Design Optimization Integrating the Outer Approximation Method with Process Simulators and Linear Genetic Programming

Larry M. Deschaine, PE
Science Applications International Corporation
Larry.M.Deschaine@alum.mit.edu

Frank D. Francone
Register Machine Leaning Technologies
ffrancone@aimlearning.com

Abstract

Fast process optimization is a challenge. Processes are often complex and the intricate simulators written to solve them can take hours or days per simulation to run. Optimization techniques that require many calls to a simulator can take days or months to solve. While advances in optimization algorithms, such as the outer approximation method have reduced the solution time by a factor of ten or more when compared to other methods, long solutions times still can occur. This work explores the development of simulating a simulator to enable optimal solution development in an accelerated time frame. The technique used to develop the simulated simulator is linear genetic programming (LGP). LGP approximated a complex industrial process simulator that took hours to execute per run with a high fitness program - applied (testing) data set R^2 fitness of 0.989. The LGP solution executes in less than a second. This success opens up the possibility of optimizing functions faster using these LGP derived high fitness simulator approximations. Since the LGP simulated process simulator now executes in less than a second, as opposed to hours, using an intensive multiple call optimization technique such as genetic algorithms and evolutionary strategies is now also feasible.

Outer Approximation Method

The optimization model employed in this work is the Outer Approximation Method, as presented by Karatzas and Pinder (1993 and 1996). The method is a global minimization technique that uses a cutting plane approach to determine the optimal solution. The algorithm starts by determining a polytope that encloses the feasible region, which is defined by a set of vertices. The feasible region is determined as the space where all of the constraints are satisfied.

The objective function, the function to be minimized, is formulated as a concave function. Based on the characteristic property of concave functions that the minimum always occurs at one of the most outer points of the feasible region, the algorithm determines the vertex of the enclosing polytope that minimizes the objective function. Next, it examines if the selected vertex is feasible. If all constraints are satisfied, it declares this vertex as the optimal solution. Otherwise, a cutting plane is introduced that eliminates this vertex and its surroundings, creates a new enclosing polytope that is a better approximation of the feasible region and the process is repeated. The goal of this process is not to determine the best approximation of the feasible region, but rather to determine the most extreme point of the feasible region without eliminating any part of it (Fig. 1).

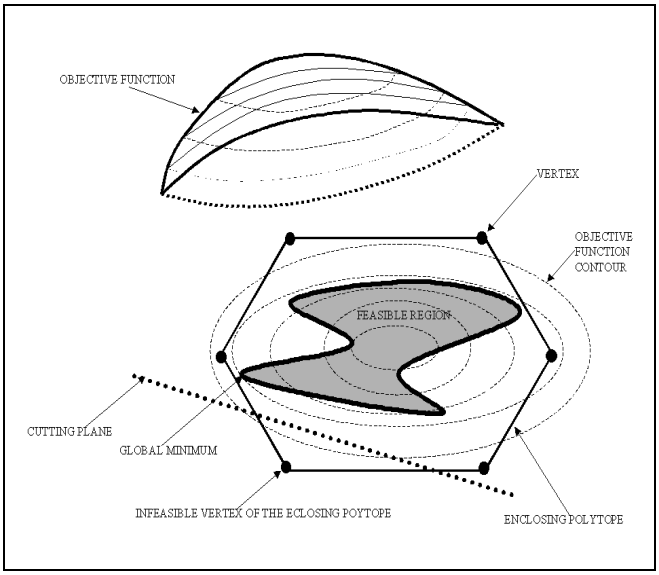

Figure 1. The Concept of the Outer Approximation Method.

This method solved a problem about 15 times faster than MINOS. While it is a very efficient, effective approach, solution speed can be accelerated even more if the simulation model can be accelerated. Between 20 and 100 calls to the simulator is common using this method.

Linear Genetic Programming

LGP is examined for possible use in simulator acceleration. LGP is the direct evolution of binary machine code through the use of evolutionary operators such as crossover and mutation. It searches for the structure of the solution [the computer program] and the constants simutaneously. In effect, it will simulate the simulator. An algorithm that specifically evolves a computer program at the machine code level [Nordin 1994, Nordin & Banzhaf 1995(a&b), Nordin, Francone & Banzhaf, 1996, and Nordin, 1999] was used in this work. An evolved LGP program is a sequence of binary machine instructions. Thus, an evolved LGP program might be comprised of a sequence of four, 32-bit machine instructions. When executed, those four instructions would cause the 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	(1)
register 3 = register 1 - 64	(2)
register 3 = register 2 * register 3	(3)
register 3 = register 2 / register 3	(4)

One of the three hardware registers in this sample LGP program is selected as the output register. Once the output register is selected, a fitness evaluation for this sample LGP program would consist of the following steps:

1. Initialize the hardware registers with the input values for the fitness instance;
2. Execute the above four instruction program {(1)-(4)} on the hardware registers as initialized; and
3. Evaluate the value in the selected output register for fitness against the fitness function.

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 [Nordin & Banzhaf 1995b]. It is this concept, whether this LGP algorithm can evolve a complex function with a high degree of fitness using various register functions that was tested.

Implementation. The process in which evolved programs are produced is the tournament. The LGP 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.

Search Parameters. Various parameters are used to direct and facilitate the search for a good solution. These are described below.

Crossover Rate. 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.

Mutation Rate. Mutation transforms programs in the LGP 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.

Reproduction Rate. 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. 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.
Number of Demes. The number of demes pertains to the how the population of programs is divided.

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 determines 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.

Inter-Deme Migration Rate. This controls the rate at which the percent of tournaments that result in migration of programs between adjacent demes is set. 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.

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.

Selection by Difficulty. The algorithm also keeps track of how difficult the population is at finding a particular training instance. By setting this parameter, solutions that are more general are found to the more difficult portions of the training set.

Stochastic Selection. This parameter is used to select training instances randomly.

Training Subset Change Frequency Equivalents. This parameter determines how frequently the training subset is changed, in generation equivalents. Since the LGP uses the tournament selection criteria, a generation equivalent is one half of the population size.

Linear Genetic Programming Analysis

The LGP technique was previously demonstrated as being resistant to developing false positive relationships [Deschaine, 2000]. The first test for this work was to create a synthetic data set based on widely known and used physical law; Darcy's Law. It is a simple linear equation that describes the flow of water through porous media. The equation is Q=KIA, where Q = flow [L^3/T], K = hydraulic conductivity [L/T], I = gradient [L/L], and A = area [L^2]. The data set was constructed by adding 10% random variation to the inputs in an attempt to confuse the algorithm. The solution (after intron removal, simplification and optimization) is precisely Darcy's Law, represented in ANSI C as:

```
f[0]+= v[2]        (1)
f[0]*= v[0]        (2)
f[0]*= v[1]        (3)
```

The second test was to use a data set that contained both the input [five production related variables] and the output from a complex process simulator [Rice, 2001]. The data set consisted of 7547 solutions generated from various values from the five input variables common to making production decisions.

The LGP analysis was designed to capture the structure of the underlying data set. This was done by randomly dividing the data set into three subsets:

Training:	2506 data points,
Validation:	2520 data points, and;
Applied (or Testing):	2521 data points.

The learning occurred on the training data set. The best evolved programs were selected using the training and validation data set. Whether the "true" structure of the solution was captured is again measured by using the applied data set. In other words, the applied data (also referred to as testing data) played no part in training or in best program selection. Accordingly, the results on the applied data measure how well the evolved solution generalizes to unseen data.

The results were as follows:

FITNESS [R^2]	Single Solution	Team Solution
Training	0.9934	0.9975
Validation	0.9893	0.9939
Applied	0.9783	0.9889

These results were obtained using the LGP system described in [Francone, 2001]. Multiple runs were conducted randomizing the values for population size, maximum program size, maximum number of FPU registers, DSS subset size, percent by difficulty, crossover rate, homologous crossover rate, and mutation rate. The reported results represent the progress at the end of the 218th run [64,290 generation equivalents]. The solution was still improving at this time, albeit slowly.

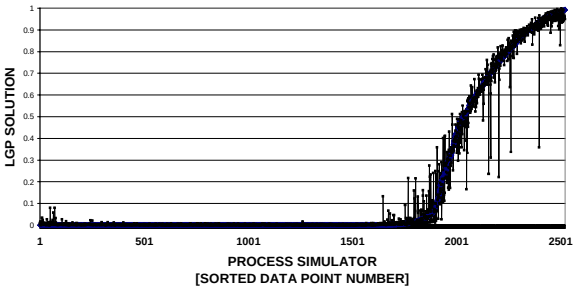

Figure 2. Process Simulator vs. LGP Solution with an R^2 of 0.9889 – Applied Data Set Results.

As can be seen on the figure, for all but six of the 2521 applied data points, the agreement is good, and for most of the data set, the agreement is very good, as evidenced by the high R^2 fitness. A second batch run achieved R^2 fitness on the applied data set of 0.989. The fitness on the training and validation data sets is 0.997. This demonstrates analysis repeatability. For comparison, using a statistical regression approach on this data set yielded an R^2 fitness of 0.80.

Summary and Conclusions

Linear Genetic Programming is demonstrated to be able to represent a complex simulator with a much faster computer code and a high degree of accuracy [R^2 =0.989 to 0.997]. This LGP solution executes on a standard PC in less than a second. The solution is written in assembler, ANSI C and JAVA. Providing a representative function of a simulator that executes with such speed and accuracy opens up the option for not only solving faster optimizations using the Outer Approximation method, but now also genetic algorithms and other evolutionary optimization techniques as well [as discussed in Deschaine, 2001]. Once an optimal solution is determined using an LGP simulated simulator, the solution cam then be input to the full simulator for verification, as warranted.

References

Deschaine, 2000. Deschaine, L. M., Tackling real-world environmental challenges with linear genetic programming, *PCAI magazine*, volume 15, number 5, September/October, 2000, pp. 35-37.

Deschaine, 2001. Deschaine, L. M. McCormack, J., Pyle, D., And Francone, F., Genetic algorithms and intelligent agents team up: techniques for data assembly, preprocessing, modeling and optimizing decisions. PCAI magazine, May June 2001, pp 38-44.

Francone, F., Discipulus Users Manual, Version 3.0. Register Machine Learning Technologies, 2001.

Karatzas 1993. Karatzas, G.P. and Pinder, G.F., Groundwater management using numerical simulation and the Outer Approximation Method for global optimization, WWR, Vol. 29, No. 10, pp. 3371-3378, October 1993.

Karatzas 1996. Karatzas, G.P. and Pinder, G.F. The solution of groundwater quality management problems with a nonconvex feasible region using a cutting plane optimization technique, WWR, Vol. 32, No. 4, pp. 1091-1100, April 1996.

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.

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

Nordin, J.P., Banzhaf, W. 1995b. 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.

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.

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

Rice, 2000. Brian S. Rice and Robert L. Walton of Eastman Kodak Company, Industrial Production Data Set, 2001.