

---

## Extending the Boundaries of Design Optimization by Integrating Fast Optimization Techniques with Machine Code Based, Linear Genetic Programming

L. M. Deschaine, F.D. Francone

### 2.1 Introduction

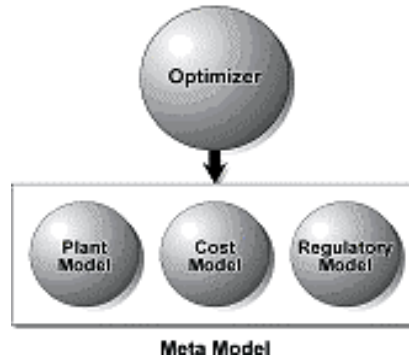
Engineers frequently encounter problems that require them to estimate control or response settings for industrial or business processes that optimize one or more goals. Most optimization problems include two distinct parts: (1) a model of the process to be optimized; and (2) an optimizer that varies the control parameters of the model to derive optimal settings for those parameters.

For example, one of the research and development (R&D) case studies included here involves the control of an incinerator plant to achieve a high probability of environmental compliance and minimal cost. This required predictive models of the incinerator process, environmental regulations, and operating costs. It also required an optimizer that could combine the underlying models to calculate a real-time optimal response that satisfied the underlying constraints. Figure 2.1 shows the relationship of the optimizer and the underlying models for this problem.

The incinerator example discussed above and the other case studies below did not yield to a simple constrained optimization approach or a well-designed neural network approach. The underlying physics of the problem were not well understood; so this problem was best solved by decomposing it into its constituent parts—the three underlying models (Figure 2.1) and the optimizer.

This work is, therefore, concerned with complex optimization problems characterized by either of the following situations.

**First:** Engineers often understand the underlying processes quite well, but the software simulator they create for the process is slow. Deriving optimal settings for a slow simulator requires many calls to the simulator. This makes optimization inconvenient or completely impractical. Our solution in this situation was to reverse engineer the existing software simulator using Linear Genetic Programming (LGP)—in effect, we simulated the simulator. Such “second-order” LGP simulations are frequently very accurate and almost always orders of magnitude faster than the hand-coded simulator. For example, for the Kodak Simulator, described below, LGP reverse engineered that simulator, reducing the time per simulation from hours to less than a second. As a result, an optimizer may be applied to the LGP-derived simulation quickly and conveniently.



**Figure 2.1** How the optimizer and the various models operate together for the incinerator solution.

**Second:** In the incinerator example given above, the cost and regulatory models were well understood, but the physics of the incinerator plant were not. However, good-quality plant operating data existed. This example highlights the second situation in which our approach consistently yields excellent results. LGP built a model of plant operation directly from the plant operation data. Combined with the cost and regulatory models to form a meta-model, the LGP model permits real-time optimization to achieve regulatory and cost goals.

For both of the above types of problems, the optimization and modeling tools should possess certain clearly definable characteristics:

- The optimizer should make as few calls to the process model as possible, consistent with producing high-quality solutions,
- The modeling tool should consistently produce high-precision models that execute quickly when called by the optimizer,
- Both the modeling and optimizing tools should be general-purpose tools. That is, they should be applicable to most problem domains with minimal customization and capable of producing good to excellent results across the whole range of problems that might be encountered; and
- By integrating tools with the above characteristics, we have been able to improve problem-solving capabilities very significantly for both problem types above.

This work is organized as follows. We begin by introducing the Evolution Strategies with Completely Derandomized Self-Adaptation (ES-CDSA) algorithm as our optimization algorithm of choice. Next, we describe machine-code-based, LGP in detail and describe a three-year study from which we have concluded that machine-code-based, LGP is our modeling tool of choice for these types of applications. Finally, we suggest ways in which the integrated optimization and modeling strategy may be applied to design optimization problems.

## 2.2 Evolution Strategies Optimization

ES was first developed in Germany in the 1960s. It is a very powerful, general-purpose, parameter optimization technique [25,26,27]. Although we refer in this work to ES, it is closely related to Fogel's Evolutionary Programming (EP) [7, 1]. Our discussion here applies equally to ES and EP. For ease of reference, we will use the term "ES" to refer to both approaches.

ES uses a population-based learning algorithm. Each generation of possible solutions is formed by mutating and recombining the best members of the previous generation. ES pioneered the use of evolvable "strategy parameters." Strategy parameters control the learning process. Thus, ES evolves both the parameters to be optimized and the parameters that control the optimization [2].

ES has the following desirable characteristics for the uses in our methodology:

- ES can optimize the parameters of arbitrary functions. It does not need to be able to calculate derivatives of the function to be optimized, nor does the researcher need to assume differentiability and numerical accuracy. Instead, ES gathers gradient information about the function by sampling. [12]
- Substantial literature over many years demonstrates that ES can solve a very wide range of optimization problems with minimal customization. [25, 26, 27, 12]

Although very powerful and not prone to getting stuck in local optima, typical ES systems can be very time-consuming for significant optimization problems. Thus, canonical ES often fails the requirement of efficient optimization.

But in the past five years, ES has been extended using the ES-CDSA technique [12]. ES-CDSA allows a much more efficient evolution of the strategy parameters and cumulates gradient information over many generations, rather than single generation as used in traditional ES.

As a rule of thumb, where  $n$  is the number of parameters to be optimized, users should allow between 100 and  $200(n+3)^2$  function evaluations to get optimal use from this algorithm [12]. While this is a large improvement over previous ES approaches, it can still require many calls by the optimizer to the model to be optimized to produce results. As a result, it is still very important to couple ES-CDSA with fast-executing models. And that is where LGP becomes important.

## 2.3 Linear Genetic Programming

Genetic Programming (GP) is the automatic 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 chapter presents three years of analysis of machine-code-based, LGP. To perform the analyses, we used Versions 1 through 3 of an off-the-shelf commercial

software package called Discipulus™ [22]. Discipulus is an LGP system that operates directly on machine code.

### 2.3.1 The Genetic Programming Algorithm

Good, detailed treatments of GP 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.

### 2.3.2 Linear Genetic Programming Using Direct Manipulation of Binary Machine Code

Machine-code-based, LGP is the direct evolution of binary machine code through GP techniques [15, 16, 17, 18, 20]. Thus, an evolved LGP program is a sequence of binary machine instructions. For example, an evolved LGP program might be comprised of a sequence of four, 32-bit machine instructions. When executed, those four instructions would cause the central processing unit (CPU) to perform operations on the CPU’s hardware registers. Here is an example of a simple, four-instruction LGP program that uses three hardware registers:

```

register 2 = register 1 + register 2
register 3 = register 1 - 64
register 3 = register 2 * register 3
register 3 = register 2 / register 3

```

While LGP programs are apparently very simple, it is actually possible to evolve functions of great complexity using only simple arithmetic functions on a register machine [18, 20].

After completing a machine-code LGP project, the LGP software decompiles the best evolved models from machine code into Java, ANSI C, or Intel Assembler programs [22]. The resulting decompiled code may be linked to the optimizer and compiled or it may be compiled into a DLL or COM object and called from the optimization routines.

The linear machine code approach to GP has been documented to be between 60 to 200 times faster than comparable interpreting systems [10, 15, 20]. As will be developed in more detail in the next section, this enhanced speed may be used to conduct a more intensive search of the solution space by performing more and longer runs.

## 2.4 Why Machine-Code-based, Linear Genetic Programming?

At first glance, it is not at all obvious that machine-code, LGP is a strong candidate for the modeling algorithm of choice for the types of complex, high-dimensional problems at issue here. But over the past three years, a series of tests was performed on both synthetic and industrial data sets—many of them data sets on which other modeling tools had failed. The purpose of these tests was to assess machine-code, LGP's performance as a general-purpose modeling tool.

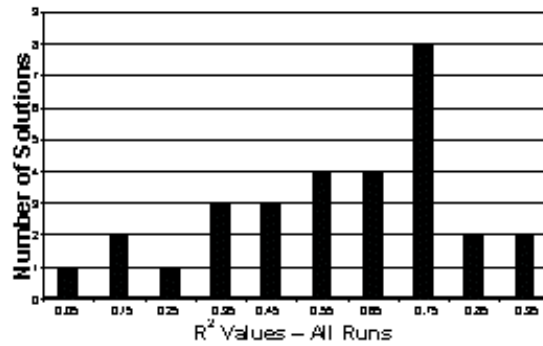
In brief summary, the machine-code-based LGP software [22] has become our modeling tool of choice for complex problems like the ones described in this work for several reasons:

- Its speed permits the engineer to conduct many runs in realistic timeframes on a desktop computer. This results in consistent, high-precision models with little customization;
- It is well-designed to prevent overfitting and to produce robust solutions; and
- The models produced by the LGP software execute very quickly when called by an optimizer.

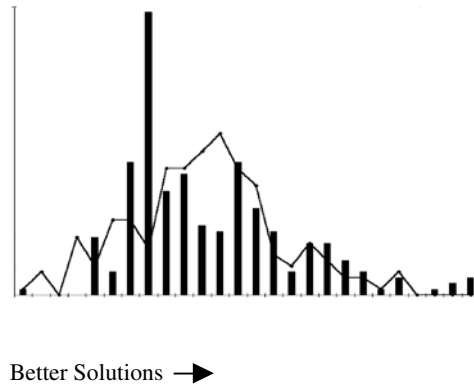
We will first discuss the use of multiple LGP runs as a key ingredient of this technique. Then we will discuss our investigation of machine-code, LGP over the past three years.

## 2.5 Multiple Linear Genetic Programming Runs

GP is a stochastic algorithm. Accordingly, running it over and over with the same inputs usually produces a wide range of results, ranging from very bad to very good. For example, Figure 2.2 shows the distribution of the results from 30 runs of LGP on the incinerator plant modeling problem mentioned in the introduction—the  $R^2$  value is used to measure the quality of the solution. The solutions ranged from a very poor  $R^2$  of 0.05 to an excellent  $R^2$  of 0.95. Our investigation to date strongly suggests the typical LGP distribution of results from multiple LGP runs includes a distributional tail of excellent solutions that is not always duplicated by other learning algorithms.



**Figure 2.2** . Incinerator control data. Histogram of results for 30 LGP runs.



**Figure 2.3** Typical comparative histograms of the quality of solutions produced by LGP runs (bars) and Neural Network runs (lines). Discussed in detail in [8].

For example, for three separate problem domains, an LGP system produced a long tail of outstanding solutions, even though the average LGP solution was not necessarily very good. By way of contrast, and in that same study, the distribution of many neural networks runs on the same problems often produced a good average solution, but did not produce a tail of outstanding solutions like LGP [4,8].

Figure 2.3 shows a comparative histogram of LGP results versus neural network results derived from 720 runs of each algorithm on the same problem. Better solutions appear to the right of the chart. Note the tail of good LGP solutions (the bars) that is not duplicated by a comparable tail of good neural network solutions. This same pattern may be found in other problem domains [4,8].

To locate the tail of best solutions on the right of Figure 2.3, it is *essential* to perform many runs, regardless whether the researcher is using neural networks or LGP. This is one of the most important reasons why a machine-code approach to GP is preferable to other approaches. It is so much faster than other approaches, that it is possible to complete many runs in realistic timeframes on a desktop computer. That makes it more capable of finding the programs in the good tail of the distribution.

## 2.6 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. Early versions of the Discipulus LGP software permitted 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 CO<sub>2</sub> data sets (Figure. 2.2) 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 new capabilities introduced into later versions of the LGP software. 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.

## 2.7 Investigation of Machine-Code-Based, Linear Genetic Programming—Phase I

We tested Versions 1.0 and 2.0 of the Discipulus LGP 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.

### 2.7.1 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

$$Q=K*I*A, \quad (2.1)$$

where  $Q$  = flow [ $L^3/T$ ],  $K$  = hydraulic conductivity [ $L/T$ ],  $I$  = gradient [ $L/L$ ], and  $A$  = area [ $L^2$ ].

To test LGP, 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 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].

### 2.7.2 Incinerator Process Simulation

The second LGP test SAIC performed was the prediction of CO<sub>2</sub> 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 CO<sub>2</sub> 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 offset 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 Figure 2.2), so we would expect to have to conduct all 30 runs to solve the problem again.

### 2.7.3 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 is 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:

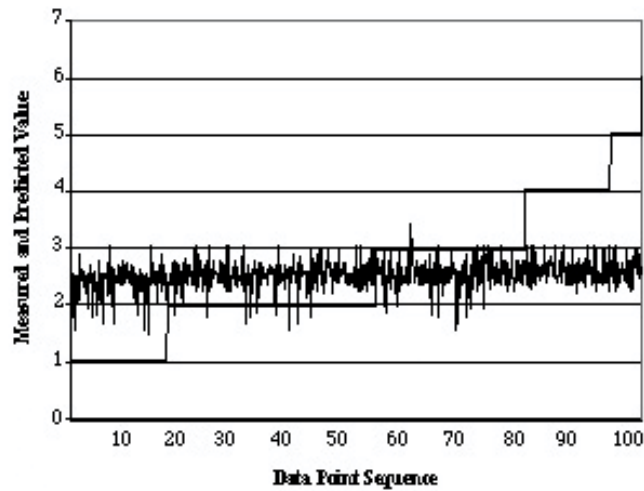
- The inputs were comprised of eight consecutive values from the drop data; and
- The target output was the next-in-sequence value of the drop data.

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

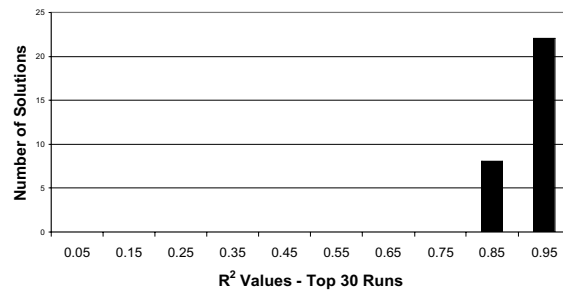
The results of this memorization test are shown on Figure 2.4. The “step” function shown in Figure 2.4 represents the measured drop data, sorted by value. The noisy data series is the output of the best LGP model of the drop data. It is clear that the LGP algorithm was not fooled by this data set. It 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.

## 2.8 Investigation of Machine-Code-based, Linear Genetic Programming—Phase II

Phase II of our investigation started when we began using Version 3.0 of the LGP software [22]. As noted above, this new version automated many aspects of conducting multiple runs, including automatically randomizing run parameters, hill-climbing to optimize run parameters, automatic determination of the appropriate termination criterion for LGP for a particular problem domain, and automatic creation of team solutions.



**Figure 2.4** Attempt to model a chaotic time series with LGP.



**Figure 2.5** Distribution of 30 best LGP runs using randomized run parameters for 300 runs on incinerator problem

### 2.8.1 Incinerator Problem, Phase II

SAIC used the new software version and reran the R&D problem involving CO<sub>2</sub> level prediction for the incinerator plant problem (described above). A total of 901,983,797 programs was evaluated to produce the distribution of the best 30 program results shown in Figure 2.5.

The enhanced LGP algorithm modeled the incinerator plant CO<sub>2</sub> levels with better accuracy and much more rapidly than earlier versions. The validation-dataset, seven-team, R<sup>2</sup> 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.

### 2.8.2 UXO Discrimination

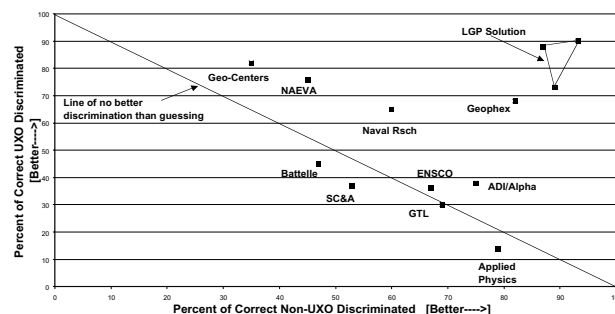
The preceding examples are regression problems. The enhanced LGP algorithm was also tested during Phase II on a difficult classification challenge the determination of the presence of subsurface unexploded ordnance (UXO).

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.

Figure 2.6 shows the performance of 10 published industrial-strength, discrimination algorithms on the Jefferson Proving Grounds UXO data—which consisted of 160 targets [13]. The horizontal axis shows the performance of each algorithm 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 Figure 2.6 represents what would be expected from random guessing.

Figure 2.6 points out the difficulty of modeling these data. Most algorithms did little better than random guessing; however, the LGP algorithm derived a best-known 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 Figure 2.6 shows the range of LGP solutions in these different configurations.



**Figure 2.6** LGP and 10 other algorithms applied to the UXO discrimination data [13].

### 2.8.3 Eight-Problem Comparative Study

In 2001, we concluded Phase II of our LGP study with a comparative study using machine-code-based, LGP, back-propagation neural networks, Vapnick Statistical Regression (VSR) [28], and C5.0 [21] on a suite of real-world modeling problems.

The test suite included six regression problems and two classification problems. LGP and VSR were used on all problems. In addition, on regression problems, neural networks were used and on classification problems, C5.0 was used.

Space constraints prevent us from detailing the experimental protocol in detail. That detail may be obtained in [9]. 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.

#### 2.8.3.1 Classification Data Sets Results

Table 2.1 reports the comparative classification error rates of the best LGP, VSR, and C5.0 results on the classification suite of problems on the held-out, testing data.

**Table 2.1** Comparison of Error Rates of Best LGP, C5.0, and Vapnick Regression Results on Two Industrial Classification Data Sets. Reported Results Are on Unseen Data. Lower is Better.

<i>Problem</i>	<i>Linear Genetic Programming</i>	<i>C5.0 Decision Tree</i>	<i>Vapnick Regression</i>
Company H spam filter	3.2%	8.5%	9.1%
Predict income from census data	14%	14.5%	15.4%

#### 2.8.3.2 Regression Data Sets Results

Table 2.2 summarizes the  $R^2$  performance of the three modeling systems across the suite of regression problems on the held-out testing data.

#### 2.8.3.3 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.

**Table 2.2** 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.

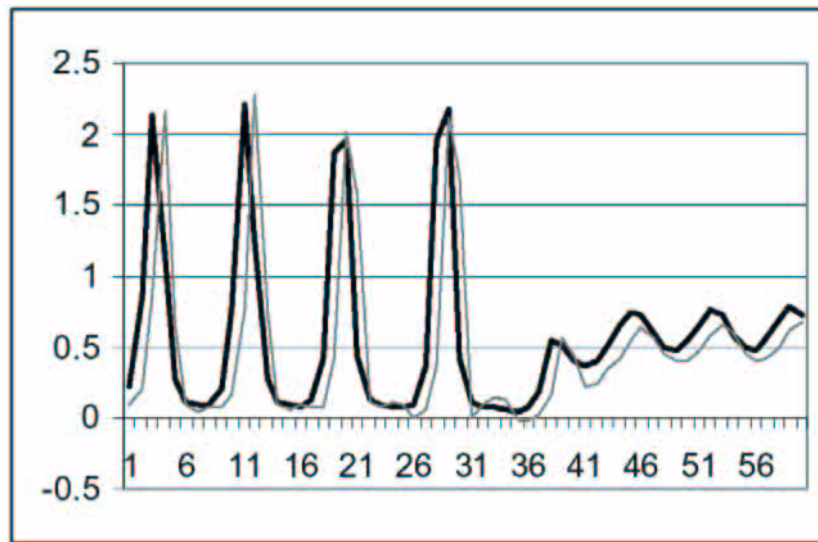
<i>Problem</i>	<i>Linear Genetic Programming</i>	<i>Neural Network</i>	<i>Vapnick Regression</i>
Dept. of Energy, cone penetrometer,	0.72	0.618	0.68
Kodak, software simulator	0.99	0.9509	0.80
Company D, chemical batch process control	0.72	0.63	0.72
Laser output prediction	0.99	0.96	0.41
Tokamak 1	0.99	0.55	N/A
Tokamak 2	0.44	.00	.12

This is an easy data set to do well upon; but it is very difficult to model the phase with precision. Most modeling tools pick up the strong periodic 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 2.7 and 2.8 show the output of VSR and LGP, respectively, plotted against a portion of the unseen laser testing data.

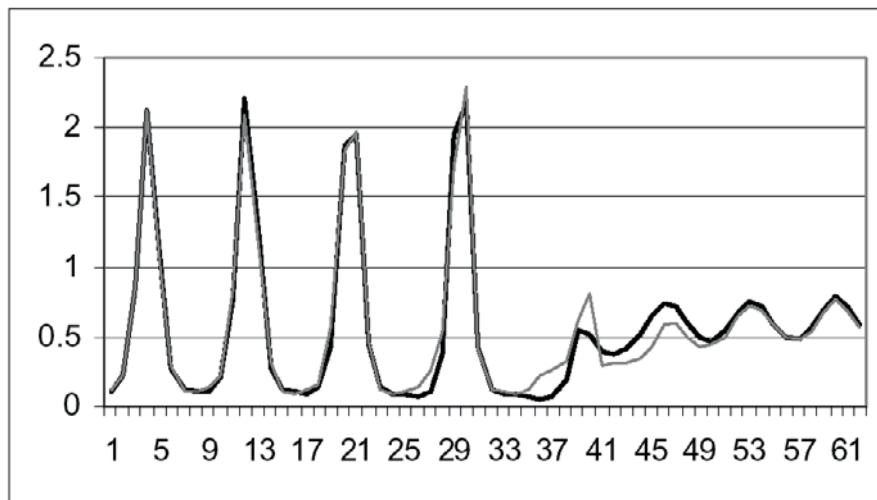
Figure 2.7 is the result from the Vapnick tool. It picks up the strong periodic element but, critically, the predicted output lags behind the actual output by a few cycles. By way of contrast, Figure 2.8 shows the results from LGP modeling. Note the almost perfect phase coherence of the LGP solution 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.

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

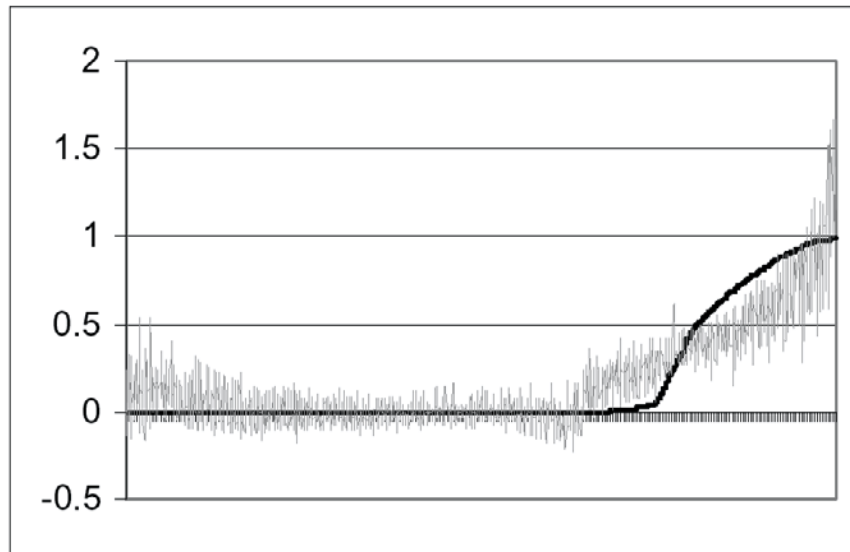
The results on the testing or held-out data for LGP, VSR, and neural networks are reported in Table 2.2. Figures 2.9 and 2.10 graph the LGP and Vapnick models against the target data.



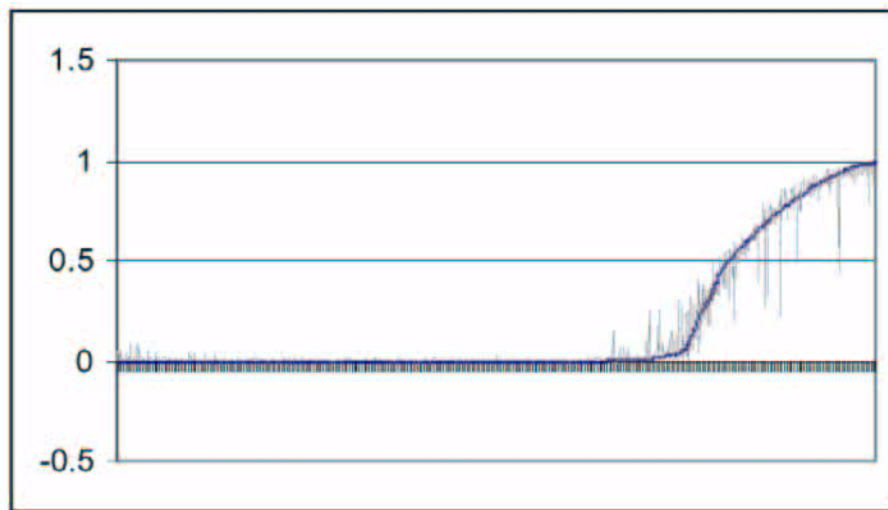
**Figure 2.7** Best VSR model on laser problem (light gray line) compared to target output (heavy line) on held-out data.



**Figure 2.8** Best LGP model (light gray line) on laser problem compared to target output (dark line) on held-out testing data.



**Figure 2.9** Best Vapnick predictions of Kodak simulator data (light-gray series) vs. the target data (dark line) on held-out data.



**Figure 2.10** Best LGP model of company K simulator problem (light gray series) vs. target data (dark series) on the held-out data.

The LGP solution (Figure 2.10) 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 ap-

plied 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 2.9) and neural network solutions were not nearly so close—the  $R^2$  for the Vapnick model was only 0.80, for example.

## 2.9 Conclusion Regarding Empirical Studies

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

**First:** The LGP software we used consistently produces excellent results on difficult, industrial modeling problems with little customization of the learning algorithm. Note: LGP 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 algorithm.

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 CO<sub>2</sub> data sets. C5.0 did well on the census problem but not well on the spam filter problem.

We speculate that one important reason behind the consistently good performance of LGP is that it performs, by default, many runs. Accordingly, it locates the tail of good performing solutions discussed above. 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:** LGP produces robust models compared to other learning algorithms. Much less attention had to be paid to overfitting problems with the LGP software than with other algorithms. This is not to say that LGP will never overfit data. Given the right data set, it will. But it does so less frequently than the neural network, Vapnick regression, and C5.0 alternatives we studied

The LGP system identifies 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.

## 2.10 Integrated System Analysis

This work is concerned with the building of a system comprised of integrated modeling and optimization tools. The integrated tool suite, comprised of (1) machine-code-based LGP for creating predictive models, and (2) ES-CDSA, is ex-



pected to ably handle a wide range of the complex problems with which we are concerned.

The remainder of this paper is devoted to discussing two application areas for the integration of these tools using two of the problems mentioned above—the incinerator R&D problem and the UXO discrimination problem.

### 2.10.1 Optimizing the Incinerator Model

The incinerator project was conceived from the beginning as a real-time control project. The models built with LGP predicted CO<sub>2</sub> levels in the secondary combustion chamber as a function of (1) The measured state of the plant at several previous time iterations; and (2) the plant control settings at previous time iterations.

Because plant control settings are part of the evolved LGP models, they may be optimized in response to each new state of the plant. That optimization may optimize for lowest cost operation, operation with a high probability of regulatory compliance, or both. Space limitations prevent a detailed description of the optimizer operation. However, details, including screenshots of the optimizer application, may be found in [4].

In terms of speed, optimization of these fast LGP models is practical and useful. The control programs evolved by LGP contain no more than 200 instructions, so they will execute on a modern Pentium computer in far less than a millisecond. So, during optimization, each call to the optimizer occupies less than a millisecond. According to the formula given above for ES-CDSA optimization,  $200 \cdot (n+3)^2$  should suffice to derive an optimal setting for a new plant state. So, to optimize five parameters would take no more than 1.3 seconds—easily enough time to determine a new group of five control settings for the plant (the LGP model predicts an hour in advance).

### 2.10.2 Optimizing the LGP-derived UXO Models

The problem of UXO or land mines affects millions of acres worldwide and includes both training areas and former battlefields. The estimated cost for remediating the U.S. training ranges alone is at least \$14 billion, and this number is likely understated [11]. The very real cost of clean-up (or nonclean-up) is the injury or death to people.

Currently, too large a portion of the resources available for responding to UXO challenges is expended by digging up sites where UXOs are predicted, but which turn out to be false alarms—that is, false positives. This, in turn, limits funding available for remediating genuine UXOs.

Machine-code-based, LGP has derived the most accurate UXO discriminator among published results to date [Jefferson Proving Grounds 1999] by a wide margin. This LGP UXO/non-UXO identification success opens up the assessment and optimization of response to the UXO issue on both the program and the project levels:

- The presence or absence of UXO can be assessed using remote, non-destructive technology such as land- or air-based sensors, including geophysics and vari-

ous wave length sensors. Developed to a high degree of accuracy, wide areas can be screened and analyzed to reduce the footprint of areas needing further investigation. This will help manage the sheer size of this challenge.

- Areas of interest identified as requiring further investigation can be prioritized and ranked using good information on the probability or absence of UXO. This ranking would integrate the LGP UXO solution with multi-criteria/multi-objective decision support models; and
- Site-specific remedial action plans, known as “dig sheets,” can be optimally designed to focus efforts on high probability, UXO-containing areas. When the decreased predicted likelihood of UXO presence and the field verified absence of UXO are demonstrated, a stopping point for remedial activities, based on scientific principals and field validation, is provided.

## 2.11 Summary and Conclusions

We are in the early stages of building a comprehensive, integrated optimization and modeling system to handle complex industrial problems. We believe a combination of machine-code-based, LGP (for modeling) and ES CDSA (for optimization) together provides the best combination of available tools and algorithms for this task.

By conceiving of design optimization projects as integrated modeling and optimization problems from the outset, we anticipate that engineers and researchers will be able to extend the range of problems that are solvable, given today’s technology.

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

## References

1. Bäck, T., Schwefel, H.P. (1993) An Overview of Evolutionary Algorithms for Parameter Optimization, *Evolutionary Computation*, 1(1):1-23.

2. Banzhaf, W., Nordin, P., Keller, R., Francone, F. (1998) *Genetic Programming, An Introduction*, Morgan Kauffman Publishers, Inc., San Francisco, CA.
3. Deschaine, L.M. (2000) Tackling real-world environmental challenges with linear genetic programming. *PCAI Magazine*, 15(5):35-37.
4. Deschaine, L.M., Patel, J.J., Guthrie, R.G., Grumski, J.T., 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, ISBN: 1-56555-238-5, pp. 41-48.
5. 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*.
6. 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.
7. Fogel, D.B. (1992) *Evolving Artificial Intelligence*. Ph.D. thesis, University of California, San Diego, CA.
8. 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.
9. Francone F. (2002) Comparison of Discipulus™ Genetic Programming Software with Alternative Modeling Tools. Available at [www.aimlearning.com](http://www.aimlearning.com).
10. 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.
11. 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.
12. Hansen, N., Ostermeier, A. (2001) Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation* 9(2):159-195.
13. 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.), MIT Press: Cambridge MA.
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. (ed.), MIT Press: Cambridge, MA.

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](http://www.aimlearning.com).
23. Brian S. Rice and Robert L. Walton of Eastman Kodak. Company, Industrial Production Data Set.
24. Scientific American (Nov. 1999). Drop Experiment to Demonstrate a Chaotic Time Series.
25. Rechenberg, I. (1994). *Evolutionsstrategie '93*, Fromann Verlag, Stuttgart, Germany.
26. Schwefel, H.P. (1995). *Evolution and Optimum Seeking*. Sixth-Generation Computer Technology Series, John Wiley & Sons, New York.
27. Schwefel, H.P. and Rudolph, G. (1995). Contemporary Evolution Strategies. In *Advances in Artificial Life*, pp. 893-907, Springer-Verlag, Berlin.
28. Vapnick V. (1998) *The Nature of Statistical Learning Theory*, Wiley-Interscience Publishing.

Information Processing with Evolutionary Algorithms  
From Industrial Applications to Academic Speculations

Grana, M.; Duro, R.J.; d'Anjou, A.; Wang, P.P. (Eds.)

2005, XVIII, 334 p. 137 illus., Hardcover

ISBN: 978-1-85233-866-4