Automated Design Methodology for Mechatronic Systems Using Bond Graphs and Genetic Programming

Goodman, Erik D., Genetic Algorithms Research and Applications Group (GARAGe), Case Center for Computer-Aided Engineering & Manufacturing, Michigan State University, goodman@egr.msu.edu

Seo, Kisung, Genetic Algorithms Research and Applications Group (GARAGe), Case Center for Computer-Aided Engineering & Manufacturing, Michigan State University, ksseo@egr.msu.edu

Rosenberg, Ronald C., Department of Mechanical Engineering, Michigan State University, rosenber@egr.msu.edu

Fan, Zhun, Electrical and Computer Engineering, Michigan State University, fanzhun@egr.msu.edu

Hu, Jianjun, Computer Science and Engineering, Michigan State University, hujianju@egr.msu.edu

Zhang, Baihai, Department of Mechanical Engineering, Michigan State University, zhangbai@egr.msu.edu

Abstract

This paper suggests an automated design methodology for synthesizing designs for multi-domain systems, such as mechatronic systems. The domain of mechatronic systems includes mixtures of, for example, electrical, mechanical, hydraulic, pneumatic, and thermal components, 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 are efficient for classification and analysis of models, allowing 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 open-ended search. The combination of these two powerful 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 two design examples. One is domain independent, an eigenvalues-placement design problem which is tested for some sample target sets of eigenvalues. The other is in the electrical domain – namely, design of analog filters to achieve specified performance over a given frequency range.

1. INTRODUCTION

Mechatronic system design is a type of multi-domain problem that 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 *et al.*[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 an 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 method 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 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. Being topological structures, they are also 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.* [2-4]. 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; however, it works in a single energy domain. Danielson *et al.*[5] use both bond graphs and a genetic algorithm to design a 2-stroke combustion engine. They start from a preliminary design, find near-optimal values for 15 physical parameters for a combustion engine, but without allowing topological variation. Tay *et al.*[6] use a genetic algorithm to vary bond graph models. This 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.

Our approach can address both limitations mentioned above. First, it enables the analysis of multi-energy-domain systems with a unified inter-domain tool. Second, it allows unlimited search for a topologically open-ended solution. Moreover efficient and rapid evaluation of individual designs is provided, using a two-stage procedure -- causal analysis of the graph, followed -- only if needed -- by appropriate detailed calculation using a derived state model. Also, a library of useful model performance evaluation tools (e.g., eigenvalue analysis, frequency response, steady-state calculation) can be embedded as need arises, since the use of these tools is encapsulated in the overall process.

We tested two different problems – one that is domain-independent and another in the electrical domain. The first design problem is eigenvalue design – *i.e.,* to realize a design having a specified set of eigenvalues. The second problem is analog filter design -- to realize an analog circuit that passes particular ranges of frequencies and rejects others. Since both problems can be studied effectively using linear components with constant parameters, we only needed to introduce one-port (generalized) resistance, capacitance, and inertance elements in these 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 some results for the eigenvalue design example, and Section 5 present analog filter results for high-pass, low-pass, and band-pass filters. Section 6 concludes the paper.

2. DESIGN METHODOLOGY

A. Design Domain

Design of 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 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.

B. Bond Graph

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. These models provide very useful insights into the structures of dynamic systems (Karnopp *et al.*[7], Rosenberg [8-9]). Much recent research has explored bond graphs as tools for design (Sharpe *et al.*[10], Youcef-Toumi [11], Redfield [12]).

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- (or multi-) port elements. They serve to interconnect other elements into subsystem 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 generating 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 3). 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 "well-formed" bond graphs can have causal properties that make it undesirable or unnecessary to derive their state models or to simulate the dynamics of the systems they represent. 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 3). Therefore it is possible to span a large search space, refining simple designs discovered initially, by adding size and complexity as needed to meet complex requirements.

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.

C. Genetic Programming

Genetic programming (GP) is an extension of the genetic algorithm method, which applies a model of natural evolution to the optimization of computer programs or algorithms to solve some task (Holland[13], Goldberg[14]). Often the model has a graph-type (or other structural) representation. The most common form of genetic programming is due to John Koza [15] 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 4 illustrates the open-ended search idea that a design in a GP system need not be of any particular size or depth compared with GA – models can thus easily be represented with arbitrary levels of complexity. Each GP tree represents 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.

3. EVOLUTION OF BOND GRAPHS

A. The Design Procedure

The overall procedure is shown in Figure 5. The designer sets the design context by specifying an embryo bond graph model (*i.e.,* driver and load ports in any number (required for the objective function to be defined), and if desired, any other fixed "plant" which the search process is not allowed to alter). Parameters for the GP search process must be set to control both the generation phase (yielding an initial population of candidate solutions in the form of GP trees) and the evolution phase. 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 (or set of them) with the highest fitness rating(s).

B. Bond Graph Construction

A typical GP system (like the one used here) evolves trees, rather than more general graphs. However, bond graphs can contain circuits, 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 and terminals specified by the tree, upon specific bonds or nodes of the bond graph, using the embryo as the starting point. The initial studies reported here use the following set of bond graph elements: {C, I, R; 0, 1}. 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: 1) *add* functions that can be applied only to a junction and which add a C, I, or R element; 2) *insert* functions that can be applied to a bond and which insert a 0-junction or 1-junction into the bond; 3) *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 4) *arithmetic* functions that perform arithmetic operations and can be used to determine the numerical values associated with components (Table 1).

Some typical operations -- insert_J0 (a 0-junction) and add_R (a 1-port resistor) -- are explained in detail as follows. 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.

In Figure 7, 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).

C. 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.*, [7]), allows rapid determination of feasibility of candidate designs, thereby sharply reducing the time needed for analysis of designs that are infeasible. For those designs "passing" the causal analysis, the state model is automatically formulated. Then one or more second-level evaluators is used (for example, eigenvalue analysis, impulse response, step response, etc.). An advantage of our evaluation method is its ability to share these common evaluation routines, without modification, among various energy domains and including multi-domain systems. The evaluation layers are depicted in Figure 8, but any particular design problem probably uses only a subset of these.

4. CASE STUDY 1 – EIGENVALUE DESIGN

A. Problem Definition

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. In the example that follows, a set of target eigenvalues is given and a bond graph model with those eigenvalues is generated. The embryo model provided is shown in Figure 9. The dotted box represents the initial modifiable site ("writehead"). The construction steps specified in the GP tree are executed at that point. The numbers in parentheses represent the parameter values found for the elements. The fitness function is defined as follows. The raw fitness, defined as $Fitness_{raw}$ is sum of the distances between target points and the closest solution points, after they are paired. Then normalized fitness is calculated according to:

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

B. Experimental Setup

We used a strongly-typed version (Luke[16]) of lil-gp (Zongker and Punch[17]) to generate bond graph models. The GP parameters were as shown below. The examples shown here were run on a single Pentium III processor rather than on a PC cluster, in order to simplify the measurement of the resources consumed. Results were first reported in Seo *et al.* [18], in July, 2001. The GP parameters used for these eigenvalue design problems were as follows (in the standard notation of the GP literature):

Number of generations:	100 – 500
Population size:	100 – 500
Initial population:	half_and_half
Max depth:	17
Initial depth:	2-6
Selection:	Tournament (size=7)
Crossover:	0.9
Mutation:	0.1

The following three sets (consisting of 2, 4, and 6 target eigenvalues, respectively) were used as targets for lilgp runs:

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

C. Results

Figure 10 gives the solution eigenvalues obtained for a typical run with targets -1±2j. The distance errors from the targets, evolved number of bonds and components, and the corresponding bond graph model are also shown. One 1-junction and three elements (one each of C, I, and R) were added to the embryo bond graph model in evolution of the solution. This final resulting bond graph was obtained after some post-processing to remove unnecessary connections and reduce the non-state variable R to its simplest equivalent form, using a well-established procedure.

Figure 11 illustrates the solution eigenvalues obtained for the target set -1±2j, -2±j. One 1-junction and six elements (two each of C, I, and R) evolved from the embryo. Four state variables were evolved – one for each C or I element.

Figure 12 illustrates the eigenvalues of the best solution for the target set -1±2j, -2±j, -3±0.5j. It shows that a bond graph with more junctions and elements was evolved, yielding a more complex structure than in the case of the four-eigenvalue problem. There are some interesting observations to be made across these examples: each bond graph model evolved had a symmetric number of storage (state variable) C and I elements (for example, three of each). Although not enough experiments have yet been run to make any statistical assertions, the results appear to be good for these small example problems. These runs were produced before the ADF (Automatically Defined Function) facility was 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 and our observation of the emergence of some useful subpatterns in the bond graphs evolved in our experiments.

5. CASE STUDY 2 – ANALOG FILTER DESIGN

A. 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 July, 2001, in Fan *et al.* [19]. Three kinds of filters were chosen to verify our approach - high-pass, low-pass, and band-pass filters. The embryo electric circuit and corresponding embryo bond graph model used in our filter design is shown in Figure 13. 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 all individual circuits passed to Matlab code for evaluation are causally valid, 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 normalized fitness is calculated according to:

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

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

B. Results

The frequency output of the high-pass filter evolved is shown in Figure 14. 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 15 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 16 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 17. 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.

6. CONCLUSION

This paper has suggested a new design methodology for automatically synthesizing designs for multi-domain, lumped-parameter dynamic systems – assembled from mixtures of electrical, mechanical, hydraulic, pneumatic, and thermal components. 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 yield an appropriate approach for 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 two design problems – a specified-target-eigenvalues design and filter design -- was tested. Experiments showed that both yielded satisfactory results in moderate times and computational expenses.

The eigenvalue design problem can be used not only to assure the stability and transient behavior of a system, but also to explore various topology issues. Bond graph representation of electrical circuits permits many kinds of circuits to be created, and can also be extended to include diodes, transistors and various types of energy sources in circuits.

These early results provide support for the conjecture that much more complex multi-domain systems with much more detailed performance specifications can be automatically designed, given longer execution times and/or using inexpensive cluster computing facilities.

To make the automated design methodology of multi-domain systems more applicable in practice, several research directions are being undertaken: 1) design novel function sets for genetic programming to improve on current designs, comparing their advantages and disadvantages with the current function set, and seeking insights into the evolutionary process of genetic programming on bond graphs; 2) use a parallel realization of genetic programming to enhance its ability to search and improve computational efficiency; and 3) because practical designs usually have many constraints and criteria entering into the evaluation of design quality, such as sensitivity of the system to variation in the parameters of its components, implement a multi-objective genetic programming system to tackle this problem.

The guiding objective continues to be to refine the approach such that it is useful for automated design of practical and important mechatronic systems.

7. 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. 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.

3. 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.

4. 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.

5. Danielson, B., J. Foster and D. Frincke [1998], “GABSys: Using Genetic Algorithms to Breed a Combustion Engine,” *Proc. of IEEE Conf. on Evolutionary Computation*, pp. 259-264

6. 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.

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

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

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

10. 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.

11. 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.

12. 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.

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

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

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

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

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

18. 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.

19. 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.

.

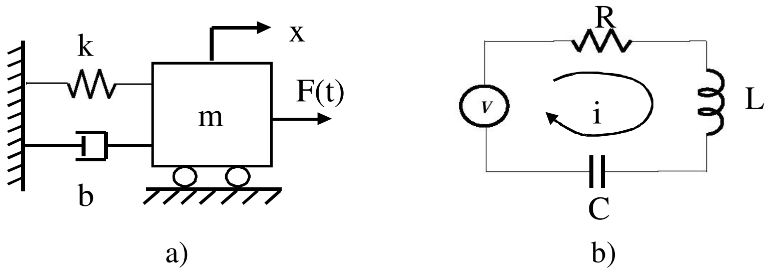


Figure 1. Example single-domain systems: a) mechanical, and b) electrical

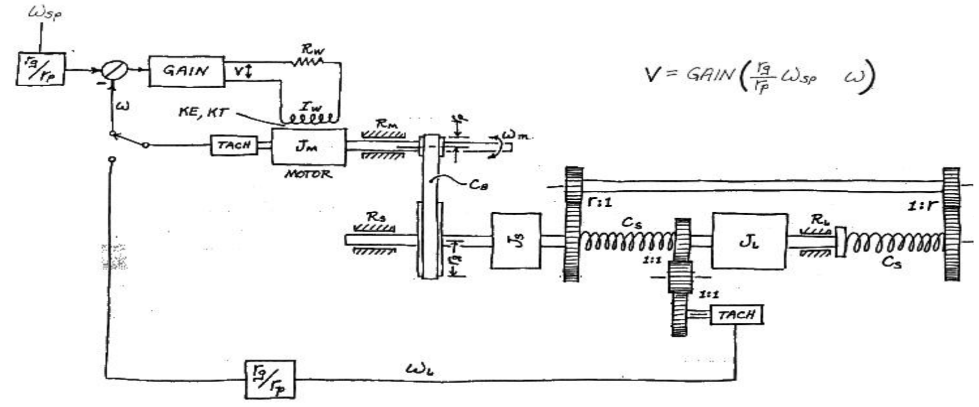


Figure 2. Schematic diagram of an example mechatronic system – the ball drive of an electric typewriter

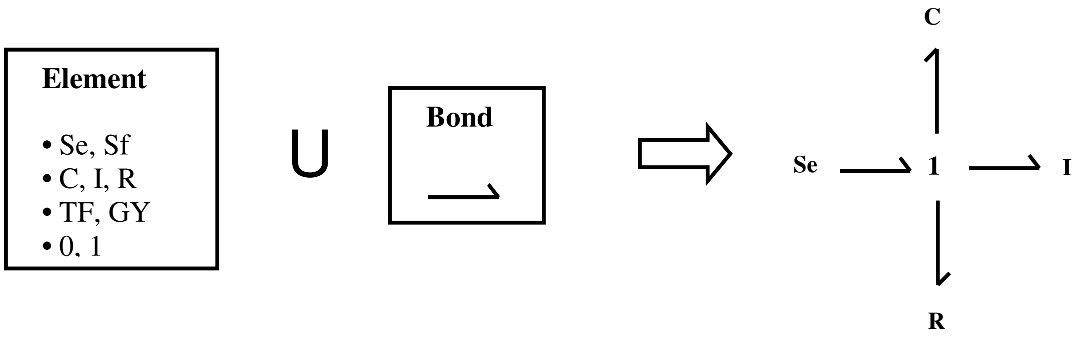


Figure 3. The combinatorial nature of bond graph generation

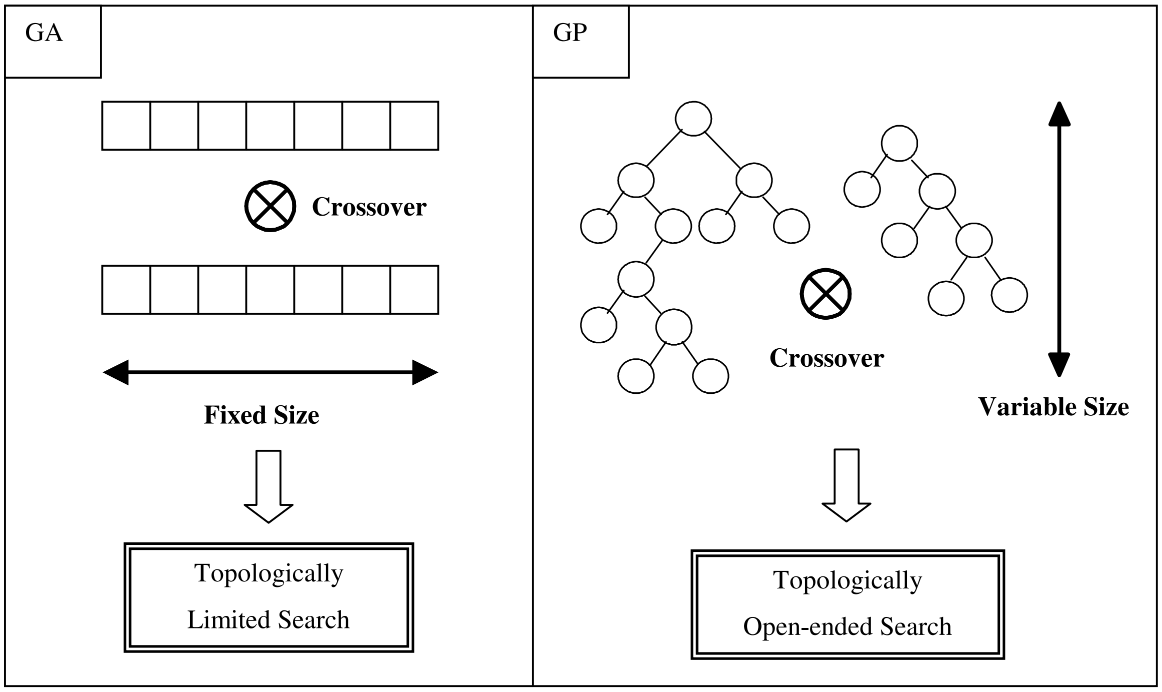


Figure 4. The open-ended search capability of GP compared with GA

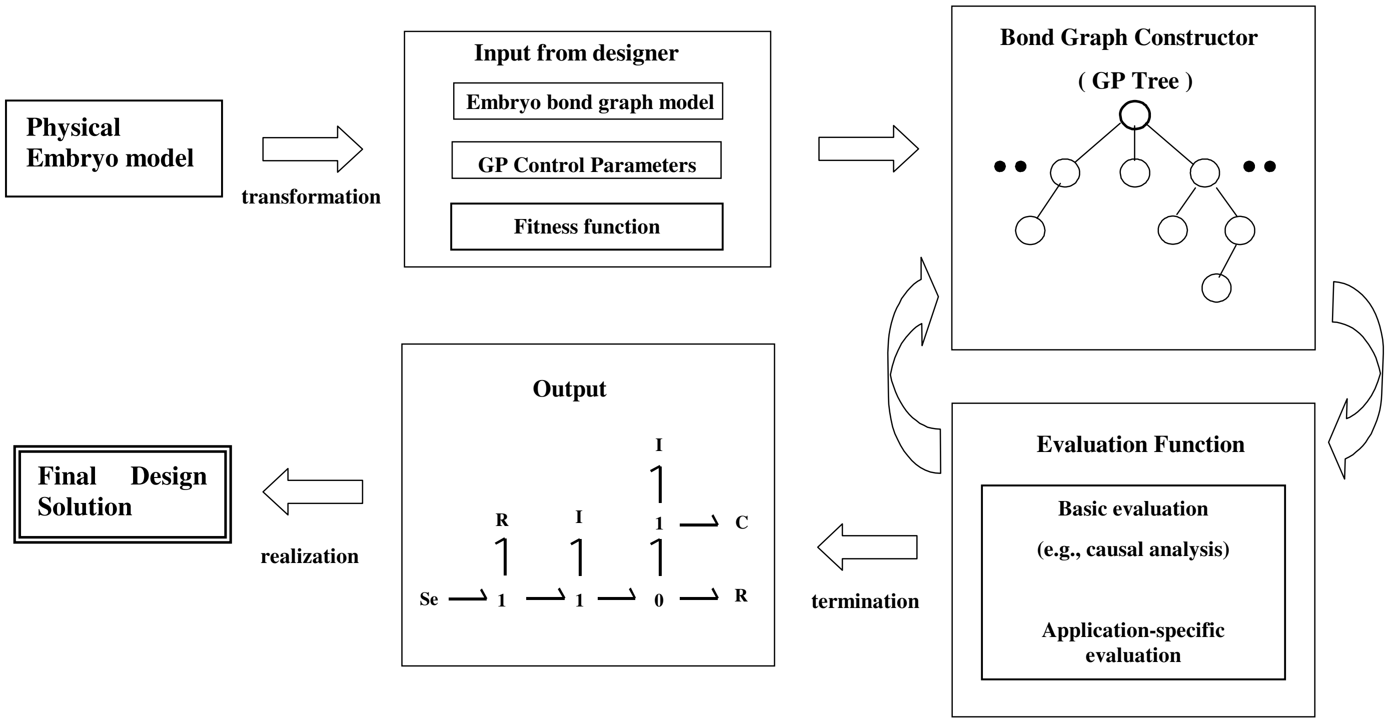


Figure 5. The GP design procedure

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)

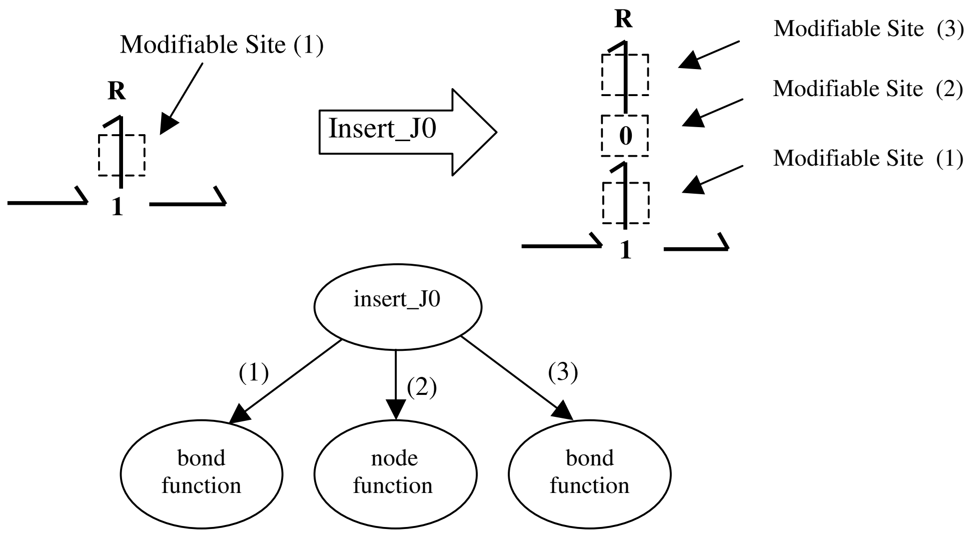


Figure 6. The insert_J0 function

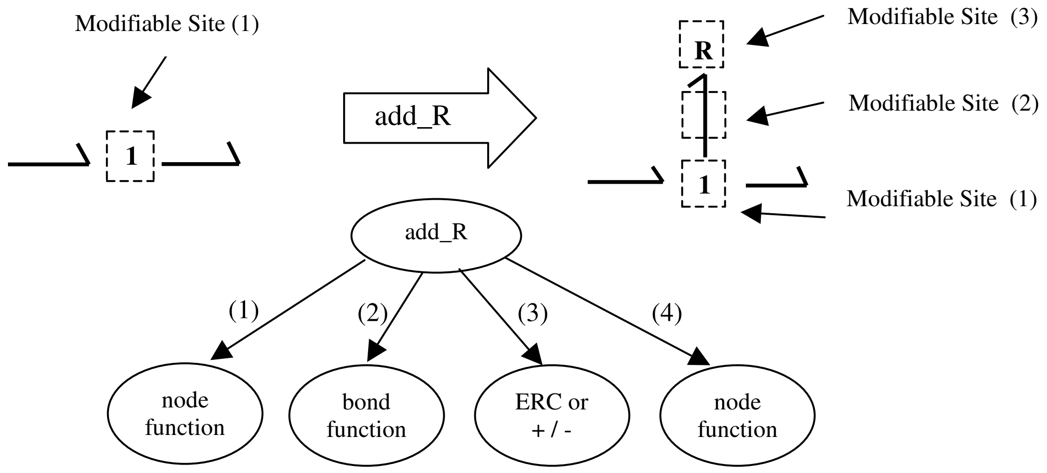


Figure 7. The add_R function

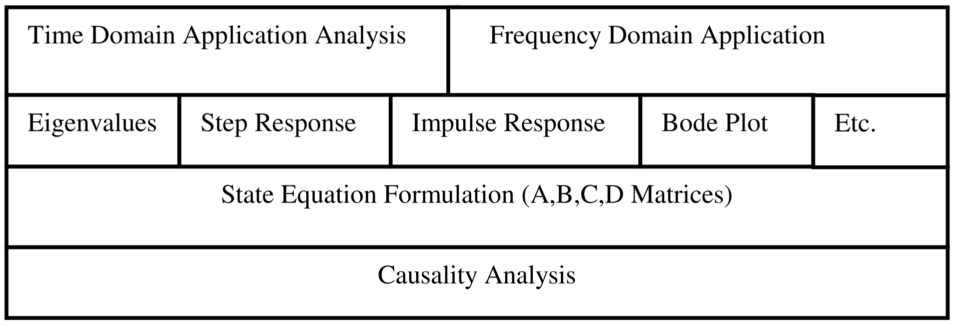

Time Domain Application Analysis
Frequency Domain Application
Eigenvalues
Step Response
Impulse Response
Bode Plot
Etc.
State Equation Formulation (A,B,C,D Matrices)
Causality Analysis

Figure 8. Evaluation layers for bond graph models

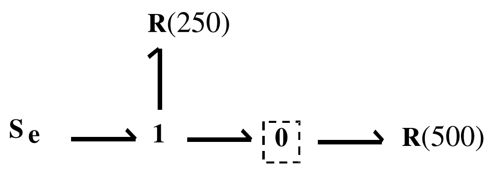


Figure 9. Embryo bond graph model

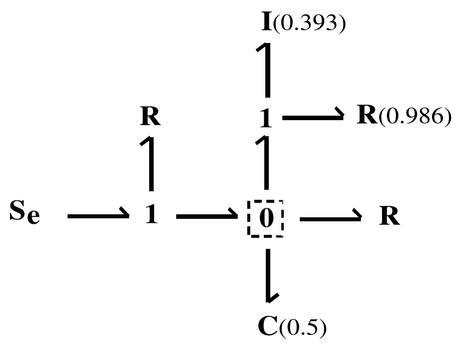
Target eigenvalues : -1 ± 2j Solution eigenvalues : -0.993 ± 2.031j Average distance error : 0.032 added junctions : 1 added elements : 3 added bonds : 4	 I(0.393) R 1 R(0.986) Se 1 0 R C(0.5)

Figure 10. Two eigenvalues result

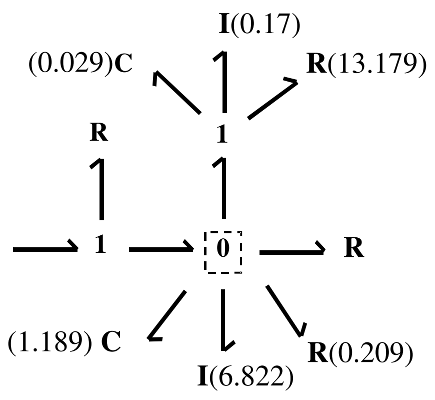
Target eigenvalues : -1 ± 2j, -2 ± j Solution eigenvalues : -0.996 ± 1.997j, -1.989 ± 1.001j Average distance error : 0.008 added junctions : 1 added elements : 6 added bonds : 6	 I(0.17) (0.029)C R(13.179) R 1 Se 1 0 R (1.189) C I(6.822) R(0.209)

Figure 11. Four eigenvalues result

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 : 0.026

added junctions : 4

added elements : 10

added bonds : 14

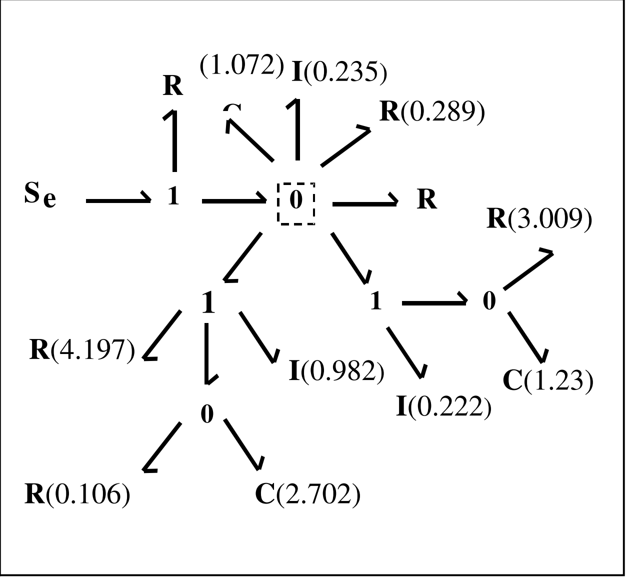


Figure 12. Six eigenvalues result

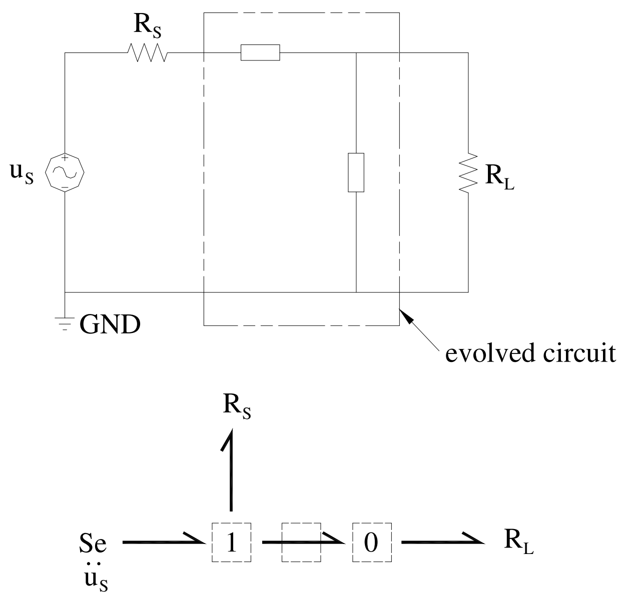


Figure 13. Embryo circuit and its bond graph representation

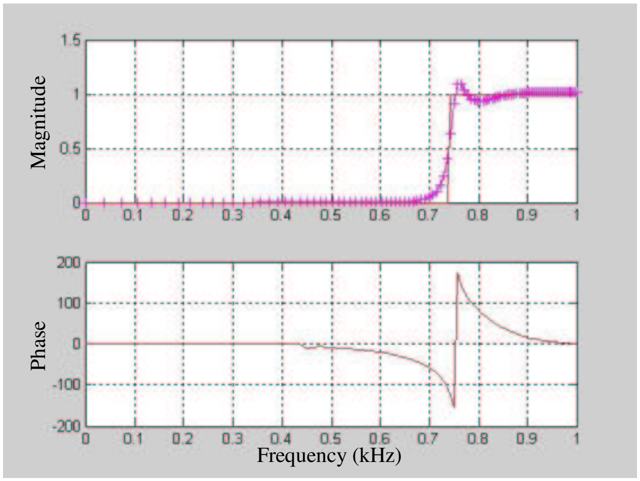


Figure 14. Frequency response of evolved high-pass filter

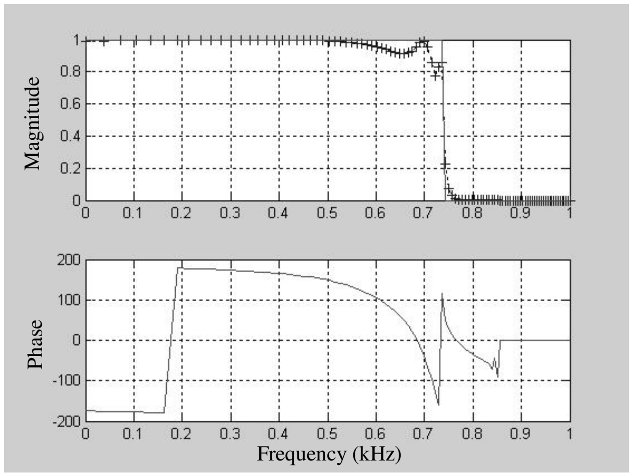


Figure 15. Frequency response of evolved low-pass filter

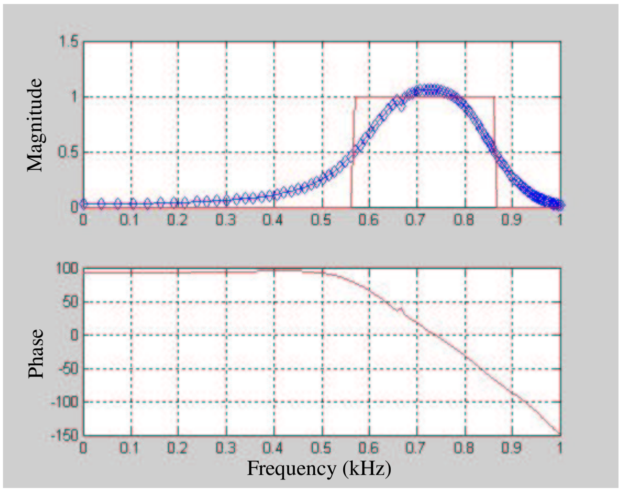


Figure 16. Frequency response of evolved band-pass filter

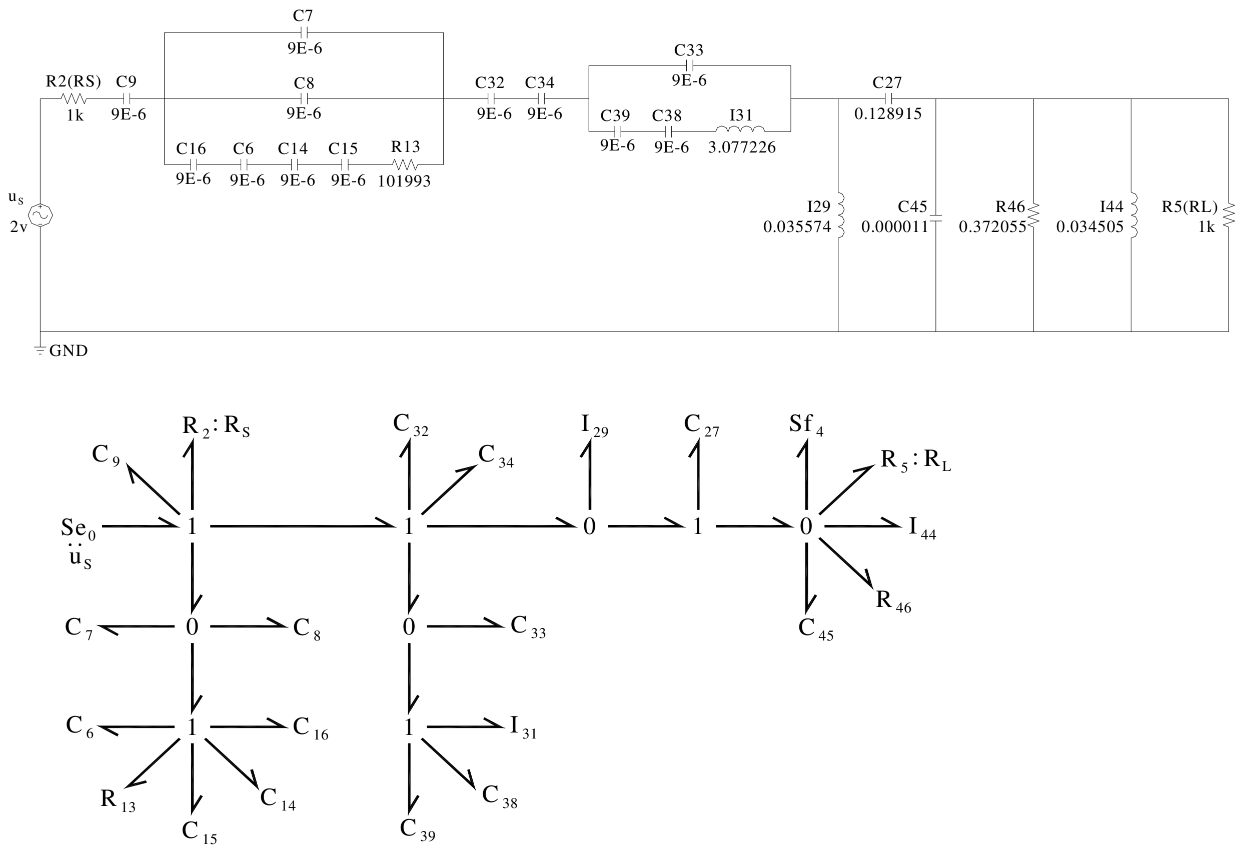


Fig 17. Evolved high-pass filter circuit and bond graph representation