Some Key Issues in using Bond Graphs and Genetic Programming for Mechatronic System Design

R.C. Rosenberg, E.D. Goodman, and Kisung Seo
rosenberg@egr.msu.edu
Michigan State University
2555 Engineering Building
East Lansing, MI 48824-1226

Keywords: Bond graphs, genetic programming, mechatronic system design

Abstract

Mechatronic system design differs from design of single-domain systems, such as electronic circuits, mechanisms, and fluid power systems, in part because of the need to integrate the several distinct domain characteristics in predicting system behavior. The goal of our work is to develop an automated procedure that can explore mechatronic design space in a topologically open-ended manner, yet still find appropriate configurations efficiently enough to be useful.

Our approach combines bond graphs for model representation with genetic programming for generating suitable design candidates as a means of exploring the design space. Bond graphs allow us to capture the common energy behavior underlying the several physical domains of mechatronic systems in a uniform notation. Genetic programming is an effective way to generate design candidates in an open-ended, but statistically structured, manner.

Our initial goal is to identify the key issues in merging the bond graph modeling tool with genetic programming for searching. The first design problem we chose is that of finding a model that has a specified set of eigenvalues. The problem can be studied using a restricted set of bond graph elements to represent suitable topologies. We present the initial results of our studies and identify key issues in advancing the approach toward becoming an effective and efficient open-ended design tool for mechatronic systems.

1. INTRODUCTION

Design of interdisciplinary (multi-domain) engineering systems, such as mechatronic systems, differs from design of single-domain systems, such as electronic circuits, mechanisms, and fluid power systems, in part because of the need to integrate the several distinct domain characteristics in predicting system behavior (Coelingh et al. [1]). However, most current modeling and simulation tools that provide for representation at a schematic, or topological, level have been optimized for a single domain. And much interdisciplinary design is still carried out by designers who have been trained in a single discipline and therefore have limited design knowledge of other fields (Sharpe and Bracewell [2]). In order to automate design of interdisciplinary multi-domain systems effectively, a new approach is required. The goal of the work reported in this paper is to develop an automated procedure capable of designing interdisciplinary systems to meet given performance specifications, subject to various constraints. The method must be able to explore the design space in a topologically open-ended manner, yet still find appropriate configurations efficiently enough to be useful. Our approach combines bond graphs for model representation with genetic programming for generating suitable design candidates as a means of exploring the design space. Bond graphs allow us to capture the common energy behavior underlying the several physical domains of mechatronic systems in a uniformly effective manner. Being topological structures, they are ideal for representing a structured design space for open-ended generation and exploration.

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.* [3, 4, 5]. 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 approach appears to be very promising, having produced a number of patentable designs for useful artifacts. It is closely related to our approach, except that it searches in a single energy domain.

Our approach, while not yet as well developed as Koza's, offers several potential advantages. First, it enables us to generate candidate designs that are implicitly multi-domain, using a unified energy-based methodology. Second, it allows us to evaluate the fitness of design candidates efficiently, using a two-stage procedure -- causal analysis, followed (if needed) by simulation using the derived state model. Third, it provides an interesting opportunity to realize the resulting best candidates in a variety of physical manifestations, depending upon the designer's ultimate objectives.

Our initial goal is to identify the key issues in merging the bond graph modeling tool with genetic programming for searching. The first design problem we chose has objective of producing model candidates that have a specified set of eigenvalues. This problem can be studied using simple one-port linear constant-parameter resistance, inertance, and capacitance components. In the balance of this paper we present the initial results of our studies, and key issues are identified in moving the approach toward an effective and efficient open-ended design tool for mechatronic systems. More specifically, Section 2 presents the design approach, Section 3 describes an example study, Section 4 identifies some key issues in design, and Section 5 concludes the paper.

2. THE DESIGN APPROACH

2.1. Bond Graphs

The bond graph provides a unified model representation across mechatronic system domains. Bond graph models describe the dynamic behavior of physical systems by the connection of idealized lumped elements based on energy behavior. It is the explicit representation of model topology that makes the bond graph a good candidate for use in open-ended design searching. For notation details and methods of system analysis related to the bond graph representation see Karnopp et al. [6] and Rosenberg [7]. Much recent research has explored the bond graph as a tool for design (Sharpe and Bracewell [2], Tay *et al*. [8], Youcef-Toumi [9], Redfield [10]).

The initial studies we have done use the following set of bond graph elements: [C, I, R; 0, 1; Se, Sf]. These are generalized capacitance, inductance, and dissipation effects; the junction elements 0 (common effort) and 1 (common flow); and the source elements Se (effort) and Sf (flow), respectively. The C and I represent energy-storage effects, the R represents losses, the 0- and 1-junctions represent the power-preserving topology, and the Se and Sf elements represent input conditions. Furthermore, in this initial study we restrict elements C, I, and R to be linear one-ports with constant parameters. This element set is sufficient to allow us to achieve designs that have practical meaning in engineering terms, while we learn how the various factors of the search method influence the results.

Bond graphs have three major advantages for open-ended automated design of physical systems. These are

- the broad range of systems that can be created because of the multi-domain nature of the representation,
- the efficiency of evaluation of design alternatives, and
- the ease of generating design candidates, due to the topological nature of the representation.

First, multi-domain systems (for example, electrical, mechanical, hydraulic, pneumatic, thermal) can be modeled using a common notation, which is especially important for design of mechatronic systems. The notation also captures the behavior of transducers between domains, such as pumps, motors, and electrohydraulic control valves, effectively. Second, automated evaluation of bond graph model candidates can be made efficient. For example, causal analysis of a model can be done with very low computational cost, but the result can be used to dismiss unacceptable candidates without the need for relatively costly analysis of behavior. Third, the graphical nature of bond graphs allows their ready generation by random combination of bond and node components, postponing the consideration of equation descriptions until needed for detailed analysis. It is possible to span a large model search space with relatively few basic elements, refining simple designs discovered initially, by adding size and/or complexity as needed to meet performance requirements.

2.2 Genetic Programming

Genetic programming (GP) is an extension of the genetic algorithm method, which applies the model of natural evolution to the optimization of computer programs or algorithms to solve some task (Holland[11], Goldberg[12]). Often the model has a graph-type (or other structural) representation. The most common form of genetic programming is due to John Koza [13] and uses trees to represent the entities to be evolved. Genetic programming can be used to "grow" trees that specify increasingly complex models, as described below. In this way it is possible to span a large search space.

Figure 1 illustrates the open-ended search idea that a design in a GP system need not be of any particular size or depth – models can thus easily be represented with arbitrary levels of complexity. The top row shows three GP trees, each

representing a particular model in a given generation of models. The next generation is generated by crossover operations, by mutation operations, and by direct copying. In our case each tree corresponds to a single bond graph, so by using GP tools we can build bond graph candidates efficiently, without the need to create an entire *ad hoc* search structure.

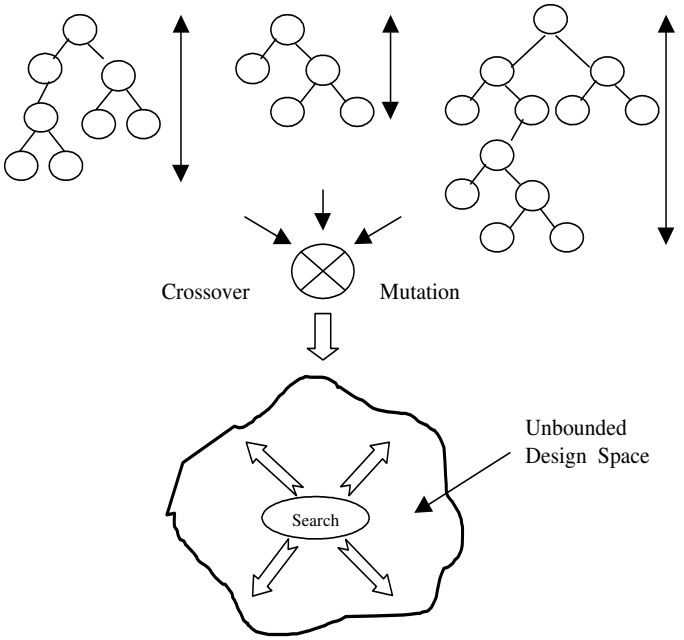

Figure 1. The open-ended search capability of GP

There are several factors involved in conducting a GP-based search. These include

- the embryo,
- the seed population of candidates,
- the fitness function,
- the statistics of evolution, and
- termination conditions for the search.

The *embryo* is an invariant part of a model which contains interfacing or boundary information associated with the problem to be solved. For example, it might include the voltage source at the driving port and the load resistor as invariants in a filter design problem. The embryo must contain the variables used to assess fitness. It also contains the modifiable sites used to create the initial seed population. The *seed population* is an initial set of trees created at random, using the embryo as a starting point. These trees correspond to an initial set of bond graph models, in our application. The *fitness function* is a measure of the relative worth of a candidate model, based on an evaluation of its dynamic performance and other characteristics, *e.g.*, size. The fitness values guide the evolution of the model population. It is an objective function to be optimized. The *parameters of evolution* guide the generation of new candidates. They apply to the basic operations - selection, crossover and mutation. For example, crossover might be applied to 70% of the existing population, while mutation is applied to 10%, and copying accounts for the rest. Selection is part of the generation process that uses the fitness measure to determine who "stays" or "reproduces" and who "dies" in the existing population. The *termination conditions* can be phrased in terms of model goodness and/or in terms of search effort. The evolution loop, including model analysis and GP operation, is iterated until the termination conditions are satisfied. The result is a “best” model (or a set of acceptable models).

2.3 Bond graphs and genetic programming

The overall procedure is shown in Figure 2. The designer sets the design context by specifying an embryo bond graph model (i.e., driver and load ports in any number). Parameters for the GP search process must be set to control both the generation phase and the evolution phase. An initial set of candidates is generated in the form of GP Trees. At each stage of evolution each of the candidates is evaluated and assigned a fitness value. The evolution phase is guided by the statistics of the selection and evolution operators. The evolution process terminates when fitness or effort conditions are met. The result is reported as a bond graph (set) that has the highest fitness rating.

2.3.1. Constructing a Bond Graph from a GP Tree

In our work a GP Tree (GPT) specifies a construction procedure for a bond graph. A bond graph is “grown” by executing the sequence of instructions coded in the GP tree, using the embryo tree as the starting point. As mentioned earlier, the initial studies reported here use the following set of bond graph elements: [C, I, R; 0, 1; Se, Sf]. C, I, and R are linear one-port elements. Se and Sf are restricted to the embryo. This set is sufficient to allow us to explore meaningful design problems, while still permitting us to use other methods as an aid in assessment of our BG/GP method.

A GP Tree is a directed graph whose single root node is the embryo. Each edge of the GPT corresponds either to a node or to a bond in the bond graph. For convenience we denote GPT node edges with a single line and a number; we denote GPT bond edges with a double line and a letter.

Figure 3 shows a single operation in the GP Tree, **Add_R**, and the corresponding effect on the bond graph. Part (a) shows a modifiable 1-junction site in the bond graph; its (arbitrary) label is "1". Part (b) shows the GP Tree for the Add_R operation. The operator acts on junction node site "1" and generates three sites. One is the existing node site "1"; the second is a new modifiable bond site "a"; the third is the new modifiable node site "2" of type R. The change to the bond graph is shown in part (c). An R element also requires a parameter value (termed "ERC" - ephemeral random constant). In the GP approach both topology and parameters (or, more generally, equations) are modified in the same operation.

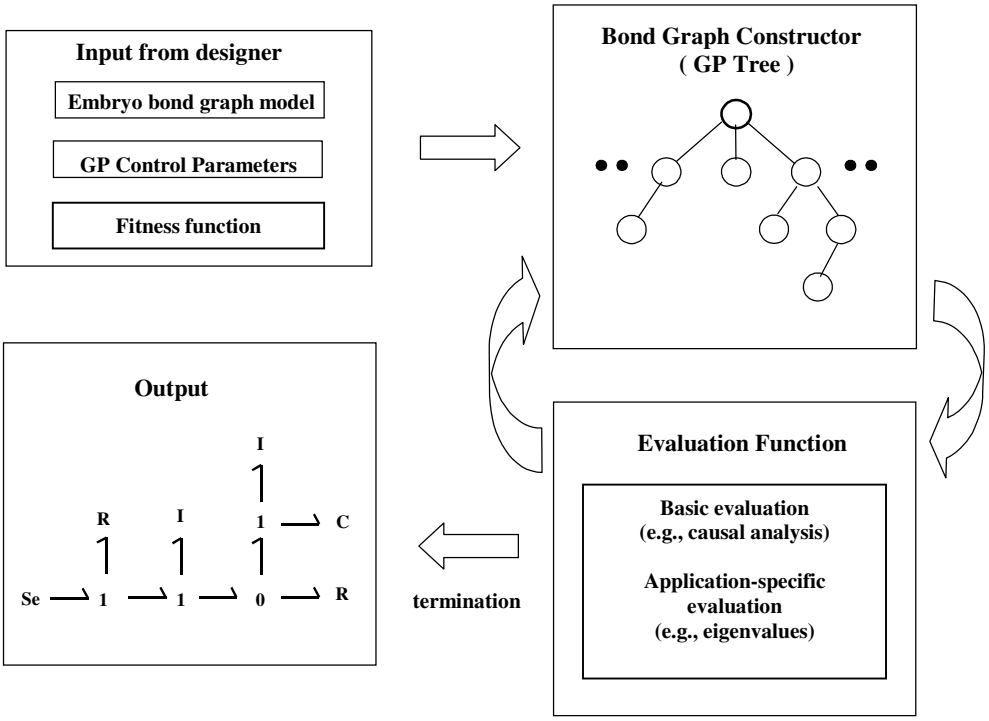

Figure 2. The GP design procedure

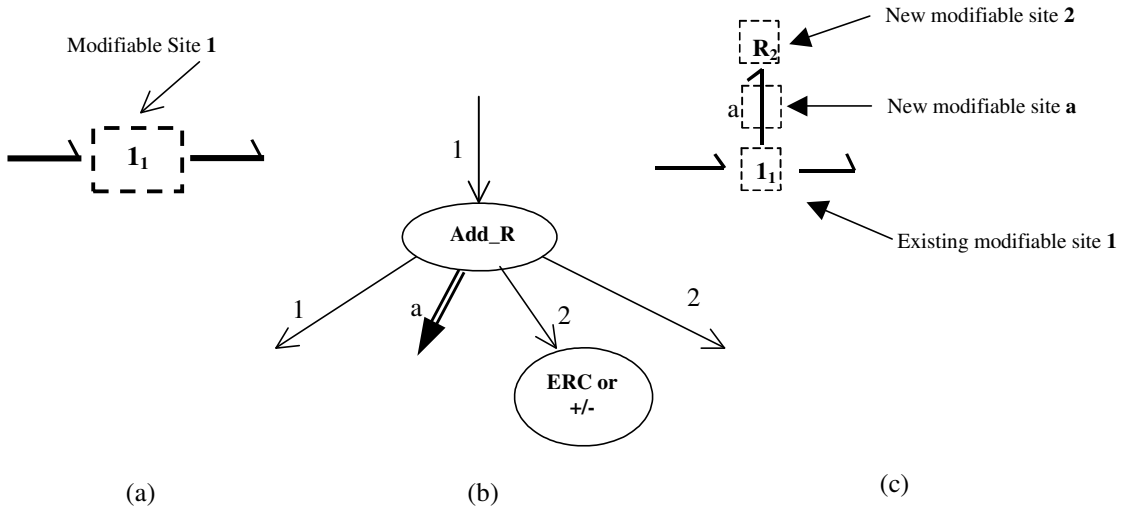

Figure 3. The *add_R* function

The fundamental group of GPT operations we use to generate bond graphs (bgs) are *Add, Replace, Insert,* and *End*. The types of functions allowable depend upon the site type. There are three site types, namely,

[0,1] node sites

- *add_C* add bond and C node to a junction site
- *add_I* add bond and I node to a junction site
- *add_R* add bond and R node to a junction site
- *end* mark site as unmodifiable

[C,I,R] node sites

- *replace_C* replace current node by C node

- *replace_I* replace current node by I node
- *replace_R* replace current node by R node
- *end* mark site as unmodifiable
- *erc* set ephemeral random constant (ERC)
- **+** add two ERCs
- **–** subtract one ERC from another ERC

Bond sites

- *insert_J0* insert 0-junction and new bond
- *insert_J1* insert 1-junction and new bond
- *end* mark site as unmodifiable

The set of functions defined above is not unique as a way to use the GPT to generate bond graphs. However, the functions are mutually exclusive and complete, as long as one is not generating bond graphs with loops. (That is a subject of our next investigations.) Also, the function set is relatively easy to implement.

Figure 4 shows an example of a GP Tree. The root node is the embryo. There are three modifiable embryo sites, denoted "1" (bond graph node), "a" (bond), and "2" (bg node). Each is denoted by an edge of the GPT. If we follow edge 1 first, we see that an I element is added to the bg, together with its parameter value and a new bond. The result is to preserve modifiable site "1" and to add modifiable sites "b" and "3". The next set of operations under *add_I* in the GPT show that all three sites are made unmodifiable by *end* functions.

Turning next to the edge labeled "a", we see that the first function applied to it is *end*. That bond site is made unmodifiable. On the other hand, site "2" is the locus of additional bond graph growth. The patient and motivated reader can verify that the bond graph of Figure 5 is generated from the GPT of Figure 4.

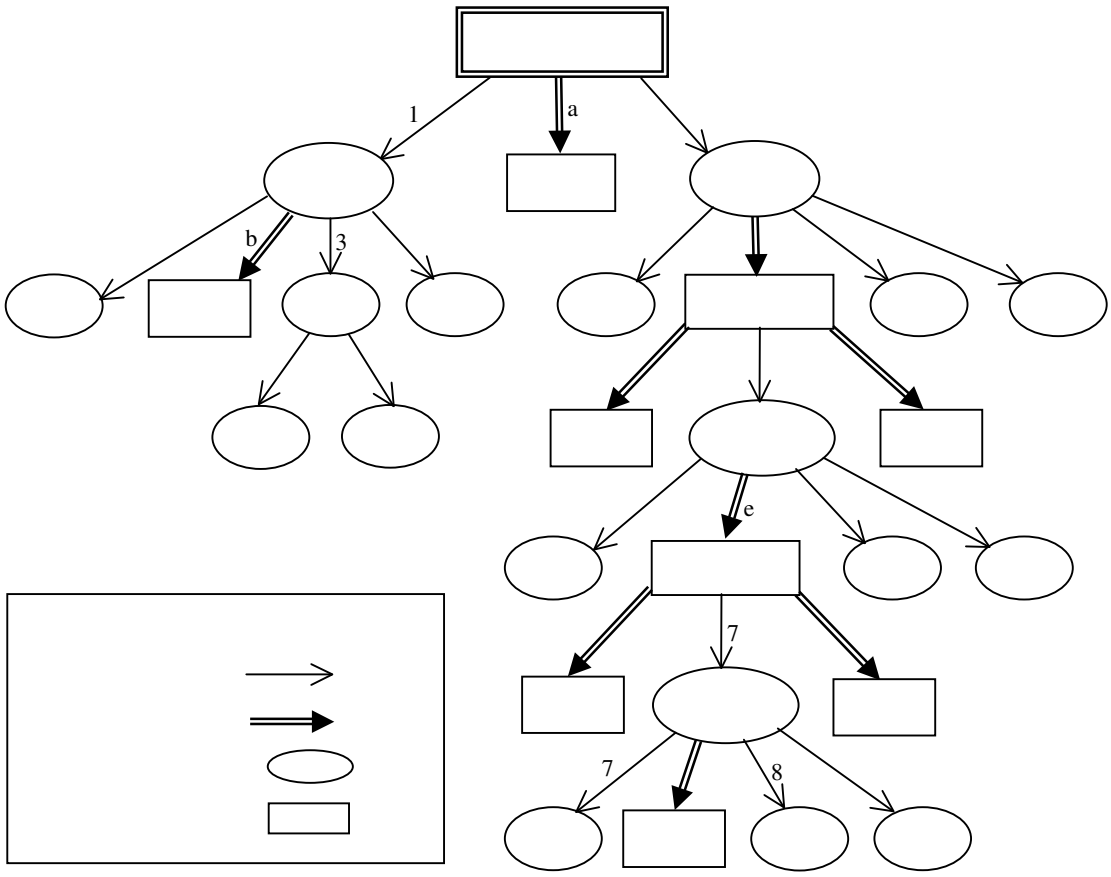

Figure 4. Example of a GP Tree.

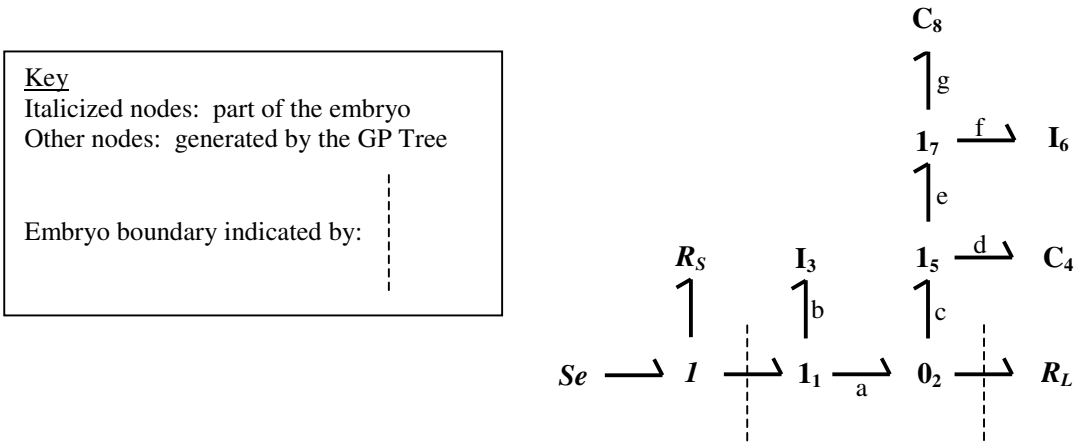

Figure 5. Bond Graph generated by GP Tree example.

2.3.2. Bond Graph Evaluation

Evaluating a model's performance and assigning a fitness value is normally the most time-consuming part of the GP search process. In this regard, the assessment of bond-graph model causality (see, e.g., Karnopp et al. [6]), which is relatively inexpensive computationally, offers the potential for increasing efficiency considerably in the evaluation process. For each design "passing" the causal analysis test, a state-space equation model is formulated automatically. Then one or more additional evaluators may be used, such as eigenvalue analysis, impulse response, step response, and/or frequency response. Because the formulation of (state-space) equations from a bond graph model transcends the specific domain, we only need one method to do this. Increasing the efficiency of this process increases the efficiency of design in all domains.

3. AN ILLUSTRATIVE EXAMPLE

3.1. The Eigenvalue Assignment Problem

The problem of eigenvalue assignment has received a great deal of attention in control system design. Design of systems to avoid instability is often an important and practical problem. For our first study of a design problem we chose to specify a set of target eigenvalues and an embryo and use the GP method to find a bond graph model with those eigenvalues. The embryo model is shown in Figure 6. The dotted box shows the initial modifiable site ("writehead" in GP terminology). The numbers in parentheses represent the parameter values specified for the fixed (embryo) elements. The embryo nodes are italicized.

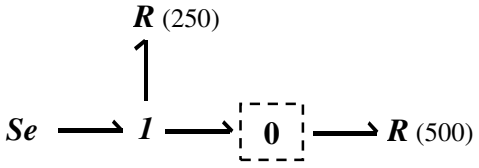

Figure 6. Embryo bond graph model

3.2. The Experimental (Computational) Setup

We used a strongly-typed version of "lilgp" to generate bond graph models (Zongker and Punch [14], Luke [15]). The examples reported here were run on a single Pentium III processor, rather than on a PC cluster, in order to make measurement of the resources consumed easier. The search parameters are grouped in two sets.

Initial population generation parameters

- Population size: 100–500 (fixed for a run)
- Growing method of initial GP Tree: half_and_half
- Depth of initial GP Tree: 2–6

Evolution process parameters

- Number of generations: 100–500 (fixed for a run)
- Maximum depth of GP Tree: 17
- Selection pool: Tournament (size=7)
- Crossover probability: 0.9 (for individual in selection pool)
- Mutation probability: 0.1 (for individual in selection pool)

3.3 Study Results

Two particular eigenvalue problems were studied. The purposes were to demonstrate feasibility, to identify key issues, and to get a feeling for the influence of size relative to effort. Eigenvalue targets were as follows:

Study 1: -1±2j, -2±j
Study 2: -1±2j, -10±j, -2±20j

Table 1 gives the best solution for a typical run for study 1. It tabulates several error distance measures. The corresponding bond graph model is shown in Figure 7. The embryo is preserved and can be identified as the elements that have no parameters shown. The final result required some post-processing to remove redundant junctions and to reduce the R to its simplest equivalent form. Such rules are well known.

Table 1. Best solution for -1±2j, -2±j target.

Solution eigenvalues	-2.001 -0.998 -2.001 +0.998 -0.995 -1.995 -0.995 +1.995
Min distance error	0.002
Max distance error	0.007
Total distance error	0.018
Average distance error	0.005

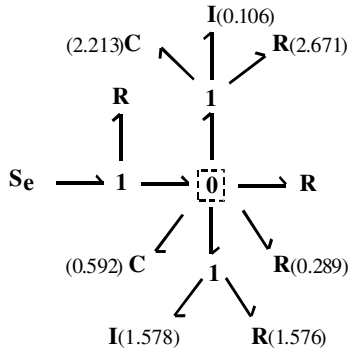

Figure 7. Best search result for Study 1.

Tables 2 gives the best solution for the target set of study 2, namely, -1±2j, -10±j, -2±20j. The target set was chosen to have a greater distances between the target pairs. Figure 8 shows that more junctions and elements were added to attain the new target.

Table 2. Best solution for -1±2j, -10±j, -2±20j

Solution eigenvalues	-1.958 –19.970 -1.958 +19.970 -10.024 –0.987 -10.024 +0.987 -0.999 –2.002 -0.999 +2.002
Min distance error	0.002
Max distance error	0.052
Total distance error	0.162
Average distance error	0.027

4. SOME KEY ISSUES

In the process of conducting these first studies we have identified a number of important issues. Some of the issues relate to speeding up the search, so that larger problems can be handled reasonably effectively. We will not discuss such issues here.

Interaction of topological and parametric searching. It is not yet clear when best to focus on topological issues and when best to focus on parametric issues for a particular class of problems. This issue will become even more important when we do nonlinear design problems. Imbedded in our work is another interesting sub-problem, that of "catalog design." By restricting C, I, and R parameter values to a small set known a priori (e.g., a choice among a dozen R values) we get an interesting practical design problem, but one with quite different features.

Fitness of infeasible candidates. Another important issue is how best to define a fitness for a candidate model to achieve the desired evolutionary goal. For example, should we give a very low fitness to a model whose causal properties seem undesirable, thereby essentially minimizing its future role in the search? Or is it better to keep it more involved in the search population, based on the idea that one more modification may render such a candidate very viable?

Topological properties of results. Because we are able to generate topologies on an open-ended basis, we can also introduce aspects of topological properties into the fitness measure. For example, we can try to get bg answers that have a chainlike structure (long path length) or ones that are more bushlike (shorter path length). A specific design problem may prefer one outcome to the other. We are just learning how to shape the fitness measure appropriately.

Exploiting subsystems that arise during a search. Perusal of results of even the small problems we have been studying thus far suggests that sub-graphs, or chunks, will play an important role in building up larger models. Can we identify and make effective use of chunks that evolve in a given search? This idea is related to the GP facility for automatically defined functions (ADFs), in which the basic building blocks are sub-graphs, rather than single elements.

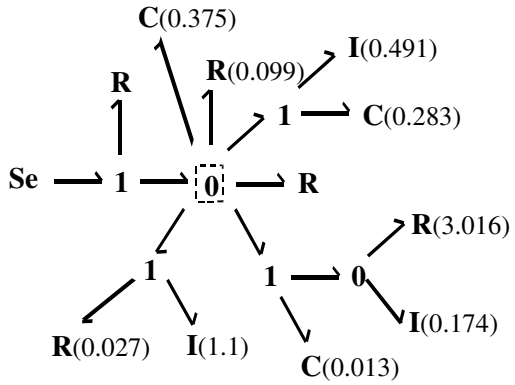

Figure 8. Best search result for Study 2.

5. CONCLUSION

This paper has described a new automated methodology for synthesizing designs for multi-domain, lumped parameter dynamic systems. Such systems are typical in mechatronics. By combining the bond graph representation with genetic

programming as a search method, we have created a design environment in which open-ended but guided topological search can be accomplished. An initial problem - designing a system to have specified eigenvalues - was studied to demonstrate the feasibility of the approach. The first objective, that of demonstrating feasibility, was achieved.

Next we searched for the key issues which would guide our efforts in making the design environment useful for difficult practical problems. We quickly found that including a basic fitness measure based on causal properties of candidates greatly increased the efficiency of the fitness assessment calculation. Reasonable efficiency in searching is a requirement of any practical environment. And we are assuming that cost of computation must be bounded realistically, although we have not quantified this factor.

In conducting the initial studies we identified a number of important issues. One issue is the need to speed up performance assessment of models as much as possible, since that is the most time-consuming part of the search process. Another issue is the need to create appropriate fitness measures that reflect the objectives of the design problem, which is not always easy to do, because the mapping between a GP for an object and the object itself is complex. Modifying GP Trees as the primary evolutionary mechanism is an indirect approach that needs careful attention. A third issue is the defining of the embryo, especially in relation to generating a good initial population. Search results can be very sensitive to the starting conditions, at least as far as efficiency is concerned.

In subsequent studies we will broaden the design problem types in a progressive manner. We will introduce additional modeling elements, such as transformers and gyrators, to perform variable scaling operations in linear systems. Then we will introduce nonlinear C, I, and R elements, followed by modulated TF and GY elements. Such a rich modeling domain will make many practical design problems available to us.

ACKOWLEDGEMENT

The authors gratefully acknowledge the support of the National Science Foundation through grant DMI 0084934.

REFERENCES

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

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

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

[4] Koza, J.R., Andre, D., Bennett, F.H., and Keane, M.A., 1997, "Evolution using genetic programming of a low-distortion 96 Decibel operational amplifier," *Proceedings of the 1997 ACM Symposium on Applied Computing*, San Jose, California, pp. 207-216.

[5] Koza, J.R., *et al*., 1999, "Automatic creation of both the topology and parameters for a robust controller by means of genetic programming," *Proceedings of the 1999 IEEE International Symposium on Intelligent Control, Intelligent Systems, and Semiotics*. Piscataway, NJ: IEEE. pp. 344-352.

[6] Karnopp, D.C., Margolis, D.L., and Rosenberg, R.C., 2000, *System Dynamics: A Unified Approach*, 3rd ed., John Wiley & Sons, New York.

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

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

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

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

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

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

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

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

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