Topological Search in Automated Mechatronic System Synthesis Using Bond Graphs and Genetic Programming

Jianjun Hu, Erik Goodman, Ronald Rosenberg

***Abstract*—We have introduced a well-defined scalable benchmark problem—the eigenvalue placement problem—to investigate scalability issues in automated topology synthesis of mechatronic systems based on bond graphs and genetic programming. This classical inverse problem shares characteristics with many other system synthesis problems, such as electric circuit and controller synthesis, in terms of epistasis and multi-modality of the search space. Critical issues of open-ended topology search by genetic programming are investigated, including encoding, population seeding, scalability and evolvability. For the eigenvalue problems, we have found there exists a correlation between structure and function that is important for efficient topology search. Standard genetic programming has been used to solve up to 20-eigen-value problems, finding the target system of bush topology out of 823,065 possibilities with only 29506 topology evaluations.**

I. Introduction

Computational synthesis as a solution to open-ended design problems has received increasing attention from many design areas [1]. Notably, many success stories with automated evolutionary synthesis have been recently reported, including automatically synthesized electric circuits infringing several recent patents [2], an automatically created general-purpose non-PID controller [2],[3] for which a patent is sought, mechatronic systems [5], [6], and digital circuits [10]. It can be observed that a large number of open-ended design problems can be summarized as: given a set of heterogeneous functional building blocks, connection/combination rules or constraints, and the functional requirements, how can one synthesize a solution to achieve the desired system behavior, automatically or interactively? The difficulty lies in that one needs to identify the appropriate topology of the system as well as its appropriate parameters, *simultaneously*, from an extraordinarily large search space, to achieve the desired behavior.

Despite the apparent difficulty, the examples cited above have been offered as an existence proof of the adequacy of genetic programming for addressing that challenge in some areas of open-ended design innovation [2]. However, critical issues remain concerning effective design of such a tool for open-ended topological exploration, leaving the prospective practitioners with many seemingly arbitrary decisions to make. While there is increasing concern over the scalability of generic evolutionary synthesis, in applications such as evolvable hardware [12], and digital circuit design [13], there are few explicit theoretical and experimental studies in the context of topological search *per se* on issues like appropriate representations, appropriate levels of building blocks, topological operations employed in the search process, scalability, and balanced topological and parameter search. One reason can be traced to the difficulty of establishing common benchmark problems, accessible to all researchers, requiring relatively small amounts of computational resources, and representative of multiple design domains.

In this paper, we introduce a set of carefully designed open-ended system synthesis benchmark problems and investigate issues involved in search-based automated topology synthesis—in particular, in genetic programming. Our goal is to enable synthesis of high-performance systems with several dozen components within a few CPU-days of computing effort, rather than taking several days of computing by hundreds of PCs, as Koza typically uses [2]. In this paper, systems to be synthesized are represented by bond graphs [14] – a multi-domain modeling scheme which can unify design domains such as analog circuits, mechatronic systems, and controllers. One of our test problems is called the eigenvalue placement problem, which has a tunable problem size in terms of order of a dynamic system. We adopt a systematic divide-and-conquer strategy to investigate critical issues in evolutionary topology search by genetic

This work was supported in part by the Dissertation fellowship program of College of Engineering, Michigan State University, East Lansing, MI, 48823 and by National Science Foundation under contract DMI0084934.

Jianjun Hu is with the Department of Computer Science and Engineering, Michigan State University, East Lansing, MI, 48823, USA (phone: 517-355-3796; e-mail: hujianju@ cse.msu.edu).

Erik D. Goodman is with Department of Electrical and Computer Engineering, Michigan State University, East Lansing, MI, 48823, USA (e-mail: goodman@egr.msu.edu).

Ronald Rosenberg is with the Department of Mechanical Engineering, Michigan State University, East Lansing, MI, 48823, USA (e-mail: rosenber@egr.msu.edu).

0-7803-8335-4/04/$17.00 ©2004 AACC

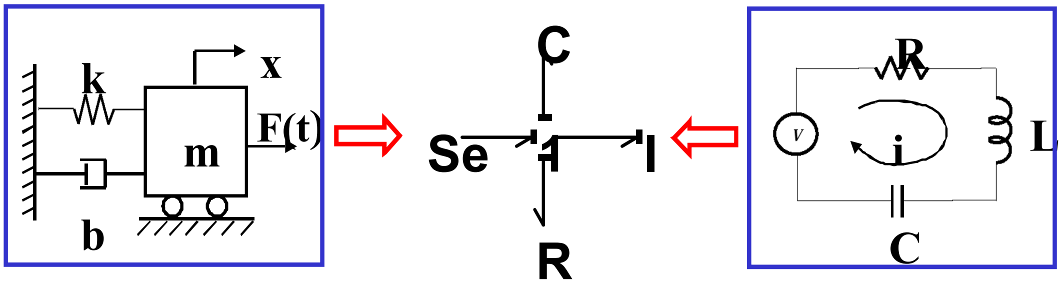


Fig 1. Bond graph as a unified modeling tool for mechanical and electric systems

programming; early progress in the first step, a topology-only search investigation, is reported in this paper.

II. Bond Graph-based System Synthesis by Genetic Programming

A. Bond Graphs

The bond graph is a multi-domain modeling tool for analysis and design of dynamic systems, especially hybrid multi-domain systems including mechanical, electrical, pneumatic, hydraulic, etc. [14]. One of the advantages of using bond graphs for open-ended design exploration is that complex loops typical in electric circuits can be transformed into tree-like structures by the bond graph's 1-junction (serial connection) and 0-junction (parallel connection) concepts, which tend to be easier to evolve in general. Details of notation and methods of system analysis related to the bond graph representation can be found in [14]. Many researchers have explored the bond graph as a tool for design [11], [15]. Design of controllers by augmenting bond graphs with signals (as used in "block-diagram" representations) has also been practiced [16]. Fig 1 illustrates a bond graph that represents either of the accompanying electrical or mechanical systems.

B. Automated synthesis by genetic programming

Genetic programming (GP) [4] is an extension of the genetic algorithm. The distinctive feature of genetic programming is that it aims to evolve variable-length open-ended structures such as computer programs, so is widely used in open-ended problems. We use a developmental strongly-typed genetic programming method as described in [4] to evolve a program tree, whose pre-ordered execution will grow a given minimal embryo bond graph into a complex design solution. Fig 2 illustrates how an embryo bond graph (a) can be developed into a complex solution (d) by executing a program (c) which manipulates the topology and parameters of the growing embryo at each step. By iterative application of evaluation, crossover, mutation, and selection operators, a genetic programming can evolve a population of better and better bond graph-generating GP trees as the one in Fig 2.(c). Further details can be found in [6]. It is important to note that the embryo, the function set, and the type of modifiable sites are all factors influencing the search efficiency.

III. The Eigenvalue Placement Problem: A Benchmark for Automated System Synthesis

Designing a good benchmark problem for investigating open-ended system synthesis is difficult for several reasons. The generic electrical circuit synthesis problem of Koza *et al.* [4], widely known as it is, requires a complicated simulation package such as SPICE, and the simulation complexity is fairly high. Other topology synthesis problems, such as molecule design, suffer from the lack of a "standard" and universally available simulation engine. Communication network design [7], neural network design [8] and other pure graph-oriented topology synthesis procedures [19], however, have quite different characteristics in terms of the search space and the fitness landscape. As a result, we introduce a set of bond graph-based benchmark problems for evaluating methods and identifying issues in automated evolutionary synthesis of mechatronic systems.

The benchmark problem proposed is called the eigenvalue placement problem, in which an analog circuit represented as a bond graph model is to be synthesized (including its topology and sizing of components) to achieve a specific behavior. The bond graph model domain to be used is composed of a set of inductors (I), resistors (R), capacitors (C), transformers (TF), gyrators (GY), and Sources of Effort (SE). Our synthesis task in the K-eigenvalue placement problem is to evolve a dynamic system with K eigenvalues that approximate a pre-specified set as closely as possible. By increasing the number of eigenvalues (K) (and, correspondingly, the order of the dynamic system), we can define a sequence of synthesis problems of increasing difficulty, for use in evaluating the scalability of various techniques. The problem difficulty can also be varied by choosing different sets of primitive building blocks (modules) to be used in the search process. In this paper, we use a set of causally well-posed primitives (CWP), in which each junction has a resistor attached, 1-junctions always have an inductor attached, 0-junctions have a capacitor attached, 0-junctions can only be adjacent to 1-junctions, and vice-versa. Such systems are readily realized in the form of mechanical systems as well as in the electrical domain. The space of all possible topologies generated using these

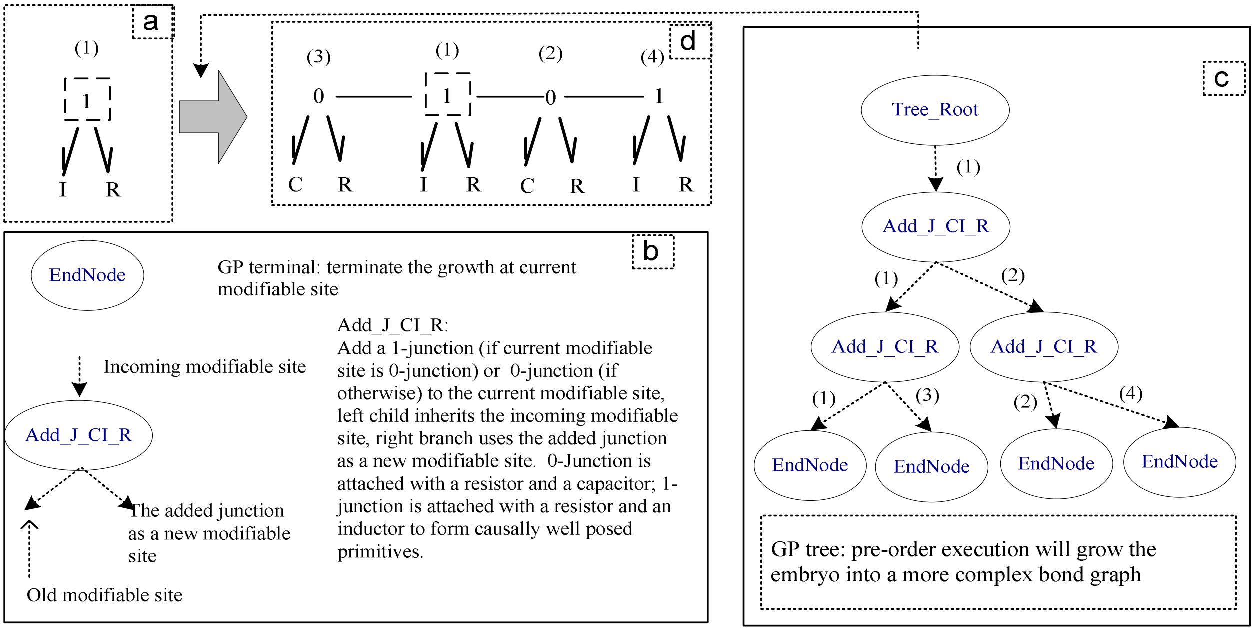

a
(1)
1
I
R
d
(3)
(1)
(2)
(4)
0
1
0
1
C
R
I
R
C
R
I
R
b
EndNode
GP terminal: terminate the growth at current modifiable site
Incoming modifiable site
Add_J_CI_R
Add_J_CI_R:
Add a 1-junction (if current modifiable site is 0-junction) or 0-junction (if otherwise) to the current modifiable site, left child inherits the incoming modifiable site, right branch uses the added junction as a new modifiable site. 0-Junction is attached with a resistor and a capacitor; 1-junction is attached with a resistor and an inductor to form causally well posed primitives.
The added junction as a new modifiable site
Old modifiable site
c
Tree_Root
(1)
Add_J_CI_R
(1)
(2)
Add_J_CI_R
Add_J_CI_R
(1)
(3)
(2)
(4)
EndNode
EndNode
EndNode
EndNode
GP tree: pre-order execution will grow the embryo into a more complex bond graph

Fig 2. (a) The embryo bond graph used in all the experiments of this paper—the 1-junction node in rectangle is a modifiable site where a new construct can be added; (b) the function set for bond graph topological modification (topology manipulating operators) called node-encoding approach; (c) An example of the GP tree, composed of topology operators; (d) the developed bond graph after the depth-first execution of (c).

primitives consists of connected graphs without loops (i.e., trees), and the conditions are strong enough to guarantee that the corresponding systems satisfy causality constraints.

Given a bond graph model, the fitness evaluation is conducted as follows:

1) Derive the A matrix of the state space model from the bond graph model [14].
2) Compute the eigenvalues of the A matrix (code to do this is widely available, or a procedure converted from Matlab may be used, for example)
3) Match the resulting eigenvalues to the target eigenvalue set to compute a relative-distance-based error measure by Procedure 1.
4) Calculate the fitness of the bond graph by normalizing that error measure into a fitness value (to be maximized) between 0 and 1 according the following scaling rule:

if *distance*/*order* <0.1 then
  fitness = 0.1 /(0.1+*distance*/*order*)
else *fitness* = 5.05 /(10+*distance*/*order*)

where *order* is the number of energy-storing components (capacitor and inductors).

The fitness scaling measure in 4) above is chosen such that 50% of the fitness range is occupied by error measures greater than 0.1, and the remainder is used for error measures less than 0.1. It is continuous, with a single slope discontinuity, and approaches 1 asymptotically as the error measure approaches 0. This transformation converts the fitness into the classical maximization form of evolutionary computation, without a singularity for an error of 0 (an exact match of candidates to the targets). Note that many evolutionary methods (using tournament or rank-based selection, for example) will not require or be sensitive to the scaling suggested in step 4,

Procedure 1. calculating distance between two sets of eigenvalues

```
Input: target eigenvalue set $(x_i^t, y_i^t), i = 1,...,O_t$,
        where $O_t$ is the number of target eigenvalues;
    candidate eigenvalue set $(x_i, y_i), i = 1,...,O_c$, where $O_c$ is
the number of candidate eigenvalues.
Output: summed relative distances between two sets of eigenvalues
if order (cardinality) of candidate set $O_c$ is different from that of target
  set, $O_t$   return $\infty$
otherwise
  calculate the average length $l_{ave}$ of target eigenvalues (distance from
  the origin), $l_{ave} = \sum_{i=1}^{O_t} l_i$, $l_i$ is the length of $i_{th}$ target eigenvalue
  label all eigenvalues as unmatched
 *distance* ← 0
  for i = 1 to $O_t$
    find an unmatched candidate eigenvalue j, which is closest to target
  eigenvalue i
    label the candidate eigenvalue j as matched
    *distance* ← *distance* + $\sqrt{(x_j - x_i^t)^2 + (y_j - y_i^t)^2} / l_{ave}$
        where $(x_j, y_j)$ is the candidate eigenvalue,
  and $(x_i^t, y_i^t)$ is the current target eigenvalue i.
  end for
  return *distance*;
```

A. Some interesting variants of this problem formulation

It is possible to use this problem formulation in at least two interesting ways. First, a set of eigenvalues may be generated arbitrarily, without knowledge of a system that would have such a set of eigenvalues. This provides an open-ended problem with no known answer and unknown properties. Second, it is possible, and often useful, to start from a known system, calculate its eigenvalues, and use that set as the target, thereby providing a target with at least one known optimal solution (error 0, fitness 1). This allows the user to readily assess the efficiency of the evolutionary design system as a global optimizer.

This problem formulation explicitly specifies the order of the system. It is useful and often pre-specified in real-world applications, as cost often correlates well with order. Alternative formulations could allow candidate systems of higher order than the target to receive finite error measures, but require that unmatched eigenvalues be outside the range of the target set, for example. Another interesting formulation of the problem, with many of the same properties but requiring much less computational effort, would use the coefficients of the characteristic polynomial of the A matrix as the targets to be matched by those coefficients of candidate systems, thereby avoiding the cost of numerically calculating the eigenvalues. However, systems with widely differing eigenvalues can possess only small differences in characteristic polynomials, so the authors have chosen here to study the eigenvalues themselves, with the hope of better illuminating structure/function relationships in the systems evolved.

IV. Exploration of topology-only search in bond graph synthesis

Solution of the dynamic system synthesis problem posed involves the need to solve simultaneously for the topology and parameters of the system—the goodness of a topology cannot be assessed without assigning values to its parameters, and a set of parameters independent of the topology to which they are assigned defines no performance at all. Here we shall begin to explore the properties of the various topology-affecting search operators by looking at a dramatically reduced sub-problem that eliminates the requirement for parameter searching by making them all identical at unity (i.e., all resistors, capacitors, and inductors have numerical parameters of 1.0). To explore such systems, the target eigenvalue sets are determined from assorted topologies with unit parameters for all of their components. Since parameter search is very expensive for high-order eigenvalue problems, this topology-only search defines a much simpler problem, but one that may illuminate properties of the topology search for the more general problems. The function set primitives used here restrict the resulting bond graphs to having C components attached to each and only 0-junctions and I components attached to each and only 1-junctions, which results in systems readily realizable in any energy domain and which need not be checked for being causally well-posed. Of course, it is only a sub-problem of the more general case in which I components might optionally be attached to 0-junctions and C components to 1-junctions, as well.

A. Properties of the topological space in bond graph synthesis with causally-well-posed sets

The size of the topology-only search space, expressed in terms of distinct bond graph junction structures, has been enumerated [18]. Since each 0-junction or 1-junction in the causally well-posed function set used has a determined set of attached components (the 1-junction has I and R, while the 0-junction has C and R), and since the embryo contains a fixed 1-junction, counting the junction topologies also counts the distinct bond graph topologies, and number of junctions and order of system are identical. For systems of order four, there are only two possible bond graphs. For order 8, there are 23 different bond graphs; for 12, there are 551; for 16, there are 19,320, for 18 there are 128,340, and for 20, it is 823,065). It is obvious that the search space increases quickly with increasing number of components. However, since multiple GP trees may map to the same bond graph topology, the genotype search space is actually much larger.

In the following experiments, we choose the three bond graph topologies in Table 1 as our targets:

Table 2 shows the search effort to find the target topologies of an 8, 12, 16, 18, and 20-eigenvalue problems using a generational GP with population size 500 for 8 and 12-eigenvalue problems, and 1000 for 16 and higher order -eigenvalue problems (with 20 runs for each target). Other primary parameters for the GP runs are as follows:

> crossover rate: 0.4; mutation rate: 0.05; tournament selection with tournament size: 7; subtree swap mutation: 0.55 (this operator chooses two separate subtrees of the same individual and swaps them, preserving system order); initialization: ramped-half-and-half initialisation to generate random trees with maximum depth of 5. Maximal evaluations: 100,000 for 8 and 12 eigenvalue problems, 200,000 for 16, 18, 20-eigenvalue problems.

From Table 2, it turns out genetic programming can easily find the target topology for 8 to 20 eigenvalue problems and the node-encoding method is a little biased for bush topologies.

B. Gene compatibility and its influence on the efficiency of topology search

In the experiments above, we used a minimal function set {EndNode, Add_J_CI_R}, which employs a node-encoding approach to cover all possible tree-structured bond graphs with the restricted set of components attached to each junction as illustrated in Fig 2. This method solves the 8-eigenvalue problems easily. However, when we first began

Table 1: Three types of target bond graph topologies (the attached C, I, and R elements are omitted for simplicity)

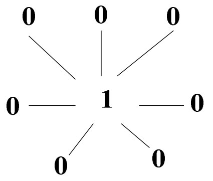
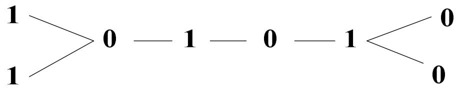
target topology	structure
bush topology	
intermediate topology	
chain topology	1 —— 0 —— 1 —— 0 —— 1 —— 0 —— 1 —— 0

Table 2: Topology search with node-encoding approach for three types of target topologies for 8 to 20-eigenvalue problems.

target topology	mean (std. dev.) order= 8	mean (std. dev.) order= 12	mean (std. dev.) order= 16	mean (std. dev.) order= 18	mean (std. dev.) order= 20
bush	1759(± 385)	3971 (± 858)	19407(± 3189)	26504(± 3470)	29506(± 3687)
intermediate	1466(± 340)	1999 (± 331)	14207(± 2029)	22356(± 3157)	25783(± 3451)
chain	1649(± 384)	3579 (± 677)	27782(± 5326)	35961(± 8058)	46651(± 10588)

Table 3: Searches with hybrid encoding approach for three types of target topologies for 8-, 12-, 16-, 18, 20-, eigenvalue problems, each with 20 runs. Parameter setting is the same with Table 2.

target topology	mean (std. dev) number of evaluations for topology search				
	order 8	order 12	order 16	order 18	order 20
bush	3031 (± 1392)	8 runs succeed 7228(± 2440)	8 runs succeed 40871(± 4908)	6 runs succeed 48920(± 8896)	All failed
intermediate	821 (± 246)	2851(± 1095)	1634 (± 4587)	24987(± 6787)	36549(± 7632)
chain	792 (± 169)	854(± 397)	5133(± 1968)	368(± 2177)	11885(± 4032)

studying this problem, we employed a hybrid encoding approach by allowing both the junctions (nodes) and the bonds (edges) to be modifiable sites, where new structures could be inserted or attached. This approach is widely used by Koza and others [4, 19]. The motivation is that the encoding should be as flexible as possible and every point can be modified. Correspondingly, we added two additional GP functions: a) Insert_J0C_J1I_R, which inserts a 0-junction and 1-junction pair into a bond to maintain causality, and b) EndBond.

Fig. 3 illustrates that all constructs in the bond graph become modifiable sites and thus might be expected to allow more rapid or effective evolution. However, experimental results (Table 3) are quite surprising. While this hybrid encoding approach has no difficulty on the 8 to 18-eigenvalue problems, its bias makes it unable to scale to the 20-eigenvalue problem for bush target topologies. Many more evaluations are needed to find bush topology, and in most cases, it can't find it at all. However, it is pretty good for finding chain topology, using 1/3 of evaluations of node-encoding methods. But in general, node encoding is more robust to search target topologies.

It is apparent that the hybrid encoding method is comparable to node-encoding approach for the easy 8-eigenvalue problems when searching for intermediate and chain topologies, but at the cost of slower search on the bush topology caused by its bias, while the unbiased node-encoding achieves similar performance for all types of topologies. But for the 12-eigenvalue problems, hybrid encoding is much worse than node-encoding when searching for bush-type topologies, 12 runs out of 20 fails to find the target within 100,000 evaluations. After examining where the difficulty comes from, it is observed that the hybrid-encoding approach can find a topology fairly similar to the target topology quite quickly. However, since there are two types of genes in the genome (bond manipulation and junction manipulation), it is highly constrained in *local* manipulations of the genome because of the gene incompatibility. It thus lacks sufficient local topology modification capability. The superficial flexibility in the phenotype space with all possible modifiable sites hides the rigidity in the genotype manipulation capability. In contrast, the node-encoding approach allows very flexible genotype manipulation and thus is able to achieve better results.

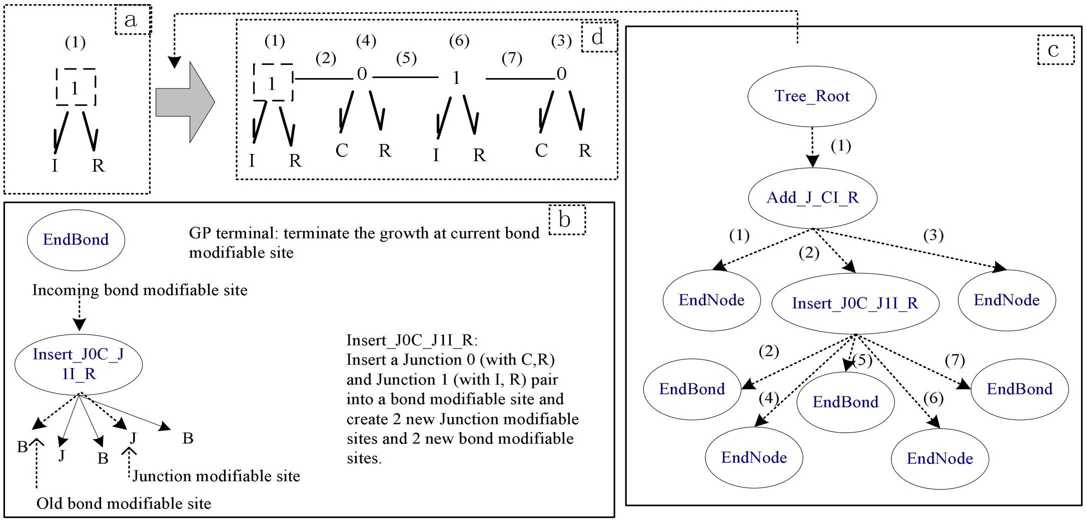

a
(1)
1
I
R
d
(1)
(2)
(4)
(5)
(6)
(7)
(3)
1
0
1
0
I
R
C
R
I
R
C
R
b
EndBond
GP terminal: terminate the growth at current bond modifiable site
Incoming bond modifiable site
Insert_J0C_J1I_R
B
J
B
J
B
Junction modifiable site
Old bond modifiable site
Insert_J0C_J1I_R:
Insert a Junction 0 (with C,R) and Junction 1 (with I, R) pair into a bond modifiable site and create 2 new Junction modifiable sites and 2 new bond modifiable sites.
c
Tree_Root
(1)
Add_J_CI_R
(1)
(2)
(3)
EndNode
Insert_J0C_J1I_R
EndNode
(2)
(5)
(7)
EndBond
(4)
EndBond
(6)
EndBond
EndNode
EndNode

Fig 3. Insert_J0C_J1I_R function and hybrid encoding approach for bond graph evolution

C. Effect of initial population seeding on the efficiency of topology search

One common practice in evolutionary synthesis is to seed the initial population with individuals of a certain property. In [17], the initial population is seeded with different types of components whose proportions are extracted from existing molecule designs. To evaluate the impact of various population initialization methods, a set of topology transformation experiments was conducted. We started from an initial population with identical individuals (bond graph constructor trees) whose execution will generate an identical source topology—e.g., the bush topology. Then we evolved this initially uniform population toward a target topology. The experimental results are summarized in Table 4, with means and standard deviations. Each source-target pair experiment was repeated 20 times, all with same parameters as in Table 2.

Table 4. Topological transformation with different initial homogenous populations

target topology / initial uniform topology	mean (std. dev) evaluations		
	Bush	Intermediate	chain
bush	N/A	2825 (± 411)	3433 (± 396)
intermediate	2348 (± 471)	N/A	1864 (± 359)
chain	3183 (± 294)	2227 (± 273)	N/A

It is interesting that since bush and chain topologies represent the two extremes, transformation between them is most difficult, while transformation from the selected initial intermediate to either chain or bush is much easier. As the intermediate topology is between bush and chain topologies in some sense, our results confirm that seeding with a more similar topology will speed up the evolution. If one does not have any idea of what the topology of the final solution looks like, it is thus advisable to use one or more intermediate topologies as the starting points. Another observation is that genetic programming is able to break the symmetry in the initial population and evolve toward the target topology even if it starts with the opposite topology type, such as evolving from bush to chain topology.

D. Evolvability and scalability of topology search with causally-well-posed primitives

As is generally true, the evolvability of topology search depends strongly on the correlation of the topology with its fitness. The less rugged and deceptive the fitness landscape, with respect to changes caused by single applications of genetic operators, the easier the problem will be to solve. In the eigenvalue problem, we find a correlation between topologies and their functional behavior. We saw above that the 8 to 20-eigenvalue problems were solved fairly quickly; Identifying the target bush topology from 823,065 possible topologies using only 29506 evaluations demonstrates the search capability of GP for topology search. Next, we used the same experiment settings to try to solve 24-eigenvalue problems, except that the population size was increased to 2000, maximum number of evaluations to 500,000, and the maximum tree depth increased to 30. Unfortunately, current node encoding approach failed to find the target bush topology.

This failure can be attributed to several causes. One is the high epistasis and multimodality of the topology search space. The resulting low neutrality and sparseness of intermediate solutions make it extremely difficult to find the target topology. The lack of effective local topology search mechanisms as discussed in section IV.B is another factor. Moreover, the standard genetic programming, with its random crossover and mutation, is random in its choice of directions to explore, and perhaps that is too unstructured to succeed in such cases. It often leads to random scrambling, most of which destroys the framework discovered so far and wastes search effort.

V. Conclusions

We have introduced a well-defined scalable benchmark problem for studying automated synthesis of dynamic systems—the eigenvalue placement problem, based on the bond graph representation. For this problem, we found that similar topologies of bond graphs lead to somewhat similar behaviors in terms of their eigenvalue distributions. This correlation between structure and function of the problem is important for efficient topology search. Without some significant correlation, such inverse problems are essentially intractable.

Using genetic programming as the topology search engine, and on a fixed-parameter reduced version of the problem, we investigated the properties of the topology-only search space, which has quite different properties from other (parametric) search problems, including low neutrality, high multi-modality, discreteness, and lack of information to guide local topology search. We find that standard genetic programming, with node-encoding only, can easily solve up to 20-eigenvalue problems, but has difficulty to scale to 24-eigenvalue problems (with a search space of 39,299,897) or higher. Two encoding schemes, node-encoding and hybrid (node and edge) encoding are compared, applied to the 8- to 20-eigenvalue problems. The result is that node encoding is much better than the hybrid encoding in terms of scalability and robustness. It turns out that incompatibility of the "genes" in the hybrid encoding introduces too strong a bias for bush topology search. However, its bias enables it to find chain topology very efficiently. With topology transforming experiments, we confirmed that appropriate population seeding is beneficial to the search process. If faced with an unfamiliar problem, it appears to be better to seed with diverse topologies. We have intentionally neglected parameter search, which we have treated elsewhere in the context of topology search [9].

Our experiments showed that current developmental genetic programming is powerful for topological search in design innovation. But it still lacks some essential mechanisms to scale up to even larger size dynamic system synthesis problems, at least unless enormous population sizes and numbers of evaluations are used. One promising direction appears to be introduction of mechanisms for framework or module discovery and reuse. A second is a local genotype operator that exploits the locality of topology. Another possible improvement may come from smarter crossover and mutation operators rather than the random swapping and mutation of GP sub-trees.

References

[1] E. K. Antonsson and J. Cagan. *Formal Engineering Design Synthesis*, Cambridge University press, Cambridge, 2001.

[2] J. R. Koza, M. A. Keane, M. Streeter, J., Mydlowec, William, J.Yu, and G. Lanza,.*Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers, 2003.

[3] J. R. Koza, M. A. Keane., J. Yu, F.H. Bennett III, and W. Mydlowec, "Automatic creation of human-competitive programs and controllers by means of genetic programming", *Genetic Programming and Evolvable Machines.* 1 (1 -2), pp.121 – 164, 2000.

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

[5] K. Seo, J. Hu, Z. Fan, E. D. Goodman, and R. C. Rosenberg, "Automated Design Approaches for Multi-Domain Dynamic Systems Using Bond Graphs and Genetic Programming". *The International Journal of Computers, Systems and Signals*, vol.3, no.1, pp.55-70, 2002

[6] K. Seo, Z. Fan, J. Hu, E. D. Goodman, and R. C. Rosenberg, "Toward an Automated Design Method for Multi-Domain Dynamic Systems Using Bond Graphs and Genetic Programming," *Mechatronics*, V. 13, Issues 8-9, pp. 851-885, 2003.

[7] B., Dengiz, F. Altiparmak & A.E. Smith, "Local search genetic algorithm for optimal design of reliable networks",*IEEE Transactions on Evolutionary Computation* v1 n3, September 1997, pp.179-188

[8] K.O. Stanley and,R. Miikkulainen, "Evolving Neural Networks through augmenting topologies", *Evolutionary Computation* 10(2), pp.99-127, 2002.

[9] J. Hu, K. Seo, S. Li, Z. Fan, R. C. Rosenberg, E. D. Goodman, "Structure Fitness Sharing (SFS) for Evolutionary Design by Genetic Programming," *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO-2002, New York, July, 2002, pp. 780-787.

[10] J. F. Miller, D. Job, and V. K. Vassilev, "Principles in the Evolutionary Design of Digital Circuits -- Part I". *Journal of Genetic Programming and Evolvable Machines*, 1(1), 2000, pp. 8-35.

[11] R.F. Ngwompo, S. Scavarda, and D. Thomasset, "Physical model-based inversion in control systems design using bond graph representation", *Journal of Systems and Control Engineering*, 215(12), pp.95-103., 2001.

[12] T.G..M. Gordon and P.J. Bentley (2002), "Towards Development in Evolvable Hardware". In *Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware*, Washington D.C., U.S.A., July 15-18, 2002, pp. 241-250.

[13] V. K. Vassilev and J. F. Miller, "Scalability Problems of Digital Circuit Evolution",in *Proceedings of the 2nd NASA/DOD Workshop on Evolvable Hardware*, IEEE Computer Society, pp. 55-64, 2000.

[14] D. C. Karnopp, D. L. Margolis and R. C. Rosenberg. *System Dynamics: Modeling anrd Simulation of Mechatronic Systems*. Third Edition. New York: John Wiley & Sons, Inc, 2000.

[15] E. Tay, W. Flowers and J. Barrus, "Automated Generation and Analysis of Dynamic System Designs", *Research in Engineering Design*, 10 (1), 15 – 29, 1998.

[16] K. Youcef-Toumi, "Modeling, Design, and Control Integration: A necessary Step in Mechatronics", *IEEE/ASME Trans. Mechatronics*, 1(1), 29-38, 1996.

[17] R. B. Nachbar, "Molecular Evolution: Automated Manipulation of Hierarchical Chemical Topology and Its Application to Average Molecular Structures", *Genetic Programming and Evolvable Machines* 1(1): 57-94, 2000.

[18] F. Harary and E. Palmer. (1973) *Graphical Enumeration*. Academic Press, New York.

[19] S. Luke, and L. Spector, "Evolving Graphs and Networks with Edge Encoding: Preliminary Report". In Koza, John R. (editor), *Late-Breaking Papers at the Genetic Programming 1996 Conference*. Palo Alto, CA.,1996.