Evolution of a Controller with a Free Variable Using Genetic Programming

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

Jessen Yu
Genetic Programming Inc., Los Altos, California
jyu@cs.stanford.edu

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

William Mydlowec
Genetic Programming Inc., Los Altos, California
myd@cs.stanford.edu

ABSTRACT

A mathematical formula containing one or more free variables is "general" in the sense that it provides a solution to an entire category of problems. For example, the familiar formula for solving a quadratic equation contains free variables representing the equation's coefficients. Previous work has demonstrated that genetic programming can automatically synthesize the design for a controller consisting of a topological arrangement of signal processing blocks (such as integrators, differentiators, leads, lags, gains, adders, inverters, and multipliers), where each block is further specified ("tuned") by a numerical component value, and where the evolved controller satisfies user-specified requirements. The question arises as to whether it is possible to use genetic programming to automatically create a "generalized" controller for an entire category of such controller design problems — instead of a single instance of the problem. This paper shows, for an illustrative problem, how genetic programming can be used to create the design for both the topology and tuning of controller, where the controller contains a free variable.

1 Introduction

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

R. Poli et al. (Eds.): EuroGP 2000, LNCS 1802, pp. 91–105, 2000.

The measure of merit for controllers typically incorporate a variety of interacting and conflicting considerations. Examples are the time required to force the plant response to reach a desired value, the degree of success in avoiding significant overshoot beyond the desired value, the time required for the plant to settle, the ability of the controller to operate robustly in the face of minor variations in the actual characteristics of the plant, the ability of the controller to operate robustly in the face of disturbances that may be introduced between the controller and the plant's input, the ability of the controller to operate robustly in the face of sensor noise that may be added to the plant output, the controller's compliance with various frequency domain constraints (e.g., bandwidth), and the controller's compliance with constraints on the magnitude of the control variable or the plant's internal state variables.

Controllers are often represented by block diagrams. A block diagram is a graphical structure containing (1) directed lines representing the (one-directional) flow of time-domain signals within the controller, (2) time-domain signal processing blocks (e.g., integrator, differentiator, lead, lag, gain, adder, inverter, and multiplier), (3) external input points from which the controller receives signals (e.g., the reference signal and the plant output that is externally fed back from the plant to the controller), and (4) output point(s) for the controller, conventionally called the *control variable*). Some signal processing blocks have multiple inputs (e.g., an adder), but they all have exactly one output. Because of this restriction, block diagrams for controllers usually contain *takeoff points* that enable the output of a block to be disseminated to more than one other point in the block diagram. Many blocks used in controllers (e.g., gain, lead, lag, delay) possess numerical parameters. The determination of these values is called *tuning*. Block diagrams sometimes also contain internal feedback (internal loops in the controller) or feedback of the controller output directly into the controller.

The design (synthesis) of a controller entails specification of both the topology and parameter values (tuning) for the block diagram of the controller such that the controller satisfies user-specified high-level design requirements. Specifically, the design process for a controller entails making decisions concerning the total number of signal processing blocks to be employed in the controller, the type of each block (e.g., integrator, differentiator, lead, lag, gain, adder, inverter, and multiplier), the interconnections between the inputs and outputs of each signal processing block and between the controller's external input and external output points, and the values of all required numerical parameters for the signal processing blocks.

Genetic programming has recently been used to create a block diagram for satisfactory controller for a particular two-lag plant and a particular three-lag plant (Koza, Keane, Yu, Bennett, and Mydlowec 2000). Both of these genetically evolved controllers outperformed the controllers designed by experts in the field of control using the criteria originally specified by the experts. However, both of these controllers were for plants having lags with a particular value of the time constant, τ.

A mathematical formula containing one or more free variables is "general" in the sense that it provides a solution to an entire category of problems. For example, the familiar formula for solving a quadratic equation contains free variables representing the coefficients of the equation. The question arises as to whether it is possible to evolve a controller for an *entire category* of plants — instead of one particular plant. In other words, is it possible to design a "generalized" controller containing one or more free variables. The fact that genetic programming permits the evolution of mathematical expressions containing free variables suggests that this might be