Automatic Synthesis Using Genetic Programming of an Improved General-Purpose Controller for Industrially Representative Plants

Martin A. Keane
Econometrics Inc.
Chicago, Illinois
makeane@ix.netcom.com

John R. Koza
Stanford University
Stanford, California
koza@stanford.edu

Matthew J. Streeter
Genetic Programming Inc.
Mountain View, California
mjs@tmolp.com

Abstract

Most real-world controllers are composed of proportional, integrative, and derivative signal processing blocks. The so-called PID controller was invented and patented by Callender and Stevenson in 1939. In 1942, Ziegler and Nichols developed mathematical rules for automatically selecting the parameter values for PID controllers. In their influential 1995 book, Astrom and Hagglund developed a world-beating PID controller that outperforms the 1942 Ziegler-Nichols rules on an industrially representative set of plants. In this paper, we approached the problem of automatic synthesis of a controller using genetic programming without requiring in advance that the topology of the plant be the conventional PID topology. We present a genetically evolved controller that outperforms the automatic tuning rules developed by Astrom and Hagglund in 1995 for the industrially representative set of plants specified by Astrom and Hagglund.

1 Introduction

Automatic controllers are ubiquitous in the real world (Astrom and Hagglund 1995; Dorf and Bishop 1998; Bryson and Ho 1975; Boyd and Barratt 1991). The purpose of a controller is to force, in a meritorious way, the response of a system (conventionally called the *plant*) to match a desired response (the *reference signal* or *setpoint*). For example, the cruise control device in a car continuously adjusts the engine (the plant) based on the difference between the speed specified by the driver (the *reference signal*) and the car's actual speed (the *plant response*).

The input to a controller typically consists of reference signal(s) and plant response(s). The output of a controller consists of control variable(s) that are passed from the controller to the plant. The individual signal-processing blocks of a controller are coupled to one another in a particular topological arrangement.

Over 90% of all real-world controllers are PID controllers. PID controllers are composed of a proportional (P), an integrative (I), and a derivative (D) block. The PID controller was invented and patented by Albert Callender and Allan Stevenson of Imperial Chemical Limited of Northwich, England in 1939. In 1942, Ziegler and Nichols developed mathematical rules for automatically selecting the parameter values associated with the proportional, integrative, and derivative blocks of a PID controller.

In their influential 1995 book *PID Controllers: Theory, Design, and Tuning*, Astrom and Hagglund (1995) identified four families of plants

> "that are representative for the dynamics of typical industrial processes."

The first of the four families of plants in Astrom and Hagglund 1995 consists of plants represented by transfer functions of the form

$$G(s) = \frac{e^{-s}}{(1+sT)^2} \quad \text{(A)}$$

where $T = 0.1, \dots, 10$.

The second family consists of the n-lag plants represented by transfer functions of the form

$$G(s) = \frac{1}{(1+s)^n} \quad \text{(B)}$$

where $n = 3, 4$, and 8.

The third family consists of plants represented by transfer functions of the form

$$G(s) = \frac{1}{(1+s)(1+\alpha s)(1+\alpha^2 s)(1+\alpha^3 s)} \quad \text{(C)}$$

where $\alpha = 0.2, 0.5$, and 0.7.

The fourth family consists of plants represented by transfer functions of the form

$$G(s) = \frac{1-\alpha s}{(s+1)^3} \quad \text{(D)}$$

where $\alpha = 0.1, 0.2, 0.5, 1.0$, and 2.0.

Astrom and Hagglund developed a method in their 1995 book for automatically tuning PID controllers for all the plants in all four of these industrially representative families of plants. They employed the widely used Ziegler-Nichols rules (Ziegler and Nichols 1942) as a "starting point." The tuning rules developed by Astrom and Hagglund in their 1995 book outperform the widely used Ziegler-Nichols tuning rules on all 16 industrially representative plants used by Astrom and Hagglund. As Astrom-Hagglund observe, "[Our] new methods give substantial improvements in control performance while retaining much of the simplicity of the Ziegler-Nichols rules."

The methods developed by Astrom and Hagglund use several parameters representing the overall characteristics of a plant. These parameters are not, of course, a complete representation of the behavior of the plant. However, these parameters offer the practical advantage of being readily obtainable for real-world plants by means of relatively straightforward testing in the field.

In one version of their method, Astrom and Hagglund use two frequency-domain parameters, namely the ultimate gain, K_u (the minimum value of the gain that must be introduced into the feedback path to cause the system to oscillate) and the ultimate period, T_u (the period of this lowest frequency oscillation). In another version, Astrom and Hagglund use two time-domain parameters, namely the plant's time constant, T_r, and the dead time, L (the period before the plant output begins to respond to a change in the reference signal). Astrom and Hagglund describe a procedure for estimating these parameters from the plant's response to a step input.

Table 2 shows the characteristics of a total of 26 different plants that are discussed in this paper. All 26 plants are members of Astrom and Hagglund's four families of plants. Columns 3, 4, 5, and 6 of the table describe the plants in terms of their ultimate gain, K_u; ultimate period, T_u; dead time, L; and the time constant, T_r; respectively. Column 7 identifies the 16 plants that Astrom and Hagglund included in their test bed. Column 8 refers to two runs described in this paper and indicates the plants that were employed in these two runs.

Figure 3 shows the world-beating solution developed by Astrom and Hagglund in their 1995 book for the 16 industrially representative plants. This controller is a PID controller.

Equation 1 in the figure is

$$0.25 * e^{\frac{0.56}{K_u} + \frac{0.12}{K_u^2}}$$

Equation 2 is

$$0.72 * K_u * e^{\frac{1.6}{K_u} + \frac{1.2}{K_u^2}}$$

Equation 3 is

$$\frac{0.72 * K_u * e^{\frac{1.6}{K_u} + \frac{1.2}{K_u^2}}}{0.59 * T_u * e^{\frac{1.3}{K_u} + \frac{0.38}{K_u^2}}}$$

Equation 4 is

$$0.108 * K_u * T_u * e^{\frac{1.6}{K_u} + \frac{1.2}{K_u^2}} * e^{\frac{1.4}{K_u} + \frac{0.56}{K_u^2}}$$

The tuning developed by Astrom and Hagglund in their 1995 book outperforms the widely used Ziegler-Nichols tuning rules (Ziegler and Nichols 1942) on all 16 industrially representative plants used by Astrom and Hagglund.

The *topology* of a controller entails the specification of

- the total number of processing blocks to be employed in the controller,
- the type of each block (e.g., gain, lead, lag, integrator, differentiator, adder, subtractor), and
- the connections (directed lines) between the input point(s) and the output point of each block in the controller,
- the connections (directed lines) between blocks of the controller and the external input(s) to the controller, and
- the connections (directed lines) between blocks of the controller and the external output(s) of the controller.

Astrom and Hagglund were, of course, seeking a PID controller that would outperform the Ziegler-Nichols rules. They prespecified the PID topology.

The question arises as to whether it is possible to find a yet better controller. An improved controller (if it exists) might possibly be a PID controller. An improved general-purpose controller would possibly employ a novel topology that is different and more complex than the PID controller shown in figure 3.

Genetic programming (Koza, Bennett, Andre, and Keane 1999; Koza, Bennett, Andre, Keane, and Brave 1999) is a method for automatically creating a computer program to solve a problem. Genetic programming has previously been used to automatically create both the topology and parameter values (tuning) for a controller for a particular two-lag plant and a particular three-lag plant (Koza, Keane, Yu, Bennett, and Mydlowec 2000). However, the two (different) evolved controllers in that work applied only to particular plants (each belonging to the same family). Genetic programming has also been used to automatically create both the topology and parameter values of a controller for plants belonging to two families (Keane, Yu, and Koza 2000).

In this paper, we approach the problem of creating an improved controller using genetic programming by building on the 1995 Astrom and Hagglund rules. We do this by inserting a terminal into the terminal set representing the output of the Astrom and

Hagglund 1995 controller (figure 3). This terminal returns the time-domain signal produced by a PID controller tuned using the Astrom and Hagglund rules. That is, the terminal causes a connection to be made to a PID controller that has been tuned for the specific plant under consideration using the current instantiations of the free variables for ultimate gain, K_u, and ultimate period, T_u. The instantiations of the values of K_u and T_u, of course, vary from fitness case to fitness case.

Genetic programming automatically creates its controller using a fitness measure built from the same elements used in Astrom and Hagglund 1995. That is, our fitness measure attempts to optimize for the integral of the time-weighted absolute error (ITAE) for a step input and also to optimize for maximum sensitivity and sensor noise attenuation.

In this paper, we present designs for both the topology and tuning for parameterized general-purpose controllers for industrially representative plants belonging to all four families of plants described by equations (A), (B), (C), and (D). These are the same industrially representative set of plants specified by Astrom and Hagglund.

The automatically designed controllers in this paper outperform the controller designed using the techniques of Astrom and Hagglund using the performance criteria of Astrom and Hagglund.

The automatically designed controller is general in the sense that it contains free variables and therefore provides a solution to an entire category of problems (i.e., all the plants in all four families).

Early controllers were built from mechanical, pneumatic, or analog electrical components (e.g., inductors and capacitors implementing differentiation and integration). Altering the basic topology of such controllers was therefore cumbersome and expensive. However, most present day controllers are electronic devices that are programmed by means of software. This means that a topology that is more complex than a PID topology can be implemented today merely by changing software.

Section 2 of this paper describes the preparatory steps and section 3 presents the results.

2 Preparatory Steps

Six major preparatory steps are required to apply genetic programming to controller synthesis.

2.1 Program Architecture

Since the to-be-synthesized controller has one output (control variable), each program tree in the population has one result-producing branch.

2.2 Terminal Set

The terminal set, T (except for the arithmetic-performing subtrees described below) contains time-domain signals.

T = {REFERENCE_SIGNAL,
CONTROLLER_OUTPUT,
PLANT_OUTPUT, LEFT_1, … , LEFT_4,
RIGHT_1, … , RIGHT_4}.

The REFERENCE_SIGNAL is the time-domain signal representing the desired plant response.

The PLANT_OUTPUT is the time-domain signal representing the plant output.

The CONTROLLER_OUTPUT is the time-domain signal representing the output of the controller (i.e., the control variable).

The eight terminals LEFT_1, … , LEFT_4 and RIGHT_1, … , RIGHT_4 are explained in the next section in conjunction with the TAKEOFF function.

The numerical parameter value for each signal processing block possessing a parameter is established by an arithmetic-performing subtree containing perturbable numerical terminals, arithmetic operations, and the four parameters for representing the overall characteristics of a plant. The value returned by an entire arithmetic-performing subtree is interpreted as a component value lying in a range of (positive values) between 10^{-3} and 10^{3}. The terminal set for the arithmetic-performing subtrees is

T_{aps} = {$\Re$, KU, TU, L, TR}.

where K_u is the ultimate gain; T_u is the ultimate period; T_r is the plant's time constant; and is L the dead time.

Here $\Re$ denotes a perturbable numerical value. In the initial random generation (generation 0) of a run, each perturbable numerical value is set, individually and separately, to a random value in a chosen range (from -3.0 and +3.0 here). In later generations, a perturbable numerical value may be changed by adding or subtracting a relatively small number drawn from a Gaussian probability distribution (with standard deviation of 1.0).

A constrained syntactic structure maintains one function and terminal set for the arithmetic-performing subtrees and a different function and terminal set (below) for the rest of the program tree.

2.3 Function Set

The function set, F (except for the arithmetic-performing subtrees described below) contains continuous-time signal processing functions.

F = {GAIN, INTEGRATOR, DIFFERENTIATOR,
DIFFERENTIAL_INPUT_
INTEGRATOR, LEAD, LAG, LAG2,

INVERTER, ADD_SIGNAL, SUB_SIGNAL, MUL_SIGNAL, DIV_SIGNAL, ADD_3_SIGNAL, TAKEOFF}.

The two-argument GAIN function multiplies the time-domain signal represented by its first argument by a constant numerical value represented by its second argument. This numerical value is constant in the sense that it is not a time-domain signal (like the first argument) and in the sense that this numerical value does not vary when the controller is operating.

The one-argument INTEGRATOR function integrates the time-domain signal represented by its one argument. That is, it applies the transfer function $1/s$, where s is the Laplace transform variable.

The one-argument DIFFERENTIATOR function differentiates the time-domain signal represented by its argument. That is, it applies the transfer function s.

The two-argument DIFFERENTIAL_INPUT_ INTEGRATOR function integrates the time-domain signal representing the difference between its two arguments.

The two-argument LEAD function applies the transfer function $1 + \tau s$, where τ is a real-valued numerical parameter. The first argument is the time-domain input signal. The second argument, τ, is a numerical parameter representing the time constant (usually expressed in seconds) of the LEAD.

The two-argument LAG function applies the transfer function $1/(1 + \tau s)$, where τ is a numerical parameter. The first argument is the time-domain input signal while the second, τ, is the time constant.

The three-argument LAG2 function applies the transfer function

$$\frac{\omega_0^2}{s^2 + 2\zeta\omega_0 s + \omega_0^2},$$

where ζ is the damping ratio, and ω_0 is the corner frequency.

The one-argument INVERTER function negates the time-domain signal represented by its argument.

The two-argument ADD_SIGNAL, SUB_SIGNAL, and MULT_SIGNAL functions perform addition, subtraction, and multiplication, respectively, on the two time-domain signals represented by their two arguments. Note that the MULT_SIGNAL function differs from the GAIN function (described earlier) in that the second argument of a GAIN function is a constant numerical value (i.e., not a time-domain signal) whose value does not vary when the controller is operating.

The three-argument ADD_3_SIGNAL adds three time-domain signals.

For additional details on these functions and other preparatory steps, see Keane, Yu, and Koza 2000, Koza, Keane, Yu, Bennett, and Mydlowec 2000, and Koza, Bennett, Andre, and Keane 1999.

The one-argument function into the function set called TAKEOFF function acts as an identity function in that it returns the value of its argument. At the same time, it stores the value of its argument for possible future use. Whenever one of the four terminals (LEFT_1, … , LEFT_4) is encountered, it returns the value stored by the first, second, third, or fourth TAKEOFF function (if any), respectively, occurring earlier (i.e., to the left) in the overall program tree. For this purpose, the earlier TAKEOFF function is determined on the basis of the usual depth-first order of evaluation from the left used in the LISP programming language. If there is no such stored value, the terminal defaults to the value of the root of the tree (i.e., the control variable). The value returned by RIGHT_1, … , RIGHT_4 is based on occurrences of the TAKEOFF function that occur later (i.e., to the right) in the overall program tree.

The function set, F_{aps}, for the arithmetic-performing subtrees is

F_{aps} = {ADD_NUMERIC, SUB_NUMERIC, MUL_NUMERIC, DIV_NUMERIC, REXP, RLOG, POW}.

The first three functions perform addition, subtraction, and multiplication of numerical values. The two-argument DIV_NUMERIC function divides the first argument by the second argument, except that the quotient is never allowed to exceed 10^5. The one-argument REXP function is the exponential function and the one-argument RLOG function is the natural logarithm of the absolute value. The two-argument POW function returns the value of its first argument raised to the power of the value of its second argument.

2.4 Fitness Measure

Genetic programming is a probabilistic algorithm that searches the space of compositions of the available functions and terminals under the guidance of a fitness measure (Koza, Bennett, Andre, and Keane 1999; Koza, Bennett, Andre, Keane, and Brave 1999). The fitness measure is a mathematical implementation of the problem's high-level requirements. The fitness measure is couched in terms of “what needs to be done.” The fitness measure for most problems of controller design is multi-objective in the sense that there are several different (usually conflicting) requirements for the controller.

The fitness of each individual in the population is determined by executing the program tree. The execution of the program tree produces an interconnected sequence of signal processing blocks

— that is, a block diagram for the individual controller. The controller is embedded into a framework containing the (fixed) plant and the (fixed) external feedback loop. A SPICE netlist is then constructed to represent the block diagram of the controller, the (fixed) plant, and the (fixed) external feedback loop. This SPICE netlist is wrapped inside an appropriate set of SPICE commands to carry out various analyses in the time domain and in the frequency domain (described below). We also provide SPICE with subcircuit definitions to implement all the signal processing functions in the function set (described above) and all the signal processing functions necessary to represent the plant. The controller is then simulated using our modified version of the original 217,000-line SPICE3 simulator (Quarles, Newton, Pederson, and Sangiovanni-Vincentelli 1994). Our modified version of SPICE is run as a submodule within our genetic programming system. The SPICE simulator returns tabular output (representing the plant output in the time domain). An interface communicates this information to our genetic programming code. See Koza, Keane, Yu, Bennett, and Mydlowec 2000 for details.

The fitness of each controller in the population is measured by means of eight separate invocations of the SPICE simulator for each plant. This fitness measure attempts to optimize step response and disturbance rejection while simultaneously imposing constraints on maximum sensitivity and sensor noise attenuation. The fitness of an individual controller is the sum of the detrimental contributions from each of these elements of the fitness measure. The smaller the sum, the better.

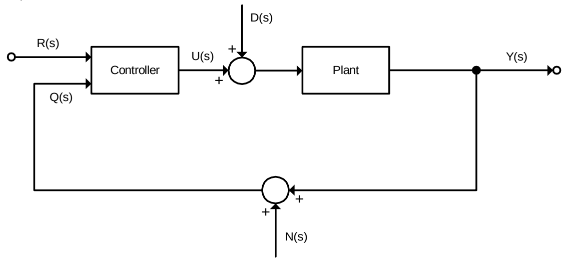

Figure 1 Overall model.

Figure 1 presents a model for the entire system containing the given plant and the to-be-evolved controller. In this figure, $R(s)$ is the reference signal; $Y(s)$ is the plant output; and $U(s)$ is the controller's output (control variable). Disturbance $D(s)$ may be added to the controller's output $U(s)$. Sensor noise $N(s)$ may be added to the plant's output $Y(s)$ yielding $Q(s)$. Here $N(s)$ is an AC signal.

The first six elements of the fitness measure for each plant are time-domain-based elements. These elements represent six choices of values for the height of the reference signal and disturbance signal (shown in table 1). The reference signal is step function that rises from 0 at time $t = 0$ to the specified height at $t = 1$ millisecond. The disturbance signal is a step function that rises from 0 at time $t = 10T_u$ to the specified height at $t = 10T_u + 1$ millisecond. The disturbance signal is added to the controller's output.

Table 1 Six combinations

Reference signal	Disturbance signal
1.0	1.0
10^{-3}	10^{-3}
-10^{-6}	10^{-6}
1.0	-0.6
-1.0	0.0
0.0	1.0

For each of these first six elements of the fitness measure, a transient analysis is performed in the time domain using the SPICE simulator. The function $e(t)$ is the difference (error) at time t between the plant output and the reference signal. The contribution to fitness is based on the sum of two integrals of time-weighted absolute error (ITAE). The first term of the integral accounts for the controller's step response while the second term accounts for disturbance rejection.

$$\frac{\int_{t=0}^{10T_u} t|e(t)|B\,dt}{T_u^2} + \frac{\int_{t=10T_u}^{20T_u} (t-10T_u)|e(t)|C\,dt}{T_u^2}.$$

The factor B in the first term of the integral multiplies each value of $e(t)$ by the reciprocal of the amplitude of the reference signal (so that all reference signals are equally influential). The factor C in the second term of the integral multiplies value of $e(t)$ by the reciprocal of the amplitude of the disturbance signals. When the amplitude of either the reference signal or the disturbance signal is zero, the appropriate factor (B or C) is set to zero. The ITAE component of fitness is such that, all other things being equal, changing the time scale by a factor of F changes the ITAE by F^2. The division of the integral by T_u^2 is an attempt to eliminate this artifact of the time scale and equalize the influence of each of the plants in the overall fitness measure. For these elements of the fitness measure, the contribution to fitness is multiplied by 20 if the element is greater than for the Astrom and Hagglund (1995).

The seventh element of the fitness measure for each plant is a frequency-domain-based element. An AC sweep is performed using the SPICE simulator from $1/(1000T_u)$ to $1000/T_u$ while holding the reference signal $R(s)$ and the disturbance signal $D(s)$ at zero. The maximum sensitivity, M_s, is a measure of the stability margin. It is desirable to minimize the maximum sensitivity (and therefore maximize the stability margin). The quantity $1/M_s$ is the minimum distance between the Nyquist plot and the point (-1,0) and is the stability margin incorporating both gain and phase margin. The maximum sensitivity is the maximum amplitude of $Q(s)$. The contribution to fitness is 0 if $M_s < 1.5$; $2(M_s - 1.5)$ for $1.5 \le M_s \le 2.0$; and $20(M_s - 2.0) + 1$ for $M_s > 2.0$. For these elements of the fitness measure (as well as the elements below), the contribution to fitness is multiplied by 10 if the element is greater than for the Astrom and Hagglund controller (1995).

The eighth element of the fitness measure for each plant is a frequency-domain-based element measuring sensor noise attenuation. Achieving favorable sensor noise attenuation is often in direct conflict with the goal of achieving a rapid response to setpoint changes and rejection of plant disturbances. An AC sweep is performed using the SPICE simulator from $10/T_u$ to $1000/T_u$ while holding both the reference signal $R(s)$ and the disturbance signal $D(s)$ at zero. The attenuation of the sensor noise is measured at plant output at $Y(s)$. $A_{\min}$ is the minimum attenuation in decibels within this frequency range. It is desirable to maximize the minimum attenuation. The contribution to fitness for sensor noise attenuation is 0 if $A_{\min} > 40$ dB; $(40 - A_{\min})/10$ if 20 dB $\le A_{\min} \le$ 40 dB; and $2 + (20 - A_{\min})$ if $A_{\min} < 20$ dB.

A controller that cannot be simulated by SPICE is assigned a high penalty value of fitness (10^8).

2.5 Control Parameters

The population size, M, was 100,000. A (generous) maximum size of 500 points (for functions and terminals) was established for the single result-producing branch. The percentages of the genetic operations for each generation are 63% one-offspring crossover on internal points of the program tree, 7% one-offspring crossover on terminals, 1% subtree mutation, 20% mutation on numerical constant terminals, and 9% reproduction. Other parameters are the same as used before (Koza, Bennett, Andre, Keane 1999).

2.6 Termination

The run was manually monitored and manually terminated when the fitness of many successive best-of-generation individuals appeared to have reached a plateau. The best-so-far individual was harvested.

2.7 Parallel Implementation

This problem was run on a home-built Beowulf-style (Sterling, Salmon, Becker, and Savarese 1999) parallel cluster computer system consisting of 1,000 350 MHz Pentium II processors (each accompanied by 64 megabytes of RAM). The system has a 350 MHz Pentium II computer as host. The processing nodes are connected with a 100 megabit-per-second Ethernet. The processing nodes and the host use the Linux operating system. The distributed genetic algorithm with unsynchronized generations and semi-isolated subpopulations was used with a subpopulation size of $Q = 100$ at each of $D = 1{,}000$ demes. Two processors are housed in each of the 500 physical boxes of the system. As each processor (asynchronously) completes a generation, four boatloads of emigrants from each subpopulation (selected probabilistically based on fitness) are dispatched to each of the four toroidally adjacent processors. The migration rate is 2% (but 10% if the toroidally adjacent node is in the same physical box).

3 Results

3.1 First Run

The fitness measure for the first run entailed eight separate invocations of the SPICE simulator for each of 20 plants (for a total of 160 separate invocations of the SPICE simulator for each individual). The average total time to evaluate the fitness of each individual was 2 minutes and 10 seconds. The population size at each node of the 1,000-Pentium Beowulf-style parallel computer was 100 (for a total population size of 100,000). It took 320 hours (13.3 days) to reach generation 88 and produce a best-of-run individual (figure 4) that equals or outperforms the Astrom and Hagglund controller on 100% of the fitness cases. Equation 1 in figure 4 is

$$10^{e^{\log(\left|\log(e^{K_u * L})/L\right|)}}$$

Equation 2 is

$$\frac{1}{1 + T_r * s}$$

Equation 3 is

$$10^{e^{\log(\log(K_u * L))}}$$

Equation 4 is

$$e^{\log(K_u / L)}$$

Equation 5 is

$$\frac{1}{1 + T_r * s}$$

Equation 6 is

$$10^{e^{\log(\log(K_u * L))}}$$

Equation 7 is

$$e^{\log(K_u)}$$

Equation 8 is

$$\frac{1}{1+T_r * s}$$

Averaged over 20 plants from all four families of plants, the best-of-run genetically evolved parameterized general-purpose controller from generation 88 of the first run has

- 78.6% of the ITAE of the Astrom-Hagglund controller,
- 59.1% of the stability penalty involving maximum sensitivity, M_s, of the Astrom-Hagglund controller, and
- 81.1% of the sensitivity penalty involving sensor noise attenuation of the Astrom-Hagglund controller.

Figure 2 compares the time-domain response of the best-of-run genetically evolved parameterized general-purpose controller from generation 88 of the first run (solid line) and the Astrom and Hagglund controller (dotted line) for the three-lag plant (one of the 20 plants) with a 1 volt reference signal. As can be seen, the Astrom and Hagglund controller overshoots the target value and takes longer to settle than the best-of-run controller from generation 88.

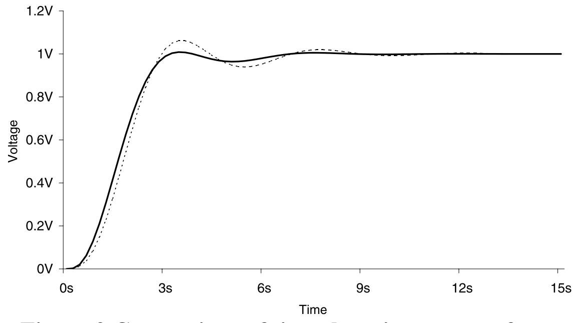

Figure 2 Comparison of time-domain response for the best-of-run controller from generation 88 of the first run (solid line) and the Astrom and Hagglund controller (dotted line) for the three-lag plant with a 1-volt reference signal.

3.2 Second Run

This second run was based on 24 plants. In addition, this run further differed from the first run in that, after all fitness cases first outperformed or equaled the Astrom and Hagglund controller, the fitness measure began to take parsimony into account.

Figure 5 shows the best-of-run genetically evolved parameterized general-purpose controller from generation 38 of the second run. It equals or outperforms the Astrom and Hagglund controller on 100% of the fitness cases.

Equation 1 in figure 5 is

$$\log\left|2T_r + K_u{}^L\right|$$

Equation 2 is

$$\log(\mathrm{abs}(T_r + K_u \wedge {}^{K_u \wedge \frac{K_u \wedge \left(\frac{0.68631}{\frac{1}{\frac{0.13031}{\frac{T_u}{\log\left|\log\left|T_r+K_u^{\left(K_u^L\right)}\right|\right|}-\left(K_u^L\right)^L-T_u}}-T_r^L-T_u}\right)}{T_r+K_u\wedge^{\frac{1}{\frac{0.69897}{T_r+K_u^L}}-T_r^L-T_u}}}))$$

Equation 3 is

$$\log\left|2T_r + K_u^{\log\left|T_r+K_u^L\right|}\right|$$

Equation 4 is

$$\log\left|T_r + K_u{}^L\right|$$

Equation 5 is

$$\log\left|T_r + \log\left(T_r + 1.2784\right)\right|$$

Equation 6 is

$$\log\left|T_r + \left(T_r + (x)^{\log\left|\log\left|K_u \wedge L\right|\right|}\right)^L\right|$$

where

$$x = T_r + K_u^{\log\left|\log\left|T_r+K_u^L\right|\right|}$$

Equation 7 is

$$\log\left|T_r + \left(T_r + K_u{}^L\right)^L\right|$$

Equation 8 is

$$\log\left|2T_r + K_u{}^L\right|$$

Averaged over all 24 plants from all four families of plants, the best-of-run genetically evolved parameterized general-purpose controller from generation 38 of the second run has

- 98.6% of the ITAE (integral of time-weighted absolute error) of the Astrom-Hagglund controller,

- 78.1% of the stability penalty involving maximum sensitivity, M_s, of the Astrom-Hagglund controller, and
- 96.1% of the sensitivity penalty involving sensor noise attenuation of the Astrom-Hagglund controller.

The second run took 397 hours (16.5 days) to produce a best-of-run individual that equals or outperforms the Astrom and Hagglund controller on 100% of the 24 plants. Unfortunately, this run was interrupted by a multi-hour power outage a few hours after the fitness measure started considering parsimony. Therefore, this second run primarily demonstrates that genetic programming is capable of solving this problem repeatedly and in slightly different ways.

3.3 Cross-Validation

Both of the best-of-run controllers for the two runs described above were cross-validated using 18 additional previously unseen plants (six plants from each of families A, C, and D). We did not create any out-of-sample plants for family B since for this family all of the possible integer values within the range specified by Astrom and Hagglund (1995) were already covered by plants used in our fitness evaluation. The parameter values associated with the out-of-sample plants were spread approximately evenly across the range of possible parameter values used by Astrom and Hagglund (1995). For each of the 18 plants, we computed the values of eight fitness cases, as described above.

The best-of-run controller from the first run outperforms the Astrom and Hagglund controller in 142 out of the 144 new fitness cases and has 82.9% of the average ITAE of the Astrom and Hagglund controller. The best-of-run controller from the second run outperforms the Astrom and Hagglund controller in 143 out of 144 cases, and has 91.7% of the average ITAE of the Astrom and Hagglund controller. The few cases in which the evolved controllers do not do better than the Astrom and Hagglund controller involve measures of either stability or sensitivity. In each of these cases, the performance of the evolved controller, although worse than that of the Astrom and Hagglund controller, is still acceptable in terms of real world control applications.

4 Conclusion

We presented a genetically evolved controller that outperforms the automatic tuning rules developed by Astrom and Hagglund in 1995 for an industrially representative set of plants The genetically evolved controller employs a novel topology that differs from the PID topology commonly used for practical controllers.

5 Future Work

The authors are actively continuing to work in this area and will report additional results and analysis in their forthcoming book *Genetic Programming IV.*

References

Astrom, Karl J. and Hagglund, Tore. 1995. *PID Controllers: Theory, Design, and Tuning.* Second Edition. Research Triangle Park, NC: Instrument Society of America.

Boyd, S. P. and Barratt, C. H. 1991. *Linear Controller Design: Limits of Performance.* Englewood Cliffs, NJ: Prentice Hall.

Bryson, Arthur E., and Ho, Yu-Chi. 1975. *Applied Optimal Control.* New York: Hemisphere Publishing.

Callender, Albert and Stevenson, Allan Brown. 1939. *Automatic Control of Variable Physical Characteristics*. United States Patent 2,175,985. Filed February 17, 1936 in United States. Filed February 13, 1935 in Great Britain. Issued October 10, 1939 in United States.

Dorf, Richard C. and Bishop, Robert H. 1998. *Modern Control Systems*. Eighth edition. Menlo Park, CA: Addison-Wesley.

Keane, Martin A., Yu, Jessen, and Koza, John R. 2000. Automatic synthesis of both the topology and tuning of a common parameterized controller for two families of plants using genetic programming. In Whitley, Darrell, Goldberg, David, Cantu-Paz, Erick, Spector, Lee, Parmee, Ian, and Beyer, Hans-Georg (editors). *GECCO-2000: Proceedings of the Genetic and Evolutionary Computation Conference, July 10 - 12, 2000, Las Vegas, Nevada*. San Francisco: Morgan Kaufmann Publishers. Pages 496 - 504.

Koza, John R., Bennett III, Forrest H, Andre, David, and Keane, Martin A. 1999. *Genetic Programming III: Darwinian Invention and Problem Solving.* San Francisco, CA: Morgan Kaufmann.

Koza, John R., Bennett III, Forrest H, Andre, David, Keane, Martin A., and Brave, Scott. 1999. *Genetic Programming III Videotape: Human-Competitive Machine Intelligence*. San Francisco, CA: Morgan Kaufmann.

Koza, John R., Keane, Martin A., Yu, Jessen, Bennett, Forrest H III, and Mydlowec, William. 2000. Automatic creation of human-competitive programs and controllers by means of genetic programming. *Genetic Programming and Evolvable Machines*. (1) 121 - 164.

Quarles, Thomas, Newton, A. R., Pederson, D. O., and Sangiovanni-Vincentelli, A. 1994. *SPICE 3 Version 3F5 User's Manual*. Department of Electrical Engineering and Computer Science, University of California. Berkeley, CA. March 1994.

Sterling, Thomas L., Salmon, John, Becker, Donald J., and Savarese, Daniel F. 1999. *How to Build a Beowulf: A Guide to Implementation and Application of PC Clusters*. Cambridge, MA: MIT Press.

Ziegler, J. G. and Nichols, N. B. 1942. Optimum settings for automatic controllers. *Transactions of ASME*. (64) 759-768.

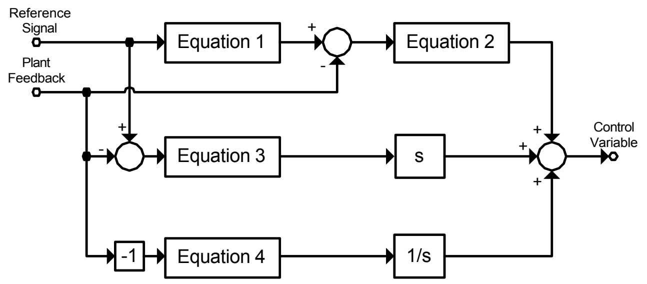

Figure 3 PID controller developed by Astrom and Hagglund (1995) for 16 plants from four industrially representative families of plants.

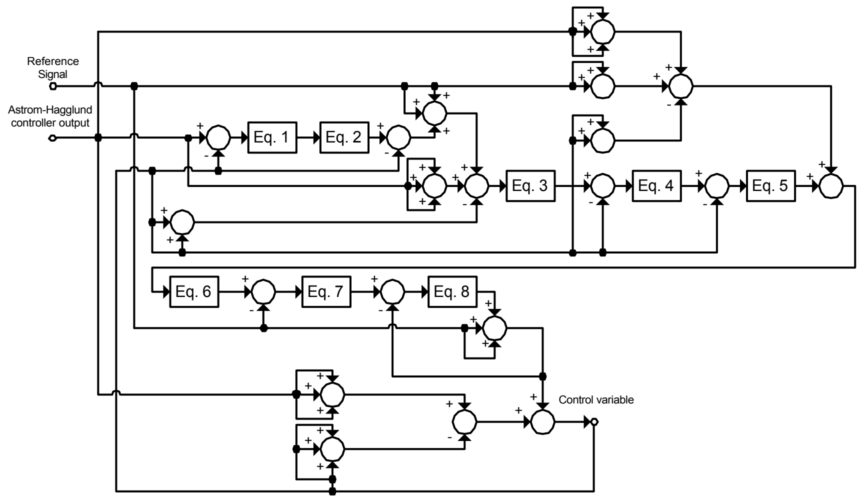

Figure 4 Best-of-run evolved parameterized general-purpose controller from generation 88 of the first run. Genetic programming produced this controller's overall topology consisting of 20 addition (or subtraction) blocks as well as the eight mathematical expressions containing free variables.

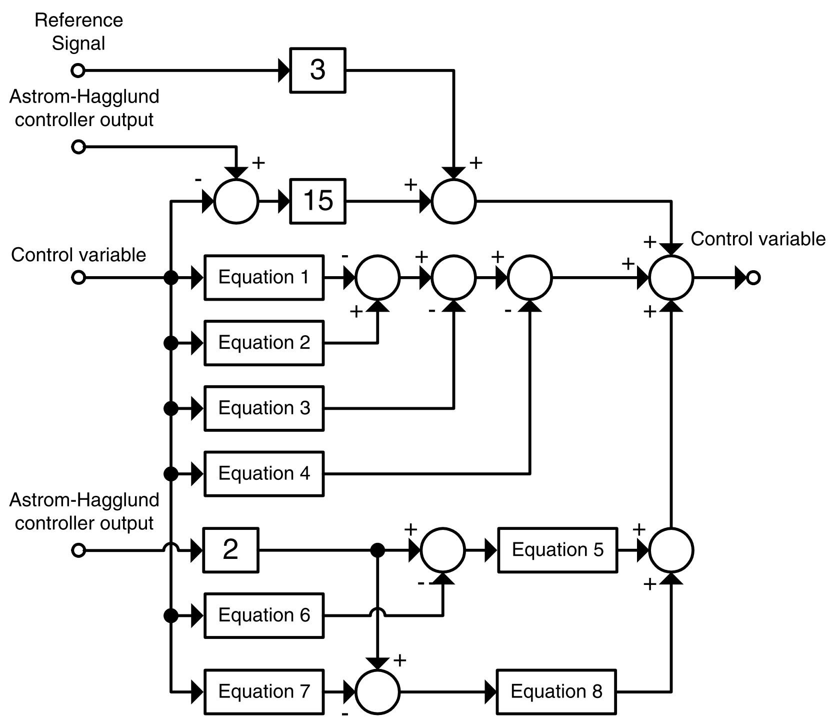

Figure 5 Best-of-run evolved parameterized general-purpose controller from generation 38 of the second run.

Table 2 Characteristics of the 26 plants

Family	Parameter value	K_u	T_u	L	T_r	16 plant set	Runs
A	$T = 0.1$	1.07	2.37	1.00	0.10	X	1,2
A	$T = 0.3$	1.40	3.07	1.01	0.29	X	1,2
A	$T = 1$	2.74	4.85	1.00	1.00	X	1,2
A	$T = 3$	6.80	7.87	1.02	2.99	X	1,2
A	$T = 6$	12.7	11.1	1.00	6.00	X	1,2
A	$T = 10$	20.8	14.2	0.92	10.1	X	2
B	$n = 3$	8.08	3.62	0.52	1.24	X	1,2
B	$n = 4$	4.04	6.27	1.13	1.44	X	1,2
B	$n = 5$	2.95	8.62	1.79	1.61		2
B	$n = 6$	2.39	10.9	2.45	1.78		1,2
B	$n = 7$	2.09	13.0	3.17	1.92		1,2
B	$n = 8$	1.89	15.2	3.88	2.06	X	1,2
C	$\alpha = 0.1$	113	0.198	-0.244	0.674		1
C	$\alpha = 0.2$	30.8	0.56	-0.14	0.69	X	1,2
C	$\alpha = 0.3$	15.0	1.04	-0.02	0.72		2
C	$\alpha = 0.4$	9.62	1.59	0.11	0.76		2
C	$\alpha = 0.5$	6.85	2.23	0.27	0.80	X	1,2
C	$\alpha = 0.6$	5.41	2.92	0.43	0.87		2
C	$\alpha = 0.7$	4.68	3.67	0.60	0.96	X	1,2
C	$\alpha = 0.9$	4.18	5.31	3.44	0.685		1
D	$\alpha = 0.1$	6.21	4.06	0.64	1.22	X	1,2
D	$\alpha = 0.2$	5.03	4.44	0.74	1.23	X	1,2
D	$\alpha = 0.5$	3.23	5.35	1.15	1.17	X	1,2
D	$\alpha = 0.7$	2.59	5.81	1.38	1.16		2
D	$\alpha = 1$	2.02	6.30	1.85	1.07	X	1,2
D	$\alpha = 2$	1.15	7.46	3.46	0.765	X	1,2