in a chromosome to 1. GA parameters are shown in Table 8. Table 9 shows the result. From Table 9, the DLX-GA was about 3 times faster than the DLX.

Table 8: Parameter setting.

Number of generations	64
Number of chromosomes	64
Chromosome length	64 bits
Crossover probability	0.6
Mutation probability	0.01
Selection operator	roulette selection
Crossover operator	1-point crossover
Mutation operator	point mutation

Table 9: Simulation result.

processor	average fitness	computation time [ms]
DLX	50.0	1029.48
DLX-GA	50.3	355.73

Now, we consider, in this case, why the DLX-GA is 3 times faster than the DLX. Compared to the DLX, for the DLX-GA, the time required for crossover and mutation was reduced roughly from 15% to 1% by using the bit-oriented operations and the SIMD operations, and the time of random number generation was reduced roughly from 55% to 1% by using the RNG related operations. Therefore, the time required for the GA execution on the DLX-GA was reduced to 32% of whole execution time of the DLX. This is the reason why the DLX-GA is 3 times faster than the DLX.

4 Conclusion

In this paper, we have proposed a new RISC architecture for high-speed execution of genetic algorithms. The proposed processor supports several types of instructions, which were devised to execute GA programs efficiently. Simulation experiments show that the proposed processor will be very effective to execute a GA program with a short computation time. The LSI design of the proposed processor has been completed, and the new processor will be fabricated as a standard cell LSI with a CMOS $0.35\mu m$ technology by July, 2001. Software environment such as a C compiler of the proposed processor is under the development. After fabricating the processor as an LSI chip, we will develop a general-purpose GA board, which consists of a DLX-GA chip, memory, and peripheral circuits to evaluate the DLX-GA processor in the real world applications. We also have a plan to realize a parallel GA system, which consists of a few tens of the above mentioned general-purpose GA board.

Acknowledgments

This research was supported in part by Grant-in-Aid for Scientific Research (C)(2)(No.12838008) from the Ministry of Education, Culture, Sports, Science and Technology, Japan. The VLSI chip in this study has been fabricated in the chip fabrication program of VLSI Design and Education Center (VDEC), the University of Tokyo with the collaboration by Rohm Corporation and Toppan Printing Corporation.

References

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

[2] P. Graham, and B. Nelson: "A hardware genetic algorithm for the traveling salesman problem on Splash 2," in *Field Programmable Logic and Applications*, ed. W. Moore and W. Luk, pp.352-361, Springer (1995).

[3] J. L. Hennessy, and D. A. Patterson: *Computer Architecture: A Quantitative Approach, 2nd Edition*, Morgan Kaufmann Publishers, Inc. (1995).

[4] Shinya Koizumi: Design and implementation of a RISC processor for high-speed execution of a genetic algorithm, *Master thesis*, Graduate School of Engineering, Hiroshima University (2001).

[5] M. M. Meysenburg, and J. A. Foster: Randomness and GA performance, revisited, in *Proc. Genetic and Evolutionary Computation Conference 1999*, pp.425–432 (1999).

[6] Z. Michalewicz: *Genetic Algorithms + Data Structures = Evolution Programs: Third, Revised and Extended Edition*, Springer-Verlag (1996).

[7] S. D. Scott, A. Samal, and S. Seth: "HGA: A hardware-based genetic algorithm," *Proc. ACM/SIGDA International Symp. on Field Programmable Gate Arrays*, pp.53-59 (1995).

[8] M. Serra, T. Slater, J. C. Mizuo *and* D. M. Miller: "The analysis of one-dimensional linear cellular automata and their aliasing properties," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 767–778 (1990).

[9] R. E. Smith, D. E. Goldberg and J. A. Earickson: "SGA-C: A C-language implementation of a simple genetic algorithm," *Tech. Rep.* No.91002, TCGA Report, Department of Engineering Mechanics, The University of Alabama (1994).

[10] D. E. Thomas, and P. R. Moorby: *The Verilog Hardware Description Language, Fourth Edition*, Kluwer Academic Publishers (1998).

[11] S. Wakabayashi, T. Koide, K. Hatta, Y. Nakayama, M. Goto, and N. Toshine: "GAA: A VLSI genetic algorithm accelerator with on-the-fly adaptation of crossover operators," *Proc. International Symposium on Circuits and Systems*, Vol.2, pp.268-271 (1998).

[12] N. Yoshida, T. Yasuoka, and T. Moriki: "Parallel and distributed processing in VLSI implementation of genetic algorithms," *Proc. 3rd International ICSC Symp. on Soft Computing*, pp.450-454 (1999).

Soft Sensor Development Using Genetic Programming

Arthur K. Kordon
The Dow Chemical Company
Freeport, TX 77541
USA

Guido F. Smits
The Dow Chemical Company
Terneuzen
The Netherlands

Abstract

A novel methodology for development of soft sensors based on sensitivity analysis and function generation by genetic programming is proposed. The main advantages of this type of soft sensor are their good generalization capabilities, explicit input/output relationships, and low implementation and maintenance cost. An example of a soft sensor generated by genetic programming in an industrial application in The Dow Chemical Company is given.

1 INTRODUCTION

Inferential sensing, also called soft sensing, involves the use of readily available process measurements to infer process state and product quality variables that are difficult to measure on-line (composition, melt index, molecular distribution, etc.). The foundation of building soft sensors is the assumption that these variables have a functional relationship with the measured process variables. Since this functional relationship is usually nonlinear, the neural network approach is a convenient choice for modeling. The common methodology of building a neural net soft sensor and the practical issues of its implementation are discussed in detail in [Qin, 1996]. Neural net based soft sensor technology is in its mature state with thousands of successful applications worldwide in all areas of manufacturing [Neelakantan and Guiver, 1998]. However, several performance and long-term operation issues had appeared, along with the benefits that soft sensors have shown in these applications. Most of the problems are related to some limitations that are typical for soft sensors based on neural nets. As it is well known, neural nets are universal approximators but usually have poor generalization capability outside the range of training data (Haykin, 1998). The result of this property is very poor performance of the soft sensor and unreliable prediction of the inferred value in new operating conditions. This problem can be avoided to some extent if the neural net has confidence limits that indicate the validity of model predictions. Another drawback of soft sensors based on back propagation neural nets is their complexity. Selection of the neural net structure is still an *ad hoc* process and very often leads to inefficient and complex solutions. This “fat” dimensionality significantly reduces the robustness of soft sensors. Of special importance is the selection of only those inputs that have a major influence on the inferred variable. In order to achieve proper input selection we need a sensitivity analysis of the influence of each input on the output. This type of analysis is very difficult to perform by the existing classical back-propagation neural nets. As a result of this non-efficient structure and reduced robustness there is a necessity of frequent re-training. The final effect of all of these problems is an increased maintenance cost and gradually decreased performance and credibility.

In order to improve soft sensor performance, to shorten its development time, and to minimize maintenance, a new hybrid intelligent system methodology was developed in The Dow Chemical Company. It is based on the use of different intelligent system components (analytic neural nets, genetic programming, support vector machines, etc.) during the development. Part of this methodology is based on genetic programming (GP) [Koza, 1992, Banzhaf *et al*,1998]. The development issues of inferential estimation models based on GP is discussed in [Willis *et al*, 1997, McKay *et al*, 1997]. Three cases (vacuum distillation column, continuous stirred tank reactor, and twin screw cooking extruder) were used to investigate the utility of this approach. The results revealed that in each case the GP algorithm generates an accurate nonlinear empirical model. Moreover, in all of the examples, the GP algorithm was able to discriminate between relevant and irrelevant inputs, evolving parsimonious system presentations. These initial encouraging results based on model simulations are a good starting point for real-world industrial applications of soft sensors based on GP.

In this paper, the implementation issues for development of industrial soft sensors generated by GP are presented. The main steps of a hybrid intelligent system methodology for robust soft sensors are discussed in Section 2 with emphasis on GP-related phases. Section 3 presents the results from a successful application of a GP soft sensor in a chemical reactor. The conclusions are summarized in Section 4.

2 A METHODOLOGY FOR DEVELOPMENT OF GP-GENERATED SOFT SENSORS

With the expanding research in the area of evolutionary algorithms and continuously increasing computational power of PCs, genetic programming is beginning to grab the attention of industry. Of special importance to industry are the following unique features of GP:

- no *a priori* modeling assumptions
- derivative-free optimization
- few design parameters
- natural selection of the most important process inputs
- parsimonious analytical functions as a final result.

The last feature has double benefit. From one side, a simple soft sensor often has better generalization capability, increased robustness, and needs less frequent re-training. From the other side, process engineers and developers prefer to use non-black box empirical models and are much more open to take the risk to implement inferential sensors based on functional relationships. An additional advantage is the low implementation cost of such type of soft sensors. It can be applied directly into the existing Distributed Control Systems (DCS) avoiding additional specialized software packages, typical for neural net-based inferential sensors.

At the same time there are still significant challenges in implementing industrial soft sensors generated by GP: function generation with noisy industrial data [Lee and Wang, 1995], dealing with time delays, sensitivity analysis of large data sets [Gilbert *et al*, 1998], to name a few. Of special importance is the main drawback of GP – the slow speed of model development due to the inherent high computational requirements of this method. For real industrial applications the calculation time is in order of days or even weeks, even with the current high-end PCs.

These problems inherent to GP can be partially overcome by integration with other approaches in soft sensor development. For example, support vector machines can detect outliers and compress the data set only with the most informative data [Vapnik, 1998], time delays can be "absorbed" by convolution functions tuned by neural networks [Tank and Hopfield, 1987], operating regime-related data can be identified by principle component analysis, etc. The objective of the integration is to supply GP with clean, informative and parsimonious data sets. In this way all the advantages of GP are enhanced and its drawbacks are reduced.

The pre-GP steps of the hybrid intelligent systems methodology for soft sensor development are the following:

<u>Step 1:</u> Representative data collection

A representative data set of a broad range of potential inputs to the soft sensor is collected with an appropriate sampling time. The average size of this initial data set for an industrial soft sensor application is of several dozens of inputs and several thousands data vectors and could be a challenge if applied directly to GP.

<u>Step 2:</u> Data preprocessing and classification

This step includes all necessary actions to assess data quality, fill data gaps, perform data transforms if necessary, etc. In case of multiple product type manufacturing with several operating regimes it is necessary to classify the data to the appropriate operating conditions. Of special importance to data cleaning is the reliable detection of all outliers. A new very effective approach for outlier elimination is the ε-insensitive support vector machine for regression [Vapnik, 1998]. By selecting a proper kernel and a width of the ε-insensitive zone one has explicit control over the threshold of outliers detection and model complexity. The final result of this step is a clean, condensed and informational rich data set.

<u>Step 3:</u> Neural net sensitivity analysis of all possible inputs

Even a condensed data set of tens of inputs and thousands of observations is a challenge for effective GP-model generation. In order to reduce the search space for GP and computational effort, a preliminary sensitivity analysis of all possible inputs is performed. The sensitivity analysis is based on stacked analytic neural nets (Smits, 1993). Typically thirty stacked neural nets are used to improve generalization and estimate confidence limits. This step begins with the most complex structure of all possible inputs. During the sensitivity analysis the initial complex structure is gradually reduced by decreasing the number of inputs. The sensitivity of each structure is the average of the calculated derivatives on every one of the stacked neural nets. The procedure performs automatic elimination of the least significant inputs and generates a matrix of input sensitivity vs. input elimination. This matrix is the basis for selection of the most influential inputs for the final nonlinear sensitivity analysis by GP.

<u>Step 4:</u> Convolution parameters' estimation

This step is necessary when we have to deal with time delays. The classical approach to handle time series by neural nets is to add additional inputs for the previous time steps. Unfortunately, this technique increases the dimensionality of the neural net significantly. For example, if one has a problem with five inputs and one wants to use the current input plus the inputs from five previous time-steps as inputs to the network, then one needs a network with 30 inputs as opposed to the original five. This increase in the dimensionality of the input vectors has a large impact on the number of required data points for a proper model identification. The problem is even bigger in the case of GP modeling. Therefore, it would be desirable to include information from previous time-steps without increasing the dimensionality of the input to the network. This can be achieved by performing a convolution on the input using an appropriately shaped function.

The steps directly related to GP are as follows:

Step 5: Final GP-based sensitivity analysis

Sensitivity analysis generates a ranking of all the input variables in terms of how important they are in modeling a certain unknown process. In linear problems the sensitivity of an input variable is related to the derivative of the output with respect to that variable. In nonlinear problems, however, the derivative becomes a local property and has to be integrated over the entire input domain to qualify as sensitivity. Since this approach is not really practical in a GP context we've opted to relate the sensitivity of a given input variable to its fitness in the population of equations. The reasoning is that important input variables will be used in equations that have a relatively high fitness. So the fitness of input variables is related to the fitness of the equations they are used in. There is however a potential problem in credit assignment i.e. what portion of the fitness goes to what variable in the equation. The easiest approach is to distribute the credit (the fitness of the equation) equally over all variables present. But probably not every variable is equally important in a given equation. In addition, most equations in a GP population are not parsimonious and posses chunks of inactive code (the problem of 'bloat' [Banzhaf *et al*, 1998]). Variables that are present in these chunks of inactive code do not contribute to the final fitness of the equation but still obtain some credit for being part of that equation. There is no direct solution for this problem on the individual equation level but still reliable answers can be obtained provided we evaluate a large number of equations. Again the reasoning is simple, if a given input variable is absolutely essential to solve the problem, it must be present in the high fitness equations. Other non-essential variables will be present in both low-fitness and high-fitness equations so their fitness will be closer to the average fitness over all equations. More important variables will obtain more credit and will have a fitness that exceeds this average value. So provided the population size is large enough we can take the fitness of each equation in the population, distribute this fitness in equal amounts over the input variables present in that equation and sum all these contributions for each input variable over the entire population. An improved version of this, at the expense of little bit extra computation, uses every sub-equation in each of the equations in the population. The extra computational step will considerably improve the statistics of the input variable fitnesses since now the number of equations is equal to the sum of all nodes in every equation-tree in the population rather than the population size itself.

Step 6: Genetic programming function generation

This step uses the GP approach to search for potential analytical relationships in a condensed data set of the most sensitive inputs. The search space is significantly reduced by the previous steps and the effectiveness of GP is considerably improved. The set of possible functions that can be generated in GP is the set of all possible functions that can be composed from the list of available terminals $T = \{X_1, X_2, ...X_n\}$ and the set of available functions $F = \{F_1, F_2, ...F_m\}$.

Various parameter settings control the type and complexity of equations that are generated. The most important parameters are the list of available functions as well as the list of available inputs. The list of available functions is set at the start of a run. The list of available inputs is usually fixed but they can vary, for example in the case of sensitivity analysis where inputs are eliminated during succesive runs. Another parameter that is quite important in controlling the average complexity of the equations being generated is the probability for function selection (default value equals 0.6). This parameter controls what the probability is to grow a specific branch of a tree by selecting a function or terminating the branch by selecting a terminal (a number or a variable) as the next node. The larger this probability value is, the higher the complexity of the functions being generated.

Both input variables and available functions can have an associated fitness that evolves during a GP run. This can significantly accelerate the search especially for problems with a large number of candidate variables of which only a few are needed for the optimal solution. There is also a danger however, the risk exists that some variables get low fitness values early in the run simply because they are under-represented in the population and receive low fitness values accordingly. Because these variables have low fitness values, the chances of being included in an equation in subsequent generations gets lower and lower, which furthur decreases their fitness value. It is important to keep a sufficient balance between exploration and exploitation of the search space when input fitnesses are also allowed to evolve.

In evaluating an equation we're not only interested in the final result of the entire equation but also in the intermediate results at every node in the tree representing the equation. For example, if the equation is x3*(x1+x2), the function would return calculated y-values and fitnesses for the set of functions: {x1, x2, x3, x1+x2, x3*(x1+x2)}. The node-based fitnesses are stored in an equation specification matrix and are used at a later stage to calculate variable fitnesses based on the evaluation of the entire population. This information will also allow us to extract subequations (which might have a better fitness than the overall equation) with a high fitness and low complexity to be used as nonlinear transforms in other applications. The final result of this important step of the methodology is a list of several analytical functions and subequations that satisfy the best solution according to a defined objective function.

Step 7: Analytical function selection/verification

The analytical function selection is still more of an art than a well-defined procedure. Very often the most parsimonious solution is not acceptable due to specific manufacturing requirements. It is preferable to deliver several potential functions with different levels of complexity and let the final user make the decision. The generalization capabilities of each soft sensor are verified for all possible data sets. Of special importance is the performance outside the training range.

Step 8: On-line Implementation

The selected off-line model (neural net or analytical function) has to be implemented on-line. This includes the on-line data set structure preparation, implementation of all necessary transforms of the data and the model itself. There are several options to apply the model on-line. Very simple analytical functions without complex data transforms can be directly coded in the existing Distributed Control Systems. If the soft sensor is a part of an integrated system, it is possible to implement it in G2 (a real time expert system shell developed by Gensym Corporation). This software environment can integrate different types of soft sensors. Those based on simple analytical functions can be implemented as G2 procedures. Complex models based on back-propagation neural nets can be implemented with Gensym NeurOnline. G2 has the capability to link all these hybrid intelligent systems components with an inference engine. In spite of its high level of complexity, the final integrated system is user-friendly and simple to be used by process operators.

Step 9: Soft sensor maintenance

This step includes the safety net that guarantees long-term robustness of the soft sensor – input data quality test, operating range test, prediction quality test, criteria for periodic re-training, etc. It is one of the most important factors in the business decision-making process for soft sensor implementation. The potential for increased robustness of the GP-generated soft sensors in comparison to neural net-based leads to less frequent re-training and respectively to lower maintenance cost.

GP-generated soft sensors have the potential to be more robust for real industrial applications than neural nets. One of the significant factors is the ability to examine the behavior of the model outside the training range. With a functional solution it can be done in an easy and direct way while this is more difficult for the case of a black-box model. Another factor in favor of GP-generated soft sensors is the ability to impose external constraints in the modeling process and to improve the extrapolation properties of the final model.

Some of the advantages of the proposed methodology for GP-generated soft sensors will be illustrated with an industrial application in a chemical reactor.

3. A GP-GENERATED SOFT SENSOR FOR A CHEMICAL REACTOR

Some of the critical parameters in chemical processes are not measured on-line (composition, molecular distribution, density, viscosity, etc.) and their values are captured either by lab samples or off-line analysis. However, for process monitoring and quality supervision the response time of these measurements with low frequency (several hours even days) is very slow. When the critical parameters are not available on-line in situations with alarm showers due to complex root causes the negative impact could be significant and eventually could lead to shutdown. One of the approaches to address this issue is through development and installation of expensive hardware on-line analyzers. Another solution is by using robust soft sensors. We will illustrate the methodology with such type of application – development of a soft sensor for prediction of a critical parameter in a chemical reactor. The objective of this soft sensor is an early detection of complex alarms.

3.1 Pre-GP processing

The original data set contains 6500 pairs of 25 potential inputs and one output. The inputs are the hourly averaged reactor temperatures, flows, and pressures. The output is the critical parameter measured by lab analysis of a grab sample every 8 hours. As a result of data preprocessing three data sets were created: reference (training) data set of 2000 data points, test data set of 2000 data points for evaluating the generalization capability of the trained model, and validation data set of 2300 data points for final assessment of soft sensor performance.

The sensitivity analysis for all inputs was performed with 30 stacked analytic neural nets in MATLAB [Smits, 1993]. In order to improve the reliability of the results, the sensitivity analysis was run several times with swapped reference and data sets. The input/output sensitivities of all potential inputs after the process of automatic input elimination are shown on Fig. 1.

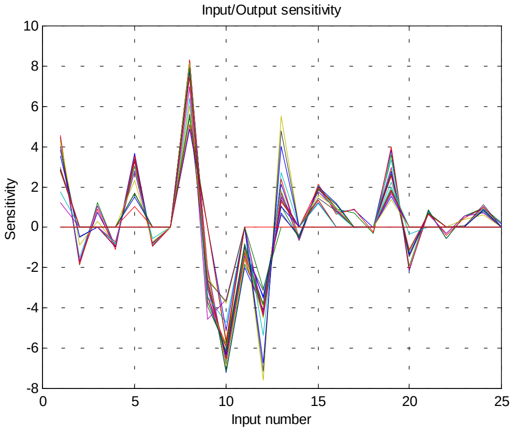


Figure 1. Input/output sensitivity of all 25 potential inputs

The dominance of several inputs is evident (especially X8, X10, X12, and X13). Only the top 10 most sensitive inputs are selected for the final GP sensitivity analysis.

One of the key tasks for developing soft sensors in chemical industry is estimation of the value of the time delay *k* between the input process variables and the output – the critical parameter lab sample. In the proposed methodology the time delay is absorbed by a convolution function. The convolution parameters are obtained by running 30 stacked neural nets with a range of peaks of time delay *k* between one and eight hours (the time span of grab sampling) . The highest correlation coefficient is for *k*=5, and this is the selected value for the peak time delay of the convolution function. The meaning of this parameter is that the soft sensor has a “predictive” capability relative to the time when the lab sample is collected. On average, the soft sensor can predict the critical parameter five hours ahead of the lab sample analysis. This is a critical feature for soft sensors related to alarm processing.

3.2 GP sensitivity analysis

The final GP sensitivity analysis is based on the selected data set of 10 inputs transformed by the convolution function with peak time delay of five hours. The procedure starts with all 10 inputs. Every evaluation period lasts 20 runs with population size of 200, number of generations of 50, number of reproductions per generation of 4, probability for function as next node as 0.6 and correlation coefficient as optimization criterion. A snapshot of input/output sensitivities of the top ten potential inputs and the sensitivity change during the run is shown on Fig. 2.

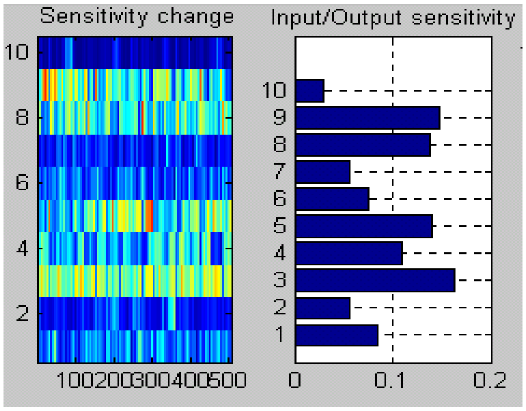


Figure 2. GP-based input/output sensitivity of the top 10 most influential inputs

At the end of any evaluation period the least significant input is eliminated and the next period begins with one input less. The final result from the GP sensitivity analysis is inputs ranking based on their final sensitivity at the end of each evaluation period.

3.3 Function generation

The initial functional set for the GP includes:{addition, subtraction, multiplication, division, square, change sign, square root, natural logarithm, exponential, cosine, sine, ispositive (1 if (x>0) else 0), power and a simple first order filter}. Function generation takes 20 runs with population size of 200, number of generations of 50, number of reproductions per generation of 4, probability for function as next node as 0.6, parsimony pressure of 0.01 and correlation coefficient as optimization criterion. The selection of the best candidate is based on a balance between function complexity and residual error. An important consideration was that the primary purpose of this soft sensor is to trigger an alarm based on the high value of the critical parameter. From the generated list of potential solutions the best fit was found for an analytical function of the type:

$$y = a + b \cdot \left(e \cdot \left| \frac{x3}{x5} - d \right| \right)^{c}$$

where x3 and x5 are the corresponding inputs from the top 10 selection, y is the predicted output, and a,b, c,d, and e are adjustment parameters. The performance of the selected functions in all three data sets was very good. The correlation coefficient for the training set was 0.87, 0.79 for the test set, and 0.81 for the validation set.

3.4 On-line performance

The simplicity of the selected function for critical reactor parameter prediction allows its implementation directly in the Distributed Control System. In addition, the GP-generated soft sensor was implemented in Gensym G2. This was done because the predictor is a critical alarm indicator in an Expert System for Alarm Troubleshooting. The system is in operation since November 1997 and initially it included a soft sensor for one reactor. An example of successful alarm detection several hours before the lab sample is shown on Fig.3.

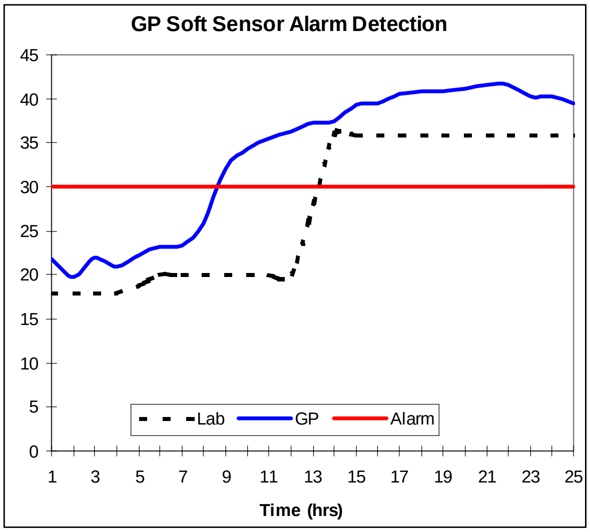


Figure 3. Fast alarm detection by the GP-generated soft sensor

The robust performance for the first six months gave the confidence of process operation to ask for leveraging the solution to all three similar chemical reactors in the unit. The only procedure that was necessary to fulfil this task was to fit the parameters of the GP-generated function to the data set from the other two reactors. Since the fall of 1998 the three soft sensors are in operation without need for re-training. The prediction quality is with standard deviation close to that of the lab measurement (between 2.9% and 4.1% vs. 2% for the lab measurement). The robust long-term performance of the GP-generated soft sensor convinced the process operation to reduce the lab sampling frequency from once a shift to once a day since July 1999. The maintenance cost after the implementation is minimal and covers only the efforts to monitor the performance of the three soft sensors.

4. CONCLUSIONS

A novel hybrid intelligent systems methodology for inferential sensing has been defined and successfully applied for fast and effective development of a soft sensor in a chemical reactor in The Dow Chemical Company. The proposed methodology is based on using different intelligent system components (stacked and convolution neural nets, genetic programming, support vector machines, etc.). A significant part of the methodology is based on the unique features of GP to deliver the final solution as a very simple analytical function. The illustrated application shows the main advantage of the proposed methodology – design of a very compact and robust empirical model that requires minimal re-training and maintenance cost. The success of this application in a complex chemical process demonstrates the great potential of GP as a very effective complement to neural net-based soft sensors.

REFERENCES

Banzhaf W., P. Nordin, R. Keller, and F. Francone, *Genetic Programming: An Introduction*, Morgan Kaufmann, San Francisco, 1998.

Gilbert, R., R. Goodacre, B. Shann, D. Kell, J. Taylor, and J. Rowland, Genetic Programming-Based Variable Selection for High-Dimensional Data, GP'98, pp.109-115, 1998.

G2 User's Manual, Gensym Corporation, 1996.

Greef, D. and C. Aldrich, Empirical Modeling of Chemical Process Systems with Evolutionary Programming, *Computers chem Engng*, **22**, pp. 995-1005, 1998

Haykin, S. *Neural Networks: A Comprehensive Foundation*, Prentice Hall, New York, 1998.

Koza, J. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, MA, 1992.

Lee J. and P.Wang. The effect of function noise on GP efficiency. In X. Yao editor, *Progress in Evolutionary Computation* , volume 956 of *Lecture Notes in Artificial Intelligence* , pp. 1-16. Springer Verlag, Heidelberg, 1995.

McKey, B, M. Willis, and G. Barton, Steady-State Modeling of Chemical Process Systems using Genetic Programming, *Computers chem Engng*, **21**, pp. 981-996, 1997.

Neelakantan R. and J. Guiver, *Applying Neural Networks*, Hydrocarbon Processing, 9, pp. 114-119, 1998.

Nordin, P. *Evolutionary Program Induction of Binary Machine Code and its Applications*, Krehl Verlag, Munster, 1999.

Qin, S., *Neural Networks for Intelligent Sensors and Control - Practical Issues and Some Solutions*. In *Neural Systems for Control*, Academic Press, New York, 1996.

Tank, D. and J. Hopfield , Neural Computation by Concentrating Information in Time, *Proc. Of the National Academy of Sciences*, **84**, pp. 1896-1900. 1987.

Smits, G. Personal communication., 1993.

Vapnik, V. *Statistical Learning Theory*, Wiley, NY, 1998.

Willis M, H. Hiden, M. Hinchliffe, B. McKay, and G. Barton, Systems Modeling using Genetic Programming, *Computers chem Engng*, **21**, Suppl. S1161 – S1166, 1997.

Personalized Email Marketing with a Genetic Programming Circuit Model

Yung-Keun Kwon and Byung-Ro Moon
School of Computer Science and Engineering
Seoul National University
Shilim-dong, Kwanak-gu, Seoul, 151-742 Korea
{airen, moon}@soar.snu.ac.kr

Abstract

Personalization is a sharply growing issue to improve customers' loyalty and maximize marketing efficiency. We try to find a personalized prediction model in email marketing. We propose a circuit model combined with genetic programming. It generates recommendation rules using customer profiles. The model showed significant improvement over general mass marketing in a field test of an email marketing company.

1 Introduction

Personalization is a sharply growing issue in modern marketing. It helps boost customers' loyalty by providing the most attractive contents to each customer or by locating the most proper set of customers for an arbitrary advertisement [3]. As the internet expands rapidly, personal information and huge activity logs are accumulated. These data implicitly contain valuable trends and patterns which are useful to improve business decisions and efficiency. Email marketing is considered one of the most promising tools for internet marketing. Its response rate is known to be much higher than direct mailing or banner ads [1].

Knowledge discovery and data mining are techniques to discover implicit knowledge hidden in a large database. Diverse data mining tools have been studied with various problems including neural networks, decision trees, rule induction, bayesian belief networks, evolutionary algorithms, fuzzy sets, association rules, and clustering, and their particular merits and demerits have been enumerated [2][4]. A particular method is not the best in all situations and it is important to locate a method suitable to the problem.

Genetic programming (GP) [11] is one of the search techniques that utilize the principle of natural evolution. In a GP, a solution is represented by a tree representing a program or a rule. It is attractive since the object of data mining is extracting a useful rule from a huge quantity of unrefined data. Examples include the search for polynomial models of financial data series [8], decision support systems [12], the strategies of animal actions [13]. We use a genetic programming technique for finding combinational targeting rules in email marketing.

There are two main issues in developing a GP: 1) how to use the raw data, and 2) how to define the terminal node set and non-terminal node set. Personal information, e.g, age, sex, job, etc., and activities can be used as input data for personalized marketing. In this study, the original input data are discrete, nominal or boolean. We first convert discrete or nominal data into binary data to fit the circuit model. For encoding in a GP, one has to first define the terminal node set and the non-terminal node set. User profile variables become terminal nodes in this study. For non-terminal nodes, we defined 31 functions. A tree produced by the GP corresponds to a combinational circuit. We also devised a local optimization heuristic to enhance the GP's fine-tuning around local optima. The combinational circuit (logic) is used for locating customers that are expected to respond to an email campaign.

The rest of this paper is organized as follows. In section 2, we explain the problem and present the objective. In section 3, we describe our genetic programming circuit model. In section 4, we provide our experimental results. Finally, conclusions are given in section 5.

Table 1: Personal Information

	Variables	Type	Range	Description
1	*Age*	Discrete	$10 \sim 80$	
2	*Sex*	Boolean	$0 \sim 1$	
3	*Marriage*	Boolean	$0 \sim 1$	
4	*Job*	Nominal	$0 \sim 9$	
5	*Job position*	Nominal	$0 \sim 9$	
6	*Kind of Job*	Nominal	$0 \sim 9$	
7	*Education*	Nominal	$0 \sim 4$	
8	*Income*	Nominal	$0 \sim 9$	
9	*Automobile*	Boolean	$0 \sim 1$	
10	*Interest1*	Boolean	$0 \sim 1$	Interested in 'DOMAIN1'?
⋮	⋮	⋮	⋮	⋮
19	*Interest10*	Boolean	$0 \sim 1$	Interested in 'DOMAIN10'?
20	*Site1*	Boolean	$0 \sim 1$	Joined in 'WebSite10'?
⋮	⋮	⋮	⋮	⋮
34	*Site15*	Boolean	$0 \sim 1$	Joined in 'WebSite15'?
35	*Magazine1*	Boolean	$0 \sim 1$	Subscribing for 'Magazine1'?
⋮	⋮	⋮	⋮	⋮
50	*Magazine16*	Boolean	$0 \sim 1$	Subscribing for 'Magazine16'?

2 Preliminaries

2.1 The Problem

As mentioned, we attack the optimal targeting problem in email marketing. It originally started in the form of spam mail. These days email marketing companies acquire the permission of customers and this type of email marketing has become well established as a legal and promising business model. However, most emails for marketing purposes fail to draw attention before being discarded. A popular approach to alleviate this problem is using a rule-based targeting which reflects marketers' experience. This approach is helpful to some extent but has limitations in handling the vast amount of complex data available today.

Personal information generates a number of independent variables; the only dependent variable is whether the customer has answered or not to the email. The entire data set used in our study is from an email marketing company, Amail Inc. Table 1 shows the personal information included in the database. Fields 20 through 34 are from the company's joint marketing data and fields 35 through 50 are from the web magazines issued by the company. Each field corresponds to an independent variable.

A variable is discrete, nominal or boolean. However, since our approach is based on the circuit model, we modify each variable with a discrete or nominal value into a number of boolean variables. For example, a discrete variable, *Age*, is divided into six boolean variables; a nominal variable, *Job*, is divided into nine boolean variables representing nine job categories, respectively. In summary, we have 86 independent variables from the 50 original variables in Table 1.

2.2 The Objective

The objective is to raise the response rate which is defined to be

$$R = \frac{C}{N}$$

where N is the number of customers receiving the email and C is the number of customers answering to the email.

2.3 The Process

We tested our approach in the following process. First, we prepare an email campaign and send an email to the customers in the training data set. GP generates a rule from the responding result. We select customers from the test data set on the basis of the rule and send the email to them. Finally, we evaluate the response rate.

Table 2: Non-Terminal Node Set in CGP

Type	Non-terminal Node	Arity	Parameters	Description
Logical function set	TRUE1	2	a, b	a
	TRUE2	2	a, b	b
	FALSE1	2	a, b	$\neg a$
	FALSE2	2	a, b	$\neg b$
	AND	2	a, b	$a \wedge b$
	NAND	2	a, b	$\neg(a \wedge b)$
	AND2	2	a, b	$a \wedge \neg b$
	AND3	2	a, b	$\neg a \wedge b$
	OR	2	a, b	$a \vee b$
	NOR	2	a, b	$\neg(a \vee b)$
	OR2	2	a, b	$\neg a \vee b$
	OR3	2	a, b	$a \vee \neg b$
	XOR	2	a, b	$(a \wedge \neg b) \vee (\neg a \wedge b)$
	NXOR	2	a, b	$(a \wedge b) \vee (\neg a \wedge \neg b)$
	NOT	1	a	$\neg a$
Conditional function set	IF0	4	a, b, c, d	always c
	IF1	4	a, b, c, d	if TRUE1(a, b), then c else d
	IF2	4	a, b, c, d	if TRUE2(a, b), then c else d
	IF3	4	a, b, c, d	if FALSE1(a, b), then c else d
	IF4	4	a, b, c, d	if FALSE2(a, b), then c else d
	IF5	4	a, b, c, d	if AND(a, b), then c else d
	IF6	4	a, b, c, d	if NAND(a, b), then c else d
	IF7	4	a, b, c, d	if AND2(a, b), then c else d
	IF8	4	a, b, c, d	if AND3(a, b), then c else d
	IF9	4	a, b, c, d	if OR(a, b), then c else d
	IF10	4	a, b, c, d	if NOR(a, b), then c else d
	IF11	4	a, b, c, d	if OR2(a, b), then c else d
	IF12	4	a, b, c, d	if OR3(a, b), then c else d
	IF13	4	a, b, c, d	if XOR(a, b), then c else d
	IF14	4	a, b, c, d	if NXOR(a, b), then c else d
	IF15	4	a, b, c, d	always d

3 Circuit Genetic Programming (CGP)

3.1 Genetic Programming Frameworks

We use a traditional GP framework for our CGP. First, it prepares initial random population where each solution is represented in a tree. It makes the population evolve through a number of generations until a stopping condition is met. Each generation evolves by selection, crossover, reproduction and mutation operations.

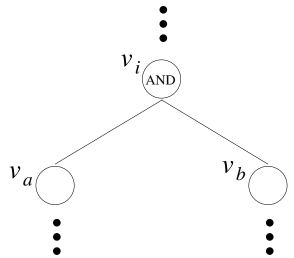
The selection operation is performed in probabilistic proportion to fitness, which is called roulette wheel selection. Reproduction simply chooses an individual in the current population and copies it into the new population. Crossover occurs on two parent trees A and B. The crossover operation is performed by randomly selecting an arbitrary node from each tree (say v_a and v_b, respectively) and exchanging the two subtrees rooted by v_a and v_b. Mutation is performed by picking a random node, deleting the corresponding subtree, and replacing it with a randomly generated subtree.

3.2 Problem Representation

Two sets have to be defined to construct trees in GP: the terminal node set and non-terminal node set (function node set). We use the 86 independent variables for terminal nodes described in Section 2.1. The non-terminal node set consists of logical functions and conditional functions. Table 2 shows the set of non-terminal nodes. The arity of a function node v_i means the number of its children nodes.

3.3 Local Optimization

After mutation, local optimization is performed on the tree. Let U be the set of all customers (in the training set) and A be the customers that have responded to the email. The quality of a tree is evaluated by the



(1) Logical function node v_i

v_a	v_b	A	$\bar{A}$
0	0	10	2400
0	1	40	100
1	0	40	100
1	1	10	2400

(2) Results in v_a, v_b

v_i	A	$\bar{A}$
0	90	2600
1	10	2400

(3) Results in v_i with "AND"

v_i	A	$\bar{A}$
0	20	4800
1	80	200

(4) Results in v_i with "XOR"

Figure 1: A situation around a logical function node

response rate defined in Section 2.2. To obtain a high response rate, the tree is desired to be a rule that produces 1 (True) for as many customers in A as possible and that produces 0 (False) for as many customers in $\bar{A}$ as possible. The usefulness of a node depends strongly on the outputs of its subtrees rooted by its children. Analyzing the evaluation results of customers in A and $\bar{A}$, respectively, we change the function with the most appropriate one. We recursively update the functions from the deepest nodes to the root node and get an improved tree.

3.3.1 Optimization of logical functions

Figure 1(1) shows a part of a tree where a node has the logical function "AND" and Figure 1(2) is a table that shows the distribution of the outputs in v_a and v_b. In A, most customers have $\langle 0, 1\rangle$ or $\langle 1, 0\rangle$ for $\langle v_a, v_b\rangle$. On the other hand, most customers in $\bar{A}$ have outputs $\langle 0, 0\rangle$ or $\langle 1, 1\rangle$. Figure 1(3) and Figure 1(4) show the distribution of the outputs in v_i when related to functions "AND" and "XOR", respectively. In this case, "XOR" is more attractive than "AND". In this way, we can find the most attractive function for the node v_i. However, when the training set is large, it is time consuming to use this heuristic. In practice, we select a random subset of the training set and apply the heuristic.

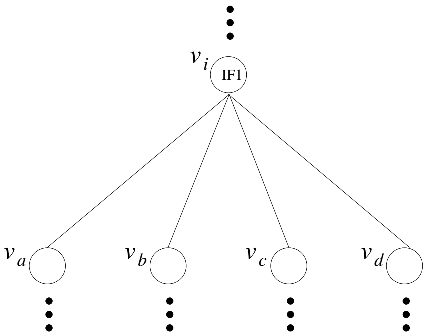
We should note that the most attractive function does not have to produce as many 1's as possible for A (as well as 0's for B). It only has to well "divide" the outputs between A and $\bar{A}$. For example, if the XOR of Figure 1(4) divided the outputs of A and $\bar{A}$ exactly in the opposite way, i.e., $\langle 20, 80\rangle$ for $\bar{A}$ and $\langle 4800, 200\rangle$ for A, its merit remains unchanged.

3.3.2 Optimization of conditional functions

Figure 2(1) shows a part of a tree where a node has the conditional function "IF1" and Figure 2(2) is a table that shows the distribution of the outputs in its four children. We wish to change the current function to the most proper one in Table 2. We have to decide which of v_c and v_d is better as the output for each pair of the outputs in v_a and v_b: $\langle 0, 0\rangle$, $\langle 0, 1\rangle$, $\langle 1, 0\rangle$, and $\langle 1, 1\rangle$. When the outputs in v_a and v_b are $\langle 0, 1\rangle$ or $\langle 1, 1\rangle$, the output in v_c divides the pattern better than the output in v_d. On the contrary, when the outputs are $\langle 1, 0\rangle$ or $\langle 0, 0\rangle$, the output in v_d divides the pattern better than the output in v_c. Thus, "IF2" is better than "IF1" for v_i. Figure 2(3) and Figure 2(4) show the distribution of the outputs in v_i when the function in v_i is "IF1" and "IF2", respectively. One can see that "IF2" divides the pattern better in A and $\bar{A}$. In this way, we select the most attractive conditional function from among those in Table 2.

3.4 Space Smoothing

Generally, the number of customers in a typical email marketing company is fairly large and it is time consuming to handle them. We apply a space smoothing technique to CGP to save time. CGP starts training with a small number of customers and increases the number gradually. CGP with space smoothing takes visibly less time than the version without it. We observed that the space smoothing did not considerably harm the performance.



(1) Conditional function node v_i

$v_a v_b$	v_c	A	$\bar{A}$	v_d	A	$\bar{A}$
0 0	0	5	500	0	10	0
	1	5	1000	1	0	1500
0 1	0	0	1000	0	20	500
	1	40	0	1	20	500
1 0	0	5	1000	0	0	1500
	1	5	500	1	10	0
1 1	0	10	1000	0	40	1000
	1	30	0	1	0	0

(2) Results in v_a, v_b

v_i	A	$\bar{A}$
0	45	2500
1	55	2500

(3) Results in v_i with "IF1"

v_i	A	$\bar{A}$
0	20	3500
1	80	1500

(4) Results in v_i with "IF2"

Figure 2: A situation around a conditional function node

Table 3: Statistics of Two Campaigns

(1) AD1

Data set	A	U	R_0
Training data	131	1800	7.28%
Test data	115	7021	1.64%
Total	246	8821	2.29%

(2) AD2

Data set	A	U	R_0
Training data	126	1800	7.00%
Test data	144	6380	2.26%
Total	270	8180	3.30%

4 Experimental Results

We tested the CGP with two email campaign data sets, say AD1 and AD2, of an email marketing company, Amail Inc. We divided each campaign data set into two disjoint sets – the training set and the test set – following the convention in data engineering. Table 3 shows the statistics. R_0 means the response rate of each set from random targeting. In these samples, R_0 of the training set is larger than that of the test set, which may seem rather unusual. The reason for this is that when the R_0 of the training set was very small, the generated rules were dominated by the customers in $\bar{A}$ and did not show satisfactory results.

Table 4 shows the experimental results. In the table, "CGP-L" means CGP with local optimization and "CGP-LS" is CGP-L with space smoothing. In both campaigns, CGPs' response rates were consistly higher than random targeting. The rates of improvement ranged from 11% to 37%. We tuned the running times of CGP and CGP-L to be comparable. CGP-L produced visibly better results than CGP. This is evidence of the effectiveness of the local optimization. In summary, CGP showed an average 20% performance improvement over random targeting; local optimization provided an additional 9.5% improvement over CGP. We should also note that the space smoothing caused negligible performance degradation with about 30% running-time reduction.

We also provide experimental results from another campaign. For the campaign AD3, we sent emails to randomly selected customers. With the remaining customers not in the random set, we selected customers by three different targeting methods: collaborative filtering (CF) [5], artificial neural network (ANN) [6], and CGP-LS. Collaborative filtering is a proven standard for personalized recommendations [7][10]. A representative company using collaborative filtering is NetPerceptions, Inc. Artificial neural networks have been used to solve a variety of problems in optimization, pattern recognition, prediction, function approximation, etc. An ANN learns from its environment through an interactive process of adjusting its weights [6][9].

Table 5 shows the results. CGP-LS showed a 4.78% response rate while ANN and CF showed 4.44% and 4.00%, respectively. This represents a 42.7% improvement over the random targeting.

Table 4: Results of Our Approaches

(1) Result of AD1

S	Training data				Test data				Trials
	$E(A)$	$E(U)$	R	R/R_0	$E(A)$	$E(U)$	R	R/R_0	
Random	131.00	1800.00	7.28%	1.00	115.00	7021.00	1.64%	1.00	
CGP	97.55	1000.85	9.75%	1.34	93.95	5149.80	1.82%	1.11	20
CGP-L	76.95	686.65	11.21%	1.54	80.75	4039.25	2.00%	1.22	20
CGP-LS	82.65	796.85	10.37%	1.42	88.10	4362.05	2.02%	1.23	20

(2) Result of AD2

S	Training data				Test data				Trials
	$E(A)$	$E(U)$	R	R/R_0	$E(A)$	$E(U)$	R	R/R_0	
Random	126.00	1800.00	7.00%	1.00	144.00	6380.00	2.26%	1.00	
CGP	49.55	377.95	13.11%	1.87	62.80	2175.95	2.89%	1.29	20
CGP-L	49.00	344.58	14.22%	2.03	63.16	2041.53	3.09%	1.37	20
CGP-LS	52.30	384.50	13.60%	1.94	67.40	2235.35	3.02%	1.34	20

5 Conclusion

In this paper, we proposed a genetic programming circuit model with a local optimization heuristic. It showed a significant performance improvement over random targeting with the field data of an email marketing company. For the circuit model, we modified every input variable with a discrete or nominal value into a number of boolean variables. We also devised a local optimization heuristic to enhance the GP's fine-tunning around local optima. This turned out to contribute to the performance improvement.

It should be pointed out that the suggested model has a limitation: Since the outputs of a circuit is always boolean, it is hard to control the number of customers for recommendation. On the other hand, other methods such as collaborative filtering and neural networks can control the number. Future study will include controling the number of customers with non-binary GP models. The proposed model is applicable not just to email marketing and may be extended to other personalization problems.

Table 5: Results of AD3

	A	U	R
Random	486	14500	3.35%
CF	553	13837	4.00%
ANN	615	13837	4.44%
CGP-LS	661	13837	4.78%

Acknowledgement

This work was supported by Brain Korea 21 Project.

References

[1] *Email Marketing Maximized, Insight Report 2000.* Peppers and Rogers Group, 2000.

[2] M. S. Chen and P. S. Han, J. Yu. Data mining: An overview from a database perspective. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):866–883, 1996.

[3] R. Dewan, B. Jing, and A. Seidmann. One-to-one marketing on the internet. In *Proceedings of the 20th international conference on Information Systems*, pages 93–102, 1999.

[4] M. Goebel and L. Gruenwald. A survey of data mining and knowledge discovery software tools. *SIGKDD Explorations*, 1:20–33, 1999.

[5] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

[6] S. Haykin. *Neural Networks, A Comprehensive Foundation.* Prentice Hall, 1975.

[7] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 230–237, 1999.

[8] H. Iba and N. Nikolaev. Genetic programming polynomial models of financial data series. In *IEEE Conf. on Evolutionary Computation*, pages 1459–1466, 2000.

[9] A. F. James and M. S. David. *Neural Networks, Algorithms, Applications, and Programming Techniques.* Addison Wesley, 1994.

[10] J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordan, and J. Riedl. Grouplens: applying collaborative filtering to usenet news. *Communications of the ACM*, 40:77–87, 1997.

[11] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* The MIT Press, 1992.

[12] V. Podgorelec, P. Kokol, and J. Zavrsnik. Medical diagnosis predictions using genetic programming. In *Computer-based Medical Systems*, pages 202–207, 1999.

[13] J. Roughgarden. *Anolis Lizards of the Caribbean: Ecology, Evolution, and Plate Tectonics.* Oxford University Press, 1992.

Improving the Performance of a Genetic Algorithm for Minimum Span Frequency Assignment Problem with an Adaptive Mutation Rate and a New Initialization Method

Shouichi Matsui **Ken-ichi Tokoro**
Communication & Information Research Laboratory (CIRL)
Central Research Institute of Electric Power Industry (CRIEPI)
2-11-1 Iwado-kita, Komae-shi, Tokyo 201-8511, JAPAN
E-mail: {matsui,tokoro}@criepi.denken.or.jp, Phone: +81-3-3480-2111

Abstract

We propose a new Genetic Algorithm (GA) for solving the minimum span frequency assignment problem (MSFAP). The MSFAP is minimizing the range of the frequencies assigned to each transmitter in a region satisfying a number of constraints. The basic framework of the GA involves finding and ordering of the cells for use in a greedy (sequential) assignment process, and it also utilizes graph theoretic constraint to reduce search space. Because the performance of the GA heavily depends on the mutation rate, we use an adaptive mutation rate. And also a novel initialization method is utilized to improve the performance. Results are given which show that our GA produces optimal solutions to several practical problems, and the performance of our GA is far better than the existing GAs.

1 INTRODUCTION

The frequency assignment problem (FAP) is a very important problem today, but is a difficult, NP-hard problem. The radio spectrum is a limited natural resource used in a variety of private and public services, the most well known example would be found in cellular mobile phone systems. To facilitate this expansion the radio spectrum allocated to a particular service provider needs to be assigned as efficiently and effectively as possible.

The minimum span frequency assignment problem (MSFAP) requires assigning frequencies to a set of transmitters in such a way that given compatibility constraints, which model potential interference between pairs of transmitters, are satisfied. In addition to satisfying the constraints the objective is to minimize the span of the assignment, i.e., to minimize the difference between the largest and the smallest frequency used.

Because the MSFAP is a very important problem in the real world and an NP-hard problem, a number of heuristic algorithms have been proposed (e.g., Hurley et al., 1997), and genetic algorithm (GA) are applied to MSFAP (e.g., Crompton et al., 1994; Cuppini, 1994; Kim et al., 1995; Hurley and Smith, 1995; Lai and Coghill, 1996; Ngo and Li, 1998; Valenzuela et al., 1998; Matsui and Tokoro, 2000).

The performance of GA had been not good enough until the work by Valenzuela et al. of 1998. The GAs based on permutation showed good performance(Valenzuela et al., 1998; Matsui and Tokoro, 2000), but the performance heavily depends on parameters, especially mutation rate (Matsui and Tokoro, 2000). Therefore these GAs are still not easily applicable to the real world problems.

This paper proposes a new genetic algorithm that performs better than the existing GAs, and is easily applicable to the real world problems. The proposed algorithm is an improvement of the GA by Matsui and Tokoro (Matsui and Tokoro, 2000). It improves the performance by using an adaptive mutation rate mechanism and a novel initialization method. The core idea of the proposed GA is the same of that it is based on, i.e., it uses GA as a meta-heuristics for a greedy (sequential) assignment method. With the GA described here the iterative transformations are applied to permutations of cells. A simple sequential assignment algorithm is then applied to each of these permutations to produce an allocation of frequencies that does not violate any constraints.

The proposed GA is tested using a set of standard benchmark problems, and the performance is superior to the existing GAs. The proposed GA can obtain better solutions than the existing GAs. The GA finds the optimal solution with less iteration than existing GAs in cases where the previously proposed GAs can find the optimal solution.

2 MINIMUM SPAN FREQUENCY ASSIGNMENT PROBLEM

In this section, we briefly describe the problem of MSFAP. The survey papers (Hale, 1980; Katzeka and Naghshineh, 1996) provide good overviews of the frequency assignment problem. This is a very brief summary of these papers.

2.1 FREQUENCY ASSIGNMENT IN GENERAL

Any given radio bandwidth can be divided into a set of non-interfering radio channels. All such channels can be used simultaneously while satisfying an acceptable received radio signal and noise ratio, or while no interference occurs between a pair of transmitters. The frequency assignment problem (FAP) is to find an assignment that satisfies these conditions. Frequencies are used as channels, therefore frequency assignment is sometimes called *channel assignment*.

Channel assignment schemes can be divided into three categories, fixed channel assignment (FCA), dynamic channel assignment (DCA), and hybrid channel assignment (HCA). Each scheme has its merits and demerits (Katzeka and Naghshineh, 1996). Because FCA schemes are simple, many mobile communication systems are still designed based on the FCA, so we consider channel assignment algorithms for FCA in this paper.

2.2 INTERFERENCE AND CONSTRAINTS

Interference can occur between a pair of transmitters if strength of the the interfering signal is sufficiently high. Whether a transmitter pair has the potential to interfere depends on many factors, e.g., distance, terrain, power, or antenna design. The higher the potential for interference between a transmitter pair is, the larger the frequency separation is required. For example, if two transmitters are sufficiently geographically separated then a frequency can be re-used, i.e., the same frequency can be assigned. At the other extreme if two transmitters are located at the same site then they may require, say, five channels separation.

To model the constraints a compatibility matrix is constructed that provides the separations needed between each transmitter pair. This matrix is usually represented by a $n \times n$ matrix C (n is the number of transmitters in the network) where each element c_{ij} denotes the frequency separation between transmitters i and j, i.e., if f_i and f_j are the frequencies assigned to transmitter i and j respectively, then the following condition should be satisfied for all i and j,

$$|f_i - f_j| \geq c_{ij}.$$

2.3 FORMULATION

Let $X = \{x_1, x_2, \cdots, x_n\}$ be a set of cells in a cellular system. A demand vector on X is an n-vector $M = (m_i)$ with nonnegative integer elements. The element m_i represents the number of radio frequencies required for cell x_i. Radio frequencies are assumed to be evenly spaced, so they can be identified with the positive integers. A compatibility matrix on X is a symmetric $n \times n$ matrix $C = (c_{ij})$ with nonnegative integer elements. The value c_{ij} prescribes the minimum frequency separation required between frequencies assigned to cell x_i and cell x_j. If $c_{ij} = \nu$, cell x_i and x_j are said to be ν-compatible with each other, then, a triple $P = (X, M, C)$ characterizes FAP. For simplicity, we assume that the requirements of each cell are ordered.

A *feasible frequency assignment for P* will be a collection $F = (f_i)$ of positive integers, $i = 1, \cdots, n, k = 1, \cdots, m_i$, $j = 1, \cdots, n, l = 1, \cdots, m_j$, such that

$$|f_i^k - f_j^l| \geq c_{ij},$$

for all indices i, j, k, l (except for $i = j, k = l$), where f_i^l is the frequency assigned to the l-th requirement of cell x_i. The span $S(F)$ of a frequency assignment F is the difference between the largest frequency and the smallest frequency assigned to the system, i.e.,

$$S(F) = \max_{i,l} f_i^l - \min_{i,l} f_i^l.$$

The objective of the MSFAP is to find a feasible frequency assignment F with the minimum span $S_o(P)$, i.e., to find

$$S_o(P) = \min\{S(F') | \text{all feasible } F' \text{ for } P\}.$$

Without loosing generality, we can assume that $\min_{i,l} f_i^l$ is 1, therefore the MSFAP is the problem of finding the feasible assignment of which $\max_{i,l} f_i^l$ is minimum.

2.4 CONSTRAINT GRAPH

The frequency assignment problem can be stated as a generalized graph coloring problem (e.g., Hale, 1980). The transmitters are vertices of a graph and the frequencies can be regarded as a set of colors to be assigned to the vertices. The edge that connects vertices x_i and x_j is labeled $c_{ij}(> 0)$. If colors are numbered from 1, and a_i^k, a_j^l are the color assigned to vertex i and j respectively then they should satisfy the condition

$$|a_i^k - a_j^l| \geq c_{ij},$$

which is equivalent to the interference constraints.

A constraint graph G is a finite, simple, undirected graph with each edge labeled with an integer that corresponds to an element of the compatibility matrix. A clique in G is a maximal complete subgraph of G. The nodes in a clique cannot be assigned the same frequency.

A constraint graph and clique are important for calculating the lower bound of span (Hurley et al., 1997). They can also improve the performance of heuristic algorithms (Smith et al., 1998) and the performance of GA (Matsui and Tokoro, 2000).

2.5 PREVIOUS WORKS

Because MSFAP is an important and very difficult problem to be solved exactly, GA based algorithms have been proposed (e.g., Crompton et al., 1994; Cuppini, 1994; Kim et al., 1995; Hurley and Smith, 1995; Lai and Coghill, 1996; Ngo and Li 1998; Valenzuela et al., 1998; Matsui and Tokoro, 2000). The performance of the previous GAs that represent possible assignment directly as bit-string or sequence of integers are not good enough, and the permutation based GAs are reported to show good performance (Valenzuela et al., 1998; Matsui and Tokoro, 2000). Assignment order of transmitters is represented by permutations and assignment is carried out using a sequential algorithm. While the scheme has overcome the weakness of the previous two schemes and the performance has improved, there still remains the problem of parameter tuning (Valenzuela et al., 1998; Matsui and Tokoro, 2000).

Matsui and Tokoro showed that utilizing the graph theoretic constraints described in the previous section gave good performance. They utilized the fact that the transmitters in a clique in a constraint graph cannot share a frequency, and showed that assigning frequencies to the transmitters in clique first, then assigning to the remaining transmitters improved performance (Matsui and Tokoro 2000). Their GA showed good performance to a typical benchmark problems, but the parameter tuning, especially the mutation rate, was still future work.

3 NEW METHODS FOR PERFORMANCE IMPROVEMENT

3.1 CELL ORDER BASED REPRESENTATION

The previous works showed that permutation based GAs showed good performance compared with the other GAs, therefore we use the same scheme. But instead of the permutation of transmitters, we use the permutation of cells, because it makes the search space smaller.

Let n be the number of cells in the system, and $M = (m_1, \cdots, m_n)$ be the demand vector, then the permutation of cell order is a sequence $S = (s_1, \cdots, s_L)$, $L = \sum_{i=1}^{n} m_i$, and cell number i occurs m_i times in a permutation . The search space of permutation of transmitters and of cells are $(\sum_{i=1}^{n} m_i)!$, and $(\sum_{i=1}^{n} m_i)!/\Pi_{i=1}^{n}(m_i!)$ respectively. Therefore the search space can be reduced greatly by the factor of $\Pi_{i=1}^{n}(m_i!)$. This term is usually very large for the real world problems, therefore we use cell order based permutation for our GA.

3.1.1 New Crossover Operator

Because the chromosome representation used is a permutation of cell numbers, and cells have multiple channel demand, a permutation contains multiple occurrence of the same cell numbers. Therefore well-known crossover operator such as partially mapped (PMX), order (OX), and cycle (CX) crossover operators cannot be used straightforwardly.

We use a modified PMX for the crossover operator. The operator builds offsprings as described below. Let us consider the case with 3 cells and the demand is $M = (3, 2, 4)$ channels. In this case channel number 1 occurs three times, 2 occurs two times, and 3 occurs four times in a permutation. A subsequence of a permutation is selected by choosing two random cut points, which are similar to the cut points of 2-points crossover. For example, the two parents (with cut points marked by "|")

$$\begin{aligned} p_1 &= (111|3333|22) \\ p_2 &= (333|1121|32) \end{aligned}$$

would produce the two offspring as follows. First ordinal 2-point crossover is done, and the elements that occurs too many times are replaced with the symbol "x" (interpreted as "at present unknown"), where the marking starts from the head. After the 2-point crossover,

$$\begin{aligned} o_1 &= (111|1121|22) \\ o_2 &= (333|3333|32) \end{aligned}$$

are the offsprings, and after the marking,

$$\begin{aligned} o_1 &= (xxx|11x1|22) \\ o_2 &= (xxx|x333|32) \end{aligned}$$

are the offsprings. Then the marked positions are replaced the elements that occurs not enough times according to the order of the parents, e.g., the elements marked by x in the offspring o_1, o_2 are modified according to the ordering of p_1, p_2 respectively. The final offsprings are as follows:

$$\begin{aligned} o_1 &= (333|1131|22) \\ o_2 &= (112|1333|32). \end{aligned}$$

3.2 SELF-ADAPTIVE MUTATION RATE

The efficiency of GA can be seen to depend on two factors, namely the maintenance of a suitable working memory, and quality of the match between the probability density function generated and the landscape being searched. The first of these factors will depend on the choice of population size and selection algorithm. The second will depend on the action of the reproductive operators, i.e., crossover and mutation operators, and the set of associative parameters on the current population (Smith, 1998).

Matsui and Tokoro reported that the performance of their GA for MSFAP heavily depends on the mutation rate (Matsui and Tokoro, 2000). Our independent extensive parameter survey of mutation rate also showed that the key factor for improving the performance is the mutation rate. A good mutation rate gave 100% convergence to the optimal solution, but inappropriate one gave bad convergence rate. And the optimal rate varied across the problem type. With these observation, we have built a self-adaptive mutation rate mechanism into our algorithm.

3.2.1 Population Level vs. Individual Level

There has been considerable effort in finding optimal values for mutation rates and parameters for mutation distributions. Good survey is in the book by Michalewicz and Fogel (Michalewicz and Fogel, 2000). These effort can be categorized into the population level control and the individual level control.

The population level control mechanisms change (decrease) the mutation rate r_m over time, and they are similar to reducing the temperature in simulated annealing. The other kind of mechanism, individual level control, changes the mutation rate of each individuals in the population.

Our extensive numerical experiments showed that the individual level control gave better results, therefore we use an individual level control in our algorithm.

3.2.2 Proposed Mutation Scheme

The mutation rate r_m^i for each individual i is coded using Gray code. The range of mutation rate is set to $[r_m^L, r_m^U]$ and it is encoded into 6-bits Gray code. The values of r_m^L, r_m^U used in the numerical experiments will be discussed later.

In the mutation step, the mutation rate r_m^i is first modified according to the value of itself. The Gray coded bit string is modified with the mutation rate of r_m^i, and the result ${r_m^i}'$ is used as the mutation rate for the individual i. Smith reported that this scheme works well for a variety of problems (Smith, 1998).

In the mutation step, the mutation and the resulting offsprings are created as follows:

Initialize: Let o_1, o_2 be the offsprings generated by crossover, and r_m^1, r_m^2 be the mutation rate of o_1, o_2 respectively.

Change Mutation Rate Change mutation rate r_m^1, r_m^2 by mutating the bit-string of r_m^1, r_m^2 with the rate of r_m^1, r_m^2 respectively, and the results be ${r_m^1}', {r_m^2}'$.

Apply Mutation Apply mutation to offspring o_1 with the rate of ${r_m^1}', {r_m^2}'$ and the results be o_1^1, o_1^2. And apply mutation to offspring o_2 with the rate of ${r_m^1}', {r_m^2}'$ and the results be o_2^1, o_2^2. Individuals inherit the mutation rate that generated them.

Select Best: Select the best individual from $o_1^1, o_1^2, o_2^1, o_2^2$ by evaluating them.

3.3 A NEW INITIALIZATION METHOD

Little attention has been given to initializing evolutionary algorithms (Micheleicz and Fogel, 2000). The straightforward approach is to sample uniformly at random from the state space of possible solutions, thereby providing an unbiased initial population. But when solving real problems, we often know some information that would help us seed the initial population with useful hints. The work by Lai and Coghil (Lai and Coghil, 1996) and by Valenzuela et al. (Valenzuela et al. 1998) used heuristics to initialize population. They first generate random population, and then modify each individual using heuristics to improve the fitness. They have reported that their initialization methods improved the performance of their GAs.

Our approach is different from theirs in the sense that our method use GA to generate initial population. In a sense, our approach can be seen as a kind of two-stage GA. In the first stage, a GA with very small population evolves to very small number of generation. The individuals are used to build the initial population of the second stage.

3.3.1 Divide and Conquer

The GA for the first stage should be fast and should generate good individuals. To achieve this goal, we should reduce the complexity of the problem. The well known strategy to reduce complexity is *divide and conquer*. To apply the strategy, we divide the problem into small one as follows.

Let $M = (m_i), i = 1, \cdots, n$ be the demand vector of a given problem P, and m_{min} is the smallest element of M, then consider the set of problems of $P_i', i = 1, \cdots, m_{\text{min}}$ with the demand vector $M' = (\lceil m_i/m_{\text{min}} \rceil) = (m_1', \cdots, m_n')$. The problem P_i' is usually easier to solve, and the assignment sequence of $S_{P_1'}, \cdots, S_{P_{m_{\text{min}}}}$ would be a good candidate for the building blocks of the original problem P.

To solve the problem P_1', we use a GA based on the *average available frequencies (AFF)* heuristics (Yokoo and Hirayama, 2000). Let N_i^F be the number of free frequencies[1] of cell i, and N_i^A be the number of assigned frequencies to cell i, then the AFF_i of cell i is defined as $AFF_i = (N_i^F)/(m_i - N_i^A)$.

Let $U = (u_1, \cdots, u_l)(l = \sum_{i=1}^n m_i')$ be a real-coded chromosome with each element takes the value of $[0, 1)$. Then the assignment order is evaluated based on the value of $v_i = AFF_i \times u_i$, i.e., the cell i of the smallest v_i among the cells with $m_i > N_i^A$ is assigned one by one. The assignment sequence generated by solving the problem P_1' is used as the assignment order of the remaining problems $P_2', \cdots, P_{m_{\text{min}}}'$. The GA evolves the population of U to find good assignment order. The uniform crossover, the adaptive mutation rate mechanism, and flip mutation are used in the first stage GA. The GA evolves until iteration reaches to the maximal number of generations, or it terminates if the optimal assignment is found, or the variance of the population decreases to zero. The number of population N_1 and the maximum number of generations are set to be small. In the performance test, they are set to 125 (1/4 of the number of population in the second stage) and 20 respectively.

3.3.2 Initialization for the Second Stage

Let N_1 be the number of population in the first stage GA, and the $S_i = (s_i^1, \cdots, s_i^l)(i = 1, \cdots, N_1)$ be the assignment sequences generated by the first stage, and N_2 be the number of population in the second stage. And also let $g_i = (g_i^j, \cdots, g_j^L)$ be the chromosome of an individual i. Then g_i is initializes as follows.

For $i \leq N_1$, S_i is used m_{min} times to initialize the g_i, i.e., g_i is initialized as the repeated sequence of S_i, namely

---

[1] *free frequency* means that is can be assigned without violating any constraints.

$g_i = (S_i, \cdots, S_i)$. For $i > N_1$, randomly chosen block of S_j is used to initialize a part of g_i, i.e., $g_i = (S_{j_1}, \cdots, S_{j_{m_{\min}}})$, where j_k is a randomly chosen integer in the range of $[1, N_1]$. Then cell numbers that occurs too many times than the demand are deleted from the permutation. And finally genes are grouped into two blocks, into cells that belong to the maximal clique in the constraint graph, and into others.

4 THE PROPOSED ALGORITHM

We propose a new algorithm for the MSFAP based on genetic algorithm. The scheme is the same as that of the GA by Matsui and Tokoro (Matsui and Tokoro, 2000), but to improve the performance, we use the cell order based permutation, the adaptive mutation rate mechanism, and the initialization method that are described in the previous section.

The simple genetic algorithm (GA) used here is an example of 'steady state', overlapping populations GA. The proposed GA is outlined in Figure 1.

4.1 GROUPING

We divide cells into 2 groups, i.e., into a cells in the maximal clique Q and others using the constraint graph G for simplicity and because the previous work reported that grouping into 2 blocks gave good results (Matsui and Tokoro, 2000).

4.2 SEQUENTIAL ASSIGNMENT ALGORITHM

Sequential assignment methods for the MSFAP are identical to the way the frequencies might be assigned manually. The simplest assignment technique is to assign the smallest acceptable frequency, i.e., the smallest frequency that can be assigned without violating any constraints. This greedy algorithm is the same one as Matsui and Tokoro used (Matsui and Tokoro, 2000).

4.3 FITNESS, OBJECTIVE FUNCTIONS

The objective function F used in the GA is defined as,

$$F = \{C_1 f_{\text{fmax}} - C_2 f^{\text{end}} - (B_1 P_1 + B_2 P_2 + P_3)\} / f_{\text{fmax}},$$

where C_1, C_2, B_1, B_2 are constants, f_{fmax} is the allowable maximum frequency number, f^{end} is the span obtained. The terms P_1, P_2 and P_3 are a type of penalty, and defined as,

$$\begin{aligned} P_1 &= \begin{cases} f^{\text{clq}} - f_{\text{fmax}} - A & f^{\text{clq}} > (f_{\text{fmax}} + A) \\ 0 & \text{otherwise,} \end{cases} \\ P_2 &= \begin{cases} f^{\text{end}} - f_{\text{fmax}} - A & f^{\text{end}} > (f_{\text{fmax}} + A) \\ 0 & \text{otherwise,} \end{cases} \\ P_3 &= \sum_{i \in X, f_i^k > f_{\text{fmax}}} (f_i^k - f_{f_{\text{fmax}}}) \end{aligned}$$

where f^{clq} is the span when the assignment to transmitters that belong to the clique is finished, and A is a constant for the threshold.

We use sigma truncation as the scaling function. The fitness function F' is defined as,

$$F' = F - (\mu_F - 2 \times \sigma_F)$$

where μ_F is the average, and σ_F is the standard deviation over the population.

The GA evolves population to maximize the F'.

4.4 SELECTION, CROSSOVER, AND MUTATION

We use roulette wheel selection, the modified PMX crossover, and swap mutation in the proposed GA. The crossover and the mutation do not generate invalid chromosome, i.e., offsprings are always valid representation of assignment ordering.

5 PERFORMANCE OF THE GA

5.1 TEST PROBLEMS

In this section, we demonstrate the performance of the proposed algorithm by considering the the same problems used in the paper by Matsui and Tokoro (Matsui and Tokoro, 2000). The examples are based on the so-called *Philadelphia problem* and were subsequently used by several authors (Gamst, 1986; Kim et al., 1995; Lai and Coghill, 1996; Hurley and Smith, 1995; Ngo and Li, 1998, Valenzuela et al., 1998). These problems are based on a cellular phone system consisting of 21 cells. The demands in each cell define the number of frequencies that need to be assigned to each of the cells. The distance between the cell centers is assumed to be 1, and all transmitters in each cell are assumed to be located at the center. The hexagonal geometry is given in Figure 2. The constraints between transmitters are generated by considering the distance between the transmitters.

Table 1 defines the Philadelphia variations used as the test problem. In Table 1, d_k denotes the smallest distance between transmitters which can use a separation of k channels, N denotes the total number of frequencies that need to be assigned, i.e., the number of transmitters and C denotes the total number of compatibility constraints that need to be satisfied

5.2 PARAMETERS

For the GA the population size (N_2) is 500, i.e., the same number as in the paper (Matsui and Tokoro 2000). The crossover ratio (P_c) is 0.8, and the number of new offsprings (N_n) is 500. The GA terminates if the span is less than or equal to the given lower bound, or terminates after 1000 generations have elapsed with no improvement to the

```
Procedure Divide
BEGIN
    Find the maximum clique $C$ of the constraint graph $G$.
    Let nodes in $C$ belong to the partition $P_1$, and the other nodes in $G$ belong to the partition $P_2$.
END
Procedure Adaptive-Mutation
BEGIN
  Initialize: Let $o_1, o_2$ be the offsprings, and $r_m^1, r_m^2$ be the mutation rate of $o_1, o_2$ respectively.
  Change Mutation Rate Change mutation rate $r_m^1, r_m^2$ by mutating the bit-string of $r_m^1, r_m^2$ with the rate of $r_m^1, r_m^2$ respectively,
    and the results be ${r_m^1}', {r_m^2}'$.
  Apply Mutation Apply mutation to offspring $o_1$ with the rate of ${r_m^1}', {r_m^2}'$ and the results be $o_1^1, o_1^2$. And apply mutation to offspring
    $o_2$ with the rate of $m_1', m_2'$ and the results be $o_2^1, o_2^2$. Individuals inherit the mutation rate that generated them.
  Select Best: Select the best individual from $o_1^1, o_1^2, o_2^1, o_2^2$ by evaluating them.
END
Procedure GA
BEGIN
  Initialize:
    Divide transmitters into 2 partitions using Divide.
    Generate $N_2$ random individuals ($N_2$ being the population size) by First-Stage GA, and set $r_m$ randomly.
    Produce $N_2$ assignments and store each one.
    Store best-so-far.
    If the span of the best-so-far is equal to the lower bound then terminate the algorithm.
  LOOP
    Generate offsprings: Generate $N_n$ offsprings.
      1. Select two parents by roulette wheel rule;
      2. Apply crossover with the probability of $P_c$, and generate two offsprings; when no crossover is applied, then two parents
         will be offsprings;
      3. Apply adaptive mutation to offsprings and generate one offspring by Adaptive-Mutation;
      4. If the offspring is better than best-so-far then replace best-so-far.
      5. If the span of the best-so-far is equal to the lower bound then terminate the algorithm.
    Selection: Select best $N_2$ individuals from the pool of old $N_2$ and new $N_n$ individuals.
  UNTIL stopping condition satisfied.
  Print best-so-far.
END
```

Figure 1: Outline of the proposed algorithm.

best-so-far, or when generation number reaches the upper bound. The upper bound is set to 9000 for all problems. The constants are set to $C_1 = 30, C_2 = 15, B_1 = 10, B_2 = 20, A = 3$ in the test, and f_{fmax} is set to the lower bound reported on the Web page (Eisenblätter Koster, 2000).

The lower and upper bound of the mutation rate are set to 0.005 and 0.05 for the problem P1, P2, P5, P6, P7, and P8, and 0.02 and 0.05 for the problem P3 and P4. This setting is based on the fact that larger lower bound gave good performance for the problems with higher channel density. The channel density is defined as $c_D = L/f_{\text{fmax}}$ where L is the total channel demand in the system, and f_{fmax} is the allowable maximum frequency number. Our extensible experiments showed that the larger lower bound gave good performance for the problems with $c_D > 3/2$.

5.3 RESULTS

We tested the performance of the GA with regard to the eight problems by running 100 times. The results are shown in Table 2. Comparisons with the results by Matsui and Tokoro (Matsui and Tokoro, 2000) are also given. In Table 2, the columns labeled *best* show the best span obtained, numbers in parentheses show the number of assignment tested, the columns labeled *mean* shows the average of spans, the columns labeled *ratio(%)* show the ratio of the obtained span is equal to the lower bound.

The table shows that our GA performs well, it found the optimal solution for all problems. It outperforms the previous GA (Matsui and Tokoro, 2000) for all cases, the span obtained by our GA is less than or equal to that obtained by the previous GA.

Table 1: Philadelphia problem variations.

Problem	d_0	d_1	d_2	d_3	d_4	d_5	Cell demands	N	C
P1	$\sqrt{12}$	$\sqrt{3}$	1	1	1	0	M	481	97,835
P2	$\sqrt{7}$	$\sqrt{3}$	1	1	1	0	M	481	76,979
P3	$\sqrt{12}$	$\sqrt{3}$	1	1	1	0	M_2	470	78,635
P4	$\sqrt{7}$	$\sqrt{3}$	1	1	1	0	M_2	470	56,940
P5	$\sqrt{12}$	$\sqrt{3}$	1	1	1	0	M_3	420	65,590
P6	$\sqrt{7}$	$\sqrt{3}$	1	1	1	0	M_3	420	44,790
P7	$\sqrt{12}$	$\sqrt{3}$	1	1	1	0	M_4	962	391,821
P8	$\sqrt{12}$	2	1	1	1	0	M	481	97,835

M = (8,25,8,8,8,15,18,52,77,28,13,15,31,15,36,57,28,8,10,13,8)
M_2 = (5,5,5,8,12,25,30,25,30,40,40,45,20,30,25,15,15,30,20,20,25)
M_3 = (20,20,20,20,20,20,20,20,20,20,20,20,20,20,20,20,20,20,20,20,20)
M_4 = (16,50,16,16,16,30,36,104,154,56,26,30,62,30,72,114,56,16,20,26,16)

Table 2: Results for Philadelphia problem variations (100 runs).

Problem	Lower bound	GA by Matsui and Tokoro			Proposed GA		
		best	mean	ratio(%)	best	mean	ratio(%)
P1	426	426 (21,500)	426.00	100	426 (8,340)	426.00	100
P2	426	426 (15,500)	426.00	100	426 (3,654)	426.00	100
P3	257	257 (778,000)	258.39	14	257 (330,688)	257.00	100
P4	252	252 (68,500)	252.33	68	252 (142,008)	252.00	100
P5	239	239 (386,000)	239.73	32	239 (2*)	239.00	100
P6	179	194 (312,500)	195.79	0	179 (8*)	179.00	100
P7	855	855 (51,500)	855.00	100	855 (25,252)	855.00	100
P8	524	524 (373,500)	524.51	64	524 (14,469)	524.00	100

Notes: (1) * denotes the best assignment was found in the initialization phase,
(2) numbers in parentheses indicate the minimum number of assignments tested to obtain the optimal assignment.

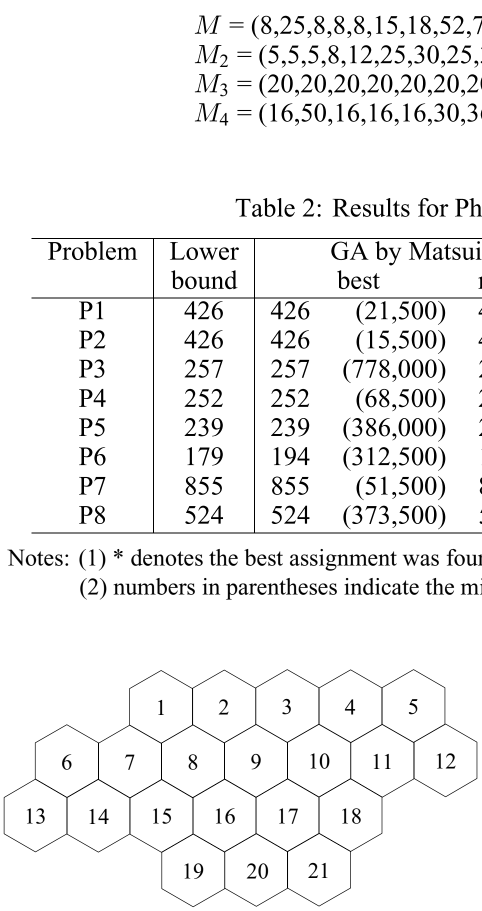


Figure 2: Cellular geometry of test problems.

The minimum number of generations, or assignments tested in iteration, to reach optimal solutions are smaller except for the case P4. For the cases where the minimum number of assignments tested is larger (P4), the convergence ratio to the optimal solution is better than the previous GA. For the problem P5 and P6, the best solutions were found in the initialization phase. The best span of P6 is reduced from 194 to 179, to the lower bound. The span of 194 is the known best one obtained by GAs (Matsui and Tokoro, 2000).

The computation time is roughly proportional to the number of assignments tested (N_t) for the both algorithms, and the N_t of the proposed GA is smaller than that of GA by Matsui and Tokoro in average, therefore the proposed GA is faster than the existing GAs.

Valenzuela et al. reported that a tabu-search based algorithm can obtain the lower bound for the problem P1, P2, P3, P4, P5, P7, and P8 if *critical subgraph* of original constraint graph is identified and a minimum span assignment for this subgraph initially found and this assignment is used as the starting (partial) assignment for the complete problem (Valenzuela et al., 1998). The performance of our GA to the seven problems are same, i.e., our GA can find the optimal solutions to the seven problems, and it can find the optimal assignment for the problem P6.

It should be emphasized that the results were obtained without tuning the mutation rate that is the key parameter for performance, therefore we can say that the proposed GA is more suitable for the real world problems than the existing GAs. With these observations, we can conclude that the performance of the proposed GA is better than the previously reported GAs.

5.4 DISCUSSION AND FUTURE WORK

The proposed GA performed well against the previously proposed GAs. It could attain the lower bounds, i.e, it could find the optimal solutions to all problems with higher probability. The performance improvement in terms of the number of evaluations is significant for the problems P1, P2, P5, P6, P7, and P8.

The experiments showed that the adaptive mutation scheme improved the convergence rate for all problems tested. They also showed that the new initialization methods decreased the number of fitness evaluation for the problem P1, P2, P3, P5, P6, P7, and P8 greatly, but it did not work well for the problem P4. More sophisticated chromosome representation and GA operators would overcome the problems, so we will investigate these. The *Set-Partition Crossover; SPX* developed for job-shop problems (Shi et al., 1996) is a promising candidate. And we will also investigate the factors that determine the complexity of problems.

6 CONCLUSIONS

We have presented here a new genetic algorithm that computes minimum span frequency assignments. The algorithm uses the GA as a meta-heuristics to determine the assignment order for the greedy algorithm, and it also utilizes graph theoretic constraints to reduce search space. And to improve the performance, an adaptive mutation rate mechanism and a new initialization methods are developed.

The proposed GA is tested using a set of standard benchmark problems, and the performance is superior to the existing GAs. The proposed GA can obtain the optimal solutions that were unable to be found using the existing GAs. The GA finds the optimal solution with less iteration than existing GAs in cases where the previously proposed GAs can find the optimal solution, and the probability of finding the optimal assignment is higher than the existing GAs. In addition, its performance is good enough compared with tabu-search based algorithm, it can find the optimal solutions that tabu-search can find.

We believe that our approach, the adaptive mutation rate mechanism and the initialization method by small GA with divide and conquer, can be applied to other real world problems.

References

Crompton, W., S. Hurley, and N. M. Stephens (1994). Applying genetic algorithms to frequency assignment problems, *Proc. SPIE Conf. Neural and Stochastic Methods in Image and Signal Processing*, vol.2304, pp.76–84.

Cuppini, M. (1994). A genetic algorithm for channel assignment problems, *Eur. Trans. Telecommun.*, vol.5, no.2, pp.285–294.

Eisenblätter, A. and A. Koster (2000). FAP web — a website about frequency assignment problems, http://fap.zib.de/, last modified July 17, 2000.

Gamst, A. (1986). Some lower bound for a class of frequency assignment problems, *IEEE Trans. Veh. Technol.*, vol.35, no.1, pp.8–14.

Hale, W. K. (1980). Frequency assignment: theory and applications, *Proc. of IEEE*, vol.68, no.12, pp.1497–1514.

Hurley, S. and D. H. Smith (1995). Fixed spectrum frequency assignment using natural algorithms, *Proc. of Genetic Algorithms in Engineering Systems: Innovations and Applications*, pp.373–378.

Hurley, S., D. H. Smith, and S. U. Thiel (1997). FASoft: a system for discrete channel frequency assignment, *Radio Science*, vol.32, no.5, pp.1921–1939.

Katzeka, I. and M. Naghshineh (1996). Channel assignment schemes for cellular mobile telecommunication systems: a comprehensive survey, *IEEE Personal Commun.*, vol.3, no.3, pp.10–31.

Kim, J.-S., S. H. Park, P. W. Dowd, and N. M. Nasrabadi (1995). Comparison of two optimization techniques for channel assignment in cellular radio network, *Proc. of IEEE Int. Conf. Commun.*, vol.3, pp.850–1854.

Lai, K. W. and G. G. Coghill (1996). Channel assignment through evolutionary optimization, *IEEE Trans. Veh. Technol.*, vol.45, no.1, pp.91–96.

Matsui, S. and K. Tokoro (2000). A new genetic algorithm for minimum span frequency assignment using permutation and clique, *Proc. of GECCO-2000*, pp.682–689, 2000.

Michalewicz, Z. and D.B. Fogel (2000). *How to solve it: modern heuristics*, Springer-Verlag.

Ngo, C. Y. and V. O. K. Li (1998). Fixed channel assignment in cellular radio networks using a modified genetic algorithm, *IEEE Trans. Veh. Technol.*, vol.47, no.1, pp.163–172.

Shi, G., H. Iima and N. Sannomiya (1996). A new encoding scheme for solving job-shop problems by genetic algorithm, *Proc. of 35th IEEE Conf. on Decision and Control*, vol.4, pp.4395–4400.

Smith, D. H., S. Hurley, and S. U. Thiel (1998). Improving heuristics for the frequency assignment problem, *Eur. J. Oper. Res.*, vol.107, no.1, pp.76–86.

Smith, J. E. (1998). Self adaptation in evolutionary algorithms, Ph.D thesis, Univ. of the West England, Bristol.

Yokoo, M. and K. Hirayama (2000). Frequency assignment for cellular mobile systems using constraint satisfaction techniques, *Proc. of IEEE VTC2000*, pp.888–894.

Valenzuela, C., S. Hurley, and D. Smith (1998). A Permutation based algorithm for minimum span frequency assignment, *Proc. 5th International Conference on Parallel Problem Solving from Nature—PPSN V*, Amsterdam, pp.907–916.

Protein Structure Prediction with EA Immunological Computation

Steven R. Michaud[1], Jesse B. Zydallis[1], Gary B. Lamont[1], Paul K. Harmer[2], and Ruth Pachter[3]
[1]Department of Electrical and Computer Engineering, Graduate School of Engineering and Management
Air Force Institute of Technology WPAFB OH 45433 {steven.michaud, jesse.zydallis, gary.lamont}@afit.edu
[2]Sensors Directorate, Air Force Research Laboratory, WPAFB OH 45433, paul.harmer@embedded.hpc.mil
[3]Materials and Manufacturing Directorate, Air Force Research Laboratory
WPAFB OH 45433, Ruth.Pachter@wpafb.af.mil

Abstract

Considerable research has been presented to develop a generalized technique to predict a polypeptide's molecular structure given its amino acid sequence. This is also known as the Protein Structure Prediction (PSP) problem which has direct applications to many scientific, medical, and engineering disciplines. Previous research with Evolutionary Algorithms (EAs) to minimize the empirical CHARMM protein energy model and generation of the associated protein structure is extended using the fast messy genetic algorithm improved through the use of secondary protein structure information integrated with artificial immune system concepts. Good statistical results using historical metrics as well as a new spatial/temporal metric are obtained, thereby making the modified memetic algorithm a viable option for solving the PSP problem.

1 INTRODUCTION

This paper presents a stochastic, immunity-based, approach to solving the protein structure prediction (PSP) problem through the integration of secondary structure information and an artificial immune system (AIS) built upon a specific evolutionary algorithm (EA), the fast messy Genetic Algorithm (fmGA). The original fmGA was developed by Goldberg, Deb, Kargupta, and Harik [5] and later modified and applied to the PSP problem [11]. The fmGA is a Genetic Algorithm (GA) that explicitly utilizes building blocks to solve optimization problems. Previous research has shown favorable results in applying the fmGA to the pentapeptide [Met]-Enkephelin [12]. Additional work was completed in applying the fmGA to a larger polyalanine model and again favorable results were obtained [13]. This work also included the use of additional domain information in the form of secondary structure information to obtain "good" solutions.

This paper focuses on incorporating immunological concepts into the fmGA. This is accomplished through the seeding of the population. The secondary structure input with fmGA search results are analyzed to determine what affects they have on the efficiency and effectiveness of the algorithm. The background information on the PSP problem is presented in the next section, followed by information on the Artificial Immune System, and a description of the improved fmGA search algorithm. The immunological seeding mechanism is then presented, followed by the design of experiments and a discussion of the results and conclusions.

2 PROTEIN STRUCTURE

A common method employed in protein structure prediction, known as energy minimization, searches a protein's conformational search space for an energy minimum. Observe that protein secondary structure refers to the backbone dihedral angle values, whereas the tertiary structure (the conformational structure) refers to the completely folded molecule including its side chains. PSP is a challenging optimization problem for a number of reasons; first, the conformational space is highly dimensionalized; second, the energy fitness function is computationally expensive; and finally, the landscape contains a very large number of local minima. In particular, a peptide molecule contains $3N_A - 6$ degrees of freedom, where N_A is the number of atoms contained in the molecule. Even relatively small proteins contain thousands of atoms and it is not uncommon to encounter proteins containing hundreds of thousands of atoms.

The ability to accurately predict the potential energy for a particular protein represented by a set of dihedral angles depends highly on the method employed to calculate the associated energy. An exact model is too computationally expensive and results in an algorithmic complexity of $\mathcal{O}(N_A^5)$ [12], thus most researchers utilize other models that have less complexity typically of only $\mathcal{O}(N_A^2)$.

The CHARMM energy model used in this research considers contributions due to non-bonded van der Waals interactions (represented by the Lennard-Jones potential), Coulomb's law, and bonded interactions. It does not consider solvent interactions. The protein energy conformation is given by

$$\begin{aligned} E = & \sum_{(i,j)\in\mathcal{B}} K_{r_{ij}}(r_{ij}-r_{eq})^2 \\ + & \sum_{(i,j,k)\in\mathcal{A}} K_{\Theta_{ijk}}(\Theta_{ijk}-\Theta_{eq})^2 \\ + & \sum_{(i,j,k,l)\in\mathcal{D}} K_{\Phi_{ijkl}}\left[1+cos(\Phi_{ijkl}-\gamma_{ijkl})\right] \\ + & \sum_{(i,j)\in\mathcal{N}} \left[\left(\frac{A_{ij}}{r_{ij}}\right)^{12}-\left(\frac{B_{ij}}{r_{ij}}\right)^{6}+\frac{q_iq_j}{\epsilon r_{ij}}\right] \quad (1) \end{aligned}$$

where $\mathcal{B}$ is the set of bonded atom pairs,
$\mathcal{A}$ is the set of atom triples defining dihedral angles,
$\mathcal{D}$ is the set of atom 4-tuples defining dihedral angles,
$\mathcal{N}$ is the set of non-bonded atom pairs,
r_{ij} is the distance between atoms i,j,
Θ_{ijk} is the angle formed by atoms i,j,k,
Φ_{ijkl} is the dihedral angle formed by atoms i,j,k,l,
q_i is the partial atomic charges of atom i,
the $K_{r_{ij}}$'s, r_{eq}'s, $K_{\Theta_{ijk}}$'s, Θ_{eq}'s, $K_{\Phi_{ijkl}}$'s, γ_{ijkl}'s $\mathcal{A}_{ij}$'s, $\mathcal{B}_{ij}$'s, and ϵ are empiric constants.

The empirical parameters are derived from the parameter files of the molecular modeling software CHARMM version 22.0 [14], as described by Brinkman, et. al. [3]. Other examples of force field energy models that could be used are AMBER and ECEPP [10]. The CHARMM energy function was chosen for the large number of parameters it contains for organic molecular systems.

3 ARTIFICIAL IMMUNE SYSTEMS

There are several computational techniques that look to biology for inspiration. Some common examples include neural networks, evolutionary algorithms, and artificial immune systems (AISs) or immunological computation. The biological immune system (BIS) has been the target of considerable research interest in the medical community from which several theories of system behavior have been developed with the hope of improving human life. The natural use of an AIS is in the field of computer security and they have has been applied to areas such as computer viruses [6, 9] and also intrusion detection [2, 8]. But, immunological computation has also been applied to other problem domains. Some examples include multi-optimization problems, anomaly detection in time series data, fault diagnosis, loan application fraud detection, and robust scheduling [6]. The similarity in all of these applications is that they utilize the pattern matching and "learning" mechanisms of the immune system model to perform a stochastic search of a large space.

Our approach captures several BIS features and infuses them within the fmGA to improve our PSP algorithm. This is similar to the methodology used by Hart, Ross, and Nelson [7] to enhance a GA with immunity concepts. Their methodology used a set of antibodies, generated from a gene library, in order to produce schedules. These schedules can account for changing conditions, in essence producing multiple schedules and binding the best one to the prevailing conditions. Our approach is similar to the BIS features of antibodies, antigens, gene libraries, and affinity maturation and utilizes the fmGA building block approach to improve the stochastic search process for protein structures.

3.1 BIOLOGICAL IMMUNE SYSTEM

The Biological Immune System (BIS) is composed of many different types of cells that are deployed in great numbers. The result is a defense framework for the protection of internal resources against foreign invaders, or pathogens. These components act in concert after the recognition of an *antigen* [1].

An *antigen* (*pathogen*) is any substance that can stimulate the immune system [1]. The most common antigens are found in bacteria and viruses. These molecules have multiple surface reaction sites that act as interaction points for the immune system's B cells [1].

B cells are studded with many "Y" shaped detectors called *antibodies.* These antibodies are generated from cell bodies. Parts of the antibody chains expose small patches on their surface, which make them highly specific antigen binders [1]. Only those *activated* antibody-antigen bindings with a large enough affinity result in an immune reaction. Once this affinity is exceeded, the B-cells divide to produce clones. These clones further undergo *hypermutation* in which they experience high mutation rates [8]. This creates daughter cells that are a little bit different than the parent with the goal of adapting to a specific antigen [8]. This works within a Darwinian selection process known as *affinity maturation.* Those B cells that better match the antigen divide, while those B cells that do not match soon die. Through this process, the remaining cells have a higher affinity for matching pathogen features that may represent similar antigens.

The diversity of antibodies required to protect the body would require more than 100 million genes. This is approximately 10% of a human's genetic material. However, through a unique combinatoric process, antibodies are generated from a library of only about 2,000 gene fragments. These BBs are further diversified though hypermutation. Note that T-cells, an integral element of the BIS, are not used here.

4 THE FAST MESSY GENETIC SEARCH ALGORITHM (fmGA)

The fmGA is an approach that exploits "good" Building Blocks (BBs) in solving optimization problems. These BBs represent "good" information in the form of partial binary strings that can be combined to obtain even better solutions. The BB approach is used in the fmGA to increase the number of "good" building blocks that are present in each subsequent generation of the algorithm. The fmGA was chosen due to its reduced computational requirements over the messy GA's Partially Enumerative Initialization Phase [5] and the fact that it explicitly manipulates BBs. The fmGA algorithm executes in three phases, the *Initialization Phase*, the *Building Block Filtering Phase*, and the *Juxtapositional Phase* [11].

The algorithm begins with the Probabilistically Complete Initialization Phase. This phase randomly generates a user specified number of population members. These population members are of the a priori specified chromosome length and each is evaluated to determine its respective fitness value. Our implementation utilizes a binary scheme in which each bit is represented with either a 0 or a 1 and the CHARMM energy model is used to calculate each string's fitness value.

```
Generate Competitive Template
    Conduct Local Search on Template
For epoch = 1 to $j$
    For n = 1 to $k$
    // Initialization Phase
        Perform Probabilistically Complete Initialization
        Evaluate Each Population Member's Fitness (w.r.t. Template)
    // Building Block Filtering Phase
        For i = 1 to Maximum Number of Building Block Filtering Generations
            If (Building Block Filtering Required Based on Input Schedule)
                Then Randomly Delete Specified Number of Bits From Each Individual
            Else
                Perform Tournament Thresholding Selection
            Endif
        End Loop
    //Juxtapositional Phase
        For i = 1 to Maximum Number of Juxtapositional Generations
            Cut-and-Splice
            Evaluate Each Population Member's Fitness (w.r.t. Template)
            Perform Tournament Thresholding Selection
        End Loop
    End Loop
End Loop
```

Figure 1: Pseudo Code for fmGA

The Building Block Filtering (BBF) Phase follows and consists of BBF alternated with a selection operator. The BBF process randomly deletes locus points and their corresponding allele values in each of the population members' chromosomes. This process completes once the length of the population member's chromosomes have been reduced to a predetermined BB size. In order to evaluate the population member's fitness values a competitive template is utilized to fill in the missing allele values [5]. This competitive template is initially generated randomly followed by a local search operation to improve the quality of the template. In subsequent generations, this template contains the allele values of the best found string in the population from the previous BB.

Through the BBF phase the length of the chromosome decreases but each chromosome must continue to be evaluated for selection purposes. During this phase these chromosomes are referred to as "underspecified" since each locus position does not have an associated allele value. To evaluate an underspecified population member, the member is overlayed upon the competitive template to fully specify the member. This process uses the allele values from the template to fill in any missing allele values in the population member to allow the fitness evaluation to take place and is repeated any time an underspecified population member needs to be evaluated. The BBF process is alternated with a selection mechanism to keep only the strings with the "best" building blocks found, or those with the best fitness value. A critical input schedule is used to specify the number of generations to execute each phase and the exact generations upon which BBF and selection occur. The resulting BBs for a given length reflect the "gene expression" for the specific protein.

The juxtapositional phase follows and uses the building blocks found through the BBF phase and recombination operators to create chromosomes that are fully specified. A chromosome is referred to as fully specified if it is not missing any locus positions, or in other words does not need to use the competitive template for evaluation. The recombination operation may result in overspecified strings which are strings containing multiple allele values for the same locus position. In this case a left to right priority is used. This scheme takes the first allele value encountered for any locus and uses that value for evaluation purposes even if subsequent values for the same locus appear in the string. Recombination is alternated with a binary thresholding tournament selection operator to keep the best solutions found in the population. Upon completing of the juxtapositional phase, the best population member found becomes the new competitive template.

Following the Juxtapositional Phase the algorithm increments the BB size by one and repeats the three phases. The completion of the three phases constitutes one BB size or one era. If there are no more eras to execute, the algorithm restarts using the first BB size and the best found individual from the previous era as the competitive template and repeats the whole process again. One repetition of all the eras constitutes an epoch. Once all of the user specified number of epochs complete, the best population member is presented as the final solution. The determination of BB size range, epoch size, and input schedule were derived originally from [5] and interpolated to the current values to work with larger problem sizes [11, 12].

4.1 MAPPING BIOLOGY TO EA IMMUNOLOGICAL COMPUTATION

The features of the BIS make it an attractive model for the stochastic search of a large space. Our approach is to place our existing fmGA integrated with AIS ideas, based on a BIS model, in order to enhance its effectiveness for protein structure prediction. The generalized BIS to fmGA mapping follows:

The BIS in part maps to an AIS that performs a PSP secondary structure search. A valid protein structure is equivalent to an antigen in this mapping, as that is the intermediate goal of the secondary structure search. The search for antigen in our AIS becomes a search for binary strings representing a set of secondary structure dihedral angles. Each antibody is associated with a single string of protein dihedral angles. This would be a member of a population in a standard GA. The antibodies are made up of strings of building blocks within the fmGA. This BB set is representative of the BIS antibody gene fragment library. The binding of antibodies to antigen in our system is completed through the evaluation of each antibody by the CHARMM energy model. The relative affinity of antigen-antibody binding is the energy returned by the CHARMM model. An affinity maturation of the antibody strings is performed over time through the use of the fmGA operators, whereby the population of antibodies is evolved in the direction of improved protein structures with appropriate secondary structure. For the PSP problem, elements of the protein dihedral angles are defined by the fmGA BBs. The fmGA objective function is associated with the CHARMM energy function. The following section provides the details of the BB variation using protein SSI and HCI methods associated with secondary structure seeding.

fmGA Init Phase
Gene Libraries ⟶ Random/Seeded BBs in individual strings
Hypermutation ⟶ BB Variation

fmGA BBF Phase
Gene Library ⟶ BBs from Init Phase
Antibody (AB) ⟶ BB + Template/String
Set of BBs ⟶ fmGA BBF Pop
AB-AG Binding ⟶ string by def.
B-cell Affinity ⟶ BB fitness eval
Affinity Maturation ⟶ BB Selection

fmGA Juxt Phase
Gene Library ⟶ BBs from BBF Phase
Antibody ⟶ Recombined BBs
Set of Antibodies ⟶ JuxtaPositional Pop
AB-AG Binding ⟶ String constraints met
Antigen (AG) ⟶ valid solution
B-Cell Affinity ⟶ fitness value
Affinity Maturation ⟶ Best Solution Selected

5 IMMUNOLOGICAL SEEDING OF SECONDARY STRUCTURE

The ability to predict a protein's three-dimensional structure or conformation from its one-dimensional amino-acid sequence is a significant problem. Often the prediction of the secondary structure has been used as a precursor to finding a "good" tertiary structure [4, 15] and the prediction of the secondary structure is considered to be an important step towards predicting the three-dimension structure of the protein [15]. This fact has lead us to use secondary structure information as a seeding mechanism in the effort to incorporate immunological concepts into the fmGA. The modified fmGA algorithm attempts to seed the initial population, for each BB size, with either secondary structure dihedral angle information and/or locally optimized hill climbed population members. Both methods were chosen with the overall intent of incorporating "good" BBs within a population member.

In the normal execution of the fmGA code, the algorithm randomly initializes the population prior to the execution of each era. Through the BBF phase these random population members are reduced in length and eventually become "good" BBs. The modified fmGA attempts to improve the results of this process by injecting potentially “good” BBs into the initial population through the seeding of population members that are derived from known secondary structures or from the use of locally optimized population members.

The Secondary Structure Initialization (SSI) method utilizes user defined optimal secondary structure angles with some $+/-$ threshold [1] to generate population members with the corresponding angular values. In order to ensure generality, the algorithm allows for the inclusion of all known secondary structures and their associated dihedral angular values. For this paper only two secondary structures were considered, α-helix and β-sheets. Additionally there are two separate methods in which the secondary structure's angular representation are incorporated into an initial population member. The first method, SSI-1, generates population members by randomly selecting an angular value between the thresholds provided for each $\psi, \phi,$ and ω dihedral angle associated with each residue that defines the protein under study. This results in all $\psi, \phi,$ and ω dihedral angles being identical for each residue in the respective population member. The second method, SSI-2, is a slight variation of this and is accomplished by randomly selecting an angular value for each $\psi, \phi,$ and ω dihedral angle of each residue, resulting in different $\psi, \phi,$ and ω dihedral angles appearing in a single population member.

The Hill-Climbing Initialization (HCI) method is designed to take a randomly generated population member and conduct a local hill-climbing search. Algorithmically this essentially sweeps a given number of times from the least-to-most significant bits (right to left) of the binary string that represents a population member. The sweeping mechanism is completed right-to-left in an attempt to conduct the local search by first manipulating the side chain dihedral angles and then altering the backbone dihedral angles (our implementation puts side-chain angles to the right of backbone dihedral angles). Bit by bit the population member's allele values are swapped from a 1 to a 0 or vise versa, completing an evaluation of the population member after each bit modification. If the modification results in an improved fitness value, the change is kept and the next bit is analyzed for modification, otherwise the bit is returned to its original value and the next bit is analyzed for modification. This process is repeated until the requested number of sweeps are completed. This was done with the hope of obtaining a semi-optimal population member based on the randomly generated strings initial backbone configuration.

The memetic algorithm is designed to allow the user to have an initial population set with either all random members, with a percentage of HCI members, with a percentage of SSI members (only one secondary structure can be selected), or a percentage of a combination of all methods. The percentage of the population must be defined as an input parameter as well as the choice of seeding the initial population.

The motivation for the SSI is to seed the population with potentially “good” BBs that will remain present through the BBF phase if they are indeed “good” BBs. If the secondary structure chosen does not exist in the protein being tested then the seeded members may not contain “good” BBs and they are filtered out through the BBF phase. The rationale behind the two methods of generating the population member via the SSI is to generate a series of $\psi, \phi,$ and ω values, which include the secondary structure present in the protein.

The first method, SSI-1 exhibits the problem of a reduced search space since it forces each occurrence of an angle to have the same value. This is done in the hope that part of the resulting conformation of that initialized population member may be close to part of the geometrical conformation of the minimized protein. The problem with this method is that very few angles are represented in the initial population set. To improve upon this, the second method somewhat bypasses this problem through the random generation of each angle within the thresholds. SSI-2 still exhibits a reduced search space but allows for more potential BBs to be present in the population members. It is noted that both of these methods restrict the size of the search space through their limitation on the angular values, but the remainder of the population is randomly generated and allows for other regions of the search space to be used. The primary goal of the SSI implementations is to seed the algorithm with “good” BBs from the start. While this may not occur with all proteins tested, for those with secondary structures, some “good” BBs will be found and used.

The secondary structure modifications that are presented in this paper is done with the goal of increasing the efficiency and effectiveness of the algorithm across a wide variety of proteins that are found in nature, i.e. to create a generalized method of incorporating problem domain information into the algorithm. The effectiveness increase is noticed through the result of better solutions and the efficiency increase through the decreased running times due to the fewer requests for

---

[1] α helix: ψ and ϕ dihedral angles are -57 and -47 degrees with a $+/-$ 15 degree threshold, and ω is 180 degrees $+/-$ a 5 degree threshold. β sheet: ψ and ϕ dihedral angles are -119 and 113 with a $+/-$ 15 degree threshold, and ω is 180 degrees $+/-$ a 5 degree threshold.

random calls. This is achieved by completing 1 random call for each angle or angle type in the SSI methods versus the 10 random calls that are completed for each bit of the respective angle in the random method.

While the model proteins presented, [Met]-Enkephelin and polyalanine, are suitable for this secondary structure analysis, but not complete, they do provide a good starting point. Better results are anticipated with testing of real-world proteins containing thousands of angles and varied secondary structures. The [Met]-Enkephelin does not contain a secondary structure but is included to illustrate the generality of the technique presented and the ability to use it against proteins with no secondary structure.

6 EXPERIMENTAL DESIGN

All of the tests were conducted on a single 933MHz Intel Pentium III machine with 256MB of RAM under Red Hat Linux version 6.2. Each test set consisted of 10 separate data runs, with different random number seeds used in each data run. Various input parameters as mentioned must be specified for executing the fmGA code. The protein selected is used to determine the BB sizes selected for execution since the BB size is related to the overall string length. The larger the protein, the longer its respective string length and larger BBs should be utilized. In the experiments conducted for the Met-Enkephalin and polyalanine proteins the respective parameters used were as follows; the string length was set to 240 and 560 bits respectively, with 10 bits associated with each dihedral angle; the BB size range 6-10, and 16-20; and population sizes of 50, 58, 62, 66, 71, 77 and 50, 80, 84, 90, 96, 103, with all experiments being conducted over 3 epochs (an epoch is one complete run of the fmGA through each BB size). The larger population and BB sizes used for the polyalanine protein were based on the work of Merkle [11]. We used Goldberg's original equation on determining the initial fmGA population size [5]. The following fmGA parameters were kept constant over all runs; cut probability = 0.02, splice probability = 1.00, BBF generations = 200, juxtapositional generations = 200, total generations = 400. An input schedule was used to specify at what generations BB filtering would occur, and the sizes of the building blocks the algorithm would utilize.

The experiments are designed to provide enough data to complete a statistical analysis of the results. For each of the population seeding methods and percentages (0%, 10%, 20%, 30%, 40%, 50%, and 100%) of seeding presented in Tables 1 and 2, 10 data runs were completed for statistical significance. The seeding percentages specify the percentage of the number of initial population members which are seeded via one of the aforementioned seeding methods. The maximum, median, minimum and average fitness values are presented along with the associated standard deviations. The minimum fitness value is also the best fitness found considering the PSP minimization problem. Both protein molecules used have unique properties that should help discriminate the effectiveness of each seeding mechanism. The Met-Enkephalin contains no secondary structure while the polyalanine has a perfect α-helix secondary structure.

Table 1: Met-Enkephalin Energy

		[Met]-Enkephelin				
	%	Max	Median	Min	Avg	SD
	0	-20.94	-23.90	**-26.35**	-23.78	1.52
Sweeps	5	-20.94	-23.90	-26.35	-23.78	1.52
	10	-24.97	-26.39	-27.69	-26.35	1.52
	20	-26.06	-26.99	-28.78	-27.21	0.76
	30	-26.08	-27.10	-28.60	-27.18	0.91
	40	-26.71	-27.93	-28.93	-27.92	0.64
	50	-27.88	-27.85	-30.01	-27.96	0.90
Combo	5	-23.75	-25.25	-27.94	-25.47	1.38
	10	-24.60	-26.20	-28.68	-26.50	1.48
	20	-25.16	-26.10	-28.28	-26.44	0.96
	30	-25.76	-26.43	-27.49	-26.46	0.55
	40	-25.10	-26.97	**-31.26**	-27.17	1.70
	50	-25.92	-27.07	-29.09	-27.29	1.14
α-helix	5	-21.41	-24.34	-27.16	-24.23	1.79
	10	-22.97	-24.67	-27.34	-24.84	1.57
	20	-22.91	-24.52	-26.94	-24.78	1.13
	30	-21.00	-24.37	-28.35	-24.40	1.87
	40	-23.12	-26.08	-29.04	-25.89	1.66
	50	-23.35	-25.48	-29.21	-25.52	1.78
β-sheet	5	-21.60	-24.24	-27.16	-24.19	1.69
	10	-22.87	-25.29	-27.90	-25.21	1.53
	20	-23.37	-25.21	-29.84	-25.61	1.69
	30	-20.96	-25.67	-29.40	-25.47	2.32
	40	-23.77	-25.46	-26.36	-25.23	0.98
	50	-21.24	-25.81	-27.02	-25.48	1.62

Note that the results for each test include 10 data runs and by the Central Limit Theorem we can assume a normal distribution. The median values and average values are relatively close indicating that this assumption is valid. Therefore given the max and min values, the distribution of points in the energy landscape imply an energy surface that is somewhat coarse. Thus for these particular proteins it is somewhat difficult to search with an evolutionary algorithm.

7 RESULTS

The fitness results obtained from each test set are presented in Tables 1 and 2. When compared to their

Table 2: Polyalanine Energy

		Polyalinine				
	%	Max	Median	Min	Avg	SD
	0	-111.99	-127.77	**-140.56**	-128.78	8.55
Sweeps	5	-119.75	-130.84	-130.76	-139.25	4.95
	10	-128.76	-135.15	-136.92	-130.76	3.28
	20	-128.77	-133.79	-137.24	-134.83	3.47
	30	-131.49	-133.79	-141.03	-134.83	3.12
	40	-132.77	-134.64	-136.76	-134.59	1.33
	50	-131.58	-135.67	-139.76	-135.94	2.71
Combo	5	-118.72	-129.21	-136.48	-128.61	5.53
	10	-124.01	-130.91	-142.41	-130.89	5.53
	20	-128.50	-132.15	-139.30	-132.83	3.57
	30	-126.36	-133.14	-137.01	-132.68	3.00
	40	-127.41	-133.12	-139.97	-133.72	4.08
	50	-129.12	-132.20	**-144.19**	-133.66	4.38
α-helix	5	-118.45	-130.22	-137.34	-129.37	7.05
	10	-116.36	-126.12	-140.59	-127.64	8.64
	20	-116.17	-132.26	-138.57	-129.63	8.50
	30	-116.46	-129.68	-138.95	-129.22	7.98
	40	-121.18	-126.41	-136.07	-127.34	4.93
	50	-117.96	-127.11	-140.08	-127.70	7.81
β-sheet	5	-118.89	-127.56	-137.34	-128.46	6.25
	10	-119.76	-130.95	-141.00	-131.18	6.02
	20	-114.75	-130.81	-138.57	-128.68	8.62
	30	-120.44	-131.61	-136.67	-131.16	4.44
	40	-107.38	-127.84	-138.02	-125.81	9.95
	50	-108.08	-127.50	-142.72	-126.33	9.30

Table 3: Protein Timing and EVE

		Met			Poly		
	%	Avg	SD	EVE	Avg	SD	EVE
	0	894	4.2	37.6	4939	16	38.35
Sweeps	5	864	4.1	37.3	5926	52	45.33
	10	1012	5.9	38.4	7426	52	55.67
	20	1254	4.3	46.1	10351	17	77.67
	30	1446	6.3	53.2	13255	20	98.31
	40	1689	4.8	60.5	16289	23	121.02
	50	1925	5.0	68.9	19359	37	142.41
Combo	5	1351	16.3	53.0	4922	170	38.3
	10	1433	18.6	54.1	5596	211	42.8
	20	1590	26.0	60.1	6551	177	49.3
	30	1675	42.1	63.3	7389	246	55.7
	40	1112	13.9	40.9	8410	190	62.9
	50	1167	25.1	42.8	9577	278	71.7
α-helix	5	889	5.5	37.0	4912	25	38.0
	10	879	4.0	37.4	4919	25	38.5
	20	906	5.7	36.1	4904	25	37.8
	30	901	3.8	35.3	4919	23	38.1
	40	887	3.1	36.8	4905	22	38.5
	50	874	3.7	35.2	4900	22	38.4
β-sheet	5	893	4.9	36.9	4951	39	38.5
	10	896	5.3	35.6	4913	23	36.9
	20	884	3.2	34.5	4907	24	38.1
	30	876	3.5	34.4	4904	16	36.4
	40	876	3.3	34.8	4913	23	39.1
	50	886	5.7	34.8	4907	15	38.8

computational performance show that while both the HCI and Combo Seeding methods obtained better averages it wasn't without cost. In fact, Table 3 shows that a considerable computational cost is required to obtain better results. It is also noted that all of the seeding methods find better minimum fitness values than the standard implementation (0%). Further we see that the combo method always achieves the best minimum fitness, and the α and β methods find solutions very close to it, thereby illustrating the usefulness of the secondary structure seeding mechanism.

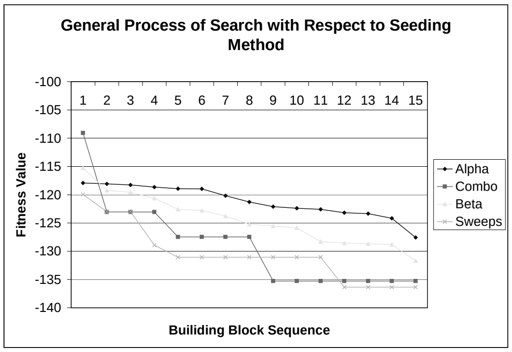


Figure 2: Building Block vs. Fitness

In order to compare the various methods employed we combine the two variables (fitness and time) into a new metric. This metric, efficiency versus effectiveness (EVE), is introduced to allow a direct comparison between different data runs. The EVE metric, Equation 2, is calculated from average fitness and timing values. This metric shows us that the α and β SSI methods find better solutions at a significantly cheaper cost (see Tables 1 and 2). A review of the new metric values show that there is a slight advantage in using these new seeding methodologies. It is also noted that the seeding mechanisms should yield improved results when applied to real-world proteins that do not contain a geometrically perfect secondary structure.

$$EVE \triangleq \left(\frac{Average\ Efficiency}{Average\ Best\ Fitness}\right) \tag{2}$$

An interesting phenomena occurred during the testing of the Combo and HCI seeding methods. Figure 2 compares a single experimental run (that is indicative of the overall behavior identified). It is interesting to note that injecting optimized solutions into the initial population often resulted in the algorithm getting stuck in a local optima area and, as a result, the fmGA algorithm was essentially ineffectual wasting precious computational time. On the other hand, both the α and β SSI seeding method, while not starting out with the best values, they did allow the fmGA to be the dominating factor in finding better solutions (as depicted by the continual improvement in the best solution found), building block after building block. Note that the conformational dihedral angles for the various "good" minimum energy values found do not reflect similar positions in the PSP energy landscape and thus a local search technique should be employed with discretion. Additionally it is worthwhile to note that the reduction in execution time achieved with the seeding methods allow for larger population sizes to be utilized, thereby increasing the size of the space actually searched in the same amount of time.

8 CONCLUSIONS

Incorporating immunological concepts into the fmGA in an attempt to improve the search for a better "semi"-optimal PSP solution was met with success. Minimum PSP conformational energy values were found in an acceptable amount of execution time. Additional research is required to determine if other domain space information can be incorporated into the algorithm as an alternative seeding method along with parameter variations. It is the authors' belief that both α and β SSI methods, depending on the protein under study, could out-perform the other two methods, especially if considering the new performance metric. The fmGA approach exercised here is an effective and efficient approach to solving the PSP problem as compared to the X-ray crystallography, Ab initio, semi-empirical, and force field approaches.

References

[1] Patricia J. Blake and Rosanne C. Perez. *Applied Immunological Concepts.* Appleton-Century-Crofts, New York, 1978.

[2] Dipankar Dasgupta. Immunity-based intrusion detection systems: A general framework. *Proceedings of the 22nd National Information Systems Security Conference (NISSC)*, October 1999.

[3] Donald Brinkman et al. Parallel genetic algorithms and their application to the protein folding problem. *Intel Supercomputer Users Group Conference Proceedings.*, 1993.

[4] Dmitrij Frishman and Patrick Argos. Seventy five percent accuracy in protein secondary structure prediction. *Proteins*, 29:443–460, 1997.

[5] David E. Goldberg, Kalyanmoy Deb, Hillol Kargupta, and Georges Harik. Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. Technical Report 93004, University of Illinois at Urbana-Champaign, Urbana, IL, USA, 1993.

[6] Paul Harmer and Gary Lamont. An agent based architecture for a computer virus immune system. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, July 2000.

[7] Emma Hart, Peter Ross, and Jeremy Nelson. Producing robust schedules via an artificial immune system. *Proceedings of the IEEE International Conference on Evolutionary Computing*, May 1998.

[8] Steven A. Hofmeyr and Stephanie Forrest. Immunity by design: An artificial immune system. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 1289 – 1296, July 1999.

[9] Gary B. Lamont, Robert E. Marmelstein, and David A. Van Veldhuizen. *New Ideas in Optimization*, chapter A Distributed Architecture for a Self-Adaptive Computer Virus Immune System, pages 167–183. Advanced Topics in Computer Science Series. McGraw-Hill, London, 1999.

[10] Scott M. LeGrand and Kenneth M. Merz Jr. The application of the genetic algorithm to the minimization of potential energy functions. *Journal of Global Optimization*, pages 49–66, 1993.

[11] Laurence D. Merkle, George H. Gates, Jr., Gary B. Lamont, and Ruth Pachter. Application of the parallel fast messy genetic algorithm to the protein structure prediction problem. *Proceedings of the Intel Supercomputer Users' Group Users Conference*, pages 189–195, 1994.

[12] Steven R. Michaud. Solving the protein structure prediction problem with parallel messy genetic algorithms. Master's thesis, Air Force Institute of Technology, Wright Patterson AFB, March 2001. AFIT/GCS/ENG/01M.

[13] Steven R. Michaud, Jesse B. Zydallis, Gary B. Lamont, and Ruth Pachter. Detecting secondary peptide structures by scaling a genetic algorithm. *First International Conference on Computational Nanoscience*, 2001.

[14] Inc. Molecular Simulations. Charmm version 22.0 parameter file. Computer Software, 1992.

[15] David R. Westhead and Janet M. Thornton. Protein structure prediction. *Current Opinion in Biotechnology*, 9:383–389, 1998.

Developing a Market Timing System using Grammatical Evolution

Michael O'Neill
Dept. Of Computer Science And Information Systems,
University of Limerick, Ireland.

Tony Brabazon
Dept. Of Accountancy,
University College Dublin, Ireland.

Conor Ryan & J.J. Collins
Dept. Of Computer Science And Information Systems,
University of Limerick, Ireland.

Abstract

This study examines the potential of an evolutionary automatic programming methodology, Grammatical Evolution, to uncover a series of useful fuzzy technical trading rules for the ISEQ, the official equity index of the Irish Stock Exchange. Index values for the period 29/03/93 to 4/12/1997 are used to train and test the model. The preliminary findings indicate that the methodology has much potential.

1 Introduction

We have previously evolved trading rules for the UK FTSE 100 index [O'Neill et.al. 2001], and now wish to extend this approach to new markets, and through the inclusion of additional technical indicators.

1.1 Technical analysis

A market index is comprised of a weighted average measure of the price of individual shares which make up that market. The value of the index represents an aggregation of the balance of supply and demand for these shares. Some market traders, known as technical analysts, believe that prices move in trends and that price patterns repeat themselves [Murphy 1999]. If we accept this premise, that there are rules, although not necessarily static rules, underlying price behaviour, it follows that trading decisions could be enhanced through use of an appropriate rule induction methodology such as Grammatical Evolution (GE).

Although controversy exists amongst financial theorists regarding the veracity of the claim of technical analysts, recent evidence has suggested that it may indeed be possible to uncover patterns of predictability in price behaviour. Brock, Lakonishok and LeBaron [Brock, Lakonishok & LeBaron 1992] found that simple technical trading rules had predictive power and suggested that the conclusions of earlier studies that technical trading rules did not have such power were "premature". Other studies which indicated that there may be predictable patterns in share price movements include those which suggest that markets do not always impound new information instantaneously [Hong, Lim, & Stein 1999] [Chan, Jegadeesh & Lakonishok 1996], that stock markets can overreact as a result of excessive investor optimism or pessimism [Dissanaike 1997] and that returns on the market are related to the day of the week [Cross 1973] or the month of the year [DeBondt & Thaler 1987]. The continued existence of large technical analysis departments in international finance houses is consistent with the hypothesis that technical analysis has proven empirically useful.

1.2 Potential for application of evolutionary automatic programming

As noted by Iba and Nikolaev [Iba & Nikolaev 2000] there are a number of reasons to suppose that the use of an evolutionary automatic programming (EAP) approach can prove fruitful in the financial prediction domain. EAP can conduct an efficient exploration of the search space and can uncover dependencies between input variables, leading to the selection of a good subset for inclusion in the final model. Additionally, use of EAP facilitates the utilisation of complex fitness functions including discontinuous, non-differentiable functions. This is of particular importance in the financial domain as the fitness criterion may be complex, usually requiring a balancing of return and risk. EAP, unlike, for example, basic neural net approaches to financial prediction, does not require the ex-ante determination of optimal model inputs and their related transformations. Another useful feature of EAP is that it produces human-readable rules that have the potential to enhance understanding of the problem domain.

1.3 Motivation for study

This study was motivated by a number of factors. Much of the existing literature concerning the application of genetic algorithms (GA) or GP to the generation of technical trading rules [Allen & Karjalainen 1999] [Colin 1994] [Bauer 1994] [Neely, Weller & Dittmar 1997] [Deboeck 1994] concentrates on the US and to a lesser extent the Japanese stock markets. Published research on this area is both incomplete and scarce. To date, only a limited number of GA / GP methodologies and a limited range of technical indicators have been considered. This study addresses these limitations by examining index data drawn from the Irish stock market and by adopting a novel evolutionary automatic programming approach.

The paper is organised as follows. Section two discusses the background to the technical indicators utilised in this study. Section three describes the evolutionary algorithm adopted, Grammatical Evolution [O'Neill & Ryan 2001] [Ryan et.al. 1998]. Section four outlines the data and function sets used. The following sections provide the results of the study followed by a discussion of these results and finally a number of conclusions are derived.

2 Background

As with any modelling methodology, issues of data preprocessing need to be considered. Rather than attempting to uncover useful technical trading rules for the ISEQ index using raw current and historical price information, this information is initially pre-processed into technical indicators. The objective of these pre-processing techniques is to uncover possible useful trends and other information in the time series of the raw index data whilst simultaneously reducing the noise inherent in the series.

2.1 Technical Indicators

The development of trading rules based on current and historic market price information has a long history [Brown, Goetzmann & Kumar 1998]. The process entails the selection of one or more technical indicators and the development of a trading system based on these indicators. These indicators are formed from various combinations of current and historic price information. Although there are potentially an infinite number of such indicators, the financial literature suggests that certain indicators are widely used by investors [Brock, Lakonishok & LeBaron 1992][Murphy 1999] [Pring 1991].

Four groupings of indicators are given prominence in prior literature:

i. Moving average indicators

ii. Momentum indicators

iii. Trading range indicators

iv. Oscillators

Given the large search space, an evolutionary automatic programming methodology has promise to determine both a good quality combination of, and relevant parameters for, trading rules drawn from individual technical indicators.

We intend to use of each of these groupings as our model is developed. Our initial study on the FTSE dataset [O'Neill et.al. 2001] included only a moving average indicator. This study also includes momentum, and trading range volatility indicators.

2.1.1 Moving Average Indicators

The simplest moving average systems compare the current share price or index value with a moving average of the share price or index value over a lagged period, to determine how far the current price has moved from an underlying price trend. As they smooth out daily price fluctuations, moving averages can heighten the visibility of an underlying trend. A variation on simple moving average systems is to use a moving average convergence divergence (MACD) oscillator. This is calculated by taking the difference of a short run and a long run moving average. In a recursive fashion, more complex combinations of moving averages of values calculated from a MACD oscillator can themselves be used to generate trading rules. For example, a nine day moving average of a MACD oscillator could be plotted against the raw value of that indicator. A trading signal may be generated when the two plotted moving averages cross. Moving average indicators are trend following devices and work best in trending markets. They can have a slow response to changes in trends in markets, missing the beginning and end of each move. They tend to be unstable in sideways moving markets, generating repeated buy and sell signals (whipsaw) leading to unprofitable trading. Trading systems using moving averages trade-off volatility (risk of loss due to whipsaw) against sensitivity. The objective is to select the lag period which is sensitive enough to generate a useful early trading signal but which is insensitive to random noise.

2.2 Momentum

The momentum of a security is the ratio of a time-lagged price to the current price ($\frac{Price_t}{Price_{t-x}}$). The belief underlying this indicator is that strongly trending shares tend to continue to move in that direction for a period of time as more investors buy or sell the trending share. There is recent evidence that momentum trading strategies can work, particularly when investing in smaller firms [Hong, Lim, & Stein 1999]. Technical analysts consider that price momentum can foretell a price turning point as momentum will tend to peak before the price peaks.

2.3 Trading Range Breakout systems

It these systems, a signal is usually generated if a price breaks out of a defined range. A simple example of a trading rule would be to buy a share when it exceeds its previous high in the last four weeks and conversely to sell a share if it falls below its previous four week low. A more complex approach is to plot an envelope at ± 'x' standard deviations above and below a moving average. Penetration of the bands by the current day's price, indicates a possible price trend reversal.

A description of the evolutionary automatic programming system used to evolve trading rules now follows.

3 Grammatical Evolution

Grammatical Evolution (GE) is an evolutionary algorithm that can evolve computer programs in any language. Rather than representing the programs as parse trees, as in traditional GP [Koza 1992], a linear genome representation is adopted. A genotype-phenotype mapping process is used to generate the output program for each individual in the population. Each individual, a variable length binary string, contains in its codons (groups of 8 bits) the information to select production rules from a Backus Naur Form (BNF) grammar. The BNF is a plug-in component to the genotype-phenotype mapping process, that represents the output language in the form of production rules. It is comprised of a set of non-terminals that can be mapped to elements of the set of terminals, according to the production rules. An example excerpt from a BNF grammar is given below. These productions state that S can be replaced with either one of the non-terminals `expr`, `if-stmt`, or `loop`.

```
S ::= expr      (0)
    | if-stmt   (1)
    | loop      (2)
```

The grammar is used in a generative process to construct a program by applying production rules, selected by the genome, beginning from the start symbol of the grammar.

In order to select a rule in GE, the next codon value on the genome is generated and placed in the following formula:

$$Rule = Codon\ Value\ MOD\ Num.\ Rules$$

If the next codon integer value was 4, given that we have 3 rules to select from as in the above example, we get $4\ MOD\ 3\ =\ 1$. S will therefore be replaced with the non-terminal `if-stmt`.

Beginning from the left hand side of the genome codon integer values are generated and used to select rules from the BNF grammar, until one of the following situations arise:

i. A complete program is generated. This occurs when all the non-terminals in the expression being mapped, are transformed into elements from the terminal set of the BNF grammar.

ii. The end of the genome is reached, in which case the *wrapping* operator is invoked. This results in the return of the genome reading frame to the left hand side of the genome once again. The reading of codons will then continue unless an upper threshold representing the maximum number of wrapping events has occurred during this individual's mapping process.

iii. In the event that a threshold on the number of wrapping events is exceeded and the individual is still incompletely mapped, the mapping process is halted, and the individual assigned the lowest possible fitness value.

GE uses a steady state replacement mechanism, such that, two parents produce two children the best of which replaces the worst individual in the current population if the child has a greater fitness. The standard genetic operators of point mutation, and crossover (one point) are adopted. It also employs a duplication operator that duplicates a random number of codons and inserts these into the penultimate codon position on the genome. A full description of GE can be found in [O'Neill & Ryan 2001] [Ryan et.al. 1998].

4 Problem Domain & Experimental Approach

We describe an approach to evolving trading rules using GE. This study uses daily data for the Irish ISEQ stock index drawn from the period 29/03/1993 to 4/12/1997. The training data set was comprised of 360 days from the first day (29/03/1993) with an additional 75 days at the beginning of this time to allow for the time lag introduced with technical indicators such as the moving average. The remaining data was divided into two hold out samples totaling 805 (i.e. two 365 day periods with a time lag of 75 days) trading days. The division of the hold out period into two segments was undertaken to allow comparison of the out of sample results across different market conditions, in order to assess the stability and degradation characteristics of the developed model's predictions.

The rules evolved by GE are used to generate one of three signals for each day of the training or test periods. The possible signals are *Buy*, *Sell*, or *Do Nothing*. Permitting the model to output a Do Nothing signal reduces the hard threshold problem associated with production of a binary output. This issue has not been considered in a number of prior studies. A variant on the trading methodology developed in Brock et al. [Brock, Lakonishok & LeBaron 1992] is then applied. If a buy signal is indicated, a fixed investment of $1,000 (arbitrary) is made in the market index. This position is closed at the end of a ten day (arbitrary) period.

On the production of a sell signal, an investment of $1,000 is sold short and again this position is closed out after a ten day period. This gives rise to a maximum potential investment of $10,000 at any point in time (the potential loss on individual short sales is in theory infinite but in practice is unlikely to exceed $1,000). The profit (or loss) on each transaction is calculated taking into account a one-way trading cost of 0.2% and allowing a further 0.3% for slippage. The total return generated by the developed trading system is a combination of its trading return and its risk free rate of return generated on uncommitted funds.

The rate adopted in this calculation is simplified to be the average interest rate over the entire data set (5.8%).

As well as the moving average, the momentum and trading range volatility technical indicators are adopted in these preliminary experiments.

In addition to the technical indicators the grammar also allows the use of the binary operators f_and, f_or, and the standard arithmetic operators, and the unary operator f_not [1].

The signals generated for each day, Buy, Sell, or Do Nothing, are post-processed using fuzzy logic. The trading rule, a fuzzy trading rule, returns values in the range 0 to 1. We use pre-determined membership functions, in this case, to determine what the meaning of this value is. The membership functions adopted were as follows:

$$Buy = Value < .33$$

$$Sell = .33 >= Value < .66$$

$$DoNothing = .66 >= Value$$

4.1 Data Preprocessing

The value of the ISEQ index increased substantially over the training and testing period, rising from 1337.44 to 3498.84. Before the trading rules were constructed, these values were normalised using a two phase preprocessing. Initially the daily values were transformed by dividing them by a 75 day lagged moving average. These transformed values are then normalised using linear scaling into the range 0 to 1. This procedure is a variant on that adopted by Allen and Karjalainen [Allen & Karjalainen 1999] and Iba and Nikolaev [Iba & Nikolaev 2000].

4.2 Selection of Fitness Function

A key decision in applying a GP methodology to construct a technical trading system is to determine what fitness measure should be adopted. A simple fitness measure such as the profitability of the system both in and out of sample is inadequate as it fails to consider the risk associated with the developed trading system. The risk of the system can be estimated in a variety of ways. One possibility is to consider market risk, defined here as the risk of loss of funds due to a market movement. A measure of this risk is provided by the maximum drawdown (maximum cumulative loss) of the system during a training or test period. This measure of risk can be incorporated into the fitness function in a variety of formats including: (return / maximum drawdown) or return - 'x'(maximum drawdown), where 'x' is a pre-determined constant dependent on an investor's psychological risk profile. For a given rate of return, the system generating the lowest maximum drawdown is preferred.

This study incorporates drawdown in the fitness function by subtracting the maximum cumulative loss during the training period from the profit generated during that period. This is a conservative approach which will encourage the evolution of trading systems with good return to risk characteristics. This will provide a more stringent test of trading rule performance as high risk / high reward trading rules will be discriminated against.

5 Results

The results from our preliminary experiments are now given. Runs were conducted with a population size of 500 for 100 generations. Trading rules were evolved with a performance superior to that of a benchmark buy and hold strategy. Under this benchmark, an amount of $10,000 is invested in the market at the beginning of each of the test periods. The gain on this investment to the end of each period is then calculated. The best individual (set of trading rules) found to date made a profit of US$5777 over the training period.

When tested on the two out of sample periods following the training data set we find that this individual was consistently profitable. Plots of the index over each of the test periods and the training period can be seen in Figure 1.

To facilitate assessment of these results, they are compared with those of the benchmark buy and hold strategy. The results of this buy and hold strategy can be seen in Table 1.

In assessing these results, the market risk profile of each trading strategy should be considered. The buy and hold strategy maintains an investment of $10,000 plus cumulative returns since the start of the investment period, in the market at all times whereas the maximum investment of the developed trading system, ignoring drawdown, is $10,000. Looking

---

[1] The operations f_and, f_or, and f_not are fuzzy logic operators returning the minimum, maximum, of the arguments, and 1 - the argument, respectively.

Trading Period (Days)	**Buy & Hold** Profit (US$)	**Best-of-run** Profit(US$)	**Best-of-run** Avg. Daily Investment
Train (1095 to 1460)	2087	5777	8233
Test 1 (1461 to 1826)	3848	1478	5055
Test 2 (1827 to 2192)	3439	2627	8808
Total	9374	9882	

Table 1: A comparison of benchmarks with the best of run individual.

at Table 1 we can see the average daily investment made by the best of run individual for each test period. Averaged over all the test periods the developed system has an investment of $6932 in the market.

There is no clear evidence that the trading system has higher market risk than the buy and hold strategy.

6 Discussion

In evaluating the performance of any market predictive system, a number of caveats must be borne in mind. Any trading model constructed and tested using historic data will tend to perform less well in a live environment than in a test period for a number of reasons. Live markets have attendant problems of delay in executing trades, illiquidity, interrupted / corrupted data and interrupted markets. The impact of these issues is to raise trading costs and consequently to reduce the profitability of trades generated by any system. An allowance for these costs ('slippage') has been included in this study but it is impossible to determine the scale of these costs ex-ante with complete accuracy. In addition to these costs, it must be remembered that the market is competitive. As new computational technologies spread, opportunities to utilise these technologies to earn excess risk-adjusted profits are eroded. As a result of this technological 'arms-race', estimates of trading performance based on historical data may not be replicated in live trading as other market participants will apply similar technology. This study ignores impact of dividends. Although a buy-and-hold strategy will generate higher levels of dividend income than an active trading strategy, the precise impact of this factor is not determinable ex-ante. It is notable that the dividend yield on most stock exchanges has fallen sharply in recent years and that the potential impact of this factor has lessened.

7 Conclusions & Future Work

The results clearly show that the model is suffering from over-fitting to the training data set, however, the risk involved with the adoption of the evolved trading rules is less than the benchmark buy and hold strategy.

The risk of the benchmark buy-and-hold portfolio exceeded that of the portfolio generated by the technical trading rules because, the benchmark buy and hold portfolio maintains a fully invested position at all times in the market, whereas the portfolio generated by the technical trading system averaged a capital investment of $7,000 over the test periods.

There is notable scope for further research utilising GE in this problem domain. Our preliminary methodology has included a number of simplifications, for example, we only considered a small set of primitive technical indicator. The incorporation of additional technical indicators may further improve the performance of our approach. Future work will also seek to remove the over-fitting that is occurring on the ISEQ data set in order to achieve the generalisation properties demonstrated by the evolved rules on the FTSE data set [O'Neill et.al. 2001].

References

[Allen & Karjalainen 1999] Allen, F., Karjalainen, R. (1999) Using genetic algorithms to find technical trading rules. *Journal of Financial Economics*, **51**, pp. 245-271, 1999.

[Bauer 1994] Bauer R. (1994). Genetic Algorithms and Investment Strategies, New York: John Wiley & Sons Inc.

[Brock, Lakonishok & LeBaron 1992] Brock, W., Lakonishok, J. and LeBaron B. (1992). 'Simple Technical Trading Rules and the Stochastic Properties of Stock Returns', Journal of Finance, 47(5):1731-1764.

[Brown, Goetzmann & Kumar 1998] Brown, S., Goetzmann W. and Kumar A. (1998). 'The Dow Theory: William Peter Hamilton's Track Record Reconsidered', Journal of Finance, 53(4):1311-1333.

[Chan, Jegadeesh & Lakonishok 1996] Chan, L. K. C., Jegadeesh, N. and Lakonishok, J. (1996). 'Momentum strategies', Journal of Finance, Vol. 51, No. 5, pp. 1681 - 1714.

[Colin 1994] Colin, A. (1994). 'Genetic Algorithms for Financial Modelling', in Guido Deboeck (Editor) (1994). Trading on the edge: neural, genetic and fuzzy systems for chaotic and financial markets, New York: John Wiley & Sons Inc.

[Cross 1973] Cross, F. (1973). 'The Behaviour of Stock prices on Friday and Monday', Financial Analysts' Journal, Vol. 29(6), pp.67-74.

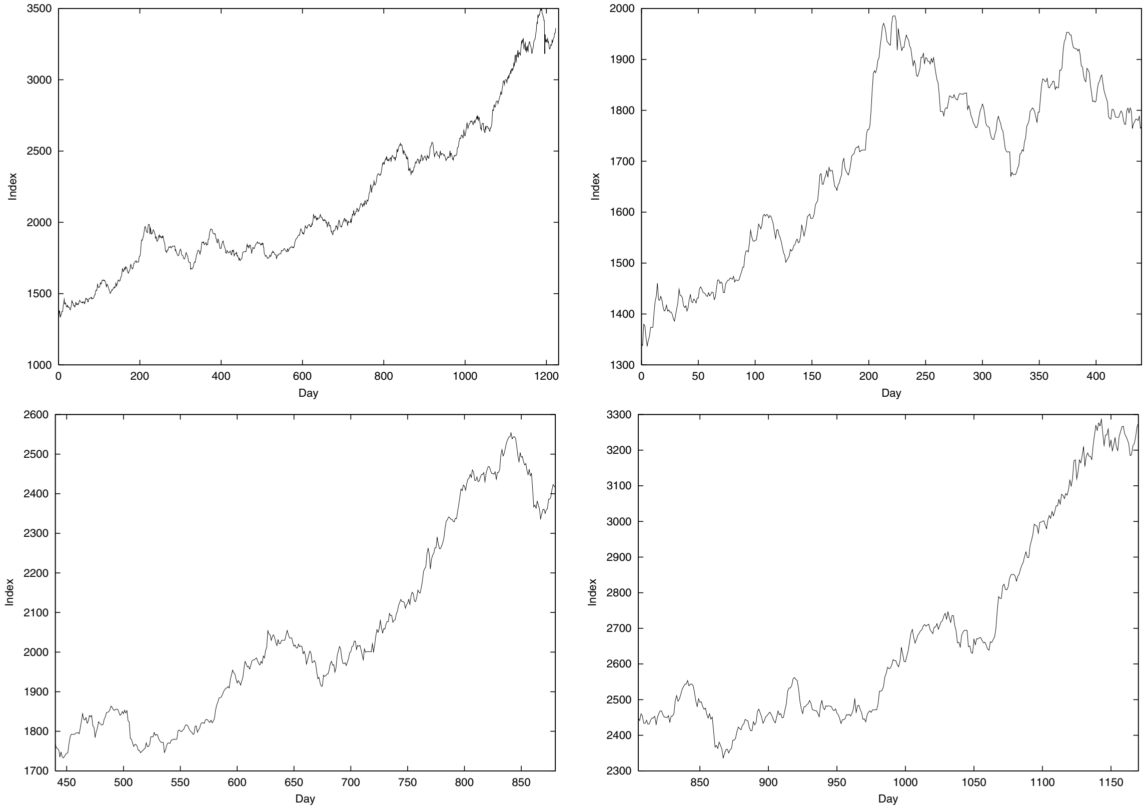


Figure 1: A plot of the ISEQ over the entire data set (top left), over the training period (top right), over the two test periods (bottom left and right respectively).

[Deboeck 1994] Deboeck G. (1994). 'Using GAs to optimise a trading system', in Guido Deboeck (Editor) (1994). Trading on the edge: neural, genetic and fuzzy systems for chaotic and financial markets, New York: John Wiley & Sons Inc.

[DeBondt & Thaler 1987] DeBondt, W. and Thaler, R. (1987). 'Further Evidence on Investor Overreaction and Stock Market Seasonality', Journal of Finance, Vol. 42(3):pp.557-581.

[Dissanaike 1997] Dissanaike, G. (1997). 'Do stock market investors overreact?', Journal of Business Finance & Accounting (UK), Vol. 24, No.1, pp. 27-50.

[Hong, Lim, & Stein 1999] Hong, H., Lim, T. and Stein, J. (1999). 'Bad News Travels Slowly: Size, Analyst Coverage and the Profitability of Momentum Strategies', Research Paper No. 1490, Graduate School of Business, Stanford University.

[Iba & Nikolaev 2000] Iba H. and Nikolaev N. (2000). 'Genetic Programming Polynomial Models of Financial Data Series', In *Proc. of CEC 2000*, pp. 1459-1466, IEEE Press.

[Koza 1992] Koza, J. (1992). *Genetic Programming*. MIT Press.

[Murphy 1999] Murphy, John J. (1999). Technical Analysis of the Financial Markets, New York: New York Institute of Finance.

[Neely, Weller & Dittmar 1997] Neely, C., Weller P. and Dittmar, R. (1997). 'Is technical analysis in the foreign exchange market profitable? A genetic programming approach", Journal of Financial and Quantitative Analysis, Vol. 32, No. 4, pp. 405 - 428.

[O'Neill et.al. 2001] O'Neill M., Brabazon, A., Ryan C., & Collins J.J. (2001). Evolving Market Index Trading Rules Using Grammatical Evolution. In Proceedings of EvoIASP 2001.

[O'Neill & Ryan 2001] O'Neill M., Ryan C. (2001). Grammatical Evolution. *IEEE Trans. Evolutionary Computation*.

[Pring 1991] Pring, M. (1991). Technical analysis explained: the successful investor's guide to spotting investment trends and turning points, New York: Mc Graw-Hill Inc.

[Ryan et.al. 1998] Ryan C., Collins J.J., O'Neill M. (1998). Grammatical Evolution: Evolving Programs for an Arbitrary Language. *Lecture Notes in Computer Science*

1391, Proceedings of the First European Workshop on Genetic Programming, pages 83-95. Springer-Verlag.

A Grammar used by GE to evolve trading rules

```
N={<code>,<expr>,<fopbi>,<fopun>,
   <matbi>,<relbi>,<var>,<int>}

T={p,=,(,),f_and,f_or,f_not,+,-,*,
   >,<,>=,<=,scale,ma,day,
   1,2,3,4,5,10}

S=<code>

P=

<code> ::=  p = <expr> ;

<expr> ::= <fopbi> (<expr>, <expr>)
        | <fopun> (<expr>)
        | <expr><matbi><expr>
        | <expr><relbi><expr>
        | <var>

<fopbi> ::= f_and
          | f_or

<fopun> ::= f_not

<matbi> ::= + | - | *

<relbi> ::= > | < | >= | <=

<var> ::= scale( <int> )
        | scale(ma( <int> , day ))
        | scale(day)
        | scale(momentum( <int> , day ))
        | scale(trb( <int> , day ))

<int> ::= 1 | 2 | 3 | 4 | 5 | 10
```

A genetic algorithm for the classification of natural corks

PECH-GOURG Nicolas
Group SABATÉ
Espace Tech Ulrich
66400 Céret – France
pech@sabate.fr

HAO Jin-Kao
Université d'Angers
2, bd Lavoisier
49045 Angers – France
hao@info.univ-angers.fr

Abstract

In this paper, we explore the use of genetic algorithms (GA) for a classification problem encountered in wine industry: the classification of natural corks according to the defects of their heads. In particular, we are interested in the task of optimizing the parameters of an existing cork classification program. For this purpose, we introduce a GA-based approach that searches for good combinations from a huge search space. Experiments on both artificial and real data show the high effectiveness of this approach. This effectiveness justifies the use of this approach for daily operations in a real environment.

1 INTRODUCTION

The cork is a well-known natural product in fine wine industry for its reliability and for its chemical and mechanic properties. The main advantage of a natural cork stopper is to allow a good gaseous diffusion adapted to the wine maturation. This is also the most appreciated cork by wine consumers. In cork industry, the production process of this product is composed of different steps [FOU97]. First, the cork is punched in cork planks. Then corks batches are washed and classified. The last steps consist in personifying the corks (picture, surface treatment) and to pack them up.

In this study, we are interested in the classification step. In fact, natural corks are classified according to their quality and proposed to vineyard with different prices. Like a lot of natural products, natural corks are heterogeneous. To classify them, a human expert would consider holes, cracks, colors and other features of a cork. The quality of a cork depends on the nature, the quantity, the size and the position of the defects. In the case of an automatic classification of corks, only some visual features are taken into account. In this study, we are only interested in the classification according to the visual aspect of the two heads of the cork. This operation allows separating a cork set into three categories. To obtain the necessary data for the classification, we use CCD cameras that give us pictures for each head of the cork. From these pictures we obtain numerical values. A classification program is then used to determine the class of each cork. This classification decision is taken by comparing the numerical values from the cameras against some internal parameters of the classification program. These internal parameters correspond in fact to a set of thresholds that must be determined carefully in order for the classification program to work correctly. The main difficulty is that these parameters are numerous (up to 30) and have large ranges for the possible values (up to 10.000 integer values).

The goal of this work is to explore a GA-based approach to determine these threshold values used by the classification program. We evaluate this approach on both artificially generated theoretical data and real data. We show the GA-based approach is able to find near optimal values for the classification parameters. Indeed, using these parameters values, the classification program produced excellent results for both the artificial and real data.

The paper is organized as follows. In section 2, we introduce our classification problem, followed by the presentation of a mathematical formulation of the problem in section 3. In section 4, we present our GA for determining the classification parameters. In section 5, we show detailed experimental results. Conclusions are given in the last section.

2 CORKS CLASSIFICATION AND CLASSIFICATION PARAMETERS

Four CCD cameras allow obtaining two pictures for each of the two heads of a cork. Each picture is analyzed in order to extract fifteen parameters that we will note CAM_{ij}: i represents the number of the camera (between 1 and 4) and j represents the number of the parameter (between 1 and 15). We will not explain the methods used to extract these parameters, neither the nature of the selected parameters. The problem that interests us in this study is in the following step. A classification program analyses the fifteen parameters given by each of the four cameras. The result of this program is the class of the cork. In Figure 1, we show the two heads of an example cork and the classification process working with the four corresponding numerical pictures of the two heads.

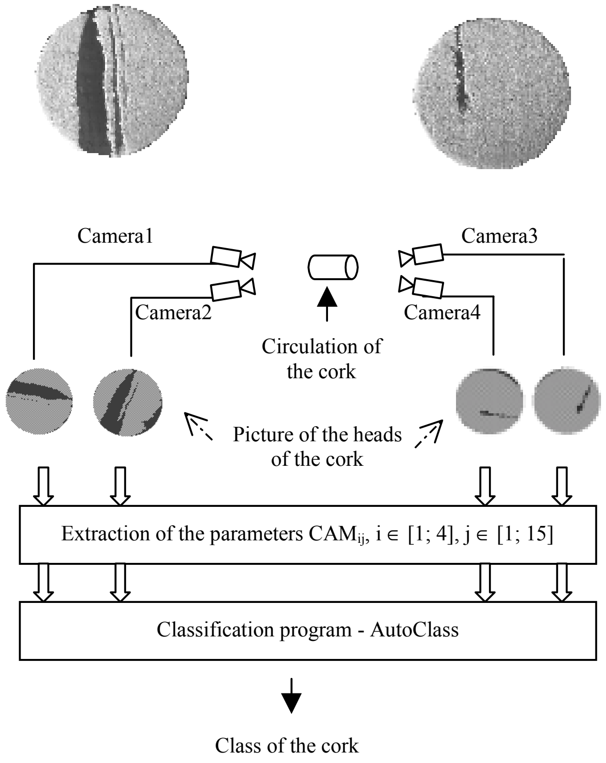

Camera1
Camera3
Camera2
Camera4
Circulation of the cork
Picture of the heads of the cork
Extraction of the parameters CAMij, i ∈ [1; 4], j ∈ [1; 15]
Classification program - AutoClass
Class of the cork

Figure 1: From the visualization to the classification of the cork

To simplify, we can say that the classification program (AutoClass) uses thirty internal parameters denoted by (P_{1i}, P_{2i}), $i \in [1; 15]$. They are the same nature as the CAM_{ij}.

The algorithm used by the classification program is quite simple: it compares the numerical values (CAM_{ij}) from the camera pictures against the classification parameters (thresholds) (P_{1i}, P_{2i}). A cork is classified to one and only one of three different classes after this comparison (Classes 1 to 3 correspond in fact to decreasing qualities).

```
IF $\forall (i, j) \in ([1 ;4], [1; 15]), CAM_{ij} < P_{1j}$
        THEN Class = 1;
ELSE    IF $\forall (i, j) \in ([1 ;4], [1 ;15]), CAM_{ij} < P_{2j}$
                THEN Class = 2;
```

Figure 2: Classification Algorithm

Clearly, the quality of the classification parameters plays a determinant role for a good classification. A good setting of these parameters $(P_{1i}; P_{2i})$, $i \in [1; 15]$ will allow to classify a cork in the class which is the most appropriated for it according to the information given by the cameras.

3 FORMULATION

In this section, we give a formulation of our problem, which is based on the CSOP model [TSA93]. Here we identify a set of (discrete) variables V, a family of value domains for the variables, a set of constraints among some variables and a cost function to be optimized.

Variables:

$$V = \{P_{1\text{-}1}; P_{1\text{-}2}; \ldots P_{1\text{-}15}; P_{2\text{-}1}; .. ; P_{2\text{-}15}\}$$
$$= \{V_i ; i \in [1; 30]\}$$

The set of variables is composed of the parameters P_{1i} and P_{2i}, that are renamed as V_i, $i \in [1; 30]$.

Domains:

$$D = \{D_i / D_i = N^+, \forall i \in [1; 30]\}$$

Each variable V_i must take a positive and entire value. More precisely, for this study, we have $D_i = [0; 800]$ for $i \in [1;15]$, $D_{15} = [5\,000; 15\,000]$ and $D_{i+15} = D_i$ for $i \in [1;15]$.

Constraints:

$$C: \forall i \in [1; 15], V_i \leq V_{i+15}$$

This constraint is used to avoid a cork that cannot be accepted in class 2, could be accepted in class 1 (Class 1 is of higher quality). This constraint is due to the classification algorithm presented before. In fact, without this constraint, we would have: $\exists\ k \in [1; 15]$ / $V_{k+15} < CAM_{ik} < V_k$. A cork can then be put to the class 1 (because $CAM_{ik} < V_k$), while it is rejected from class 2 (because $V_{k+15} < CAM_{ik}$). The set of the proposed constraints allows us to avoid this undesirable situation.

Cost function:

This is the sum of corks that are classified in the right way. These classified corks are those for which the class determined by the classification algorithm is the same as the known class given by the human expert. The aim is of course to maximize this function.

4 A GA-BASED RESOLUTION APPROACH

From the literature, one may find several studies concerning the automatic classification of corks by analyzing pictures of corks and by employing different classification techniques. For example, some researchers take interests in picture analysis to determine the quality of cork boards [MOL93]. Others are interested in the picture analysis and in the classification of corks with the help of artificial neuronal networks [CHA97]. In a more general context, genetic algorithms have been successfully applied to various classification-related problems [PUN93], [SIE88], [VAF91], [FAL93]. These previous studies on similar problems constitute one important factor motivating the choice of genetic algorithms for our classification problem.

Since the very beginning of the GA [HOL75], its principle becomes well known. For a comprehensive introduction, the reader is invited to consult books on the subject, for example [GOL89]. We give here only a brief remainder necessary to describe our genetic algorithm. A GA may be considered to be composed of three essential elements:

1. A set of *potential solutions* called individuals or chromosomes that will evolve during a number of iterations (generations). This set of solutions is also called *population*.
2. An *evaluation* mechanism (fitness function) that allows assessing the quality or fitness of each individual of the population.
3. An *evolution* procedure that is based on some “genetic” operators such as selection, crossover and mutation.

Crossover and Mutation

- The crossover takes two individuals to produce two new individuals. For example, the application of the well-known one-point crossover to α=abcd and ß=bbaa can produce two individuals γ=ab*aa* and η=bb*cd*.
- The mutation consists in modifying randomly a gene of an individual. A mutation of γ=ab*a*a could lead to a new individual γ=ab*e*a.

Fitness function and selection

The quality of the individuals is assessed with a fitness function. The result is a real value for each individual. The best individuals will survive and are allowed to produce new individuals.

Stop condition

The stop condition is used to determine the end of the algorithm. Well-known stop conditions are:

- a pre-defined number of generations or evaluations,
- a pre-defined value to reach for the fitness function,
- a number of generation without improvement.

Our genetic algorithm

For our problem of determining the parameters for cork classification, each individual is defined by a vector: $V^i=(P^i_1, ., P^i_{10}, ., P^i_{30})$, each gene corresponding to one of the thirty parameters of the problem and taking its value from its value domain (c.f. §3). A population of 40 individuals is used in this study.

The classical one-point crossover is used to generate new individuals. For the mutation, the following technique is used. Suppose we decide to mutate the k^th^ gene V^i_k of an individual. Then the new value for the gene is determined by V^i_k + (random(1)-0.5) x V^i_k. Selection is carried out over the whole population and half of the best individuals are kept. The best individual is always record in a variable (V*) and updated each time a better solution is found. The stop condition concerns the number of generations without improvement of the best solution found so far. This number is empirically fixed at 50 generations.

To evaluate the fitness of an individual, we run the classification program AutoClass (§2) with the parameter values coded by the individual on a learning database. The learning database is composed of a set of corks with a known class number for each cork. According to the number of corks that are correctly classified, a score is assigned to the individual that is being assessed. Since we use an external program for fitness evaluation, it is clear that the evaluation constitutes the most time-consuming part of the algorithm.

In addition to these conventional mechanisms, our GA uses a diversification function: if the best individuals of the population do not evolve during 10 generations, then the whole population undergoes a mutation (each individual is mutated). This diversification function allows modifying the population more importantly than by a crossover or a classical random mutation. It helps in some cases avoid the problem of premature convergence of the population. The overall algorithm is described by the following flowchart (Figure 3).

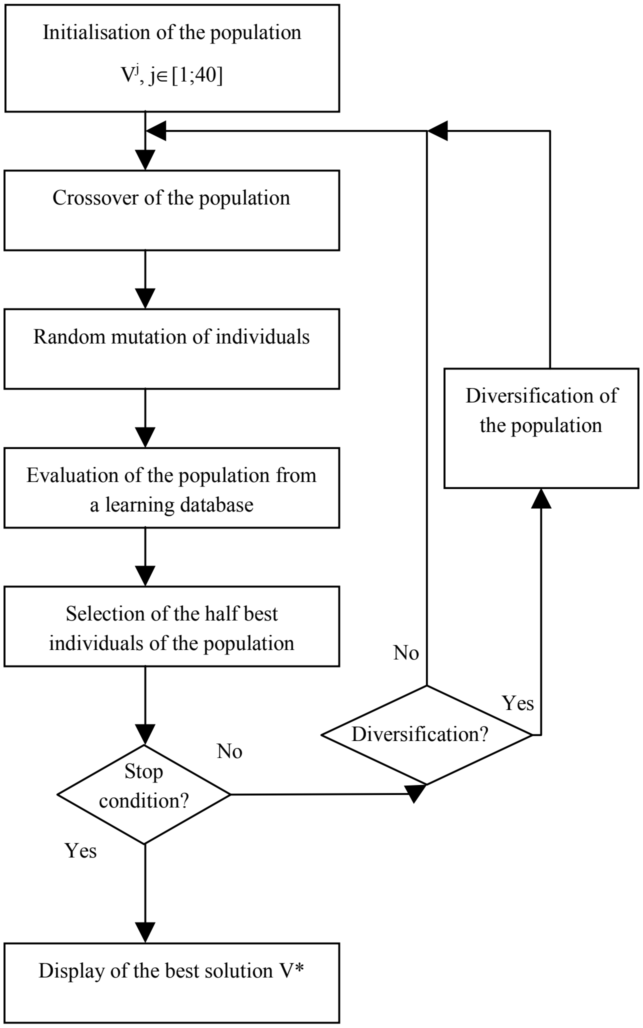

Initialisation of the population
V^j, $j\in[1;40]$
Crossover of the population
Random mutation of individuals
Evaluation of the population from a learning database
Selection of the half best individuals of the population
Stop condition?
No
Yes
Diversification?
No
Yes
Diversification of the population
Display of the best solution V*

Figure 3: A GA for a classification system of natural corks

5 EXPERIMENTAL RESULTS

5.1 RESULTS ON ARTIFICIAL DATA

In order to assess the approach just described, we apply the approach to a set of artificial, random data for which an optimal solution is known, that is, for each cork, we know its class. Using such a data set, we may compare directly the results of the GA with the optimal ones, consequently. These data are generated in the following way. We create a 2-dimentional N x M table with N=5000 (the number of theoretical corks) and M=61 (60 simulated numerical values that are usually given by 4 cameras plus the class of the cork).

More precisely, the first line is randomly computed and the following data are calculated from a function that takes into account the value of the cell of the first line and a random value. For each of the N lines, there are the 15 parameters given by each of the four cameras for each cork. We obtain the information on the four pictures of five thousand theoretical corks. In order to assign a cork to a class, we proceed as follows. We take randomly a combination for the thirty parameters V_i, $i\in[1;\ 30]$ used by the classification program AutoClass. We run then AutoClass with these parameters to classify all the 5000 corks. In this case, we know the class of each cork and we know also the parameters necessary to find this classification (These parameters may be considered to be optimal for the classification of these corks). Now we can run our GA on these data to see whether it is able to find these optimal (or near-optimal) parameters to classify correctly all the corks of these data.

We test the program on data sets with different sizes (50, 100, 200, 500, 1000 and 5000 corks). We run 10 times the algorithm on each data set. The tests were realized on a Pentium II with 200 MHz and 64 MB of RAM. The results are given in the following table.

Table 1: Solutions found for 10 different runs on theoretical corks

N = number of corks	Case where f = N	Case where f < N	Average value of f (in %)	Average solving time for one run
50	3	7	40/50 (80%)	1 min 14 s
100	3	7	73/100 (73%)	3 min 50 s
200	2	8	162/200 (81%)	7 min 28 s
500	3	7	465/500 (93%)	26 min 10 s
1000	1	9	873/1000 (87%)	59 min 13 s
5000	2	8	4533/5000 (87%)	5 h 50 min

(population size: 40, stop condition: 50 generations without improvement)

From table 1 we observe, for example, that with 200 corks, the algorithm finds twice out of ten the optimal solution (f = N), that is, it finds twice a combination of the classification parameters V_i that allows classifying correctly all the 200 corks. On average, the algorithm leads to a right classification for 162 of 200 corks (81%). The last column indicates the average time for a run.

Let us note that the resolution time increases according to the size of the data set. This increase is due to the evaluation step that uses an external classification program (AutoClass, see §4). The more important the data set is, the higher the evaluation time is.

This experiment is very satisfactory for a practical point of view. Indeed, it shows that the algorithm is able to find the best (optimal) solution at least once out of four in the previous example. Here, we can speak of the optimal solution because it is known and we know that it is possible to reach it. With real data we will see that this is no more possible because an optimal classification is not known in advance for a given set of corks. Moreover, it is almost impossible to classify a set of corks exactly in the same way as a human expert. We discuss this issue in the next section.

5.2 A CASE STUDY ON REAL DATA: THE CLASSIFICATION OF 173 CORKS

From a visual selection realized by a human expert, 173 corks were classified according to their heads into three classes. The following table gives the result of this manual classification done by the expert.

Table 2: Classification by an expert of a batch of 173 corks

	Class 1	Class 2	Class 3	*Total*
Quantity	70	46	57	*173*
Percentage	40.5 %	26.5 %	33 %	*100 %*

We analyze the corks of each class with the four cameras to extract the sixty parameters from the cork. The data are recorded in a 61-columns table. The class determined by the human expert is indicated in the 61st column. Then, we run our algorithm to determine the 30 classification parameters V_i, $i \in [1;30]$ such that the classification is the same as that determined by the human expert.

We run twenty times the algorithm before selecting the best solution. The results are summarized in table 3.

Table 3: Results of 20 runs on 173 real corks

	Maximal value of the fitness function f (correctly classified corks for the 173 corks)	Number of generations
Run 1	130	144
Run 2	129	239
Run 3	130	158
Run 4	130	252
Run 5	130	161
Run 6	127	109
Run 7	129	135
Run 8	130	194
Run 9	129	138
Run 10	130	194
Run 11	129	174
Run 12	127	168
Run 13	130	140
Run 14	130	197
Run 15	130	212
Run 16	128	197
Run 17	130	171
Run 18	130	188
Run 19	130	204
Run 20	129	168

The next figure (figure 4) shows the typical evolution of the fitness function of the best individual with the number of generations. From the figure, we observe that the fitness of the best individuals increases quickly for the first 60 generations. Then the evolution slows down and stops around 181 with a best fitness of 130.

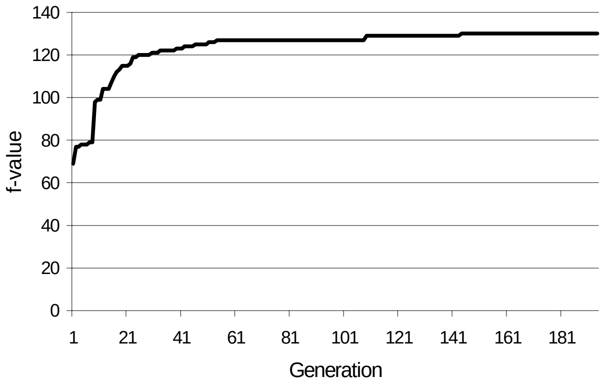


Figure 4: Evolution of the best individuals of the population

From these results, we know that the classification parameters determined by the GA allow 130 out of 173 corks to be classified as the human expert suggested. Now, we want to know exactly which cork is classified into which class. For this purpose, we take one of the best individuals (with $f_{max} = 130$). We re-run the classification program with the classification parameters given by the chosen individual. Applying to our 173 corks, we obtain the following results (table 4):

Table 4: Confusion matrix for a total of 173 corks.

Expert \ Machine	Class 1	Class 2	Class 3	Total
Class 1	**61**	8	1	*70*
Class 2	10	**23**	13	*46*
Class 3	7	4	**46**	*57*

Classified in the right class: 61 + 23 + 46 = 130

Satisfaction Percentage: 75.1%

From table 4, we can see that on the 70 corks that are classified by the expert in the class 1, 61 of them are classified by the classification system in class 1, 8 in class 2, and 1 in class 3. For the 173 corks, the algorithm leads to a classification that has an overlap of 75.1% with that of the human expert.

If we compare these results with those obtained on theoretical corks (§5.1), we may conclude that the results on real data are “less good”. Two factors can explain the difference between these two experiments. The first one is due to the classification made by the human expert (cf. table 2). Just like we realized a confusion matrix between a human expert and a classification program (cf. table 4), we also could realize a confusion matrix between two experts or with the same expert but in different conditions. Without any doubts, the traces of the matrix would never be equal to the number of corks to be classified. This result is well known in cork industry and certainly also in other domains that use the human intervention of man to classify products.

The second factor is a more bothering one that is related to the classification algorithm currently used (AutoClass). The data themselves we use may not allow classifying correctly the set of corks. Take an example with two variables, noted Var_1 and Var_2, and two classes to be separated: the circles and the triangles (cf. figure 5). There is an obvious manner to separate these elements: the straight θ. However, the classification algorithm AutoClass is not able to separate these elements by using ψ and φ (perpendicular to the axes represented by the variables). In the case presented here, there is no way to separate the two classes with ψ and φ.

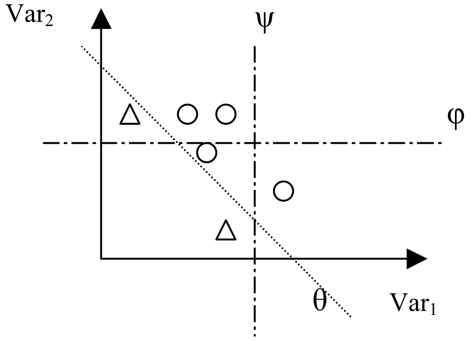


Figure 5: Separation of classes

These two factors explain the difference between the quality of theoretical data and the tested real data.

Let us mention that other tests have been carried out on very large set of non-classified corks (up to 15 000 corks). Assessed by human expert, the classification results on these real data are considered to the best one known today for the daily industrial classification task. For this reason, the system is currently used in daily operation.

6 CONCLUSIONS AND FUTURE WORK

The classification of natural corks is a very important topic in wine industry. In this paper, we have studied a parameter optimization problem for an automatic classification system. The problem involves thirty variables with a huge number (up to 10 000) of possible values for these parameters. To solve the problem, we have developed a GA-based approach to search for good combinations for the thirty parameters of the problem. The proposed approach has been evaluated on both (supervised) artificial data and real data. These evaluations have led to highly satisfactory and concluding results on the tested data. Moreover, results on unsupervised data were favorably approved by human expert and were the best ones known.

The analysis of results showed that it would still be possible to improve the effectiveness of the classification system by modifying other steps of the classification process (including the classification program used currently). One possibility would be to use a GA to find more pertinent classification rules. We studied in this paper the classification only according to the defects of the heads of the cork. Classification is also done using defects of boards of the cork. We would use the approach proposed in the paper to this kind of classification. Finally, we plan to apply the proposed approach to other classification problems encountered in wine industry. For example, for champagnes corks, one distinguishes even more classification steps: the classification of the two slices before pasting them, and the classification of corks according to the specification of customers.

Acknowledgements

This work was realized within the framework of a CIFRE convention between the group SABATÉ SA and the University of Angers. The authors would like to thank the "Agence Nationale de la Recherche Technique" (ANRT) for the financial support. Special thanks go to M. Bussac from SABATÉ S.A. for his help for this research and M. Hyppolyte and M. Nougalliat from VGA S.A. for their technical support about cork classification. We thanks also the reviewers of the paper for their comments.

References

[CHA97] CHANG J., HAN G., VALVERDE J.M., GRISWOLD N.C., DUQUE-CARRILLO J.F., SANCHEZ-SINENCIO E., Cork quality classification system using a unified image processing and fuzzy-neural network methodology, *IEEE Transactions on neural networks*. Vol. 8 No. 4, pages 964-974, 1997.

[FAL93] FALKENAUER E., GASPART P., Creating part families with a grouping genetic algorithm, *International Symposium on Intelligent Robotics*, India, 1993.

[FOU97] FOUCAULT V., Code international des pratiques bouchonnières, *Confédération européenne du liège*, 1997.

[GOL89] GOLDBERG D.E., *Genetic algorithm in search, optimization and machine learning*, Addison-Wesley Publishing Campany, Inc, 1989.

[HOL75] HOLLAND J.H., *Adaptation in natural and artificial systems*, The University of Michigan Press, 1975.

[MOL93] MOLINAS M., CAMPOS M., Aplicacion del analisis digital de imagenes al estudio de la calida del corcho, *Congreso forestal espanol*, Lourizan, Ponencias y Comunicaciones, Vol. 6, pages 347-352, 1993.

[PUN93] PUNCH W.F., GOODMAN E.D., PEI M., CHIA-SHUN L., HOVLAND P., ENBODY R., Further research on feature selection and classification using genetic algorithms, *ICGA93*, pages 557-564, 1993.

[SIE88] SIEDLECKI W., SKLANSKY J., On automatic feature selection, *International Journal of Pattern Recognition and Artificial Intelligence*, Vol. 2, No. 2, pages 197-220, 1988.

[TSA93] TSANG E., *Foundations of constraint satisfaction*, Academic Press, 1993.

[VAF92] VAFAIE H., JONG DE K., Genetic algorithms as a tool for feature selection in machine learning, *Proceedings of the Intl. Conf. on Tools with AI*, Arlington, VA, pages 200-204. IEEE CS Press, 1992 .

Evolutionary Optimization of Logic-Oriented Systems

Witold Pedrycz, Marek Reformat
Department of Electrical & Computer Engineering
University of Alberta
Edmonton T6G 2G7 Canada

Abstract

This study is concerned with an evolutionary methodology of designing logic-based models. These models dwell on a logic fabric of granular computing and learning capabilities of fuzzy neural networks. The proposed design comprises two fundamental phases, namely an evolutionary optimization (via Genetic Programming, GP) of the generic structure of the model that is followed by its parametric refinement completed in the form of a detailed gradient-based learning. We discuss the underlying algorithm and elaborate on the way in which GP helps cope with high dimensionality of the modeling problem (it is known that a significant number of variables leads to the failure of the parametric learning). The study is illustrated with the aid of a numeric example that provides a detailed insight into the performance of the logic-oriented models and quantifies crucial design issues.

1 INTRODUCTION

The main challenges of fuzzy (granular) modeling that are continuously facing this rapidly growing area remain the same as they were at the very inception of this paradigm. The agenda of granular modeling has to cope with two highly conflicting requirements such as developing models that are transparent yet accurate. Interestingly, neurofuzzy models (that form a significant trend) tend to gravitate towards addressing the requirement of high accuracy and this happens at a substantial expense of lowering their transparency. This is somewhat inevitable considering the underlying black-box processing paradigm and various topologies existing in neurocomputing. In many extreme cases, calling these constructs fuzzy models would be inappropriate, as fuzzy sets resulting through the optimization process may not exhibit any semantics. Moreover the basic computing carried out may be quite distinct from logic-based processing that is pertinent to computing with fuzzy sets.

Two general observations are essential in addressing the two modeling aspects being raised above

- To retain the transparency of the model that make it easily understood by an end user, we have to adhere to the solid logic-oriented structure of the model itself.
- The accuracy of the model and its high generalization capabilities call for a multiphase model design where naturally we start with a “skeleton” (more qualitative than quantitative) of the model and then proceed with its further numeric refinement. As the required optimization needs to be comprehensive including structural development of the model, there is an evident need for using mechanisms of global optimization such as evolutionary computing.

The objective of this study is to develop a hybrid design methodology of logic-based (fuzzy) modeling, come up with a logic-based structure of such models and propose a comprehensive evolutionary development environment in which the optimization of the models can be efficiently carried out both at the structural as well as parametric level. First, we exploit fuzzy neural networks [11] [12] that are aimed at capturing the essence of logic-oriented systems. Second, various topologies of these networks are developed through the use of Genetic Programming (GP) [1] [5] [7] that is one of the approaches readily available in evolutionary computing.

The experimental results illustrating the performance of the evolutionary fuzzy modeling include synthetic data (a multivalued logic function) and a case study, which exploits a Boston housing data. As far as a basic notation is concerned, we adhere to the one commonly used in fuzzy sets. In particular, fuzzy sets are denoted by capital letters. The basic logic operations on fuzzy sets are realized with the aid of triangular norms (t - and s - norms) [10].

2 THE ARCHITECTURE AND DEVELOPMENT PHASES OF LOGIC-BASED MODELS

The crux of the logic-based model discussed lies in its transparent form along with existing learning capabilities (structural and parametric plasticity). The form of the model is that of a series of rules involving information granules (fuzzy sets) that are combined *and*-wise to form a condition part of the rule and afterwards all the rules are combined *or*-wise. The overall architecture implies a certain development process of the model. At the level of structural optimization, we exploit evolutionary computing, especially Genetic Programming (GP). Furthermore the structural optimization is carried out independently from parametric optimization. By distinguishing between the structure and the parameters we attempt to concentrate on the topology of the model and make it disjoint (as much as possible) from the phase concentrated on parameter adjustment. By following this path, we search for the structure by exploiting the space of all possible structures, find the best and afterwards proceed with their refinement occurring at the parametric (numeric) level. There is a crucial reason behind the use of GP. First, the structural optimization is not supported by gradient-based techniques. Second, the space of the structures is large and this calls for the use of evolutionary techniques. As to the structure itself, we proceed with a standard two-level OR-AND representation of Boolean functions of symbols (in this phase fuzzy sets are used as symbols). Interestingly, this representation is in line with the well-known structures of rules (if-then statements) composed of fuzzy sets standing in their condition and conclusion parts. The OR-AND representation of the Boolean functions is equivalent to a logic network (combinational system). Third, the once the topology of the logic (Boolean) network has been established during the previous phase, the network is subject to some parametric refinement. To make this process possible, the network is augmented by modifiable connections and this gives rise to an idea of fuzzy neural networks [8]. In these networks two types of processing units (fuzzy neurons) are encountered. An OR neuron generalizes an *or*-type of aggregation. An *and*-type of aggregation is realized by using an AND fuzzy neuron. The connections of these neurons help calibrate the inputs and contribute to the improved performance of the model expressed at the level of the information granules (fuzzy sets) now being treated at the numeric end.

3 GP MODEL REPRESENTATION

The architecture of the fuzzy model follows the geometry of multidimensional data and reflects the main objective of such modeling that is to cover the data by a series of "patches" [13]. Each patch is a fuzzy relation formed with the use of fuzzy sets defined in each variable. Then a fuzzy model arises as a union of the patches. The geometry of the model implies its detailed architecture and dictates pertinent operational details.

As advocated in [10], fuzzy modeling or logic-based modeling is realized at the conceptual level formed by a collection of fuzzy sets defined in each variable. These are also regarded as linguistic landmarks whose choice implies a certain point of view at the data (system) under discussion. Each fuzzy set conveys a well-defined semantics. When dealing with many variables (that is usually the case), the fuzzy sets are aggregated and give rise to their granular manifestations in the form of fuzzy relations (Cartesian products of contributing fuzzy relations). As we require several patches, these are combined together by a union operation. This gives rise to a two-level topology of the model that captures the geometry of data, see Figure 1.

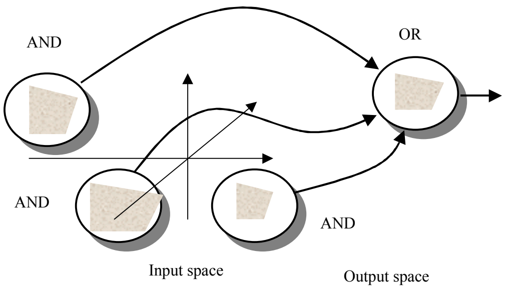


Figure 1. Geometry of data implying a topology of the model and its underlying logic fabric

Evidently, the geometry of the model stands in a one-to-one correspondence of its logic fabric. The essence of this geometry can be captured in the form of AND and OR nodes (aggregation operations) as illustrated in Figure 2. This figure emphasizes the structural nature of this construct. Considering the specific information granules shown there, we can translate it into the description

$$(A_1 \text{ and } B_3 \text{ and } C_4) \text{ or } (B_1 \text{ and } F_2) \text{ or } (A_2 \text{ and } G_2)$$

where each list is composed of fuzzy sets defined in the corresponding spaces (A, B, C,…). Each list includes a number of granules; their number could differ from list to list.

The list structure forms an essence of the model. It is worth stressing that even though the information granules convey detailed numeric information in the format of their membership functions, the resulting structure, Figure 2, does not include any other numeric quantification. A calibration of the structure is possible by equipping it with some parametric flexibility. The refinement of this nature is completed by introducing fuzzy AND and OR neurons [8] in place of the existing nodes. There exists a direct correspondence between these nodes and the fuzzy neurons. As a consequence, we come up with a fuzzy neural network whose learning is equivalent to the parametric optimization of the connections.

The network in Figure 2 pertains to a single information granule (fuzzy set) in the output space. The models with many outputs come in the form of a collection of such lists.

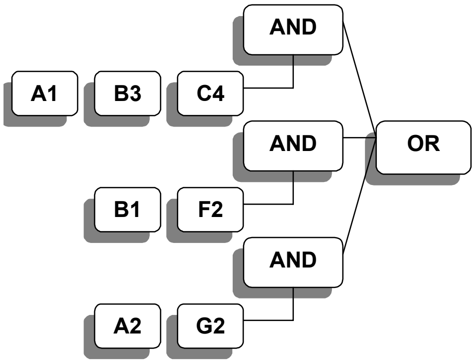


Figure 2. The structure of the logic-based model represented as a list of lists of information granules

4 FUZZY NEURAL NETWORKS: FROM A BINARY BLUEPRINT OF THE MODEL TO ITS PARAMETRIC REFINEMENT

The structure of the fuzzy neural network is fully determined by the logic fabric of the model. The ensuing learning of the network leads to its further refinements that appear at the numeric level. Proceeding with the architectural details, the fuzzy neural network is governed by the following expressions, refer also to Figure 3.

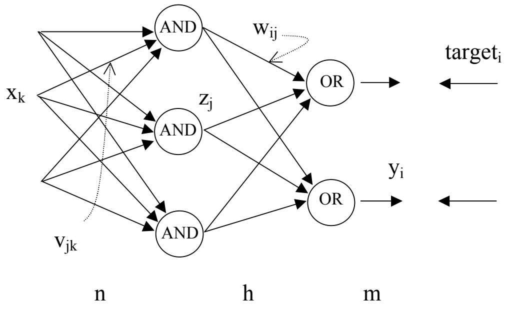


Figure 3. A structure of the fuzzy neural network along with a detailed notation

As we noted, the mapping from the structure to the fuzzy neural network is straightforward. Recall that an h-input single output OR neuron is described in the form

$$y_i = \mathrm{OR}\ (\mathbf{z}; \mathbf{w})$$

where $\mathbf{z}$, $y_i \in [0,1]$. The connections $w_1, w_2, \ldots, w_h$ are arranged in a vector form ($\mathbf{w}$). Rewriting the above expression in a coordinate wise manner we obtain

$$y_i = \mathop{\mathrm{S}}_{j=1}^{h} (z_j \ t\ w_{ij})$$

meaning that the neuron realizes an s-t composition of the corresponding finite sets $\mathbf{z}$ and $\mathbf{w}$.

The AND neuron $z_j = \mathrm{AND}\ (\mathbf{x}; \mathbf{v})$ is governed by the expression

$$z_j = \mathop{\mathrm{T}}_{k=1}^{n} (x_k \ s\ v_{jk})$$

Computationally, this neuron realizes a t-s composition of $\mathbf{x}$ and $\mathbf{v}$.

The role of the connections in both neurons is to weight the inputs and in this way furnish them with required parametric flexibility. A monotonicity property holds. In case of OR neurons, the higher the connection, the more essential the associated input. For AND neurons an opposite situation holds: lower connection indicates that the respective input is more essential. In general, a certain threshold operation can be sought. For any OR neuron, we consider the input irrelevant if the associated connection assumes values lower than 0.5. An input of the AND neuron is viewed irrelevant if the connection exceeds 0.5.

The learning is realized as a gradient-based optimization scheme [10]. The parametric learning of the fuzzy neural network has been well developed and documented in the literature [9] [10]. Several general observations are worth summarizing

- The gradient-based learning supports optimization that may result in a local minimum of the performance index. Global minimum could be out of reach of this learning mechanisms
- The efficiency of learning depends upon the choice of the triangular norms and co-norms. Here the minimum and maximum operators deserve particular attention as they lead to optimization traps. One of the reasons is that both minimum and maximum are non-interactive meaning that the results depends on an extreme value encountered there and the final outcome does not reflect the remaining arguments of these t- and s-norms. The other hand, for most other t-norms we may end up with a saturation effect that may be extremely difficult to handle in case of higher dimensions of the

problem. For instance, consider the product as a model of the t-norm. If the number of arguments increases, the result of aggregation carried out in this way tends to zero. Now if one envisions such an AND neuron located in the input layer of the fuzzy neural network and assume that all connections are the same and equal to zero, the output of the neuron reads as $z = \prod_{i=1}^{n} x_i$.

For any input less than one, say 1- γ we end up with the output equal to $(1-\gamma)^n$. One can easily check that a situation in which $\gamma = 0.5$ and n = 40 inputs produces the output of the neuron equal to $9.095 * 10^{-13}$. This activation level reduces quickly once the dimensionality of the problem goes up.

- The learning may be very slow especially when the size of the network gets large. A way in which the connections are initiated (random values) associated with no preliminary knowledge about the structure of the network (that implies its fully connected topology where all neurons are connected with the neurons in the neighboring layer), we are not guarded against the curse of dimensionality.

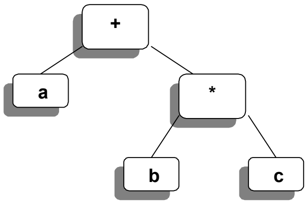
In light of these observations, the general design paradigm proposed in this study is strongly supported. Instead of learning the fuzzy neural network from scratch (the process which may fail quite easily), we concentrate first on establishing a structural blueprint of the network and then continue with the learning of the connections. Effectively, this skeleton of the network reduces the number of connections to be learned. The structural optimization of the network is out of reach of parametric (gradient-based) optimization and requires methods along the line of Evolutionary Computing [6] [14].

5 GENETIC PROGRAMMING AS A VEHICLE OF STRUCTURAL OPTIMIZATION OF THE NETWORKS

The algorithmic area of EC is diverse embracing a number of population-based optimization techniques such as Genetic Algorithms, Evolutionary Programming and Genetic Programming, to name a few of them. In this study, we concentrate on the use of Genetic Programming (GP) [7]. In comparison to Genetic Algorithms (that are indisputably the most commonly exploited in the area of fuzzy modeling), GP comes with greater flexibility and far lower brittleness that helps carry out an efficient search.

In what follows, we use a simple example making use of the logic structures (fuzzy neural networks) introduced in the previous section. This example will help explain the concepts of GP and underline any specific points arising in this setting. The fundamental point of evolutionary computing is in a population-based optimization [1] [5] [7] and this aspect is retained in GP.

GP can be seen as an extension of genetic paradigm into the area of programs. It means, that objects, which constitute population, are not fixed-length character strings that encode possible solutions to the given problem, but they are programs, which "are" the candidate solutions to the problem. In general, these programs are expressed as parse trees, rather than as lines of code. For instance, the simple program "***a + b*c***" would be viewed in the following way:



Such representation of candidate solutions combined with some constrains regarding their structure allows for straightforward representation of fuzzy models such as fuzzy neural networks.

GP operates on a population of lists, which are blueprints of fuzzy models. In other words, each individual of population – a list – represents a single fuzzy model, refer to Figure 2. A fuzzy neural network of single output is a tree with an OR node as the root, AND nodes at the first level, and nodes representing inputs at the second level. Such structure is presented in Figure 6. The OR and AND nodes can have multiple inputs. Additionally, in order to represent fuzzy neural networks with multiple outputs, a single AND node can be connected to more than one OR node.

A population of fuzzy models evolves according to the rules of selection and genetic operations such as crossover and mutation. Each individual in the population is evaluated by means of a certain fitness function. Based on this a selection procedure is performed. In this process individuals are chosen to form the next population. The choice is made on the basis of favoring individuals with higher fitness values [1] [5] [7].

Crossover and mutation are the two standard operations leading to the search of the solution space (viz. the space of the logic – based models, i.e. , a collection of lists). The role of the fitness function is to assess how well the model matches the experimental data. We consider the fitness function regarded as a sum of squared errors

$$Q = \sum_{k=1}^{N} (F(k) - \hat{F}(k))^T (F(k) - \hat{F}(k))$$

with N being the number of data points used for training. $\mathrm{F}(k)$ and $\hat{\mathrm{F}}(k)$ are the outputs of the model and target values, respectively. The dimensionality of F depends on the number of outputs (m) of the model.

6 THE DETAILED DESIGN PROCESS OF THE FUZZY MODEL

As we have already discussed the main phases of fuzzy modeling in the evolutionary setting, they can be put together in a form of a coherent design platform. In particular, it is essential to elaborate on the computational interfaces between the successive phases.

Selection of fuzzy sets Fuzzy sets serve as information granules quantifying a given variable (input or output). We choose these fuzzy sets in advance and keep them unchanged during the successive phases of the model development. There are two main reasons behind this. First, fuzzy sets are semantically sound constructs that have to retain key properties including well-delineated identity [10]. Their number should be limited to a few in order to allow for their linguistic interpretation (such as small, medium, etc). This means that if fuzzy sets are to be involved in the optimization process all these semantic integrity requirements should be maintained and this is not straightforward. Second, because of the overlap of successive fuzzy sets, we maintain continuity between changes of position of fuzzy sets and the amount of data embraced by them. In this sense, some changes to the position of the fuzzy sets as well as their parameters will not cause abrupt changes in the performance of the fuzzy model. In this sense we may anticipate that for a fixed collection of fuzzy sets, we may realize an efficient optimization of the model through structural optimization.

Structure optimization The structure of the family of lists becomes a point of optimization at this phase of model development. A formation of these lists is about a structure of the model (more specifically, the form of the patches covering the data). The GP terminates once a fitness function does not change its values.

The structure of the resulting network (a collection of lists) obtained through GP may not be unique. More than that: it is unlikely to get the same structure for optimal structures, as the data set is quite sparse in the space of fuzzy sets. For instance, for "n" variables and "p" fuzzy sets defined in each space we end up with p^n combinations (Cartesian products) of fuzzy sets. A lot of don't care conditions are present in the space. GP attempts to use them in order to come up with a simple logic expression for the data set yet their usage is not unique.

Parametric optimization of the network The topology of the network derived during the structural optimization completed by GP is now refined through learning of the induced fuzzy neural network. This network maps directly the collection of lists formed by GP. As a result the fuzzy neural network is not fully connected. The significant initial connections are those one that are identified by GP. In the sequel only those are modified. Evidently this selection reduces a size of the learning problem as we concentrate only on a subset of the connections. This reduction is especially visible for AND neurons where the number of input variables has been confined to a small fraction of all inputs.

To underline that only selected connections are modified, we introduce a mask **M** that allows the connections that are not masked to be adjusted. In other words, the original update formula reads as

$$[\mathbf{connections}\,(\mathrm{iter}+1)]_{\mathbf{M}} = [\mathbf{connections}(\mathrm{iter})]_{\mathbf{M}} - \alpha \nabla_{\mathrm{connections}} Q$$

The network can be represented in an equivalent rule-based format

- if condition$_i$ and condition$_j$ and … then conclusion$_l$

The format of the rules varies as each rule may have a different number of conditions. In this setting, the connections of the fuzzy neural network can be interpreted as calibration factors of the conditions and rules

- the connections of the AND neuron modify the membership functions of the fuzzy sets contributing to the Cartesian product of the overall condition part of the rule. For instance, the expression $(A_1 s w_1)$ t $(B_3\ s\ w_2)$ can be interpreted as an *and* combination of the modified (less specific) fuzzy sets A_1' t B_3' where A_1' is a modified version of A_1, i.e. $A_1' = A_1\ s\ w_1$. Similarly, we get a modified version of B_3, that is $B_3\ s\ w_2$. The higher the value of the connection, the less specific is the modified fuzzy set. We have $A_1\ s\ w_1 \geq A_1$. In limit, when the connection is equal to 1, we end up with A_1 being eliminated from the rule (in this way the rule becomes more general).
- The connections of the OR neuron determine confidence of the rule meaning that the Cartesian product (overall condition of the rule) is quantified in terms of its relevance.

7 EXPERIMENTAL STUDIES

Two datasets are used in the experimental part of the study. In the first experiment we exploit some synthetic data representing some multivalued logic function. The second one, known as Boston housing data (http:// www.ics.uci.edu/~mlearn/ MLSummary.html) concerns a description of real estate in the Boston area where housing is characterized by a number of features including crime rate, size of lots, number of rooms, age of houses, etc. and median price of houses.

In all the experiments we use GP as an environment of evolutionary optimization. The parameters of the GP are as follows: population size: 200, number of generations: 1000, probability of crossover: 0.9, selection of crossover points (i.e. AND level vs. inputs): 0.5, probability of mutation: 0.1, generation of initial population: a grow method (lists have variable length), selection method: fitness-proportionate reproduction, elitist strategy. Maximal sizes of lists used in the experiments: number of ANDs – 15 in the first experiment and 5 in the second, maximal number of inputs to AND nodes – 10 in the first and 5 in the second experiment.

7.1 MULTIVALUED LOGIC FUNCTION

Here we consider a multivalued XOR function $[0,1]^n \to [0,1]$

$$y = \phi_1 x_1 \oplus \phi_2 x_2 \oplus ... \oplus \phi_n x_n$$

where the logic operations (*and* and *or*) are realized by means of some t- and s-norms (s-norm: probabilistic sum, t-norm: product). Some out of 20 (n=20) variables contribute to the function (as indicated by the indicator function ϕ_i being equal to 1 if x_i contributes to the XOR function and 0 otherwise. More specifically, there are five variables contributing to the output. The training set consists of 300 data points. Owing to the dimensionality of the problem, the FNN is not successful in completing a parametric learning. Then we confine ourselves to the GP optimization. The structural optimization reduced the performance index from 22.5861 to 5.9928 in 1,000 generations, refer to Figure 4.

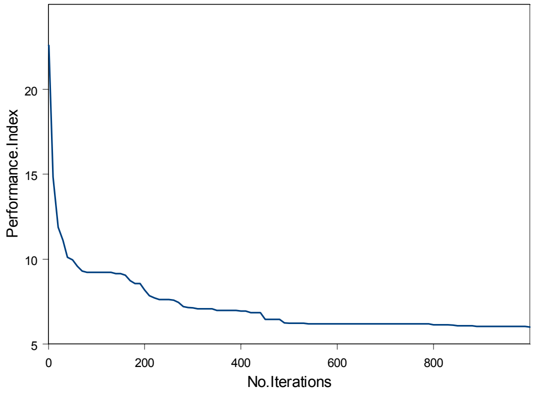


Figure 4. Performance index in successive generations of GP

The optimal structure of the model is the following

x8 x10 x2’
x4 x13 x19 x12’ x14’ x17’
x9 x2’ x12’ x15’ x19’
x2 x12’ x13’ x16’ x19’
x8 x14 x19 x1’ x12’ x17’ x20’
x12 x2’ x3’
x1 x15 x8’
x2 x19 x1’ x12’
x1 x4’ x8’ x12’ x14’ x15’ x16’ x17’
x1 x3 x8’
x8 x19 x1’ x4’ x7’ x12’ x14’ x16’
x3 x8 x10’ x12’

(in the above expressions x’ stands for a complement of the variable, x’=1-x, variables of a single line represent operands of a t-norm operation, and all lines are operands of s-norm operation). The parametric optimization leads to some further improvement of the model by reducing the performance index to 2.6002.

7.2 BOSTON DATA HOUSING

The Boston dataset consists of 504 14-dimensional points, each representing a single attribute. The construction of the fuzzy model is completed for 336 data points treated as a training set (the rest of the data set is retained for testing purposes). The number of the fuzzy sets defined in each input space (variable) is equal to 3 while for the output space we define 2 fuzzy sets. All fuzzy sets are Gaussian, uniformly distributed in the space and with an overlap of 0.5 between two successive fuzzy sets.

The results of the structural learning process are shown in Figure 5 where the values of the performance index Q in successive generations are presented. The optimal structure is shown in Figure 6. It contains only 5 out of 13 attributes of Boston data. The remaining have been found to be of lower relevance.

The normalized performance index $\tilde{Q}$ (that is the value of Q divided by the number of data points) of the optimal structure is equal to 0.0321. While using a testing dataset, the value of the normalized performance index raises a bit and now equals to 0.0363.

The network is then optimized parametrically through a parametric learning of the corresponding fuzzy neural network. The learning rate (α) is set to 0.005 and the method is run for 1,500 learning epochs. As a matter of fact, most of improvement happened at the beginning of the learning process.

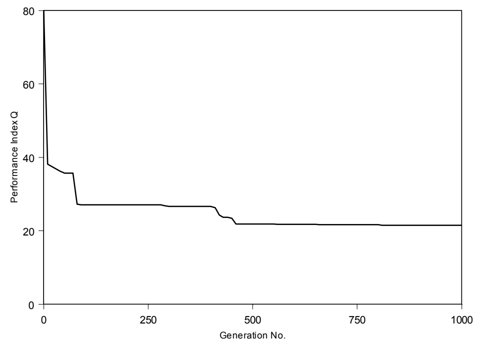
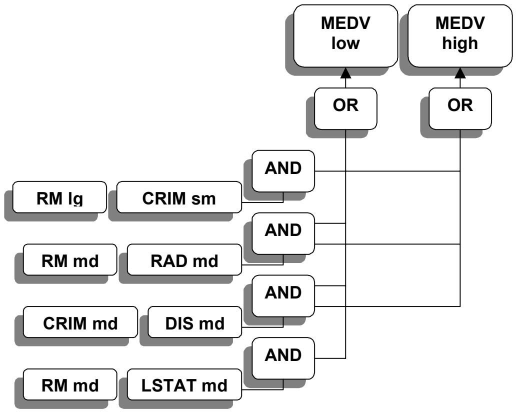


Figure 5. Performance index Q showing GP optimization



CRIM: per capita crime rate by town, RM: average number of rooms per dwelling, DIS: weighted distance to five Boston employment centers, RAD: index of accessibility to radial highways, LSTAT: %lower status of the population, MEDV: median value of owner-occupied homes
sm: small, md: medium, lg: large

Figure 6. An optimal structure of the two-output network derived through GP optimization

The normalized performance index $\tilde{Q}$ of the optimal fuzzy neural network, after structural and parametric optimization, is equal to 0.0173, and for the testing set becomes equal to 0.0167. The improvement after the parametric learning accounts for 46% (training) and 54% (testing) of the initial value of the performance index (the one after the structural optimization)

The parametrically optimized structure is shown in Figure 7; essentially it is the same as in Figure 6 but now being augmented by the values of the connections. It is easy to observe how gradient-based learning process changed the significance of some input sets and the rules. The dominant *if-condition* for membership function *low* is "**CRIM** is *medium* AND **DIS** is *medium*", and in the case of function *high* the essential *if-condition* is "**RM** is *large* and **CRIM** is *small*".

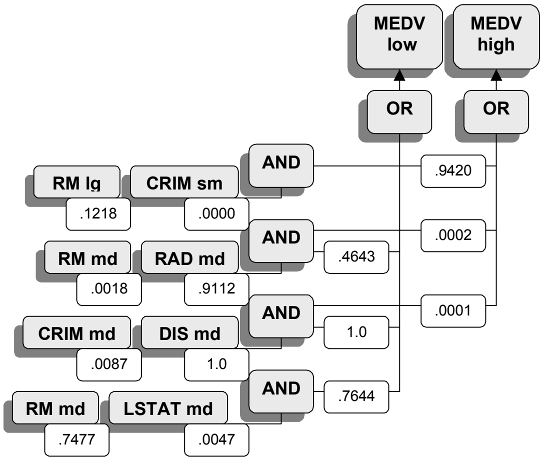


Figure 7. Structure of the two-output fuzzy neural network after parametric optimization

8 CONCLUSIONS

In this study, we have proposed a general design methodology for fuzzy models. The three-phase development process conforms to the two fundamental requirements of granular modeling that is accuracy and transparency. The optimization tandem of evolutionary computing (more specifically, genetic programming) and gradient-based learning of fuzzy neural networks naturally supports structural and parametric optimization of the models that helps us achieve accuracy of the overall model. The transparency of the model is accomplished by subscribing to the logic-oriented architecture of the fuzzy neural networks. The proposed methodology fully applies to highly dimensional modeling and comes as a remedy to the curse of dimensionality associated with rule-based fuzzy models.

There are several possible extensions worth considering. First, the fuzzy sets may be constructed by capturing the nature of the data. This could be done by various techniques of fuzzy clustering [12]. Second, it is worth investigating various architectures of fuzzy neural networks.

Acknowledgments

Support from the Natural Sciences and Engineering Research Council (NSERC) and Alberta Software Engineering Research Consortium (ASERC) is gratefully acknowledged.

References

[1] T.Back, D.B. Fogel, Z. Michalewicz (eds.), *Evolutionary Computations I*, Institute of Physics Publishing, Bristol, 2000.

[2] S. Bengio, Y. Bengio, and J. Cloutier, Use of genetic programming for the search of a learning rule for neural networks, in *Proceedings of the First Conference on Evolutionary Computation*, IEEE World Congress on the Computational Intelligence, 1994, pp. 324-327.

[3] A. Bastian, Identifying fuzzy models utilizing genetic programming, *Fuzzy Sets and Systems*, 113(3), 2000, pp. 333-350.

[4] A.I. Esparcia-Alcázar, and K.C. Sharman, Evolving Recurrent Neural Network Architectures by Genetic Programming, *Genetic Programming 97: Proceedings of the Second Annual Conference*, Stanford University, USA, July 1997, pp.89-94.

[5] D.B. Fogel, *Evolutionary Computation, Toward a New Philosophy of Machine Intelligence*, IEEE Press, Piscataway, 1995.

[6] P. G. Korning, Training neural networks by means of genetic algorithms working on very long chromosomes, *International Journal of Neural Systems*, 6(3), 1995, pp. 299-316.

[7] J.R. Koza, J. R., *Genetic Programming*, The MIT Press, 1992.

[8] W. Pedrycz, *Computational Intelligence: An Introduction*, CRC Press, 1997.

[9] W. Pedrycz and F. Gomide, *An Introduction to Fuzzy Sets; Analysis and Design*, The MIT Press, 1998.

[10] W.Pedrycz, *Fuzzy Sets Engineering*, CRC Press, Boca Raton, FL, 1995.

[11] W. Pedrycz, Fuzzy equalization in the construction of fuzzy sets, *Fuzzy Sets & Systems*, to appear.

[12] W. Pedrycz, Conditional fuzzy clustering in the design of radial basis function neural networks, *IEEE Transactions on Neural Networks*, 9(4),1998, pp. 601- 612.

[13] M. Setnes, Supervised fuzzy clustering for rule extraction, *IEEE Transactions on Fuzzy Systems*, 8(4), 2000, pp. 416-424.

[14] X.Yao, Evolving Artificial Neural Networks, *Proceedings of IEEE*, 87(9), 1999, pp.1423-1447.

[15] B. Zhang, and Muehlenbein, Synthesis of sigma-pi neural networks by the breeder genetic programming, in *Proceedings of IEEE International Conference on Evolutionary Computation (ICEC)*, World Congress on Computational Intelligence, Orlando, Florida, USA, 1994, IEEE Computer Society Press, pp. 318-323.

Aircraft Ground Traffic Optimisation using a Genetic Algorithm

Brankica Pesic
Faculty of Transp and Traffic Eng
University of Belgrade
Vojvode Stepe 305, 11000 Belgrade, Yu
381 11 3091 352
dean@sf.sf.bg.ac.yu

Nicolas Durand
CENA
7, av Ed Belin
31055 Toulouse, France
(33) 562 17 40 54
durand@tls.cena.fr

Jean-Marc Alliot
CENA
7, av Ed Belin
31055 Toulouse, France
(33) 562 17 41 24
alliot@dgac.fr

Abstract

The development of air traffic during the last years, has greatly increased the density of aircraft in the airspace, and congestion on major airports. Indeed, on many airports, the taxi operation of aircraft between parking positions and runways, causes delays. The problem is increased by the development of hubs. In this article, a taxi optimisation tool using a Genetic Algorithm is introduced and tested on Roissy Charles De Gaulle Airport. The tool can help choosing the best taxiways to reduce the time spent from the gate to the runway or the runway to the gate, respecting the separation with other aircraft. It can also help choosing one way taxiways regarding to traffic and wind, and also measuring the impact of opening a new taxiway or closing an existing taxiway. Simulations are presented on a one day traffic at Paris Roissy. Delays are correlated to the traffic density on the airport.

1 Introduction

Development of hubs have generated new problems for ground operations, as all aircraft are tending to move at the same time on the airport. Thus, new delays are introduced on major airports due to ground congestion. Airport designers are tempted to build new taxiways to reduce congestion and improve the efficiency of ground operations, but by the moment, no tool can help them to measure the efficiency of their choices.

As many research projects are concentrated on decision making tools for airspace controllers, little work has been done on ground control. The SIMMOD[1] project developed by the FAA[2] is a heavy software that was not designed to give any advice to ground controllers. The SMA[3] project was developed by the FAA and NASA[4] to help current airport facilities to operate more efficiently. Many efforts were concentrated on improving the information sharing of the different operators on the ground. The DP[5] project ([IDA+98]) only focuses on improving the performance of departure operations, without taking into account the taxi problem. The TAAM[6] project ([Gro99]) is developed by The Preston Group. Trajectory optimisation partly exists and it uses notions of reachable gates. The conflict detection and resolution is not developed. Finally, a component of the TARMAC[7] project, developed by the DLR[8] Institute of Flight Guidance, focuses on the ATC[9]-related traffic planning systems for airport movements, but does not introduce any optimisation tool to taxi aircraft.

In this article, a taxi optimisation tool is introduced and tested on Roissy Charles De Gaulle Airport. The tool chooses the best trajectory to reduce the time spent from the gate to the runway or the runway to the gate, respecting the separation with other aircraft. It can also help choosing one way taxiways regarding to traffic and wind, and also measuring the impact of opening a new taxiway or closing an existing taxiway. The problem is introduced and modelled in the first part. The different algorithms used to solve the

---

[1] Simulation Model (http://www.atac.com/simmod/)
[2] Federal Aviation Administration
[3] Surface Movement Adviser (http://surface.arc.nasa.gov/sma/)
[4] National Aeronautics and Space Administration
[5] Departure Planer
[6] Total Airspace and Airport Modeller
[7] Taxi and Ramp Management And Control (http://dv.bs.dlr.de/ff/fl/24/tarmacs-as)
[8] Deutsches Zentrum fur Luft und Raumfahrt
[9] Air Traffic Control

problem are detailed in part 2. The last part gives the results of a full simulation on a one day traffic on Roissy Airport.

2 Problem modelling

The problem is to find for each aircraft an optimal path from its parking to a given runway holding position or from its runway exit to its parking, and respect a given separation between aircraft.

An optimal path can have different definitions as for example the length of the path or the total taxiing time. Holding on a taxiway can be more or less penalising than increasing the length of the path. It can be cheaper to hold at the parking position than on a taxiway.

It can be better, for example, to lengthen slightly the routes of two aircraft than to make one aircraft wait a long time. Therefore a global optimum criteria will have to be defined in the following. However, the purpose of this article is not to discuss the choice of such criteria, which depend on many different factors related to the airport geometry, the traffic, and airlines preferences…

By the way, it is quite difficult to predict with a good accuracy the future positions of aircraft on taxiways. First of all, the exact departure time is generally known only a few minutes in advance (many factors can cause delays), and the exact landing time depends on the runway sequencing. A modelling that can afford these uncertainties is necessary in order to build a realistic tool.

2.1 Airport structure

The airport is defined by a graph: links represent taxiway segments whereas nodes are taxiway intersections or connections, parking, holding positions, and runway exits. Figure 2 represents the graph of Roissy airport. The graph is obviously connected. A Dijkstra algorithm [AMO93] can be used to compute the minimum length from any node of the graph to every parking, holding point, or runway exit. An A^* algorithm [Pea84] can as well be used to compute the minimum length from any node of the graph to every parking, holding point or runway exit, taking into account the limitations in terms of turning rate. Therefore, extra time is added depending on the turning rate (see figure 1).

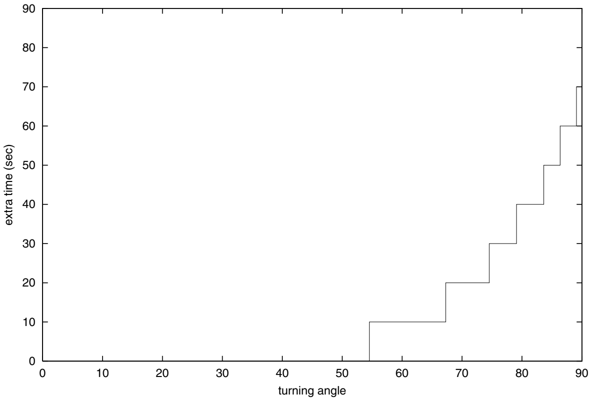


Figure 1: Additional delay as a function of the turning angle

2.2 Aircraft possible manoeuvres

In order to minimise the delays and ensure the separations, the path of aircraft can be modified, or aircraft can hold position at their parking, on taxiways or at the holding point before taking off. Two aircraft are in conflict if the distance between them is less than 60 meters at every time, except on the parking position on which this distance can be reduced. In order to have simple manoeuvres, only one holding order should be given to the pilot at a time (starting at t_0 and ending at t_1). The path should also respect some constraints: turning angles are limited by the aircraft performance, an aircraft should not use the same taxiway twice in the same direction, there cannot not be more than one aircraft on a runway at the same time …In order to simplify the problem, aircraft are supposed to have constant speed except when they turn.

Alternate paths lengthening the trajectory less than a certain distance can be computed with a simple Branch and Bound algorithm [HT95].

Figure 2 gives an example of the shortest path calculated between a runway exit and a gate. The 467 alternate paths lengthening the trajectory less than 500 meters are also represented.

2.3 AGTO modelling

As the aircraft future positions and movements are not known with a good accuracy, it is necessary to regularly update the situation, every Δ minutes for example. By the same time, looking a long period ahead is not possible as predictions are not good enough. Consequently a time window $T_w > \Delta$ is defined.

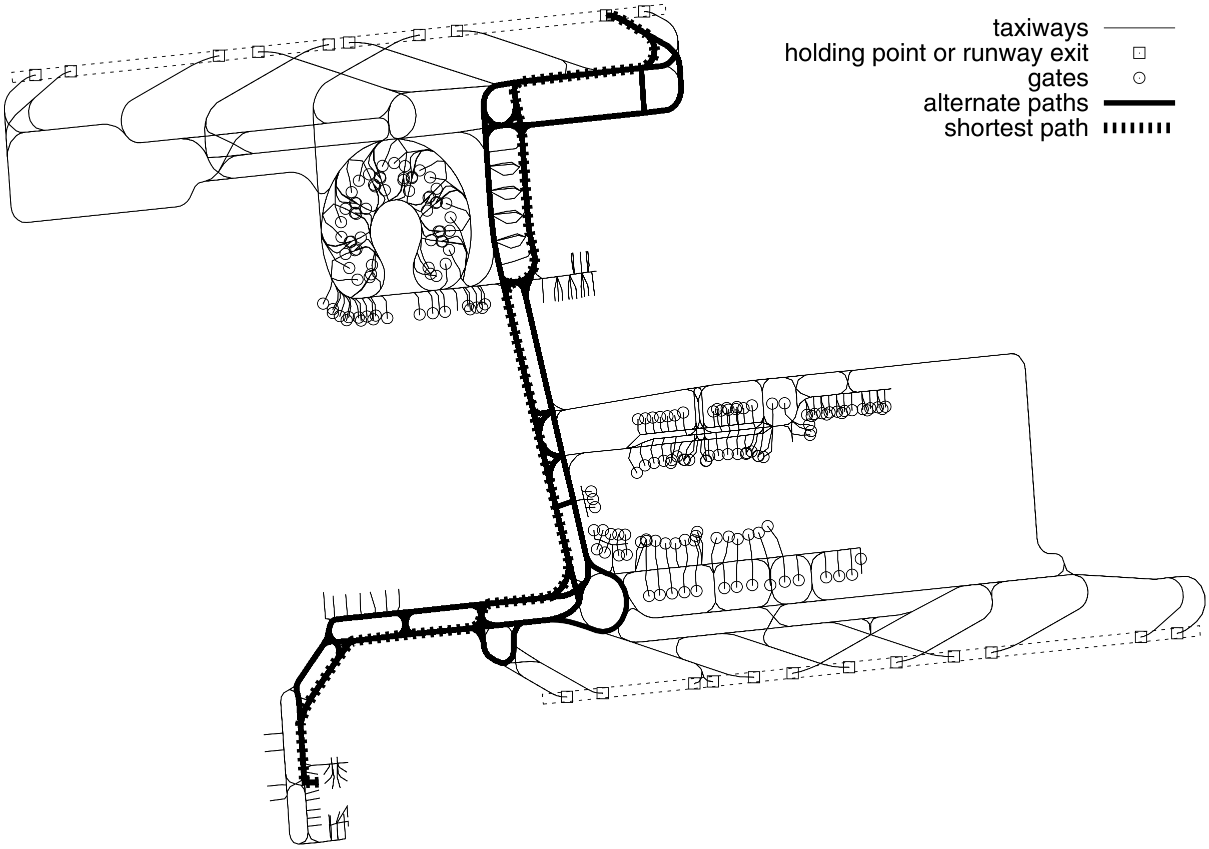


Figure 2: Roissy airport graph - Example of shortest and alternate paths

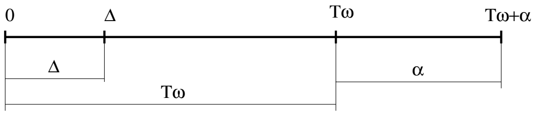


Figure 3: Time window

Only aircraft arriving or taking off in the time window will be considered. The time window will be shifted every Δ minutes, the problem reconsidered and a new optimisation performed. As an aircraft may taxi more than T_w minutes, the horizon effect problem can appear. Indeed, as we only look at aircraft during the T_w period, a conflict may appear a few seconds after the end of the T_w period without being detected. In order to prevent the optimisation process from building solution that would not be acceptable at the next shift, the time window is increased of α seconds during the optimisation process (see figure 3).

3 GAs applied to AGTO

In this paper, classical Genetic Algorithms and Evolutionary Computation principles such as described in the literature [Gol89, Mic92] are used. The algorithm is used every Δ minutes on the problem defined in section 2.3.

3.1 Data structure

During each optimisation process, each aircraft trajectory is described by 3 numbers (n, t_0, t_1). n is the number of the path : as detailed in section 2.2, all the alternate paths lengthening the aircraft trajectory less than some distance can be initially computed and sorted. The aircraft may hold position at t_0 and resume taxi at t_1 (if $t_0 = t_1$, the aircraft does not stop). When N aircraft are simultaneously taxiing, the problem is defined by $3N$ variables.

3.2 Fitness function

The fitness function must ensure that a solution without any conflict is always better than a solution with a conflict. Consequently it was decided that the fitness of a solution without conflict should be less than $\frac{1}{2}$ and the fitness of a solution with a conflict more than $\frac{1}{2}$. The different conflicts between each pair of aircraft can be initially computed in a $(n \times n)$ matrix (see table 1). A conflict during 3 time steps between aircraft i and j sets elements (i, j) and (j, i) to 3. Element (i, i) is filled with the trajectory lengthening due to the path chosen and holding time $(t_1 - t_0)$.

	(1)	...	...	(j)	...	(i)	...	...	(n)
(1)	80	0	0	0	0	0	0	0	0
(2)	0	0	0	0	0	0	0	0	0
...	0	0	30	0	0	0	0	0	0
(j)	0	0	0	0	0	3	0	0	0
...	0	0	0	0	25	0	0	0	0
(i)	0	0	0	3	0	0	0	0	0
...	0	0	0	0	0	0	0	0	0
...	0	0	0	0	0	0	0	0	0
(n)	0	0	0	0	0	0	0	0	0

Table 1: Fitness matrix

Using the fitness matrix M_f, it is possible to compute the fitness value as follows :

If the matrix is diagonal :

$$F = \frac{1}{2} + \frac{1}{1 + \sum_{i=0}^{n} M_f(i,i)}$$

Else :

$$F = \frac{1}{2 + \sum_{i<j} M_f(i,j)}$$

3.3 Crossover operator

The conflict resolution problem is partially separable as defined in [DA98, DAN96]. In order to increase the probability of producing children with a better fitness than their parents, principles applied in [DA98] were applied. For each aircraft i of a population element, a local fitness F_i value is defined as the sum of the i^{th} line (or column) of the fitness matrix (except the diagonal element).

$$F_i = \sum_{j \neq i} M_f i, j$$

The crossover operator is presented on the figure 4. First two population elements are randomly chosen. For each parent A and B, fitness A_i and B_i of aircraft i are compared. If $A_i < B_i$), the children will take aircraft i of parent A. If $B_i < A_i$, the children will take aircraft i of parent B. If $A_i = B_i$ children randomly choose aircraft A_i or B_i or even a combination of A_i and B_i.

3.4 Mutation operator

For each candidate to mutation, parameters of an aircraft having one of the worst local fitness are modified (n, t_0, t_1). If every conflict is solved, an aircraft is randomly chosen and its parameters changed.

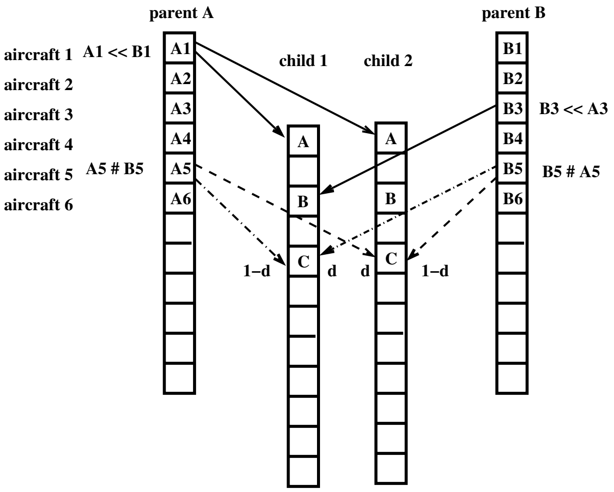


Figure 4: Crossover operator

The crossover and mutation operators are quite deterministic at the beginning because there are many conflicts to solve. They focus on making feasible solutions. When the solutions without conflict appear in the population, they become less deterministic.

3.4.1 Sharing

The problem is very combinatorial and may have many local optima. In order to prevent the algorithm from a premature convergence, the sharing process introduced by Yin and Germay [YG93] is used. The complexity of this sharing process has the great advantage to be in $n\log(n)$ (instead of n^2 for classical sharing) if n is the size of the population.

A distance between two chromosomes must be defined to implement a sharing process. Defining a distance between two sets of N trajectories is not very simple. In the experiments, the following distance is used introduced:

$$D(A,B) \quad = \quad \frac{\sum_{i=0}^{N} |l_{A_i} l_{B_i}|}{N}$$

l_{A_i} (resp l_{B_i}) is the i^{th} aircraft path length of chromosome A (resp B). As the paths are sorted according to their length, the distance increases with the difference of lengths.

3.5 Ending criteria

As time to solve a problem is limited, the number of generations is limited, as follows: as long as no available solution is found, the number of generation is limited to 100. The algorithm is stopped 20 generations after the first acceptable solution (with no remaining conflict) is found.

4 Experimental results

The experimental results presented in this section have been computed with real flight plans on a complete day at Roissy Airport (May 22^{nd} 1999). During that day, some aircraft land, other aircraft take off and some aircraft land and take off. Aircraft are assigned to terminals according to the airline they belong to (for example an Air France flight is assigned to Roissy 2).

When taking off or landing, aircraft are randomly assigned one of the two runways. They are sequenced on runways every minute using the first in first out principle.

Three hypotheses are done:

- in the **"random hypothesis"**, taking off and landing aircraft are randomly allocated both runways.
- in the **"deterministic hypothesis"**, taking off and landing aircraft are allocated the runway that minimises the distance to the allocated parking.
- in an **"middle hypothesis"**, taking off aircraft are randomly allocated both runways and landing aircraft are allocated the runway that minimises the distance to the parking.

The three hypotheses are tested with the genetic algorithm. The last hypotheses is tested with a 1-to-n strategy that uses an A^* algorithm: aircraft are sorted according to their time of departure or arrival, each aircraft trajectory is then optimised considering previous aircraft trajectory as a constraint.

4.1 Parameters

- $T_w = 12mn$
- $\alpha = \Delta = 3mn$
- Population size: 300
- Max number of generations: 100
- Crossover rate: 60%
- Mutation rate: 15%
- Selection principle: stochastic reminder without replacement

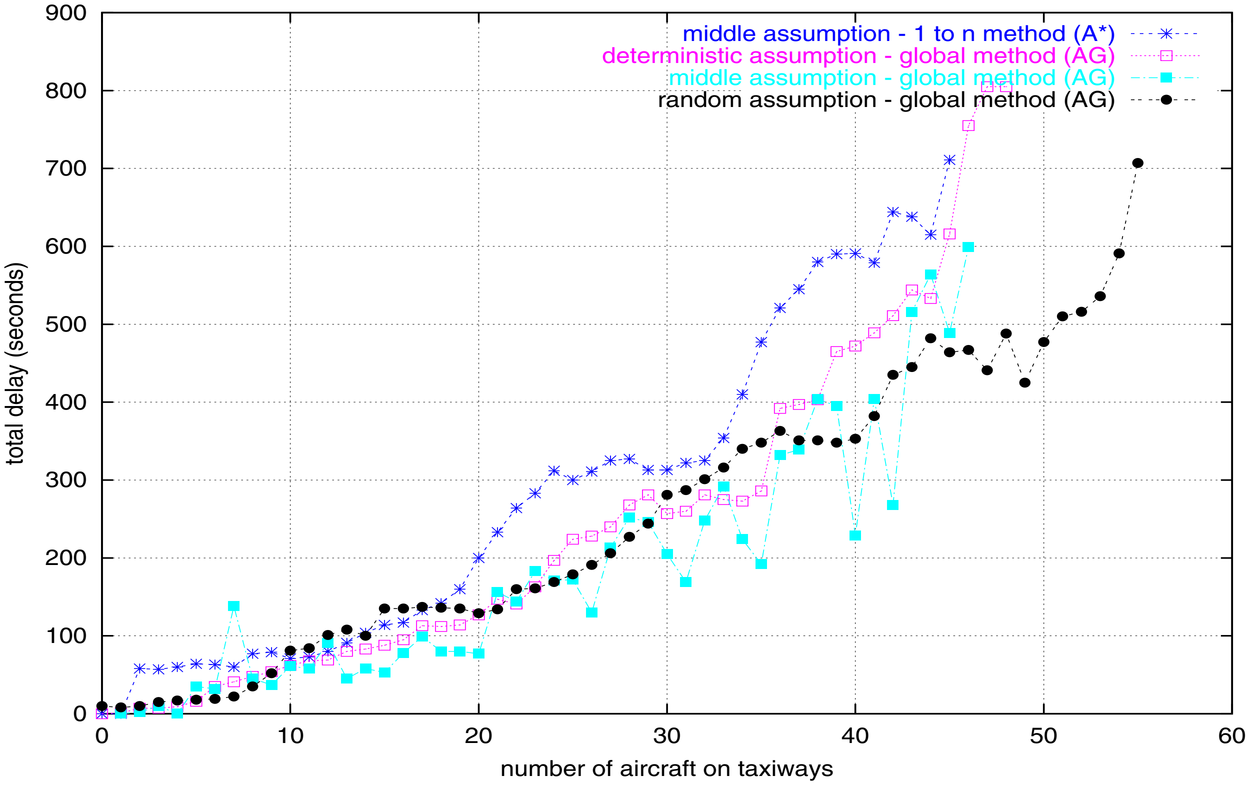


Figure 5: Total delay as a function of the number of aircraft on taxiways.

Hypothesis	mean delay	maximum number of aircraft
random (GA)	255	55
deterministic (GA)	198	48
middle (GA)	195	46
middle (A^*)	271	45

Table 2: Mean delay and maximum number of aircraft for the different hypotheses

4.2 Comparing 1-to-n to the global strategy

Figure 5 gives the mean delays as a function of the number of aircraft moving on the taxiways for the different hypotheses. The 1 to n method using an A^* algorithm produces more delays than the global method using the Genetic Algorithm, whatever the chosen hypothesis.

Table 2 gives for the different hypotheses the mean total delay and the maximum number of aircraft simultaneously moving.

The middle hypothesis (GA) penalises less aircraft than the other hypotheses and a smaller number of aircraft are moving at a time. The random hypothesis (GA), which is probably more in accordance with reality (the parking position does generally not influence the runway allocation), is more penalising (each aircraft is delayed 1 minute more). The 1-to-n strategy is more penalising for a number of aircraft that is not bigger, which can be explained by the weakness of the strategy.

4.3 Genetic algorithm efficiency

In order to observe the GA efficiency, figure 6 gives the number of generations required by the GA as a function of time. the different peaks appearing at 7, 8, 10, 11 am and 5 pm are the traffic peaks. Figure 7 shows the correlation between the number of generation required by the GA and the number of moving aircraft on the ground.

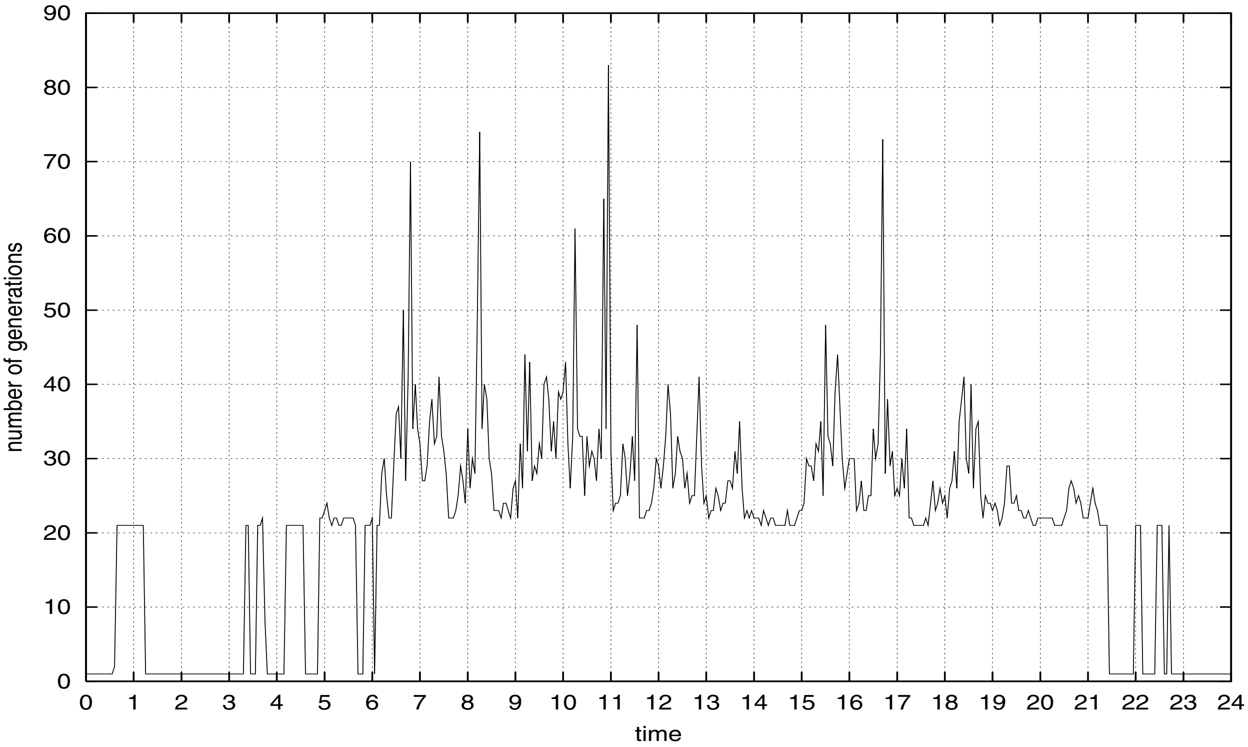


Figure 6: Number of generations (random strategy) as a function of time

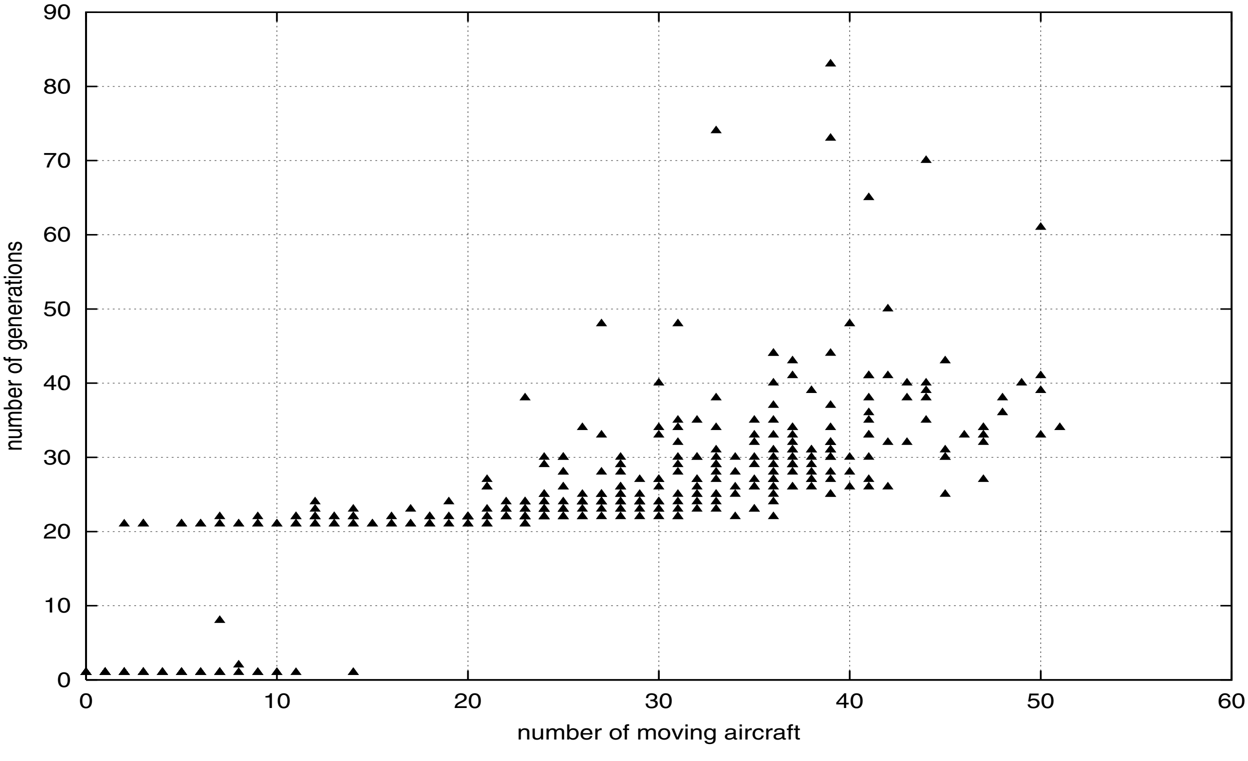


Figure 7: Number of generations (random strategy) as a function of the number of moving aircraft

5 Conclusion and further work

This preliminary work has shown that it was possible to build a taxiway adviser in order optimise the aircraft ground traffic on big airports such as Roissy Charles de Gaulle. If many hypotheses have been simplified in order to focus on the algorithm, it can be noticed that the modelling can be improved in order to take into account different speeds, uncertainties on speeds etc... without changing the algorithm itself. Further work will focus on these improvements. Genetic Algorithms are very efficient on the problem as they search the global optimum of the problem whereas a deterministic algorithm such as an A^* algorithm can only reasonably be used with a 1-to-n strategy, which is very poor.

References

[AMO93] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows, Theory, Algorithms and Applications.* Prentice Hall, 1993.

[DA98] N. Durand and J. M. Alliot. Genetic crossover operator for partially separable functions. In *Proceedings of the third annual Genetic Programming Conference*, 1998.

[DAN96] Nicolas Durand, Jean-Marc Alliot, and Joseph Noailles. Automatic aircraft conflict resolution using genetic algorithms. In *Proceedings of the Symposium on Applied Computing, Philadelphia.* ACM, 1996.

[Gol89] D.E Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Reading MA Addison Wesley, 1989.

[Gro99] The Preston Group. *TAAM Reference Manual.* The Preston Group, 1999.

[HT95] Reiner Horst and Hoang Tuy. *Global Optimization, Deterministic Approaches.* Springler, 1995.

[IDA$^+$98] A.H Idris, B Delcaire, I Anagnostakis, W.D Hall, J.P Clarke, R.J Hansman, E Feron, and A.R Odoni. Observations of Departure Processes at Logan Airport to Support the Development of Departure Planning Tools. In *2nd USA/Europe Air Traffic Management R & D Seminar*, Orlando, December 1998.

[Mic92] Z Michalewicz. *Genetic algorithms + Data Structures = Evolution Programs.* Springer-verlag, 1992.

[Pea84] Judea Pearl. *Heuristics.* Addison-Wesley, 1984. ISBN: 0-201-05594-5.

[YG93] Xiaodong Yin and Noel Germay. A fast genetic algorithm with sharing scheme using cluster analysis methods in multimodal function optimization. In C.R. Reeves R.F.Albrecht and N.C. Steele, editors, *In proceedings of the Artificial Neural Nets and Genetic Algorithm International Conference, Insbruck Austria.* Springer-Verlag, 1993.

Using Cultural Algorithms to Improve Knowledge Base Maintainability

Nestor Rychtyckyj
Ford Motor Company
New Business and Strategic Planning
Dearborn, MI 48121

Robert G. Reynolds
Wayne State University
Department of Computer Science
Detroit, MI 48202

Abstract

In this paper we discuss the use of a specific form of evolutionary computation known as Cultural Algorithms to improve the maintainability of knowledge bases in dynamic problem environments. One such dynamic problem environment involves process planning for vehicle assembly. Since 1990 Ford Vehicle Operations has used the Direct Labor Management System (DLMS) as an automated solution to managing the automobile manufacturing process system at Ford's vehicle assembly plants. Maintainability becomes very difficult over time due to changes in all of the following areas: the external business environment, the processes and physical concepts being modeled, and the underlying hardware and software architecture. We will discuss how Cultural Algorithms are applied using a bottom-up approach to re-engineer the DLMS semantic network knowledge base and improve the maintainability of the system. Our results show that Cultural Algorithms can be used to discover emergent knowledge by combining building blocks in our bottom-up semantic network re-engineering application. We demonstrate how the semantic networks generated by Cultural Algorithms compare favorably to both a decision tree approach and to the results obtained manually by the developers of the system in terms of reduced complexity.

1 INTRODUCTION

The use of KL-ONE and associated knowledge representation systems for building large complex knowledge bases to support real-world problems has been demonstrated in various application areas (Brachman et al 1991). One such system is Ford's Direct Labor Management System (DLMS) that has been used since 1990 in the very dynamic domain of process planning for vehicle assembly (Rychtyckyj 1999). The long-term maintenance of the DLMS knowledge base has demonstrated both the flexibility and reliability of semantic network-based knowledge bases in a rapidly changing industrial setting. The most critical issue in utilizing knowledge-based systems over a long period of time is the maintainability of the system.

Previous work (Rychtyckyj and Reynolds, 1998, 1999, 2000) has concentrated on utilizing a form of Evolutionary Computation known as Cultural Algorithms to re-engineer existing semantic-network based knowledge bases using a top-down approach in order to reduce network complexity and improve performance. The complexity that is inherent in real world problems has led to the use of semantic networks to represent the many relationships that exist in these problem domains. Originally developed to explain the organization of semantic information in human cognitive systems, semantic networks have been expanded to represent various complex environments. The graphical nature of semantic network models provides a solid framework for engineering and maintaining knowledge based systems that model these complex environments (Brachman and Schmolze 1985). The use of semantic network based knowledge representation systems, also known as Description Logics, has been successfully applied to a variety of dynamic industrial problem domains (McGuiness and Wright 1998).

The most critical issue in utilizing knowledge-based systems over a long period of time is the maintainability of the system. This maintainability can become very difficult due to changes in the external business environment and the processes and physical performance that the system is modeling. The solution to this maintenance problem lies in the process of re-engineering the application to keep it current with the changing problem environment. The requirement for the constant adaptation of the system to a dynamic environment motivated us to explore the use of evolutionary computational techniques as a tool for re-engineering semantic networks.

One specific model of evolutionary computation, known as Cultural Algorithms, has been successfully used to re-engineer both a commercial rule based expert system (Sternberg and Reynolds 1997) as well as a knowledge discovery system utilizing decision trees (Al-Shehri 1997). In this work we focus our maintenance efforts on a knowledge based system, known as the Direct Labor Management System (DLMS) (Rychtyckyj 1999), that is used by Ford Vehicle Operations to manage the vehicle assembly process at Ford's plants in North America and Europe. The vehicle manufacturing process planning environment is extremely dynamic as competitive pressures, advances in technology, and the globalization and consolidation of the automobile industry create a very dynamic environment. DLMS utilizes a large-scale semantic network architecture with over 10,000 nodes to model the vehicle assembly process at Ford. This knowledge base must be constantly updated and maintained to keep it current with the changing business environment. These modifications often increase the complexity of the system, the cost of maintenance, and the time required to make the necessary changes. These factors have motivated us to utilize Cultural Algorithms as a tool to learn how to analyze and re-engineer the DLMS semantic network in order to reduce complexity and increase maintainability.

This paper discusses the results that we have achieved using the Cultural Algorithm approach in a bottom-up fashion to re-engineer the DLMS semantic network knowledge base. The results that we have obtained using Cultural Algorithms are compared against the results that were obtained by the human developers and a decision tree-based approach. We show that in most cases Cultural Algorithms are a much more efficient tool than manual inspection in terms of reducing the cost of subsumption. Our results also demonstrate that as the complexity of the network increases, the performance of Cultural Algorithms produces significant improvements over current manual techniques. These results demonstrate the usefulness of Cultural Algorithm for knowledge base re-engineering, and show how an evolutionary computational approach may be utilized as a tool for knowledge base maintenance.

Section 2 of this paper will provide a brief background on semantic networks and the DLMS system. A discussion and description of Cultural Algorithms will be given in Section 3. The results of using Cultural Algorithms to cluster concepts in a semantic network will be discussed in Section 4. Section 5 will compare and analyze the Cultural Algorithm results with the previous manual results. The paper concludes with a discussion of our results and a description of future work.

2 SEMANTIC NETWORKS IN DLMS

The knowledge representation scheme utilized in DLMS is based on the KL-ONE (Brachman and Schmolze 1985) family of semantic network based representation schemes. The use of semantic networks for modeling knowledge bases requires the development of a mapping scheme between the physical model and its representation in the network. A node contains information about an entity or class of entities that have certain features in common. These features can be represented as properties or attributes of that node. The relationship between the nodes is represented in the association links that connect these two nodes. These associations are based on the characteristics of the structure that we are modeling. A semantic network model of a natural language processing system requires links that denote the relationships between tokens in a sentence. Models of physical processes, such as a vehicle assembly process, utilize a representation that describes how automobile subsystems relate to each other. The structure of the semantic network also reflects the entities that it is modeling. Generally, a network is broken down into classes containing similar objects that are distinct from objects with dissimilar features.

A further requirement of some semantic network models is the presence of an algorithm that correctly puts new or modified nodes into their proper place. This process is known as classification, and it involves utilizing the attribute values of the target node to place this node into the most suitable position in the semantic network. The classification process requires the use of a technique called *subsumption* to determine if a target node can be added to a potential parent node. Subsumption is the inferential relationship that determines if one node in the network is a parent of another node. The subsumption relationship is determined by comparing attribute values between the two nodes. Each attribute value in the child node must be a member of that attribute class in the parent node. The algorithm for subsumption is as follows:

For any two nodes S and T, S subsumes T if the following conditions are true:

1. Node T has at least all of the attributes possessed by Node S.
2. For each attribute A that is common to both T and S, the value of T(A) must belong to the allowable values of S(A).

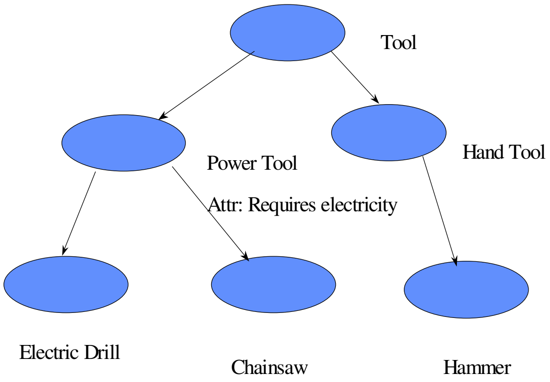


Figure 1: An example of a portion of the DLMS semantic network

The utilization of a semantic network model for knowledge representation is based on building an internal model of an application within the computer. The entities that are identified in the model will be represented as concepts or nodes within the network. The properties or attributes that define an entity are represented as slots within a particular node. These slots may be defined as accepting any value or they may be edited to accept a particular type or domain of acceptable values.

3 CULTURAL ALGORITHMS

A Cultural Algorithm is an evolutionary computational approach that utilizes culture as a vehicle for storing relevant information that is accessible to all members of the population over the course of many generations. In this context, culture can be viewed as an evolving source of data that influences the patterns of behavior that are practiced by various members of the population. As in human societies, culture changes over time, but it provides a baseline for interpreting and documenting an individual's behavior within a society. Cultural Algorithms were developed to model the evolution of the cultural component over time as it learns and acquires knowledge. Cultural Algorithms can be viewed as an extension of Genetic Algorithms, where the belief space acts as a conduit of knowledge between each generation that is being evolved. Based on this approach, Cultural Algorithms can be used to drive the self-adaptation process within evolutionary systems in a variety of different application areas (Reynolds 1999).

Initially, a population of individuals that represent the solution space are randomly generated to create the first generation. The population model used by Cultural Algorithms is based on a Genetic Algorithm approach with the addition of the belief space to guide the learning process. The initial belief space is empty. For each generation, the Cultural Algorithm will evolve a population of individuals utilizing the Vote-Inherit-Promote (VIP) framework. During the Vote phase of this process, the population members are evaluated for their contribution to the belief space using the acceptance function. Those beliefs that contribute the most to the problem solution are selected or voted to contribute to the current belief space. The belief space is modified when the inherited beliefs are combined with the beliefs that have been added from the current generation using the belief space update reasoning process. Next, the updated belief space is used to influence the evolution of the population by favoring those individuals for reproduction whose traits most closely reflect the contents of the belief space. A set of evolutionary operators, usually including crossover and mutation, is then used to produce the new population. This new population will be evaluated and this cycle continues again. The VIP cycle ends when a termination condition is met. The termination condition is usually achieved when little or no change is detected in the population through several generations, or when certain knowledge structures have emerged in the belief space.

4 CULTURAL ALGORITHMS FOR SEMANTIC NETWORKS

In this section we describe a bottom-up approach that uses Cultural Algorithms to re-engineer a semantic network by building a new network structure that that is more efficient in terms of information extraction time. This new semantic network contains all of the concepts and attributes described previously, but the structure will be created using learning heuristics and the Cultural Algorithm learning process. The goal of Cultural Algorithms in this application is to classify the input concepts into graphical clusters that are most efficient for subsumption and classification. A set of concepts with all of their properties is used as the input and a semantic network is generated as an output. This approach allows us to create a new semantic network without relying on any of the previous design information. We will also discuss how Cultural Algorithms are used to discover emergent knowledge by combining building blocks in our bottom-up semantic network re-engineering application.

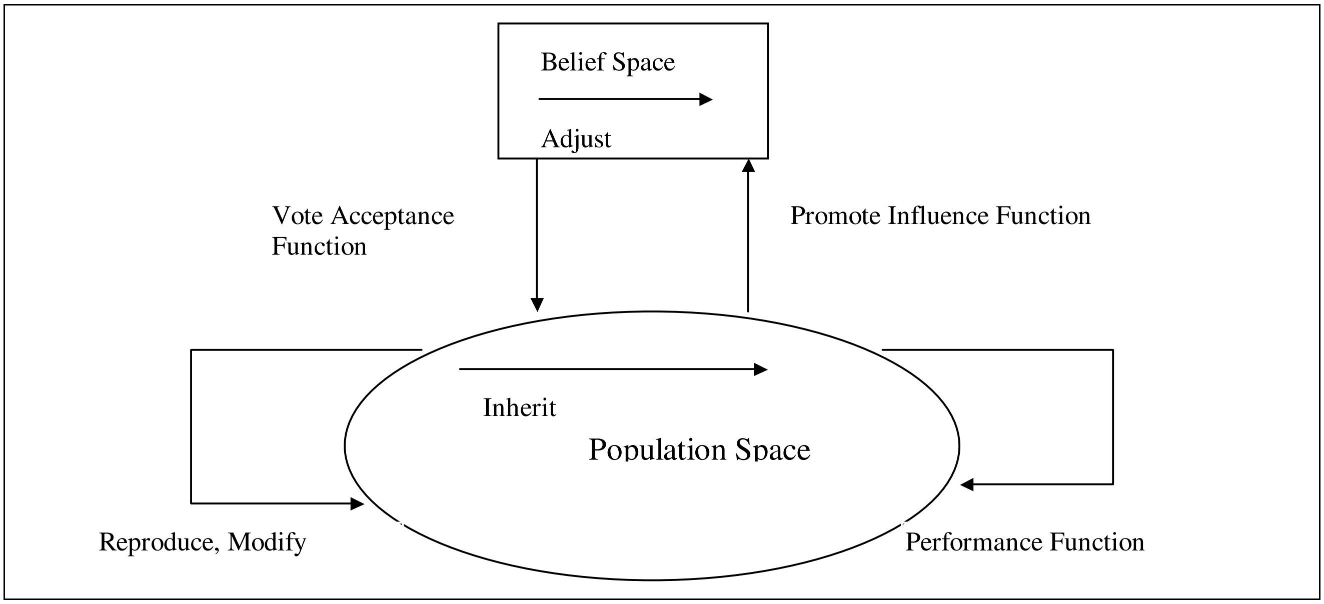


Figure 2: Cultural Algorithm Components

The following algorithm describes our application of Cultural Algorithms for bottom-up semantic network re-engineering.

1. Generate at random an initial population *p* of individuals each of whom represents a clustering scheme for a given level of the network.
2. Analyze the input data set and develop a semantic network framework based on the attribute usage and values found in the input data set. This analysis will derive the value for *n* (the number of concepts in the input data) and *C* (the clusters needed to classify this data).
3. Create an initial belief space *B* containing a vector representing the clustering of concepts in the semantic network. The vector *B(1….n)* (where *n* is the number of concepts in the input data set) contains the "don't care" value for all entries in *B*.
4. For each individual *I* in population *p*, build a semantic network representation using the clustering distribution represented in *p(I).* Each individual *I* in the population *p* is a vector of length *n* that consists of a single entry for every concept in the input data set. Each concept has a value of *Cn* that represents the cluster *C* that this concept is being assigned to.
5. Evaluate the semantic network created for each individual in *p* using the performance function *PF*. *PF* for any individual *P(i)* in the population is computed as:
    - Select each entry *I(j)* in the individual *I* from *I(i)* to *I(n)* where *n* is the length of *I*.
    - This entry *I(j)* represents the cluster *Cj*, to which the node represented in *I(j)* is assigned to.
    - Compare the attributes of *I(j)* with the attributes present in the assigned cluster *C*.
    - When there is a match between the attributes values for *I(j)* and cluster *C* then we increment the fitness value for *I(j)*; otherwise we do not do anything to the fitness value.
6. Update the belief space *B* by accepting input from the top performing 20% of the individuals using the following algorithm:
    - Select the voters from the general population by using the fitness function to rank each individual.
    - For each entry *B(i)* in the Belief Space *B* check to see what value each voter has for the corresponding attribute definition.
    - If more than 50% of the voters agree for any entry *B(i)* then update *B(i)* with the value that the majority of voters propose.
7. Generate new *p* offspring solutions by applying variation operators that are modified by the influence function. This creates *2p* solutions in the population.
8. Conduct a tournament between the individuals based on the fitness score.

9. Select the *p* individuals that have the most wins in the tournament to be the parents for the next generation.
10. Check for the termination condition
11. The process will return back to Step 4 unless an acceptable solution has been found.

4.1 BOTTOM-UP CULTURAL ALGORITHM CLUSTERING

The use of Cultural Algorithms for building a semantic network is based on defining the problem as finding the best fit for a concept in a search space of clusters or classes. The clusters are differentiated from each other by their attributes and the values that those attributes contain. The goal of classifying these concepts into their appropriate clusters is analogous to building a semantic network that minimizes the number of clusters but preserves the accuracy and correctness required for information retrieval. This technique can be also be used as a data mining tool by creating a semantic network that has its concepts classified into the appropriate cluster. These clusters can then be used as a basis for investigating and interpreting the knowledge that is contained in any particular class. The process of building a semantic network also preserves the subsumption relationship between the clusters or classes, as we traverse the network from the leaf nodes to the root.

The initial step in building a semantic network is to analyze the input data set to determine the depth, or number of levels that should be present in the proposed network. This process is accomplished by separating the concepts in the data set by the number of attributes that they each contain. The concepts with the same number of attributes will be classified at the same level of the network. The next step of the analysis process is to determine how many unique attribute values are contained for each level of the proposed semantic network. We will temporarily create a new class for each attribute/value pair that is contained in the input data set. Later, some of these classes will be pruned if the learning process discovers that a class is contained within a larger class.

Once the basic structure of the semantic network is in place, we can utilize the Cultural Algorithms to start classifying the input data concepts into their appropriate class or cluster. Our goal is to use Cultural Algorithms to evolve a solution where the concepts at each level of the network are properly classified into their appropriate class or cluster. Here, the population of the Cultural Algorithm contains individuals that represent a possible clustering solution for a set of given concepts. The tree that is built using these clusters is then evaluated for complexity and accuracy by the performance function. The belief space contains a list of possible clusters that is used to guide the search process.

The mutation genetic operator is used to create new individuals in the population. The mutation operator randomly flips one of the bits in the individual based on a given random variable factor. The mutation operation is constrained by the fact that the resulting bit must also be a valid cluster that is utilized for this population of concepts. The selection process utilized the evaluation function to select those members of the population that are most likely to produce an individual with high fitness. This approach enables us to evolve a population containing individuals that represent networks that are both accurate and have low complexity in terms of subsumption cost. Those networks that have been selected contain those clusters that best represent the concepts and properties in the input data set while minimizing the cost of subsumption.

Each individual in the population represents a potential partitioning of the input data concepts into clusters. Each member in the population contains the number of the cluster that this concept belongs to. The belief space structure is a vector as shown in Figure 3. The belief space represents our current knowledge of the attribute data that is most useful in guiding the system to a solution. The value for each entry in the belief space is either the number of the cluster or a "negative one", which represents "don't care". The initial belief space contains all "negative ones" as we don't know anything about the solution at this point.

The main loop of the Cultural Algorithm is executed continuously for a specific number of generations or until a termination condition is achieved. In our case the belief space contains knowledge that has emerged from our search process and is utilized within the termination condition. Our system utilizes the belief space to in order to guide the evolutionary search throughout the learning process for all levels of the network. After the learning is completed at one level, the belief space utilizes the knowledge that it has learned at that level as building blocks to guide the process at the next level. This knowledge contains information about the clusters that are being used at this level of the network. The Cultural Algorithms use heuristic knowledge about the relationship between clusters at different levels of the network to prune those clusters that are already subsumed by a higher-level cluster. In effect, clusters at lower levels of the network can be combined into larger subsuming classes in the belief space as we build up the network from the leaves to the root. By combining these building blocks of knowledge at each level of the network, Cultural Algorithms are able to discover emergent knowledge about the optimal structure of the semantic network. This allows the system to incorporate the knowledge it has learned about the relationships between classes at different levels to build a more efficient semantic network. If the belief space is not significantly modified for a period of three generations, we conclude that the evolution process has stopped and we terminate the program.

The evaluation function describes how the fitness of a particular individual is judged, and it greatly impacts the potential for this individual to contribute to the solution for the given problem. For each member of the population we generate a fitness measure based on how accurately each concept in the population has been classified into a cluster in the semantic network. This evaluation is based on the similarity between each concept's attributes and values with those of the cluster that this concept has been assigned to.

The top 20% of the individuals in the population based on performance are then selected to vote for the beliefs that will be accepted in to the belief space. If the majority of the members in the voting population agree on a particular cluster for a given concept then the belief space will be modified according to the algorithm described in Section 3. Any concepts that do not receive a majority of votes for a particular cluster remain at "don't care" status

The belief space that has been modified by the population is then utilized to assist in the evolution of the new population. This new generation is created by the influence function with input from the belief space using the mutation genetic operator. The members of the new population are created by comparing the values in each member of the existing population with the corresponding member in the belief space.

The next generation of the population is then created following a single elimination tournament between the existing population and the population that was created in the previous step. The fitness is calculated for each individual in the existing population and in the new population. Subsequently each member in the old population is compared to the offspring in the new population based on their fitness value. The individual with the highest fitness value is selected for inclusion into the next generation. The learning process terminates when the belief space stops evolving. At the conclusion of the Cultural Algorithm process, we have created a semantic network that contains all of the knowledge from the input data set that has been optimized for the efficiency of information retrieval

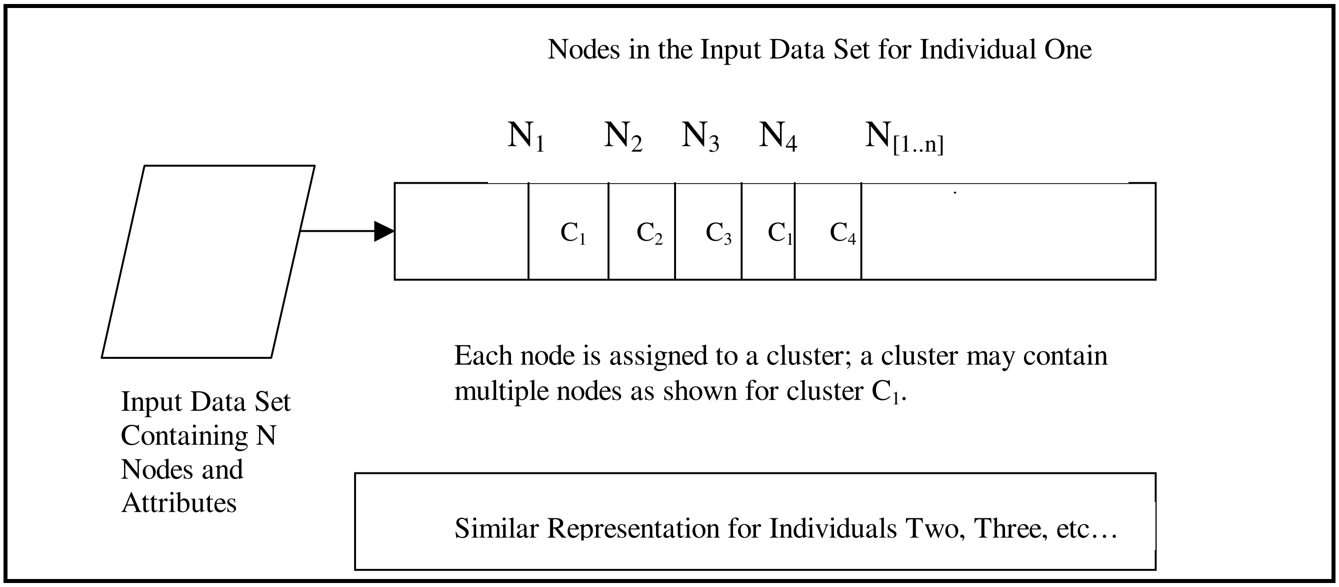

Nodes in the Input Data Set for Individual One
N_1
N_2
N_3
N_4
$N_{[1..n]}$
C_1
C_2
C_3
C_1
C_4
Each node is assigned to a cluster; a cluster may contain multiple nodes as shown for cluster C_1.
Input Data Set Containing N Nodes and Attributes
Similar Representation for Individuals Two, Three, etc…

Figure 3: Representation of Individuals in Population Space

4.2 USING CULTURAL ALGORITHMS FOR BOTTOM-UP RE-ENGINEERING

This section describes the results of the application of Cultural Algorithms to the clustering of nodes in the DLMS semantic network. The input to this system is a data file containing a set of concepts describing an automotive assembly process planning knowledge base that also include a list of attributes and values describing these attributes. Each concept represents a node whose attributes describe the properties that make the concept unique. Our system reads in these concepts and utilizes a Cultural Algorithm approach to create a network representation of the data. This network representation is then compared against both a decision-tree based representation produced using a decision tree algorithm on the same input set and the manual representation that was constructed by human developers. This result is displayed in Table 1.

5 DISCUSSION OF RESULTS

The use of Cultural Algorithms for the clustering of concepts provides a decided advantage over both the decision tree and manual approaches in terms of reducing the cost of subsumption. As shown in Table 1, this advantage is apparent when we view the network as a single entity. The advantage of Cultural Algorithms over human developers is about 15% in terms of reducing the number of clusters needed to classify the input data. However, a closer comparison of the Cultural Algorithm results with those of the human developers shows that there is some variation in this ratio at the individual levels of the network. In terms of minimizing the number of clusters needed, the human developers were more efficient than Cultural Algorithms at certain levels of the network, even though their entire network was more complex than the Cultural Algorithms approach. All of the following observations discuss efficiency in terms of minimizing the number of clusters needed to classify the input data set.

Table 1 – Results of Using Cultural Algorithms for DLMS Re-Engineering

Level	Population Size	Number of Generations	Number of Concepts to be Classified	Clusters Needed with Decision Trees	Clusters Required Using CA's	Pct Used (CA vs. Decision Trees)	Manual Clusters Used	Pct (CA Clusters vs. Manual)	Cluster Correlation Between Manual and CA's
1	50	12	311	2	2	1	64	0.03125	1
2	50	20	4416	87	87	1	297	0.292929	0.91938406
3	50	21	956	83	83	1	181	0.458564	0.43410042
4	50	28	1857	761	692	0.90932983	666	1.039039	0.73882604
5	50	32	801	345	313	0.90724638	244	1.282787	0.26716605
6	50	39	198	177	123	0.69491525	144	0.854167	0.35858586
7	50	15	127	220	89	0.40454545	119	0.747899	0.44094488
8	50	21	180	249	128	0.51405622	115	1.113043	0.7777778
9	50	27	322	323	216	0.66873065	190	1.136842	0.7329193
10	50	35	164	147	87	0.59183673	131	0.664122	0.9207317
11	50	22	17	31	10	0.32258065	18	0.555556	1
12	50	8	2	15	2	0.13333333	2	1	1
Total		23.333333	9351	2440	1832	0.75081967	2171	0.843851	0.71586968

A close observation of the data displayed in Table 1 shows that the Cultural Algorithms were more efficient than the human developers at 8 levels of the network, while the human developers had an advantage in 4 of those levels. In three of those four cases, the Cultural Algorithm approach is slightly more complex than the manual approach. In these cases, there is agreement between the clusters produced by Cultural Algorithms and the manual approach. The one case where the Cultural Algorithm approach is much worse (28%) than the manual one is at Level 5. This particular result showed a low correlation between the clusters as well as a significantly higher number of clusters required for the Cultural Algorithm solution. A closer examination of this data showed that some of the attributes used at this level of processing were no longer actively used in the DLMS system, but had never been removed from the network. This may happen because the developers felt that this knowledge may again be required in the future and should not be completely removed from the system. The human developers have obviously recognized this and made the required adjustments in the network to avoid using this inactive data. The Cultural Algorithms had no knowledge of these changes and performed the clustering based on the irrelevant input, which resulted in a poor solution to the problem compared to the manual approach that ignored the input. However, if these concepts were being actively used the Cultural Algorithm solution would be an acceptable one.

The results from the other 8 levels of the network show that the Cultural Algorithms generate a very good solution to the clustering problem relative to the human developers. The number of clusters required by the Cultural Algorithms was significantly lower than that of the human developers needed and the correlation between the two solutions was high in 5 of the cases. This demonstrated that the Cultural Algorithms approach finds excellent solutions that follow the general approach of the human developers, but result in a higher performance. The other 3 cases exhibit an even more interesting result. In these cases the Cultural Algorithms produce a superior result to the human developers by finding a novel network configuration. The low correlation between the human and Cultural Algorithm clusters demonstrates that it is possible for the Cultural Algorithms to develop new ways to organize the concepts in a knowledge base.

Further examination of the results show that the Cultural Algorithms approach exhibit improvement over the human developers at both the lowest nodes of the network and at the highest levels of the network. The correlation between the two approaches is also very high at these two extremes and demonstrates that the Cultural Algorithms approach can solve the clustering problem for both simple concepts with few attributes and highly complex concepts with many attributes that exist at the root of the semantic network. The Cultural Algorithm performance, in terms of minimizing the number of clusters needed to classify the input data is comparable with the human developers at all levels of the network and can actually define new methods of clustering the concepts that were not apparent to the human developers

6 CONCLUSIONS

In this paper we described our approach to the bottom-up re-engineering problem for semantic networks using Cultural Algorithms. This paper presents a method of bottom-up re-engineering where Cultural Algorithms are used to build a semantic network from an input data set that contains a list of concepts and their attributes. The bottom-up re-engineering system uses the population space to represent sets of clusters for each concept at the given level of the network.

Our bottom-up re-engineering approach was tested on the DLMS semantic network. The results obtained here showed that Cultural Algorithms often reduced the number of clusters that are needed to classify a set of concepts in the semantic network. The results obtained by using Cultural Algorithms for bottom-up re-engineering provide a decided advantage over both the manual results and a decision tree based approach in terms of reducing the cost of subsumption and network complexity. These results also show that Cultural Algorithms can be used to discover novel configurations of semantic networks that were not apparent to the human developers of the system.

References

Al-Shehri, Hasan, (1997), "Evolution-Based Decision Tree Optimization Using Cultural Algorithms," Ph.D. Dissertation, Wayne State University.

Brachman, R., Schmolze, J., (1985), "An Overview of the KL-ONE Knowledge Representation System," *Cognitive Science* 9(2), pp. 171-216.

Brachman, R., McGuiness, D., Patel-Schneider, P., Resnick, L., Borgida, A., (1991) "Living With Classic: When and How to Use a KL-ONE-Like Language" in *Principles of Semantic Networks*, ed. J. Sowa, pp. 401-456, Morgan Kaufmann Publishers.

Holland, J., (1975), *Adaptation in Natural and Artificial Systems*, (1992 edition), MIT Press.

McGuiness, D., Wright, J., (1998), “An Industrial-Strength Description Logic-Based Configurator Platform”, *IEEE Intelligent Systems & Their Applications*, Volume 13, Number 4, pp. 69-77.

Reynolds, R.G., (1999), “Cultural Algorithms: Theory and Applications” in *New Ideas in Optimization*, pg. 367-377, McGraw Hill.

Rychtyckyj, N., (1999), “DLMS: Ten Years of AI for Vehicle Assembly Process Planning”, *AAAI-99/IAAI-99 Proceedings*, Orlando, FL, July 18-22, 1999, pp. 821-828, AAAI Press.

Rychtyckyj, N., Reynolds, R.G., (1998), “Learning to Re-Engineer Semantic Networks Using Cultural Algorithms” in *Evolutionary Programming VII*, Springer-Verlag, pg. 181-190.

Rychtyckyj, N., Reynolds, R.G., (1999), “Using Cultural Algorithms to Improve Performance in Semantic Networks”, in *Proceedings of the 1999 Congress of Evolutionary Computation*, Washington D.C, July 6-9, vol. 3, pp. 1651-1656, IEEE Press.

Rychtyckyj, N., and Reynolds, R., (2000), “Assessing the Performance of Cultural Algorithms for Semantic Network Re-Engineering”, *Proceedings of the 2000 Congress on Evolutionary Computation*, July 16-19, 2000, La Jolla, CA, pp. 1482-1491, IEEE Press.

Sternberg, M., Reynolds, R. (1997), “Using Cultural Algorithms to Support Re-Engineering of Rule-Based Expert Systems in Dynamic Performance Environments: A Case Study in Fraud Detection” in *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 4, pp. 225-243.

Selecting Dimensions and Delay Values for a Time-Delay Embedding Using a Genetic Algorithm

James B. Vitrano
Department of Electrical and
Computer Engineering
Marquette University
Milwaukee, WI 53201 USA
jimv@jimv.net

Richard J. Povinelli
Department of Electrical and
Computer Engineering
Marquette University
Milwaukee, WI 53201 USA
richard.povinelli@marquette.edu

Abstract

This paper describes a novel technique for determining a useful dimension for a time-delay embedding of an arbitrary time series, along with the individual time delays for each dimension. A binary-string genetic algorithm is designed to search for a variable number of time delays that minimize the standard deviation of the distance between each embedded data point and the centroid of the set of all data points, relative to the mean distance between each data point and the centroid. The geometric transformations of rotation and scaling are added to the algorithm to allow it to identify attractors that are not aligned with the data axes. Several artificial and real-world attractors and time series are analyzed to describe the types of attractors favorable to the use of this technique.

1 INTRODUCTION

Time-delay embedding, or establishing a phase space representation of a system using current and delayed values from a sampled time series, is a useful technique for characterizing nonlinear behavior of a system (Abarbanel, 1995; Povinelli, 1999). Takens (1981) showed that an embedding of dimension greater than twice the dimension of a smooth manifold containing an attractor is a true embedding; i.e., the phase space is topologically equivalent to the state space of the attractor. Sauer and Yorke (1993) extended Takens' continuous-time work into discrete time and found that in many circumstances a lower embedding dimension is sufficient to represent the dynamics of the system.

When performing a time-delay embedding of a sampled time series, the two key questions to be answered are (1) how many embedding dimensions are required, and (2) what are the proper time delays, or lags, to use for each dimension? As described above, Takens, Sauer, and Yorke have theoretical answers to the first question. However, when facing a system with an attractor of unknown dimensionality, their theorems provide only general guidance. To test the adequacy of a particular embedding dimension, the false nearest neighbors technique (Kennel, Brown, and Abarbanel, 1992) examines the relative location of neighboring data points in the next higher dimension to determine whether the neighboring points remain neighbors in the higher dimension. Even with these techniques, selecting the proper embedding dimension for a particular time series seems to be as much art as it is science (Abarbanel, 1995).

Some more specific techniques are available to help answer the second question, finding the individual time delays for each dimension. Zeros or minima of the autocorrelation function of the time series have been mentioned as useful choices for time delays (Kantz and Schreiber, 1997), along with the first minimum of the time-delayed mutual information function (Fraser and Swinney, 1986). However, if these delays do not produce a useful embedding, little additional guidance is available.

This paper proposes the use of a binary-string genetic algorithm (GA) to search for the dimensionality and individual delay values for an embedding that best fits a given criterion – in this case, the minimum standard deviation of estimates of the radius of the attractor, compared to the mean of those radius estimates. While the GA amounts to a solution by trial and error, it represents an improvement in that it is an automated and directed trial-and-error solution.

2 CHARACTERISTICS OF THE GENETIC ALGORITHM

A genetic algorithm (GA) (Dumitrescu *et al.*, 2000), designed to emulate the natural principles of evolution, is an iterative technique for searching a large set of possible solutions to a problem for an optimal solution. In most GAs, a population of random solutions is generated, and the "fitness" of each solution in the population is calculated. Based on the fitness of each solution, a new generation of solutions is created such that the "fittest" solutions survive and combine into new possible solutions. Typically, some level of mutation is introduced into the new population to help prevent the GA from converging to a solution that is only locally optimal. This process is then repeated until a stopping criterion is met (e.g., a fixed number of generations, exceeding a fitness threshold, or domination of the population by one particular solution).

In a binary-string GA, each solution is represented by a series of binary digits, known as a "chromosome". After decoding each chromosome, evaluating the fitness of each solution, and selecting two "parents" to be combined, the combination is often performed using a "crossover" technique, where a portion of one parent's chromosome is combined with a portion of the other parent's chromosome. Mutations are usually performed by inverting one or more bits within the chromosome.

The binary-string GA used in this paper was selected because a software implementation of the GA was already available to the authors. The GA uses a fixed, predetermined population size and number of generations.

The most dramatic difference from the "standard" GA described above is in the method of selecting and combining parents. A preset number of the least-fit members of the population are not allowed to be selected as parents. A preset number of the most-fit members of the population are copied directly into the next generation, and in addition can be selected as parents. Members in the remaining "middle-fit" portion of the population are also able to be selected as parents. For each slot in the new population not occupied by the copies of the most-fit members, two parents are selected at random (with equal probability) from the set of eligible parents, and a byte-wise crossover is performed where each byte of the child's chromosome has a 50% probability of being copied from either parent. Single-bit mutations are also placed in the child's chromosome randomly at a preset rate.

The specific GA implementation was not studied in much detail. This may be an area for future research and improvement. In particular, a more efficient breeding strategy may result in more rapid convergence to an optimal solution.

3 THE TIME-DELAY EMBEDDING CHROMOSOME

The chromosome used with the GA is designed to be simple to decode into its corresponding time-delay embedding. The chromosome contains a fixed number of possible embedding dimensions. These dimensions are combined with a fixed first dimension, which (when not rotated as described later) corresponds to $x(t)$, the current sample from the time series. The maximum number of possible dimensions is preset by the user. This represents one of the methods of limiting the dimensionality of the set of possible solutions.

The format of the chromosome is shown in Figure 1 below:

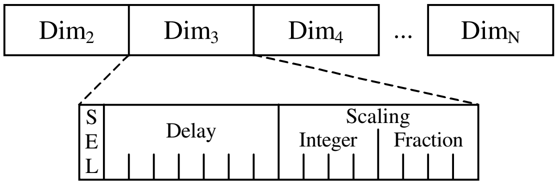


Figure 1: The format of the time-delay embedding chromosome (without rotation).

Each embedding dimension contains a single "selector" bit, which controls whether the dimension is considered when the chromosome is decoded into a time-delay embedding. Another seven bits are allocated for the time delay value corresponding to that dimension, allowing each dimension to contain a delay of between 0 and 127 samples. Another eight bits contain a scaling factor along that dimension's axis, ranging from 1/16 to 15 15/16 in increments of 1/16. This allows, for example, a properly aligned, oval-shaped, two-dimensional attractor to be expanded or compressed along the two embedding axes to nearly form a circle, which the fitness function described below recognizes as optimal.

The chromosome generation and decoding routines used with the GA may also be configured to allow the time-delay embedding to be rotated in space. In the scaling example above, the oval-shaped two-dimensional attractor needed to be "properly aligned", i.e., its major and minor axes needed to be roughly parallel to the coordinate axes. Allowing the GA to search through possible rotations allows the GA to rotate a misaligned oval so that it is properly aligned, then scale it to be roughly circular, thus producing a nearly optimal fitness value.

A rotation operation affects only two coordinates of a point, regardless of the number of dimensions (Burbanks, 1996). If rotation is enabled, an additional 8-bit field is appended to the chromosome for each possible pair of dimensions, resulting in $n_d(n_d-1)/2$ possible rotations, where n_d is the maximum number of dimensions allowed in the embedding. Only those rotations where both dimensions in the pair are enabled by their respective selector bits are performed. The 8-bit field allows for 256 possible rotations in the dimension pair, resulting in resolution of approximately 1.4 degrees. The rotation is performed by multiplying a transformation matrix by the coordinate vector of the data point (Hoggar, 1992). For example, Equation 1 shows a rotation in dimensions 1 and 3 of a 5-dimensional point:

$$\begin{bmatrix} x_1' \\ x_2' \\ x_3' \\ x_4' \\ x_5' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ \sin(\theta) & 0 & \cos(\theta) & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \qquad (1)$$

If rotation is enabled, it is performed before the scaling operation. This was done with the misaligned oval-shaped attractor in mind: performing the scaling along the coordinate axes before the rotation would have made the transformation from oval to circle impossible. There may be other cases where performing the scaling first would provide a benefit. Providing for both a pre-rotation and a post-rotation scaling may be another possible improvement to this technique.

None of the field sizes chosen for this chromosome appear to be "magical", i.e., they can most likely be varied to suit an individual application without harming the ability of the GA to find a useful embedding. If a user has reason to believe that, for example, a scaling factor larger than 16 may be needed in some dimension, the chromosome can certainly be modified to allow this. Adding multiple selector bits in each dimension, which are XOR'ed together to determine whether a given dimension is included, may also provide interesting results by taking increased advantage of the mutation feature of the GA.

4 THE FITNESS FUNCTION

The fitness function is a key component of the GA: it controls which members of the population are represented in the next generation. Because the "most fit" members are selected most often for reproduction, the GA tends to find the maximum of the fitness function over many generations (Dumitrescu *et al.*, 2000). If a minimization is needed instead, a simple approach is to make the fitness function the negative of the original function.

The fitness function used in this technique assumes that all data points lie near an attractor in phase space, and that the attractor can be rotated and scaled to produce a roughly constant radius in all dimensions. The GA locates the centroid of the data points in phase space, calculates the Euclidean distance between each data point and the centroid, and then uses statistical properties of the distance values d to provide a fitness judgment:

$$f(d, n_d) = -\frac{\sigma_d}{\mu_d} \cdot b^{n_d} \qquad (2)$$

In Equation 2 above, σ_d represents the standard deviation of the distances, and μ_d represents the mean of the distances. The standard deviation is scaled by the reciprocal of the mean so that the GA does not favor smaller attractors over larger ones. The n_d parameter represents the number of dimensions, and b is a constant bias (≥ 1) toward a smaller number of dimensions. The bias causes a lower-dimensionality embedding to be rated as more fit than a higher-dimensionality embedding that is otherwise slightly more fit. This behavior may be desirable, for example, when seeking to view an embedding in two or three dimensions, or when working with the resulting embedding with limited computing resources. The bias causes the GA to add dimensions only when the added dimensions result in a fitness improvement. Values of b of 1.05 and 1.2 were used for the examples in this paper, and appeared to yield good general-purpose results.

The fitness function is negative to cause the GA to seek embeddings that minimize the relative standard deviation in the distance measurements.

Based on the description above, it is clear that the attractor geometry for which this technique is ideally suited is a hypersphere. With a sufficient number of noise-free samples, the centroid will be calculated at the center of the hypersphere, and thus all samples will have an equal distance from the centroid, yielding an optimal fitness value of 0. However, the technique is not necessarily limited to attractors that are hyperspheres. Many other geometric shapes and real-world attractors have roughly uniform radii, as shown in Table 1 below. The noise-free fitness values in Table 1 were calculated by randomly placing 1,000 points on or near the surface of each attractor, and removing the dimensionality bias shown in Equation 2. The noisy fitness values in Table 1 were calculated similarly, except that random Gaussian noise with RMS magnitude $0.1 \bullet \mu_d$ was added to each point:

Table 1: Partial fitness values ($-\sigma_d/\mu_d$ ratios) of several geometric and non-time-delay real-world attractors.

Dimension	**Attractor Description**	**Noise-Free Fitness**	**Noisy Fitness**
2	Circle	-0.0130	-0.0885
2	Hexagon	-0.0510	-0.0869
2	Square	-0.1093	-0.1353
2	Van der Pol oscillator limit cycle (Vidyasagar, 1993)	-0.1723	-0.1849
3	Sphere	-0.0094	-0.0638
3	Cube	-0.1229	-0.1386
3	Torus (o.d.=2•i.d.)	-0.2265	-0.2316
3	Lorenz attractor (Abarbanel, 1995)	-0.5453	-0.5445
3	Rössler attractor (Frazer and Swinney, 1986)	-0.4086	-0.4060

The Lorenz and Rössler attractors both exhibit a “folded” geometry, i.e., most of the samples fall near one of two planes that intersect at nearly right angles. Because this geometry is quite different than the spherical geometry that this technique was designed to seek, the fitness values for these two attractors are quite low. These examples point out one of the limitations of this technique: it searches for a time-delay embedding that best meets its goal of a uniform radius between the samples and the centroid, even if an attractor with a different geometry is responsible for the dynamical behavior of the system. However, if a reasonable guess about the geometry of the attractor can be made, a different fitness function that favors that particular geometry can be used.

5 RESULTS

To illustrate the technique, a simple test pattern was devised. A two-dimensional time-delay embedding of a sinusoidal signal should produce a circle if a proper delay (for example, ¼ of the oscillation period) is chosen. To test the technique, 1,000 samples of the time series shown in Equation 3 were presented to the GA to find an embedding with a maximum dimension of 7:

$$x(t) = \sin\left(\frac{2\pi}{\sqrt{2455}}t\right) + noise \qquad (3)$$
$$noise \in N(0, 0.1)$$

Table 2 below shows the two-dimensional embedding found by the GA.

Table 2: The time-delay embedding parameters found by the GA for a noisy sinusoidal time series.

	Dim. 1	**Dim. 2**
Delay value	0	15
Rotation vs. dim. 1 (radians)	N/A	2.90
Scaling factor	1	0.9375
Overall fitness value	-0.1665	
$-\sigma_d/\mu_d$ ratio	-0.1156	

Figure 2, a plot of the time-delay embedding, shows that the GA did indeed find a circular attractor. The dots in the plot represent the time series samples, and the cross represents the calculated centroid:

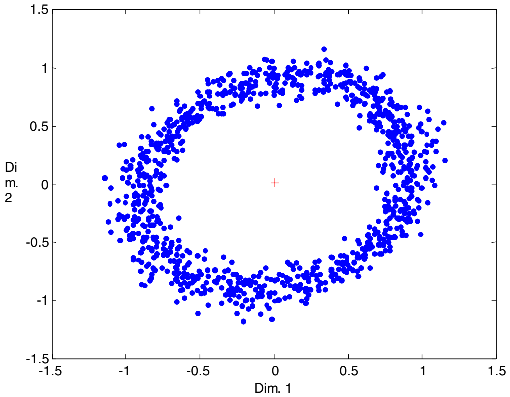


Figure 2: Plot of the time-delay embedding found by the GA for a noisy sinusoidal time series.

Interestingly, another run of the sinusoidal time series with noise recalculated from the same distribution found a three-dimensional solution shown in Table 3 and Figure 3, with rotation and scaling that produce a ring-shaped attractor in the three-dimensional phase space:

Table 3: Parameters for a three-dimensional time-delay embedding found by the GA for a noisy sinusoidal time series.

	Dim. 1	**Dim. 2**	**Dim. 3**
Delay value	0	25	62
Rotation vs. dim. 1 (radians)	N/A	4.52	5.82
Rotation vs. dim. 2 (radians)	N/A	N/A	0.07
Scaling factor	1	3.8750	1.5625
Overall fitness value		-0.1742	
$-\sigma_d/\mu_d$ ratio		-0.1008	

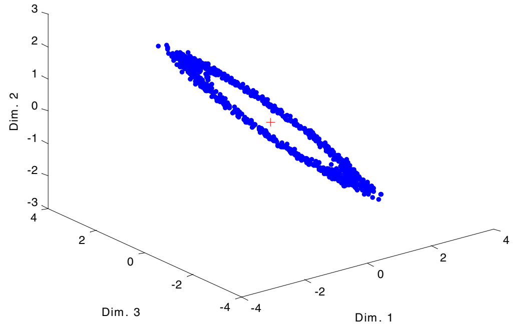


Figure 3: Plot of the three-dimensional time-delay embedding described in Table 3.

The three-dimensional result points out another characteristic of this technique: noise can cause the GA to find a more complex embedding (e.g., higher dimensionality or non-intuitive time delay values, rotation, or scaling) than might be required for a particular data set. The *b* parameter in the fitness function can be varied to compensate for the effect of noise on the dimensionality of the embedding found by the GA. Since this technique does not provide a similar mechanism for constraining rotation or scaling, minimizing measurement noise makes the GA more likely to find an attractor that is based on the actual dynamics of the system instead of the noise.

A more complex test was also presented to the GA, a simulation of a system governed by two attractors. An interleaved sinusoidal time series was developed by generating a time series using Equation 3 and doubling every second *x*(*t*) value. Depending on the time delay chosen, an embedding of this series may either separate or combine the two attractors. In this series, a time delay of an even number of samples results in an embedding that appears as two concentric circles, thus allowing the attractors to be separated visually. A delay of an odd number of samples gives an embedding that combines the attractors into one shape. The GA determined that the combined attractor had a more uniform radius than the union of the two separated attractors, and thus found an odd embedding delay:

Table 4: Parameters for a two-dimensional time-delay embedding found by the GA for the dual interleaved sinusoidal time series.

	Dim. 1	**Dim. 2**
Delay value	0	13
Rotation vs. dim. 1 (radians)	N/A	3.14
Scaling factor	1	0.8125
Overall fitness value	-0.3730	
$-\sigma_d/\mu_d$ ratio	-0.2590	

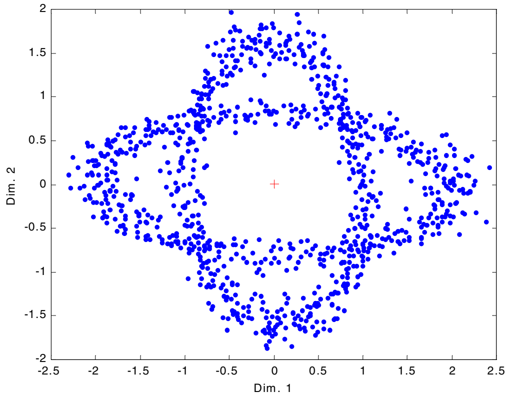


Figure 4: Plot of the time-delay embedding described in Table 4.

Having produced reasonable results with test data, this technique was then applied to several real-world attractors. One example is a time series representing the temperature of a room heated with a boiler and radiator. The time series is a set of 1,000 samples taken every two minutes during a simulation of a nonlinear model of the heating system. The time series was provided to the GA with a maximum dimensionality of 7. The GA reported the results in Table 5 and Figure 5:

Table 5: Parameters for a two-dimensional time-delay embedding found by the GA for the boiler/radiator time series.

	Dim. 1	**Dim. 2**
Delay value	0	88
Rotation vs. dim. 1 (radians)	N/A	0.61
Scaling factor	1	5.5

Overall fitness value	-0.5515
-σ_d/μ_d ratio	-0.3830

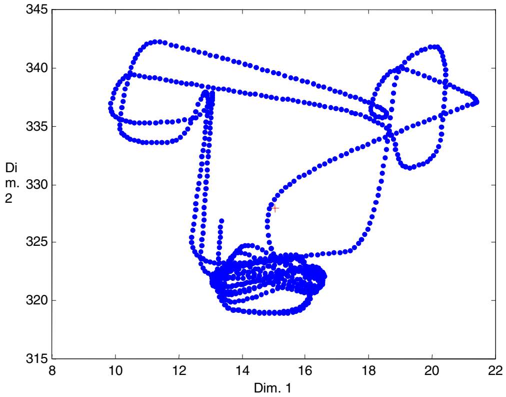


Figure 5: Plot of the boiler/radiator time-delay embedding described in Table 5.

In this case, the GA has found an interesting two-dimensional attractor. Three lobes appear in the plot, one of which is more densely populated than the others. While the attractor found by the GA is not circular, a dominant radius range does exist: a clear majority (693 of the 1,000) of the data points fall within a radius range of between 4 and 10 units. Figure 6 shows a histogram of the radii calculated for the data points. More importantly, the attractor does have some potentially useful structure.

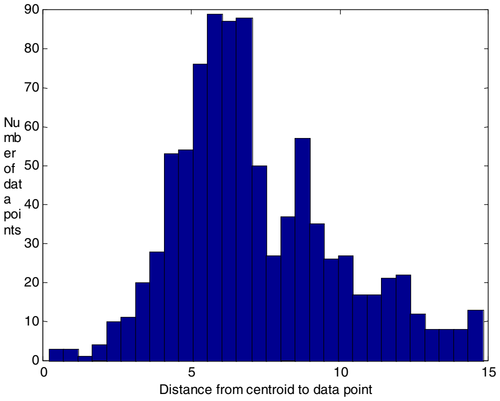


Figure 6: Histogram of radii between data points and centroid for the boiler/radiator time delay embedding described in Table 5.

Another interesting, and potentially lucrative, problem is to attempt to produce a model of the price of a stock. A 1,263-point time series containing the daily percentage price changes in the common stock of General Electric Co. between November 27, 1995, and November 24, 2000, was assembled and provided to the GA to find an attractor with a maximum dimensionality of 20. The results, shown in Table 6 and Figure 7 below, are essentially a three-dimensional cloud that indicates that the GA did not truly find an attractor:

Table 6: Parameters for a three-dimensional time-delay embedding found by the GA for a stock price time series.

	Dim. 1	**Dim. 2**	**Dim. 3**
Delay value	0	76	108
Rotation vs. dim. 1 (radians)	N/A	2.63	4.54
Rotation vs. dim. 2 (radians)	N/A	N/A	3.02
Scaling factor	1	12.75	14.75

Overall fitness value	-1.1257
-σ_d/μ_d ratio	-0.6514

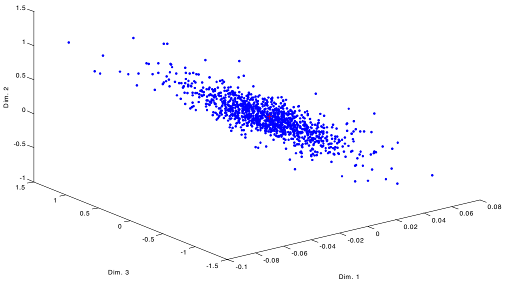


Figure 7: Plot of the stock price time-delay embedding described in Table 6.

Because the GA always reports the solution that it found to be best, there is no guarantee that the GA's solution is meaningful. In this case, it clearly is not. Unfortunately, the GA gives little guidance as to why it did not produce a meaningful result: perhaps the attractor requires more than 20 dimensions to become apparent, or perhaps the real attractor in this system has a geometry that is significantly different than a sphere. The higher-dimensionality case can be tested by allowing a larger maximum dimension, although there seems to be a point of diminishing returns in increasing dimensionality. The different geometry case can also be tested with a set of fitness functions tailored to different attractor geometries.

In some ways, this is a lucky result in that the three-dimensional view clearly shows that no attractor was truly found. This raises the question of how to detect a failure of this technique in higher dimensions, where a plot becomes infeasible. Two possible methods emerge from this example. One is to examine the fitness value of the GA's best solution. Knowing the fitness value, the dimensionality bias b, and the number of dimensions found by the GA, the ratio $-\sigma_d/\mu_d$ can be back-calculated and used as a measure of the uniformity of the attractor radius. This ratio can be compared directly to the values shown in Table 1. Based on the values in Table 1 and the examples in this paper, a ratio value of above approximately -0.4 tends to suggest that a meaningful attractor was found, and a ratio value of below -0.6 tends to suggest failure. Ratio values between these cutoffs are probably inconclusive, but may very well indicate success with noisy data.

Another possible method for determining the meaningfulness of a result is to examine the scaling factors on the various dimensions. If the scaling factors are dissimilar in magnitude, as in the stock price example, this suggests that the GA may have maximized the $-\sigma_d/\mu_d$ ratio with a meaningless combination of rotations and scaling.

One final example attempts to find a model for seismic activity. A 1,000-sample time series was taken from the east-west, broadband, high-gain sensor at the MA2 seismograph station at Magadan, Russia, during an earthquake of magnitude 6.4 that occurred on July 30, 2000, south of the Japanese island of Honshu (33.92 degrees north, 139.28 degrees east) (IRIS web site). Because previous investigation by the authors suggested that the attractor is likely to be of high dimensionality, the dimensionality bias b was relaxed to a value of 1.05 so as not to excessively penalize higher-dimensionality solutions. A maximum dimensionality of 20 was set.

The GA produced an 8-dimensional solution, summarized in Table 7. The rotations between the eight dimensions are omitted for clarity. Because of the high dimensionality, it is impractical to plot the data in the 8-dimensional phase space.

Table 7: Selected parameters for an eight-dimensional time-delay embedding found by the GA for a seismic time series.

	Delay value	**Scaling factor**
Dim. 1	0	1
Dim. 2	7	8.4375
Dim. 3	19	1.1875
Dim. 4	24	9.5
Dim. 5	26	9.625
Dim. 6	41	10.5625
Dim. 7	107	6.6875
Dim. 8	110	6.875

Overall fitness value	-0.6136
$-\sigma_d/\mu_d$ ratio	-0.4153

When examining this result, the natural question to ask is whether the result is meaningful. Because of the high dimensionality, the simplest tool to help determine this, plotting the data in the phase space, is not practical.

Examining the $-\sigma_d/\mu_d$ ratio and applying the benchmarks described earlier in this paper provide some guidance. The ratio value of -0.4153 falls in the inconclusive range, but it is very near the ratio found with the boiler/radiator system. Since the time series was sampled during an earthquake, presumably with a fair amount of noise present from sources such as seismic wave reflections from the earth's surface, the GA's result seems to be believable. The dimensionality of 8 also seems plausible, based on earlier investigation by the authors that showed no clear or emerging pattern in a number of embeddings in two and three dimensions.

The other tool developed in the stock price example, examining the scaling factors on each dimension for similarity, tends to lend some believability to the meaningfulness of the embedding: six of the eight dimensions have scaling factors between 6.6875 and 10.5625. The remaining two dimensions with scaling factors of 1 and 1.1875 may prove to be unnecessary. However, for now, all that can be said about this embedding is that it is an interesting candidate for further study.

6 CONCLUSIONS

Based on the examples in this paper, the GA technique appears to be a viable method for identifying appropriate time-delay embeddings for certain types of attractors. The technique works well when the attractor has a relatively uniform radius in phase space.

The main factor to consider when using this technique is that the GA finds an embedding that optimizes its fitness function, given the time series provided. Noise in the data can cause the GA to converge to an embedding that is more complex than necessary. In cases where the true attractor has a radius that is far from uniform, the GA still converges to the embedding that it found gives the data the most uniform radius. In some cases, this embedding represents the data as a random jumble of points, yielding no useful information. This paper discusses some techniques to help determine if this has occurred on a given data set, but at some point the user must decide whether to pursue higher dimensionality, different attractor geometries, or an entirely different technique.

The area of alternate attractor geometries appears to be an interesting area for future research. To change this technique to operate on a different attractor geometry, a new fitness function that detects that particular geometry is needed. It is even conceivable that the GA could select among a number of geometries for a given embedding, assuming that the fitness functions can be balanced so that a particular fitness value represents a the same level of fitness across the different geometries.

Another interesting area for future research is optimization of the GA itself. Faster convergence and speed optimization translate directly to decreased time to reach a result. The GA used in this paper is a simple, generic GA; it is likely possible to tailor the GA for faster convergence in this application.

While this technique is certainly not a "silver bullet" to find the optimal time-delay embedding for any time series, it is another tool that, in many circumstances, can provide useful results.

References

Henry D. I. Abarbanel, *Analysis of observed chaotic data*. New York: Springer-Verlag, 1995.

Andy Burbanks, *Gallery of mathematics: hyperspace structures – the hypercube*, Oct. 28, 1996, http://info.lboro.ac.uk/departments/ma/gallery/hyper/cube.html.

D. Dumitrescu, B. Lazzerini, L. C. Jain, and A. Dumitrescu. *Evolutionary computation*. Boca Raton, Fla.: CRC Press, 2000.

A. M. Fraser and H. L. Swinney, "Independent coordinates for strange attractors from mutual information", *Physical Review A*, no. 33, p. 1134-1140, 1986.

S. G. Hoggar, *Mathematics for computer graphics*. Cambridge, U.K.: Cambridge University Press, 1992.

Incorporated Research Institutions for Seismology, http://www.iris.washington.edu/.

Holger Kantz and Thomas Schreiber, *Nonlinear time series analysis*. Cambridge, U.K.: Cambridge University Press, 1997.

M. B. Kennel, R. Brown, and H. D. I. Abarbanel, "Determining minimum embedding dimension using a geometrical construction", *Physical Review A*, no. 45, p. 3403-3411, 1992.

Richard J. Povinelli, *Time series data mining: Identifying temporal patterns for characterization and prediction of time series events*. Doctoral dissertation. Milwaukee, Wis.: Marquette University, 1999.

T. Sauer and J. A. Yorke, "How many delay coordinates do you need?", *International Journal of Bifurcation and Chaos*, no. 3, 1993.

F. Takens, "Detecting strange attractors in turbulence", *Lecture Notes in Mathematics*, no. 898, 1981.

M. Vidyasagar, *Nonlinear systems analysis*, 2nd ed. Upper Saddle River, N.J.: Prentice-Hall, 1993.

A Genetic Algorithm for Generating a Steiner Tree with Wire Sizing and Buffer Insertion

Shin'ichi Wakabayashi **Masakazu Ohsako**
Graduate School of Engineering, Hiroshima University
4-1, Kagamiyama 1 chome, Higashi-Hiroshima 739-8527 JAPAN
E-mail: wakaba@computer.org

Abstract

This paper proposes a genetic algorithm for generating a rectilinear Steiner tree with wire sizing and buffer insertion for the interconnect optimization problem in VLSI layout design. In the proposed genetic algorithm, each chromosome represents the topological structure of a Steiner tree. An evaluation function is given to map it into the layout of a Steiner tree with wire sizing and buffer insertion. Experimental results show that the algorithm effectively produces Steiner trees better than ones produced by the previous method.

1 Introduction

The routing problem in VLSI physical design is generally formulated as the problem of finding a rectilinear Steiner tree, which connects a given source with a set of sinks with vertical and horizontal wire segments. The minimum Steiner tree construction problem is an NP-hard problem, and thus many heuristic algorithms have been proposed [6, 14]. On the other hand, with the advent of sub-micron geometries in semiconductor technology, wire resistance becomes a significant contributor to signal delay, and thus routing should be performed under the timing constraints with an appropriate delay model [1, 2, 3, 4].

There are various optimization techniques that can be applied to solve the interconnect optimization problem under the timing constraints. Those are wire-length minimization, interconnect topology optimization, device sizing, buffer insertion, and wire-size optimization [3]. Those techniques could be discussed independently, and in fact, there have been many works on each subject. However, it is apparent that we would obtain better results if more than one techniques were effectively combined. For example, Okamoto and Cong proposed an algorithm to produce a rectilinear Steiner tree with wire sizing and buffer insertion [13]. References [1, 3] gave a summary on those previous works.

In this paper, we propose a genetic algorithm to produce a Steiner tree with wire sizing and buffer insertion. Genetic algorithms (GAs) [8] are known to be robust heuristic algorithms to solve optimization problems, and for VLSI design areas, a number of GA based algorithms have been presented [11, 12]. For the (rectilinear) Steiner construction problem, a few GAs have been also proposed [9, 10]. In those previous GAs, a chromosome directly represents the geometry of a Steiner tree. We have a different approach to representing a solution of the problem. In the proposed GA, each chromosome represents a topological structure of a Steiner tree. An evaluation function is given to map it into the layout of a Steiner tree with wire sizing and buffer insertion. As the interconnect delay model, we adopt the Elmore delay model [7].

The proposed algorithm was implemented and compared with the previous algorithm by Okamoto and Cong [13]. Experimental results show that the algorithm efficiently produces better Steiner trees in VLSI interconnect optimization.

This paper is organized as follows. In Section 2, the delay model is given and the problem is formulated. In Section 3, the proposed algorithm is presented. Section 4 shows experimental results to evaluate the proposed algorithm, and finally, Section 5 concludes with possible directions for future research.

2 Preliminaries

2.1 Delay Model

As in most previous work on interconnect layout optimization, we adopt the Elmore delay model [7] for interconnects. For wire e, let l_e, c_e and r_e denote its length, capacitance, and resistance, respectively. Further, let e_v denote the wire entering node v from its parent. We use the following model for interconnect delay D_{wire} and buffer delay D_{buff}:

$$\begin{aligned} c_e &= (c_a \cdot w_e + c_f), \\ r_e &= r_0 \cdot l_e / w_e, \\ D_{wire}(e_v) &= r_{e_v} \times (c_{e_v}/2 + c(T_v)), \\ D_{buff}(b, c_l) &= d_b + r_b \cdot c_l, \end{aligned}$$

where c_a, c_f and r_0 are area capacitance, fringing capacitance, and resistance for unit-width unit-length wire, respectively, T_v is the subtree rooted at v, and $c(T_v)$ is the capacitance of *dc-connected subtree* [1] in T_v rooted at T_v's root. d_b and r_b are buffer b's intrinsic delay and output resistance, respectively, and c_l is the load on buffer b.

The Elmore delay from source s_0 to sink s_i is

$$\begin{aligned} t_{Elmore}(s_0, s_i) &= \sum_{e_v \in path(s_0, s_i)} D_{wire}(e_v) \\ &\quad + \sum_{b \in path(s_0, s_i)} D_{buff}(b, c_l). \end{aligned}$$

2.2 Problem Formulation

As the same discussed in [13], we use required arrival time and total capacitance as our optimization objectives. The required arrival time at the root of tree T_v is defined as follows:

$$q(T_v) = \min_{u \in sinks(T_v)} (q_u - delay(v, u)),$$

where q_u is the required arrival time of sink u, $sinks(T_v)$ is a set of sinks of tree T_v, and $delay(v, u)$ is the delay from v to u defined by the delay model discussed above. The total capacitance of tree T_v, denoted $c_{total}(T_v)$, is defined as follows:

$$c_{total}(T_v) = \sum_{e \in T_v} c_e + \sum_{u \in buffers(T_v)} c_u + \sum_{u \in sinks(T_v)} c_u,$$

where $buffers(T_v)$ is a set of buffers in tree T_v and c_u is loading capacitance of buffer or sink u.

The timing-driven rectilinear Steiner problem in this paper is formulated as follows: Given a source s_0 and sinks s_1, s_2, $\ldots$, s_n of a signal net S with given positions and a required arrival time associated with s_i $(1 \leq i \leq n)$, find a rectilinear Steiner tree T_{s_0} that connects S and has wire sized and buffers inserted. The objective is to maximize $q(T_{s_0})$ with minimization of $c_{total}(T_{s_0})$ as the secondary objective.

An alternative formulation of the problem can be also obtained by defining the objective as minimization of $c_{total}(T_{s_0})$ as the main objective under the constrain of $q(T_{s_0}) > 0$. The proposed algorithm can handle both versions of the problem.

3 The Algorithm

3.1 Outline of the Algorithm

The algorithm presented in the following is a *genetic algorithm (GA)*. GA is known to be a robust heuristic algorithm to complex optimization problems [8]. For VLSI physical design, many GAs have been also proposed [11, 12].

The proposed GA is a generational GA, and maintains the population consisting of m chromosomes, each representing a rectilinear Steiner tree. Chromosomes in the current generation are recombined and mutated, and selection is performed to produce a new generation. Mapping from a chromosome to a tree is given, and the total wire length and the maximum source-sink delay of the tree are regarded as the fitness values of the chromosome. The proposed mapping is very effective to explore the search space efficiently. Based on the fitness values, tournament selection is performed to construct a new generation with elitist strategy. The algorithm repeats those procedures within the user specified number. An overview of the algorithm is shown in Figure 1.

```
generate($P_C$);
evaluate($P_C$);
repeat noOfGenerations times:
  $P_N := \emptyset$;
  repeat noOfOffspring times:
    select $p_1, p_2 \in P_C$;
    $P_N := P_N \cup$mutate(crossover($p_1$, $p_2$));
  end;
  evaluate($P_C \cup P_N$);
  $P_C :=$ select($P_C \cup P_N$);
end;
```

Figure 1: Overview of the algorithm.

3.2 Genotype

In GA, a *genotype* is a coding of the information constituting a chromosome. In our problem, we should repre-

---

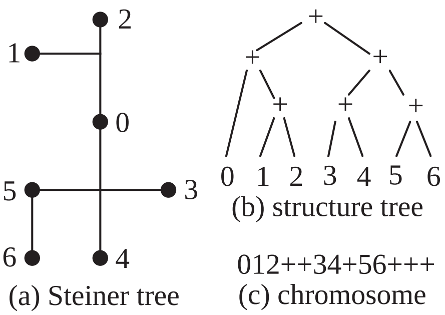
[1] A *dc-connected subtree* is a subtree, whose edges are directly connected (i.e., there is no buffer among them) to its root.

sent a rectilinear Steiner tree with an appropriate coding. There may be a number of possible representations of a Steiner tree. In general, for representing a Steiner tree, there are two types of information, that is, topological information and geometrical information. The former specifies the parent-children relations among nodes, and the latter specifies how each wire segment in the tree is actually laid out. In the previously proposed GAs for the Steiner tree problem, those two types of information were both coded in a chromosome. In [9], a GA for the Steiner tree problem was proposed, in which a chromosome was an assembly of the x, y-positions of a fixed number of Steiner points. Since the layout area of a Steiner tree in VLSI layout design is large in general, using this chromosome, it would be very difficult to realize the efficient search in the solution space of the problem. In [10], a GA for the rectilinear Steiner tree problem was proposed, in which a chromosome consists of $n-1$ binary symbols and $n-2$ symbols selected from an alphabet of n symbols, where n is the number of points to be connected with a Steiner tree. Due to the Cayley's Formula and Hanan's theorem, from this chromosome, a Steiner tree could be constructed. Since, in this paper, we should take not only the wire length but also the signal propagation delay into account, this coding scheme may not be appropriate.

In this paper, a chromosome only specifies the topological information of a Steiner tree. Geometrical information of the tree will be determined during the fitness evaluation, that is, a Steiner tree will be constructed according to the topological information specified by the chromosome.

Topological information of a Steiner tree is coded as follows. The coding is defined recursively. Let T be a rectilinear Steiner tree, and T_u be a subtree of T. Assume that T_u consists of two subtrees T_v and T_w, each of which contains at least one source or sink. Let $code(T_v)$ and $code(T_w)$ be the coded strings of T_v and T_w, respectively. Then the coded string of T_u is defined as the concatenation of $code(T_v)$, $code(T_w)$, and the special symbol $+$, that is, $code(T_u) = code(T_v)code(T_w)+$. If T_u contains only one source or sink, then $code(T_u)$ is defined as s_i $(0 \leq i \leq n)$.

A chromosome defined here specifies how the tree is constructed. The coding can be interpreted as a tree, whose leaves are a source and sinks, and internal nodes are the special symbols $+$. We call this tree the *structure tree*. From the structure tree, its geometrical information is determined during the fitness evaluation. Figure 2 shows an example of a Steiner tree and its corresponding structure tree as well as the chromosome.



(a) Steiner tree (b) structure tree (c) chromosome

Figure 2: Genotype.

3.3 Fitness

3.3.1 Getting the geometry of a tree

In a GA, to evaluate each chromosome, a *fitness function* is required. Since each chromosome only specifies the topological information, to evaluate its fitness, the Steiner tree should actually be laid out according to the topological information given by the structure tree.

Steiner tree construction in the proposed algorithm is performed as follows. Let S_u be a subtree of the structure tree given by the chromosome, and S_v and S_w be the subtrees constituting S_u, where u, v, and w are roots of those subtrees, and v and w are the two children of u. Assume that S_v and S_w have been already laid out. Then, we determine a route from a node in S_v to a node in S_w to construct a Steiner subtree corresponding to S_u. Note that, in this case, nodes to be connected in the subtrees may be existing nodes or intermediate points in the existing edges.

This Steiner construction algorithm is similar to the Kruskal's algorithm for finding a minimum spanning tree of a given graph [5]. However, in the proposed algorithm, the ordering of matching nodes or edges is specified by the structure tree. To complete the description of the algorithm, we should specify how to choose the pair of two nodes or edges to connect two subtrees. One possible idea is to choose the node (edge) pair with the minimum distance. However, this idea has two drawbacks. First, this heuristic is so strong that it would cause the premature convergence into a local optimum. Second, since all node pairs (i.e., edges) are required to be checked, time complexity to construct a tree would become $O(n^2)$. Since the fitness evaluation is executed in many times in one GA execution, this is not feasible.

To reduce the computation time, and to avoid causing premature convergence, we restrict the range of searching in combining two subtrees. Let v and w be two roots of subtrees, T_v and T_w, to be combined. Let $E(v)$ and $E(w)$ be sets of edges in T_v and T_w, in which each edge resides within the distance *max_edge_level* from v and w, respectively. The parameter *max_edge_level* is specified by the user. When combining the two subtrees, only edges in $E(v)$ and $E(w)$ are examined so that the time complexity in constructing the whole tree is reduced from $O(n^2)$ to $O(n)$.

3.3.2 Calculating the objective function

Once the Steiner tree is constructed from the chromosome, we perform wire sizing and buffer insertion, and calculate the required arrival time and the total capacitance of tree. The algorithm to produce a buffer-inserted, and wire-sized rectilinear Steiner tree, from which the objective function is evaluated, is based on the dynamic programming technique, that was originally proposed in [13]. By carefully looking how the Elmore delay is calculated for a given tree, we see that the delay calculation is done by two phases, one is the bottom up phase to calculate the delay of each node, and the other is the top down phase to calculate the total delay from the source to each sink. If more than one kinds of wire width are allowed to be used, and/or buffers may be inserted within each wire segment, then there are possible combinations of them, and for each node, the node delay is formulated as dynamic programming. Let T'_i denote T_i with e_i. In the algorithm, at each node i, a set of triples $(q(T'_i), c_{total}(T'_i), c(T'_i))$ is computed and maintained for possible configuration of buffer insertion and wire sizing for T'_i. In [13], this triple is called an *option*, and an algorithm for finding a set of options was proposed. Note that, in [13], calculation of options is done during the construction of a Steiner tree. On the other hand, in the proposed algorithm, since the Steiner tree was already laid out from a given chromosome, we only calculate a set of options for each node.

As mentioned, options are defined at each node. However, in the option computation, we also introduce the set of options for each edge. Let Z_u and Z_v be sets of options at node u and v, respectively. Assume that there is an edge $e = (u, v)$, and Z_v has been computed. Then, a set of options for edge $e = (u, v)$, denoted Z'_e, is computed with the procedure shown in Figure 3. In this procedure, W means the set of wire width, which can be used in the layout, and e_w and $c(e_w)$ show the edge laid out with the wire width $w \in W$, and its capacitance. There may be redundant options, which will be no use in the future computation of options. That is, there are two options, (q, p, r) and (q', p', r'), and if $q > q'$, $p < p'$, and $r < r'$, then (q', p', r') is said to be redundant, and it is removed from the set of options in the end of procedure.

For each node u, a set of options is computed as follows. If u is a sink (i.e., a leaf of the tree), q is set to the required arrival time at sink u, denoted $rat(u)$, and p and r are set to the load capacitance of sink u, denoted c_l. If u is an internal node of the tree, without loss of generality, we assume that u has two descendant nodes, denoted v and w. If node u has more than two descendant nodes, we can transform the tree so that any internal node has two descendant nodes by splitting such nodes and inserting zero-length edges into those splitted nodes. Let $e1 = (u, v)$, and $e2 = (u, w)$. Then, a set of options at node u, denoted Z_u, is computed from Z'_{e1} and Z'_{e2} with the procedure shown in Figure 4.

```
procedure edge_options($e = (u, v)$);
begin
  $Z'_e := \emptyset$;
  for each $(q, p, r) \in Z_v$ do
    for each $w \in W$ do
      $Z'_e := Z'_e \cup \{(q - D_{wire}(e_w), p + c(e_w), r + c(e_w))\}$;
  for each $(q, p, r) \in Z'_e$ do
    $Z'_e := Z'_e \cup \{(q - D_{buff}(b, p), c_l, r + c_l)\}$;
  remove_redundant($Z'_e$);
end;
```

Figure 3: The procedure to compute edge options.

```
procedure node_options(u);
begin
  if $u$ is a sink then $Z_u := \{(rat(u), c_l, c_l)\}$
  else begin
    $Z_u := \emptyset$;
    /* Assume that $e1 = (u, v)$ and $e2 = (u, w)$ */
    for each $(q1, p1, r1) \in Z'_{e1}$ do
      for each $(q2, p2, r2) \in Z'_{e2}$ do
        $Z_u := Z_u \cup \{(min(q1, q2), p1 + p2, r1 + r2)\}$;
    for each $(p, q, r) \in Z_u$ do
      $Z_u := Z_u \cup \{(q - D_{buff}(b, p), c_l, r + c_l)\}$;
    remove_redundant($Z_u$);
  end;
end;
```

Figure 4: The procedure to compute node options.

As mentioned, node options are computed in the bottom up manner from sinks to the root of the tree. After option computation, the second phase is performed to determine buffer insertion and wire width assignment in the top down manner. If the problem to be solved is to find a solution maximizing the required arrival time of the source, then we choose an option of the root with the maximum required arrival time, and then determine the wire width and buffers. If the problem to be solved is to find a solution minimizing the total capacitance of routing, then, we choose an option of the root with the minimum total capacitance, and then determine the wire width and buffers.

3.4 Crossover Operator

To generate a set of chromosomes belonging to the population in the next generation, two chromosomes are randomly selected and recombined with a crossover operator with probability p_c. The crossover operator adopted in the proposed algorithm is called the *subtree interchange*. In the following, we treat each chromosome as its corresponding structure tree. Let T and U be the structure trees to be recombined. First, in each structure tree, a $+$ node is randomly selected from all $+$ nodes except its root. Let T_v and U_w be two subtrees whose roots are selected $+$ nodes. Second, two subtrees, T_v and U_w, are interchanged. Interchanging subtrees usually causes the inconsistency in other parts in T and U. To fix this inconsistency, duplicate leaves may be renamed or removed, and new leaves may be added if necessary.

We explain how the crossover is performed using an example shown in Figure 5. There are two parent chromosomes, denoted T for *parent 1* and U for *parent 2*. Assume that subtrees surrounded with dotted lines are interchanged. After interchanging subtrees, in *child 1*, leaves 4 and 6 are duplicate, and there is no leaf 2 in the whole tree. Hence leaf 4 outside the subtree is removed, and leaf 6 outside the subtree is renamed with 2. In *child 2*, in the subtree, leaf 2 is duplicate and there is no leaf 4 nor 6 in the whole tree. Hence leaf 2 outside the subtree is renamed with 6, and new leaf 4 is added as the leftmost leaf.

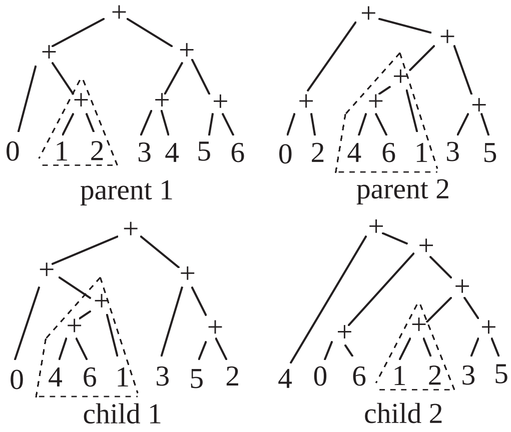


Figure 5: Crossover.

3.5 Mutation Operator

After performing the crossover, each newly created chromosome is mutated with the mutation operator. The mutation operator adopted in the algorithm is as follows. We regard a chromosome to be mutated as a structure tree T. First, two distinct nodes in T are arbitrarily chosen, and let those nodes be u and v. Let T_u and T_v be subtrees whose roots are u and v, respectively. If there is no node, which is contained in both T_u and T_v, then two subtrees are actually interchanged. Otherwise, the mutation failed. The probability of applying mutation is denoted as p_m.

3.6 Selection

After applying mutation, the population of next generation is constructed with tournament selection with elitist strategy. The tournament size is 2.

4 Experimental Results

We have implemented the proposed algorithm with the C language. To show the effectiveness of the proposed method, we performed experiments to compare the proposed method with the previous method proposed by Okamoto and Cong [13], which was implemented with also the C language. In the experiments, both methods were executed on a UltraCOMPstation model 60 workstation (CPU: UltraSPARC-II, 300MHz). As the test data, we randomly generated two sets of four signal nets, and applied the algorithms to those. For one set of test data, the chip area was $10 \times 10mm^2$, and the required arrival time of each sink was set to 1.0ns. For the other set, the chip area was $15 \times 15mm^2$, and the required arrival time of each sink was set to 1.5ns. For each set, the numbers of sinks of test data were 5, 10, 20, and 30, respectively, and the source was located at the center of the chip area, and each sink was randomly generated in the chip area.

In the experiments, the delay of a Steiner tree was calculated with the Elmore delay model described in Subsection 2.1 with the following parameter values: $c_a = 0.044$fF/μm, $c_f = 0.055$fF/μm, $r_0 = 0.076\Omega$/μm, $d_b = 36.4ps$, $r_b = 180\Omega$, and $c_l = 23.4$fF. We assume that there were three wiring layers, and the wire width of each layer is 0.18, 0.25, and 0.5 μm, respectively. As the GA parameters, we set the population size to 30, crossover and mutation probabilities to 0.6 and 0.01, respectively, generation gap to 0.6, and the maximum number of generations to 200. We also set max_edge_level in the fitness evaluation phase to 3. Those parameter values were determined according to the results of preliminary experiments.

For each test data, we executed the proposed algorithm in 20 times to solve a respective instance of the problem. For each run of the algorithm, a set of 30 individuals were randomly generated as the initial population for the proposed algorithm. The previous method has a parameter, denoted α, to control the tradeoff between signal delay and wire

Table 1: Experimental results for maximizing the required arrival time ($area = 10 \times 10mm^2$, $timing = 1.0ns$).

$\#sink$		RAT[ns]	C_{total}[pF]	WL[μm]	$\#B$	Gen	CPU[sec]
5	Proposed ($best$)	**0.68**	1.58	18509	3	0	51.6
	Proposed (ave)	0.68	1.48	17516	4.1	8.9	48.9
	[13]	0.64	1.29	16131	2	—	0.4
10	Proposed ($best$)	**0.68**	2.72	31300	7	93	197.9
	Proposed (ave)	0.68	2.61	29320	7.4	66.9	186.4
	[13]	0.64	2.25	26281	5	—	2.4
20	Proposed ($best$)	**0.62**	5.06	53927	20	104	398.6
	Proposed (ave)	0.61	4.68	49682	18.3	142.4	400.4
	[13]	0.45	3.93	43186	10	—	5.5
30	Proposed ($best$)	**0.62**	7.37	79180	31	164	704.7
	Proposed (ave)	0.59	6.22	64250	27.9	141.2	706.4
	[13]	0.48	4.72	50246	14	—	11.9

Table 2: Experimental results for minimizing the total capacitance ($area = 10 \times 10mm^2$, $timing = 1.0ns$).

$\#sink$		RAT[ns]	C_{total}[pF]	WL[μm]	$\#B$	Gen	CPU[sec]
5	Proposed ($best$)	0.05	**1.14**	15547	1	0	68.4
	Proposed (ave)	0.02	1.15	15547	1.0	4.1	72.2
	[13]	0.06	1.16	16131	0	—	0.2
10	Proposed ($best$)	0.002	**1.78**	23050	2	0	164.4
	Proposed (ave)	0.003	1.78	23185	1.9	29.6	174.5
	[13]	0.04	1.88	25107	0	—	1.1
20	Proposed ($best$)	0.03	**3.11**	39177	3	142	346.1
	Proposed (ave)	0.02	3.19	39942	3.6	126.9	347.8
	[13]	0.02	3.44	43186	3	—	8.0
30	Proposed ($best$)	0.001	**3.99**	46383	5	182	553.9
	Proposed (ave)	0.005	4.14	48287	6.5	138.9	582.9
	[13]	0.01	4.26	49763	8	—	17.9

length [13]. We set α to 0.4 as recommended in [13]. Note that the previous method was a deterministic one, and thus for each instance of the problem, the method was executed once.

Table 1 shows the experimental results for the problem of maximizing the required arrival time at the source node for the first set of test data. In the table, $\#sink$ means the number of sinks. RAT is the required arrival time at the source of the obtained solution. C_{total} is the total capacitance of the obtained solution. WL shows the total wire length of the solution, and $\#B$ is the number of buffers used in the solution. Gen shows the generation, at which the best solution was obtained in the run of the proposed method. CPU shows the average CPU time in seconds to execute the algorithms. As mentioned, the proposed algorithm was executed in 20 times for each test data, and hence we show the best and average results of the proposed method. Figures in the bold type style show the best values among the proposed and previous methods. Similarly, for the first set of test data, Table 2 shows the experimental results for the problem of minimizing the total capacitance under the constraint that the required arrival time of the source should be positive. Tables 3 and 4 show the experimental results of the same two cases for the second set of test data. Note that there was a case that $Gen = 0$. It means that the best solution was randomly generated as an initial solution.

From those tables, we see that the proposed algorithm produced much better solutions than the previous method. From Table 1, the required arrival time of the solution of the proposed method was improved by a 19.9% ($best$) and a 17.8% (ave) on average, and from Table 2, the total capacitance of the solution of the proposed method was improved by a 5.7% ($best$) and a 4.1% (ave) on average. From Table 3, the required arrival time of the solution of the proposed method was improved by a 49.8% ($best$) and a 48.6% (ave) on average, and from Table 4, the total capacitance of the solution of the proposed method was improved by a 10.2% ($best$) and a 8.3% (ave) on average.

Table 3: Experimental results for maximizing the required arrival time ($area = 15 \times 15mm^2$, $timing = 1.5ns$).

$\#sink$		RAT[ns]	C_{total}[pF]	WL[μm]	$\#B$	Gen	CPU[sec]
5	Proposed ($best$)	**0.92**	2.60	30015	9	0	79.6
	Proposed (ave)	0.92	2.29	26387	7.4	1.2	83.2
	[13]	0.46	1.98	24246	3	—	0.3
10	Proposed ($best$)	**0.88**	3.33	38028	9	34	105.8
	Proposed (ave)	0.88	3.25	37095	9.9	52.5	122.1
	[13]	0.76	3.14	39534	6	—	3.1
20	Proposed ($best$)	**0.87**	5.84	65610	21	152	370.4
	Proposed (ave)	0.86	5.69	62592	20.9	116.1	345.1
	[13]	0.59	4.54	54306	10	—	4.5
30	Proposed ($best$)	**0.87**	7.96	86353	31	115	490.7
	Proposed (ave)	0.85	8.01	86460	32.4	131.2	571.2
	[13]	0.64	6.32	74075	14	—	8.7

Table 4: Experimental results for minimizing the total capacitance ($area = 15 \times 15mm^2$, $timing = 1.5ns$).

$\#sink$		RAT[ns]	C_{total}[pF]	WL[μm]	$\#B$	Gen	CPU[sec]
5	Proposed ($best$)	0.04	**1.65**	23394	1	0	82.2
	Proposed (ave)	0.04	1.65	23394	1.0	9.9	94.6
	[13]	0.03	1.84	24246	2	—	0.2
10	Proposed ($best$)	0.02	**2.45**	33713	2	148	118.4
	Proposed (ave)	0.01	2.45	33317	1.9	60.8	122.4
	[13]	0.01	2.71	36952	2	—	1.5
20	Proposed ($best$)	0.01	**3.67**	47570	4	107	328.3
	Proposed (ave)	0.01	3.80	48019	5.4	100.3	332.6
	[13]	0.01	4.19	54306	5	—	4.0
30	Proposed ($best$)	0.02	**5.34**	65391	7	188	525.1
	Proposed (ave)	0.01	5.60	68959	9.8	145.9	534.4
	[13]	0.004	5.84	74075	7	—	11.0

Superiority of the proposed method may be explained by pointing out the fact that the ability of the proposed algorithm to search in the solution space is much powerful than the previous method. The proposed method was based on a GA, and a GA in general has a quite good capability to explore in the solution space. It is also often said that a GA has less powerful in the exploitative search (i.e., search in the neighborhood of a current search point). Since, in the proposed algorithm, the neighborhood search is realized in the deterministic manner as the fitness evaluation by determining the geometry of a Steiner tree, the proposed algorithm can successfully find a good solution.

The proposed algorithm has another advantage. Since the proposed algorithm produces a set of Steiner trees, and they are different in their characteristics, such as the geometry, wire length, and the wire delay, those trees can be considered as a set of alternative routes for a given net. The designer and/or the routing program can choose one of them as a final routing solution based on their own criterion.

A disadvantage of the proposed method was its large computation time. Most of the computation time of the proposed algorithm was devoted to perform the computation of options in the fitness evaluation. Thus, developing an effective pruning method in the option computation will be required. From the experimental results, for the small size data such as the case with 10 sink nodes, 100 generations will be enough to get a good solution, and reducing the maximum number of generations contributes to reduce the computation time.

5 Conclusion

In this paper, we proposed a genetic algorithm for constructing a rectilinear Steiner tree with wire sizing and buffer insertion in VLSI interconnect optimization. Experimental results show that the proposed algorithm effectively produces Steiner trees better than ones produced by the previous method.

As the future work, to reduce the computation time is necessary. Since the framework of the proposed algorithm is so flexible that it is easy to change the delay model as well as the objective functions used in the algorithm. For example, the clock tree synthesis may be solved by the proposed algorithm with a minor modification.

Acknowledgments

This research was supported in part by Grant-in-Aid for Scientific Research (C)(2)(No.12838008) from the Ministry of Education, Culture, Sports, Science and Technology, Japan.

References

[1] J. Cong, L. He, C. -K. Koh, and P. H. Madden, Performance optimization of VLSI interconnect layout, *Integration, the VLSI Journal,* 21, 1&2, pp.1–94 (1996).

[2] J. Cong, Modeling and layout optimization of VLSI devices and interconnects in deep submicron design, *Proc. Asia and South Pacific Design Automation Conference*, pp.121–126 (1997).

[3] J. Cong, Z. Pan, L. He, C. -K. Koh, and K. -Y. Khoo, Interconnect design for deep submicron ICs, *Proc. International Conference on Computer-Aided Design*, pp.478–485 (1997).

[4] J. Cong, and C. -K. Koh, Interconnect layout optimization under higher-order RLC model, *Proc. International Conference on Computer-Aided Design,* pp.713–720 (1997).

[5] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms,* The MIT Press (1990).

[6] D.-Z. Du, J. M. Smith, and J. H. Rubinstein (Eds.), *Advances in Steiner Trees,* Kluwer Academic Publishers (2000).

[7] W. C. Elmore, The transient response of damped linear network with particular regard to wideband amplifier, *J. Applied Physics,* 19, pp.55–63 (1948).

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

[9] J. Hesser, R. Männer, and O. Stucky, Optimization of Steiner trees using genetic algorithms, *Proc. 3rd International Conference on Genetic Algorithms,* pp.231–236 (1989).

[10] B. A. Julstrom, A genetic algorithm for the rectilinear Steiner problem, *Proc. 5th International Conference on Genetic Algorithms,* pp.474–480 (1993).

[11] J. Lienig, and J. P. Cohoon, Genetic algorithms applied to the physical design of VLSI circuits: A survey, *Proc. Parallel Problem Solving from Nature IV,* pp.839–848 (1996).

[12] P. Mazumder, and E. M. Rudnick, *Genetic Algorithms for VLSI Design, Layout & Test Automation,* Prentice-Hall PTR (1999).

[13] T. Okamoto, and J. Cong, Buffered Steiner tree construction with wire sizing for interconnect layout optimization, *Proc. International Conference on Computer-Aided Design*, pp.44–49 (1996).

[14] M. Sarrafzadeh, and C. K. Wong, *An Introduction to VLSI Physical Design,* McGraw-Hill (1996).

Reconstruction of Particle Flow Mechanisms with Symbolic Regression via Genetic Programming

Klaus Weinert and Marc Stautner

Dept. of Machining Technology
University of Dortmund
D-44227 Dortmund, Germany

Abstract

Modeling the particle flow mechanisms in orthogonal cutting of turning processes is a vital task in mechanical engineering. This paper presents a new approach that differs from techniques like finite element analyzes (FEA) or molecular dynamics (MD). Using symbolic regression, a genetic programming system evolves mathematical formulae that describe the trajectories of single particles of steel recorded during the turning process by a high-speed camera.

1 INTRODUCTION

Modelling the chip-building process in metal cutting has been in the centre of interest for a long time. The chip geometry, the material movement and the thermal processes which take place at the centre of cutting (the so called "contact zone") are decisive for manufacturing quality, reduction of machining times and tool wear. Most existing approaches incorporate FE- or MD-methods as well as analytical techniques based on cutting force models. Some of them can be found in [Ina97, SB93, SITU94, XBZ98]. The ideas presented in this paper differ from those. No external knowledge is provided besides the images of the process taken by a high-speed camera. The principle of the system works similar to a human observer [War74] but of course, a computer can deal much better with the large amount of information provided by the process images. In order to "simulate" a human observer, as he is searching for the process-describing formulae, Genetic Programming is utilized. This method has proven to be helpful to search for solutions within a search space, especially when dealing with finding formulae matching some given discrete requirements, a task which is known as "symbolic regression". The evolved formulae are expected to reveal geometrical aspects of the cutting process. They will be used as a basis for advanced simulation tools, as well as for analytical investigations in the field of metal cutting. Before they are applicable, they have to be validated through correspondence with existing formulae or with observed properties of the cutting process. Furthermore, the system may produce some unexpected results, which do not correspond to any known properties of the process. If such phenomena occur, new knowledge about the process has been generated. This knowledge may be helpful to support the development of new simulation tools, which are able to increase the productivity of the process as well as its reliability.

The task of describing the particle flow mechanisms in orthogonal cutting leads to the problem of describing the particle trajectories of the crystalline particles in the metallic workpiece. This implies the problem of finding the mathematical functions which underly these trajectories.

These functions may be illustrated by relations of this type:

$$\vec{f}(t) = \begin{pmatrix} f_x(t) \\ f_y(t) \end{pmatrix} \quad resp. \quad f(x,y) = 0$$

To succeed in this objective, some problems have to be solved. The first task is to obtain the trajectories of the particles in the crystalline structure of the metal workpiece. Until now the studies are based on movies which were recorded by Warnecke [War74] in the 1970's. A concept for the experimental setup, shown in figure 1, has been developed:

A "Boehringer M-670" turning lathe will be used. The cutting process will be filmed by a high speed camera "Weinberger SpeedCam+", which allows to

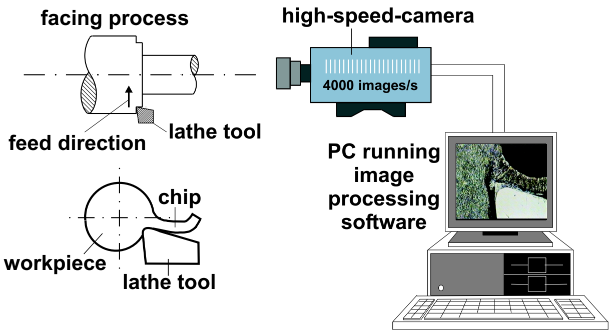


Figure 1: Experimental Setup of Camera and Turning Machine

take 4000 images per second. The size of the display window is 0.44 mm by 0.33 mm. In Figure 2 the trajectory of a single particle has been sketched on to the frame of the digitized film.

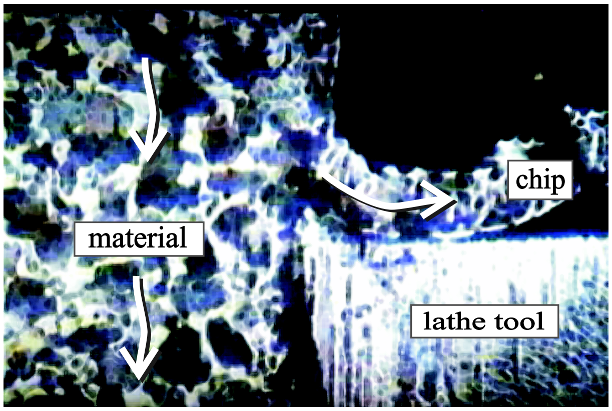


Figure 2: Single Frame Taken from the Film Sequence

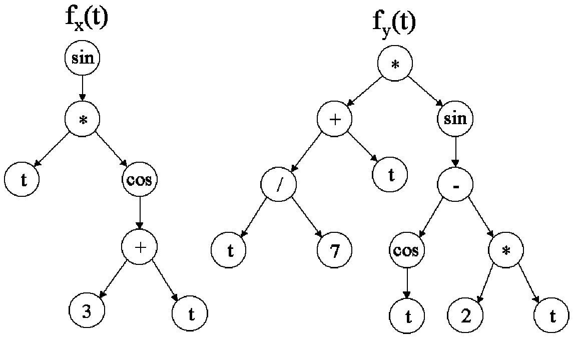
In order to extract the particle trajectories, the programm "WINAnalyze" developed by Mikromak will be used. This program generates ASCII trace files that allow to analyze the trajectories of the particles in the GP-kernel. The positions of the particles are interpreted as values of a parametric function which is generated using symbolic regression and genetic programming.

2 MAIN ALGORITHM

The evolution starts with a randomly generated set of functions (initial population). The genetic operations are applied to this first generation of functions and a succeeding generation is produced. The fitness of an individual is measured directly by evaluating the evolved formulae. Technically the formulae are compiled as a C-program and linked dynamically to the main GP-program, so evaluation can take place in a fast and generic way. It is also possible to utilize an interpreter, what may be more flexible in some cases (e.g. for debugging tasks), but slows down evaluation. The basic structure of the program follows the classic GP-scheme [Koz92]. The data structure and the genetic operators will be described next.

2.1 DATA STRUCTURES

In analogy to the infix notation of mathematical functions a tree based representation forms the genotype. A sample individual is shown in Figure 3.



$$f_x(t) = sin(t * cos(3 + t))$$

$$f_y(t) = t/7 + t * sin(cos(t) - 2 * t)$$

Figure 3: Sample structure of a single individual

The primary goal of getting the mathematical representations of two-dimensional curves leads to the phenotypic representation of an individual as graph of a parametric function.

This difference between genotypic and phenotypic representation and the need for genetic operators in which small changes in the genotype shall result in small changes in the phenotype yields two problems. First a set of genetic operators which allows to do these small changes on the genotype must be used. Second a fitness function which correctly evaluates the fitness of one individual has to be defined.

2.2 FITNESS FUNCTION

The naive way of applying a fitness function is to compare the function values of the individual with corresponding points of the trajectory. If all points of the trajectory lie on the function plot this individual will represent a perfect solution to the problem. The fitness values will be represented by a weighted point to value distance scheme. See Figure 4. In order to improve the efficiency of the naive approach, some variations have been realized.

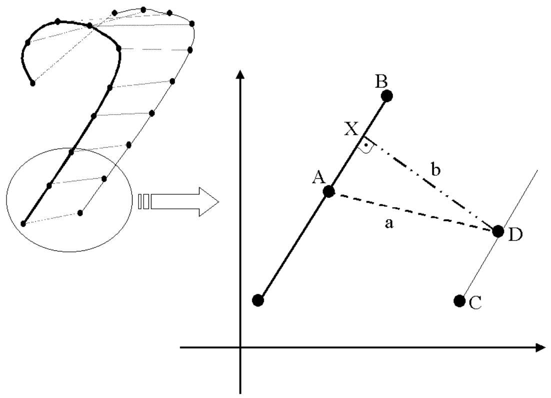


Figure 4: The Point to Value Distance Scheme

The evaluation of the distance of the sampling points to the evaluated function has been simplified to point-to-point distances instead of the mathematically correct orthogonal distances. This simplification is sufficient if the density of the sampling points is high enough.

Due to the fact that many solutions of the GP-algorithm show shapes that are similar to the correct solution but differ only in orientation and size, a deterministic step is inserted between mutation and the fitness calculation of the individual.

The translation vector $\vec{a}$ is determined by setting the starting point of the individual at the position of the first point of the trajectory. See Figure 5. The translation is performed by adding $\vec{a}$ to each point of the genetically generated function.

In a second step the individual will be scaled to the size of the trajectory curve. The scaling values are determined by setting the maxima of the individual to the same value as the maxima of the trajectory points. See Figure 6.

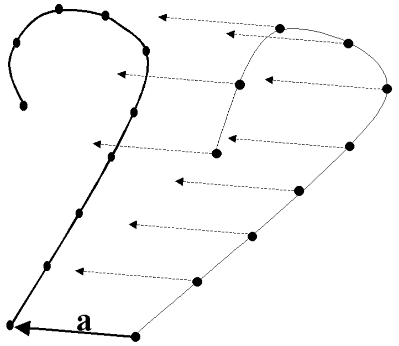


Figure 5: Translation

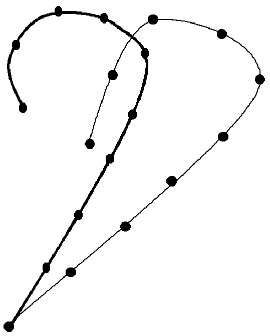


Figure 6: Rotation

2.3 GENETIC OPERATORS

The basic algorithmic structure of the GP algorithm follows the description of Koza [Koz92]. Due to the fact that the algorithm has to evolve parametric functions, two symbolic representations have to be generated in parallel.

Figure 7 illustrates the variation scheme of the algorithm consisting of recombination (crossover) and mutation. A tournament selection operator defines two groups (dark gray and light gray) of individuals that compete with each other. A letter is assigned to each individual. Every survivor of a tournament replaces the inferior individual. A survivor of a tournament is the individual which has a higher fitness value compared to its competitor. In Figure 7 **K** and **G** are the winners that replace **B** and **E**. The survivors undergo recombination realized by crossover. Each resulting individual **K´** and **G´** is varied by mutation yielding **K´´** and **G´´**.

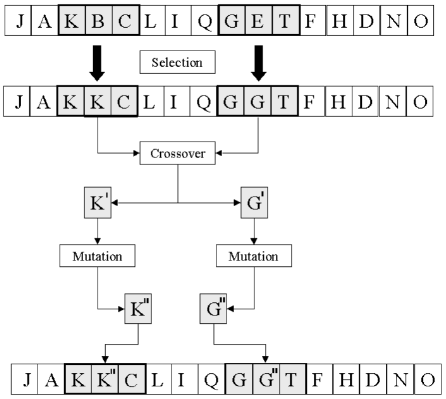

J A K B C L I Q G E T F H D N O
Selection
J A K K C L I Q G G T F H D N O
Crossover
K'
G'
Mutation
Mutation
K''
G''
J A K K'' C L I Q G G'' T F H D N O

Figure 7: Sample of the Variation Scheme

The probability to choose a constant during mutation, the maximum size of an individual or the probability to change a function or a terminal and the number of changes is defined in a parameter file.

3 RESULTS

3.1 TEST FUNCTION

In the first example the following simple function had to be reconstructed:

$$\vec{f}(t) = \begin{pmatrix} f_x(t) \\ f_y(t) \end{pmatrix} = \begin{pmatrix} \frac{cos(t)}{t+t^2} \\ \frac{sin(t)}{6t} \end{pmatrix}$$

In this case a known function was reconstructed in order to illustrate the power of the symbolic regression algorithm. The shape of the graph of $\vec{f}(t)$ resembles a chip. For simulation purpose this function was sampled yielding discrete points which represent points which could have been sampled from a film sequence. The size of the population was 200 and 400 generations were evolved. After this test run the evolved formula was,

$$f_x(t) = \frac{cos(t)}{t * (t + 1.02618) * cos(\frac{1.272908745}{(t+1.21945)*(t+3.81393)})}$$

$$f_y(t) = \frac{sin(t)}{t}$$

After rescaling with the factor $1\frac{2}{3}$ the target function has been approximated with a negligible error. See Figure 8.

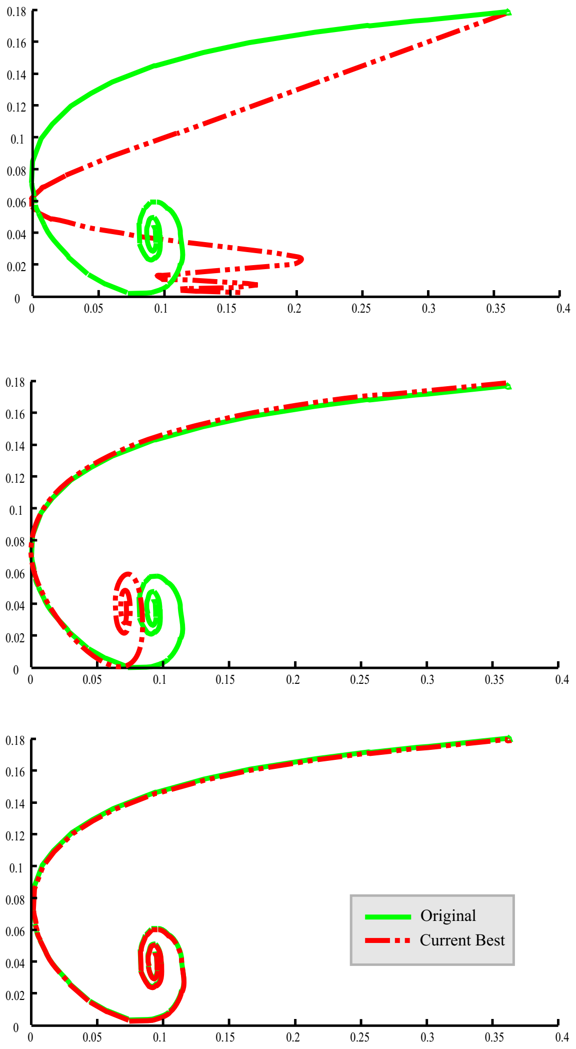


Figure 8: Approximation of the Test Function, after 200, 300 and 400 Generations

3.2 REAL_WORLD DATA

Next the algorithm is used on the data extracted from the film data of the turning process. The size of the filmed window is 0.44 mm by 0.33 mm, so the trajectory which is to be reconstructed stretches from 0.01 mm to 0.08 mm and from 0.06 mm to 0.16 mm.

Terminal Set	t , $K \in \{1.0; 5.0\}$
Func. Set	+ - * / sin() cos()
Selection	tournament
# individuals	250
# programs	2 programs per individual

Table 1: Parameter Table for real world test data

Table 1 shows the parameters for this test run. The result after 500 generations is shown in Figure 9.

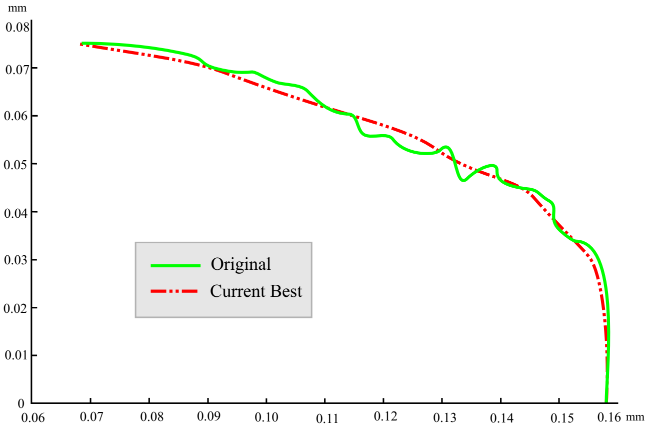


Figure 9: Approximation of real-world data after 500 Generations

The figure illustrates the tendency of the algorithm to equalize the trajectories. The approximations can be improved locally by increasing the number of sampling points. This way the reconstructions are forced to follow also small bendings of the particle flow.

4 CONCLUSION

This paper presents a method to reconstruct and analyze unknown mathematical correlations. In this case one particle trajectory of material movement in the turning process has been reconstructed.

This reconstruction uses symbolic regression and genetic programming. First tests show the general applicability of this approach. Complex two dimensional functions can be reconstructed by assembling terminal functions.

In order to develop a physical model of the turning process, there are additional considerations to be done. For example adding parameters, like the cutting speed or the tool angle to the reconstruction process. Furthermore, restrictions had to be developed to constrain the length of the reconstructed individuals.

The proposed tracking of particles by GP is the first step towards an overall system, which will include another CI-method, namely a CA (Cellular Automaton). This automaton will be provided with the rules evolved by the GP part of the system. The authors intend to use the modeling properties of CAs (simple local interactions can model complex global behavior) together with the GP-evolution of formulae to develop a complete model of the geometric properties of the chip-building process in orthogonal cutting which can be a basis for systems as described in the introduction section.

Acknowledgments

This research was supported by the Deutsche Forschungsgemeinschaft as part of the Collaborative Research Center “Computational Intelligence” (SFB 531).

References

[Ina97] T . Inamura. Brittle/ductile phenomena observed in computer simulations of machining defect-free monocrystalline silicon. *Annals of the CIRP*, 46:31–34, 1997.

[Koz92] J. R Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, 1992.

[SB93] H. Schulz and K. Bimschas. Optimisation of precision machining by simulation of the cutting process. *Annals of the CIRP*, 42:55–58, 1993.

[SITU94] S. Shimada, N. Ikawa, H. Tanaka, and J. Uchikoshi. Structure of micromachined surface simulated by molecular dynamics analysis. *Annals of the CIRP*, 43:51–54, 1994.

[War74] G. Warnecke. *Spanbildung bei metallischen Werkstoffen.* Technischer Verlag Resch, Munich, 1974.

[XBZ98] J. Q. Xie, A. E. Bayoumi, and H. M. Zbib. Fea modeling and simulation of shear localized chip formation in metal cutting. *Intl. J. of Machine Tools and Manufacture.*, 38:1057–1087, 1998.