Graph Design by Graph Grammar Evolution

Martin H. Luerssen, *Member, IEEE*, and David M. W. Powers, *Senior Member, IEEE*

Abstract— **Determining the optimal topology of a graph is pertinent to many domains, as graphs can be used to model a variety of systems. Evolutionary algorithms constitute a popular optimization method, but scalability is a concern with larger graph designs. Generative representation schemes, often inspired by biological development, seek to address this by facilitating the discovery and reuse of design dependencies and allowing for adaptable exploration strategies. We present a novel developmental method for optimizing graphs that is based on the notion of directly evolving a hypergraph grammar from which a population of graphs can be derived. A multi-objective design system is established and evaluated on problems from three domains: symbolic regression, circuit design, and neural control. The observed performance compares favorably with existing methods, and extensive reuse of subgraphs contributes to the efficient representation of solutions. Constraints can also be placed on the type of explored graph spaces, ranging from tree to pseudograph. We show that more compact solutions are attainable in less constrained spaces, although convergence typically improves with more constrained designs.**

I. Introduction

Natural and artificial instances of systems that can be represented as graphs are ubiquitous and many problems of practical interest may be formulated as questions about graphs. While a variety of graphs are the product of self-organization, other graphs, such as the circuit of a microprocessor, require to be designed. With competent human designers an ever scarce resource, automatic design of graphs is therefore eminently useful. Evolutionary algorithms (EAs) are a class of heuristic optimization algorithms that have been applied to various problems, including design. However, they often scale poorly with the combinatorial explosion of configurations that exist for large graphs. Yet a large graph is not necessarily complex, and this is where self-organization can benefit even the designer. A few simple rules can describe a huge graph if it exhibits some form of regularity. A precedent exists in biological development, where genes (the rules) are expressed into a complex organism (the graph).

This paper begins with a general review of the evolutionary optimization of graph designs and, in particular, the application of developmental methods in this context. We then introduce a simple approach adapted from the formal technique of hyperedge replacement and combine it with a novel algorithm for grammar evolution to produce a design system titled G/GRADE (Graph GRAmmar Design by Evolution), which can capture patterns in evolved graph designs and facilitate their reuse in new solution candidates. As this constitutes a notable step beyond the existing emphasis in EAs on string and tree data structures, we explore the benefits and drawbacks of generalizing to graphs and compare results to other established techniques.

This work was in part supported by the 2006/2007 Flinders University Faculty of Science and Engineering Program Grant.

The authors are with the Artificial Intelligence Laboratory, School of Informatics and Engineering, Flinders University of South Australia, Bedford Park SA 5042, Australia (email: martin.luerssen@flinders.edu.au; powers@ieee.org).

II. Background

A directed graph is a quadruple (V, E, s, t) where V is a finite set of vertices, E is a finite set of edges, and $s, t : E \to V$ assign a source $s(e)$ and a target $t(e)$ to each $e \in E$. Pseudographs are graphs that exhibit loops joining a vertex to itself or multiple edges connecting the same pair of vertices. We will refer to solution candidates as networks, independent of whether they describe simple graphs or pseudographs, except when the distinction is relevant. The most straightforward representation of a network is to directly encode it as an adjacency matrix, the rows of which can be concatenated into a string for optimization by a genetic algorithm. As the string scales with the size of the network rather than its complexity, however, large networks become difficult to optimize even if they exhibit symmetry – a property common to many useful designs.

A. Biological Embryogeny

Biological designs exploit symmetries by employing a generative, highly indirect mapping between the evolved (genotype) and evaluated (phenotype) representations. The developmental process that mediates this, commonly also referred to as an embryogeny [1], is characterized by polygeny (multiple genes define a single phenotypic variable) and pleiotropy (changes to a single gene affect multiple phenotypic variables), which respectively facilitate the neutrality and modularity of design. Neutrality is defined by genotypic variations that fail to affect the phenotype, which has implications for the evolution of evolvability, an effect known as canalization [2]. Canalization is a form of genetic buffering which affects the exploration strategy of evolution by reducing the impact of new mutations and thus allowing a build-up of hidden genetic variation. A change in the selection objective or further variation may break down the canalizing system and lead to more rapid directional change than would otherwise be expected to occur. Neutral variations therefore allow distinct exploration strategies to be encoded in – and ultimately evolved with – the genotype [3]. In contrast, modularity concerns the effective partition of sets into distinct subsets that can be optimized independently [4]. Network designs may be encoded efficiently in terms of modules, thus reducing the dimensionality of the configuration space that must be searched. In conjunction, neutrality and modularity contribute to an adaptive evolutionary process that we know to scale favorably with a variety of challenges.

1-4244-1340-0/07/$25.00 ©2007 IEEE

B. Artificial Embryogeny

Existing computational models of embryogeny differ in their faithfulness to biology. Emphasis is often placed on chemical or mechanical factors of cellular development. Nolfi and Parisi [5] evolved artificial neural networks (ANNs) by modeling the position and branching properties of axonal trees spreading out from neurons. Fleischer [6] and Kitano [7] established more sophisticated simulation frameworks for this purpose, based on equations and rules describing chemical reactions and changes within cells. A crucial aspect of cellular development is gene expression, with genes forming networks of complex interactions termed Genetic Regulatory Networks (GRNs). Numerous notable GRN models have been presented for the purpose of ANN evolution, e.g., [8], [9], [10]; some recent models employ evolved programs [11] or recurrent ANNs [12] to satisfy this role. A common issue here is the complexity involved, which implies not only a considerable computational cost, but also a general difficulty in analyzing such systems (and rarity; we are not aware of any detailed studies).

C. Grammatical Development

We suggest that a proper compromise between the power of a realistic model and the practicality of something simpler is to model embryogeny as a generative grammar. A formal grammar G is a quadruple (N, T, P, S), where N and T are finite sets of nonterminal and terminal symbols, respectively, P is a set of production rules, and S (in N) is an axiom (starting symbol). Each production rule is an ordered pair $p = (P, S)$, where predecessor $P \in (N \cup T)^*$ denotes a string of symbols that is to be replaced by the successor $S \in (N \cup T)^*$. The grammar G defines a formal language L of all the strings that can be generated by a derivation (series of rule applications) from the axiom. Production rules are typically applied in sequence, but L-systems, originally introduced by Lindenmayer [13] to replicate the growth characteristics of plants, rewrite all the symbols of a string concurrently.

Kitano [14] evolved ANNs using a matrix L-system, where each production rewrites a node or edge symbol within a node or edge matrix into a 2×2 node or edge matrix. Boers and Sprinkhuizen-Kuyper [15] used a string L-system to also evolve ANNs by interpreting the final string that results from a given number of rewrites as a graph. The grammar of GENRE [16], an evolutionary design framework based on a parametric L-system, is evolved by a simple EA with specialized operators. Strings are rewritten and then translated into designs, with successful application to table designs, neural networks, and robot controllers.

Graph design as provided above is based on defining a number of operations on graphs and considering a string language of expressions over these operations; a 'graph grammar in disguise' is established [17]. Data structures other than strings are less common; a notable exception is Cellular Encoding, introduced by Gruau [18], where the rewriting rules are represented as a tree evolved by genetic programming (GP) [19]. The nodes of the tree are references to graph operators applied successively to develop a single ancestor cell into a neural network. The choice of operators imposes a bias on the kind of networks that are discovered [20], with both node [18] and edge [21] operators having been advocated for various applications.

D. Hyperedge Replacement

Operating directly on graphs curbs the need for predefined graph operations, since graph operations constitute graph replacements which can be evolved like any other graph. Over the last 30 years a great many graph rewriting techniques have been devised [22]. Hyperedge replacement is one of the most elementary and frequently used techniques and constitutes a solid foundation to work with, as it is rich with theoretical results corresponding to the properties of context-free Chomsky languages [23]. It is a type of edge rewriting extended to hyperedges, which, unlike binary edges, may have multiple sources and targets, $s, t : E \to V^*$, connecting several vertices via a set of incoming tentacles and a set of outgoing tentacles. A graph with hyperedges is known as a hypergraph. Formally, a directed, labeled hypergraph over a label set C is a quintuple (V, E, s, t, l) where:

- V is a finite set of nodes,
- E is a finite set of hyperedges,
- $s : E \to V^*$ assigns sources $s(e)$ to each $e \in E$,
- $t : E \to V^*$ assigns targets $t(e)$ to each $e \in E$,
- and $l : E \to C$ labels each hyperedge.

A multi-pointed hypergraph H is a hypergraph with additional *begin* and *end* nodes, which are also referred to as the external nodes of H. Formally, a multi-pointed hypergraph over C is a septuple $(V, E, s, t, l, begin, end)$ where (V, E, s, t, l) is a hypergraph over C and $begin, end \in V^*_{ext}$. H_c is the set of all multi-pointed hypergraphs. A hypergraph production is an ordered pair $p = (A, R)$ with predecessor, or left-hand side (LHS), $A \in N$ and successor, or right-hand side (LHS), $R \in H_c$. A hyperedge replacement grammar *HRG* is a quadruple (N, T, P, Z) where $N \in C$ and $T \in C$ are finite sets of nonterminal and terminal symbols, P is a finite set of hypergraph productions, and $Z \in H_c$ is the axiom. Given a hyperedge e in a hypergraph H, if there is a hypergraph production $p = (e, R)$ and the begin and end nodes of the multi-pointed hypergraph R match the available attachments in H, then e may be replaced by R by handing over each hyperedge tentacle that is attached to a begin or end node within R to the corresponding source or target attachment node of the replaced hyperedge e.

III. Cellular Graph Grammars

It is generally presumed that the replacement is well-typed, i.e., the hyperedge being replaced has a set of tentacles that match the external nodes of the multi-pointed hypergraph. However, type correctness may be difficult to maintain in the context of evolutionary optimization, as there needs to be scope for productions being reused within and between solution candidates. The handover operation typically fuses the i-th source/target with the i-th begin/end node, so type-correctness could be ignored by simply not trying to fuse

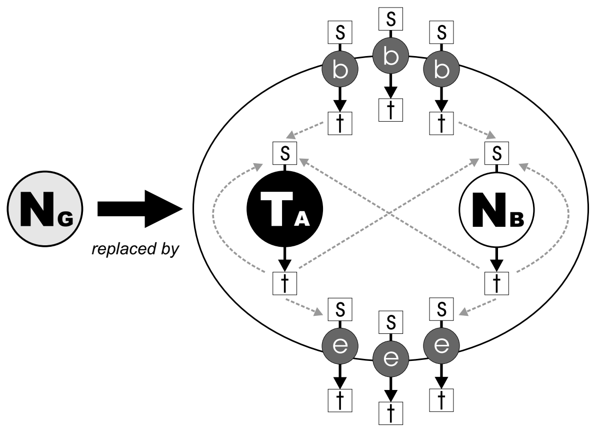

Fig. 1. A diagrammatic representation of a cellular production: nonterminal N_G is replaced by a cellular graph, where T_A is a terminal, N_B is a further nonterminal, b and e are begin and end nodes, and s and t are source labels and target labels of each node. Dotted arrows indicate scope.

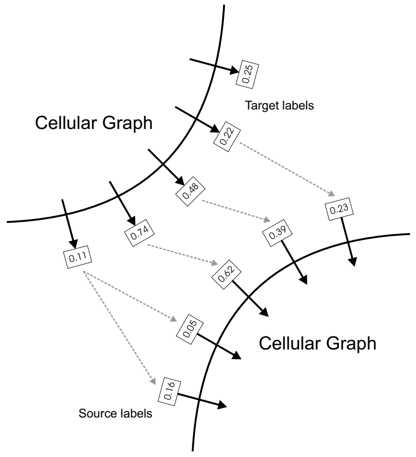

Fig. 2. Soft label matching involves distance-based matching on a large set of labels (shown here as decimal numbers).

any nodes beyond those that are present, but this may lead to position-dependent side effects if the involved hypergraphs are later modified, e.g., during evolution.

Position independence resolves this issue, and it has previously been achieved in the Messy GA [24] by allowing the ordering of genes within a chromosome to evolve. A similar principle can be applied here. An identifying label $l \in C$ is assigned to each external and internal node, so that $l(v)$ is the label of node v. The order of nodes may be restored by using l as an index; however, this achieves position independence only for nodes, not for the mappings s and t, i.e., the tentacles of the hyperedge. A solution is to extend the mappings s and t so that the label l of the external node of the multi-pointed hypergraph is specified; the mappings hence become $s : E(l) \rightarrow V^*$ and $t : E(l) \rightarrow V^*$.

A directed hypergraph can be described by an incidence structure, which contains a point for each vertex or hyperedge of the hypergraph and a line (i, j) if vertex i of the hypergraph is in hyperedge j. We can store this as an adjacency list and, to obtain position independence, decorate each vertex and hyperedge with an identifier as above. A drawback of this approach is that adding or deleting a single element in this list is rarely sufficient to substantially change the graph. We address this by encapsulating those parts of a hyperedge or vertex that define how it attaches to other components into a descriptive unit referred to as a cellular graph, which is illustrated in Fig. 1.

External nodes in the cellular graph are represented as triples (s, t, d), where $s \in C$ is a source label, $t \in C$ is a target label, and $d \in \{0, 1\}$ is the tentacle directionality, either incoming (for begin nodes) or outgoing (for end nodes). A cellular graph G is defined as a triple (N, T, X) where $N \in C$ is a finite set of nonterminal symbols (i.e., hyperedges), $T \in C$ is a finite set of terminal symbols (i.e., internal vertices), and X is a finite set of external nodes. G_c constitutes the set of all cellular graphs. A cellular production is an ordered pair (A, G) with $A \in N$ and $G \in G_c$. Cellular productions can be treated as simple hypergraph productions in a hyperedge replacement system, except that all edges need to be explicitly defined by cellular graphs. To satisfy this requirement, each terminal must manually be wrapped into a user-defined cellular graph, which acts as an interface specification. A network is constructed from a grammar of cellular productions by replacing each nonterminal (and terminal) by the associated cellular graph, as shown in Fig. 3. Each cellular graph may be glued to other expressed cellular graphs to form a cohesive network. For this, fusion between begin and end nodes is established by finding target labels that match source labels.

It was previously suggested that a system that can be decomposed into modules may be more easily optimized. For this to be practical, the representation of modules must be accounted for. A module is expected to have minimal dependences with components external to the module. These dependencies usually relate to a well-specified interface of the module that acts as a dependency bottleneck. In the graph domain, achieving structural modularity translates into restricting the number of vertices inside a module that have edges to vertices outside the module. The begin and end nodes of the multi-pointed hypergraph provide a natural feature for restricting such edges, since it is only these nodes that allow binding to components external to the hypergraph. When matching labels, we thus restrict ourselves to a specific scope for each label type. This is shown in Fig. 1 and represents the baseline scope of a cellular graph. No label outside the scope boundary is visible from within the cellular graph, which, for a graph composed of many cellular graphs, greatly reduces the number of possible sources and targets for which labels must be matched.

The most apparent interpretation of the term "match" is to mean that both labels are identical. However, this approach has previously been found to perform rather poorly, and a "soft" matching scheme has been suggested instead [25] –

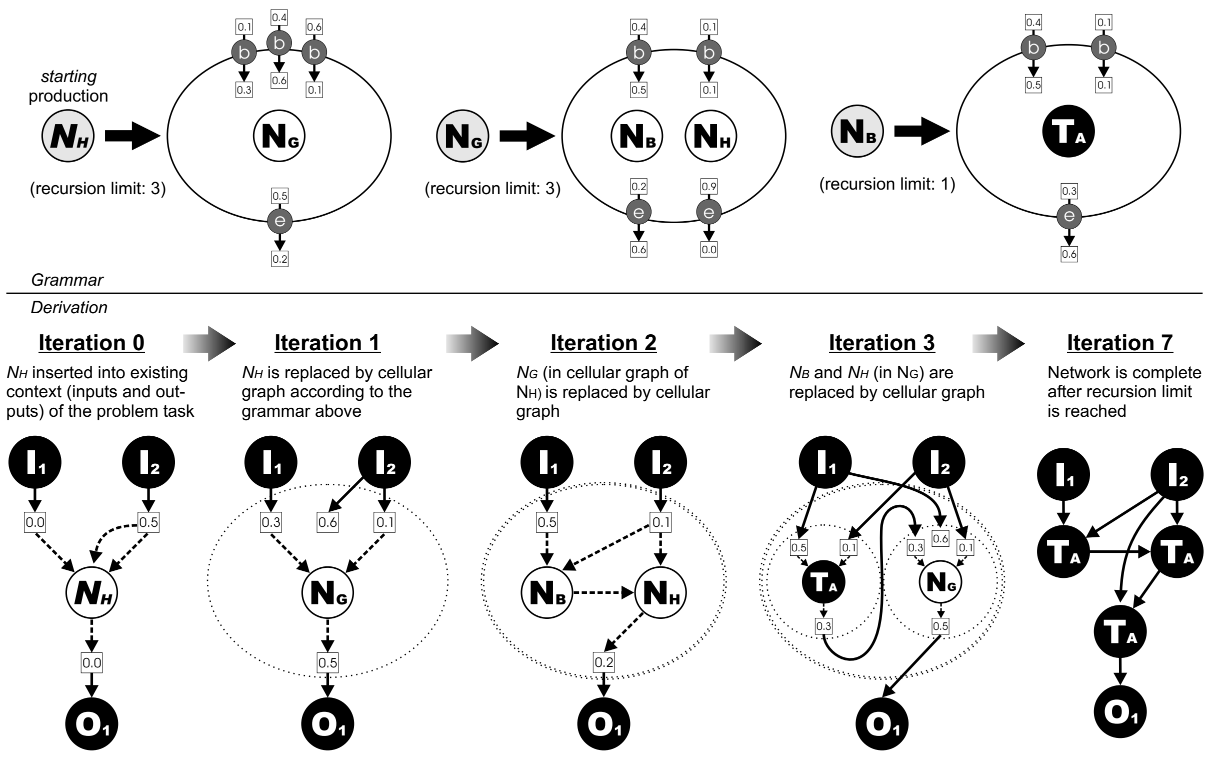
starting production
(recursion limit: 3)
(recursion limit: 3)
(recursion limit: 1)
Grammar
Derivation
Iteration 0
Iteration 1
Iteration 2
Iteration 3
Iteration 7
NH inserted into existing context (inputs and outputs) of the problem task
NH is replaced by cellular graph according to the grammar above
NG (in cellular graph of NH) is replaced by cellular graph
NB and NH (in NG) are replaced by cellular graph
Network is complete after recursion limit is reached

Fig. 3. An example network is derived from a cellular graph grammar over several iterations of replacement.

labels are selected from a very large set and matched with the nearest, not necessarily identical, label. Arithmetic difference is used here as the distance metric (see Fig. 2). Offset labels, which add to all the labels of associated cellular graphs, can also be applied to terminals and nonterminals. Labels may therefore change and combine in various ways without affecting the phenotype. Subgraphs can be partially or fully disconnected from the host graph, allowing building blocks to neutrally accumulate and later be activated through possibly minor label changes.

A. Variation Operators

Cellular graphs are changed randomly during evolution by applying simple mutations. For this, the components of a cellular graph are organized into a list of nonterminal symbols (hyperedges of the graph), a list of terminal symbols (terminal nodes of the graph), a list of $(s, t, 0)$ label triples (begin nodes of the graph), and a list of $(s, t, 1)$ label triples (end nodes of the graph). Having separate lists allows separate mutation probabilities to apply to each component type. Two operators may be applied to each list:

- *insert*, which adds a new element into a random position in the list, where the new element is defined by randomly selecting a new symbol and new labels from a global set of all possible choices
- *remove*, which randomly selects an element from the list and deletes it

A probability is assigned to each *(operation, list)* pair, so that all probabilities sum to 1. A mutation involves randomly selecting an *(operation, list)* pair from these probabilities. The above operators are supplemented by the *increase recursion* and the *decrease recursion* operators, which increase or decrease the recursion limit of the cellular production by one. The recursion limit prevents infinite recursion by defining the maximum recursion depth of a production calling itself during derivation.

B. Shared Grammar Evolution

In the L-systems discussed in Section II-C, a grammar is evolved for each population member, which has a notable drawback: since productions need to be in the same grammar if they reference each other, any group of interacting productions is likely destroyed by crossover operations between genotypes. Consequently, instead of distributing productions across multiple grammars, G/GRADE maintains just a single shared production set [26]. This representation is fully deterministic, as each predecessor is unique and axioms are specially tagged starting productions whose expression leads to a previously evaluated network.

Evolution with G/GRADE is achieved by the algorithm shown in Fig. 4. For every network derived from its associated starting production, a single expressed production is spontaneously replaced by a mutated variant. Since mutating a production that is expressed by several different networks may result in greater or lesser fitness depending on the graph, the mutations apply specifically to a single network and nowhere else. After testing all the mutated networks, the least fit solutions, both from the mutated set and the existing network population, are eliminated, as are all productions not involved in any fitter solutions.

Grammars have previously found widespread use in evolutionary computation as a means of syntactic constraint [27] or as probabilistic models [28]. These aspects of grammar evolution are also highly relevant, but, because of limited space here, will have to be discussed separately.

C. Multiple Objectives

A network derived from a specific starting production may be evaluated on a given problem task and the performance recorded with this starting production. However, performance is not the only important property of networks; size is another. We define size as the sum of terminals, nonterminals, and external nodes of each instance of each production expressed during network derivation. Having both performance and size objectives implies that there is not one optimal solution, but a set of compromises. Multi-objective evolutionary algorithms (MOEAs) can be employed to find this set, with the majority of recently published MOEAs based on Pareto-domination. The implementation for G/GRADE matches the NSGA-II [29], except that we define population density as the distance between a solution and its nearest neighbor. Controlling size with a MOEA has been achieved previously, mainly for trees in GP [30].

New networks in graph grammar evolution are defined from productions that already exist, and these must come from somewhere. The alternative to obtaining diverse building blocks from an initialization method is to generate them during evolution. MOEAs can facilitate diversity if we add diversity as another objective, with the most available target being the solution error. Two solutions i and j are regarded as performing differently if

$$S_{ij} = \begin{cases} 1 & \text{if } \sum_{c=0}^{C} |E_{ci} - E_{cj}| > 0 \\ 0 & \text{otherwise,} \end{cases}$$

where C is the number of fitness cases and E_{ci} is the error of solution i on fitness case c. The diversity of solution i can be determined as an entropy over the proportion of solutions that are different in performance,

$$H(i) = -\log(1 - \frac{\sum_{j=0}^{N} S_{ij}}{N}). \quad (1)$$

This phenotypic objective has been found to be a good compromise between efficiency and observed performance improvements in evolutionary convergence [26].

D. Graph Constraints

Graphs or pseudographs are the most natural representation of a solution for a variety of problems, but G/GRADE is also capable of solving any problem that necessitates strings or trees, since the string/tree space is a subset of pseudograph space. The flexibility of representation may have some benefits, e.g., the arithmetic concept of $2x$ can be represented by two edges both incident on nodes representing an adder and an x. In the absence of such shortcuts, however, the larger search space of pseudographs is expected to lead to slower convergence. Being able to constrain the evolvable data structures for a given problem may hence

```
FUNCTION Evolve-Grammar($C$,$G$,$P_t$)
  $C$ – problem cases
  $G$ – global production set
  $P_t$ – network performance statistics at generation $t$
while termination conditions not met do
  generation $t \leftarrow t+1$
  $S \leftarrow$ all starting productions in $G$
  for starting production $s \in S$ do
    (assume a separate S loop for each statement)
    $E(s) \leftarrow$ productions expressed when deriving $S$
    $O(s) \leftarrow$ random choice of productions from $E(s)$
    for production $o \in O(s)$ do
      $m \leftarrow$ copy of $o$
      apply mutations to $m$
      $M(s) \leftarrow M(s) \cup \{m\}$
    $X(s), R(s), P_t(s) \leftarrow$
      Derive-Graph($s$,$G$,$O(s)$,$M(s)$)
    for problem case $c \in C$ do
      $P_t(s)$.errors $\leftarrow$ simulate $X(s)$ on $c$
    for generation $r \leftarrow t-1$ to $t$ do
      $P_r(s)$.diversity $\leftarrow$ compute diversity of $P_r(s)$
        against all $P_t$ and $P_{t-1}$
      $N_r(s) \leftarrow$ compute nondominance of $P_r(s)$
        against all $P_t$ and $P_{t-1}$
    if $N_t(s)$ within $n$ best of $N$ then
      $G \leftarrow G \cup$ {copy of productions in $R(s)$}
      apply changes in $R(s)$ to $G$
    else
      delete $M(s)$
    if $N_{t-1}(s)$ not within popmax best of $N$ then
      $s$.LHS loses axiom status
  delete productions not called by axioms
return updated productions G, performances $P_t$

FUNCTION Derive-Graph($p$,$G$,$O$,$M$)
  $p$ – expressed production
  $G$ – global production set
  $O$ – productions to be replaced with $M$
  $M$ – productions to replace $O$ by
for nonterminal $n \in p$ do
  while $i <$ O.length do
    if $O(i)$.LHS $= n$ then
      $X, R, P \leftarrow$ Derive-Graph($M(i)$,$G$,$O$,$M$)
      $R \leftarrow R \cup \{(i,\ M(i).LHS,\ s.LHS)\}$
    else
      $sn \leftarrow$ production from $G$ with LHS $= n$
      $X, R, P \leftarrow$ Derive-Graph($sn$, $G$, $O$, $M$)
      if $R$ modified then
        $R \leftarrow R \cup \{(i,\ O(i).LHS,\ s.LHS)\}$
$P$.size $\leftarrow$ size of network $X$ after extension by $p$
return network $X$, replacement list $R$, performance $P$
```

Fig. 4. Simplified pseudocode for grammar evolution via graph derivation with applied mutations.

TABLE I

DEFAULT PARAMETERS FOR ALL PROBLEM TASKS. (SEE FIG. 5 FOR SOLUTION EXAMPLES.)

	Binomial-3 Regression	**2-bit Multiplier**	**Pole Balancing**	**Binary Sequence**
Objective	Infer the mapping $y = f(x)$, where $f(x)$ is the binomial-3 polynomial $(x+1)^3$	Design a Boolean circuit that multiplies 2×2 bit arguments into a 4-bit number	Optimize topology and weights of a neural network balancing 2 poles fixed to a cart moving on a finite track	Reproduce a binary time sequence
Terminals	0/1/2-ary: +, -, $\times$, % (protected division)	0/1/2-ary: AND, XOR	neurons with transfer function $\varphi(x) = \frac{1}{1+e^{-4.9x}}$	0/1/2-ary: AND, XOR
Fitness Case(s)	21 equidistant points generated by the objective function over the interval of $x = [-1, 1]$	All 16 combinations of the 4 Boolean arguments	Pole balancing setup and simulation employed in a previous study by Stanley and Miikkulainen [31]	16-bit sequence given by a 4-bit de Bruijn Counter (with seed 0000)
Simulation	Tree/Acyclic Graph: single pass; Graph/Pseudograph: relaxed for 10 cycles		Relaxed for 3 cycles; weights are assigned to edges by randomization with a standard Gaussian distribution ($\mu = 0$, $\sigma = 1$), at 0.3 probability, or by differential evolution [32], at 0.7 probability, with parameter $F = 0.2$ and a crossover probability of 0.9	Simulation for 32 simulation cycles (+4 cycles lead-in), sampled every 2 cycles; to allow many different designs to be synchronized with the sampling rate, line delays are assigned to edges with a geometric probability of 0.5 of longer delays
Error Measure	Mean squared error		Reciprocal number of cycles both poles remain balanced	Proportion of incorrectly reproduced bits
Mutation	A single production is selected for mutation and a single mutation is applied at a time, with *insert* mutations applied at twice the rate as *remove* mutations, recursion mutations at half this rate again, and a geometric probability of 0.5 that further mutations are applied			
Population	20 networks, each defined by a maximum of 1000 productions and 1000 terminals per production			

improve on this. Cellular graph grammars naturally produce pseudographs, as multi-edges can be formed between any two nodes. The system can be constrained to produce simple graphs by requiring the external nodes of a cellular graph to always match different nodes. If the node with the closest label has been selected before, then the next-closest node is chosen. Maintaining this approach across all cellular graphs ensures that no multi-edges can form, although at a penalty to position independence.

If the solution domain demands it, an acyclic (i.e., feed-forward) topology can also be imposed on the network by ensuring that nonterminals are limited to only receiving incoming edges from begin nodes, and terminals can receive incoming edges from all nodes except directly from other terminals. Further constraining the grammar into generating only trees requires more than a change to label matching; the changes to the productions themselves must be constrained. We accomplish this by defining the proper template productions, one for each kind of terminal receiving inputs from a number of nonterminals matching the arity of the terminal. The templates have one output and no inputs, terminals are added to represent each possible input, and mutations can only change symbols, not insert or delete them.

IV. EXPERIMENTS

G/GRADE permits optimization of pseudographs, while also maintaining the capability of competing with existing techniques on simpler data structures. An experiment is therefore performed to investigate the efficacy of G/GRADE in general, but also to observe the impact of constraining the search space from pseudographs to graphs and trees. Four tasks described in Table I are selected that encompass different natural requirements of the representation, although all are known to be solvable with trees. Each task is evaluated in tree, acyclic graph, (cyclic) graph, and pseudograph space for up to 5000 generations or until optimal performance has been obtained. Results are averaged over 100 independent runs and presented in Table II. Statistical significance (at $p < .01$), where mentioned in the text, is determined using a non-parametric Wilcoxon rank sum test on the error mean of the best solutions.

V. RESULTS

The best results on all problems apart from the pole balancing are obtained by constraining the search to trees, which is significantly superior to using cyclic designs. Exploring the space of acyclic graphs performs comparably well and is only significantly worse than searching trees on the multiplier circuit and, again, the pole balancing. Pole balancing is indeed the only problem which favors a cyclic design, as the solution can be evolved in a single production. Convergence is thus mainly dependent on the weight evolution scheme, which, however, was not the focus of our efforts here, so these results remain inferior to another recent, specialized pole balancing system requiring a minimum computational effort (MCE) of just 20,918 evaluations [33].

On the other tasks, a general trend can be observed towards improved performance with stricter graph constraints. Notably poor results with the Boolean tasks can be attributed to the size objective, as replacing it with a dynamic size limit has solutions grow considerably (up to 59$\times$ larger), while performance is also significantly lifted, with perfect success rates on the sequence task. The MCE of 34,860 on the multiplier task, or 66,060 with a size objective, compares

TABLE II

PERFORMANCE STATISTICS FOR ALL RUNS, USING SIZE AND DIVERSITY OBJECTIVES.

Experiment		**Success** Rate	**MCE** $\times 1000$	**Min Error** Mean	**Size (0-Error)** Mean	**Size (All)** Mean	**Verbosity** Mean
Binomial-3 Regression	Tree	100%	4	0.0000± 0.0000	44.37± 10.88	27.43± 5.00	0.41± 0.05
	Acyclic Graph	99%	29	0.0003± 0.0030	46.79± 20.08	24.72± 13.71	0.70± 0.10
	Graph	77%	54	0.0062± 0.0150	27.08± 12.72	17.68± 9.09	0.83± 0.12
	Pseudograph	87%	63	0.0050± 0.0171	29.34± 26.95	17.03± 5.88	0.81± 0.12
2-bit Multiplier	Tree	99%	66	0.0003± 0.0031	115.14± 15.03	72.67± 9.65	0.33± 0.04
	Acyclic Graph	16%	1937	0.0175± 0.0165	61.13± 22.50	26.18± 7.02	0.79± 0.11
	Graph	4%	9648	0.0597± 0.0247	30.50± 4.93	18.46± 5.94	0.89± 0.11
	Pseudograph	1%	32965	0.0738± 0.0315	32.00± 0.00	16.46± 4.44	0.90± 0.11
Pole Balancing	Tree	46%	181	0.0013± 0.0017	96.50± 131.21	100.40± 172.44	0.63± 0.21
	Acyclic Graph	9%	3912	0.0041± 0.0028	86.11± 73.42	23.58± 16.65	0.79± 0.15
	Graph	87%	75	0.0003± 0.0010	17.87± 15.03	14.54± 6.05	0.93± 0.11
	Pseudograph	68%	70	0.0013± 0.0025	15.63± 7.30	14.49± 5.73	0.93± 0.11
Binary Sequence	Tree	87%	65	0.0081± 0.0211	103.41± 79.06	50.91± 23.97	0.38± 0.08
	Acyclic Graph	68%	331	0.0206± 0.0308	87.40± 76.24	37.57± 17.44	0.56± 0.17
	Graph	36%	633	0.0450± 0.0378	36.94± 21.34	17.59± 7.66	0.73± 0.13
	Pseudograph	52%	420	0.0319± 0.0350	42.12± 24.59	18.76± 9.55	0.72± 0.14

well with the 136,080 evaluations reported with an extended GP model using the same terminal set [34], although much of this advantage may be due to the diversity selection that is specific to our system. G/GRADE is also a powerful tool for symbolic regression, as we achieve a perfect success rate within 330 generations and a population of 20, whereas a previous detailed study using GP obtained only 84% over 200 generations and a larger population of 500 [35].

More compact solutions are generated as we move from stricter, acyclic graph constraints to cyclic designs. Since fewer optimal solutions are found for the cyclic designs, this may reflect the exponential increase in possible configurations as we reduce the constraints, so only smaller solutions are likely to be found. However, evolution operates here not on networks but on a grammar. The effectiveness of the grammar representation is given as its verbosity, which is the inverse of the average production reuse over the whole population. The overall results show that larger networks inversely correlate with verbosity (Spearman $\rho = -0.92$), but the size increase with lesser constraints is often matched by a reduction in verbosity. Thus, for a given number of generations and size of population, G/GRADE optimizes a similar number of productions for each solution candidate, with the size of the resulting graph partially dependent on the applied graph constraints.

VI. CONCLUSION

For many design problems the most natural representation is the graph, but it is often easier and more transparent to evolve strings and trees. This paper introduces a novel framework for graph evolution that emphasizes scalability and universality, since real-world problems literally come in all shapes and sizes. Earlier research noted that generative representations are an appropriate starting point for this, as they facilitate reuse of design and can incorporate a bias of the design problem into their structure. The system presented here is of this nature, but based on the concept of hypergraph grammars, which allows graph transformations to be adapted as part of the representation. Conversely, it also maintains substantial flexibility in constraining graph spaces to suit the problem task at hand. We provide results in support of this approach, with problems from multiple domains and various levels of constraint ranging from pseudograph to tree. As might be expected, expanding the search space to graphs for problems that can be solved with trees mostly leads to poorer convergence properties, although also to more compact solutions that are for at least one problem – the pole balancing – easier to find. We hence conjecture that graph optimization is only advantageous with problems where a graph representation is substantially more efficient than otherwise, for example in the case of extensive reuse or recurrency. Future work is intended to investigate this further.

Despite the prevalent use of grammars in artificial evolution, it is quite atypical to find a grammar being directly evolved. In the context of graph design, this paper employs an innovative technique for evolving productions of a grammar that describes a population of solutions. Each solution is represented by a production that calls upon other productions, potentially shared with other solutions, to iteratively construct the solution. Representational efficiency is promoted by extensive grammatical reuse, but a downside is the emergence of exceedingly large networks. Co-optimization towards a size objective can inhibit this, but at a potential cost to convergence performance. Overall, this paper should be seen as an early milestone, with more to come, towards a better understanding of graph evolution, grammar evolution, and the intersection of these important new fields.

REFERENCES

[1] K. O. Stanley and R. Miikkulainen, “A taxonomy for artificial embryogeny,” *Artificial Life*, vol. 9, no. 2, pp. 93–130, 2003.

[2] G. Gibson and G. P. Wagner, “Canalization in evolutionary genetics: a stabilizing theory?” *BioEssays*, vol. 22, no. 4, pp. 372–380, 2000.

[3] M. Toussaint, “The evolution of genetic representations and modular neural adaptation,” Ph.D. dissertation, Institut für Neuroinformatik, Ruhr-Universität Bochum, Bochum, Germany, 2003.

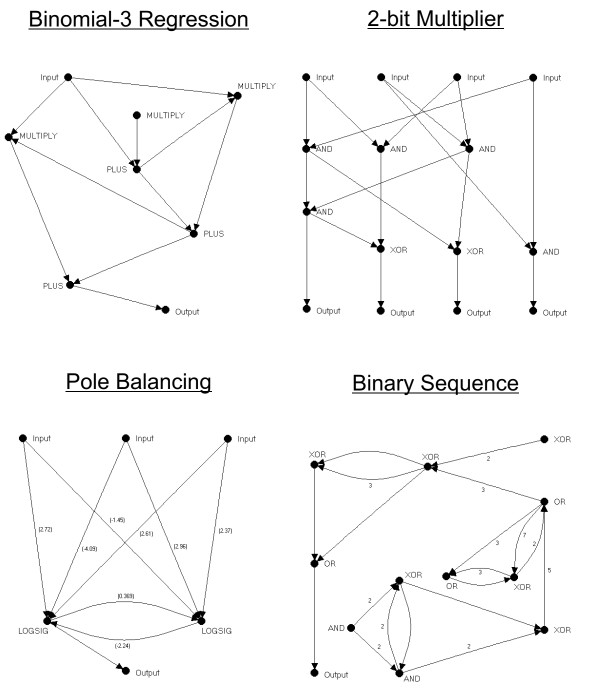

Fig. 5. A sample of error-optimal networks evolved by G/GRADE on the four problem tasks. Connection weights are shown for the pole balancing ANN and line delays for the sequence circuit.

[4] H. A. Simon, *The Sciences of the Artificial*, 3rd ed. Cambridge, MA, USA: The MIT Press, 1996.

[5] S. Nolfi and D. Parisi, "Growing neural networks," Department of Cognitive Processes and Artificial Intelligence, Institute of Psychology, National Research Council, Rome, Italy," Technical Report PCIA-91-15, 1991.

[6] K. W. Fleischer, "A multiple-mechanism developmental model for defining self-organizing geometric structures," Ph.D. dissertation, Department of Computation and Neural Systems, California Institute of Technology, Pasadena, USA, 1995.

[7] H. Kitano, "A simple model of neurogenesis and cell differentiation based on evolutionary large-scale chaos," *Artificial Life*, vol. 2, pp. 79–99, 1995.

[8] D. G. Stork, B. Jackson, and S. Walter, "Non-optimality via pre-adaptation in simple neural systems," in *Artificial Life II: Proceedings of the Workshop on Artificial Life*. Addison Wesley, 1992, pp. 409–429.

[9] J. C. Astor and C. Adami, "A developmental model for the evolution of artificial neural networks," *Artificial Life*, vol. 6, pp. 189–218, 2000.

[10] N. Jakobi, "Harnessing morphogenesis," in *On Growth, Form and Computers*, P. Bentley and S. Kumar, Eds. London, UK: Academic Press, 2003, pp. 392–404.

[11] K. L. Downing, "Developmental models for emergent computation," in *Proceedings of the 5th International Conference on Evolvable Systems*, ser. Lecture Notes in Computer Science, vol. 2606. Springer-Verlag, 2003, pp. 105–116.

[12] M. Federici, "Evolving developing spiking neural networks," in *Proceedings of the IEEE Congress on Evolutionary Computation*. IEEE Press, 2005, pp. 543–550.

[13] A. Lindenmayer, "Mathematical models for cellular interaction in development, parts I and II," *Journal of Theoretical Biology*, vol. 18, pp. 280–315, 1968.

[14] H. Kitano, "Designing neural networks using genetic algorithms with graph generation systems," *Complex Systems*, vol. 4, no. 4, pp. 461–476, 1990.

[15] E. Boers and I. Sprinkhuizen-Kuyper, "Combined biological metaphors," in *Advances in the Evolutionary Synthesis of Intelligent Agents*, M. Patel, V. Honavar, and K. Balakrishnan, Eds. Cambridge, MA, USA: The MIT Press, 2001, pp. 153–183.

[16] G. S. Hornby, "Generative representations for evolutionary design automation," Ph.D. dissertation, Department of Computer Science, Brandeis University, Waltham, USA, 2003.

[17] J. J. Vereijken, "Graph grammars and operations on graphs," Master's dissertation, Department of Computer Science, Leiden University, Leiden, The Netherlands, 1993.

[18] F. Gruau, "Genetic synthesis of Boolean neural networks with a cell rewriting developmental process," in *Proceedings of the International Workshop on Combinations of Genetic Algorithms and Neural Networks*. IEEE Computer Society, 1992, pp. 55–74.

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

[20] G. Hornby, "Shortcomings with using edge encodings to represent graph structures," *Genetic Programming and Evolvable Machines*, vol. 7, no. 3, pp. 231–252, 2006.

[21] S. Luke and L. Spector, "Evolving graphs and networks with edge encoding: preliminary report," in *Late Breaking Papers at the Genetic Programming Conference*. Stanford University Bookstore, 1996, pp. 117–124.

[22] G. Rozenberg, Ed., *Handbook on Graph Grammars and Computing by Graph Transformation: Volume I. Foundations*. River Edge, USA: World Scientific, 1997.

[23] A. Habel, *Hyperedge Replacement: Grammars and Languages*, ser. Lecture Notes in Computer Science. Berlin, Germany: Springer-Verlag, 1992, vol. 643.

[24] D. E. Goldberg, K. Deb, and B. Korb, "Messy genetic algorithms: motivation, analysis, and first results," *Complex Systems*, vol. 3, pp. 493–530, 1989.

[25] M. H. Luerssen and D. M. W. Powers, "Graph composition in a graph grammar-based method for automata network evolution," in *Proceedings of the IEEE Congress on Evolutionary Computation*. IEEE Press, 2005, pp. 1653–1660.

[26] M. H. Luerssen, "Phenotype diversity objectives for graph grammar evolution," in *Recent Advances in Artificial Life*, ser. Advances in Natural Computation, H. A. Abbass, T. Bossamaier, and J. Wiles, Eds. Singapore: World Scientific, 2005, vol. 3, pp. 159–170.

[27] C. Ryan, J. J. Collins, and M. O'Neill, "Grammatical evolution: evolving programs for an arbitrary language," in *Proceedings of the 1st European Workshop on Genetic Programming*, ser. Lecture Notes in Computer Science, vol. 1391. Springer-Verlag, 1998, pp. 83–95.

[28] Y. Shan, R. I. McKay, R. Baxter, H. A. Abbass, D. Essam, and H. X. Nguyen, "Grammar model-based program evolution," in *Proceedings of the IEEE Congress on Evolutionary Computation*. IEEE Press, 2004, pp. 478–485.

[29] K. Deb, S. Agrawal, A. Pratab, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II," in *Proceedings of the Parallel Problem Solving from Nature VI Conference*, ser. Lecture Notes in Computer Science, vol. 1917. Springer-Verlag, 2000, pp. 849–858.

[30] S. Bleuer, M. Braek, L. Thiele, and E. Zitzler, "Multiobjective genetic programming: reducing bloat using SPEA2," in *Proceedings of the IEEE Congress on Evolutionary Computation*. IEEE Press, 2001, pp. 536–543.

[31] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[32] K. Price, "An introduction to differential evolution," in *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, Eds. London, UK: McGraw-Hill, 1999, pp. 79–108.

[33] K. O. Stanley, "Efficient evolution of neural networks through complexification," Ph.D. dissertation, Department of Computer Science, University of Texas, Austin, USA, 2004.

[34] M. Lones, "Enzyme genetic programming: Modelling biological evolvability in genetic programming," Ph.D. dissertation, 2003.

[35] J. M. Daida, R. R. Bertram, S. A. Stanhope, J. C. Khoo, S. A. Chaudhary, O. A. Chaudhri, and J. A. Polito II, "What makes a problem GP-hard? Analysis of a tunably difficult problem in genetic programming," *Genetic Programming and Evolvable Machines*, vol. 2, no. 2, pp. 165–191, 2001.