Toward an Automated Design Method for Multi-Domain Dynamic Systems Using Bond Graphs and Genetic Programming

Kisung Seo*, Zhun Fan*, Jianjun Hu*, Erik D. Goodman*, Ronald C. Rosenberg+

*Genetic Algorithms Research and Applications Group (GARAGe), +Department of Mechanical Engineering, Michigan State University

Abstract

This paper describes a unified and automated design method for synthesizing designs for multi-domain systems, such as mechatronic systems. A multi-domain dynamic system, of which a mechatronic system is a popular example, includes a combination of components drawn from electrical, mechanical, hydraulic, pneumatic, and/or thermal systems, making it difficult to design a system to meet specified performance goals with a single design tool. The multi-domain design approach is not only efficient for mixed-domain problems, but is also useful for addressing separate single-domain design problems with a single tool. Bond graphs are domain independent, allow free composition, and allow efficient classification and analysis of models, permitting rapid determination of various types of acceptability or feasibility of candidate designs. This can sharply reduce the time needed for analysis of designs that are infeasible or otherwise unattractive. Genetic programming is well recognized as a powerful tool for topologically open-ended search. The combination of these two methods is therefore an appropriate target for a better system for synthesis of complex multi-domain systems. The approach described here will evolve new designs (represented as bond graphs) with ever-improving performance, in an iterative loop of synthesis, analysis, and feedback to the synthesis process. The suggested design methodology has been applied here to three design examples. The first is domain independent – an eigenvalues-placement design problem which is tested for some sample target sets of eigenvalues. The second is in the electrical domain – namely, design of analog filters to achieve specified performance over a given frequency range. The third is in the mechanical domain – design of an electric typewriter drive system to obtain the required steady state rotational position of the printing head for each character to be printed.

1. Introduction

A multi-domain dynamic system design differs from conventional design of electronic circuits, mechanical systems, and fluid power systems in part because of the need to integrate several types of energy behavior as part of the basic design (Coelingh [1]). Multi-domain design is difficult because such systems tend to be complex and most current simulation tools operate over only a single domain. In order to automate design of multi-domain systems, such as mechatronic systems, a new approach is required. The goal of the work reported in this paper is to develop a unified and automated procedure capable of designing mechatronic systems to meet given performance specifications, subject to various constraints. The most difficult aspect of the research is to develop a single tool that can explore the design space in a topologically open-ended manner, yet find appropriate configurations efficiently enough to be useful. Our approach combines bond graphs for representing the mechatronic system models with genetic programming as a means of exploring the design space.

Bond graphs (Karnopp *et al*. [2], Rosenberg [3-6]) allow us to capture the energy behavior underlying the physical aspects (as opposed to the information aspects) of mechatronic systems in a uniformly effective way across domains. This enables the analysis of multi-energy-domain systems with a unified inter-domain tool. Being topological structures, bond graphs are also ideal for representing a structured design space for open-ended generation and exploration. Finally, they allow efficient and rapid evaluation of individual designs, using a two-stage procedure – causal analysis of the graph, followed only if needed by appropriate detailed calculation using a derived state model.

Sharpe and Bracewell [7] present the use of bond graph reasoning for the design of interdisciplinary schemes. They describe how conceptual scheme synthesis may be assisted and structured by the use of functions-mean trees developed by the application of bond-graph-inspired rules. Coelingh *et al*. [1] present a computer-based

design tool for conceptual design of mechatronic motion systems that addresses these problems and that will give a feasible design proposal that is likely to meet the desired performance [BUT???]. Youcef-Toumi [8] introduces an algorithm which identifies automatically the physical components and/or subsystems that are responsible for zero dynamics. Redfield [9] demonstrates the value of using bond graphs as a conceptual or configurational design tool for dynamic systems, using as an example a continuously variable transmission. Tay, Flowers and Barrus [10] use a genetic algorithm to vary bond graph models. Their approach adopts a variational design method, which means they make a complete bond graph model first, then change the bond graph topologically using a GA, yielding new design alternatives. Their goal is to provide a wider range of possible designs, and is closely related to that presented here, but within a topologically more limited search space.

Genetic programming is an effective way to generate design candidates in an open-ended, but statistically structured, manner. A critical aspect of the procedure is a fitness measure, which must guide the evolution of candidate designs toward a suitable result in a reasonable time. There have been a number of research efforts aimed at exploring the combination of genetic programming with physical modeling to find good engineering designs. Perhaps most notable is the work of Koza *et al.* [11-13]. He presents a single uniform approach using genetic programming for the automatic synthesis of both the topology and sizing of a suite of various prototypical analog circuits, including low-pass filters, operational amplifiers and controllers. This system has already shown itself to be extremely promising, having produced a number of patentable designs for useful artifacts, and is the most closely related approach to that proposed here; however, it works in a single energy domain.

Our approach seeks to combine all the potential advantages of both bond graph and genetic programming to allow automated, multi-domain, topologically open-ended design. As our first class of design problems we chose one in which the objective is to realize a design having a specified set of eigenvalues. Since the problem can be studied effectively using linear components with constant parameters, we only needed to introduce one-port (generalized) resistance, capacitance, and inductance elements in our designs. Section 2 discusses the inter-domain nature, efficient evaluation and graphical generation of bond graphs. Section 3 describes evolution of bond graphs by genetic programming. Section 4 presents the eigenvalue design problem; Section 5 presents a filter design example; and Section 6 discusses the typewriter drive mechanism, followed by conclusions in Section 7.

2. Design Domain and Methodology

2.1 Multi-Domain Dynamic Systems

Multi-domain system design differs from conventional design of electronic circuits, mechanical systems, and fluid power systems in part because of the need to integrate several types of energy behavior as part of the basic design. For example, in addition to appropriate "drivers" (sources), lumped-parameter dynamical mechanical systems models typically include at least masses, springs and dampers (Figure 1 a)) while "RLC" electric circuits include resistors, inductors and capacitors (Figure 1 b)). Figure 2 shows a drive system for a typewriter that involves the drive shaft and the load, with important physical properties modeled. The input is the driving torque, T_d., generated through the belt coupling back to the motor.

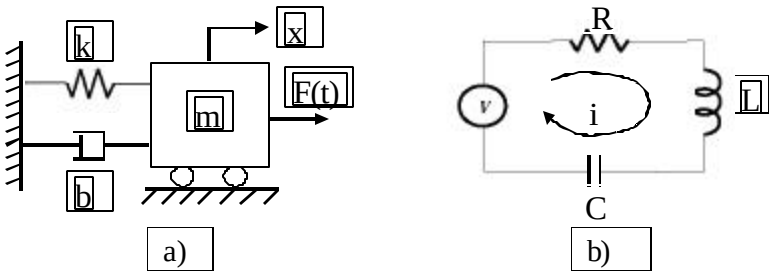

Figure 1. Example single-domain systems: a) mechanical, and b) electrical

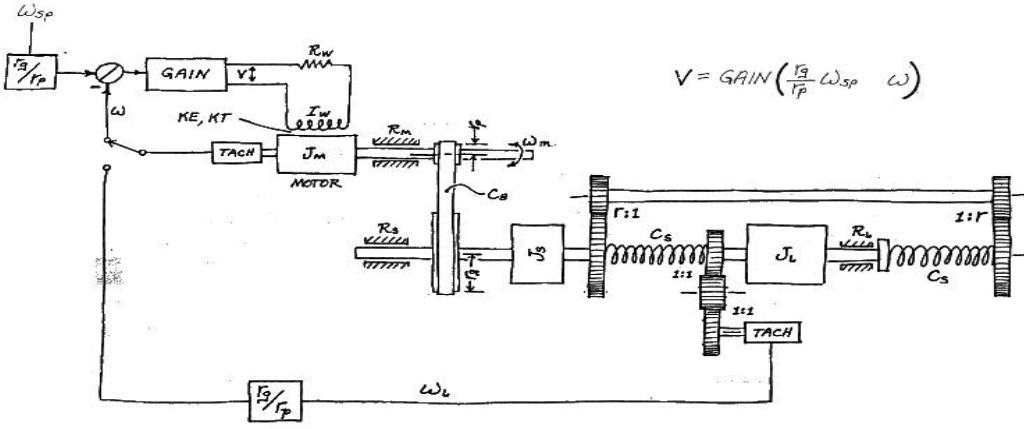

Figure 2. Schematic diagram of an example mechatronic system – the ball drive of an electric typewriter

2.2 Unified and Automated Method

Due to the complexity of the engineering design problem, the need of an efficient design method is greatly increased. Most design tools or methods require user interaction, automating the analysis, but not the synthesis, of the system. The user is involved in a trial/error search, guided by the user's engineering knowledge and intuition. Because design problems are heavily domain-oriented, a change in application often requires the use of a new design/analysis tool, that in turn requires a significant investment in time to learn the new tool. Our design method, which combines bond graphs and genetic programming, can provide both capabilities – design automation and a unified approach (Figure 3). The proposed BG/GP(Bond Graph with Genetic Programming) design method requires, for problem definition, only an embryo model and a fitness (or performance measure) definition – the remaining procedures are automatically executed by genetic programming search.

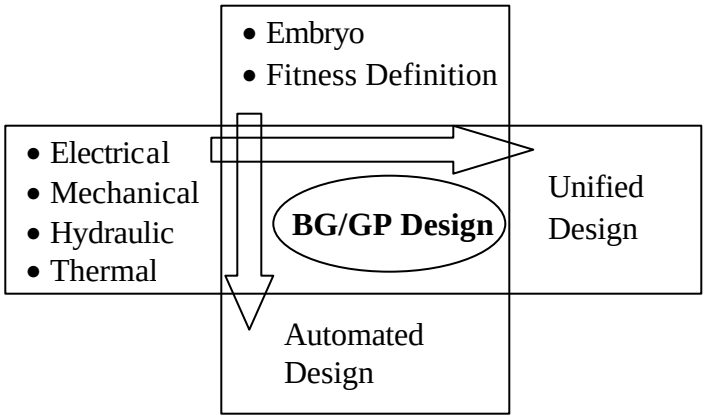

Figure 3. The features of the BG/GP design method

2.3 Bond Graphs

The bond graph is a modeling tool that provides a unified approach to the modeling and analysis of dynamic systems. Bond graph models can describe the dynamic behavior of physical systems by the connection of idealized lumped elements based on the principle of conservation of power.

Bond graphs consist of elements and bonds. There are several types of elements, each of which performs analogous roles across energy domains. The first type -- C, I, and R elements -- are passive one-port elements that contain no sources of power, and represent capacitors, inductors, and resistors (in the electrical domain). The second type, S_e and S_f, are active one-port elements that are sources of power, and that represent effort sources and flow sources, respectively (for example, sources of voltage or current, respectively, in the electrical domain). The third type, TF and GY, are two-port elements, and represent transformers and gyrators, respectively. Power is conserved in these elements. A fourth type, denoted as 0 and 1 on bond graphs, represents junctions, which are two-port (or more) elements. They serve to interconnect other elements into subsystems or system models.

Bond graphs have three embedded strengths for design applications – the wide scope of systems that can be created because of the multi- and inter-domain nature of bond graphs, the efficiency of evaluation of design alternatives, and the natural combinatorial features of bond and node components for generation of design alternatives. First, multi-domain systems (electrical, mechanical, hydraulic, pneumatic, thermal) can be modeled using a common notation, which is especially important for design of mechatronic systems. For example, the mechanical system and the electrical circuit in Figure 1 have the *same* bond graph model (Figure 4). Second, this representation of dynamic systems is also efficient for computational implementation. The evaluation stage is composed of two steps: 1) causality analysis, and, when merited, 2) dynamic simulation. In causality analysis, the causal relationships and power flow among elements and subsystems can reveal various system properties and inherent characteristics that can make the model unacceptable, and therefore make dynamic simulation unnecessary. While the strong typing used in the GP system (see below) will not allow the GP system to formulate "ill-formed" bond graphs, even a "well-formed" bond graph can have causal properties that make it undesirable or unnecessary to derive its state models or to simulate the dynamics of the system it represents. Causality analysis is fast, and can rapidly eliminate further cost for many models that are generated by the genetic programming system, by performing assignment of effort and flow variables and making checks for violations of the appropriate constraints. This simple "filtering" cuts the evaluation workload dramatically. For systems passing causal analysis, state equations are easily and systematically derived from bond graph models. Then various analyses (of eigenvalues, for example) or simulation based on the state model allow computation of the desired performance measures. Third, the graphical (topological) structure characteristic of bond graphs allows their generation by combination of bond and node components, rather than by specification of equations. This means that any system model can be generated by a combination of bond and node components, because of their free composition and unbounded growth capabilities (Figure 4). Therefore it is possible to span a large search space, refining simple designs discovered initially, by adding size and complexity as needed to meet problem requirements.

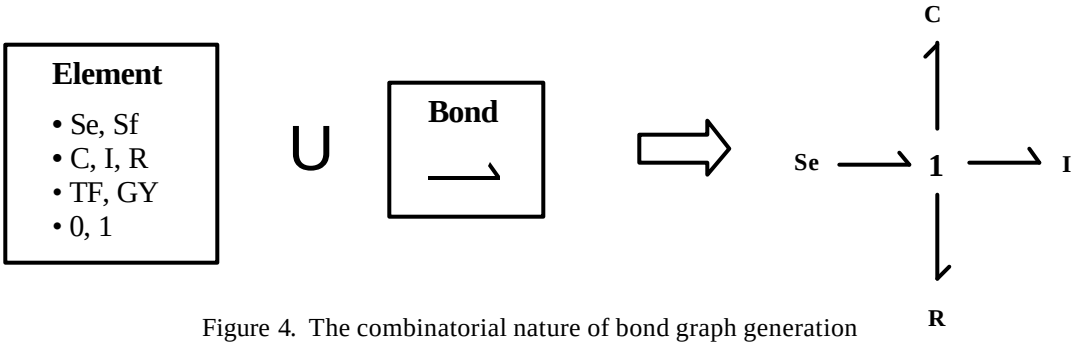

Figure 4. The combinatorial nature of bond graph generation

The particular procedures used for synthesis of bond graph models are a developing and crucial part of this work, since they determine the search space within which design solutions will be contained.

2. 2 Combining Bond Graphs with Genetic Programming

Genetic programming is an extension of the genetic algorithm, using evolution to optimize actual computer programs or algorithms to solve some task (Holland [15], Goldberg [16]), typically involving a graph-type (or other variable-length) representation. The most common form of genetic programming is due to Koza [11-13], and uses trees to represent the entities to be evolved. Genetic programming can manipulate variable-sized data

structures and can be used to "grow" trees that specify increasingly complex bond graph models, as described below. If the scope and analysis efficiency of the bond graph model can be successfully integrated with the impressive search capability of genetic programming when utilized to its full potential, an extremely capable automated synthesis procedure, without need for user intervention, should result.

As with any fairly general system for design automation, the user must, as part of the specification of the problem to be solved, indicate the target performance that is desired and how it is to be evaluated. That generally includes identifying some input variable(s) or driver(s) and some output(s) at which the desired behavior is to be observed, and the desired relationships among them. For a system to be represented as a bond graph, this amounts to specifying an "embryonic" physical model for the target system, which will remain *invariant* during the design process. That embryo should include any inputs, usually specified as time-varying sources of effort or flow (e.g., voltages, currents, forces, velocities, pressures, etc.). It must include any outputs required to evaluate fitness (for example, voltages across a given load resistance, flow rates through pipes, etc.). That these components should NOT be allowed to be changed/eliminated during design evolution is obvious – the problem is not defined without their presence. When the user has formulated the problem (i.e., the external boundaries of the physical model with its environment and the performance measures to be used), the user must specify it as an embryonic bond graph model and a "fitness" function (objective function to be extremized). The user also specifies one or more "sites" in the embryo model where modifications/insertions are allowed. Then an initial population of trees is created at random, using that embryo as a common starting point. For each tree ("individual"), the bond graph analysis is performed. This analysis, including both causal analysis and (under certain conditions) state equation analysis, results in assignment of a fitness to the individual. Then genetic operations – selection, crossover and mutation – are performed on the evaluated population, generating new individuals (designs) to be evaluated. The loop, including bond graph analysis and GP operation, is iterated until the termination condition is satisfied. The result is a "best" bond graph ready for physical realization – there is no guarantee that it is globally optimal in any sense, but the tools are being structured to attempt to make it likely that the solution evolved is at least good, in some broad sense, for a wide range of problem complexity.

3. Evolutionary Design with Bond Graphs

3.1 Bond Graph Construction

A typical GP system (like the one used here) evolves GP trees, rather than more general graphs. However, bond graphs can contain loops, so we do not represent the bond graphs directly as our GP "chromosomes." Instead, a GP tree specifies a *construction procedure* for a bond graph. Bond graphs are "grown" by executing the sequence of GP functions specified by the tree, using the bond graph embryo as the starting point.

Table 1. GP terminals and functions

Name	#Args	Description
add_C	4	Add a C element to a junction
add_I	4	Add an I element to a junction
add_R	4	Add an R element to a junction
insert_J0	3	Insert a 0-junction in a bond
insert_J1	3	Insert a 1-junction in a bond
replace_C	2	Replace the current element with a C element
replace_ I	2	Replace the current element with an I element
replace_ R	2	Replace the current element with an R element
+	2	Add two ERCs
-	2	Subtract two ERCs
enda	0	End terminal for add element
endi	0	End terminal for insert junction
endr	0	End terminal for replace element
erc	0	Ephemeral random constant (ERC)

Some initial studies have been reported in Seo *et al*.[17] and Fan *et al*.[18]. The following set of bond graph elements: {C, I, R; 0, 1}, plus any sources in the embryo, are used. This set is sufficient to allow us to achieve designs that have practical meaning in engineering terms, while still permitting other methods to be used for comparison, as an aid in assessment of our work.

We define the GP functions and terminals for bond graph construction as follows. There are four types of functions: first, *add* functions that can be applied only to a junction and which add a C, I, or R element; second, *insert* functions that can be applied to a bond and which insert a 0-junction or 1-junction into the bond; third, *replace* functions that can be applied to a node and which can change the type of element and corresponding parameter values for C, I, or R elements; and fourth, *arithmetic* functions that perform arithmetic operations and can be used to determine the numerical values associated with components (Table 1).

Some typical operations -- add_R (a 1-port resistor) and insert_J0 (a 0-junction) -- are explained in detail as follows. In Figure 5, the R element is added to an existing junction by the add_R function. This function adds a node with a connecting bond. An R element also requires an additional parameter value (ERC -- ephemeral random constant).

The insert_J0 function can be applied only at a bond, and performs insertion of a 0-junction at the given modifiable site (Figure 6). Inserting a 0-junction between node R and a 1-junction yields a new bond graph (the right side of Figure 6). As a result, three new modifiable sites are created in the new bond graph. At each modifiable site, various bond growth functions can be applied, in accordance with its type. In GP terminology, this is a *strongly typed* GP.

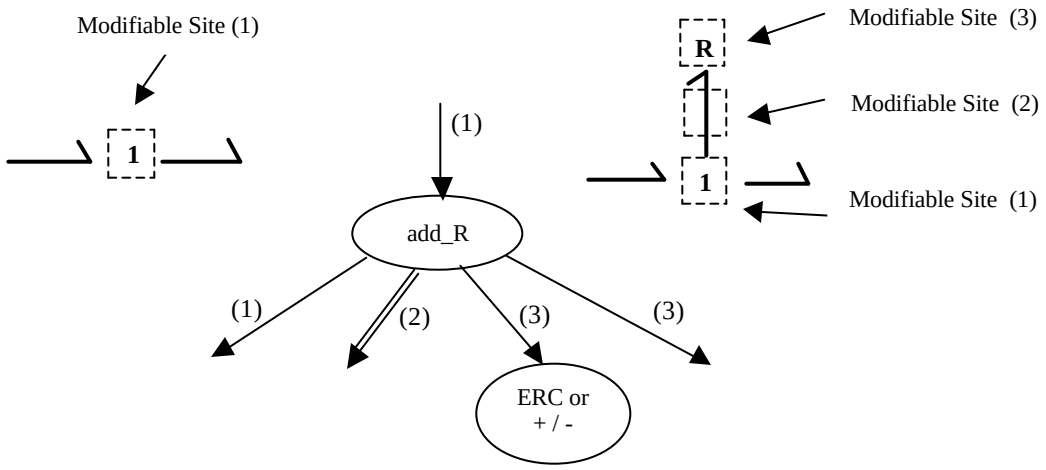

Figure 5. The add_R function

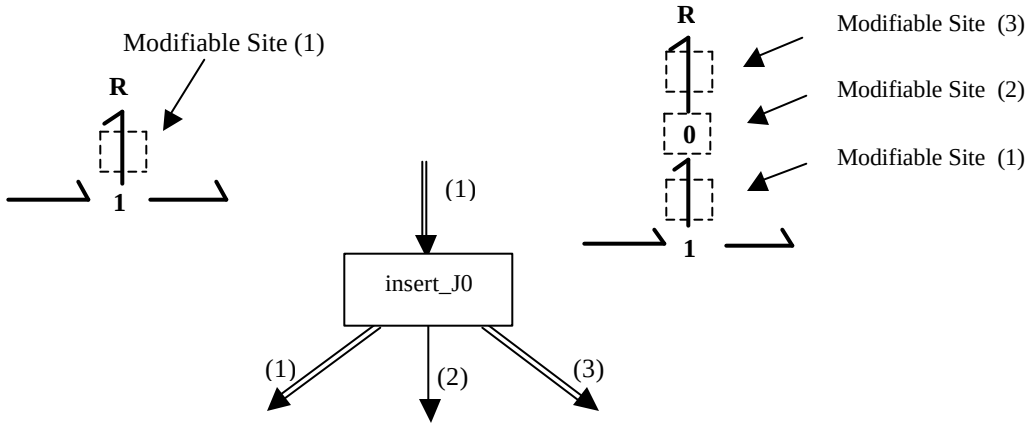

Figure 6. The insert_J0 function

3.2 Overall Design Procedure and Algorithm

The flow of the entire algorithm is shown in Figure 7. The user must specify the embryonic physical model for the target system (*i.e.*, its interface to the external world, in terms of which the desired performance is specified). That determines an embryonic bond graph model and corresponding embryo (starting) element for a GP tree. From that, an initial population of GP trees is randomly generated. Bond graph analysis is then performed on the bond graph specified by each tree. This analysis consists of two steps – causal analysis and state equation analysis. Based on those two steps, the fitness function is evaluated. For each evaluated and sorted population, genetic operations – selection, crossover and mutation – are performed. This loop of bond graph analysis and GP operation is iterated until a termination condition is satisfied. The final step in instantiating a physical design would be to realize the highest-fitness bond graph in physical components, which is not done in the current work.

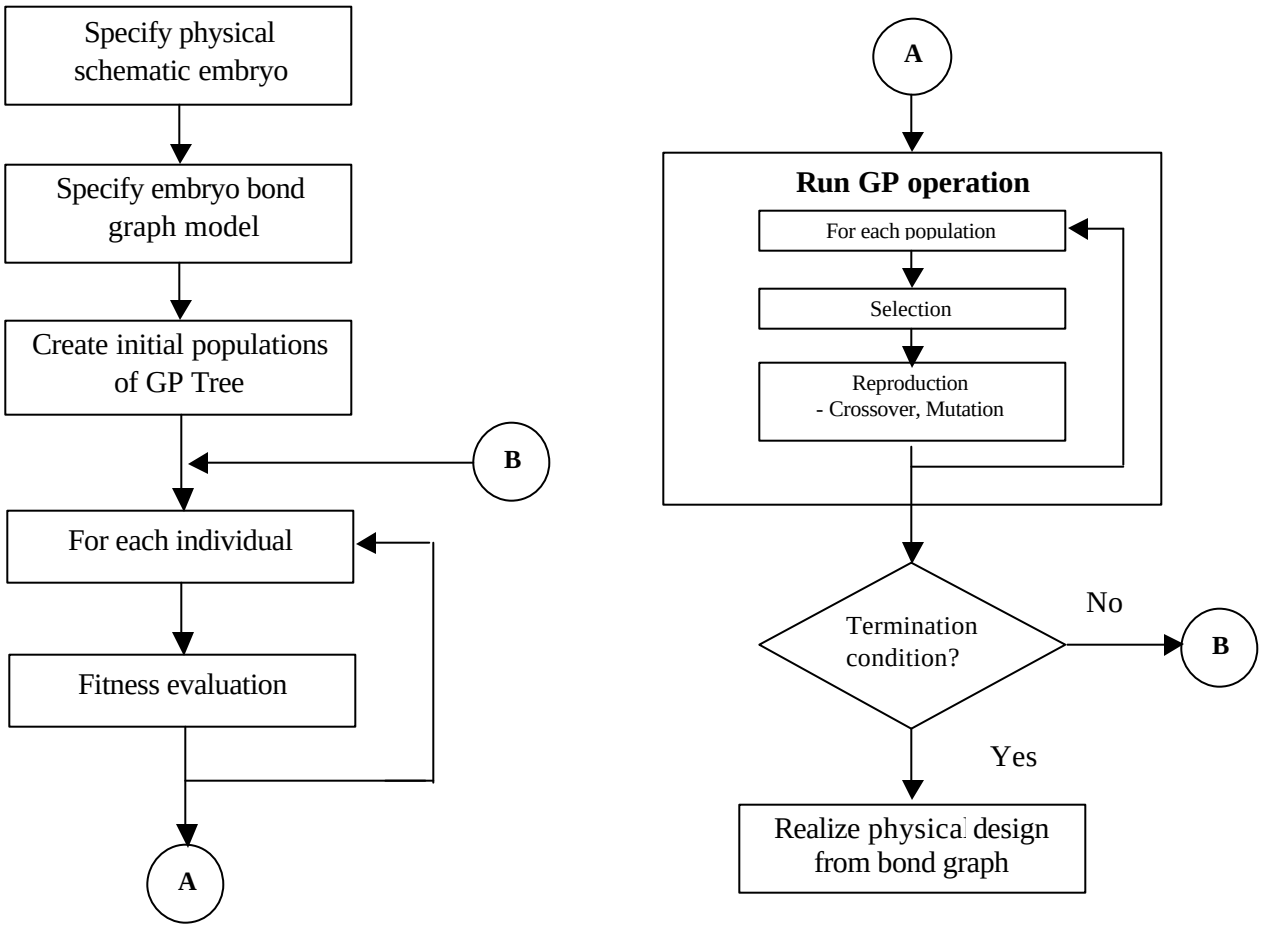

Figure 7. The entire algorithm

3.3 Bond Graph Evaluation

As mentioned earlier, a two-stage evaluation procedure is executed to evaluate bond graph models. The first, causal analysis (see Karnopp *et al*. [2]), allows rapid determination of feasibility of candidate designs, thereby

sharply reducing the time needed for analysis of designs that are infeasible. Designs "failing" the causal analysis step are simply assigned a low fitness (poor performance). For those designs "passing" the causal analysis, the state model is automatically formulated. The evaluation procedure is shown in Figure 8.

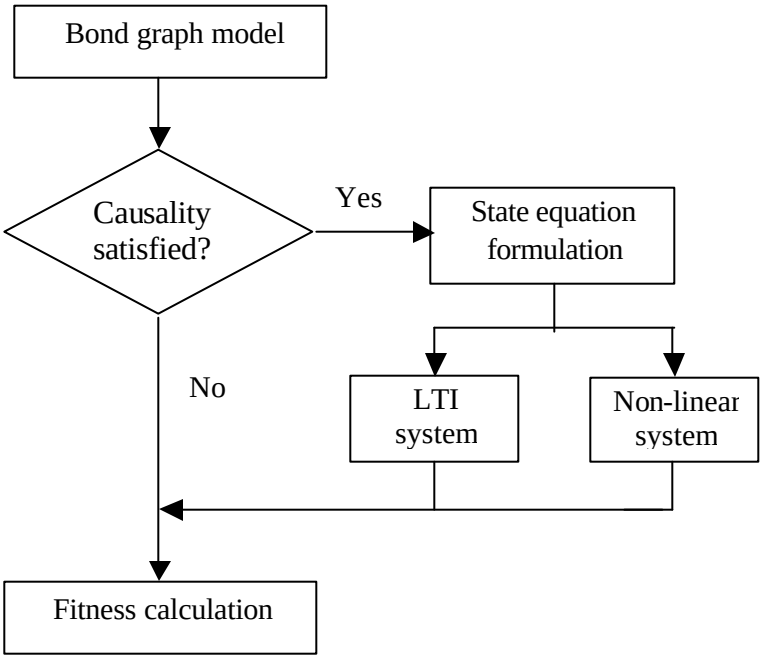

Figure 8. Evaluation of bond graph models

4. Case Study 1 - Eigenvalue Assignment

Although the final design of practical multi-domain systems still requires physical realization of the best generated bond graph model, it is sufficient to design a bond graph model with the desired performance in order to demonstrate the utility of our automated design methodology for multi-domain systems. In the first example problem, the main design objective is to find bond graph models with minimal distance errors from the target sets of eigenvalues. The problem of eigenvalue assignment has received a great deal of attention in control system design. Design of systems to avoid instability and to provide response characteristics as determined by their eigenvalues is often an important and practical problem. The following experiments were done to illustrate the performance of genetic programming on this problem and to explore the topological and parametric behaviors of the bond graph models evolved.

4.1 Problem Definition

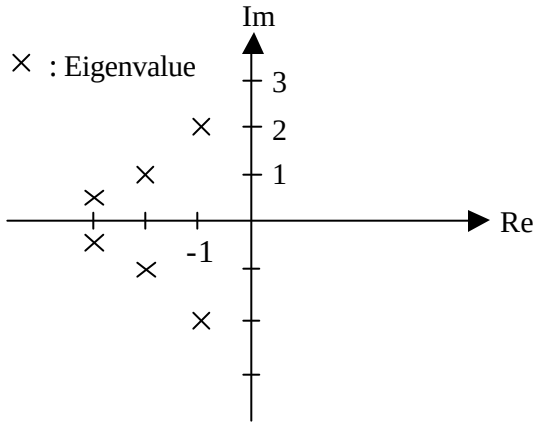

Figure 9. An example of a target set of eigenvalues

In the example that follows, first, a set of target eigenvalues is given and a bond graph model with those eigenvalues is generated. An example of a target set of eigenvalues is shown in Figure 9. The following three sets (consisting of 2, 4, and 6 target eigenvalues, respectively) were used as targets for example genetic programming runs:

{-1±2j}
{-1±2j, -2±j}
{-1±2j, -2±j, -3±0.5j}

The following sets of experiments (six total) were conducted, with each run repeated (with different random number seeds) 5 times for 6-eigenvalue problems and 10 times each for 2- and 4-eigenvalue problems, and with

1) Each target set using an embryo with one modifiable site
2) Each target set using an embryo with three modifiable sites

The two types of embryo model used are shown in Figure 10. Figure 10a represents an embryo bond graph with one initial modifiable site, while the embryo bond graph in Figure 10b has three initial modifiable sites. Each dotted box represents an initial modifiable site ("writehead", in GP terminology). In each case, the fixed components of the embryo are sufficient to allow definition of the system input and output, yielding a system for which the eigenvalues can be evaluated, including appropriate impedances. The construction steps specified in the GP tree are executed at that point. The numbers in parentheses represent the parameter values of the elements.

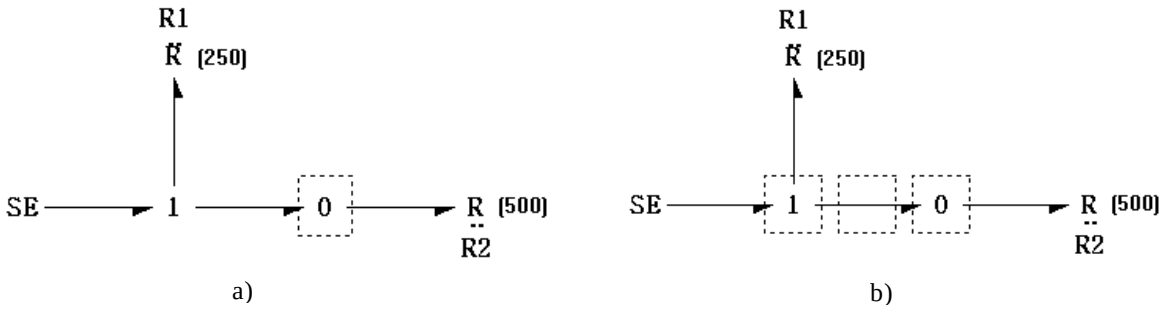

Figure 10. Two types of embryo bond graph model with
a) one modifiable site, b) three modifiable sites

We used a strongly-typed version Luke [19] of lilgp (Zongker and Punch[20]) to generate bond graph models. These examples were run on a single Pentium III 800MHz PC with 256MB RAM. The GP parameters were as shown below.

Number of generations: 100 – 500
Population sizes: 100 – 2500 for single population runs for 2 and 4 eigenvalues
200 in each of 15 subpopulations for multiple population runs for 6 eigenvalues
Initial population: half_and_half
Max depth: 17
Initial depth: 3-6
Selection: Tournament (size=7)
Crossover: 0.9
Mutation: 0.1

4.2 Results

Figure 11 gives the solution eigenvalues obtained for a typical run with targets -1±2j, and summarizes the solutions, average distance errors from the targets, structure and parameter information. The corresponding bond

graph model obtained is shown on the right side of Figure 11. One 1-junction and three elements (one each of C, I, and R) were added to the embryo bond graph model of Figure 10a in evolution of the solution. This final resulting bond graph is obtained after post-processing to remove unnecessary connections and reduce the non-state-variable R to its simplest equivalent form, using well-established rules. In the other runs, topologically similar structures, with C, I, and R elements and a 1-junction attached to the 0-junction, as in Figure 12, dominated the results.

Eigenvalues
Target eigenvalues : -1 ± 2j Solution eigenvalues: -0.993 ± 2.031j Average distance error : 0.032
Bond Graph Structure
added junctions : 1 added elements : 3 added bonds: 4
Bond Graph Parameters
C: 0.5 I : 0.393 R3 : 0.986

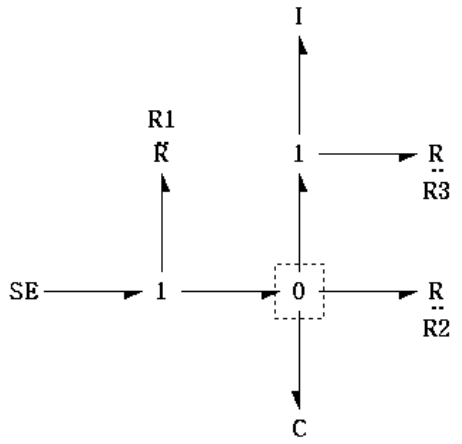

Figure 11. Two eigenvalues result from one modifiable site

Figure 11 illustrates solution eigenvalues and a corresponding bond graph model obtained for the same target set -1±2j, but starting from an embryo with three initial modifiable sites (Figure 9b). Nonetheless, in this case, the C, I and R elements all evolved from the third modifiable site in the embryo, because only two state variables were needed. Nine of 10 runs had the same structure as Figure 12. Only in one case were the C, I, and R attached to the 0-junction (first modifiable site)

Eigenvalues
Target eigenvalues : -1 ± 2j Solution eigenvalues: -1.022 ± 2.051j Average distance error : 0.056
Bond Graph Structure
added junctions : 0 added elements : 3 added bonds: 3
Bond Graph Parameters
C: 0.215 I : 0.885 R3 : 24.191

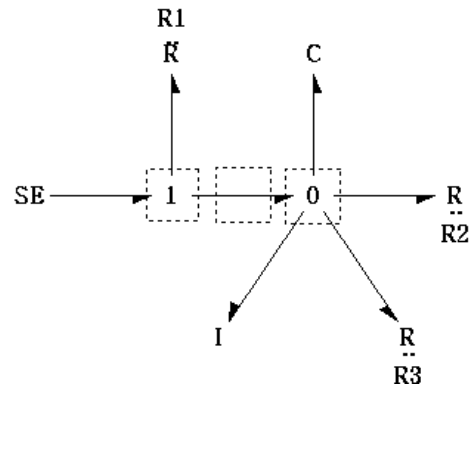

Figure 12. Typical two-eigenvalue result from three modifiable sites

Figure 13 illustrates the solution eigenvalues obtained for the target set -1±2j, -2±j, along with the corresponding bond graph model. One 1-junction and six elements (two each of C, I, and R) evolved from the embryo. Four state variables thus evolved – corresponding to each C or I element. In the 4-eigenvalue problem, the topological search space is larger than in the 2-eigenvalue problem, and a greater variety of structures is discovered. Half of them for the one modifiable site case have very similar structures to that of Figure 13. It is interesting that in 3 of 10 cases, the number of C's and I's in the state vector are not the same (for example, some had one C and three I's). However, in most cases, especially with three modifiable sites, the C's and I's evolved in matched pairs.

Eigenvalues
Target eigenvalues : -1 ± 2j, -2 ± j Solution eigenvalues : -0.996 ± 1.997j, -1.989 ± 1.001j Average distance error : 0.008
Bond Graph Structure
added junctions : 1 added elements : 6 added bonds: 6
Bond Graph Parameters
C1 : 0.029 C2 : 1.189 I 1 : 0.17 I 2 : 6.822 R3 : 13.179 R4 : 0.209

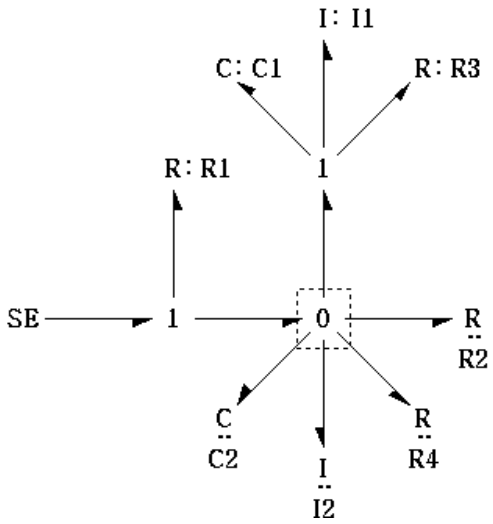

Figure 13. Four-eigenvalue result from one initial modifiable site

Eigenvalues
Target eigenvalues : -1 ± 2j, -2 ± j Solution eigenvalues : -1.028 ± 1.929j, -2.008 ± 0.995j Average distance error : 0.043
Bond Graph Structure
added junctions : 0 added elements : 5 added bonds : 5
Bond Graph Parameters
C1 : 0.0029 C2 : 7.499 I 1 : 121.581 I 2 : 0.027 R3 : 0.033

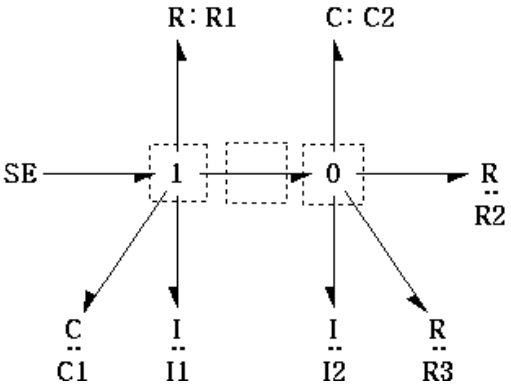

Figure 14. Four-eigenvalue result from three initial modifiable sites

Figure 14 illustrates the result for the target set -1±2j, -2±j when started with three modifiable sites. Two C's, two I's and one R element evolved from the 0- and 1-junctions. Unlike the case of one modifiable site, components evolved at three different modifiable sites. In 8 cases of 10, the resulting bond graph appeared more balanced than when evolved from a single modifiable site.

The computation time for the 6-eigenvalues problem was much larger than for the 4-eigenvalue problem. It took 6-8 hours on a Pentium III 800Mhz PC with 256MB RAM for each solution. It was more difficult to achieve acceptable error distances. (Further work is underway to improve the performance of the system so that problems with up to about 20 eigenvalues are addressed practically.)

Eigenvalues
Target eigenvalues : -1 ± 2j, -2 ± j , -3±0.5j Solution eigenvalues : -0.998 ± 2.005j, -1.983 ± 0.961j -2.970 ± 0.492j Average distance error : 0.02
Bond Graph Structure
added junctions : 4 added elements : 10 added bonds : 14
Bond Graph Parameters
C1 : 1.072, C2 : 2.702 C3 : 1.23 I 1 : 0.235, I2 : 0.982 I3 : 0.222 R3 : 0.289, R4 : 4.197 R5 : 0.106, R6 : 3.009

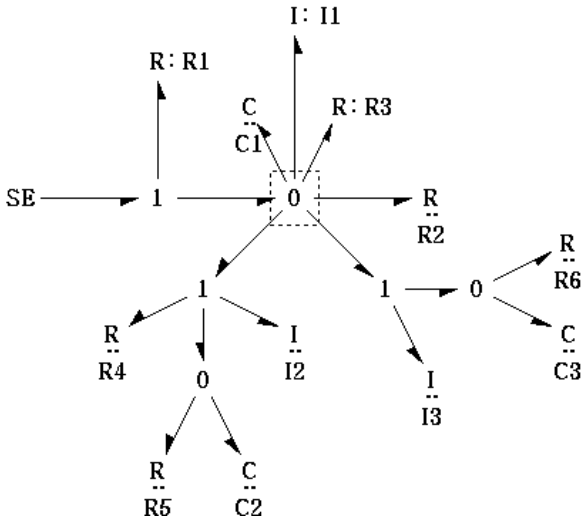

Figure 15. Six-eigenvalue result with one initial modifiable site

Figures 15 and 16 illustrate two typical 6-eigenvalue solutions for the target set -1±2j, -2±j, -3±0.5j. Both figures show that more junctions and elements were added, yielding a more complex structure than in the case of the four-eigenvalue problem. The solution starting from three modifiable sites has a more balanced structure, just as was found in the 4-eigenvalue problems, although stronger conclusions must await additional testing. These runs were produced before the ADF (Automatically Defined Function) facility had been implemented for this strongly typed bond graph representation, but we expect that ADF will allow production of better results for much larger problems, based on the prior work of Koza [12] and the emergence of some useful subpatterns in the bond graphs in our experiments.

The results reported here illustrate two things: 1) Many topological forms of bond graph are capable of satisfying the specified design objectives, and 2) the form of embryo and GP operations used can strongly influence the form of the design evolved. Therefore, care and understanding of the evolutionary process are important in generating designs that maintain desirable topological properties while satisfying the specified design objectives.

Although the experiments run to date are not sufficient to allow making strong statistical assertions, it appears that the search capability of genetic programming is good enough to make feasible the automated design methodology proposed here for multi-domain systems.

Eigenvalues
Target eigenvalues : -1 ± 2j, -2 ± j , -3±0.5j Solution eigenvalues : -1.014 ± 2.012j, -2.053 ± 1.016j -2.998 ± 0.503j Average distance error : 0.18
Bond Graph Structure
added junctions : 3 added elements : 10 added bonds : 13
Bond Graph Parameter
C1 : 0.106, C2 : 0.309 C3 : 0.000 I 1 : 100.574, I2 : 10.0 I3 : 570.908 R3 : 21.39, R4 : 40.393 R5 : 1.116, R6 : 0.002

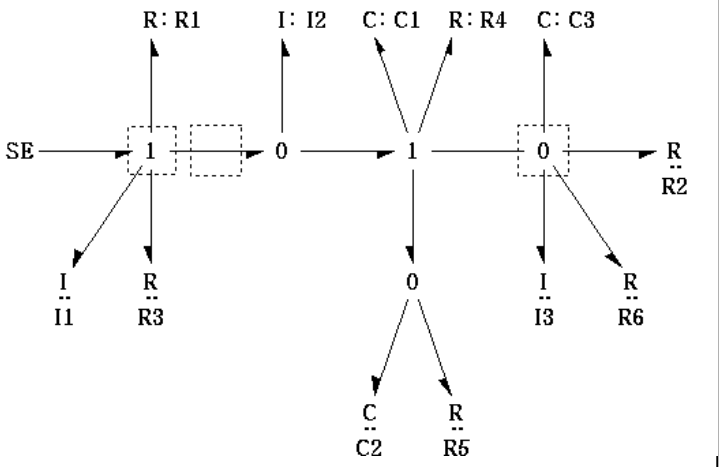

Figure 16. Six-eigenvalue result with three initial modifiable sites

5. Case Study 2 – Analog Filter Design

5.1. Problem Definition

A filter design problem was used as a test of our approach for evolving electrical circuits with bond graphs, as first reported in Fan *et al.* [18]. Three kinds of filters were chosen to verify our approach - high-pass, low-pass, and band-pass filters. The embryo electrical circuit and corresponding embryo bond graph model used in our filter design is shown in Figure 17. We used converted Matlab routines to evaluate frequency response of the filters created. As Matlab provides many powerful toolboxes for engineering computation and simulation, it facilitates development of source codes for our genetic programming evaluation dramatically. In addition, as only causally valid circuits were passed to Matlab code for evaluation, the occurrence of singularities is excluded, which enables the program to run continuously without interruption. The fitness function is defined as follows: within the frequency range of interest, uniformly sample 100 points; compare the magnitudes of the frequency response at the sample points with target magnitudes; compute their differences and obtain the squared sum of differences as raw fitness. Then calculate normalized fitness (to be maximized) according to:

$$Fitness_{norm} = 0.5 + 1/(1 + Fitness_{raw})$$

The GP parameters used for eigenvalue design were as follows:

Number of generations: 500
Population size: 500
Initial population: half_and_half
Max depth: 16
Initial depth: 4-6
Selection: Tournament (size=7)
Crossover: 0.8
Mutation: 0.2

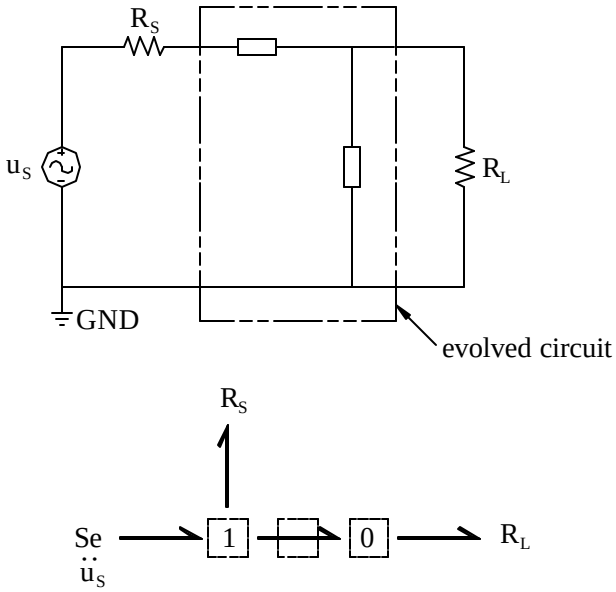

Figure 17. Embryo circuit and its bond graph representation

5.2 Results

The frequency output of the high-pass filter evolved is shown in Figure 18. The upper part is the magnitude of the frequency response, while the lower part describes its phase angle. The figure illustrates that the result is quite satisfactory. To get this result, the program ran in a P-III 550 for 121.9 minutes. It took the genetic programming algorithm 272 generations to evolve it.

Figure 19 gives the frequency response of an evolved low-pass filter. The program ran for 151.8 minutes, in which it evolved for 157 generations to get this result. Figure 20 gives the frequency response of the evolved band-pass filter. The program ran for 193.8 minutes, for a total of 270 generations, to get this result. Obviously, it is the most difficult of the three filter design problems.

The evolved high pass filter circuit and bond graph are shown in Figure 21. From the evolved bond graph, it is clear that the topological structure could be simplified. This topological redundancy is frequently observed in our research. We believe that allowing (or even fostering) this kind of topological redundancy is useful for ensuring the robustness of the evolution process. It may help the search process to traverse rugged landscapes and to avoid getting stuck in local minima.

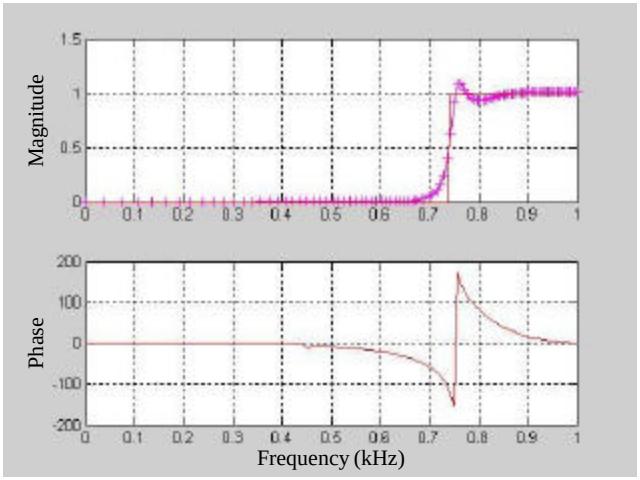

Figure 18. Frequency response of evolved high-pass filter

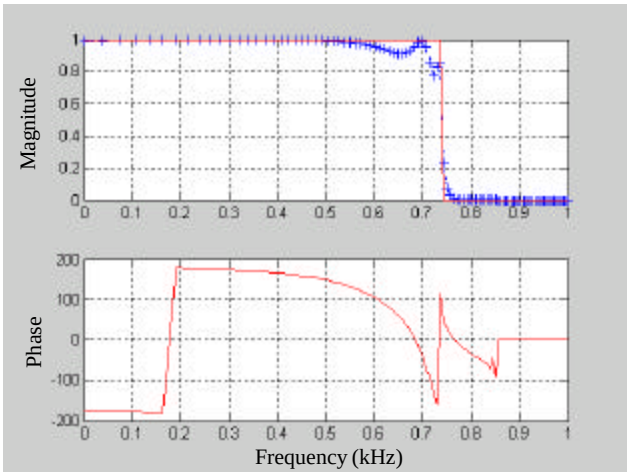

Figure 19. Frequency Response of Evolved Low Pass Filter

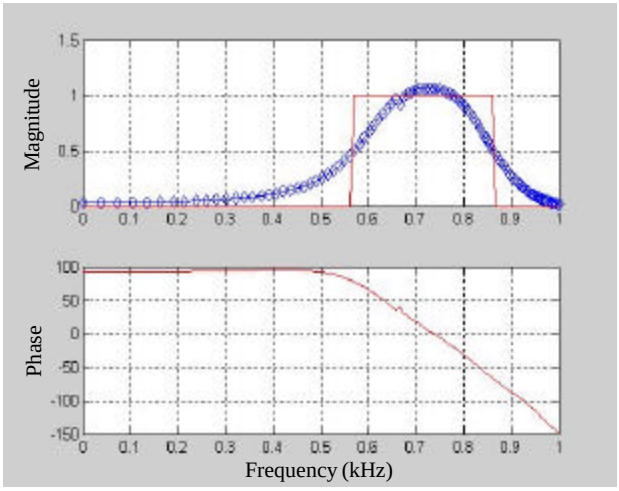

Figure 20. Frequency Response of Evolved Band Pass Filter

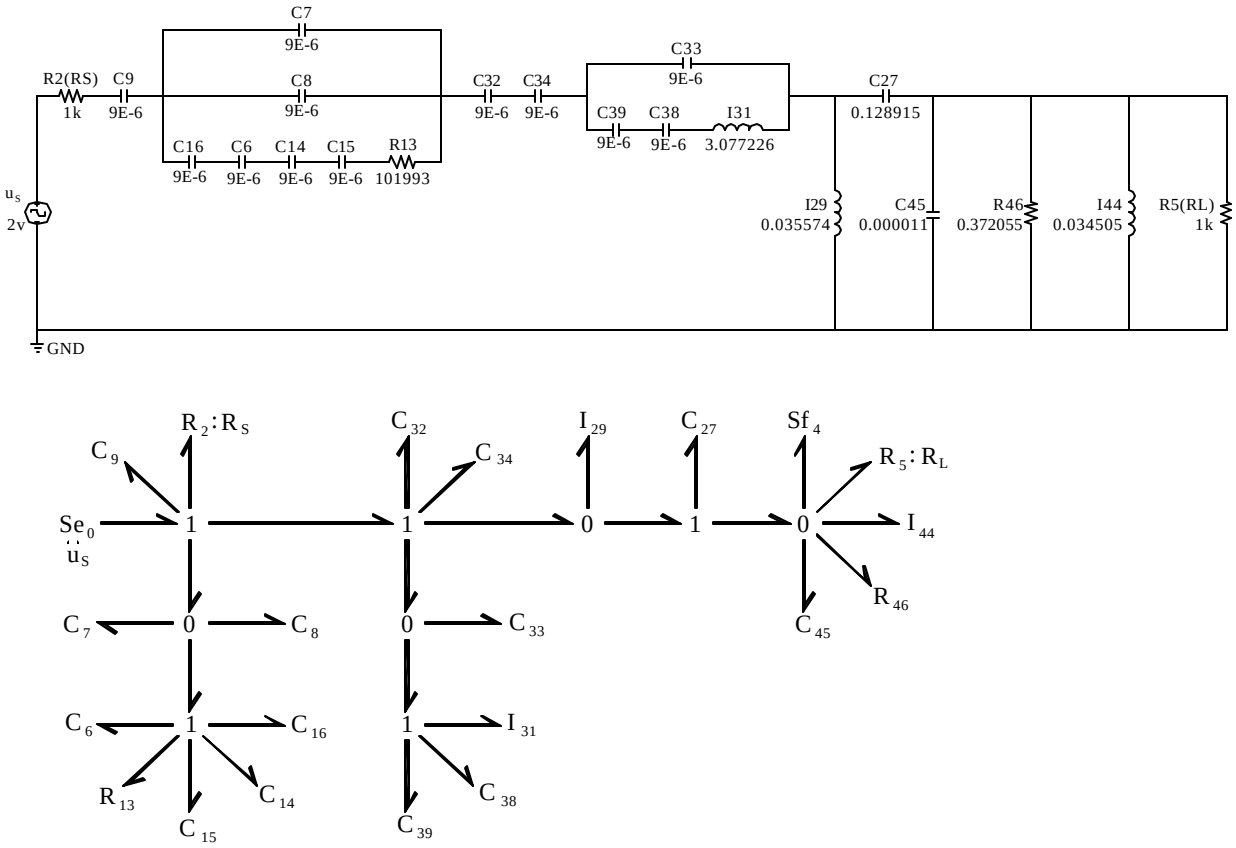

Fig 21. Evolved High Pass Filter Circuit and Bond Graph Representation

6. Case Study 3 – Typewriter Drive Redesign

6.1. Problem Definition

This example involves a drive system for an electric typewriter. The original problem was presented to one of the investigators by C. Denny and W. Oates of IBM, Lexington, KY, in 1972. Figure 2 (in section 2) shows a closed-loop control system to position a rotational load (inertia) denoted as J_L and Figure 22 shows the initial design of a subsystem. The detailed specification involved reducing the vibration of the load to an acceptable level, given certain command conditions for input position.

For comparison with our design solution by the BG/GP method, a simulation was first performed for the initial drive system. The corresponding bond graph model of Figure 23 was derived. A simulation result using 20SIM [ref.] is as shown in Figure 24. The input is a step function and the feedback gain is one(K=1). Output represents position of rotational load (J_L). This design is found to be unsatisfactory because there were unacceptable vibrations in the load for 1700-1800ms.

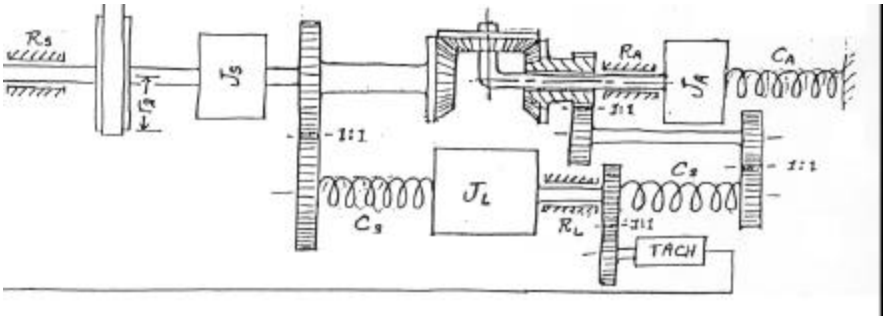

Figure 22. The initial printer drive subsystem.

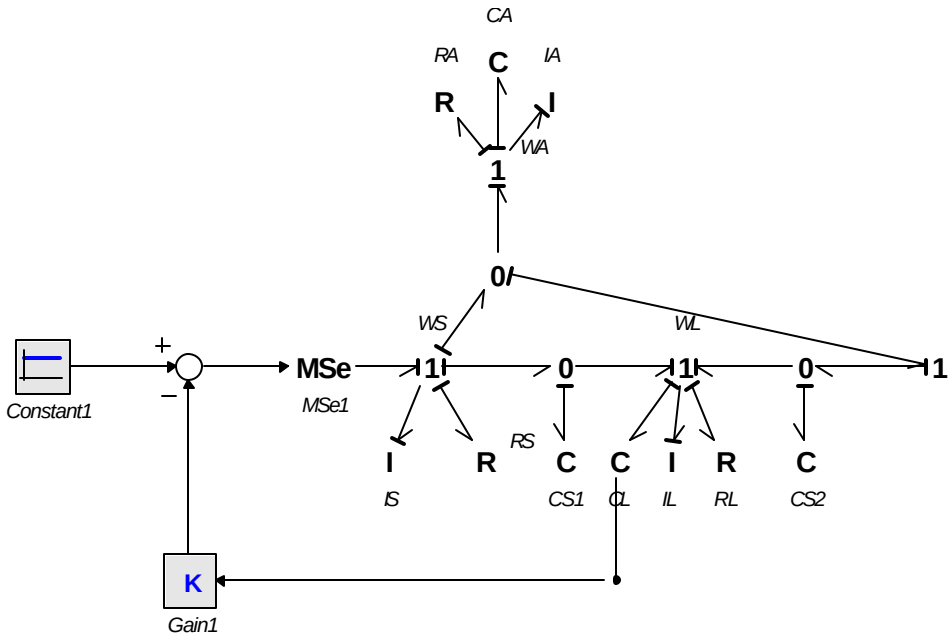

Figure 23. The bond graph model for the initial typewriter drive subsystem.

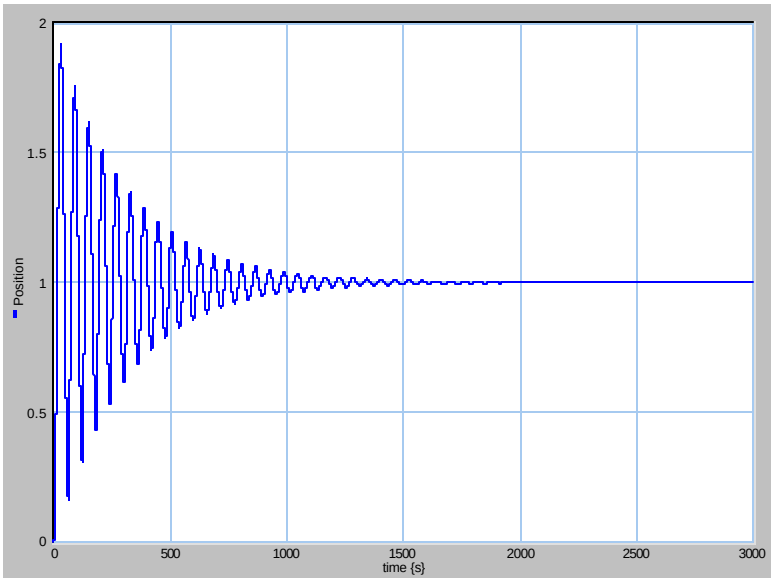

Figure 24. Simulation result of initial typewriter drive subsystem.

To search for a new design using the BG/GP design tool, an embryo model is required. Figure 25 shows an embryo subsystem that involves the drive shaft and the load, with important physical properties modeled. The input is the driving torque, T_d, generated through the belt coupling back to the motor (not shown). The subsystem is open-loop, with the feedback path disconnected. Since the vibration problem seemed to be local, this subsystem was deemed a logical place to begin the design problem.

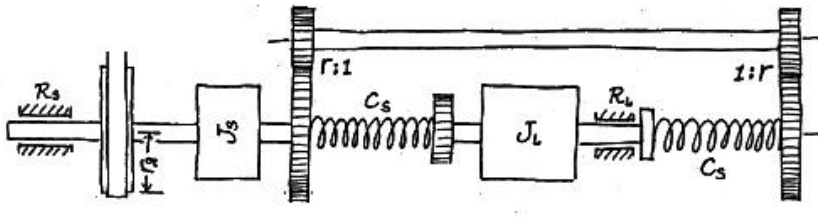

Figure 25. The initial typewriter drive subsystem.

To begin the automated design process it is necessary to seed the search with an embryonic model that, essentially, constitutes a boundary condition for the design to be developed. The corresponding embryonic bond graph model of the drive subsystem is given in Figure 26. The two 1-junctions denote the angular velocities of the shaft inertia (w_S) and the load inertia (w_L), respectively. Also critical to the search procedure is the identification of sites in the model where modifications can be initiated. We allowed three such sites, denoted by 1, 2, and 3 and dashed squares. Sites 1 and 3 permit addition to the model; site 1 is essentially the rigid drive shaft section (w_S), and site 3 is at the right end of the drive shaft spring. Site 2 is an insertion location, where the connecting shaft can be "broken" and other subsystem effects inserted.

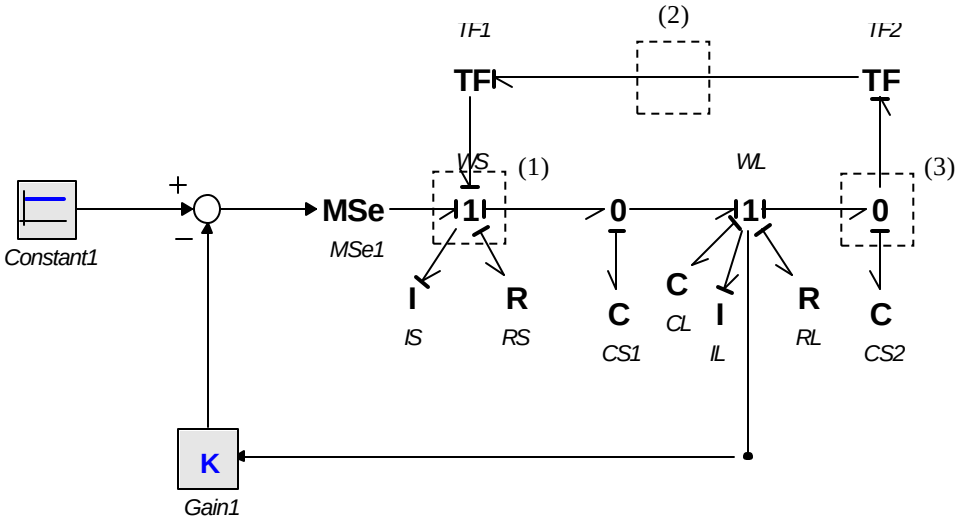

Figure 26. The embryonic bond graph model with feedback loop

The following examples were run on a single Pentium III 800Mhz PC with 256MB RAM. The GP parameters were as shown below.

Number of generations: 100
Population sizes: 200 in each of 15 subpopulations for multiple population runs
Initial population: half_and_half
Max depth: 17
Initial depth: 3-6
Selection: Tournament (size=7)
Crossover: 0.9
Mutation: 0.1

6.2 Results

Although all of our planned experiments for this printer drive redesign problem have not yet been completed, some very good solutions have already been found. One of them is shown Figure 27. It was generated in only 20 generations with 200 designs in each of 15 subpopulations, and has a simple structure. Three elements – one 0-junction, a C and an I – evolved at modifiable site 1 of the embryo model (Figure 26). The performance of this model is shown Figure 28. The position response for a step function input was damped almost completely within 70msec, which constituted an acceptable solution to the problem.

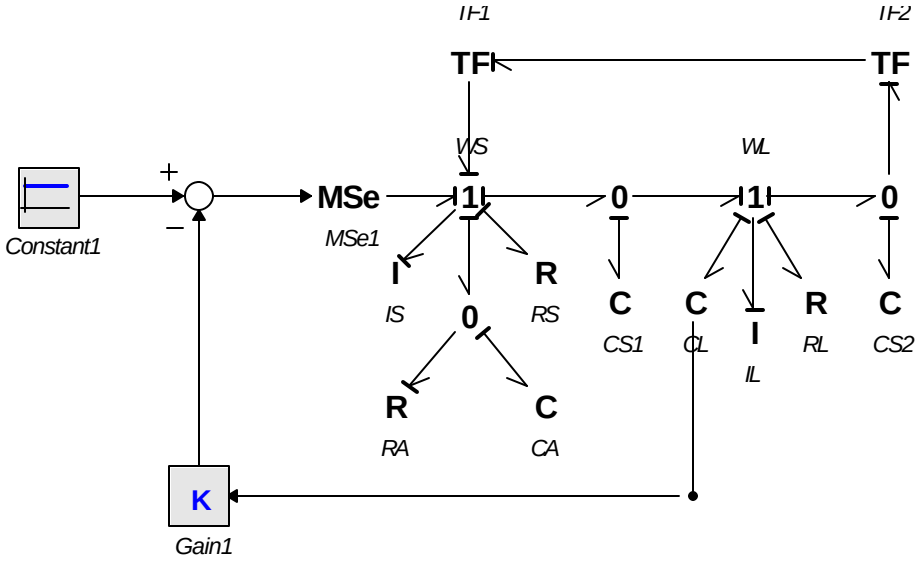

Figure 27. The best evolved bond graph model .

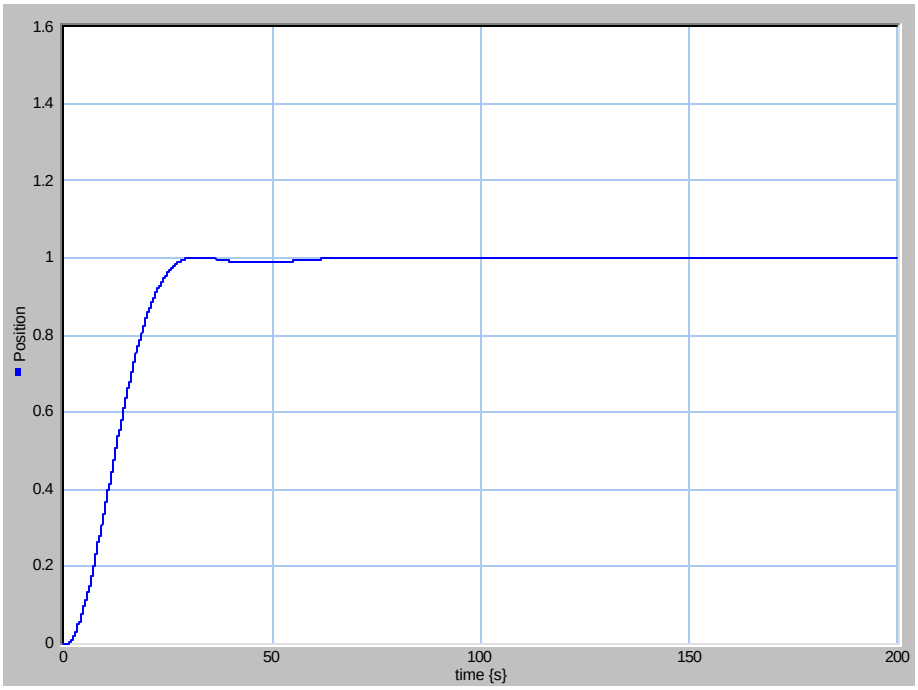

Figure 28. Simulation result of best evolved bond graph model

7. Conclusion

This paper has suggested a new design method for automatically synthesizing designs for multi-domain, lumped parameter dynamic systems, using a unified tool. A careful combination of bond graphs and genetic programming, including a multi-step evaluation procedure that greatly increases the efficiency of fitness assessment, appears to be an appropriate approach to development of a method for synthesis of complex multi-domain systems, such as mechatronic systems.

As a proof of concept for this approach, evolution of a limited set of bond graphs for three different domain design problems – a specified-target-eigenvalues design, electrical filter design, and typewriter drive design – was tested. Experiments showed that the system produced satisfactory results on all three problems in moderate times and computational expenses.

This provides some support for the conjecture that much more complex multi-domain systems with more detailed performance specifications can be automatically designed, given longer execution times and/or using inexpensive cluster computing facilities. Further study will aim at extension and refinement of the design method and its application to design of more complex and multi-domain mechatronic systems.

References

[1] Coelingh, E., T. de Vries, J. Amerongen, "Automated Performance Assessment of Mechatronic Motion Systems during the Conceptual Design Stage," *Proc. 3rd Int'l Conf. on Adv. Mechatronics*, Okayama, Japan, 1998.

[2] Karnopp, D. C., R. C. Rosenberg,, D. L. Margolis [2000] *System Dynamics, A Unified Approach*, 3nd ed., John Wiley & Sons, 2000.

[3] Rosenberg, R. C., J. Whitesell, and J. Reid, "Extendable Simulation Software for Dynamic Systems," *Simulation, 58*(3), 1992, pp.175-183.

[4] Rosenberg, R. C., "Reflections on Engineering Systems and Bond Graphs," *Trans. ASME J. Dynamic Systems, Measurements and Control*, v.115, 1993, pp.242-251

[5] Rosenberg, R. C., *The ENPORT User's Manual*, Rosencode Associate Inc., E. Lansing, MI, 1996.

[6] Rosenberg, R. C., M.K.Hales, and M.Minor, "Engineering Icons for Multidisciplinary Systems," *Proc. ASME IMECE 1996*, DSC-V.58, 1996, pp.665-672.

[7] Sharpe, J. E., R. H. Bracewell, " The Use of Bond Graph Reasoning for the Design of Interdisciplinary Schemes," *1995 International Conference on Bond Graph Modeling and Simulation*, 1995, pp. 116-121.

[8] Youcef-Toumi, K., Ye. A. Glaviano, P. Anderson, "Automated Zero Dynamics Derivation from Bond Graph Models," *1999 International Conference on Bond Graph Modeling and Simulation*, 1999, pp. 39-44

[9] R. C. Redfield, , "Bond Graphs in Dynamic Systems Designs: Concepts for a Continuously Variable Transmission," 1999 International Conference on Bond Graph Modeling and Simulation, 1999, pp. 225-230.

[10] Tay, E., W. Flowers and J. Barrus, "Automated Generation and Analysis of Dynamic System Designs," *Research in Engineering Design*, vol 10, 1998, pp. 15-29.

[11] Koza, J. R., *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, The MIT Press, 1992.

[12] Koza, J. R., *Genetic Programming II: Automatic Discovery of Reusable Programs*, The MIT Press, 1994

[13] Koza, J. R., F. H. Bennett, D. Andre, M. A. Keane, *Genetic Programming III, Darwinian Invention and Problem Solving*, Morgan Kaufmann Publishers, 1999.

[14] Koza J. R., F. H. Bennett, D. Andre, M. A. Keane, F. Dunlap, "Automated Synthesis of Analog Electrical Circuits by Means of Genetic Programming," *IEEE Trans. Evolutionary Computation., 1*(2), 1997, pp.109-128.

[15] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

[16] D. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.

[17] K. Seo, E. Goodman, and R. Rosenberg, 2001, "First Steps toward Automated Design of Systems Using Bond Graphs and Genetic Programming", *Proc. Genetic and Evolutionary Computation Conference*, San Francisco, p. 189 (1-page abstract) and poster.

[18] Z. Fan, J. Hu, K. Seo, E. Goodman, R. Rosenberg, and B. Zhang, 2001, "Bond Graph Representation and GP for Automated Analog Filter Design," *Genetic and Evolutionary Computation Conference Late-Breaking Papers*, San Francisco, pp. 81-86.

[19] Luke, S., 1997, *Strongly-Typed, Multithreaded C Genetic Programming Kernel*, http://www.cs.umd.edu/users/-seanl/gp/patched-gp/.

[20] Zonker and Punch, W.F., III, 1998, *lil-gp 1.1 User's Manual*, GARAGe, College of Engineering, Michigan State University.