Experimental Investigations into Graph Grammar Evolution

by

Martin Holger Luerssen, *B.Sc. (Hons)*
School of Informatics and Engineering,
Faculty of Science and Engineering

May 29, 2006

A thesis presented to the
Flinders University of South Australia
in fulfillment of the requirements for the degree of
Doctor of Philosophy

Adelaide, South Australia, 2006

Contents

List of Figures

List of Tables

Abstract

Artificial and natural instances of networks are ubiquitous, and many problems of practical interest may be formulated as questions about networks. Determining the optimal topology of a network is pertinent to many domains. Evolutionary algorithms constitute a well-established optimisation method, but they scale poorly if applied to the combinatorial explosion of possible network topologies. Generative representation schemes aim to overcome this by facilitating the discovery and reuse of design dependencies and allowing for adaptable exploration strategies. Biological embryogenesis is a strong inspiration for many such schemes, but the associated complexities of modelling lead to impractical simulation times and poor conceptual understanding. Existing research also predominantly focuses on specific design domains such as neural networks.

This thesis seeks to define a simple yet universally applicable and scalable method for evolving graphs and networks. A number of contributions are made in this regard. We establish the notion of directly evolving a graph grammar from which a population of networks can be derived. Compact cellular productions that form a hypergraph grammar are optimised by a novel multi-objective evolutionary design system called G/GRADE. A series of empirical investigations are then carried out to gain a better understanding of graph grammar evolution. G/GRADE is applied to four domains: symbolic regression, circuit design, neural networks, and telecommunications. We compare different strategies for composing graphs from randomly mutated productions and examine the relationship between graph grammar diversity and fitness, presenting both the use of phenotypic diversity objectives and an island model to improve this. Additionally, we address the issue of bloat and demonstrate how concepts from swarm intelligence can be applied to production selection and mutation to improve grammatical convergence. The results of this thesis are relevant to evolutionary research into networks and grammars, and the wide applicability and potential of graph grammar evolution is expected to inspire further study.

Certification

I certify that this thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person except where due reference is made in the text.

As requested under Clause 14 of Appendix D of the *Flinders University Research Higher Degree Student Information Manual* I hereby agree to waive the conditions referred to in Clause 13(b) and (c), and thus

- Flinders University may lend this thesis to other institutions or individuals for the purpose of scholarly research;
- Flinders University may reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Signed Dated

Martin Holger Luerssen

Acknowledgements

Although this thesis has only a single name on its cover, it would not have been accomplished without the support, academic and otherwise, of other people. I therefore wish to extend my sincerest thanks to the following individuals who have facilitated this work:

- My supervisor, Dr. David M. W. Powers, for his many creative, inspiring, sometimes impossible ideas, and who still let me have my own (stubborn) way
- My office roommate, Dr. Trent Lewis, for putting up with my many eccentric ideas and not cleaning them off the whiteboard
- My fellow postgraduates in the AI group and beyond, for being good friends as well as good intellectual sounding boards: Aaron Ceglar, Denise de Vries, Kirsty Kitto, Richard Leibbrandt, Takeshi Matsumoto, Darius Pfitzner, and Dongqiang Yang
- My anonymous paper reviewers, whoever they are, and innumerable fellow academics I've bumped into at conferences, for all their constructive feedback and interesting conversations
- My parents, Dr. Holger Luerssen and Maike-Vogt Luerssen, for the support and happiness they provided to me, unconditionally, over these tough years

Chapter 1

Introduction

This dissertation addresses the general issue of designing solutions for any domain that allows solutions to be described as networks. Towards this goal, we establish a framework for network optimisation that is based on the artificial evolution of a (hyper-)graph grammar. We provide a detailed understanding of graph grammar evolution by describing a series of precisely designed experiments that investigate multiple strategies for construction, modularity, diversity, and convergence of the networks and the associated grammar. This brief introductory chapter is organised into multiple sections as follows. First, an essential synopsis of the conceptual background for this research is presented in section 1.1, succeeded in section 1.2 by a description of the research question and our approach to answering it. For the impatient reader, section 1.3 summarises the contributions of this thesis, and the chapter ends with 1.4, an outline of the upcoming chapters.

1.1 Conceptual Background

Networks permeate our lives. They describe a multitude of systems, natural and artificial, ranging from the cell, a network of chemicals linked by reactions, to the internet, a network of computers linked by cables or radio waves. Many of these networks are defined by self-organisation, a topic of intense study in recent years. Yet not every network is an emergent product of distributed agents. Sometimes a network needs to be designed. The circuit of a microprocessor is an obvious example, as is the architecture of a building or the mechanics of a jet engine, each a network of carefully placed components.

Good designers for these networks are rare, however, and as technology becomes ever more complex, they will only become rarer. It is thus a valuable skill to be able to optimise the configuration of a network so that it solves a particular problem to a satisfactory degree. A computer can become an effective designer of a network if it is given a proper definition of the problem. For instance, an exhaustive search of all possible configurations is assured to find us the optimal solution, although this is computationally infeasible for many larger problems. A heuristic search is more likely to determine a satisfactory solution in a reasonable amount of time, assuming that we have the right heuristic.

Evolutionary algorithms constitute a class of heuristic optimisation algorithms that have been applied to a great many problems, including design. Evolutionary design of networks is fraught with a number of challenges, however. Artificial evolution is inspired by natural evolution, yet lacks the sheer scale and sophistication of the latter. Since the computational cost of evolution tends to scale exponentially with the number of features needing evolution, the magnificent designs of nature are out of practical reach of any evolutionary algorithm. Just how complex a design can be achieved by artificial evolution?

The number of variables defining a simple graph scales at least quadratically with the number of graph nodes, so the answer within our intended domain appears to be: not very complex at all. A large network is not necessarily complex, however, and this is where self-organisation can benefit even the designer. A few simple rules can describe a huge network if it exhibits some form of regularity. Thus, the assumption that a network arises from a set of rules may bring about the most compact description of the network, but to benefit from this we require a system in which not the network but the rules that define the network are optimised.

A precedent for this is given by the growth and development of biological life. Genes (the rules) are expressed into a fully functional organism (the network) that has many orders of magnitude more features than there are base pairs in the chromosome. Proposing and confirming models of biological development is a popular research activity in the life sciences, but for this to be of use in artificial evolution the computationally expensive details need to be abstracted away. *Grammars* represent elementary but very applicable models of development. For our purposes the sentences (solutions) that must be derived from a grammar are networks, which implies that a graph grammar is needed. The evolution of a graph grammar can capture patterns in the design of a network and facilitate their reuse in new solutions, consequently allowing network designs to be represented and searched more efficiently.

1.2 Research Strategy

The hypothesis that searching a space of grammatical rules is more effective than directly searching the space of networks forms the basis of the following research. However, this dissertation is not about directly proving this hypothesis; as will be shown over the next two chapters, existing research already offers extensive support for it. The novelty here lies in the utilisation of a graph grammar, a formal model that is quite distinct from the powerful but convoluted biology-inspired development models or the less expressive direct coding strategies. Moreover, unlike most research into evolutionary algorithms, the strategy is not to evolve string or tree representations using existing techniques and then translate them into networks, but to evolve the graph grammar directly.

The question that this study responds to concerns the nature of an effective framework for graph grammar evolution, which incorporates the novel challenges of directly evolving a grammar and composing a graph from disparate production rules. Naturally, an answer would not be meaningful unless a benchmark is provided. The proposed framework will therefore be developed and tested against a number of practical problems: symbolic regression, circuit design, neural networks, and communication networks. What works and what does not will hence be determined in an empirically sound manner.

1.3 Contributions

Through exploring the idea of graph grammar evolution this work renders several contributions towards an understanding of network design by evolution. The following important inputs to scientific knowledge are made:

1. A survey and analysis of how networks can be optimised demonstrates the merit of developmental models in an evolutionary search context.

2. A novel method of graph grammar evolution is presented, which transforms the formal model of a hypergraph grammar towards being directly evolvable by a multi-objective evolutionary algorithm.

3. The nature of the grammars and graphs obtained from graph grammar evolution is investigated, and a comparison between alternative heuristics is given.

4. The relevance of diversity in graph grammars is recognised. Phenotypic diversity objectives and island models are introduced to improve diversity.
5. The structure of evolved grammars is analysed, and an adaptive search scheme is established for improving grammatical convergence.

Details for each of these contributions are provided within the chapters and summarised in section 8.1.

1.4 Overview

The thesis starts with a review of essential concepts and existing literature that underline this research, followed by a technical description of the intended graph grammar evolution framework. The second half of the thesis covers extensions to this framework and experimental evaluation of these, with an analysis of the outcomes and further improvements suggested. Specifically, the remaining chapters of the thesis are organised as below.

Chapter 2 introduces network theory, reviews networks as dynamic processing structures, and discusses how networks can change to solve a problem. Computational learning and evolution are appraised within this context. This constitutes a fundamental introduction to concepts applied in subsequent chapters.

Chapter 3 presents the benefits and drawbacks of generative representations, and reviews previous literature in developmental models of artificial evolution. Graph grammars are exposed as a formally elegant basis for network development; this is concluded by an introduction to hypergraph grammars.

Chapter 4 establishes a novel variant of hypergraph grammar suitable for artificial evolution. A complete system for evolving such a grammar and deriving networks from it is detailed.

Chapter 5 addresses the many different ways in which graph production rules may interact and connect to each other. This includes explicit or implicit representations, modularity, and differences between simple graphs and pseudographs; for empirical comparison, several experiments are performed on different problems.

Chapter 6 looks at the importance of diversity in evolving a graph grammar, introducing and experimenting with several different phenotypic diversity objectives and island models.

Chapter 7 analyses the evolutionary convergence of the productions of a graph grammar, testing the impact of the mutation rates, evaluation length, and adaptive search inspired by swarm intelligence. The development of bloat and Pareto efficiency is compared, and the network of references that forms between evolved productions is also studied.

Chapter 8 states the conclusions, lists the contributions and summarises the problem-specific results obtained throughout the previous chapters. It finishes with recommendations for future research.

Appendix A provides tables of the results from all experiments. No tables will be presented in the chapters, so please refer here for details on the performance and size of the evolved networks.

Chapter 2

Network Adaptation

If done by hand, designing a network can be a trivial task or an impossible task, depending on the number of components involved and the complexity of interactions between them. Why do we need to design a network? The human brain is certainly a spectacular design of a network, as revealed by its evident ability to read this very sentence. Yet nobody would claim to have a designed those circuits: a primary school teacher may have provided valuable feedback on the reading task, but nothing more. It would certainly be convenient if other networks, like a silicon circuit or power-line network, self-designed just by exposure to the problem. For this to be viable, a fundamental requirement is that the network can change its own state. This chapter begins with an introduction to and review of the processing models that represent this type of network: the *automata networks*. We extend from there into how networks self-organise and can be made to self-organise within the context of computational intelligence, leading to a brief survey of general learning, neural networks, and artificial evolution.

2.1 A Brief Note on Graphs & Networks

Before continuing with this chapter, however, we would like to clarify any potential confusion between the terms *network* and *graph*. In fact, these terms can be used interchangeably, although here a network will usually refer to an automata network (see below) that is assessable on a problem task, whereas the graph (or cellular space) is a formal description of the structure of the network. Specifically, we define a *directed graph* as a quadruple (V, E, s, t) where V is a finite set of vertices, E is a finite set of edges, and $s, t : E \rightarrow V$ assign a source $s(e)$ and a target $t(e)$ to each $e \in E$. In general usage, a *graph* is a simple graph without

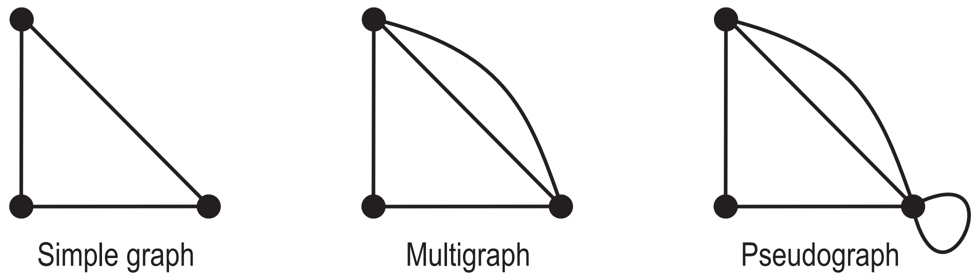


Figure 2.1: Graph concepts.

loops or multiple edges. A loop is an edge that joins a vertex to itself; multiple edges are two or more edges connecting the same two vertices. Graphs with loops and multiple edges are called *pseudographs* (see figure 2.1). Thus, every graph in the accepted sense is a pseudograph, but not every pseudograph is also a (simple) graph. We will, however, refer to pseudographs also as graphs and networks, unless the distinction is relevant (as in section 5.4).

2.2 Automata Networks

Automata networks are dynamical systems that are discrete in time and space (Goles and Martínez, 1990). Variants of these networks operating in continuous time also exist and play an important role in the modelling of physical systems, although we will not consider them here in detail. Formally, an automata network is a system (C, A, S_n, N, L) where C is a set of cells, A is an alphabet of states, $S_n : C \rightarrow A$ is the state at time n, where $n = 0, 1, 2, \ldots, M$. $\forall c \in C : N(c)$ is the neighbourhood of c, where $N : C \rightarrow P(C)$ is the neighbourhood system and $P(C)$ the power-set of C. The triplet $(C,\ S,\ N)$ is commonly also referred to as the *cellular space* of the automata network. S is updated according to $L : C \rightarrow D$ where D is a set of local dynamic rules $\delta_{N(c)} : S(c)_{n-1} \rightarrow S(c)_n$. The global dynamic rule is formed by $\Delta(S) = \cup c \in C(\delta_{N(c)}(c))$ and describes the behaviour of the system.

The defining aspects of an automata network are its cellular space and the local dynamic rules (or *transition functions*). The cellular space describes the presence of, and connectivity between, processors (or *cells*) of the network. It consists of a directed graph and a set of states associated with the vertices of the graph. Each processor occupies a vertex of the graph and operates on the associated state. The state is updated at successive time steps depending on the local transition function, which takes the current state and the states of adjacent vertices as its inputs. States can be updated all at once, i.e. synchronously, or in some prescribed (or random) order, i.e. sequentially. Input into the automata network is encoded as a pattern of states initially assigned to part or all of the machines, and the output is similarly retrieved after a certain number of cycles of the network. This is a closed, autonomous model that – unlike a systolic array – does not exchange information with its environment in the course of its computation. However, the transition between systolic arrays and automata networks is difficult to pinpoint, and not everyone takes this definition literally (e.g. Zambonelli and Roli, 2001).

Unless we place further constraints on what constitutes an automata network, the dynamical behaviour, i.e. the evolution of states, of the network has no (known) mathematical shortcut – the only way to describe it is to simulate it. Consequently, automata networks have been further constrained in order to make them amenable to mathematical analysis and prediction. Depending on the applied constraints, such networks are known as cellular automata, artificial neural networks, and by various other names. The following sections provide a review of these, as several associated concepts will be taken up in subsequent chapters.

2.2.1 Cellular Automata

Cellular automata were introduced by Von Neumann (1966) to establish a framework for studying the behaviour of complex, self-reproducing systems. They have since become a subject of interest in a variety of fields, particularly in modelling physical phenomena (Wolfram, 2002). Cellular automata are principally distinguished from other automata networks by the homogeneity of their cellular space (Garzon, 1995). The underlying digraph is not just any graph, but a Cayley graph – a regular graph with regular colouring. In practice, this implies that the interconnection pattern between vertices is spatially invariant, i.e. it appears the same at every vertex. A common instance of such a graph is a Euclidean lattice (see figure 2.2).

A neighbourhood on such a graph may consist of just the four cells in the principle directions (the von Neumann neighbourhood), or the corner cells as well (the Moore neighbourhood), or a block or diamond of even larger size; in fact any arbitrary shape. What is common, however, is a sense of locality, that nearby cells have greater influence upon each other than remote ones. Information must be

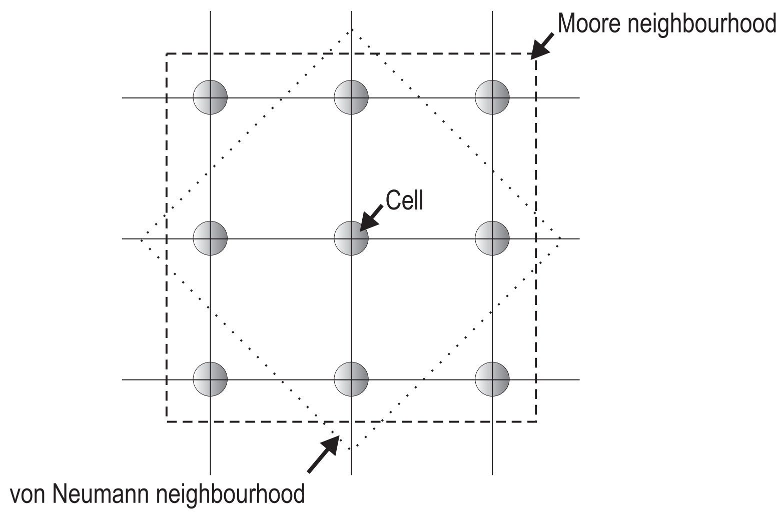


Figure 2.2: Two common definitions of neighbourhood as they apply to a Euclidean lattice, a typical cellular space used for cellular automata.

propagated from neighbour to neighbour in order to reach a remote cell, but this locality constraint also exists in nature, because of physical constraints on how quickly information can spread from one place to another. However, in cellular automata the local transition rules of each cell are also identical, i.e. spatially invariant, which is less naturally plausible, since, for example, the human body is constructed of a variety of cells that interact differently. A *non-uniform cellular automaton* has cells that can obey diverse rules. Sipper (1996) demonstrated the superior performance of non-uniform cellular automata over uniform cellular automata on the density classification task, but other research in this field remains sparse.

The most common way of expressing a local transition rule is a transition table analogous to a truth-table, with rows describing the state of the neighbourhood and a next-state column indicating the next-state of the cells. The simplest rules are those that obey the superposition principle: the next state of a cell depends linearly on the local distribution of its neighbours. Linear cellular automata exhibit only a limited repertoire of global behaviours, but are reasonably well understood. On the other hand, totalistic cellular automata, such as Conway's Game of Life, are significantly more complex. Totalistic rules update a cell's state dependent on the density of its neighbours. Even though this represents only the mildest generalisation of linear rules, Conway's Game of Life can simulate a Universal Turing Machine. The computational universality of such a cellular automaton can rarely be exploited, however. There are two fundamental problems in the study of a cellular automaton. Given a local transition rule, we would like to characterise the global effect of this rule – this is known as the *forward problem*. From a design point of view, however, it is the *inverse problem* that is of paramount importance: given a desired global effect, we would like to know the local rules that induce this effect.

2.2.2 Threshold Automata

A popular approach for addressing the inverse problem is inspired by the operation of biological neurons in the brain. Signal transmission between biological neurons is a complex chemical process in which specific transmitter substances raise or lower the electrical potential across the membrane of the receiving neuron (Bear et al., 2006). Once this potential has reached a certain threshold, a pulse (or *action potential*) is emitted that induces further transmitter release to other cells. McCulloch and Pitts (1943) proposed a simple computational model of the neuron as a binary threshold unit. Specifically, the model neuron computes the weighted sum of its inputs from other neurons, and outputs a one or zero according to whether this sum is above or below a certain threshold value. We can replicate this model on a totalistic cellular automaton by allowing the transition function of each cell to be a compound rule formed of a sequence of two steps. In the first step, the states of the neighbouring cells are multiplied by real numbers acting as weights, and the products are added to obtain a weighted sum. In the second step, the threshold operation – actually a special type of totalistic rule – is applied to this sum, and the cell is assigned a state accordingly.

Cellular automata that employ this rule are referred to as linear-threshold automata, a widely applied variant of which are the Cellular Neural Networks (CNNs) introduced by (Chua and Yang, 1988a,b). Each cell of the CNN has a continuous-valued internal state and an output. The internal state is computed as a sum of a local bias value and the inputs derived from two so-called templates: The feedback template, which constitutes the outputs of the neighbouring cells; and the control template, which constitutes the input into the CNN, e.g. an image to be processed. Both templates are associated with parameter arrays that typically define the inter-cell connection weights. Templates are usually considered to be spatially invariant, although non-uniform CNNs have also been investigated (Roska and Chua, 1992).

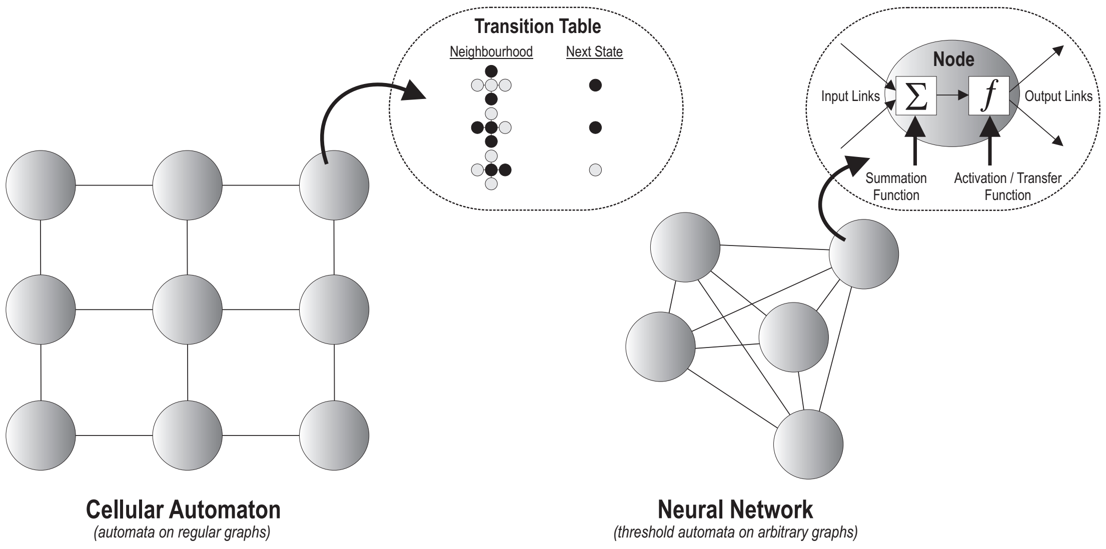


Figure 2.3: Automata networks are defined by their topology and local transition functions, illustrated here for two popular instances of automata networks.

2.2.3 Neural Networks

Research into artificial neural networks (ANNs) was originally aimed towards modelling networks of real neurons in the brain, but has gradually shifted towards the more practical aspects of employing such networks for signal processing and other tasks. The previous section introduced an extremely simplified model of neural communication, where a crucial feature was omitted, however. Biological neurons conduct their signals along long fibres (called *axons*) that can project to very remote regions of the human body. To some extent this circumvents the constraints of locality by providing remote communication without having to pass through other cells. Neural networks are generalised threshold automata where no locality constraints are placed on the digraphs of their cellular space. Furthermore, the local transition function is not homogeneous – it varies from site to site, depending on the weight of its connections.

Neural networks have been intensively researched since the days of McCulloch and Pitts' (1943) seminal work on neural modelling. An early wave of activity in the 1960s was centred on networks called *perceptrons* (Minsky and Papert, 1969), which have their neurons organised into layers with feedforward connections between one layer and the next, that is, the digraph of the cellular space is a unidirectional bipartite graph. Given appropriate inter-cell connection weights, passing an input pattern into one layer is expected to produce a desirable output from another. Following a revival of interest in the 1980s, perceptrons, specifically those with multiple fully connected layers and sigmoid threshold functions, have become the most studied and utilised of networks. The main other focus in neural network theory is on associative content-addressable memories, such as the Hopfield network (Hopfield, 1982), which can store several binary patterns in such a way that the attractor dynamics of the system carries any state of the network to a state matching the nearest of the stored patterns. The majority of proposed networks differ with respect to the learning algorithms that are applied; the undoubtedly most popular one – backpropagation – will be covered in section 2.4.1.

2.2.4 Random Boolean Networks

Random Boolean networks (RBNs) are a generalisation of cellular automata, where, as with neural networks, each node can be affected by potentially any node in the network. RBNs were originally developed by Kauffman (1969, 1993) as a model of genetic regulatory networks, which describe the complex interactions between genes. In the classic RBN, nodes have binary states, but more recent work has also investigated RBNs where nodes can take multiple values (Ballesteros and Luque, 2005). Each node is connected to a fixed number of other (or the same) nodes, with the topology typically defined randomly. Local transitions rules are also chosen randomly for each node, as is the initial state.

Consequently, a vast number of different networks are conceivable, yet there are general properties that apply to all (Gershenson, 2004). The state space of a network is finite, so eventually a state will be repeated. Since the state update is assumed to be deterministic, the network dynamics have thereby reached an attractor, either a fixed-point attractor if it is a fixed state, or a periodic attractor, if the state moves back and forth between two or more configurations. A categorisation of networks is possible in terms of these attractors and the phase transitions (changes in behaviour) based on different initial conditions or online perturbations of the network. These characteristics are highly sensitive to changes in network topology, as demonstrated in a study by Oosawa and Savageau (2002). Networks with uniform degree distributions for their nodes have more and longer attractors but less correlation in their dynamics, whereas skewed topologies show the opposite. A scale-free topology, a feature typical of self-organised complex networks (see section 2.3 below), balances these parameters much more favourably than the extreme topologies.

2.3 Complex Networks

Networks that are composed of parts interacting in non-trivial ways are referred to as complex networks and are found to characterise a wide variety of biological, social, and technological systems. Recent empirical studies have focused on the role played by topological structures in the dynamics of these systems. Examples include the Internet and the World Wide Web, cellular networks, citation networks, ecological networks, neural networks, power networks, and others; an extensive review of these results is provided by Albert and Barabási (2002). Despite exhibiting otherwise very distinct interaction dynamics, all of these systems share common principles of structural self-organisation – and they are all remarkably effective at what they do. It may be assumed that these principles are valuable for the design of networks in general.

Traditionally the study of complex network has been the domain of graph theory, which proposes random graphs, i.e. graphs with no apparent design principle, as the simplest and most straightforward realisation of a complex network. It is intuitively clear, however, that complex systems such as the Internet exhibit some form of organisation encoded into their topology. The desire to understand such interwoven systems has brought along significant challenges. Statistical mechanics has given researchers an arsenal of successful tools to predict the behaviour of a system as a whole from the properties of its constituents; for example, it forms our basis of understanding how Hopfield networks operate as associative memories (cf. section 2.2.3). Frequently, the capacity to predict depends on the simplicity of interactions between elements of the networks, yet there are many systems where these interactions are not simple. Motivated by these circumstances, three concepts have come to occupy a prominent place in contemporary thinking about complex networks: clusters, small worlds, and scale-free degree distributions.

The inherent tendency to cluster is a common property of complex networks and is quantified by the clustering coefficient. In a random network, the probability that two neighbours of any node of the network are themselves neighbours of each other (i.e. the clustering coefficient) is equal to the probability that two randomly selected nodes are neighbours. Watts and Strogatz (1998) determined that complex networks have much larger clustering coefficients than random networks and proposed a new class of network models, collectively called *small-world* models. These models interpolate between highly clustered regular lattices and random networks, but retain a very short characteristic path length between any two nodes of the network, around $\ln(N)$ for a network with N nodes.

A famous manifestation of the small-world model are the "six degrees of separation" uncovered by Milgram (1967), who concluded that any pair of people in the United States are connected by a path of six acquaintances. The cause of short path lengths in small-world networks is the existence of shortcuts between nodes (Watts, 1999). Every shortcut is likely to connect widely separated parts of the network and thus has a significant impact on the characteristic path length of the entire network. Watts and Strogatz (1998) proposed an algorithm to construct small-world networks. Starting from a regular ring lattice where each node is connected to its nearest neighbours on either side, a small fraction of connections is then re-wired to other randomly selected nodes. Barabási and Albert (1999) realised that this didn't reflect any natural process. Most physical networks represent open systems which grow by the continuous addition of new nodes. Furthermore, new nodes exhibit preferential attachment, that is, the likelihood of connecting to an existing node depends on that node's degree. As an analogy: having many friends makes it easy to have even more.

The resulting number of edges each node has, the node degree, is characterised by a distribution $P(k)$, which described the probability that a randomly selected node has exactly k edges. In a random network, the majority of nodes have approximately the same degree, close to the average degree k. In a complex network defined by growth and preferential attachment, the degree distribution often has a power-law tail, that is, $P(k) \sim k^{-\lambda}$. Such networks are referred to as being *scale-free* (Barabási and Albert, 1999). Cohen and Havlin (2003) demonstrated that scale-free networks can also be, and are likely to be, small-worlds. In fact, scale-free networks with an exponents $2 < \lambda < 3$ have a characteristic path length of approximately $\ln(\ln(N))$, notably shorter that what is usually regarded as a small-world. Short path lengths are desirable because they maximise information flow across the network, which can have a significant effect on the dynamics of the network. For example, Bohland and Minai (2000) observed that the performance of an associative memory based on a small-world network can match that of a random network, whilst requiring only a fraction of the total connection length.

Assimilating concepts from complex network science into the intentional design of networks merits investigation, because the former reflects natural constraints of design. Conversely, the constraints assumed previously for cellular automata stem from the reductionist thinking prevalent in the physical sciences – the attempt to find a minimal set of laws that uniformly applies to all of space and time. At higher levels of abstraction, however, these constraints do not hold. Locality is undermined by the possibility of signals propagating at different speeds. Spatial invariance similarly breaks down if contrasted with the diversity of cells in the human body and brain. This is not to say that this particular set of constraints, or any other for that matter, is somehow wrong. Cellular automata, neural networks and automata networks are computationally equivalent (Garzon, 1995). Yet brain and computer are computationally equivalent, too, and both outperform each other on different tasks. Some approaches to solving a particular problem have simply proven themselves more effective than others. Constraints need to be chosen accordingly, based on the task at hand and the resources available.

2.4 Network Learning

The previous section hinted at the sheer diversity of connected machines, both natural and artificial, that are possible, each of which may exhibit its own unique dynamics. The classification of these dynamics has become a science by itself, as exemplified by Wolfram's well-cited observations about a very limited set of cellular automata (Wolfram, 1983). Wolfram attempted to categorise the behaviour of comparatively simple one-dimensional cellular automata into various dynamical classes, but whether this has been of much practical use to anyone is debatable. What this exercise did reveal is that even remarkably simple networks operating on simple rules can produce enough complexity to require years to reach human understanding.

Such dynamical richness, if mastered, also promises to be a useful tool in solving problems. To solve a specific problem it becomes necessary to find the automata network that exhibits the precise dynamics required for its solution. For a Von Neumann machine we can conveniently write a series of instructions that – akin to an assembly line – transform a problem into a solution. Conversely, for an automata network we must orchestrate all the processing units and their interactions from a problem configuration into a solution configuration. As the stock market demonstrates to us on a daily basis, the prediction or even control of massively parallel and potentially chaotic systems constitutes a serious challenge. Nonetheless, there are countless such systems to be found in nature, the human brain being amongst them, and they fulfil their purpose rather effectively. Nobody designed these systems, they designed themselves – they adapted. Adaptation is an automatic solution-finding process. Its replication on a computer comprises the very core of the field of artificial intelligence, and there are a variety of ways for accomplishing it.

2.4.1 Weight Learning

A term closely associated with adaptation is learning. A learning system may determine a solution to a problem by exposure to this problem. In the above context, the system would have to learn the nature of a particular automata network, i.e. its structure and any attributes associates with it. By far the most common attribute that is being learned in this context are the edge weights of perceptrons. The popularity of perceptrons is in no small part due to the effective learning algorithms that can be and have been devised for them, which are typically based on the idea of *gradient descent*.

Each configuration of weights produces a different error in a supervised learning environment. This grants us a function of error against weights and hence an error landscape, or error surface. By evaluating the derivative of this function, we can determine the weight change needed to reduce this error. Learning by gradient descent involves adjustment of the weights in small steps along the gradient given by the derivative, until a minimum of the function – i.e. a minimal error – is attained. A parallel implementation of this would call for the error information to be propagated from the output nodes back through the network – hence we also refer to this approach as *backpropagation* learning (Rumelhart et al., 1986).

Gradient descent has an inherent propensity to become easily trapped by local optima. Two kinds of local optima can be distinguished (Bianchini and Gori, 1996): those inherent to the structure of the chosen problem (*structural local optima*) and those resulting from mathematical idiosyncrasies of the chosen network (*spurious local optima*). In the presence of such optima, there is no assurance that the best possible weight configuration will be found. Furthermore, while backpropagation can be applied to recurrent networks, the smallest weight changes may lead to vastly different dynamics if networks are simulated over many iterations (Doya, 1992). The application of backpropagation loses its effectiveness in such cases, as the possible learning trajectories along the error surface become increasingly unstable. Recurrent neural networks can be trained by alternative means, such as the Hebb rule, which determines that connection weights between neurons should change relative to the correlation between their firing patterns (Hebb, 1949). However, success of this is only assured for very constrained network designs, such as symmetric designs, e.g. Hopfield networks.

2.4.2 Structure Learning

Gradient descent, as presented previously, is feasible if an appropriate derivative function with respect to the weights has been found. For weight learning in a perceptron with differentiable threshold functions this is a straightforward exercise. For the majority of other learning problems, however, it is not. Weights are meant to model the synapses between biological neurons. Weight learning hence reflects synaptic change, which is a major source of functional change in the brain – but it is not the only source of change. Adult neurogenesis, i.e. the perpetual incorporation of new neurons into neural structures, has been long conjectured and shown in models to play a crucial role in brain plasticity and learning (Cecchi et al., 2001). The classic example is the seasonal death and regrowth of neurons in canaries, which renew their yearly repertoire of songs in this manner (Alvarez-Buylla et al., 1992).

Weight learning is not the only possible – or even necessary – form of adaptation in an artificial neural network either. In fact, the success of weight learning is often dependent on the number and arrangement of neurons and edges within the network, which is not in any way adjusted by backpropagation. Too many neurons and links, and the network will fail to generalise; too few, and the network won't be able to learn the task at all. Very little theoretical knowledge exists about how to find a suitable network architecture for a given task. Beyond simple trial and error, a range of heuristic algorithms have been suggested to address this problem. Incremental algorithms initially assume a simple network and add neurons and links until the network can learn the task (e.g. Fahlman and Lebière, 1990; Frean, 1990), while pruning methods start with a large network and prune off superfluous components, usually to improve generalisation (e.g. Mozer and Smolensky, 1989; Cun et al., 1990). Both types of algorithm represent a form of structural gradient descent that can only explore a subset of topologies and is intrinsically limited by the lack of gradient information for the discrete topology space, leading to slow and unreliable convergence (Angeline et al., 1994). Exploring a wider range of topologies beyond small, fully connected feedforward designs is not directly feasible with any of these methods.

2.5 Evolutionary Computation

For the majority of problems we can define an error function that will tell us (numerically) how good a particular solution is at solving the problem. If we ran-

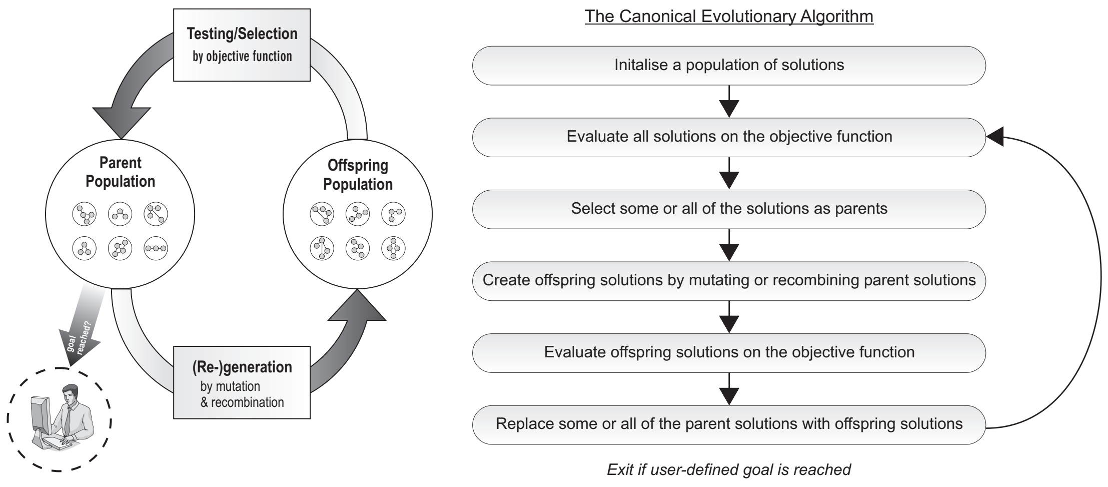

Testing/Selection
by objective function
Parent Population
Offspring Population
goal reached?
(Re-)generation
by mutation
& recombination
The Canonical Evolutionary Algorithm
Initalise a population of solutions
Evaluate all solutions on the objective function
Select some or all of the solutions as parents
Create offspring solutions by mutating or recombining parent solutions
Evaluate offspring solutions on the objective function
Replace some or all of the parent solutions with offspring solutions
Exit if user-defined goal is reached

Figure 2.4: Although there are many different instances of evolutionary algorithms, the iterative process of generation, selection, and regeneration is common among all.

domly generate more solutions, we are likely to find solutions that are better than others. By sampling the error function in this manner, we generate information about the associated error space that can help us deduce where superior solutions might be found. Unlike gradient descent, this information is obtained not from a single solution, but a population of solutions. Evolutionary optimisation is the umbrella term for optimisation techniques of this kind.

Evolutionary optimisation is inspired by the concept of biological evolution, which was first formulated by Darwin (1859) and is believed to be responsible for the complexity of life as we see it. The Darwinian theory of evolution explains the adaptation of species by the principle of natural selection, which favours those species that are fittest, i.e. most adapted to their environment, and consequently most successful at reproducing. The notion of universal Darwinism (Dawkins, 1983) asserts that the same characteristics that make life susceptible to evolutionary change can also be found in other systems. Plotkin (1993) has proposed the g-t-r heuristic as the fundamental characteristic of the general evolutionary process, comprising the three phases of generation, testing, and regeneration.

Numerous evolutionary algorithms (EAs) have been developed which exploit this process for function optimisation, the principal ones being: evolution strategies, evolutionary programming, genetic algorithms, and genetic programming. Common to all of these is the notion of increasing the overall fitness of a population of 'species', say, particular network configurations, by replacing poor species with improved ones (Bäck et al., 1997). For this purpose, a population of offspring

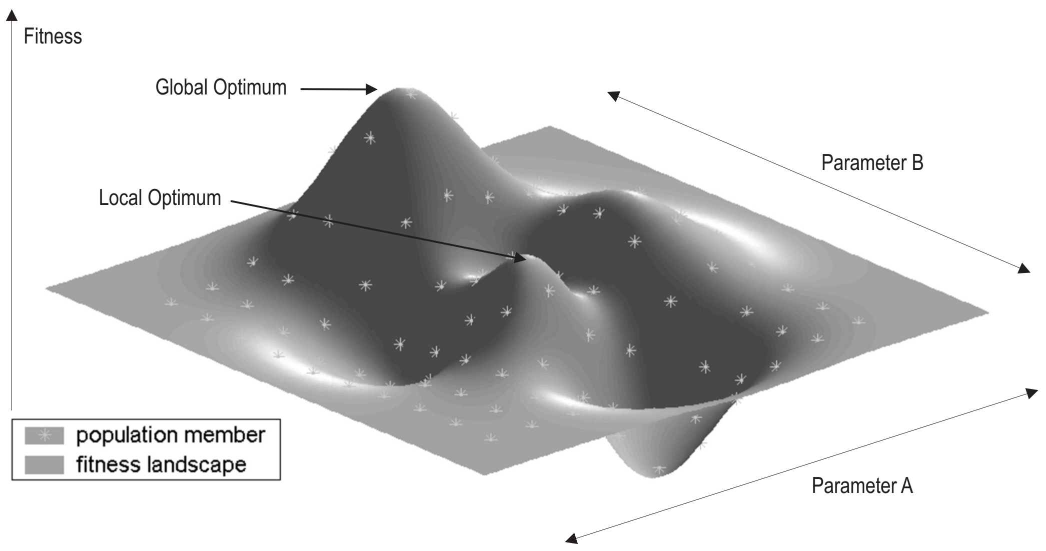


Figure 2.5: A problem that requires the optimisation of two parameters might have a fitness landscape as shown above. An evolutionary algorithm can find the best solution – the global optimum – by sampling the fitness landscape with a population of parameter configurations. If the number of samples is too small, a local optimum may be discovered instead.

is generated from the existing population by means of special operators applied to selected multi-sets of species. Those that are selected form a new population; the cycle is repeated until a species is deemed sufficiently fit to solve whatever problem we had in mind in the first place.

2.5.1 Exploring the Fitness Landscape

The concept of a fitness landscape, an energy landscape modelling the evolutionary process, was introduced by Wright (1967). At each point of this landscape, the vertical position is given by the fitness of the corresponding species (the higher, the fitter), and the neighbourhood of points is given by corresponding variations thereof (see figure 2.5). Mutations represent small and typically undirected variations to a species, analogous to the random fluctuations of simulated annealing. Mutation and natural selection alone are commonly proposed as the key mechanisms for any macroscopic explanation of evolution, i.e. the development of species over time (Bäck, 1996). The evolutionary process can hence be understood to ascend local gradients of the fitness landscape by making small changes in the neighbourhood of the population mean and selecting for variants that are fitter. Evolutionary programming (EP), developed by Fogel et al. (1966), and evolution strategies (ES), developed by Schwefel (1981), are algorithms that pursue this strategy to discover solutions to given optimisation problems. EP uses finite state machines as the representation of individuals, although later iterations of EP use real-valued vectors. ES is targeted at parameter optimisation problems right from the start; consequently, a solution is represented as a pair of real-valued vectors $\overline{v} = (\overline{x}, \overline{\sigma})$, where $\overline{x}$ represents a point in the search space and $\overline{\sigma}$ is a vector of standard deviations. Mutations are realised by replacing $\overline{v}$ so that

$$\overline{\sigma} = \overline{\sigma} \cdot e^{N(0,\Delta\overline{\sigma})} \text{ and } \overline{x} = \overline{x} + N(0, \overline{\sigma}'), \tag{2.1}$$

where $N(0, \overline{\sigma})$ is a vector of independent random Gaussian numbers with a mean of zero and standard deviation $\overline{\sigma}$, and $\Delta\overline{\sigma}$ is a parameter of the method. In the $(\mu + \lambda)$ notation also common with ES and adopted in this thesis, μ marks the number of parents, λ stands for the number of offspring, and the $+$ specifies elitist selection (the best of parents and offspring survive).

2.5.2 Genetic Algorithms

Genetics has extended evolutionary theory by the concept of heredity, with genes acting as transfer units. The genes of an organism constitute its *genotype*, or genome, which is encoded into several chromosomes of DNA. Natural selection applies to the *phenotype*, which is the collection of all measurable characteristics of the organism representing an expression of the genotype within a given environment. In practice, the distinction between genotype and phenotype is somewhat obscured, as, for example, the DNA itself is a phenotypic feature and, conversely, certain properties of the mother's ovum are inherited to the child. Furthermore, the mapping between genotype and phenotype is highly intricate. Genes composed of DNA are transcribed into RNA, translated into polypeptides, and then processed into proteins which self-organise into phenotypic traits (Futuyma, 1998). Complex feedback loops orchestrate the continued expression of genes.

Genetic algorithms (GAs) (Holland, 1992) acknowledge the genotype-phenotype distinction, but the complexities of molecular genetics are typically omitted, and a far simpler interpretation function translates what usually amounts to a binary string into a construct whose fitness can be assessed. (The next chapter is dedicated to more plausible models of the genotype-phenotype map.) GAs are also distinguished from other evolutionary algorithms by creating offspring through recombination, which is meant to reflect the crossover of chromosomes in biological reproduction.

2.5.3 Genetic Programming

Genetic programming (GP) is a genetic algorithm proposed by Koza (1992), which is applicable to any genome that represents its information as an n-ary syntax tree, such as a computer program. A tree is a graph in which any two vertices are connected by exactly one path, which is less restrictive than the linear string representation of the canonical GA. The recombination operator is therefore also different, as it swaps subtrees of the program tree rather than substrings of the genome. By constraining the possible meaning of each subtree, we can enforce that any such recombination will produce a syntactically valid tree as offspring. The size and shape of the solution – and hence of the genome – need not be specified in advance, because any number of subtrees can be added to the existing genome. In contrast, most other genetic algorithms assume fixed size strings, which greatly narrows the range of solutions that can be explored by the algorithm.

Automatically defined functions (ADFs) can be evolved during the run of genetic programming as separate branches of a particular program and may be called from other branches concurrently being evolved (Koza, 1994). This, together with the exchangeability of sub-trees, facilitates the reuse of successful genetic components – and essentially adds cycles into the expression of the trees. Genetic programming with automatically defined functions has been shown to scale well with the problem size, both in terms of the required computational effort and the solution size (Koza, 1994). The theoretical foundations of genetic programming are summarised by Langdon and Poli (2002).

2.5.4 Learning vs. Evolution

Through our distinction of phenotype from genotype, two kinds of adaptation have become possible: those that change the genotype permanently, and those that change the phenotype during its lifetime. Learning is often associated with the latter, as a reintegration of the phenotypic improvements into the genotype is tricky, if at all possible, within the context of biological development and Darwinian evolution. It is, however, the defining attribute of Lamarckian evolution (Futuyma, 1998). Lamarckian evolution is known to be more effective than Darwinian evolution in a stationary environment (i.e. where fitness optima don't shift) (Ackley and Littman, 1994), although less so in a changing environment (Sasaki and Tokoro, 1997).

Baldwin (1896) suggested Darwinian evolution might also be facilitated by learning. Genotypes with inherently low fitness can acquire the knowledge to survive and reproduce through learning. Learning thus produces an enlargement and smoothing of the surface area around a fitness optimum, thereby simplifying its discovery by evolution (Nolfi and Floreano, 1999). Not only does learning benefit evolution, evolution can also improve learning. Since learning affects the fitness of individuals and consequently the choice of individuals selected for reproduction, evolution will tend to select individuals that display good learning traits.

Despite the apparent bidirectionality of the relationship between learning and evolution, it forms part of what in fact is a larger hierarchy. According to the No Free Lunch theorem by Wolpert and Macready (1997), all optimisation techniques, including learning and evolution, must perform identically on average across all possible optimisation problems. A specific technique can be more effective than other techniques when applied to a specific problem, but will be less effective when applied to other problems (Droste et al., 1998). Thus, there exists a universal trade-off between generality and effectiveness of an optimisation technique.

A general technique will make few assumptions about a problem domain; it can enjoy wide applicability, but will scale poorly with more difficult problems (Michalewicz, 1993). This can be avoided by making strong assumptions in the problem solving method, which will limit its applicability. In trying to optimise any type of network design, the only assumptions we can make is with respect to the nature of networks in general. The aim of this research is therefore to establish an optimisation technique that intrinsically operates within the space of graphs, rather than strings or trees, yet also addresses the common difficulty with networks: that they are large. Evolutionary algorithms constitute a solid foundation for this endeavour; the remainder of this chapter reviews further aspects of these and their application to network design.

2.6 Offspring Generation

Evolutionary algorithms create new solutions by variation of existing solutions, with *mutation* and *recombination* being the two principle means by which the variation is generated. Alternatively, offspring may be obtained from an explicit, statistical model of the fitness landscape; methods of this nature are known as

Estimation of Distribution Algorithms. This section serves as an introduction to essential concepts in this area that will be of relevance in later chapters.

2.6.1 Mutation

In biology, the term ‘mutation’ refers to a variety of different, generally random, changes affecting the genome, some of which may also affect the fitness of the phenotype. Since the likelihood of a fitness improvement sharply decreases with the magnitude of a random change, mutations tend to be quite small (Futuyma, 1998). In accordance with this observation, GAs tend to apply only nominal amounts of mutation. In fact, the role of mutation in GAs is often constrained to providing the recombination operator with a full range of alleles (i.e. possible variations on the genetic sequence), rather than to directly contribute to the search process (Holland, 1992). Conversely, ES and EP place an emphasis on the mutation scheme as the sole provider of variations. The choice of mutation scheme makes a significant difference on the search performance here.

An efficient mutation scheme should aim to produce changes that push the phenotype by the right amount into the right direction of the fitness landscape. Achieving this goal depends critically on the probability distribution that underlies these changes. Yet no single distribution can exhibit the versatility that would be required for an efficient exploration of every imaginable fitness landscape. If, for instance, the fitness optimum is close in one dimension, but distant in another, the mutations must be scaled accordingly.

An ES achieves this by adaptively scaling and correlating the output of a predefined multi-variate probability distribution – see section 2.5.1. Unfortunately, this scheme becomes computationally very expensive if used on high-dimensional landscapes. A functional approximation of the probability distribution can be attained by the comparatively simple process of randomly sampling the differences between population members. *Differential evolution* applies these differences as mutations to its population members (Price, 1999). The distribution of mutations is hence determined by the distribution of population members, which in turn depends on the topography of the fitness landscape. The result of this is an adaptive scheme with superior convergence properties, as demonstrated by numerous experiments and its successful application in engineering (Lampinen and Storn, 2004).

2.6.2 Recombination

GAs strongly emphasise recombination over mutation as a source of improved offspring (Bäck et al., 1997). Recombination creates offspring individuals which are combinations of their parents. The resulting adaptive change is very different from the accumulation of mutational changes in a population. A common recombination operator is *crossover*, which exchanges short segments of the parents' genotypes. According to the building block hypothesis, segments, or *schemata*, that confer above-average fitness (so-called building blocks) become increasingly dominant in subsequent generations and form instances of larger segments that confer even greater fitness (Holland, 1992).

Environments where this is not true are called deceptive (Goldberg, 1989a). Several forms of deception may arise when using crossover to evolve networks. For instance, several different genotypes may translate into networks with a common topology and dynamics. This is known as the *competing conventions* (or *permutations*) *problem* (Hancock, 1992). Although these networks behave identically, they represent different optima in the recombination space, so a crossover between such parents is likely distant to either optima. Similarly, any pair of networks whose building blocks are sufficiently incompatible will produce poor offspring. In fact, the success of the building block hypothesis rests on several assumptions that are not necessarily true. For instance, not all possible building blocks are represented in a typical population, and crossover cannot create building blocks; it only breaks them into smaller pieces by converting mutual dependencies into entropy (Toussaint, 2003b). Likewise, in GP, a comparison by Angeline (1997) between subtree crossover and subtree macro-mutation revealed no statistically significant difference.

A key aspect of the apparent ineffectiveness of recombination is that most genetic representations determine the meaning of each gene by its absolute or relative position. If genes that belong to an optimal building block are too far apart, random crossover is more likely to destroy than exchange a building block between chromosomes. Biological crossover is more refined in this respect, as homologous genes contributing to the same trait are lined up by a special protein called *RecA* in a process called synapsis (Radding, 1982). *Linkage learning* in artificial evolution is motivated by the desire that all alleles of a building block should also be inherited together during recombination. A simple system for accomplishing this is to allow the ordering of genes within a chromosome to evolve.

2.6.2.1 Messy GAs

The messy genetic algorithm (mGA) is the earliest algorithm to use such a *floating representation* to achieve linkage learning (Goldberg et al., 1989). The mGA dispenses with the fixed-locus assumption of the canonical GA by defining each gene of a chromosome as an ordered pair $g = (name, value)$. A similar scheme will also be introduced within our system; accordingly, some noteworthy features of the mGA are presented in this section.

Consider a 3-bit problem that requires 3 genes in its solution; consider also the chromosome string $((1,1)(2,0)(1,0))$. The first entry of this string is gene 1 with value 1, the second entry is gene 2 with value 0, and the third entry is gene 1 with value 0. The name of the gene denotes its location in the solution, so the actual chromosome order is less relevant to its fitness (but not irrelevant; see below). Linkage between genes can now be established by having constituent genes of the building block in close proximity to each other.

The drawback of the variable-locus representation of the mGA is that it may be under- or overspecified with respect to the problem being solved. For example, the 3-bit problem presented above is underspecified, because gene 3 is absent; and it is overspecified, because gene 1 appears twice. mGA addresses overspecification by adopting a first-come-first-served policy on a left-to-right scan of the chromosome, i.e. the above chromosome would be expressed $((1,1)(2,0))$ because the second instance of gene 1 would be ignored. Underspecification is solved by employing *competitive templates*. A string specifying a value for each gene name is chosen as a template and any unspecified genes in a chromosome are borrowed from this template. The template is established over several evolutionary eras. In the first era, a random template is used; at the end of each other era, the best chromosome evolved so far becomes the template for the next era.

The mGA further divides each era into two distinct phases: a *primordial phase* and a *juxtapositional phase*. In the primordial phase, the population is initialised to contain all possible building blocks of a specified length, and reproduction and selection are then applied for several generations to increase the proportion of good building blocks. The juxtapositional phase continues this, but also applies genetic operators. Since the mGA representation is of variable length, the crossover is replaced by two operators, *cut* and *splice*. Cut partitions a string with the probability $p_c = (\lambda - 1)p_k$ that grows with the string length λ, while splice joins two string together with a fixed probability p_s.

2.6.3 Estimation of Distribution Algorithms

Estimations of Distribution Algorithms (EDAs) replace the aforementioned variation operators with probabilistic model building based on fit solutions and the sampling of this model to generate offspring solutions. Instead of the parent population, a set of other parameters, e.g. a Bayesian network, determines the offspring distribution. The model is constructed so that it reflects the experienced fitness distribution and targets the search accordingly; this requires the sampling and evaluation of the population by some heuristic rule.

Early EDAs did not estimate any interactions between variables and only generated new solutions by preserving the probabilities of individual variables of a solution; EDAs in this category are the Population Based Incremental Learning Algorithm (PBIL) (Baluja and Caruana, 1995), the Univariate Marginal Distribution Algorithm (UMDA) (Mühlenbein and Paaß, 1996), and the Compact Genetic Algorithm (cGA) (Harik et al., 1999). Because the presence of significant interactions between variables handicaps these algorithms, other EDAs model such interactions.

Pairwise interactions are addressed by the Mutual-Information-Maximizing Input Clustering algorithm (MIMIC) (De Bonet et al., 1997) and the Bivariate Marginal Distribution Algorithm (BMDA) (Pelikan and Mühlenbein, 1999). EDAs that deal with higher order interactions include the Factorized Distribution Algorithm (FDA) (Mühlenbein and Mahnig, 1999), which uses a factorization of the Boltzmann distribution, and the Bayesian Optimization Algorithm (BOA) (Pelikan et al., 1999), which uses Bayesian networks to represent the distribution. EDAs are principally suited only for linear genetic representation, but can be extended to tree and graph representations by the use of grammar models, which will be further discussed in section 3.7.

2.7 Network Evolution

Evolutionary optimisation has been successfully and extensively applied to automata networks, particularly to the weights and structure of neural networks, but also to the transition rules of cellular automata (Mitchell et al., 1997). The former holds the main importance here, since we are principally interested in the design of networks. A comprehensive survey is provided by Yao (1999), to which we direct the reader for a historical overview of the field. The following sections address several points of note regarding weight evolution and basic approaches to structure evolution, before discussing general tree and graph evolution using GP and Cartesian GP, respectively.

2.7.1 Weight Evolution

Weight training by backpropagation may either be inapplicable if the error function is nondifferentiable or become trapped by local optima if the error function is multimodal. Consequently, an attractive alternative for training neural network weights is the use of evolutionary algorithms, which do not require gradient information about the error surface and can avoid local optima by sampling many different points on the error surface at once. In practice, however, the error surface of neural networks is not sufficiently multimodal for this to overcome the computational expense of evolutionary algorithms, especially with large networks, although there are numerous exceptions. Whether to apply backpropagation or an evolutionary algorithm is thus highly problem dependent, and in practice hybrid algorithms often perform better on many problems than either approach alone (Yao, 1999).

Optimising the real-valued weights of a neural network requires these weight to be represented appropriate to an evolutionary algorithm. The canonical GA uses binary chromosomes to encode solutions (Goldberg, 1989b), so much of the early work in weight evolution follows this approach (Caudell and Dolan, 1989). Each connection weight is represented by a fixed number of bits, and the chromosome is simply a concatenation of all these connection weights. Binary representations fit well with the classic GA operators like 1-point crossover and random bit-flipping mutations, but can produce a somewhat arbitrary mapping to connection weights. Weights are ultimately assumed to be real-valued, and very close points in a real-valued space may be represented by very Hamming-distant binary strings. Alternative coding schemes, such as *Gray coding* (Collins and Eaton, 1997), can be used to map real values to special bit patterns that retain the natural ordering of real space.

Directly operating on a real-valued representation is more appropriate for evolving connection weights, but this requires special operators to be designed that replace the classic binary ones. Montana and Davis (1989) trained neural networks using this approach. EP and ES were designed for real-vector optimisation, which also makes them well-suited for finding connection weights. Fogel et al. (1990) used EP to train feedforward networks on some classic connectionist problems. Greenwood (1997) trained partially recurrent neural networks on viseme recognition using ES, which outperformed backpropagation techniques. Results superior to backpropagation were also achieved on a noisy function approximation task, using differential evolution in combination with a simple network architecture evolution (Abbass and Sarker, 2001).

2.7.2 Structure Evolution

Optimising the structure of a neural network may not seem to matter much if the initial network is large and fully connected, as weights can always be set to zero if certain edges or nodes are not needed. However, as elucidated in section 2.4.2, finding a good topology can assist in the objective of finding good weights, and evolutionary algorithms are well-suited for discontinuous search spaces of this kind. Importantly, in problems with high sensitivity to network topology, such as recurrent networks, using an automated approach such as evolution can be a significant time saver for the human user (Gruau et al., 1996).

To evolve a network topology one must first decide on how to represent it. Possible choices are a direct (or *blueprint*) encoding, or one of two types of indirect (or *recipe*) encodings: either a parametric encoding or a generative (or *developmental*) encoding. Each differs in how much information about the architecture is encoded into the genotype and what kinds of architectures are easiest to describe with this information. One of the earliest examples of ANN structure evolution that we know of, by Miller et al. (1989), employs a typical direct encoding. A topology of n nodes is represented by an adjacency matrix M of dimensions $n \times n$ in which element M_{ij} denotes the presence ($M_{ij} = 1$) or absence ($M_{ij} = 0$) of a connection from node i to node j. By concatenating the rows of the matrix, a binary chromosome for evolution by a GA is obtained. The simplicity of this approach allows for easy implementation but has a drawback in that the size of the chromosome scales with the size of the network rather than its complexity. Highly regular networks are just as difficult to find as highly random ones, which means that large networks become difficult to optimise in practice. Kitano (1990) evaluated the direct encoding method for evolving a neural network and found that the speed of evolutionary convergence degrades as the network scales up.

An indirect encoding may be more appropriate when seeking large networks. An indirect encoding typically represents not individual nodes and connections of the network, but interactions between groups of such components. With a parametric encoding, the architecture of the neural network is described in terms of coarse parameters, e.g. the number of nodes in a hidden layer, which are then optimised. This requires certain assumptions about the architecture, e.g. there is a hidden layer, and thereby limits the novelty of the architectures that can ultimately be discovered. Nevertheless, parametric encodings can still be very expressive, as in the case of Harp et al.'s (1989) encoding, which was presented around the same time as Miller above. It employs a binary string to represent only modules of the network and the outward connections from these modules. A module is defined here as a rectangular grid of nodes, fully connected in a feedforward pattern. Within a module, only the nodes in the highest layer have external connections. Each module is assigned an identification number so the target of outgoing projections can be specified.

Generative encodings are constituted by rule sets that can produce a variety of possible networks, typically not just a limited subset as for parametric encodings, although some networks are still more easily expressed than others. The reader is directed to the next chapter for details on generative encodings. Roberts and Turega (1995) compared all three encoding schemes and found that the direct encoding outperforms the indirect encoding on small problems, but vice versa for larger problem sizes; the generative encoding represented an effective compromise.

2.7.3 Cartesian Genetic Programming

Unlike GP, Cartesian Genetic Programming (CGP) uses directed graphs instead of trees to represent programs (Miller and Thomson, 2000). CGP was developed principally for digital circuit evolution, although it has since been applied to artificial life and related domains as well (Rothermich and Miller, 2002). Nodes of a CGP graph must be part of a rectangular grid. The grid consists of $n \times m$ nodes arranged into columns and rows, n_i inputs, and n_o outputs, as shown in figure 2.6. Each component of the grid – nodes as well as inputs and outputs of the grid – has a unique integer index assigned automatically. The grid is populated with nodes according to a fixed length genotype, which is composed of a linear sequence of integers arranged into groups. The last element of a group specifies the function of a node, say a XOR gate; all other elements specify the indices of the grid nodes that the inputs of this node should connect to. Note that a fixed arity is assumed for every function. Nodes can connect to nodes in preceding columns; the number of column to include is defined by the global *levels-back* parameter. Although each node must have a function and resolved inputs for this function, its output need not be used by other nodes. This leads to

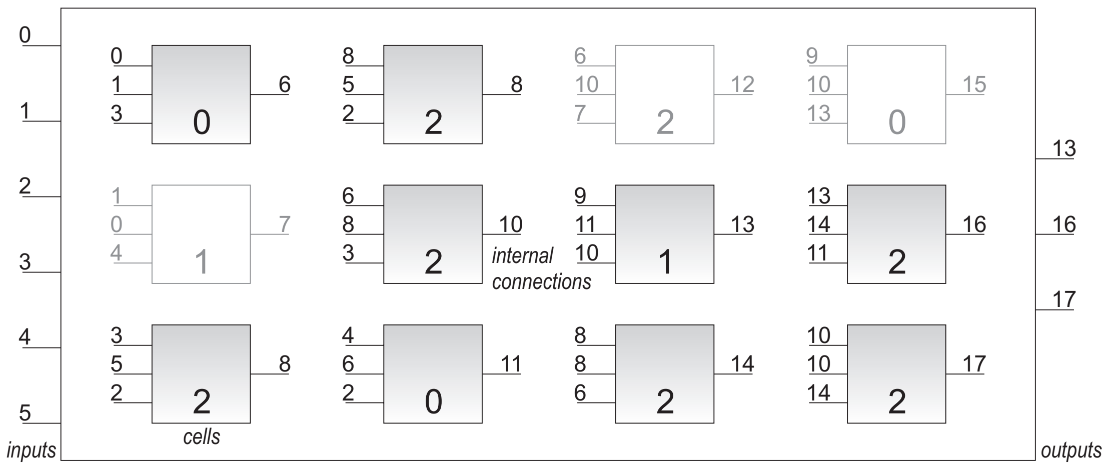


Figure 2.6: Cartesian GP evolves a population of strings, which can be translated into a rectangular array of cells connected according to the input and output indices of each cell. Shown here is program with 6 inputs, 3 outputs, and 3 functions (0, 1, 2 inside cells) (adapted from Miller and Thomson, 2000).

some neutrality in the genotype-phenotype mapping (see also section 3.3 on this topic), which has been claimed to be very beneficial (Vassilev and Miller, 2000; Yu and Miller, 2001). The genotype itself is evolved with a $(1 + \lambda)$ evolution strategy.

2.8 Summary

This chapter provides the reader with a perspective on the nature of change in networks, taking account of the dynamics of networks and how networks can be optimised towards a goal. With respect to network design, artificial evolution suggests itself as a proper balance between search effectiveness and the kind of generality needed for optimising within multiple domains. The efficacy of evolution at this task, however, ultimately depends on its ability to handle the exponential growth of possible network configurations with network size. Unless evolution can utilise knowledge about repetitions and other patterns within these networks, it may quickly become intractable. In view of this and the sheer simplicity of the Darwinian principles, the capacity for complex functional adaptation in biological systems is remarkable. However, the likely source of this is not a sophisticated adaptation mechanism, but a sophisticated functional representation upon which simple mechanisms operate (Toussaint, 2003a). The next chapter will review this idea in the context of artificial evolution, and present a rational foundation for the system presented subsequently.

Chapter 3

From Embryogeny to Graph Grammar

The human brain contains roughly 10^{11} neurons, each with an average of 10^5 connections to other neurons (Bear et al., 2006). Explicitly describing every connection would hence require a minimum of $10^{11} \times 10^5 \times log_2(10^{11}) = 1.7 \times 10^{17}$ bits of storage. In contrast, the human genome is composed of about 3×10^9 nucleotide bases (Venter et al., 2001), representing about twice as many bits of information. These numbers are already several orders of magnitude apart, despite most of the genome not even coding for the brain. If optimising a neural network of the order of the brain was indeed our goal, having such an efficient representation would likely be the deciding factor in our success, quite independent of the optimisation algorithm. This is because the number of samples needed to estimate the space of all possible configurations increases exponentially with the number of parameters constituting each configuration. Unless the topography of the configuration space is trivial, deducing the location of better samples from previous samples amounts to a futile endeavour, as the samples are simply lost in the vastness of high-dimensional space. We also refer to this as the *curse of dimensionality*, which plagues most statistical learning problems and is not easily overcome.

However, redundancy in the phenotype can allow for a lower-dimensional, more efficient genetic representation. We ideally would wish to eliminate all redundancy and determine the most compact representation. This idea is formalised by *Kolmogorov complexity*, which is (roughly) the shortest computer program capable of generating a given message (Li and Vitányi, 1997). Kolmogorov complexity has its roots in probability theory, information theory, and philosophical notions of randomness. Its central notions are perhaps best explained in contrast with classic information theory (Shannon, 1948), which assigns a quantity of information, expressed as bits, to an ensemble of possible messages. The minimal mean length for the bit-string encoding a given message should be proportional to the negative logarithm of the probability of the message, i.e. the more surprising the message, the more information it will carry: $I(m_i) = -\log p(m_i)$.

However, this does not say anything about the number of bits needed to convey any individual message in the ensemble. If a message was composed of a sequence of a million 1's, then we do not need a million 1's to describe this message; we can encode it by expressing the million-multiplier in binary and attaching the repeated pattern '1', i.e. only about 21 bits would be required. Although determining the absolutely smallest program is typically not feasible, the idea is nevertheless applicable. The description of some messages can thus be compressed by a substantial amount, provided they exhibit enough regularity. A requirement for this to work is that we have agreed on an algorithm that decodes the encoded message.

The elaborate mapping from genotype to phenotype fulfills this role in the biological context, and we can likewise exploit it for optimising large network designs. The following sections investigate the expected benefits of *artificial embryogeny* for this purpose and provide a comprehensive survey of earlier work in this area. *Grammars* are subsequently established as formally neat and simply models of generativity, and graph grammars are presented as a suitable means of obtaining networks from such models. Finally, we touch upon the prospect of facilitating the evolutionary search process by building grammar-based models of offspring distributions.

3.1 Embryogeny

Unlike the direct one-to-one mapping function often employed in artificial evolution to translate genes into phenotypic traits, the biological mapping function is highly complex. Genetic changes are not directly manifested in phenotypic changes; rather, a complex developmental machinery mediates between genetic information and the phenotype. *Embryogeny* is the process of growth that defines how a genotype is mapped onto a phenotype up until the fetal stage of development. It is an appropriate and common term for the kind of generative representation that we are investigating here (Stanley and Miikkulainen, 2003).

In particular, embryogeny should exhibit one or both of the related properties of polygeny and pleiotropy, which are not possible with the simple one-to-one mapping (Bentley and Kumar, 1999).

Polygeny refers to multiple genes acting in combination to produce a phenotypic trait, which can have a significant impact on how this trait will evolve in relation to other traits. Conversely, if changes to a single gene affect whole groups of phenotypic variables, this is called *pleiotropy*. Biological embryogeny exhibits some dramatic instances of this. Halder et al. (1995) forced the mutation of a single control gene called *eyeless* in the early ontogenesis of a *Drosophila Melanogaster*. This comparatively minor genotypic variation leads to a highly correlated phenotypic variation: the growth of additional, functionally complete eyes on the wings, legs and antennae of the fruit fly. Pleiotropy here defines the eye as a functional module that can be reused phenotypically by triggering a single gene. By describing the phenotype in terms of such modules, it can be compressed into a much smaller genotype, thereby greatly reducing the number of variables requiring optimisation and allowing evolution to progress in big leaps rather than small steps. However, while pleiotropy enables modularity, it does not guarantee it; additional biases in the representation are needed to facilitate the emergence of modules like *eyeless*.

3.2 Modularity

Modularity concerns the effective partition of sets into distinct subsets. Given a set of parameters, the optimality of any subset of parameters will depend on the configuration of the remaining parameters. Thus, a subset that has been optimised in one particular context may be far from optimal in another context. A system can be understood as being *modular* if it can be described in terms of parameter subsets – modules – that are more tightly coupled internally, i.e. between parameters of a single subset, than externally, i.e. between parameters of different subsets (Simon, 1996). If a subset were entirely independent of external parameters, there can only be a single optimal configuration of this subset – this would be the ideal module. More realistically, a system may be composed of parameter subsets that are marginally dependent upon each other, so that the number of optimal configurations of each subset is – if perhaps not one – at least low. An adaptive mechanism that is able to discover and manipulate these modules in an effective manner would greatly reduce the configuration space that must be searched. Also referred to as *divide and conquer*, this complexity-reducing strategy should be familiar to anyone who has ever tried to solve a tricky problem.

As the fruit fly example indicates, nature follows the same path, but the modularity of biological design is not just visible in experiments on insects. Even the human brain, our poster child for massive complexity, exhibits widespread modularity at various scales of description, e.g. the organisation of the cortex into horizontal and vertical layers defining minicolumns, which are grouped into macrocolumns, which in turn compose larger entities (Mountcastle, 1978). Indeed, it has become apparent that modularity, at either the genetic or phenotypic level, or both, is a necessary characteristic of complex and highly evolvable systems (Wagner, 1995). It would therefore seem sensible to also make provisions for modularity when trying to artificially evolve any large-scale network design.

Extensive practical research on modularity has occurred within the context of tree evolution with GP. The most popular means of modularisation is to use ADFs, which hide functionality into separate branches and can be called as if they were terminals (Koza, 1994). A significant drawback is that the user must define the number of ADFs and the arguments each takes before the evolution commences. *Architecture altering operations* (Koza, 1995) and *adaptive representation* (Rosca and Ballard, 1994) are natural extensions to ADFs, as they allow these parameters to be changed during the run and without requiring user input. Alternatively, with *encapsulation* a point in a tree is chosen and the subtree from that point down is henceforth treated as a new terminal (with no arguments) (Koza, 1992). *Module acquisition* is similar to this, but the subtree is only encapsulated up to a given depth (Angeline and Pollack, 1993). The encapsulated section is shielded from any modification, which has been shown to decrease the time required to find a solution by reducing the amount of manipulations that can take place in the genotype. Walker and Miller (2004) extended module acquisition to CGP and thereby evolved solutions to the even-8 parity problem about 20 times quicker than before.

In the context of networks, modularity entails that inter-module edges are sparser than intra-module edges. Previous studies into hard-coding modularity into fixed-architecture neurocontrollers for game-playing (Togelius and Lucas, 2005) and evolvable-architecture neural networks for visual perception (Di Ferdinando et al., 2001) have presented a strong case for improved evolvability arising from modular designs. However, the structural description of a network module informs us only about the probability of immediate effects between one module and another, which is not necessarily indicative of the extent of consequent state

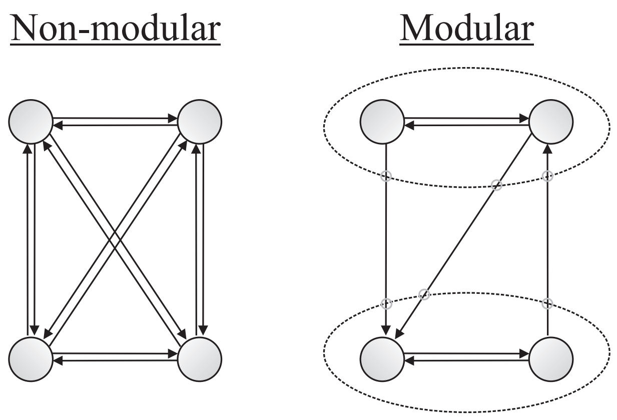


Figure 3.1: Modularity is a reduction in external dependencies between subnetworks, which can be accomplished by encapsulating these subnetworks within interfaces that explicitly restrict the possible structural dependencies (see also section 5.2 on this).

changes over time (Watson and Pollack, 2005). One module may be strongly and non-linearly sensitive to small state changes in another module, despite being sparsely connected. In practice, the optimisation of a system is therefore not as straightforward as its potential for structural modularity might imply. Yet this kind of functional interdependency also has little impact on the choice of representation that we ought to use, as it is not a property that would correlate with a network design in a domain-independent manner. Structural modularity, on the other hand, can be imposed upon a representation, as will be demonstrated for network design in the next chapter.

3.3 Neutrality

If a variation to the genotype of an organism does not affect its phenotype, it is called *neutral*. Neutrality is common in biological genotype-phenotype mappings, because they tend to incorporate some redundancy. One of the pressures on natural evolution is to code the information in a fault-tolerant manner, so that encodings are robust to deleterious mutations. A genetic sequence that has evolved to be robust will have an improved likelihood of successful transmission, and thus will continue to possess a selective advantage. We speak here of *neutrality selection* that acts to increase the probability that mutations are selectively neutral (Ofria et al., 2003).

The existence of neutrality selection also has implications for the evolution of evolvability, an effect otherwise known as *canalisation* (Gibson and Wagner, 2000). Canalisation is a form of genetic buffering where multiple genetic factors within the genome stabilise a developmental pathway (Wilkins, 1992). The expressed number of variations of a phenotypic trait are decreased this way, which affects the evolvability of a phenotypic trait, i.e. its capacity to evolve, since the rate of evolution is proportional to the amount of additive genetic variance. Canalisation also controls the exploration strategy of evolution. By reducing the effects of new mutation, canalisation can allow a build-up of what is referred to as *hidden genetic variation*. This variation is not expressed as phenotypic variation and is hence independent of the selection process. It can continue to accumulate until the canalising system breaks down, either as a result of a change in the selection objective, or after admixture of new variation. The resulting expression of previously hidden variations may produce a more rapid change than might otherwise be expected to occur.

The exploration strategy of an evolutionary algorithm is principally defined by the choice of mutation operators and the representation to which they are applied. An obvious but misleading conclusion is that the adaptability of an exploration strategy requires the adaptability of either the operators or the representation, which, given that the phenotype space is defined by the problem, would entail the adaptability of the genotype-phenotype mapping (Wagner and Altenberg, 1996). However, if the mapping allows of canalisation, then the exploration can adapt even with fixed operators and fixed representation. Neutral variations to the genotype affect phenotype evolution by changing the fitness effects of later mutations, thereby allowing for distinct exploration strategies to be encoded in the same representation (Toussaint, 2003a). The obvious drawback of direct encoding schemes is that they allow for no neutrality beyond what is intrinsic to the phenotype space. Canalisation requires a non-trivial means of representation as made possible by embryogeny.

3.4 Categories of Artificial Embryogeny

Embryogeny allows for a genotype with fewer redundancies yet the facility to encode variable exploration strategies. Exploiting these properties for network evolution requires the design of an artificial embryogeny. Establishing a model of the biological development process is difficult, however, because it is defined by a complex web of interactions between genes, their phenotypic effects, and the environment in which the development occurs. Consequently, existing implementations of artificial embryogeny differ in their degree of sophistication and can be categorised into external (non-evolved), explicit (evolved), and implicit (evolved) (Bentley and Kumar, 1999). The majority of such embryogenies are *external* to the genotype and designed by a researcher. *Evolutionary art* systems, for instance, often use a fixed, non-evolvable embryogeny that specifies how each gene affects the construction of the phenotype (Todd and Latham, 1992). The advantage lies in the extent of control that the designer has over the embryogeny, but because the embryogeny remains static during evolution, the quality of results depends on the quality of the embryogeny the designer came up with in the first place.

Explicit embryogenies grow designs by following the instructions of a developmental program or grammar that is typically also the target of the evolutionary search. *Implicit* embryogenies rely on indirect chains of interacting rules that allow for self-adaptation and independent dynamics of the embryogeny. Implicit embryogenies are types of constrained generating procedures, which, as Holland (1998) describes, resemble neural nets, game theory and classifier systems. In practice, the term tends to refer to systems based on models of gene expression and artificial chemistry. A survey of explicit and implicit embryogenies in the context of network evolution is given below.

3.4.1 A Survey of Artificial Embryogeny

Several different factors of morphogenesis need to be taken into account in order to remain faithful to the biological archetype. This includes chemical factors such as Turing's (1952) mathematical theory of cell-cell interaction via chemical substances and mechanical factors such as Odell et al. (1981)'s physical model of cell membranes. Fleischer and Barr (1994) constructed a simulation framework combining multiple such developmental mechanisms by means of ordinary differential equations (ODEs) which are coupled via *if*-clauses. Simulated chemical gradients are used to determine cell growth and differentiation. This model can reproduce a wide range of biologically relevant developmental phenomena and was also used to evolve desirable ANN topologies (Fleischer, 1995). Kitano (1995) likewise evolved ANNs from a model of cell metabolism and division. The genome encodes metabolic rules linked to ODEs describing chemical reactions and changes within cells, with explicit modelling of diffusion, active transport of chemicals, and special growth factors.

A key to the development of cells is gene expression, with each gene typically interpreted as an instruction on how to modify some aspect of the development process. In biology, the *operon model* explains how genes form networks of complex interactions termed Genetic Regulatory Networks (GRNs). GRNs in biological DNA contain master control switch genes, known as *Hox* genes, which orchestrate the transcription of other genes to grow high-level repeated structure. Lewis (1978) demonstrated that mutations of *Hox* genes can lead to large-scale but localised changes in phenotype. Raff (1996) argues that in some cases, differentiation and/or duplication of a feature may allow evolution to co-opt one copy of the feature to perform a different function role. This process is known as *exaptation* (Gould and Vrba, 1982). A similar mechanism has been shown to occur at the gene level (Ohno, 1970).

Babloyantz and Hiernaux (1974) established one of the earliest artificial systems based on the operon model. It encompasses gene regulation and cell differentiation, and implements chemical reactions as ODEs, but is restricted to modelling only a single cell. Stork et al. (1992) would later be the first to present a gene expression system evolving ANNs. Here the activities of two different types of genes, control genes and structural genes, are directly encoded in an activity table where the state of each gene is determined by evolution. A biologically more plausible encoding scheme has been developed by Jakobi (1995). In his model, genes code for proteins and proteins activate or suppress genes, forming GRNs which constitute independent dynamical systems that can be moved from one basin of attraction into another by internal or external stimuli to the neuron. Each protein also has a unique effect on the gross behaviour of the neuron, including the growth of excitatory or inhibitory dendrites. These neurons then compose recurrent ANNs for controlling a simulated Khepera robot around obstacles.

Astor and Adami (2000) presented an ambitious gene expression model emphasising the artificial life paradigm: to design a system in such a way that complex higher-order structures can emerge from low-level descriptions inspired by biology. In practice, however, only a few neurons were grown in a two-dimensional hexagonal grid. Other complex models of gene expression have been developed by Eggenberger (1997a,b), used to evolve primitive Hebbian neural networks and bilaterally symmetric (i.e. also rather primitive) 3D shapes, and Bolouri et al. (1998), who demonstrated the evolution of an edge detecting retina through an implicit embryogeny. Most recently, Bowers (2005) presented an embryogeny involving diffusion of 20 chemicals, physical cell interaction, and a GRN with

26 chemical-sensitive genes, and applied this to the problem of evolving a visually recognisable French Flag (Wolpert, 1969). The phenotype was shown to be very robust to damage due to the modularity intrinsic to the complex genotype-phenotype mapping. It appears that useful traits must be encapsulated in the genotype to be effectively utilised in evolution, and this happens if the genes that function together are also grouped together.

Growth of shape was earlier explored also by Furusawa and Kaneko (1998) and Hogeweg (2000) in studies on multi-cellularity. A simpler model of establishing shape was presented by Sims (1994) for the evolution and co-evolution of three-dimensional virtual creatures. The genotypes are structured as directed graphs of nodes and connections, which are used to build the morphology of each creature, and can be mutated and recombined during evolution. Each morphological node also constrains graphs of neural nodes and connection that define the dynamics of each part of the resulting creature. Bongard and Pfeifer (2001) combined a gene expression model with cellular encoding (see section 3.5.3) to establish the morphology – including limbs and sensors – and neural control of a multi-articulated simulated agent.

Morphology has also been the focus of early models of neural network growth. Nolfi and Parisi (1991) evolved the position and branching properties of axonal trees spreading out from artificial neurons, and added grammar-based cell division in later work (Cangelosi et al., 1994). The disadvantage of this model is that the number of the genes in the genome grows with the number of neurons, which leads to a poor scaling behaviour. Dellaert and Beer (1996) presents a more sophisticated attempt at implementing a embryogeny model and applying it to neural network growth. Testing it on a basic avoidance task, however, revealed that the size and structure of the search space was intractable. A simplified GRN based on a Random Boolean Network was proposed instead and used to successfully evolve agents that were capable of following a curved line. The genome consists of a fixed number of network nodes, each with two inputs (other nodes are specified by their numerical indices) and a truth table determining the state of the node, which in turn controls various properties of cell division and neural innervation.

Cell development can certainly be modelled without biologically realistic regulatory networks. Downing (2003) ran pieces of Push-like code in modules of a network containing other Push programs. *Push* is a programming language intended primarily for use in evolutionary computation systems (Spector and Robinson, 2002). A genome in Push encodes a list of primitive operations that can act on a variety of data types. Push achieves this by having multiple data stacks, one for each data type. Operators take arguments from the appropriate stack and push them onto the correct stack as well. A module in the network communicates by pushing data onto the stacks of neighbours, migrates by swapping places with a neighbour, and reproduces by copying its Push code into a newly formed neighbour cell. Downing gives examples of how particular Push programs lead to emergent processing behaviours within the resulting cellular ecologies.

Another kind of cell program was demonstrated by Miller (2003) on the French Flag problem. A feedforward Boolean circuit is evolved by CGP to control the quantities of chemical a cell will produce, and how it will grow and change. Finally, a GRN can also be modelled by a recurrent neural network, as implemented by Federici (2005) for the evolution of spiking neurocontrollers for simulated Khepera robots. Compared to direct encoding, the resulting controllers were more parsimonious and revealed a steeper performance increase.

3.5 Grammar Evolution

Many of the above systems maintain an emphasis on biological plausibility and consequently a high degree of complexity, which implies not only a considerable computational cost in simulating the embryogeny, but also a general difficulty in analysing such systems (and rarity; we are not aware of any detailed studies). Consequently, artificial embryogenies often originate from intuition rather than experimentation. A more transparent approach is to model the developmental process as a generative grammar rather than a realistic simulation of biology.

3.5.1 Generative Grammars

A formal generative grammar G is a quadruple (N, T, P, S), where N is a finite set of nonterminal symbols, T is a finite set of terminal symbols (disjoint from N), P is a set of production rules, and S (in N) is a starting symbol, also known as the axiom. 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 $S \in (N \cup T)^*$. A derivation is a series of rule applications to a string. A formal grammar G defines a formal language L of all the strings that can be generated by a derivation from a starting symbol. For example, the following grammar with terminals $\{a, b\}$, nonterminals $\{S, A, B\}$, and production rules

$S \rightarrow ABS$

$S \rightarrow \epsilon$

$BA \rightarrow AB$

$BS \rightarrow b$

$Bb \rightarrow bb$

$Ab \rightarrow ab$

$Aa \rightarrow aa,$

where S is the starting symbol (and ϵ is the empty string), defines the language of all strings of the form $a^n b^n$ (i.e. n copies of a followed by n copies of b). Note that all symbols in N that do not appear as the left side of a production rule are, by default, rewritten into themselves. These default rewriting rules are usually not included in P. In a context-free grammar (CFG), the left hand side of a production rule may only be formed by a single nonterminal symbol, whereas no such restriction applies to context-sensitive grammars such as the one above.

3.5.2 Lindenmayer-Systems

Lindenmayer (1968) introduced Lindenmayer-systems (commonly referred to as *L-systems*), which replicate the growth characteristics of linear and branching structures observed in plant morphogenesis. L-systems are best known for the plant-like fractals they generate, but are also able to model the morphology of a variety of organisms or structures. An L-system is a special kind of string grammar. Whereas the production rules of most grammars are applied sequentially, an L-system rewrites all the symbols of a string concurrently to form the new string. (This is not unlike the updating of cellular automata discussed earlier in section 2.2.1, but with the additional freedom of changing the number of cells.) For illustration, assume an L-system with

$N = \{A, B\},$

$T = \{\},$

$P = \{(A \rightarrow AB),\ (B \rightarrow A)\},$

$S = \{A\}.$

This is Lindenmayer's original model for the growth of algae. After the n-th replacement iteration the following strings would be generated:

$n = 1 : A \rightarrow AB$

$n = 2 : AB \rightarrow ABA$

$n = 3 : ABA \rightarrow ABAAB$

$n = 4 : ABAAB \rightarrow ABAABABA$

$n = 5 : ABAABABA \rightarrow ABAABABAABAAB$

By iteratively applying the production rules from the axiom, a more complex string appears to arise from a succession of simpler ones. Obtaining this string is rarely the final objective, however. The symbols of the string can be interpreted as instructions to produce another artifact, e.g. as commands for a LOGO-style turtle, such as "turn left" and "draw line" (Abelson and diSessa, 1981). To draw naturally branching plants this way, bracketed L-systems are usually employed, which include [and] as symbols to respectively remember or restore the last position and direction of the turtle.

For modelling interactions between neighbouring cells, it is possible to base L-systems on a context-sensitive grammar. An L-system without context is known as a 0L-system. If some production rules have a one-sided context (i.e. $cA \rightarrow B$ or $Ac \rightarrow B$, where c is not replaced), it is a 1L-system, and a 2L-system may have context on both sides ($cAc \rightarrow B$). Another extension of the L-system is the class of parametric L-systems (PL-systems) (Lindenmayer, 1974). These differ from the basic L-systems in that production rules are associated with parameters. Parameter values can be modified by algebraic expressions and may have conditions that determine which production rule to apply. The principal advantage of a PL-system over a basic L-system is that the given PL-system can produce a family of strings depending on the chosen parameter values. For instance, assume a PL-system similar to the example above, with

$N = \{A, B\},$

$T = \{\},$

$P = \{(A(n) : \ n > 0 \rightarrow A(n-1) \ B(n-1)), \ (B(n) : \ n > 0 \rightarrow A(n-1))\},$

$S = \{A(4)\}.$

After the n-th replacement iteration the following strings would be generated:

$n = 1 : A(4) \to A(3)B(3)$

$n = 2 : A(3)B(3) \to A(2)B(2)A(2)$

$n = 3 : A(2)B(2)A(2) \to A(1)B(1)A(1)A(1)B(1)$

$n = 4 : A(1)B(1)A(1)A(1)B(1) \to A(0)B(0)A(0)A(0)B(0)A(0)B(0)A(0)$

$n = 5 : A(0)B(0)A(0)A(0)B(0)A(0)B(0)A(0)$ {no replacement}

The conditions of the two productions rules effectively terminate the further growth of the L-systems after 4 iterations, because the initial parameter of A was 4.

3.5.2.1 Graph L-System

Kitano (1990) and Boers and Kuiper (1992) are pioneering examples of evolving grammar-based encodings of graphs. Kitano (1990) employs a context-free, deterministic *graph L-system*, which is an extension to the conventional L-system apparently first proposed by Doi (1988). The graph L-system G_{GL} is the sextuple $(S_N, S_E, M_N, M_E, P, S)$, where

- S_N is a finite set of node symbols,
- S_E is a finite set of edge symbols,
- M_N is a finite set of 2×2 matrices whose (i, j)-th element is $a_{ij} \in S_N$,
- M_E is a finite set of 2×2 matrices whose (i, j)-th element is $b_{ij} \in S_E$,
- P is a set of production rules, and
- $S \in S_N$ is the axiom (i.e. the starting symbol).

Each production rewrites a node or edge symbol within a node or edge matrix into another node or edge matrix until all symbols that can be rewritten are rewritten. Kitano (1990) uses graph L-systems to provide a compact representation of neural networks for evolutionary optimisation. The production rule set for each network is translated into a binary string and stored in a chromosome; the rule sets are then evolved in this format by a genetic algorithm. Kitano applied neural network evolution to an encoder/decoder problem and observed that the graph L-system encoding produced a faster convergence than the direct encoding. It also scaled better on harder problems where larger networks were required.

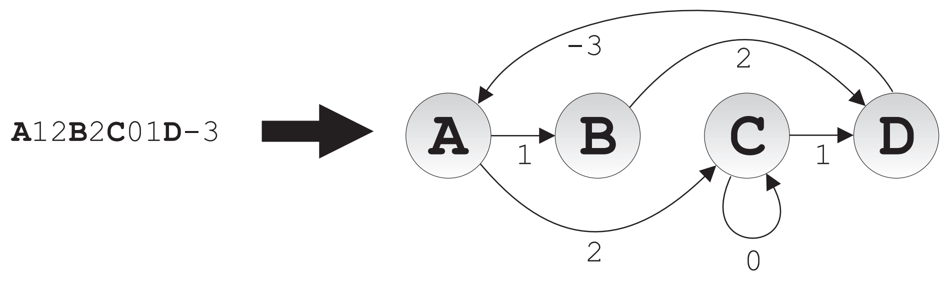


Figure 3.2: The string on the left is translated into the graph on the right using the G2L-system (adapted from Boers and Sprinkhuizen-Kuyper, 2001).

3.5.2.2 G2L-System

A more conventional 2L-system is used by Boers and Kuiper (1992), where the string that results from the rewriting is interpreted as a graph. This was later named the G2L-system and generalised to cyclic graphs (Boers and Sprinkhuizen-Kuyper, 1995). Neural network optimisation is again the sole application of this system. A string in the G2L-system is the set

$$\Sigma_g = \Im \ \cup \ \{[,]\} \ \cup \ \Sigma$$

with Σ being the finite set of symbols of the language and $\Im$ the set of integers where each $j \in \Im$ is read as a *connector* symbol. The square brackets [and] are used to denote modular subgraphs. Each $n \in \Sigma$ in the string represents a node in the corresponding graph. The integer j behind n or] connects the node or subgraph, respectively, with a directed edge to the j^{th} node or subgraph to the right of the string if j is positive, or the left of the string if j is negative. For an illustration of this, refer to figure 3.2.

When seeking the j^{th} node or subgraph, each subgraph is seen as a unity and regarded as a single node. The connectors immediately to the right of] attach all output nodes of the subgraph, while any connection made to the subgraph is attached to all of the subgraph's input nodes. Output nodes are defined as those nodes that have no outgoing edges to other nodes of the same subgraph; input nodes are those nodes that do not have any incoming edges from within the same subgraph. Each production of the G2L-system has the format

$$L < P > R \rightarrow S$$

where L is the left context, R is the right context, P is the predecessor, and S is the successor. P and S contain fully defined modules and nodes. The meaning of L and R differ from normal 2L-systems, where the context of a symbol being rewritten is part of the string directly to the left and right of that symbol. In the G2L-system, L is instead matched against the coding symbols of the nodes from which nodes in P receive their input, while R is matched against the nodes attached to the output from one or more nodes in P. Matching occurs in a left to right order, and more specific production rules are preferred if there are multiple matches; the specificity of a production is defined as the number of symbols on the left-hand side of the production.

Each symbol of a production rule is represented by a 6-bit string. An asterisk acts as a start and stop marker, separating each L, P, R and S and each complete production. The genotype of a network is the sequence of symbols that define the production set for this network. The fittest neural network topology is evolved from a population of genotypes by a standard GA; the weights of the network are optimised using error backpropagation. Very large modular neural network architectures can be evolved from comparatively small genotypes using the G2L-system, although, to the author's knowledge, no experiments have been done to show this to be applicable in practice.

3.5.2.3 GENRE

Hornby and Pollack (2001a) extended the use of evolved L-systems beyond neural networks to general design. This research followed earlier work by Lipson and Pollack (2000) on evolving simple robots represented as graph-based data structures with edges connecting bars, actuators, and artificial neurons. A direct encoding scheme was used here. It was shown that systems beyond a certain level of complexity were impractical to evolve and that the systems obtained had unnatural, highly asymmetric morphologies (Hornby and Pollack, 2001b). In contrast, Hornby's (2003) evolutionary design system, called GENRE, is based on a P0L-system. The final string produced by the L-system after a given number of rewrites is passed on to a design constructor that interprets the string symbols as instructions to build a particular design.

The L-system is evolved directly by a simple evolutionary algorithm that uses specialised variation operators. It is initialised from a blank template of a fixed number of production rules (i.e. predecessors), each with a fixed number of conditions and successors. Conditions are created by randomly choosing a parameter and a constant value to compare against; successors are random sequences of production and terminal symbols. Mutating the L-system involves random selection of a rule, which is then changed by randomising, incrementing, decrementing, adding, or deleting symbols or conditions; alternatively, a sequence of symbols may also be encapsulated and placed into a previously unused production rule. The recombination operator randomly takes symbols and conditions from two existing L-systems and creates a new L-system from these.

GENRE was evaluated on four design classes: table designs, neural networks, robots controlled by oscillator networks, and robots controlled by neural networks (Hornby, 2003). In comparison to a non-generative representation, variations to designs (especially large variations) encoded with an L-system were more likely to be successful during evolution, leading to fitter solutions being evolved within a shorter time span.

3.5.3 Cellular Encoding

L-systems rewrite strings according to rules, and the systems presented above store these rules as strings and evolve these strings. Use of other data structures has been uncommon, but a notable exception is Cellular Encoding (CE), which was introduced by Gruau (1992). CE represents the graph rewriting rules as a tree, or a list of trees, known as *grammar trees*. The nodes of the tree are references to graph transformations applied successively to develop a single *ancestor cell* into a neural network (specifically in Gruau's work). Figure 3.3 illustrates this process. Each cell carries duplicate copies of the grammar trees and a pointer to a particular tree node, whose associated transformation is executed before the pointer moves on to the following node.

Gruau's (1995) basic CE framework defines a number of instructions for graph transformation, some of which are also shown in figure 3.3. A *sequential division* produces two daughter cells that replace the mother cell; the first cell inherits the incoming edges of the mother cell, while the second child inherits its outgoing edges. In a *parallel division* both daughter cells inherit the incoming and outgoing edges of their mother. Other instructions allow modifications to the thresholds of neurons or the weights (or existence) of incoming edges. A *recurrence* instruction permits the reuse of other instructions by moving the cell pointer to the root of a grammar tree; recurrence continues until a cell-specific limit (called *life*) is reached for the tree root. CE can thereby efficiently represent repeating structures in problems such as parity and symmetry. The growth process otherwise stops on an *end program* instruction.

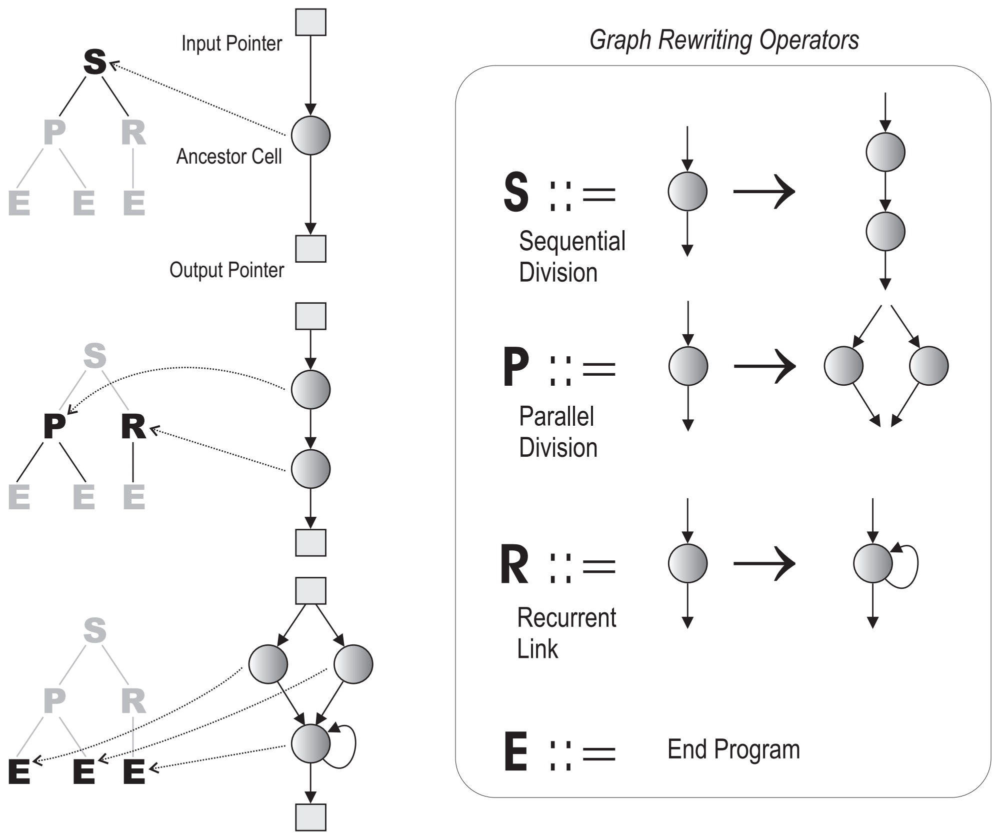


Figure 3.3: Network construction according to cellular encoding: the left half of this figure depicts a tree of graph rewriting operators, together with the developing network of cells. Dotted arrows point to the instruction that applies to the cell at each step. The subset of CE operators employed here is defined on the right-hand side.

Gruau (1994) proved that CE can be translated into a proper graph grammar (see also section 3.6). The principle benefit of the tree representation is that it allows for optimisation by GP. Subtrees can be swapped with those of other trees, and as the network specification is hierarchically organised (as a tree), this facilitates the identification of useful modules. However, reuse is otherwise quite restricted, because the recurrence instruction only points at roots and uses relative indexing of trees, so a recurrence is not always portable between trees.

The choice of defined graph transformation also imposes a strong bias on the kind of networks that are discovered. Inevitably, there has been some disagreement on what is the best set of transformations. CE as described above uses node operators, i.e. all operators apply to cells and may result in one or more other cells, with only weak control over the edges between nodes and often generating highly connected networks. Luke and Spector (1996) therefore propose cellular encoding by edge operators rather than node operators, and De Jong and Pollack (2001) use both node and edge operators in evolving recurrent neural networks for signal reproduction.

3.6 Graph Grammars

The theory of graph grammars can be viewed as an extension of the theory of formal languages to graphs. Just like sets of strings can be characterised by string grammars, sets of graphs can be characterised by graph grammars. Graph grammars therefore provide an intuitive description for the manipulation of graphs and graphical structures in any applicable domain. Over the last 30 years of research a great many graph rewriting mechanisms have been devised, which can generate a variety of classes of graphs with a diverse range of properties. The reader is referred to Rozenberg (1997) for a comprehensive review of this area.

Graph rewriting can be separated into node rewriting (or *vertex replacement*) and edge rewriting (or *hyperedge replacement*, see section 3.6.1), depending on whether graphs are seen as sets of vertices linked by edges or as sets of edges glued by vertices. Vertex replacement grammars are regarded as more powerful than hyperedge replacement grammars, but hyperedge replacement grammars are easier to implement. In both instances, the mechanism is defined by what is being replaced, what it will be replaced by, and how either connects to the rest of the graph. Advanced rewriting mechanisms in each category, such as *double pushout rewriting* and *NCE-rewriting*, respectively, can rewrite entire subgraphs at once.

A rather different approach to graph rewriting is to define a number of operations on graphs and consider a string language of expressions over these operations (Vereijken, 1993). With a string grammar and an appropriate function that interprets each string of the generated string language, a “graph grammar in disguise” is established, which can indeed by very powerful; e.g. see Lucas (1995) for how a perceptron that internally performs backpropagation can be coded from this. The grammatical development models presented in section 3.5 are further instances of this approach. In notable contrast, the research here will pursue a different direction by operating directly on a graph grammar as described below, thereby avoiding the need for predefined graph operations and the interpretation stage.

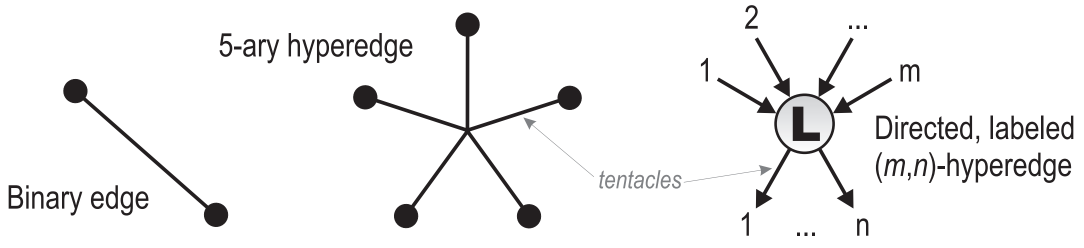


Figure 3.4: A hyperedge is an atomic item with an ordered set of incoming tentacles and an ordered set of outgoing tentacles, which are typically not singleton sets, unlike for the binary edge.

3.6.1 Hyperedge Replacement Systems

Hyperedge replacement is one of the most elementary and frequently used concepts of graph rewriting and was originally introduced by Feder (1971) and Pavlidis (1972). It constitutes a solid foundation to work with, as it is rich with theoretical results corresponding to the properties of context-free Chomsky languages. In essence, hyperedge replacement is a type of edge rewriting extended to hyperedges. Edges in a graph normally have arity two, that is, they connect two vertices. A *hyperedge* may instead have multiple sources and targets; it connects several vertices via a set of incoming tentacles and a set of outgoing tentacles, as depicted in figure 3.4. A graph with hyperedges is known as a *hypergraph.* Formally, a directed, labelled 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,
- $s : E \rightarrow V^*$ assigns a sequence of sources $s(e)$ to each $e \in E$,
- $t : E \rightarrow V^*$ assigns a sequence of targets $t(e)$ to each $e \in E$,
- and $l : E \rightarrow C$ labels each hyperedge.

Whereas in graphs the nodes are generally regarded as objects and edges as relationships, the role of nodes and edges is reversed in hypergraphs. Edges represent the primary objects that are related via nodes.

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 $(V, E, s, t, l, \mathit{begin}, \mathit{end})$ where (V, E, s, t, l) is a hypergraph over C and $\mathit{begin}, \mathit{end} \in V^*$. Let H_c be the set of all multi-pointed hypergraphs. A *hypergraph production* is an ordered pair $p = (A, R)$ with $A \in N$ and $R \in H_c$. A and R will also be referred to as the left-hand side (LHS) and right-hand side (RHS) of the production, respectively. A *hyperedge replacement grammar HRG* is a quadruple (N, T, P, Z) where:

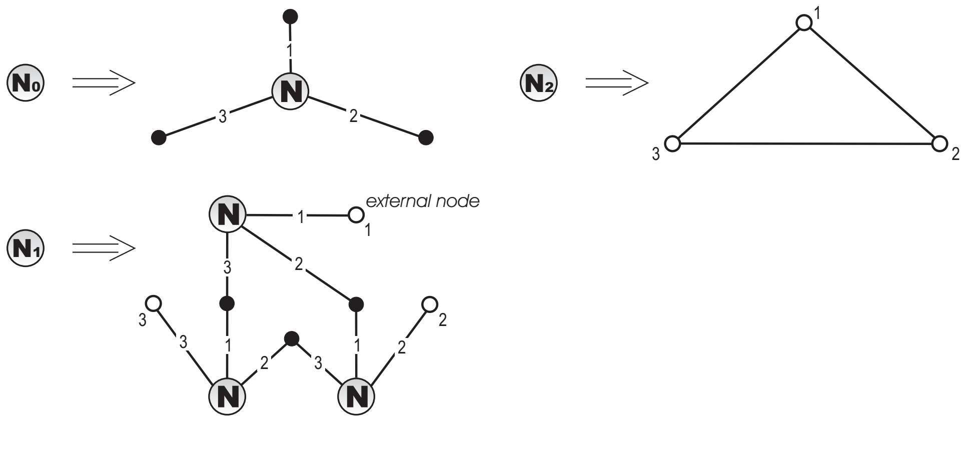
- $N \in C$ is a finite set of nonterminal symbols,
- $T \in C$ is a finite set of terminal symbols,
- P is a finite set of hypergraph productions,
- and $Z \in H_c$ is the axiom.

Hyperedges of a hypergraph may be replaced by other hypergraphs according to hypergraph productions. 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 attachment nodes in H, then e may be replaced by R. This occurs by removing the hyperedge and adding the hypergraph R, except for the begin and end nodes; each tentacle of a hyperedge within R that is attached to a begin or end node is handed over to the corresponding source or target attachment node of the replaced hyperedge e.

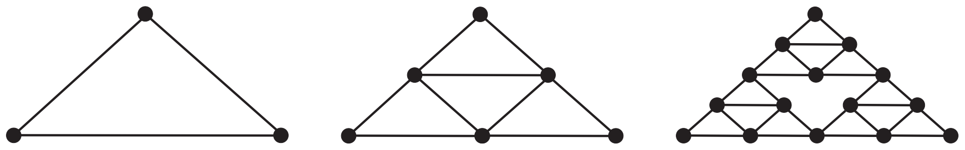
Habel (1992) provides an illustration of hyperedge replacement, which is reproduced below. The hypergraph grammar is

$$HRG = (\{S, N\}, T, \{(S, N_0), (N, N_1), (N, N_2)\}, S),$$

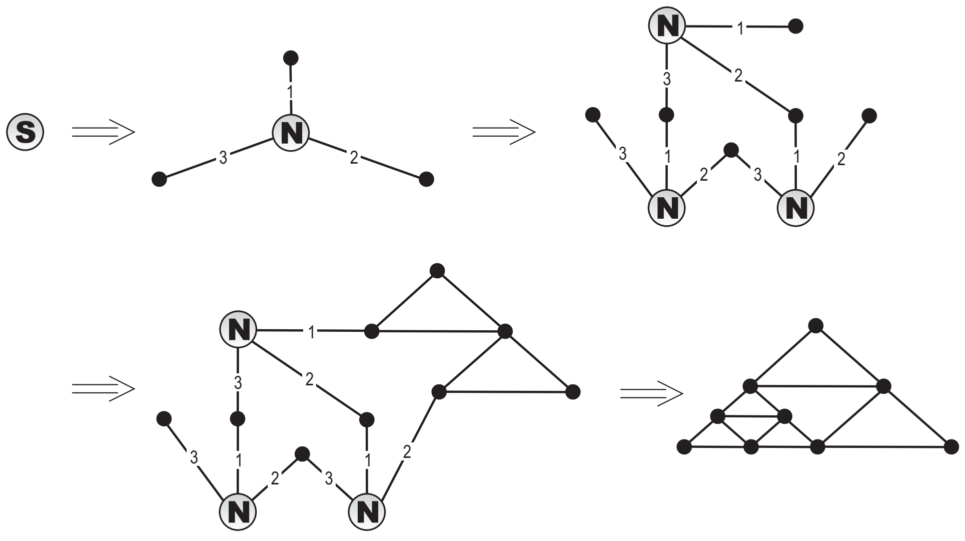
where



This grammar generates irregular Sierpinski triangles, for example,



or, alternatively, the triangle generated by the following derivation:



A complete formal treatment is available in Habel (1992) for the curious reader. Note that hyperedge replacement can be extended beyond a CFG into *parallel hyperedge replacement*, which generalises L-systems to graphs (Kreowski, 1993). However, for the purposes of evolutionary optimisation, we will only use the context-free model in this thesis.

3.7 Grammar Model-Building

Grammars in general have found widespread use in evolutionary computation. A classic example is Grammatical Evolution (GE) (Ryan et al., 1998), a GA designed to evolve programs in any language that can be generated by a CFG. It is based on earlier work by Paterson and Livesey (1997). The GE genotype is a variable-length bit string that is read left to right to generate 8-bit integers (referred to as *codons*). The modulo of each codon and the total number of productions is used to specify the production rule to be applied for the currently replaced nonterminal, starting from the axiom and ending when all nonterminals have been replaced. If all codons are read but not all nonterminals are replaced, the expansion continues from the start of genome. The parse tree of the phenotype is generated depth-first, i.e. the left-most branch of the tree is completed first. The use of a simple one-point crossover in GE means that left-most and shallow sections of the tree are more stable than right-most and deep sections, since the tail of the chromosome will often map into a completely different set of productions after crossover. Instead of exploring all branches in parallel, the tree is optimised sequentially left-to-right, which increases the risk of being trapped by local optima. Kubalík et al. (2003) have suggested a bidirectional representation to address this.

The purpose of the grammar in GE is to solve the issue of *closure*: the requirement that any evolved solution is syntactically legal, which is a constraint that applies to any problem that involves typed arguments. With CFG-GP, Whigham (1995) first introduced the use of CFGs to overcome closure issues in GP, but also went a step further. GE samples solutions from a static grammar – there is no mechanism for changing the grammar. What if the grammar could be adapted as the evolution of solutions progresses, so that the grammar would not just produce legal solutions, but better solutions? Whigham (1995) biased a CFG towards this goal by choosing the fittest member of a population and propagating one of its terminals up the program tree to the next level of nonterminals. This creates either a new production that is added to the grammar, or if that production already exists, increases its *merit value*. The merit value changes the probability of a production being applied when a solution is generated by the grammar.

Hoai and McKay (2001) also used CFGs but in conjunction with lexicalized tree-adjunct grammars (LTAGs), which are tree-rewriting systems. Productions in this system consist of so-called *elementary trees*, each of which must have at least one terminal node. This is easier to handle than CFG-GP because it uses a linear genome like GE, but unlike GE, all genotypes will produce legal phenotypes. In GE this is not always guaranteed, because, for example, a production sequence where nonterminals are only replaced by nonterminals will never finish. Abbass et al. (2002) presented AntTAG to explore tree derivation from LTAGs using ant colony optimisation (ACO) (see section 7.4.1 for a discussion of ACO), but the grammar itself remains unchanged. Conversely, PEEL (Program Evolution with Explicit Learning), which also applies an ACO, but to a stochastic parametric L-system, allows a change of grammar structure (Shan et al., 2003). The production rules used to generate the fittest individuals of the population, and rules with the same LHS, are split into multiple productions and/or mutated with a specific probability. Grammar Model-based Program Evolution (GMPE) constitutes a further refinement of this process of modifying a grammar to produce the best possible solutions (Shan et al., 2004). GMPE applies a stochastic hill-climbing search to learn a stochastic CFG from the best solutions in the existing population. A grammar that specifically describes only the fittest population members is established at each generation and then generalised by merging rules with the goal of minimising the minimum description length of the grammar. A fraction of the next generation is then sampled using this grammar, and the procedure repeated. Novelty arises from adding random solutions to the population. The practical outcome of this is an extension of EDA to tree space, but using a grammar, not a probability distribution.

A changing grammar is also feasible in GE, as shown by O'Neill and Ryan (2004), although the grammar is evolved here and not learned. The meta-Grammar Genetic Algorithm (O'Neill, 2005) consists of a universal grammar and a solution grammar. Two chromosomes are used: the first generates the solution grammar from the predefined universal grammar, and the second generates the solution itself; crossover only operates between homologous chromosomes. In this framework, both the solution grammar and the solution based on this grammar must coevolve. Experiments reveal significant performance gains relative to static grammars.

3.8 Summary

This chapter highlights important features of embryogeny as seen in biology, including the potential for pleiotropy, modularity, and neutrality. The associated benefits of such features encourage us to employ a generative representation in the evolution of network designs. The previous use of artificial embryogeny for related purposes has been reviewed here and can be separated into biologically realistic models and more abstract grammatical models. We intend to move against the general trend by choosing an abstract model, because of its transparency and consequent openness to a systematic investigation.

Network design by L-systems or CE is typically based around a fixed set of user-defined graph transformations, involving specific assumptions being made on how to describe the network. It does not need to be this way, however, as our discussion of hypergraph grammars has made evident: the graph transformation can itself be a production rule and thereby be modifiable by evolution. An evolved bias of graph transformation thus becomes possible. We strive to design a generative representation that maintains this property, based on a hypergraph grammar. The proposed framework is named *Cellular Graph Grammars* and will be presented, together with a method for evolving the grammars, in the following chapter.

Chapter 4

Cellular Graph Grammar Evolution

If networks are to be represented by graph grammars, then the problem of optimising a network becomes one of optimising a grammar. The previous chapters introduced evolutionary algorithms as optimisation tools and the hypergraph grammar as a powerful type of graph grammar. Combining these two raises some challenges. Evolution is based on random change, which here means random change to grammatical productions. However, in literature on hypergraph grammars it is typically assumed that the replacement of a hyperedge is well-typed, i.e. the hyperedge being replaced has a set of tentacles that match the external nodes of the multi-pointed hypergraph.

For illustration, let us assume that a nonterminal hyperedge labelled N_2 is added to the RHS hypergraph R_1 of the production associated with the hyperedge labelled N_1. When rewriting the hypergraph, hyperedge N_2 is replaced by multi-pointed hypergraph R_2, which must attach to the sources and targets of the hyperedge. If R_2 is changed so that its external nodes do not match the replaced hyperedge, then the type-correctness of the replacement is only maintained if R_1 is also changed – and this may require further changes to other hypergraphs that R_1 includes or that include R_1. Obviously, it would be far simpler and less restrictive if type differences are allowed and somehow resolved implicitly without needing to explicitly address all the possible interactions with other hypergraphs.

A further issue with the evolution of hypergraph grammars is that each production in the grammar must define a multi-pointed hypergraph, which must be somehow represented. Yet even a direct encoding of graphs in general, e.g. as matrices, requires rather sophisticated variation operators if the number of nodes is variable, because any new node will have to be connected to existing nodes if it is intended to make a difference. The novel representation that we introduce in this chapter uses a set of productions of hypergraph fragments called *cellular graphs* to describe a hyperedge replacement grammar. We subsequently present an evolutionary algorithm for optimising a *cellular graph grammar* from which a population of networks may be derived. The algorithm is packaged into a system (and corresponding software suite) called G/GRADE. Several applications for network evolution using G/GRADE are also described.

4.1 Cellular Graphs

The handover operation usually defined for hyperedge replacement fuses the i-th source with the i-th begin node and the j-th target with the j-th end node. Thus, the type-correctness of multi-pointed hypergraphs R_1 and R_2 can be ignored by simply not trying to fuse any nodes beyond those that are present. However, this may lead to further side-effects if R_1 or R_2 are later modified, e.g. during evolution. For instance, if the first begin node is removed, the formerly i-th begin node will now attach to the $(i-1)$-th source node, as illustrated in figure 4.1. The longer the sequence of nodes in which such a mutation occurs, the larger the ripple effect on the topology. A minor mutation can thus produce major changes to the fitness of the derived graph, which may lead to an evolutionary bias against changing external nodes that exist early in the sequence. As likewise noted for crossover in Grammatical Evolution (Kubalík et al., 2003), this leads to a left-to-right optimisation of the nodes, which has a greater chance of being trapped by local optima.

Position independence (for the purpose of linkage learning) has been achieved in GAs such as the Messy GA (Goldberg et al., 1989) 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. So while a begin node with label $l(k)$ is replaced by the k-th source, the k-th source may have changed because a tentacle was added to or removed from the hyperedge. The solution is to extend the mappings s and t so that the label of the external node of the multi-pointed hypergraph is specified; the mappings hence become $s : E(l) \rightarrow V^*$ and $t : E(l) \rightarrow V^*$.

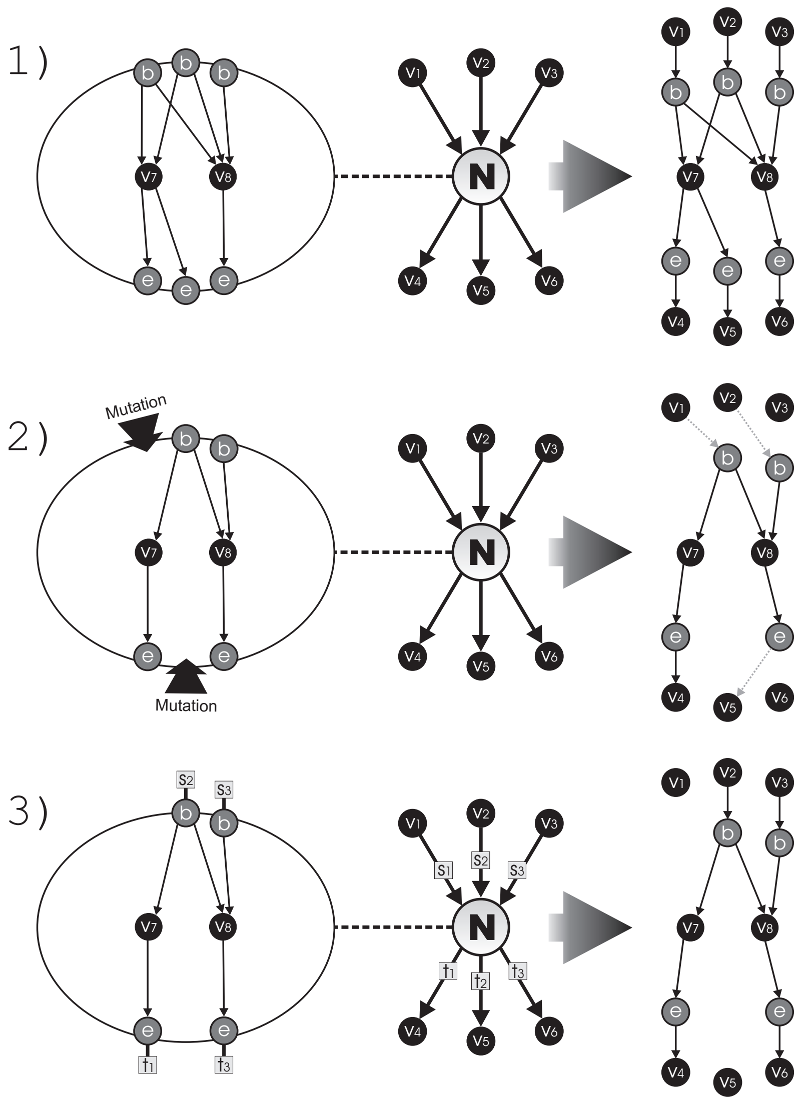


Figure 4.1: 1) Hyperedge N is replaced by the graph on the left, producing the graph shown on the right if the hyperedge tentacles are connected in a fixed order. 2) The graph associated with N is mutated by deleting a begin and end node, but this has also changed the attachments of the other nodes. 3) These side-effects are avoided by applying additional node and tentacle labels; now the node labels can be matched directly against the tentacle labels.

A multi-pointed hypergraph with node and tentacle labeling is a notably larger construct than without. This raises the question of how to represent the hypergraph for evolutionary optimisation. A directed hypergraph with n vertices and m hyperedges can be represented as an $n \times m$ incidence matrix in which a nonzero entry exists in M_{ij} if and only if vertex i is incident to edge j. Alternatively, it can be described by an incidence structure. The incidence structure 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. Since this can be represented as an adjacency list, it is hence particularly suited for describing sparse hypergraphs, which are analogous to the kind of large, modular network designs we are intending to optimise.

As aforementioned, however, we need a more reliable means of identifying vertices and hyperedges than a sequential numbering such as i and j above. The hypergraph representation must assign a label l to each vertex and hyperedge. The exact nature of the labels will be explored in the next chapter; for now, it is sufficient to know that the set of labels can be partitioned into $L_E \in C$ for hyperedges, $L_V \in C$ for internal vertices, $L_b \in C$ for begin nodes, and $L_e \in C$ for end nodes. A hypergraph is represented as an adjacency list of hypergraph elements identified by these labels. Specifically, it is a set of ordered pairs (L_S, L_T), where $L_S \in \{L_E, L_V, L_b\}$ and $L_T \in \{L_E, L_V, L_e\}$, but excluding any adjacency between hyperedges (i.e. (L_E, L_E)) and adjacency between external nodes (i.e. (L_b, L_e)).

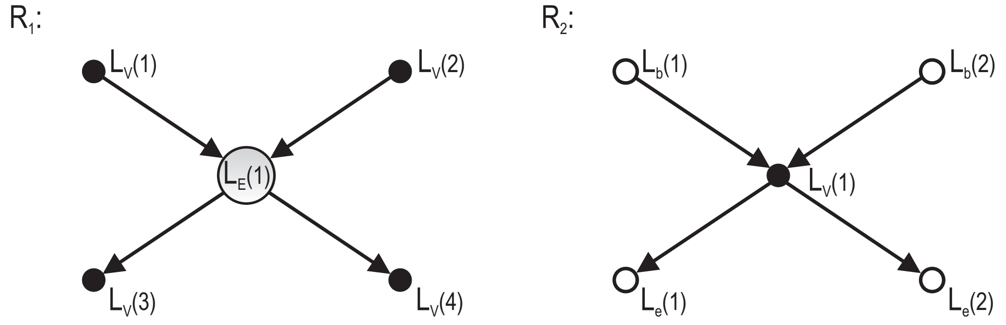


Figure 4.2: Two labelled hypergraphs assumed to be used in a hypergraph grammar that includes the production $L_E(1) \to R_2$.

The following example uses this representation for the two hypergraphs R_1 and R_2 depicted in figure 4.2:

R_1: $(L_V(1), L_E(1)), (L_V(2), L_E(1)), (L_E(1), L_V(3)), (L_E(1), L_V(4))$

R_2: $(L_b(1), L_V(1)), (L_b(2), L_V(1)), (L_V(1), L_e(1)), (L_V(1), L_e(2))$

No position independence with respect to the tentacle mappings has been established in this instance. For example, if one were to remove the item $(L_V(1), L_E(1))$ from R_1, then $L_b(1)$ (instead of $L_b(2)$) would be handed over to $L_V(2)$. A more informative mapping requires $L_S \in \{(L_E, \hat{L}_e), L_V, L_b\}$ and $L_T \in \{(L_E, \hat{L}_b), L_V, L_e\}$, so that each hyperedge source or target is associated with an external node of the embedded hypergraph. This reflects the desired extension of the s

and t mappings discussed earlier. Labels that belong to elements of the embedded hypergraph are hatted ($\hat{}$) here for ease of readability. Hypergraphs R_1 and R_2 would hence be represented like this:

R_1: $(L_V(1), (L_E(1), \hat{L}_b(1))), (L_V(2), (L_E(1), \hat{L}_b(2))),$
$((L_E(1), \hat{L}_e(1)), L_V(3)), ((L_E(1), \hat{L}_e(2)), L_V(4))$

R_2: $(L_b(1), L_V(1)), (L_b(2), L_V(1)), (L_V(1), L_e(1)), (L_V(1), L_e(2))$

Now, suppose we have a hypergraph R_0 that only consists of vertices $L_V(1)$, $L_V(2)$, $L_V(3)$, $L_V(4)$ and we want to embed R_2 in this hypergraph. The s and t mappings to and from the begin and end nodes of the embedded hypergraph need to be defined, which, without further knowledge on hand, implies randomising (or otherwise guessing) such a mapping to produce something like the above R_1. However, this is not required if the mapping labels are not part of the host hypergraph representation, but of the embedded hypergraph representation. So, instead of the above, we could have $L_S \in \{L_E, L_V, (\hat{L}_V, L_b), (\hat{L}_b, L_b)\}$ and $L_T \in \{L_E, L_V, (\hat{L}_V, L_e), (\hat{L}_e, L_e)\}$, where hatted labels are those of the host hypergraph, and,

R_1: $(L_V(1), L_E(1)), (L_V(2), L_E(1)), (L_E(1), L_V(3)), (L_E(1), L_V(4))$

R_2: $((\hat{L}_V(1), L_b(1)), L_V(1)), ((\hat{L}_V(2), L_b(2)), L_V(1)),$
$(L_V(1), (\hat{L}_V(3), L_e(1))), (L_V(1), (\hat{L}_V(4), L_e(2))$

R_2 thus defines its own connectivity wherever it is embedded. However, the hypergraph descriptions are not independent anymore, and can only be interpreted as a set. Note that a vertex label in the definition of R_2 refers to either a vertex of R_1 or of R_2, depending on whether it is paired with an external node (i.e. has a hat) or not.

A certain amount of redundancy arises with the node and tentacle labelling that we have described so far. In particular, it is apparent from the definition of R_2 that some vertices of R_1 are connected via hyperedge $L_E(1)$. Consequently, not all of the adjacency pairs are needed for a complete definition of R_1 and R_2. It is possible to simplify the representation so that each hypergraph is defined only by the labels of the vertices and hyperedges of which it is composed, plus the source/target association, so, formally, each hypergraph is a set of $L \in \{L_E, L_V, (\hat{L}_V, L_b), (\hat{L}_b, L_b), (\hat{L}_V, L_e), (\hat{L}_e, L_e)\}$. Consider our two hypergraphs described as follows:

R_1: $L_V(1), L_V(2), L_V(3), L_V(4), L_E(1)$

R_2: $(\hat{L}_V(1), L_b(1)), (\hat{L}_V(2), L_b(2)), (\hat{L}_V(3), L_e(1)), (\hat{L}_V(4), L_e(2)), L_V(1)$

V_1: $\hat{L}_b(1), \hat{L}_b(2), \hat{L}_e(1), \hat{L}_e(2)$

where V_1 is the vertex labelled by $L_V(1)$. As the description of R_1 fails to specify the adjacency between elements of the same hypergraph, R_1 is underdefined, unless R_2, which provides this information, is also defined. R_2 remains underdefined, however, because no edges between external nodes and the vertex are described. The missing information can be specified for each vertex separately, as shown with V_1 above. This constitutes a wrapper for the vertex (see also section 5.3 for details), which is a set of $L \in \{\hat{L}_b, \hat{L}_e\}$. The benefit of this approach is that position dependence is re-established for the incident edges of the vertex, which is e.g. necessary if the vertex represents a noncommutative operation.

The above representation of R_1 or R_2 will be referred to as a *cellular graph* to reflect its atomic nature and the need for multiple cellular graphs to define an actual (hyper-)graph. We represent external node pairs (i.e. $(\hat{L}, L)$ pairs) as triples (s, t, d) called *cellular tentacles*, where $s \in C$ is called a *source label*, $t \in C$ is called a *target label*, and $d \in \{0, 1\}$ is the directionality of a tentacle, either incoming (0, for a $(\hat{L}_S, L_b)$ pair) or outgoing (1, for a $(\hat{L}_T, L_e)$ pair). A cellular graph G can thus be 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 cellular tentacles. G_c constitutes the set of all cellular graphs.

A cellular production is an ordered pair (A, G) with $A \in N$ and $G \in G_c$, as illustrated in figure 4.3. Cellular productions can be used in place of hypergraph productions in a hyperedge replacement system. A cellular production has a simpler data structure and a shorter description length than a hypergraph production with additional labels. The main advantage of the cellular production lies in the simple variation operations that can be used on it, and which will be detailed in section 4.3.1. Cellular productions do not define internal vertex connectivity, which requires additional labels to be associated with the vertices, as aforementioned. We will implement this by explicitly wrapping each vertex into a dedicated cellular production, so that the cellular graph becomes the universal unit of graph construction. Ultimately, many more cellular productions are required to describe a given graph than if one were to use the more complex hypergraph productions, but the expressiveness of the graph rewriting system has not been changed by this.

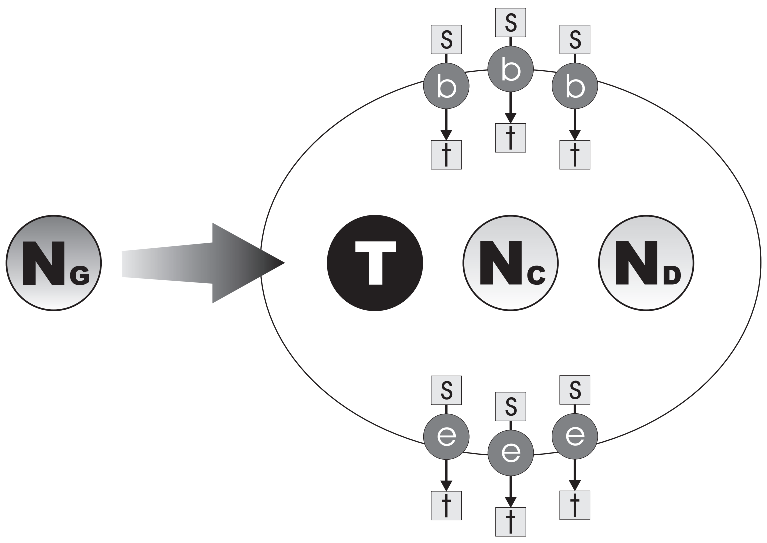


Figure 4.3: Diagrammatic representation of a cellular production. Nonterminal N_G on the left is replaced by a cellular graph, where T is a terminal, N_C and N_D are further nonterminals, b and e are begin and end nodes, and s and t are source labels and target labels of each node.

4.2 Unified Grammar Model

How can a hypergraph grammar be applied to the problem of network design? An obvious approach would be to evolve the network using Grammatical Evolution on a hypergraph grammar. For this, a set of hyperedge productions needs to first be defined from which any network that could be a solution can be constructed. In principle, this is not very challenging and can be done manually; however, the choice of production set decides which networks are easiest to construct, and this cannot be addressed unless one has *a priori* knowledge about what constitutes a fit network. Having a static grammar also precludes us from using it for the sort of model-building discussed in section 3.7.

It is consequently more appealing to have a graph grammar that can also change. As with string grammars there are multiple approaches for accomplishing this, but we will focus here on the simplest one: to evolve the grammar directly. In the previously described instances of L-system evolution (see section 3.5.2), this is interpreted as evolving a grammar for each member of the population. From each grammar a single network can then be derived and tested on the objective function; the concept is visualised in figure 4.4. The grammar is represented in the chromosome as a string that is interpreted as a set of productions, which can be modified by mutation or recombination of the string. If a production makes

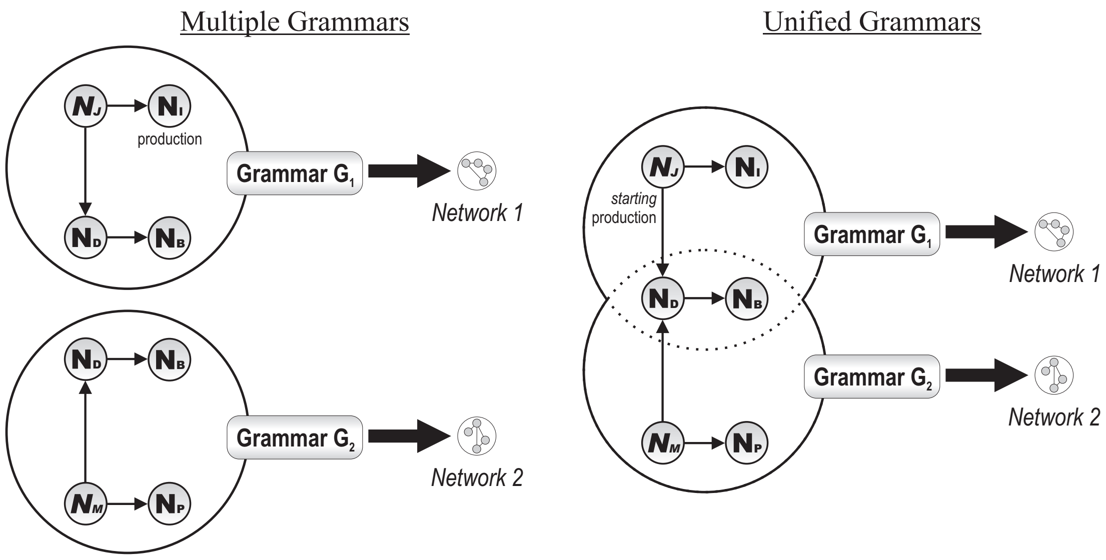


Figure 4.4: Previous studies into using a grammar as a deterministic representation (rather than a stochastic model) involved a population of grammars – one set of productions per solution. The alternative proposed here is to merge these grammars into a deterministic grammar set with unique production labels.

a reference to another production, then that other production must be defined on the same chromosome. Typically, the variation operators are not "aware" of the interpretation of the chromosome as a graph or a grammar, yet the fitness landscape is likely even more rugged than for a conventional GA. For example, if part of the chromosome is exchanged through crossover, then any building block of interacting productions is likely to be destroyed.

Addressing this problem is difficult because the productions form a network of relations that is only observed once the string has been interpreted. The model we propose here is that instead of distributing productions across multiple grammars and chromosomes, we maintain only a single unified production set. Having a single grammar usually implies the availability of choice (i.e. the existence of multiple productions with identical LHSs), but we would also like to use the grammar as a (deterministic) representation, so as to be able to derive any specific network from it as needed. Since the grammar is being evolved, applying a choice list, as with Grammatical Evolution, is not appropriate, because the choices will have evolved, too.

Multiple productions with the same LHS can also exist within a grammar of the aforementioned L-systems, but this is addressed by always choosing the first production in the string that matches. If all productions of all grammars were to be concatenated, however, always choosing the first of a kind would cause many

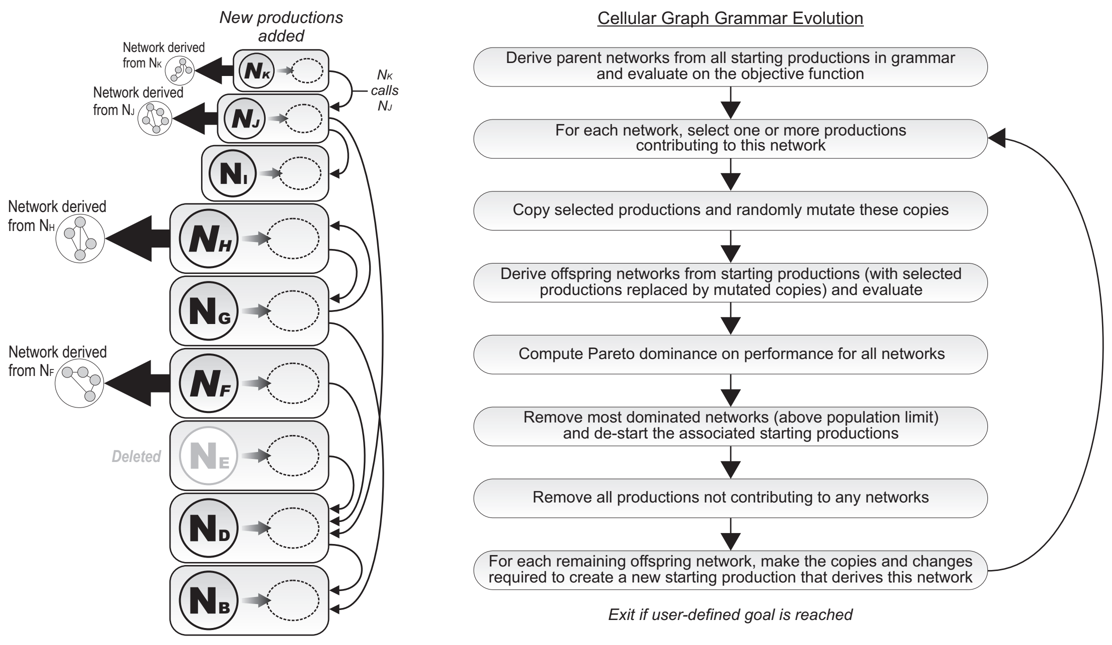

New productions added
Network derived from N_K
Network derived from N_J
Network derived from N_H
Network derived from N_F
Deleted
N_K calls N_J
N_K
N_J
N_I
N_H
N_G
N_F
N_E
N_D
N_B
Cellular Graph Grammar Evolution
Derive parent networks from all starting productions in grammar and evaluate on the objective function
For each network, select one or more productions contributing to this network
Copy selected productions and randomly mutate these copies
Derive offspring networks from starting productions (with selected productions replaced by mutated copies) and evaluate
Compute Pareto dominance on performance for all networks
Remove most dominated networks (above population limit) and de-start the associated starting productions
Remove all productions not contributing to any networks
For each remaining offspring network, make the copies and changes required to create a new starting production that derives this network
Exit if user-defined goal is reached

Figure 4.5: The different steps involved in cellular graph grammar evolution are shown on the right. The algorithm operates on a population of interlinked productions, as shown on the left, where each generation adds new productions and removes unused ones.

networks to become underivable. To restore the information contained by the division into multiple grammars, each production could be labelled with an identifier of the grammar to which it belongs. The concatenation of all productions into a single set would then induce no information loss.

Yet there is no need to have both a production identifier and a grammar identifier as part of the production LHS. Once productions are allowed to call productions from other grammars, this label duality is no more informative for representing solutions than having a single unique label for each production (although it may still be informative in other respects, as will be noted in section 8.3.3). With unique labels assigned to each production it becomes evident that we now have a fully deterministic grammar with no choices, although this also implies that there is only a single network that can be derived. For multiple networks, multiple starting productions are needed. The concatenated production set with multiple starting productions is hence not a grammar in formal terms, but a set of grammars where the LHS of each production of each grammar is unique across the entire set (see figure 4.4).

Each network is associated with a starting production from which it can be derived. So a population of networks is represented as a population of starting productions and other productions that are called by these. Unlike with multiple grammars, only one instance of a particular production has to exist, even if it is involved in the derivation of different networks. This production set shared by all networks is analogous to the gene pool of biological organisms. It is not the genes that an organism holds that define it as much as the genes that it expresses.

Just as an organism is therefore a sample of the biological gene pool, the derived networks are samples of the production set. The total genotypic size of the population may thus be reduced depending on the degree of reuse of productions. If nature is any indication, reuse is prevalent within complex solutions. It is common for individual genes to contribute to multiple phenotypic traits; moreover, humans share at least 31% of their genes with yeast, 50% with the fruit fly, and 99% with the mouse (Pines, 2001). The extensive reuse of proven components allows natural evolution to operate on a much smaller design space than would otherwise be possible – perhaps we can do the same for network design.

4.3 Evolving the Grammar

The grammar evolution system first described by Luerssen and Powers (2003) is specifically targeted at graph grammars and evolves a unified grammar set as described above. 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 network. Productions are neither predefined nor learned from any existing population, but obtained through copying and mutation of existing productions.

Mutations are the only means of change; no recombination (crossover) operator is modelled. The mutation of nonterminals already results in a recombination of networks, which is comparable to subtree-swapping in GP. The mutation operators are described in the next section. Evolution in this framework is viewed as a repeated growing and pruning of the production set. 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

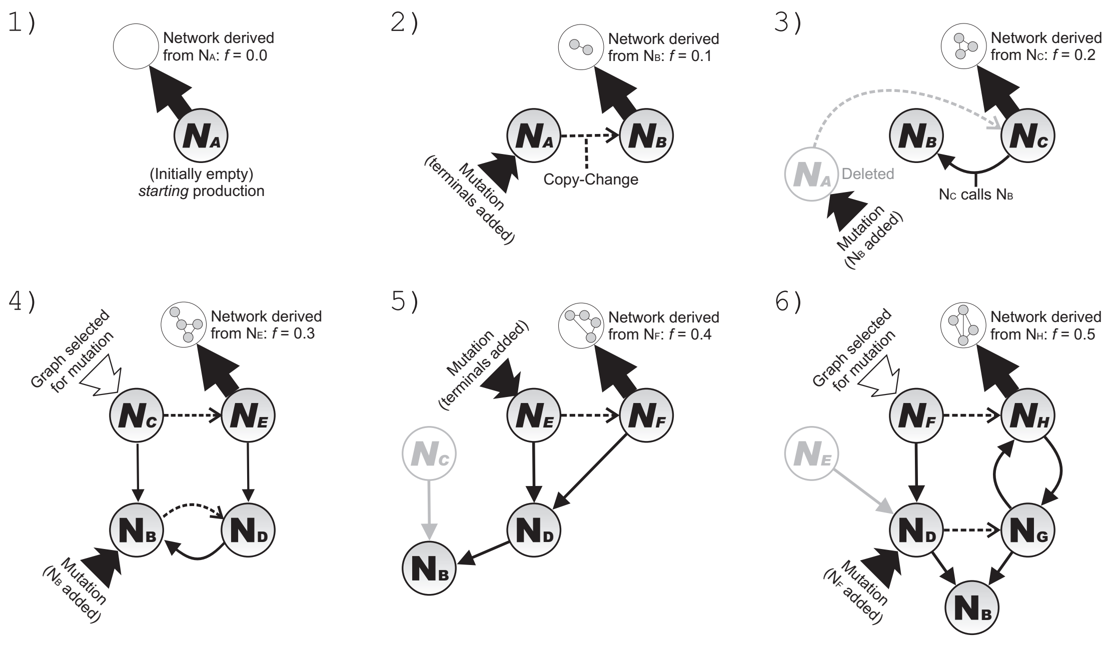

1)
Network derived from NA: f = 0.0
NA
(Initially empty) starting production
2)
Network derived from NB: f = 0.1
NA
NB
Copy-Change
Mutation (terminals added)
3)
Network derived from NC: f = 0.2
NB
NC
NA
Deleted
NC calls NB
Mutation (NB added)
4)
Network derived from NE: f = 0.3
Graph selected for mutation
NC
NE
NB
ND
Mutation (NB added)
5)
Network derived from NF: f = 0.4
Mutation (terminals added)
NE
NF
NC
ND
NB
6)
Network derived from NH: f = 0.5
Graph selected for mutation
NF
NH
NE
ND
NG
NB
Mutation (NF added)

Figure 4.6: Illustration of graph grammar evolution with a maximum population of two graphs. Starting with an empty production/graph N_A in generation (1), terminals are added to a copy N_B of this production in (2), then N_B is added to itself, producing N_C in (3), while the graph of N_A has least fitness f and is thus removed. N_B in the graph of N_C is then mutated in (4), producing N_D and a copy of N_C, N_E, with a reference to N_D. The graph of N_B is now uncompetitive, but remains as a production used by N_C. Further offspring is created in (5) and (6), leading to N_H, which exhibits a recursive self-reference.

mutated set and the existing graph population, are eliminated, as are all productions not involved in any fitter solutions. Conversely, if a mutation survived, the grammar is modified so that the mutated network becomes one of the networks derivable from the grammar. The mutated production is inserted into the grammar; then copies are made of all the network'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. The process is illustrated in figure 4.6. Note that if a production is mutated that is called, directly or indirectly, by many other productions of the same network, then all of these productions must change – i.e. most of the "genome" of the network will be copied. Conversely, changes to a production referenced by no other production, e.g. a non-recursive starting production, require no other changes being necessary, greatly reducing the effort in creating new offspring.

4.3.1 Variation Operators

The mutation operators for graph grammar evolution as presented here are comparatively trivial, as they lack even simple knowledge of graph theoretic concepts. Cellular graphs are changed randomly; for this, the components of a cellular graph are organised into three lists:

- N : a list of nonterminal symbols (hyperedges of the graph)
- T : a list for terminal symbols (wrapped vertices of the graph)
- X : a list of (s, t, d) label triple (begin $(d = 0)$ and end $(d = 1)$ nodes of the graph)

Keeping these components of a cellular graph in separate lists ultimately makes it easier to also have separate mutation probabilities for each component type. Three operations may be applied to each list:

- insert
- remove
- modify

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 operations are implemented as follows.

The ***insert*** operation adds a new element into a random position in the list. For terminal symbols, the new element is randomly selected from the user-defined list of all terminal symbols. For nonterminal symbols, the new element is randomly selected from the evolved population of cellular productions. For label triples, both s and t are each selected randomly from a user-defined label set, with a 50-50 probability of d being 0 or 1.

The ***remove*** operation randomly selects an element from the list and deletes it.

The ***modify*** operation randomly selects an element from the list and changes its value. For terminal and nonterminal symbols, the existing symbol is replaced by a new symbol randomly selected from the available sets. For label pairs, either the s or t member is selected (again at 50-50 probability) and replaced by a random new label; the other members remain as is. The *modify* operation is only used in a select number of experiments in the following chapters, as it is not an essential mutation: the same result may be achieved by a combination of *remove* and *insert* operations.

Since, unlike a hypergraph, a cellular graph defines the attachments of the hyperedge it replaces, the insertion or modification of nonterminal symbols or wrapped terminal symbols requires no additional mutations to establish connectivity between the inserted hyperedge and its host hypergraph (assuming that the inserted graph has any attachments at all). Given a rich set of existing cellular graphs, constructing a specific graph hence becomes as simple as adding nonterminal symbols. On the other hand, obtaining label triples by randomisation fails to ensure that the label triple will match existing labels and therefore properly constitute a hyperedge tentacle in the graph. This issue will be a subject of elaboration in the next chapter.

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 depth is thus adapted with each component separately (see also section 4.4). Note that since this has no effect on either performance or size of the graph unless the production already calls itself, these mutations are typically very neutral (see also section 4.4 on the nature of the recursion limits).

4.3.1.1 Variation Operator Probabilities

None of the suggested operators are adaptive, neither in their probability of being applied nor in the effect they have. We rely solely on population distributions to produce any form of guidance during the search process. That being said we have observed that choosing the right probabilities for a given problem can improve the convergence characteristics, so an adaptive approach would likely reveal benefits as well. The reader is pointed to Hinterding et al. (1997) for a survey of parameter and operator adaptation.

We did not pursue this path, however, because of the need for additional numeric optimisation and also the possibly complex interactions with the existing framework. Because manual fine-tuning would be unrealistic in practice, all operators are instead applied at equal probabilities, except for mutations to external nodes, which are considered separate for begin and end nodes, and the *increase recursion* operator, which is applied with only half probability, so that the incidence of high recursion numbers in the population is limited.

4.3.2 Size Control

In GP terminology, *bloat* refers to the phenomenon that solution trees may grow during evolution more so than necessary for solving the problem (Langdon and Poli, 1997). Several related explanations have been put forward for this. Firstly, programs composed largely of introns – code segments that have no phenotypic effect – are more likely to survive mutation or recombination than programs without introns, as the phenotypic effect of the variation is more likely to be neutral as well (Blickle and Thiele, 1994; McPhee and Miller, 1995). Miller (2001) proposes that bloat is caused by exploration of a solution's neutral variations, most of which are larger than the existing solution. It has also been argued that above a certain size threshold there are exponentially more longer solutions than shorter or equally sized solutions with the same fitness as a currently discovered solution (Langdon and Poli, 2002).

Certainly, if bloat is left unchecked, it is known to grow at quadratic rates (Langdon, 2000a). This puts excessive strain on the available computing resources and thus slows the optimisation process, which is clearly undesirable. The standard method of size control is due to Koza (1992) and involves a static population cardinality and a maximum tree depth for solutions, which, however, puts an absolute limit on GP's ability to explore solutions of greater complexity and thus possibly higher quality. Several alternative methods have therefore been proposed. One simple yet effective approach is to just increase the maximum tree depth when needed to accommodate an individual that is deeper than the maximum but is better than any other individual found during the run (Silva and Almeida, 2003). Langdon (2000b) suggested the use of special, size-aware crossover operators that reduce the growth of solutions trees. Wagner et al. (2004) describe a population control method based on a variable population cardinality, with no limit on solution tree depth, but a global limit on the total number of tree nodes in the population.

In graph grammar evolution, bloat is also an issue of significant concern. Performance is not the only important property of the evolved networks; size is another. A trade-off between network performance and network size is to be expected, with larger networks performing better and smaller networks requiring less evaluation time and space. Large networks that perform poorly are clearly not desirable, and hence should be selected against. But what makes a network large? As a measure of size, we may simply count the number of edges and nodes of the network. This is formally appropriate, but fails to consider the genetic representation of networks as cellular productions. A cellular production need not directly contribute to the number of nodes and edges of a network; it may simply refer to other productions. Consequently, a small network may result from a large number of cellular productions, which therefore needs to be restricted to avoid a potential production bloat.

Conversely, measuring network size as the number of cellular productions involved in a network might lead to very large networks, as cellular productions can be called recursively. We therefore define size as the sum of the terminals, nonterminals, and external nodes (i.e. label triples) of each cellular production expressed during the derivation of a network, including cellular productions expressed before. The trade-off between performance and size will be balanced by using a multi-objective evolutionary algorithm for grammar optimisation. This approach has previously been followed successfully in a variety of related domains, including tree size control in GP (Bleuer et al., 2001; De Jong and Pollack, 2003) and evolution of a hidden layer in a neural network (Abbass, 2003).

4.3.3 Multi-objective Optimisation

The presence of multiple objectives in a problem frequently results in not just a single optimal solution, but an entire set of so-called Pareto-optimal solutions. Multi-objective evolutionary algorithms (MOEAs) can be employed to find this set of Pareto-optimal solutions. The majority of recently published MOEAs use fitness assignment based on Pareto-domination (Deb, 2001). A solution S_1 is said to dominate another solution S_2 if S_1 is no worse than S_2 in all objectives and better than S_2 in at least one objective. If a solution is not dominated, then it is potentially a member of the Pareto-optimal set, from which the user can ultimately pick the most appropriate compromise between the objectives. Note, however, that even though the MOEA takes multiple objectives into account simultaneously, it must still transform all of these objectives into one fitness measure, so that the evolutionary algorithm can distinguish fit individuals from less fit ones. The transformation is typically made by assigning each solution a measure of its nondominatedness.

If the size of the population is smaller than the size of the Pareto-optimal set, then the MOEA is meant to return a set of nondominated solutions that

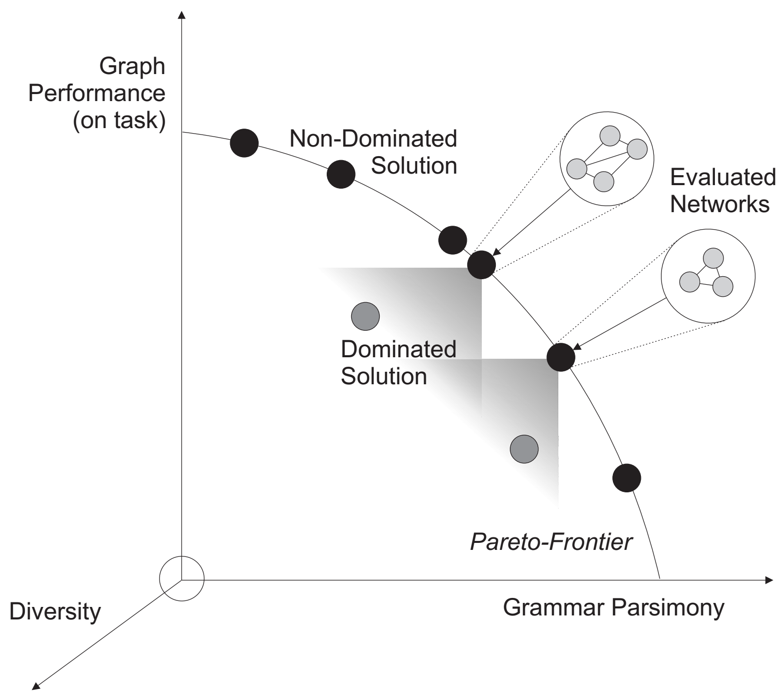


Figure 4.7: Schematic depiction of a Pareto frontier for the various objectives optimised in this study. (Diversity will be covered in chapter 6.)

are spread evenly along the Pareto boundary. Most MOEAs apply some form of phenotypic *niching* to achieve this, which means that the spread is based on the objective function values and not on structural differences within the solutions themselves. Niching is used only as a secondary measure of fitness: If individual S_1 is more nondominated than S_2, S_1 is preferred regardless of niching, whereas if S_1 and S_2 have the same degree of nondominatedness, the one residing in the most sparsely populated region of the search-space is preferred. In our multi-objective implementation of graph grammar evolution, we assess population density as simply the distance between a chosen solution and its nearest neighbour, with a bias towards the lowest error solution in case of a tie (or the newest solution, if this fails). Otherwise the implementation of a MOEA for this project matches the NSGA-II presented by Deb et al. (2000).

4.4 Evaluating the Network

See figure 4.8 for an illustration of how a network is constructed from a set of cellular productions. The graph rewriting is performed in parallel. Although for a context-free grammar this has no effect on the shape of the generated network,

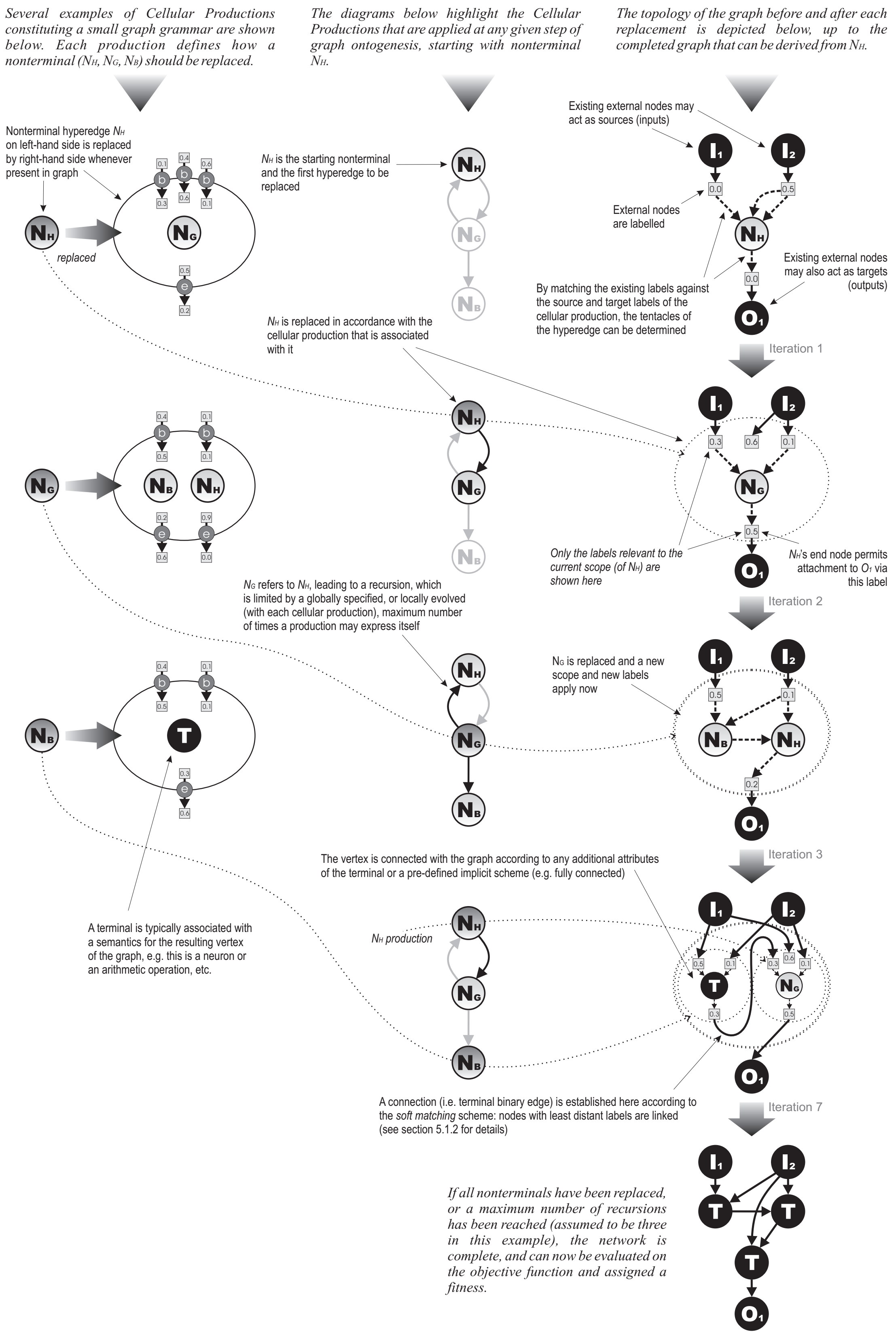

Several examples of Cellular Productions constituting a small graph grammar are shown below. Each production defines how a nonterminal (NH, NG, NB) should be replaced.
The diagrams below highlight the Cellular Productions that are applied at any given step of graph ontogenesis, starting with nonterminal NH.
The topology of the graph before and after each replacement is depicted below, up to the completed graph that can be derived from NH.
Nonterminal hyperedge NH on left-hand side is replaced by right-hand side whenever present in graph
replaced
NH is the starting nonterminal and the first hyperedge to be replaced
Existing external nodes may act as sources (inputs)
External nodes are labelled
By matching the existing labels against the source and target labels of the cellular production, the tentacles of the hyperedge can be determined
Existing external nodes may also act as targets (outputs)
NH is replaced in accordance with the cellular production that is associated with it
Iteration 1
Only the labels relevant to the current scope (of NH) are shown here
NH's end node permits attachment to O1 via this label
NG refers to NH, leading to a recursion, which is limited by a globally specified, or locally evolved (with each cellular production), maximum number of times a production may express itself
Iteration 2
NG is replaced and a new scope and new labels apply now
Iteration 3
The vertex is connected with the graph according to any additional attributes of the terminal or a pre-defined implicit scheme (e.g. fully connected)
A terminal is typically associated with a semantics for the resulting vertex of the graph, e.g. this is a neuron or an arithmetic operation, etc.
NH production
A connection (i.e. terminal binary edge) is established here according to the soft matching scheme: nodes with least distant labels are linked (see section 5.1.2 for details)
Iteration 7
If all nonterminals have been replaced, or a maximum number of recursions has been reached (assumed to be three in this example), the network is complete, and can now be evaluated on the objective function and assigned a fitness.

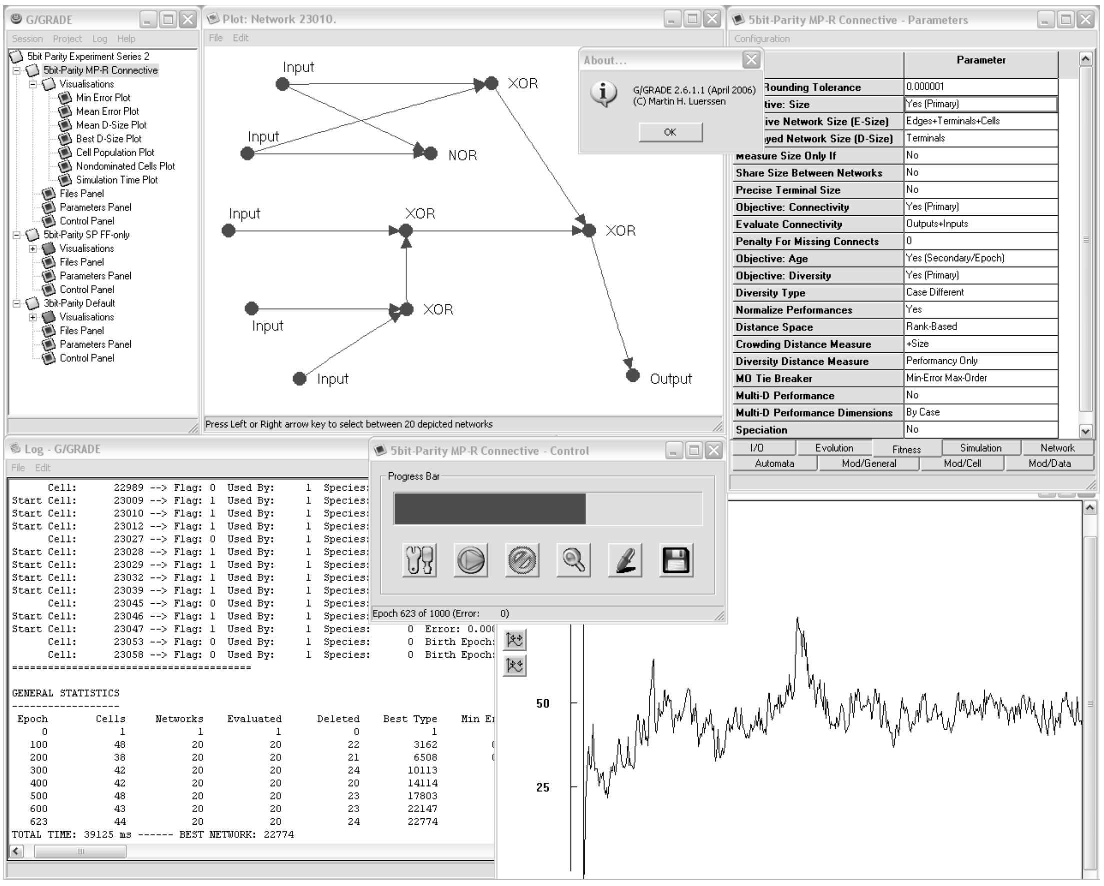
Figure 4.8: Example derivation of a network from a cellular graph grammar; the right column depicts the network at various stages of construction.

it allows us to express the graph rewriting as a distributed developmental process. For this purpose, let us define a developmental unit as a system $U = (P, H, N, U_0)$, where P is the set of all productions, H is an existing hypergraph, N is the label of a hyperedge to be replaced, and U_0 is an associated ancestor developmental unit (can be empty). Starting from a single U_1, a graph can be generated as follows. U_1 retrieves a production p matching label N from set P and replaces the occurrence of hyperedge e in H with the RHS of the production p. For each occurrence of a hyperedge e in the inserted subhypergraph a new $U_e = (P, H, N_e, U_1)$ is generated and the replacement process repeated in parallel for each new U_e, until either no more hyperedges require replacement or the number of previous instances of a production in the replacement path exceeds a parameter $m \in \Im^+$ (comparable to the life of the cell, as with Cellular Encoding, or a condition, as in PL-systems) that is co-evolved with each production. Thus, each production's recursion depth is limited individually rather than globally.

From the graph rewriting process we obtain a hypergraph H as well as a set of U that are interlinked into a (development) tree. This U tree can be discarded if H is complete; however, the initial H_0 of U_1 may change for a new fitness case (see below), in which case the final H_n is incomplete or false. The U tree can be used to modify H_n according to the changes in H_0, instead of having to fully re-grow H_n. The generated graph becomes an automata network by adding the appropriate semantics to the nodes – for a neural network these become threshold automata, for example.

The developed network is usually evaluated on some optimisation problem. Unless a network has been precluded from having recurrent connectivity (see section 5.3.1 on how this is achieved), it cannot be evaluated layer by layer as with a perceptron. Instead, each automaton is simulated in parallel for several cycles until a user-defined cycle limit is reached. We achieve parallelism by splitting the automaton update into two phases. First, each automaton updates its hidden state based on the visible state of all the neurons to which it is connected. Subsequently, each automaton turns its hidden state into a visible state and continues with step one.

If the problem requiring optimisation is a typical pattern/label-classification task, the network needs to be linked to the appropriate sources (inputs) and targets (outputs). For each possible source and target defined by a particular pattern/label pair (or time series thereof), an automaton is spawned whose state matches that of the source or target for each cycle. A vertex set of automata representing a pattern/label pair constitutes an initial H_0, to which every network



Press Left or Right arrow key to select between 20 depicted networks

Epoch 623 of 1000 (Error: 0)

```
      Cell:      22989 --> Flag: 0  Used By:     1  Species:
Start Cell:      23009 --> Flag: 1  Used By:     1  Species:
Start Cell:      23010 --> Flag: 1  Used By:     1  Species:
Start Cell:      23012 --> Flag: 1  Used By:     1  Species:
      Cell:      23027 --> Flag: 0  Used By:     1  Species:
Start Cell:      23028 --> Flag: 1  Used By:     1  Species:
Start Cell:      23029 --> Flag: 1  Used By:     1  Species:
Start Cell:      23032 --> Flag: 1  Used By:     1  Species:
Start Cell:      23039 --> Flag: 1  Used By:     1  Species:
      Cell:      23045 --> Flag: 0  Used By:     1  Species:
Start Cell:      23046 --> Flag: 1  Used By:     1  Species:
Start Cell:      23047 --> Flag: 1  Used By:     1  Species:      0  Error: 0.000
      Cell:      23053 --> Flag: 0  Used By:     1  Species:      0  Birth Epoch:
      Cell:      23058 --> Flag: 0  Used By:     1  Species:      0  Birth Epoch:
========================================

GENERAL STATISTICS
------------------
 Epoch        Cells     Networks    Evaluated     Deleted    Best Type    Min E
     0            1            1            1           0            1
   100           48           20           20          22         3162         (
   200           38           20           20          21         6508         (
   300           42           20           20          24        10113
   400           42           20           20          20        14114
   500           48           20           20          23        17803
   600           43           20           20          23        22147
   623           44           20           20          24        22774
TOTAL TIME: 39125 ms ------ BEST NETWORK: 22774
```

	Parameter
Rounding Tolerance	0.000001
tive: Size	Yes (Primary)
ive Network Size (E-Size)	Edges+Terminals+Cells
yed Network Size (D-Size)	Terminals
Measure Size Only If	No
Share Size Between Networks	No
Precise Terminal Size	No
Objective: Connectivity	Yes (Primary)
Evaluate Connectivity	Outputs+Inputs
Penalty For Missing Connects	0
Objective: Age	Yes (Secondary/Epoch)
Objective: Diversity	Yes (Primary)
Diversity Type	Case Different
Normalize Performances	Yes
Distance Space	Rank-Based
Crowding Distance Measure	+Size
Diversity Distance Measure	Performancy Only
MO Tie Breaker	Min-Error Max-Order
Multi-D Performance	No
Multi-D Performance Dimensions	By Case
Speciation	No

Figure 4.9: The G/GRADE software suite.

in the evolved population will link according to the source and target mappings of their respective starting production. States retrieved by the target automata after several update cycles automata are compared to the expected outputs and an error for the network may thus be computed. Once all networks are evaluated on this pattern/label pair, they are connected to another H_0 matching the next pair and the evaluation continues. Unless otherwise specified, the order in which the pattern/label pairs are presented is random.

4.5 The G/GRADE System

We shall refer to the system of graph grammar evolution described above as G/GRADE (for *G*raph *Gra*mmar *D*esign by *E*volution). G/GRADE is also the name of a software suite that implements this system and several simulators for the applications described in the next section. The G/GRADE software was coded in C++ and allows multiple projects, involving different problem domains, input and output files, and parameters, to be managed concurrently. It generates and visualises various statistics of the optimisation run, and can also illustrate the networks that are discovered through this process. Any ongoing optimisation run can be interrupted, recorded to storage, and continued with different parameters. G/GRADE will be employed in all of the experiments described in chapters 5 to 7, which were run on a single PC with Windows XP, Athlon 64 3000+ CPU, and 1GB of RAM. The interested reader may obtain executables or source code from the author. Figure 4.9 displays a screenshot of the version of G/GRADE that was current at the time of writing.

4.6 Applications

G/GRADE is only useful if it can solve problems of practical relevance. Fortunately, network design is essential to a wide range of problem tasks. The subsequent chapters will employ several problem tasks as testbeds for evaluating the system presented here and any possible extensions or variations to it. For this purpose the chosen problem tasks must be quick to simulate, so that statistically meaningful results are achievable within a realistic time frame. On the other hand, the tasks should also provide a challenge to the optimisation algorithm, as well as highlight notable features of the system, and preferably be used in other research to allow for easy comparison. With this in mind we decided on a mixed set of tasks, which are described below. Illustrations of some of the networks evolved by G/GRADE to solve these problem tasks are given in figures 4.11 and 4.13.

4.6.1 Symbolic Regression

Regression is about inferring a functional mapping $y = f(x)$ between a set of independent variables x and a dependent variable y. Regression by neural networks assumes an auxillary transfer function g (such as the sigmoid), so that $f(x) = W_O \cdot g(W_H)$, where W_O are the weights from hidden to output layer, and W_H are the weights from input to hidden layer. Since multilayer neural networks of sufficient complexity can approximate any mapping, the problem of regression primarily becomes that of optimising weights within the context of a specific model. In contrast, symbolic regression is about generating answers directly in the symbolic language of mathematics (Koza, 1992). This can be advantageous if the black box nature of neural networks is not acceptable, as in some engineering applications or when scientific understanding is required.

Symbolic regression is commonly used in theoretical studies of GP. GP can directly construct the actual mapping function $f(x)$, as is typical for most test problems, or a mapping function $f_1(x)$ that best approximates the actual $f(x)$. The mapping function is represented by a tree of functions selected from a set of pre-specified low-level elementary functions. Applying graph grammar evolution to symbolic regression produces graphs of elementary functions; this may include any tree that GP could discover, but cycles in the graphs may also represent subfunction reuse, or recurrence equations. The functions being regressed will be described in the method sections of the respective experiments.

4.6.2 Neural Networks

The evolution of artificial neural networks is an extensive field of study, and given the many practical uses of neural networks, the range of possible applications for neuro-evolution is virtually boundless. Two tasks are therefore chosen to reflect common challenges in neuro-evolution. The first involves the evolutionary design of a simple multilayer perceptron (MLP), with weights being concurrently determined by backpropagation. The MLP is intended to classify the well-known Fisher Iris dataset (Fisher, 1936). Finding a multilayer architecture is essential for good performance here. Although the design task is quite simple, the evolutionary algorithm has to interact with a rather fickle learning process. Details on this will again be provided later in the experimental methods.

The second task requires the evolution of architecture as well as weights of a neural network on a common reinforcement learning problem, the balancing of two poles fixed to a cart moving on a finite track (Wieland, 1991). The neural

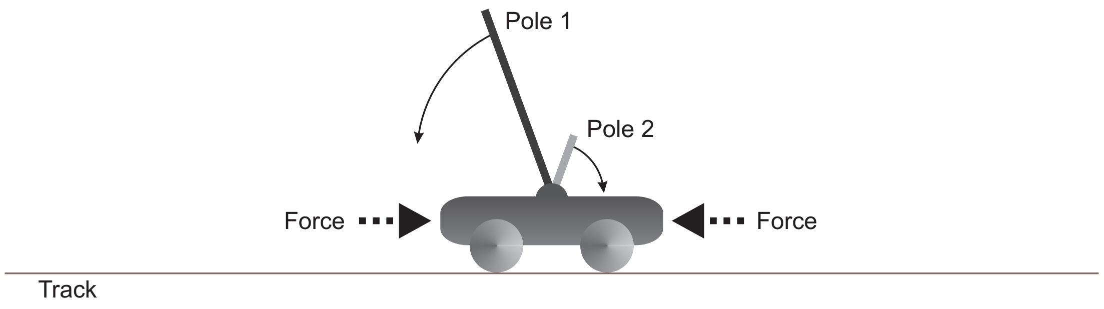


Figure 4.10: The pole balancing task.

network acts as a controller for keeping the poles balanced by applying a force to the cart to push it to the left or right (see figure 4.10). This is viable because the poles have different lengths and therefore respond differently to accelerations of the cart. Pole balancing is a common test problem in neuro-evolution; some results from other studies will be provided in section 7.7.

The input to the neural network is the system state defined by the cart position and velocity, and the position and angular velocity of each pole. Our physics model matches that of Stanley and Miikkulainen (2002). A Runge-Kutta fourth-order method is used to implement the dynamics of the system, with a step size of 0.01s. All state variables are scaled to $[-1, 1]$ before being fed to the network, which outputs a variable force to the cart. The initial position of the long pole is 4.5° and the short pole is upright; the track is 4.8m long, and poles are only regarded as balanced if between $-36°$ and 36° from vertical. The fitness of a neural network is given by the number of time steps that both poles remain balanced without the cart exceeding the track.

4.6.3 Circuit Design

Automation has been applied to circuit design for many years now, as circuit diagrams drawn up by humans are mapped into actual layouts through the application of simple minimization, placement and routing rules. Evolvable hardware promises to take this much further by either eliminating or at least lessening the need for a designer and thus also diminishing the production costs (Gordon and Bentley, 2006). However, modern circuit designs are large and may involve millions of transistors. Evolving circuits is therefore only practical if we can successfully address the problem of scalability.

To accomplish this, the idea of employing a developmental process has also been raised in this field. Several of the embryogenic approaches that we previously reviewed have been applied to the problem. Koza et al. (1999) modified Cellular Encoding for use on circuit design and other engineering problems. Haddow et al. (2001) evolved L-systems that grow into a two-dimensional grid of configurable logic blocks. Gordon and Bentley (2005) likewise employed a grid of cells, where each cell contains evolved rules that produce virtual proteins; the resulting pattern of the cells is then mapped into a circuit design. Miller and Thomson (2003) also explored a growth-based system for circuit design, but each cell in this system contains a circuit evolved with Cartesian GP that maps input conditions to output conditions of the cell from which the main circuit is grown.

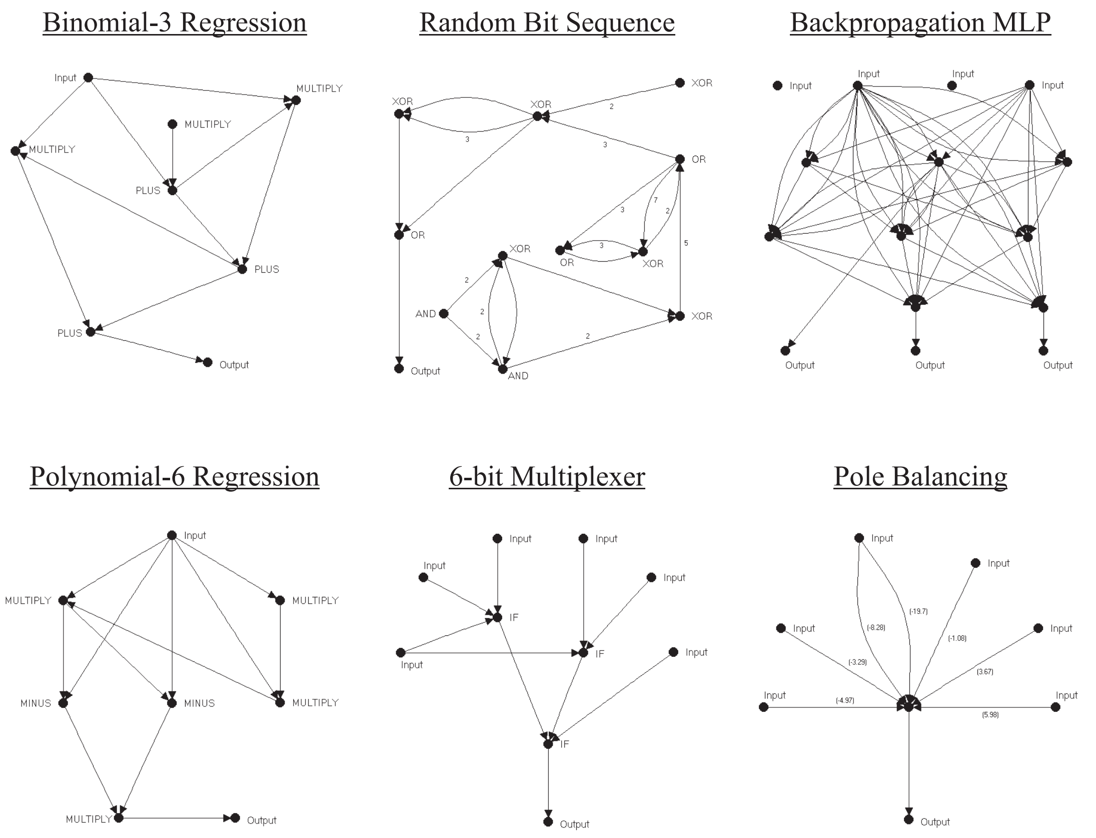


Figure 4.11: A sample of networks evolved by G/GRADE on 6 evaluated problem tasks. Line delays are shown for the RBS circuit. Note that while all of the depicted networks are performance-optimal, we did not verify that they are the smallest possible optimal networks as well.

Circuit design is a domain that can easily be addressed by the G/GRADE framework, although actual hardware evolution was not intended as the emphasis of this research. We have therefore selected two tasks requiring Boolean network design as representative instances of this domain. The first task concerns the design of a multiplexer (Koza, 1992). Multiplexers are logic circuits that are frequently used in communication and input/output operations for transmitting a number of separated signals simultaneously over a single channel. The 6-bit Boolean multiplexer problem involves decoding a 2-bit binary address $(00, 01, 10, 11)$ and returning the value of the corresponding data register $(d0, d1, d2, d3)$. Thus, the multiplexer has 6 inputs here: two to determine the address, and four to determine the answer. We implement the multiplexer as a network of the Boolean operators AND, OR, NOT, and IF that is optimised by evolution.

A more challenging design problem that involves recurrent circuitry is the design of a random number generator (RNG). Specifically, we are aiming at a 4-bit de Bruijn Counter, which is a single bit RNG based on a linear feedback shift register (LFSR) (Chu and Jones, 1999). The LFSR is mathematically represented by the recurrence equation:

$$x_n = a_1 \bullet x_{n-1} \oplus a_2 \bullet x_{n-2} \oplus \ldots \oplus a_m \bullet x_{n-m}, \tag{4.1}$$

where x_i is the i^{th} number generated, a_i is a pre-determined binary constant, and $\bullet$ and $\oplus$ are AND operator and XOR (exclusive-OR) operator respectively. Through this sequence of AND and XOR operators a new number can be generated from m previous values $(x_{n-1}, x_{n-2}, ..., x_{n-m})$. The maximum period achieved by an LFSR is $2^m - 1$, but this requires a special set of a_is to be used. For instance, when m is 4, the optimal set of a_is describes the simple equation

$$x_n = x_{n-1} \oplus x_{n-4}. \tag{4.2}$$

Assuming an initial seed (i.e. x1, x2, x3, x4) of 1000, the random sequence obtained from this equation is a pattern repeating itself every 15 bits:

100011110101100 100011110101100 . . .

LFSRs generate only zeroes if the initial seed is all-zero. We can address this by adding a *de Bruijn Counter*, which consists of an $m - 1$ input AND gate and an extra XOR gate. For a seed of 0000, the random sequence of a 4-bit de Bruijn Counter is

1111010110010000 1111010110010000 . . .

which repeats every 16 numbers, as de Bruijn Counters have a $2m$ period. The specific problem that will be presented to G/GRADE is to find a circuit with the above gates that produces the previous sequence with an initial seed of 0000. We will refer to this as the *Random Bit Sequence* (RBS) circuit.

4.6.4 Telecommunications

Satisfying the ever-increasing demands for bandwidth and coverage in a very competitive marketplace means that there is little room for error when building expensive telecommunication networks. This has prompted researchers to develop

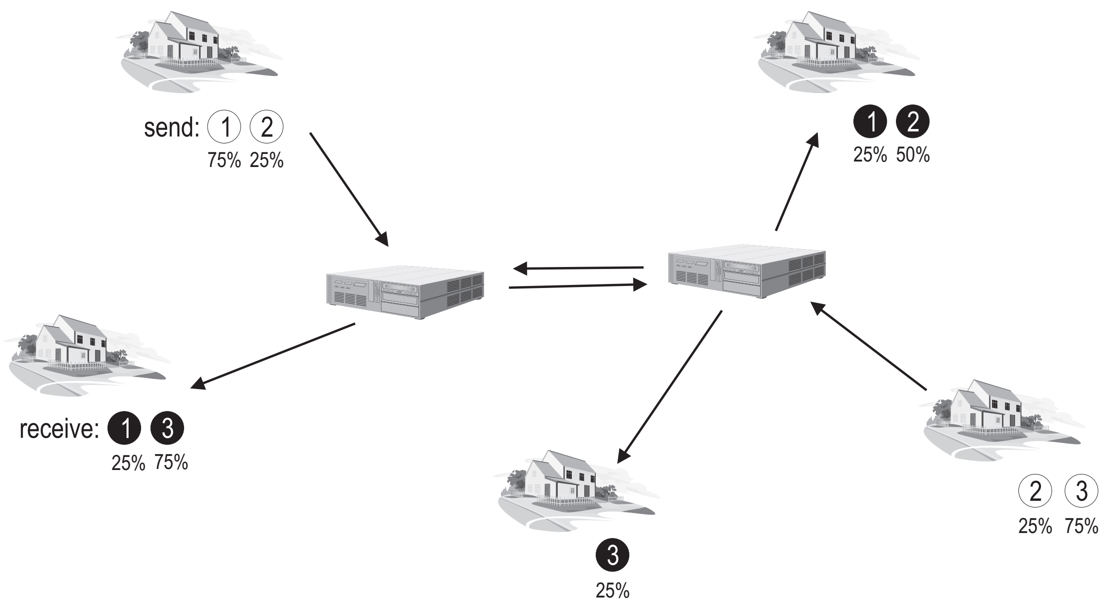


Figure 4.12: The data streaming computer network: switches, depicted here as computers, must connect units (houses) via cables so that data streams 1, 2, and 3 are accessible to all receiving units at the indicated bandwidth fraction (not all units have maximum bandwidth).

new methodologies for finding optimal networks, which also includes the application of evolutionary algorithms. Network design problems can be classified under four inter-related categories: network topology design, network routing and flow control, network performance, and network reliability.

The most obvious avenue to explore with G/GRADE is, of course, network topology design. Much of the previous research concerns minimum spanning trees between a set of sites, as this defines a network with a minimum of wiring cost. Michalewicz (1991) evolved minimum spanning tree topologies for computer networks using a two-dimensional binary adjacency matrix representation and problem-specific mutation and crossover operators. Kumar et al. (1993) optimised networks towards minimum network diameter, minimum average hop count, and maximum reliability using a single-objective GA operating on bit-strings, but with a network-aware crossover. A simple, string-based GA is also used by Dengiz et al. (1995) to determine a minimum-cost topology subject to a reliability constraint. Tanaka and Berlage (1996) addressed the design of video-on-demand distribution networks, not only in terms of topology, but also storage locations; the GA again uses a binary matrix as representation. Sinclair (1995) has focused on multi-wavelength all-optical transport networks, applying first a simple bit-string GA, but later also hybrid GA's (Sinclair, 1997) and GP (Aiyarak et al., 1997) to mesh network design.

Notable among all these studies is the use of a weighting model to obtain a fitness measure from multiple objectives. Use of a MOEA is still uncommon, although Flores et al. (2003) achieved good results with a multi-objective GA on the problem of optimising a fibre optic network between universities. However, simple GAs are still frequently found even in more recent research into network topology design (Berryman et al., 2004; Tsenov, 2005), and developmental methods for addressing the scalability problem are visibly absent. Thus, a useful application for G/GRADE is to optimise a communications network. Although there is much diversity and few standard problems in network topology design, the general problem is that of allocating customers (clients) to suppliers (servers) through links and additional nodes. The problem defined below combines a site generation problem, where nodes can be placed freely in a continuous space, with network topology design.

4.6.4.1 Data Streaming Computer Network

For the purpose of evaluating G/GRADE, a problem is needed that captures the essence of the design challenge inherent in the above examples. The suggested task is depicted in figure 4.12 and involves data streaming between multiple sources and sinks; we will refer to them simply as *units*. These units are located in a bounded two-dimensional landscape of size one-by-one kilometres. Any unit may send and/or may desire to receive one or more specific data streams. Each data stream has a bandwidth requirement B that needs to be satisfied between the unit that sends and the one that receives; GB/s will be used as a unit of bandwidth here. The units and their data streams are predefined in the problem. It is the responsibility of the evolutionary system to optimise the satisfaction of bandwidth by placing *switches* into the landscape. A switch has one or more one-directional lines (cables) that connect to a unit each. The unit can request a desired data stream from the connected switch; this request is passed on to other connected switches and units from which the switch can receive streams. Any unit or switch that receives a request for a stream to which it has access will pass this stream on.

There are, however, several hardware constraints. A unit or switch cannot send or receive more than 1 GB/s. If it receives more than 1 GB/s the incoming streams are equally scaled to a total of 1 GB/s (i.e. packets are lost). If it receives requests for streams, the bandwidth of all streams that are passed on are scaled

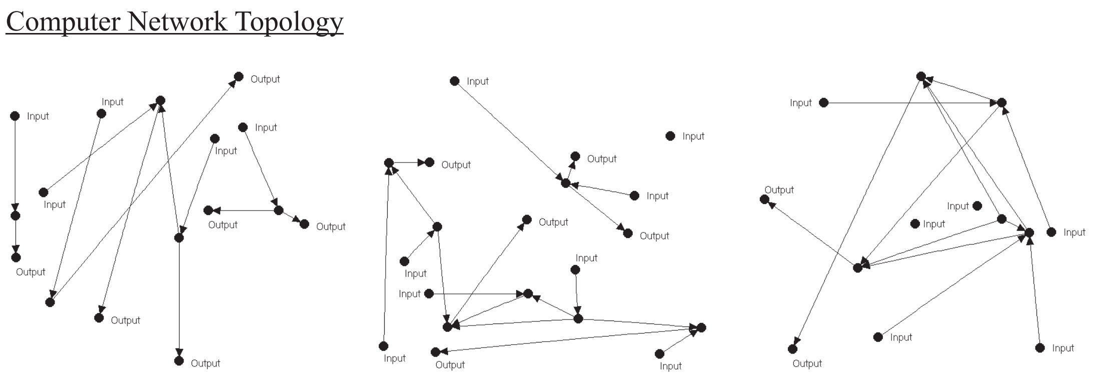


Figure 4.13: A sample of networks evolved by G/GRADE on 3 of the 10 randomly designed computer network topology problems.

so that the unit or switch does not send more than 1 GB/s. Another constraint is that each unit can only be connected to a single switch, although switches can be connected to several other switches (with a maximum of five incoming connections). Finally, each switch and cable in the network has a monetary cost m associated with it. A switch costs $m_s = 1000$ and a cable costs $m_c = 1000$ times each kilometre laid (or fractions thereof). The goal is to minimize the cost $C =$ (number of switches) $\times\, m_s + \sum m_c \times$ (length of cable), while also minimizing the amount of bandwidth absent from the desired data streams:

$$B_d = \sum_{i=0}^{n} (\begin{cases} 0 & \text{if } d_i = 0 \\ 0 & \text{if } o_i \geq d_i \\ 1 - \frac{o_i}{d_i} & \text{if } d_i > 0, o_i < d_i \end{cases}),$$

where n is the number of units, o is the bandwidth of the arriving data stream and d is the desired bandwidth of the unit. G/GRADE will be tested on 10 different unit/data-stream configurations, randomly generated under the condition that desired bandwidth never exceeds provided bandwidth. Figure 4.13 is an indication of the kinds of networks that need to be evolved here.

4.7 Summary

Our contribution with this chapter is to have transformed hypergraph grammars into cellular graph grammars that are amenable to evolutionary exploration using simple variation operators. We also introduce a novel unified grammar model for evolving a population of networks as a grammar. As with other grammar-model building schemes, changes to the grammar lead to new networks, yet the grammar also deterministically encodes every network being evolved, which can simply be derived as needed. Since there is consequently no distinction between the grammar of one network and the grammar of another, recombination operators become superfluous. The resulting system constitutes a powerful foundation for optimising network design, but not every aspect of it has been defined yet. The following chapters will address the outstanding questions and empirically determine some answers using the application tasks that have been presented here.

Chapter 5

Methods of Graph Construction

Deriving a network from a cellular graph grammar entails the expression of a sequence of cellular productions that will ultimately result in a set of vertices and (binary) edges connecting these vertices, i.e. the solution network. Each cellular production defines a cellular graph, which has set of begin and end nodes, each in turn defined by a pair of source and target labels. Fusion between begin and end nodes is established by finding labels that match other labels. Clearly, there is some freedom in the interpretation of what it means to match a label and what the exact consequences are. This chapter addresses four major issues in this context: the nature of labels and a suitable definition of a label match; the scope of the label lookup; possibilities for simplifying the model; and the nature of the graphs evolved using this model.

5.1 Graph Gluing Models

Our fundamental assumption for label matching is that we are looking for target labels that match a particular source label, not vice versa. The node associated with a target may be adjacent to multiple nodes of matching sources, but the node of a source may only be adjacent to a single node of a matching target (see 5.1). The reason for this is that begin and end nodes are never actually manifested as actual vertices of the graph, nor are the edges between them; they are only placeholders for the internal vertices defined by the terminals. So when we pick any particular begin or end node, it is possible to trace it back to one (and only one) specific terminal vertex of the graph. If, alternatively, each begin or end node were able to represent multiple terminal vertices, we would lose the

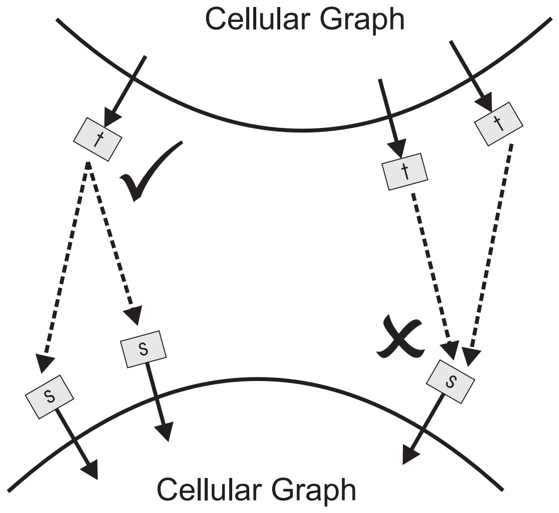


Figure 5.1: A node with a given source label is assigned to a node with a target label that matches the source label. During graph construction, the former node is replaced by the latter; we do not permit a single node to be assigned to, and thus replaced by, multiple nodes.

ability to distinguish these vertices, but this is very important if the network defines operations that are noncommutative or require a fixed number of inputs.

5.1.1 Strict Matching

The key to gluing the nodes of various cellular graphs into a cohesive network lies in the nature of the label matching. The most apparent interpretation of the term “match” is to mean that both labels are identical – we will refer to this definition from now on as *strict matching*. It is a sensible approach if you want to manually design a graph, but, once again, the random modifications required by evolutionary search can make this problematic. Assume the simple case of adding a begin node to a production. For the added node to make any difference, the source of the begin node needs to match the target of a node in its scope. Two exceptions might occur here: either A) a matching target is not found, or B) multiple targets with the same label match the source. In case A), the addition of the begin node has no effect: no edge is created. In case B), multiple different edges are possible – but which one should we choose? Since only one edge is permitted for the begin node, this edge is therefore the only edge that will be created for this particular cellular graph. All other edges are essentially unavailable to the search process, because every choice we make within our gluing models must be deterministic, so that a particular network always arises from the same starting production.

Now, consider that the binomial probability mass function is

$$b(x, n, p) = \begin{pmatrix} n \\ x \end{pmatrix} p^x (1-p)^{(n-x)}. \tag{5.1}$$

The probability of a target matching a specific source (i.e. the probability of an edge being created from all the possible edges that could be created) is the probability of the label of the source occurring once in the target set plus the probability of the label of the source occurring multiple times in the target set multiplied by the probability that the target that is selected is the one that is wanted, i.e.

$$P_{create} = \sum_{y=1}^{g} b(y, g, \frac{1}{a}) \times \frac{1}{y}, \tag{5.2}$$

where g is the number of targets and a the number of possible labels (selected from a global set). P_{create} is shown against g and a in figure 5.2. This plot reveals not only that there is a generally rather slim chance that a particular target is picked, but also that there should be a correspondence between the number of targets in the scope and the number of labels in the global set. While a larger 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. Estimating the appropriate size of the global label set is difficult because target numbers vary depending on the nature of the problem and individual cellular graphs.

During the evolution of cellular graphs, a source label that is guaranteed to match a target can be obtained by selecting from one of the targets within the host graph. This requires a lookup of perhaps multiple cellular graphs that are within this scope, a rather complex operation for what would constitute just part of a single mutation step. If targets are also added using this principle, then no new labels can ever arise within a scope – and hence no new edges. Selecting labels randomly is thus at least occasionally absolutely necessary. A different approach to edge creation is to modify both the production harbouring the source as well as the production harbouring the target, but since with each begin or end node that is added a new source and a new target are needed, a cascade of label modifications may suddenly be required. We therefore avoid this option.

In the presence of multiple targets with identical labels, labels alone are not

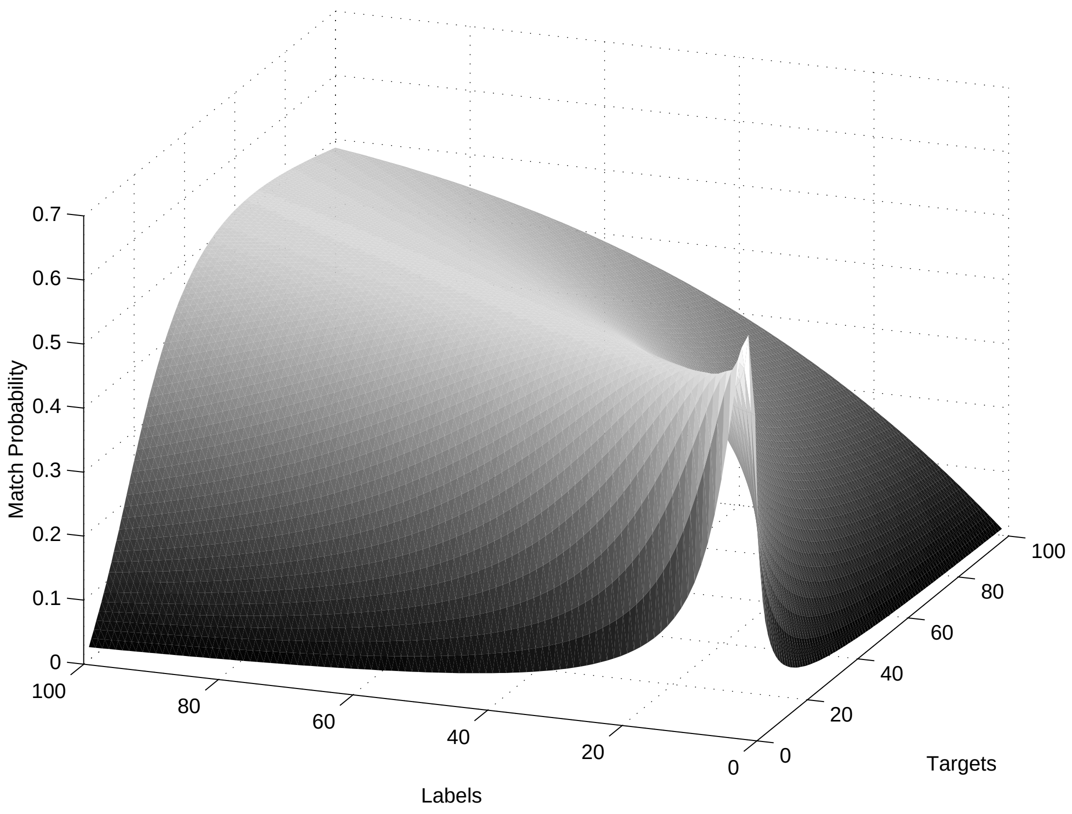


Figure 5.2: The probability of a successful strict match against the size of the label set and the number of potential targets in the current scope.

sufficient for choosing a target. With random labels the probability of duplicate labels is

$$P_{dup} = 1 - b(1, g, \frac{1}{a}), \tag{5.3}$$

where b is the binomial probability mass function, g is the number of targets and a the number of possible labels. With label selection from scope for both sources and targets a becomes the number of different targets or sources, respectively, which is in all cases less than or equal to the number of possible labels, so the probability of duplicates is further increased. In certain problem domains, it may be possible to choose between – or combine – these duplicates (e.g. by applying an OR operation), but this cannot be generalised. How do we choose between identically-labelled targets in a domain-independent manner? An additional bias is necessary, which should give a preference of one target over another – yet this was the main purpose of having labels in the first place.

The most apparent solution involves an implementation-based ordering, or an additional, unique (e.g. “age”) label, so that one can simply choose the first (“oldest”) target that matches the label. Targets are sorted into a queue according to this identification, with the first member of this queue becoming the selected target. This would imply that all subsequent targets of the same label are never chosen, which, if there are many such targets, also implies that most of the possible edges are never explored. A feasible means of addressing this is to equally distribute connections among identically labelled nodes. The first member of the queue becomes the selected target, but is then pushed to the back. The next target of that label will hence be a different first queue member. This resembles chained hashing in that a single key leads to a list of slots, but the access order then determines the exact slot. The downside of either of these approaches is that the position independence of the representation is lost.

5.1.2 Soft Matching

It is evident from the various instances above that strict matching could limit the effectiveness with which the space of possible edges is explored. Although solutions for overcoming these instances have also been suggested, they are computationally expensive and have their own limitations. An alternative method of label matching has been suggested and described by Luerssen and Powers (2005) and will be referred to as *soft matching*. Instead of using a small set of labels, a label set that is maximally large is used, so multiple instances of the same label within the same scope are less likely to occur. Secondly, because this means labels would rarely match according to the strict matching criterion, a label is now matched with the label that is nearest. A distance metric must hence be defined for the label space. An obvious advantage is that a label will always match another label, unless no other label exists.

Determining connectivity with a diversity of labels means less dependence on any additional node order (e.g. age, implementation order). This establishes a high degree of position independence in the representation. Yet, although the likelihood of multiple identical labels is reduced, they can still occur; for instance, from multiple identical nonterminals in a production, all of which will be replaced by cellular graphs with the same begin and end node labels. Given the large number of possible labels and the small subset present within the scope of a cellular production, a suggested fix is to append an additional source and/or target label to both the nonterminals and terminals. The additional source label (from now on referred to as the *source offset*) adds to the source labels of all begin

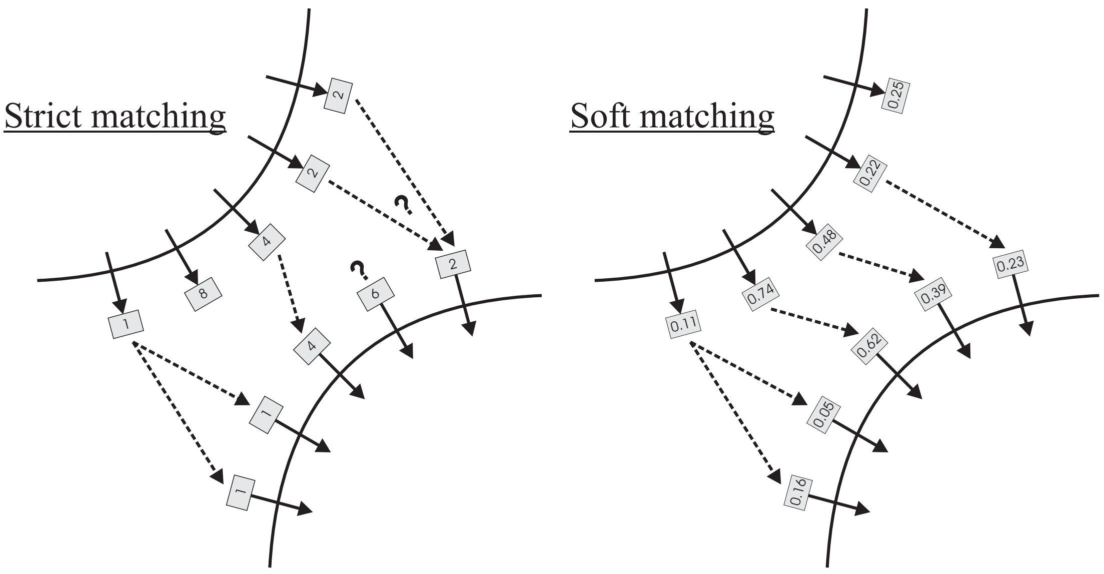


Figure 5.3: Soft matching involves distance-based matching on a large set of labels (shown here as decimal numbers) and is therefore a more complex scheme than strict matching, which requires exact matches on a small set of labels, but is also prone to duplicate labels and sources that miss out on targets.

nodes of the associated hyperedge (or inputs of the terminal), and the additional target label (from now referred to as the *target offset*) adds to the target labels of all end nodes (or outputs of the terminal).

The probability of two identical labels being identical after such a transformation is equal to the probability of any two labels being identical ($P = 2^{-64}$ for a 64-bit implementation), which virtually eliminates the problem of deciding between identical labels arising from identical terminals or nonterminals. Should identical labels or equidistant labels still occur, duplicate labels and distances are resolved by representation order, as with strict matching. We did not follow up on a further solution to the duplicates problem – which is to ensure that all labels in the entire population are unique. This is permissible as, unlike with strict matching, no two labels ever need to be identical, but the implementation would be complex (e.g. a global list, a high-period random number generator, etc.) and identical distances are still not avoided.

When applying both a source and target offset, two identical terminals or nonterminals within the same scope will behave as if they had different source labels – hence different inputs – as well as different target labels, which make them separately accessible to other cellular graphs. Applying only a target offset also allows this kind of differentiation, but the node outputs would be identical because of identical inputs (unless the represented operations are non-deterministic). Thus, without a source offset there is only a very limited benefit to having more than one identical terminal or nonterminal in a production, and more complex structures have to be established using multiple productions.

Source and target offsets can also be imagined as the positions of the terminal or nonterminal in a Euclidean space (that is, if source and target are equal; otherwise they form what might be interpreted as a tunnel or wormhole). Sources and targets of begin and end nodes are offsets with respect to this position. Extending this to multi-dimensional labels allows a physical interpretation of the system as a simulation of growth, based on relative distances between components (i.e. nodes) of the organism (i.e. graph) being grown. The term *relative* is important here, as the neighbourhoods are defined by the cellular graphs, so unlike neural growth simulations such as Nolfi and Parisi (1991) and De Jong et al. (2001), no absolute coordinate system exists. The potential advantage of this is that we never run out of space for placing nodes and edges, since space can be "created" each time a cellular production is applied.

5.1.3 Method

Does soft matching deliver any benefits over strict matching in practice? To answer this question, both these schemes have been evaluated on three distinct problems selected from the application domains listed in section 4.6. Please note that some of the conclusions of later experiments in this chapter are already applied – as indicated – to these experiments, since they are a culmination of earlier experiments, which were reported in Luerssen (2005a) and Luerssen and Powers (2005).

5.1.3.1 Binomial-3 Regression

As a target for symbolic regression, we chose the binomial-3 polynomial, which is defined as

$$f(t) = (t+1)^3. \tag{5.4}$$

Fitness cases are 21 equidistant points generated by this function over the interval of $t = [-1, 1]$. The binomial-3 polynomial has found previous use as a regression target for analysing the difficulty of GP problems (Daida et al., 2001). It is less trivial than the quadratic polynomial, the default target in the field, and it is highly multimodal.

We employ G/GRADE to evolve a population of 20 symbolic regression networks for each of 1000 generations. A $(\mu + \lambda)$ evolution strategy is used, with all parents producing a single offspring each ($\mu = 20; \lambda = 1000$). The context of a networks consists of the independent variable t as input, and the function value $f(t)$ as output. Each network may only be composed of automata that implement the operations $\{+, -, \times, div\}$, where div returns 1 if the divisor is zero, otherwise it returns the actual result of the division. If an automaton has less than two inputs, the missing inputs are assumed to be 0 for $\{+, -, div\}$ and 1 for $\{\times\}$. Networks are evaluated by making a feedforward pass from input t to output $f(t)$; recurrent connections are not permitted (see section 5.3.1 on how this is achieved).

5.1.3.2 Random Bit Sequence

The second problem involves the design of a small Boolean circuit that approximates the output of a 4-bit de Bruijn Counter, as detailed in section 4.6.3. The fitness of the network is defined as the MSE between this random bit sequence and the sequence produced as an output by the network. The network must be composed of automata that implement Boolean operations from the set $\{AND, OR, XOR\}$; if an automaton has less than two inputs, the missing inputs are assumed to be 0. The population consists of 20 network evolved for 5000 generations.

Each network is simulated for 36 cycles (2×16 cycles + 4 cycles lead-in), with outputs compared every 2 cycles for the last 32 cycles against the target sequence. A faster evaluation (e.g. in 16 cycles) is not viable, since within a single cycle each automaton can only affect its immediate neighbours, so a larger graph may take multiple cycles to generate the next output. The limit of 36 cycles was chosen based on our informal observation that solutions are indeed possible within this limit and rare when using fewer cycles. To make it possible for a variety of networks to be synchronized with the intended rate of output sampling, we additionally simulate line delays for automata inputs. Longer line delays are applied randomly with a geometric probability of 0.5 to each input of any new terminal added to a cellular graph. These delays are then stored with the graph; during simulation, they increase the number of cycles required by a signal on the affected edge to become available to the automaton.

5.1.3.3 Backpropagation MLP

The third problem task for this experiment is finding a multilayer MLP that classifies the Iris dataset (see also section 4.6.2). Unlike with the previous tasks, the population here consists only of one network that is evolved for 1000 generations (i.e. a (1+1) strategy is used). The nodes of the network are log-sigmoid neurons trained with standard backpropagation and an adaptive learning rate ($LR_{start} = 0.1$, $LR_{inc} = 1.05$, $LR_{dec} = 0.7$, $LR_{thres} = 1.04$). The network space is restricted to feedforward architectures only.

Weights are initialised randomly and uniformly within the range [0,1]. The reasons for evolving only one network are,

- the problem is easy,
- backpropagation is computationally expensive,
- we want to be at least competitive with random guessing by the user, so the effort required by this evolutionary approach should be minimal.

The network and its offspring are therefore trained for just one cycle at each generation, and any weight changes are inherited by their respective offspring (i.e. we apply Lamarckian evolution). Weights are stored with neurons rather than as tentacle attributes. Only once network derivation is complete are the weights mapped onto the inputs of a neuron deterministically using the associated edge labels as indices into a weight vector, which also allows for a limited degree of position independence. We regarded the dissociation of weights from the grammar representation as the most immediate solution to the concern that edges are formed from potentially multiple tentacles. Simply adding up tentacle weights would imply that the size of a single weight is dependent on the number of productions required to describe the network, which is not a sensible bias.

75 patterns from the Iris set are presented in random order to the networks, a forward and backward pass are applied, then 75 different patterns from the Iris set are also presented, and the fitness MSE is computed by comparing the actual output against the target output. The total computational effort for this setup is twice that of training the mean-generation network for 1000 epochs, or somewhat less than if the user were to guess the topology of and then train a neural network with validation.

5.1.3.4 Computer Network Topology

The final problem task, the optimisation of a communications network first introduced in section 4.6.4.1 and denoted *Computer Network Topology* (*CNT*), is only applied to evaluating the impact of offsets on soft matching. The simulator is incompatible with strict matching, as it exploits the target offsets for the positioning of switches and cables in the virtual landscape. Accordingly, (x, y) label pairs instead of single labels are assigned to each node, so that we may operate on a two-dimensional landscape. A Euclidean distance function is employed to match labels, but also to determine distances between nodes, which contribute to the cost objective of the solution. As previously, the labels are initialised randomly and not changed by evolution, so the cost optimisation is unlikely to be as effective as if we had evolved the labels and hence the positioning. However, the choice of method for doing so and the potential consequences of this were not something we wished to address here. G/GRADE has to optimise 10 different, randomised landscapes 10 times each. The performance objective is given by the mean percentage of unfulfilled data streams of requesting units.

5.1.3.5 Experimental Configuration

The experiments will compare 12 variations of strict matching and 4 variations of soft matching against each other. Results are averaged over 100 separate runs, each with a different random seed. Statistical significance is determined using a non-parametric Wilcoxon rank sum test on the best solutions of the final generation of each run. Differences with $p < 0.05$ are regarded as significant, but no Bonferroni correction is applied to any of these comparisons, unless otherwise noted. (p-values will be shown, unless they are exceptionally small.)

Across all problems, a maximum of 1000 productions and 1000 terminals per production are permitted for each network, although no networks actually reached this limit in these experiments. Terminals are implicitly wrapped into productions, as further described in section 5.3. The permitted mutations are those discussed in section 4.3.1, but excluding the *modify* operator. A single production is selected for mutation and a single mutation is applied at a time, with a geometric probability of 0.5 that further mutations are applied. Selection favours solutions that are nondominated across two objectives: the function error, which is the mean squared error over all training samples, and the size of the network, as defined in section 4.3.2.

Strict matching selects randomly from a set of integer labels in an interval I (as defined below), or, at a probability P_R selects randomly from the labels of the immediately visible scope (see section 5.2), unless there is no other label in the scope. Interval I is either $[1, 5]$ or $[1, 100]$, and three values for P_R are suggested: 0.0, 0.5, and 0.9, which produce different proportions of random selection against selection from context. (1.0 is avoided, as only a single label value would ever arise from this.) Additionally, strict matching is also tested either with or without redistribution for duplicate labels on combinations of the above parameters, leading to a total of 12 configurations.

Soft matching always selects randomly a floating-point label from a uniform distribution between $[0, 1)$. The possible coordinate space with offsets is limited by having a wrap-around from 1 to 0 and vice versa. The minimum difference (with possible wrap-around) between two labels is used as the default distance measure. The independent variable for the soft matching experiments is the addition of nonterminal and terminal offsets. Besides the elementary case of no offsets, we also test using offsets for only the target labels of a terminal or nonterminal (marked *Target Offset*), using the same offset for source labels and target labels (marked *Point Offset*), and using different offsets for source labels and target labels (marked *Tunnel Offset*).

5.1.4 Results & Discussion

The results for the experiments described above are shown as box plots in figures 5.5 and 5.6. The error rates and sizes of the networks are also summarised in tables A.1, A.3 and A.5 in Appendix A. Please refer to Appendix A for tabulations of the numerical results of all the experiments in this thesis.

The results reveal that the performance of strict label matching is dependent on the parameter P_R. As was expected from equation 5.2, having a large label set to choose from reduces the likelihood of randomly finding a label that actually matches one in the local scope. This obstacle is overcome by selecting from existing labels in the scope. Selecting 50% or 90% of labels by this method leads to a significantly better final MSE of the best solution ($p < 4 \times 10^{-10}$). $P_R = 0.9$ exhibits less error than $P_R = 0.5$ for all problems, but the difference is only significant with the RBS circuit ($p < 0.0007$). P_R evidently needs to be greater than 0 to ensure that there is a diversity of distinct labels, but there appears to be some tolerance as to exactly what P_R is set to.

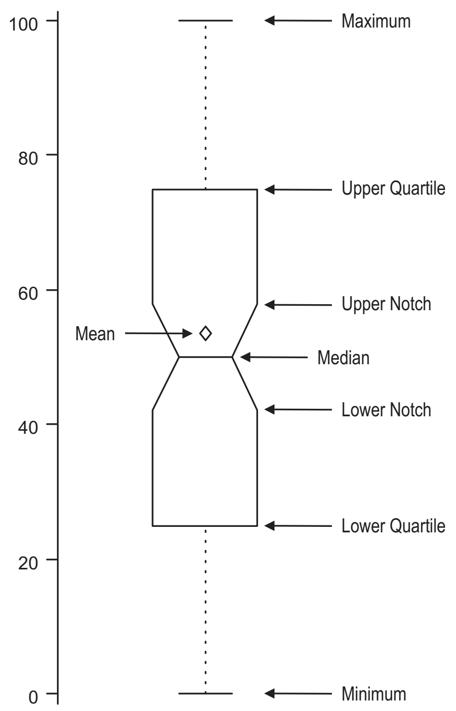


Figure 5.4: Box plots are helpful in visualising the distribution of results. This is a simple box-and-whiskers plot, as the whiskers extend to the minimum and maximum and outliers are not separately plotted here. Notches show the approximate confidence intervals around the medians for box to box comparison. The medians of two boxes differ at the 0.05 significance level if the notches of either box do not overlap.

While selecting labels from the scope results in a performance improvement, determining which labels are in the scope of the cellular graph being mutated is not a trivial matter: it requires a partial expansion of the grammar into the graph embedding this cellular graph. This increases the overall complexity and computation cost associated with the system. A simpler alternative is to reduce the total set of labels that we can choose from, which increases the likelihood of randomly selecting a label that already exists in the scope. The results support this hypothesis. Using a set of 5 labels instead of 100 labels leads to significantly better performance for $P_R = 0$, but for instances of non-zero P_R performance is inconsistently better or worse depending on the problem.

Reducing the label set can be problematic for two reasons. Firstly, if there are only n labels but the problem involves $n+1$ inputs, it becomes unsolvable because the representation becomes constrained in its ability to distinguish between nodes. This should not be an issue with the problems tested here, but nonetheless represents a potential limitation of the framework. Secondly, the likelihood

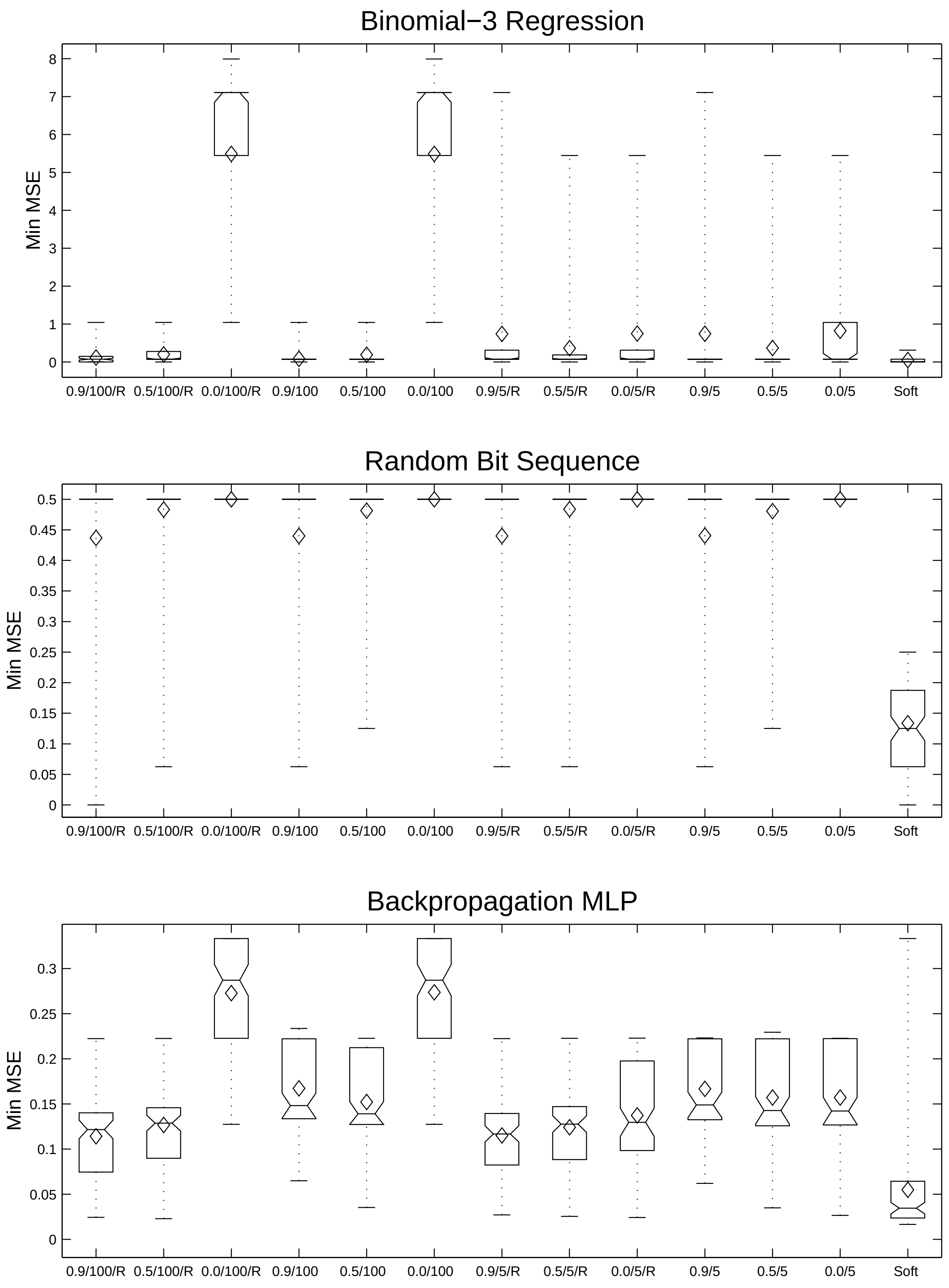


Figure 5.5: Minimum MSE box plot of different configurations of the strict matching scheme compared to soft matching with target offsets. Configurations are denoted $P/L/R$, where P is the probability of using an existing label, L is the size of the label set, and edge redistribution is specified by the R. Refer to figure 5.4 for a brief description of how to read box plots.

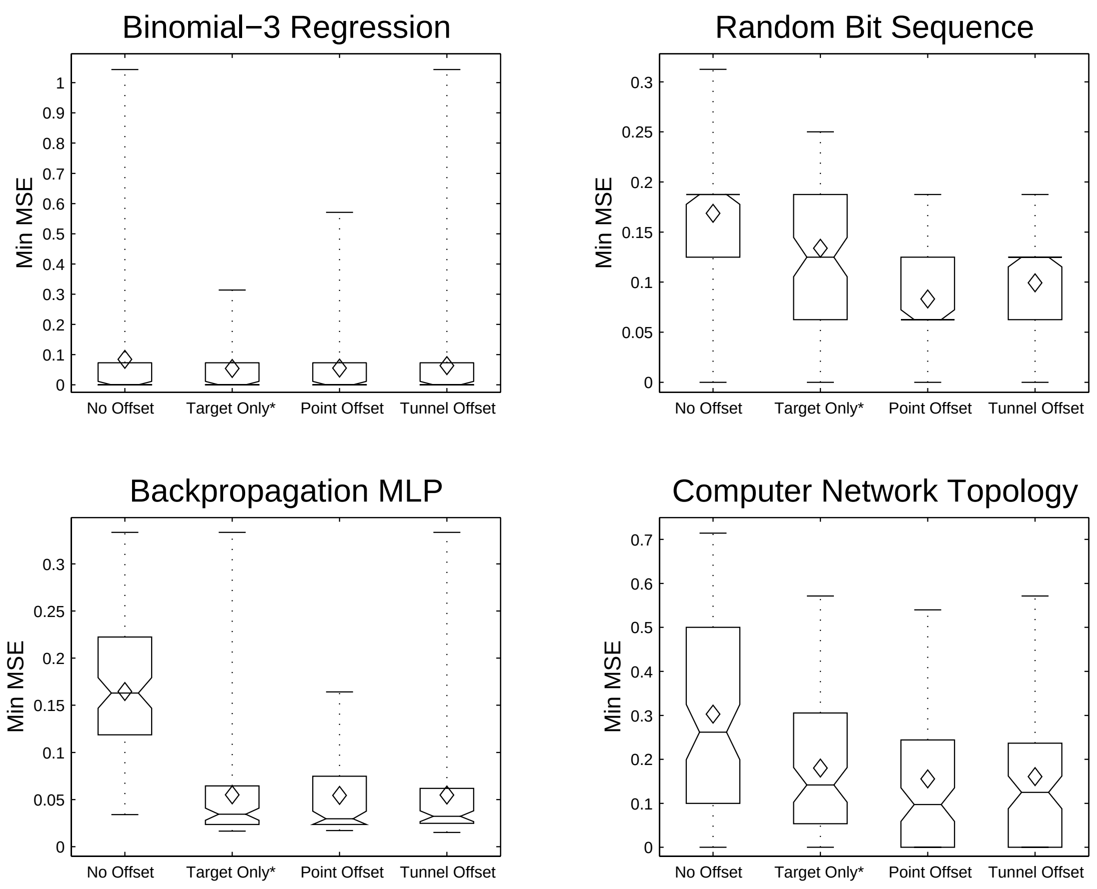


Figure 5.6: Box plot of results from using different offset schemes for soft matching. The asterisk (*) denotes the default configuration, i.e. the configuration also used in other experiments.

of label duplication in the same scope increases. Without any additional bias, selecting a target label that already exists in the scope constitutes a neutral mutation in at least half of all instances, since the other existing targets may be preferred. We have suggested redistributing labels of the same source across all targets of the same label as a means of addressing this. Indeed, applying the queue-based edge redistribution significantly improves the success rates (i.e. the fraction of runs that produce optimal solutions) on the Binomial-3 regression and the performance on the neural network task, 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 RBS circuit. In the study by Luerssen and Powers (2005), some problems similarly benefited from edge redistribution, while others did not. Since edge redistribution only applies to instances where duplicate labels occur, duplicates must hence be a hindrance in solving some problems; but what makes a problem sensitive to this is not clear from these results. With this reasoning in mind, having a small label set, and hence a greater probability of duplicates, should also benefit more from edge redistribution, but the results are not conclusive about this.

Soft matching avoids this issue for the most part, as labels are rarely duplicates by default. Soft matching without additional offsets is significantly superior to the best parameterization of strict matching on the Binomial-3 regression and the RBS circuit (no comparison exceeds $p < 10^{-8}$), but not on the Backpropagation MLP. With Target Offset the difference is significant on every tested problem (no comparison exceeds $p < 10^{-12}$). Adding offsets to each terminal and non-terminal leads to significantly better performance than without any offset. The compositional freedom added to the graph grammar by these offsets appears to outweigh any complexity issues caused by the increase in the number of labels requiring optimisation. The target offset alone is mostly sufficient, however. It is only significantly worse than the other offsets on the RBS circuit, especially on the success rate, where only one in a hundred runs is a winner, as compared to 11% for Point Offset.

The conclusion from these results is that strict matching is not an appropriate means of label matching in an evolutionary optimisation system; soft matching with offsets is certainly superior. The simplest effective soft matching, with target offsets only, will therefore be used as the default configuration for all subsequent experiments, unless noted otherwise.

5.2 Graph Modularity

Section 3.2 proposes that a system that can be decomposed into modules may be more easily optimised. For this to be practical, the optimisation system must account for the representation of modules. 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. This way a successful design can be protected from being affected by changes to other components of the system.

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. By limiting the number

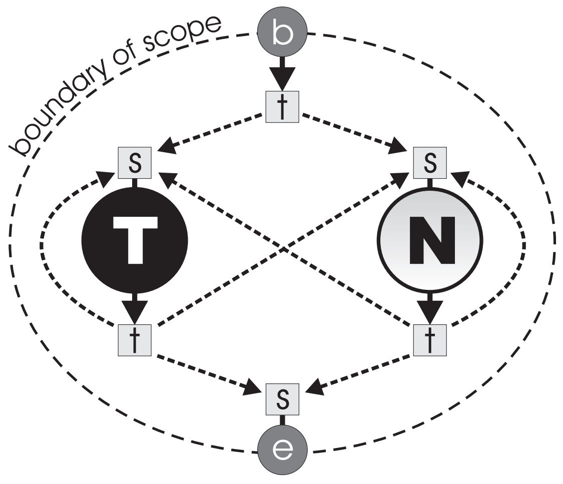


Figure 5.7: Source labels can only match target labels within the scope of the production; this includes the source labels of begin nodes and target labels of end nodes of included cellular graphs. No connections between target labels of begin nodes and source labels of end nodes are permitted, as they are not required for describing any graph, yet increase the difficulty of implementation by a considerable amount.

of begin and end nodes or reducing the number of edges from begin and to end nodes within the hypergraph we directly reduce the number of possible structural dependencies and thereby turn the hypergraph into a module. In the cellular production model, the same effect is accomplished by limiting the cellular tentacles of a cellular graph – we will hence also refer to these tentacles as the interface of the cellular graph.

Since all dependencies between cellular graphs involve the interface, the label matching approach that was presented in the previous section cannot just match any labels, but needs to be restricted to a specific scope for each label type. This is shown in figure 5.7 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. There are drawbacks to this, however. To connect nodes that are separated by multiple scope boundaries, numerous begin or end nodes need to be defined among intermediate productions in order to bridge these scope boundaries. This is akin to passing down a parameter in a sequence of function calls.

The purpose of modularity is to make connections between widely separated modules difficult, but there are instances in which this limitation is a hassle. For instance, in a proper modular framework we should assume that the problem nodes (or variables, in a non-graph representation) are external to the solution, i.e. they constitute inputs and outputs to the network module. Thus, by the scoping rules suggested above only the starting cellular production has direct access to these top-level nodes. Connecting a node in a deeply embedded production with an input and an output of the graph requires edges from the input to the node back to the output to be established for each intermediate scope boundary. Moreover, if the number of edges between modules is increased (e.g. on a problem with many data lines), the probability of bridging multiple consecutive scope boundaries drops exponentially, unless fitness also scales reasonably smoothly with the degree of completion of the interface (which is rather unlikely in the current framework).

In GP, only edges from the node to the output (the root of the tree) must be determined, because variables can be located anywhere in the tree. Unlike with the above approach, looking at the root of the GP tree tells us nothing about the variables accessed, so this approach is not very modular, but it requires only half as many edges on average to be established during evolution. So a simple method for making modularity more selective and less restrictive is to permit global variables as in GP. This can be achieved by assigning a flag to each begin and end node indicating whether it matches labels in the local scope or the global scope. In the global scope, only top-level sources and targets are matched. The benefit of such *global tentacles* is expected to apply mainly to graphs with many incoming and outgoing edges at the top-level, as boundary bridging to the top-level can be skipped by applying the global flag. The disadvantage is a loss of assured modularity; you cannot determine the inputs and outputs of the network solely by looking at the starting production anymore.

One of the principal issues arising with problems of many variables is that, unless global tentacles are used, the starting production must define tentacles to access each variable, and this can only be done by adding these tentacles through mutation of the production. If two separate productions each solve a different half of the problem of accessing all the variables, then each of them has to independently evolve the other half as well, as there is no mechanism by which both could combine. Hence, the second modification to our algorithm is to assign a modularity-flag to each cellular graph that defines whether the scope boundary should apply at all (Luerssen and Powers, 2005). If the flag is off, then the scope boundary of the next including cellular graph that is modular is used instead.

In practice, this means that if a cellular production N_H is referring to a non-modular cellular production N_o, it is equivalent to the definitions of N_H and N_o

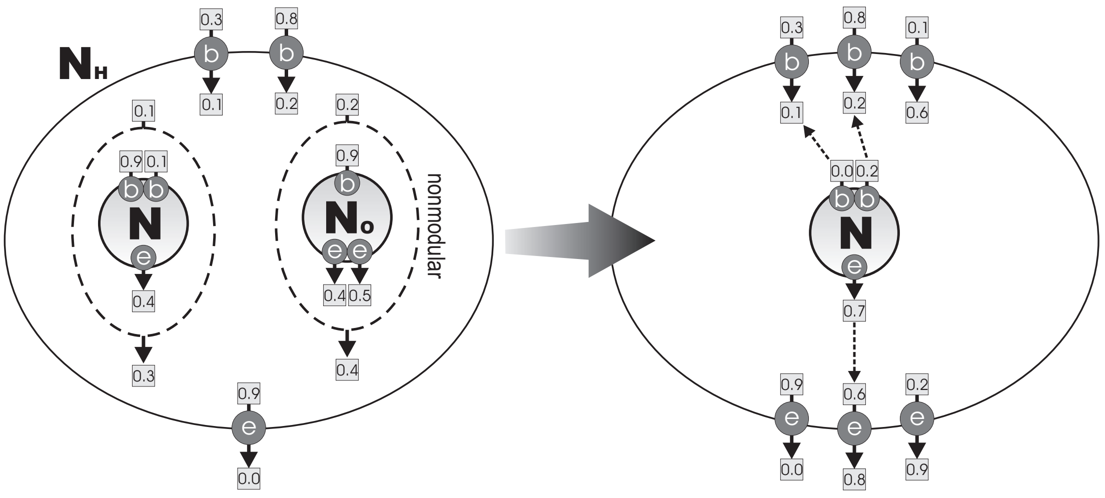


Figure 5.8: Label offsets add to the labels of associated hypergraphs (or vertices, not shown). A non-modular production N_o concatenates to the referring production N_H, hence moving the nodes of the embedded cellular graph into the scope of the host cellular graph. Connections are established subsequently. Please note that not all labels are actually shown on the left.

being concatenated (see figure 5.8 for an illustration of this). A cellular graph with many incoming and outgoing edges can now be defined in terms of non-modular cellular graphs, which may then also be reused for defining other cellular graphs. The need to re-discover the same or similar interfaces by adding each tentacle individually is thus greatly reduced. Additionally, there is now the potential for linkage learning in the concatenation of interfaces, as the position independence of external nodes allows for a floating representation as with mGAs (cf. section 2.6.2.1).

5.2.1 Method

The experimental method of section 5.1.3 is reapplied here with the same parameters on the same problem tasks: Binomial-3 Regression, RBS circuit, Backpropagation MLP, and the CNT design. For label matching we use soft matching with target offset.

The first experiment introduces global tentacles. Instead of matching labels in the local scope, global tentacles match labels (and hence nodes) in the global scope of the starting production of the graph. Begin and end nodes added to a production during mutation have a probability of $P_{globalIO}$ of constituting global tentacles. Values of $P_{globalIO} = 0.1$ and $P_{globalIO} = 0.5$ are evaluated in this

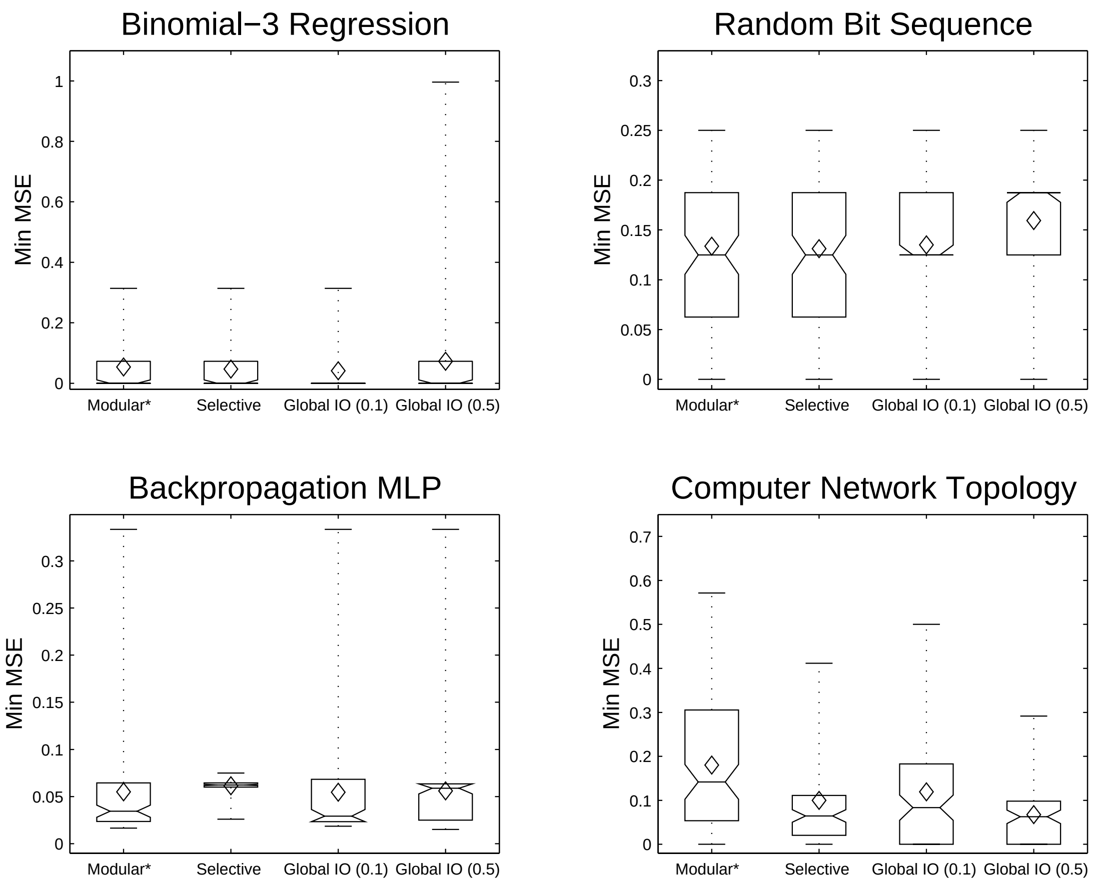


Figure 5.9: Minimum MSE box plot comparing the default modular configuration against using selectively modular productions and global tentacles at 10% and 50% insertion probability.

experiment against the default of no global tentacles.

Selective modularity is implemented in the second experiment. The initial cellular graph is modular. However, anytime a cellular graph is mutated, there is a 10% chance of changing the modularity flag, i.e. making the graph non-modular if it is modular, or vice versa. This is in addition to any other conventional mutations that may occur.

5.2.2 Results & Discussion

The results, shown in figure 5.9, fail to provide clarity on the evolutionary impact of modularity. Neither the addition of non-modular productions nor of global tentacles produces consistent changes across all problem tasks. The stated hypothesis was that modularity would increase the number of graph grammar components required to describe a network, and this was slowing evolutionary convergence. However, significant performance gains from making modularity optional are only observed with the CNT design ($p < 0.02$), but not on the other problems. The likely reason for this is that they involve far fewer variables than the CNT design – indeed, the circuit design problem only has a single output. The problem variables are therefore highly available and increasing their availability any further (with $P_{globalIO} = 0.5$) can lead to a performance dip. For problems of this nature there appears to be no notable overhead imposed by modularity and the consequent need to pass problem variables through multiple interfaces, but the results for the CNT design indicate that this statement cannot be generalised to all other problems.

5.3 Implicit and Explicit Gluing Models

Cellular graphs do not define how terminal vertices connect. Additional information is needed to establish edges from a terminal to another terminal or any component of the cellular graph. It is clear that with label matching as the sole means of defining connectivity, terminals must be assigned labels as well. One method consistent with our framework involves wrapping each terminal into a cellular production so that it may be attached like a hyperedge by a cellular graph. The actual connection between the terminal and the external nodes of the cellular graph into which it is wrapped would need to be defined manually in this case. If the cellular graph is modular, then the terminal is hidden from other cellular graphs and could be treated like any other cellular graph.

However, with some of the terminals of the evaluated problem tasks, the degree of the vertex may be variable, e.g. a neuron can have any number of inputs. This would imply that we need to define a different production for each vertex degree of a particular terminal, which seems perhaps a bit too much manual effort. One possibility, and the approach also used in the previous experiments, is to implicitly wrap each terminal into a default production, which connects to up to $n + 1$ nodes of any label, where n is the maximum number of inputs defined for that terminal type, and the terminal has one output. If source and target offsets are defined for cellular graphs, a source label and a target label are also attached to each terminal in the cellular graph. Nodes are chosen as inputs in order of proximity to the source label (or 0 for setups without source offset), while the terminal output is connected to the source matching a target label (or 0 for setups without target offset); nodes are never chosen more than once. A drawback of

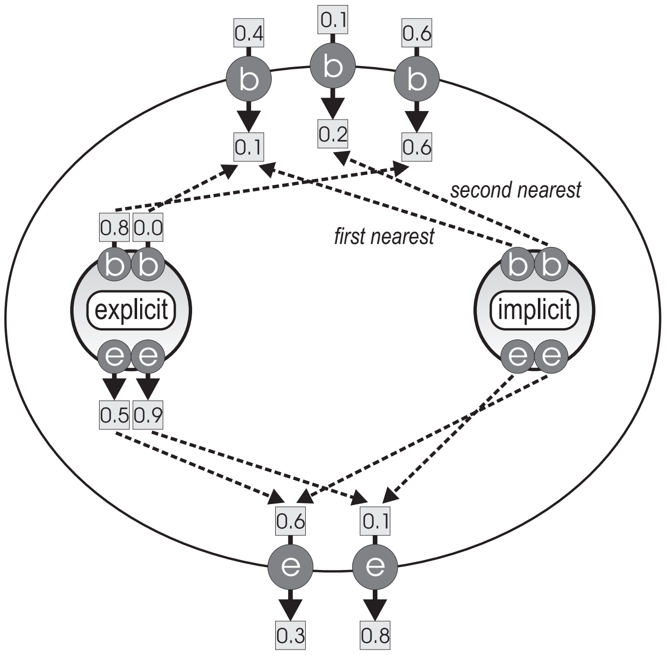


Figure 5.10: The explicit gluing scheme employed so far involves matching label pairs. Fewer labels are needed if we assume that the n-th external node is connected to the n-th possible source/target. This, however, increases position dependency within the representation.

this method is that it is formally less "pure" than with explicitly user-defined cellular graphs, as we now need to distinguish between evolved cellular graphs (for nonterminals) and implicitly generated cellular graphs (for terminals).

5.3.1 Feedforward Networks

We can exploit the possible differentiation between the implicitly wrapped terminals and the nonterminals of explicitly evolved cellular productions to impose feedforwardness (i.e. a cycle-free topology) on the solution networks. Two of the four problem tasks – the regression and the neural network – used in the experiments of this chapter require only feedforward architectures. The search process can be facilitated here by restricting the search space to only such architectures. The following rule is applied: nonterminals only receive incoming edges from begin nodes, and terminals can receive incoming edges from all nodes except directly from other terminals. This changes the label matching process so that if a selected source does not satisfy the rule the next best source that does satisfy the rule is selected. Thus, the only permitted interaction between nonterminals and/or terminals within the same scope is the nonterminal providing an input to the terminal. Interactions between two or more nonterminals or between two or more terminals is not possible with this rule, which results in cycle-free networks being derived.

5.3.2 Method

We wish to evaluate whether implicitly wrapping terminal vertices into cellular productions is a good idea by comparing it against hand-coded cellular productions that also wrap the same terminals. In the latter instance, all terminals are hidden to the system, so enforcing feedforwardness by distinguishing between nonterminals and terminals is not possible anymore. We will first see what impact the removal of the feedforward constraint has on the Binomial-3 Regression and the Backpropagation MLP, although self-loops will still be prohibited and signals along any cyclic links are ignored. A performance decrease is expected, because fewer of the possible network configurations are now suitable for solving the problem.

Subsequently, terminals for all problem tasks will be replaced by hand-coded productions reflecting the different input arrangements that these terminals might encounter. For the Binomial-3 Regression, there will be a production for each of $\{+, -, \times, div\}$ with two incoming tentacles and one outgoing; a production for $\{+\}$ with one incoming tentacle and one outgoing; and a production for each of $\{+, \times\}$ with only an outgoing tentacle, representing the constants 0 and 1 that are implicitly established by this. For the RBS circuit, there will be a production for each of $\{$*AND, OR, XOR*$\}$ with two incoming tentacles and one outgoing, as well as two additional productions for *OR* with one or zero incoming tentacles, respectively, and one outgoing. For the neural network, there will be 5 productions wrapping the sigmoid neuron, for each of one to five incoming tentacles (and one outgoing tentacle always). Likewise, for the CNT design, there will be five possible switches with one to five inputs.

The mutations for inserting or removing terminals have been removed, and the mutation probabilities for inserting or removing nonterminals have been doubled instead. 50% of inserted nonterminals are allocated to our hand-designed productions; this avoids having their insertion probability dependent on the total number of productions in the population.

Finally, to round out this exploration of implicitly defined productions, we will also evaluate implicit definitions of nonterminals. The target nearest to the

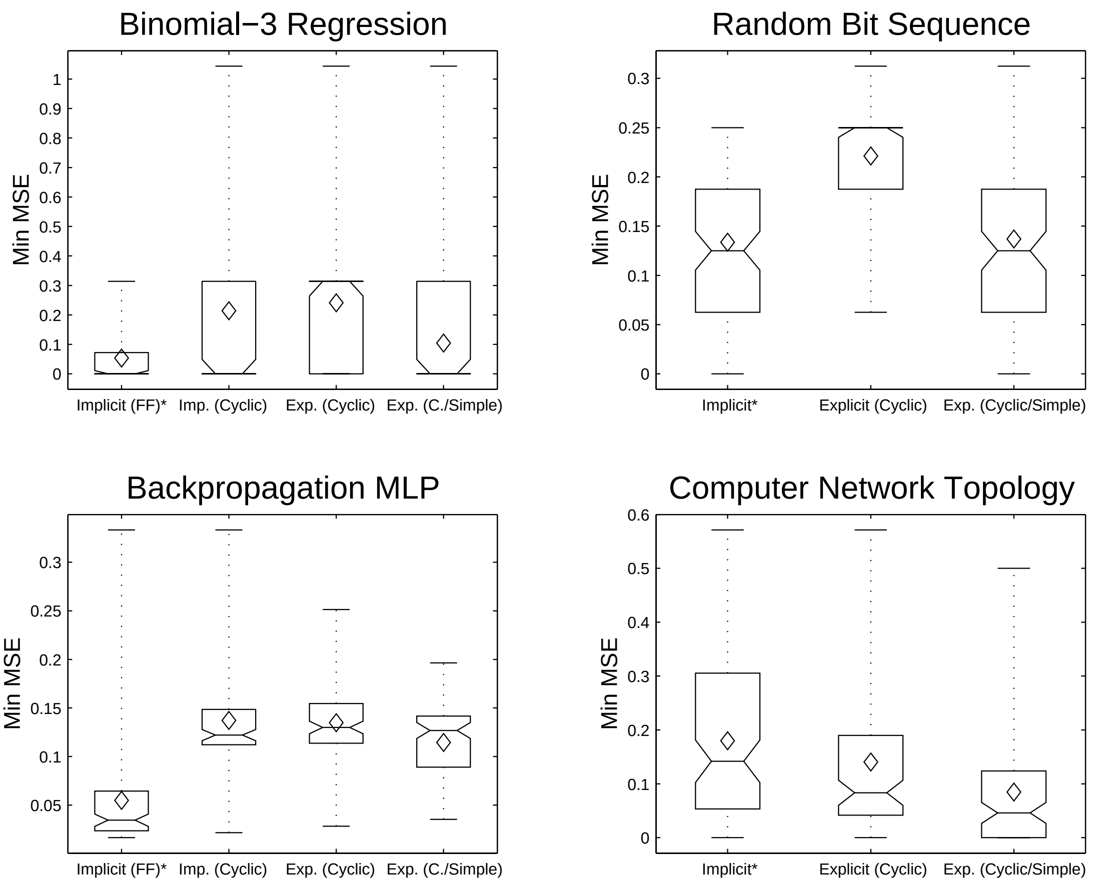


Figure 5.11: In this box plot, implicit gluing models for terminal vertices are compared against explicit, user-defined production models. FF signifies a feed-forward constraint on the architecture (see section 5.3.1), whereas *simple* denotes that only simple graphs are allowed (using the *Simple (Terminal + Nonterminal)* configuration, see section 5.4).

source offset of the nonterminal is fused with the earliest begin node automatically without matching it against a source label (which therefore becomes superfluous); terminals are also defined implicitly, but target labels are still defined as previously. A major drawback here is that certain topologies, such as a XOR-like architecture, require more productions to be constructed. Moreover, it greatly increases the position dependency of external nodes within the representation, which we originally (in section 4.1) intended to avoid.

5.3.3 Results & Discussion

Results are averaged over 100 runs and can be seen in figure 5.11 (for implicit terminals) and figure 5.12 (for implicit nonterminals). Particularly noteworthy

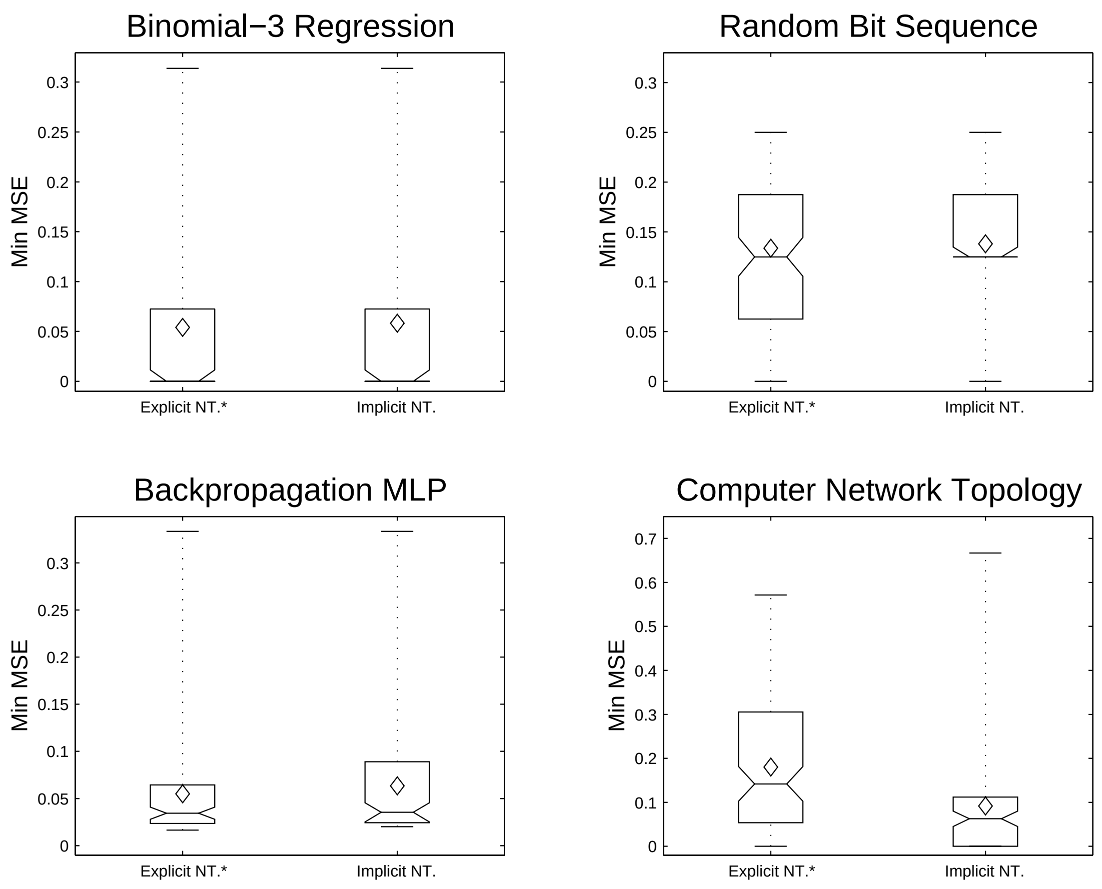


Figure 5.12: Implicit gluing is extended to also include nonterminals (i.e. cellular graphs), so that external nodes can be defined without source labels.

is the negative effect of allowing cycles in the solution graphs of problems that only require a feedforward graph. This significantly reduces both median performance and the success rate when compared to feedforward graphs ($p < 0.0004$ for each), which is expected, given that the search space is increased without a corresponding increase in viable solutions. Much better results with cyclic graphs are obtained in Luerssen (2005a) and Luerssen and Powers (2005), where the graphs are simulated for multiple passes. Recurrent feedback may affect fitness in those cases, and it appears that there exist some compact recurrent solutions, which may explain the performance gap compared to this experiment.

Performance on three out of the four problem tasks is not significantly different with hand-coded terminal productions than with implicit terminal productions, although the lack of feedforward constraint remains a drawback. Only the RBS circuit reveals a significant advantage in favour of implicit terminal productions ($p < 0.002$). Any differences must be attributed to either the different mutation distributions when selecting terminals (e.g. there are 3 terminals with the RBS circuit, but 5 hand-coded productions); the different sizes of the resulting graph grammar, as grammars with implicit terminal productions grow relatively less in size with each added terminal; or the fact that nodes are never chosen twice, which may bias the graph design (see section 5.4).

Assessing hand-coded terminal productions differently from other production would address these issues, but the results indicate that implicit terminal productions are quite effective. Since they are also more convenient than defining productions by hand, we will continue to use them throughout other experiments. On the other hand, implicitly wrapped nonterminals will not be used due to our concern about the aforementioned drawbacks – although this is not supported by the results on these problem tasks. Performance with implicitly wrapped nonterminals is remarkably similar, except on the CNT design, where it is even significantly better ($p < 0.00004$). It thus appears that a simpler model of graph gluing than the cellular graph model might be entirely viable in practice, if not in theory.

5.4 Enforcing Simple Graphs

It is not graphs that we derive from the evolved graph grammar, but pseudographs, because the grammar can easily describe a graph with a self-looping edge or more than one edge joining the same pair of vertices. In fact, within a modular graph grammar it is impossible to determine whether two or more of the incident edges of a node are multi-edges, because this would require knowing whether they are incident on the same other node. Yet this other node may be outside the scope of the first node and hence not open to inspection. Consequently, the variation operators cannot, without actually deriving the network, ensure that an evolved network is always a simple graph. However, the graph grammar itself can be constrained to only produce simple graphs. We suggest two such schemes for preventing pseudographs. Neither scheme involves removing loops or combining multi-edges, as this would likely starve the terminal vertices of any available edges.

The first scheme is based on the implicit gluing presented in the previous section. When deciding on the connectivity of a terminal we verify that each possible input source is different (i.e. not just the external node, but the terminal it represents). Any input source that is identical to a previous input source is eliminated from consideration, so no pseudographs can arise. In some cases there may be very few different input sources within the scope, but if there exist multi-edges between nonterminals (i.e. “multi-tentacles” between hyperedges), then by eliminating identical input sources for nonterminals as well, we improve the diversity of input sources within the scope. This is our second, extended scheme against pseudographs.

Clearly, these schemes make the process of connecting nodes a more elaborate process. Is it truly desirable to avoid pseudographs? The additional representational flexibility of pseudographs has some potential benefits. For example, the arithmetic concept of $2x$ can be represented by two edges both incident on a node representing x and a node representing an adder; in a simple graph, two instances of x would either be necessary or such kind of reuse would require a more elaborate graph with more nodes and edges. On the other hand, if the intended solution is not a pseudograph, then the larger search space constituted by pseudographs will merely slow down the evolutionary convergence.

5.4.1 Method

This experiment is intended to show whether pseudographs are of help or hindrance in finding solutions to our standard set of problem tasks. Our investigation relies on the two suggested schemes for avoiding pseudographs described above; we compare this against the default (pseudograph) setup used previously in this chapter. No other parameters are changed. Avoiding multiple edges during implicit terminal gluing is labelled *Simple (Terminal)*, while avoiding multiple edges during the matching of nonterminal tentacles is labelled *Simple (Terminal + Nonterminal)*; allowing multiple edges is labelled *Pseudograph*.

5.4.2 Results & Discussion

Results are shown in figure 5.13. On most problem tasks, no significant differences in performance are observed between pseudograph and simple graph. The sole exception is the CNT design where the *Simple (Terminal + Nonterminal)* configuration more than halves the error rate ($p < 9 \times 10^{-6}$). No such improvement arises from constraining just the edges incident on terminal vertices, so, clearly, the occurrence of multi-tentacles has practical relevance in describing simple graphs with a cellular graph grammar. Since only one in four problem tasks responds to the simple graph constraints, however, this reasoning cannot easily be generalised. The overall results in fact imply that there is no universal

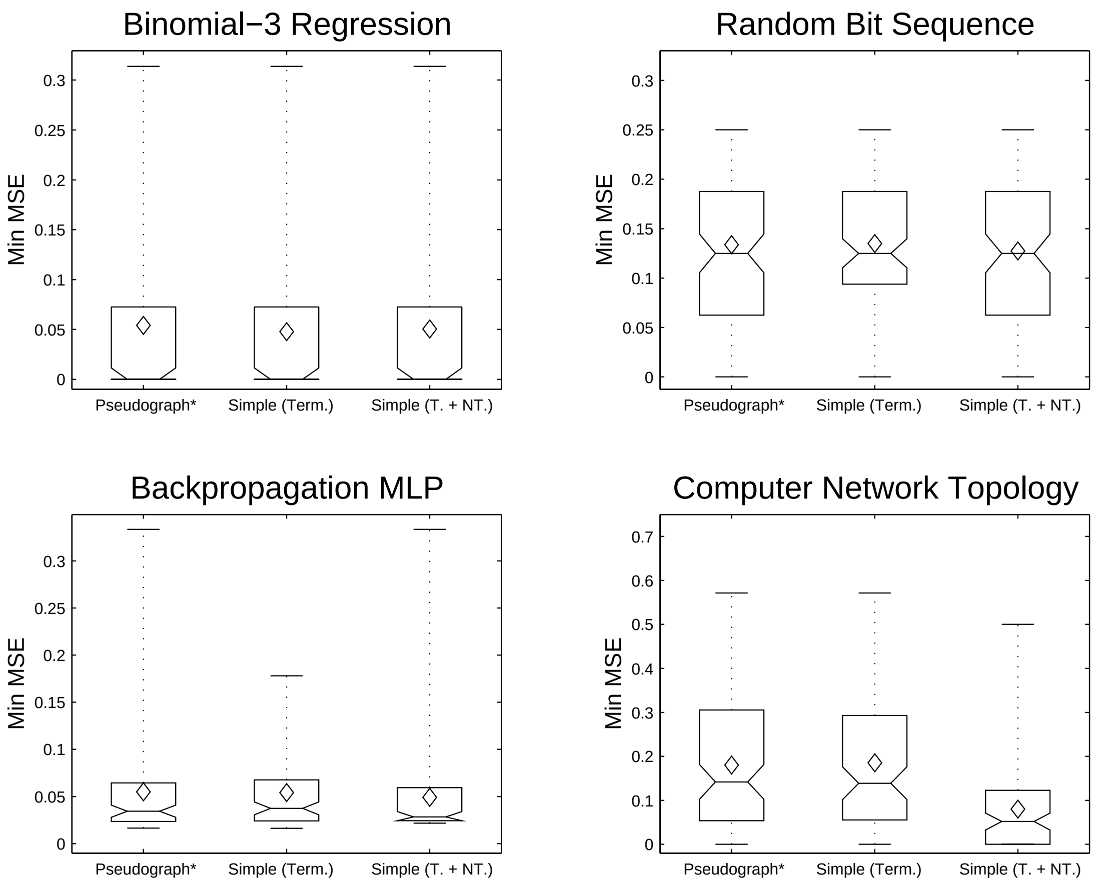


Figure 5.13: The default model of graph construction can create pseudographs. Two extensions of the label matching scheme are compared here, which constrain the search to simple graphs. The first prohibits a terminal node from forming multiple edges with another node, the second also prohibits hyperedge nonterminals from doing so.

penalty for trying to solve problems as pseudographs, even though they can easily be solved by simple graphs, and there are far more pseudograph topologies imaginable than simple graph topologies. A possible explanation for this lies in the implicit terminal wrapping used here, which already biases against multi-edges by never choosing a node twice, so the benefits of the simple graph constraint are marginalised. This is supported by the notable performance improvements observed in figure 5.11 when combining the simple graph constraint with hand-coded productions. Yet various problems that emphasise the benefits of pseudographs are certainly also imaginable, e.g. a problem where properties of a single node are required many times by another node. Taking into account the additional effort of applying the constraints, it thus appears that there is little justification for artificially constraining the output of the evolved graph grammar – which is intrinsically a pseudograph – into simple graphs, at least for performance reasons on most problems.

5.5 Summary

In this chapter we made several discoveries towards an effective model of graph construction. The most obvious method of label matching that arises from the framework of hyperedge replacement, the idea of strict matching, performed rather poorly. We provided a theoretical justification of why strict matching may fail and this was fully born out by the results. The soft labelling model that we alternatively suggested, which involves nearest neighbour label matching on a diverse set of labels, was comparatively much more successful. It also provides the option of additional node offsets, which reduce the problem of label duplicates and indeed give better performance, although it is not clear just how much offset is actually needed. On the issue of label lookup, employing a strictly modular scope of only looking up the immediate host graph is quite effective, despite this also implying that the root cellular production must define the interface for the entire network. However, only a single problem benefited from making modularity more selective or allowing global variables as terminals.

We also investigated whether source labels could be eliminated by implicitly allocating tentacles according to the distance of a source node from a target node. Indeed, this implicit gluing was our default approach for terminal definition, as otherwise the user would have to specify the cellular graph extension for each terminal type. Implicit gluing was found to be mostly beneficial, and it can be extended to nonterminals, although it appears to make less of a difference there and could, in theory, complicate the evolution of certain topologies. The reason why it is beneficial at all may have to do with the nature of the networks derived from the graph grammar: they aren't graphs, but pseudographs. Forcing the derivation to generate simple graphs leads to only few notable performance improvements, but the implicit gluing already biases strongly against pseudographs (without excluding them) by never choosing a node twice. We will therefore continue to use implicit gluing for terminals, but explicit gluing for nonterminals.

Chapter 6

A Case for Phenotypic Diversity

Initialisation is an important phase in many evolutionary algorithms. GAs require an initial population of random strings that contain a diversity of schemata that can be recombined to form strings of potentially higher fitness. In GP, a variety of methods exist for generating an initial population of syntax trees (Luke and Panait, 2001). Most commonly used is the ramped half-n-half method introduced by Koza (1992), which either randomly picks functions and terminals or just functions to build a tree no deeper in any branch than a user-specified limit. This population of trees again provides a reservoir of diverse building blocks from which potentially fitter trees can be constructed.

Building blocks are also essential for graph grammar evolution. New graphs are defined from productions that already exist, and these must come from somewhere. Starting with random productions, however, is not viable, because recursive relationships between the productions can make it difficult to control the size of the resulting graphs. Additionally, unlike with trees, vertices are not required to be adjacent to other vertices, so a random initialisation will often produce disconnected graphs. In the context of automata networks we already know in advance that such solutions will not be fit. They also constitute a poor source of building blocks, as cellular graphs define their own connectivity, and disconnected cellular graphs are therefore likely to remain disconnected when reused in different graphs.

The alternative to obtaining diverse building blocks from some kind of initialisation method is to generate them during evolution. Diversity refers to the differences between members of a population. *Genotypic diversity* is the diversity among genomes in the population, whereas *phenotypic diversity* is the diversity among fitness values in the population. Maintaining a diverse population is essential to the long-term success of any evolutionary system, as this implies the continued global exploration of productive regions of the configuration space instead of local search trapped by local optima. An effective method for facilitating diversity should therefore not just eliminate the need for initialisation, but also improve the search process throughout all generations. This chapter will address this issue in the unique framework of graph grammar evolution. The principal emphasis will be on phenotypic diversity, as genotypic diversity is difficult to estimate for a graph grammar – more on this below.

6.1 Diversity Objectives

Several methods have been proposed to improve diversity and combat premature convergence in evolutionary algorithms, including spatial separation, fitness sharing, and crowding. *Spatial separation* is a major topic in its own right and therefore reserved for section 6.2, but the other methods are discussed here.

Fitness sharing involves penalising the fitness of a solution if it is similar to other population members (Goldberg and Richardson, 1987). Rosca (1995) used fitness values to define an entropy and free energy measure for phenotypic diversity. High entropy reveals the presence of many unique fitness values, with the population evenly distributed over these. Bersano-Begey (1997) tracked the number of solutions that solved specific fitness cases (i.e. training cases), which was used to discover and promote more distinctive solutions. Fitness sharing among different fitness cases has also been applied to GP, reducing the occurrence of premature convergence (McKay, 2000; McKay and Abbass, 2001).

Crowding forces new individuals to replace those that are genotypically similar (De Jong, 1975). This is closely related to niching (previously discussed in section 4.3.3), which is applied in MOEAs such as NSGA-II (Deb et al., 2000). MOEAs select for solutions that represent Pareto-optimal trade-offs between multiple objectives. Niching strategies can evenly spread solutions across the Pareto-boundary (Deb et al., 2003), but this is not guaranteed to lead to diverse building blocks and can indeed be detrimental to the scalability of the algorithm (Sastry et al., 2005). While our purpose for using MOEAs is size control, selecting against size is known to lead to premature convergence on small solutions (De Jong and Pollack, 2003).

MOEAs can facilitate diversity if we add diversity as another objective; but for this we need a measure of diversity. A common method for determining genotypic diversity is to compute an edit distance based on string matching, which can also be used to determine distance between GP trees (O'Reilly, 1997). De Jong et al. (2001) achieved both smaller and more diverse trees by using tree distance as a genotypic diversity objective in the multi-objective optimisation of n-parity problems. Bui et al. (2005) explored several diversity objectives, including mean and minimum genotype distances; the latter was also implemented by Toffolo and Benini (2003), and competitive results were achieved in all instances. The principal drawback of genotype distance measures is their limited applicability to graph grammars, as the extensive neutrality intrinsic to graph grammars would allow these to improve distance while remaining isomorphic. Since genotypic and phenotypic diversity are closely intertwined – a decrease in genotypic diversity will often cause a decrease in phenotypic diversity – a possible solution is to employ a phenotypic diversity objective instead. We suggest a number of simple phenotypic diversity measures in the following section.

6.1.1 Measures of Phenotypic Diversity

The error returned by the objective function is the most available phenotypic trait of a solution and hence a solid basis for measuring phenotypic diversity. To reduce any bias attributable to the nature of the specific objective function used, the solutions can be ranked against each other on this function; distances are then computed as differences of ranks. We suggest six different rank-based distance measures here:

The mean distance of solution i is the absolute difference between ranks,

$$D_i = \frac{\sum_{j=0}^{N} |R_i - R_j|}{N} \tag{6.1}$$

where N is the number of other solutions. Since it is often easier to attain worst rank than best rank, using this measure encourages poor performance. A measure less biased towards poor performance is to compare whether two solutions i and j show non-identical performance,

$$S_{ij} = \begin{cases} 1 & \text{if } R_i \neq R_j \\ 0 & \text{otherwise.} \end{cases}$$

The diversity of solution i can be defined as the proportion of solutions that are different in performance,

$$D_i = \frac{\sum_{j=0}^{N} S_{ij}}{N} \tag{6.2}$$

This 'difference' measure is logarithmically related to the phenotypic entropy of the solution:

$$H(i) = -\log(1 - \frac{\sum_{j=0}^{N} S_{ij}}{N}), \tag{6.3}$$

but since the diversity objective is ranked as well, we can use the simpler difference measure while obtaining the exact same effect.

The above approach encourages solutions to be different, but no worse than necessary to achieve a difference. For numeric optimisation, this would obviously lead to a population of very similar solutions. In the case of graph optimisation similar performance can be attained by very different graphs, so this is arguably less of a concern. However, solutions with equal mean performance can still be different, and the measures presented so far do not recognise this. Distinguishing these solutions without comparing their genotypes is only feasible if there are multiple fitness cases that can be compared separately. Then the mean rank distance can be averaged across each case c,

$$D_i = \frac{\sum_{c=0}^{C} \sum_{j=0}^{N} |R_{ci} - R_{cj}|}{C \times N}, \tag{6.4}$$

where C is the number of fitness cases. Two solutions perform non-identically if

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

so that diversity may again be defined as the proportion of non-identical solutions,

$$D_i = \frac{\sum_{j=0}^{N} S_{ij}}{C \times N}. \tag{6.5}$$

Pareto-dominance across all fitness cases can also be established, so that dominated solutions can be excluded from the above measures. Thus, satisfying fewer fitness cases is only regarded as diversity if these fitness cases are different. The rank distance of a solution i is its distance to another solution that does not dominate it,

$$S_{ij} = \begin{cases} \left|\sum_{c=0}^{C} |R_{ci} - R_{cj}\right| & \text{if } j \text{ does not } \in\text{-dominate } i \\ 0 & \text{otherwise,} \end{cases}$$

and the proposed diversity measure is the mean of these,

$$D_{ij} = \frac{\sum_{j=0}^{N} S_{ij}}{C \times P_i}, \tag{6.6}$$

where P_i is the number of solutions that do not dominate i. Within this dominance framework, two solutions can also be defined to differ if

$$S_{ij} = \begin{cases} 1 & \text{if } \sum_{c=0}^{C} |R_{ci} - R_{cj}| = 0 \text{ and } j \text{ does not } \in\text{-dominate } i \\ 0 & \text{otherwise,} \end{cases}$$

and diversity can be the proportion of non-identical solutions that do not dominate,

$$D_{ij} = \frac{\sum_{j=0}^{N} S_{ij}}{C \times P_i}. \tag{6.7}$$

For comparison, using each of the fitness cases as a separate objective in the MOEA will also be evaluated. Instead of a single performance objective, there is now a performance objective for each separate fitness case. Solutions thereby remain non-dominated as long as they are superior to all other solutions in at least one fitness case. In this instance, ensuring a diversity of Pareto efficient solutions becomes the principal responsibility of the niching mechanism. It has to spread the solutions along a Pareto frontier in $n+1$ dimensions, where n is the number of fitness cases (i.e. performance objectives), and, as by default, there is also a single size objective.

6.1.2 Age as a Diversity Heuristic

Over many generations and large populations the effort of calculating entropy or distance for each solution may have a noticeable impact on optimisation speed. A much simpler objective is based on assigning a time stamp to each solution, depending on when the solution was first evaluated (Luerssen, 2005a). In contrast to a previous study by Abbass and Deb (2003) that also employed this idea, we favour newer solutions over older solutions, thus mirroring the biological aging process and health benefits of youth. A significant benefit to diversity here is that any changes to offspring that are otherwise neutral will survive into the next generation because of the youth of the offspring. Two different time stamps are suggested for maximisation: one is the birth generation of the solution (we mark this as the solution *age* on tables and figures), the other is the *sequence* number of the solution, which is incremented for every solution and hence unique. The main effective difference between using age or sequence is that with sequence the newest solution is never dominated, and older solutions are more strongly dominated than with age, because they may be dominated by otherwise identical solutions of the same generation. Note that the order in which solutions produce offspring (and hence their sequence order) is randomised in each generation.

6.1.3 Method

The proposed diversity measures are evaluated on the Binomial-3 Regression, the RBS circuit, and the CNT design. The Backpropagation MLP task is replaced by the task of balancing two poles on a cart controlled by a neural network that we evolve (as described in section 4.6). This change was deemed appropriate because the MLP design task involved a population of one, leaving no room for phenotypic diversity. Moreover, the outputs, and hence diversity, of the network is affected by learning, which greatly complicates the relationship between evolution and diversity.

Solutions to the first three tasks are evolved using the same method as described in the last chapter, with a population of 20 graphs and soft matching, target offsets, default modularity, and pseudographs. We address the pole balancing in the same way, but the terminals are constituted by log-sigmoid neurons and real-valued weight vectors are initialised randomly with a standard Gaussian distribution ($\mu = 0$, $\sigma = 2$). However, unlike with the CNT design, we will not just rely on randomisation for generating the weight values. New weight vectors are determined by adding the weighted difference vector between two weight vectors (of different, randomly chosen neurons) to a third vector, a method based on Differential Evolution (Price, 1999) with parameter $F = 0.2$ and a crossover probability of 0.9. A new mutation is added to the set of allowed mutations that generates such a new weight vector for a randomly chosen neuron; it is applied during evolution so that one third of all mutations are weight changes.

Two additional problems from the domains of symbolic regression and circuit design are also investigated: the regression of the sixth-order polynomial

$$f(x) = x^6 - 2x^4 - x^2 \tag{6.8}$$

with fitness cases constituted by 21 equidistant points generated by this function over the interval of $x = [-1, 1]$; and the design of a 6-bit Boolean multiplexer, as presented in section 4.6.3. Results for these two problems have already been published previously in Luerssen (2005b) and will be repeated here for further comparison and interpretation. A new addition are the results for the age and sequence measures of diversity, which are unique for this experiment. The method is otherwise detailed in the aforementioned paper and differs in a few details from the method used for the other tasks: the population size is 10 for the regression, and 30 for the multiplexer; the *modify* mutation operation is applied at equal probability as the other operations; and each run ends after 5000 generations instead of 1000. The paper also evaluates pole balancing over 100000 cycles with these parameters (but a population size of 50); however, due to performance and consistency issues with respect to the next section, the pole balancing was reevaluated here with a population of 20 members simulated for 1000 cycles. See section 7.2 in the next chapter for additional 100000 cycle runs.

We optimise the network designs with a MOEA applied to three objectives: the function error, the network size, and a diversity measure. On all problems except the pole balancing, 9 different diversity measures are evaluated. The first three measures are based on entropy (i.e. 'difference' in our implementation), including mean rank entropy (equation 6.2, marked *Entropy* on the figures and in tables), mean rank entropy across fitness cases (equation 6.5, marked *Case Entropy*), and the mean rank entropy across fitness cases for non-dominated solutions only (equation 6.7, marked *ND Entropy*). The next three measures are based on distance and include the mean rank distance (equation 6.1, marked *Distance*), the mean rank distance across fitness cases (equation 6.4, marked *Case Distance*), and the mean rank distance across fitness cases for non-dominated solutions only (equation 6.6, marked *ND Distance*). Finally, we also use each fitness case as a separate performance objective in Pareto optimisation (marked *Case Pareto*), and the age of the solution (marked *Age*) and the sequence number of the solution (marked *Sequence*) as alternative diversity measures. For pole balancing, only the mean rank entropy and distance measures, and the age and sequence measures are evaluated, due to the absence of multiple fitness cases.

6.1.4 Results & Discussion

The performance outcomes of using the different diversity measures are shown in figures 6.1 and 6.2; the success rates are listed in tables A.1 to A.7 in Appendix A. On the Binomial-3 regression, the best results are obtained with the simple entropy measure, which gives a mean error of 0.0155 for the best solutions. With a diversity measure, the mean error is 0.054, which appears a lot worse, yet the

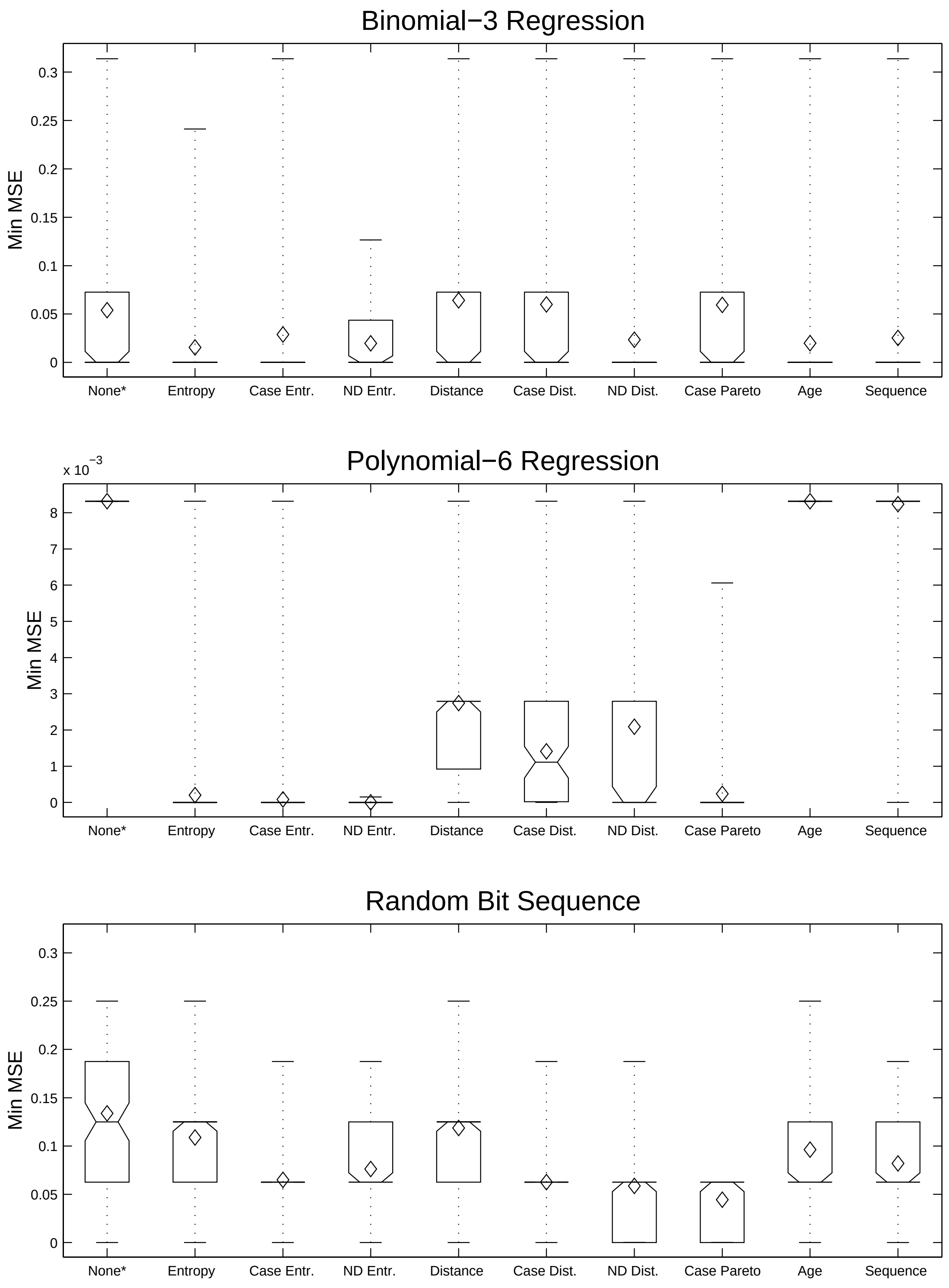


Figure 6.1: Minimum MSE box plot of the default configuration against 9 different diversity measures involving additional objectives for optimisation. The asterisk (*) denotes the default configuration, i.e. the configuration also used in other experiments.

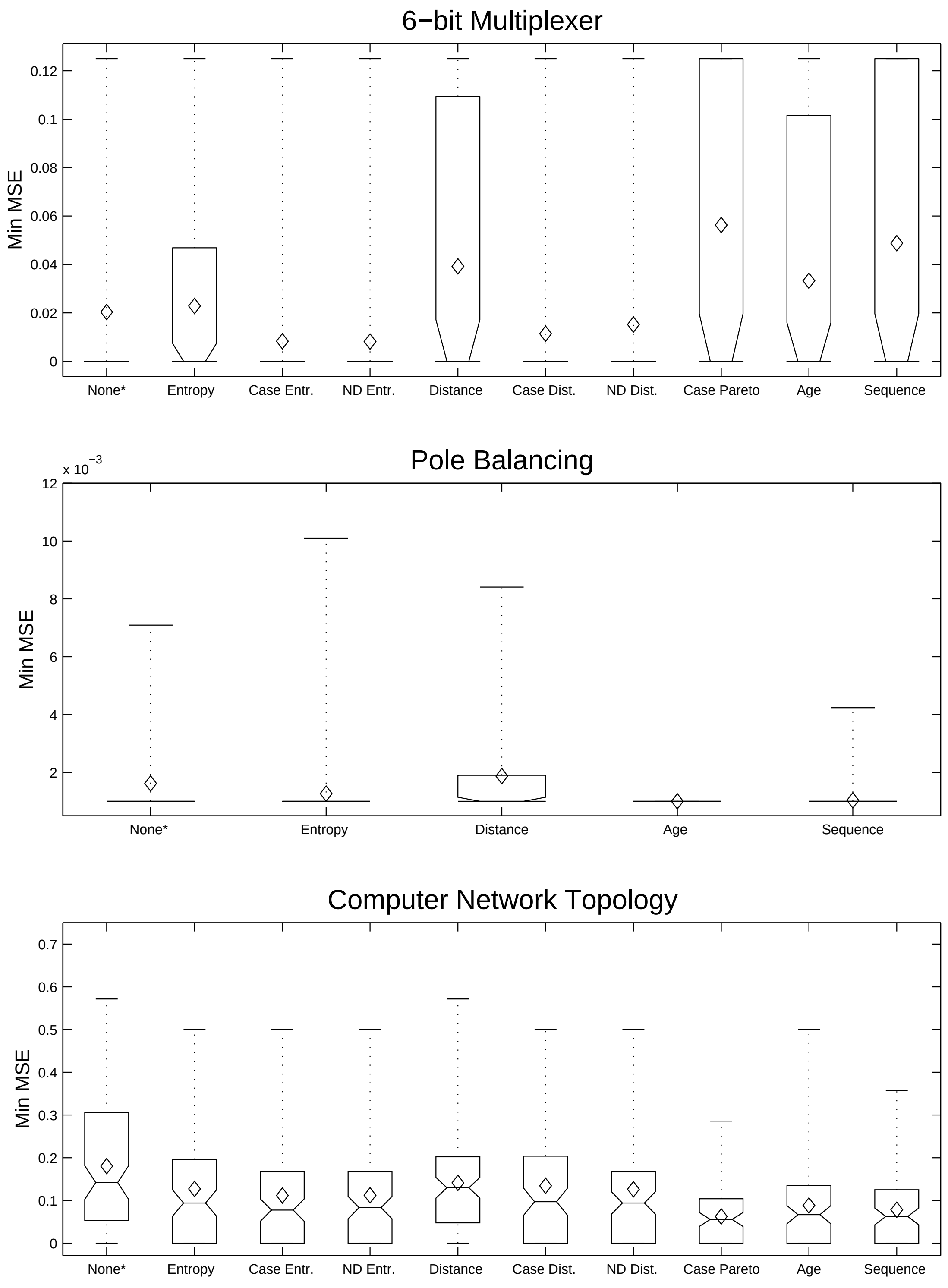


Figure 6.2: Minimum MSE box plot of the default configuration against 9 different diversity measures involving additional objectives for optimisation. The asterisk (*) denotes the default configuration, i.e. the configuration also used in other experiments.

difference is only borderline significant ($p < 0.03$). The other entropy measures, the non-dominated distance measure, and the age/sequence measures show lesser improvements that are not significant. We can attribute this to the very high standard deviations in the results, which cause uncertainty about the visible performance increases. Conversely, a decline in performance is observed when using either simple distance or fitness case distance as a diversity objective, or using fitness case Pareto performance instead. The success rates for both the simple distance measure and fitness case Pareto are 57%, which is significantly worse than the 71% without a diversity measure ($p < 0.005$) and also significantly worse on the less sensitive non-parametric test than any of the entropy measures (all below $p < 0.003$). Using fitness case distance or non-dominated distance leads to some improvement, with non-dominated distance having a significantly better MSE than either simple distance ($p < 0.002$) or fitness case distance ($p < 0.05$).

While the benefits of diversity are not clear-cut with the Binomial-3 regression, they are very prominent with the 6th-order polynomial. No convergence at all happens on the regression of the latter unless we explicitly facilitate diversity. In contrast, a 100% success rate is attained with the non-dominated entropy measure; a perfect solution is found after 6058 evaluations on average. Entropy measures are all very effective, and there are no significant differences between these. Only if we do a Z-test on success rates do we find that applying entropy to fitness cases is significantly superior to the simple entropy measure ($p < 0.003$). Selecting for distance is overall significantly less effective at obtaining good MSEs than selecting for entropy ($p < 10^{-10}$), but the distance measure improves significantly if applied to fitness cases ($p < 0.002$), especially if only non-dominated solutions are considered ($p < 0.00003$). Overall, the use of entropy objectives, distance objectives, or fitness case Pareto leads to significantly better outcomes than without any diversity measure. Unlike on the other problem tasks, age and sequence objectives have no notable impact on performance here.

RBS circuit evolution clearly benefits from diversity as well. All diversity measures improve performance, and this improvement is significant except for the simple distance objective. The distance objective is significantly better on the MSE if applied to fitness cases than otherwise ($p < 4 \times 10^{-13}$), but solutions obtained by use of any distance measure are also very large in size, typically more than 20 times of what is obtained otherwise. The required increase in evaluation time does not appear justifiable in practice compared to the other measures. The best performance, and also without the degree of bloat associated with the distance objectives, is observed with the fitness case Pareto approach. Although we would expect the Pareto frontier to become far too large to be sampled appropriately in this way, the results tell a different story and are significantly superior to those of all other measures.

The benefits of facilitating diversity are less pronounced with the 6-multiplexer problem. 81% of runs already produce an optimal solution in the absence of a diversity measure, which increases to a maximum of 91% when using the non-dominated entropy measure ($p = 0.01$ on the success rate, $p < 0.04$ on the non-parametric test). Smaller improvements and in many instances reductions in performance are produced by the other diversity measures. Indeed, the distance measure produces significantly worse MSEs than without any diversity measure ($p < 0.006$), but taking fitness cases into account improves this significantly ($p < 0.00006$), which is also true when applied to the entropy measure ($p < 0.004$). Likewise, using age or sequence objectives has a negative impact on performance, which is significant for the sequence objective ($p < 0.0006$). The lack of strong results is surprising given that performance benefits of diversity are quite large with GP on this problem (McKay, 2000).

Only a single fitness case is used for pole balancing, so none of the fitness case-based diversity measures can be applied. MSEs are generally low, but not all solutions manage to balance the pole for the entire cycle sequence. Surprisingly, the age measure produces a 100% success rate (as compared to 82% without), and the sequence measure is close with 98%, which is significantly better than all other evaluated diversity measures (no comparison exceeds $p < 10^{-7}$ on the success rate, $p < 0.03$ on the non-parametric test). The entropy measure also leads to an improvement, but this is not significant, whereas the distance measure (with only 71% success rate) is significantly worse than every other measure on the success rate ($p < 0.02$).

The CNT design problem benefits significantly from every diversity measure except the simple distance objective, which had a negative, but not significant, influence on performance. The best performance is produced by the fitness case Pareto approach, with an MSE of 0.0627 (compared to 0.1801 without diversity), but the solutions were typically twice as large as otherwise. The performance, however, is significantly better than for any other diversity measure except the age and sequence objectives, which also performed well, with the highest success rate of 42% provided by the use of the sequence objective (as compared to 23% without diversity).

In conclusion, using phenotypic difference (i.e. entropy) as a diversity measure is generally quite effective, particularly if differences between fitness cases are taken into account. Mean distance measures are less effective; performance reductions are observed with the simple phenotypic distance, but computing distance over fitness cases produces better results. Solutions arising from these measures are frequently much larger than otherwise, with a correspondingly negative impact on evaluation time. Evaluation on the chosen problem tasks is computationally far more costly than any other aspect of evolution, including the diversity estimation. Since the preservation of diversity may have a notable impact on solution size, however, the choice of diversity measure is still critical in determining the mean computation time. Selecting for age or sequence is a particularly efficient strategy in this regard, as it is not only exceptionally simple, but generally improves performance without significant size increases, although it clearly also fails at dislodging solutions out of some local optima, such as with the 6th-order polynomial regression. The use of multiple performance objectives for Pareto optimisation is a mixed bag, with some problems benefiting, others not, and solution bloat being a frequent problem as well.

6.2 Island Models

Separating individuals spatially from each other qualitatively changes evolutionary behaviour by slowing down the flow of information between subsets of the population, thus encouraging their diversity by allowing them to evolve more independently of other parts of the population. A classification of the approaches for separating individuals includes two main models: the coarse-grained *island* model (Martin et al., 1997) and the fine-grained *neighbourhood*, or *cellular*, model (Pettey, 1997). These models were originally intended for distributed, parallel implementations of evolutionary algorithms, but they often not only produce time savings, but better solutions, too (Whitley et al., 1999).

In the island model the population is explicitly divided into several smaller subpopulations, called *demes*. An evolutionary algorithm evolves each deme independently, but, periodically, information is exchanged by migrating individuals from one deme to another. Based on concepts from population genetics, the idea here is that random genetic drift will cause each deme to explore different regions of the search space, and migration will communicate important discoveries among the demes. The migration rate is an important parameter here, as high rates cause global mixing, reducing the isolation advantage, whereas a low rate may lead to each deme converging prematurely.

The neighbourhood model defines a cellular space for the population by associating each individual with a cell of the grid. Different neighbourhoods can be defined for the cells (such as presented in section 2.2.1). Fitness evaluation is done simultaneously for all the individuals, but individuals can only interact with their direct neighbours when it comes to mating, reproduction, and selection. Once again, the idea is that information slowly diffuses across the space forming semi-isolated subgroups of individuals, each exploring different regions of the search space.

6.2.1 Graph Grammar Islands

In G/GRADE each solution consists of productions that are part of a global gene pool; new solutions are created by randomly selecting from this pool, so a production may become part of several solutions. Applying an island model to G/GRADE entails a partitioning of this gene pool. Because of shared use, however, moving a solution from one deme to another cannot be done by moving all of its associated productions – they need to be copied. Copying is computationally expensive in this framework and since a conventional island model such as this has been explored time and again, a G/GRADE-specific alternative is investigated instead. The need for copies comes from the implicit requirement that productions can only refer to productions on the same deme. Without this requirement, any production can be moved to any deme; if it is a starting production, then the solution itself is considered to have moved there. Solutions compete only against those of the same deme, so the island model would be applied to the selection process alone. There would be no effect on the choice of productions for new solutions, i.e. the mating aspect of the island model.

This latter aspect may be included by making production choices dependent on the deme. Production choice matters only in two instances: when we choose a production for mutation, and when we add a production to another production as part of the insert mutation. Choosing a production for mutation by a deme that matches that of the solution does not achieve any sort of localisation; it just modifies the effective mutation rate for each production. Consequently, we only explored choosing productions for insertion according to deme, which we will refer to as *local mating*. The deme to match in this case is the deme of the production that is being mutated, as this causes localisation along the call chain between productions – neighbouring productions will tend be on the same deme and so will alternative choices for these productions, which focuses the search

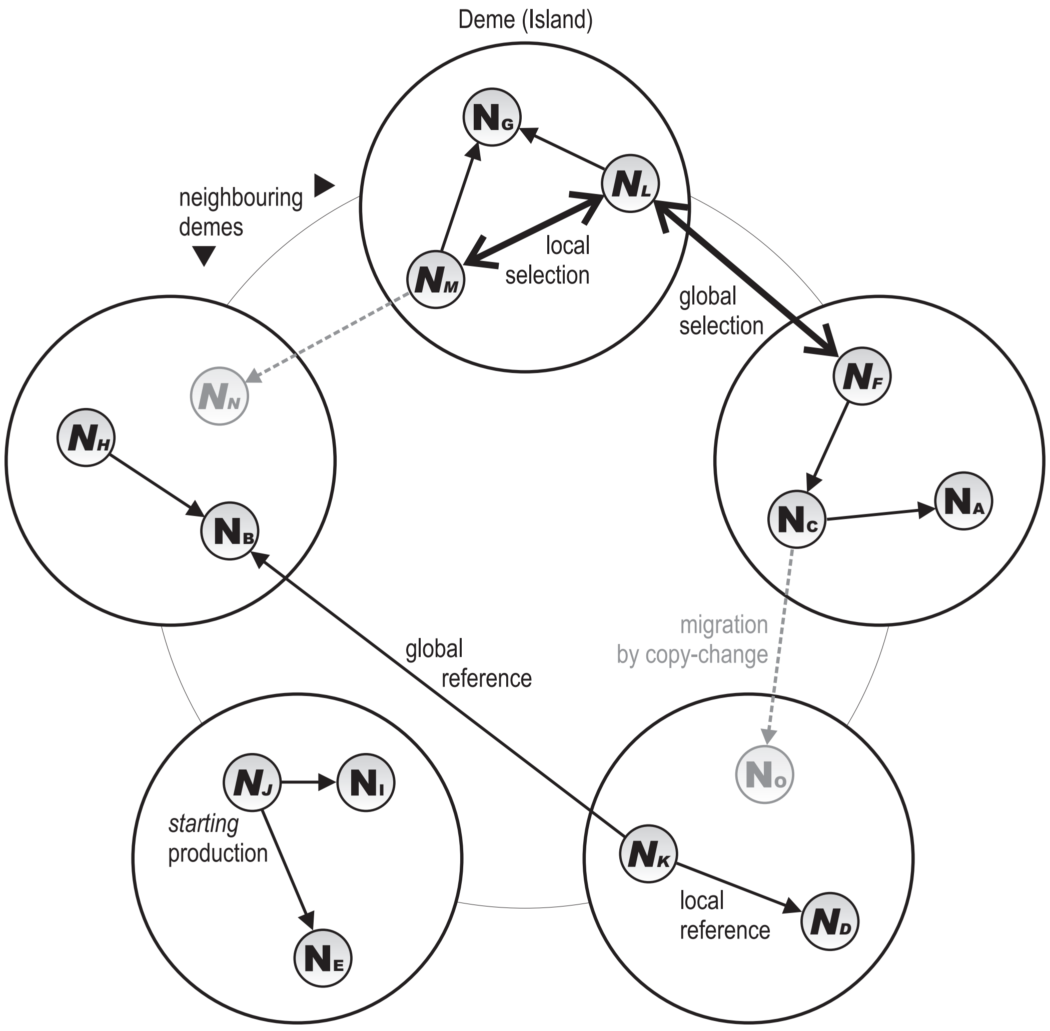


Figure 6.3: The island model as applied to graph grammar evolution: islands are arranged in a ring, with offspring being allowed to move to neighbouring islands.

along these choices. The various interactions of productions across islands are visualised in figure 6.3.

One confounding factor is how multi-objective selection occurs in the context of multiple islands. Since solutions only need to compete against other solutions on the same island, the probability of being dominated (and hence the average degree of dominatedness) increases with the number of solutions on that island. This balances the distribution of the population across the islands, because the chance of survival is higher for solutions from less populated islands. Globally, however, each solution has fewer solutions to compete against, so there are many more nondominated solutions in the whole population than without an island model. Thus, the number of nondominated solutions is more likely to exceed the global size limit for the population. To maintain this limit we must rely strongly on multi-objective niching estimation (see section 4.3.3), yet niches on individual islands tell us little about global niches. For example, each island might hold a copy of an empty solution, because it is a Pareto efficient solution, but this would contribute nothing to the search process. To avoid this kind of predicament but still retain the diversity benefits of the island model, we evaluate the niching globally across all islands, so that being on a different island only affects domination, not niching.

6.2.2 Method

The population size in our previous experiments has been 20 for the most part; this is very small compared to what you might expect in an island model. Using a Moore- or Von Neumann-neighbourhood with 4 or 8 neighbours, respectively, would necessitate at least that many demes (and thus very small deme populations) and allow for very easy transitions between any pair of demes. Consequently, our model here is based on a 2-neighbour cellular space forming a ring, where transition is possible from any island to any of its two neighbours (see figure 6.3).

We compare the island model with local selection but a fully global production pool against the island model with local selection and local mating. The number of demes is fixed at 5, and there is no limit to how many solutions or productions may exist in a deme. The population starts, as previously, with a single empty starting production in the first deme. The choice of mutations has been expanded so that productions (and starting productions, hence solutions) can transit randomly to one of the two neighbouring demes. This *transition* mutation is applied at the same probability as any *insert* or *remove* mutation. The total number of mutations remains identical, so other mutations are applied at a somewhat lower effective probability than before. However, this ought to worsen performance according to the trends observed in section 7.1; if performance nevertheless improves it should be safe to say that this is due to the island model.

These models are tested against running 5 populations (of 4 or 16 members) in complete seclusion to each other to establish whether there is any benefit to transitions at all. Each deme starts with a single empty production, and productions are exclusive to each deme. Finally, the effect of different deme numbers is also evaluated, but only on the model without local mating. Rings with 2, 5, and a more fine-grained 20 demes are used.

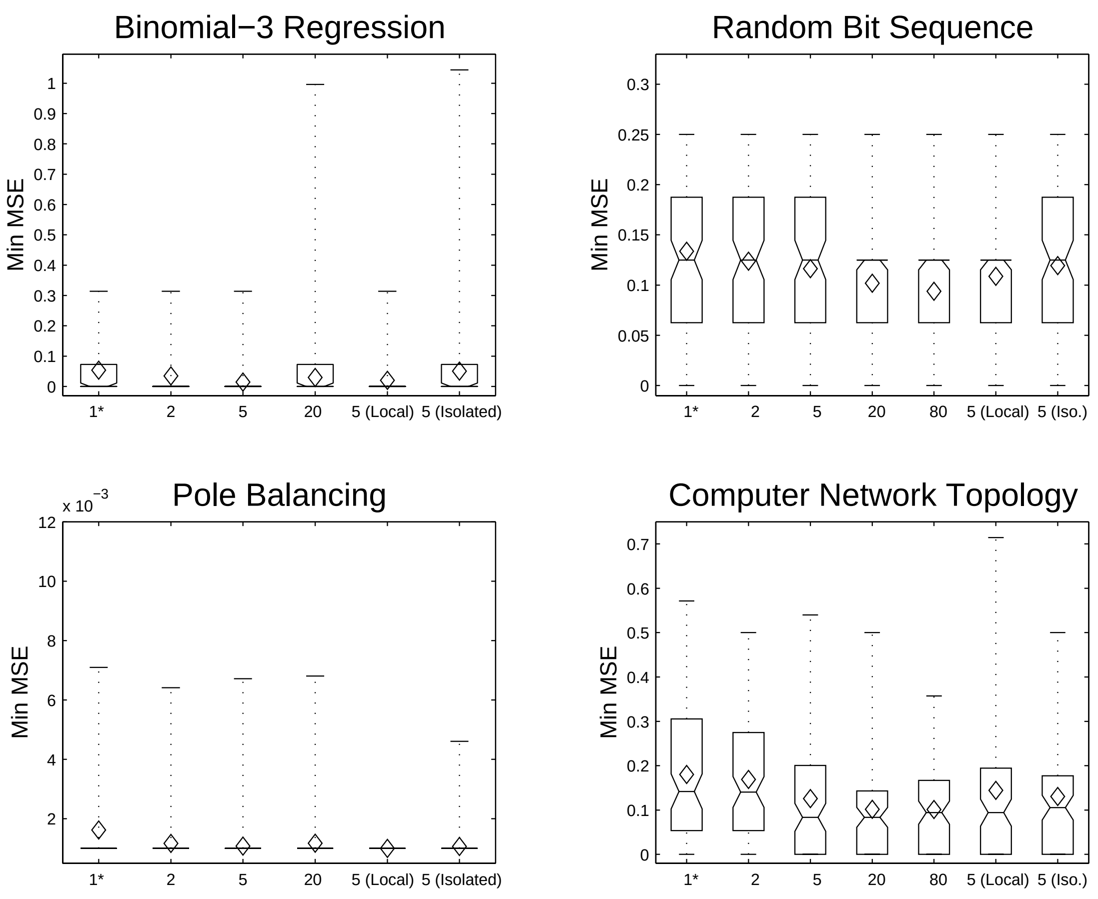


Figure 6.4: Minimum MSE box plot on different numbers of islands and constraints on global interactions: with *Local*, only local productions can be added to a network; with *Isolated*, no migration between islands is allowed.

6.2.3 Results & Discussion

Dividing the population into any configuration of islands leads to an improved MSE on every problem – there are no instances in which the performance is actually diminished. The improvements are not always significant, however, and the number of islands appears to matter. For the Binomial-3 regression and the pole balancing, the best results are obtained with 5 islands (with a significance of $p < 0.009$ and $p < 0.0002$, respectively, against the single island). On the CNT design, the success rate of 35% is also best with 5 islands ($p < 0.02$), but the results distribution continues to become better with more islands ($p < 0.003$ for 80 islands). This trend is most visible with the RBS circuit, where the optimum corresponds to the maximum number of islands, both with respect to success rate (13% against 1%, $p < 0.0004$) and the mean MSE (0.09 against 0.13, $p < 0.000005$).

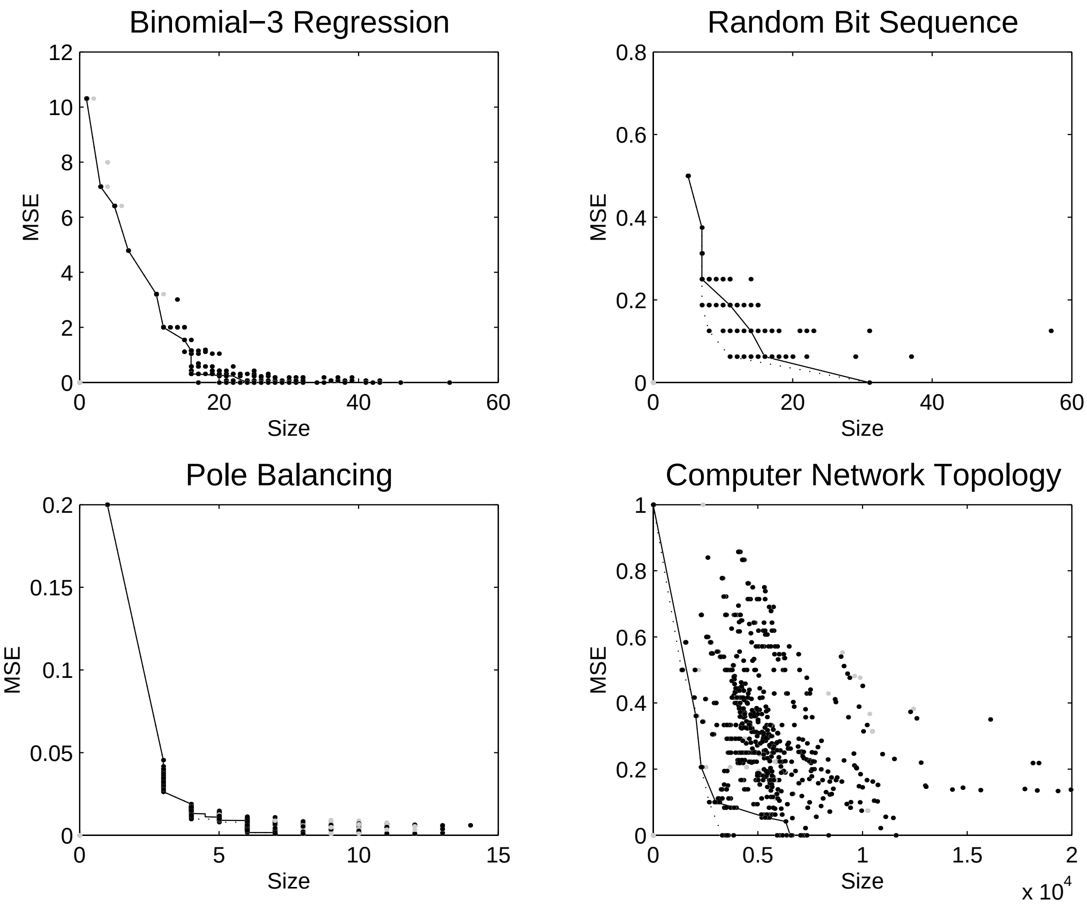


Figure 6.5: Final generation solutions for the default experimental configuration plotted against performance and size. Black dots are nondominated solutions, grey dots are dominated solution. Additionally, the minimum Pareto frontier (dotted line) and median Pareto frontier (solid line) are superimposed. Note that if the minimum frontier is not visible, it is actually identical to the median frontier; hence, the more dotted line is visible, the less effective the evolutionary search at determining the best possible frontier.

Choosing productions locally from an island rather than from a global repository appears to make no significant different on the tested 5 island configuration. Since networks that migrate from one island to another can still refer to the original, but now remote, productions, the practical differences to a fully global production repository are rather minor, so this is perhaps not a surprise. On the other hand, evolving populations on 5 isolated islands (which is equivalent to running G/GRADE 5 times with $1/5^{th}$ of the population) leads to worse performance than with the standard island model. Migration between islands is evidently essential for obtaining performance benefits from the island model.

Besides the improved solutions that can be obtained from the island model,

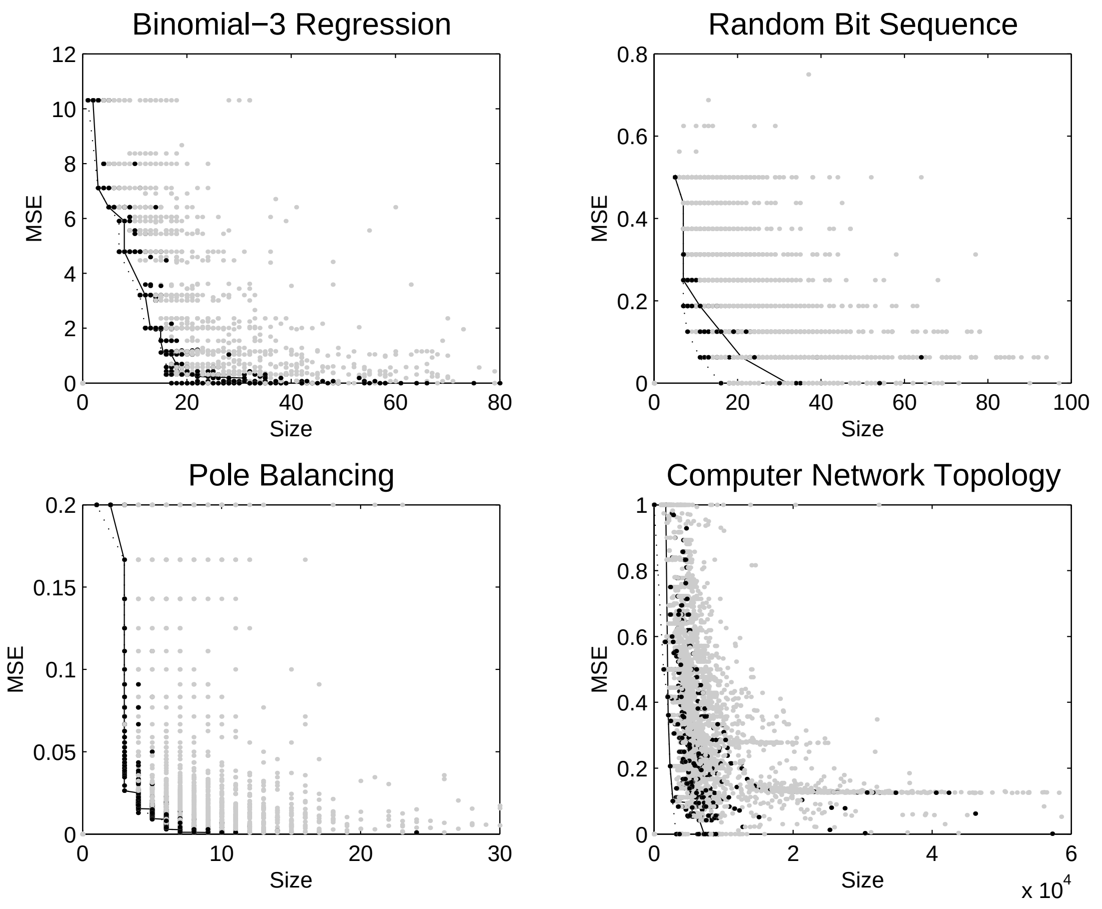


Figure 6.6: Pareto frontier for final generation solutions using the 5 island model, plotted against performance and size. See caption of figure 6.5 for legend.

the computational cost can be reduced as well, as fewer comparisons are required for evaluating the Pareto domination. However, the benefit is minimal in the implementation presented here, because we estimate the crowding across all islands. We are not aware of any papers that have tried to use MOEAs in conjunction with island models. Since dominance is evaluated for each island separately, a proper Pareto frontier indeed seems unlikely to arise. Figure 6.5 plots the minimum Pareto frontier (what is truly Pareto-optimal) and the median Pareto frontier (what is typically found), as well as each performance/size compromise in the final generations of all evolutionary runs using the default configuration. The Pareto frontier for the 5 island configuration is likewise shown in figure 6.6. Immediately visible with the island model is the much greater diversity of solutions that make up the population. Notably, however, the Pareto frontier remains intact, and compromises between size and performance can be determined even when using an island model. The median Pareto frontier is not as optimal as for the default configuration, but the differences appear surprisingly small.

6.3 Summary

Several means of facilitating phenotypic diversity were evaluated in this chapter, and it has become quite evident that diversity is a crucial requirement for the effective evolution of the graph grammar. Significant performance improvements were observed with most measures, including entropic diversity objectives, age/sequence objectives, and island models, but typically excluding distance-based diversity objectives, which tend to directly encourage poorly performing and bloated solutions. Potential speed improvements were noted (but not realised) for island models, and age/sequence objectives perform remarkably well (with some exceptions) despite their simplicity. The actual differences between the measures were otherwise quite minor, and without an investigation into the evolutionary dynamics affected by diversity no conclusion can be made on what constitutes an ideal measure. However, the last word on diversity is not spoken yet, as the next chapter will, among other things, address some of the interactions between diversity and size and reveal some additional, interesting results.

Chapter 7

Convergence Outcomes and Analysis

A population is said to have converged if the genotypic or phenotypic diversity of the population has collapsed to a small value with little chance of recovery. Sometimes this is a positive event, as evolutionary algorithms are meant to converge upon optima as they are discovered. However, in a multi-modal fitness landscape these optima are frequently only local optima. If the population fully converges upon a local optimum, we speak of *premature convergence*. This may lead to stasis, with no subsequent progress in optimisation, or the search will continue locally, but with minimal diversity and the associated risk of again being trapped by another local optimum. Evolutionary algorithms can avoid local optima by facilitating diversity as presented in the previous chapter, but also if the population is large enough or if the mutation rates are high enough so that the search resembles simulated annealing. This is rather undesirable in practice, however, as it slows convergence down. A proper compromise between fast convergence and reliable convergence is needed, and the aim of this chapter to establish one for graph grammar evolution. To begin with, we extend upon the previous chapter by investigating how evolution responds to changes in the rate of mutation and the size of the population. A model for accelerating convergence by adapting the search bias is then presented. We also explore the influence of the size objective on convergence and, ultimately, take a closer look at the graph grammars that result from convergence and how this compares to other systems.

7.1 Rates of Change

The force that drives convergence is the rate of change, which is a function of the size or number of mutations being applied to the productions that describe the networks. For any optimisation to occur, some of these mutations must bring about new networks that are fitter than their parents. If mutations were indeed mostly beneficial, raising the mutation rate would lead to faster convergence. Realistically, any problem worth addressing will have a search space composed of mostly unfit solutions, so random changes – and, above all, large random changes – are likely to lead away from the fit regions. Yet large mutations are requisite for overcoming local optima, and local optima are very common with cellular graph grammars, because a small change often has no effect on the network design unless another change is also applied. Since adding something to a cellular graph increases its size and thereby reduces fitness, we may encounter a circumstance where no single mutation can produce a fitter network than what currently exists.

The mutation rate used in our previous experiments is 1 mutation plus a 50% chance of additional mutations (i.e. a geometric probability distribution). Thus, a wide range of mutation numbers applies throughout evolution, with higher numbers less common. Nevertheless, as we clearly observed with the sixth-order polynomial regression in the last chapter, this alone is not a guarantee that a better solution is found. The purpose of this section is therefore to explore how graph grammar evolution responds to changes in the mutation rate. Although this is trivial in principle, there are some possible snags here. Until now it was conveniently assumed that only one production is mutated for each network. Raising the mutation rate for that one production may produce extensive changes to a network, because of the many other productions that can be called. As the productions involved must already exist in the population, however, this is analogous to doing multiple crossovers, and only very little change has been accomplished, namely to a single production. If two productions need to change in concert to describe a fitter network and neither is independently viable in the population, then the search will stall.

Clearly, we need to apply mutations to multiple productions simultaneously, but this is associated with several drawbacks. Firstly, multiple modified productions diminish some of the benefits of G/GRADE that were espoused in sections 4.2 and 4.3, because most of the genome will now be copied during reproduction. Furthermore, changes to a production may cause different productions to be expressed than before – potentially none of the productions originally chosen

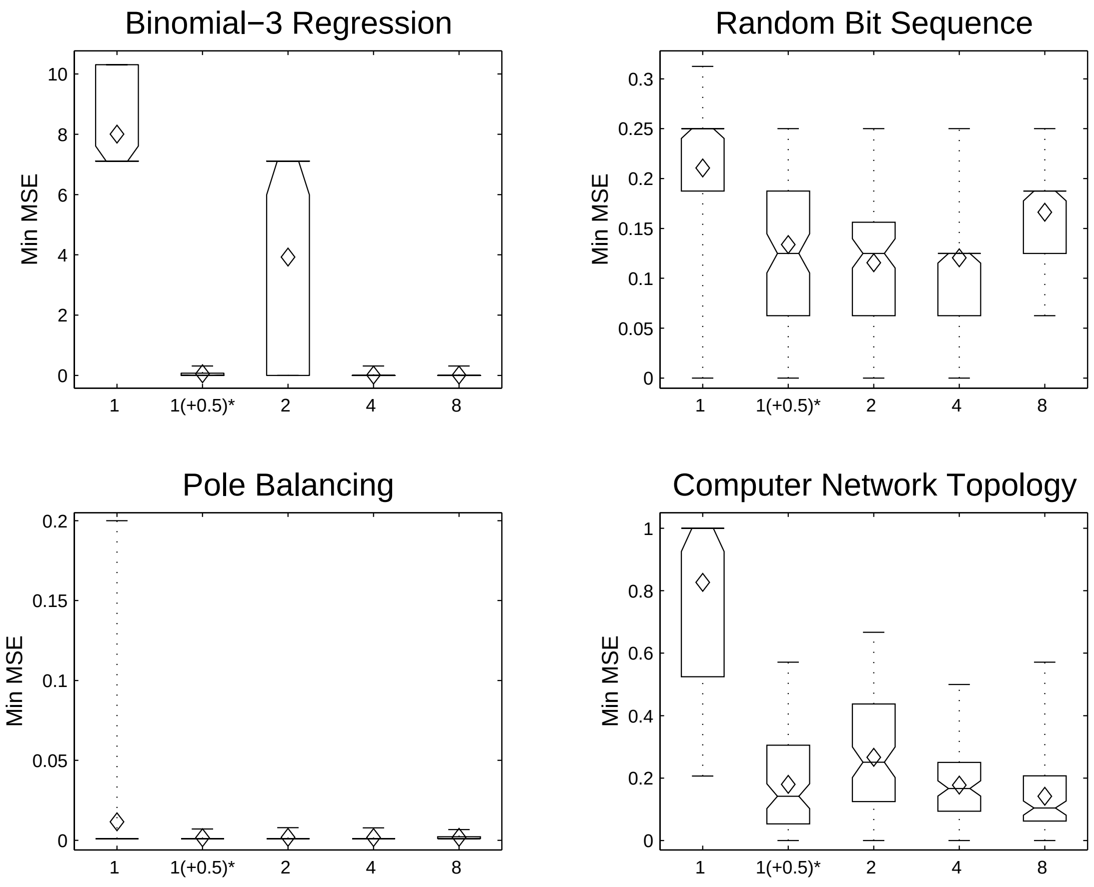


Figure 7.1: Performance box plot for different mutation rates for each mutated production. 1(+0.5) denotes a 50% probability of additional mutations beyond 1; the asterisk (*) again indicates the default value used in other experiments.

for mutation! An effective means of resolving this is to choose new productions for mutation whilst the network is being derived. Since the outcome of the new derivation is unknown, however, a fair (i.e. controlled) distribution of mutations among the expressed productions becomes intractable, and changes to a specific production P at a later stage of development cannot easily apply to earlier expressed instances of P. We therefore choose a much simpler alternative, which is to just discard all the mutations to the productions not expressed, although this limits how many mutations can be effectively applied at once.

7.1.1 Method

We compare the default mutation rate of 1 plus 50% probability of additional mutations, denoted 1(+0.5), applied to a single production against applying 1, 2, 4, or 8 mutations to the production. Additionally, the application of 1(+0.5)

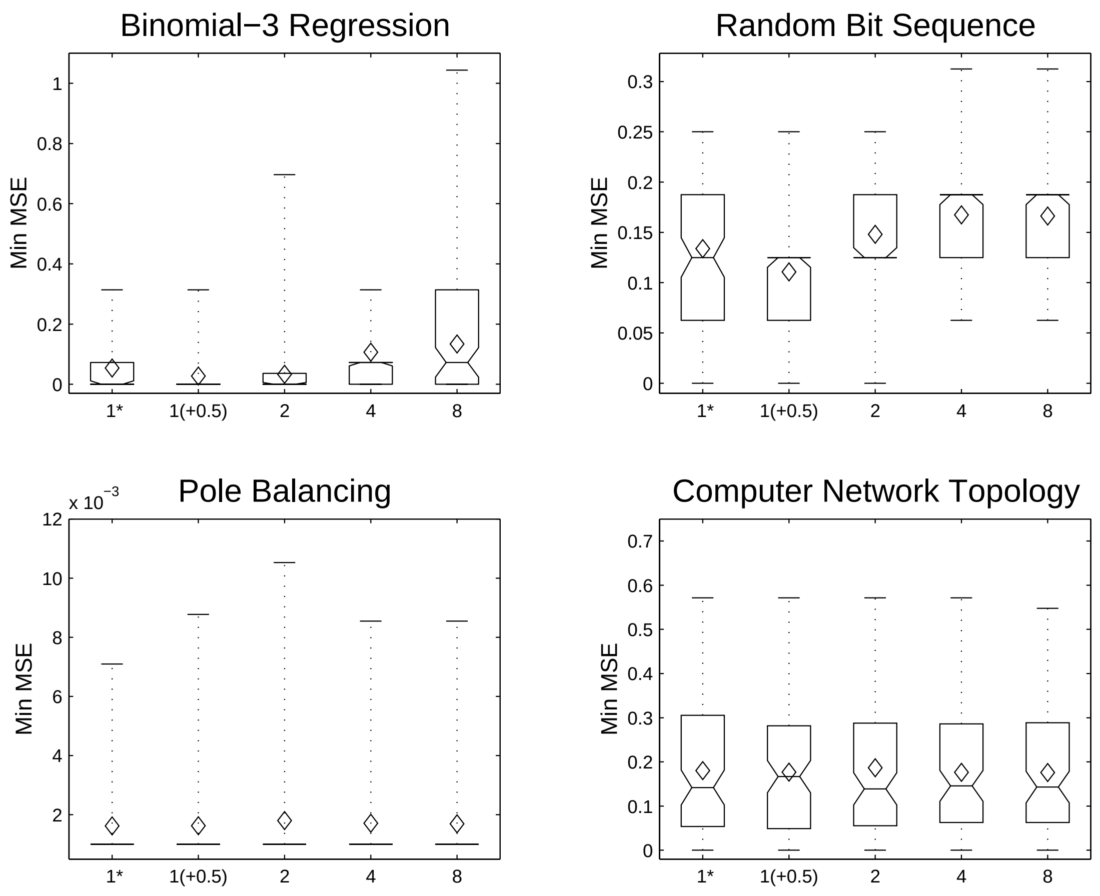


Figure 7.2: Performance box plot for different maximum numbers of mutated productions for each network. 1(+0.5) denotes a 50% probability of additional productions being chosen for mutation.

mutations to each of 1(+0.5), 2, 4, or 8 productions is evaluated. Only different productions are mutated, there are no duplicate selections; thus, if there are fewer productions in a network than are meant to be chosen, all productions will end up being mutated. The test problems are the Binomial-3 regression, the RBS circuit, the pole balancing, and the CNT design, optimised with G/GRADE and using the previously established default parameter choices (and applying no diversity objectives).

7.1.2 Results & Discussion

The results revealed in figures 7.1 and 7.2 indicate that there is little additional performance to be achieved by just scaling up the existing random mutation mechanism, although convergence suffers if only one mutation is applied at a time (and significantly so, $p < 10^{-13}$, on every problem except the pole balancing). Increasing the number of mutations on a single production leads to an improvement in performance. Although there is a fall-off observed for 8 mutations (except for the CNT design), the system seems to be quite robust to high mutation rates on single productions.

In contrast, raising mutation rates by targeting multiple productions generally causes worse results to arise, with the exception of mutating additional productions at a 50% chance. This actually seems to improve the outcome, although it is only significant for the RBS circuit ($p < 0.004$). The results reflect that an increase in the number of mutated productions should not have much of an impact, because mutated productions are less likely to call other mutated productions and some solutions may require less than 4 or 8 different productions.

In summary, for both the mutation rate per production and the number of productions being mutated it is advantageous to employ variable – rather than static – rates and numbers, respectively. This suggests that different networks in different contexts require different mutations and that there may hence be a better way of choosing mutations than randomly from a uniform or geometric distribution – more on this in section 7.4.

7.2 Evaluation Length

The previous chapters have uncovered various successful strategies for improving the outcome of network optimisation, but this outcome has always been with respect to a fixed number of evaluations, i.e. the population size multiplied by the total number of generations. Since the computational effort of evolution is a function of the number of evaluations, achieving global convergence with fewer evaluations is an important goal. However, merely reducing the population or the number of generations will typically just produce worse solutions, so our goal in this section is to determine the trade-off between these parameters. We will compare different population sizes against different lengths of evolution, but the total number of evaluations remains fixed. If the system behaves like a local search (i.e. gradient descent) algorithm, then a reduction in the population size should hence lead to a performance improvement.

In practice, the length of a single run may be less significant than the total number of evaluations, over all possible runs, required to find the optimal solution. As discussed previously, evolution of graph grammars can end in premature convergence, but it has been difficult to observe consistent stalling within just

1000 generations (cf. figures 7.14 to 7.17 in section 7.6.4 later in this chapter), so we are curious to see how evolution progresses, if at all, over further generations. The length of evolution will thus be increased by a factor of 50 (to 50000 generations). Since this also increases the time required for the experiment, we cannot compare each possible parameter configuration; only the default configuration and a combination of several effective extensions of the G/GRADE framework will be evaluated (see below).

7.2.1 Method

Recall that the default population size for the Binomial-3 regression and the pole balancing task is 20; for the RBS circuit and the CNT design it is 80. Here we will employ population sizes of 5, 20, and 80 for each problem task. The default number of generations is 1000, but is changed for the other evaluated population sizes, so that the total number of evaluations remains constant. Results are averaged over 100 runs, as previously.

For the extended runs, a generation limit of 50000 is applied, but to complete these runs in a reasonable amount of time, we average the results over only 30 runs. The default configuration is compared against an *enhanced* configuration, combining the following framework extensions introduced previously: using point offsets for nodes (see section 5.1), allowing for more than one production to be mutated (see section 7.1), and using fitness case entropy as a diversity objective (except for pole balancing, where in the absence of a potent diversity objective, we will use the 5 island model; see sections 6.1 and 6.2). Pole balancing will also be performed over 100000 cycles in this experiment, in order to produce results that are comparable to previous studies.

7.2.2 Results & Discussion

Figure 7.3 has performance box plots for the tested combinations of population size and run length. It seems a safe bet to employ a small population over many generations, but the performance differences are mostly not large enough to allow us to define a "correct" population size. Clearly, G/GRADE exploits sequential (local) and parallel (global) exploration of the search space similarly well. Significant differences are only observed on the pole balancing task, where the 80 network population performs significantly worse ($p < 7 \times 10^{-6}$); and the RBS circuit, where the default configuration used so far turns out to be the worst

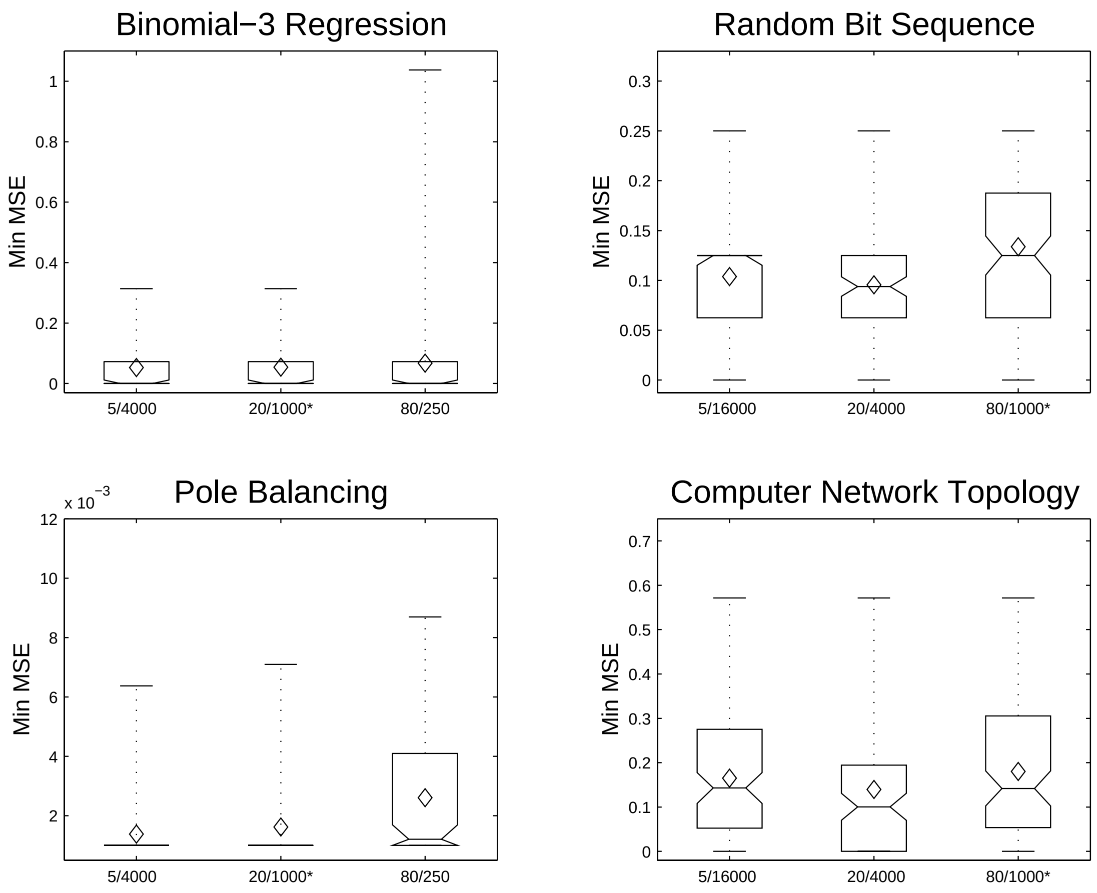


Figure 7.3: Performance box plot for the default population, marked with an asterisk (*), against evolving smaller and larger populations against correspondingly more or fewer generations (shown as *Population/Generations*).

($p < 0.0004$ for each comparison). Tables A.1 to A.7 also include results for decreasing or increasing population size without adjusting the number of generations, which, as expected, causes reductions and improvements in performance, respectively.

Figure 7.4 plots the performance of individual runs over 50000 generations. We observe in several runs of each problem that convergence indeed stalls. The use of the enhanced configuration greatly reduces this, leading to 100% success rates on all problems except the RBS circuit, which is constrained by the size objective – see section 7.5 for a discussion. The improved results are, at least over the first 1000 generations, clearly due to the diversity measure being applied, which performs insignificantly worse on its own.

Balancing poles over 100000 cycles is slightly more difficult than over 1000 cycles, with a success rate of 73% as compared to 82% over 1000 generations

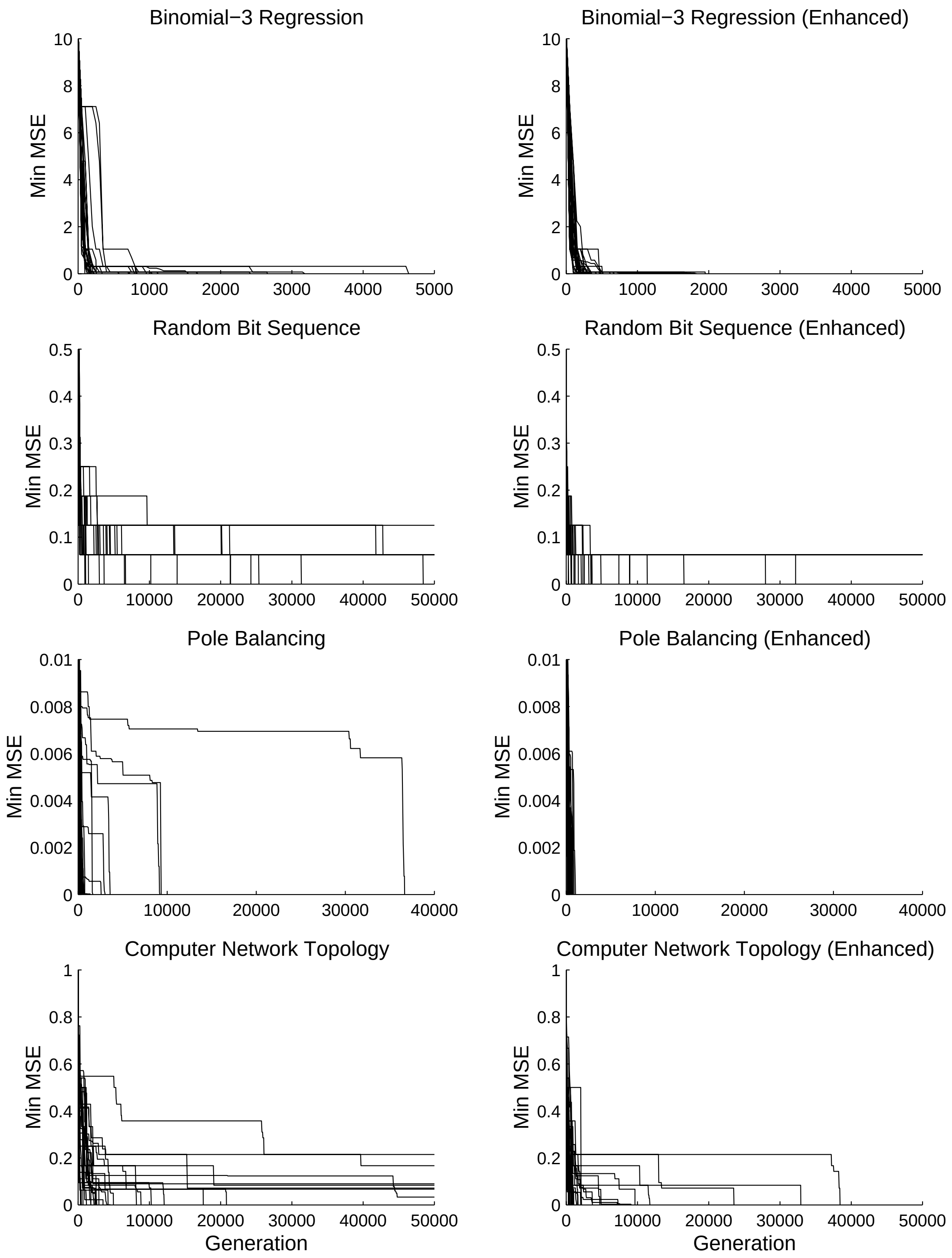


Figure 7.4: Each figure shows the performance of the lowest error solution for each of 30 runs over 50000 generations (or until all runs converge). Left-hand figures are from the default configuration of G/GRADE; right-hand figures are from a enhanced configuration that uses point offsets, multiple production mutation, and a fitness case entropy objective (or, for pole balancing, a 5 island model).

using the default configuration. With the enhanced configuration, this rises to 97%, which is competitive to the island model results obtained for 1000 cycles, so it appears unlikely that the results were greatly biased by previously simulating pole balancing over only 1000 cycles; at best, the benefits of various framework extensions were underestimated.

7.3 A Basic Convergence Proof

If certain optimisation runs fail to find the global optimum, we might ponder whether this is because G/GRADE is somehow constrained from finding the global optimum at all. However, a convergence guarantee can be established, with some conditions, for G/GRADE as for any GA; the following is inspired by a GA convergence proof by Hartl (1990). For a finite set of feasible solutions S and an associated error E,

$$F : S \rightarrow E \tag{7.1}$$

is the objective function. If $x \in S$ then the goal of optimisation is

$$F(x) \rightarrow min(E). \tag{7.2}$$

Let's define a neighborhood $N(x)$ of x according to the mapping $N : S \rightarrow S$, so that each $y \in N(x)$ is a neighbour of x. A locally optimal solution is an $x \in S$ such that

$$\forall y \in N(x) : F(x) \leq F(y) \tag{7.3}$$

and a globally optimal solution is an $x \in S$ such that

$$\forall y \in S : F(x) \leq F(y). \tag{7.4}$$

In G/GRADE we can mutate productions of a given solution x to obtain solution y from the neighborhood $N(x)$. This gives us a matrix R of transition probabilities such that

$$R(x, y) = P\{X_{t+1} = y | X_t = x\} \tag{7.5}$$

where X_t denotes a state of the system at generation t. A state y is reachable from state x with a non-zero probability if there exists $z_1, \ldots, z_s \in S$ such that $z_1 \in N(x), z_2 \in N(z_1), \ldots, y \in N(z_s)$.

Let $m \in S^p$ be the combined vector of p elements of S, where p is the size of the population. A Markov chain now describes the sequence of populations that would arise from applying G/GRADE. To allow convergence, the global optimum state m_0, must be absorbing, so at least one best population member must be selected during transition. Likewise, to ensure that m_0 is reachable from all m, at least one newly mutated member must be selected randomly, so that every state $m \neq m_0$ is transient. In G/GRADE, this is only guaranteed if we also employ the sequence diversity objective, so this will be assumed here. It is known that a Markov chain with a finite state space and irreducible transition matrix visits every state infinitely often with probability one regardless of the initial distribution (Iosifescu, 1980), thus:

$$\lim_{t \to \infty} P\{\text{at least one solution in the population is globally optimal}\} = 1. \quad (7.6)$$

Consequently, by exploring randomly and keeping the best solution even G/GRADE is destined to find the global optimum, but the low probabilities associated with certain moves in state space, such as the successful creation and use of building blocks, also ensure that this is not a very meaningful statement in practice. By increasing the diversity of the population through measures presented in the last chapter, the transition matrix should become more uniform and raise the chance of otherwise improbably changes. The next important step is to maximally exploit such changes if they turn out to be beneficial.

7.4 Adaptive Search

The grammatical representation established for networks in this thesis belongs into the group of *indirect context* representations. These are typified by associating each component with a reference that any other component can use to specify interactions, rather than relying on their position within a representation, as for *explicit context* representations (Lones and Tyrrell, 2004). Cartesian GP is classic example of this, as each component is assigned a coordinate by which other components can access its output (Miller and Thomson, 2000). However, these references or coordinates are often assigned arbitrarily, so that there is no correlation with the behaviour of the component. Moving a component from one solution to another is unlikely to work out, as the references are either different or refer to different components – and therefore the context is disrupted.

In G/GRADE, references can adapt to the extent that cellular productions with "compatible" labels are more likely to produce successful offspring, yet when applying variation we take no account of any measure of compatibility. A sophisticated method of determining this property is to create a profile of the functionality of each component and its context, and ensure that components are suitably matched during evolution (Lones and Tyrrell, 2004). However, this option raises the difficult problem of how to assess and match functionality. We may alternatively, and more straightforwardly, assume that ancestry provides direct evidence of common functionality (Stanley and Miikkulainen, 2002). Our intention is to combine this idea with a more adaptive means of changing solutions, so that the chance of successful offspring is improved. Changes proven to be effective should occur more frequently, but instead of evolving a mutation distribution with evolution (as with ES, see section 2.5.1), we will pursue the promising idea of swarm intelligence as it applies to graph grammar evolution.

7.4.1 Swarm Intelligence

Ants are remarkably proficient at establishing shortest route paths between their colony and food sources. Since ants only have severely limited perception and intelligence, this seems to be quite an ethological paradox. To resolve it we have to look at more than just the individual ant, but the ant collective. Ant communication is primarily through chemicals called pheromones, which a moving ant lays on the ground as a trail (Wilson, 1971). An ant will wander randomly, until it encounters a trail, which it might decide to follow. This reinforces the trail, which can lead to a positive feedback (*autocatalytic*) mechanism where more and more ants follow the trail. Since a shorter trail will have more ants traversing it back and forth, the shortest path to a food source emerges over time from these group dynamics (Dorigo et al., 1991).

Ant Colony Optimization (ACO) (Dorigo et al., 1999) is a class of algorithms striving to make use of this phenomenon by defining a set of simple computational agents – *ants* – that solve a problem. The ants explore a graph of states corresponding to partial solutions of the problem. A solution to the problem is incrementally constructed by the ants moving between these states. Where an ant moves depends on the feasible expansions of its current state and the probability distribution of the respective moves. Only local information may be used and no look ahead to predict future states is allowed.

The probability of a move is defined by its trail, which indicates how beneficial this move has been in the past, and optionally its attractiveness, based on *a priori* information about the problem space. Trails are updated after the ants have finished their solution, increasing or decreasing the pheromone levels depending on whether the corresponding moves were part of good or bad solutions, respectively. Pheromones also evaporate at the end of each algorithm iteration, so that stagnation – every ant following the same path – is avoided. In the earliest ACO model, the Ant System (AS) (Dorigo et al., 1996), all ants lay trails, but in the subsequent Ant Colony System (ACS) (Dorigo and Gambardella, 1997) only the best solution updates (and evaporates) the pheromones. This reduces convergence time by focusing the search on the neighbourhood of the best solution discovered so far. The trail update rule is

$$\tau_{ij}(t) \leftarrow (1-\varphi)\tau_{ij}(t) + \Delta\tau_{ij}(t) \tag{7.7}$$

where $\Delta\tau_{ij}(t) = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t)$, m is the number of ants at each iteration, and $\varphi \in (0,1]$ is the pheromone trail decay coefficient (Dorigo et al., 1999).

Further changes from AS and ACS are the use of a candidate list, which is a list of preferred states to be reached from the current state, and the use of a pseudo-random-proportional state transition rule. This adds a probability q that instead of making moves according to the established probability distribution, only the highest probability move is chosen. Tuning q allows us to modulate the trade-off between exploration and exploitation. An ant should exploit moves that have been found effective in the past, but – in order to discover these – must also explore moves not previously selected.

ACO is perhaps the best-known technique implementing the concept of swarm intelligence (SI) (Ramos et al., 2005). SI refers to the emergence of coherent functional global patterns from the collective behaviour of simple agents interacting locally. This constitutes a foundation for distributed problem solving without centralized control. Particle Swarm Optimisation (PSO) is another instance of this, which originates in the simulation of the social dynamics of flocking organisms, like insect swarms governed by basic rules such as nearest neighbour velocity matching (Eberhart et al., 2001).

Aggregation patterns of this kind can be observed in societies of organisms as simple as bacteria and as complex as humans (Chowdhury et al., 2004). One of the mediating factors that allows the collective to exceed the capacities of the individual, as we have observed in the case of ants, is *stigmergy* (Theraulaz and Bonabeau, 1999). Stigmergy refers to the elicitation of environment-changing behaviours based on the sensory effects of environmental changes produced by previous and past behaviour of the collective. By making changes to a common environment (creation of artefacts, changes in position, etc.), individuals indirectly interact with each other, because these changes affect the way further changes will be made. This establishes a form of distributed learning and memory among the whole society, ultimately leading to such wonders as the construction of termite nests – far beyond the comprehension of the individual termite.

7.4.2 Adaptive Production Search

We aim to absorb some of the principles of swarm intelligence into searching the space of cellular graph grammars. The existing process is essentially a variation on an evolutionary algorithm. Evolutionary algorithms and ACO differ in that the evolutionary algorithm represents the knowledge about the problem as a population of solutions, whereas ACO maintains a memory of past performance in the form of pheromone trails. Within G/GRADE such a memory may provide useful guidance in exploring the grammar, as the premise of grammar-guided search has so far only been partially fulfilled. G/GRADE presently generates offspring by copying and changing one or more of the productions that make up an existing network. A large network can only arise if productions are added to the network, that is, nonterminals are added to one of its constituent productions. Productions can only survive if they are useful in some existing network – thus, unlike any random construct, these productions also have a higher probability of being useful in a new network that solves the same problem. Naturally, this probability diminishes if there are niches for many different networks in the population, because we might randomly pick a production and use it in a context for which it was not evolved.

Reinterpreted as ACO, each production is a partial solution, and the addition of a production constitutes a move. Unlike ACO, G/GRADE has no explicit probabilities assigned to each move. Consequently, production choice is highly random and depends solely on the composition of the grammar, which is a function of the problem task and not very well understood at this stage. Multiple near-identical production in a population may reinforce their own exploitation, but this conflicts with the ease of reuse in G/GRADE. Having an adaptable probability distribution as with ACO would provide superior guidance, but partial solutions in G/GRADE are exceedingly short-lived: productions are added and removed with every generation. The path that one ant builds can hence rarely be followed by another. ACO for G/GRADE does not appear directly feasible, but a more limited model is possible and will be presented next.

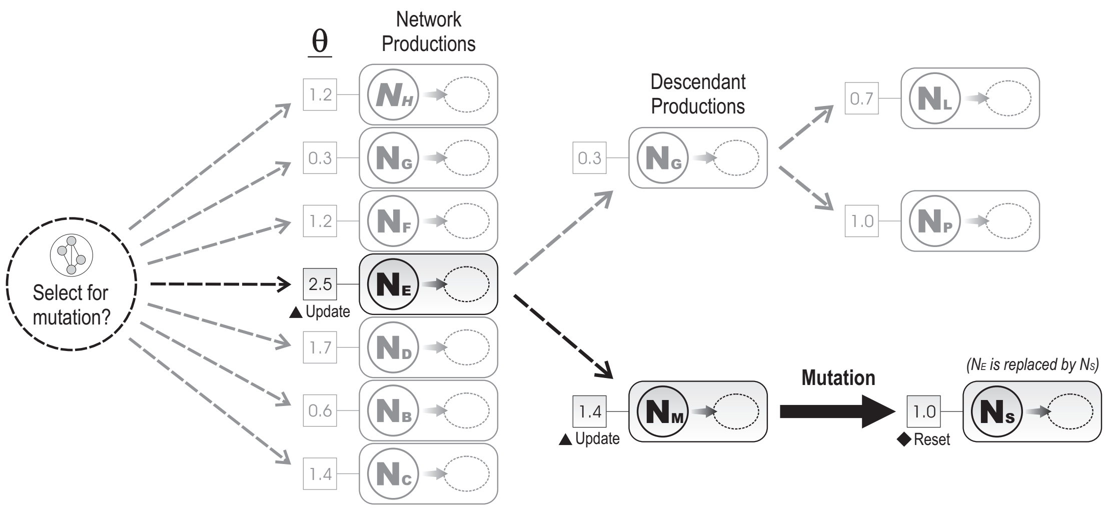


Figure 7.5: Adaptive production search: When mutating a network, the probability of a production being chosen for mutation is dependent on θ, which is then replaced by a descendant also determined by θ before random mutation is applied.

At least two choices must be made when generating offspring from a network. First, we choose one of the productions expressed during derivation of this network. Instead of randomly choosing from a uniform distribution, as in the existing framework, the following heuristic inspired by PBIL (Baluja and Caruana, 1995), an early EDA (see section 2.6.3), is implemented: the chance of a production being chosen is decreased if it rarely results in successful offspring. A real value θ is stored with each production. When choosing a production to mutate, the chance of a specific production R_i being chosen from m productions is

$$P(R_i) = \frac{\theta_i}{\sum_{j=1}^{m}\theta_j}. \tag{7.8}$$

θ is multiplied or divided by a user-defined factor $\rho > 1$ depending on whether the new offspring of this production survives into the next generation or is eliminated, respectively. No evaporation of θ occurs here. θ is simply reset to 1 for every new production, as the expected success of mutating a new production may be independent of the success of mutating the original production (e.g. in a different context). The presented mechanism should globally reduce the mutation of productions that rarely lead to good offspring (e.g. productions fully optimised for their context) and focus on other productions that do.

The second choice concerns the nature of mutations. Applying the above method to the different variation operators would potentially allow for a good balance between the possible variations to be achieved, but as this has been done for numerous other evolutionary algorithms previously, we did not follow it up. However, some of the graph grammar variations involve additional choices, such as a choice of label and a choice of production being added, all from potentially very large sets. As, in some cases, multiple variations may be needed to produce fit offspring, a highly specific sequence of such variations is just not likely to occur. Yet if it does occur, it never needs to occur again, as it will be stored as a new production.

Our suggestion here is to make use of the genetic lineage when applying variation. Recording a lineage from a production to all its descendants provides a list of moves that are known to be successful. This includes any moves that involve new references to other productions. A descendant is likely to be located in a context similar to its ancestors, so replacing an ancestor with a descendant seems a promising move. Following this line, we will replace a production, once chosen for mutation, by one of its descendants – and then apply additional variations (as we would without the replacement).

The descendant is chosen according to the above system of preferred mutation targets, as illustrated in figure 7.5. The upshot of this is that the replacement effectively applies previously successful variations immediately, so the search can emphasise the neighbourhood of productions with high offspring ratios. In the context of multi-objective evolution (starting from the smallest network), it also shifts the search effort away from those sections of the frontier (of other small networks) that have been explored already.

7.4.3 Method

We apply the above SI-inspired extensions separately and in combination. Choosing a production for mutation according to real value θ will be referred to as the *Target* choice; replacing a production by a descendant is the *Lineage* choice. The pheromone factor is $\rho = 1.2$ and the probability of replacement by a descendant is $\psi = 0.9$. These values were chosen *a priori*: ρ should be set so that a production's respective θ is substantially different – but not excessively different – after several successful (or failed) mutations, whereas ψ should allow for some cases where no descendant is chosen, but also have a large value so as to increase the experimental effect here. The parameters are otherwise as with the default configuration. Results from 100 runs are summarised in figure 7.6.

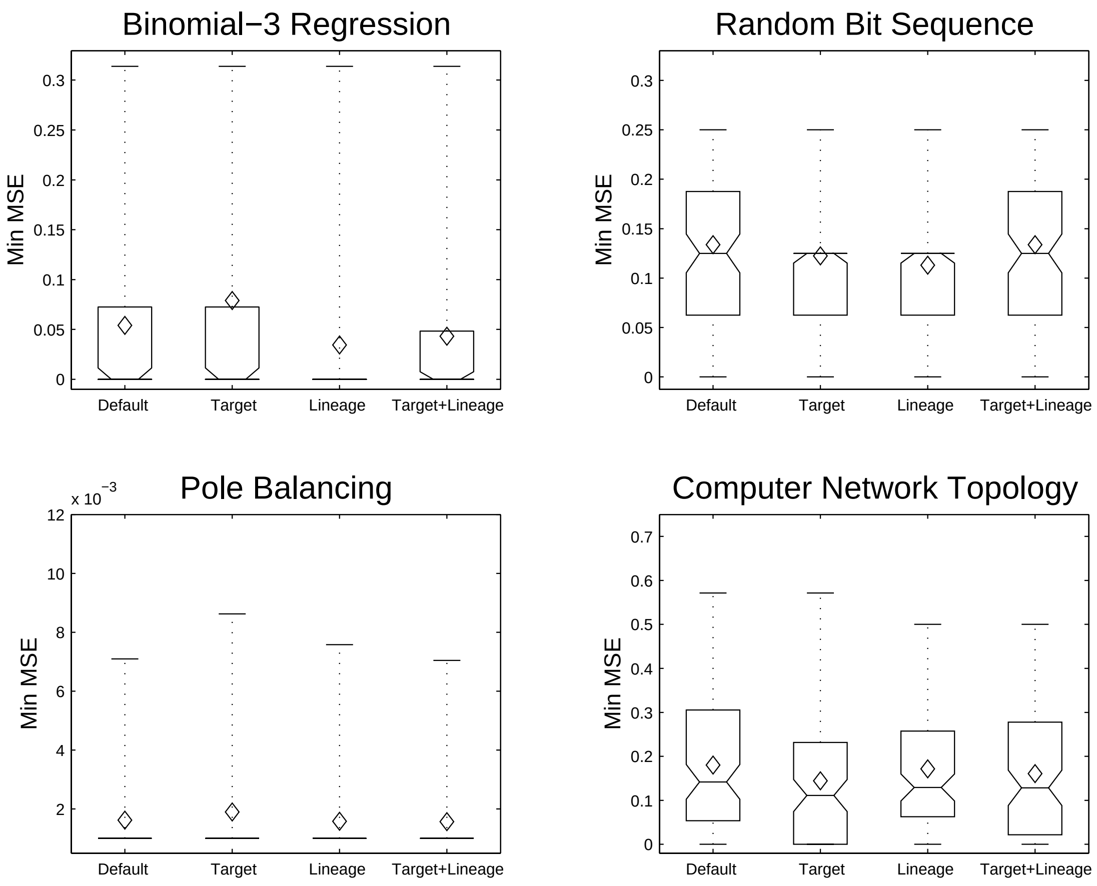


Figure 7.6: Performance box plot for applying adaptive production target choice, adaptive production lineage choice, and the complete model.

7.4.4 Results & Discussion

Figure 7.6 reveals the results of using the SI-inspired extensions. No evident trend can be observed across the different problem tasks. Applying lineage replacement results in better performance on all tasks compared to the default configuration, but the difference is not significant. Further informal experiments on our part, using different parameters and diversity measures all produced similar results. On the whole, this is not a favourable outcome for this attempt at a more adaptive search process. Why does it fail? It is likely that characterising a production through a single parameter θ is an oversimplification, as it fails to take the dynamic context into account – unlike with a GA, where the context of a gene is static.

Furthermore, most of the productions in the grammar do not have many descendants, or in fact any: only 29.8% (averaged across all problem tasks) of final generation productions had at least one descendant present in the grammar (see figure 7.11 in section 7.6.1 for the distribution). Network evolution is characterised by a punctuated equilibrium, and only a few offspring survive each generation, often to replace their parents in the same niche. Under these conditions any production not replaced by a descendant is equal or better than its descendants. Although the descendants are also the only known successful mutation transitions, the results suggest that a preference for the offspring over its parent is indeed not very beneficial to overall performance.

Although performance appears unaffected by the adaptive search, it has a notable impact on the composition of the grammar, which is much smaller despite no decrease in average network size. We discuss this further in section 7.6.2, which takes a closer look at the evolved graph grammars. Beyond this, it is unclear where else an SI-like process may be applicable and useful. Determining the complete probability distribution for good mutations appears infeasible given the multitude of possible production contexts and mutation types, including those involving other productions and continuous-valued labels. Learning the kind of changes that need to be made is clearly the way to go, and neither the graph grammar itself nor the SI-inspired techniques are quite sufficient. We will make some specific suggestions on this issue in the final chapter.

7.5 Balancing Bloat against Performance

The objective function MSE has so far been the only quality measure that we have relied on in comparing the different algorithms and parameter settings. However, section 4.3.2 highlighted the issue of bloat in graph grammar evolution, and thus a size measure was introduced as a second objective to be optimised. The size objective restrains network growth and also provides the user with the freedom to choose a compromise between network performance and derivation size.

However, in some instances only the best performing solution is acceptable and searching the entire Pareto frontier would then seem a disproportionate effort. Yet completely removing the size objective leads to excessive bloat – and hence resource consumption – that is unacceptable in practice and difficult to study in a systematic way. A viable middle ground is constituted by having size become a secondary objective: only solutions with equal performance will compete on the size objective, so any solution with better performance will dominate another solution independent of its size.

We shall also use this opportunity to evaluate a variation to the size objective.

The present measure of size is the sum total of nonterminals, terminals, and external nodes expressed during network derivation. It is not the same as a node/edge count of the network, but any less inclusive measure would permit bloat. There is a potential drawback, however, in that defining the same network with fewer productions leads to a smaller size, which encourages fewer productions overall and hence discourages a diverse set of general and reusable productions from ever arising. Yet the aim should be to discourage bloat, not to discourage reuse. We suggest to address this by *sharing* the size of a production among all the network in the population that make use of this production, i.e. each network only bears its fraction of the total size of the production population (excluding recurrency, which is still scored cumulatively). A production contributing to many networks therefore becomes cheaper, which facilitates its reuse as any network with this production, rather than a unique production, is under less selection pressure on the size objective.

7.5.1 Method

Experiments are performed using our standard set of problem tasks and the default configuration, with three variations being tested: 1) the size of a production is shared among that networks that utilise it; 2) the size objective is delegated to being a secondary objective, consulted only on equally performing solutions (for brevity, we denote this as *No Size* in the charts and tables); and 3) in addition to size as a secondary objective, fitness case entropy (or simple entropy for pole balancing) is also included as a primary objective. This last variation was prompted by the results obtained for 2) – we hence felt it was necessary to evaluate it as well. Results for these experiments are exhibited in figure 7.7.

7.5.2 Results & Discussion

Allowing production size to be shared among networks triggers a significant decline in performance across all tasks where reuse would be expected to matter (i.e. all except pole balancing). At first this may seem puzzling, but analysis of the solutions provides a clue. Using shared size causes the evolution of a grammar where some productions call on many more (10+) other productions than is otherwise typical. A coevolution appears to happen where networks minimise their effective size by maximising references to each other's productions. Part of the population is thus optimised on the size objective at the expense of actual

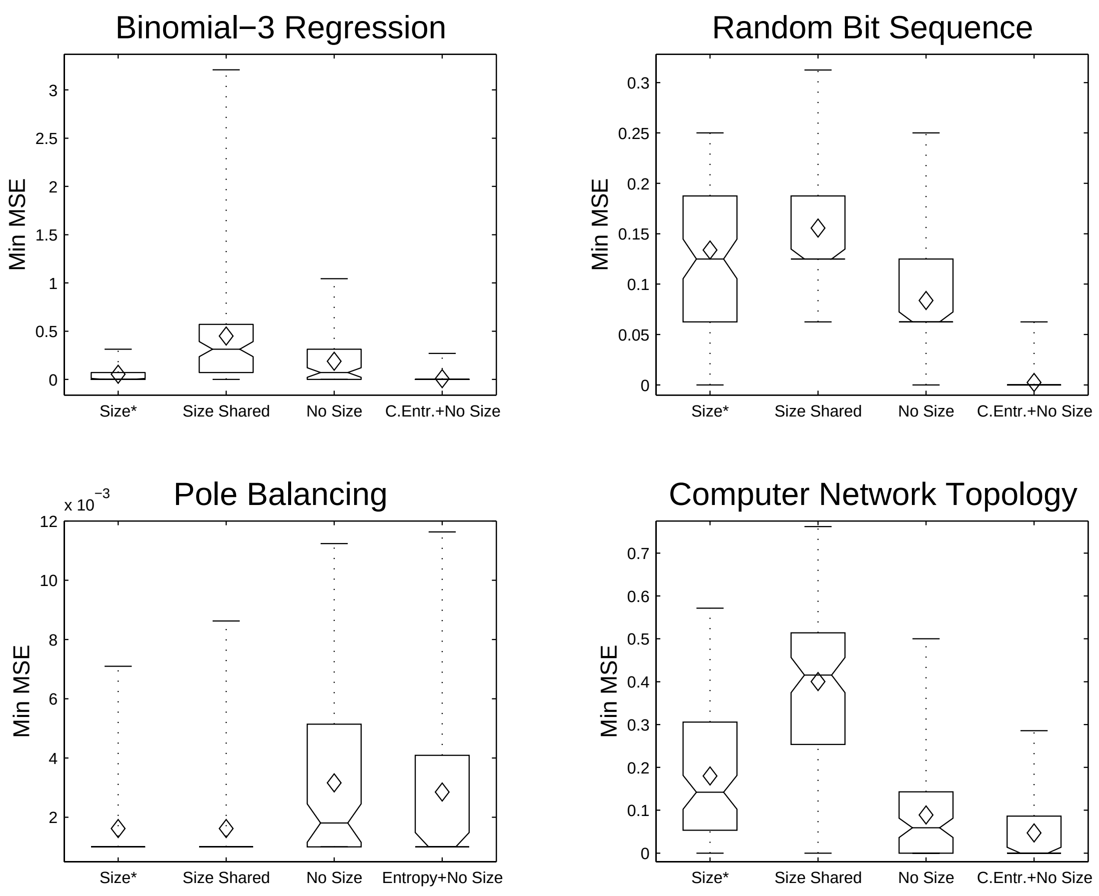


Figure 7.7: Performance box plot for experimental runs with a shared primary size objective, without a primary size objective (labelled *No Size*), and without a primary size objective but a diversity objective.

task performance. We learn from this that care must be taken when postulating complex objectives; it is easy to create new evolutionary niches that divert the search from the objectives that we are really aiming for. In this case, the algorithm exploited a means of reducing the size objective without reducing the actual size of either the networks or the grammar.

Eliminating size as a primary objective also leads to significant performance differences, with the Binomial-3 regression and the pole balancing performing worse ($p < 5 \times 10^{-8}$), yet the RBS circuit and the CNT design performing better ($p < 3 \times 10^{-8}$). This is likewise reflected in the success rates and is not a particularly helpful outcome, unless we can determine what makes these two pairs of problems produce such contrary results. Reviewing the tables in Appendix A reveals an interesting pattern across previous experiments. The problem pair that performs better in this experiment has improvements in MSE mainly correlate with increases in size of the minimum MSE solution; the opposite is true for the

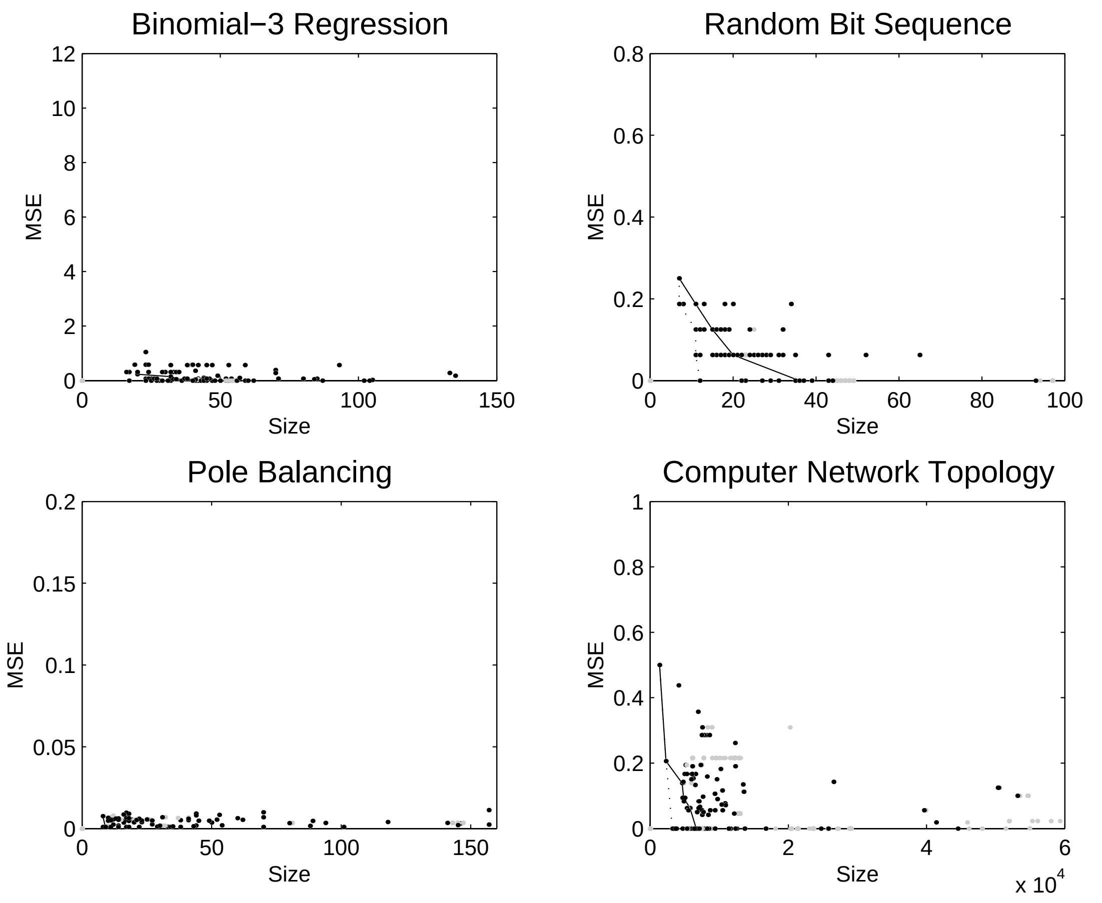


Figure 7.8: Pareto frontier for final generation solutions evolved without a primary size objective, plotted against performance and size. See caption of figure 6.5 for legend.

other pair. The problems appear to evolve differently with respect to changes in size.

In order to better understand this, the trade-off between size and performance of each problem must be examined. Figure 6.5 (in the previous chapter) revealed the Pareto frontier for evolution with a primary size objective. By having size as a primary objective, search effort is invested into finding solutions smaller than the current minimum MSE solution. This is because many solutions in the population are indeed smaller than the minimum MSE solution, and each solution is allowed a single offspring, which is likely similar to its parent. Figure 7.8 shows the Pareto frontier for evolution with a secondary size objective, which clearly eliminates most of the smaller solutions from the population. Thus, if we achieve a better final minimum MSE solution by having a primary size objective, then this may indicate that the typical minimum MSE solution is larger than the final minimum MSE solution. While this hypothesis is supported by the very small size of optimal pole balancing solutions (which are therefore never practically limited by size), the Binomial-3 regression solutions are no smaller than solutions of the RBS circuit or CNT design.

Another explanation is needed – and diversity may be the key. Figures 7.14 to 7.17 (see section 7.6.4) reveal that the mean entropy of solutions is generally higher with a primary size objective than without. Chapter 6 established that diversity is important in the evolution of the graph grammar, so the deterioration of performance when using a secondary size objective may be accounted for by the loss of diversity. The results would reflect a problem-dependent trade-off between the general benefit of not selecting against size and the detriment of losing diversity as a consequence of this. In line with this, both the Binomial-3 regression and the pole balancing have a relatively much lower diversity (and hence greater performance penalty) than the RBS circuit and CNT design when using a secondary size objective. Presumably, if a diversity objective were to be employed in conjunction with a secondary, rather than primary, size objective, performance improvements should be observed for any problem where size might be constraining factor.

Accordingly, we carried out a separate experiment, which indeed corroborates this hypothesis. Performance on the Binomial-3 regression, RBS circuit, and CNT design improves significantly compared to the default configuration and also compared to the previous outcomes of using diversity measures (except for the regression, where $p = 0.11$). Particularly noteworthy is the improvement with the RBS circuit, where the success rate rises to 96%, in stark contrast to 1% for the default configuration (and 18% with only a secondary size objective). Since the evolution of the RBS circuit frequently stalls over 50000 generations (see section 7.2), it is thus clear that the primary size objective restricts G/GRADE from properly solving the problem, although selecting for diversity remains crucial as well.

The disadvantage of this setup is the increase in solution size and hence computational cost: while using a secondary size objective approximately doubles the mean solution size, in combination with the diversity objective it increases almost seven-fold on the RBS circuit – and more than 20$\times$ for the regression and CNT design. The large mean size is caused by a small number of very big solutions, as suggested by the large standard deviation. A big solution to the Binomial-3 regression is shown in figure 7.9, which exhibits a lot of bloat in the form of nodes that fail to contribute to the output of the network. Much smaller solutions are

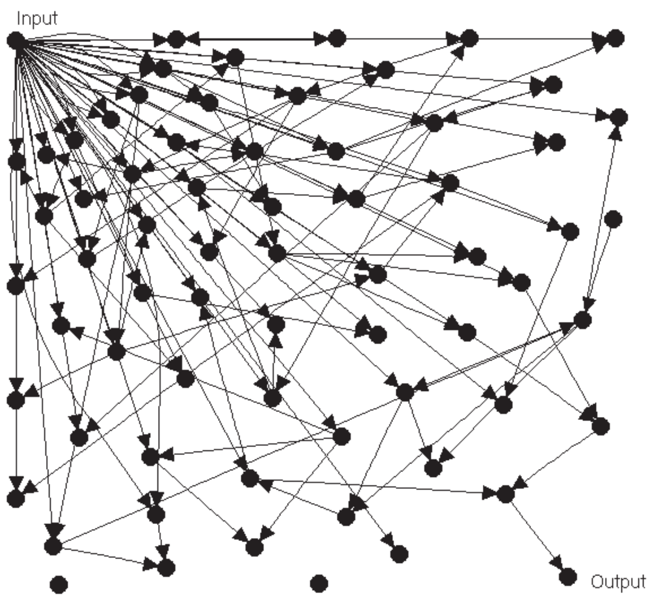


Figure 7.9: A very large solution for the Binomial-3 regression arising from the combined use of a secondary size objective and a diversity objective. Note the large number of nodes without outgoing edges – they cannot affect performance. (To improve readability, node labels are not shown.)

possible, of course, as evidenced by the earlier solution depicted in figure 4.11. A technique for controlling size without significantly impacting on performance, however, continues to elude us. It is not clear how much of the neutral overhead observed in the figure 7.9 is actually required for an effective search. We will have to leave it to future studies to address this issue.

7.6 Observations on Evolved Graph Grammars

This section constitutes an analysis and discussion of the nature of the cellular productions being created during graph grammar evolution. Because the total number of evolved productions is quite large, the focus will be on the productions that make up the final generation of evolution (using the default configuration, unless noted otherwise). The networks derivable from these productions represent the best Pareto-dominant solutions discovered within 1000 generations. It is worth emphasising that the final generation is often already converged and hence in a state of stasis; during phases of rapid convergence the results below may be different, as increases in size would be tolerated because they coincide with increases in performance. We will also later on in section 7.6.4 have a look at how certain statistics of the populations are developing over the 1000 generations.

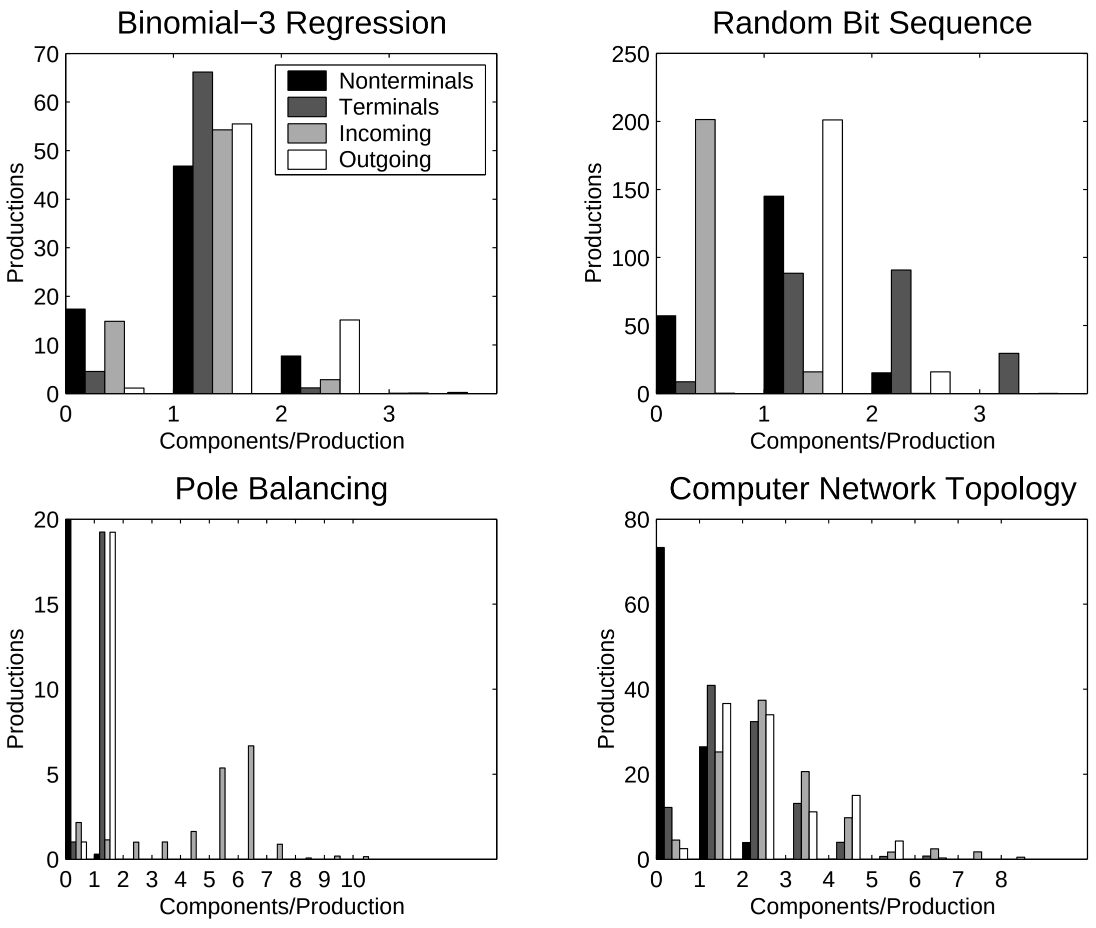


Figure 7.10: Histogram of the amount of different components in final generation cellular graphs.

7.6.1 Cellular Graph Components

As described earlier, each cellular graph consists of nonterminal nodes, terminal nodes, and two kinds of external nodes that represent directed tentacles, marked *incoming* and *outgoing* here. Figure 7.10 displays a histogram of the quantity of such cellular graph components in final generation productions, averaged over all 100 runs on the specified problem tasks. We observe that the productions contributing to solutions of the Binomial-3 regression have a structure that is reminiscent of regular grammars: the typical production defines a cellular graph with one nonterminal and one terminal. Also noteworthy is that most of the cellular graphs have only one incoming tentacle, although all the terminal operators are 2-ary. For this arrangement to be workable, the original input must be passed straight to the deepest cellular graphs, and the actual result is computed when the outputs of subnetworks defined by these cellular graphs are combined by the terminal automata of the host networks.

Solutions to the RBS circuit exhibit a similar design, but few cellular graphs define incoming tentacles, as this problem requires no inputs to the network at all. Without inputs to cellular graphs, however, the recurrency needed for generating a sequence can only arise locally between terminal automata, which suggests itself as the reason why many cellular graphs include multiple terminals. In contrast, six inputs are necessary for success on the pole balancing task, which is reflected in its histogram. Few references to nonterminals are observed here, which implies that the optimal solution can be described by a single production. This, unfortunately, also means that the pole balancing task is not very useful for addressing questions about production bloat and reuse (as in section 7.5). Several productions correspond to bloat here, as they have no outputs, a phenomenon we observe for the other problem tasks as well. The histogram for the CNT design is difficult to make definite statements about, as it summarises distinct populations trying to solve 10 different problems. However, the high average number of terminals and low average number of nonterminals indicate that many of the solution networks are represented by very shallow development trees, so a direct encoding scheme for networks may have been just as effective on this problem.

7.6.2 Production Reuse

Figure 7.11 displays the mean number of productions in a final generation population that have the indicated number of instances of being used or having a descendant. Being *used* is here defined as being referenced by another production in the derivation of some network; a *descendant* is an offspring of a production, either arising by its mutation or by copy-changing to accommodate the mutation of a referenced production. The figure reveals that a great majority of productions are exclusive to only the network they are part of: they have not been successful at reproduction – or not had the opportunity yet. A minority of productions in all of the problem tasks have high usage or many descendants. The distribution appears to be scale-free, which is expected as the referencing and reproduction processes exhibit strong preferential attachment. If a production is used or copied often, it is more likely to end up being used or copied elsewhere as well.

A measure of reuse (expressed in terms of the verbosity of the network description) is shown in figure 7.12 for some of the main experimental configurations explored in previous sections. The problems previously identified as making more

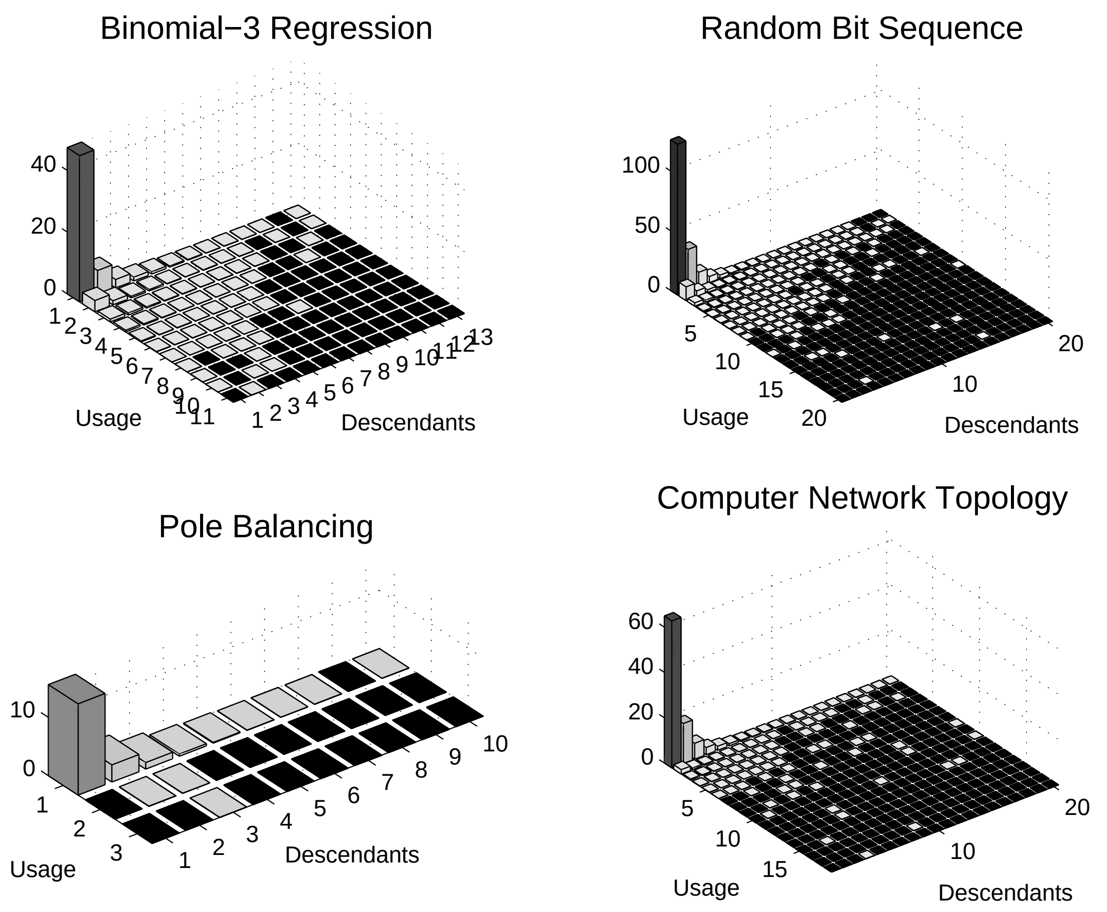


Figure 7.11: Histogram of the usage, i.e. networks to which a production contributes, and the number of descendants (present in the population) of the final generation productions.

use of multiple productions – the Binomial-3 regression and the RBS circuit – exhibit a moderate degree of reuse, while pole balancing naturally shows very little. Some extreme outliers are noted; the occurrence of a small number of large recursive solutions in the population could be a possible explanation for this. We have noted earlier that the use of adaptive search has little impact on performance, but this figure reveals that it encourages reuse on the problems where reuse should matter. Also noteworthy is the significant increase in reuse associated with removing the size objective as a primary objective. Reuse appears to scale with the size of the evolved solutions, which indicates that the efficiency of the graph grammar representation would be most evident with problems that require larger solutions than the ones evaluated here.

Overall, the results on production components and reuse imply that solutions originating from deep, reusable development trees occur seldomly, especially for

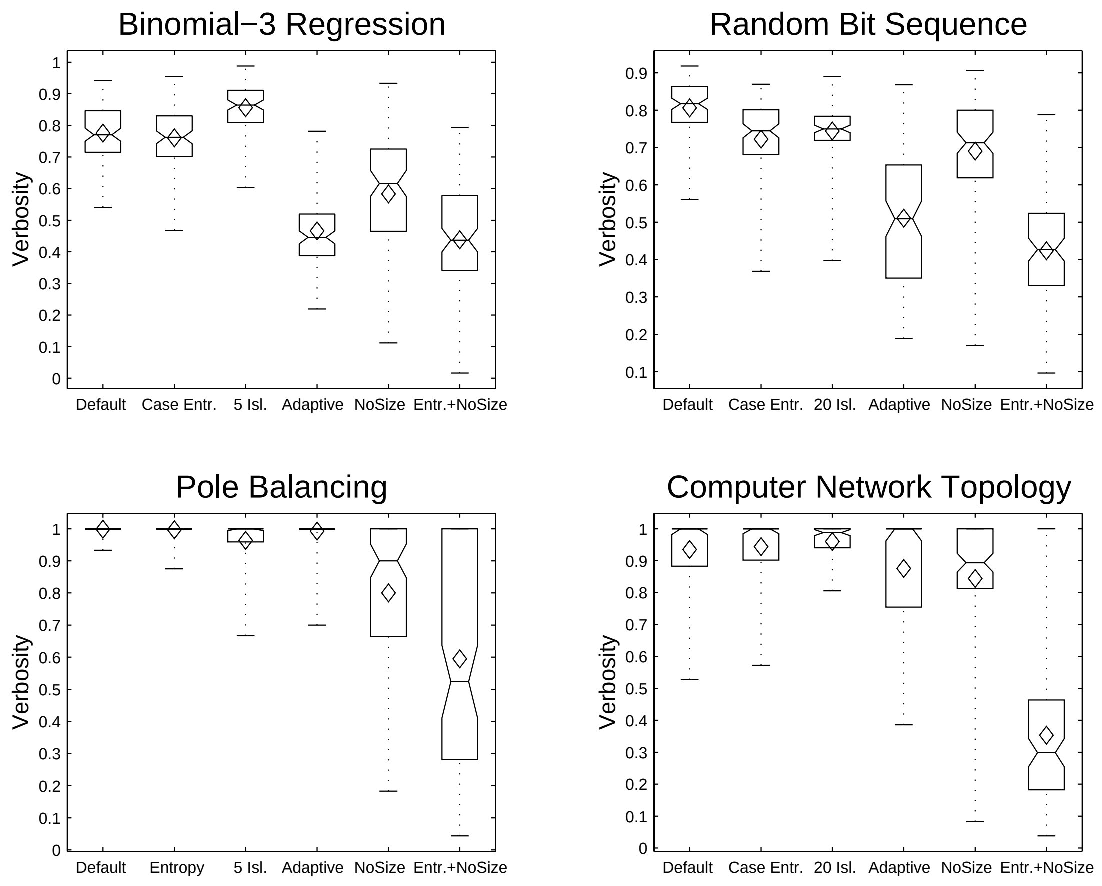


Figure 7.12: Verbosity box plot for the final generation of the main different experimental configurations. The verbosity is defined as the ratio of the total number of productions expressed when deriving all networks over the total number of productions in the population. If verbosity is 1, then each production belongs to a single network and there is no recursion, whereas a low verbosity indicates that productions are being referenced multiple times (i.e. high reuse). *Adaptive* refers to the use of adaptive production search (section 7.4.2), *No Size* is a setup where size is only used to decide between solutions of equal MSE (section 7.5), while *Entr. + No Size* combines the latter with a fitness case entropy objective.

the CNT design and pole balancing. This is likely due to a low probability of suitable productions arising by chance, paired with a selection process favouring small size, which further penalises against groups of interacting productions. Since our earlier attempts at improving reuse either did not lead to performance improvements (with *shared size* and *adaptive production search*) or triggered a rather substantial growth in solution sizes (with *secondary size objective*), it is still not clear how the supposed benefits of reuse, and hence scalability, are appropriately realised and exploited in practice. At this stage, reuse is shunned

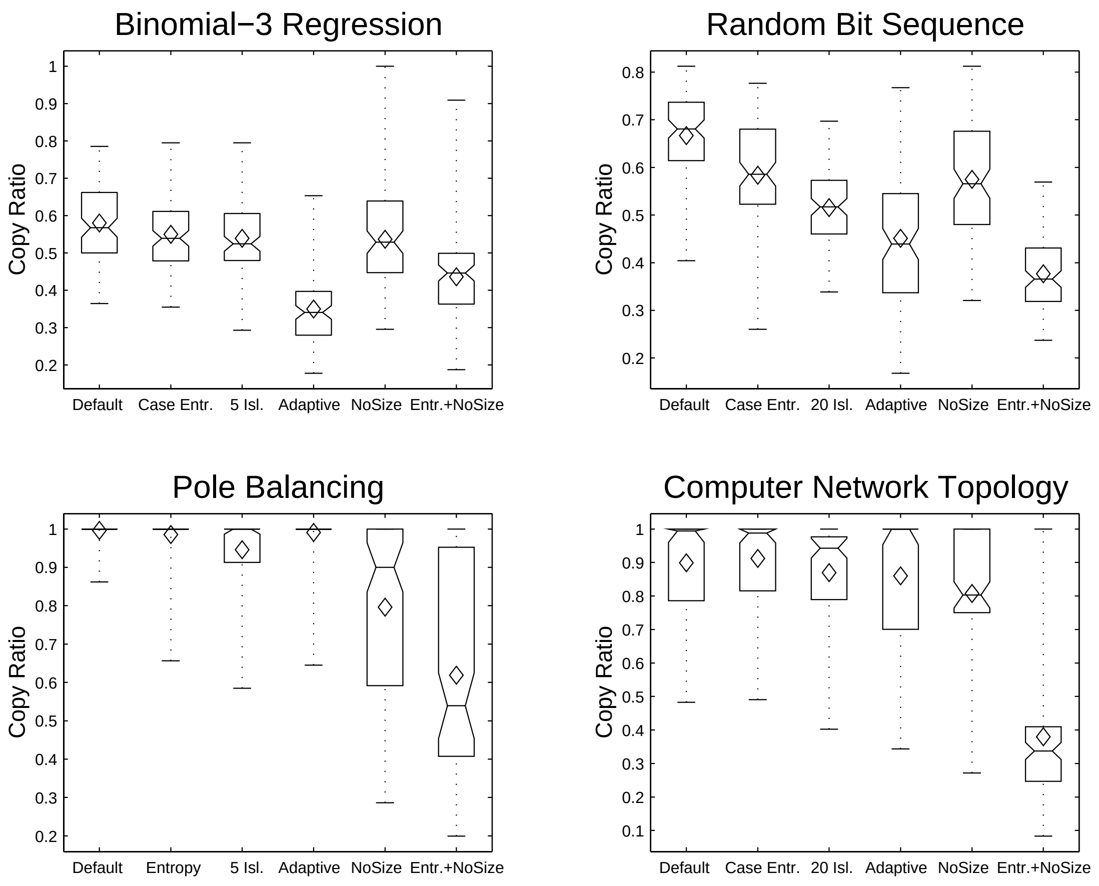


Figure 7.13: Copy ratio box plot for the final generation of the main different experimental configurations. The copy ratio is defined as the ratio of the number of productions copied in order to add the network to the grammar over the number of productions contributing to the network.

more often than not, which is also a likely explanation of why more complex label offsets have resulted in performance improvements in section 5.1.4: they allow a production to construct more complex networks without having to resort to as many other productions.

7.6.3 Production Copying

The need to copy only part of the genome of a solution was one of the salient features of reproduction in the graph grammar framework. Yet by randomly choosing a production for mutation, we are likely to pick a production from the lower half of the derivation tree. Many references must therefore be changed, which also requires many productions to be copied. Figure 7.13 shows the copy ratio, the average ratio of the productions that are copied against the productions that are part of the original network. Extensive copying is needed for problems that use fewer productions, although in absolute terms the number of copies is also less because of this. Not using a primary size objective has the opposite effect, producing ratios that are closer to what we would expected from randomly picking a production in the tree. Adaptive search also reduces the copy ratio, which suggests that productions near the root of derivation tree are preferred. Deeper productions may be less mutable because they have existed for longer and have already been optimised as building blocks.

7.6.4 Generational Development

The figures on the following pages depict the changes of various population and solution statistics over 1000 generations, for each of the different problem tasks and the main different configurations. The plots show, in clockwise order starting from the legend: the MSE of the best performing solution; the mean entropy of all population members (across all different fitness cases, i.e. fitness case entropy, except for pole balancing); the proportion of solutions that are replaced by new solutions each generation; the total number of productions in the population; the mean size of all solutions, defined by the number of expressed terminals, nonterminals, and external nodes; and the size of the best performing solution.

Network evolution without a primary size objective but with a diversity objective (denoted *C. Entr. + No Size*) converges most quickly among all alternative configurations and produces the best outcome on the majority of problem tasks, with the exception of pole balancing, where we have previously noted that the small size of the optimal solution actually penalises any relaxation of the size constraint. The evolutionary convergence is characterised by a rapid increase in solution size, which starts to flatten out after a few generations and remains comparatively steady throughout subsequent generations, although the exact nature of this is somewhat problem dependent.

Without either a primary size or a diversity objective (*No Primary Size*) the growth of solution size is much more contained, but solutions also exhibit inferior performance. Of the configurations with a primary size objective, the island model produces the largest solutions, most likely for the reasons discussed previously in section 6.2.3. Reductions in size during the evolution seem to be uncommon, particularly when applying the primary size objective, which suggests that either the optimisation towards a minimum size is quite ineffective, or, more likely, that solutions much larger than the current performance optimum have a propensity to do poorly on the performance objective. Nevertheless, it is apparent from the success of the *C. Entr. + No Size* configuration that the best outcomes are obtained when such solutions make up part of the population.

The number of productions necessary for representing the entire network population is dependent on the size of the networks and the extent of reuse in the representation. We have noted that without a primary size objective the verbosity of the representation is decreased, but as the solutions are also very large, the total number of productions remains high. The smallest grammars are obtained with the adaptive production search model, which encourages reuse without penalty to performance, although there seem to be no practical benefits to this. It was originally expected that adaptive search could accelerate convergence, yet as the plots indicate, adaptive search behaves very similarly to the default configuration.

On the entropy statistic, we note that using the entropy as an objective improves average entropy of the solutions, but also not nearly as much as the island model does. An explanation for this may be found in the comparatively low rate of solution replacement that we observe with the entropy objective. It suggests that not much opportunity is given for introducing the kind of structural novelty into the population that is not directly reflected in phenotypic diversity. The age/sequence measures for diversity (not shown here) also produce greater improvements in entropy by maintaining an inherently high replacement ratio.

However, it is not clear why the replacement ratio for the island model is so high – this may have nothing to do with the island model par se, but with our attempt at establishing a Pareto frontier across the islands. The resultant asymmetric elimination of solutions from the islands should cause new niches (in the shape of underpopulated islands) to arise with each generation, allowing otherwise unfit, but potentially novel, solutions to survive for more than one generation.

At any rate, the phenotypic entropy is not a reliable indicator of performance. The least entropy is observed with the secondary size objective on its own, which reflects the loss of the diversity that was provided by having a Pareto frontier of different solutions, but this configuration is not the worst performer. However, in the Binomial-3 regression and the CNT design it is inclined to flatten out early, which is indicative of a premature convergence of many runs. Adding the diversity objective raises entropy (and, of course, greatly raises performance), but in both the Binomial-3 regression and pole balancing tasks the entropy still remains below the corresponding configuration with a primary size objective.

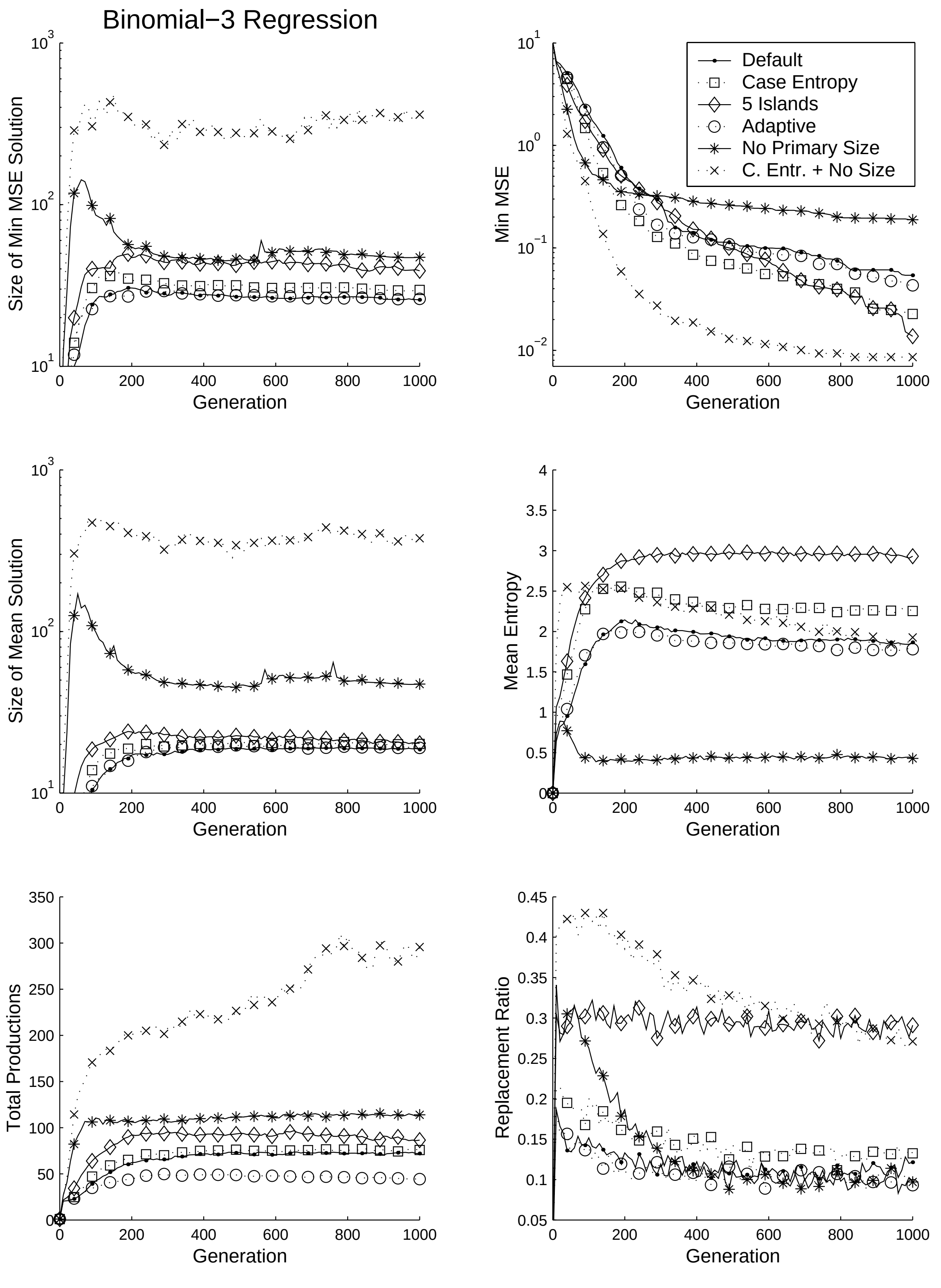


Figure 7.14: Generational development of performance, size, and diversity statistics for the Binomial-3 regression problem. (MSE and size are shown on a logarithmic Y-axis to improve readability.)

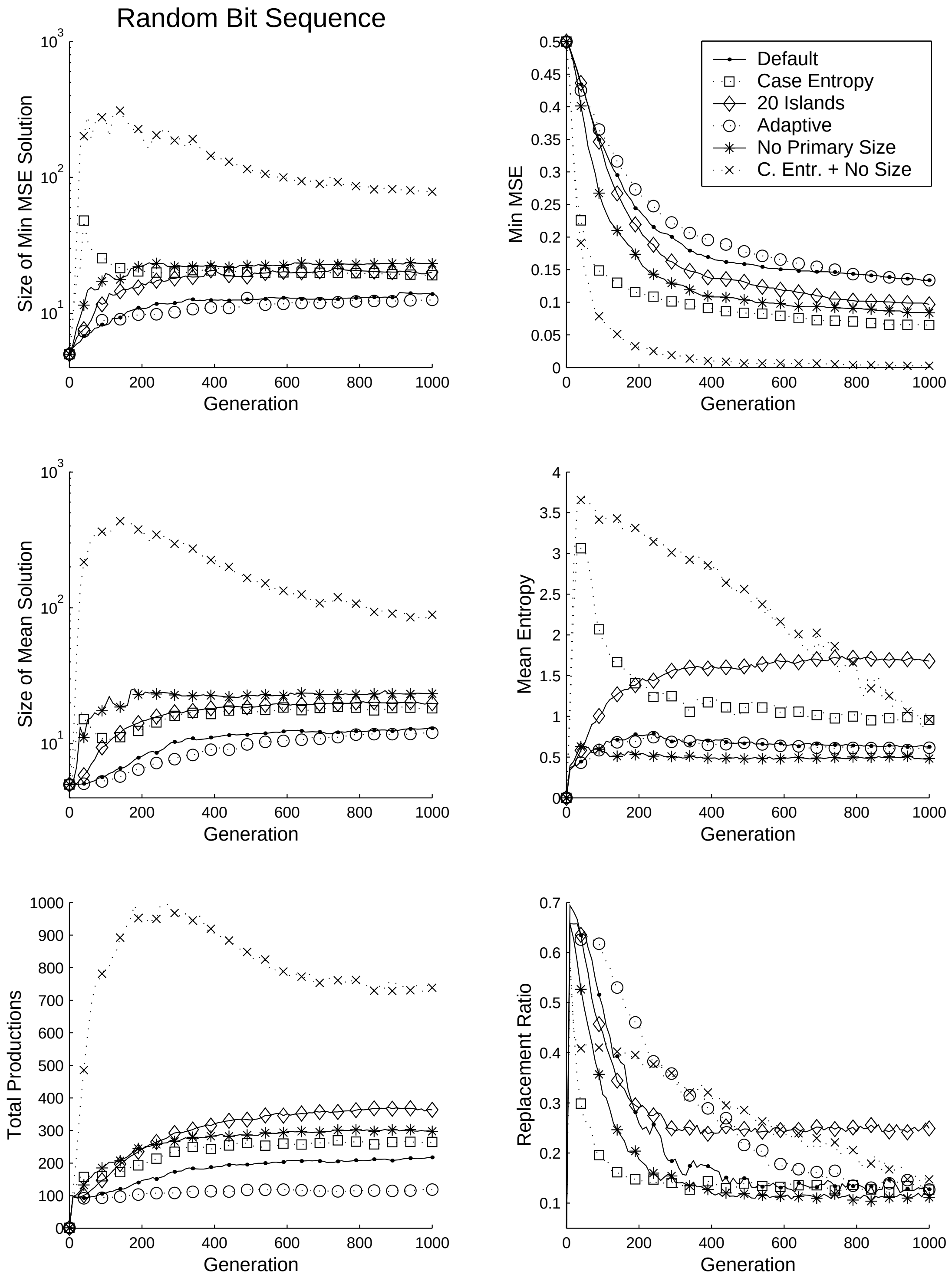


Figure 7.15: Generational development of performance, size, and diversity parameters for the RBS circuit problem. (Size is shown on a logarithmic Y-axis to improve readability.)

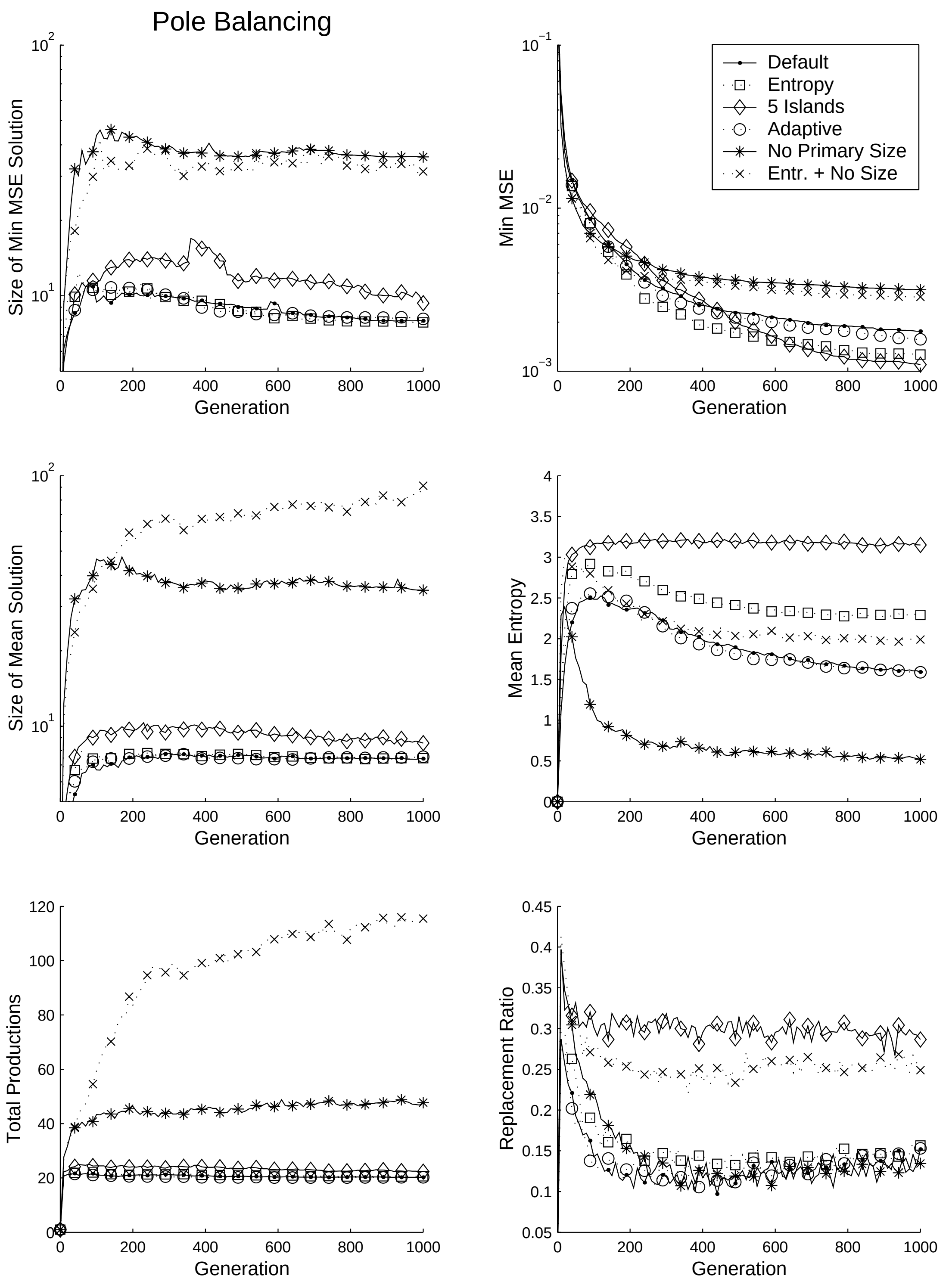


Figure 7.16: Generational development of performance, size, and diversity parameters for the pole balancing problem. (MSE and size are shown on a logarithmic Y-axis to improve readability.)

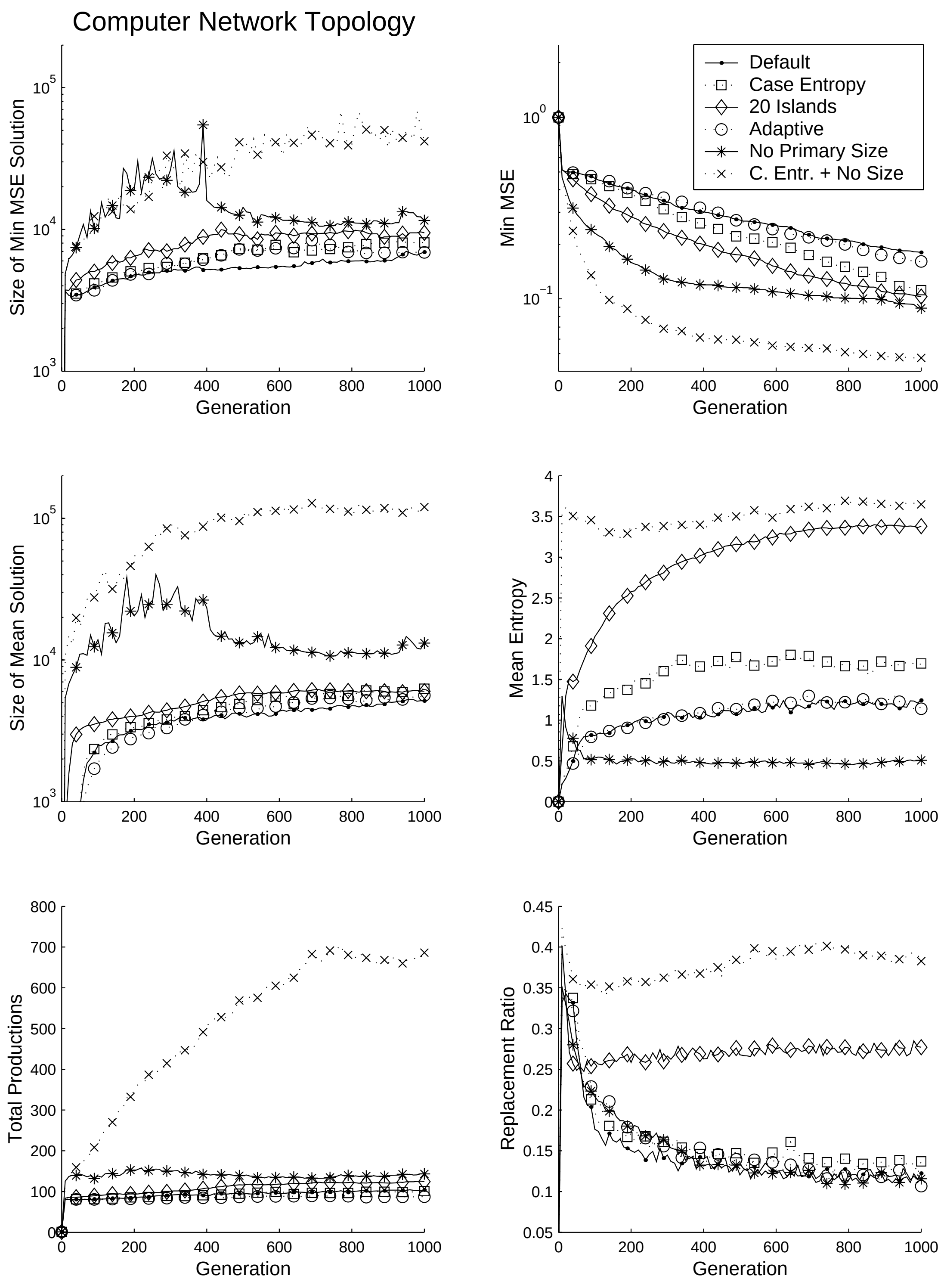


Figure 7.17: Generational development of performance, size, and diversity parameters for the CNT design problem. (MSE and size are shown on a logarithmic Y-axis to improve readability.)

7.7 Comparisons against Other Systems

G/GRADE is a tool for optimising the topology of networks. Few assumptions are made with respect to what these networks represent, thus G/GRADE can be applied to a large number of domains. Problems from several of these domains were used in this thesis as test cases for evaluating various aspects of the G/GRADE framework. Some of these problems have been used in previous research, which permits a comparison of G/GRADE against alternative systems.

The Binomial-3 regression problem was studied in detail by Daida et al. (2001), who evolved solutions to this problem using standard GP. The success rate that was obtained for a population of 500 solutions evolved over 200 generations was 84%. In comparison, G/GRADE achieves 84% with the island model, and 71% without, for a population of 20 solutions evolved over 1000 generations. On the 6th-order polynomial regression task, CGP has a success rate of 61% with a population of 10 solutions (and a maximum of 10 nodes per solution) over 8000 generations (Miller and Thomson, 2000), and GP achieves 64% with a population of 4000 and 50 generations (Langdon, 2000b). G/GRADE obtains success rates of up to 100% with a population of 20 over 1000 generations depending on the choice of diversity objective applied (no convergence is achieved without).

G/GRADE is not very competitive on the 6-multiplexer problem, where the minimum computational effort is 225100 evaluations if a diversity objective is applied, and 413000 without. Koza (1992) reports a minimum effort of 245000 for standard GP on this problem, although diversity measures can further improve this result to less than 50000 evaluations, as shown by McKay (2000). As the 6-multiplexer can be represented as a tree, we project that the performance of G/GRADE would be improved by constraining the search space from multigraphs to trees. Preliminary investigations into this have indicated that G/GRADE can solve the 6-multiplexer problem with a minimum effort of 34800 evaluations if we preserve diversity, and 67760 evaluations otherwise. Further study will be needed to generalise these findings, however.

For the Backpropagation MLP, we trained a user-defined 2-layer neural network with 2 hidden neurons and 3 output neurons, using standard backpropagation for 2000 epochs (Luerssen, 2005a). The mean MSE resulting from this setup is 0.0157, which is also what the best evolved networks approximately achieved (using half that many evaluations). However, the average evolved network has a notably worse error of 0.0549, which is more typical of the performance of a single-layer perceptron on this problem. The latter is very simple to describe as a single production, so it is a prominent local optimum upon which G/GRADE converges. To determine the other neural network, the pole balancing solution over 100000 cycles, G/GRADE requires about 12700 evaluations (Luerssen, 2005b), which is much less than the 80000 evaluations of Wieland (1991), and also compares well to the 34000 evaluations of Cellular Encoding (Gruau et al., 1996) and 12600 evaluations of Symbiotic Adaptive Neuro-Evolution (Moriarty and Miikkulainen, 1996). It converges not nearly as fast as the 3600 evaluations reported for Neuro-Evolution of Augmenting Topologies (Stanley and Miikkulainen, 2002), although that study also uses a different experimental setup with a starting angle of 1° (rather than the more difficult 4.5° used by us).

No comparisons are available for the RBS circuit and the CNT design. Both problems are unique, as they make use of G/GRADE specific features that are uncommon in other systems.

7.8 Summary

The purpose of this chapter was to observe and improve some of the convergence properties of G/GRADE, but – perhaps attributable the relative complexity of the underlying algorithm – we seem to have raised many more questions than answers. Our effort at accelerating convergence by implementing swarm-intelligence inspired search adaptation failed to produce improvements to performance, although it did improve grammatical convergence towards greater reuse. However, this may not have been the limiting factor in graph grammar evolution at all, as the most significant outcome relates to size control. We have ascertained that the search process is severely constrained by co-optimisation towards a size objective, yet excessive bloat occurs as soon as the effective importance of size is reduced. The representational effectiveness of graph grammars becomes evident with the latter, but at great computational cost; a proper balance has not been found. However, the notion that diversity is a critical aspect of graph grammar evolution has been further reinforced. We have also noted that G/GRADE, despite its generality, is quite competitive to other evolutionary optimisation systems on the problem tasks evaluated here.

Chapter 8

Conclusions

The most natural representation for so many design problems is a network, but it is often easier and more transparent to evolve strings and trees. The system presented in this thesis is a significant step towards achieving a simple, formal, comprehensive basis for network evolution. Of particular significance is the potential scalability and generality of this approach, because solutions with real-world relevance literally come in all shapes and sizes. Earlier research has noted that generative representations are an appropriate starting point for this, since they facilitate reuse of design and can incorporate a bias of the design problem into their structure.

The novelty of the research presented here is in the application of a generative representation based on hypergraph grammars. The hypergraph grammar is a simple formal model, which avoids some of the complexity pitfalls of existing models that try to be biologically realistic. It hence permits for a more systematic study, as we have demonstrated on a diverse set of design problems. Additionally, unlike previous attempts at utilising graph grammars, the graph transformations are not predefined and fixed, but fully evolvable, allowing for an automatic optimisation of the network design bias and thus a greater degree of domain independence.

Despite the prevalent use of grammars in the field of artificial evolution, it is quite atypical to find a grammar being directly evolved. This thesis establishes an innovative technique for evolving productions of a grammar that describes a population of solution networks. Each solution is represented by a production that calls upon other productions, potentially shared with other solutions, to iteratively construct the solution. Grammar evolution begins with a single empty production, so each production either represents a solution or is exapted by another solution as a component of itself – a form of endosymbiosis, reflecting the emergence of complexity from basic matter.

The existing structure of the grammar has strong influence over the direction into which the grammar can evolve further. Avoiding local optima and ensuring continued progress necessitates a diversity of productions to be present in the grammar, and our research has surveyed various schemes for facilitating this. Tentative steps have also been made towards establishing an adaptive search model for direct grammar evolution, which better exploits the knowledge contained by the grammar. Specific contributions are summarised below. We expect that these are merely the first milestones – with more to come – towards a better understanding of graph evolution, grammar evolution, and the intersection of these important fields.

8.1 Contributions

A survey and analysis of how networks can be optimised demonstrates the merit of developmental models in an evolutionary search context.

With a principal focus on the connectionist paradigm of computing, Chapter 2 examines the notion of what a network is and how it can change. Various methods for adapting the weights of network links and optimising the network design are discussed. Evolutionary algorithms appear the best suited for the latter, but the scalability of this approach is limited. We can address this by applying some of the basic engineering principles also employed in biological life, which include modularity, neutrality, and reuse. Chapter 3 surveys previous evolutionary research into artificial embryogenies that are applicable to network design. The complexity of many features of these systems is justified by biological plausibility, yet also greatly increases the difficulty of assessing their practical usefulness. This leads us to grammars, which capture the essence of what is desirable about biological development in a simpler framework. As existing models typically apply to strings or trees, we present the case for evolving a graph grammar directly.

A novel method of graph grammar evolution is presented, which transforms the formal model of a hypergraph grammar towards being directly evolvable by a multi-objective evolutionary algorithm.

Hypergraph grammars constitute a unique conceptual foundation for the *cellular graph grammar* described in Chapter 4. An evolutionary optimisation system named G/GRADE is presented that directly operates on the grammar and the population of networks indirectly represented by this grammar. The performance of networks derived from the grammar determines the addition or removal of productions from the grammar, with new productions obtained from applying a set of simple mutation operators. The problem of bloat is addressed by selecting in accordance with an additional size objective in a multi-objective optimisation scheme.

The nature of the grammars and graphs obtained from graph grammar evolution is investigated, and a comparison between alternative heuristics is given.

Constructing a graph from the productions of a hypergraph grammar constitutes a surprising challenge when the source of new productions is random change. Chapter 5 explores different schemes for node and edge matching during graph rewriting and evaluates these on several problems from domains including symbolic regression, circuit design, neural networks, and communications networks. The best results are obtained with a nearest neighbour matching scheme for large label sets used in conjunction with additional label offsets for terminals and non-terminals. Additionally, we investigate how to constrain graph construction so as to generate graphs rather than pseudographs, and also discover that modularity in the scope of matching is not only simple and efficient, but also exhibits more consistent performance than less modular alternatives.

The relevance of diversity in graph grammars is recognised. Phenotypic diversity objectives and island models are introduced to improve diversity.

Randomly initialising a grammar is likely to lead to networks that are either disconnected or bloated or both, yet complex networks can only arise from multiple correct productions. To ensure that these exist, a diversity of productions must be maintained. Chapter 6 suggests two representation-independent methods for accomplishing this by promoting the phenotypic diversity of networks. Firstly, we explore the use of a variety of diversity objectives in addition to the existing performance and size objectives. Most successful is the entropy measure of phenotypic diversity across fitness cases, which leads to significant performance improvements on the majority of problem tasks. Further significant improvements are obtained in combination with a less restrictive size objective, but notable increases in solution size become an issue here. Alternatively, we present an island model for multi-objective evolution, which exhibits performance benefits comparable to the first method.

The structure of evolved grammars is analysed, and an adaptive search scheme is established for improving grammatical convergence.

Chapter 7 investigates the generational development and statistics on the grammars evolved for each problem task. We suggest the application of concepts from swarm intelligence to accelerate convergence. The associated experiments fail to produce significant performance improvements, but reveal significant increases in production reuse, leading to a more compact grammar. Graph grammar evolution is additionally found to be robust to high mutation rates, with an optimal performance obtained with a variable rate of mutation within a single production and across multiple productions of a network. Finally, while G/GRADE maintains a high level of generality, it is also shown to exhibit similar performance to other systems on the evaluated problem tasks.

8.2 Limitations

The volume of experiments in this study is quite extensive, yet we mostly evaluate variations of only a single property of the system against an assumed default configuration. Testing every combination of factors would have required more time or computational resources that were not available. Thus, not much about the possible interactions between these factors can be known, such as whether the benefits of the proposed model extensions actually stack or whether there would be interference. We are therefore fairly restricted in making any conclusions on what constitutes the optimal configuration of G/GRADE.

Another noteworthy limitation of our research is the lack of a proper assessment of the scalability of the presented framework. Despite the considerable theoretical potential and the success of other researchers in showing scalability of generative representations, none of the design tasks employed here can be regarded as substantial enough to allow proper conclusions on this issue. Large problems either require evaluation times that are not sufficiently tolerable to allow for a comprehensive study, or constitute very artificial problems, which we wished to avoid here. However, some of the original justifications for the graph grammar systems therefore remain unsupported, since the chosen set of problem tasks, albeit diverse, does not exclude that alternative coding schemes, including simple direct schemes, could have also produced competitive results. This is an open question for future research, together with the items proposed in the next section.

8.3 Future Work

Our understanding of graph grammar evolution is necessarily limited by the novelty of this approach and the lack of systematic research in related fields of study. There are many unanswered questions that remain and unsatisfactory results that must be addressed. Consequently, we can suggest a plethora of possible directions for future investigation, the following being the most immediately promising.

8.3.1 Strong Graph Bias

The most significant impact on performance in our study was due to a label matching scheme that improves the likelihood of edges being formed. Thus, the method by which graphs are constructed is evidently important, and there is certainly further room for improvement. In particular, we have observed many productions in the population that define no outgoing tentacles and thus have no bearing on performance (e.g. as seen in figure 7.9). Even without specific knowledge of the problem domain, we hold some knowledge of what makes a good graph and hence a good graph grammar, and this knowledge should be applied.

Section 5.4 presented a construction scheme that precludes pseudographs, but the extent to which we can bias the graph by changing the interpretation of productions is limited. What is needed is a less random approach to choosing cellular graph components for network variation. Each component should be seen as fulfilling a local objective consistent with the expected solution, such as leaving no dangling tentacles. A simple way of achieving this is to build networks from predefined graph templates with a greatly restricted set of variation operators. Many self-defeating sequences of changes would thereby be avoided, and the changes that do take place have a higher likelihood of actually affecting the performance of the network and move evolution forward. Open questions, however, exist in the definition and matching of templates to a given problem task and on the exact trade-off between design freedom and performance.

8.3.2 Real Space Evolution

The use of continuous labels and addition of node offsets has effectively assigned positions to the components of the graph model, and these positions can be extended into any dimension. The computer network is an example problem where we exploited this to actually be part of the solution. However, as labelling is performed randomly, the success potential of this is limited. Unifying the evolution of the topology, with evolution of labels, and optionally the evolution of node or edge attributes, would open up a number of additional real-world problem domains in design, such as architecture, for which the current framework is not well suited. Indeed, finding the optimal connectivity between nodes is often only a small part of the problem. The challenge lies in ensuring that its optimisation does not interfere with the optimisation of the other parts. In this thesis, for example, it required the separation of weights from topology (in section 5.1.3.3), so as to avoid constraints in the representation of the latter affecting the former. This is not the most elegant solution, however, and finding a more cohesive model of graph grammar and general attribute evolution would certainly go a long way towards making G/GRADE more widely applicable.

8.3.3 Production Inheritance

The presented system of grammar evolution requires productions to be copied and changed if they call other productions that have been copied and changed. Not only is this computationally expensive, it also affects the composition of the grammar in potentially adverse ways, e.g. by producing large quantities of near-duplicates of the same production. The straightforward alternative to this, also discussed in section 4.2, is to separate the unified grammar into many smaller, network-specific grammars, so that changes to a network have no effect on other networks. It is quite difficult, however, to preserve modularity while exchanging productions between grammars, because of the complex network of production references and possible identity conflicts with exchanged productions. Quite a bit of copying and changing is inevitable here as well, so we believe the best choice is a hybrid model: networks are represented separately, but can cross-reference productions according to an inheritance tree. The model is illustrated in figure 8.1 and introduces inheritance, as typically employed in object-oriented design, into grammar evolution.

Each production would belong to a class and can call other productions by their identifiers. If a called production is not present in the class, we progressively look up the parent class and its ancestors until the production is found. Productions from other classes can be explicitly referenced, but otherwise follow the same rules. Only one production identifier corresponds to the starting production, but now each class is tagged as to whether it constitutes a network or

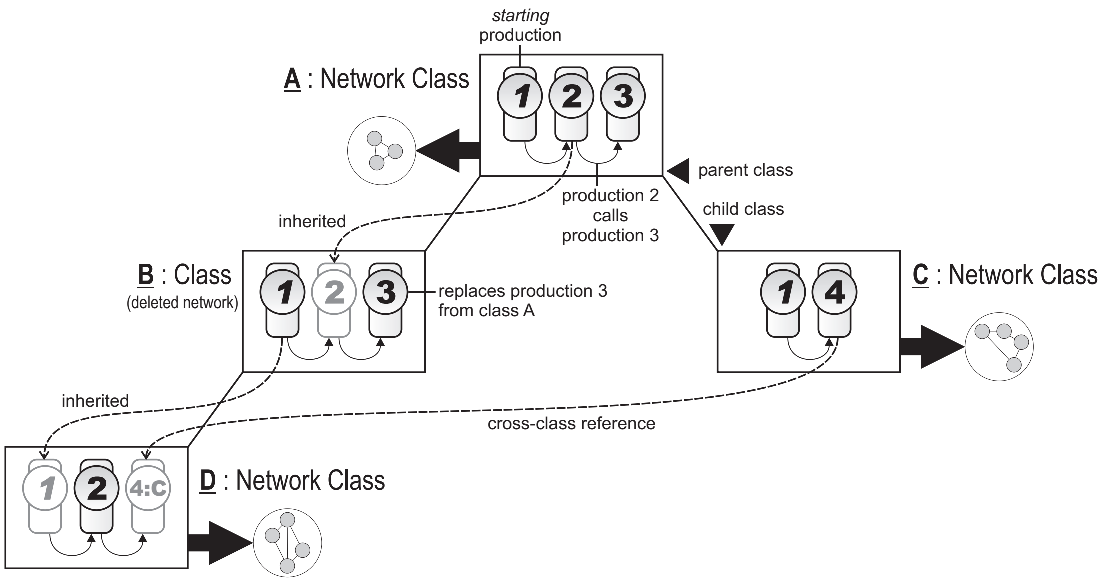


Figure 8.1: The proposed production inheritance scheme defines different networks as classes that define productions, but not all the required productions are defined in each class. Productions of other classes can be reused through references to these classes (as with production 4 of class C) or through inheritance, as shown.

not. A new class is created from an existing class by creating a production in the new class that differs from a production in the existing class – no copying of productions, other than the ones directly mutated, is needed. In other aspects the above model would match G/GRADE, as classes represent networks only if the networks are selected, and productions (and classes) only remain as long as they contribute to networks.

Introducing object-oriented concepts into graph grammar evolution seems a promising approach because of the common emphasis on reuse; it has been previously suggested by Lucas (2002) and fits well with current trends towards object-oriented evolution (Agapitos and Lucas, 2006). As there may be numerous benefits because of (and beyond) representational efficiency, more research into this is patently required.

8.3.4 Grammar-Guided Search

Finally, this study has a major outstanding question – how does the grammar help guide evolution? From a given starting production only one and the same network is derived; any changes to expressed productions are random. Choosing productions from a pool of productions that all relate to the problem being solved may have possible benefits for the search process, but no theoretical model of this has yet been established, and it is unlikely to represent a potent guidance measure. Section 7.4.2 hence introduced an additional statistic that is assigned to productions and used to bias search, yet it has no significant effect other than to reduce redundancy in the grammar. Indeed, we are not clear on how search guidance might benefit a stalled search. However, exploring larger neutral networks beyond existing productions may be the key to this. We would need to establish a more permissible selection scheme so that exploratory mutations into distant search regions become viable, which, in combination with ACO-like exploitation, could ultimately improve the convergence characteristics of graph grammar evolution.

Appendix A

Complete Tables of Results

The performance statistics of all the experiments referred to in this document are shown in the tables below, grouped by the problem task. Please see chapters 5, 6, and 7 for visualisations and discussions of these results. Each table displays the statistics for every tested parameter setup of G/GRADE on a particular problem task. The left-most column describes the experiment and experiment series; the tick ($\surd$) signifies the success rate of the experiment (i.e. the percentage of runs that produce performance-optimal solutions); *MCE* denotes the minimum computational effort for a success probability of 99% (see Koza, 1992); *Min MSE* is the mean squared error of the minimum-error solution of the final generation (showing the minimum, mean, and maximum of the distribution); and *Size* is the average size of a solution in the final generation or the size of the minimum-error solution, as indicated. Standard deviations are listed after each ±. If the success rate is shown in *italic*, it is significantly different than the default (according to a two-tailed Z-test; $p < 0.05$). If the mean performance is shown in italic, the distribution of results is significantly different for this experiment than for the default (according to a Wilcoxon rank sum test; $p < 0.05$). Please note that no Bonferroni correction is applied, so not all the significant results we found will hold at once.

A.1 Binomial-3 Regression

Parameters	$\surd$	**MCE**	**Min MSE**			**Size**	
		×1000	Min	Mean	Max	Mean	Min Error
Default	71%	55	0.0000	0.0540±0.1064	0.3138	18.60±4.27	25.77±7.49
Discrete Labels							
0.9/100/R	*26%*	184	0.0000	*0.1163*±0.1459	1.0440	12.97±2.18	20.36±5.83
0.5/100/R	*12%*	567	0.0000	*0.1980*±0.2843	1.0440	12.39±2.05	18.86±6.31

0.0/100/R	*0%*	N/A	1.0440	*5.4877*±2.3418	7.9916	5.25±2.82	6.12±4.11
0.9/100	*15%*	564	0.0000	*0.0805*±0.1175	1.0440	13.89±1.72	22.42±5.61
0.5/100	*1%*	6711	0.0000	*0.1898*±0.2985	1.0440	13.06±2.09	19.82±5.14
0.0/100	*0%*	N/A	1.0440	*5.4877*±2.3418	7.9916	5.17±2.64	5.92±3.55
0.9/5/R	*17%*	297	0.0000	*0.7386*±1.7317	7.1074	12.02±2.57	18.03±5.66
0.5/5/R	*12%*	457	0.0000	*0.3638*±1.0639	5.4440	12.09±1.79	17.95±4.24
0.0/5/R	*4%*	1281	0.0000	*0.7502*±1.6021	5.4440	11.48±2.12	16.75±5.04
0.9/5	*14%*	513	0.0000	*0.7456*±1.7379	7.1074	12.97±3.11	20.33±6.88
0.5/5	*5%*	1208	0.0000	*0.3690*±1.0755	5.4440	12.98±2.40	20.07±6.39
0.0/5	*8%*	932	0.0000	*0.8244*±1.5956	5.4440	11.83±2.85	17.57±6.35
Node Offset							
No Offset	66%	72	0.0000	0.0844±0.1639	1.0440	19.69±6.16	27.35±10.12
Point Offset	69%	76	0.0000	0.0550±0.1115	0.5710	19.99±5.10	28.84±9.38
Tunnel Offset	66%	85	0.0000	0.0631±0.1623	1.0440	20.13±5.99	28.93±10.86
Modularity							
Selective	65%	72	0.0000	0.0468±0.0874	0.3138	19.91±6.52	28.00±12.47
Global IO (0.1)	76%	66	0.0000	0.0414±0.0929	0.3138	18.65±5.10	25.81±9.67
Global IO (0.5)	62%	90	0.0000	0.0729±0.1483	0.9963	19.93±5.25	27.32±9.21
Gluing Models							
Implicit (Cyclic)	*53%*	120	0.0000	*0.2138*±0.3169	1.0440	17.09±4.48	22.84±7.80
Explicit (Cyclic)	*37%*	184	0.0000	*0.2415*±0.2659	1.0440	27.77±6.30	36.20±10.59
Exp. (C./Simple)	62%	93	0.0000	0.1044±0.1675	1.0440	27.90±6.06	34.97±10.61
Implicit Nonterm.	69%	63	0.0000	0.0582±0.1109	0.3138	18.76±3.73	23.83±5.61
Simple Graphs							
Terminal Only	*58%*	87	0.0000	0.0478±0.0852	0.3138	21.80±4.98	29.40±9.01
Term. + Nont.	*60%*	82	0.0000	0.0504±0.0894	0.3138	20.76±4.46	27.92±7.44
Diversity							
Entropy	*82%*	46	0.0000	*0.0155*±0.0389	0.2413	20.72±15.98	30.40±28.21
Case Entropy	*79%*	45	0.0000	0.0289±0.0743	0.3138	19.21±5.17	29.67±10.80
ND Entropy	73%	58	0.0000	0.0196±0.0339	0.1266	20.41±5.80	30.88±21.95
Distance	*57%*	104	0.0000	0.0642±0.0997	0.3138	40.93±79.09	29.08±10.71
Case Distance	64%	84	0.0000	0.0599±0.1072	0.3138	72.41±110.28	39.42±63.86
ND Distance	76%	69	0.0000	0.0235±0.0515	0.3138	21.60±39.64	32.38±17.45
Case Pareto	*57%*	91	0.0000	0.0594±0.0862	0.3138	52.09±32.10	46.15±20.70
Age	*79%*	60	0.0000	0.0198±0.0476	0.3138	12.42±3.00	26.12±10.34
Sequence	*81%*	52	0.0000	0.0253±0.0684	0.3138	20.80±13.91	39.05±20.94
Island Model							
2 Islands	76%	62	0.0000	0.0350±0.0862	0.3138	15.03±2.85	25.38±8.10
5 Islands	*84%*	51	0.0000	*0.0145*±0.0428	0.3138	20.35±5.78	38.90±21.46
20 Islands	71%	77	0.0000	0.0298±0.1028	0.9963	30.45±9.23	46.11±21.80
5 Isl. (Local)	78%	63	0.0000	0.0200±0.0469	0.3138	19.89±5.35	35.14±15.36
5 Isl. (Isolated)	66%	83	0.0000	0.0507±0.1295	1.0440	16.59±3.87	27.66±9.59
Mutation Rate							
1 Mutation	*0%*	N/A	7.1074	*8.0034*±1.4440	10.3074	2.37±0.86	2.44±0.90
2 Mutations	*34%*	118	0.0000	*3.9241*±3.5374	7.1074	10.80±9.38	14.46±13.94
4 Mutations	*90%*	40	0.0000	*0.0108*±0.0407	0.3138	15.97±3.14	24.46±7.97
8 Mutations	*88%*	44	0.0000	*0.0125*±0.0437	0.3138	14.26±4.41	31.91±11.36
1(+0.5) Targets	78%	58	0.0000	0.0274±0.0665	0.3138	18.73±5.62	27.36±10.21
2 Targets	75%	71	0.0000	0.0328±0.0907	0.6964	14.14±3.67	25.04±7.49
4 Targets	*31%*	243	0.0000	*0.1064*±0.1197	0.3138	13.64±3.57	34.71±16.09
8 Targets	*28%*	297	0.0000	*0.1341*±0.1623	1.0440	14.46±4.37	41.20±23.17
Population Size							
5 (1000 Gen.)	*28%*	68	0.0000	*0.3374*±0.7538	7.1074	19.59±9.94	32.41±16.13
5 (4000 Gen.)	70%	61	0.0000	0.0529±0.1038	0.3138	19.00±8.82	30.59±13.83
80 (250 Gen.)	71%	80	0.0000	0.0673±0.1489	1.0376	20.34±5.88	24.70±7.05
80 (1000 Gen.)	*96%*	77	0.0000	*0.0029*±0.0143	0.0726	22.34±4.27	23.94±4.64

50k Generations							
Default	*100%*	53	0.0000	*0.0000*±0.0000	0.0000	17.21±5.06	34.36±13.02
Enhanced	*100%*	41	0.0000	*0.0000*±0.0000	0.0000	18.96±5.15	41.27±14.03
Enhanced @ 1K	77%	63	0.0000	*0.0137*±0.0274	0.0726	20.38±6.41	42.7±14.86
Adaptive Search							
Target	64%	77	0.0000	0.0791±0.1264	0.3138	17.96±4.50	24.07±7.42
Lineage	79%	50	0.0000	0.0345±0.0863	0.3138	19.74±4.67	27.64±8.46
Target+Lineage	74%	53	0.0000	0.0432±0.0943	0.3138	19.01±5.18	26.11±8.50
Size Objective							
Size Shared	*21%*	360	0.0000	*0.4511*±0.6435	3.2090	8.84±5.87	51.20±54.03
No Size	*31%*	399	0.0000	*0.1885*±0.2464	1.0440	47.29±37.22	47.19±37.15
C.Entr.+No Size	*86%*	20	0.0000	*0.0086*±0.0329	0.2709	377.6±850.3	359.7±900.4

Table A.1: Results for the Binomial-3 Regression problem.

A.2 6th-order Polynomial Regression

Parameters	$\surd$	**MCE**	**Min MSE**			**Size**	
		×1000	Min	Mean	Max	Mean	Min Error
Default	0%	N/A	0.0083	0.0083± 0.0000	0.0083	1.00± 0.00	1.00± 0.00
Diversity							
Entropy	*96%*	13	0.0000	*0.0002*± 0.0012	0.0083	30.20± 26.18	43.56± 49.33
Case Entropy	*99%*	14	0.0000	*0.0001*± 0.0008	0.0083	26.47± 15.69	35.56± 29.64
ND Entropy	*100%*	16	0.0000	*0.0000*± 0.0000	0.0000	27.75± 19.93	43.78± 52.15
Distance	*16%*	757	0.0000	*0.0027*± 0.0025	0.0083	64.98± 45.52	100.9± 94.10
Case Distance	*15%*	406	0.0000	*0.0014*± 0.0014	0.0083	103.4± 78.30	127.3± 110.0
ND Distance	*53%*	123	0.0000	*0.0021*± 0.0034	0.0083	47.51± 58.73	74.40± 96.69
Case Pareto	*84%*	72	0.0000	*0.0002*± 0.0009	0.0061	96.16± 91.98	118.9± 126.9
Age	0%	N/A	0.0083	0.0083± 0.0000	0.0083	2.08± 0.36	1.00± 0.00
Sequence	1%	11663	0.0000	0.0082± 0.0008	0.0083	5.85± 15.36	5.36± 43.60

Table A.2: Results for the 6th-order Polynomial Regression problem.

A.3 Random Bit Sequence

Parameters	$\surd$	**MCE**	**Min MSE**			**Size**	
		×1000	Min	Mean	Max	Mean	Min Error
Default	1%	36390	0.0000	0.1338±0.0569	0.2500	12.97±4.87	13.89±6.93
Discrete Labels							
0.9/100/R	2%	17893	0.0000	*0.4369*±0.1375	0.5000	2.89±4.90	4.11±7.49
0.5/100/R	*0%*	N/A	0.0625	*0.4831*±0.0710	0.5000	1.93±4.49	2.40±6.45
0.0/100/R	*0%*	N/A	0.5000	*0.5000*±0.0000	0.5000	1.00±0.00	1.00±0.00
0.9/100	*0%*	N/A	0.0625	*0.4400*±0.1287	0.5000	4.35±11.97	5.38±13.19
0.5/100	*0%*	N/A	0.1250	*0.4813*±0.0777	0.5000	1.94±4.36	2.61±7.15
0.0/100	*0%*	N/A	0.5000	*0.5000*±0.0000	0.5000	1.00±0.00	1.00±0.00
0.9/5/R	*0%*	N/A	0.0625	*0.4400*±0.1314	0.5000	4.09±9.52	5.24±12.43
0.5/5/R	*0%*	N/A	0.0625	*0.4838*±0.0730	0.5000	1.51±2.30	1.52±2.36
0.0/5/R	*0%*	N/A	0.5000	*0.5000*±0.0000	0.5000	1.00±0.00	1.00±0.00
0.9/5	*0%*	N/A	0.0625	*0.4406*±0.1298	0.5000	4.21±9.62	5.18±11.44

0.5/5	*0%*	N/A	0.1250	*0.4806*±0.0783	0.5000	1.66±2.85	2.20±5.22
0.0/5	*0%*	N/A	0.5000	*0.5000*±0.0000	0.5000	1.00±0.00	1.00±0.00
Node Offset							
No Offset	1%	9951	0.0000	*0.1688*±0.0593	0.3125	13.15±5.92	13.73±6.11
Point Offset	*11%*	2782	0.0000	*0.0831*±0.0471	0.1875	11.91±4.72	12.72±5.29
Tunnel Offset	*7%*	4926	0.0000	*0.0994*±0.0487	0.1875	11.90±4.28	12.23±4.61
Modularity							
Selective	3%	9503	0.0000	0.1313±0.0579	0.2500	13.14±6.00	14.36±7.20
Global IO (0.1)	4%	6646	0.0000	0.1350±0.0594	0.2500	12.42±4.60	12.84±4.74
Global IO (0.5)	1%	27944	0.0000	*0.1594*±0.0618	0.2500	11.10±5.69	11.68±6.19
Gluing Models							
Explicit (Cyclic)	*0%*	N/A	0.0625	*0.2213*±0.0605	0.3125	21.07±6.99	22.06±7.91
Exp. (C./Simple)	2%	18076	0.0000	0.1369±0.0675	0.3125	25.86±8.58	28.35±13.12
Implicit Nonterm.	1%	12154	0.0000	0.1381±0.0578	0.2500	12.73±5.12	13.45±6.73
Simple Graphs							
Terminal Only	1%	18764	0.0000	0.1350±0.0560	0.2500	12.44±5.39	13.57±8.29
Term. + Nont.	*6%*	4866	0.0000	0.1275±0.0621	0.2500	13.62±8.48	14.13±8.79
Diversity							
Entropy	4%	7874	0.0000	*0.1088*±0.0566	0.2500	12.91±4.49	13.78±5.76
Case Entropy	*21%*	1442	0.0000	*0.0650*±0.0443	0.1875	15.64±6.43	17.40±12.83
ND Entropy	*16%*	1796	0.0000	*0.0763*±0.0483	0.1875	15.08±6.90	15.83±7.48
Distance	2%	12522	0.0000	0.1188±0.0506	0.2500	374.58±229.64	31.79±35.68
Case Distance	*24%*	1321	0.0000	*0.0625*±0.0453	0.1875	341.66±240.34	26.78±16.60
ND Distance	*27%*	1153	0.0000	*0.0588*±0.0443	0.1875	299.00±110.55	49.57±38.63
Case Pareto	*29%*	1064	0.0000	*0.0444*±0.0285	0.0625	30.79±7.87	36.47±24.60
Age	*8%*	2450	0.0000	*0.0963*±0.0536	0.2500	11.55±5.00	16.34±9.81
Sequence	*11%*	2915	0.0000	*0.0819*±0.0423	0.1875	14.50±3.80	17.03±7.24
Island Model							
2 Islands	4%	6970	0.0000	0.1238±0.0582	0.2500	15.36±9.05	16.65±10.90
5 Islands	*6%*	4687	0.0000	0.1163±0.0576	0.2500	15.61±9.19	16.95±11.48
20 Islands	*10%*	2944	0.0000	*0.1019*±0.0600	0.2500	19.80±7.93	20.54±12.50
80 Islands	*13%*	2532	0.0000	*0.0938*±0.0537	0.2500	22.41±7.84	20.53±10.08
5 Isl. (Local)	*6%*	5046	0.0000	*0.1088*±0.0593	0.2500	15.52±7.41	16.07±7.90
5 Isl. (Isolated)	*8%*	4395	0.0000	0.1194±0.0653	0.2500	13.96±8.41	16.45±20.00
Mutation Rate							
1 Mutation	1%	26475	0.0000	*0.2106*±0.0680	0.3125	9.35±3.61	9.50±3.75
2 Mutations	*6%*	4255	0.0000	*0.1156*±0.0637	0.2500	13.70±6.12	14.09±6.37
4 Mutations	2%	18258	0.0000	0.1206±0.0472	0.2500	10.45±3.89	16.94±27.73
8 Mutations	*0%*	N/A	0.0625	*0.1663*±0.0512	0.2500	5.99±1.07	14.16±6.20
1(+0.5) Targets	4%	6970	0.0000	*0.1106*±0.0568	0.2500	12.95±4.31	14.43±6.00
2 Targets	1%	36390	0.0000	*0.1481*±0.0580	0.2500	8.15±2.53	13.65±6.39
4 Targets	*0%*	N/A	0.0625	*0.1675*±0.0509	0.3125	6.78±0.87	12.50±3.71
8 Targets	*0%*	N/A	0.0625	*0.1663*±0.0496	0.3125	6.93±0.83	12.55±4.02
Population Size							
5 (1000 Gen.)	0%	N/A	0.0625	*0.3238*±0.1386	0.5000	8.44±4.92	11.69±14.39
5 (16000 Gen.)	*7%*	2940	0.0000	*0.1038*±0.0556	0.2500	11.08±4.60	16.99±12.08
20 (1000 Gen.)	3%	2405	0.0000	*0.1688*±0.0882	0.5	13.93±12.23	15.42±14.80
20 (4000 Gen.)	*13%*	2405	0.0000	*0.0956*±0.0586	0.2500	16.36±9.46	18.19±11.26
50k Generations							
Default	*50%*	5076	0.0000	*0.0333*±0.0357	0.1250	18.14±8.75	25.53±19.24
Enhanced	*77%*	1649	0.0000	*0.0146* ±0.0269	0.0625	14.48±7.72	23.73±16.04
Enhanced @ 1K	*17%*	1649	0.0000	*0.0625* ±0.0367	0.1250	14.33±5.33	17.67±7.59
Adaptive Search							
Target	3%	8044	0.0000	0.1225±0.0525	0.2500	13.70±7.32	14.40±7.67
Lineage	4%	8416	0.0000	0.1131±0.0545	0.2500	13.74±6.69	14.66±7.61
Target+Lineage	3%	9685	0.0000	0.1338±0.0628	0.2500	12.06±4.55	12.57±4.61

Size Objective							
Size Shared	*0%*	N/A	0.0625	*0.1556*±0.0606	0.3125	18.36±8.55	26.68±37.82
No Size	*18%*	1376	0.0000	*0.0838*±0.0604	0.2500	23.32±19.46	23.25±19.28
C.Entr.+No Size	*96%*	79	0.0000	*0.0025*±0.0123	0.0625	89.05±73.09	78.58±42.91

Table A.3: Results for the Random Bit Sequence circuit design problem.

A.4 6-bit Multiplexer

Parameters	√	**MCE**	**Min MSE**			**Size**	
		×1000	Min	Mean	Max	Mean	Min Error
Default	81%	413	0.0000	0.0203± 0.0438	0.1250	30.46± 2.21	32.87± 7.31
Diversity							
Entropy	74%	432	0.0000	0.0228± 0.0419	0.1250	30.91± 2.00	34.46± 11.31
Case Entropy	*90%*	226	0.0000	*0.0083*± 0.0265	0.1250	33.55± 5.13	35.65± 12.76
ND Entropy	*91%*	236	0.0000	*0.0081*± 0.0276	0.1250	33.14± 4.13	36.46± 15.63
Distance	*63%*	660	0.0000	*0.0392*± 0.0542	0.1250	31.67± 2.60	34.17± 11.55
Case Distance	86%	368	0.0000	0.0114± 0.0301	0.1250	37.12± 7.20	36.46± 18.45
ND Distance	85%	341	0.0000	0.0152± 0.0379	0.1250	37.35± 7.12	36.58± 17.20
Case Pareto	*54%*	726	0.0000	*0.0563*± 0.0619	0.1250	42.42± 24.40	39.25± 16.08
Age	*72%*	583	0.0000	0.0333± 0.0544	0.1250	35.48± 5.57	37.93± 15.82
Sequence	*61%*	745	0.0000	*0.0488*± 0.0613	0.1250	31.28± 6.72	27.52± 13.01

Table A.4: Results for the 6-bit Multiplexer problem.

A.5 Backpropagation MLP

Parameters	**Min MSE**			**Size**
	Min	Mean	Max	
Default	0.0165	0.0549±0.0534	0.3333	67.65±34.05
Discrete Labels				
0.9/100/R	0.0242	*0.114127*±0.0478	0.2223	52.03±43.51
0.5/100/R	0.0229	*0.126753*±0.0504	0.2224	53.26±37.43
0.0/100/R	0.1273	*0.2727*±0.0643	0.3333	17.73±24.22
0.9/100	0.0649	*0.167219*±0.0452	0.2336	51.59±38.33
0.5/100	0.0353	*0.152215*±0.0464	0.2226	44.92±34.52
0.0/100	0.1273	*0.273557*±0.0633	0.3333	18.34±24.12
0.9/5/R	0.0270	*0.115099*±0.0463	0.2223	45.04±28.75
0.5/5/R	0.0254	*0.124085*±0.0479	0.2227	46.43±37.26
0.0/5/R	0.0241	*0.137174*±0.0582	0.2229	44.12±36.38
0.9/5	0.0619	*0.166669*±0.0465	0.2230	50.56±37.71
0.5/5	0.0349	*0.157057*±0.0520	0.2294	52.34±34.34
0.0/5	0.0264	*0.156989*±0.0535	0.2227	63.35±37.88
Node Offset				
No Offset	0.0340	*0.1647*±0.0668	0.3333	49.08±31.88
Point Offset	0.0172	0.0544±0.0432	0.1642	72.79±30.58
Tunnel Offset	0.0151	0.0548±0.0535	0.3333	75.38±37.02
Modularity				
Selective	0.0259	*0.0614*±0.0068	0.0749	23.48±25.72

Global IO (0.1)	0.0184	0.0545±0.0489	0.3333	78.56±35.69
Global IO (0.5)	0.0151	0.0560±0.0446	0.3333	73.26±39.50
Gluing Models				
Implicit (Cyclic)	0.0216	*0.1371*±0.0731	0.3333	51.88±33.44
Explicit (Cyclic)	0.0284	*0.1348*±0.0525	0.2513	32.99±18.09
Exp. (C./Simple)	0.0353	*0.1145*±0.0392	0.1965	46.01±17.69
Implicit Nonterm.	0.0202	0.0634±0.0572	0.3333	59.51±22.83
Simple Graphs				
Terminal Only	0.0164	0.0541±0.0392	0.1781	58.16±32.39
Term. + Nont.	0.0217	0.0491±0.0476	0.3333	60.13±29.60

Table A.5: Results for the Backpropagation MLP (Iris dataset) problem.

A.6 Pole Balancing

Parameters	√	**MCE**	**Min MSE**			**Size**	
		×1000	Min	Mean	Max	Mean	Min Error
Default	82%	40	0.0010	0.0016±0.0015	0.0071	7.65±1.42	8.99±3.41
Diversity							
Entropy	*90%*	32	0.0010	0.0013±0.0011	0.0101	7.45±0.72	7.82±1.58
Distance	*71%*	71	0.0010	0.0019±0.0017	0.0084	6.16±0.72	9.15±2.21
Age	*100%*	14	0.0010	*0.0010*±0.0000	0.0010	7.11±1.03	9.71±2.14
Sequence	*98%*	23	0.0010	*0.0010*±0.0003	0.0042	9.14±4.38	14.02±13.23
Island Model							
2 Islands	*96%*	26	0.0010	*0.0012*±0.0009	0.0064	6.97±0.73	8.68±1.82
5 Islands	*98%*	20	0.0010	*0.0011*±0.0006	0.0067	8.29±2.21	10.41±3.74
20 Islands	*95%*	23	0.0010	*0.0012*±0.0009	0.0068	12.09±9.20	12.56±9.93
5 Isl. (Local)	*100%*	20	0.0010	*0.0010*±0.0000	0.0010	8.13±1.44	9.86±2.83
5 Isl. (Isolated)	*96%*	32	0.0010	*0.0011*±0.0004	0.0046	7.72±1.87	9.25±1.96
Mutation Rate							
1 Mutation	76%	42	0.0010	0.0116±0.0435	0.2000	7.15±1.56	8.20±2.14
2 Mutations	76%	52	0.0010	0.0019±0.0019	0.0079	7.58±1.03	8.92±2.65
4 Mutations	78%	55	0.0010	0.0017±0.0016	0.0078	6.61±1.77	9.64±3.43
8 Mutations	*64%*	96	0.0010	*0.0019*±0.0015	0.0067	8.32±6.47	17.99±19.41
1(+0.5) Targets	82%	40	0.0010	0.0016±0.0016	0.0088	7.51±0.90	8.69±2.12
2 Targets	76%	46	0.0010	0.0018±0.0019	0.0105	7.46±0.72	8.77±1.86
4 Targets	79%	47	0.0010	0.0017±0.0016	0.0085	7.45±0.64	8.76±1.74
8 Targets	80%	45	0.0010	0.0017±0.0016	0.0085	7.49±0.60	8.73±1.59
Population Size							
5 (1000 Gen.)	*58%*	27	0.0010	*0.0028*±0.0029	0.0109	8.43±3.54	10.56±4.95
5 (4000 Gen.)	*89%*	36	0.0010	0.0014±0.0012	0.0064	7.41±4.49	10.23±7.90
80 (250 Gen.)	*49%*	141	0.0010	*0.0026*±0.0021	0.0087	6.61±1.75	9.02±2.34
80 (1000 Gen.)	*89%*	140	0.0010	0.0013±0.0011	0.0058	8.52±1.21	8.41±1.27
50k Generations							
100k Cycles							
Default	*100%*	51	1.0e-5	*1.0e-5*±0.0000	1.0e-5	7.13±1.22	11.00±2.52
Enhanced	*100%*	22	1.0e-5	*1.0e-5*±0.0000	1.0e-5	8.56±2.97	16.50±12.02
Default @ 1k	73%	51	1.0e-5	0.0012±0.0026	0.0086	7.19±1.20	10.40±2.66
Enhanced @ 1k	*97%*	28	1.0e-5	*4.4e-5*±1.9e-4	0.0010	8.41±2.61	15.3±9.59

Adaptive Search							
Target	77%	48	0.0010	0.0019±0.0018	0.0086	7.54±0.95	8.22±2.27
Lineage	86%	34	0.0010	0.0016±0.0016	0.0076	7.27±0.43	7.74±1.76
Target+Lineage	84%	37	0.0010	0.0016±0.0015	0.0070	7.56±0.83	8.08±1.97
Size Objective							
Size Shared	87%	46	0.0010	0.0016±0.0017	0.0086	7.50±0.74	8.22±2.25
No Size	*42%*	76	0.0010	*0.0032*±0.0025	0.0112	34.91±41.94	35.79±42.71
Entropy+No Size	*51%*	47	0.0010	*0.0028*±0.0026	0.0116	91.13±119.06	31.28±35.75

Table A.6: Results for the double Pole Balancing problem.

A.7 Computer Network Topology

Parameters	√	**MCE**		**Min MSE**		**Cost** (×1000000)	
		×1000	Min	Mean	Max	Mean	Min Error
Default	23%	1366	0.0000	0.1801± 0.1668	0.5714	5.14± 2.40	6.91± 6.76
Node Offset							
No Offset	*5%*	6127	0.0000	*0.3028*± 0.2108	0.7143	7.34± 2.33	7.72± 2.76
Point Offset	28%	1117	0.0000	0.1556± 0.1647	0.5400	5.81± 3.25	7.08± 5.98
Tunnel Offset	31%	947	0.0000	0.1606± 0.1621	0.5714	5.30± 2.25	6.42± 2.92
Modularity							
Selective	25%	227	0.0000	*0.0999*± 0.1161	0.4118	5.38± 2.38	5.93± 2.79
Global IO (0.1)	27%	1177	0.0000	*0.1194*± 0.1259	0.5000	5.67± 2.58	7.24± 4.90
Global IO (0.5)	*35%*	695	0.0000	*0.0674*± 0.0703	0.2917	5.21± 1.72	6.54± 2.42
Gluing Models							
Explicit (Cyclic)	24%	1142	0.0000	0.1405± 0.1479	0.5714	5.31± 1.76	6.28± 2.25
Exp. (C./Simple)	*46%*	601	0.0000	*0.0848*± 0.1161	0.5000	6.37± 3.20	7.88± 4.79
Implicit Nonterm.	*36%*	817	0.0000	*0.0918*± 0.1261	0.6667	6.44± 4.12	8.75± 8.09
Simple Graphs							
Terminal Only	21%	1472	0.0000	0.1853± 0.1706	0.5714	5.08± 2.59	6.55± 5.26
Term. + Nont.	*42%*	654	0.0000	*0.0798*± 0.0989	0.5000	6.04± 2.32	7.85± 5.90
Diversity							
Entropy	32%	932	0.0000	*0.1270*± 0.1332	0.5000	6.11± 3.62	8.04± 8.05
Case Entropy	29%	1051	0.0000	*0.1117*± 0.1273	0.5000	6.28± 2.45	8.09± 4.42
ND Entropy	31%	1031	0.0000	*0.1120*± 0.1209	0.5000	6.39± 4.33	8.71± 11.26
Distance	20%	1463	0.0000	0.1413± 0.1202	0.5714	6.04± 3.57	8.82± 12.65
Case Distance	25%	1293	0.0000	*0.1342*± 0.1330	0.5000	6.16± 2.80	7.57± 5.60
ND Distance	29%	1051	0.0000	*0.1261*± 0.1333	0.5000	5.97± 2.53	7.29± 3.94
Case Pareto	*40%*	683	0.0000	*0.0627*± 0.0675	0.2857	9.86± 10.46	13.04± 12.91
Age	*36%*	692	0.0000	*0.0881*± 0.0977	0.5000	4.69± 3.09	7.92± 7.52
Sequence	*42%*	422	0.0000	*0.0781*± 0.0894	0.3571	5.01± 2.79	7.91± 5.52
Island Model							
2 Islands	21%	1570	0.0000	0.1690± 0.1494	0.5000	5.91± 3.35	7.52± 7.53
5 Islands	*35%*	722	0.0000	*0.1256*± 0.1427	0.5400	5.93± 3.89	8.89± 14.35
20 Islands	28%	935	0.0000	*0.1015*± 0.1084	0.5000	6.12± 5.32	9.89± 14.33
80 Islands	30%	987	0.0000	*0.1012*± 0.1000	0.3571	7.28± 5.98	9.58± 13.41
5 Isl. (Local)	30%	1041	0.0000	0.1441± 0.1607	0.7143	6.06± 4.55	8.23± 8.51
5 Isl. (Isolated)	27%	1081	0.0000	0.1305± 0.1278	0.5000	5.56± 2.57	8.10± 7.46
Mutation Rate							
1 Mutation	*0%*	N/A	0.2063	*0.8270*± 0.2528	1.0000	1.38± 2.03	1.41± 2.06
2 Mutations	*8%*	4440	0.0000	*0.2663*± 0.1757	0.6667	4.75± 3.77	7.77± 11.81
4 Mutations	*9%*	771	0.0000	0.1775± 0.1165	0.5000	3.37± 2.71	7.06± 7.10

8 Mutations	*15%*	499	0.0000	0.1415± 0.1145	0.5714	2.13± 2.04	7.75± 6.71
1(+0.5) Targets	22%	1430	0.0000	0.1768± 0.1592	0.5714	5.16± 3.24	6.53± 5.29
2 Targets	16%	2114	0.0000	0.1867± 0.1634	0.5714	5.09± 2.23	7.09± 3.69
4 Targets	19%	1462	0.0000	0.1763± 0.1516	0.5714	5.14± 2.28	7.28± 5.45
8 Targets	16%	1988	0.0000	0.1758± 0.1487	0.5476	5.26± 2.21	7.39± 5.63
Population Size							
5 (1000 Gen.)	*0%*	N/A	0.0556	*0.4099*± 0.1609	0.7619	3.00± 1.73	4.31± 2.66
5 (16000 Gen.)	22%	212	0.0000	0.1651± 0.1522	0.5714	4.83± 5.14	7.81± 8.52
20 (1000 Gen.)	*4%*	743	0.0000	*0.3163*± 0.1714	0.7143	4.36± 2.86	5.54± 4.86
20 (4000 Gen.)	29%	743	0.0000	0.1395± 0.1407	0.5714	5.68± 3.70	6.91± 6.76
50k Generations							
Default	*77%*	1366	0.0000	*0.0242*± 0.0531	0.2143	7.65± 4.21	11.45± 8.10
Enhanced	*100%*	600	0.0000	*0.0000*± 0.0000	0.0000	6.74± 2.88	11.78± 3.82
Enhanced @ 1k	*47%*	626	0.0000	*0.0978*± 0.1265	0.5000	5.27± 1.77	8.49± 3.00
Adaptive Search							
Target Choice	*26%*	1281	0.0000	0.1445± 0.1371	0.5714	5.47± 2.86	6.96± 4.57
Lineage Choice	21%	1506	0.0000	0.1716± 0.1523	0.5000	5.47± 3.27	7.21± 6.34
Target+Lineage	24%	1378	0.0000	0.1609± 0.1539	0.5000	5.62± 2.18	6.88± 3.06
Size Objective							
Size Shared	*1%*	30882	0.0000	*0.4003*± 0.1759	0.7619	0.22± 0.68	5.50± 2.82
No Size	*37%*	450	0.0000	*0.0889*± 0.1078	0.5000	13.17± 24.21	11.57± 17.31
C.Entr.+No Size	*54%*	202	0.0000	*0.0473*± 0.0683	0.2857	120.1± 133.3	41.88± 62.62

Table A.7: Results for the Computer Network Topology design problem.

Bibliography

Abbass, H. A. (2003). Speeding up back-propagation using multiobjective evolutionary algorithms. *Neural Computation*, 15(11):2705–2726.

Abbass, H. A. and Deb, K. (2003). Searching under multi-evolutionary pressures. In Fonseca, C., Fleming, P., Zitzler, E., Deb, K., and Thiele, L., editors, *Proceedings of the Second International Conference on Evolutionary Multi-Criterion Optimization*, volume 2632 of *Lecture Notes in Computer Science*, pages 391–404. Springer Verlag.

Abbass, H. A., Hoai, N. X., and McKay, R. I. (2002). AntTAG: a new method to compose computer programs using colonies of ants. In Fogel, D., editor, *Proceedings of the 2002 Congress on Evolutionary Computation*, pages 1654–1659. IEEE Press.

Abbass, H. A. and Sarker, R. (2001). Simultaneous evolution of architectures and connection weights in ANNs. In Kasabov, N. and Woodford, B., editors, *Proceedings of the Fifth Biannual Conference on Artificial Neural Networks and Expert Systems*, pages 16–21. IEEE Press.

Abelson, H. and diSessa, A. A. (1981). *Turtle Geometry.* The MIT Press, Cambridge, USA.

Ackley, D. E. and Littman, M. L. (1994). A case for Lamarckian evolution. In Langton, C. G., editor, *Artificial Life III: Proceedings of the Workshop on Artificial Life*, pages 487–509. Addison-Wesley.

Agapitos, A. and Lucas, S. M. (2006). Learning recursive functions with object oriented genetic programming. In Collet, P., Tomassini, M., Ebner, M., Gustafson, S. M., and Ekárt, A., editors, *Proceedings of the 9th European Conference on Genetic Programming*, volume 3905 of *Lecture Notes in Computer Science*, pages 166–177. Springer Verlag.

Aiyarak, P., Saket, A. S., and Sinclair, M. C. (1997). Genetic programming approaches for minimum cost topology optimisation of optical telecommunication networks. In Zalzala, A., editor, *Proceedings of the 2nd IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, pages 415–420. Institution of Electrical Engineers.

Albert, R. and Barabási, A.-L. (2002). Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47–97.

Alvarez-Buylla, A., Ling, C. Y., and Nottebohm, F. (1992). High vocal center growth and its relation to neurogenesis, neuronal replacement and song acquisition in juvenile canaries. *Journal of Neurobiology*, 23:396–406.

Angeline, P. J. (1997). Subtree crossover: Building block engine or macromutation? In Koza, J. R., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M., Iba, H., and Riolo, R. L., editors, *Proceedings of the Second Annual Conference on Genetic Programming*, pages 9–17. Morgan Kaufmann.

Angeline, P. J. and Pollack, J. B. (1993). Evolutionary module acquisition. In Fogel, D. B. and Atmar, W., editors, *Proceedings of the Second Annual Conference on Evolutionary Programming*, pages 154–163. The MIT Press.

Angeline, P. J., Saunders, G. M., and Pollack, J. B. (1994). An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, 5(1):54–65.

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

Babloyantz, A. and Hiernaux, J. (1974). Models for positional information and positional differentiation. *Proceedings of the National Academy of Sciences, USA*, 71(4):1539–1544.

Bäck, T. (1996). *Evolutionary Algorithms in Theory and Practice.* Oxford University Press, New York, USA.

Bäck, T., Graaf, J. M., Kok, J. N., and Kosters, W. A. (1997). Theory of genetic algorithms. *Bulletin of the EATCS*, 63:161–192.

Baldwin, J. (1896). A new factor in evolution. *American Naturalist*, 30:441–451.

Ballesteros, F. J. and Luque, B. (2005). Order-disorder phase transition in random-walk networks. *Physical Review E*, 71:031104.

Baluja, S. and Caruana, R. (1995). Removing the genetics from the standard genetic algorithm. In Prieditis, A. and Russel, S., editors, *Proceedings of the Twelfth International Conference on Machine Learning*, pages 38–46. Morgan Kaufmann.

Barabási, A.-L. and Albert, R. (1999). Emergence of scaling in random networks. *Science*, 286(5439):509–512.

Bear, M., Connors, B., and Paradiso, M. (2006). *Neuroscience: Exploring the Brain.* Lippincott Williams & Wilkins, Philadelphia, USA.

Bentley, P. J. and Kumar, S. (1999). Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem. In Angeline, P., Michalewicz, Z., Schoenauer, M., Yao, X., and Zalzala, A., editors, *Proceedings of the 1999 Congress on Evolutionary Computation*, pages 35–43. IEEE Press.

Berryman, M. J., Allison, A., and Abbott, D. (2004). Optimizing genetic algorithm strategies for evolving networks. In White, L. B., editor, *Noise in Communication*, volume 5473 of *Proceedings of the SPIE*, pages 122–130. SPIE.

Bersano-Begey, T. (1997). Controlling exploration, diversity and escaping local optima in GP. In Koza, J. R., editor, *Late Breaking Papers at the Genetic Programming 1997 Conference*, pages 7–10. Stanford University.

Bianchini, M. and Gori, M. (1996). Optimal learning in artificial neural networks: a theoretical view. *Neurocomputing*, 13:313–346.

Bleuer, S., Braek, M., Thiele, L., and Zitzler, E. (2001). Multiobjective genetic programming: reducing bloat using SPEA2. In Kim, J.-H., Zhang, B.-T., Fogel, G., and Kuscu, I., editors, *Proceedings of the 2001 Congress on Evolutionary Computation*, volume 1, pages 536–543. IEEE Press.

Blickle, T. and Thiele, L. (1994). Genetic programming and redundancy. In Hopf, J., editor, *Genetic Algorithms within the Framework of Evolutionary Computation (Workshop at KI-94, Saarbrücken)*, pages 33–38. Max-Planck-Institut für Informatik.

Boers, E. J. W. and Kuiper, H. (1992). *Biological metaphors and the design of modular artificial neural networks*. Master's thesis, Departments of Computer Sciences and Experimental and Theoretical Psychology, Leiden University, Leiden, The Netherlands.

Boers, E. J. W. and Sprinkhuizen-Kuyper, I. G. (1995). Using L-systems as graph grammars: G2L-systems. Technical report 95-30, Department of Computer Science, Leiden University, The Netherlands.

Boers, E. J. W. and Sprinkhuizen-Kuyper, I. G. (2001). Combined biological metaphors. In Patel, M. J., Honavar, V., and Balakrishnan, K., editors, *Advances in the evolutionary synthesis of intelligent agents*, book chapter 6, pages 153–183. The MIT Press.

Bohland, J. W. and Minai, A. A. (2000). Efficient associative memory using small-world architecture. *Neurocomputing*, 38-40:489–496.

Bolouri, H., Adams, R., George, S., and Rust, A. G. (1998). Molecular self-organisation in a developmental model for the evolution of large-scale artificial neural networks. In Usui, S. and Omori, T., editors, *Proceedings of the International Conference on Neural Information Processing and Intelligent Information Systems*, pages 797–800. Springer Verlag.

Bongard, J. C. and Pfeifer, R. (2001). Repeated structure and dissociation of genotypic and phenotypic complexity in artificial ontogeny. In Spector, L. and Goodman, E. D., editors, *Proceedings of the 2001 Genetic and Evolutionary Computation Conference*, pages 829–836. Morgan Kaufmann.

Bowers, C. P. (2005). Formation of modules in a computational model of embryogeny. In Corne, D., Michalewicz, Z., McKay, B., Eiben, G., Fogel, D., Fonseca, C., Greenwood, G., Raidl, G., Tan, K. C., and Zalzala, A., editors, *Proceedings of the 2005 Congress on Evolutionary Computation*, pages 537–542. IEEE Press.

Bui, L. T., Branke, J., and Abbass, H. A. (2005). Multiobjective optimization for dynamic environments. In Corne, D., Michalewicz, Z., McKay, B., Eiben, G., Fogel, D., Fonseca, C., Greenwood, G., Raidl, G., Tan, K. C., and Zalzala, A., editors, *Proceedings of the 2005 Congress on Evolutionary Computation*, pages 2349–2356. IEEE Press.

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

Caudell, T. P. and Dolan, C. P. (1989). Parametric connectivity: training of constrained networks using genetic algorithms. In Schaffer, J. D., editor, *Proceedings of the Third International Conference on Genetic Algorithms and Their Applications*, pages 370–374. Morgan Kaufmann.

Cecchi, G. A., Petreanu, L. T., Alvarez-Buylla, A., and Magnasco, M. O. (2001). Unsupervised learning and adaptation in a model of adult neurogenesis. *Journal of Computational Neuroscience*, 11(2):175–182.

Chowdhury, D., Nishinari, K., and Schadschneider, A. (2004). Self-organized patterns and traffic flow in colonies of organisms: from bacteria and social insects to vertebrates. *Phase Transitions*, 77:601–624.

Chu, P. and Jones, R. (1999). Design techniques of FPGA based random number generator. In Katz, R., editor, *Proceedings of the 2nd Annual Military and Aerospace Applications of Programmable Devices and Technologies (MAPLD) Conference*. The Johns Hopkins University – Applied Physics Laboratory.

Chua, L. O. and Yang, L. (1988a). Cellular neural networks: applications. *IEEE Transactions of Circuits and Systems*, 35(10):1273–1290.

Chua, L. O. and Yang, L. (1988b). Cellular neural networks: theory. *IEEE Transactions of Circuits and Systems*, 35(10):1257–1272.

Cohen, R. and Havlin, S. (2003). Scale-free networks are ultrasmall. *Physical Review Letters*, 90(5):058701.

Collins, J. J. and Eaton, M. (1997). A global representation scheme for genetic algorithms. In Reusch, B., editor, *Proceedings of the 1997 International Conference on Computational Intelligence*, volume 1226 of *Lecture Notes in Computer Science*, pages 1–16. Springer Verlag.

Cun, Y., Denker, J., and Solla, S. (1990). Optimal brain damage. In Touretzky, D. S., editor, *Advances in neural information processing systems*, volume 2, pages 598–605. Morgan Kaufmann.

Daida, J. M., Bertram, R. R., Stanhope, S. A., Khoo, J. C., Chaudhary, S. A., Chaudhri, O. A., and II, J. A. P. (2001). What makes a problem GP-hard? Analysis of a tunably difficult problem in genetic programming. *Genetic Programming and Evolvable Machines*, 2(2):165–191.

Darwin, C. (1859). *On the Origin of Species By Means of Natural Selection*. Murray, London.

Dawkins, R. (1983). Universal darwinism. In Bendall, D. S., editor, *Evolution from molecules to man*, pages 403–425. Cambridge University Press.

De Bonet, J. S., Isbell, Jr., C. L., and Viola, P. (1997). MIMIC: Finding optima by estimating probability densities. In Mozer, M. C., Jordan, M. I., and Petsche, T., editors, *Advances in Neural Information Processing Systems*, volume 9, pages 424–430. The MIT Press.

De Jong, E. D. and Pollack, J. B. (2001). Utilizing bias to evolve recurrent neural networks. In Marko, K. and Werbos, P., editors, *Proceedings of the 2001 International Joint Conference on Neural Networks*, pages 2667–2672. IEEE Press.

De Jong, E. D. and Pollack, J. B. (2003). Multi-objective methods for tree size control. *Genetic Programming and Evolvable Machines*, 4(3):211–233.

De Jong, E. D., Watson, R. A., and Pollack, J. B. (2001). Reducing bloat and promoting diversity using multiobjective methods. In Spector, L. and Goodman, E. D., editors, *Proceedings of the 2001 Genetic and Evolutionary Computation Conference*, pages 11–18. Morgan Kaufmann.

De Jong, K. (1975). *An analysis of the behavior of a class of genetic adaptive systems*. Ph.D. thesis, The University of Michigan, Ann Arbor, MI, USA.

Deb, K. (2001). *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley & Sons, Chichester, UK.

Deb, K., Agrawal, S., Pratab, A., and Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Merelo, J. J., and Schwefel, H.-P., editors, *Proceedings of the Parallel Problem Solving from Nature VI Conference*, volume 1917 of *Lecture Notes in Computer Science*, pages 849–858. Springer Verlag.

Deb, K., Mohan, M., and Mishra, S. (2003). Towards a quick computation of well-spread Pareto-optimal solutions. In Fonseca, C., Fleming, P., Zitzler, E., Deb, K., and Thiele, L., editors, *Proceedings of the Second International Conference on Evolutionary Multi-Criterion Optimization*, pages 222–236. Springer Verlag.

Dellaert, F. and Beer, R. (1996). A developmental model for the evolution of complete autonomous agents. In Maes, P., Mataric, M., Meyer, J.-A., Pollack, J. B., and Wilson, S., editors, *From Animals to Animats: Proceedings of the*

Fourth International Conference on Simulation of Adaptive Behavior, pages 393–402. The MIT Press.

Dengiz, B., Altiparmak, F., and Smith, A. E. (1995). A genetic algorithm approach to optimal topological design of all terminal networks. In Dagli, C., Akay, M., Chen, C., Fernandez, B., and Ghosh, J., editors, *Proceedings of the Artificial Neural Networks in Engineering Conference (ANNIE'95)*, pages 405–411. ASME Press.

Di Ferdinando, A. D., Calabretta, R., and Parisi, D. (2001). Evolving modular architectures for neural networks. In French, R. M. and Sougné, J. P., editors, *Proceedings of the Sixth Neural Computation and Psychology Workshop*, pages 253–262. Springer Verlag.

Doi, Y. (1988). *Morphogenesis of Life Forms.* Saiensu-sha, Tokyo, Japan.

Dorigo, M., Caro, G. D., and Gambardella, L. M. (1999). Ant algorithms for discrete optimization. *Artificial Life*, 5(2):137–172.

Dorigo, M. and Gambardella, L. M. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66.

Dorigo, M., Maniezzo, V., and Colorni, A. (1991). Positive feedback as a search strategy. Technical report 91-016, Dipartimento Di Elettronic, Politecnico Di Milano, Milan, Italy.

Dorigo, M., Maniezzo, V., and Colorni, A. (1996). The ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics*, 26(1):29–41.

Downing, K. L. (2003). Developmental models for emergent computation. In Tyrell, A. M., Haddow, P., and Torresen, J., editors, *Proceedings of the 5th International Conference on Evolvable Systems*, volume 2606 of *Lecture Notes in Computer Science*, pages 105–116. Springer Verlag.

Doya, K. (1992). Bifurcations in the learning of recurrent neural networks. In *Proceedings of the 1992 International Symposium on Circuits and Systems*, pages 2777–2780. IEEE Press.

Droste, S., Jansen, T., and Wegener, I. (1998). Perhaps not a free lunch but at least a free appetizer. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakeila, M., and Smith, R. E., editors, *Proceedings of the 1998 Genetic and Evolutionary Computation Conference*, pages 833–839. Morgan Kaufmann.

Eberhart, R. C., Shi, Y., and Kennedy, J. (2001). *Swarm Intelligence.* Morgan Kaufmann, San Francisco, USA.

Eggenberger, P. (1997a). Creation of neural networks based on developmental and evolutionary principles. In Gerstner, W., Germond, A., Hasler, M., and Nicoud, J.-D., editors, *Proceedings of the 7th International Conference on Artificial Neural Networks*, volume 1327 of *Lecture Notes in Computer Science*, pages 337–342. Springer Verlag.

Eggenberger, P. (1997b). Evolving morphologies of simulated 3D organisms based on differential gene expression. In Husbands, P. and Harvey, I., editors, *Proceedings of the 4th European Conference on Artificial Life*, pages 205–213. The MIT Press.

Fahlman, S. and Lebière, C. (1990). The cascade-correlation learning architecture. In Touretzky, D. S., editor, *Advances in neural information processing systems*, volume 2, pages 524–532. Morgan Kaufmann.

Feder, J. (1971). Plex languages. *Information Science*, 3:225–241.

Federici, M. (2005). Evolving developing spiking neural networks. In Corne, D., Michalewicz, Z., McKay, B., Eiben, G., Fogel, D., Fonseca, C., Greenwood, G., Raidl, G., Tan, K. C., and Zalzala, A., editors, *Proceedings of the 2005 Congress on Evolutionary Computation*, pages 543–550. IEEE Press.

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

Fleischer, K. W. (1995). *A multiple-mechanism developmental model for defining self-organizing geometric structures*. Ph.D. thesis, Department of Computation and Neural Systems, California Institute of Technology, Pasadena, USA.

Fleischer, K. W. and Barr, A. H. (1994). A simulation testbed for the study of multicellular development: The multiple mechanisms of morphogenesis. In Langton, C. G., editor, *Artificial Life III: Proceedings of the Workshop on Artificial Life*, pages 389–416. Addison Wesley.

Flores, S. D., Cegla, B. B., and Cáceres, D. B. (2003). Telecommunication network design with parallel multi-objective evolutionary algorithms. In *Proceedings of the 2003 IFIP/ACM Latin America conference on Towards a Latin American agenda for network research*, pages 1–11. ACM Press.

Fogel, D. B., Fogel, L. J., and Porto, V. W. (1990). Evolving neural networks. *Biological Cybernetics*, 63:487–493.

Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966). *Artificial Intelligence through Simulated Evolution*. John Wiley & Sons, New York, USA.

Frean, M. (1990). The upstart algorithm: a method for constructing and training feedforward neural networks. *Neural Computation*, 2(2):198–209.

Furusawa, C. and Kaneko, K. (1998). Emergence of multicellular organisms with dynamic differentiation and spatial pattern. *Artificial Life*, 4(1):79–93.

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

Garzon, M. H. (1995). *Models of Massive Parallelism: Analysis of Cellular Automata and Neural Networks.* Springer Verlag, Berlin, Germany.

Gershenson, C. (2004). Introduction to Random Boolean Networks. In Bedau, M., Husbands, P., Hutton, T., Kumar, S., and Suzuki, H., editors, *Workshop and Tutorial Proceedings, Artificial Life IX: Ninth International Conference on the Simulation and Synthesis of Living Systems*, pages 160–173. The MIT Press.

Gibson, G. and Wagner, G. P. (2000). Canalization in evolutionary genetics: a stabilizing theory? *Bioessays*, 22(4):372–380.

Goldberg, D. E. (1989a). Genetic algorithms and Walsh functions. *Complex Systems*, 3:129–171.

Goldberg, D. E. (1989b). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, MA.

Goldberg, D. E., Deb, K., and Korb, B. (1989). Messy genetic algorithms: motivation, analysis, and first results. *Complex Systems*, 3:493–530.

Goldberg, D. E. and Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. In Grefenstette, J. J., editor, *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 41–49. Lawrence Erlbaum Associates.

Goles, E. and Martínez, S. (1990). *Neural and automata networks: dynamical behaviour and applications.* Mathematics and Its Applications. Kluwer Academic Publishers, Dordrecht, The Netherlands.

Gordon, T. G. W. and Bentley, P. J. (2005). Development brings scalability to hardware evolution. In Lohn, J., Gwaltney, D., Hornby, G., Zebulum, R., Keymeulen, D., and Stoica, A., editors, *Proceedings of the 7th NASA/DoD Conference on Evolvable Hardware*, pages 272–279. IEEE Press.

Gordon, T. G. W. and Bentley, P. J. (2006). Evolving hardware. In Zomaya, A., editor, *Handbook of Innovative Computational Paradigms*, pages 343–386. Springer Verlag.

Gould, S. J. and Vrba, E. S. (1982). Exaptation – a missing term in the science of form. *Paleobiology*, 8:4–15.

Greenwood, G. W. (1997). Training partially recurrent neural networks using evolutionary strategies. *IEEE Transactions on Speech and Audio Processing*, 5(2):192–194.

Gruau, F. (1992). Genetic synthesis of Boolean neural networks with a cell rewriting developmental process. In Schaffer, J. D. and Whitley, D., editors, *Proceedings of the International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 55–74. IEEE Press.

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

Gruau, F. (1995). Automatic definition of modular neural networks. *Adaptive Behaviour*, 3(2):151–183.

Gruau, F., Whitley, D., and Pyeatt, L. (1996). A comparison between cellular encoding and direct encoding for genetic neural networks. In Koza, J. R., Goldberg, D. E., Fogel, D. B., and Riolo, R. L., editors, *Proceedings of the First Annual Conference on Genetic Programming*, pages 81–89. The MIT Press.

Habel, A. (1992). *Hyperedge Replacement: Grammars and Languages*, volume 643 of *Lecture Notes in Computer Science.* Springer Verlag, Berlin, Germany.

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

Halder, G., Callaerts, P., and Gehring, W. (1995). Induction of ectopic eyes by targeted expression of the eyeless gene in drosophila. *Science*, 267:1788–1792.

Hancock, P. J. B. (1992). Genetic algorithms and permutation problems: a comparison of recombination operators for neural net structure specification. In Whitley, D. and Schaffer, J. D., editors, *Proceedings of the International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 108–122. IEEE Press.

Harik, G. R., Lobo, F. G., and Goldberg, D. E. (1999). The compact genetic algorithm. *IEEE Transactions on Evolutionary Computation*, 3(4):287–297.

Harp, S., Samad, T., and Guha, A. (1989). Towards the genetic synthesis of neural networks. In Schaffer, J. D., editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 360–369. Morgan Kaufmann.

Hartl, R. F. (1990). A global convergence proof for a class of genetic algorithms. Technical report, Institute of Econometrics, Operations Research and Systems Theory, Vienna University of Technology, Vienna, Austria.

Hebb, D. O. (1949). *The Organization of Behavior: A Neuropsychological Theory.* John Wiley & Sons, New York, USA.

Hinterding, R., Michalewicz, Z., and Eiben, A. E. (1997). Adaptation in evolutionary computation: a survey. In *Proceedings of the 4th IEEE International Conference on Evolutionary Computation*, pages 65–69. IEEE Press.

Hoai, N. X. and McKay, R. I. (2001). A framework for tree-adjunct grammar guided genetic programming. In Abbass, H. A. and Barlow, M., editors, *Proceedings of the Post-graduate ADFA Conference on Computer Science*, pages 93–99. ADFA.

Hogeweg, P. (2000). Evolving mechanisms of morphogenesis: on the interplay between differential adhesion and cell differentiation. *Journal of Theoretical Biology*, 203:317–333.

Holland, J. (1992). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* The MIT Press, Cambridge, USA, 2nd edition.

Holland, J. (1998). *Emergence: From Chaos to Order.* Oxford University Press, Oxford, UK.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences, USA*, 79:2554–2558.

Hornby, G. S. (2003). *Generative representations for evolutionary design automation.* Ph.D. thesis, Brandeis University Dept. of Computer Science.

Hornby, G. S. and Pollack, J. B. (2001a). The advantages of generative grammatical encodings for physical design. In Kim, J.-H., Zhang, B.-T., Fogel, G., and Kuscu, I., editors, *Proceedings of the 2001 Congress on Evolutionary Computation*, pages 600–607. IEEE Press.

Hornby, G. S. and Pollack, J. B. (2001b). Evolving L-systems to generate virtual creatures. *Computers and Graphics*, 25(6):1041–1048.

Iosifescu, M. (1980). *Finite Markov Processes and Their Applications.* John Wiley & Sons, Chichester, UK.

Jakobi, N. (1995). Harnessing morphogenesis. Cognitive science research paper 423, School of Cognitive and Computing Sciences, University of Sussex, Brighton, UK.

Kauffman, S. A. (1969). Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology*, 22:437–467.

Kauffman, S. A. (1993). *The Origins of Order.* Oxford University Press, New York, USA.

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

Kitano, H. (1995). A simple model of neurogenesis and cell differentiation based on evolutionary large-scale chaos. *Artificial Life*, 2:79–99.

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

Koza, J. R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs.* The MIT Press, Cambridge, USA.

Koza, J. R. (1995). Evolving the architecture of a multi-part program in genetic programming using architecture-altering operations. In McDonnell, J. R., Reynolds, R., and Fogel, D. B., editors, *Proceedings of the Fourth Annual Conference on Evolutionary Programming*, pages 695–717. The MIT Press.

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.

Kreowski, H.-J. (1993). Five facets of hyperedge replacement beyond context-freeness. In Ésik, Z., editor, *Proceedings of 9th International Conference on Fundamentals of Computation Theory*, volume 710 of *Lecture Notes in Computer Science*, pages 69–86. Springer Verlag.

Kubalík, J., Koutník, J., and Rothkrantz, L. J. M. (2003). Grammatical evolution with bidirectional representation. In Ryan, C., Soule, T., Keijzer, M., Tsang, E. P. K., Poli, R., and Costa, E., editors, *Proceedings of the 6th European Conference on Genetic Programming*, volume 2610 of *Lecture Notes in Computer Science*, pages 354–363. Springer Verlag.

Kumar, A., Pathak, R. M., Gupta, M. C., and Gupta, Y. P. (1993). Genetic algorithm based approach for designing computer network topology. In *Proceedings of the 21st ACM conference on Computer Science*, pages 358–365. ACM Press.

Lampinen, J. and Storn, R. (2004). Differential evolution. In Onwubolo, G. C. and Babu, B. V., editors, *New Optimization Techniques in Engineeering*, volume 141 of *Studies in Fuzziness and Soft Computing*, book chapter 6, pages 123–166. Springer Verlag.

Langdon, W. B. (2000a). Quadratic bloat in genetic programming. In Whitley, D., Goldberg, D. E., Cantú-Paz, E., Spector, L., Parmee, I., and Beyer, H.-G., editors, *Proceedings of the 2000 Genetic and Evolutionary Computation Conference*, pages 451–458. Morgan Kaufmann.

Langdon, W. B. (2000b). Size fair and homologous tree crossovers for tree genetic programming. *Genetic Programming and Evolvable Machines*, 1(1/2):95–119.

Langdon, W. B. and Poli, R. (1997). Fitness causes bloat. In Chawdhry, P. K., Roy, R., and Pan, R. K., editors, *Proceedings of the 2nd On-line World Conference on Soft Computing in Engineering Design and Manufacturing*, pages 13–22. Springer Verlag.

Langdon, W. B. and Poli, R. (2002). *Foundations of Genetic Programming.* Springer Verlag, London, UK.

Lewis, E. B. (1978). A gene complex controlling segmentation in drosophila. *Nature*, 276:565–570.

Li, M. and Vitányi, P. (1997). *An Introduction to Kolmogorov Complexity and Its Applications*. Springer Verlag, New York, USA, 2nd edition.

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

Lindenmayer, A. (1974). Adding continuous components to L-systems. In Rozenberg, G. and Salomaa, A., editors, *L Systems*, volume 15 of *Lecture Notes in Computer Science*, pages 53–68. Springer Verlag.

Lipson, H. and Pollack, J. B. (2000). Automatic design and manufacture of robotic lifeforms. *Nature*, 406:974–978.

Lones, M. A. and Tyrrell, A. M. (2004). Modelling biological evolvability: implicit context and variation filtering in enzyme genetic programming. *BioSystems*, 76(1–3):229–238.

Lucas, S. M. (1995). Towards the open ended evolution of neural networks. In Zalzala, A. M. S., editor, *Proceedings of the 1st International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, pages 388–393. Institution of Electrical Engineers.

Lucas, S. M. (2002). Evolving spring-mass models: a test-bed for graph encoding schemes. In Fogel, D., editor, *Proceedings of the 2002 Congress on Evolutionary Computation*, pages 1952–1957. IEEE Press.

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

Luerssen, M. H. (2005b). Phenotype diversity objectives for graph grammar evolution. In Abbass, H. A., Bossamaier, T., and Wiles, J., editors, *Recent Advances in Artificial Life*, volume 3 of *Advances in Natural Computation*, book chapter 12, pages 159–170. World Scientific.

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

Luerssen, M. H. and Powers, D. M. W. (2005). Graph composition in a graph grammar-based method for automata network evolution. In Corne, D., Michalewicz, Z., McKay, B., Eiben, G., Fogel, D., Fonseca, C., Greenwood, G., Raidl, G., Tan, K. C., and Zalzala, A., editors, *Proceedings of the 2005 Congress on Evolutionary Computation*, pages 1653–1660. IEEE Press.

Luke, S. and Panait, L. (2001). A survey and comparison of tree generation algorithms. In Spector, L. and Goodman, E. D., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 81–88. Morgan Kaufmann.

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

Martin, W. N., Lienig, J., and Cohoon, J. P. (1997). Island (migration) models: evolutionary algorithms based on punctuated equilibria. In Bäck, T., Fogel, D. B., and Michalewicz, Z., editors, *Handbook of Evolutionary Computation*, pages C6.3:1–16. Oxford University Press.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133.

McKay, R. I. (2000). Fitness sharing in genetic programming. In Whitley, D., Goldberg, D., Cantu-Paz, E., Spector, L., Parmee, I., and Beyer, H.-G., editors, *Proceedings of the 2000 Genetic and Evolutionary Computation Conference*, pages 435–442. Morgan Kaufmann.

McKay, R. I. and Abbass, H. A. (2001). Anticorrelation measures in genetic programming. In Kasabov, N. and Whigham, P., editors, *Australasia-Japan Joint Workshop on Intelligent and Evolutionary Systems*, pages 45–51. University of Otago Press.

McPhee, N. F. and Miller, J. D. (1995). Accurate replication in genetic programming. In Eshelman, L., editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 303–309. Morgan Kaufmann.

Michalewicz, Z. (1991). A step towards optimal topology of communications networks. In Libby, V., editor, *Data Structures and Target Classification*, volume 1470 of *Proceedings of the SPIE*, pages 112–122. SPIE.

Michalewicz, Z. (1993). A hierarchy of evolution programs: an experimental study. *Evolutionary Computation*, 1(1):51–76.

Milgram, S. (1967). The small world problem. *Psychology Today*, 1(1):60–67.

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

Miller, J. F. (2001). What bloat? Cartesian genetic programming on Boolean problems. In Goodman, E. D., editor, *Late Breaking Papers at the 2001 Genetic and Evolutionary Computation Conference*, pages 295–302. ISGEC Press.

Miller, J. F. (2003). Evolving developmental programs for adaptation, morphogenesis, and self-repair. In Banzhaf, W., Christaller, T., Dittrich, P., Kim,

J. T., and Ziegler, J., editors, *Proceedings of the 7th European Conference on Artificial Life*, volume 2801 of *Lecture Notes in Artificial Intelligence*, pages 256–265. Springer Verlag.

Miller, J. F. and Thomson, P. (2000). Cartesian genetic programming. In Poli, R., Banzhaf, W., Langdon, W. B., Miller, J., Nordin, P., and Fogarty, T. C., editors, *Proceedings of the Third European Conference on Genetic Programming*, pages 121–132. Springer Verlag.

Miller, J. F. and Thomson, P. (2003). A developmental method for growing graphs and circuits. In Tyrell, A. M., Haddow, P., and Torrensen, J., editors, *Proceedings of the 5th International Conference on Evolvable Systems: From Biology to Hardware*, volume 2606 of *Lecture Notes in Computer Science*, pages 93–104. Springer Verlag.

Minsky, M. L. and Papert, S. A. (1969). *Perceptrons.* The MIT Press, Cambridge, USA.

Mitchell, M., Crutchfield, J. P., and Das, R. (1997). Evolving cellular automata to perform computations. In Bäck, T., Fogel, D. B., and Michalewicz, Z., editors, *Handbook of evolutionary computation.* Oxford University Press.

Montana, D. and Davis, L. D. (1989). Training feedforward neural networks using genetic algorithms. In Sridharan, N. S., editor, *Proceedings of the 11th International Joint Conferences on Artificial Intelligence*, pages 762–767. Morgan Kaufmann.

Moriarty, D. E. and Miikkulainen, R. (1996). Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, 22(1-3):11–32.

Mountcastle, V. B. (1978). An organizing principle for cerebral function: the unit module and the distributed system. In Edelman, G. M. and Mountcastle, V. B., editors, *The mindful brain*, pages 17–49. The MIT Press.

Mozer, M. C. and Smolensky, P. (1989). Skeletonization: a technique for trimming the fat from a network via relevance assessment. In *Advances in neural information processing systems*, volume 1, pages 107–115. Morgan Kaufmann.

Mühlenbein, H. and Mahnig, T. (1999). FDA – a scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evolutionary Computation*, 7(4):353–376.

Mühlenbein, H. and Paaß, G. (1996). From recombination of genes to the estimation of distributions I, binary parameters. In Voigt, H.-M., Ebeling, W., Rechenberg, I., and Schwefel, H.-P., editors, *Proceedings of the Parallel Problem Solving from Nature IV Conference*, volume 1411 of *Lecture Notes in Computer Science*, pages 178–187. Springer Verlag.

Nolfi, S. and Floreano, D. (1999). Learning and evolution. *Autonomous Robots*, 7(1):89–113.

Nolfi, S. and Parisi, D. (1991). Growing neural networks. Technical report PCIA-91-15, Department of Cognitive Processes and Artificial Intelligence, Institute of Psychology, National Research Council, Rome, Italy.

Odell, G. M., Oster, G., Alberch, P., and Burnside, B. (1981). The mechanical basis of morphogenesis. *Developmental Biology*, 85:446–462.

Ofria, C., Adami, C., and Collier, T. C. (2003). Selective pressures on genomes in molecular evolution. *Journal of Theoretical Biology*, 222(4):477–483.

Ohno, S. (1970). *Evolution by Gene Duplication.* Springer Verlag, New York, USA.

O'Neill, M. (2005). mGGA: the meta-grammar genetic algorithm. In Keijzer, M., Tettamanzi, A., Collet, P., van Hemert, J. I., and Tomassini, M., editors, *Proceedings of the 8th European Conference on Genetic Programming*, volume 3447 of *Lecture Notes in Computer Science*, pages 311–320. Springer Verlag.

O'Neill, M. and Ryan, C. (2004). Grammatical evolution by grammatical evolution: the evolution of grammar and genetic code. In Keijzer, M., O'Reilly, U.-M., Lucas, S. M., Costa, E., and Soule, T., editors, *Proceedings of the 7th European Conference on Genetic Programming*, volume 3003 of *Lecture Notes in Computer Science*, pages 138–149. Springer Verlag.

Oosawa, C. and Savageau, M. A. (2002). Effects of alternative connectivity on behavior of randomly constructed Boolean networks. *Physica D: Nonlinear Phenomena*, 170(2):143–161.

O'Reilly, U.-M. (1997). Using a distance metric on genetic programs to understand genetic operators. In Tien, J. M., editor, *Proceedings of the 1997 International Conference on Systems, Man, and Cybernetics*, pages 4092–4097. IEEE Press.

Paterson, N. and Livesey, M. (1997). Evolving caching algorithms in C by genetic programming. In Koza, J. R., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M. H., Iba, H., and Riolo, R. L., editors, *Proceedings of the Second Annual Conference on Genetic Programming*, pages 262–267. Morgan Kaufmann.

Pavlidis, T. (1972). Linear and context-free graph grammars. *Journal of the ACM*, 19(1):11–23.

Pelikan, M., Goldberg, D. E., and Cantú-Paz, E. (1999). BOA: the Bayesian optimization algorithm. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., and Smith, R. E., editors, *Proceedings of the 1999 Genetic and Evolutionary Computation Conference*, pages 525–532. Morgan Kaufmann.

Pelikan, M. and Mühlenbein, H. (1999). The bivariate marginal distribution algorithm. In Roy, R., Furuhashi, T., and Chawdhry, P. K., editors, *Advances in Soft Computing – Engineering Design and Manufacturing*, pages 521–535, London. Springer Verlag.

Pettey, C. C. (1997). Diffusion (cellular) models. In Bäck, T., Fogel, D. B., and Michalewicz, Z., editors, *Handbook of Evolutionary Computation*, pages C6.4:1–16. Oxford University Press.

Pines, M. (2001). *The Genes We Share with Yeast, Flies, Worms, & Mice: New Clues to Human Health & Disease.* Howard Hughes Medical Institute, Chevy Chase, USA.

Plotkin, H. C. (1993). *Darwin machines and the nature of knowledge.* Harvard University Press, Cambridge, USA.

Price, K. (1999). An introduction to differential evolution. In Corne, D., Dorigo, M., and Glover, F., editors, *New ideas in optimization*, pages 79–108. McGraw-Hill.

Radding, C. M. (1982). Homologous pairing and strand exchange in genetic recombination. *Annual Review of Genetics*, 16:405–437.

Raff, R. A. (1996). *The Shape of Life.* The University of Chicago Press, Chicago, USA.

Ramos, V., Fernandes, C., and Rosa, A. C. (2005). On ants, bacteria and dynamic environments. In Abraham, A. and Dumitrescu, D., editors, *Proceedings of the Natural Computing and Applications Workshop.* IEEE Press.

Roberts, S. G. and Turega, M. (1995). Evolving neural network structures: an evaluation of encoding techniques. In Pearson, D., Steele, N., and Albrecht, R., editors, *Proceedings of the Second International Conference on Artificial Neural Networks and Genetic Algorithms*, pages 96–99. Springer Verlag.

Rosca, J. P. (1995). Entropy-driven adaptive representation. In Rosca, J. P., editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 23–32, Tahoe City.

Rosca, J. P. and Ballard, D. H. (1994). Learning by adapting representations in genetic programming. In *Proceedings of the 1994 World Congress on Computational Intelligence*, pages 407–412. IEEE Press.

Roska, T. and Chua, L. O. (1992). Cellular neural networks with non-linear and delay-type template elements and non-uniform grids. *International Journal of Circuit Theory and Applications*, 20(5):469–481.

Rothermich, J. A. and Miller, J. F. (2002). Studying the emergence of multicellularity with Cartesian genetic programming in artificial life. In Cantú-Paz, E., editor, *Late Breaking Papers at the 2002 Genetic and Evolutionary Computation Conference*, pages 397–403. AAAI Press.

Rozenberg, G., editor (1997). *Handbook on Graph Grammars and Computing by Graph Transformation: Volume I. Foundations.* World Scientific, River Edge, USA.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representation by back-propagating errors. *Nature*, 323:533–536.

Ryan, C., Collins, J. J., and O'Neill, M. (1998). Grammatical evolution: evolving programs for an arbitrary language. In Banzhaf, W., Poli, R., Schoenauer, M., and Fogarty, T. C., editors, *Proceedings of the First European Workshop on Genetic Programming*, volume 1391 of *Lecture Notes in Computer Science*, pages 83–95. Springer Verlag.

Sasaki, T. and Tokoro, M. (1997). Adaptation toward changing environments: why darwinian in nature? In Husbands, P. and Harvey, I., editors, *Proceedings of the Fourth European Conference on Artificial Life*, pages 378–387. The MIT Press.

Sastry, K., Pelikan, M., and Goldberg, D. E. (2005). Decomposable problems, niching, and scalability of multiobjective estimation of distribution algorithms. IlliGAL Report No. 2005004, Illinois Genetic Algorithms Laboratory, University of Illinois, Urbana-Champaign, USA.

Schwefel, H.-P. (1981). *Numerical Optimization of Computer Models*. John Wiley & Sons, Chichester, UK.

Shan, Y., McKay, R. I., Abbass, H. A., and Essam, D. (2003). Program evolution with explicit learning: a new framework for program automatic synthesis. In Sarker, R., Reynolds, R., Abbass, H. A., Tan, K. C., McKay, R. I., Essam, D., and Gedeon, T., editors, *Proceedings of the 2003 Congress on Evolutionary Computation*, pages 1639–1646. IEEE Press.

Shan, Y., McKay, R. I., Baxter, R., Abbass, H. A., Essam, D., and Nguyen, H. X. (2004). Grammar model-based program evolution. In *Proceedings of the 2004 Congress on Evolutionary Computation*, pages 478–485. IEEE Press.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423.

Silva, S. and Almeida, J. (2003). Dynamic maximum tree depth – a simple technique for avoiding bloat in tree-based gp. In Cantú-Paz, E., Foster, J. A., Deb, K., Davis, L. D., Roy, R., O'Reilly, U.-M., Beyer, H.-G., Standish, R. K., Kendall, G., Wilson, S., Harman, M., Wegener, J., Dasgupta, D., Potter, M. A., Schultz, A. C., Dowsland, K. A., Jonoska, N., and Miller, J. F., editors, *Proceedings of the 2003 Genetic and Evolutionary Computation Conference*, volume 2724 of *Lecture Notes in Computer Science*, pages 1776–1787. Springer Verlag.

Simon, H. A. (1996). *The Sciences of the Artificial*. The MIT Press, Cambridge, USA, 3rd edition.

Sims, K. (1994). Evolving 3d morphology and behaviour by competition. In Brooks, R. A. and Maes, P., editors, *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, pages 28–39. The MIT Press.

Sinclair, M. C. (1995). Minimum cost topology optimisation of the COST 239 European optical network. In Pearson, D., Steele, N., and Albrecht, R., editors, *Proceedings of the Second International Conference on Artificial Neural Networks and Genetic Algorithms*, pages 26–29. Springer Verlag.

Sinclair, M. C. (1997). NOMAD: applying a genetic-algorithm/heuristic hybrid approach to optical network topology design. In Smith, D. G., Steele, N. C., and Albrecht, R. F., editors, *Proceedings of the Third International Conference on Artificial Neural Networks and Genetic Algorithms*, pages 299–303. Springer Verlag.

Sipper, M. (1996). Co-evolving non-uniform cellular automata to perform computations. *Physica D*, 92:193–208.

Spector, L. and Robinson, A. (2002). Genetic programming and autoconstructive evolution with the push programming language. *Genetic Programming and Evolvable Machines*, 3:7–40.

Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127.

Stanley, K. O. and Miikkulainen, R. (2003). A taxonomy for artificial embryogeny. *Artificial Life*, 9(2):93–130.

Stork, D. G., Jackson, B., and Walter, S. (1992). Non-optimality via pre-adaptation in simple neural systems. In Langton, C. G., Taylor, C. J., Farmer, J. D., and Rasmussen, S., editors, *Artificial Life II: Proceedings of the Workshop on Artificial Life*, volume 10, pages 409–429. Addison-Wesley.

Tanaka, Y. and Berlage, O. (1996). Application of genetic algorithms to VOD network topology optimization. *IEICE Transactions on Communication*, E79-B(8):1046–1053.

Theraulaz, G. and Bonabeau, E. W. (1999). A brief history of stigmergy. *Artificial Life*, 5(2):97–116.

Todd, S. and Latham, W. (1992). *Evolutionary Art and Computers.* Academic Press, London, UK.

Toffolo, A. and Benini, E. (2003). Genetic diversity as an objective in multi-objective evolutionary algorithms. *Evolutionary Computation*, 11(2):151–168.

Togelius, J. and Lucas, S. M. (2005). Forcing neurocontrollers to exploit sensory symmetry through hard-wired modularity in the game of cellz. In Kendall, G. and Lucas, S., editors, *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pages 37–43. IEEE Press.

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

Toussaint, M. (2003b). The structure of evolutionary exploration: on crossover, buildings blocks and estimation-of-distribution algorithms. In Cantú-Paz, E., Foster, J. A., Deb, K., Davis, L., Roy, R., O'Reilly, U.-M., Beyer, H.-G., Standish, R. K., Kendall, G., Wilson, S. W., Harman, M., Wegener, J., Dasgupta, D., Potter, M. A., Schultz, A. C., Dowsland, K. A., Jonoska, N., and Miller, J. F., editors, *Proceedings of the 2003 Genetic and Evolutionary Computation Conference*, volume 2724 of *Lecture Notes in Computer Science*, pages 1444–1456. Springer Verlag.

Tsenov, A. (2005). Simulated annealing and genetic algorithm in telecommunications network planning. *International Journal of Intelligent Technology*, 1(1):28–33.

Turing, A. M. (1952). The chemical basis of morphogenesis. *Philosophical Transactions B*, 237:37–72.

Vassilev, V. K. and Miller, J. F. (2000). The advantages of landscape neutrality in digital circuit evolution. In Miller, J. F., Thompson, A., Thomson, P., and Fogarty, T. C., editors, *Proceedings of the Third International Conference on Evolvable Systems: From Biology to Hardware*, volume 1801 of *Lecture Notes in Computer Science*, pages 252–263. Springer Verlag.

Venter, J. C. et al. (2001). The sequence of the human genome. *Science*, 291(5507):1304–1351.

Vereijken, J. J. (1993). *Graph grammars and operations on graphs*. Master's thesis, Department of Computer Science, Leiden University, Leiden, The Netherlands.

Von Neumann, J. (1966). *Theory of Self-Reproducing Automata*. A. W. Burks (Ed.), University of Illinois Press, Urbana, USA.

Wagner, G. P. (1995). Adaptation and the modular design of organisms. In Moran, F., Moreno, A., Merelo, J. J., and Chacon, P., editors, *Proceedings of the Third European Conference on Artificial Life*, volume 929 of *Lecture Notes in Computer Science*, pages 317–328. Springer Verlag.

Wagner, G. P. and Altenberg, L. (1996). Complex adaptations and the evolution of evolvability. *Evolution*, 50(3):967–976.

Wagner, N., Michalewicz, Z., Khouja, M., and McGregor, R. R. (2004). Time series forecasting for dynamic environments: the DyFor genetic program model. In Hryniewicz, O., Kacprzyk, J., Koronacki, J., and Wierzchoń, S. T., editors, *Issues in Intelligent Systems – Paradigms*. Exit.

Walker, J. A. and Miller, J. F. (2004). Evolution and acquisition of modules in Cartesian genetic programming. In Keijzer, M., O'Reilly, U.-M., Lucas, S. M., Costa, E., and Soule, T., editors, *Proceedings of the 7th European Conference on Genetic Programming*, volume 3003 of *Lecture Notes in Computer Science*, pages 187–197. Springer Verlag.

Watson, R. A. and Pollack, J. B. (2005). Modular interdependency in complex dynamical systems. *Artificial Life*, 11(4):445–458.

Watts, D. J. (1999). *Small Worlds: The Dynamics of Networks between Order and Randomness*. Princeton University Press, Princeton, USA.

Watts, D. J. and Strogatz, S. H. (1998). Collective dynamics of 'small-world' networks. *Nature*, 393:440–442.

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

Whitley, D., Rana, S., and Heckendorn, R. (1999). The island model genetic algorithm: on separability, population size and convergence. *Journal of Computing and Information Technology*, 7(1):33–47.

Wieland, A. (1991). Evolving neural network controllers for unstable systems. In *Proceedings of the 1991 International Joint Conference on Neural Networks*, pages 667–673. IEEE Press.

Wilkins, A. S. (1992). *Genetic Analysis of Animal Development*. John Wiley & Sons, New York, USA, 2nd edition.

Wilson, E. O. (1971). *The Insect Societies*. Belknap Press, Cambridge, USA.

Wolfram, S. (1983). Cellular automata. *Los Alamos Science*, 9:2–21.

Wolfram, S. (2002). *A New Kind of Science*. Wolfram Media, Champaign, USA.

Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82.

Wolpert, L. (1969). Positional information and the spatial pattern of cellular differentiation. *Journal of Theoretical Biology*, 25(1):1–47.

Wright, S. (1967). Surfaces of selective value. *Proceedings of the National Academy of Sciences, USA*, 58:165–179.

Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE*, 87:1423–1447.

Yu, T. and Miller, J. F. (2001). Neutrality and the evolvability of Boolean function landscape. In Miller, J. F., Tomassini, M., Lanzi, P. L., Ryan, C., Tettamanzi, A., and Langdon, W. B., editors, *Proceedings of the 4th European Conference on Genetic Programming*, volume 2038 of *Lecture Notes in Computer Science*, pages 204–217. Springer Verlag.

Zambonelli, F. and Roli, A. (2001). On the emergence of macro spatial structures in dissipative cellular automata, and its implications for agent-based distributed computing. Technical report no. DISMI-14, University of Modena and Reggio Emilia, Modena and Reggio Emilia, Italy.