24

An Empirical Study of Graph Grammar Evolution

Martin Luerssen and David Powers
Flinders University
Australia

1. Introduction

Finding an optimal topology for a graph is relevant to many problem domains, as graphs can be used to model a variety of systems. Evolutionary algorithms (EAs) constitute a popular class of heuristic optimization algorithms, but have mainly been applied to what constitutes just a small subset of graphs, namely string and trees. Methods for evolving graphs typically involve the interpretation of a string or tree into a graph (e.g. Shirakawa et al., 2007). Accordingly, they rely on classical variation operators that are proven and easy to implement, but were fundamentally never designed for graphs and may struggle with their intrinsically greater complexity. Yet operating directly on graphs does not necessarily address this problem either. What is needed is a representation that facilitates the discovery and reuse of design dependencies within graphs. Graph grammars are the key to this, and their application to evolutionary graph building will be the focus of this chapter.

Grammars have mainly performed two distinct roles in the context of evolutionary comptuation: (1) as a means of establishing search bias, both declarative and preferential, which restrict and guide the search process, respectively; and (2) as a scalable representation that separates the complexity of the genotype from that of the phenotype. Both of these are eminently useful capabilities that are rarely found in conjunction. We therefore start by reviewing past research and trends in these fields and then describe the technique of Shared Grammar Evolution (SGE), which synergistically combines both roles into one coherent framework. SGE is subsequently applied to evolve a Cellular Graph Grammar, a graph representation tailored for evolutionary change. We experimentally explore the impact of diversity and spatial separation on evolutionary convergence, and propose a new evolutionary model inspired by swarm intelligence. Finally, the issue of graph bloat and the efficacy of the representational model are analysed so as to provide a practical insight into this unique scheme.

2. Grammar-Based evolution

For clarity, let us establish some introductory concepts first. In formal terms, a grammar G is a quadruple (N, T, P, A), 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 $A \in N$ is the axiom (or starting symbol). Each production rule is an ordered pair $p = (P, S)$, where predecessor $P \in (N \cup T)^*$ denotes a string of symbols to be replaced by the successor $S \in (N \cup T)^*$.

Source: Evolutionary Computation, Book edited by: Wellington Pinheiro dos Santos,
ISBN 978-953-307-008-7, pp. 572, October 2009, I-Tech, Vienna, Austria

Context-free grammars (CFGs) are a popular class of grammars constrained to $P \in N$, so that the predecessor can only be formed by a single nonterminal. A derivation involves applying a sequence of productions, starting from the axiom and typically generating a string.

2.1 Grammatical bias

Grammatical Evolution (GE) is a well-studied method for guiding evolution with a grammar (Ryan et al., 1998). The evolved genome in GE is a linear sequence of choices that are applied to a pre-defined CFG, which acts as a declarative bias, i.e. it restricts the search space to a limited set of predetermined, "sensible" possibilities. The GE grammar can itself be evolved using the meta-Grammar Genetic Algorithm (mGGA) (O'Neill, 2005). It thereby models a preferential bias, which dynamically adapts to the search process, i.e. produces better derivations with each generation. For this purpose, a pre-defined universal CFG is employed from which GE grammars are derived that then generate the actual solutions.

The use of a linear representation makes GE straightforward to implement, but a major drawback is the lack of guarantee that a sequence of choices will end in a terminal. The standard solution is to wrap the genome and repeat the choice sequence, but this can lead to never-ending derivations. CFG-GP is an alternative scheme based on Genetic Programming (GP) does not have this drawback as it derives trees rather than strings from a CFG (Whigham, 1995; Koza, 1992). CFG-GP is also capable of evolving its grammar by creating new productions from subtrees of the fittest solutions in the population. Grammatical GP (Augusto et al., 2008) is a recent, simplified variant of CFG-GP, where subtrees can be replaced only by other type-compatible subtrees. The grammar is not directly modified here, but subtree quantities will implicitly affect derivation probabilities.

Hoai et al. (2003) further formalize the tree-based approach by employing Lexicalized Tree-Adjunct Grammars (LTAGs). Each production in an LTAG consists of elementary trees, each of which must have at least one terminal node. The broad objective of this work is to extend the notion of probabilistic model building from string representations to trees (Mühlenbein & Paaá, 1996). Grammar Model-based Program Evolution (GMPE) correspondingly performs a hill-climbing search to learn a stochastic CFG from the best solutions in an existing population (Shan et al., 2004). A grammar that specifically describes only the fittest population members is established at each generation and then generalized by merging rules with the goal of minimizing the minimum description length of the grammar. A fraction of the next generation is then sampled using this grammar, and the procedure repeated, with novelty arising from the intermediate addition of random solutions.

2.2 Grammatical development

Grammars have also found popular use as models of developmental processes in biology and as such can provide further benefits to evolutionary search. A developmental process, or embryogeny, separates the representation of what is modified during evolution (the genotype) from the actual solution (the phenotype). If one merely applies a one-to-one mapping from genotype to phenotype, then the complexity of the former has to match the complexity of the latter. Large solutions therefore become difficult to optimize, even if they exhibit symmetry, which is common in many useful designs. Biological designs exploit symmetries by employing a highly indirect, developmental representation that has DNA transcribed into RNA, translated into polypeptides, and then processed into proteins which self-organize into phenotypic traits (Futuyma, 1998). Complex feedback loops within this

system produce iterative and recursive algorithms of development that are characterized by polygeny (multiple genes define a single phenotypic variable) and pleiotropy (changes to a single gene affect multiple phenotypic variables). Two desirable properties in evolutionary search are facilitated by this: neutrality and modularity. Neutrality is defined by genotypic variations that fail to affect the phenotype, which may lead to a build-up of hidden genetic variation that, once exposed, may produce a more rapid directional change than would otherwise be expected to occur. Neutral variations therefore allow distinct exploration strategies to be encoded in – and ultimately evolved with – the genotype (Toussaint, 2003). Modularity concerns the effective partition of sets into distinct subsets that are more tightly coupled internally than externally (Simon, 1996). The indirection of embryogeny enables the encoding of modules and of graph designs in terms of these modules, thus potentially reducing the configuration space that must be searched.

Embryogenic models that wish to be faithful to the biological archetype must establish detailed developmental mechanisms based on chemical, mechanical, and genetic regulatory factors. Yet the complexity of such a system implies not only a considerable computational cost, but also a general difficulty in analyzing it. In comparison, modelling embryology as a grammar can combine much of the power of a realistic model with the practicality of something simpler. A popular instance thereof is the L-system, which uses a grammar to rewrite all the symbols in a string in parallel and was originally introduced by Lindenmayer (1968) for replicating the growth characteristics of plants. Kitano (1990) evolved neural networks using a matrix L-system, where each production rewrites a node or edge symbol within a node or edge matrix into a 2×2 node or edge matrix. Boers and Sprinkhuizen-Kuyper (2001) used a string L-system to likewise evolve neural networks by interpreting a rewritten string as a graph. The grammar of GENRE (Hornby, 2003), an evolutionary design framework based on a parametric L-system, is evolved by a simple EA with specialized operators. Strings are rewritten and then translated into solutions, with successful applications to table designs, neural networks, and robot controllers.

In most instances of grammatical development the grammar generates a string that is interpreted as some solution construct. Data structures other than strings are less common; a notable exception is Cellular Encoding (CE) (Gruau, 1995). CE represents graph rewriting rules as a list of grammar trees, which can be evolved by GP. The nodes of the tree are references to graph operators applied successively to develop a single ancestor cell into a neural network or circuit design (Koza, 1999). Yet whether it is the choice of graph operators or the interpretation function for strings, a bias is imposed on the evolvable outcomes that is usually not well understood. It would therefore be desirable to operate as directly as possible on the graph itself – without necessarily abandoning the benefits of a grammar.

3. Graph operations

A directed graph is a quadruple (V, E, s, t) where V is a finite set of vertices, E is a finite set of edges, and $s, t: E \to V$ assign a source $s(e)$ and a target $t(e)$ to each $e \in E$. Natural and artificial instances of systems that can be represented as graphs are ubiquitous, and many problems of practical interest may be formulated as questions about graphs. Some graphs, such as the circuit of a microprocessor, need to be designed, and this is where EAs can assist. EAs traditionally operate on strings, with more recent methods such as GP operating on trees, a larger subset of graphs. For proper graph evolution we need a way to manipulate 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 a great many graph rewriting mechanisms have been devised; a comprehensive review is provided by Rozenberg (1997).

3.1 Hyperedge replacement

Hyperedge replacement constitutes one of the most elementary and frequently used concepts of graph rewriting (Habel, 1992). Edges in a graph normally have arity two, that is, they connect two vertices. A hyperedge may instead have multiple sources and targets, $s, t: E \to V^*$, connecting several vertices via a set of incoming tentacles and a set of outgoing tentacles. A graph with hyperedges is known as a hypergraph. Formally, a directed, 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, $s: E \to V^*$ assigns a sequence of sources $s(e)$ to each $e \in E$, $t: E \to V^*$ assigns a sequence of targets $t(e)$ to each $e \in E$, and $l: E \to C$ labels each hyperedge.

A multi-pointed hypergraph H is a hypergraph with additional *begin* and *end* nodes, which are also referred to as the external nodes of H. Let H_C be the set of all multi-pointed hypergraphs. A hypergraph production is an ordered pair $p = (A, R)$ with predecessor $A \in N$ and successor $R \in H_C$. 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 attachments 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.

3.2 Cellular graph grammars

Evolution of graphs implies directed change, which can be perceived as either a change to the graph, compliant with a grammar; or as a change to the grammar itself, as is common in grammatical development models. These two choices are not exclusive, as graph operations can be defined as graph replacements, i.e. grammatical operations, which can be evolved like any other graph. However, in graph grammar theory it is generally presumed that a replacement is well-typed, so that the hyperedge being replaced matches the external nodes of the multi-pointed hypergraph. The classic handover operation fuses the i-th source with the i-th begin node and the j-th target with the j-th end node, assuming these exist. In this context, not fusing any nodes beyond those that are present can lead to ripple effects on the topology of the final graph. Position independence resolves this problem and can be achieved by allowing the ordering of nodes to evolve (Goldberg et al., 1989). An identifying label $l \in C$ is assigned to each external and internal node, so that $l(v)$ is the label of node v. Additionally, we extend the mappings s and t so that the label l of the external node of the multi-pointed hypergraph is specified; the mappings hence become $s: E(l) \to V^*$ and $t: E(l) \to V^*$, respectively.

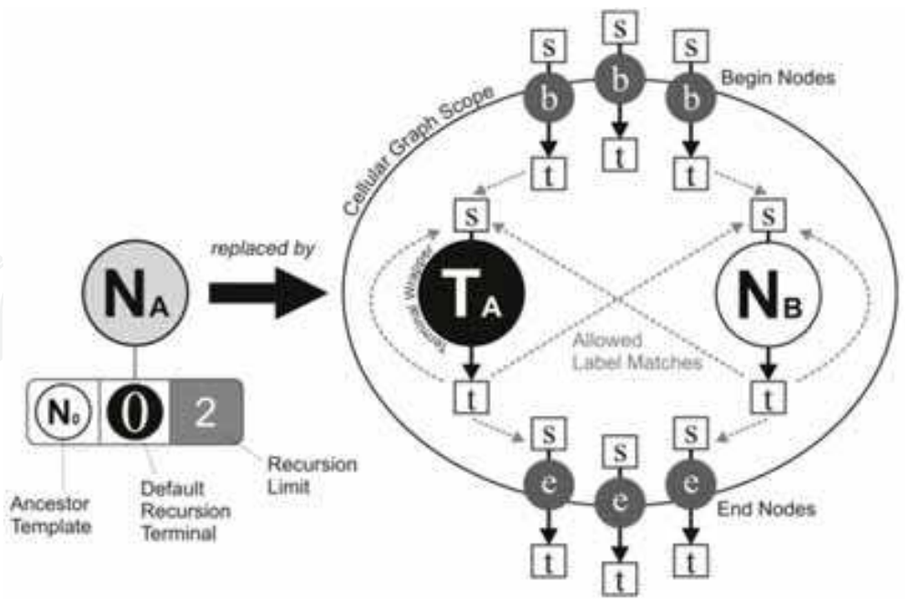

Fig. 1. A cellular production where nonterminal N_A is replaced by a cellular graph; T_A is a terminal, N_B is a nonterminal, b and e are begin and end nodes, and s and t are source labels and target labels.

A directed hypergraph can be described by an incidence structure, which contains a point for each vertex or hyperedge of the hypergraph and a line (i,j) if vertex i of the hypergraph is in hyperedge j. Storing these structures in an adjacency list has the drawback that adding or deleting a single structure would rarely be sufficient to substantially change the graph, as e.g. adding a hyperedge does not imply that it connects to anything. We address this by encapsulating those parts of a hyperedge or vertex that define how it attaches to other components into a descriptive unit referred to as a cellular graph, illustrated in Figure 1. A cellular production is a production with a cellular graph as its successor. It can be treated as a simplified hypergraph production in a hyperedge replacement system, except that all edges must be defined by cellular graphs, including those of the terminals. A graph is constructed from a grammar of cellular productions by replacing each nonterminal (or terminal wrapper) by the associated cellular graph, as shown in Figure 2. Fusion between begin and end nodes is established by finding target labels that match source labels.

We previously argued that a system that can be decomposed into modules may be more easily optimized. 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.

When matching labels, we thus restrict ourselves to a specific scope for each label type. 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. Labels are selected from a very large set (e.g.

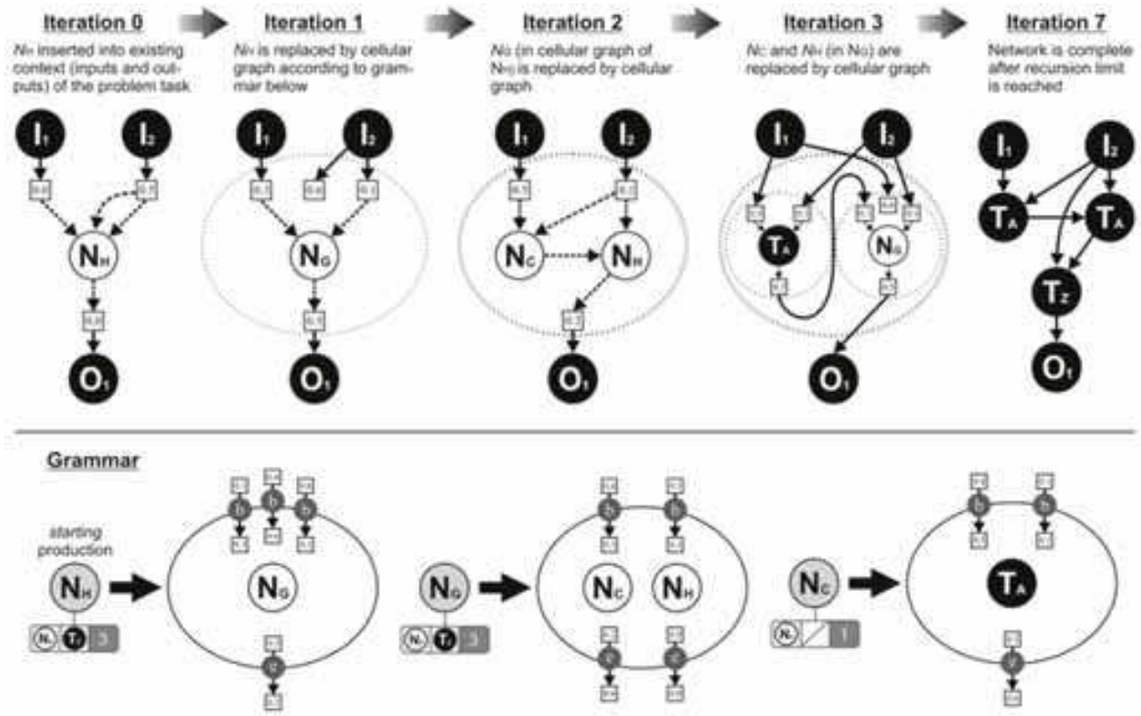

Fig. 2. A graph is derived from a cellular graph grammar over several iterations of replacement.

real numbers) and matched with the nearest, not necessarily identical, label – arithmetic difference is used here as the distance metric. Offset labels, which add to all the labels of associated cellular graphs, can also be applied to terminals and nonterminals. Labels may therefore change and combine in various ways without affecting the final solution. Subgraphs can be partially or fully disconnected from the host graph, allowing building blocks to neutrally accumulate and later be activated through possibly minor label changes.

4. Shared Grammar Evolution

Shared Grammar Evolution (SGE) provides a framework for evolving grammars such as the above (Luerssen & Powers, 2008). Each solution graph is described by its own individual grammar, referred to as an *i*-grammar, which is composed of a set of productions from which this graph (and only this graph) can be derived. Productions within an *i*-grammar may refer to each other, leading to recursion and a high degree of pleiotropy, as individual productions can trigger many other productions. This is comparable to the L-system evolution discussed previously, but instead of representing each solution by a separate set of productions, SGE combines these sets into a single set, the *p*-grammar. Productions with identical successors can be eliminated from the *p*-grammar, as only one instance of a particular production has to exist, even if it is involved in the derivation of different programs. Depending on the reuse of productions, the total number of productions in the population may thus be reduced, as shown in Figure 3.

Since a grammar with alternatives cannot uniquely represent – that is, describe deterministically – a specific solution, no productions with identical predecessors are

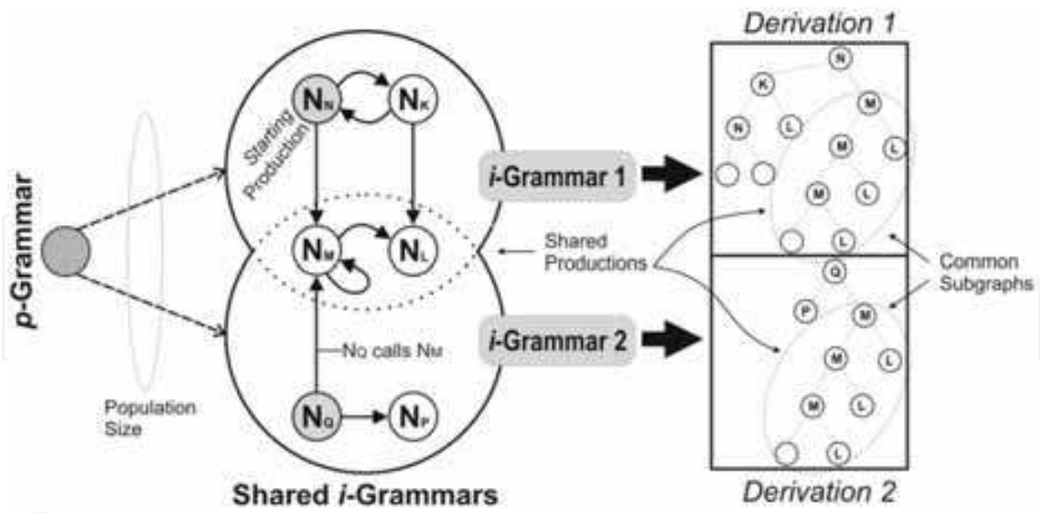

Fig. 3. Each graph is derived from an associated grammar, but grammars may share productions.

permitted in *i*-grammars. *i*-grammars are initially generated from a user-defined set of template productions, which define the cellular graphs and terminals that are permissible. Template productions can have alternatives, since their role is not as a representation, but as a declarative bias. They delimit the range of graphs that can evolve, as every new production is a variation of a template production. In case *a priori* knowledge about an optimal static template grammar is poor, cellular graphs can also be mutated directly during evolution. For this, the components of a cellular graph are organized into a list of nonterminal symbols (hyperedges of the graph), a list of terminal symbols (terminal nodes of the graph), and a list of *(source, target, direction)* label triples (begin and end nodes of the graph). Three operators may be applied to each list:

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

A probability is assigned to each *(operation, list)* pair, so that all probabilities sum to 1. A mutation involves randomly selecting a pair from these probabilities. If an inserted label is obtained from the template grammar, it is changed to point to a new production instance (and its associated subgraph) generated from the template grammar. The above operators are supplemented by the *increase recursion* and *decrease recursion* operators, which increase or decrease the recursion limit of the cellular production by one. During derivation, a production is redirected to an associated default terminal if it calls itself, directly or indirectly, more often than specified by this limit.

SGE can be viewed as a repeated growing and pruning of the *p*-grammar; an illustration of this is given in Figure 4. For every graph derived from its associated starting production, a single expressed production is spontaneously replaced by a mutated variant. After testing all the mutated graphs, the least fit solutions, both from the mutated set and the previously evaluated graphs, are discarded by eliminating all associated productions that are not involved in any fitter solutions. Conversely, if a mutation survived, the *p*-grammar is modified so that the mutated graph becomes one of the graphs derivable from it.

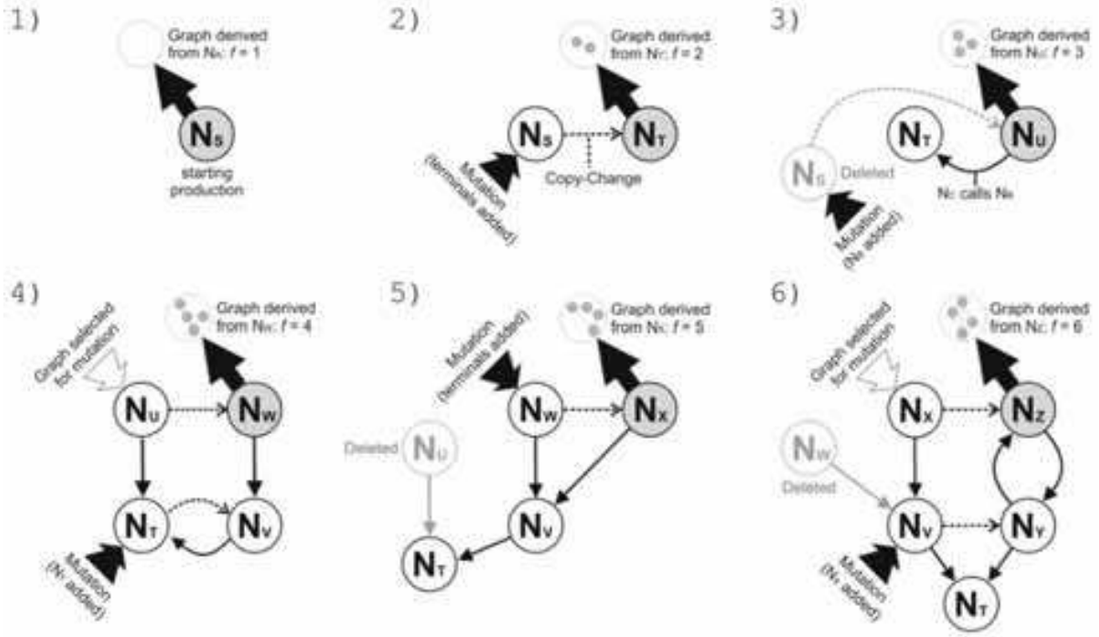

Fig. 4. Illustration of SGE with a maximum population of two graphs. Starting with an empty production/ graph N_S in generation (1), terminals are added to a copy N_T of this production in (2), then N_T is added to itself, producing N_U in (3), while the graph of N_S has least fitness f and is thus removed. N_T in the graph of N_U is then mutated in (4), producing N_V and a copy of N_U, N_W, with a reference to N_V. The graph of N_T is now uncompetitive, but remains as a production used by N_U. Further offspring is created in (5) and (6), leading to N_Z, which exhibits a recursive self-reference.

5. Promoting diversity

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. Since genetic lineages often reduce to one lineage early in the evolutionary process (McPhee & Hopper, 1999), maintaining diversity in a population is necessary for the long-term success of any evolutionary system, as it allows the population to continue searching for productive regions of the search space and thus avoid becoming trapped by local optima. Several methods have been proposed to improve diversity and combat premature convergence in EAs; we will investigate two of these in the context of graph grammar evolution: phenotypic diversity objectives and spatial separation.

5.1 Phenotypic diversity objectives

The principal drawback of any genotypic diversity measure is its limited applicability to a grammar, as the extensive neutrality intrinsic to this representation would allow it to improve diversity while remaining isomorphic. A possible solution is to employ a phenotypic diversity objective instead (Luerssen, 2005). 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. The mean distance of solution i is the absolute difference between ranks,

$$D_i = \frac{\sum_{j=0}^{N} |R_i - R_j|}{N} \tag{5.1}$$

where N is the number of other solutions. A measure less biased towards rewarding poor performance is to compare whether two solutions i and j show non-identical performance,

$$S_{ij} = \begin{cases} 1 \; if \; R_i \neq R_j \\ 0 \; otherwise \end{cases} \tag{5.2}$$

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

$$D_i = \frac{\sum_{j=0}^{N} S_{ij}}{N} \tag{5.3}$$

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

$$H(i) = -\log(1 - \frac{\sum_{j=0}^{N} S_{ij}}{N}) \tag{5.4}$$

but since the diversity objective will also be ranked for selection purposes, we can use the simpler difference measure while obtaining the same effect.

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,

$$D_i = \frac{\sum_{c=0}^{C} \sum_{j=0}^{N} |R_{ci} - R_{cj}|}{C \times N} \tag{5.5}$$

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

$$S_{ij} = \begin{cases} 1 \; if \; \sum_{c=0}^{C} |R_{ci} - R_{cj}| > 0 \\ 0 \; otherwise \end{cases} \tag{5.6}$$

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

$$D_i = \frac{\sum_{j=0}^{N} S_{ij}}{C \times N} \tag{5.7}$$

5.1.1 Handling multiple objectives

Having both a performance and a diversity objective implies that there is not one optimal solution, but a set of compromises known as a Pareto-optimal set. Multiobjective evolutionary algorithms (MOEAs) are typically based on Pareto-domination, where 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 on at least one objective (Deb, 2001). If the population size is smaller than the size of the Pareto-optimal set, then the MOEA is meant to return solutions spread evenly along the Pareto boundary. Most MOEAs apply some form of phenotypic niching to achieve this: 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 the multi-objective implementation of SGE, 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 MOEA for SGE matches the NSGA-II presented by Deb et al. (2000).

	Binomial-3 Regression	**Random Bit Sequence (RBS)**	**Pole Balancing**	**Computer Network Topology (CNT)**
Objective	Infer the mapping $y = f(x)$, where $f(x)$ is the binomial-3 polynomial $(x+1)^3$	Reproduce a binary time sequence	Optimize topology and weights of a neural network balancing 2 poles fixed to a cart moving on a finite track	Create a virtual computer network that efficiently connects data sinks with sources
Terminals	0/ 1/ 2-ary: +, -, ×, % (protected division)	0/ 1/ 2-ary: AND, XOR	Neurons with transfer function: $\varphi(x) = \frac{1}{1+e^{-4.9x}}$	Virtual sinks, sources, and switches
Fitness Case(s)	21 equidistant points generated by the objective function over $x = [-1, 1]$	16-bit sequence given by a 4-bit de Bruijn Counter (with seed 0000)	Pole balancing setup and simulation, see Stanley & Miikkulainen (2002)	10 randomly pregenerated unit/ data stream configurations
Simulation	Graph relaxed for 10 cycles	Simulation for 32 simulation cycles (+4 cycles lead-in), sampled every 2 cycles; to allow many different designs to be synchronized with the sampling rate, line delays are assigned to edges with a geometric probability of 0.5 of longer delays	Relaxed for 3 cycles; weights are assigned to edges by randomization with a standard Gaussian distribution (μ = 0, σ = 1), at 0.3 prob., or by DE (Price, 1999), at 0.7 prob., with parameter F = 0.2 and a crossover prob. of 0.9	Simplified network simulation, see Luerssen (2009)
Error Measure	Mean squared error	Proportion of incorrectly reproduced bits	Reciprical number of cycles both poles remain balanced	Fraction of data requests not satisfied
Mutation	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 to the same production			
Population	20 graphs, each defined by a maximum of 1000 productions and 1000 terminals per production			
Generations	1000			

Table 1. Description and default parameters for all experimental problem tasks.

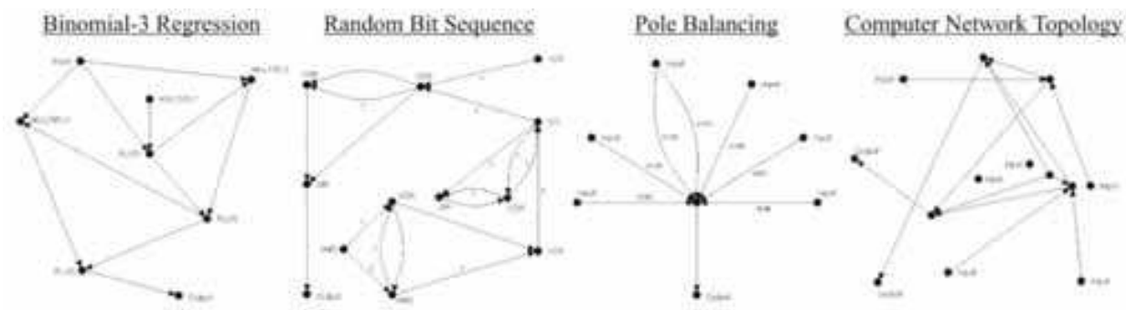

Fig. 5. A sample of graphs evolved by SGE on the four problem tasks. Connection weights are shown for the pole balancing ANN and line delays for the sequence circuit.

5.1.2 Experiment

To evaluate the effect of diversity on graph grammar evolution, we selected four problem tasks, which are described in Table 1. They are quick to evaluate, yet also challenging, and encompass different natural requirements of the representation. Sample solutions for each problem are shown in Figure 5. Solution candidates are evolved with a MOEA applied to three objectives: the function error, the solution size (see also section 8), and one of four diversity measures: entropy, fitness case entropy, distance, and fitness case distance. Results are averaged over 100 runs. Statistical significance is determined using a Z-test or non-parametric Wilcoxon rank sum test on the best solutions of the final generation of each run.

5.1.3 Results

The performance outcomes of using the different diversity measures are listed in tables 2 and 3 later in the chapter. 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. Without a diversity measure, the mean error is 0.054, which appears a lot worse, yet the difference is only borderline significant ($p < 0.03$). 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 $p < 0.003$).

All diversity measures improve performance on the RBS evolution, and this improvement is significant except with 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. Only a single fitness case is used for pole balancing, so the fitness case-based diversity measure is inapplicable. MSEs are generally low, but not all solutions manage to balance the pole for the entire cycle sequence. The entropy measure leads to an improvement, but this is not significant, whereas the distance measure (with only 71% success rate) is significantly worse 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.

Overall, 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. However, solutions arising from these measures are frequently much larger than otherwise, with a correspondingly negative impact on evaluation time (not shown).

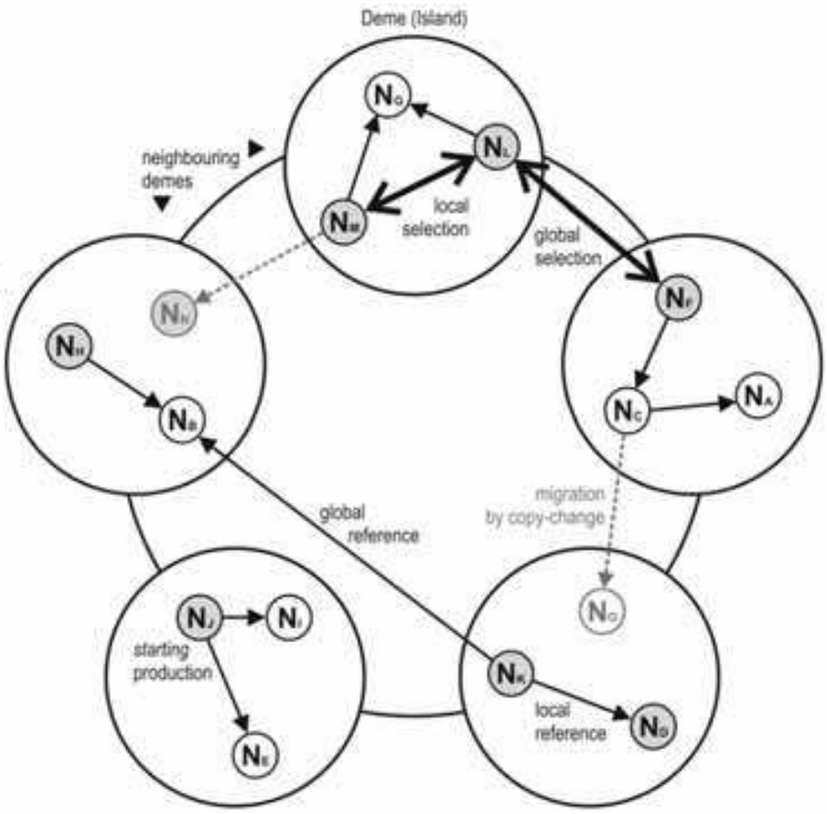

Fig. 6. The island model as applied to graph grammar evolution: islands are arranged in a ring, with offspring being allowed to move to neighbouring islands.

5.2 Island models

Separating individuals spatially may promote their diversity by allowing them to evolve more independently of other parts of the population. The most popular model of this, the island model, coarsely divides the population into several smaller subpopulations, called demes (Martin et al., 1997). An EA evolves each deme independently, but, periodically, information is exchanged by migrating individuals from one deme to another. 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. Production sharing in SGE complicates this migration of an individual – and all its associated productions – to a different deme, so the island model will be applied to the selection process alone: solutions compete only against those on the same deme, but productions can refer to any others globally. There is 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. 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 explore 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 along these choices. The various interactions of productions across islands are visualised in Figure 6.

5.2.1 Experiment

Our model is based on a simple 2-neighbour cellular space forming a ring, where transition is possible from any island to any of its two neighbours. 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 in this case is fixed at 5, and there is no limit to how many solutions or productions may exist in a deme. The population starts with a single empty starting production in the first deme. The choice of mutations will be expanded so that productions 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. These models are tested against running 5 populations (of 4 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. Note that multiobjetive niching will be applied globally across all islands, so that being on a different island only affects domination, not niching.

5.2.2 Results

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 ($p < 0.02$). Choosing productions locally from an island rather than from a global repository appears to have no significant impact on the tested 5 island configuration. Since solutions 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 leads to worse performance than with the standard island model. Migration between islands is clearly essential for gaining performance benefits from the island model.

6. Adaptive search

Ant Colony Optimization (ACO) is perhaps the best-known implementation of swarm intelligence (Dorigo et al., 1999). It is inspired by the ability of ants to establish shortest route paths between their colony and food sources. In ACO, a set of simple computational agents – artificial 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. Ants lay down a pheromone trail that indicates how beneficial a move was, which affects the probability distribution of future moves.

EAs differ from ACO in that the former represents the knowledge about the problem as a population of solutions, whereas the latter maintains a memory of past performance in the form of pheromone trails. For graph grammar evolution, such a memory may provide useful guidance in exploring the grammar. Productions can only survive if they are useful in some existing solution – thus, unlike any random construct, such productions also have a higher probability of being useful in a new solution. Naturally, this probability diminishes if there are niches for many different solutions in the population, because we might randomly pick a production and use it in a context for which it was not evolved. Reinterpreted for ACO, each production is a partial solution, and the addition of a production constitutes a move. Unlike ACO, SGE has no explicit probabilities assigned to each move. Consequently, production choice is highly random and depends solely on the composition of the grammar. Having an adaptable probability distribution as with ACO would provide superior guidance, but partial solutions in SGE are exceedingly short-lived: productions are added and removed with every generation. The path that one ant builds can rarely be followed by another. SGE thus does not appear easy to adapt to swarm intelligence, but a more limited model is possible and will be presented next.

6.1 Graph grammar swarms

At least two choices must be made when generating offspring from a graph. First, we must choose one of the productions expressed during derivation of this graph. Instead of randomly choosing from a uniform distribution, as in the existing framework, the following heuristic inspired by PBIL (Baluja & Caruana, 1995) 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

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

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. The presented mechanism should globally reduce the mutation of productions that rarely lead to good offspring (e.g. productions fully optimized for their context) and focus on other productions that do.

Some of the graph grammar mutations involve novel 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 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 second proposal therefore 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. 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 replace a production, once chosen for mutation, by one of its descendants – and then apply additional variations. The descendant

is chosen according to the above system of preferred mutation targets, as illustrated in Figure 7. 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.

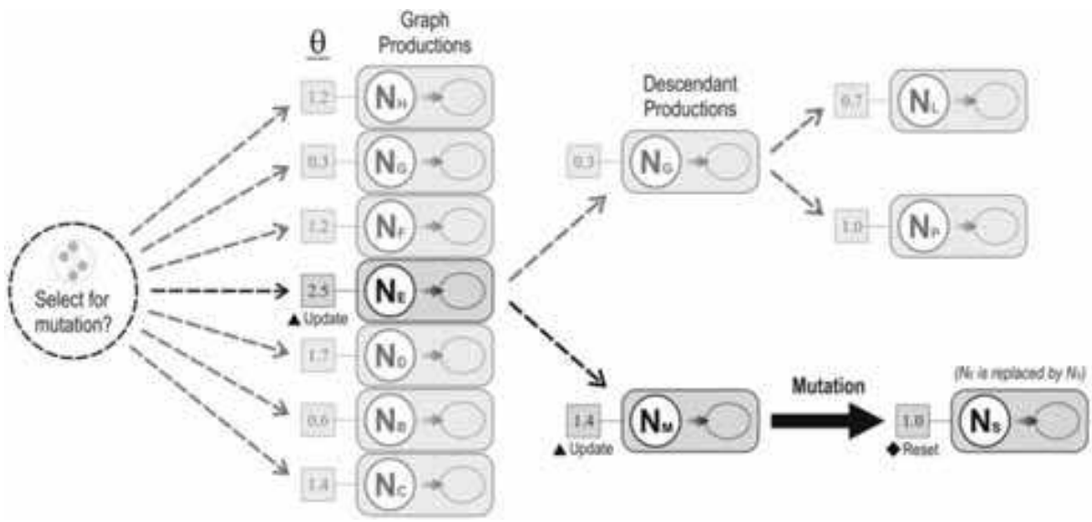

Fig. 7. Adaptive production search: The probability of a production being chosen for mutation is dependent on $\boldsymbol{\theta}$; it is then replaced by a descendant also determined by $\boldsymbol{\theta}$ before mutation is applied.

6.1.1 Experiment

We apply the above 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 $\varphi = 0.9$. 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 φ should allow for some cases where no descendant is chosen, but also have a large value so as to increase the experimental effect here.

6.1.2 Results

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. 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. 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. Graph 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 the only known successful mutation transitions, the results suggest that a preference for offspring over parent is indeed not very beneficial to overall performance.

Problem		Parameters	Success Rate	MCE ×1000	Min MSE		Size	
					Min	Mean	Mean	Min Err.
Binomial-3 Regression		Default	71%	55	0.000	0.054±0.106	18±7	25±7
	Diversity	Entropy	*82%*	46	0.000	*0.016*±0.039	21±16	30±28
		Case Entropy	*79%*	45	0.000	0.029±0.074	19±5	30±11
		Distance	*57%*	104	0.000	0.064±0.100	41±79	29±11
		Case Distance	64%	84	0.000	0.060±0.107	72±110	39±64
	Island Model	2 Islands	76%	62	0.000	0.035±0.086	15±3	25±8
		5 Islands	*84%*	51	0.000	*0.015*±0.043	20±6	39±21
		20 Islands	71%	77	0.000	0.030±0.103	30±9	46±22
		5 Isl. (Local)	78%	63	0.000	0.020±0.047	20±5	35±15
		5 Isl. (Isolated)	66%	83	0.000	0.051±0.130	17±4	28±10
	Adaptive Search	Target	64%	77	0.000	0.079±0.126	18±5	24±7
		Lineage	79%	50	0.000	0.035±0.086	20±5	28±8
		Target+Lineage	74%	53	0.000	0.043±0.094	19±5	26±9
	Size Objective	Size Shared	*21%*	360	0.000	*0.451*±0.644	9±6	51±54
		No Primary Size	*31%*	399	0.000	*0.189*±0.246	47±37	47±37
		C.Entr.+No Size	*86%*	20	0.000	*0.009*±0.033	378±850	360±900
Random Bit Sequence Circuit		Default	1%	36390	0.000	0.134±0.057	13±5	14±7
	Diversity	Entropy	*4%*	7874	0.000	*0.109*±0.057	13±4	14±6
		Case Entropy	*21%*	1442	0.000	*0.065*±0.044	16±6	17±13
		Distance	*2%*	12522	0.000	0.119±0.051	375±230	32±36
		Case Distance	*24%*	1321	0.000	*0.063*±0.045	342±240	27±17
	Island Model	2 Islands	*4%*	6970	0.000	0.124±0.058	15±9	17±11
		5 Islands	*6%*	4687	0.000	0.116±0.058	16±9	17±11
		20 Islands	*10%*	2944	0.000	*0.102*±0.060	20±8	21±13
		5 Isl. (Local)	*6%*	5046	0.000	*0.109*±0.059	16±7	16±8
		5 Isl. (Isolated)	*8%*	4395	0.000	0.119±0.065	14±8	16±20
	Adaptive Search	Target	*3%*	8044	0.000	0.123±0.053	14±7	14±8
		Lineage	*4%*	8416	0.000	0.113±0.055	14±7	15±8
		Target+Lineage	*3%*	9685	0.000	0.134±0.063	12±5	13±5
	Size Objective	Size Shared	*0%*	N/ A	0.063	*0.156*±0.061	18±9	27±38
		No Primary Size	*18%*	1376	0.000	*0.084*±0.061	23±19	23±19
		C.Entr.+No Size	*96%*	79	0.000	*0.003*±0.012	89±73	79±43

Table 2. Performance statistics for experiments, averaged over 100 runs. MCE denotes minimum computational effort for a success probability of 99% (see Koza, 1992).

Problem		Parameters	Success Rate	MCE ×1000	Min MSE		Size	
					Min	Mean	Mean	Min Err.
Pole Balancing		Default	82%	40	0.001	0.002±0.002	8±1	9±3
	Diversity	Entropy	*90%*	32	0.001	0.001±0.001	7±1	8±2
		Distance	*71%*	71	0.001	0.002±0.002	6±1	9±2
	Island Model	2 Islands	*96%*	26	0.001	*0.001*±0.001	7±1	9±2
		5 Islands	*98%*	20	0.001	*0.001*±0.001	8±2	10±4
		20 Islands	*95%*	23	0.001	*0.001*±0.001	12±9	13±10
		5 Isl. (Local)	*100%*	20	0.001	*0.001*±0.000	8±1	10±3
		5 Isl. (Isolated)	*96%*	32	0.001	*0.001*±0.000	8±2	9±2
	Adaptive Search	Target	77%	48	0.001	0.002±0.002	8±1	8±2
		Lineage	86%	34	0.001	0.002±0.002	7±0	7±2
		Target+Lineage	84%	37	0.001	0.002±0.002	8±1	8±2
	Size Objective	Size Shared	87%	46	0.001	0.002±0.002	8±1	8±2
		No Primary Size	*42%*	76	0.001	*0.003*±0.003	35±42	36±43
		C.Entr.+No Size	*51%*	47	0.001	*0.003*±0.003	91±119	31±36
ComputerNetwork Topology Design		Default	23%	1366	0.000	0.180±0.167	5±2	7±7
	Diversity	Entropy	32%	932	0.000	*0.127*±0.133	6±4	8±8
		Case Entropy	39%	1051	0.000	*0.112*±0.127	6±2	8±4
		Distance	20%	1463	0.000	0.141±0.120	6±4	9±13
		Case Distance	25%	1293	0.000	*0.134*±0.133	6±3	8±6
	Island Model	2 Islands	21%	1570	0.000	0.169±0.149	6±3	8±8
		5 Islands	*35%*	722	0.000	*0.126*±0.143	6±4	9±14
		20 Islands	28%	935	0.000	*0.102*±0.108	6±5	10±14
		5 Isl. (Local)	30%	1041	0.000	0.144±0.161	6±5	8±9
		5 Isl. (Isolated)	27%	1081	0.000	0.131±0.128	6±3	8±7
	Adaptive Search	Target	*26%*	1281	0.000	0.145±0.176	5±3	7±5
		Lineage	21%	1506	0.000	0.172±0.152	5±3	7±6
		Target+Lineage	24%	1378	0.000	0.161±0.154	6±2	7±3
	Size Objective	Size Shared	*1%*	30882	0.000	*0.400*±0.176	0±1	6±3
		No Primary Size	*37%*	450	0.000	*0.089*±0.108	13±24	12±17
		C.Entr.+No Size	*54%*	202	0.000	*0.047*±0.068	120±133	42±63

Table 3. (Continued from Table 2) Standard deviations are listed after each ±. *Italic* success rates are signifantly different from default (Z-test, $p < 0.05$); *italic* MSEs are significantly different from default (Wilcoxon rank sum test, $p < 0.05$)

7. Complexity growth

Solutions may grow during evolution more so than necessary, a phenomenon referred to as bloat (Langdon & Poli, 1997). Targeting solution size as an evolutionary objective is an effective means of addressing this and also leaves the user with a Pareto-optimal set of solutions balancing performance and size. This has been our default setup so far. In most instances this is desirable; in others only the best performing solution is acceptable and searching the entire Pareto boundary would then seem a disproportionate effort. A viable alternative is provided 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.

Size is defined here as the sum of the terminals, nonterminals, and external nodes (i.e. label triples) of each cellular production expressed during the derivation of the graph. This is not the same as a node + edge count of the graph, but any less inclusive measure would permit bloat. With any form of size constraint, however, some degree of redundancy will be needed to balance exploitation, i.e., greedy search, against exploration, i.e., diversity in the population. Hidden variation can accumulate within redundant code and, once revealed, fuel the kind of rapid adaptation needed to escape from any suboptima that have trapped the search (Hansen, 2006). We suggest to accommodate this by sharing the size of a production among all the solutions in the population that make use of this production (excluding recurrency, which is still scored cumulatively). A production contributing to many graphs therefore becomes in effect cheaper, which facilitates its reuse.

7.1 Experiment

Experiments are performed using our standard set of problem tasks and parameters, with four variations being tested: 1) size is targeted as an evolutionary objective (the experimental default); 2) size is shared among that solutions that utilize it; 3) size is consulted only on equally performing solutions (denoted *No Size* for brevity); and 4) fitness case entropy is included as a primary objective when size is secondary (see reasons below).

7.2 Results

Allowing production size to be shared among solutions triggers a significant decline in performance across all tasks where reuse would be expected to matter (i.e. all except pole balancing, which is structurally trivial). 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 solutions minimise their effective size by maximising references to each other's productions. Part of the population is thus optimized on the size objective at the expense of actual task performance.

Significant performance differences are also observed when making size a secondary objective, with the Binomial-3 regression and the pole balancing performing worse ($p < 5 \times 10^{-8}$), yet the RBS and the CNT design performing better ($p < 3 \times 10^{-8}$). These contradictory results 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 than the RBS and CNT design when using a secondary size objective.

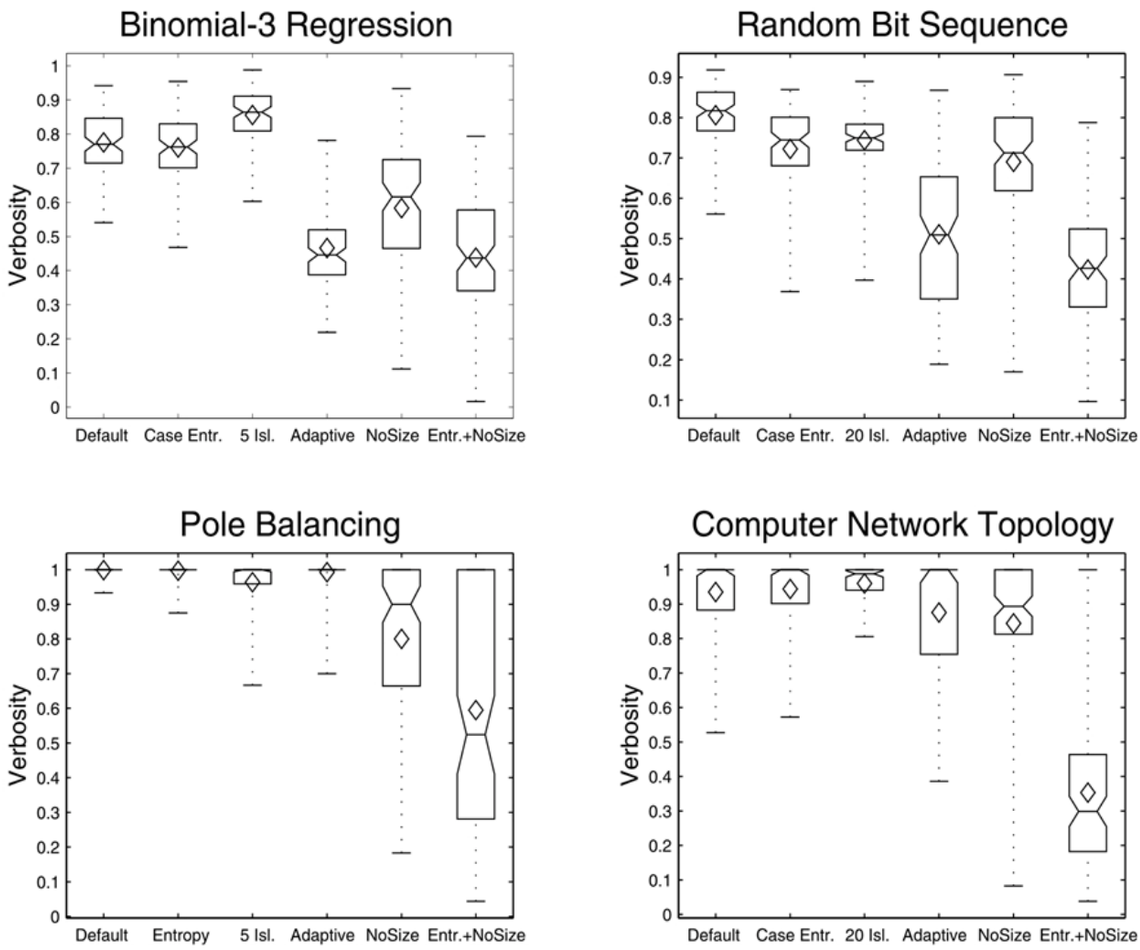

Fig. 8. Verbosity box plots for final generation that show median, quartiles, and outliers. Verbosity is defined as the ratio of the total production number expressed when deriving all graphs over the total production number in the population. If verbosity is 1, each production would belong to a single graph and there is no recursion, whereas low verbosity indicates high reuse.

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.

Results indeed corroborate this hypothesis. Performance on the Binomial-3 regression, RBS, 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, where the success rate rises to 96% from just 1% for the default. 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 7× on the RBS – and more than 20× 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.

7.3 Reuse

SGE is unique in sharing productions between multiple solutions. A measure of this reuse is shown in Figure 8. Problems previously identified as making more use of multiple productions – the Binomial-3 regression and the RBS – exhibit a moderate degree of reuse, while pole balancing 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 noted earlier that the use of adaptive search (target + lineage in this experiment) has little impact on performance, but this figure reveals that it encourages reuse on the problems where reuse should matter, resulting in much smaller grammars for the same graph size. 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 grammar representation would be most evident with problems that require larger solutions than the ones evaluated here.

8. Generational trends

Figures 9 to 12 depict the changes of various population and solution statistics over 1000 generations for the major different configurations. The individual 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); 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; and the size of the best performing solution.

Graph 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 the small size of the optimal solution actually penalises any relaxation of the size constraint. Without either a primary size or a diversity objective the growth of solution size is much more contained, but solutions also exhibit inferior performance. Reductions in size during evolution seem to be uncommon, particularly when applying the primary size objective, which suggests that either the optimization 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 total production numbers are dependent on solution size and the extent of reuse. 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. However, it is not clear why the replacement ratio for the island

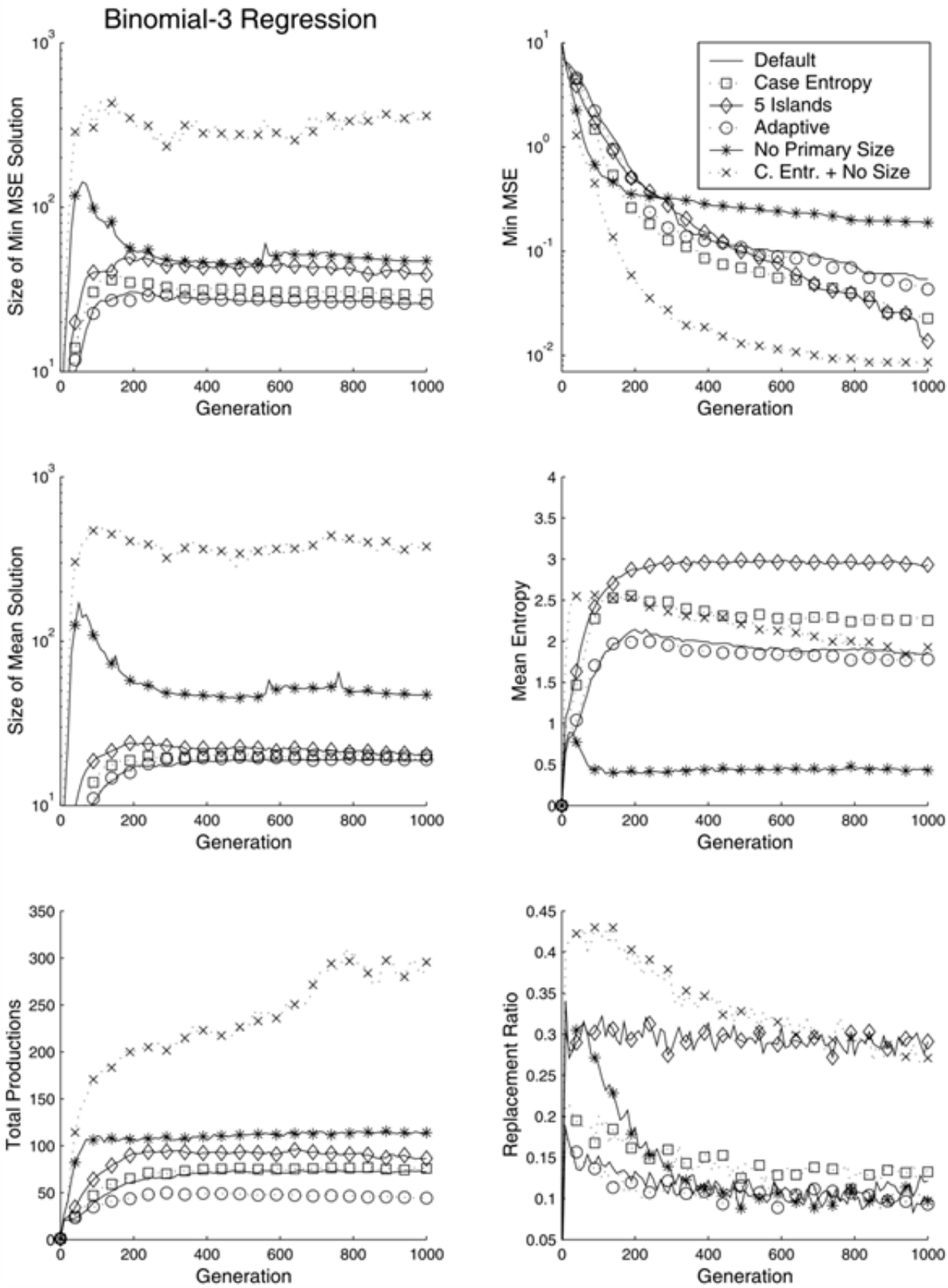

Fig. 9. Generational development for the Binomial-3 regression problem. (MSE and size are shown on a logarithmic Y-axis to improve readability.)

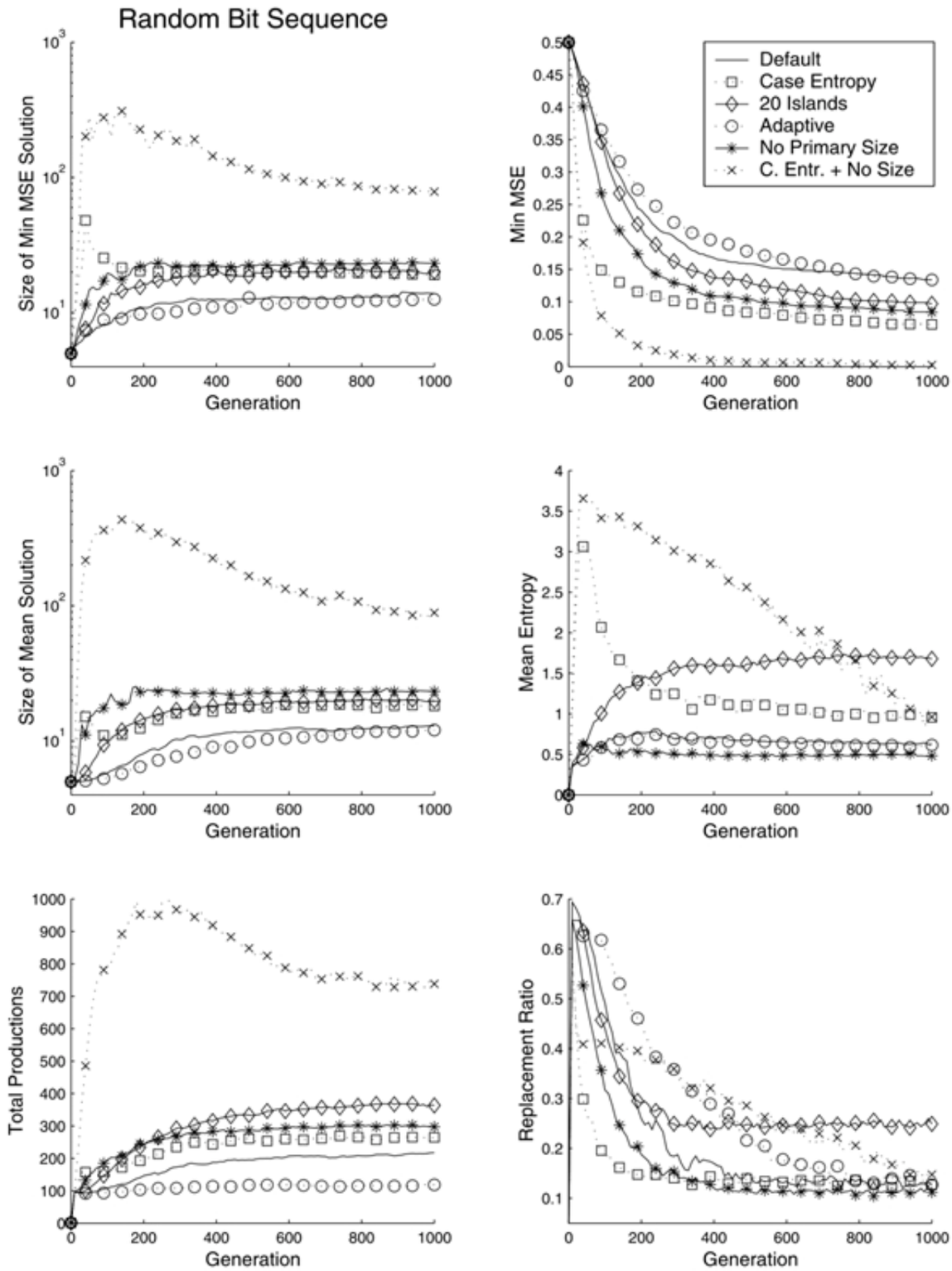

Fig. 10. Generational development for the RBS circuit problem. (Size is shown on a logarithmic Y-axis to improve readability.)

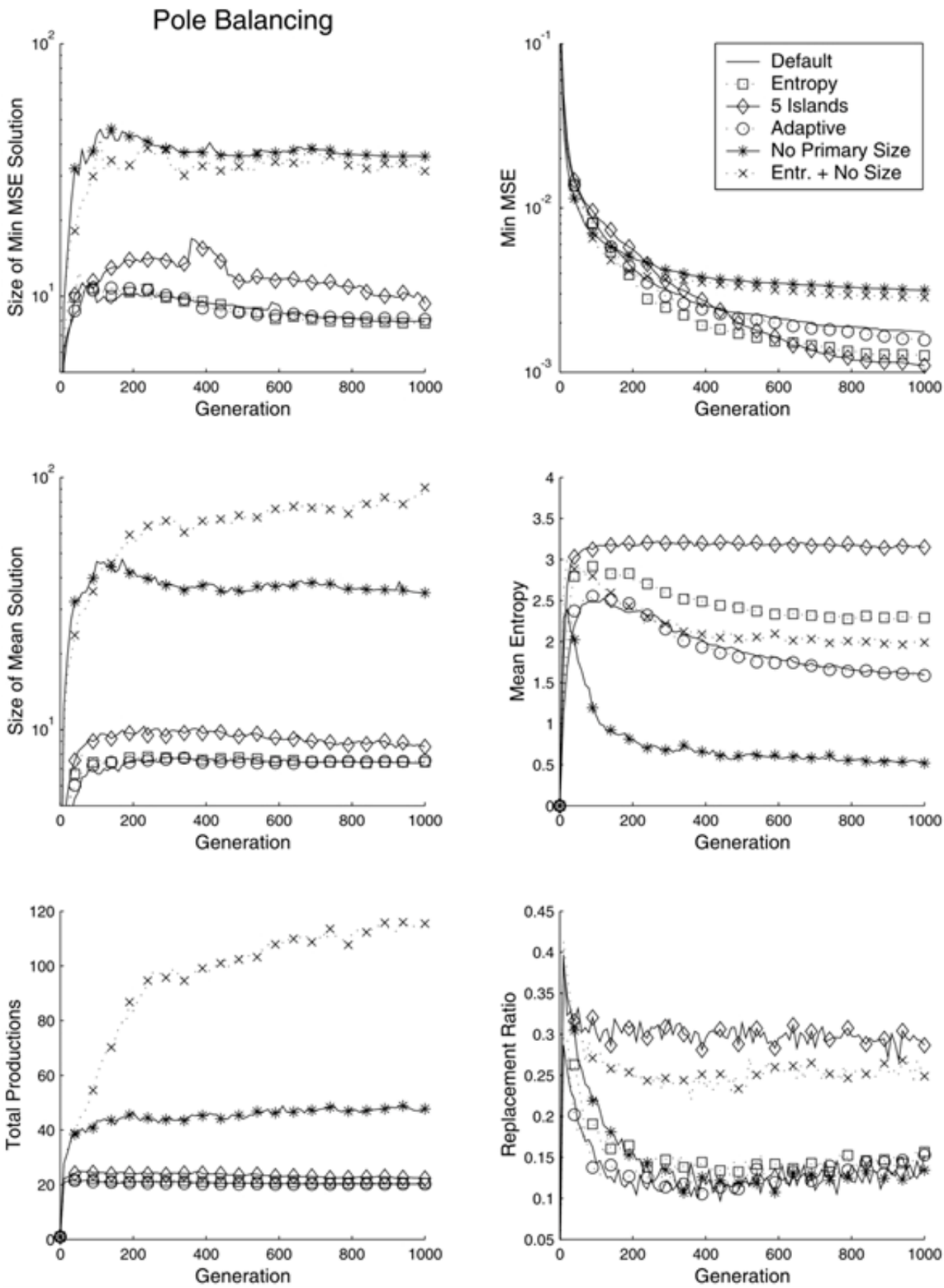

Fig. 11. Generational development for the pole balancing problem. (MSE and size are shown on a logarithmic Y-axis to improve readability.)

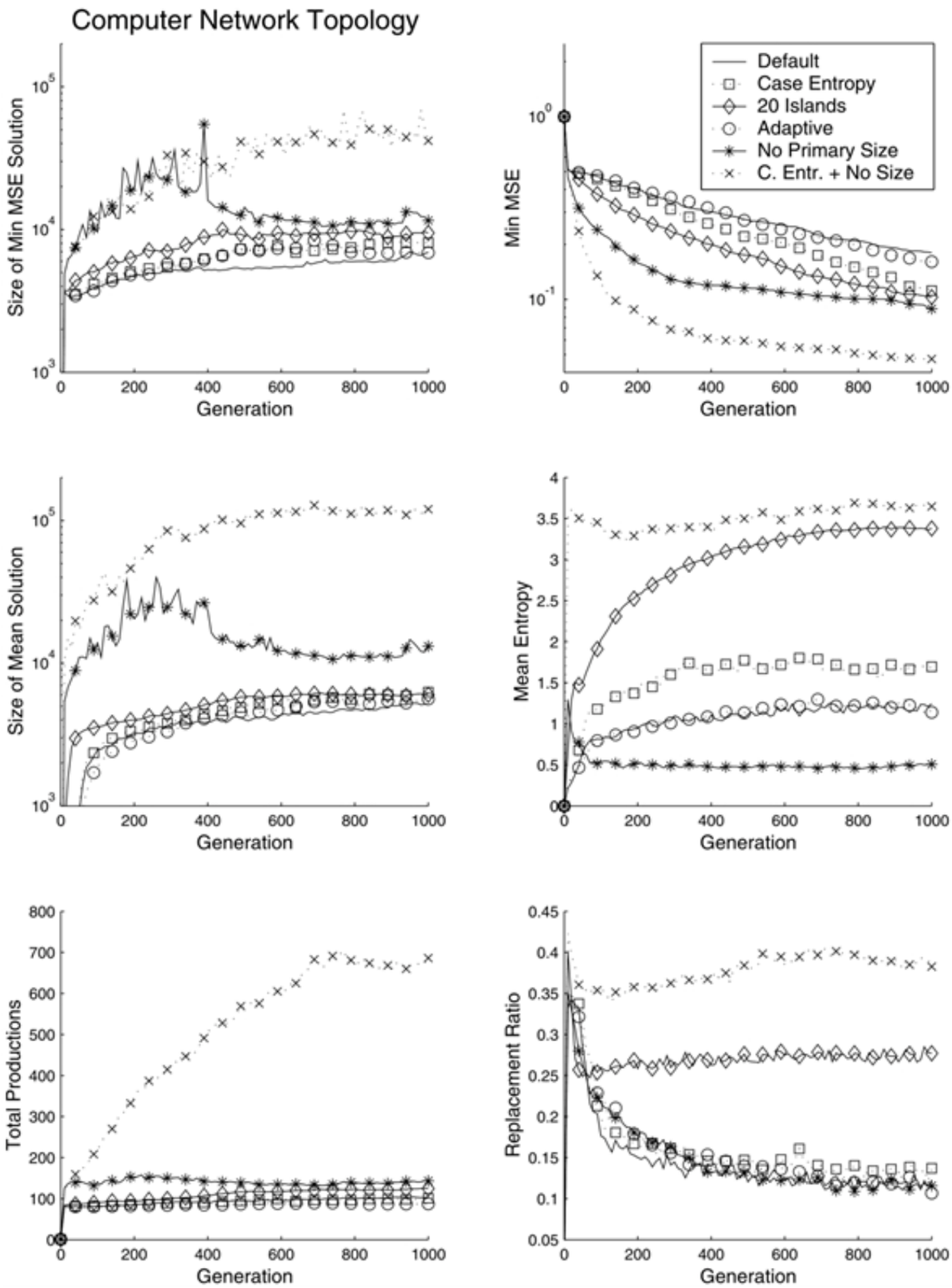

Fig. 12. Generational development for the CNT design problem. (MSE and size are shown on a logarithmic Y-axis to improve readability.)

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.

9. Conclusion

The system presented in this chapter is a significant step towards achieving a simple, formal, comprehensive basis for graph evolution. Its main significance arises from simplifying hypergraph grammars for the purpose of evolutionary optimization, which avoids many of the complexity pitfalls of "biologically realistic" models. Yet unlike other simpler models, the graph transformations are not predefined and fixed here, but fully evolvable, allowing for an automatic optimization of the graph design bias and thus a greater degree of domain independence. It assumes, however, that we have a method for evolving such grammar. Shared grammar evolution unites several aspects of grammatical bias and developmental systems into an effective method that can evolve anything derivable from a CFG, including graphs. The success of this approach is governed by a number of factors, and through application to a diverse set of design problems, we have gained some perspective on these. Firstly, significant performance improvements can be obtained when emphasizing diversity in the grammar population. This can be accomplished most effectively by adding an entropy measure of phenotypic diversity as an evolutionary objective. 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 have also presented a multi-objective island model that exhibits performance benefits comparable to the entropy method. We further propose the application of concepts from swarm intelligence to accelerate convergence, but associated experiments fail to produce significant performance improvement, although they reveal significant increases in production reuse that lead to a more compact grammar. In relation to this, we ascertained that the search process is severely constrained by co-optimization 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 yet been found. Future performance improvements should arise from a better understanding of how the grammar establishes a preferential bias. We need to develop a more intelligent selection scheme that makes exploratory mutations into distant search regions viable, which, in

combination with ACO-like exploitation, could ultimately improve the convergence characteristics of graph grammar evolution.

10. References

Augusto, D. A.; Barbosa, H. J. C. & Ebecken, N. F. F. (2008). Coevolution of data samples and classifiers integrated with grammatically-based genetic programming for data classification. *Proceedings of the 10th annual conference on Genetic and evolutionary computation (GECCO'08)*, Atlanta, USA, pp. 1171-1178, ACM Press.

Baluja, S. & Caruana, R. (1995). Removing the genetics from the standard genetic algorithm. *Proceedings of the Twelfth International Conference on Machine Learning, ML-95*, pp. 38-46, Morgan Kaufmann.

Boers, E. & Sprinkhuizen-Kuyper, I. (2001). Combined biological metaphors. *Advances in the evolutionary synthesis of intelligent agents*, pp. 153-183, MIT Press.

Deb, K.; Agrawal, S.; Pratab, A. & Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. *Proceedings of the Parallel Problem Solving from Nature VI Conference*, LNCS 1917, pp. 849-858, Springer-Verlag.

Deb, K. (2001). *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley.

Dorigo, M.; Di Caro, G. & Gambardella, L. (1999). Ant algorithms for discrete optimization. *Artificial Life*, vol. 5, no. 2, pp. 137-172.

Futuyma, D. (1998). *Evolutionary biology* (3rd ed.), Sinauer Associates, Inc.

Gruau, F. (1995). Automatic definition of modular neural networks. *Adaptive Behaviour*, vol. 3, no. 2, pp. 151-183.

Goldberg, D.; Deb, K. & Korb, B. (1989). Messy genetic algorithms: motivation, analysis, and first results. *Complex Systems*, vol. 3, pp. 493-530.

Habel, A. (1992). *Hyperedge Replacement: Grammars and Languages*. LNCS 643, Springer-Verlag.

Hansen, T. F. (2006). The evolution of genetic architecture. *Annual Review of Ecology Evolution and Systematics*, vol. 37, no. 1, pp. 123-157.

Hoai, N.; McKay, R. & Abbass, H. (2003). Tree adjoining grammars, language bias, and genetic programming. *Proceedings of the 6th European Conference on Genetic Programming (EuroGP 2003)*, LNCS 2610, pp. 335-344, Springer-Verlag.

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

Kitano, H. (1990). Designing neural networks using genetic algorithms with graph generation systems. *Complex Systems*, vol. 4, no. 4, pp. 461-476.

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

Koza, J.; Bennett III, F.; Andre, D. & Keane, M. (1999). *Genetic Programming III: Darwinian invention and problem solving*, Morgan Kaufmann.

Langdon, W. & Poli, R. (1997). Fitness causes bloat. *Second On-line World Conference on Soft Computing in Engineering Design and Manufacturing*, pp. 13-22, Springer-Verlag.

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

Luerssen, M. (2005). Phenotype diversity objectives for graph grammar evolution. *Recent Advances in Artificial Life*, vol. 3, pp. 159-170, World Scientific.

Luerssen, M. & Powers, D. (2008). Evolving encapsulated programs as shared grammars. *Genetic Programming and Evolvable Machines*, vol. 9, no. 3, pp. 203-228.

Luerssen, M. (2009). *Experimental Investigations into Graph Grammar Evolution*: *A novel approach to evolutionary design*. VDM Verlag Dr. Müller.

Martin, W.; Lienig, J. & Cohoon, J. (1997). Island (migration) models: evolutionary algorithms based on punctuated equilibria. *Handbook of Evolutionary Computation*, pp. 1-16, Oxford University Press.

McPhee, N. & Miller, J. (1995). Accurate replication in genetic programming. *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA'95)*, pp. 303-309, Morgan Kaufmann.

McPhee, N. & Hopper, N. (1999). Analysis of genetic diversity through population history. *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1112-1120, Morgan Kaufmann.

Miller, J. (2001). What bloat? Cartesian Genetic Programming on Boolean problems. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'01) Late Breaking Papers*, pp. 295-302, ISGEC Press.

Mühlenbein, H. & Paaá, G. (1996). From recombination of genes to the estimation of distributions I, binary parameters. *Parallel Problem Solving from Nature (PPSN IV)*, LNCS 1411, pp. 178-187, Springer-Verlag.

O'Neill, M. (2005). mGGA: The meta-Grammar Genetic Algorithm. *Proceedings of the 8th European Conference on Genetic Programming*, Lausanne, Switzerland, LNCS 3447, pp. 311-320, Springer-Verlag.

Price, K. (1999). An introduction to differential evolution. *New Ideas in Optimization*, pp. 79–108, McGraw-Hill.

Rozenberg, G. (1997). *Handbook of Graph Grammars and Computing by Graph Transformation: volume I. Foundations*. World Scientific.

Ryan, C.; Collins, J. & O'Neill, M. (1998). Grammatical evolution: evolving programs for an arbitrary language. *Proceedings of the First European Workshop on Genetic Programming (EuroGP'98)*, pp. 83-95, Springer-Verlag.

Shan, Y.; McKay, R.; Baxter, R.; Abbass, H.; Essam, D. & Nguyen, H. (2004). Grammar model-based program rvolution. *Proceedings of the 2004 IEEE Congress on Evolutionary Computation (CEC 2004)*, Portland, Oregon, vol. 1, pp. 478-485.

Shirakawa, S.; Ogino, S. & Nagao, T. (2007). Graph structured program evolution. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'07)*, pp. 1686-1693, ACM Press.

Simon, H. (1996). *The Sciences of the Artificial* (3rd ed.), MIT Press.

Stanley, K. O. & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, vol. 10, no. 2, pp. 99-127.

Toussaint, M. (2003). On the evolution of phenotypic exploration distributions. *Foundations of Genetic Algorithms 7 (FOGA VII)*, pp. 169-182, Morgan Kaufmann.

Whigham, P. (1995). Rosca, J. (ed.) Grammatically-based genetic programming. *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications,* Tahoe City, USA, pp. 33-41, Morgan Kaufmann Publishers.

Evolutionary Computation

Edited by Wellington Pinheiro dos Santos

ISBN 978-953-307-008-7
Hard cover, 572 pages
Publisher InTech
Published online 01, October, 2009
Published in print edition October, 2009

This book presents several recent advances on Evolutionary Computation, specially evolution-based optimization methods and hybrid algorithms for several applications, from optimization and learning to pattern recognition and bioinformatics. This book also presents new algorithms based on several analogies and metafores, where one of them is based on philosophy, specifically on the philosophy of praxis and dialectics. In this book it is also presented interesting applications on bioinformatics, specially the use of particle swarms to discover gene expression patterns in DNA microarrays. Therefore, this book features representative work on the field of evolutionary computation and applied sciences. The intended audience is graduate, undergraduate, researchers, and anyone who wishes to become familiar with the latest research work on this field.

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen