White Paper

Comparison of Discipulus™ Linear Genetic Programming Software with Support Vector Machines, Classification Trees, Neural Networks and Human Experts

Larry M. Deschaine[1]

Frank D. Francone[2]

Executive Summary

Discipulus™ is multiple-run, linear, genetic-programming software. Various versions have been available commercially since 1998 (*see*, www.aimlearning.com). Discipulus creates models directly from data, like neural networks or support vector machines.

This white paper reports on the result of a multi-year study of the performance of Discipulus by Science Applications International Corp (SAIC) and RML Tech-

[1] Mr. Deschaine is an Engineering Physicist at Science Applications International Corp. He has published about 80 works in the open literature on three continents in multiple languages (including journals, book chapters and conference proceedings). He invented the UXO/MineFinder in 2001 for SAIC. He recently received a "job well done" medal from Fort Knox for the work he presented on UXO and plume finding, optimal remedial design, provable TI determination and optimal long term monitoring.

He has eighteen national and international awards for his work. These include back-to-back world records in underground construction (horizontal bioairsparge wells) that achieved in excess of a 40% cost savings ($1.84M) in remediation. He received a citation from the Vice-President of the United States for leading the development of an algorithm which saved $90M over a five year period.

[2] Frank Francone has been President of RML Technologies, Inc. since 1996. As such, he designed and marketed the world's first, and still best selling, Genetic Programming software, Discipulus™. He also leads RML's consulting business in data analysis and solutions. That practice concentrates on business applications of evolutionary computation, genetic programming, optimization, risk analysis, and statistics.

In addition to his business interests, as one of the authors of the leading university textbook on Genetic Programming, he is a well known research scientist. His many articles on Genetic Programming, Machine Learning, data analysis, and UXO cleanup have appeared in scientific journals, trade magazines, and conference proceedings since 1995.

nologies, Inc. This study compared Discipulus to several other powerful modeling tools on a wide variety of industrial problems including regression and classification problems, CRM problems, time series problems, complex signal discrimination problems and others.

We compared the modeling capability of Discipulus to the following competitive modeling technologies:

- Vapnick Statistical Learning,
- Neural Networks,
- Decision Trees, and
- Rule-Based Systems.

In brief summary, the other modeling tools performed inconsistently—sometimes they produced very good results and sometimes mediocre or even very poor results. None of these tools produced high quality results across the board. ***In contrast, Discipulus (at its default settings) always produced results that were the same as or better than the best results from other modeling techniques.***

The results described in this white paper have all been previously published in peer-reviewed scientific publications.

Introduction

Discipulus™ is multiple-run, linear, genetic-programming software. Various versions have been available commercially since 1998 (see, www.aimlearning.com). Discipulus creates models directly from data, like neural networks, decision-trees, or support vector machines. Genetic Programming software produces models in the form of computer programs. Discipulus, for example, creates models in C, Java and Intel inline assembler.

This white paper reports on the result of a multi-year study of the performance of Discipulus by Science Applications International Corp (SAIC) and RML Technologies, Inc.

This study compared Discipulus with several other powerful modeling tools on a wide variety of industrial problems including regression and classification problems, CRM problems, time series problems, complex signal discrimination problems and others. We compared the quality of models produced by Discipulus with the following competitive modeling tools:

- Support Vector Machines/Vapnick Statistical Learning,
- Neural Networks,
- Decision Trees, and
- Human Experts.

This white paper is organized as follows:

First: we briefly describe machine-code-based, linear genetic programming (LGP) in detail and the Discipulus software, in particular; and

Second: we detail the results of a three-year study of the performance of Discipulus as compared to the performance of these other powerful modeling tools. The study occurred in two phases:

- Phase I. LGP modeling was performed with Discipulus Lite and Discipulus Standard Versions.
- Phase II. LGP modeling was performed with Discipulus Professional Version.

Finally, we conclude with a summary of our results.

Linear Genetic Programming with Discipulus

Genetic Programming (GP) is the automatic, computerized creation of computer programs to perform a selected task using Darwinian natural selection. GP developers give their computers examples of how they want the computer to perform a

task. GP software then writes a computer program that performs the task described by the examples.

GP is a robust, dynamic, and quickly growing discipline. It has been applied to diverse problems with great success—equaling or exceeding the best human-created solutions to many difficult problems [14,3,4,2].

This paper presents approximately three years of analysis of machine-code-based, LGP. To perform the analyses, we used the Lite, Standard and Professional Versions an off-the-shelf commercial software package called Discipulus™ [22]. Discipulus is a LGP system that operates directly on machine code.

Genetic Programming

Good, detailed treatments of Genetic Programming may be found in [2,14]. In brief summary, the LGP algorithm in Discipulus is surprisingly simple. It starts with a population of randomly generated computer programs. These programs are the "primordial soup" on which computerized evolution operates. Then, GP conducts a "tournament" by selecting four programs from the population—also at random—and measures how well each of the four programs performs the task designated by the GP developer. The two programs that perform the task best "win" the tournament.

The GP algorithm then copies the two winner programs and transforms these copies into two new programs via crossover and mutation transformation operators—in short, the winners have "children." These two new child programs are then inserted into the population of programs, replacing the two loser programs from the tournament. GP repeats these simple steps over and over until it has written a program that performs the selected task.

GP creates its "child" programs by transforming the tournament winning programs. The transformations used are inspired by biology. For example, the GP mutation operator transforms a tournament winner by changing it randomly—the mutation operator might change an addition instruction in a tournament winner to a multiplication instruction. Likewise, the GP crossover operator causes instructions from the two tournament winning programs to be swapped—in essence, an exchange of genetic material between the winners. GP crossover is inspired by the exchange of genetic material that occurs in sexual reproduction in biology.

Configuration Issues in Performing Multiple LGP Runs

Our investigation into exploiting the multiple run capability of machine-code-based LGP had two phases—largely defined by software versioning. We started with Discipulus Lite and Discipulus Standard. These versions of Discipulus permit multiple runs, but only with user-predefined parameter settings.

As a result, our early multiple run efforts (described below as our Phase I investigation) just chose a range of reasonable values for key parameters, estimated an

appropriate termination criterion for the runs, and conducted a series of runs at those selected parameter settings. For example, the chart of the LGP results on the incinerator CO2 data sets (Fig. 2) was the result of doing 30 runs using different settings for the mutation parameter.

By way of contrast, the second phase of our investigation was enabled by four, key capabilities introduced into Discipulus Professional Version and up. Those capabilities were:

- The ability to perform multiple runs with randomized parameter settings from run to run;
- The ability to conduct hillclimbing through LGP parameter space based on the results of previous runs;
- The ability to automatically assemble teams of models during a project that, in general, perform better than individual models; and
- The ability to determine an appropriate termination criterion for runs, for a particular problem domain, by starting a project with short runs and automatically increasing the length of the runs until longer runs stop yielding better results.

Accordingly, the results reported below as part of our Phase II investigation are based on utilizing these additional four capabilities of Discipulus Professional Version.

Discipulus Investigation—Phase I

We tested Discipulus Lite and Standard software on a number of problem domains during this first phase of our investigation. This Phase I investigation covered about two years and is reported in the next three sections.

Deriving Physical Laws

Science Applications International Corporation's (SAIC's) interest in LGP was initially based on its potential ability to model physical relationships. So the first test for LGP to see if it could model the well-known (to environmental engineers, at least) Darcy's Law. Darcy's Law describes the flow of water through porous media. The equation is:

$$Q=K*I*A, \tag{1}$$

where Q = *flow [L3/T*], K = *hydraulic conductivity [L/T*], I = *gradient [L/L*], and A = *area [L2*].

We generated a realistic input set and then used Darcy's law to produce outputs. We then added 10% random variation to the inputs and outputs, and ran the LGP

software on these data. After completing our runs, we examined the best program it produced.

The best solution derived by the LGP software from these data was a four-instruction program that is precisely Darcy's Law, represented in ANSI C as:

```
Q = 0.0
Q += I
Q *= K
Q *= A
```

In this LGP evolved program, Q is an accumulator variable that is also the final output of the evolved program.

This program model of Darcy's Law was derived as follows. First, it was evolved by LGP. The "raw" LGP solution was accurate though somewhat unintelligible. By using intron removal [19] with heuristics and evolutionary strategies to simplify and optimize the evolved solution, the specific form of Darcy's Law was evolved. This process is coded in the LGP software; we used the "Interactive Evaluator" module, which links to the "Intron Removal" and automatic "Simplification" and "Optimization" functions. These functions combine heuristics and ES optimization to derive simpler versions of the programs that LGP evolves [22].

Incinerator Process Simulation

The second LGP test SAIC performed was the prediction of CO2 concentrations in the secondary combustion chamber of an incinerator plant from process measurements from plant operation. The inputs were various process parameters (e.g., fuel oil flow, liquid waste flow, etc.) and the plant control settings. The ability to make this prediction is important because the CO2 concentration strongly affects regulatory compliance.

This problem was chosen because it had been investigated using neural networks. Great difficulty was encountered in deriving any useful neural network models for this problem during a well-conducted study [5].

The incinerator to be modeled processed a variety of solid and aqueous waste, using a combination of a rotary kiln, a secondary combustion chamber, and an off-gas scrubber. The process is complex and consists of variable fuel and waste inputs, high temperatures of combustion, and high velocity off-gas emissions.

To set up the data, a zero and one hour off-set for the data was used to construct the training and validation instance sets. This resulted in a total of 44 input variables. We conducted 30 LGP runs for a period of 20 hours each, using 10 different random seeds for each of three mutation rates (0.10, 0.50, 0.95) [3]. The stopping criterion for all simulations was 20 hours. All 30 runs together took 600 hours to run.

Two of the LGP runs produced excellent results. The best run showed a validation data set R2 fitness of 0.961 and an R2 fitness of 0.979 across the entire data set.

The two important results here were: (1) LGP produced a solution that could not be obtained using Neural Networks; and (2) Only two of the 30 runs produced good solutions (see Fig. 2), so we would expect to have to conduct all 30 runs to solve the problem again.

Data Memorization Test

The third test SAIC performed was to see whether the LGP algorithm was memorizing data, or actually learning relationships.

SAIC constructed a known, chaotic time series based on the combination of drops of colored water making their way through a cylinder of mineral oil. The time series used was constructed via a physical process experimental technique discussed in [24].

The point of constructing these data was an attempt to deceive the LGP software into predicting an unpredictable relationship, that is, the information content of the preceding values from the drop experiment are not sufficient to predict the next value. Accordingly, if the LGP technique found a relationship on this chaotic series, it would have found a false relationship and its ability to generalize relationships from data would be suspect.

The LGP was configured to train on a data set as follows:

- The inputs were comprised of eight consecutive values from the drop data; and
- The target output was the next-in-sequence value of the drop data.

Various attempts were tried to trick the LGP technique, including varying parameters such as the instructions that were available for evolving the solution.

The results of this memorization test are shown on Fig. 4. The “step” function shown in Fig. 4 represents the measured drop data, sorted by value. The noisy data series is the output of the best LGP model of the drop data.

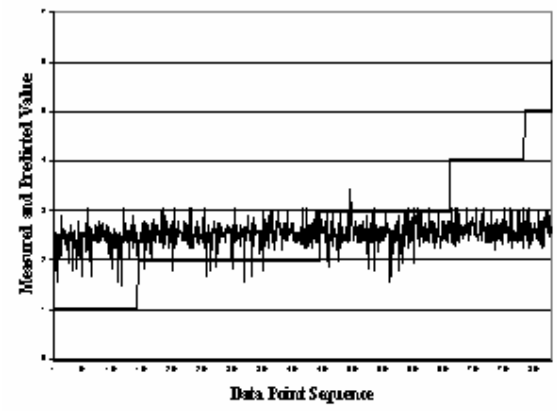

Fig. 1. Attempt to model a chaotic time series with Linear Genetic Programming.

It is clear that the LGP algorithm was not fooled by this data set. It evolved a program that was approximately a linear representation of the average value of the data set. But it did not memorize or fit the noise.

Discipulus Study—Phase II

Phase II of our investigation started when we began using the Professional Version of Discipulus [22]. As noted above, this new version automated many aspects of conducting multiple runs, including automatically randomizing run parameters, hillclimbing to optimize run parameters, automatic determination of the appropriate termination criterion for LGP for a particular problem domain, and automatic creation of team solutions.

Incinerator Problem, Phase II

SAIC used the new software version and re-ran the R&D problem involving CO2 level prediction for the incinerator plant problem (described above). A total of 901,983,797 programs were evaluated to produce the best 30 programs.

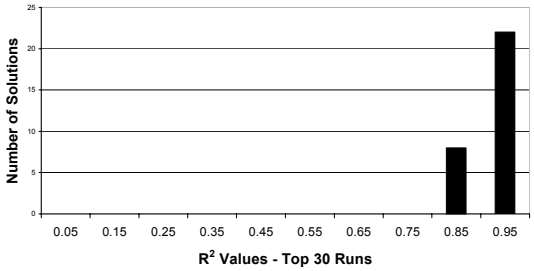

Fig. 2. Distribution of 30 Best LGP Runs using Randomized Run Parameters for 300 Runs on Incinerator Problem

The enhanced LGP algorithm modeled the Incinerator plant CO_2 levels with better accuracy and much more rapidly than earlier versions. The validation-dataset, seven-team, R^2 fitness was 0.985 as opposed to 0.961 previously achieved by multiple single runs. The CPU time for the new algorithm was 67 hours (using a PIII-800 MHz/100 MHz FSB machine), as opposed to 600 hours (using a PIII 533 MHz /133 FSB machine) that was needed in Phase I. It is important to note that the team solution approach was important in developing a better solution in less time.

UXO Discrimination

The preceding examples are regression problems. The enhanced LGP algorithm was also tested during Phase II on a difficult classification problem–the determination of the presence of subsurface unexploded ordnance (UXO).

The Department of Defense has been responsible for conducting UXO investigations at many locations around the world. These investigations have resulted in the collection of extraordinary amounts of geophysical data with the goal of identifying buried UXO.

Evaluation of UXO/non-UXO data is time consuming and costly. The standard outcome of these types of evaluations is maps showing the location of geophysical anomalies. In general, what these anomalies may be (i.e., UXO, non-UXO, boulders, etc.) cannot be determined without excavation at the location of the anomaly.

The Department of Defense has prepared 'test-beds' for testing performance of contractors in distinguishing UXO from non-UXO. The Jefferson Proving Grounds Phase IV test-bed consists of 160 buried targets. Ten contractors were engaged by the DoD to survey this test-bed with magnetic or electromagnetic sensors and then tell the DoD whether each target was UXO or not. The various contractors collected data and then submitted the data to their geophysicists who are experts at UXO discrimination.

Figure 1 shows the published performance of the ten contractors [13]. The horizontal axis shows the performance of each contractor in correctly identifying points that *did not* contain buried UXO. The vertical axis shows the performance of each algorithm in correctly identifying points that *did* contain buried UXO. The angled line in Fig. 1 represents what would be expected from random guessing.

Figure 1 points out the difficulty of modeling these data. Most contractors did little better than random guessing; however, the LGP algorithm derived a best-know model for correctly identifying UXO's and for correctly rejecting non-UXO's using various data set configurations [5,13]. The triangle in the upper right hand corner of Fig. 1 shows the range of LGP solutions in these different configurations.

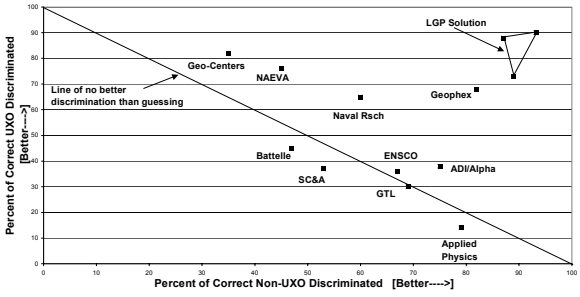

Fig. 1. LGP and Ten other Contractors Results on JPG-IV UXO Discrimination Data [13]

Eight-Problem Comparative Study

In 2001, we concluded Phase II of our LGP study with a comparative study using machine-code-based, Linear Genetic Programming, back-propagation neural networks, Vapnick Statistical Regression/Support Vector Machines [28], and C5.0 Decision Trees [21] on a suite of real-world, modeling problems.

The test suite included six regression problems and two classification problems. LGP and Vapnick Statistical Regression were used on all problems. In addition, on regression problems, Neural Networks were used and on classification problems, C5.0 was used.

In summary, each algorithm was trained on the same data as the others and was also tested on the same held-out data as the others. The figures reported below are the performance on the *held-out, testing data.* Each algorithm was run so as to maximize its performance, except that the LGP system was run at its default parameters in each case.

Classification Data Sets Results

Table 1 reports the comparative classification error rates of the best LGP, Vapnick Regression, and C5.0 results on the classification suite of problems on the held-out, testing data.

Table 1. Comparison of Error Rates of Best LGP, C5.0, and Vapnick Regression Results on Unseen Data for Two Industrial Classification Data Sets.

Problem	*Linear Genetic Programming*	*C5.0 Decision Tree*	*Vapnick Regression*
Company H Spam Filter	3.2%	8.5%	9.1%
Predict Income from Census Data	14%	14.5%	15.4%

Regression Data Sets Results

Table 2 summarizes the R^2 performance of the three modeling systems across the suite of regression problems on the held-out testing data.

Table 2. Comparison of LGP, Neural Networks and Vapnick Regression on Six Industrial Regression Problems. Value Shown is the R^2 Value on Unseen Data Showing Correlation between the Target Function and the Model's Predictions. Higher Values are Better.

Problem	*Linear Genetic Programming*	*Neural Network*	*Vapnick Regression*
Dept. of Energy, Cone Penetremeter,	0.72	0.618	0.68
Kodak, Software Simulator	0.99	0.9509	0.80
Company D, Chemical Batch Process Control	0.72	0.63	0.72
Laser Output Prediction	0.99	0.96	0.41
Tokamak 1	0.99	0.55	N/A
Tokamak 2	0.44	.00	.12

Two Examples from the Eight-Problem Study

This section will discuss two examples of results from the eight-problem comparison study—the Laser Output prediction data and the Kodak Simulator data.

Laser Output Problem. This data set comprises about 19,000 data points with 25 inputs. This is sequential data so the last 2,500 data points were held out for testing. The problem is to predict the output level of a ruby laser, using only previously measured outputs.

This is easy data set on which to generate a decent R2 value; but it is very difficult to model the phase with precision. Most modeling tools pick up the strong pe-

riodic element but have a difficult time matching the phase and/or frequency components—they generate their solutions by lagging the actual output by several cycles. Figures 2 and 3 show the output of Vapnick Regression and LGP, respectively, plotted against a portion of the unseen laser testing data.

Figure 2 is the result from the Vapnick tool. It picks up the strong periodic element but critically, the predicted output lags behind the actual output by a few cycles. By way of contrast, Figure 3 shows the results from LGP modeling. Note the almost perfect phase coherence of the LGP solution and the actual output of the laser both before and after the phase change. The phase-accuracy of the LGP models is what resulted in such a high R^2 for the LGP models, compared to the others.

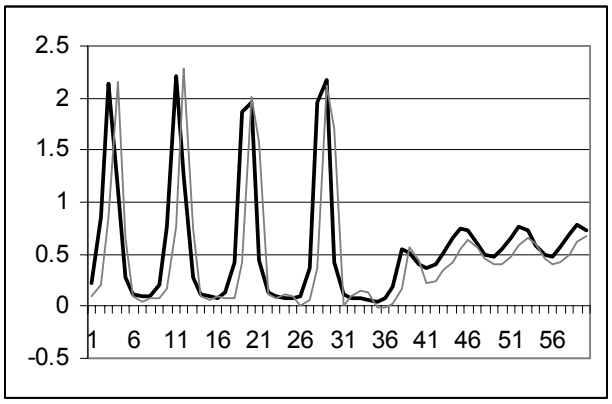

Fig. 2. Best Vapnick Regression Model on Laser Problem (Light Gray Line) Compared to Target Output (Heavy Line) on Held-Out, Data

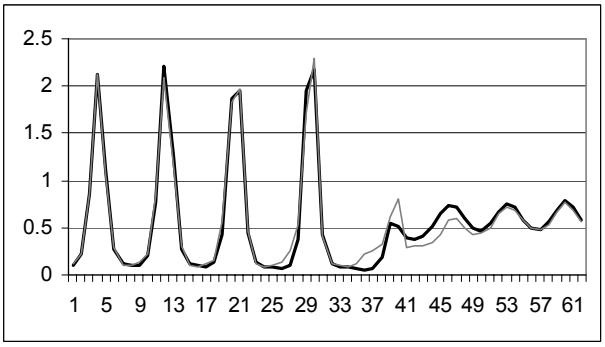

Fig. 3. Best LGP Model (Light Gray Line) on Laser Problem Compared to Target Output (Dark Line) on Held-Out, Testing Data

Simulating a Simulator. In the Kodak Simulator problem, the task was to use LGP to simulate an existing software simulator. Past runs of the existing simulator provided many matched pairs of inputs (five production related variables) and the output from [23]. The data set consisted of 7,547 simulations of the output of a chemical batch process, given the five input variables common to making produc-

tion decisions. Of these data points, 2,521 were held out of training for testing the model.

The results on the testing or held-out data for LGP, Vapnick Regression, and neural networks are reported in Table 2. Figures 4 and 5 graph the LGP and Vapnick Models against the target data.

The LGP solution (Figure 5) so closely models the Target Output that the predictions completely obscure the target output line. In fact, for all but six of the 2,521 data points, the agreement between the LGP prediction and the actual value is very good. The R^2 fitness on the applied data set for the best team solution was 0.9889. (A second batch of 232 LGP runs achieved a similar R^2 fitness on the applied data set of 0.9814, using a team of seven programs. The range of R^2 for the top 30 programs of this second batch was 0.9707 to 0.9585. This demonstrates analysis repeatability using LGP.)

The Vapnick (Figure 4) and Neural Network solutions were not nearly so close—the R^2 for the Vapnick Model was only 0.80, for example.

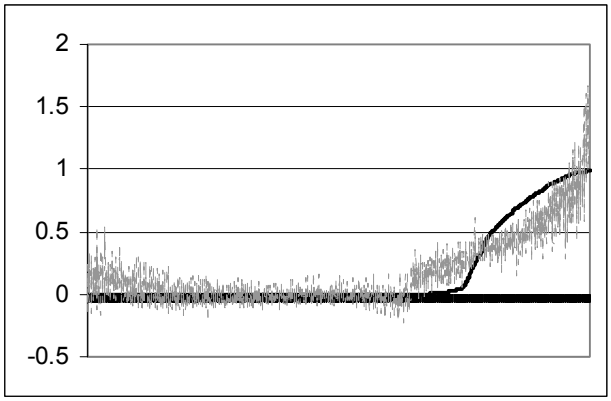

Fig. 4. Best Vapnick Predictions of Kodak Simulator Data (light-gray series) vs. the Target Data (dark line) on Held-out Data.

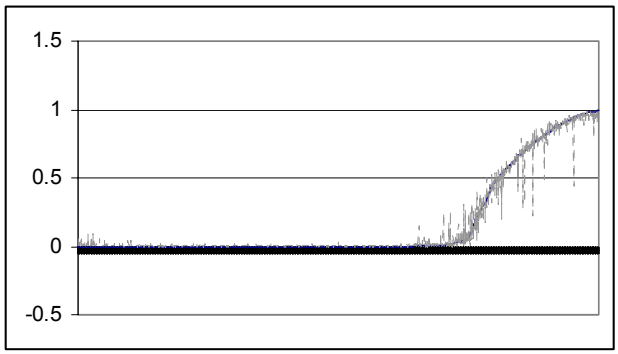

Fig. 5. Best LGP Model of Company K Simulator Problem (light gray series) vs Target Data (dark series) on the Held-out Data.

Conclusion Regarding Empirical Studies

The key results of the two phases of our empirical studies of the LGP algorithm are as follows.

First: Discipulus consistently produces excellent results on difficult, industrial modeling problems with little customization of the learning algorithm. Note: it did not always produce *better* results than all other algorithms studied. However, on every problem studied, LGP produced a model that was as good as, or better than, any other approach.

The performance of other learning algorithms was decidedly up-and-down. For example, Vapnick Regression did quite well on the Cone Penetrometer and Company D data but quite poorly on the laser and Company K problems. Neural Networks did quite well on the laser and Company K problems but not so well on the Tokamak and incinerator CO2 data sets. C5.0 did well on the census problem but not well on the spam filter problem.

Our comfort level that LGP will arrive at a good solution to most problems without customization or 'tweaking" is one of the principal reasons we have settled on LGP as our modeling algorithm of choice for complex and difficult modeling problems.

Second: Discipulus produces robust models compared to other learning algorithms. Much less attention had to be paid to overfitting problems with Discipulus than with other algorithms. This is not to say that LGP will never overfit data. Give the right data set, it will. But it does so less frequently than the Neural Network, Vapnick Regression, and C5.0 alternatives we studied

Discipulus identifies which are the important inputs, and which are not. For example, we screened a wastewater treatment plant with 54 inputs and identified 7 important ones. This reduces the number of inputs to monitor, allows assessment of what will happen if an input goes off-line (for security and contingency planning), and enhances accelerated optimization by reducing the number of decision variables, as discussed below.

Acknowledgments

The results presented in this work are part of a three-plus year collaborative effort between SAIC and Register Machine Learning Technologies to advance the state of the-art of evolutionary computation as applied to complex systems. Specifically thanked for funding and latitude from SAIC are Joseph W. Craver, John Aucella, and Janardan J. Patel. Dr. Gregory Flach, Dr. Frank Syms, and Mr. Robert Walton are gratefully acknowledged for providing the input data sets used in some of the work – as are the researchers who need to keep their identity confidential for business reasons. Christopher R. Wellington -Ad Fontes Academy, Centreville, Vir-

ginia conducted the chaotic drop experiment. All computations were performed by, and responsibility for their accuracy lies with, the authors.

References

1. Banzhaf, W., Nordin, P., Keller, R., Francone, F. (1998) Genetic Programming, An Introduction, Morgan Kauffman Publishers, Inc., San Francisco, CA.
2. Deschaine, L.M., (2000) Tackling real-world environmental challenges with linear genetic programming. PCAI Magazine, Volume 15, Number 5, September/October, pp. 35-37.
3. Deschaine, L.M., Patel, J.J., Guthrie, R.G., Grumski, J.T., and Ades, M.J. (2001) "Using Linear Genetic Programming to Develop a C/C++ Simulation Model of a Waste Incinerator," The Society for Modeling and Simulation International: Advanced Simulation Technology Conference, Seattle, WA, USA April, ISBN: 1-56555-238-5, pages 41-48.
4. Deschaine, L.M., Hoover, R.A. Skibinski, J. (2002) "Using Machine Learning to Complement and Extend the Accuracy of UXO Discrimination Beyond the Best Reported Results at the Jefferson Proving Grounds," (in press), Proceedings of Society for Modeling and Simulation International, April.
5. Fausett, L.V. (2000). A Neural Network Approach to Modeling a Waste Incinerator Facility, Society for Computer Simulation's Advanced Simulation Technology Conference, Washington, DC, USA April.

Francone, F. D., and Deschaine, L.M., Extending the Boundaries of Design Optimization by Integrating Fast Optimization Techniques with Machine-Code-Based Linear Genetic Programming, Information Sciences Journal, Elsevier Press, Vol. 161/3-4 pp 99-120: 2004. Amsterdam, the Netherlands.

6. Francone, F., Nordin, P., and Banzhaf. W. (1996) Benchmarking the Generalization Capabilities of a Compiling Genetic Programming System Using Sparse Data Sets, In Koza et al. Proceedings of the First Annual Conference on Genetic Programming, Stanford, CA.
7. Fukunaga, A., Stechert, A., Mutz, D. (1998) A Genome Compiler for High Performance Genetic Programming, in Proceedings of the Third Annual Genetic Programming Conference, pp. 86-94, Morgan Kaufman Publishers, Jet Propulsion Laboratories, California Institute of Technology Pasadena, CA.
8. Government Accounting Office (2001) "DOD Training Range Clean-up Cost Estimates are Likely Understated," Report to House of Representatives on Environmental Liabilities, USA General Accounting Office, April, Report no. GAO 01 479.
9. Hansen, N., and Ostermeier, A. (2001) Completely Derandomized Self-Adaptation in Evolution Strategies. In Evolutionary Computation 9(2): 159-195.
12. Jefferson Proving Grounds (1999) Jefferson Proving Grounds Phase IV Report: Graph ES-1, May, Report No: SFIM-AEC-ET-CR-99051.
14. Koza, J., Bennet, F., Andre, D., and Keane, M. (1999) Genetic Programming III. Morgan Kaufman, San Francisco, CA.

15. Nordin, J.P. (1994) A Compiling Genetic Programming System that Directly Manipulates the Machine Code. In Advances in Genetic Programming, K. Kinnear, Jr. (ed.), Cambridge MA: MIT Press.

16. Nordin, J.P. (1999) Evolutionary Program Induction of Binary Machine Code and its Applications, Krehl Verlag.

17. Nordin, J.P., Banzhaf , W. (1995) Complexity Compression and Evolution. In Proceedings of Sixth International Conference of Genetic Algorithms, Morgan Kaufmann Publishers, Inc.

18. Nordin, J.P., Banzhaf, W. (1995) Evolving Turing Complete Programs for a Register Machine with Self Modifying Code. In, Proceedings of Sixth International Conference of Genetic Algorithms, Morgan Kaufmann Publishers, Inc.

19. Nordin, J.P., Francone, F., and Banzhaf, W. (1996) Explicitly Defined Introns and Destructive Crossover in Genetic Programming. Advances in Genetic Programming 2, K. Kinnear, Jr. (Editor), Cambridge MA: MIT Press.

20. Nordin, J.P., Francone, F., and Banzhaf, W. (1998) Efficient Evolution of Machine Code for CISC Architectures Using Blocks and Homologous Crossover. In Advances in Genetic Programming 3, MIT Press, Cambridge MA.

21. Quinlan, R. (1998) Data Mining Tools See5 and C5.0. ,Technical report, RuleQuest Research.

22. Register Machine Learning Technologies, Inc. (2002) Discipulus Users Manual, Version 3.0. Available at www.aimlearning.com.

23. Brian S. Rice and Robert L. Walton of Eastman, Kodak.Company, Industrial Production Data Set.

24. Scientific American (November 1999). Drop Experiment to Demonstrate a Chaotic Time Series.

28. Vapnick V. (1998) The Nature of Statistical Learning Theory, Wiley-Interscience Publishing.