# Using Linear Genetic Programming to Develop a C/C++ Simulation Model of a Waste Incinerator.

**Larry M Deschaine PE**
**Science Applications International Corporation**
**Augusta, GA**
**(706) 724-5589 x227**
Larry.M.Deschaine@alum.mit.edu

**(Released for Distribution – Complimentary Advance Copy)**

**Abstract.**

We explore whether Genetic Programming (GP) can evolve a C/C++ computer simulation model that models the performance of a hazardous waste incinerator accurately. Human expert written simulation models are used worldwide in a variety of industrial and business applications. They are expensive to develop, may or may not be valid for the specific process that is being modeled, and may contain bugs.

Genetic Programming is a machine learning technique that uses information about a process's inputs and outputs to simultaneously write the simulation model, calibrate and optimize the model's constants, and verify the solution. The end result is a calibrated, validated, bug-free C/C++ computer model specific to the desired process.

To evaluate whether this is feasible for complex industrial processes, we test the approach on data obtained from the operation of a hazardous waste incinerator. This process is a difficult problem to model. Previously, in a well-conducted study, the popular machine learning technique, analytic neural networks, was unable to derive useful solutions to this problem. The present study used various mutation rates (95%, 50%, and 10%), ten random initial seeds per mutation rate, and a large number (1,280 to 4,461) of generations. The GP system evolved excellent solutions to this problem -- the best validation data measure of fitness, $R^2$, was 0.961.

This work demonstrates the value of Genetic Programming for process simulation. This study confirms previously published work, which found that the distribution of outputs from multiple GP runs tends to include an extended "tail" of outstanding solutions. Such a tail was not found in previous studies of neural networks. This finding emphasizes the need for employing a strategy of multiple runs using various initial seeds and mutation rates to find good solutions with GP to difficult problems. It also demonstrates the value of a fast Genetic Programming algorithm implemented at the machine code level for both static scientific data mining and real-time process control. The work consumed 600 hrs of CPU time; other GP algorithms would have required on the order of between four and 136 years of CPU time to achieve these same results.

## 1. Introduction

With the increasing complexity of modern manufacturing [Popovic98] and processes [Popovic90], industries demand fast techniques for adaptive real-time control [Francone00a]. Today, many industries allocate in excess of 10% of their plant investment capital outlays for instrumentation and control [Murrill00]. This percentage has doubled over the past thirty years and shows no signs of diminishing. The industrial processes are often non-linear and the mathematical representation of the process is not known [Sinha00]. Hence simulation models are not available for many of the processes that exist today. Without simulation models, optimal process control is very difficult to achieve. This results in unnecessarily wasting resources.

Genetic Programming is a promising machine learning technology that has been the subject of intensive academic research since 1988. Since 1998, commercial applications have arrived on the market [Francone00b]. GP can be used to automatically develop a simulation model of complex industrial processes. In this work, we specifically examine whether Genetic Programming (GP) can evolve a C/C++ computer simulation model that mimics the performance of the concentrations of carbon dioxide in a hazardous waste incinerator. Waste incineration was chosen as a test case as it is a very complex process. It

involves variable input material properties (solids, liquids and gaseous), high temperature, large temperature swings, variable-input energy sources, compressible gas flow, density effects and the like. The simulated variable, the concentration of carbon dioxide in the secondary combustion chamber, varies widely from zero to 5000 ppm. This process has been demonstrated resistant to solution by machine learning via the analytical neural network technique (ANN), as evidenced by a well-conducted study [Fausett, 00].

Despite the complexity of incineration processes, developing a computer simulation code that maps the process variables to the carbon dioxide emission concentration should at least be feasible using Genetic Programming. The strength of Genetic Programming is its ability to abstract an underlying principle from a finite set of fitness cases [Banzhaf98, Koza99]. This principle can be considered the essence of the regularities that determine the appearance of concrete fitness cases. Genetic Programming evolves both the structure and the constants of the solution simultaneously, the goal being to extract these regularities in the form of an algorithm or computer program, in this example a simulation model.

## 2. The Genetic Programming Algorithm

### 2.1 The Goal

The task given to the Genetic Programming algorithm is to develop a C/C++ computer simulation program that accurately maps the important waste incineration process variables with the concentration of carbon dioxide in the waste incineration process. The goal of GP is to evolve that one super solution that solves the problem excellently, as opposed evolving many average or good solutions. Identifying the important inputs will reduce the cost of the real-time process monitoring and control system. A computer program that predicts the concentration of air contaminants can reduce the cost of monitoring and provide a tool to simulate emissions prior to incinerator loading, thereby improving compliance with the Clean Air Act. Optimal strategy for incinerating waste, as defined by maximizing incineration rate (i. e. profit) subject to no permit violations can then be developed.

### 2.2 The Genetic Programming Algorithm

For this work, we used a GP algorithm that specifically evolves a computer program at the machine code level [Nordin, 97] on a Von Neumann machine, a machine where the data and program resides in the same storage location. It is called Automated Induction of Machine Code by Genetic Programming (AIMGP) system. This system was formally known as the Compiling Genetic Programming System or CGPS. The algorithm, since it manipulates the machine code directly, avoids compilation overhead and has been estimated to be over 60 times faster when compared to an interpreting C-language implementation and up to 1500 to 2000 times faster when compared to a LISP implementation. While we could have used any of the number of available GP algorithms that have been developed [Banzhaf98], the speed of AIMGP means that programs that would take months or years to evolve on a single processor can be evolved less than a day. For example, this research consumed 600 hrs of CPU time. A GP approach that relied on C-language interpreting of solutions for evaluation would have consumed about four years of CPU time, LISP on the order of 100 to 136 years. Since the ANN had difficulty with this challenge – and this was the reason we chose this problem to solve– we expected that we would need a lot of computing power. The algorithm was recently re-written for the CISC processor and extended from its original form [Nordin99, Francone, 00b].

### 2.3 Machine Learning Facilitating Human Learning

Genetic Programming is a machine learning technique that writes computer programs. During a GP run, intron explosion occurs. An intron is an instruction that occurs in a program that has little to no consequence on the output. Introns are believed to occur to protect a good program from the destructive effects of crossover [Banzhaf98]. To the human observer, these introns make the code confusing, if not intelligible. Specific algorithms have been developed that transform the raw machine learned program to a human understandable program. These algorithms, which clean, simplify, and optimize the code, facilitate the process of transferring the knowledge derived during machine learning to the human expert, facilitating acceptance of the solution as well as increasing the knowledge of the process.

### 2.3.1 Parsimony Pressure

Parsimony pressure is a term used to refer to techniques that tend to make the evolved program shorter. Parsimony pressure causes "natural selection in evolutionary learning systems to favor the selection of shorter and more compact programs. It is a penalty function on program length. The function calculates the percentage difference between two programs evolved using the tournament selection algorithm. If the shorter program meets acceptance criteria when compared to competing individuals, than the shorter program is selected as the winner. Shorter programs are often easier to interpret. Parsimony was not deemed necessary in these simulations, as discussed in the result section.

### 2.3.2 Intron Removal

Introns are removed by substituting the no operation instruction into the program and comparing the two output values. If the output value does not change, than the instruction is deemed to be an intron and is removed.

### 2.3.3 Custom Fitness Functions

A custom fitness function can be incorporated that allows the researcher to specify precisely how the results of the GP evolutions are scored, in addition to the standard linear and least squares error approach.

### 2.3.4 Interactive Evaluation of Evolved Program

Interactive evaluation of the evolved program allows the researcher simplify, modify and optimize the program as well as explore the results of the modification (s) on the program's fitness. This algorithm also allows the researcher to perform what-if scenarios on the code, including importing solutions from known sources or other GP runs.

## 3. The Test Case

We found that GP generated excellent solutions to the incinerator problem. Heretofore, this incineration prediction challenge had no known solution; hence a benchmark with which to compare either the results or the resulting program does not exist. We rely on the results of the program fit to the validation data set for verification for success determination.

To test the GP technique, the performance of a waste incineration plant was modeled using real-time typical operational data collected hourly over a one-week period at the Consolidated Incinerator Facility (CIF) at the Department of Energy Savannah River Site (DOE-SRS). The CIF processes a variety of solid and aqueous waste, using a combination of a rotary kiln, a secondary combustion chamber, and an off gas scrubber system. The process was chosen, as it is a very complex system to simulate as it consists of variable fuel and waste inputs high temperatures of combustion and high velocity off gas emissions. Variables that describe the CIF process were collected as follows:

- **Process Parameters**

  Rotary Kiln Incinerator (lb./hr): Fuel Oil (flow), Liquid Waste (flow), Solids (airflow), Solids (flow), Solids (average flow), Fuel Oil (airflow), Liquid Waste (airflow), and Aqueous Waste (flow). Temperature(C). Time (hrs.)

  Secondary Combustion Chamber (lb./hr): Fuel Oil (flow), Fuel Oil (airflow), and Steam (flow), as well as Temperature (C) and $O_2$ (percentage).

  Offgas: 4 measurements of $CO_2$ (ppm), 2 measurements of $O_2$ (percentage), and one measurement of duct flow (scfm).

- **Output Parameter**

  Secondary Combustion Chamber: $CO_2$ (concentration as ppm)

If the GP is to be successful, it must develop the relationship that maps the variables of the incineration process to the carbon dioxide concentration in the secondary combustion chamber. This parameter was chosen as the ANN had great difficulty developing any useful model for this mapping [Fausett00]. We used a zero and one hour offset for the data when constructing the training and validation instance sets. This resulted in a total of 44 input variables. We ran the thirty GP simulations for a period of 20 hrs each using ten different random seeds for each of three mutation rates. We then compare the results of the GP simulations for solution convergence.

### 3.1 The GP System Parameters

A description of the parameters and values that were used in this run, as well as the rationale for their selection, are presented below.

#### 3.1.1 Genetic Programming

**Population Size.** A population in GP is the number of programs that the system will evolve. Generally speaking, the larger the population the more difficult a problem that will be solved and the longer the run will take. The size of the population is limited by the random access memory (RAM) of the computer. A population size of 25,000 was used, and the AIMGP algorithm consumed a total of 95,692 Kbytes of RAM.

**Maximum Number of Tournaments.** A GP tournament is the process in which evolved programs are produced. The tournament algorithm is constructed as follows:

1. Initialize a Population of Programs. Create a population of randomly generated programs.

2. Tournament Contest. Randomly select four programs from the population. Evaluate them for how well they map the input data to the output data. This step is known as the program "fitness" evaluation. Two programs are selected as winners, and the other two are tagged as losers.

3. Transform the "Winner" Programs. The two "winner" programs are then copied and transformed probabilistically by:

   - Exchanging parts of the "winner" programs with each other to create two new programs (crossover); and/or

   - Randomly changing each of the tournament winners to create two new programs (mutation).

4. Replace the "Loser" Programs. Replace the "loser" programs in the population with the transformed "winner" programs. The winners of the tournament remain in the population unchanged.

5. Iterate Until Convergence. Repeat steps two through four until a program is developed that predicts the behavior sufficiently.

Time and its C variable declaration only limit the number of tournaments. We used an arbitrarily large maximum number of tournaments to be 2147483647, as the stopping criterion was 20 hrs.

**Search Parameters.**

- **Mutation Rate.** Mutation is one of the principal search operators used to transform programs in the GP algorithm. Mutation has the effect of causing random changes to occur in tournament winners. The mutation is applied probabilistically to all programs that have won tournaments, regardless of whether a winner has been selected for crossover. While many GP systems use either minor or no mutation operators, increasing the mutation rate has been shown to significantly improve the generalization capabilities of GP [Banzhaf96]. The mutation rate can range from 0% - no mutation to 100%, we selected 95%, 50%, and 10% in concert with, but slightly higher than, the referenced research which used 80% as the upper limit.

- **Crossover Rate.** The crossover rate is another key search parameter in GP. The crossover operates by exchanging sequences of instructions between two tournament winners. This results in two programs being inserted into the population in place of the two losers in that tournament. The crossover rate can vary from 0% to 100%. We used a constant crossover rate of 50% for all thirty simulations.

- **Reproduction Rate.** Reproduction rate is another search operator in GP. Reproduction copies a program and places the copy in the population in addition to the original program. It is a function of the crossover and mutation rates as follows: 100-mutation- (crossover*(1-mutation)).

**Demes.**

- **Number of Demes.** The number of demes pertains to the how the population of programs is divided. A deme is a subset of the program population to essentially isolate groups of populations from each other. This paradigm mimics biologists belief that genetic diversity is enhanced when populations are separated from each other geographically. We chose a value of 10 demes for this run with 2500 programs per deme.

- **Crossover Percentage Between Demes.** As discussed above, cross over occurs between programs in the same population. By dividing the population into demes, crossover now occurs both within each deme as well as between demes. The percentage of crossover between demes sets the percent of tournaments that will result in crossover between programs in adjacent demes. The algorithm works as follows:

    1. Select a deme at random
    2. Select one of the two adjacent demes at random
    3. Select two programs from each of the selected demes, the better of which is chosen for crossover.
    4. Crossover the selected program from each deme. The offspring of the crossover replace the two tournament losers.

    We used a deme crossover percentage value of 10%; it can range from 0% to 100%.

- **Inter-Deme Migration Rate.** This controls the rate at which the percent of tournaments that result in migration of programs between adjacent demes. The algorithm works as follows:

    1. Randomly select a deme
    2. Randomly select one of the two adjacent demes
    3. Randomly select one program from each deme.
    4. Evaluate the fitness of each program, and replace the worse program with the better one from the other deme.

- **Dynamic Subset Selection.** Dynamic subset selection uses a subset of the training set to help evolve solutions that are more generalized. It works by periodically changing which subset of the training set is used for training purposes, and hence helps avoid over training or memorization.

- **Target Subset Size.** This controls the size of subset of the training set that will be used. The total data set consisted of 166 instances, 151 training instances and 15 validation instances chosen as every $10^{th}$ point. We used a value of 120 training instances out of the total training set of 151 for the target subset size.

- **Selection by Age.** The algorithm keeps track of the usage of each individual training instance. The training set is chosen in proportion of the time since the training instance was last used. The least recent training instances are preferentially chosen during the next training set assembly. We chose a weight selection of 50% for this parameter.

- **Selection by Difficulty.** The algorithm also keeps track of how difficult the population is at finding a particular training instance. By setting this parameter, more general solutions are found to the tougher portions of the training set. We chose a selection weight of 50% for this parameter.

- **Stochastic Selection.** This parameter is used to select training instances randomly. We set a selection weight for this criterion to 0%, which means we did not use stochastic selection in our runs.

- **Training Subset Change Frequency Equivalents.** This parameter determines how frequently the training subset is changed, in generation equivalents. Since the GP uses the tournament selection criteria, a generation equivalent is one half of the population size, in our case 12,500 tournaments. We used a value of changing the training subset each (1) generation equivalent for our runs.

## 4. Results

The GP algorithm produced a very good solution to the incineration simulation challenge. Table 1.0 provides a summary of the results. The table shows the summary of the information from the best-validated program from each simulation.

Table 1.0 Comparison of $R^2$ for Various Mutation Rates and Random Seeds.

| **Fitness Case ($R^2$)** | **Mutation Rate = 95%** | **Mutation Rate = 50%** | **Mutation Rate = 10%** |
|---|---|---|---|
| | | | |
| **Validation Data Set** | | | |
| Best Validated Fitness | 0.961 | 0.785 | 0.852 |
| Avg. Validated Fitness | 0.720 | 0.477 | 0.552 |
| Worst Validated Fitness | 0.477 | 0.088 | 0.168 |
| | | | |
| **Complete Data Set** (Validation and Training) | | | |
| Best Fitness | 0.979 | 0.903 | 0.912 |
| Avg. Fitness | 0.709 | 0.569 | 0.531 |
| Worst Fitness | 0.483 | -0.252 | -0.117 |

The stopping criterion for all simulations was 20 hrs. The average number of generations that were evolved was 3037; this ranged from a low of 1,280 generations to a maximum of 4,461 generations.

The fitness of the solution was calculated by comparing the measured and predicted values using the linear trend-line option that is in Microsoft Excel, with the y-intercept set to zero. The best-validated data results - $R^2$ of 0.961 - were obtained using a high mutation rate of 95%. In fact the top two solutions were obtained at the 95% mutation rate. The mutation rate of 50% and 10% failed to produce an $R^2$ validated fitness above 0.900. This is thought to have occurred due to the high mutation causes a more aggressive search of the solution space as opposed to lower mutation rates. The best solution is presented on Figure 1.0. The distribution of validated solution fitness is shown on Figure 2.0.

The GP algorithm was run thirty times for a total of 600hrs 09min 30sec of CPU time. The computer consisted of a dual PentiumIII © 533Mhz PC with a 133 FSB, 256MB of ECC 100 RAM, the Intel © 840 chipset on a Supermicro PIIIDME motherboard using the Microsoft Windows 2000 Professional operating system. The GP algorithm consumed about 95,692KB of memory. The total memory requirement, including the operating system was about 204,784KB when run as a single application.

## 5. Conclusions and Suggestions for Further Research

Earlier research suggests that, for three machine-learning problems, multiple runs on linear GP systems tend to produce a distribution of outputs with a long tail of outstanding quality solutions [Francone96]. Interestingly, this long tail of outstanding solutions exists even though the average solution generated may not be very good. By way of contrast, that same study found that neural networks produce a large number of runs that are normally distributed around a mean of average quality solutions; but that multiple neural network runs do not produce the tail of outstanding solutions characteristic of linear GP.

The present study is consistent with the earlier results in [Francone96]. Like the previous study, neural network technology was unable to generate outstanding solutions to the problem at hand. And, like the previous study, multiple linear GP runs produced a tail of outstanding solutions, which included solutions considerably better than any solution produced by neural network technology. To wit, linear Genetic Programming evolved a validated program that explains 96.1% of the relationship between the process inputs and output variable for a machine learning unfriendly incineration process.

In real world applications, the goal of machine learning is normally to produce that one super-solution that solves the problem -- as opposed to generating many average solutions. Two studies (including this present study) now suggest that a linear GP approach meets this criterion better than do neural networks. GP performed extremely well on this very difficult process control problem. However, the GP technique did not perform superbly on all of the runs. This confirms how tough this problem is.

The results of this research also demonstrate the importance of solving problems by performing multiple GP runs at different mutation rates and initial seeds. This process facilitates finding the super-solution because of the unique distribution of the results of multiple, linear GP runs discussed above. But multiple runs are very time consuming because machine learning in general (and GP in particular) is very computationally intensive. Ordinary GP or neural network software would not have been capable of performing the runs in this study in remotely realistic time frames.

Accordingly, these conclusions underline the importance of AIMGP's great speed in solving process control problems. Process control frequently involves difficult learning domains that benefit from multiple runs. Thus, AIMGP's speed may well open other, historically-intractable dynamic control of complex process problems to widespread solution by machine learning techniques.

This success demonstrates the promise of using linear GP algorithms to simulate the waste incineration process. Further work includes testing this technique on other complex manufacturing processes. To date, success has been demonstrated on manufacturing of solid materials [Deschaine00a] and liquid [Deschaine00b] processes.

With the simulation model developed, what-if scenario simulations of a planned waste incineration activity can help keep emissions to within acceptable limits. The C/C++ process description also allows process optimization to be explored via linkage with cost information and other management resource planing and optimization tools. Reducing the number of monitoring locations for data collection by focussing on the evolved solution inputs has obvious benefit with respect to capital expenditures, reduced operations and maintenance of the control system, and automated reporting.

## 6. Acknowledgements

The author greatly acknowledges Dr. Wolfgang Banzhaf, Dr. Peter Nordin, Dr. Clinton W. Kelly III, Mr. Frank D. Francone, Dr. Laurene Fausett and Mr. Edward Dilkes for their comments and suggestions on this work. Register Machine Learning Technologies is thanked for providing the AIMGP system used in this work which transformed a multi-year research project into one that was completed within one month. Mr. Joseph W. Craver III, Mr. Janardan J. Patel and Mr. Frederic A. Zafran of Science Applications International Corporation are thanked for the latitude and freedom to explore this new and valuable technology.

**Figure 1. Predicted vs. Actual $CO_2$ SCC Emissions**

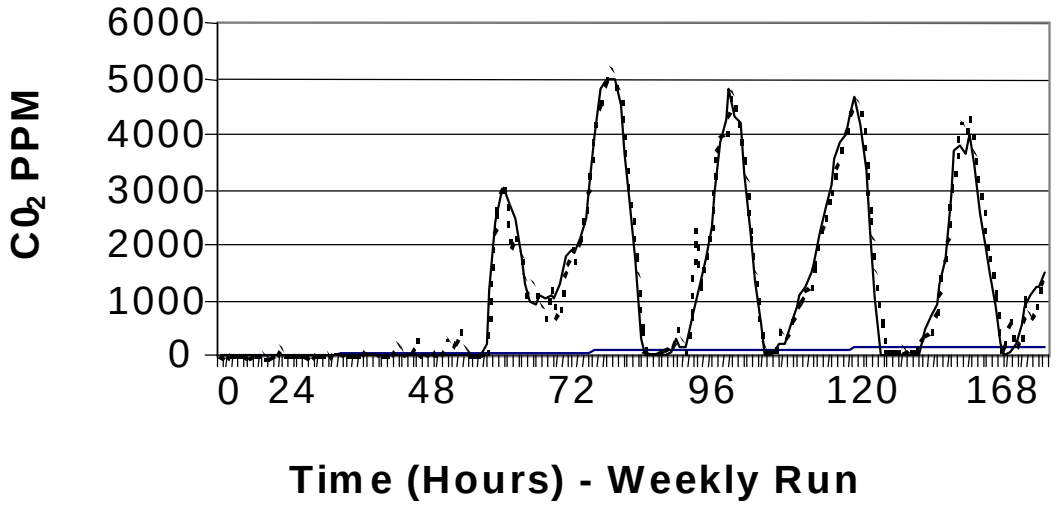
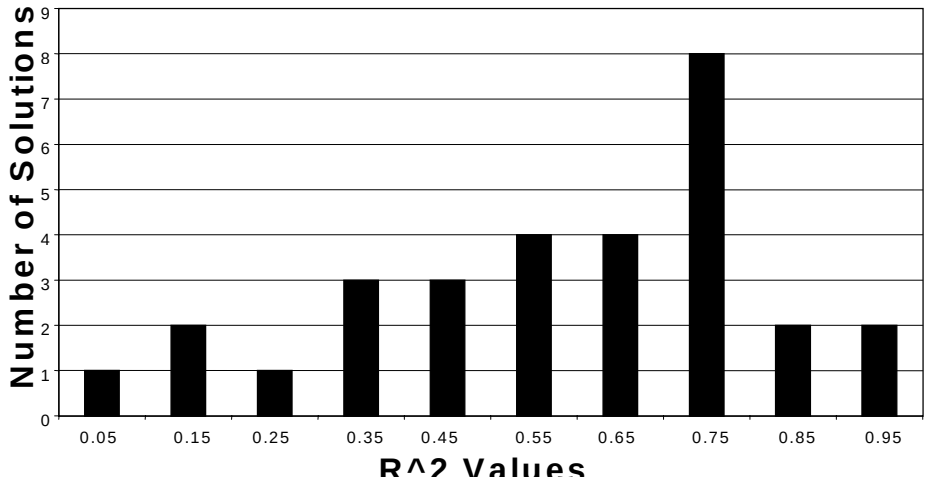



**Figure 2.0 Distribution of GP Validation R^2**



## 7. References


[Banzhaf98] Banzhaf, Nordin, Keller, Francone, *Genetic Programming – An Introduction. On the Automatic Evolution of Computer Programs and its Applications.* 1998 Morgan Kaufmann Publishers, Inc. 470 pp.

[Banzhaf96] Banzhaf, W., Francone, F., and Nordin, P. (1996). *The Effect of Extensive use of the Mutation Operator on Generalization in Genetic Programming Using Sparse Data Sets.* In Voigt, H., Ebling, W., Rechenberg, I., and Schwefel, H., P., editors, Parallel Problem Solving From Nature IV. Proceedings of the International Conference on Evolutionary Computation, Vol. 1141 of Lecture Notes in Computer Science, pages 300-310, Berlin. Springer-Verlag, Berlin.

[Deschaine00a] Deschaine, L. M., Zafran, F. A., Patel, J. J., Amick, D., Pettit, R., SAIC, Francone, F. D., Nordin, P., Dilkes, E., and Fausett, L. V., *Solving the Unsolved-Using Machine Learning to Model a Complex Production Process – Case Example Applying Three Machine Learning Techniques,* Society for Computer Simulation's Advanced Simulation Technology Conference, Washington, DC, USA April 2000.

[Deschaine00b] Deschaine, L. M., *An Evaluation of the Feasibility of Genetic Programming to Solve a Waste Waster Treatment Process*, unpublished work in progress.

[Fausett00] Fausett, L. V., *A Neural Network Approach to Modeling a Waste Incinerator Facility*, Society for Computer Simulation's Advanced Simulation Technology Conference, Washington, DC, USA April 2000.

[Francone96] Francone, Nordin, J.P. and Banzhaf W. (1996) Benchmarking The Generalization Capabilities of a Compiling Genetic Programming System Using Sparse Data Sets. In: *Proceedings of The First International Conference on Genetic Programming*, Stanford, USA

[Francone00a] Francone, F. D., Nordin, P., Banzhaf, W. Dilkes, E., Deschaine, L. M. *AIM Learning™ Adaptive, Real-Time, Control Technologies*. Society for Computer Simulation's Advanced Simulation Technology Conference, Washington, DC, USA April 2000.

[Francone00b] Francone, F. D. et. al., *Discipulus Owners Manual*, Register Machine Learning Technologies, Inc. Version 2.0 2000.

[Koza99] Koza, Bennett, Andre, Keane, *GENETIC PROGRAMMING III – Darwinian Invention and Problem Solving*, 1999 Morgan Kaufmann Publishers, Inc. 1154 pp.

[Murrill00] Murrill, Paul W., *Fundamentals of Process Control Theory, 3rd Edition*, ISA: the International Society for Measurement and Control, North Carolina, USA, 333 pages.

[Nordin97] Nordin, Peter, *Evolutionary Program Induction of Binary Machine Code and its Applications*, Krehl Verlag, Munster, 1997, 290 pp.

[Nordin99] Nordin, P., Banzhaf, W., Francone, F.D., *Efficient Evolution of Machine Code for CISC Architectures Using Instruction Blocks and Homologous Crossover, Advances in Genetic Programming Vol. III*, The MIT Press, 1999 Chap. 12.

[Popovic98] D. Popovic and L. Vlacic, *Mechatronics in Engineering Design and Product Development*, Marcel Dekker, New York, 1998.

[Popovic90] D. Popovic and V.P. Bhatkar, *Distributed Computer Control for Industrial Control*, Marcel Dekker, New York, 1990.

[Sinha00] Sinha, N.K., and Gupta, M.M., *Soft Computing & Intelligent Systems, Theory and Applications*, Academic Press, UK, Chapter 18 (D. Popovic), 2000.