Graph Composition in a Graph Grammar-Based Method for Automata Network Evolution

Martin H. Luerssen
School of Informatics & Engineering
Flinders University of South Australia
GPO Box 2100 Adelaide 5001 Australia
luer0001@infoeng.flinders.edu.au

David M. W. Powers
School of Informatics & Engineering
Flinders University of South Australia
GPO Box 2100 Adelaide 5001 Australia
powers@ieee.org

Abstract- The dynamics of neural and other automata networks are defined to a large extent by their topologies. Artificial evolution constitutes a practical means by which an optimal topology can be determined. Constructing a grammar of good graphs and then deriving new graphs from this grammar can facilitate this process. The following paper presents a simple but novel method of evolving a hypergraph grammar for this purpose. Different strategies for composing graphs within this framework are evaluated on problems of symbolic regression, time series approximation, and neural networks. The results favour a selectively modular approach that connects nodes with the most similar, rather than identical, labels.

1 Introduction

An automata network (Goles & Martinez 1990) is a system (C, A, S_n, N, L) where C is a set of cells, A is an alphabet of states, and $S_n : C \rightarrow A$ is a state at a discrete time n. S is updated according to $L : C \rightarrow D$ where D is a set of local dynamic rules, which interact with the neighbourhood $N(c)$ of each $c \in C$, to define the global dynamics of the system. Many parallel distributed models of computation, including biological and artificial neural networks (ANNs), are specific instances of automata networks.

An often-overlooked issue in the design of automata networks is how to determine the optimal topology for N. This inevitably requires a search of the space of possible graphs. Evolutionary algorithms (EAs) are a well-established method for searching discontinuous spaces of this kind where little domain-specific knowledge is available (Bäck 1996). However, unless the EA operates on an efficient graph representation, its computational cost will be prohibitive. Hypergraph grammars can represent graphs efficiently by facilitating the discovery and reuse of topological patterns in these graphs.

A system for evolving such a grammar within the context of automata networks is presented by Luerssen (2005). This paper serves as a further elaboration on this system and investigates several different strategies for determining node adjacency during composition of graphs from other graphs.

2 Evolving Graphs

EAs operate on a population of diverse solutions from which an offspring population is generated by applying mutations and/or recombinations; the fittest solutions are then selected to form a new population. Genetic algorithms (GAs) (Holland 1992) explicitly model an additional, genetic level of representation, inspired by the biological mechanism of heredity, which postulates a transferable genotype encoding the phenotype that is hence subject to natural selection (Futuyma 1998).

Evolving a graph requires a way of representing the graph for this purpose. Most of the early research on this issue relates to the evolution of ANN topologies, exemplified by Miller et al. (1989), who employ a classical blueprint encoding as the genetic representation. A topology of M nodes is represented by an adjacency matrix of dimensions $M \times M$ in which element c_{ij} denotes the presence ($c_{ij} = 1$) or absence ($c_{ij} = 0$) of a connection from node i to node j. The simplicity of this encoding has a drawback in that the genotype size scales with the size of the graph rather than its complexity. Consequently, highly regular phenotypes are just as difficult to find as highly random ones, unless the search strategy can exploit correlations between genes. Linkage learning is one such approach (Kargupta and Bandyopadhyay 2000); another is to allow for neutral variations to the genotype which affect only the exploration probabilities, not the phenotype itself (Toussaint 2003).

Since Dawkins (1989) demonstrated that the genotype-phenotype encoding can affect evolvability as well, the idea of evolving a developmental system has been widely explored. Many studies place an emphasis on biological plausibility by simulating various aspects of biological ontogenesis, such as neural morphology, e.g. Cangelosi et al. (1994), or chemistry, e.g. Astor and Adami (2000). This typically comes at a high computational cost. Alternatively, the development process can be abstracted to a grammar that models genes as production rules from which phenotypes are derived. Design regularities are thus implicitly reproduced by relations between these productions, and the indirectness of this encoding allows for neutral variations as well.

Kitano (1990) and Boers and Kuiper (1992) are pioneering examples of evolving grammar-based encodings of graphs. Both optimise ANN topologies using

0-7803-9363-5/05/$20.00 ©2005 IEEE.

Lindenmayer-systems (L-systems), which are parallel string rewriting systems introduced by Lindenmayer (1968) and originally intended for describing plant morphogenesis. Haddow et al. (2001) apply this idea to circuit design, and Hornby (2003) to a more diverse range of structures.

Another common approach to ANN optimisation is Cellular Encoding (CE) (Gruau 1994). CE explicitly represents each developmental step as a node in a tree of graph-transforming operators. Koza et al. (1999) also apply CE to circuit design. The tree is evolved by Genetic Programming (GP) (Koza 1992), but the expressive power of CE depends mainly on the appropriate choice of operators (Luke & Spector 1996). A related variant of GP, Cartesian Genetic Programming (CGP) (Miller & Thomson 2000), directly constructs graphs from nodes with labelled edges. Luerssen and Powers (2003) and Luerssen (2005) extend this concept to a rewriting system, which derives graphs from an evolved hypergraph grammar.

3 Cellular Productions

Edges in a graph typically have arity two, i.e. they connect two vertices. A hyperedge connects several vertices (via tentacles), and a graph with hyperedges is called a hypergraph. Formally, a directed hypergraph over a label set C is a system (V, E, s, t, l) where V is a finite set of nodes, E is a finite set of hyperedges, $s : E \rightarrow V^*$ and $t : E \rightarrow V^*$ assign a sequence of sources $s(e)$ and a sequence of targets $t(e)$ to each $e \in E$, and $l : E \rightarrow C$ labels each hyperedge (Habel 1992).

A multi-pointed hypergraph is a hypergraph with additional begin and end nodes. A hyperedge can be replaced by a multipointed hypergraph by matching these nodes with the respective sources and targets. Let $N \subseteq C$ be the set of nonterminals, $T \subseteq C$ be a set of terminals, and H be the set of all multi-pointed hypergraphs. A hypergraph production is an ordered pair $p = (LHS, RHS)$ with $LHS \in N$ and $RHS \in H$. A hypergraph grammar is a system $HGG = (N, T, P, Z)$ where P is a finite set of hypergraph productions over N and $Z \in H$ is the axiom.

To illustrate the difficulty of evolving a hypergraph grammar, assume a hyperedge nonterminal N_C is added on the *RHS* of production N_G. The edge mappings s and t need to be fully defined for this node, which, without further knowledge on hand, implies randomizing an adjacency matrix matching the begin and end nodes of the *RHS* hypergraph of production N_C. If, however, a begin or end node of this *RHS* hypergraph is later removed by mutation of N_C, then the adjacency matrix for N_G becomes invalid, as shown in Figure 1. Only with additional identifying labels for the tentacles can the matrix correctly map to the new begin and end nodes, but this leads to even larger hypergraph descriptions on the *RHS* of each production.

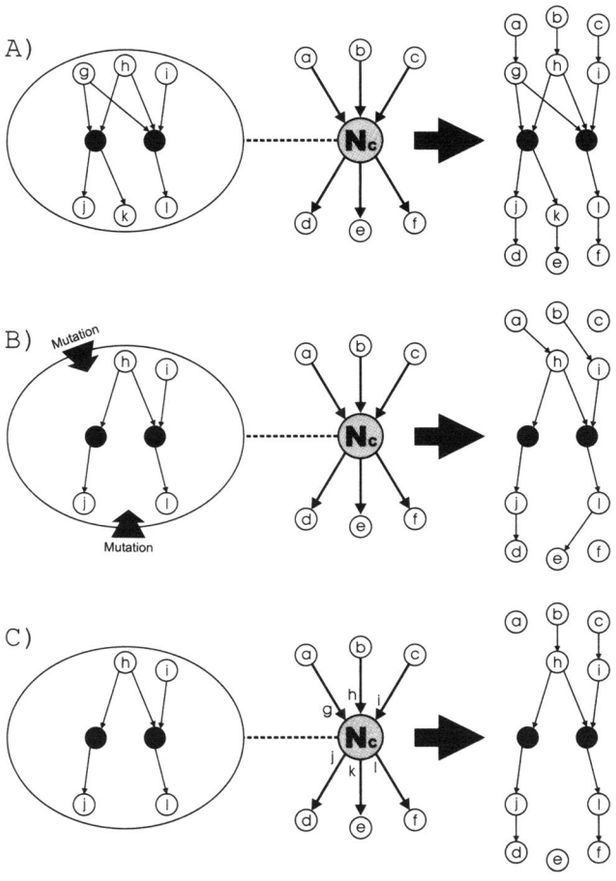

Figure 1: A) Hyperedge N_C is replaced by the graph on the left, producing the graph shown on the right if the hyperedge tentacles are connected in a fixed order. B) The graph associated with N_C is mutated by deleting a begin and end node, but this also changes the connectivity of the remaining nodes. C) By uniquely labelling tentacles, these mutation side-effects are avoided.

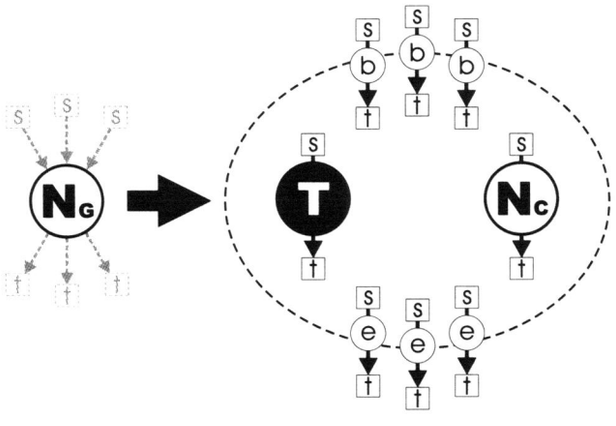

Figure 2: Diagrammatic representation of a cellular production. Nonterminal N_G on the left is replaced by a simplified hypergraph, where T is a terminal, N_C is a nonterminal, b and e are begin and end nodes, and s and t are source labels and target labels of each node. Source labels for terminals and source and target labels for nonterminals are optional – see section 5.1.

Luerssen (2005) describes a means for simplifying the *RHS* by replacing s and t with a mapping limited to the begin and end nodes of the hypergraph (see Figure 2). The *RHS* of the production is consequently not a hypergraph, but corresponds to a row in a labelled adjacency list. Each such production is referred to as a *cellular production*, which independently defines its connectivity to the graph context. By applying a set of cellular productions, the connectivity of the original hypergraph *RHS* can be established. Although more cellular productions than hypergraph productions are ultimately required to describe the same graph, each cellular production is a much more compact data structure, which scales linearly with the number of nodes specified.

4 Evolving Grammars

Generating a population of solutions from a grammar has been previously studied within the context of GP (Whigham 1995). A recent instance of this is Grammar Model-based Program Evolution (GMPE) (Shan et al. 2004), which applies a stochastic hill-climbing search to learn a stochastic context-free grammar from the best solutions in the existing population. Grammatical Evolution (GE), a GA evolving a genotype that indexes productions from a predefined grammar (Ryan et al. 1998), is also widely applied today and is based on earlier work by Paterson and Lively (1997).

In contrast, the grammar evolution system first proposed by Luerssen and Powers (2003) is specifically targeted at graph grammars and evolves a fully deterministic graph grammar directly. Each nonterminal of the grammar is unique, a constraint that allows for only a fixed number of derivations exactly matching the intended population of graphs. Starting productions are specially tagged productions whose expression leads to a previously evaluated graph. There are no separate genomes for different graphs; only one instance of a production has to exist, even if it is involved in the derivation of different graphs. Productions are neither predefined nor learned from any existing population, but obtained through copying and mutation of existing productions. The mutation operators comprise the simple addition, deletion and replacement of all possible terminal types and labels, non-terminal types, and source and target labels of begin and end nodes of the production.

Evolution in this model is viewed as a repeated growing and pruning of the production set. For every graph 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 graphs may result in greater or lesser fitness depending on the graph, the mutations apply specifically to a single graph and nowhere else. After testing all the mutated graphs, the least fit solutions, both from the mutated set and the existing graph population, are eliminated, as are all productions not involved in any fitter solutions.

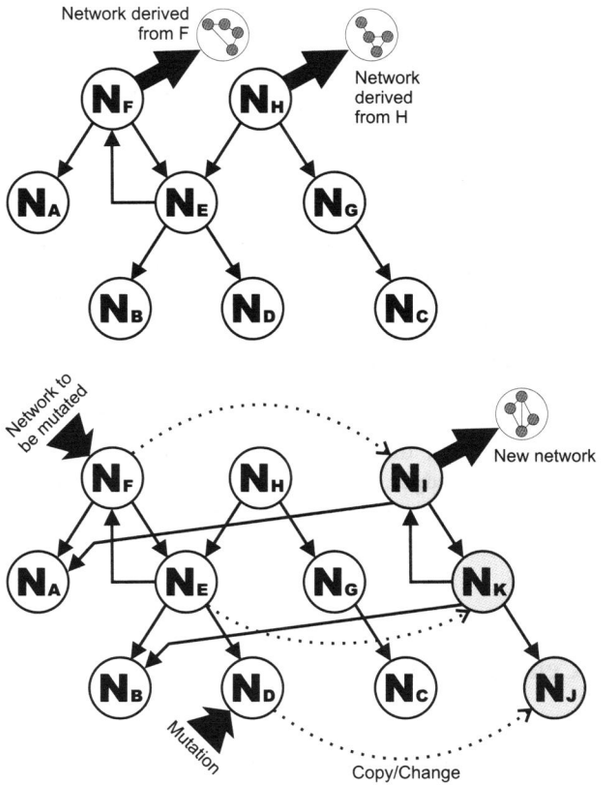

Figure 3: Example relationships between productions in the proposed system. Top: Starting productions with left-hand sides N_F and N_H call upon other productions to generate two different networks. Bottom: A new network is created by mutating N_D while deriving the N_F network. New productions N_J, N_K, and N_I are created to represent the mutation and new relationships.

Conversely, if a mutation survived, the grammar is modified so that the mutated graph becomes one of the graphs derivable from the grammar (see Figure 3). The mutated production is inserted into the grammar; then copies are made of all the graph's productions that need to refer to this mutated production and modified so as to refer to the mutated instance, not the original. This is repeated for all the productions referring to the now modified productions, including the starting production, from which the graph can now also be derived. To keep this process tractable only one production is mutated for each graph at a time.

5 Composing Graphs

5.1 Modular

All hyperedges in a hypergraph constitute nonterminals that must be replaced by hypergraphs according to the cellular production set. This replacement necessitates that hypergraphs connect to other hypergraphs, which can be accomplished through a process of label matching. Each begin and end node has a source and target label. A

connection is established if the source label of one node matches the target label of another. To reduce the search effort, this matching is limited to the immediate scope of the nonterminal (see Figure 4). Nodes in different scopes can only be connected via begin and end nodes that bridge these scopes. Unnecessary coupling between productions is thus discouraged, which facilitates a sense of modularity as defined by Simon (1996).

The label-matching model described in Luerssen (2005) employs a set of integer labels, and a match requires labels to be identical – this will be called *strict matching*. While a large set of distinct labels diminishes the likelihood of a match, a smaller set increases the occurrence of multiple nodes with identical labels within the same scope. This may be addressed by equally distributing connections among identically labelled nodes. Nodes are first sorted into a queue according to type and age of node. The first member of this queue becomes the selected node, which is then pushed to the back. The downside is that individual mutations can again have side effects on connectivity, as previously illustrated in Figure 2.

This paper proposes an alternative means of resolving connectivity: *soft matching*. The label set is maximally large, implemented here as a real number in the range [0,1). In place of matching only identical labels, the label with the smallest difference (also across the range boundaries) is selected. The diversity of possible labels reduces the likelihood of multiple identical labels, but these can still occur; for instance, from multiple identical nonterminals in a production, all of which will have the same begin and end node labels. The suggested fix is to add a source/target label pair to both the nonterminals and terminals. The source label adds to the source labels of all begin nodes of the associated hyperedge (or inputs of the terminal), and the target label adds to the target labels of all end nodes (or output of the terminal). This approach leads to a consistent labelling of all nodes (begin, end, terminal, nonterminal) with a source and a target, and allows for greater flexibility in connecting terminals. Determining connectivity also depends much less on the original node order, thus establishing a high degree of order independence in the representation.

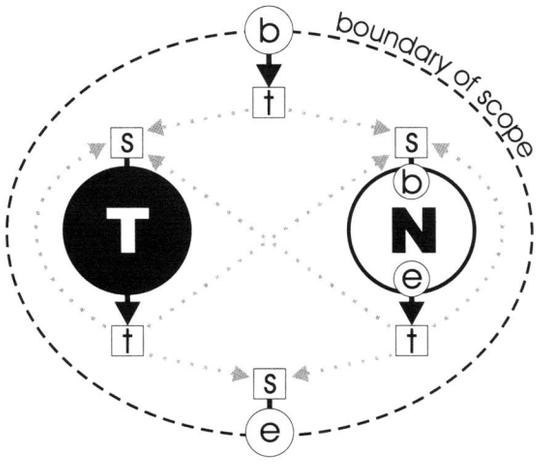

Figure 4: Source labels can only match target labels within the scope of the production; this also comprises the source labels of begin nodes and target labels of end nodes of included hypergraphs. No connection is permitted between target labels of begin nodes and source labels of end nodes, since this would allow graphs without terminals to occur.

In both label-matching models, terminals are implicitly wrapped into a production that connects to up to n nodes of any label, where n is the maximum number of inputs to the terminal type. This ensures correct connectivity even without the user explicitly wrapping terminals into cellular productions (although this is also allowed). The terminal output is connected to the source of an end node matching the terminal target label.

5.2 Non-Modular

Mutations are the only means of changing productions; no recombination (crossover) operator is modelled, since the mutation of nonterminals already results in a recombination of networks. When this occurs, however, each production and associated graph establishes its own scope, which is not always beneficial to graph composition. If, for instance, a node in a deeply embedded production is to be connected to an input of the network, then numerous begin nodes need to be defined among intermediate productions in order to bridge the scope boundaries. Allowing composition to occur without modularity constraints would provide additional flexibility in describing graphs. The proposal here is to assign a flag to productions that indicates whether modularity should apply. Thus, in practice, if a production N_A is referring to a non-modular production N_B, it is equivalent to the definitions of N_A and N_B being concatenated (see Figure 5 for an illustration of this). A mutation that turns this flag on or off during evolution is also added to the set of permitted mutations.

The following experiment evaluates this approach as well as the aforementioned different label matching models. The tested setups in particular are 1) strict matching without edge redistribution, 2) strict matching with edge redistribution, 3) soft matching without additional source/target labels, 4) soft matching with additional source/target labels, 5) soft matching with additional source/target labels and non-modular productions, and finally, for evaluating the impact of modularity, 6) soft matching with additional source/target labels and only non-modular productions.

6 Method

Graph optimisation is required by a diverse range of applications, but for this theoretical study tasks were selected for their transparency rather than utility. Symbolic regression concerns the inference of a functional mapping $y = f(x)$ between a set of independent variables x and a dependent variable y. Within the context of automata networks, this necessitates construction of a

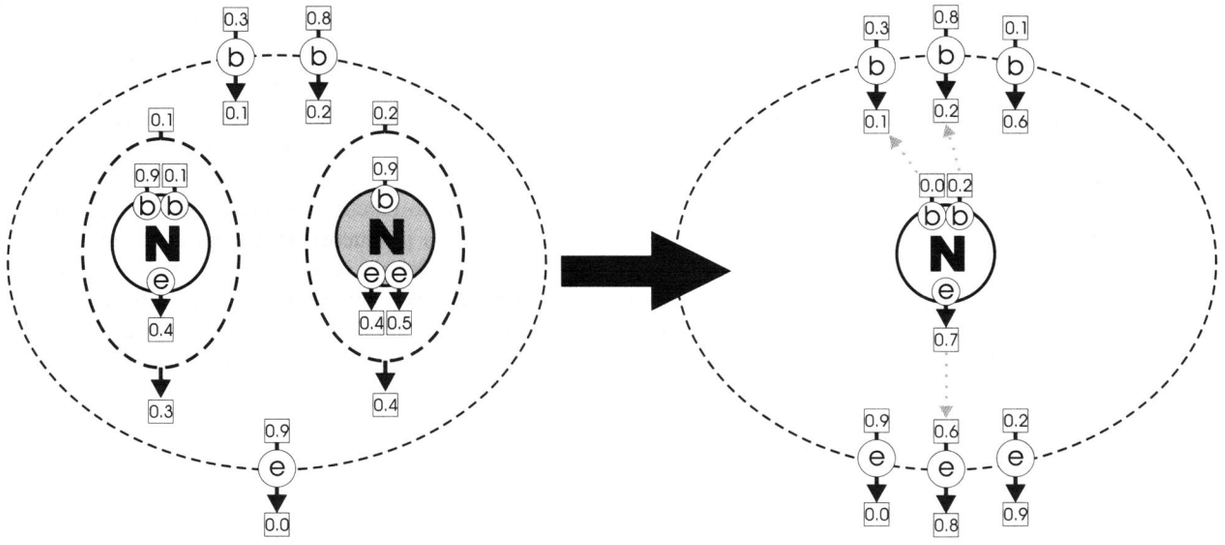

Figure 5: Nonterminal labels and nonmodular productions being applied. Nonterminal labels add to the labels of associated hypergraphs. The nonmodular production (grey *N* node) concatenates to the referring production, hence moving the nodes of the included hypergraph into the scope of the including. Connections are established subsequently. Please note that not all labels are shown on the left.

network of arithmetic function automata that best approximates the desired output when simulated. The regression targets for these experiments are the binomial-3 polynomial $f(t) = (t + 1)^3$ and the cycloid parametric equation $x = t - \sin(t)$, $y = 1 - \cos(t)$, for which x and y are separate output objectives. Fitness cases are 21 equidistant points generated by these functions over the interval of t = [-1,1]. Additionally, the time series $t(n) = \sin(t(n - 1)^2) + \cos(t(n - 2)^2)$, $t(0) = 0$, has to be approximated over 20 time steps. No performance comparison to other evolutionary techniques is provided here, but interested readers are referred to Luerssen (2005) for evaluations against GP.

Starting from an empty production the system evolves a population of 100 networks for each of 200 generations. A (μ + λ) evolution strategy is used, with all parents producing a single offspring each (μ = 100; λ = 100). For the binomial regression, each network is composed of automata that implement the binary functions {+, −, ×}; for the cycloid regression and the time series approximation the functions {sin, cos} are also included. To restrict execution time, a maximum of 100 productions and 100 terminals per production is permitted for each network. Terminal are prohibited from directly connecting to themselves. Strict matching selects randomly from the labels of the immediately visible scope, or at a 0.1 probability (or if there is no other label in the scope) selects randomly from a set of integer values in the interval [0, 10]. Soft matching always selects randomly a label from a uniform distribution between [0, 1), and is therefore intrinsically simpler.

For both regression tasks, networks are simulated for 10 time steps before an output from the network is sampled. Automata states are then reset, a new input is provided, and this is repeated until all inputs have been tested. For the time series, networks are sampled every 4 time steps for 20 × 4 time steps, for a total of 90 time steps including 10 time steps of initial margin. The system can assign line delays to the automata inputs to facilitate synchronization; longer line delays are applied randomly with a geometric probability of 0.5.

The second problem task (marked BP-NN) is to evolve the topology of a neural network that can classify the well-known Fisher Iris dataset (Fisher 1936). Finding a multilayer architecture is essential for good performance here. A population of only 25 networks is evolved for 50 generations on 75 patterns from the Iris set. Patterns are presented in random order to each network (neuron states are not reset), with each network being simulated for 10 cycles before an MSE is computed. Terminals are log-sigmoid neurons trained with standard backpropagation at a learning rate of 0.1. Weights are initialized randomly and uniformly within the range [-1,1]. No restrictions are made on cyclic connections, thus the classical method of instantaneously evaluating the feedforward and backward passes becomes intractable; instead, the network is relaxed over the existing 10 cycles.

On all tasks, the likelihood of a production being mutated is inversely proportional to how deeply it is embedded in the derivation tree of the network. This reduces the copying effort based on the assumption that deeply embedded productions constitute elementary building blocks that have shown their usefulness in composing larger networks and are thus less likely to require change. The geometric probability of selecting a deeper mutation is set at 0.25. Mutation operators are applied at equal probabilities with the exception of the addition operators, which have double probability. A single mutation is applied at a time, with a geometric probability of 0.5 that further mutations are applied.

Fitness is based on a multi-objective criterion of pareto-dominance (Deb, 2001) along the following dimensions: the objective function error, which is the mean squared error (MSE) over all samples; the size of the network, which is a simple count of the expressed begin, end, terminal, and nonterminal nodes; and the age of the network in terms of generations, used as a simple measure of novelty. The most dominated members are eliminated if the base population size (100) is exceeded. In the case of equally dominated members, each member is ranked on each performance objective, and the least diverse members (with the lowest mean rank distance) are eliminated.

7 Results & Discussion

Results from the above experiments are presented in Figure 6 and Table 1. A substantial amount of variability is present between evolutionary runs, but a few general trends can still be observed. Strict matching without any edge distribution is clearly the least successful connection strategy on the MSE objective of all problem tasks. Applying the queue-based edge redistribution greatly improves performance on the binomial-3 and BP-NN tasks, where a lack of unique labels seems to be an impediment to finding the best solution. However, this strategy fails to have any notable impact on the other problems.

Soft matching without additional labels is universally superior to simple strict matching and also performs more consistently than strict matching with edge redistribution. When allowing for additional terminal and non-terminal labels, further performance improvements with soft matching are observed on the majority of problems. The compositional freedom added by these labels evidently outweighs any complexity issues thus caused.

If the production modularity is disabled, the system relies solely on the source and target labels of terminals to establish the topology of the network. This has presented itself as remarkably effective on most of the evaluated problem tasks. On the binomial-3 regression it consistently stalls, however, so the expressive power of this approach is limited – modular productions are clearly

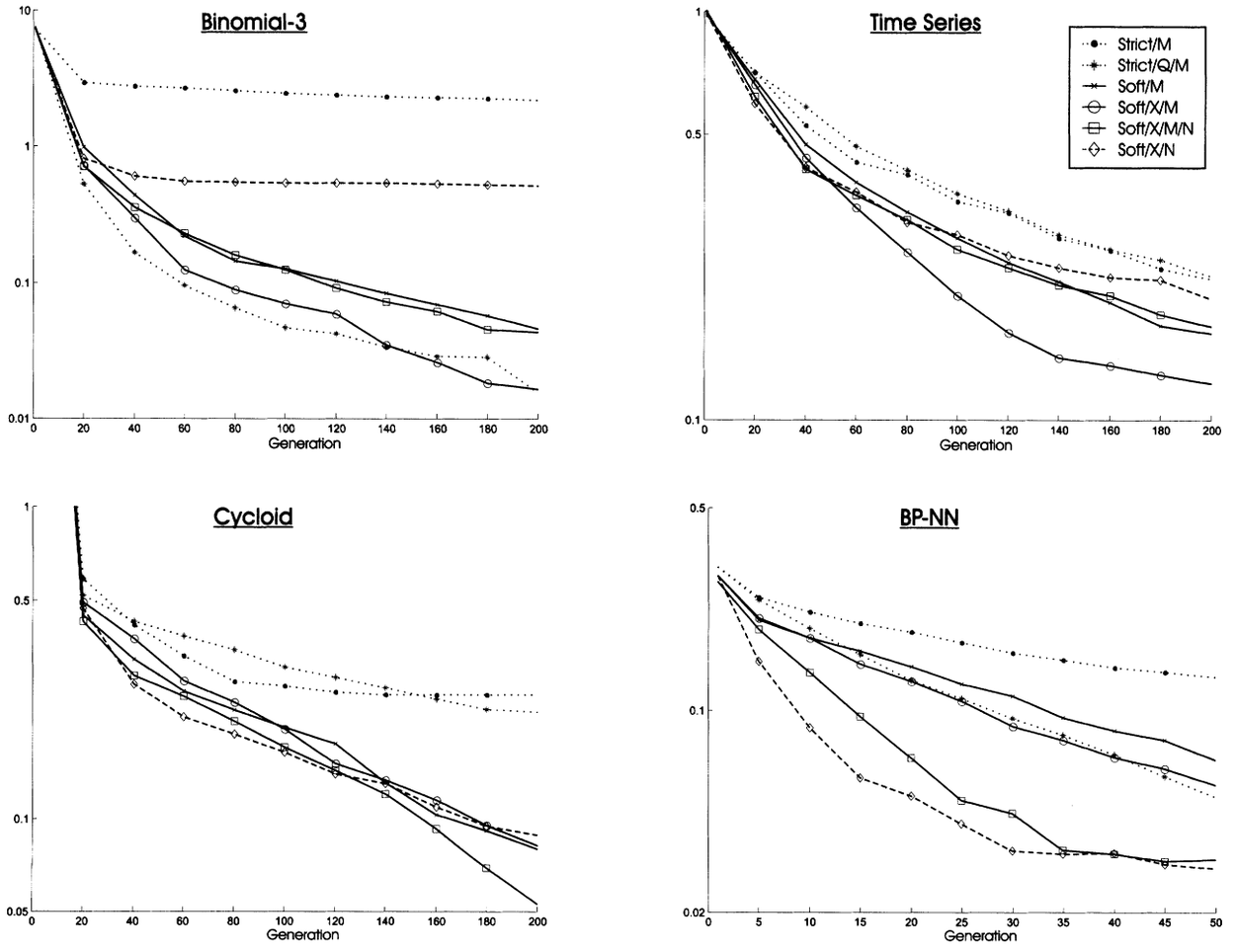

Figure 6: Mean error of the lowest error solution at each generation for all the different experimental parameter sets. Strict and Soft denote the labelling approach; Q indicates edge distribution; X indicates additional source/target labels; M and N show whether modular and/or nonmodular productions, respectively, are applied.

required for some solutions. On the other hand, modularity is not a free lunch either, as modular composition performs much worse on the BP-NN task. This presumably occurs because high connectivity is preferable here, since the learning process can reduce the effect of undesirable links but cannot create desirable links where there are none. Allowing for both modular and non-modular composition constitutes a reliable compromise between the modular and non-modular approaches, although it is worth noting that the addition of non-modularity does not appear to produce consistent benefits when modular approaches already perform well.

8 Conclusions

This paper presents a novel system for encoding and evolving automata networks using a mutable hypergraph grammar. The connectivity between a graph and its replaced hyperedges is a central issue in deriving these networks. An argument is made in favour of a simplified hyperedge replacement model, and, in extension to the work done by Luerssen (2005), several different methods for labelling and matching nodes are described and evaluated on a set of small but distinct problem tasks. The most consistent, superior performance is exhibited by the proposed soft matching approach. Extending this to multi-dimensional labels, although not attempted here, would result in a simulation of growth in a physically realistic space and may be worth pursuing in future. Thus, despite no biological plausibility being intended for this system, analogies to problem solving in nature appear unavoidable.

Modularity has also revealed itself as a potential drawback if high connectivity is desirable. A hybrid model of modular and non-modular composition is presented here as well, which provides for superior performance in this circumstance. Nevertheless, it is likely that larger networks are needed to observe the principal advantages of this approach and the system in general. Since any graph that can be assigned a fitness value can be optimized by the proposed system, numerous applications, from circuit design to programming, are expected to benefit from the expressive power of graph grammars.

	Binomial-3					
Parameters	E_{min}	E_{mean}	E_{std}	S_{min}	S_{mean}	S_{std}
Strict / M	0.20	2.18	0.88	5	10.54	6.83
Strict / Q / M	0 (78%)	0.02	0.03	11	23.58	13.56
Soft / M	0 (38%)	0.05	0.11	7	20.76	9.44
Soft / X / M	0 (72%)	0.02	0.07	10	19.10	9.56
Soft / X / M / N	0 (70%)	0.04	0.13	7	22.06	13.08
Soft / X / N	0.31	0.52	0.07	7	18.86	14.65

	Cycloid					
Parameters	E_{min}	E_{mean}	E_{std}	S_{min}	S_{mean}	S_{std}
Strict / M	0.21	0.25	0.01	11	18.32	10.21
Strict / Q / M	0.03	0.22	0.09	11	36.52	19.87
Soft / M	0 (20%)	0.08	0.08	9	28.54	21.77
Soft / X / M	0 (26%)	0.08	0.09	10	30.12	21.45
Soft / X / M / N	0 (32%)	0.05	0.07	13	35.30	23.69
Soft / X / N	0 (14%)	0.09	0.08	9	35.44	28.81

	Time Series					
Parameters	E_{min}	E_{mean}	E_{std}	S_{min}	S_{mean}	S_{std}
Strict / M	0.05	0.22	0.20	5	15.58	17.97
Strict / Q / M	0.08	0.22	0.23	5	13.48	8.35
Soft / M	0.06	0.16	0.10	5	16.74	16.18
Soft / X / M	0.07	0.12	0.06	5	21.70	21.85
Soft / X / M / N	0.06	0.17	0.11	5	18.10	11.08
Soft / X / N	0.05	0.20	0.11	5	11.14	5.79

	BP-NN					
Parameters	E_{min}	E_{mean}	E_{std}	S_{min}	S_{mean}	S_{std}
Strict / M	0.02	0.13	0.06	6	30.32	30.41
Strict / Q / M	0.02	0.05	0.02	8	29.12	22.18
Soft / M	0.02	0.07	0.05	3	18.40	14.45
Soft / X / M	0.02	0.06	0.04	3	16.08	12.13
Soft / X / M / N	0.02	0.03	0.01	8	18.78	10.24
Soft / X / N	0.02	0.03	0.01	9	19.98	10.60

Table 1: Final generation statistics for all experiments. E is error, S is size of lowest-error network. Fraction in brackets for E_{min} shows percentage of runs that reached zero error. See the Figure 6 caption for a parameter legend.

Bibliography

Astor, J. C. and Adami, C. (2000). A developmental model for the evolution of artificial neural networks. *Artificial Life*, 6:189–218.

Bäck, T. (1996). *Evolutionary algorithms in theory and practice*. Oxford University Press, New York, USA.

Boers, E. J. W. and Kuiper, H (1992). *Biological metaphors and the design of modular artificial neural networks*. Master's Thesis. Leiden University, The Netherlands.

Cangelosi, A., Parisi, D., and Nolfi, S. (1994). Cell division and migration in a 'genotype' for neural networks. *Network*, 5:497–515.

Dawkins, R. (1989). The evolution of evolvability. In C. G. Langton, editor, *Artificial Life: The Proceedings of an Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems*, Los Alamos, USA, pages 201–220. Addison Wesley, Santa Fe, USA.

Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms*. Wiley, Chichester, UK.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annual Eugenics*, 7: 179–188.

Futuyma, D. J. (1998). *Evolutionary biology*. Sinauer Associates, Inc., Sunderland, USA.

Goles, E. and Martinez, S. (1990). *Neural and automata networks: dynamical behavior and applications*. Kluwer Academic Publishers, Dordrecht, Germany.

Gruau, F. (1994). *Neural network synthesis using cellular encoding and the genetic algorithm*. Ph.D. Thesis. l'Ecole Normale Supérieure de Lyon, France.

Habel, A. (1992). *Hyperedge replacement: grammars and languages*. Springer-Verlag, Berlin, Germany.

Haddow, P. C., Tufte, G., and Van Remortel, P. (2001). Shrinking the genotype: L-systems for evolvable hardware. In Y. Liu et al., editors, *Proceedings of the 4th International Conference on Evolvable Systems: From Biology to Hardware*, Tokyo, Japan. Lecture Notes in Computer Science, volume 2210, pages 128–139. Springer-Verlag, Berlin, Germany.

Holland, J. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT Press, Cambridge, USA.

Hornby, G. S. (2003). *Generative representations for evolutionary design automation*. Ph.D. Thesis. Brandeis University, USA.

Kargupta, H. and Bandyopadhyay, S. (2000). A perspective on the foundation and evolution of the linkage learning genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186, 269–294.

Kitano, H. (1990). Designing neural networks using genetic algorithms with graph generation systems. *Complex Systems*, 4(4):461–476.

Koza, J. R. (1992). *Genetic programming: on the programming of computers by means of natural selection*. MIT Press, Cambridge, USA.

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

Lindenmayer, A. (1968). Mathematical models for cellular interaction in development, parts I and II. *Journal of Theoretical Biology*, 18(3):280–315.

Luerssen, M. H. and Powers, D. M. W. (2003). On the artificial evolution of neural graph grammars. In P. Slezak, editor, *Proceedings of the 4th International Conference on Cognitive Science (ICCS/ASCS-2003)*, Sydney, Australia, pages 369–377. University of New South Wales, Sydney, Australia.

Luerssen, M. H. (2005). Graph grammar encoding and evolution of automata networks. In V. Estivill-Castro, editor, *Proceedings of the 28th Australasian Computer Science Conference*, Newcastle, Australia, pages 229–238. Australian Computer Society, Inc., Sydney, Australia.

Luke, S. and Spector, L. (1996). Evolving graphs and networks with edge encoding: preliminary report. In J. R. Koza, editor, *Late Breaking Papers at the Genetic Programming 1996 Conference*, Stanford University, USA, pages 117–124. Stanford Bookstore, Stanford, USA.

Miller, G. F., Todd, P. M., and Hegde, S. U. (1989). Designing neural networks using genetic algorithms. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms and Their Applications*. pages 379–384. Morgan Kaufmann, San Mateo, USA.

Miller, J. F. and Thomson, P. (2000). Cartesian genetic programming. In R. Poli et al., editors, *Proceedings of the Third European Conference on Genetic Programming (EuroGP2000)*, Edinburgh, UK. Lecture Notes in Computer Science, volume 1802, pages 121–132. Springer-Verlag, Berlin, Germany.

Paterson, N. and Livesey, M. (1997): Evolving caching algorithms in C by genetic programming. In J. R. Koza et al., editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, Stanford University, USA, pages 262–267. Morgan Kaufmann, San Mateo, USA.

Ryan, C., Collins, J. J., and O'Neill, M. (1998): Grammatical evolution: evolving programs for an arbitrary language. In W. Banzhaf et al., editors, *Proceedings of the First European Workshop on Genetic Programming*. Lecture Notes in Computer Science, volume 1391, pages 83–95. Springer-Verlag, Berlin, Germany.

Shan, Y., McKay, R. I., Baxter, R., Abbass, H. A., Essam, D. L., and Nguyen, H. X. (2004). Grammar model-based program evolution. In G. Greenwood et al., editors, *Proceedings of the 2004 IEEE Congress on Evolutionary Computation (CEC 2004)*, Portland, USA, volume 1, pages 478–485. IEEE Press, Piscataway, USA.

Simon, H.A. (1996). *The sciences of the artificial*. MIT Press, Cambridge, USA.

Toussaint, M. (2003): *The evolution of genetic representations and modular neural adaptation*. Ph.D Thesis. Institut für Neuroinformatik, Ruhr-Universität Bochum, Germany.

Whigham, P. A. (1995). Grammatically-based genetic programming. In J. P. Rosca, editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, Tahoe City, USA, pages 33–41. Morgan Kaufmann, San Francisco, USA.