A Statistical Study of a Class of Cellular Evolutionary Algorithms

Mathieu Capcarrère
Logic Systems Laboratory
Swiss Federal Institute of Technology
1015 Lausanne, Switzerland
Mathieu.Capcarrere@epfl.ch

Marco Tomassini
Institute of Computer Science
University of Lausanne
1015 Lausanne, Switzerland
Marco.Tomassini@iismail.unil.ch

Andrea Tettamanzi
Department of Computer Science
University of Milan
Via Bramante 65, 26013 Crema (CR), Italy
tettaman@dsi.unimi.it

Moshe Sipper
Logic Systems Laboratory
Swiss Federal Institute of Technology
1015 Lausanne, Switzerland
Moshe.Sipper@epfl.ch

Abstract

Parallel evolutionary algorithms, over the past few years, have proven empirically worthwhile, but there seems to be a lack of understanding of their workings. In this paper we concentrate on cellular (fine-grained) models, our objectives being: (1) to introduce a suite of statistical measures, both at the genotypic and phenotypic levels, which are useful for analyzing the workings of cellular evolutionary algorithms; and (2) to demonstrate the application and utility of these measures on a specific example—the cellular programming evolutionary algorithm. The latter is used to evolve solutions to three distinct (hard) problems in the cellular-automata domain: density, synchronization, and random number generation. Applying our statistical measures, we are able to identify a number of trends common to all three problems (which may represent intrinsic properties of the algorithm itself), as well as a host of problem-specific features. We find that the evolutionary algorithm tends to undergo a number of phases which we are able to quantitatively delimit. The results obtained lead us to believe that the measures presented herein may prove useful in the general case of analyzing fine-grained evolutionary algorithms.

Keywords

Parallel evolutionary algorithms, cellular automata, cellular programming, genetic algorithms, statistical analysis.

1 Introduction

The field of evolutionary computation encompasses a wide range of methodologies, including genetic algorithms, evolution strategies, evolutionary programming, and genetic programming. Central to all these approaches is the notion of solving problems by evolving an initially random population of candidate solutions, through the application of operators inspired by natural genetics and natural selection, such that "fitter" (i.e., better) solutions emerge. Evolutionary algorithms have been gaining ubiquity over the past decade and are commonly used in industry and academia. They have been successfully applied to numerous problems from different domains including optimization, automatic programming,

machine learning, economics, immune systems, ecology, population genetics, and social systems.

One of the basic aspects of evolutionary algorithms is their inherent parallelism: the existence of a population implies that there are several individuals evolving in parallel. This has not escaped practitioners in the field, who have explored the issue of parallel evolutionary algorithms. One can cite two basic motivations underlying these parallelization efforts. First, there is the wish to reduce the necessary run time, thus expediting the emergence of a solution to the problem at hand. A second motivation lies in the algorithmic benefits resulting from a parallel implementation that echoes evolution in nature—the field's fundamental inspiration.

A basic tenet of parallel evolutionary algorithms is that the population has a spatial structure. A number of models based on this observation have been proposed, the two most important being the *island* model and the *grid* model. The coarse-grained island model features geographically separated subpopulations of relatively large size. Subpopulations exchange information by having some individuals migrate from one subpopulation to another with a given frequency and according to various migrational patterns (see Figure 1a). This can work to offset premature convergence by periodically re-injecting diversity into otherwise converging subpopulations. In the fine-grained grid model individuals are placed on a toroidal d-dimensional grid (where $d = 1, 2, 3$ is used in practice), one individual per grid location. This approach is also known as *cellular* (Whitley, 1993; Tomassini, 1993)(see Figure 1b). Fitness evaluation is done simultaneously for all individuals with genetic operators taking place locally within a small neighborhood. From an implementation point of view, coarse-grained island models, where the ratio of computation to communication is high, are more adapted to multiprocessor systems or workstation clusters. Fine-grained cellular models are better suited for massively parallel machines or specialized hardware. Hybrid models are also possible, e.g., one might consider an island model in which each island is structured as a grid of locally interacting individuals. For recent reviews of parallel evolutionary algorithms the reader is referred to Cantú-Paz (1995) and Tettamanzi and Tomassini (1998).

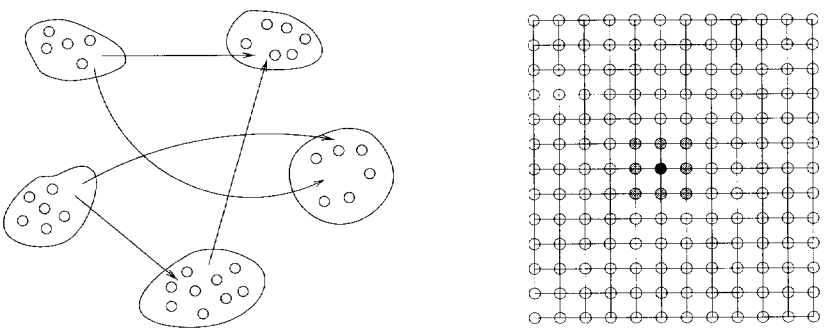


Figure 1: The two basic models of parallel evolutionary algorithms: (a) the coarse-grained island model, and (b) the fine-grained grid (or cellular) model.

Though such parallel models have empirically proven worthwhile (Cohoon et al., 1987; Starkweather et al., 1991; Loraschi et al., 1995; Manderick and Spiessens, 1989; Tomassini, 1993; Oussaidene et al., 1997; Andre and Koza, 1996), there lacks a better understanding

of their workings. Gaining insight into the mechanisms of parallel evolutionary algorithms calls for the introduction of statistical measures of analysis. This is the underlying motivation of our paper. Specifically, concentrating on cellular models, our objectives are: (1) to introduce several statistical measures of interest, both at the genotypic and phenotypic levels that are useful for analyzing the workings of fine-grained parallel evolutionary algorithms; and (2) to demonstrate the application and utility of these measures on a specific example, that of the cellular programming evolutionary algorithm (Sipper, 1997a). Among the few theoretical works carried out to date, one can cite Mühlenbein (1991), Cantú-Paz and Goldberg (1997), and Rudolph and Sprave (1995). The latter treated a special case of fine-grained cellular algorithms, studying its convergence properties. However, they did not present statistical measures as done herein.

We begin Section 2 with a presentation of basic formal definitions. Section 3 introduces the various statistical measures used in the analysis of cellular evolutionary algorithms. Section 4 describes cellular programming, our choice of evolutionary algorithm for demonstrating the viability and applicability of our approach. In Section 5 we apply the statistics of Section 3 to analyze the cellular programming algorithm when used to evolve solutions to three different problems: density, synchronization, and random number generation. We conclude in Section 6.

2 Basic Definitions and Notation

We now formally define the basic elements used in this paper (a summary is provided in Table 1). A *population* is a collection of individuals, each represented by a genotype. A genotype is not necessarily unique—it may occur several times in the population. In addition, if the population has a topology, the spatial distribution of the genotypes is of interest. Let n be the number of individuals in the system. Let R_i, $1 \leq i \leq n$ be the genome of the ith individual. Let Γ be the space of genotypes and $G(\Gamma)$ be the space of all possible populations. Let $f(\gamma)$ be the fitness of an individual having genotype $\gamma \in \Gamma$. When the cells are arranged in a row, as in the examples in Section 4, a population can be defined as a vector of n genotypes $x = (R_1, \ldots, R_n)$. We have $G(\Gamma) = \Gamma^n$, provided the row of cells is not folded into a circle by connecting the extremity cells. For example, with a three-bit genotype, the space of genotypes Γ is: $\{000, 001, 010, ..., 111\}$; for a population of size $n = 2$, $G(\Gamma)$ is: $\Gamma^2 = \{\{000, 000\}, \{000, 001\}, ..., \{111, 111\}\}$.

For all populations $x \in G(\Gamma)$, an occupancy function $n_x: \Gamma \rightarrow N$ is defined, such that, for all $\gamma \in \Gamma$, $n_x(\gamma)$ is the number of individuals in x sharing the same genotype γ, i.e., the occupancy number of γ in x. The size of population x, $\|x\|$, is defined as $|x\| \equiv \sum_{\gamma \in \Gamma} n_x(\gamma)$.

We can now define a share function $q_x: \Gamma \rightarrow [0, 1]$ giving the fraction $q_x(\gamma)$ of individuals in x that have genotype γ, i.e., $q_x(\gamma) = n_x(\gamma)/\|x\|$.

Consider the probability space $(\Gamma, 2^\Gamma, \mu)$, where 2^Γ is the algebra of the parts of Γ and μ is any probability measure on Γ. Let us denote by $\tilde{\mu}$ the probability of generating a population $x \in G(\Gamma)$ by extracting n genotypes from Γ according to measure μ. It can be shown that it is sufficient to know either of the two measures—μ (over the genotypes) or $\tilde{\mu}$ (over the populations) in order to reconstruct the other.

The fitness function establishes a morphism from genotypes into real numbers. If genotypes are distributed over Γ according to a given probability measure μ, then their fitness will be distributed over the reals according to a probability measure ϕ obtained from

Table 1: Nomenclature.

Γ	Space of genotypes
γ	Genotype in Γ
$f(\gamma)$	Fitness of genotype γ
$G(\Gamma)$	Space of populations
μ	Probability measure over Γ
$\tilde{\mu}$	Probability measure over $G(\Gamma)$
n	Population size
t	Generation
x	Population in $G(\Gamma)$
$n_x(\gamma)$	Occupancy number of genotype γ in population x
$q_x(\gamma)$	Share of genotype γ in population x
P	Probability measure over the trajectories
ϕ	Probability function over fitness
R_i	Genotype of ith individual in the population
X_t	(Random) population at tth generation
Ω	Space of all possible evolutionary trajectories
ω	Evolutionary trajectory

μ by applying the same morphism. This can be summarized by the following:

$$\begin{array}{ccc} \Gamma & \xrightarrow{f} & \mathbb{R} \\ \wr & & \wr \\ \mu & & \phi \end{array} \tag{1}$$

The probability $\phi(v)$ of a given fitness value $v \in [0, +\infty)$ is defined as the probability that an individual extracted from Γ according to measure μ has fitness v (or, if we think of fitness values as a continuous space, the probability density of fitness v). For all $v \in [0, +\infty)$, $\phi(v) = \mu(f^{-1}(v))$, where $f^{-1}(v) \equiv \{\gamma \in \Gamma : f(\gamma) = v\}$.

An evolutionary algorithm can be regarded as a time-discrete stochastic process

$$\{X_t(\omega)\}_{t=0,1,2,\ldots}, \tag{2}$$

having the probability space $(\Omega, \mathcal{F}, P)$ as its base space, $(G(\Gamma), 2^{G(\Gamma)})$ as its state space, and the natural numbers as the set of times, called *generations*. Ω might be thought of as the set of all the evolutionary trajectories, $\mathcal{F}$ is a σ-algebra on Ω, and P is a probability measure over $\mathcal{F}$.

The transition function of the evolutionary process, in turn based on the definition of the genetic operators, defines a sequence of probability measures over the generations.

Let $\tilde{\mu}_t$ denote the probability measure on the state space at time t; for all populations $x \in G(\Gamma)$,

$$\tilde{\mu}_t(x) = P\{\omega \in \Omega : X_t(\omega) = x\} \tag{3}$$

In the same way, let μ_t denote the probability measure on space $(\Gamma, 2^{\Gamma})$ at time t; for all $\gamma \in \Gamma$,

$$\mu_t(\gamma) = P[\kappa = \gamma | \kappa \in X_t(\omega)] \tag{4}$$

Similarly, we define the sequence of probability functions $\phi_t(\cdot)$ as follows: for all $v \in [0, +\infty)$ and $t \in N$,

$$\phi_t(v) = \mu_t(f^{-1}(v)) \tag{5}$$

3 Statistical Measures for Cellular Evolutionary Algorithms

In this section we shall introduce several statistics pertaining to cellular evolutionary algorithms, which can be divided into two classes: genotypic statistics, which embody aspects related to the genotypes of individuals in a population; and phenotypic statistics, which concern properties of individual performance (fitness) for the problem at hand. Table 2 provides a summary of our statistics.

Table 2: Summary of statistics introduced in this section.

Notation	Formula	Explanation
$n_x(\gamma)$		Occupancy number of γ in population x
$q_x(\gamma)$	$n_x(\gamma)/\|\|x\|\|$	Share of γ in population x
$\nu(x)$	$\frac{\sum_{i=1}^{n}\sum_{j\in N(i)}[R_i\neq R_j]}{\sum_{i=1}^{n}\|\|N(i)\|\|}$	Frequency of transitions in population x
$H(x)$	$\sum_{\gamma\in\Gamma} q_x(\gamma)\log\frac{1}{q_x(\gamma)}$	Entropy of population x
$D(x)$	$\frac{n}{n-1}\sum_{\gamma\in\Gamma} q_x(\gamma)(1-q_x(\gamma))$	Genotypic diversity of population x
$E[\phi_x]$		Performance of population x
$\sigma^2(x)$	$\mathrm{Var}[\phi_x]$	Phenotypic diversity of population x
$\rho^2(x)$	$\frac{1}{n}\sum_{i=1}^{n}\left[1-\frac{1+\|\|N(i)\|\|f(R_i)}{1+\sum_{j\in N(i)}f(R_j)}\right]^2$	Ruggedness of population x

3.1 Genotypic Statistics

One important class of statistics consists of various genotypic diversity indices (within the population) whose definitions are based on the occupancy and share functions delineated below.

Occupancy and Share Functions

At any time $t \in N$, for all $\gamma \in \Gamma$, $n_{X_t}(\gamma)$ is a discrete random variable with binomial distribution

$$P[n_{X_t}(\gamma) = k] = \binom{n}{k}\mu_t(\gamma)^k[1-\mu_t(\gamma)]^{n-k}; \tag{6}$$

thus, $E[n_{X_t}(\gamma)] = n\mu_t(\gamma)$ and $\mathrm{Var}[n_{X_t}(\gamma)] = n\mu_t(\gamma)[1-\mu_t(\gamma)]$. The share function $q_{X_t}(\gamma)$ is perhaps more interesting, because it is an estimator of the probability measure $\mu_t(\gamma)$; its mean and variance can be calculated from those of $n_{X_t}(\gamma)$, yielding

$$E[q_{X_t}(\gamma)] = \mu_t(\gamma) \quad \text{and} \quad \mathrm{Var}[q_{X_t}(\gamma)] = \frac{\mu_t(\gamma)[1-\mu_t(\gamma)]}{n} \tag{7}$$

Structure

Statistics in this category measure properties of the population structure, that is, how individuals are spatially distributed. Obviously, such statistics apply only to populations that have a spatial structure (see Figure 1b).

Frequency of Transitions

The frequency of transitions $\nu(x)$ of a population x of n individuals (cells) is defined as the number of borders between homogeneous blocks of cells having the same genotype divided by the number of distinct couples of adjacent cells. In other words, $\nu(x)$ is the probability that two adjacent individuals (cells) have different genotypes.

Formally, the frequency of transitions $\nu(x)$ can be expressed as

$$\nu(x) = \frac{\sum_{i=1}^{n} \sum_{j \in N(i)} [R_i \neq R_j]}{\sum_{i=1}^{n} \|N(i)\|}, \tag{8}$$

where $[P]$ denotes the indicator function of proposition P, and $N(i)$ is the neighborhood of cell i. In the example studied in Section 4, we have a one-dimensional grid structure, whereby Equation 8 reduces to

$$\nu(x) = \frac{1}{n} \sum_{i=1}^{n} \left[R_i \neq R_{(i \bmod n)+1}\right] \tag{9}$$

Diversity

There are a number of ways to measure genotypic diversity; we define two: population entropy, and the probability that two individuals in the population have different genotypes.

Entropy. The entropy of a population x of size n is defined as

$$H(x) = \sum_{\gamma \in \Gamma} q_x(\gamma) \log \frac{1}{q_x(\gamma)} \tag{10}$$

Entropy takes on values in the interval $[0, \log n]$ and attains its maximum, $H(x) = \log n$, when x comprises n different genotypes.

Diversity Indices. The probability that two individuals randomly chosen from x have different genotypes is denoted by $D(x)$.

Index $D(X_t)$ is an estimator of quantity

$$\sum_{\gamma \in \Gamma} \mu_t(\gamma)\,(1 - \mu_t(\gamma)) = 1 - \sum_{\gamma \in \Gamma} \mu_t(\gamma)^2, \tag{11}$$

which relates to the "breadth" of measure μ_t.

PROPOSITION 1: *Let x be a population of n individuals with genotypes in Γ. Then,*

$$D(x) = \frac{n}{n-1} \sum_{\gamma \in \Gamma} q_x(\gamma)(1 - q_x(\gamma)) \tag{12}$$

PROOF: We choose the first individual at random with uniform probability: it will have genotype γ with probability $q_x(\gamma)$. We then choose a second individual without replacement from the remaining $n-1$ individuals. This time, the probability that it will have genotype γ is $\frac{n_x(\gamma)-1}{n-1}$, given that the first individual has genotype γ. Therefore, the probability that it will *not* have genotype γ is

$$1 - \frac{n_x(\gamma) - 1}{n-1} = \frac{n - n_x(\gamma)}{n-1} \tag{13}$$

Now, we have to sum this probability over all possible genotypes that can be extracted first, weighted by the probability of extracting them, which is q_x:

$$D(x) = \sum_{\gamma \in \Gamma} q_x(\gamma) \frac{n - n_x(\gamma)}{n-1} \tag{14}$$

Observe that

$$\frac{n - n_x(\gamma)}{n-1} = \frac{n - nq_x(\gamma)}{n-1} = \frac{n}{n-1}(1 - q_x(\gamma)) \tag{15}$$

Substituting in Equation 14 yields the thesis.

We observe that for all populations $x \in G(\Gamma)$,

$$D(x) \geq \frac{H(x)}{\log n} \tag{16}$$

In other words, $D(x)$ rises more steeply than entropy as diversity increases.

An interesting relationship between D and ν is given by the following proposition.

PROPOSITION 2: *Given a random one-dimensional linear population x of size n, the expected frequency of transitions will be given by*

$$E[\nu(x)] = D(x) \tag{17}$$

PROOF: We express the expected frequency of transitions:

$$E[\nu(x)] = \frac{1}{n} \sum_{i=1}^{n} P[\gamma_i \neq \gamma_{(i \bmod n)+1}] = \frac{1}{n} \sum_{i=1}^{n} D(x) = D(x) \tag{18}$$

(Note that this can also be proven for dimensions higher than one by generalizing the above proof.)

3.2 Phenotypic Statistics

Phenotypic statistics deal with properties of phenotypes, primarily, fitness. Associated with a population x of individuals, there is a fitness distribution. We will denote by ϕ_x its (discrete) probability function.

Performance

The performance of population x is defined as its average fitness, or the expected fitness of an individual randomly extracted from x, $E[\phi_x]$.

Diversity

The most straightforward measure of phenotypic diversity of a population x is the variance of its fitness distribution, $\sigma^2(x) = \mathrm{Var}[\phi_x]$.

Structure

Statistics in this category measure how fitness is spatially distributed across the individuals in a population.

Ruggedness

Ruggedness measures the dependency of an individual's fitness on its neighbors' fitness. Ruggedness of population $x \in G(\Gamma)$ of size n can be defined as follows:

$$\rho^2(x) = \frac{1}{n}\sum_{i=1}^{n}\left[1 - \frac{1 + \|N(i)\|f(R_i)}{1 + \sum_{j\in N(i)} f(R_j)}\right]^2 \tag{19}$$

Notice that $\rho^2(x)$ is independent of the fitness magnitude in population x, i.e., of performance $E[\phi_x]$.

For a one-dimensional population of size n, Equation 19 becomes:

$$\rho^2(x) = \frac{1}{n}\sum_{i=1}^{n}\left[1 - \frac{1 + 2f(R_i)}{1 + f(R_{(i \bmod n)+1}) + f(R_{(i-2 \bmod n)+1})}\right]^2 \tag{20}$$

4 Cellular Programming

In this section we present the cellular programming algorithm, a fine-grained evolutionary algorithm. We shall demonstrate the applicability of our measures in the next section.

4.1 Cellular Automata

Our evolving machines are based on the cellular automata model. Cellular automata (CA) are dynamic systems in which space and time are discrete. A cellular automaton consists of an array of cells, each of which can be in one of a finite number of possible states, updated synchronously in discrete time steps, according to a local, identical interaction rule. The state of a cell at the next time step is determined by the previous states of a surrounding neighborhood of cells. This transition is usually specified in the form of a *rule table*, delineating the cell's next state for each possible neighborhood configuration (Wolfram, 1994; Toffoli and Margolus, 1987). The cellular array (grid) is d-dimensional, where $d = 1, 2, 3$ is used in practice (in this paper we shall concentrate on $d = 1$). A one-dimensional CA is illustrated in Figure 2 (based on Mitchell (1996)).

CAs exhibit three notable features: massive parallelism, locality of cellular interactions, and simplicity of basic components (cells). As such they are naturally suited for hardware implementation with the potential of exhibiting extremely fast and reliable computation that is robust to noisy input data and component failure. A major impediment to ubiquitous computing with CAs stems from the difficulty of using their complex behavior to perform

Rule Table:

neighborhood:	111	110	101	100	011	010	001	000
output bit:	1	1	1	0	1	0	0	0

Grid:

$t = 0$	0	1	1	0	1	0	1	1	0	1	1	0	0	1	1
$t = 1$	1	1	1	1	0	1	1	1	1	1	1	0	0	1	1

Figure 2: Illustration of a one-dimensional, 2-state CA. The connectivity radius is $r = 1$, meaning that each cell has two neighbors, one to its immediate left and one to its immediate right. Grid size is $n = 15$. The rule table for updating the grid is shown on top. The grid configuration over one time step is shown at the bottom. Spatially periodic boundary conditions are applied, meaning that the grid is viewed as a circle, with the leftmost and rightmost cells each acting as the other's neighbor.

useful computations. Designing CAs to exhibit a specific behavior or to perform a particular task is complicated and limits their applications. This results from the local dynamics of the system, which renders the design of local rules to perform global computational tasks extremely arduous. Automating the design (programming) process would greatly enhance the viability of CAs (Mitchell et al., 1994; Sipper, 1997a).

The model investigated in this paper is an extension of the CA model, termed *non-uniform cellular automata* (Sipper 1996, 1997a, 1998). Such automata function in the same way as uniform ones, the only difference being in the cellular rules that need not be identical for all cells. Our main focus is on the *evolution* of non-uniform CAs to perform computational tasks using the cellular programming approach. Rather than seek a *single* rule that must be universally applied to all cells in the grid, each cell is allowed to "choose" its own rule through evolution.

4.2 The Cellular Programming Algorithm

We study 2-state, non-uniform CAs, in which each cell may contain a different rule. A cell's rule table is encoded as a bit string (the "genome"), containing the next-state (output) bits for all possible neighborhood configurations (see Figure 2). Rather than employ a population of evolving, uniform CAs, as with standard genetic algorithm approaches, our algorithm involves a single, non-uniform CA of size n, where the population of cell rules is initialized at random. Initial configurations are then generated at random in accordance with the task at hand, and for each one the CA is run for M time steps. Each cell's fitness is accumulated over $C = 300$ initial configurations, where a single run's score is 1 if the cell is in the correct state after M iterations, and 0 otherwise. After every C configurations evolution of rules occurs by applying crossover and mutation. This evolutionary process is performed in a completely *local* manner, where genetic operators are applied only between directly connected cells. It is driven by $nf_i(c)$, the number of fitter neighbors of cell i after c configurations. The pseudo-code of the algorithm is delineated in Figure 3.

Crossover between two rules is performed by selecting at random (with uniform prob-

```
for each cell i in CA do in parallel
    initialize rule table of cell i
    f_i = 0 {fitness value}
end parallel for
c = 0 {initial configurations counter}
while not done do
    generate a random initial configuration
    run CA on initial configuration for M time steps
    for each cell i do in parallel
        if cell i is in the correct final state then
            f_i = f_i + 1
        end if
    end parallel for
    c = c + 1
    if c mod C = 0 then {evolve every C configurations}
        for each cell i do in parallel
            compute nf_i(c) {number of fitter neighbors}
            if nf_i(c) = 0 then rule i is left unchanged
            else if nf_i(c) = 1 then replace rule i with the fitter neighboring rule,
                                    followed by mutation
            else if nf_i(c) = 2 then replace rule i with the crossover of the two fitter
                                    neighboring rules, followed by mutation
            else if nf_i(c) > 2 then replace rule i with the crossover of two randomly
                                    chosen fitter neighboring rules, followed by mutation
                                    (this case can occur if the cellular neighborhood includes
                                    more than two cells)
            end if
            f_i = 0
        end parallel for
    end if
end while
```

Figure 3: Pseudo-code of the cellular programming algorithm.

ability) a single crossover point and creating a new rule by combining the first rule's bit string before the crossover point with the second rule's bit string after the crossover point. Mutation is applied to the bit string of a rule with probability 0.001 per bit.

There are two main differences between the cellular programming algorithm and the standard genetic algorithm approach (e.g., Mitchell et al., 1994). First, the latter involves a population of evolving, uniform CAs; all CAs are *ranked* according to fitness with crossover occurring between *any* two individuals in the population. Thus, while the CA runs in accordance with a local rule, evolution proceeds in a *global* manner. In contrast, the cellular programming algorithm proceeds *locally* in the sense that each cell has access only to its locale, not only during the run but also during the evolutionary phase, and no global fitness ranking is performed. Second, the standard genetic algorithm involves a population of *independent* problem solutions; the CAs in the population are assigned fitness values independent of one another and interact only through the genetic operators in order to produce the next generation. In contrast, our CA *coevolves* since each cell's fitness depends upon its evolving neighbors. This may also be considered a form of symbiotic cooperation, which falls, as does coevolution, under the general heading of "ecological" interactions (see Mitchell (1996), 182-183). In summary, cellular programming is a local, coevolutionary,

parallel genetic algorithm.

4.3 The Computational Tasks

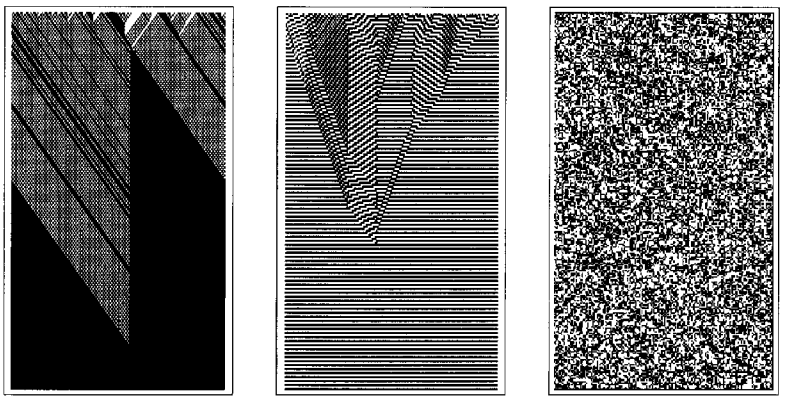
In the next section we study three computational tasks using one-dimensional, 2-state, $r = 1$ CAs: density, synchronization, and random number generation. Spatially periodic boundary conditions are applied, resulting in a circular grid. The tasks are described below.

The Density Task. The one-dimensional density task is to decide whether or not the initial configuration contains more than 50% 1s, relaxing to a fixed-point pattern of all 1s if the initial density of 1s exceeds 0.5, and all 0s otherwise (see Figure 4a). As noted by Mitchell et al. (1994), the density task comprises a non-trivial computation for a small radius CA ($r \ll n$, where n is the grid size). Density is a global property of a configuration whereas a small-radius CA relies solely on local interactions. Since the 1s can be distributed throughout the grid, propagation of information must occur over large distances (i.e., $O(n)$). The minimum amount of memory required for the task is $O(\log n)$ using a serial-scan algorithm, thus the computation involved corresponds to recognition of a non-regular language. Note that the density task cannot be perfectly solved by a uniform, two-state CA, as proven by Land and Belew (1995). This result applies to the above problem statement where the CA's final pattern (output) is specified as a fixed-point configuration. Interestingly, it has recently been proven that by changing the output specification, a two-state, $r = 1$ uniform CA exists that can perfectly solve the density problem (Capcarrère et al., 1996).

The Synchronization Task. The one-dimensional synchronization task was introduced by Das et al. (1995) and studied by Sipper (1997a, 1997b) using non-uniform CAs. In this task the CA, given any initial configuration, must reach a final configuration, within M time steps, that oscillates between all 0s and all 1s on successive time steps (see Figure 4b). As with the density task, synchronization also comprises a non-trivial computation for a small-radius CA.

Random Number Generation (RNG). Random numbers are needed in a variety of applications, yet finding good random number generators is a difficult task (Park and Miller, 1988). To generate a random sequence on a digital computer, one starts with a fixed-length seed and iteratively applies some transformation to it, progressively extracting as long as possible a random sequence. Such numbers are usually referred to as *pseudo*-random, as distinguished from true random numbers resulting from some natural physical process. In the last decade, cellular automata have been used to generate random numbers (Hortensius et al., 1989; Wolfram, 1994).

Sipper and Tomassini (1996a, 1996b) applied the cellular programming algorithm to evolve random number generators. Essentially, the cell's fitness score for a single configuration (see Figure 3) is the entropy of the temporal bit sequence of that cell with higher entropy implying better fitness (this should not be confused with the entropy measures defined in Section 3). This fitness measure was used to drive the evolutionary process after which standard tests were applied to evaluate the quality of the evolved CAs. The results obtained suggest that good generators can indeed be evolved (see Figure 4c). These exhibit behavior at least as good as that of previously described CAs with notable advantages arising from the existence of a "tunable" algorithm for obtaining random number generators.



(a) (b) (c)

Figure 4: Demonstration of three evolved non-uniform CAs. Grid is one-dimensional, with radius $r = 1$ and size $n = 150$. White squares represent cells in state 0, black squares represent cells in state 1. The pattern of configurations is shown through time, which increases down the page. Initial configurations were generated at random. Figure (a) shows the density task. Initial density of 1s is greater than 0.5 and the CA relaxes to a fixed pattern of all 1s, correctly classifying the initial configuration. Figure (b) shows the synchronization task. Final pattern consists of an oscillation between a configuration of all 0s and a configuration of all 1s. Figure (c) shows random number generation. Essentially, each cell's sequence of states through time is a pseudo-random bit stream.

5 Results and Analysis

Using the different measures presented in Section 3, we analyzed the processes taking place during the execution of the cellular programming algorithm presented in the previous section. This was carried out for all three tasks delineated in Subsection 4.3 (density, synchronization, and RNG). From previous experiments, we knew that they present different degrees of difficulty for the evolutionary algorithm, with density being the hardest and the other two being easier (though non-trivial (Sipper 1996, 1998)). All experiments were run for CAs of size 150. The number of experiments varied with the tasks: 20 for RNG, 56 for density, and 75 for synchronization.

First we present features commonly observed for all tasks, which may thus represent properties intrinsic to the algorithm itself rather than idiosyncrasies of a particular task. Specific properties are described in the subsequent subsections.

5.1 Common Features

There are a number of trends common to all three tasks. A notable feature we observed is the significant decrease in entropy (H) which is initially high, due to the random nature of the initial population. This can be considered as an increase in order, reflected in the trends exhibited by other measures.

In all runs the entropy falls from a high of approximately 0.8 to a low of approximately 0.7 within the first 20 generations, and subsequently tends to decline. Though this decline is

not monotonous, regardless of the outcome the entropy ends up below 0.5 for all three tasks. This fall in entropy is due to two factors. First, we can observe in all runs a steep drop in the transition frequency (ν) in the first few generations followed by an almost continuous drop in the subsequent generations whose slope is task dependent (see Figure 5). Though it may be intuitive that blocks will tend to form given the possibility of rule replication between neighboring cells after each generation, our measures now provide us with quantitative evidence. Note that the transition frequency (ν) progresses towards an oscillatory state about values below 0.3 (including the synchronization case, though this occurs around generation 160). The second factor involved in the lower entropy is the number of rules. One can see directly that a low ν implies few rules. This is corroborated by the decreasing trend of diversity (D).

For the three tasks studied herein the objective is to reach a high average fitness over the entire population rather than consider just the highest-fitness individual cell. Intuitively, we can expect that the phenotypic variance will tend to be minimized, and we can factually check that both the fitness variance (σ^2) and ruggedness (ρ^2) are always very low towards the end of an evolutionary run. A clear correlation between the genotypic and phenotypic measures we observed is that a higher fitness variance results in a much steeper decrease of ν, rendering the grid more uniform. Usually the evolved CA had less than 10 different rules out of the 256 possible ones. We found that fitness variance (σ^2) is about ten times higher for RNG and density than for synchronization, which underlies the difference in the declivity of the ν curves in Figure 5.

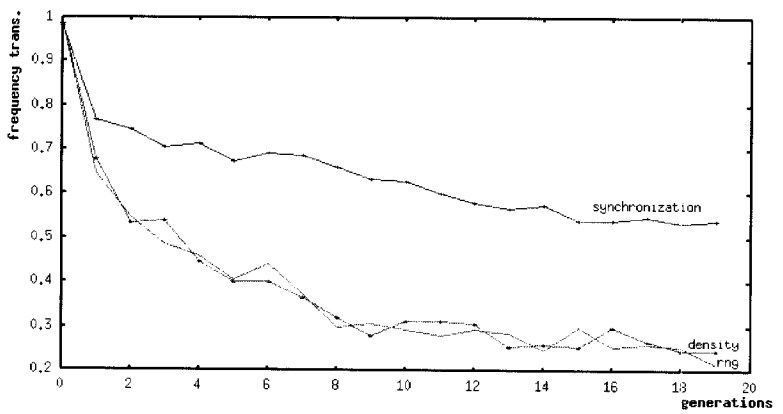


Figure 5: Progression of the transition frequency (ν) over the first 20 generations of typical runs.

In conclusion, there is an increase in order. This means that the population tends to evolve towards a small number of rules organized in blocks, i.e., a low H and ν. As we will see in the following subsections, which detail properties specific to each task, the exact values of H, D, and ν differ.

5.2 Random Number Generation (RNG)

The evolution of random number generators is highly successful in terms of final fitness. Though high fitness does not necessarily entail good random behavior, Sipper and Tomassini

(1996a, 1996b) have shown that such evolved CAs do fare well on several randomness tests. For all runs, we obtained normalized fitness values higher than 0.99 in relatively few generations (approximately 100).

One can note that there is a steady increase in fitness during the first 60 generations, accompanied by a relatively sharp descent in entropy (H) dropping from a high of 0.8 to a low of approximately 0.3. We see in Figure 6 that the fitness average is already high (approximately 0.98) when the entropy reaches the low-value range. We then observe that the entropy stabilizes, and the average fitness remains generally high. During the stabilization phase the genotypic variance (ν) declines to almost 0. The evolved CA usually contained less than 10 different rules.

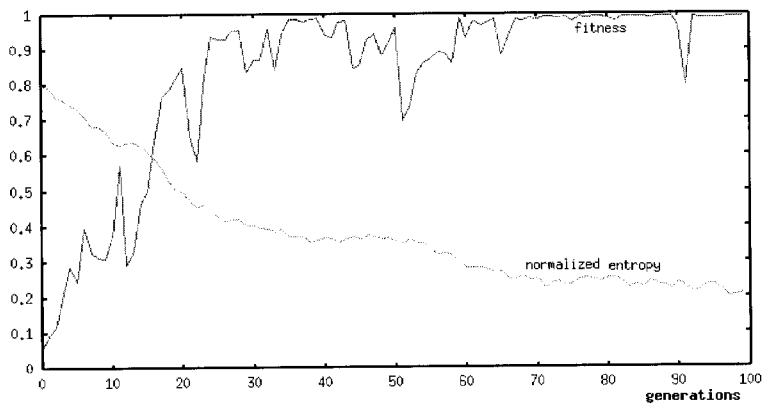


Figure 6: Fitness and entropy (H) vs. time for a typical run of the RNG task. We observe that H declines while fitness increases, both eventually leveling off.

5.3 Synchronization

The evolutionary dynamics of the synchronization task were found to exhibit at most three fitness phases: a low-fitness phase, a rapid-increase phase, and a high-fitness phase. Note that for this task a successful run is considered to be one where a perfect fitness value of 1.0 is attained. The evolutionary runs can be classified in four distinct classes. Two of represent successful runs (Figures 7a and 7b) and the two represent unsuccessful runs (Figures 7c and 7d). The classification is based on the number of phases exhibited during the evolution. We now present the results of our experiments according to these three fitness phases.

Phase I: Low Fitness. This phase is characterized by an average fitness of 0.5, with an extremely low variance. However, while exhibiting phenotypic (fitness) "calmness," this phase is marked by high underlying genotypic activity: the entropy (H) steadily decreases, and the number of rules strongly diminishes. An unsuccessful type-c run (Figure 7c) results from "genotypic failure" in this phase. To explain, let us first note that for the synchronization task, only rules with neighborhood 111 mapped to 0 and 000 mapped to 1 (cf. Figure 2) may appear in a successful solution. Let us call this the "good" quadrant of the rule space and define the "bad" quadrant to be the one that comprises rules mapping 111 to 1 and 000 to 0. In some experiments, evolution

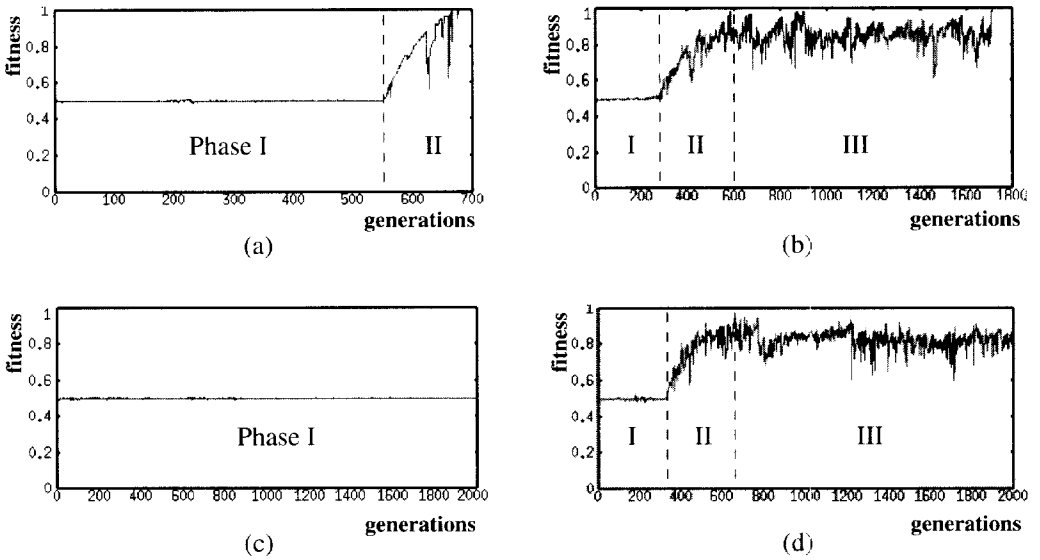


Figure 7: The evolutionary runs for the synchronization task can be classified in four distinct classes based on the three observed fitness phases: phase I (low fitness), phase II (rapid fitness increase), and phase III (high fitness). (a) represents a successful run exhibiting the first two phases. The solution is found at the end of phase II. (b) represents a successful run exhibiting all three phases. The solution is found at the end of phase III. (c) represents an unsuccessful run "stuck" in phase I. (d) represents an unsuccessful run, exhibiting all three phases. Phase III does not give rise to a perfect solution.

falls into the bad quadrant, possibly due to a low fitness variance. Only the mutation operator can possibly hoist the evolutionary process out of this trap. However, it is usually insufficient with the mutation rate used here. In such a case, the algorithm is stuck in a local minimum, and fitness never ascends beyond 0.53 (see Figure 7c).

Phase II: Rapid Fitness Increase. A rapid increase of fitness characterizes this phase. Its onset is marked by the attainment of a 0.54 fitness value (at least). This comes about when a sufficiently large block of rules from the good quadrant emerges through the evolutionary process. In a relatively short time after this emergence (less than 100 generations), evolved rules over the entire grid all end up in the good quadrant of the rule space. This is coupled with a high fitness variance (σ^2). This variance then drops sharply while the average fitness steadily increases, reaching a value of 0.8 at the end of this phase. Another characteristic of this phase is the sharp drop in entropy. On certain runs, a perfect CA was found directly at the end of this stage, thus bringing the evolutionary process to a successful conclusion (see Figure 7a).

Phase III: High Fitness. The transition from phase II to phase III is not clear cut, but we observed that when a fitness of approximately 0.82 is reached, the fitness average then begins to oscillate between 0.65 and 0.99. During this phase the fitness variance also oscillates between approximately 0 and 0.3. While low, this variance is still higher than that of phase I. Whereas in phases I and II we observed a clear decreasing trend for entropy (H), in this phase entropy exhibits an oscillatory pattern between values of

approximately 0.3 and 0.5 (see Figure 8). We conclude that when order (low entropy) is too high, disorder is re-injected into the evolutionary process, while remaining in the good quadrant of the rule space—hence the oscillatory behavior. On certain runs it took several hundred generations in this phase to evolve a perfect CA—this is a success of type b (see Figure 7b). Finally, on other runs no perfect CA was found, though phase III was reached and very high fitness was attained. This is a type-d unsuccessful run (see Figure 7d) which does not differ significantly from type-b successful runs.

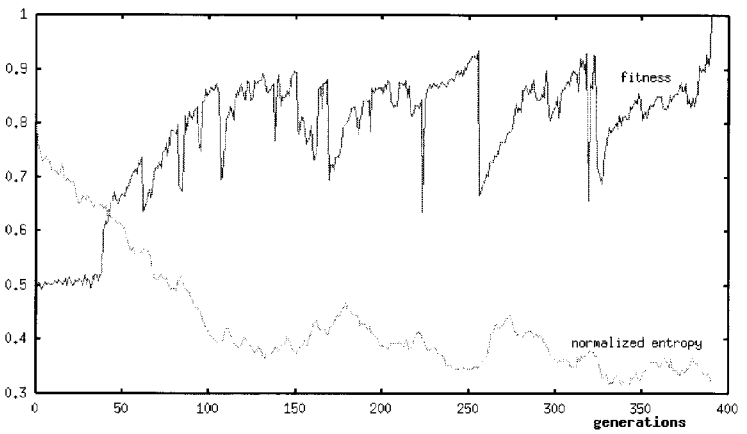


Figure 8: Synchronization task: Entropy and average fitness for a successful run of type b. We can observe phase I (generations 0–38), phase II (generations 39–110), and phase III (generations 111 and onward).

5.4 Density

Density is arguably the most difficult task of the three. Indeed, this task necessitated the longest evolution times, with fitness never exceeding 0.95. Thus, following Sipper (1996), we define a successful density CA to be one with average fitness higher than 0.9.

One can distinguish three fitness phases: a rapid increase phase, followed by a stable phase, ending with an unstable phase. Note that they are different from the synchronization task phases. These phases also differ from those identified by Mitchell et al. (1994), when applying a genetic algorithm to the evolution of uniform CAs that solve the density task. They observed four phases dubbed "epochs of innovation". The difference between the dynamics of their algorithm and ours is not altogether surprising in view of the very different natures of the two.

Phase I: Rapid Fitness Increase. During this phase, the frequency transition (ν) and the entropy (H) rapidly decrease from approximately 0.8 to 0.4 within the first 10 to 20 generations, while average fitness increases to approximately 0.8. With this task, rules that are from the "bad" quadrant can nonetheless exhibit high fitness, between approximately 0.75 and 0.85 (here the "bad" quadrant of the rule space is the one in which the genome maps 111 to 0 and 000 to 1). This presents an obstacle to evolution, possibly leading to a local minimum, where only the mutation operator may be of help.

Phase II: Fitness Stabilization. After this initial burst of high evolutionary activity in phase I, phase II average fitness levels off and begins to oscillate in a narrow band of values between 0.8 and 0.85. Entropy oscillates between 0.1 and 0.3. Note that phase II for a successful run exhibits more oscillatory behavior than that of its counterpart in an unsuccessful run (see Figure 9). This again shows that almost total phenotypic inactivity is often counterbalanced by high genotypic activity. Nevertheless, we observed in certain runs that the entropy stabilized at low values (below 0.1). This always meant that phase III was never reached and resulted in an unsuccessful run.

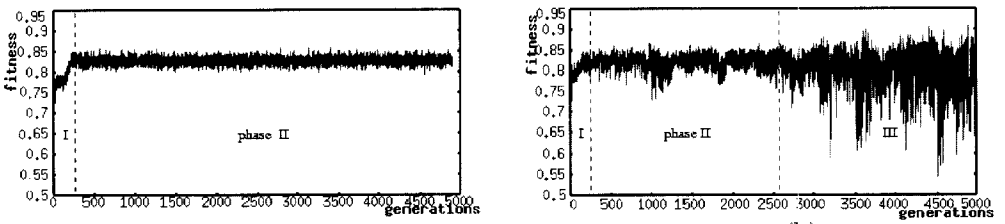


Figure 9: The evolutionary runs for the density task can be classified in two distinct classes, based on the three observed fitness phases: phase I (rapid fitness increase), phase II (fitness stabilization), and phase III (unstable fitness). (a) shows an unsuccessful run exhibiting only the first two phases. (b) shows a successful run exhibiting all three phases. The solution is found during phase III. (Note that phase II for a successful run exhibits more oscillatory behavior than that of its counterpart in an unsuccessful run; however, phase III can still be readily distinguished by a net increase in oscillations.)

Phase III: Unstable Fitness. This phase, the onset of which is the hallmark of successful runs, is characterized by the de-stabilization of the average fitness (see Figure 10). It exhibits highly oscillating fitness behavior with values possibly differing by as much as 0.3 (between approximately 0.65 and 0.95) from one generation to the next. This suggests that the fitness landscape is highly rugged, which may explain the relative difficulty of the task.

6 Concluding Remarks

In this paper we introduced several statistical measures of interest, at both the genotypic and phenotypic levels, which are useful for analyzing the workings of fine-grained parallel evolutionary algorithms. We then demonstrated their application and utility on a specific example, the cellular programming evolutionary algorithm, which we employed to evolve solutions to three different problems: density, synchronization, and random number generation.

We observed a number of features common to all three tasks, which may represent inherent algorithmic properties. First is the notable difference between activity at the genotypic level and at the phenotypic level. At several stages of the evolutionary process, though seemingly little phenotypic (observable) activity is taking place, the population is

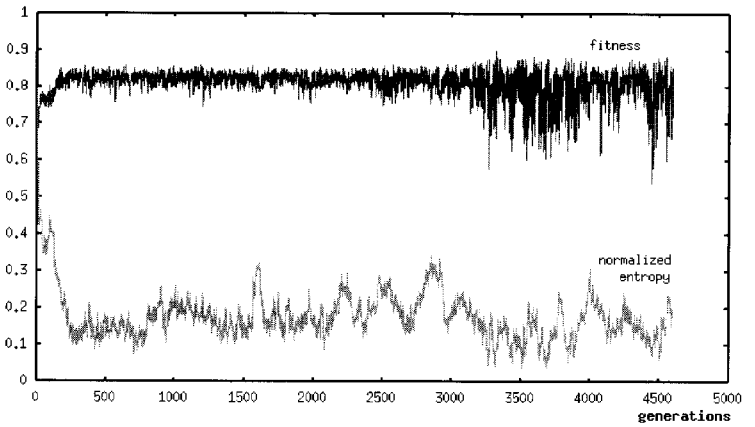


Figure 10: Density task: Entropy and average fitness for a successful run. We can see the beginning of the unstable fitness phase at generation 3100 during which the best CA is found (here at generation 3315).

in fact teeming with underlying genotypic activity. The latter gives rise at certain points in time to discernible phenotypic effects evident by fitness phase transitions. We noted a strong decrease in diversity (D), transition frequency (ν), entropy (H), and the number of rules. This provides quantitative evidence that the population tends to evolve towards a more ordered state with respect to the initial state which is random and therefore highly disordered.

Each task also exhibited a number of specific properties. For the RNG task a high-fitness solution was always evolved in usually no more than 100 generations. The synchronization task was seen to undergo (at most) three fitness phases: a low-fitness phase, followed by a rapid increase in fitness, ending with a high-fitness phase. The nature of these stages, or the absence of some of them, served to distinguish between four types of evolutionary runs. Finally, the density task was seen to undergo (at most) three evolutionary phases: a rapid-increase phase, followed by a highly stable phase, ending with an unstable phase. A successful solution was observed to emerge only during the third phase (which was not necessarily reached, e.g., in certain failed runs).

Parallel evolutionary algorithms have been receiving increased attention in recent years. Gaining a better understanding of their workings and of their underlying mechanisms presents an important research challenge. We feel that the work presented here represents a step in this direction.

References

Andre, D. and Koza, J. R. (1996). Parallel genetic programming: A scalable implementation using the transputer network architecture. In Angeline, P. and Kinnear, K., editors, *Advances in Genetic Programming 2*, MIT Press, Cambridge, Massachusetts.

Cantú-Paz, E. (1995). A summary of research on parallel genetic algorithms. Technical Report 95007, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Urbana, Illinois.

Cantú-Paz, E. and Goldberg, D. E. (1997). Modeling idealized bounding cases of parallel genetic algorithms. In Koza, J. R., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M., Iba, H. and Riolo, R. L., editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 353–361, Morgan Kaufmann, San Francisco, California.

Capcarrère, M. S., Sipper, M. and Tomassini, M. (1996). Two-state, r=1 cellular automaton that classifies density. *Physical Review Letters*, 77(24):4969–4971.

Cohoon, J. P., Hedge, S. U., Martin, W. N. and Richards, D. (1987). Punctuated equilibria: A parallel genetic algorithm. In Grefenstette, J. J., editor, *Proceedings of the Second International Conference on Genetic Algorithms*, page 148, Lawrence Erlbaum Associates, Hillsdale, New Jersey.

Das, R., Crutchfield, J. P., Mitchell, M. and Hanson, J. E. (1995). Evolving globally synchronized cellular automata. In Eshelman, L. J., editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 336–343, Morgan Kaufmann, San Francisco, California.

Hortensius, P. D., McLeod, R. D. and Card, H. C. (1989). Parallel random number generation for VLSI systems using cellular automata. *IEEE Transactions on Computers*, 38(10):1466–1473.

Land, M. and Belew, R. K. (1995). No perfect two-state cellular automata for density classification exists. *Physical Review Letters*, 74(25):5148–5150.

Loraschi, A., Tettamanzi, A., Tomassini, M. and Verda, P. (1995). Distributed genetic algorithms with an application to portfoli selection problems. In Pearson, D. W., Steele, N. C. and Albrecht, R. F., editors, *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms*, pages 384–387, Springer-Verlag, New-York, New York.

Manderick, B. and Spiessens, P. (1989). Fine-grained parallel genetic algorithms. In Schaffer, J. D., editor, *Proceedings of the Third International Conference on Genetic Algorithms*, page 428, Morgan Kaufmann, San Mateo, California.

Mitchell, M. (1996). *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, Massachusetts.

Mitchell, M., Crutchfield, J. P. and Hraber, P. T. (1994). Evolving cellular automata to perform computations: Mechanisms and impediments. *Physica D*, 75:361–391.

Mühlenbein, H. (1991). Evolution in time and space–the parallel genetic algorithm. In Rawlins, G. J. E., editor, *Foundations Of Genetic Algorithms I*. Morgan Kaufmann, San Mateo, California.

Oussaidene, M., Chopard, B., Pictet, O. and Tomassini, M. (1997). Parallel genetic programming and its application to trading model induction. *Parallel Computing*, 23:1183–1198.

Park, S. K. and Miller, K. W. (1988). Random number generators: Good ones are hard to find. *Communications of the ACM*, 31(10):1192–1201.

Rudolph, G. and Sprave, J. (1995). A cellular genetic algorithm with self-adjusting acceptance threshold. In *First IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, pages 365–372, IEE, London, England.

Sipper, M. (1996). Co-evolving non-uniform cellular automata to perform computations. *Physica D*, 92:193–208.

Sipper, M. (1997a). *Evolution of Parallel Cellular Machines: The Cellular Programming Approach*. Springer-Verlag, Heidelberg, Germany.

Sipper, M. (1997b). The evolution of parallel cellular machines: Toward evolware. *BioSystems*, 42:29–43.

Sipper, M. (1998). Computing with cellular automata: Three cases for nonuniformity. *Physical Review E*, 57(3):3589–3592.

Sipper, M. and Tomassini, M. (1996a). Co-evolving parallel random number generators. In Voigt, H.-M., Ebeling, W., Rechenberg, I. and Schwefel, H.-P., editors, *Parallel Problem Solving from Nature - PPSN IV*, volume 1141 of *Lecture Notes in Computer Science*, pages 950–959, Springer-Verlag, Heidelberg, Germany.

Sipper, M. and Tomassini, M. (1996b). Generating parallel random number generators by cellular programming. *International Journal of Modern Physics C*, 7(2):181–190.

Starkweather, T., Whitley, D. and Mathias, K. (1991). Optimization using distributed genetic algorithms. In Schwefel, H.-P. and Männer, R, editors, *Parallel Problem Solving from Nature*, volume 496 of *Lecture Notes in Computer Science*, page 176, Springer-Verlag, Heidelberg, Germany.

Tettamanzi, A. and Tomassini, M. (1998). Evolutionary algorithms and their applications. In Mange, D. and Tomassini, M., editors, *Bio-Inspired Computing Machines: Toward Novel Computational Architectures*, pages 59–98, Presses Polytechniques et Universitaires Romandes, Lausanne, Switzerland.

Toffoli, T. and Margolus, N. (1987). *Cellular Automata Machines*. MIT Press, Cambridge, Massachusetts.

Tomassini, M. (1993). The parallel genetic cellular automata: Application to global function optimization. In Albrecht, R. F., Reeves, C. R. and Steele, N. C., editors, *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms*, pages 385–391, Springer-Verlag, Wien, Germany.

Whitley, D. (1993). Cellular genetic algorithms. In Forrest, S., editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, page 658, Morgan Kaufmann, San Mateo, California.

Wolfram, S. (1994). *Cellular Automata and Complexity*. Addison-Wesley, Reading, Massachusetts.